Augmented Reality
Camera Calibration

projection centre

image point

M = (X, Y, Z)

object

optical axis

[R, t]
Agenda

- Homogeneous Coordinates
- Projective Transformation
- Pinhole Camera Model
- Camera Calibrations
- Zhengyou Zhang's Camera Calibration
- Augmented Reality
Homogeneous Coordinates

lb@dongseo.ac.kr
Homogeneous coordinates

Homogeneous representation of lines

\[ ax + by + c = 0 \quad (a, b, c)^T \]

\[(ka)x + (kb)y + kc = 0, \forall k \neq 0 \quad (a, b, c)^T \sim k(a, b, c)^T \]

equivalence class of vectors, any vector is representative

Set of all equivalence classes in \( \mathbb{R}^3 - (0,0,0)^T \) forms \( \mathbb{P}^2 \)

Homogeneous representation of points

\[ x = (x, y, 1)^T \text{on } l = (a, b, c)^T \text{ if and only if } ax + by + c = 0 \]

\[(x, y, 1)(a, b, c)^T = (x, y, 1)1 = 0 \quad (x, y, 1)^T \sim k(x, y, 1)^T, \forall k \neq 0 \]

The point \( x \) lies on the line \( l \) if and only if \( x^T l = l^T x = 0 \)

*Homogeneous coordinates* \( (x, y, z)^T \) but only 2DOF

*Inhomogeneous coordinates* \( (x, y)^T \)
Points from lines and vice-versa

Intersections of lines
The intersection of two lines \( l \) and \( l' \) is \( x = l \times l' \)

Line joining two points
The line through two points \( x \) and \( x' \) is \( l = x \times x' \)

Example

\[
\begin{align*}
\mathbf{a} \times \mathbf{b} &= \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} \\
\mathbf{a} \times \mathbf{b} &= \begin{vmatrix} a_2 & a_3 \\ b_2 & b_3 \end{vmatrix} \mathbf{i} - \begin{vmatrix} a_1 & a_3 \\ b_1 & b_3 \end{vmatrix} \mathbf{j} + \begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} \mathbf{k}
\end{align*}
\]
Ideal points and the line at infinity

Intersections of parallel lines

\[ \mathbf{l} = (a, b, c)^T \quad \text{and} \quad \mathbf{l}' = (a, b, c')^T \quad \mathbf{l} \times \mathbf{l}' = (b, -a, 0)^T \]

Example

\[ (b, -a) \quad \text{tangent vector} \]
\[ (a, b) \quad \text{normal direction} \]

Ideal points

\[ \begin{pmatrix} x_1 \\ x_2 \\ 0 \end{pmatrix}^T \]

Line at infinity

\[ \mathbf{l}_\infty = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}^T \]

\[ \mathbb{P}^2 = \mathbb{R}^2 \cup \mathbf{l}_\infty \]

Note that in \( \mathbb{P}^2 \) there is no distinction between ideal points and others.
A model for the projective plane

exactly one line through two points
exactly one point at intersection of two lines
Duality principle:

To any theorem of 2-dimensional projective geometry there corresponds a dual theorem, which may be derived by interchanging the role of points and lines in the original theorem.
Projective Transformations
Projective 2D Geometry
Projective transformations

Definition:

A *projectivity* is an invertible mapping \( h \) from \( P^2 \) to itself such that three points \( x_1, x_2, x_3 \) lie on the same line if and only if \( h(x_1), h(x_2), h(x_3) \) do.

Theorem:

A mapping \( h: P^2 \rightarrow P^2 \) is a projectivity if and only if there exist a non-singular 3x3 matrix \( H \) such that for any point in \( P^2 \) represented by a vector \( x \) it is true that \( h(x) = Hx \)

Definition: Projective transformation

\[
\begin{bmatrix}
  x'_1 \\
  x'_2 \\
  x'_3
\end{bmatrix} =
\begin{bmatrix}
  h_{11} & h_{12} & h_{13} \\
  h_{21} & h_{22} & h_{23} \\
  h_{31} & h_{32} & h_{33}
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  x_3
\end{bmatrix}
\]

or \( x' = Hx \)

8DOF

projectivity=collineation=projective transformation=homography
Mapping between planes

central projection may be expressed by $x' = Hx$
(application of theorem)
Removing projective distortion

select four points in a plane with known coordinates

\[ \begin{align*}
x' &= \frac{x'_1}{x'_3} = \frac{h_{11} x + h_{12} y + h_{13}}{h_{31} x + h_{32} y + h_{33}} \\
y' &= \frac{x'_2}{x'_3} = \frac{h_{21} x + h_{22} y + h_{23}}{h_{31} x + h_{32} y + h_{33}}
\end{align*} \]

\[ \begin{align*}
x'(h_{31} x + h_{32} y + h_{33}) &= h_{11} x + h_{12} y + h_{13} \\
y'(h_{31} x + h_{32} y + h_{33}) &= h_{21} x + h_{22} y + h_{23}
\end{align*} \]
(linear in \( h_{ij} \))

(2 constraints/point, 8DOF \( \Rightarrow \) 4 points needed)
Smarter Presentations: Exploiting Homography in Camera-Projector Systems

Rahul Sukthankar\textsuperscript{1,2}, Robert G. Stockton\textsuperscript{1}, Matthew D. Mullin\textsuperscript{1}

\textsuperscript{1}Just Research \hspace{1em} \textsuperscript{2}The Robotics Institute
4616 Henry Street \hspace{1em} Carnegie Mellon
Pittsburgh, PA 15213 \hspace{1em} Pittsburgh, PA 15213

\{rahuls,rgs,mdm\}@justresearch.com

At first glance, it may appear that this mapping is impossible to determine in the presence of so many unknowns. Fortunately, we can exploit the fact that all of the observed points in the scene lie on some unknown plane (the flat projection screen), and this establishes a homography between the camera and projector frames of reference. Thus, we can show that the compounded transforms mapping \((x, y)\) in the projector frame, to some unknown point on the projection screen, and then to pixel \((X, Y)\) in the camera frame, can be expressed by a single projective transform,

\[
(x, y) = \begin{pmatrix} p_1 X + p_2 Y + p_3 \\ p_7 X + p_8 Y + p_9 \end{pmatrix},
\]

with eight degrees of freedom. \(\vec{\rho} = (p_1 \ldots p_9)^T\) constrained by \(|\vec{\rho}| = 1\). The same transform is more concisely expressed in homogeneous coordinates as:

\[
\begin{pmatrix} x w \\ y w \\ w \end{pmatrix} = \begin{pmatrix} p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 \\ p_7 & p_8 & p_9 \end{pmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}
\]

\(\vec{\rho}\) can be determined from as few as four pixel correspondences\textsuperscript{2}; when more than four correspondences are available, the system finds the best estimate in a least-squares sense. Given \(n\) feature point matches, \\{(\(x_i, y_i\), \((X_i, Y_i)\)\}, let \(A\) be the following \(2n \times 9\) matrix:

\[
\begin{pmatrix}
X_1 & Y_1 & 1 & 0 & 0 & 0 & -X_1 x_1 & -Y_1 y_1 & -x_1 \\
0 & 0 & 0 & X_1 & Y_1 & 1 & -X_1 y_1 & -Y_1 y_1 & -y_1 \\
X_2 & Y_2 & 1 & 0 & 0 & 0 & -X_2 x_2 & -Y_2 y_2 & -x_2 \\
0 & 0 & 0 & X_2 & Y_2 & 1 & -X_2 y_2 & -Y_2 y_2 & -y_2 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
X_n & Y_n & 1 & 0 & 0 & 0 & -X_n x_n & -Y_n y_n & -x_n \\
0 & 0 & 0 & X_n & Y_n & 1 & -X_n y_n & -Y_n y_n & -y_n
\end{pmatrix}
\]

The goal is to find the unit vector \(\vec{\rho}\) that minimizes \(|Ap|\), and this is given by the eigenvector corresponding to the smallest eigenvalue of \(A^T A\).
More Examples
Transformation for lines

For a point transformation
\[ x' = Hx \]

Transformation for lines
\[ l' = H^{-T}l \]
Isometries

\[
\begin{pmatrix}
x' \\
y' \\
1
\end{pmatrix} =
\begin{bmatrix}
\varepsilon \cos \theta & -\sin \theta & t_x \\
\varepsilon \sin \theta & \cos \theta & t_y \\
0 & 0 & 1
\end{bmatrix}
\begin{pmatrix}
x \\
y \\
1
\end{pmatrix}
\]

\[\varepsilon = \pm 1\]

- orientation preserving: \(\varepsilon = 1\)
- orientation reversing: \(\varepsilon = -1\)

\[x' = H_E x = \begin{bmatrix}
R & t \\
0^\top & 1
\end{bmatrix} x\]

\[R^\top R = I\]

3DOF (1 rotation, 2 translation)

special cases: pure rotation, pure translation

**Invariants:** length, angle, area
Similarities

\[
\begin{pmatrix}
x' \\ y' \\ 1
\end{pmatrix} = \begin{bmatrix}
s \cos \theta & -s \sin \theta & t_x \\ s \sin \theta & s \cos \theta & t_y \\ 0 & 0 & 1
\end{bmatrix} \begin{pmatrix}
x \\ y \\ 1
\end{pmatrix}
\]

\[x' = H_s x = \begin{bmatrix} sR & t \\ 0^T & 1 \end{bmatrix} x \quad \text{with} \quad R^T R = I\]

4DOF (1 scale, 1 rotation, 2 translation)
also known as equi-form (shape preserving)

metric structure = structure up to similarity (in literature)

**Invariants:** ratios of length, angle, ratios of areas, parallel lines
Affine transformations

\[
\begin{pmatrix}
x' \\
y' \\
1
\end{pmatrix} =
\begin{bmatrix}
a_{11} & a_{12} & t_x \\
a_{21} & a_{22} & t_y \\
0 & 0 & 1
\end{bmatrix}
\begin{pmatrix}
x \\
y \\
1
\end{pmatrix}
\]

\[
x' = H_A x = \begin{bmatrix} A & t \\ 0^T & 1 \end{bmatrix} x
\]

\[
A = R(\theta)R(-\phi)DR(\phi)
\]

\[
D = \begin{bmatrix}
\lambda_1 & 0 \\
0 & \lambda_2
\end{bmatrix}
\]

6DOF (2 scale, 2 rotation, 2 translation)
non-isotropic scaling! (2DOF: scale ratio and orientation)

**Invariants:** parallel lines, ratios of parallel lengths, ratios of areas
Projective transformations

\[ x' = H_p x = \begin{bmatrix} A & t \\ v^T & v \end{bmatrix} x \]

\[ v = \left(v_1, v_2\right)^T \]

8DOF (2 scale, 2 rotation, 2 translation, 2 line at infinity)
Action non-homogeneous over the plane

**Invariants**: cross-ratio of four points on a line (ratio of ratio)

\[ H = H_s H_A H_p = \begin{bmatrix} sR & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} K & 0 \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} I & 0 \\ v^T & v \end{bmatrix} = \begin{bmatrix} A & t \\ v^T & v \end{bmatrix} \]

\[ A = sRK + tv^T \]

decomposition unique (if chosen \( s > 0 \))

\[ K \text{ upper-triangular, } \det K = 1 \]
### Overview Transformations

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Parameters</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Projective</strong></td>
<td>$h_{11}$, $h_{12}$, $h_{13}$</td>
<td>Concurrency, collinearity, order of contact (intersection, tangency, inflection, etc.), cross ratio</td>
</tr>
<tr>
<td><strong>8dof</strong></td>
<td>$h_{21}$, $h_{22}$, $h_{23}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$h_{31}$, $h_{32}$, $h_{33}$</td>
<td></td>
</tr>
<tr>
<td><strong>Affine</strong></td>
<td>$a_{11}$, $a_{12}$, $t_x$</td>
<td>Parallellism, ratio of areas, ratio of lengths on parallel lines (e.g. midpoints), linear combinations of vectors (centroids).</td>
</tr>
<tr>
<td><strong>6dof</strong></td>
<td>$a_{21}$, $a_{22}$, $t_y$</td>
<td><strong>The line at infinity</strong> $l_\infty$</td>
</tr>
<tr>
<td><strong>Similarity</strong></td>
<td>$s_{r_{11}}$, $s_{r_{12}}$</td>
<td>Ratios of lengths, angles.</td>
</tr>
<tr>
<td><strong>4dof</strong></td>
<td>$s_{r_{21}}$, $s_{r_{22}}$</td>
<td><strong>The circular points</strong> $I, J$</td>
</tr>
<tr>
<td><strong>Euclidean</strong></td>
<td>$r_{11}$, $r_{12}$, $t_x$</td>
<td>lengths, areas.</td>
</tr>
<tr>
<td><strong>3dof</strong></td>
<td>$r_{21}$, $r_{22}$, $t_y$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$0$</td>
<td></td>
</tr>
</tbody>
</table>
Line at infinity

\[
\begin{bmatrix}
A & t \\
0^T & v
\end{bmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
0
\end{pmatrix}
= \begin{pmatrix}
A(x_1) \\
A(x_2) \\
0
\end{pmatrix}
\]

Line at infinity stays at infinity

\[
\begin{bmatrix}
A & t \\
v^T & v
\end{bmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
0
\end{pmatrix}
= \begin{pmatrix}
A(x_1) \\
A(x_2) \\
v_1x_1 + v_2x_2
\end{pmatrix}
\]

Line at infinity becomes finite, allows to observe vanishing points, horizon,
The line at infinity $l_\infty$ is a fixed line under a projective transformation $H$ if and only if $H$ is an affinity.

\[ l'_\infty = H^{-T}_A l_\infty = \begin{bmatrix} A^{-T} & 0 \\ -A^T & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = l_\infty \]
Affine properties from images

Projection

\[ \mathbf{H}_P \]

Rectification

\[ \mathbf{H}_P' \]

\[ \mathbf{H}_A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{bmatrix} \]

\[ 1_\infty = \begin{bmatrix} l_1 \\ l_2 \\ l_3 \end{bmatrix}, l_3 \neq 0 \]
Affine Rectification

\[ v_1 = l_1 \times l_2 \]

\[ v_2 = l_3 \times l_4 \]
Projective 3D geometry

- **Projective 15dof**
  $$\begin{bmatrix} A & t \\ v^T & v \end{bmatrix}$$
  Intersection and tangency

- **Affine 12dof**
  $$\begin{bmatrix} A & t \\ 0^T & 1 \end{bmatrix}$$
  Parallellism of planes,
  Volume ratios, centroids,
  **The plane at infinity** $\pi_\infty$

- **Similarity 7dof**
  $$\begin{bmatrix} sR & t \\ 0^T & 1 \end{bmatrix}$$
  **The absolute conic** $\Omega_\infty$

- **Euclidean 6dof**
  $$\begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix}$$
  Volume
Pinhole Camera Model

lb@dongseo.ac.kr
Pinhole Camera Model

\[(X, Y, Z)^T \mapsto (fX / Z, fY / Z)^T\]

\[
\begin{bmatrix}
X \\
Y \\
Z \\
1
\end{bmatrix}
\mapsto
\begin{bmatrix}
fX \\
fY
\end{bmatrix} =
\begin{bmatrix}
f & 0 \\
f & 0 \\
1 & 0
\end{bmatrix}
\begin{bmatrix}
X \\
Y \\
Z \\
1
\end{bmatrix}
\]

\[
\begin{bmatrix}
fX \\
fY
\end{bmatrix} =
\begin{bmatrix}
f & 1 & 0 \\
1 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix}
\]

![Diagram of pinhole camera model showing camera center, principal axis, and image plane.](image)
Pinhole Camera Model

$$x = K[I\mid 0]X_{\text{cam}}$$

$$K = \begin{bmatrix} f & p_x \\ f & p_y \\ 1 \end{bmatrix}$$

Principal point offset

Calibration Matrix
Internal Camera Parameters

\[
\begin{bmatrix}
  u' \\
  v' \\
  w'
\end{bmatrix} =
\begin{bmatrix}
  \alpha_x & s & x_0 & 0 \\
  0 & \alpha_y & y_0 & 0 \\
  0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
  x_s \\
  y_s \\
  z_s \\
  1
\end{bmatrix}
\]

with

\[
\alpha_x = f \ k_x \quad \text{and} \quad \alpha_y = -f \ k_y
\]

\[
x_{\text{pix}} = \frac{u'}{w'} \quad \text{and} \quad y_{\text{pix}} = \frac{v'}{w'}
\]

\[
\begin{bmatrix}
  \alpha_x & s & x_0 & 0 \\
  0 & \alpha_y & y_0 & 0 \\
  0 & 0 & 1 & 0
\end{bmatrix}
= K \begin{bmatrix} I_3 & 0_3 \end{bmatrix}
\]

- \( \alpha_x \) and \( \alpha_y \) “focal lengths” in pixels
- \( x_0 \) and \( y_0 \) coordinates of image center in pixels
- Added parameter \( s \) is skew parameter
- \( K \) is called calibration matrix. Five degrees of freedom.
  - \( K \) is a 3x3 upper triangular matrix
Camera rotation and translation

\[
\tilde{X}_{\text{cam}} = R(\tilde{X} - \tilde{C})
\]

\[
X_{\text{cam}} = \begin{bmatrix} R & -R\tilde{C} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} R & -RC \\ 0 & 1 \end{bmatrix} X
\]

\[
x = K[I \mid 0]X_{\text{cam}}
\]

\[
x = KR[I \mid -\tilde{C}]X
\]

\[
x = PX
\]

\[
P = K[R \mid t] \quad t = -R\tilde{C}
\]
Camera Parameter Matrix $P$

- Further simplification of $P$:

$$x = K \begin{bmatrix} I_3 & 0_3 \end{bmatrix} \begin{bmatrix} R & -R \tilde{C} \\ 0_3^T & 1 \end{bmatrix} X$$

$$[I_3 \ | \ 0_3] \begin{bmatrix} R & -R \tilde{C} \\ 0_3^T & 1 \end{bmatrix} = [R \ | \ -R \tilde{C}] = R[I_3 \ | \ -\tilde{C}]$$

$$x = KR[I_3 \ | \ -\tilde{C}]X$$

$$P = KR[I_3 \ | \ -\tilde{C}]$$

- $P$ has 11 degrees of freedom:
  - 5 from triangular calibration matrix $K$,
  - 3 from $R$ and 3 from $\tilde{C}$
- $P$ is a fairly general $3 \times 4$ matrix
  - left $3 \times 3$ submatrix $KR$ is non-singular
CCD Cameras
Correcting Radial Distortion of Cameras

\[ x_u = c_x + (x_d - c_x) f_2(r_d^2) \]
\[ = c_x + (x_d - c_x)(1 + k_1 r_d^2 + k_2 r_d^4 + k_3 r_d^6) \]
\[ y_u = c_y + (y_d - c_y) f_2(r_d^2) \]
\[ = c_y + (y_d - c_y)(1 + k_1 r_d^2 + k_2 r_d^4 + k_3 r_d^6) \]
\[ r_d^2 = (x_d - c_x)^2 + (y_d - c_y)^2 \]

\[ x_p = \frac{m_0 x_u + m_1 y_u + m_2}{m_6 x_u + m_7 y_u + 1} \]
\[ y_p = \frac{m_3 x_u + m_4 y_u + m_5}{m_6 x_u + m_7 y_u + 1} \]
Correcting Radial Distortion of Cameras with Wide Angle Lens Using Point Correspondences

Leonardo Romero and Cuauhtemoc Gomez

División de Estudios de Postgrado, Facultad de Ingeniería Eléctrica
Universidad Michoacana de San Nicolás de Hidalgo
Morelia, Michoacán, México

\[ e_{xk} = x_p(\Theta, x_{dk}, y_{dk}) - x_{rk} \]
\[ e_{yk} = y_p(\Theta, x_{dk}, y_{dk}) - y_{rk} \]
\[ E(\Theta) = \sum_{k=1}^{n} (e_{xk}^2 + e_{yk}^2) \]
\[ \Theta = \text{argmin} E(\Theta) \]

5.1 Non-Linear Optimization
The Gauss-Newton-Levenberg-Marquardt method (GNLM) (Press et al., 1986) is a non-linear iterative technique specifically designated for minimizing functions which has the form of sum of square functions, like \( E \). At each iteration the increment of parameters, vector \( \delta \Theta \), is computed solving the following linear matrix equation:

\[ [A + \lambda I] \delta \Theta = B \] (8)

If there is \( n \) point correspondences and \( q \) parameters in \( \Theta \), \( A \) is a matrix of dimension \( q \times q \) and matrix \( B \) has dimension \( q \times 1 \) and \( \delta \Theta = [\delta \theta_1, \delta \theta_2, ..., \delta \theta_q] \). \( \lambda \) is a parameter which is allowed to vary at each iteration. After a little algebra, the elements of \( A \) and \( B \) can be computed using the following formulas,

\[ a_{i,j} = \sum_{k=1}^{n} \left( \frac{\partial x_{pk}}{\partial \theta_i} \frac{\partial x_{pk}}{\partial \theta_j} + \frac{\partial y_{pk}}{\partial \theta_i} \frac{\partial y_{pk}}{\partial \theta_j} \right) \]
\[ b_i = -\sum_{k=1}^{n} \left( \frac{\partial x_{pk}}{\partial \theta_i} e_{xk} + \frac{\partial y_{pk}}{\partial \theta_i} e_{yk} \right) \]
5.4 The Calibration Process

The calibration process starts with one image from the camera, \( I_d \), another image from the calibration pattern, \( I_r \), and initial values for parameters \( \Theta \). In the following algorithm, \( \Theta \) and \( \delta \theta \) are considered as vectors. We start with \((c_x, c_y)\) at the center of the image, \( k_1 = k_2 = k_3 = 0 \) and the identity matrix for \( M \). The calibration algorithm is as follows:

1. From the reference image, compute the reference feature points \((x_{rk}, y_{rk})\), \( k = 1, \ldots, n \).
2. From \( \Theta \) and the distorted image, compute a corrected image.
3. From the corrected image compute the set of feature points \((x_{pk}, y_{pk})\), \( k = 1, \ldots, n \).
4. From \((x_{pk}, y_{pk})\) \( k = 1, \ldots, n \) and \( \Theta \) compute \((x_{dk}, y_{dk})\) \( k = 1, \ldots, n \).
5. Find the best \( \Theta \) that minimize \( E \) using the GNLM algorithm:
   (a) Compute the total error, \( E(\Theta) \) (eq. 7).
   (b) Pick a modest value for \( \lambda \), say \( \lambda = 0.001 \).
   (c) Solve the linear system of equations (8), and calculate \( E(\Theta + \delta \Theta) \).
   (d) If \( E(\Theta + \delta \Theta) \geq E(\Theta) \), increase \( \lambda \) by a factor of 10, and go to the previous step. If \( \lambda \) grows very large, it means that there is no way to improve the solution \( \Theta \).
   (e) If \( E(\Theta + \delta \Theta) < E(\Theta) \), decrease \( \lambda \) by a factor of 10, replace \( \Theta \) by \( \Theta + \delta \Theta \), and go to step 5a.
6. Repeat steps 2-5 until \( E(\Theta) \) does not decrease.

When \( \lambda = 0 \), the GNLM method is a Gauss-Newton method, and when \( \lambda \) tends to infinity, \( \delta \theta \) turns to so called steepest descent direction and the size of \( \delta \theta \) tends to zero.

The calibration algorithm apply several times the GNLM algorithm to get better solutions. At the beginning, the clusters of the distorted image are not perfect squares and so point features can not match exactly the feature points computed using the reference image. Once a corrected image is ready, point features can be better estimated.
Correcting Radial Distortion of Camera with Rotation Angle

\[ x_p = \frac{m_0 x_u + m_1 y_u + m_2}{m_6 x_u + m_7 y_u + 1} \]
\[ y_p = \frac{m_3 x_u + m_4 y_u + m_5}{m_6 x_u + m_7 y_u + 1} \]

\[ x_u = c_x + (x_d - c_x)(1 + k_1 r_d^2 + k_2 r_d^4 + k_3 r_d^6) \cos \theta \]
\[-(y_d - c_y)(1 + k_1 r_d^2 + k_2 r_d^4 + k_3 r_d^6) \sin \theta \]
\[ y_u = c_y + (x_d - c_x)(1 + k_1 r_d^2 + k_2 r_d^4 + k_3 r_d^6) \sin \theta \]
\[ + (y_d - c_y)(1 + k_1 r_d^2 + k_2 r_d^4 + k_3 r_d^6) \cos \theta \]
Virtual Interactive Board  U-Pointer

Wants to have an interactive electronic board on any wall in your room? Place the U-pointer beside the beam projector and connect it to a computer and the wall of your choice becomes an interactive electronic board. It is a Virtual Interactive Board.

Main Feature of U-Pointer

**Easy Installation**
- Place the U-Pointer on top of the projector and connect it to the PC via a USB cable.
- Align the U-Pointer on the beamed screen and adjust it by knob.
- Light weight along with easy installation enables the U-Pointer to be your presentation partner.

**Interaction with PC via the projected screen**
- Click, double click and drag functions of conventional mouse are all fully supported by the U-Pointer’s pen.
- It can control PC from the projected screen interactively.
- It provides the freedom of a wireless presentation in a seminar or conference, and the size of the projected screen is up to 150 inches.

**Accurate Pen Work**
- The high resolution of the U-Pointer provides the capability of writing smaller characters, than those written with a tablet.
- The fast response time and over 59 frames per second of sampling rate give you the same feeling as if you were using a white board.
Camera Calibration

lb@dongseo.ac.kr
Camera Models

Coordinate transformations in Tsai, Heikkilä and Zhang’s camera models

<table>
<thead>
<tr>
<th>Tsai</th>
<th>Heikkilä</th>
<th>Zhang</th>
</tr>
</thead>
</table>
| \[
\begin{bmatrix}
X_c \\
Y_c \\
Z_c
\end{bmatrix} = R \begin{bmatrix}
X_w \\
Y_w \\
Z_w
\end{bmatrix} + t
\] | \[
\begin{bmatrix}
X_c \\
Y_c \\
Z_c
\end{bmatrix} = R \begin{bmatrix}
X_w \\
Y_w \\
Z_w
\end{bmatrix} + t
\] | \[
\begin{bmatrix}
X_c \\
Y_c \\
Z_c
\end{bmatrix} = R \begin{bmatrix}
X_w \\
Y_w \\
Z_w
\end{bmatrix} + t
\] |

\[
\begin{bmatrix}
x_u \\
y_u
\end{bmatrix} = \frac{f}{Z_c} \begin{bmatrix}
X_c \\
Y_c
\end{bmatrix}
\] | \[
\begin{bmatrix}
x_u \\
y_u
\end{bmatrix} = \frac{f}{Z_c} \begin{bmatrix}
X_c \\
Y_c
\end{bmatrix}
\] | \[
\begin{bmatrix}
x_u \\
y_u
\end{bmatrix} = \frac{1}{Z_c} \begin{bmatrix}
X_c \\
Y_c
\end{bmatrix}
\]

where \( r = \sqrt{x_d^2 + y_d^2} \)

\[
\begin{bmatrix}
x_u \\
y_u
\end{bmatrix} = (1 + k_1^{(H)} r^2 + k_2^{(H)} r^4) \begin{bmatrix}
x_d \\
y_d
\end{bmatrix}
\] | \[
\begin{bmatrix}
x_u \\
y_u
\end{bmatrix} = (1 + k_1^{(H)} r^2 + k_2^{(H)} r^4) \begin{bmatrix}
x_d \\
y_d
\end{bmatrix}
\] | \[
\begin{bmatrix}
x_u \\
y_u
\end{bmatrix} = (1 + k_1^{(Z)} r^2 + k_2^{(Z)} r^4) \begin{bmatrix}
x_u \\
y_u
\end{bmatrix}
\]

where \( r = \sqrt{x_u^2 + y_u^2} \)

\[
\begin{bmatrix}
x_p \\
y_p \\
1
\end{bmatrix} = \begin{bmatrix}
s_x/d_x & 0 & c_x \\
0 & 1/d_y & c_y \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
x_d \\
y_d \\
1
\end{bmatrix}
\] | \[
\begin{bmatrix}
x_p \\
y_p \\
1
\end{bmatrix} = \begin{bmatrix}
s_x D_x & 0 & c_x \\
0 & D_y & c_y \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
x_d \\
y_d \\
1
\end{bmatrix}
\] | \[
\begin{bmatrix}
x_p \\
y_p \\
1
\end{bmatrix} = \begin{bmatrix}
\alpha & \gamma & c_x \\
0 & \beta & c_y \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
x_d \\
y_d \\
1
\end{bmatrix}
\]

“An empirical evaluation of factors influencing camera calibration accuracy using three publicly available techniques”, Wei Sun and Jeremy R. Cooperstock
Machine Vision and Applications Volume 17, Number 1 (2006), 51-67,
A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses

ROGER Y. TSAI

Abstract—A new technique for three-dimensional (3D) camera calibration for machine vision metrology using off-the-shelf TV cameras and lenses is described. The two-stage technique is aimed at efficient computation of camera external position and orientation relative to object reference coordinate system as well as the effective focal length, radial lens distortion, and image scanning parameters. The two-stage technique has advantage in terms of accuracy, speed, and versatility over existing state of the art. A critical review of the state of the art is given in the beginning. A theoretical framework is established, supported by comprehensive proof in five appendices, and may pave the way for future research on 3D robotics vision. Test results using real data are described. Both accuracy and speed are reported. The experimental results are analyzed and compared with theoretical prediction. Recent effort indicates that with slight modification, the two-stage calibration can be done in real time.

A Flexible New Technique for Camera Calibration

Zhengyou Zhang, Senior Member, IEEE

Abstract—We propose a flexible new technique to easily calibrate a camera. It only requires the camera to observe a planar pattern shown at a few (at least two) different orientations. Either the camera or the planar pattern can be freely moved. The motion need not be known. Radial lens distortion is modeled. The proposed procedure consists of a closed-form solution, followed by a nonlinear refinement based on the maximum likelihood criterion. Both computer simulation and real data have been used to test the proposed technique and very good results have been obtained. Compared with classical techniques which use expensive equipment such as two or three orthogonal planes, the proposed technique is easy to use and flexible. It advances 3D computer vision one more step from laboratory environments to real world use. The corresponding software is available from the author’s Web page.

Our current research is focused on a desktop vision system (DVS) since the potential for using DVSs is large. Cameras are becoming inexpensive and ubiquitous. A DVS aims at the general public who are not experts in computer vision. A typical computer user will perform vision tasks only from time to time, so they will not be willing to invest money for expensive equipment. Therefore, flexibility, robustness, and low cost are important. The camera calibration technique described in this paper was developed with these considerations in mind.

The proposed technique only requires the camera to observe a planar pattern shown at a few (at least two) different orientations. The pattern can be printed on a laser printer and attached to a “reasonable” planar surface (e.g., a hard book cover). Either the camera or the planar pattern can be moved by hand. The motion need not be known. The proposed approach, which uses 2D metric information, lies between the photogrammetric calibration, which uses explicit 3D model, and self-calibration, which uses motion rigidity or equivalently implicit 3D information. Both computer
2.1 Notation

A 2D point is denoted by \( \mathbf{m} = [u, v]^T \). A 3D point is denoted by \( \mathbf{M} = [X, Y, Z]^T \). We use \( \tilde{\mathbf{x}} \) to denote the augmented vector by adding 1 as the last element: \( \tilde{\mathbf{m}} = [u, v, 1]^T \) and \( \tilde{\mathbf{M}} = [X, Y, Z, 1]^T \). A camera is modeled by the usual pinhole: the relationship between a 3D point \( \mathbf{M} \) and its image projection \( \mathbf{m} \) is given by

\[
s\tilde{\mathbf{m}} = \mathbf{A} [\mathbf{R} \quad \mathbf{t}] \tilde{\mathbf{M}},
\]

where \( s \) is an arbitrary scale factor, \((\mathbf{R}, \mathbf{t})\), called the extrinsic parameters, is the rotation and translation which relates the world coordinate system to the camera coordinate system, and \( \mathbf{A} \), called the camera intrinsic matrix, is given by

\[
\mathbf{A} = \begin{bmatrix}
\alpha & \gamma & u_0 \\
0 & \beta & v_0 \\
0 & 0 & 1
\end{bmatrix}
\]

with \((u_0, v_0)\) the coordinates of the principal point, \(\alpha\) and \(\beta\) the scale factors in image \(u\) and \(v\) axes, and \(\gamma\) the parameter describing the skewness of the two image axes.

We use the abbreviation \(\mathbf{A}^{-T}\) for \((\mathbf{A}^{-1})^T\) or \((\mathbf{A}^T)^{-1}\).
2.2 Homography between the model plane and its image

Without loss of generality, we assume the model plane is on $Z = 0$ of the world coordinate system. Let's denote the $i^{th}$ column of the rotation matrix $R$ by $r_i$. From (1), we have

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = A \begin{bmatrix} r_1 & r_2 & r_3 & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix}$$

$$= A \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}.$$

By abuse of notation, we still use $\tilde{M}$ to denote a point on the model plane, but $\tilde{M} = [X, Y]^T$ since $Z$ is always equal to 0. In turn, $\tilde{M} = [X, Y, 1]^T$. Therefore, a model point $\tilde{M}$ and its image $\tilde{m}$ is related by a homography $H$:

$$s\tilde{m} = H\tilde{M} \quad \text{with} \quad H = A \begin{bmatrix} r_1 & r_2 & t \end{bmatrix}.$$  \hspace{1cm} (2)

As is clear, the $3 \times 3$ matrix $H$ is defined up to a scale factor.
A Estimation of the Homography Between the Model Plane and its Image

There are many ways to estimate the homography between the model plane and its image. Here, we present a technique based on maximum likelihood criterion. Let \( M_i \) and \( m_i \) be the model and image points, respectively. Ideally, they should satisfy (2). In practice, they don’t because of noise in the extracted image points. Let’s assume that \( m_i \) is corrupted by Gaussian noise with mean \( 0 \) and covariance matrix \( \Lambda_{m_i} \). Then, the maximum likelihood estimation of \( H \) is obtained by minimizing the following functional

\[
\sum_i (m_i - \hat{m}_i)^T \Lambda_{m_i}^{-1} (m_i - \hat{m}_i),
\]

where

\[
\hat{m}_i = \frac{1}{h_3^T M_i} \begin{bmatrix} \bar{h}_1^T M_i \\ \bar{h}_2^T M_i \end{bmatrix} \quad \text{with} \quad \bar{h}_i, \text{the } i^{th} \text{ row of } H.
\]

In practice, we simply assume \( \Lambda_{m_i} = \sigma^2 I \) for all \( i \). This is reasonable if points are extracted independently with the same procedure. In this case, the above problem becomes a nonlinear least-squares one, i.e., \( \min_H \sum_i \|m_i - \hat{m}_i\|^2 \). The nonlinear minimization is conducted with the Levenberg-Marquardt Algorithm as implemented in Minpack [18]. This requires an initial guess, which can be obtained as follows.

Let \( x = [\bar{h}_1^T, \bar{h}_2^T, h_3^T]^T \). Then equation (2) can be rewritten as

\[
\begin{bmatrix}
\bar{M}^T & 0^T & -u\bar{M}^T \\
0^T & \bar{M}^T & -v\bar{M}^T
\end{bmatrix} x = 0.
\]

When we are given \( n \) points, we have \( n \) above equations, which can be written in matrix equation as \( Lx = 0 \), where \( L \) is a \( 2n \times 9 \) matrix. As \( x \) is defined up to a scale factor, the solution is well known to be the right singular vector of \( L \) associated with the smallest singular value (or equivalently, the eigenvector of \( L^T L \) associated with the smallest eigenvalue).

In \( L \), some elements are constant \( 1 \), some are in pixels, some are in world coordinates, and some are multiplication of both. This makes \( L \) poorly conditioned numerically. Much better results can be obtained by performing a simple data normalization, such as the one proposed in [12], prior to running the above procedure.
Basic Equations

2.3 Constraints on the intrinsic parameters

Given an image of the model plane, an homography can be estimated (see Appendix A). Let’s denote it by \( \mathbf{H} = \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix} \). From (2), we have

\[
\begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix} = \lambda \mathbf{A} \begin{bmatrix} r_1 & r_2 & t \end{bmatrix},
\]

where \( \lambda \) is an arbitrary scalar. Using the knowledge that \( r_1 \) and \( r_2 \) are orthonormal, we have

\[
\begin{align*}
    h_1^T \mathbf{A}^{-T} \mathbf{A}^{-1} h_2 &= 0 \\
    h_1^T \mathbf{A}^{-T} \mathbf{A}^{-1} h_1 &= h_2^T \mathbf{A}^{-T} \mathbf{A}^{-1} h_2.
\end{align*}
\]

These are the two basic constraints on the intrinsic parameters, given one homography. Because a homography has 8 degrees of freedom and there are 6 extrinsic parameters (3 for rotation and 3 for translation), we can only obtain 2 constraints on the intrinsic parameters. Note that \( \mathbf{A}^{-T} \mathbf{A}^{-1} \) actually describes the image of the absolute conic [16]. In the next subsection, we will give an geometric interpretation.
Solving Camera Calibration

3.1 Closed-form solution

Let

\[
\mathbf{B} = \mathbf{A}^{-T} \mathbf{A}^{-1} = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{12} & B_{22} & B_{23} \\ B_{13} & B_{23} & B_{33} \end{bmatrix}
\]

\[
= \begin{bmatrix}
\frac{1}{\alpha^2} - \frac{\gamma}{\alpha^2 \beta} & -\frac{\gamma}{\alpha^2 \beta} + \frac{1}{\beta^2} & -\frac{v_0 v_0 - u_0 \beta}{\alpha^2 \beta^2} - \frac{v_0}{\beta^2} \\
-\frac{\gamma}{\alpha^2 \beta} - \frac{\gamma}{\alpha^2 \beta^2} + \frac{1}{\beta^2} & \frac{\gamma}{\alpha^2 \beta^2} - \frac{v_0 v_0 - u_0 \beta}{\alpha^2 \beta^2} - \frac{v_0}{\beta^2} \\
-\frac{v_0 v_0 - u_0 \beta}{\alpha^2 \beta^2} - \frac{v_0}{\beta^2} & -\frac{v_0 v_0 - u_0 \beta}{\alpha^2 \beta^2} + \frac{v_0^2}{\beta^2} + 1
\end{bmatrix}.
\]  

(5)

Note that \( \mathbf{B} \) is symmetric, defined by a 6D vector

\[
\mathbf{b} = [B_{11}, B_{12}, B_{22}, B_{13}, B_{23}, B_{33}]^T.
\]  

(6)

Let the \( i^{th} \) column vector of \( \mathbf{H} \) be \( \mathbf{h}_i = [h_{i1}, h_{i2}, h_{i3}]^T \). Then, we have

\[
\mathbf{h}_i^T \mathbf{B} \mathbf{h}_j = \mathbf{v}_{ij}^T \mathbf{b}
\]  

(7)

with

\[
\mathbf{v}_{ij} = [h_{i1} h_{j1}, h_{i1} h_{j2} + h_{i2} h_{j1}, h_{i2} h_{j2},
\]

\[
h_{i3} h_{j1} + h_{i1} h_{j3}, h_{i3} h_{j2} + h_{i2} h_{j3}, h_{i3} h_{j3}]^T.
\]
Solving Camera Calibration

Therefore, the two fundamental constraints (3) and (4), from a given homography, can be rewritten as 2 homogeneous equations in $b$:

$$
egin{bmatrix}
    v_{12}^T \\
    (v_{11} - v_{22})^T
\end{bmatrix}
\begin{bmatrix}
    b
\end{bmatrix} = 0 .
$$

If $n$ images of the model plane are observed, by stacking $n$ such equations as (8) we have

$$
Vb = 0 ,
$$

where $V$ is a $2n \times 6$ matrix. If $n \geq 3$, we will have in general a unique solution $b$ defined up to a scale factor. If $n = 2$, we can impose the skewless constraint $\gamma = 0$, i.e., $[0, 1, 0, 0, 0, 0]b = 0$, which is added as an additional equation to (9). (If $n = 1$, we can only solve two camera intrinsic parameters, e.g., $\alpha$ and $\beta$, assuming $u_0$ and $v_0$ are known (e.g., at the image center) and $\gamma = 0$, and that is indeed what we did in [19] for head pose determination based on the fact that eyes and mouth are reasonably coplanar.) The solution to (9) is well known as the eigenvector of $V^T V$ associated with the smallest eigenvalue (equivalently, the right singular vector of $V$ associated with the smallest singular value).
B Extraction of the Intrinsic Parameters from Matrix B

Matrix B, as described in Sect. 3.1, is estimated up to a scale factor, i.e., \( B = \lambda A^{-\top} A \) with \( \lambda \) an arbitrary scale. Without difficulty\(^\dagger\), we can uniquely extract the intrinsic parameters from matrix B.

\[
\begin{align*}
v_0 &= (B_{12}B_{13} - B_{11}B_{23})/(B_{11}B_{22} - B_{12}^2) \\
\lambda &= B_{33} - [B_{13}^2 + v_0(B_{12}B_{13} - B_{11}B_{23})]/B_{11} \\
\alpha &= \sqrt{\lambda/B_{11}} \\
\beta &= \sqrt{\lambda B_{11}/(B_{11}B_{22} - B_{12}^2)} \\
\gamma &= -B_{12}\alpha^2\beta/\lambda \\
u_0 &= \gamma v_0/\beta - B_{13}\alpha^2/\lambda .
\end{align*}
\]
Once $b$ is estimated, we can compute all camera intrinsic matrix $A$. See Appendix B for the details.

Once $A$ is known, the extrinsic parameters for each image is readily computed. From (2), we have

$$r_1 = \lambda A^{-1} h_1$$
$$r_2 = \lambda A^{-1} h_2$$
$$r_3 = r_1 \times r_2$$
$$t = \lambda A^{-1} h_3$$

with $\lambda = 1/\|A^{-1} h_1\| = 1/\|A^{-1} h_2\|$. Of course, because of noise in data, the so-computed matrix $R = [r_1, r_2, r_3]$ does not in general satisfy the properties of a rotation matrix. Appendix C describes a method to estimate the best rotation matrix from a general $3 \times 3$ matrix.
### 3.2 Maximum likelihood estimation

The above solution is obtained through minimizing an algebraic distance which is not physically meaningful. We can refine it through maximum likelihood inference.

We are given \( n \) images of a model plane and there are \( m \) points on the model plane. Assume that the image points are corrupted by independent and identically distributed noise. The maximum likelihood estimate can be obtained by minimizing the following functional:

\[
\sum_{i=1}^{n} \sum_{j=1}^{m} \| \mathbf{m}_{ij} - \hat{\mathbf{m}}(\mathbf{A}, \mathbf{R}_i, \mathbf{t}_i, \mathbf{M}_j) \|^2 ,
\]  

where \( \hat{\mathbf{m}}(\mathbf{A}, \mathbf{R}_i, \mathbf{t}_i, \mathbf{M}_j) \) is the projection of point \( \mathbf{M}_j \) in image \( i \), according to equation (2). A rotation \( \mathbf{R} \) is parameterized by a vector of 3 parameters, denoted by \( \mathbf{r} \), which is parallel to the rotation axis and whose magnitude is equal to the rotation angle. \( \mathbf{R} \) and \( \mathbf{r} \) are related by the Rodrigues formula [5]. Minimizing (10) is a nonlinear minimization problem, which is solved with the Levenberg-Marquardt Algorithm as implemented in Minpack [18]. It requires an initial guess of \( \mathbf{A}, \{\mathbf{R}_i, \mathbf{t}_i | i = 1..n\} \) which can be obtained using the technique described in the previous subsection.
3.3 Dealing with radial distortion

Up to now, we have not considered lens distortion of a camera. However, a desktop camera usually exhibits significant lens distortion, especially radial distortion. In this section, we only consider the first two terms of radial distortion. The reader is referred to [20, 2, 4, 26] for more elaborated models. Based on the reports in the literature [2, 23, 25], it is likely that the distortion function is totally dominated by the radial components, and especially dominated by the first term. It has also been found that any more elaborated modeling not only would not help (negligible when compared with sensor quantization), but also would cause numerical instability [23, 25].

Let \((u, v)\) be the ideal (nonobservable distortion-free) pixel image coordinates, and \((\check{u}, \check{v})\) the corresponding real observed image coordinates. The ideal points are the projection of the model points according to the pinhole model. Similarly, \((\check{x}, \check{y})\) and \((\check{x}, \check{y})\) are the ideal (distortion-free) and real (distorted) normalized image coordinates. We have [2, 25]

\[
\check{x} = x + x[k_1(x^2 + y^2) + k_2(x^2 + y^2)^2]
\]

\[
\check{y} = y + y[k_1(x^2 + y^2) + k_2(x^2 + y^2)^2],
\]

where \(k_1\) and \(k_2\) are the coefficients of the radial distortion. The center of the radial distortion is the same as the principal point. From* \(\check{u} = u_0 + \alpha \check{x} + \gamma \check{y}\) and \(\check{v} = v_0 + \beta \check{y}\) and assuming \(\gamma = 0\), we have

\[
\check{u} = u + (u - u_0)[k_1(x^2 + y^2) + k_2(x^2 + y^2)^2] \tag{11}
\]

\[
\check{v} = v + (v - v_0)[k_1(x^2 + y^2) + k_2(x^2 + y^2)^2]. \tag{12}
\]
Solving Camera Calibration

**Estimating Radial Distortion by Alternation.** As the radial distortion is expected to be small, one would expect to estimate the other five intrinsic parameters, using the technique described in Sect. 3.2, reasonable well by simply ignoring distortion. One strategy is then to estimate $k_1$ and $k_2$ after having estimated the other parameters, which will give us the ideal pixel coordinates $(u, v)$. Then, from (11) and (12), we have two equations for each point in each image:

$$
\begin{bmatrix}
(u-u_0)(x^2+y^2) & (u-u_0)(x^2+y^2)^2 \\
(v-v_0)(x^2+y^2) & (v-v_0)(x^2+y^2)^2 
\end{bmatrix}
\begin{bmatrix}
k_1 \\
k_2
\end{bmatrix}
= \begin{bmatrix}
\tilde{u}-u \\
\tilde{v}-v
\end{bmatrix}.
$$

Given $m$ points in $n$ images, we can stack all equations together to obtain in total $2mn$ equations, or in matrix form as $Dk = d$, where $k = [k_1, k_2]^T$. The linear least-squares solution is given by

$$
k = (D^T D)^{-1} D^T d. \quad (13)
$$

Once $k_1$ and $k_2$ are estimated, one can refine the estimate of the other parameters by solving (10) with $\hat{m}(A, R_i, t_i, M_j)$ replaced by (11) and (12). We can alternate these two procedures until convergence.
Solving Camera Calibration

**Complete Maximum Likelihood Estimation.** Experimentally, we found the convergence of the above alternation technique is slow. A natural extension to (10) is then to estimate the complete set of parameters by minimizing the following functional:

$$
\sum_{i=1}^{n} \sum_{j=1}^{m} \left\| \mathbf{m}_{i,j} - \hat{\mathbf{m}}(\mathbf{A}, k_1, k_2, \mathbf{R}_i, \mathbf{t}_i, \mathbf{M}_j) \right\|^2,
$$

(14)

where $\hat{\mathbf{m}}(\mathbf{A}, k_1, k_2, \mathbf{R}_i, \mathbf{t}_i, \mathbf{M}_j)$ is the projection of point $\mathbf{M}_j$ in image $i$ according to equation (2), followed by distortion according to (11) and (12). This is a nonlinear minimization problem, which is solved with the Levenberg-Marquardt Algorithm as implemented in Minpack [18]. A rotation is again parameterized by a 3-vector $\mathbf{r}$, as in Sect. 3.2. An initial guess of $\mathbf{A}$ and $\{\mathbf{R}_i, \mathbf{t}_i | i = 1..n\}$ can be obtained using the technique described in Sect. 3.1 or in Sect. 3.2. An initial guess of $k_1$ and $k_2$ can be obtained with the technique described in the last paragraph, or simply by setting them to 0.
3.4 Summary

The recommended calibration procedure is as follows:

1. Print a pattern and attach it to a planar surface;
2. Take a few images of the model plane under different orientations by moving either the plane or the camera;
3. Detect the feature points in the images;
4. Estimate the five intrinsic parameters and all the extrinsic parameters using the closed-form solution as described in Sect. 3.1;
5. Estimate the coefficients of the radial distortion by solving the linear least-squares (13);
6. Refine all parameters by minimizing (14).
Camera Calibration with One-Dimensional Objects

Zhengyou Zhang, Senior Member, IEEE

Abstract—Camera calibration has been studied extensively in computer vision and photogrammetry and the proposed techniques in the literature include those using 3D apparatus (two or three planes orthogonal to each other or a plane undergoing a pure translation, etc.), 2D objects (planar patterns undergoing unknown motions), and 0D features (self-calibration using unknown scene points). Yet, this paper proposes a new calibration technique using 1D objects (points aligned on a line), thus filling the missing dimension in calibration. In particular, we show that camera calibration is not possible with free-moving 1D objects, but can be solved if one point is fixed. A closed-form solution is developed if six or more observations of such a 1D object are made. For higher accuracy, a nonlinear technique based on the maximum likelihood criterion is then used to refine the estimate. Singularities have also been studied. Besides the theoretical aspect, the proposed technique is also important in practice especially when calibrating multiple cameras mounted apart from each other, where the calibration objects are required to be visible simultaneously.
Zhengyou Zhang

Microsoft Corp., One Microsoft Way, Redmond WA 98052-6399, USA
zhanyu@microsoft.com


Pattern Recognition Special Issue on "Sparse representation for event recognition in video surveillance". Deadline of submission: March 31, 2012.


Principal Researcher at Microsoft Research (MSR), Redmond, USA. My research is in computer vision, speech signal processing, multi-sensory fusion, multimedia computing, real-time collaboration and human-machine interaction. I manage the Multimedia, Interaction, and Communication (MIC) Group. I was affiliated with the Communication and Collaboration Systems Group, and the Speech Technology Group.

Research interests

- Computer vision and graphics: calibration, matching, stereo, motion, 3D modeling, 3D display
- Audio processing and rendering, speech processing, spatial audio, multichannel AEC
- Audio-visual fusion, active object detection and tracking
- Multimedia, human-computer interaction, human-human communication and collaboration
- Biology-inspired learning, autonomous mental development
- Human information processing: face/speaker recognition/verification, activity recognition and understanding

Full publication list is available through Google Scholar.

Recent publications and some downloadable papers are available from here.

Please visit my home page at INRIA for information prior to my arrival (March 30, 1998) at Microsoft Research.
OpenCV

- [OpenCV documentation](http://opencv.willowgarage.com/documentation/camera_calibration_and_3d_reconstruction.html)

Camera Calibration and 3D Reconstruction

The functions in this section use the so-called pinhole camera model. That is, a scene view is formed by projecting 3D points into the image plane using a perspective transformation.

\[
\begin{bmatrix}
    x' \\
    y'
\end{bmatrix} = A[R]t [X]
\]

Where \((X, Y, Z)\) are the coordinates of a 3D point in the world coordinate space, \((u, v)\) are the coordinate camera matrix, or a matrix of intrinsic parameters \((cx, cy)\) is a principal point (that is usually at the im expressed in pixel-related units). Thus, if an image from camera is scaled by some factor, all of these parameters are scaled respectively by the same factor. The matrix of intrinsic parameters does not depend on the scene viewed at the focal length is fixed (in case of zoom lens). The joint rotation-translation matrix \([M]\) is called a matrix of camera motion around a static scene, or vice versa, rigid motion of an object in front of still camera. This is \([M]\) to some coordinate system, fixed with respect to the camera. The transformation above is equivalent to the following projection:

\[
\begin{bmatrix}
    X' \\
    Y' \\
    Z'
\end{bmatrix} = R \begin{bmatrix}
    X \\
    Y \\
    Z
\end{bmatrix} + t
\]

\[
x' = x/z \\
y' = y/z \\
u = fx' + u' \\
v = fy' + v'
\]

Real lenses usually have some distortion, mostly radia distortion and slight tangential distortion. So, the above transform does not hold for these cases and the distortion correction is used in CV.

[OpenCV documentation](http://opencv.willowgarage.com/documentation/camera_calibration_and_3d_reconstruction.html)
Camera Calibration Toolbox

http://www.vision.caltech.edu/bouguetj/calib_doc/

Camera Calibration Toolbox for Matlab

This is a release of a Camera Calibration Toolbox for Matlab® with a complete documentation. This document may also be used as a tutorial on camera calibration since it includes general information about calibration, references and related links.
Please report bugs/questions/suggestions to Jean-Yves Bouguet at bouguet at gmail dot com.

The implementation of this toolbox is included in the Open Source Computer Vision library distributed by Intel and freely available online.

Content:
- System requirements
- Getting started
- Calibration examples
- Description of the calibration parameters
- Description of the functions in the calibration toolbox
- Doing your own calibration
- Undocumented features of the toolbox
- References
Camera Calibration

Calibration results after optimization (with uncertainties):

Focal Length: \( f_c = [661.67001, 662.82958] \pm [1.17913, 1.26567] \)
Principal point: \( c_c = [386.09590, 248.78987] \pm [2.38443, 2.17401] \)
Skew: \( \alpha_c = [0.00000] \pm [0.00000] \) \( \Rightarrow \) angle of pixel axes = 90.00000 ± 0.00000 degrees
Distortion: \( k_c = [-0.26425, 0.22645, 0.00020, 0.00023, 0.00000] \pm [0.00034, 0.00326, 0.00052, 0.00053, 0.00000] \)
Pixel error: \( err = [0.45330, 0.38916] \)
Calibrating a Stereo System

**Stereo calibration parameters after loading the individual calibration files:**

**Intrinsic parameters of left camera:**

- **Focal Length:** \( f_{\text{left}} = [533.00371, 533.15260] \pm [1.07629, 1.10913] \)
- **Principal point:** \( c_{\text{left}} = [341.58812, 234.25940] \pm [1.24041, 1.33005] \)
- **Skew:** \( \alpha_c_{\text{left}} = [0.00000] \pm [0.00000] \Rightarrow \text{angle of pixel axes} = 90.00000 \pm 0.00000 \text{ degrees} \)
- **Distortion:** \( k_{\text{left}} = [-0.28947, 0.10326, 0.00103, -0.00029, 0.00000] \pm [0.00596, 0.02055, 0.00030, 0.00037, 0.00000] \)

**Intrinsic parameters of right camera:**

- **Focal Length:** \( f_{\text{right}} = [536.98262, 536.56938] \pm [1.19786, 1.15677] \)
- **Principal point:** \( c_{\text{right}} = [326.47209, 240.32527] \pm [1.36588, 1.34252] \)
- **Skew:** \( \alpha_c_{\text{right}} = [0.00000] \pm [0.00000] \Rightarrow \text{angle of pixel axes} = 90.00000 \pm 0.00000 \text{ degrees} \)
- **Distortion:** \( k_{\text{right}} = [-0.28936, 0.10677, 0.00078, 0.00029, 0.00000] \pm [0.00488, 0.00866, 0.00027, 0.00062, 0.00000] \)

**Extrinsic parameters (position of right camera w.r.t left camera):**

- **Rotation vector:** \( \text{on} = [0.00411, 0.00499, -0.00559] \)
- **Translation vector:** \( T = [-99.84929, 0.82221, 0.435647] \)
Robust Multi-camera Calibration

Multi-Camera Self Calibration


Matlab package for a complete and fully automatic calibration of multi-camera setups (3 cams min). A standard laser pointer is the only hardware you need. No calibration object and user interaction required.

**Keywords:** multi-camera calibration, multicamera calibration, self-calibration, multi-camera calibration, calibration of a camera network.

**Authors of the code**
- **Tomas Svooboda.** Corresponding author. Design of the package, Euclidean stratification, Finding points, I/O operations, interfacing, robust reconstruction for calib validations ...
- **Daniel Martinez** and **Tomas Pajdla.** Filling points in projective reconstruction via rank-4 factorization.
- **Jean-Yves Bouguet.** Radial distortion routines.
- **Tomas Werner.** Projective Bundle Adjustment.
- **Onkel Chum.** RANSAC implementation

**History**
- **February, 2011.** Code slightly modified and made Octave compatible by Andrew Straw and his collaborators. See the README for more details. Thanks Andrew!
- **May 24, 2002.** Version 1.0 released.
- **October 29, 2004.** Samples data available for download.
- **July 16, 2004.** Our journal paper accepted.
- **20 August, 2003.** Documentation upgrade.
Multi-Camera Self Calibration

Problem definition:
From $u^i_j$ points, for which $\lambda^i_j u^i_j = P^i X_j$ holds, estimate Euclidean projection matrices $P^i$ and coordinates of the 3D points $X_j$. 
ARToolKit Camera Calibration

**Accurate two-step method (2/3)**
- **Step 1:** Getting distortion parameters: `calib_dist`
  - Selecting dots with mouse
  - Getting distortion parameters by automatic line-fitting
  - Take pattern pictures as large as possible
  - Slant in various directions with big angle
  - 4 times or more

**Accurate two-step method (3/3)**
- **Step 2:** Getting perspective projection matrix: `calib_cparam`
  - Manual line-fitting
  - Grid cardboard box to be moved in the perpendicular direction of the plane. Camera should be placed in almost perpendicular direction of the plane.

**Easy one-step method: `calib_camera2`**
- Same operation as `calib_dist`
- Getting all camera parameters including distortion parameters and perspective projection matrix
- Not require careful setup
- Accuracy is good enough for image overlay
  - [But, Not good enough for 3D measurement.]

**Camera Parameter Implementation**
- Camera parameter structure
  ```c
  typedef struct {
    int width, height;
    double mat[1][4];
    double dist_coeffs[4];
  } ARParam;
  ```
- Adjust camera parameter for the input image size
  ```c
  int arParamChangeSize(ARParam *source, int new_width, int new_height, ARParam *newparam);
  ```
- Read camera parameters from the file
  ```c
  int arParamLoad(char *filename, int num, ARParam *param, ...);
  ```
OpenCV

http://opencv.willowgarage.com/documentation/camera_calibration_and_3d_reconstruction.html
OpenCV 2.1 Camera Calibration

Camera Calibration and 3d Reconstruction

The functions in this section use the so-called pinhole camera model. That is, a scene view is formed by projecting 3D points into the image plane using a perspective transformation.

\[ s m' = A[R|t]M' \]

or

\[
\begin{bmatrix}
    u \\
    v \\
    1
\end{bmatrix} =
\begin{bmatrix}
    f_x & 0 & c_x \\
    0 & f_y & c_y \\
    0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    r_{11} & r_{12} & r_{13} & t_1 \\
    r_{21} & r_{22} & r_{23} & t_2 \\
    r_{31} & r_{32} & r_{33} & t_3
\end{bmatrix}
\begin{bmatrix}
    X \\
    Y \\
    Z \\
    1
\end{bmatrix}
\]

Where \((X, Y, Z)\) are the coordinates of a 3D point in the world coordinate space, \((u, v)\) are the coordinates of the projection point in pixels. \(A\) is called a camera matrix, or a matrix of intrinsic parameters. \((c_x, c_y)\) is a principal point (that is usually at the image center), and \(f_x, f_y\) are the focal lengths expressed in pixel-related units. Thus, if an image from camera is scaled by some factor, all of these parameters should be scaled (multiplied/divided, respectively) by the same factor. The matrix of intrinsic parameters does not depend on the scene viewed and, once estimated, can be re-used (as long as the focal length is fixed (in case of zoom lens)). The joint rotation-translation matrix \([R|t]\) is called a matrix of extrinsic parameters. It is used to describe the camera motion around a static scene, or vice versa, rigid motion of an object in front of still camera. That is, \([R|t]\) translates coordinates of a point \((X, Y, Z)\) to some coordinate system, fixed with respect to the camera. The transformation above is equivalent to the following (when \(z \neq 0\)):
OpenCV 2.1 Camera Calibration

\[
\begin{bmatrix}
x \\
y \\
z \\
\end{bmatrix}
= R \begin{bmatrix}
X \\
Y \\
Z \\
\end{bmatrix} + t
\]

\[
x' = \frac{x}{z}
\]

\[
y' = \frac{y}{z}
\]

\[
u = f_x * x' + c_x
\]

\[
v = f_y * y' + c_y
\]

Real lenses usually have some distortion, mostly radial distortion and slight tangential distortion. So, the above model is extended as:

\[
\begin{bmatrix}
x \\
y \\
z \\
\end{bmatrix}
= R \begin{bmatrix}
X \\
Y \\
Z \\
\end{bmatrix} + t
\]

\[
x' = \frac{x}{z}
\]

\[
y' = \frac{y}{z}
\]

\[
x'' = x' \left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6 \right) + 2 p_1 x' y' + p_2 (r^2 + 2 x'^2)
\]

\[
y'' = y' \left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6 \right) + p_1 (r^2 + 2 y'^2) + 2 p_2 x' y'
\]

where \( r^2 = x'^2 + y'^2 \)

\[
u = f_x * x'' + c_x
\]

\[
v = f_y * y'' + c_y
\]

\(k_1, k_2, k_3\) are radial distortion coefficients, \(p_1, p_2\) are tangential distortion coefficients. Higher-order coefficients are not considered in OpenCV. In the functions below the coefficients are passed or returned as
Using the Polynomial Distortion Model to Correct Tangential Distortion

The polynomial distortion model uses two parameters, $P_1$ and $P_2$, to characterize tangential distortion. The distortion model for tangential distortion can be represented as:

$$x_{\text{corrected}} = x + [2P_1xy + P_2(r^2 + 2x^2)]$$

$$y_{\text{corrected}} = y + [P_1(r^2 + 2y^2) + 2P_2xy]$$

OpenCV 2.1 Camera Calibration

$k_1$, $k_2$, $k_3$ are radial distortion coefficients, $p_1$, $p_2$ are tangential distortion coefficients. Higher-order coefficients are not considered in OpenCV. In the functions below the coefficients are passed or returned as

$$(k_1, k_2, p_1, p_2[, k_3])$$

vector. That is, if the vector contains 4 elements, it means that $k_3 = 0$. The distortion coefficients do not depend on the scene viewed, thus they also belong to the intrinsic camera parameters. *And they remain the same regardless of the captured image resolution.* That is, if, for example, a camera has been calibrated on images of $320 \times 240$ resolution, absolutely the same distortion coefficients can be used for images of $640 \times 480$ resolution from the same camera (while $f_x$, $f_y$, $c_x$ and $c_y$ need to be scaled appropriately).

The functions below use the above model to

- Project 3D points to the image plane given intrinsic and extrinsic parameters
- Compute extrinsic parameters given intrinsic parameters, a few 3D points and their projections.
- Estimate intrinsic and extrinsic camera parameters from several views of a known calibration pattern (i.e. every view is described by several 3D–2D point correspondences).
- Estimate the relative position and orientation of the stereo camera “heads” and compute the *rectification* transformation that makes the camera optical axes parallel.
double calibrateCamera(const vector<vector<Point3f>> &objectPoints, const vector<vector<Point2f>> &imagePoints, Size imageSize, Mat& cameraMatrix, Mat& distCoeffs, vector<Mat>& rvecs, vector<Mat>& tvecs, int flags=0)

Finds the camera intrinsic and extrinsic parameters from several views of a calibration pattern.

Parameters:
• objectPoints – The vector of vectors of points on the calibration pattern in its coordinate system, one vector per view. If the same calibration pattern is shown in each view and it’s fully visible then all the vectors will be the same, although it is possible to use partially occluded patterns, or even different patterns in different views – then the vectors will be different. The points are 3D, but since they are in the pattern coordinate system, then if the rig is planar, it may have sense to put the model to the XY coordinate plane, so that Z-coordinate of each input object point is 0
• imagePoints – The vector of vectors of the object point projections on the calibration pattern views, one vector per a view. The projections must be in the same order as the corresponding object points.
• imageSize – Size of the image, used only to initialize the intrinsic camera matrix
• cameraMatrix – The output 3x3 floating-point camera matrix

\[
A = \begin{bmatrix}
    f_x & 0 & c_x \\
    0 & f_y & c_y \\
    0 & 0 & 1
\end{bmatrix}
\]

If CV_CALIB_USE_INTRINSIC_GUESS and/or CV_CALIB_FIX_ASPECT_RATIO are specified, some or all of fx, fy, cx, cy must be initialized before calling the function
• distCoeffs – The output 5x1 or 1x5 vector of distortion coefficients (k_1, k_2, p_1, p_2[, k_3])
calibrateCamera

Parameters:

- `rvecs` - The output vector of rotation vectors (see Rodrigues2), estimated for each pattern view. That is, each k-th rotation vector together with the corresponding k-th translation vector (see the next output parameter description) brings the calibration pattern from the model coordinate space (in which object points are specified) to the world coordinate space, i.e. real position of the calibration pattern in the k-th pattern view (k=0..M-1)
- `tvecs` - The output vector of translation vectors, estimated for each pattern view.
- `flags` - Different flags, may be 0 or combination of the following values:
  - `CV_CALIB_USE_INTRINSIC_GUESS` cameraMatr ix contains the valid initial values of fx, fy, cx, cy that are optimized further. Otherwise, (cx, cy) is initially set to the image center (imageSize is used here), and focal distances are computed in some least-squares fashion. Note, that if intrinsic parameters are known, there is no need to use this function just to estimate the extrinsic parameters. Use FindExtrinsicCameraParams2 instead.
  - `CV_CALIB_FIX_PRINCIPAL_POINT` The principal point is not changed during the global optimization, it stays at the center or at the other location specified when `CV_CALIB_USE_INTRINSIC_GUESS` is set too.
  - `CV_CALIB_FIX_ASPECT_RATIO` The functions considers only fy as a free parameter, the ratio fx/fy stays the same as in the input cameraMatrix. When `CV_CALIB_USE_INTRINSIC_GUESS` is not set, the actual input values of fx and fy are ignored, only their ratio is computed and used further.
  - `CV_CALIB_ZERO_TANGENT_DIST` Tangential distortion coefficients (p1, p2) will be set to zeros and stay zero.
The function estimates the intrinsic camera parameters and extrinsic parameters for each of the views. The coordinates of 3D object points and their correspondent 2D projections in each view must be specified. That may be achieved by using an object with known geometry and easily detectable feature points. Such an object is called a calibration rig or calibration pattern, and OpenCV has built-in support for a chessboard as a calibration rig (see `FindChessboardCorners()`). Currently, initialization of intrinsic parameters (when CV_CALIB_USE_INTRINSIC_GUESS is not set) is only implemented for planar calibration patterns (where z-coordinates of the object points must be all 0’s). 3D calibration rigs can also be used as long as initial cameraMatrix is provided.

The algorithm does the following:

1. First, it computes the initial intrinsic parameters (the option only available for planar calibration patterns) or reads them from the input parameters. The distortion coefficients are all set to zeros initially (unless some of CV_CALIB_FIX_Ki are specified).
2. The initial camera pose is estimated as if the intrinsic parameters have been already known. This is done using `FindExtrinsicCameraParams2`.
3. After that, the global Levenberg-Marquardt optimization algorithm is run to minimize the reprojection error, i.e., the total sum of squared distances between the observed feature points imagePoints and the projected (using the current estimates for camera parameters and the poses) object points objectPoints; see `ProjectPoints2`.

The function returns the final re-projection error. Note: if you’re using a non-square (=non-NxN) grid and `findChessboardCorners()` for calibration, and `calibrateCamera` returns bad values (i.e. zero distortion coefficients, an image center very far from \((w/2 - 0.5, h/2 - 0.5)\), and / or large differences between \(f_x\) and \(f_y\) (ratios of 10:1 or more)), then you’ve probably used `patternSize=cvSize(rows,cols)` , but should use `patternSize=cvSize(cols,rows)` in `FindChessboardCorners`. 
Camera Calibration & AR@Home

lbg@dongseo.ac.kr
Capture chessboard image

- Chessboard no of row and col
- 10 : no of calibration images
- Auto : automatically add images after 3*150 milliseconds
- Start : start calibration
Calibration parameters

- Focal length \( f_x, f_y \)
- Center \( c_x, c_y \)
- Radial: \( k_1, k_2, k_3 \) (check box)
- Tangential: \( p_1, p_2 \)
- Undistort: Undistorted live image
### Images

<table>
<thead>
<tr>
<th>이름</th>
<th>크기</th>
<th>종류</th>
<th>수정한 날짜</th>
<th>사전 적은 날짜</th>
<th>그림 크기</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capture072309511500</td>
<td>901KB</td>
<td>알пись BMP 파일</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
<tr>
<td>Capture072309511601</td>
<td>901KB</td>
<td>알пись BMP 파일</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
<tr>
<td>Capture072309511802</td>
<td>901KB</td>
<td>알пись BMP 파일</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
<tr>
<td>Capture072309511903</td>
<td>901KB</td>
<td>알пись BMP 파일</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
<tr>
<td>Capture072309512004</td>
<td>901KB</td>
<td>알пись BMP 파일</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
<tr>
<td>Capture072309512205</td>
<td>901KB</td>
<td>알пись BMP 파일</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
<tr>
<td>Capture072309512306</td>
<td>901KB</td>
<td>알пись BMP 파일</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
<tr>
<td>Capture072309512407</td>
<td>901KB</td>
<td>알пись BMP 파일</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
<tr>
<td>Capture072309512508</td>
<td>901KB</td>
<td>알пись BMP 파일</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
<tr>
<td>Capture072309512609</td>
<td>901KB</td>
<td>알пись BMP 파일</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
<tr>
<td>Corner072309511500</td>
<td>901KB</td>
<td>알пись BMP 파일</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
<tr>
<td>Corner072309511601</td>
<td>901KB</td>
<td>알пись BMP 파일</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
<tr>
<td>Corner072309511802</td>
<td>901KB</td>
<td>알пись BMP 파일</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
<tr>
<td>Corner072309511903</td>
<td>901KB</td>
<td>알пись BMP 파일</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
<tr>
<td>Corner072309512004</td>
<td>901KB</td>
<td>알пись BMP 파일</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
<tr>
<td>Corner072309512205</td>
<td>901KB</td>
<td>알пись BMP 파일</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
<tr>
<td>Corner072309512306</td>
<td>901KB</td>
<td>알пись BMP 파일</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
<tr>
<td>Corner072309512407</td>
<td>901KB</td>
<td>알пись BMP 파일</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
<tr>
<td>Corner072309512508</td>
<td>901KB</td>
<td>알пись BMP 파일</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
<tr>
<td>Corner072309512609</td>
<td>901KB</td>
<td>알пись BMP 파일</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
<tr>
<td>Distortion0723095127</td>
<td>1KB</td>
<td>XML 문서</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
<tr>
<td>Image0723095127</td>
<td>17KB</td>
<td>XML 문서</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
<tr>
<td>Intrinsic0723095127</td>
<td>1KB</td>
<td>XML 문서</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
<tr>
<td>Object0723095127</td>
<td>15KB</td>
<td>XML 문서</td>
<td>2012-07-23 오전 ...</td>
<td>640 x 480</td>
<td></td>
</tr>
</tbody>
</table>
Filenames

- Capture072309511500
- month day hour min sec index
- Corner072309511500
- Image0723095115.xml
- Object0723095115.xml
- Distortion0723095115.xml
- Intrinsics0723095115.xml
Image & Object xml

```xml
<?xml version="1.0"?>
<opencv_storage>
<Image0723095127 type_id="opencv-matrix">
  <rows>480</rows>
  <cols>2</cols>
  <dt>f</dt>
  <data>
    2.25150803e+002 1.92720139e+002 2.42047302e+002 1.88396301e+002
    2.59520660e+002 1.83857040e+002 2.77741913e+002 1.79775314e+002
    2.96598541e+002 1.75898483e+002 3.15514801e+002 1.72283661e+002
  </data>
</Image0723095127 type_id="opencv-matrix">
</opencv_storage>

<?xml version="1.0"?>
<opencv_storage>
<Object0723095127 type_id="opencv-matrix">
  <rows>480</rows>
  <cols>3</cols>
  <dt>f</dt>
  <data>
    0. 1.37500000e+001 0. 2.75000000e+000 1.37500000e+001 0.
    5.50000000e+000 1.37500000e+001 0. 8.25000000e+000 1.37500000e+001
    0. 11. 1.37500000e+001 0. 1.37500000e+001 1.37500000e+001 0.
  </data>
</Object0723095127 type_id="opencv-matrix">
</opencv_storage>
```
Distortion & Intrinsics xml

```xml
<?xml version="1.0"?>
<opencv_storage>
  <Distortion0723095127 type_id="opencv-matrix">
    <rows>4</rows>
    <cols>1</cols>
    <dt>f</dt>
    <data>
      -4.46110874e-001 2.08227739e-001 9.08007473e-003 -3.17067979e-003
    </data>
  </Distortion0723095127>
</opencv_storage>

<?xml version="1.0"?>
<opencv_storage>
  <Intrinsics0723095127 type_id="opencv-matrix">
    <rows>3</rows>
    <cols>3</cols>
    <dt>f</dt>
    <data>
      2.66292694e+002 0. 3.24350769e+002 0. 2.58133820e+002 2.33235382e+002 0. 2.58133820e+002 0. 1.
    </data>
  </Intrinsics0723095127>
</opencv_storage>
```
bool CCamCalib::FindChessboard(IplImage *src, IplImage *dest)
{
    IplImage *gray = cvCreateImage(cvGetSize(src), IPL_DEPTH_8U, 1);
    cvCvtColor(src, gray, CV_BGR2GRAY);
    CvPoint2D32f* corners = new CvPoint2D32f[_no_corners_total];
    int corner_count = 0;
    int found = cvFindChessboardCorners(src, cvSize(_no_corners_width, _no_corners_height), corners, &corner_count, CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_FILTER_QUADS);
    cvFindCornerSubPix(gray, corners, corner_count, cvSize(11,11), cvSize(-1,-1), cvTermCriteria(CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, 30, 0.1));
    cvDrawChessboardCorners(dest, cvSize(_no_corners_width, _no_corners_height), corners, corner_count, found);
    bool ret = false;
    if (found && corner_count == _no_corners_total) {
        for( int i=success_count*_no_corners_total, j=0; j<_no_corners_total; ++i, ++j ) {
            CV_MAT_ELEM(*image_points, float, i, 0) = corners[j].x;
            CV_MAT_ELEM(*image_points, float, i, 1) = corners[j].y;
            CV_MAT_ELEM(*object_points,float, i, 0) =
                (float)((j%_no_corners_width)*_grid_width);
            CV_MAT_ELEM(*object_points,float, i, 1) =
                (float)((_no_corners_height - j/_no_corners_width - 1)*_grid_height);
            CV_MAT_ELEM(*object_points,float, i, 2) = 0.0f;
        }
        CV_MAT_ELEM(*point_counts, int, _success_count, 0) = _no_corners_total;
        ret = true;
    }
    delete[] corners;
    cvReleaseImage(&gray);
    return ret;
}
void CCamCalib::CalibrateCamera(CvSize &image_size, bool k3flag, bool debug)
{
    if(_intrinsic_matrix) cvReleaseMat(&_intrinsic_matrix);
    if(_distortion_coeffs) cvReleaseMat(&_distortion_coeffs);
    if(_mapx) cvReleaseImage(&_mapx);
    if(_mapy) cvReleaseImage(&_mapy);
    _intrinsic_matrix = cvCreateMat(3, 3, CV_32FC1);
    if(k3flag) _distortion_coeffs = cvCreateMat(5, 1, CV_32FC1);
    else _distortion_coeffs = cvCreateMat(4, 1, CV_32FC1);
    CV_MAT_ELEM(*_intrinsic_matrix, float, 0, 0 ) = 1.0f;
    CV_MAT_ELEM(*_intrinsic_matrix, float, 1, 1 ) = 1.0f;
    cvCalibrateCamera2(_object_points, _image_points, _point_counts,
                        image_size, _intrinsic_matrix, _distortion_coeffs, NULL, NULL, 0);
    _mapx = cvCreateImage(image_size, IPL_DEPTH_32F, 1);
    _mapy = cvCreateImage(image_size, IPL_DEPTH_32F, 1);
    cvInitUndistortMap(_intrinsic_matrix, _distortion_coeffs, _mapx, _mapy);
}

void CCamCalib::Undistort(IplImage *src, IplImage *dest)
{
    assert(_mapx);
    assert(_mapy);
    cvRemap(src, dest, _mapx, _mapy);
}
bool CCamCalib::FindExternalParameter(IplImage *src, IplImage *dest, bool flag)
{
    IplImage *gray = cvCreateImage(cvGetSize(src), IPL_DEPTH_8U, 1);
    cvCvtColor(src, gray, CV_BGR2GRAY);
    CvPoint2D32f* corners = new CvPoint2D32f[_no_corners_total];
    int corner_count = 0;
    int found = cvFindChessboardCorners(src, cvSize(_no_corners_width, _no_corners_height),
                                         corners, &corner_count, CV_CALIB_CB_ADAPTIVE_THRESH |
                                         CV_CALIB_CB_FILTER_QUADS);
    cvFindCornerSubPix(gray, corners, corner_count, cvSize(11,11),
                       cvSize(-1,-1), cvTermCriteria(CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, 30, 0.1));
    if(flag) cvDrawChessboardCorners(dest, cvSize(_no_corners_width, _no_corners_height),
                                      corners, corner_count, found);
    bool ret = false;
    if (found && corner_count == _no_corners_total) {
        for(int i=0; i<_no_corners_total; ++i) {
            CV_MAT_ELEM(*_image_points_for_external, float, i, 0) = corners[i].x;
            CV_MAT_ELEM(*_image_points_for_external, float, i, 1) = corners[i].y;
            CV_MAT_ELEM(*_object_points_for_external, float, i, 0) =
                (float)((float)((i%_no_corners_width-_no_corners_width-1.)/2)*_grid_width);
            CV_MAT_ELEM(*_object_points_for_external, float, i, 1) =
                (float)((float)((_no_corners_height-i/_no_corners_width-1.)*_no_corners_height-1.)/2)*_grid_height);
            CV_MAT_ELEM(*_object_points_for_external, float, i, 2) = 0.0f;
        }
        cvFindExtrinsicCameraParams2(_object_points_for_external, _image_points_for_external,
                                      _intrinsic_matrix, _distortion_coeffs, _rotation_vector_for_external,
                                      _translate_vector_for_external, 0);
        ret = true;
    }
    cvReleaseImage(&gray);
    delete [] corners;
    return ret;
}
DrawExternalParameter AR

```c
void CCamCalib::DrawExternalParameter(IplImage *src, IplImage *dest, double height)
{
    static int loop=0, inc=1;
    int loops=50;
    int radius = 3;
    int thickness = 2;
    int connectivity = 8;
    CvPoint pt, pt1;
    CvScalar green = CV_RGB(0,250,0);
    CvScalar blue = CV_RGB(0,0,250);
    CvScalar red = CV_RGB(250,0,0);
    int x, y;

    height+=(float)(loop-loops)/loops*5.;
    loop+=inc;
    if(loop==0) inc=1;
    else if(loop==2*loops) inc=-1;

    for(int i=0; i<_no_corners_total; ++i) {
        x=(int) (cvmGet(_image_points_for_external, i, 0)+.5);
        y=(int) (cvmGet(_image_points_for_external, i, 1)+.5);
        pt = cvPoint(x,y);
        cvCircle(dest,pt,radius,red,thickness,connectivity);
    }
}```
radius = 2;
cvProjectPoints2(_object_points_for_external,
_rotation_vector_for_external, _translate_vector_for_external,
_intrinsic_matrix, _distortion_coeffs, _image_points_for_external);

for(int i=0; i<_no_corners_total; ++i) 
CV_MAT_ELEM(*_object_points_for_external, float, i, 2) = height;

cvProjectPoints2(_object_points_for_external,
_rotation_vector_for_external, _translate_vector_for_external,
_intrinsic_matrix, _distortion_coeffs, _image_points_for_external_height);

for(int i=0; i<_no_corners_total; ++i) {
    x=(int) (cvmGet(_image_points_for_external_height, i, 0)+.5);
    y=(int) (cvmGet(_image_points_for_external_height, i, 1)+.5);
    pt = cvPoint(x,y);
    cvCircle(dest,pt,radius,blue,thickness,connectivity);

    x=(int) (cvmGet(_image_points_for_external, i, 0)+.5);
    y=(int) (cvmGet(_image_points_for_external, i, 1)+.5);
    pt1 = cvPoint(x,y);
    cvLine(dest,pt,pt1,green,1,connectivity);
}
for (int i = 0; i < _no_corners_total; ++i) {
    CV_MAT_ELEM(*_object_points_for_external, float, i, 0) = height*cos((float)i/_no_corners_total*2.*PI);
    CV_MAT_ELEM(*_object_points_for_external, float, i, 1) = height*sin((float)i/_no_corners_total*2.*PI);
    CV_MAT_ELEM(*_object_points_for_external, float, i, 2) = 0.;
}

cvProjectPoints2(_object_points_for_external,
_rotation_vector_for_external, _translate_vector_for_external,
_intrinsic_matrix, _distortion_coeffs, _image_points_for_external);

for (int i = 0; i < _no_corners_total-1; ++i) {
    x = (int) (cvmGet(_image_points_for_external, i, 0)+.5);
    y = (int) (cvmGet(_image_points_for_external, i, 1)+.5);
    pt = cvPoint(x, y);

    x = (int) (cvmGet(_image_points_for_external, i+1, 0)+.5);
    y = (int) (cvmGet(_image_points_for_external, i+1, 1)+.5);
    pt1 = cvPoint(x, y);
    cvLine(dest, pt, pt1, green, 1, connectivity);
}
Thanks you!!
lbg@dongseo.ac.kr
http://kowon.dongseo.ac.kr/~lbg/
Distortion Parameters

- Assumed perspective geometry: the perspective center, the object point, and the corresponding image point are collinear.
- During camera calibration, we try to compensate for all deviations from the assumed perspective geometry:
  - Radial Lens Distortion
  - Decentric Lens Distortion
  - Atmospheric Refraction
  - Affine Deformations
  - Out of Plane Deformations

Radial Lens Distortion

- The light ray changes its direction after passing through the perspective center.
- Radial lens distortion is caused by:
  - Large off-axial angle
  - Lens manufacturing flaws
- Radial lens distortion occurs along a radial direction from the principal point.
- Radial lens distortion increases as we move away from the principal point.

Pin Cushion Type Radial Lens Distortion
Radial Lens Distortion

Barrel Type Radial Lens Distortion

Radial Lens Distortion

Fiducial Center ≈ Principal Point

Radial Lens Distortion

\[ \Delta x = \Delta r \times \frac{(x - x_p)}{r} \]
\[ \Delta y = \Delta r \times \frac{(y - y_p)}{r} \]
Radial Lens Distortion

\[ \Delta x_{\text{Radial Lens Distortion}} = (x - x_p) (k_1 r^2 + k_2 r^4 + k_3 r^6 + \ldots) \]
\[ \Delta y_{\text{Radial Lens Distortion}} = (y - y_p) (k_1 r^2 + k_2 r^4 + k_3 r^6 + \ldots) \]

where: \( r = \sqrt{(x - x_p)^2 + (y - y_p)^2} \)^{0.5}

Decentric Lens Distortion

- Decentric lens distortion is caused by miss alignment of the components of the lens system.
- Decentric lens distortion has two components:
  - Radial component
  - Tangential component
Decentric Lens Distortion

\[ P(r) = J_1 r^2 + J_2 r^4 + \ldots \]
\[ \Delta r = 3 \ P(r) \sin(\phi - \phi_o) \]
\[ \Delta t = P(r) \cos(\phi - \phi_o) \]

- \( P(r) \) is the profile along the axis with the maximum tangential distortion.
- \( \phi_o \) is the direction of the axis with the maximum tangential distortion.

Decentric Lens Distortion

\[ \Delta x_{\text{Decentric Lens Distortion}} = (1 + p_3^2 r^2) \left( p_1 (r^2 + 2 \bar{x}^2) + 2 p_2 \bar{x} \bar{y} \right) \]
\[ \Delta y_{\text{Decentric Lens Distortion}} = (1 + p_3^2 r^2) \left( 2 p_1 \bar{x} \bar{y} + p_2 (r^2 + 2 \bar{y}^2) \right) \]

where: \( r = \left( (x - x_p)^2 + (y - y_p)^2 \right)^{0.5} \)
\[ \bar{x} = x - x_p \]
\[ \bar{y} = y - y_p \]

- \( p_1 = J_1 \sin \phi_o \)
- \( p_2 = J_1 \cos \phi_o \)
- \( p_3 = J_2 / J_1 \)