Computational Optical Imaging - Optique Numerique

-- Calibration and Features --

Winter 2013

Ivo Ihrke

with slides by Thorsten Thormaehlen
Estimating the Camera Matrix
3D Calibration target
DLT algorithm

- Have correspondences: \( x \leftrightarrow X \); unknown \( A \)

- Projecting a point: \( x = AX \)

- Performing the perspective division yields

\[
\begin{align*}
x &= \frac{a_{11}X + a_{12}Y + a_{13}Z + a_{14}}{a_{31}X + a_{32}Y + a_{33}Z + a_{34}} \\
y &= \frac{a_{21}X + a_{22}Y + a_{23}Z + a_{24}}{a_{31}X + a_{32}Y + a_{33}Z + a_{34}}
\end{align*}
\]

\[
\Rightarrow a_{31}XX + a_{32}XY + a_{33}XZ + a_{34}X - a_{11}X - a_{12}Y - a_{13}Z - a_{14} = 0 \\
a_{31}YY + a_{32}YY + a_{33}YZ + a_{34}Y - a_{21}X - a_{22}Y - a_{23}Z - a_{24} = 0
\]

\[
\begin{bmatrix}
-X & -Y & -Z & -1 & 0 & 0 & 0 & 0 & Xx & Yx & Zx & x \\
0 & 0 & 0 & 0 & -X & -Y & -Z & -1 & Xy & Yy & Zy & y \\
\vdots & & & & & & & & & & & \\
\end{bmatrix}
\begin{pmatrix}
a_{11} \\
a_{12} \\
a_{13} \\
a_{14} \\
a_{21} \\
\vdots \\
\end{pmatrix} = \mathbf{Ba} = \begin{pmatrix} 0 \\
0 \\
\vdots \end{pmatrix}
\]
DLT algorithm II

- Solve homogeneous system
  (avoid trivial solution $a=0$)

- Perform SVD:

- Extract lower-most row of $V^T$
  (right-most column of $V$)

- Yields least-squares optimal approximation to null-space

$$Ba = 0$$

$$UDV^T = \text{svd}(B)$$
3D Target – calibrated coordinate system

Plotted 3D points: (0,0,0), (7,0,0), (0,7,0), (0,0,-9)
3D Target – calibrated coordinate system

Plotted 3D points: (0,0,0), (7,0,0), (0,7,0), (0,0,-9)
Things to note:

- Computer vision coordinate systems are often left-handed
- OpenGL systems are right-handed
- Can simply load $A$ into matrix stack
  - BUT: coordinate system (and clipping) might behave strangely
  - Convert by “flip matrix”
Accuracy considerations

- Extract focal length from intrinsic matrix $K$
  
  \[
  \frac{(K_{11}+K_{22})}{2} \times \text{pixel size} \quad \text{(Nikon D7000: 4.78 um)}
  \]

- View 1: $f = 33.65\text{mm}$
  - EXIF info: 34.0 mm

- View 2: $f = 32.93\text{mm}$
  - EXIF info: 34.0 mm
Accuracy considerations

- Extract focal length from intrinsic matrix K
- Variance on image measurements:
  - sigma = 2 pixels

- View 1: f = 33.65 (sigma = 0.67mm)
  - EXIF info: 34.0 mm

- View 2: f = 32.93mm (sigma = 0.72mm)
  - EXIF info: 34.0 mm
How to estimate the parameter covariance – analytical - I

- 1D case – mapping \( f : x \in \mathbb{R} \mapsto p \in \mathbb{R} \)

from measurement space to parameter space

\[
\bar{p} + \Delta_p = f(\bar{x} + \Delta_x) \approx f(\bar{x}) + \frac{df}{dx} \Delta_x + O(\Delta_x^2)
\]
How to estimate the parameter covariance – analytical - II

- **Taylor expansion**
  \[ \bar{p} + \sigma_p = f(\bar{x} + \sigma_x) \approx f(\bar{x}) + \frac{df}{dx} \sigma_x + O(\sigma_x^2) \]

  - Valid as long as linearization within \([\bar{x} - 2\sigma_x, \bar{x} + 2\sigma_x]\) holds

  \[ \Rightarrow \sigma_p \approx \frac{df}{dx} \sigma_x \quad \text{or} \quad \frac{dx}{df} \sigma_p \approx \sigma_x \]

- **Higher Dimensions:**
  - Description of variance \(\rightarrow\) covariance matrix

  \[ \Sigma_x = \begin{pmatrix}
  \sigma_x^2 & \cdots & \sigma_x \cdot \sigma_{x_n} \\
  \vdots & \ddots & \vdots \\
  \sigma_{x_n} \cdot \sigma_{x_1} & \cdots & \sigma_{x_n}^2
  \end{pmatrix} \]
How to interpret the covariance matrix

- Error ellipsoid (quadratic, positive (semi-) definite form)

\[
(x - \bar{x})^T \Sigma_x^{-1} (x - \bar{x}) < k
\]

- \(k\) determined from \(\chi^2\) distribution \((k, 2)\) [Zhang96]

- E.g. contains
  - 70% of the data for \(k = 2.41\)
  - 95% of the data for \(k = 5.99\)
How to estimate the parameter covariance – analytical - III

- Higher dimensional Taylor:

\[
\bar{x} + \Delta x = f^{-1}(\bar{p} + \Delta p) \approx f^{-1}(\bar{p}) + J_p \Delta p + \mathcal{O}(\Delta p^2)
\]

with Jacobian matrix

\[
J_p \in \mathbb{R}^{M \times N} := \begin{pmatrix}
\frac{\partial x_1}{\partial p_1} & \cdots & \frac{\partial x_1}{\partial p_N} \\
\vdots & \ddots & \vdots \\
\frac{\partial x_M}{\partial p_1} & \cdots & \frac{\partial x_M}{\partial p_N}
\end{pmatrix}
\]

\[\Rightarrow \Delta x \approx J_p \Delta p\]

\[\Rightarrow \Delta x^T \Sigma_x^{-1} \Delta x = \Delta p^T J_p^T \Sigma_x^{-1} J_p \Delta p\]

\[\Rightarrow \Sigma_p = \left( J_p^T \Sigma_x^{-1} J_p \right)^{-1}\]
Example: line fit, direct parametrization

\[ y = m \cdot x + n \]

- measurements

\[ x \] - measurement positions

\[ p = (m, n)^T \]

- parameter vector

fitting linear system \( (N = 100, \sigma_y = 0.5, \sigma_x = 0.25) \)

\[
A \mathbf{p} = \begin{pmatrix}
    x_1 & 1 \\
    \vdots & \vdots \\
    x_N & 1
\end{pmatrix}
\begin{pmatrix}
    m \\
    n
\end{pmatrix}
= \begin{pmatrix}
    y_1 \\
    \vdots \\
    y_N
\end{pmatrix}
= \mathbf{y}
\]

Ground truth values: \( m = 1.5, \ n = 10 \)
Example: line fit, direct parametrization

\[ y = m \cdot x + n \]

\( y \) - measurements

\( x \) - measurement positions

\[ p = (m, n)^T \]

- parameter vector

Solve normal equations \( \rightarrow \) least squares solution

\[ p = (A^T A p)^{-1} A^T y \]

\( \Rightarrow p = (1.516, 9.827)^T \)
Example: line fit, direct parametrization

\[ y = m \cdot x + n \]

Jacobian w.r.t. measurements:

\[
\frac{\partial y}{\partial m} = x, \quad \frac{\partial y}{\partial n} = 1
\]

\[ \Rightarrow J_{py} = \begin{pmatrix} x_1 & \cdots & x_N \\ 1 & \cdots & 1 \end{pmatrix}^T \]

\[ \Sigma_y = \begin{pmatrix} \sigma_{y_1}^2 & 0 & \cdots & 0 \\ 0 & \sigma_{y_2}^2 & \cdots & 0 \\ 0 & \cdots & \cdots & 0 \\ 0 & \cdots & 0 & \sigma_{y_N}^2 \end{pmatrix} = \begin{pmatrix} 0.5^2 & 0 & \cdots & 0 \\ 0 & 0.5^2 & \cdots & 0 \\ 0 & \cdots & \cdots & 0 \\ 0 & \cdots & 0 & 0.5^2 \end{pmatrix} \]

\[ \Sigma_{py} = (J_{py}^T \Sigma_y^{-1} J_{py})^{-1} = 10^{-3} \cdot \begin{pmatrix} 0.301 & -0.017 \\ -0.017 & 2.501 \end{pmatrix} \]
Example: line fit, direct parametrization

\[
y - n \quad \frac{m}{m} = x
\]

Jacobian w.r.t. input uncertainty:

\[
\frac{\partial x}{\partial m} = \frac{n - y}{m^2}, \quad \frac{\partial x}{\partial n} = -\frac{1}{m}
\]

\[
\Rightarrow J_{px} = \begin{pmatrix}
(n - y_1)/m^2 & \cdots & (n - y_N)/m^2 \\
-1/m & \cdots & -1/m
\end{pmatrix}^T
\]

\[
\Sigma_x = \begin{pmatrix}
\sigma_{x_1}^2 & 0 & \cdots & 0 \\
0 & \sigma_{x_2}^2 & \cdots & 0 \\
0 & \cdots & \cdots & 0 \\
0 & \cdots & 0 & \sigma_{x_N}^2
\end{pmatrix} = \begin{pmatrix}
0.25^2 & 0 & \cdots & 0 \\
0 & 0.25^2 & \cdots & 0 \\
0 & \cdots & \cdots & 0 \\
0 & \cdots & 0 & 0.25^2
\end{pmatrix}
\]

\[
\Sigma_{px} = \left( J_{px}^T \Sigma_x^{-1} J_{px} \right)^{-1} = 10^{-3} \cdot \begin{pmatrix}
0.116 & 0.005 \\
0.005 & 1.376
\end{pmatrix}
\]
Combining covariances from different sources

- If the sources are independent, the covariance matrices add
  - Line example – error due to incorrectly known \( x \) and noisy measurements:

\[
\Sigma_p = \Sigma_{px} + \Sigma_{py}
\]

- Line example: \( \Sigma_p = 10^{-3} \cdot \begin{pmatrix} 0.408 & -0.04 \\ -0.04 & 3.794 \end{pmatrix} \)
  - Covariances are negligible

\[
\sigma_m = \sqrt{\Sigma_p^{1,1}}, \quad \sigma_n = \sqrt{\Sigma_p^{2,2}}
\]

\[\Rightarrow \ p = (1.516 \pm 0.0408, 9.827 \pm 0.123)^T\]

with 95% certainty \( (p \pm (2 \cdot \sigma_m, 2 \cdot \sigma_n)^T) \)
$$\Sigma_p = 10^{-2} \begin{pmatrix} 0.441 & -0.277 \\ -0.277 & 4.037 \end{pmatrix}$$

$$\Rightarrow p = (1.516 \pm 0.133, 9.827 \pm 0.402)^T$$

with 95% certainty \(p \pm (2 \cdot \sigma_m, 2 \cdot \sigma_n)^T\)
Monte-Carlo Estimation

- Idea: generate *synthetic* measurements around *true measurement* and analyze sensitivity of solution
  - Collect all parameter estimates from perturbed measurements and estimate their statistics

- Generating data:
  - Line example:
    - Perturbing the measurements \( y' = y + \mathcal{N}_{\Sigma_y} \)
    - Perturbing the input positions \( x' = x + \mathcal{N}_{\Sigma_x} \)
Monte-Carlo Estimation

- Generating noise according to some covariance matrix

\[ N_\Sigma = N(0, 1) \cdot \text{chol}(\Sigma) \]
Over-Parameterized Models

- In practice: (linear) estimates often over-parameterized
  - Example line equation: \( a \cdot x + b \cdot y + c = 0 \)
  - Line is parameterized by \( \mathbf{p} = (a, b, c)^T \)

- Problem: \( \lambda \mathbf{p} = (\lambda a, \lambda b, \lambda c)^T \)
  \[ \Rightarrow \lambda (a \cdot x + b \cdot y + c) = 0 \] is the same equation
  \[ \Rightarrow J_p^T \Sigma_x^{-1} J_p \] is ill-conditioned (not full rank) and cannot be inverted
Over-Parameterized Models

- Idea: invert model in lower-dimensional space where it is full-rank and transfer coordinates

- Example: normalized coordinates: \( \| p \| = 1 \)
  - Describes a manifold (the 2-sphere \( S^2 \)) in 3D Euclidean parameter space

Ivo Ihrke / Winter 2013
Over-Parameterized Models

- Parameter error may vary off the surface of allowed models
  \[ \| (a + \sigma_a, b + \sigma_b, c + \sigma_c)^T \| \neq 1 \]

- Solution:
  - Project onto tangent space \( T_p \)
  - Correct to first order
  - Tangent space
    - can be computed by Jacobian of some direct parameterization
Over-Parameterized Models

- Example: normalized coordinates
  - Describes a sphere of radius 1
  - Use spherical coordinates
    \[
    \begin{align*}
    x &= r \cdot \cos(\theta) \sin(\phi) \\
    y &= r \cdot \sin(\theta) \sin(\phi) \\
    z &= r \cdot \cos(\theta)
    \end{align*}
    \]
  - Set \( r = 1 \)

\[
J_\theta = \begin{pmatrix}
\frac{\partial x}{\partial \theta} & \frac{\partial x}{\partial \phi} \\
\frac{\partial y}{\partial \theta} & \frac{\partial y}{\partial \phi} \\
\frac{\partial z}{\partial \theta} & \frac{\partial z}{\partial \phi}
\end{pmatrix}
\begin{pmatrix}
\hat{\theta} \\
\hat{\phi}
\end{pmatrix}
\]
Then, the covariance transfer can be computed by

\[ \Sigma_p = J_\theta \left( J_\theta^T J_p^T \Sigma_x^{-1} J_p J_\theta \right)^{-1} J_\theta^T \]

1. Error ellipsoid in over-parameterized space

2. Error ellipsoid in tangent space (tangent space coordinates)

3. Error ellipsoid in tangent space (over-parameterized space coordinates)
Over-Parameterized Models

- Example: line fit, implicit representation, same data as before
- For comparison, make sure to use same constraint

\( N = 10 \)
Summary – Uncertainty Estimation

- Monte-Carlo
  + Very general
  + Easy to implement
  + Variable dependency and over-parameterization simple
  - Slow

- Analytical
  + Fast
  - Sometimes difficult derivation
  - Unflexible (e.g. matrix decompositions are tricky to handle)
  - Explicit handling of over-parameterization and variable dependencies
Automatic Checkerboards

[Atcheson et al. “CALTag”]
Automatic Checkerboards

- Principal steps:
  - Detected linked edge segments
  - Fit lines
  - Intersect lines to obtain corners
  - Fit homography to each patch
  - Sample Code
  - Classify patch
  - Optional: if planar, fit homography to all patches and retry to find additional patches

- In all steps: outlier filtering
What about single-view “natural” scenes?

- World-parallel lines are distorted
- → vanishing points
Vanishing points

\[ \hat{x}, \hat{y}, \hat{z} \]
Vanishing points
Vanishing points

- Vanishing points are projections of the directions of the coordinate axes

\[
\hat{x} = A \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \hat{y} = A \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \quad \hat{z} = A \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}
\]

\[
A = KR \begin{bmatrix} 1 & -C \end{bmatrix}
\]

\[
\Rightarrow V := [\hat{x}\hat{y}\hat{z}] = A \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = KR
\]

Points at infinity = intersection of parallel lines
Vanishing points

\[ V := [\hat{x}\hat{y}\hat{z}] = A \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} = KR \]

- can decompose \( V \) with RQ decomposition
- Fix origin of coordinate system

\[ \hat{o} = A \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \]

\[ \Rightarrow \hat{o} = -KRC \]

\[ -(KR)^{-1} \hat{o} = C \]
Accuracy considerations

 Extract focal length from intrinsic matrix K

 View 1: \( f = 34.10 \text{mm} \) (sigmas ?)
  - EXIF info: 34.0 mm

\[
K_1 = \begin{pmatrix}
7400.47 & -829.20 & 3678.15 \\
0 & 6870.23 & 888.23 \\
0 & 0 & 1.0
\end{pmatrix}
\]

 View 2: \( f = 33.45 \text{mm} \) (sigmas ?)
  - EXIF info: 34.0 mm

\[
K_2 = \begin{pmatrix}
7279.19 & 862.23 & 1169.35 \\
0 & 6712.14 & 1104.74 \\
0 & 0 & 1.0
\end{pmatrix}
\]
3D Reconstruction of Architecture

1. Original uncalibrated photographs
2. Primitive definition and localisation
3. Finding vanishing points and camera calibration
4. Computation of projection matrices
5. Triangulation, 3D reconstruction and texture mapping

[Cipolla’99]
Towards Multiple Views and Self-Calibration
Foundations of Camera Motion Estimation

- Feature detection
- Feature tracking
- 2D homography
- Fundamental matrix
Why bother?

Quelle: Transformers, DreamWorks, Director: Michael Bay, Special effects by Industrial Light & Magic and Digital Domain, USA
Towards Multiple Views and Self-Calibration

-- Feature Detection, Matching and Tracking --
Feature Point Detection

- Harris Corner Detector
- corners at high image signal gradients in two perpendicular directions
Harris Corner Detector

image point \( \mathbf{n} = (n_x, n_y) \)

window point \( \mathbf{m} = (m_x, m_y) \) (e.g. 5 x 5 pixels)

- calculate

\[
G(\mathbf{n}) = \sum_{m_x} \sum_{m_y} \begin{bmatrix}
g_x^2(\mathbf{n} + \mathbf{m}) & g_x(\mathbf{n} + \mathbf{m})g_y(\mathbf{n} + \mathbf{m}) \\
g_x(\mathbf{n} + \mathbf{m})g_y(\mathbf{n} + \mathbf{m}) & g_y^2(\mathbf{n} + \mathbf{m})
\end{bmatrix}
\]

\[
\frac{\partial I(x, y)}{\partial x} \approx g_x = \frac{I(x + 1, y) - I(x - 1, y)}{2}
\]

\[
\frac{\partial I(x, y)}{\partial y} \approx g_y = \frac{I(x, y + 1) - I(x, y - 1)}{2}
\]
Harris Corner Detector

Eigenvalues of the symmetric matrix $G(n)$:

$$ G v_i = \lambda_i v_i $$

$$ G(n) = WVW^T = V \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} V^T $$

$\lambda_{\text{min}}$ $\lambda_{\text{max}}$
Harris Corner Detector

Classification of image points using eigenvalues

\[ \lambda_1 \text{ and } \lambda_2 \text{ are small; } E \text{ is almost constant in all directions} \]

\[ \lambda_1 \text{ and } \lambda_2 \text{ are large, } \lambda_1 \sim \lambda_2; \]

\[ \lambda_1 \gg \lambda_2 \]

“Corner”
“Edge”
“Flat” region

Ivo Ihrke / Winter 2013
Harris Corner Detector

- Harris Cornerness Response Function

\[ CRF(n) = \det(G(n)) + K_{CRF} \ \text{trace}^2(G(n)) \]

- Determinant of a matrix

\[ \det(G) = \lambda_1 \cdot \lambda_2 \]

- Trace of a matrix

\[ \text{trace}(G) = \lambda_1 + \lambda_2 \]
Harris Corner Detector

- Input image
- Cornerness response
- Threshold
- Non-maxima suppression
Correspondence Analysis – KLT Tracker

Correspondence Analysis

- For video KLT-Tracker (Kanade, Lucas and Tomasi)
- Minimization of the SSD between two windows in subsequent camera images to find the displacement vector

\[ d = (n_K - n_{K-1}) \]

\[
SSD(d) = \sum_{m_x} \sum_{m_y} [I_K(n_{K-1} + d + m) - I_{K-1}(n_{K-1} + m)]^2 \rightarrow \min
\]
### KLT Tracker

\[
SSD(d) = \sum_{m_x} \sum_{m_y} \left[ I_K(n_{K-1} + d + m) - I_{K-1}(n_{K-1} + m) \right]^2 \rightarrow \min
\]

- Linearization of the image signal with a Taylor series

\[I_K(n_{K-1} + d + m) \approx I_K(n_{K-1} + m) + \begin{pmatrix} g_x(n_{K-1} + m) \\ g_y(n_{K-1} + m) \end{pmatrix}^T d\]

- Finding the minimum by setting the derivative to zero

\[
\sum_{m_x} \sum_{m_y} 2 \left[ I_K(n_{K-1} + m) + \begin{pmatrix} g_x(n_{K-1} + m) \\ g_y(n_{K-1} + m) \end{pmatrix}^T d \right] - I_{K-1}(n_{K-1} + m) \begin{pmatrix} g_x(n_{K-1} + m) \\ g_y(n_{K-1} + m) \end{pmatrix} = 0
\]
\[
\sum_{m_x} \sum_{m_y} 2 \left[ I_K(n_{K-1} + m) + \begin{pmatrix} g_x(n_{K-1} + m) \\ g_y(n_{K-1} + m) \end{pmatrix} \right]^\top d \\
- I_{K-1}(n_{K-1} + m) \begin{pmatrix} g_x(n_{K-1} + m) \\ g_y(n_{K-1} + m) \end{pmatrix} = 0
\]

\[
d = G(n_{K-1})^{-1} b(n_{K-1})
\]

with

\[
b(n_{K-1}) = \sum_{m_x} \sum_{m_y} \left[ I_{K-1}(n_{K-1} + m) - I_K(n_{K-1} + m) \right] \begin{pmatrix} g_x(n_{K-1} + m) \\ g_y(n_{K-1} + m) \end{pmatrix}
\]
Optical Flow – Horn & Schunck’81

- Continuous version of tracking: **optical flow**
- Apparent motion of brightness patterns in an image sequence (typically two frames)
- For images: \( \mathbf{u}(\mathbf{x}): \mathbb{R}^2 \rightarrow \mathbb{R}^2 \), is a vector valued fct.
- Often visualized as vector field or color coded
Example Yosemite sequence

Flow field (middlebury coding)

Flow field (IPOL coding)

middlebury coding

IPOL coding
Optical Flow - Derivation

- Assume a video $I(\vec{x}, t): R^3 \rightarrow R$
- Brightness constancy implies
  $I(\vec{x} + \vec{u}(\vec{x}, t), t) = I(\vec{x}, t + 1)$
- Look at one particular time step with flow vectors $\vec{u} = (u_x, u_y)$
  - perform Taylor expansion of $I(\vec{x}, t + 1)$:
    $$I(\vec{x}, t + 1) \approx I(\vec{x}, t) + \frac{\partial I}{\partial x}u_x + \frac{\partial I}{\partial y}u_y + \frac{\partial I}{\partial t} + O(\nabla^2)$$
  - implies
    $$\frac{\partial I}{\partial x}u_x + \frac{\partial I}{\partial y}u_y + \frac{\partial I}{\partial t} = 0$$
  - Alternative form: $\nabla I \cdot \vec{u} + \frac{\partial I}{\partial t} = 0$
KLT-Tracker: Resolution pyramids

- KLT and optical flow descriptions are **differential** in nature → only suitable for **small** displacements (up to 2 pixels)

- Large displacements are **small** on a coarser scale
  - compute displacements on coarse scale
  - Upsample
  - Iteratively compute residuals on finer scales
KLT-Tracker: Affine mapping for longer sequences
KLT-Tracker summary

- For video (small displacements)
- Minimization of the SSD between two windows in subsequent camera images
- Linearization of the image signal with a Taylor series → linear equation system
- Linearization only valid for small displacements → Resolution Pyramids
References

   implemented in Toolbox:
   Matlab Calibration Toolbox, Jean-Yves Bouguet 2000
   http://www.vision.caltech.edu/bouguetj/calib_doc/
   implemented in Toolbox:
   Implemented in: Voodoo Camera Tracker,
   http://www.digilab.uni-hannover.de.
References