Image formation: camera calibration

Peter Veelaert, Hiep Luong

February 15, 2024

Table of Contents

Table of Contents

Range, rank and nullspace of a matrix

Let **A** be an $m \times n$ matrix.

- The set of all **x** that satisfy $\mathbf{Ax} = 0$ is called the **nullspace** of **A** (*all linear dependencies that exist between the column vectors*)
- The set of all vectors **b** of the form $\mathbf{Ax} = \mathbf{b}$ is called the **range** of **A** (*the space spanned by the column vectors*)
- the nullspace is a subspace of $\mathbb{R}^n$
- the range is a subspace of $\mathbb{R}^m$
- the dimension of the range is called the **rank** of **A**
- rank $\leq \min(m, n)$

Orthogonal matrices

An orthogonal matrix $\mathbf{A}$ is a real square matrix whose columns (rows) are orthonormal vectors.

Main properties:

- $\mathbf{A}\mathbf{A}^T = \mathbf{A}^T\mathbf{A} = \mathbf{I}$
- $\mathbf{A}^{-1} = \mathbf{A}^T$
- $\det \mathbf{A} = \pm 1$
- orthogonal matrices form a group: the product of two orthogonal matrices is orthogonal
- a rotation matrix is an orthogonal matrix with $\det \mathbf{A} = +1$
- rotation matrices form a subgroup of the group of orthogonal matrices

Singular value decomposition

Let $\mathbf{A}$ be a real $m \times n$ matrix.

$\mathbf{A}$ can always be decomposed as

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T$$

with

- $\mathbf{U}, \mathbf{V}$ orthogonal matrices
- $\mathbf{D}$ a rectangular diagonal matrix, with non-negative real numbers on its diagonal
- $\mathbf{U}$ a $m \times m$ matrix
- $\mathbf{D}$ a $m \times n$ matrix
- $\mathbf{V}$ a $n \times n$ matrix

Singular value decomposition

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T$$

with

- the elements d_i on the diagonal of **D** are called the **singular values** of **A**
- common practice: choose decomposition so that singular values are in descending order
- the columns of **U** associated with nonzero singular values form an orthonormal base for the range of **A**
- the columns of **V** associated with the singular values that are zero form an orthonormal base for the nullspace of **A**

Singular value decomposition: example

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 2 \\ 1 & 2 & 3 \\ 1 & 3 & 4 \\ 1 & 4 & 5 \end{pmatrix} = \mathbf{U}\mathbf{D}\mathbf{V}^T =$$

$$\begin{pmatrix} 0.252 & 0.798 & 0.535 & -0.120 \\ 0.399 & 0.375 & -0.805 & -0.239 \\ 0.546 & -0.047 & 0. & 0.837 \\ 0.692 & -0.470 & 0.267 & -0.478 \end{pmatrix} \begin{pmatrix} 9.344 & 0. & 0. \\ 0. & 0.829 & 0. \\ 0. & 0. & 0. \\ 0. & 0. & 0. \end{pmatrix} \begin{pmatrix} 0.202 & 0.791 & -0.577 \\ 0.584 & -0.570 & -0.577 \\ 0.786 & 0.220 & 0.577 \end{pmatrix}^T$$

- Since σ_1, σ_2 are non-zero while $\sigma_3 = 0$, the columns of **A** span a 2D linear space.
- The two first columns of **U** form an orthonormal basis for this space.
- Since $\sigma_3 = 0$, there is a 1D linear space of linear dependencies between the columns of **A**.
- The third column of **V** forms an orthonormal basis for this space of linear dependencies.

Table of Contents

Purpose of LS

Find the least square solution of an over-determined system of linear equations

$$\mathbf{Ax} = \mathbf{b}$$

Least square minimization is ubiquitous in computer vision:

- to find the fundamental matrix
- find a homography, affine, rigid body transformation between two images
- find a projection matrix, intrinsic and extrinsic camera parameters
- fit a curve (line, parabola, circle, ...) to a set of data points
- fit a plane to a set of 3D points
- ...

Least squares

Consider the system of linear equations

$$\mathbf{Ax} = \mathbf{b}$$

with $\mathbf{A}$ a real $m \times n$ matrix. (not necessarily square!)

If $m > n$, the system is **over-determined**, and in general it will not have a solution.

However, we can seek the vector $\mathbf{x}$ that minimizes

$$||\mathbf{Ax} - \mathbf{b}||$$

This is the **least squares** solution of the over-determined system.

Least squares

Using the SVD of $\mathbf{A}$:

$$||\mathbf{Ax} - \mathbf{b}|| = ||\mathbf{UDV}^T\mathbf{x} - \mathbf{b}||$$

Since $\mathbf{U}$ is orthogonal, it does not change the norm:

$$||\mathbf{UDV}^T\mathbf{x} - \mathbf{b}|| = ||\mathbf{DV}^T\mathbf{x} - \mathbf{U}^T\mathbf{b}||$$

Let $\mathbf{y} = \mathbf{V}^T\mathbf{x}$ and $\mathbf{b}' = \mathbf{U}^T\mathbf{b}$,

then we seek $\mathbf{y}$ that minimizes

$$||\mathbf{Dy} - \mathbf{b}'||$$

Least squares

$$||\mathbf{Dy} - \mathbf{b}'||$$

is of the form

$$|| \left[\begin{array}{cccc} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \\ \hline 0 & \dots & & 0 \\ \dots & & & \\ 0 & \dots & & 0 \end{array}\right] \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix} - \left[\begin{array}{c} b'_1 \\ b'_2 \\ \dots \\ b'_n \\ \hline b'_{n+1} \\ \dots \\ b'_m \end{array}\right] ||$$

When $d_i \neq 0$ this norm is clearly minimized when

$$y_i = b'_i / d_i$$

When $d_i = 0$, we set $y_i = 0$.

$\mathbf{Vy} = \mathbf{x}$ is then the solution that minimizes $||\mathbf{Ax} - \mathbf{b}||$

Least squares

Summarizing:

$$||\mathbf{Ax} - \mathbf{b}||$$

is minimized by

$$\mathbf{x} = \mathbf{VD'U}^T\mathbf{b}$$

where $\mathbf{D}'$ is the transpose of $\mathbf{D}$ with each non-zero element d_i replaced by its reciprocal $1/d_i$.

Least squares when $n = m$

When $\mathbf{A}$ is a non-singular square matrix, all $d_i \neq 0$, and

$$\mathbf{A}^{-1} = (\mathbf{U}\mathbf{D}\mathbf{V}^T)^{-1} = (\mathbf{V}^T)^{-1}\mathbf{D}^{-1}\mathbf{U}^{-1} = \mathbf{V}\mathbf{D}'\mathbf{U}^T$$

Hence the LS solution

$$\mathbf{x} = \mathbf{V}\mathbf{D}'\mathbf{U}^T\mathbf{b}$$

coincides with the standard solution of a linear system of n equations in n unknowns:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

A related problem

Find $\mathbf{x}$ with $||\mathbf{x}|| = 1$ that minimizes

$$||\mathbf{Ax}||$$

The optimal solution is

$$\mathbf{x} = \mathbf{v}_n,$$

where $\mathbf{v}_n$ is the last column of $\mathbf{V}$.

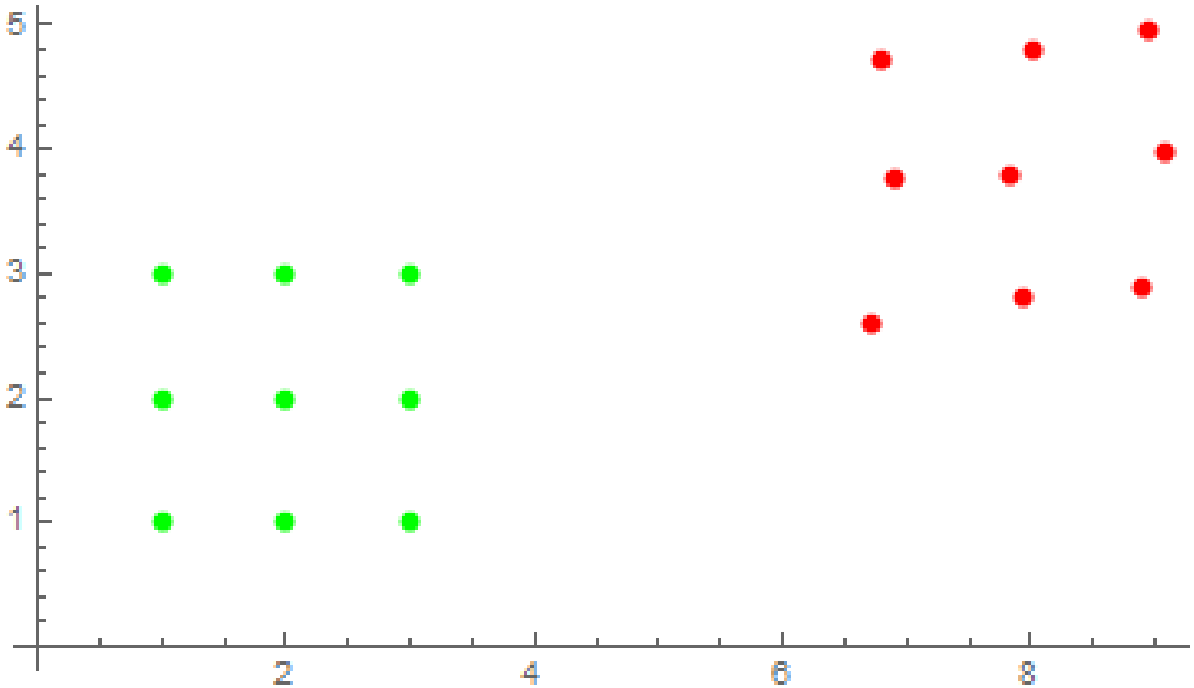
Why?

$$||\mathbf{Ax}|| = ||\mathbf{UDV}^T\mathbf{x}|| = ||\mathbf{DV}^T\mathbf{x}|| = ||\mathbf{Dy}||$$

Since σ_n is the smallest singular value, the vector $\mathbf{y}$ that minimizes $\mathbf{Dy}$, subject to $||\mathbf{y}|| = 1$, is $\mathbf{y} = (0, \ldots, 0, 1)^T$.

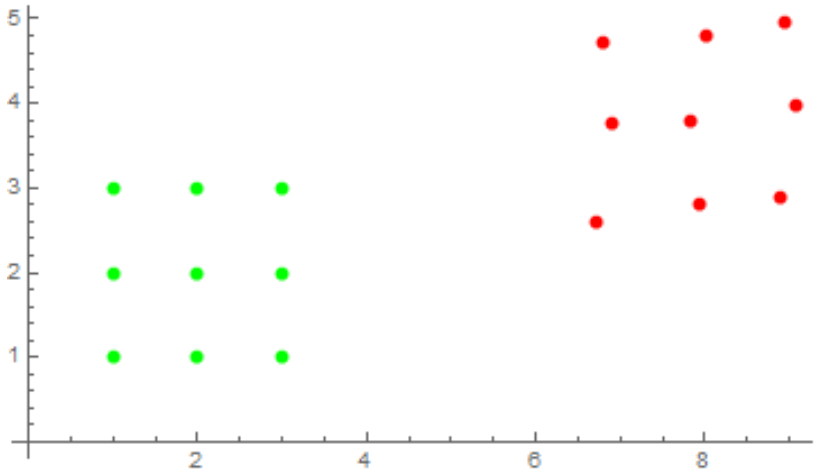
Hence, the vector $\mathbf{x}$ that minimizes $||\mathbf{Ax}||$ is $\mathbf{Vy}$, with $\mathbf{y} = (0, \ldots, 0, 1)^T$, that is $\mathbf{x} = \mathbf{v}_n$.

An example: best affine transformation



Suppose we want to find the best affine transformation that maps the red points onto the green points.

An example: best affine transformation



The general form of an affine transformation is:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Each point pair $(x, y) \rightarrow (x', y')$ yields constraints of the form

$$\begin{aligned} x' &= ax + by + c \\ y' &= dx + ey + f \end{aligned}$$

Since we have 9 point pairs, we have 18 constraints for 6 unknowns.
Clearly this is an overdetermined system, for which we will find a LS solution.

An example: best affine transformation

$$\begin{pmatrix} 6.72291 \\ 6.89613 \\ 6.79445 \\ 7.93342 \\ 7.84109 \\ 8.01707 \\ 8.88669 \\ 9.08627 \\ 8.95496 \\ 2.59573 \\ 3.75595 \\ 4.71335 \\ 2.81457 \\ 3.7993 \\ 4.79671 \\ 2.89362 \\ 3.96986 \\ 4.96737 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 \\ 1 & 3 & 1 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 & 0 & 0 \\ 2 & 2 & 1 & 0 & 0 & 0 \\ 2 & 3 & 1 & 0 & 0 & 0 \\ 3 & 1 & 1 & 0 & 0 & 0 \\ 3 & 2 & 1 & 0 & 0 & 0 \\ 3 & 3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 1 & 3 & 1 \\ 0 & 0 & 0 & 2 & 1 & 1 \\ 0 & 0 & 0 & 2 & 2 & 1 \\ 0 & 0 & 0 & 2 & 3 & 1 \\ 0 & 0 & 0 & 3 & 1 & 1 \\ 0 & 0 & 0 & 3 & 2 & 1 \\ 0 & 0 & 0 & 3 & 3 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix}$$

or $\mathbf{b} = \mathbf{A}\mathbf{a}$, where the column vector $\mathbf{a}$ contains the unknowns $a, \ldots, f$.

An example: best affine transformation

According to previous slides, the LS solution is

$$\mathbf{a} = \mathbf{V}\mathbf{D}'\mathbf{U}^T\mathbf{b}$$

If we apply a singular value decomposition to **A** we find

- a 18×18 matrix **U**
- a 6×6 matrix **V**
- a 18×6 matrix **D** with singular values σ_i:

$$\mathbf{D} = \begin{pmatrix} 9.29381 & 0. & 0. & 0. & 0. & 0. \\ 0. & 9.29381 & 0. & 0. & 0. & 0. \\ 0. & 0. & 2.44949 & 0. & 0. & 0. \\ 0. & 0. & 0. & 2.44949 & 0. & 0. \\ 0. & 0. & 0. & 0. & 0.790685 & 0. \\ 0. & 0. & 0. & 0. & 0. & 0.790685 \\ 0. & 0. & 0. & 0. & 0. & 0. \\ \dots & & & & & \\ 0. & 0. & 0. & 0. & 0. & 0. \end{pmatrix}$$

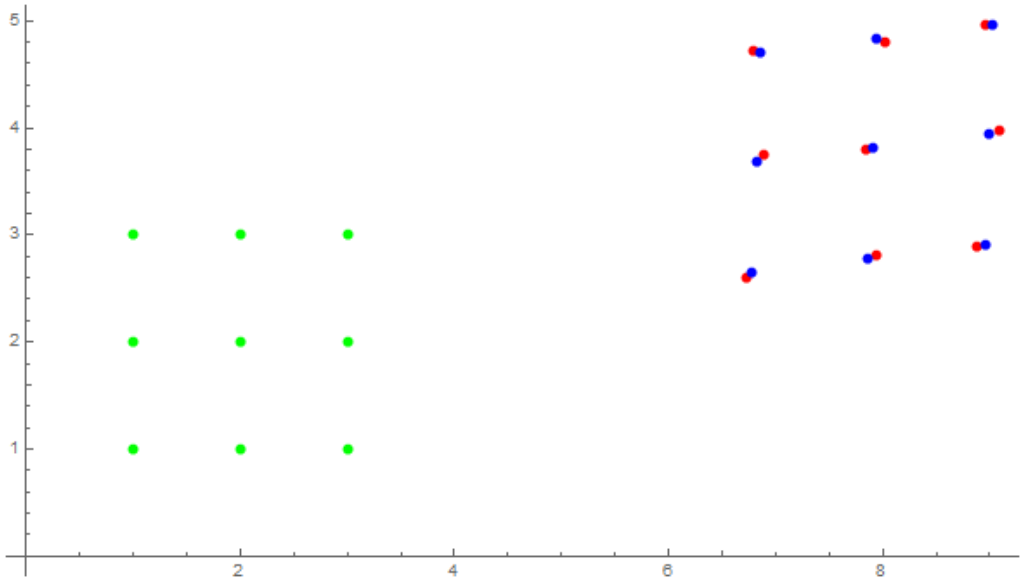
Note. Since all 6 singular values are non-zero, the nullspace of **A** is empty, and the 6 column vectors of **A** span a 6 dimensional linear subspace.
What would happen if all the red points would lie on a common line?

An example: best affine transformation

$\mathbf{D}'$ is a 6×18 matrix with the reciprocals $1/\sigma_i$:

$$\mathbf{D}' = \begin{pmatrix} 0.107599 & 0. & 0. & 0. & 0. & 0. & 0. & \dots & 0. \\ 0. & 0.107599 & 0. & 0. & 0. & 0. & 0. & & 0. \\ 0. & 0. & 0.408248 & 0. & 0. & 0. & 0. & & 0. \\ 0. & 0. & 0. & 0.408248 & 0. & 0. & 0. & & 0. \\ 0. & 0. & 0. & 0. & 1.26473 & 0. & 0. & & 0. \\ 0. & 0. & 0. & 0. & 0. & 1.26473 & 0. & & 0. \end{pmatrix}$$

An example: best affine transformation



From

$$\mathbf{a} = \mathbf{V}\mathbf{D}'\mathbf{U}^T\mathbf{b}$$

we find the affine transformation

$$\mathbf{T}' = \begin{pmatrix} 1.08574 & 0.0372444 & 5.6577 \\ 0.127639 & 1.02892 & 1.49871 \\ 0 & 0 & 1 \end{pmatrix}$$

which maps the green points onto the blue points.

Estimating a 2D homography

The best 2D homography is found in the same way:

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \mathbf{Hx} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

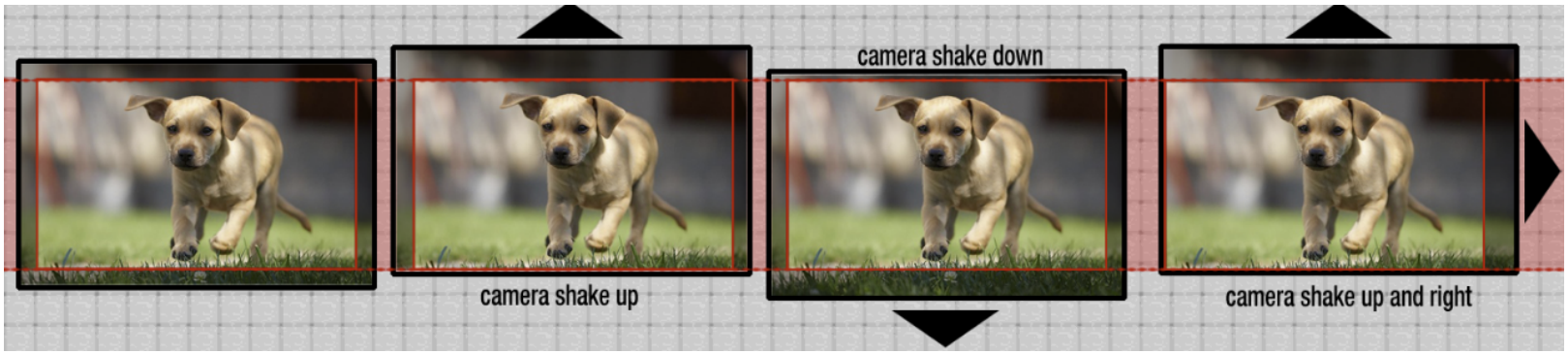
Hence

$$x'_i(h_{31}x_i + h_{32}y_i + h_{33}) = h_{11}x_i + h_{12}y_i + h_{13}$$

$$y'_i(h_{31}x_i + h_{32}y_i + h_{33}) = h_{21}x_i + h_{22}y_i + h_{23}$$

2 linear equations in 9 unknowns

Concluding remarks

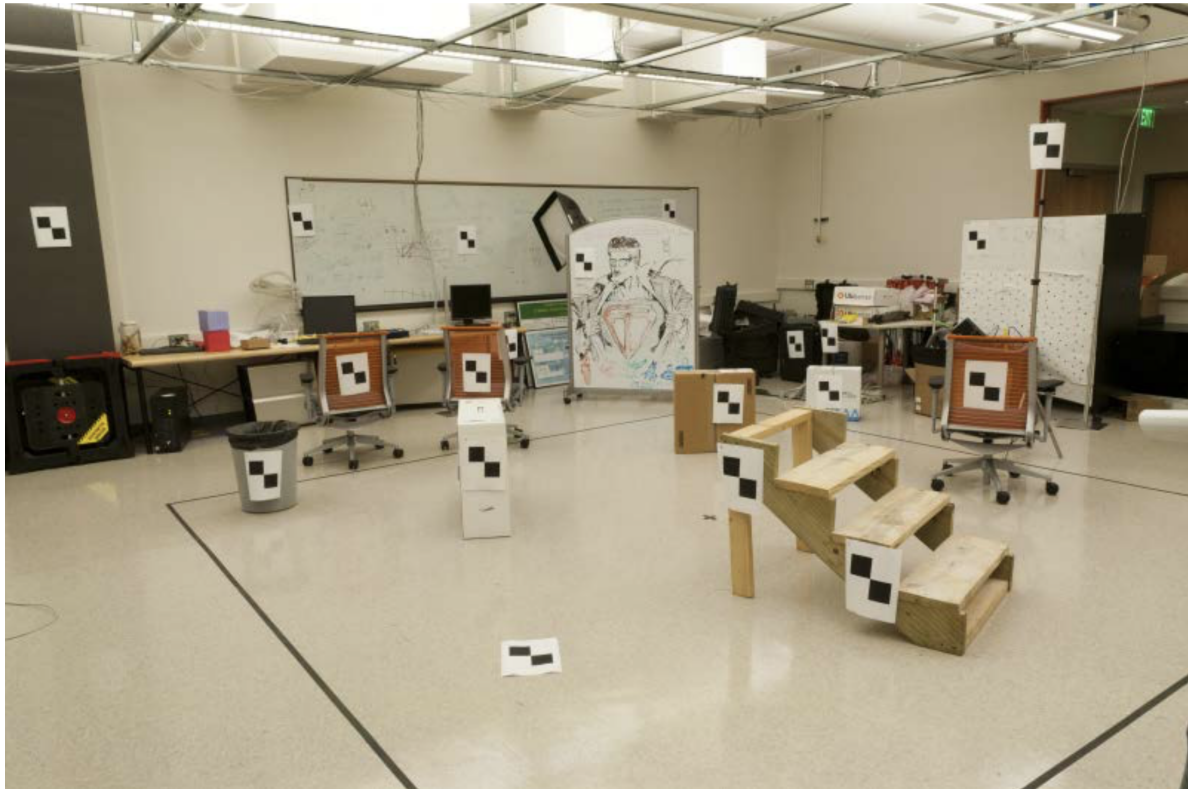


picture from invensense.tdk.com

- Affine transformations are often used for image stabilization, motion estimation, correction of lens distortion, panorama stitching, and image registration
- 2D homographies can be found with the same ease
- Similarities and Euclidean transformations are more difficult since the rotation angle θ is not a linear variable (*requires non-linear optimization, such as Levenberg-Marquardt*)
- We assumed all point correspondences are known. (*When this is not the case we need additional tools: feature detectors + descriptors, FLANN, Ransac*)

Table of Contents

Camera calibration from known 3D points



Find **P** such that

$$\mathbf{x} = \mathbf{PX}$$

for known pairs $(\mathbf{x}, \mathbf{X})$.

3D points should not lie in a common plane.

Camera calibration from known 3D points

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{PX} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Hence

$$x_i(p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34}) = p_{11}X_i + p_{12}Y_i + p_{13}Z_i + p_{14}$$

$$y_i(p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34}) = p_{21}X_i + p_{22}Y_i + p_{23}Z_i + p_{24}$$

Camera calibration from known 3D points

Each pair of points defines two linear equations in the unknowns p_{ij}:

$$x_i(p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34}) = p_{11}X_i + p_{12}Y_i + p_{13}Z_i + p_{14}$$

$$y_i(p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34}) = p_{21}X_i + p_{22}Y_i + p_{23}Z_i + p_{24}$$

Define

$$\mathbf{p} = (p_{11}, p_{12}, p_{13}, p_{14}, p_{21}, \ldots, p_{34})^T$$

Then

$$\begin{pmatrix} X_i & Y_i & Z_i & 1 & 0 & 0 & 0 & 0 & -x_iX_i & -x_iY_i & -x_iZ_i & -x_i \\ 0 & 0 & 0 & 0 & X_i & Y_i & Z_i & 1 & -y_iX_i & -y_iY_i & -y_iZ_i & -y_i \end{pmatrix} \mathbf{p} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

For N points we obtain a homogeneous system

$$\mathbf{Ap} = 0$$

where $\mathbf{A}$ is a $2N \times 12$ matrix. Since $\mathbf{p}$ is only defined up to scale, we may use SVD to find $\mathbf{p}$ that minimizes $\mathbf{Ap}$ subject to $||\mathbf{p}|| = 1$.

Intrinsic vs extrinsic

Now that we have **P**, how can we find the intrinsic (**K**), and extrinsic parameters (**R**, **T**)?

$$\mathbf{P} = \mathbf{K}\,(\mathbf{I}|\mathbf{0}) \begin{pmatrix} \mathbf{R} & \mathbf{T} \\ 0 & 1 \end{pmatrix}$$

where **K** is an upper triangular matrix.

Write $\mathbf{P} = (\mathbf{H}|\mathbf{h})$ such that $\mathbf{H} = \mathbf{KR}, \quad \mathbf{h} = \mathbf{KT}$

K and **R** ar found by a so-called *QR*-decomposition of $\mathbf{H}^{-1}$:

$$\mathbf{H}^{-1} = \mathbf{AB}$$

where **A** is an orthogonal matrix, and **B** an upper triangular matrix.

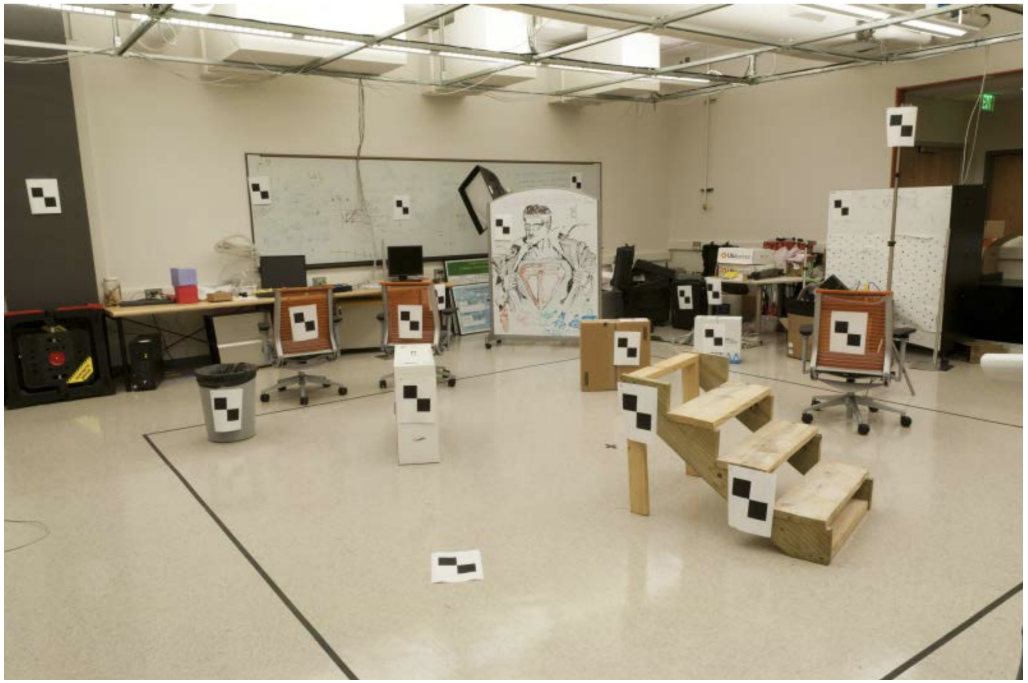
Hence

$$\mathbf{H} = \mathbf{B}^{-1}\mathbf{A}^{-1} = \mathbf{B}^{-1}\mathbf{A}^{T}$$

and

$$\mathbf{K} = \mathbf{B}^{-1}, \quad \mathbf{A}^{T} = \mathbf{R}, \quad \mathbf{K}^{-1}\mathbf{h} = \mathbf{T}$$

Camera calibration from known 3D points



Advantages:

- simple (note: this approach is also known as the Direct Linear Transform)

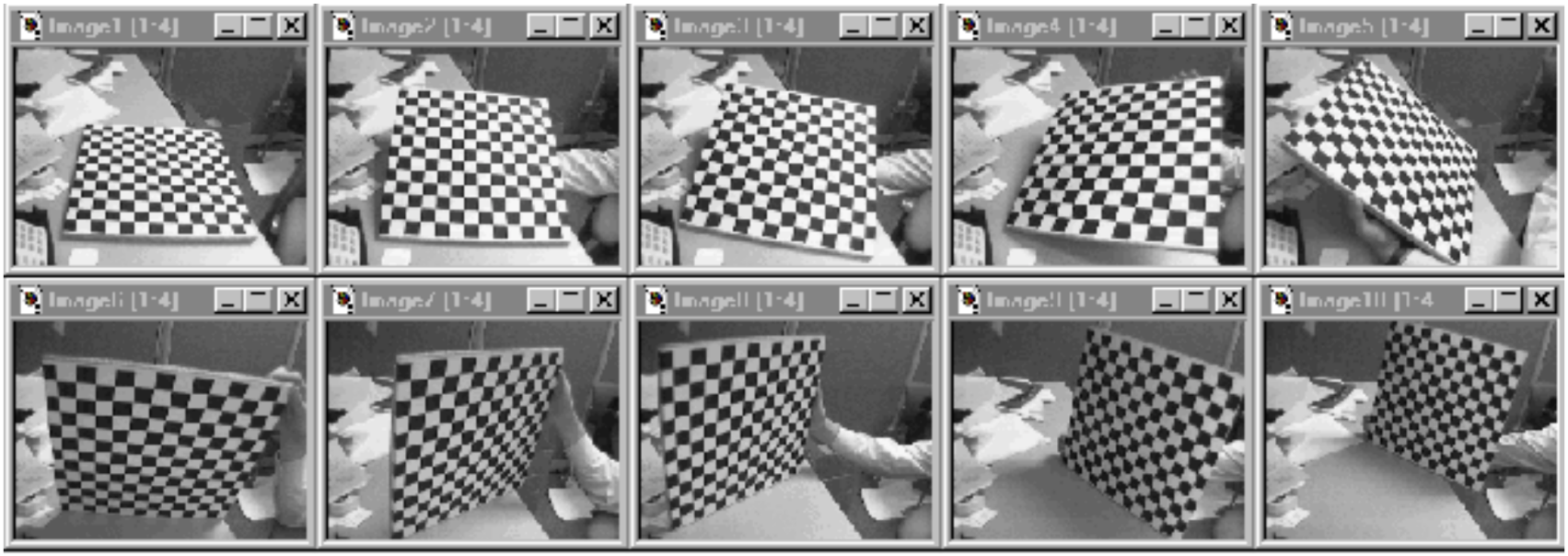
Drawbacks:

- position 3D points not always easy to measure
- minimizes an algebraic error, not a geometric error
- doesn't model **lens distortion**

Table of Contents

Multiplane calibration: Zhang's method

Chessboard waved in front of camera:



Advantages:

- only requires **planar object** with known features (chessboard)
- finds 5 intrinsic parameters
- can be extended to find lens distortion correction coefficients
- code in Matlab, OpenCV (Zhang's method)

Zhang's method

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{PX} = \mathbf{K}(\mathbf{R}|\mathbf{t})\mathbf{X} = \begin{bmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Choose world coordinate frame such that $Z = 0$ for chessboard (we are free to do so)

Then

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \mathbf{H} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

Where **H** is a 2D homography.

We know how we can estimate **H** (see previous section)

Once we have **H**, how can we find **K**?

Zhang's method

Once we have $\mathbf{H}$, how can we find $\mathbf{K}$?

$$\mathbf{H} = (\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3) = \mathbf{K}(\mathbf{r}_1, \mathbf{r}_2, \mathbf{t})$$

Hence, $\mathbf{r}_1 = \mathbf{K}^{-1}\mathbf{h}_1$, $\mathbf{r}_2 = \mathbf{K}^{-1}\mathbf{h}_2$.

We will use the orthonormality constraints of $\mathbf{r}_1, \mathbf{r}_2$:

$$\mathbf{r}_1^T\mathbf{r}_2 = 0, \quad ||\mathbf{r}_1|| = ||\mathbf{r}_2|| = 1$$

Therefore,

$$\mathbf{h}_1^T\mathbf{K}^{-T}\mathbf{K}^{-1}\mathbf{h}_2 = 0$$

$$\mathbf{h}_1^T\mathbf{K}^{-T}\mathbf{K}^{-1}\mathbf{h}_1 - \mathbf{h}_2^T\mathbf{K}^{-T}\mathbf{K}^{-1}\mathbf{h}_2 = 0$$

After introducing the symmetric matrix $\mathbf{B} = \mathbf{K}^{-T}\mathbf{K}^{-1}$, we have

$$\mathbf{h}_1^T\mathbf{B}\mathbf{h}_2 = 0, \quad \mathbf{h}_1^T\mathbf{B}\mathbf{h}_1 - \mathbf{h}_2^T\mathbf{B}\mathbf{h}_2 = 0$$

Zhang's method

$$\mathbf{h}_1^T \mathbf{B} \mathbf{h}_2 = 0, \quad \mathbf{h}_1^T \mathbf{B} \mathbf{h}_1 - \mathbf{h}_2^T \mathbf{B} \mathbf{h}_2 = 0$$

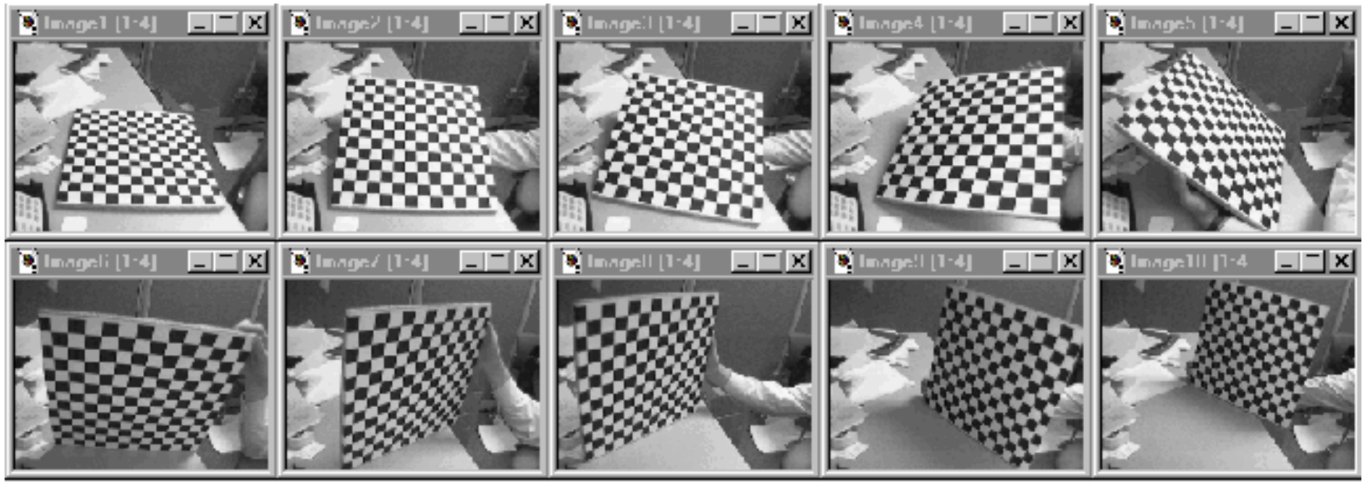
or

$$\begin{aligned} &b_{11} h_{11} h_{21} + b_{22} h_{12} h_{22} + b_{33} h_{13} h_{23} + \\ &b_{12}(h_{12} h_{21} + h_{11} h_{22}) + b_{13}(h_{11} h_{23} + h_{13} h_{21}) + b_{23}(h_{13} h_{22} + h_{12} h_{23}) = 0 \end{aligned}$$

$$\begin{aligned} &b_{11}(h_{11}^2 - h_{21}^2) + b_{22}(h_{12}^2 - h_{22}^2) + b_{33}(h_{13}^2 - h_{23}^2) + \\ &2b_{12}(h_{11} h_{12} - h_{21} h_{22}) + 2b_{13}(h_{11} h_{13} - h_{21} h_{23}) + 2b_{23}(h_{12} h_{13} - h_{22} h_{23}) = 0 \end{aligned}$$

- Two linear equations in 6 unknowns $b_{11}, \ldots, b_{33}$
- By waving the chessboard in front of the camera and computing homographies for each image, we produce a large number of equations

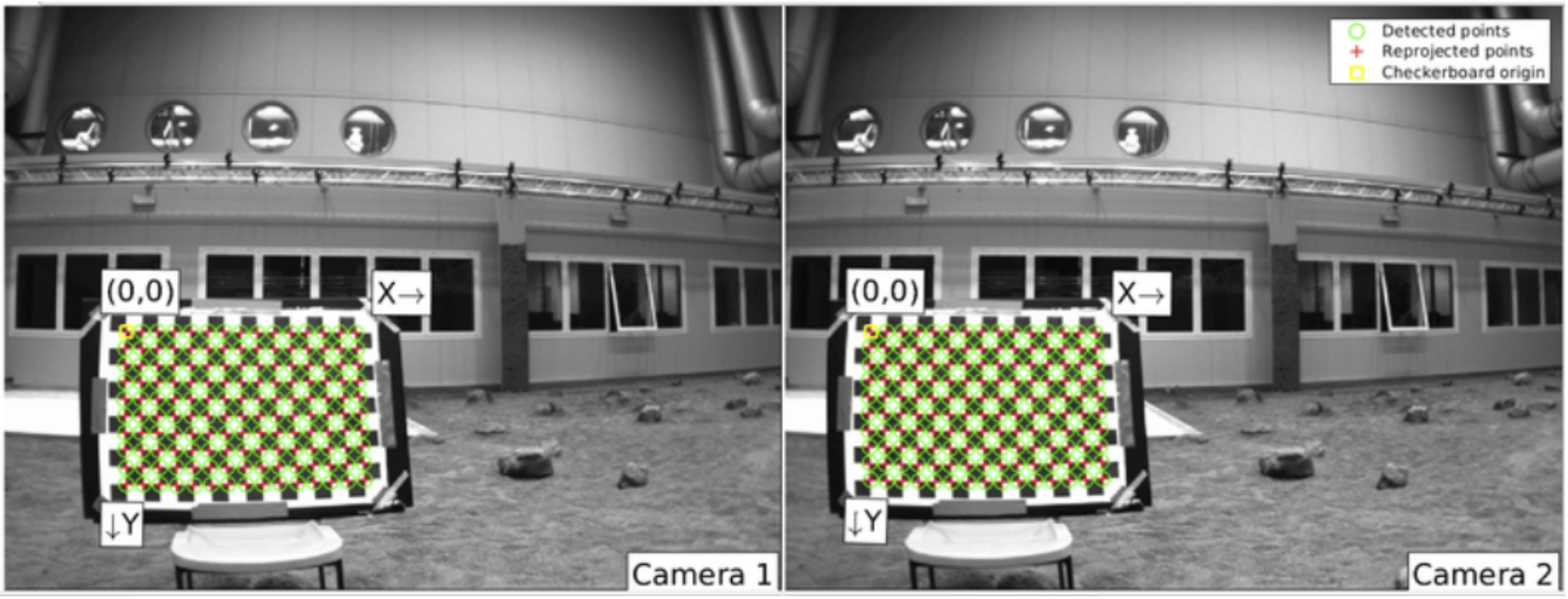
Zhang's method



- Each position of the chessboard defines a different homography **H**, and therefore 2 additional linear equations in 6 unknowns $b_{11}, \ldots, b_{33}$.
- $n >> 3$ chessboard images define a system with $2n$ equations for which we can find an LS solution via SVD
- **K** can be recovered through a **Cholesky decomposition** of **B**: $\mathbf{B} = \mathbf{A}\mathbf{A}^T$, $\mathbf{K} = \mathbf{A}^{-T}$.

Note: Cholesky decomposition of a real symmetric matrix, is a decomposition of the form $\mathbf{B} = \mathbf{A}\mathbf{A}^T$, where **A** is a lower triangular matrix.

Zhang's method: camera pose estimation



When chessboard is waved in front of two cameras, we can easily find their relative poses.

Zhang's method: camera pose estimation

Consider one position of the chessboard. Then for the first camera

$$\mathbf{r}_1 = \mathbf{K}^{-1}\mathbf{h}_1, \quad \mathbf{r}_2 = \mathbf{K}^{-1}\mathbf{h}_2, \quad \mathbf{t} = \mathbf{K}^{-1}\mathbf{h}_3$$

Since, $\mathbf{R}$ is an orthogonal matrix, we easily find $\mathbf{r}_3$.

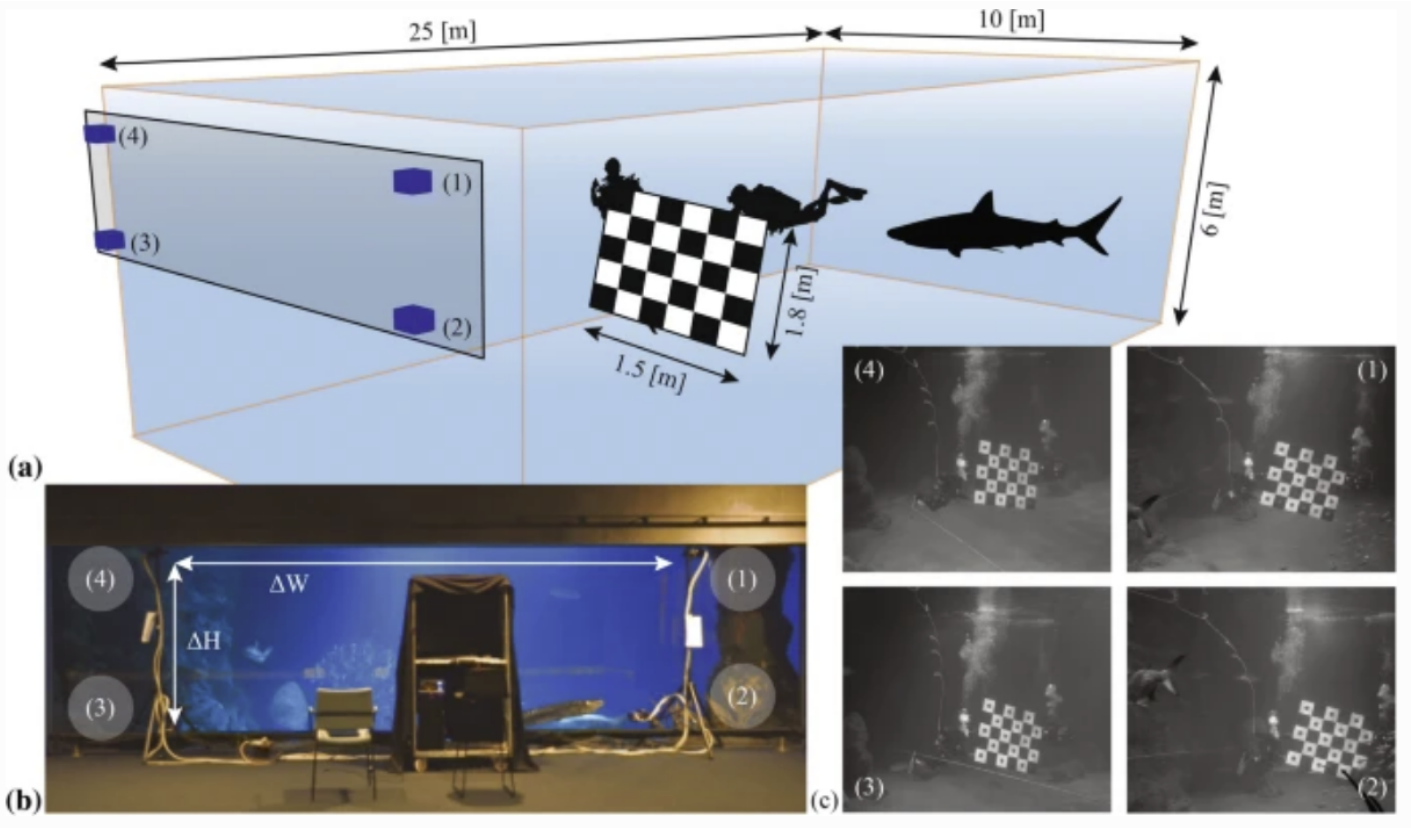
Hence we know the relation between the chessboard coordinate frame and the camera frame:

$$\begin{pmatrix} \mathbf{R} & \mathbf{T} \\ 0 & 1 \end{pmatrix}$$

Similarly, for the second camera, we find $\begin{pmatrix} \mathbf{R}' & \mathbf{T}' \\ 0 & 1 \end{pmatrix}$

and the two camera frames are related as $\begin{pmatrix} \mathbf{R}' & \mathbf{T}' \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{R} & \mathbf{T} \\ 0 & 1 \end{pmatrix}^{-1}$

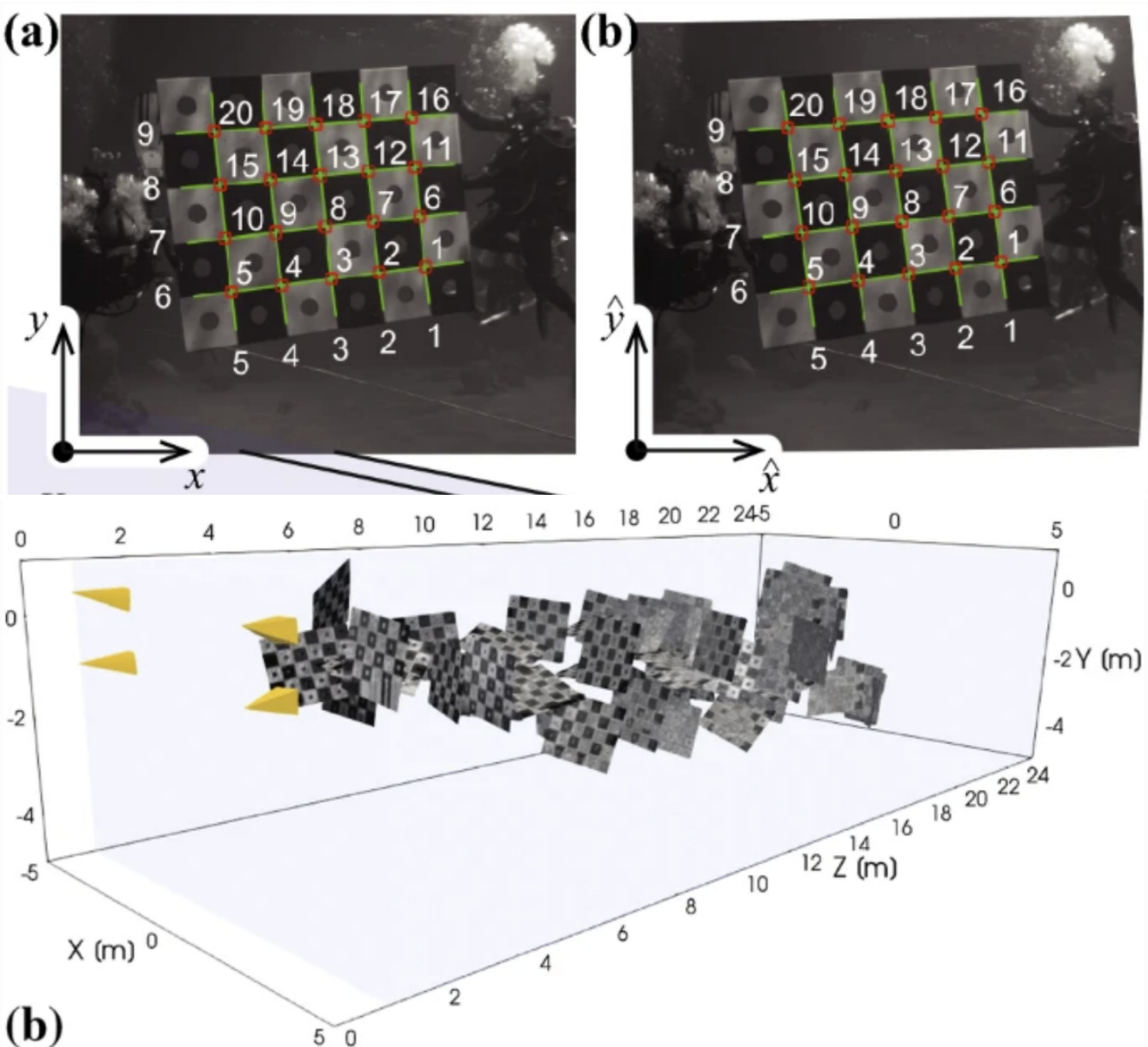
Zhang's method: camera pose estimation



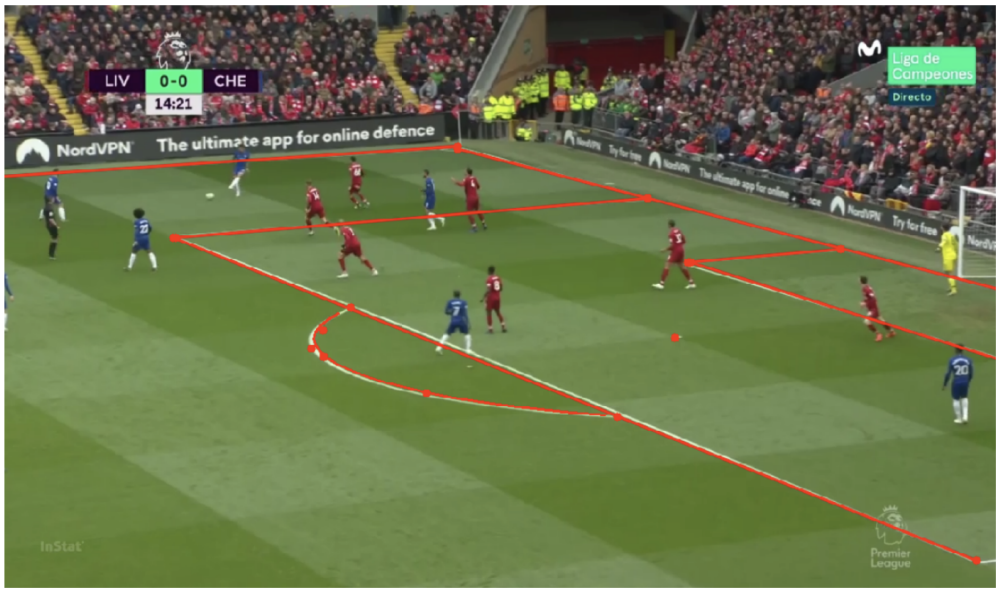
By pairwise calibration we find all the intrinsic parameters and relative poses in a multicamera system. One of the camera frames can be selected as the world coordinate frame.

Picture from Calibration of multiple cameras for large-scale experiments using a freely moving calibration target, Muller et al, Experiments in fluids 61, 2019.

Zhang's method: camera pose estimation



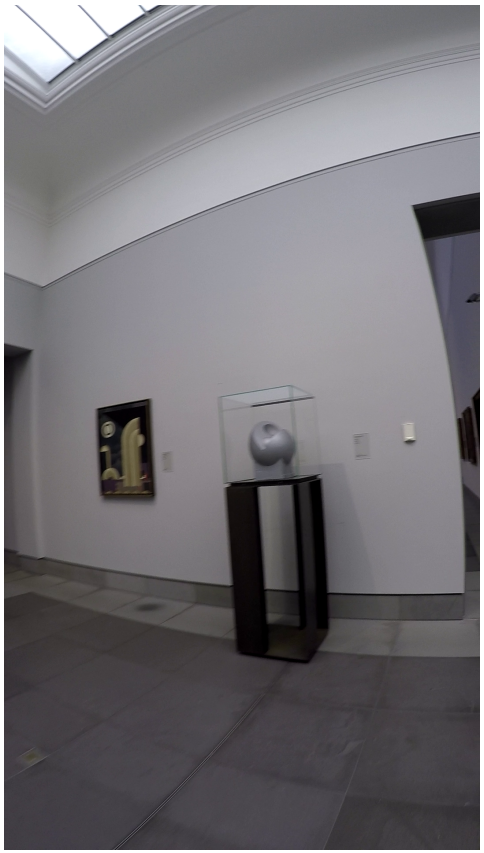
Camera calibration for large scenes



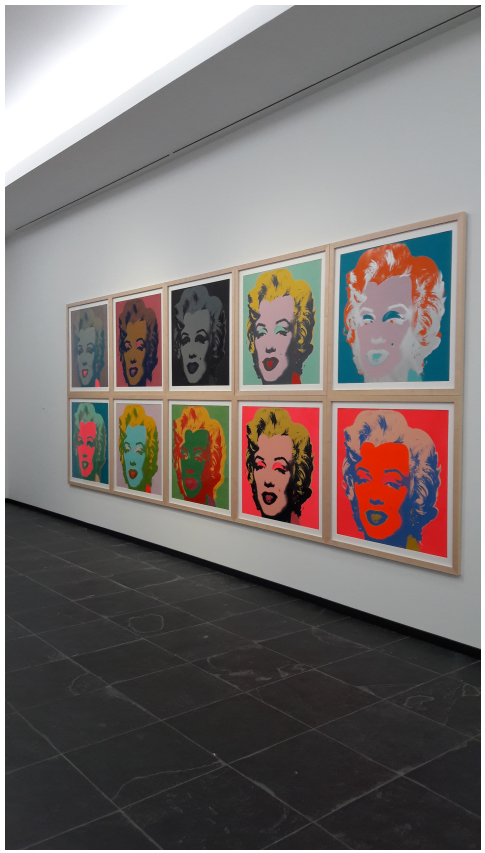
- when cameras are far apart, we don't have a chessboard large enough to be seen by all cameras
- dimensions of large objects are often known (playing fields, cars, buildings)
- intrinsic calibration matrix $\mathbf{K}$ of each camera can still be found with chessboard calibration
- extrinsic camera parameters are found from known 3D points: find $\mathbf{P}$, then $\mathbf{K}^{-1}\mathbf{P} = (\mathbf{R}|\mathbf{t})$
- for very large scenes dimensions may even be retrieved from google maps (e.g., drones flying over agriculture fields)

Table of Contents

Lens distortion



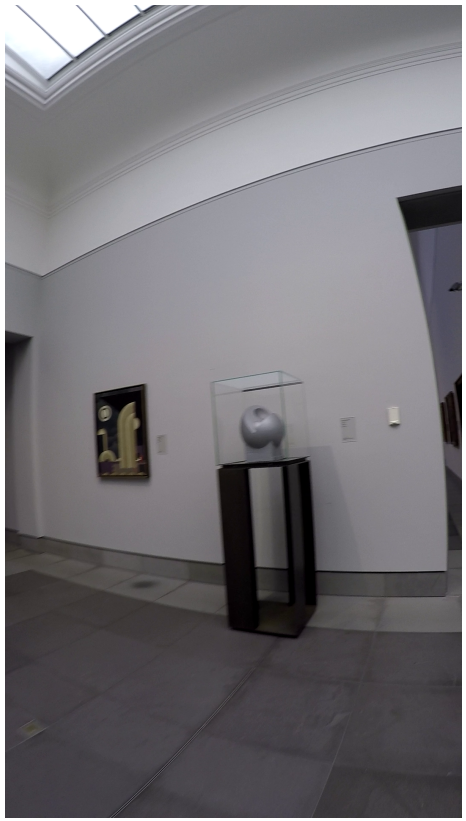
GoPro Hero 4

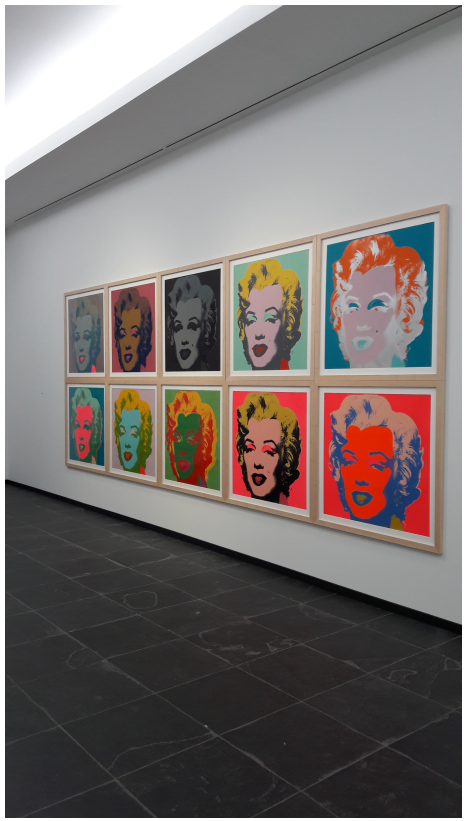


Samsung Galaxy A5

Many wide-angle lenses have noticeable **radial distortion**.

Lens distortion





Compensation of lens distortion is needed for

- detection of straight lines, circles, rectangles, squares, ...
- computation fundamental matrix, extrinsic calibration, triangulation, ...
- stereo matching, stitching, rectification, ...

Lens distortion

A common way to model radial distortion:

$$\begin{aligned} x'_c &= x_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4) \\ y'_c &= y_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4) \end{aligned}$$

- r_c denotes the radial distance of pixel (x_c, y_c) from the optical center (which may differ slightly from the image center)
- given $n >> 1$ point pairs $\{(x_c, y_c), (x'_c, y'_c)\}$, κ_1, κ_2 can be estimated with LS and SVD
- κ_1, κ_2 and the position of the optical center can also be obtained with Zhang's **multiplane camera calibration** method (e.g., see Bouguet's Camera Calibration Toolbox for Matlab)
- distortion compensation is always the first step before any further processing takes place