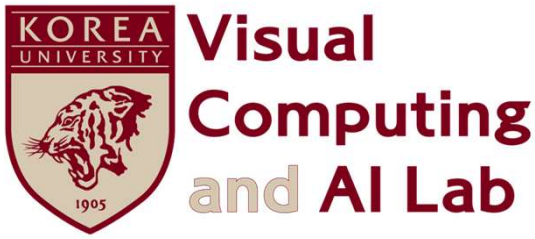
INTRODUCTION TO COMPUTER VISION

Lecture 4 – Camera Calibration

Gyeongsik Moon

Visual Computing and AI Lab

Korea University



Slides Credit: Prof. Dr. Davide Scaramuzza

1. Camera calibration

2. Camera localization

1. Camera calibration

Camera Calibration

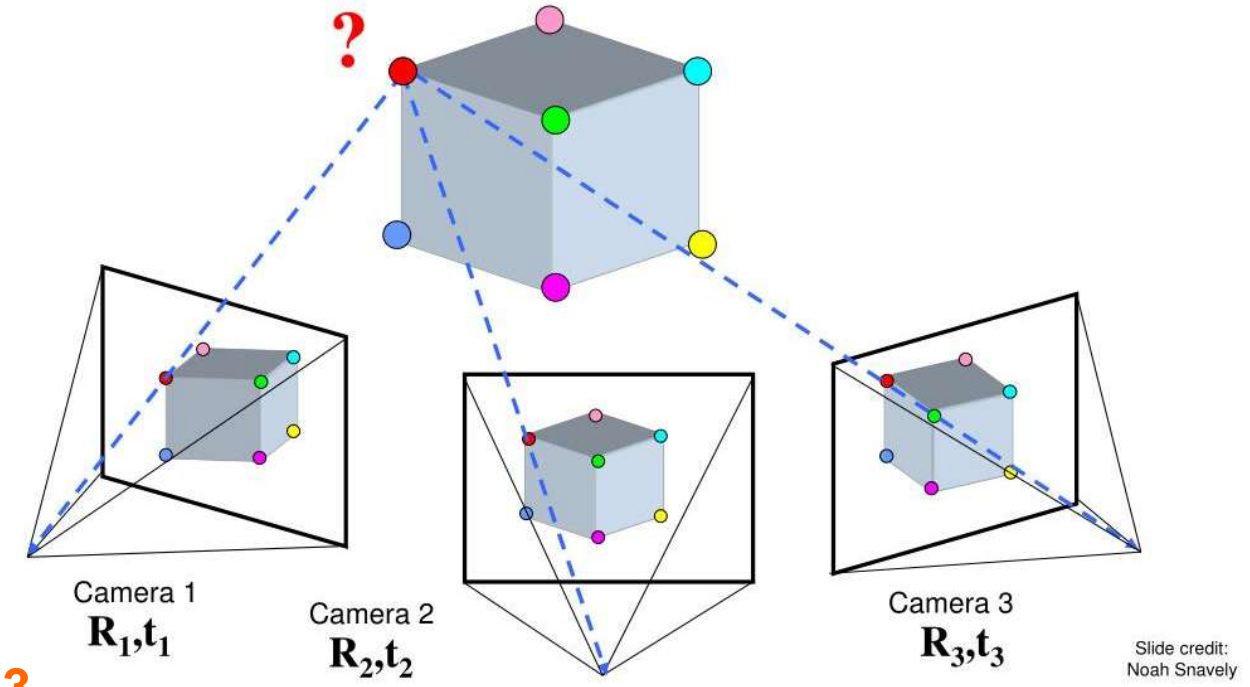
Goal:
Getting intrinsic/extrinsic camera parameters

Multi-View Geometry (MVG) Theory

According to MVG (we'll learn this later), if we know

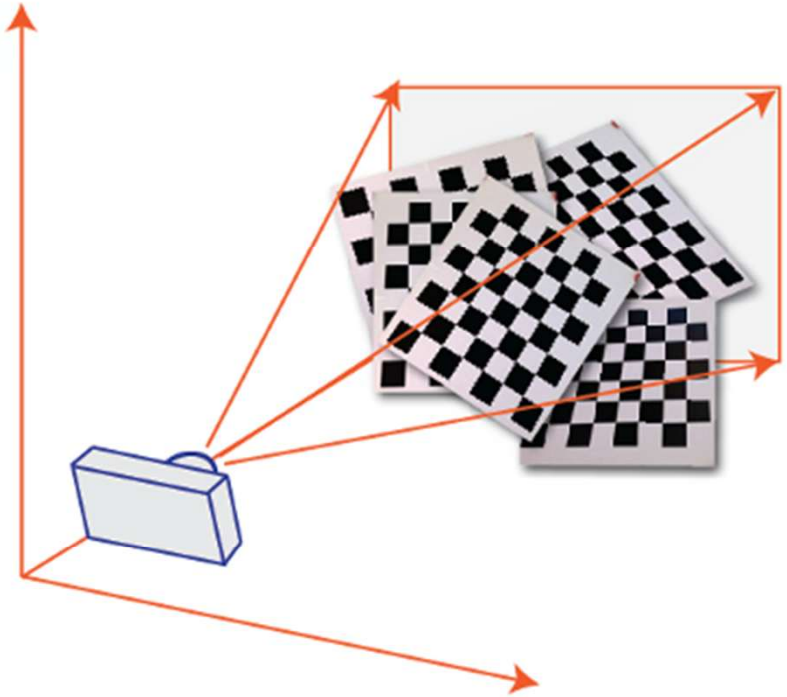
- Camera intrinsic/extrinsic parameters (we learnt what are they in prev. classes)
- *Matched points across multiple viewpoints (same-colored dots in images)*

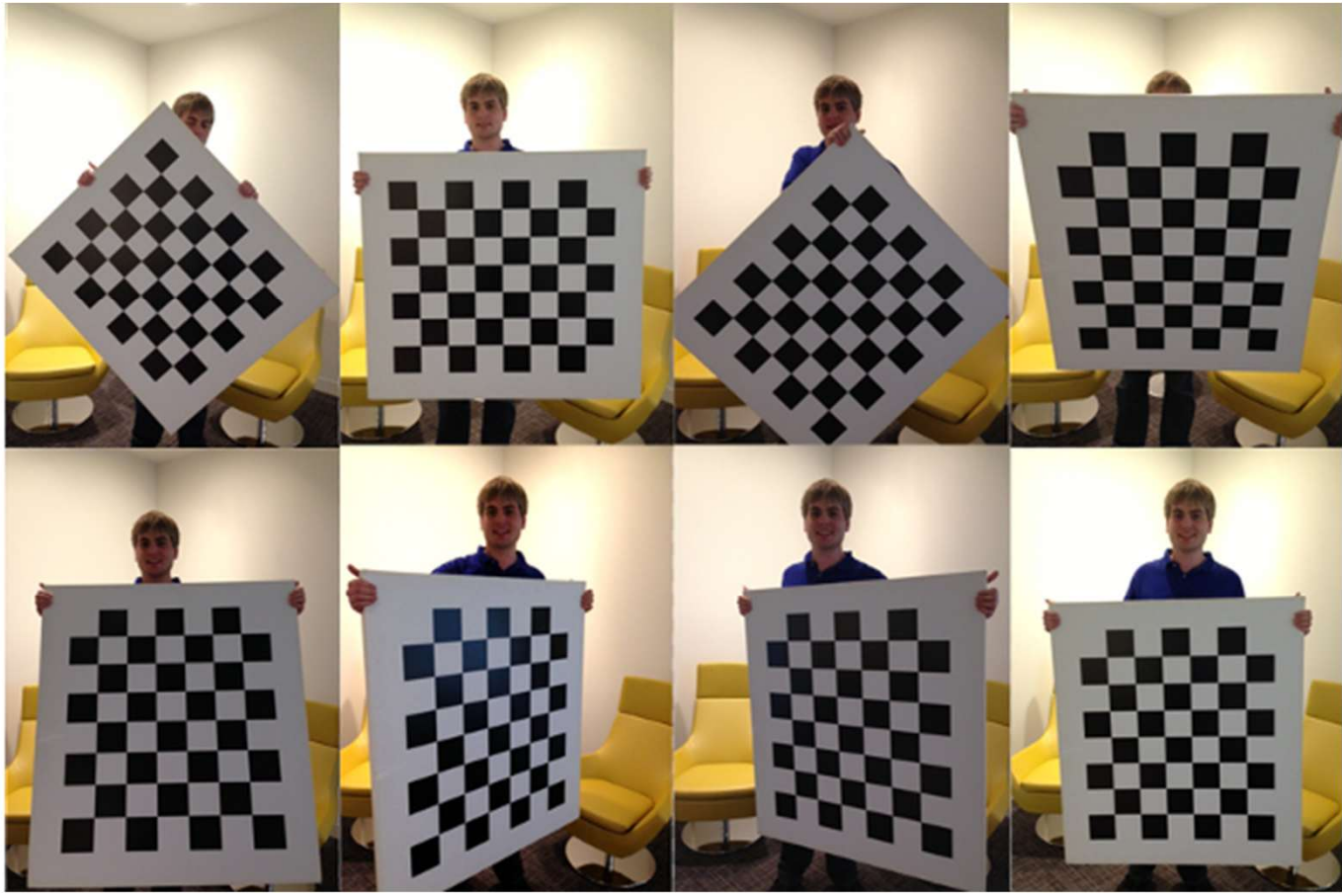
, then, we can lift the multi-view observations to the 3D space



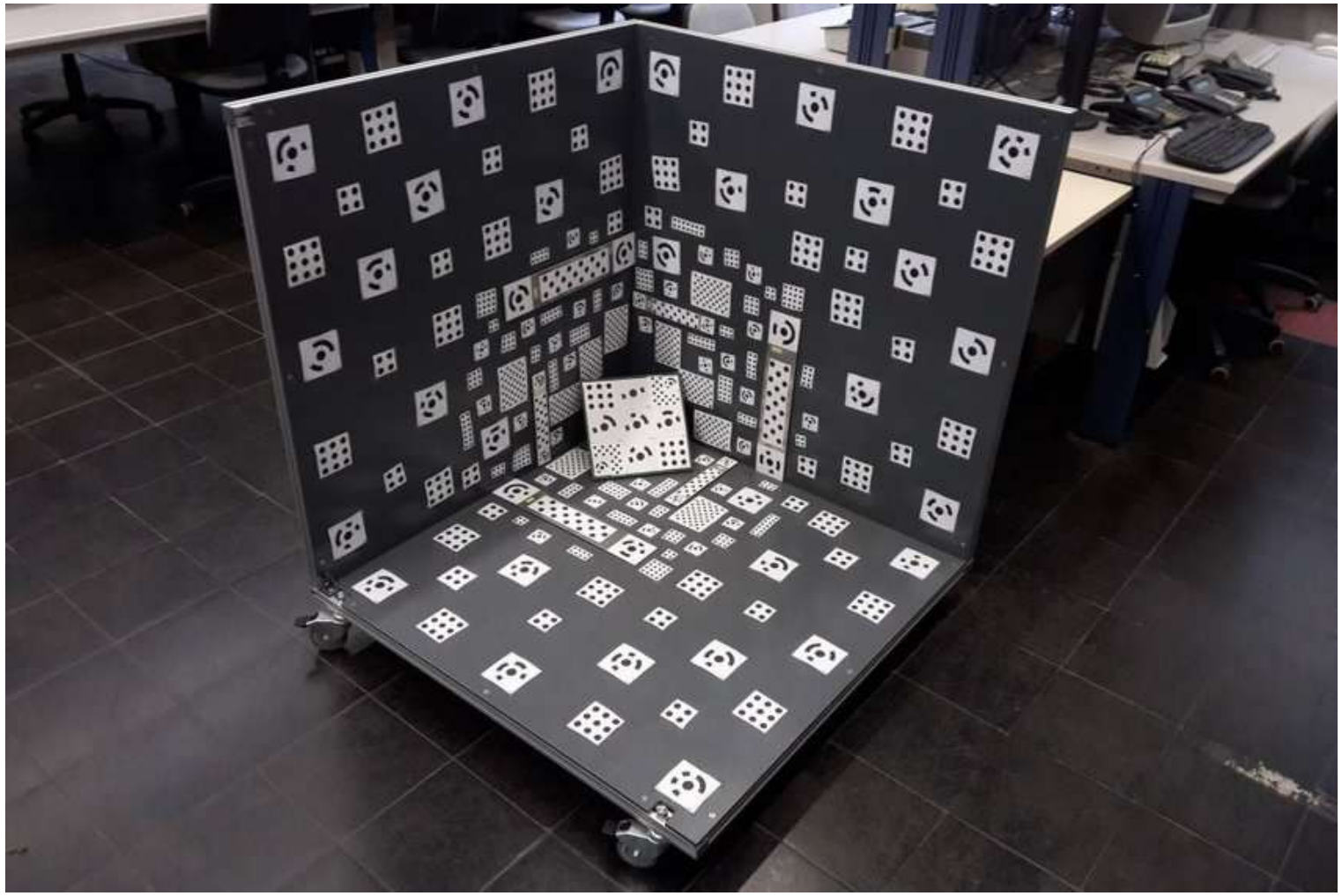
Slide from Lecture 3

How to calibrate cameras?





How to calibrate cameras?



Camera Calibration

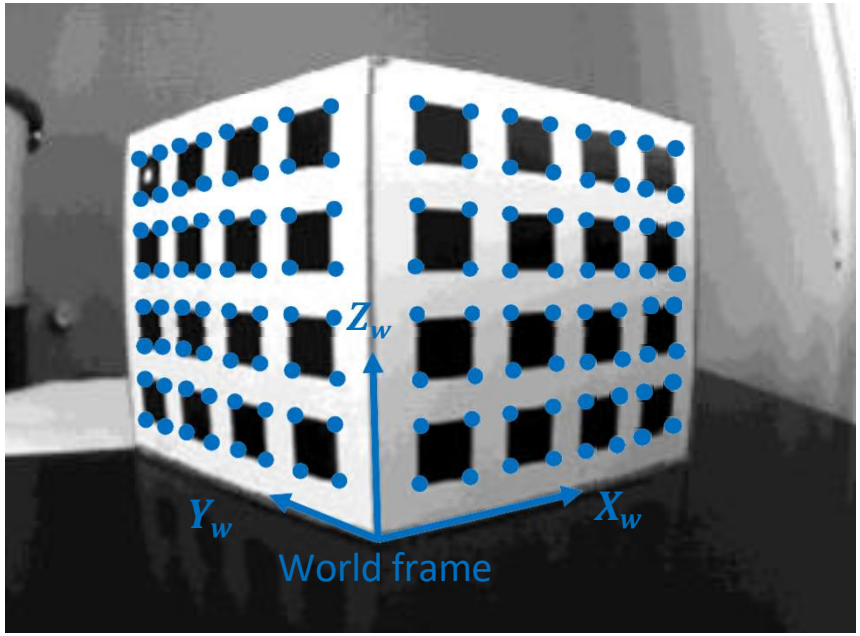
- Calibration is the process to determine the **intrinsic** (K plus lens distortion) **and extrinsic** (R, T) parameters of a camera. For now, we will **neglect the lens distortion** and see later how it can be determined.
- The solution for K, R, T can be found by applying the perspective projection equation:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K\left[R \mid T\right] \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

- There are two popular methods:
  - **Tsai's method**: uses 3D objects
  - **Zhang's method**: uses planar grids

Tsai's Method: Calibration from 3D Objects

- This method was proposed in 1987 by Tsai and consists of measuring the 3D position of $\boldsymbol{n \geq 6}$ **control points** on a 3D calibration target and the **2D coordinates of their projection** in the image.
- Assumption: we know 2D and 3D coordinates of control points ← distinctive points, (corner...)
  - 2D: image pre-processing (e.g., corner detectors)
  - 3D: we know actual size of the 3D object and actual positions of control points as well



Tsai, Roger Y. (1987) "A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses,'' *IEEE Journal of Robotics and Automation*, 1987. PDF.

Applying the Direct Linear Transform (DLT) algorithm

The idea of the DLT is to rewrite the perspective projection equation as a **homogeneous linear equation** and solve it by standard methods. Let's write the perspective equation for a generic 3D-2D point correspondence:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K[R \mid T] \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \Rightarrow$$

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_u r_{11} + u_0 r_{31} & \alpha_u r_{12} + u_0 r_{32} & \alpha_u r_{13} + u_0 r_{33} & \alpha_u t_1 + u_0 t_3 \\ \alpha_v r_{21} + v_0 r_{31} & \alpha_v r_{22} + v_0 r_{32} & \alpha_v r_{23} + v_0 r_{33} & \alpha_v t_2 + v_0 t_3 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

M

Applying the Direct Linear Transform (DLT) algorithm

The idea of the DLT is to rewrite the perspective projection equation as a **homogeneous linear equation** and solve it by standard methods. Let's write the perspective equation for a generic 3D-2D point correspondence:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K\left[R \mid T\right] \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \Rightarrow$$

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

Applying the Direct Linear Transform (DLT) algorithm

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = M \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

goal is getting M.

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} m_1^{\mathrm{T}} \\ m_2^{\mathrm{T}} \\ m_3^{\mathrm{T}} \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

where m_i^{T} is the i-th row of M

Applying the Direct Linear Transform (DLT) algorithm

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} m_1^{\mathrm{T}} \\ m_2^{\mathrm{T}} \\ m_3^{\mathrm{T}} \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \rightarrow P$$

Conversion back from homogeneous coordinates to pixel coordinates leads to:

$$u = \frac{\lambda u}{\lambda} = \frac{m_1^{\mathrm{T}} \cdot P}{m_3^{\mathrm{T}} \cdot P}$$
$$v = \frac{\lambda v}{\lambda} = \frac{m_2^{\mathrm{T}} \cdot P}{m_3^{\mathrm{T}} \cdot P}$$
$$\Rightarrow \qquad (m_1^{\mathrm{T}} - u_i m_3^{\mathrm{T}}) \cdot P_i = 0$$
$$(m_2^{\mathrm{T}} - v_i m_3^{\mathrm{T}}) \cdot P_i = 0$$

Applying the Direct Linear Transform (DLT) algorithm

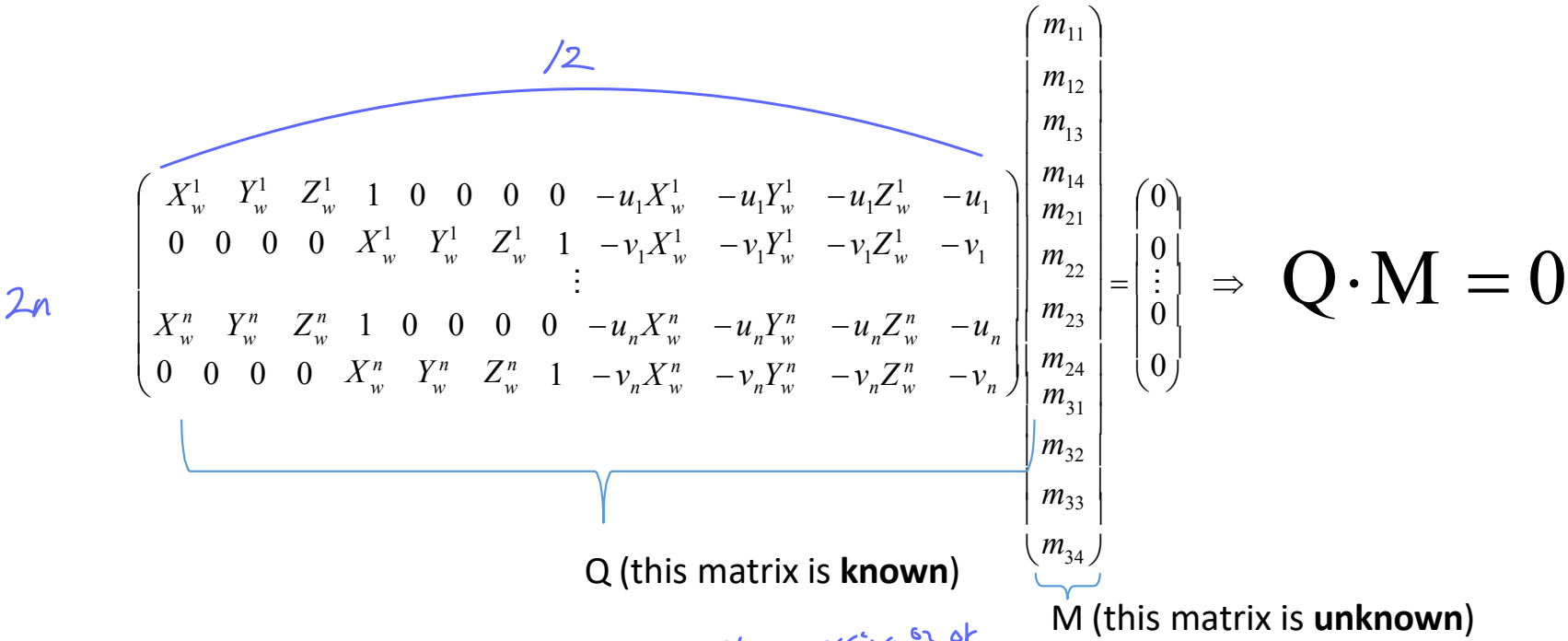
- By re-arranging the terms, we obtain

$$\begin{aligned}(m_1^{\mathrm{T}} - u_i m_3^{\mathrm{T}}) \cdot P_i = 0 \\ (m_2^{\mathrm{T}} - v_i m_3^{\mathrm{T}}) \cdot P_i = 0\end{aligned} \Rightarrow \begin{pmatrix} P_1^{\mathrm{T}} & 0^{\mathrm{T}} & -u_1 P_1^{\mathrm{T}} \\ 0^{\mathrm{T}} & P_1^{\mathrm{T}} & -v_1 P_1^{\mathrm{T}} \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \\ m_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

- For n points, we can stack all these equations into a big matrix:

$$\begin{pmatrix} P_1^{\mathrm{T}} & 0^{\mathrm{T}} & -u_1 P_1^{\mathrm{T}} \\ 0^{\mathrm{T}} & P_1^{\mathrm{T}} & -v_1 P_1^{\mathrm{T}} \\ & \vdots & \\ P_n^{\mathrm{T}} & 0^{\mathrm{T}} & -u_n P_n^{\mathrm{T}} \\ 0^{\mathrm{T}} & P_n^{\mathrm{T}} & -v_n P_n^{\mathrm{T}} \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \\ m_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}$$

Applying the Direct Linear Transform (DLT) algorithm



Applying the Direct Linear Transform (DLT) algorithm

$$\mathbf{Q} \cdot \mathbf{M} = 0$$

Minimal solution

- $Q_{(2n\times 12)}$ should have rank 11 to have a unique (up to a scale) *non-zero* solution M
- Because each 3D-to-2D point correspondence provides 2 independent equations, then $5+\frac{1}{2}$ point correspondences are needed (in practice **6 point** correspondences!) 6개를 알면 M을 구할수 있음

Over-determined solution

- For $n \geq 6$ points, a solution is the **Least Square solution**, which minimizes the sum of squared residuals, $\|QM\|^2$, subject to the constraint $||M||^2 = 1$. It can be solved through Singular Value Decomposition (SVD). The solution is the eigenvector corresponding to the smallest eigenvalue of the matrix Q^TQ (because it is the unit vector x that minimizes $||Qx||^2 = x^TQ^TQx$.
- Matlab instructions: practice에선 6개보다 많은 points 사용.
  - `[U,S,V] = SVD(Q);`
  - `M = V(:,12);`

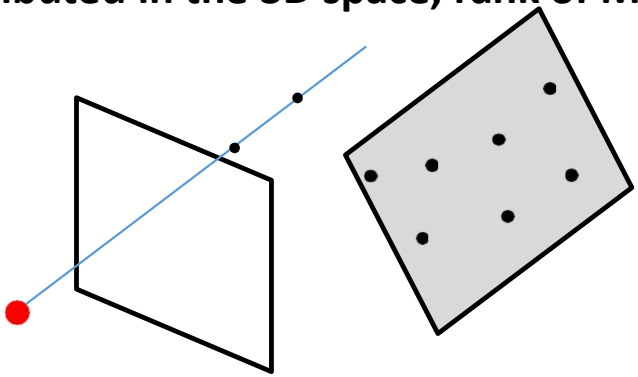
6개를 최소로.

Applying the Direct Linear Transform (DLT) algorithm

$$Q \cdot M = 0$$

Degenerate configurations
(cases where DLT does not work)

1. Points lying on a **plane** and/or along a single **line** passing through the **center of projection**
- **If points are not sufficiently distributed in the 3D space, rank of M is not enough to get solutions**



2. Camera and points on a **twisted cubic** (i.e., smooth curve in 3D space of degree 3)

- Points on a twisted cubic can be represented with polynomial equation, which can strongly limit rank of M

Applying the Direct Linear Transform (DLT) algorithm

- Once we have determined M, we can recover the intrinsic and extrinsic parameters by remembering that:

$$\mathrm{M} = \mathrm{K}(\mathrm{R} \mid \mathrm{T})$$

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}$$

Applying the Direct Linear Transform (DLT) algorithm

- Once we have determined M, we can recover the intrinsic and extrinsic parameters by remembering that:

$$\mathrm{M} = \mathrm{K}(\mathrm{R} \mid \mathrm{T})$$

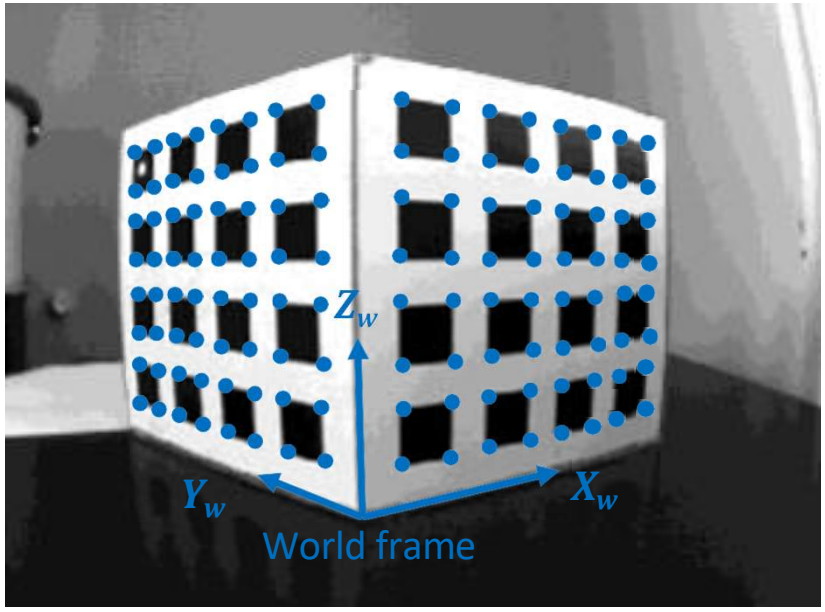
$kt(R) = 1$
$RR^T = I$

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} = \begin{bmatrix} \alpha r_{11} + u_0 r_{31} & \alpha r_{12} + u_0 r_{32} & \alpha r_{13} + u_0 r_{33} & \alpha t_1 + u_0 t_3 \\ \alpha r_{21} + v_0 r_{31} & \alpha r_{22} + v_0 r_{32} & \alpha r_{23} + v_0 r_{33} & \alpha t_2 + v_0 t_3 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}$$

- However, notice that we are not enforcing the constraint that $\boldsymbol{R}$ **is orthogonal, i.e.,** $\boldsymbol{R} \cdot \boldsymbol{R}^{\boldsymbol{T}} = \boldsymbol{I}$
- To do this, we can use the so-called **QR factorization of** $\boldsymbol{M}$, which decomposes M into a R (orthogonal), T, and an upper triangular matrix (i.e., K)

Example of Tsai's Calibration Results

Recommendation: use many **more than 6 points** (ideally more than 20) **and non coplanar**



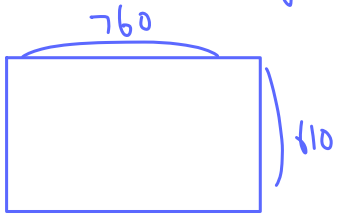
Corners can be detected with accuracy < 0.1 pixels

α_u, α_v: focal lengths (pixel) u_0, v_0: principal points (pixel)

K_{12}: skew parameter

α_u	α_u/α_v	K_{12}	u_0	v_0	Average Reprojection error
1673.3	1.0063	1.39	379.96	305.78	0.365

$$\begin{pmatrix} \alpha_u & K_{12} & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}$$

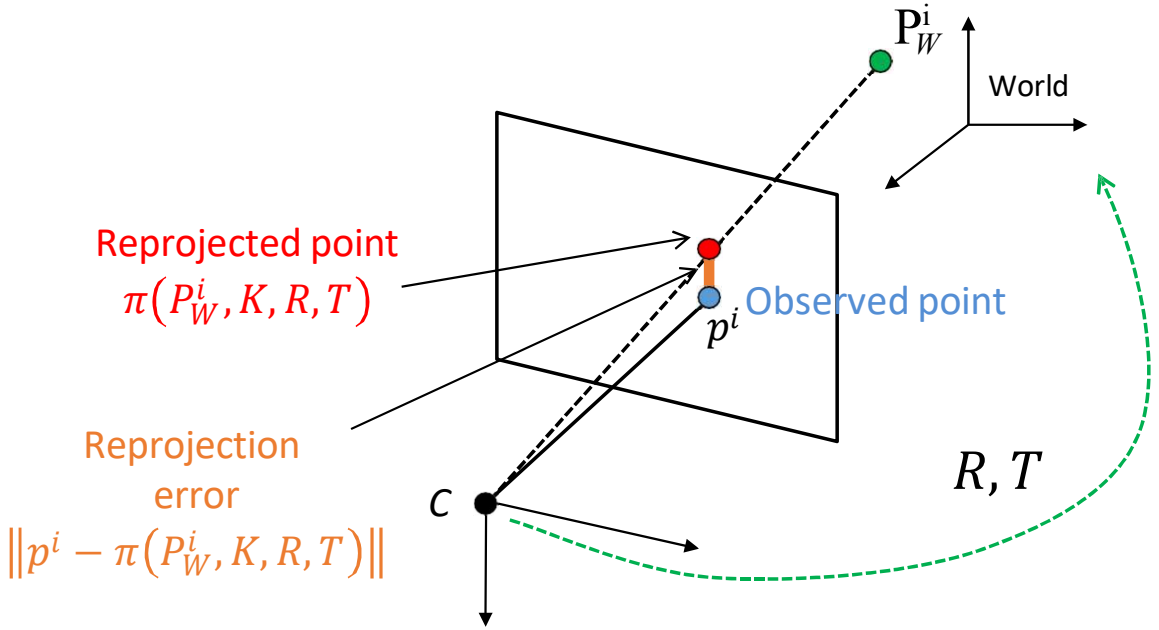


How can we estimate the lens distortion parameters?

How can we enforce $\alpha_u = \alpha_v$ and $K_{12} = 0$?

Reprojection Error

- The reprojection error is the **Euclidean distance** (in pixels) between an **observed image point** and the **corresponding** 3D point **reprojected** onto the camera frame.
- The reprojection error gives us a **quantitative measure of the accuracy** of the calibration (**ideally it should be zero**).



u, v

assumption.

this is accurate

u, v

observed

minimize

u', v'

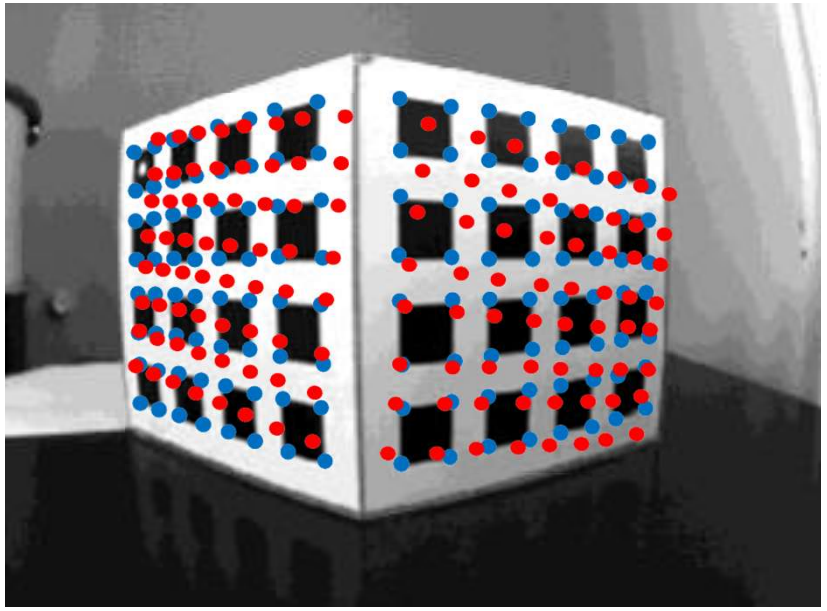
reprojected

Reprojection Error

- The reprojection error can be used to assess the quality of the camera calibration
- What are the sources of the reprojection error? -> limitation of DLT

noise of observation.

1. Lens distortion
2. Errors in corner detections
3. Least square optimization is prone to noise



Control points (observed points)

Reprojected points $\pi\left(P_W^i, K, R, T\right)$

Non-Linear Calibration Refinement

Tsai로 calibration 구하고 이걸 minimize,

- The calibration parameters K, R, T determined by the DLT can be refined by minimizing the following cost:

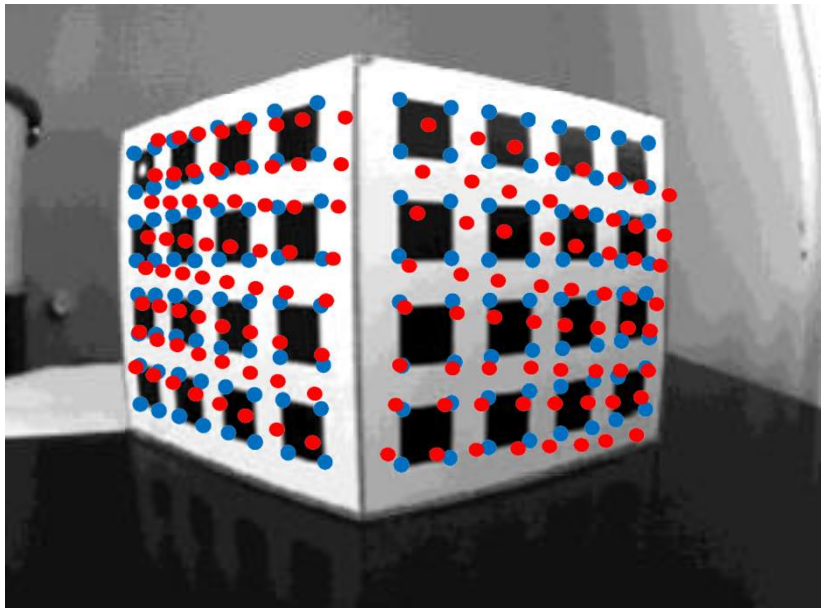
$$K, R, T, lens\ distortion = argmin_{K,R,T,lens} \sum_{i=1}^{n} \left\| p^i - \pi\left(P_W^i, K, R, T\right) \right\|^2$$

minimize

- This time we also include the **lens distortion** (can be set to 0 for initialization)
- Can be minimized using **Levenberg–Marquardt** (more robust than Gauss-Newton to local minima)

Hessian

Jacobian (SGD)



Control points (observed points)

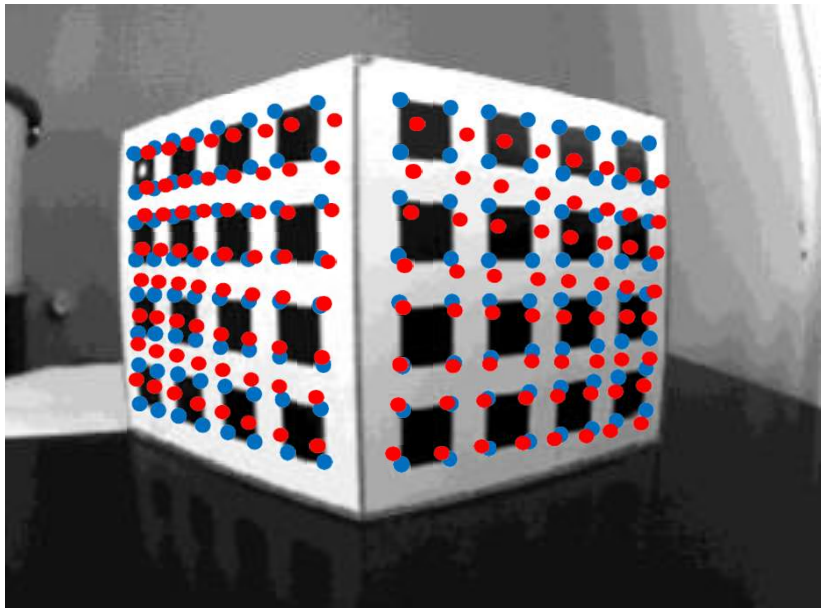
Reprojected points $\pi\left(P_W^i, K, R, T\right)$

Non-Linear Calibration Refinement

- The calibration parameters K, R, T determined by the DLT can be refined by minimizing the following cost:

$$K, R, T, lens\ distortion = argmin_{K,R,T,lens} \sum_{i=1}^{n} \left\| p^i - \pi\left(P_W^i, K, R, T\right) \right\|^2$$

- This time we also include the **lens distortion** (can be set to 0 for initialization)
- Can be minimized using **Levenberg–Marquardt** (more robust than Gauss-Newton to local minima)



Control points (observed points)

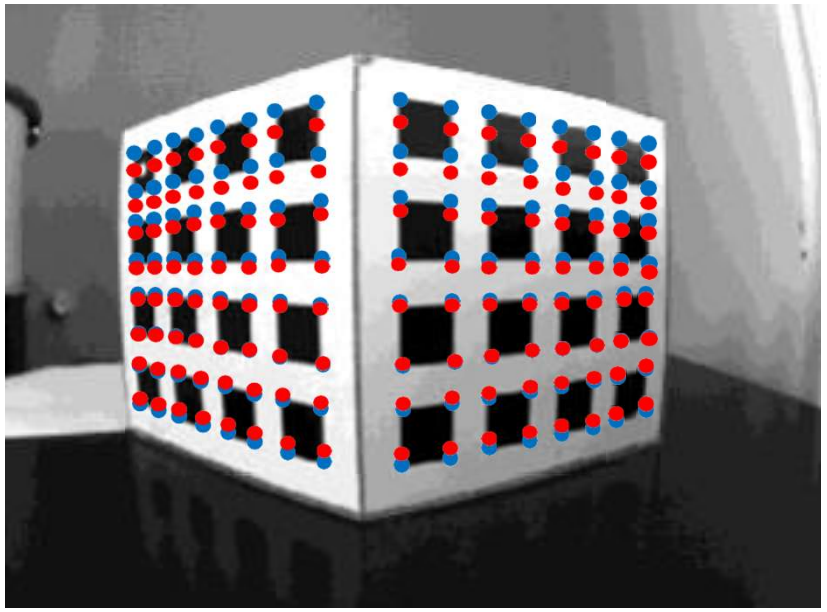
Reprojected points $\pi\left(P_W^i, K, R, T\right)$

Non-Linear Calibration Refinement

- The calibration parameters K, R, T determined by the DLT can be refined by minimizing the following cost:

$$K, R, T, lens\ distortion = argmin_{K,R,T,lens} \sum_{i=1}^{n} \left\| p^i - \pi\left(P_W^i, K, R, T\right) \right\|^2$$

- This time we also include the **lens distortion** (can be set to 0 for initialization)
- Can be minimized using **Levenberg–Marquardt** (more robust than Gauss-Newton to local minima)



Control points (observed points)

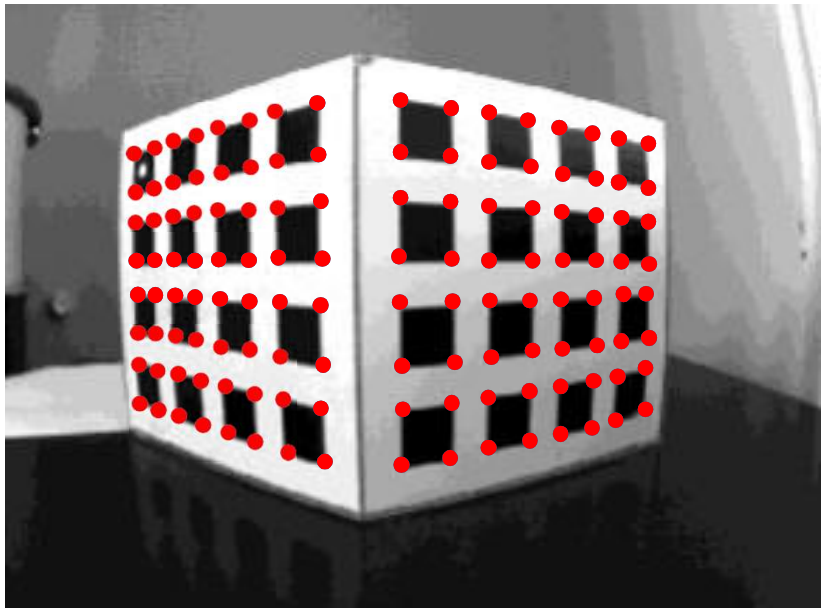
Reprojected points $\pi\left(P_W^i, K, R, T\right)$

Non-Linear Calibration Refinement

- The calibration parameters K, R, T determined by the DLT can be refined by minimizing the following cost:

$$K, R, T, lens\ distortion = argmin_{K,R,T,lens} \sum_{i=1}^{n} \left\| p^i - \pi\left(P_W^i, K, R, T\right) \right\|^2$$

- This time we also include the **lens distortion** (can be set to 0 for initialization)
- Can be minimized using **Levenberg–Marquardt** (more robust than Gauss-Newton to local minima)

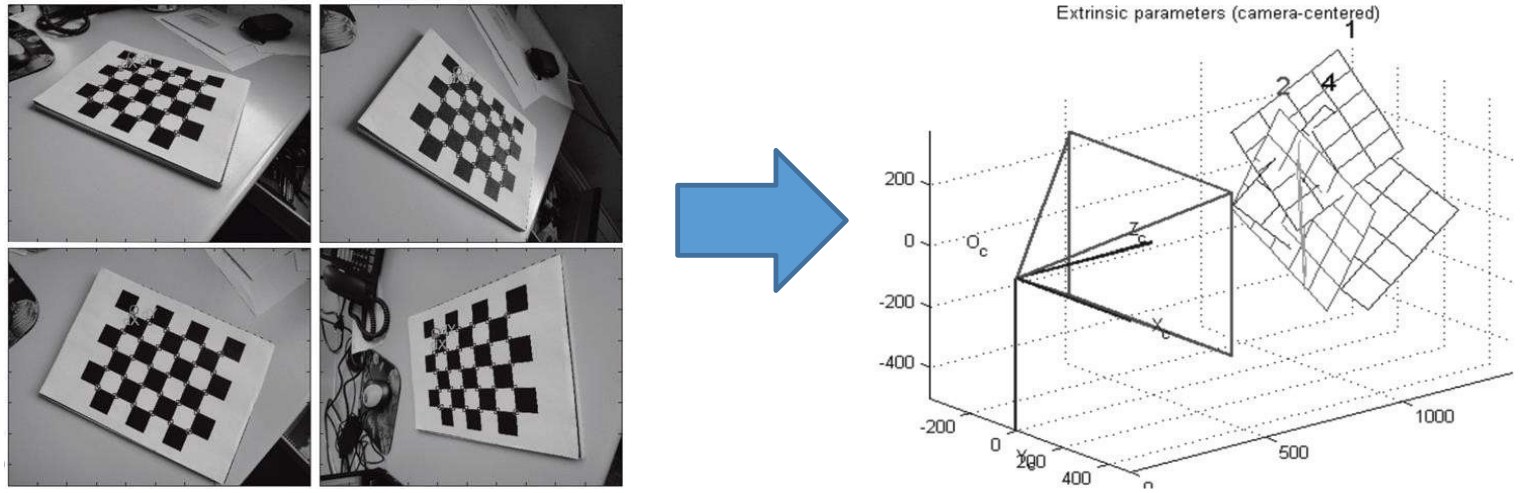


Control points (observed points)

Reprojected points $\pi\left(P_W^i, K, R, T\right)$

Zhang's Algorithm: Calibration from Planar Grids

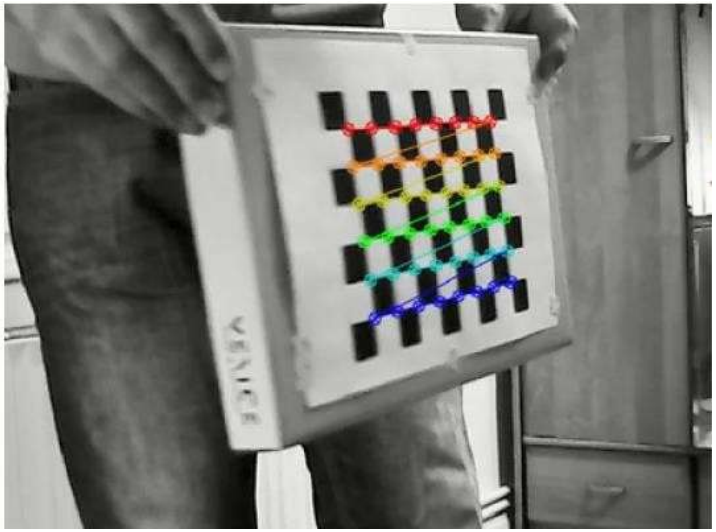
- **Tsai's calibration** requires that the world's 3D points are non-coplanar, which is **not very practical**
- **Today's camera calibration toolboxes** (Matlab, OpenCV) use **multiple views** of a **planar grid** (e.g., a checker board)
- They are based on a method developed in 2000 by Zhang (Microsoft Research)



Zhang, A flexible new technique for camera calibration, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000. PDF.

Zhang's Algorithm: Calibration from Planar Grids

- **Tsai's calibration** requires that the world's 3D points are non-coplanar, which is **not very practical**
- **Today's camera calibration toolboxes** (Matlab, OpenCV) use **multiple views** of a **planar grid** (e.g., a checker board)
- They are based on a method developed in 2000 by Zhang (Microsoft Research)



Zhang, A flexible new technique for camera calibration, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000. PDF.

Applying the Direct Linear Transform (DLT) algorithm

As in Tsai's method, we start by writing the perspective projection equation (again, we neglect the radial distortion). However, in **Zhang's method the points are all coplanar**, i.e., $\boldsymbol{Z_w = 0}$, and thus we can write:

in World Coord system

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K\left[R \mid T\right] \cdot \begin{bmatrix} X_w \\ Y_w \\ 0 \\ 1 \end{bmatrix} \Rightarrow$$

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ 0 \\ 1 \end{bmatrix}$$

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix}$$

Applying the Direct Linear Transform (DLT) algorithm

As in Tsai's method, we start by writing the perspective projection equation (again, we neglect the radial distortion). However, in **Zhang's method the points are all coplanar**, i.e., $\boldsymbol{Z_w = 0}$, and thus we can write:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} R \mid T \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ 0 \\ 1 \end{bmatrix} \Rightarrow$$

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ 0 \\ 1 \end{bmatrix}$$

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix}$$

Applying the Direct Linear Transform (DLT) algorithm

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix}$$

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = H \cdot \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix}$$

This matrix is called Homography

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} h_1^{\mathrm{T}} \\ h_2^{\mathrm{T}} \\ h_3^{\mathrm{T}} \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix}$$

where h_i^{T} is the i-*th* row of H

Applying the Direct Linear Transform (DLT) algorithm

$$\Rightarrow \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} h_1^{\mathrm{T}} \\ h_2^{\mathrm{T}} \\ h_3^{\mathrm{T}} \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix} \rightarrow P$$

Conversion back from homogeneous coordinates to pixel coordinates leads to:

$$u = \frac{\lambda u}{\lambda} = \frac{h_1^{\mathrm{T}} \cdot P}{h_3^{\mathrm{T}} \cdot P}$$
$$v = \frac{\lambda v}{\lambda} = \frac{h_2^{\mathrm{T}} \cdot P}{h_3^{\mathrm{T}} \cdot P}$$
$$\Rightarrow \quad (h_1^{\mathrm{T}} - u_i h_3^{\mathrm{T}}) \cdot P_i = 0$$
$$(h_2^{\mathrm{T}} - v_i h_3^{\mathrm{T}}) \cdot P_i = 0$$

Applying the Direct Linear Transform (DLT) algorithm

- By re-arranging the terms, we obtain:

$$\begin{aligned}(h_1^{\mathrm{T}} - u_i h_3^{\mathrm{T}}) \cdot P_i = 0 \\ (h_2^{\mathrm{T}} - v_i h_3^{\mathrm{T}}) \cdot P_i = 0\end{aligned} \quad \Rightarrow \quad \begin{aligned}P_i^{\mathrm{T}} \cdot h_1 + 0 \cdot h_2^{\mathrm{T}} - u_i P_i^{\mathrm{T}} \cdot h_3^{\mathrm{T}} = 0 \\ 0 \cdot h_1^{\mathrm{T}} + P_i^{\mathrm{T}} \cdot h_2^{\mathrm{T}} - v_i P_i^{\mathrm{T}} \cdot h_3^{\mathrm{T}} = 0\end{aligned} \quad \Rightarrow \begin{pmatrix} P_i^{\mathrm{T}} & 0^{\mathrm{T}} & -u_1 P_i^{\mathrm{T}} \\ 0^{\mathrm{T}} & P_i^{\mathrm{T}} & -v_1 P_i^{\mathrm{T}} \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

- For n points (from a **single view**), we can stack all these equations into a big matrix:

$$\begin{pmatrix} P_1^{\mathrm{T}} & 0^{\mathrm{T}} & -u_1 P_1^{\mathrm{T}} \\ 0^{\mathrm{T}} & P_1^{\mathrm{T}} & -v_1 P_1^{\mathrm{T}} \\ \cdots & \cdots & \cdots \\ P_n^{\mathrm{T}} & 0^{\mathrm{T}} & -u_n P_n^{\mathrm{T}} \\ 0^{\mathrm{T}} & P_n^{\mathrm{T}} & -v_n P_n^{\mathrm{T}} \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix} \quad \Rightarrow \quad \mathrm{Q} \cdot \mathrm{H} = 0$$

Q (this matrix is **known**) H (this matrix is **unknown**)

Applying the Direct Linear Transform (DLT) algorithm

$$\mathrm{Q}\cdot\mathrm{H}=0$$

$M \in R^{2n \times 12}$

$H \in R^{2n \times 9}$

Minimal solution

- $Q_{(2n\times 9)}$ should have rank 8 to have a unique (up to a scale) non-trivial solution H
- Each point correspondence provides 2 independent equations x, y.
- Thus, a minimum of **4 non-collinear points** is required $8/2 = 4$

Over-determined solution

- $n \geq 4$ points
- It can be solved through Singular Value Decomposition (SVD) (same considerations as before)

How to recover K, R, T

- H can be decomposed by recalling that:

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix}$$

- Differently from Tsai's, the decomposition of H into K, R, T requires **at least two views**
- In practice **the more views the better**, e.g., 20-50 views spanning the entire field of view of the camera for the best calibration results
- Notice that now **each view j has a different homography H^j** (and so a different R^j and T^j). However, **$\boldsymbol{K}$ is the same for all views**:

$$\begin{bmatrix} h_{11}^j & h_{12}^j & h_{13}^j \\ h_{21}^j & h_{22}^j & h_{23}^j \\ h_{31}^j & h_{33}^j & h_{33}^j \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11}^j & r_{12}^j & t_1^j \\ r_{21}^j & r_{22}^j & t_2^j \\ r_{31}^j & r_{32}^j & t_3^j \end{bmatrix}$$

How to recover K, R, T from H and from multiple views?

Details won't be asked at the exam ☺

1. Estimate the homography H_i for each i-th view using the DLT algorithm.

2. Determine the intrinsics K of the camera from a set of homographies:
   1. Each homography $H_i \sim K(\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{t})$ provides two *linear* equations in the 6 entries of the matrix $\mathrm{B} \coloneqq K^{-\mathrm{T}}K^{-1}$. Letting $\boldsymbol{w}_1 \coloneqq K\boldsymbol{r}_1, \boldsymbol{w}_2 \coloneqq K\boldsymbol{r}_2$, the rotation constraints $\boldsymbol{r}_1^{\mathrm{T}}\boldsymbol{r}_1 = \boldsymbol{r}_2^{\mathrm{T}}\boldsymbol{r}_2 = 1$ and $\boldsymbol{r}_1^{\mathrm{T}}\boldsymbol{r}_2 = 0$ become $\boldsymbol{w}_1^{\mathrm{T}}B\boldsymbol{w}_1 - \boldsymbol{w}_2^{\mathrm{T}}B\boldsymbol{w}_2 = 0$ and $\boldsymbol{w}_1^{\mathrm{T}}B\boldsymbol{w}_2 = 0$.
   2. Stack $2N$ equations from N views, to yield a linear system $A\boldsymbol{b} = \boldsymbol{0}$. Solve for $\boldsymbol{b}$ (i.e., B) using the Singular Value Decomposition (SVD).
   3. Use Cholesky decomposition to obtain K from B.

3. The extrinsic parameters for each view can be computed using K:
   $\boldsymbol{r}_1 \sim \lambda K^{-1}H_i(:,1)$, $\boldsymbol{r}_2 \sim \lambda K^{-1}H_i(:,2)$, $\boldsymbol{r}_3 = \boldsymbol{r}_1 \times \boldsymbol{r}_2$ and $T_i = \lambda K^{-1}H_i(:,3)$, with $\lambda = 1/K^{-1}H_i(:,1)$.
   Finally, build $R_i = (\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3)$ and enforce rotation matrix constraints.

R은 orthogonal matrix 이므로.

$r_1 = k^{-1}h_1$, $r_2 = k^{-1}h_2$

$r_1^T r_2 = (k^{-1}h_1)^T(k^{-1}h_2) = h_1^T k^{-T}k^{-1}h_2 = 0$

$= h_1^T B h_2 = 0$

$\|h_1\|^2 = \|r_2\|^2$

$\Rightarrow (k^{-1}h_1)^T(k^{-1}h_1) = h_1^T B h_1 = h_2^T B h_2$

$h_i^T B h_j = \sum_{m=1}^{3}\sum_{n=1}^{3} h_{im} B_{mn} h_{jn} = v_{ij}^T b$

$B_{mn} = B_{nm}$

$B = k^{-T}k^{-1}$

k는 square matrix $k^{-T} = (k^{-1})^T$

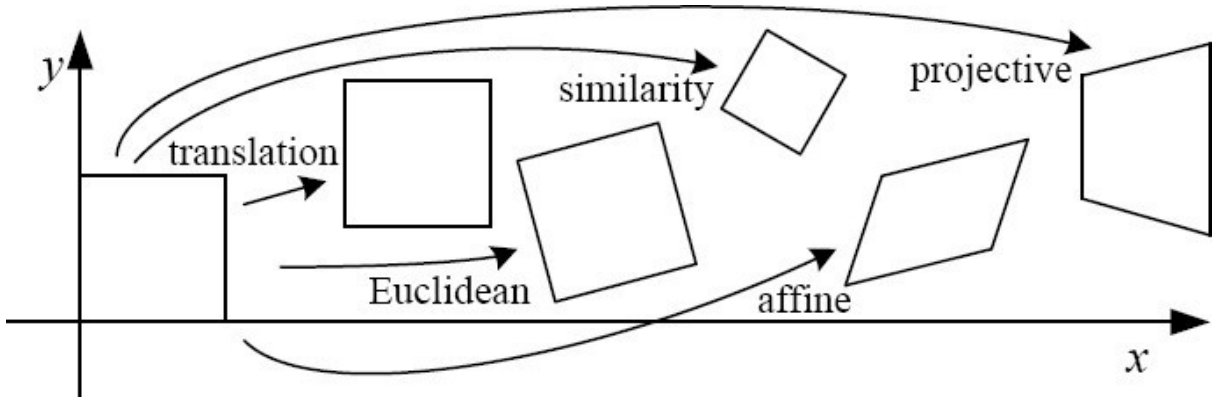
$B^T = (k^{-T}k^{-1})^T = k^{-T}k^{-1} = B$.

$B^T = B$ 면 대칭행렬.

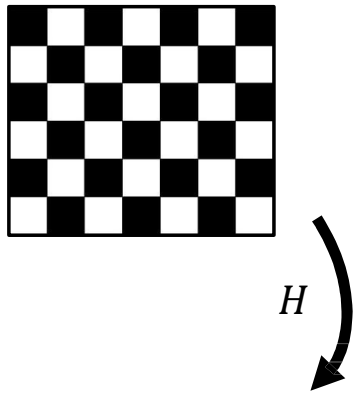
B는 대칭행렬이므로 $\begin{bmatrix} B_{11} & B_{12} & B_{13} \\ & B_{22} & B_{23} \\ & & B_{33} \end{bmatrix}$ 해만 알면 된다

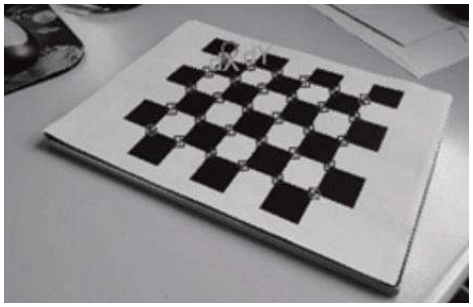
$b = [B_{11}, B_{12}, B_{22}, B_{13}, B_{23}, B_{33}]^T$

Types of 2D Transformations (from Lecture 2)



Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\left[\, \boldsymbol{I} \,\middle|\, \boldsymbol{t} \,\right]_{2\times3}$	2	orientation $+\cdots$	
rigid (Euclidean)	$\left[\, \boldsymbol{R} \,\middle|\, \boldsymbol{t} \,\right]_{2\times3}$	3	lengths $+\cdots$	
similarity	$\left[\, s\boldsymbol{R} \,\middle|\, \boldsymbol{t} \,\right]_{2\times3}$	4	angles $+\cdots$	
affine	$\left[\, \boldsymbol{A} \,\right]_{2\times3}$	6	parallelism $+\cdots$	
projective	$\left[\, \tilde{\boldsymbol{H}} \,\right]_{3\times3}$	8	straight lines	





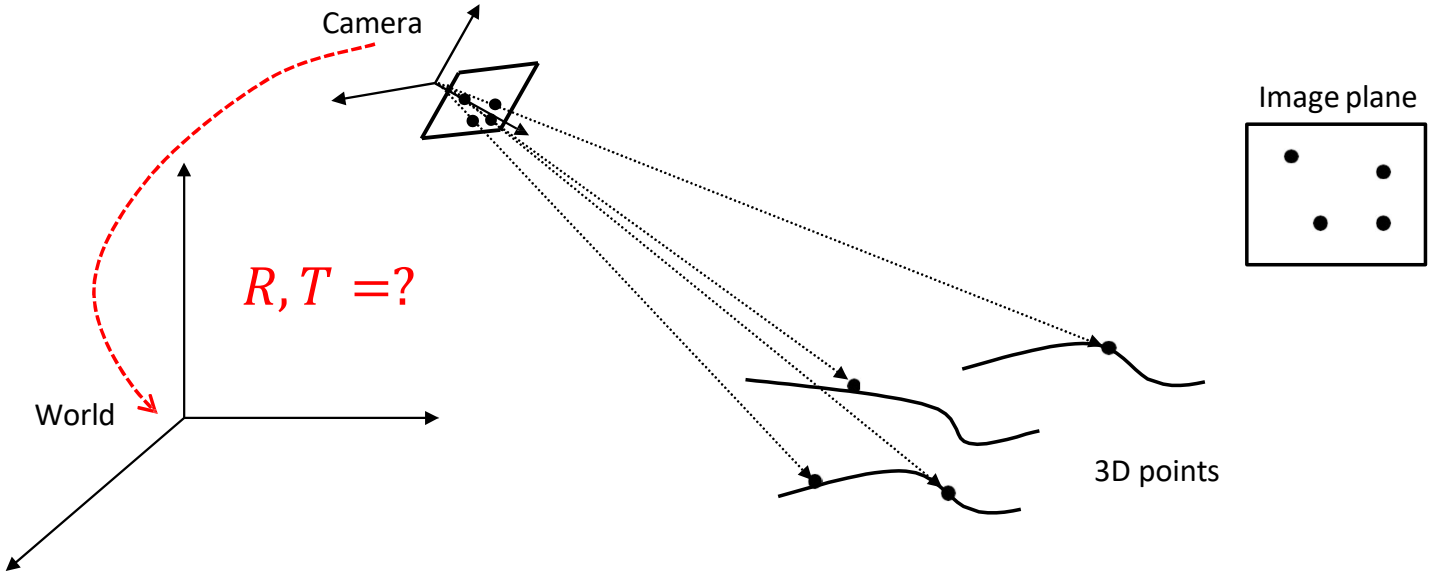
This matrix is called **Homography**

2. Camera localization

Camera Localization (or Perspective from n Points: PnP)

Degree of Freedom

- This is the problem of determining the **6DoF pose of a camera** (position and orientation) with respect to the world frame **from a set of 3D-2D point correspondences**.
- It assumes the **camera** to be **already calibrated**
- **In other words, the goal is getting extrinsics (R and T) while intrinsics are given**
- The **DLT can be used** to solve this problem **but is suboptimal**. We want to study **algebraic solutions** to the problem.



How Many Points are Enough?

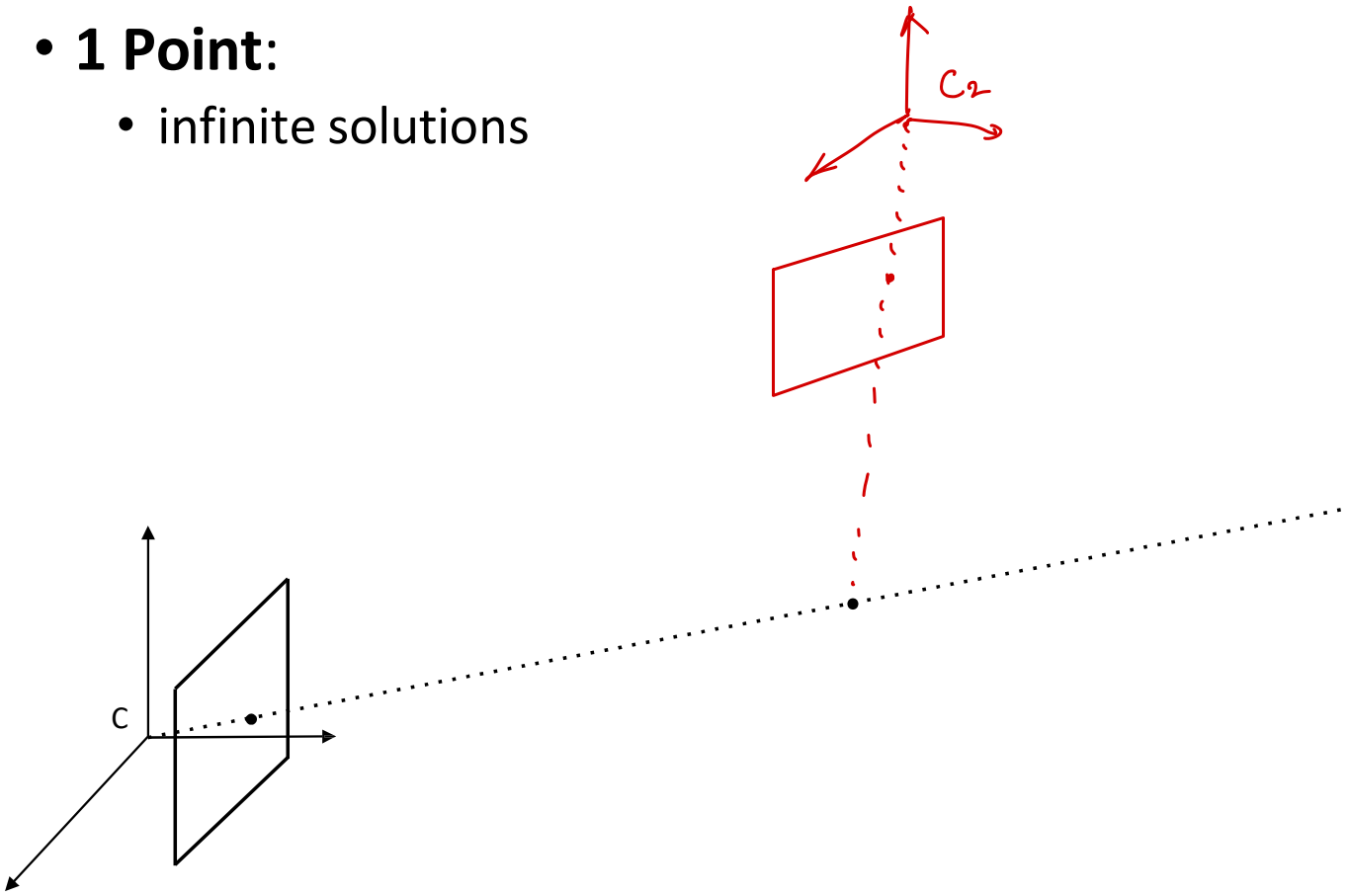
- Here, ‘points’ means correspondence between 3D and 2D of points
- E.g., ‘2 points’ means we know 1) 3D and 2D positions of point A and 2) 3D and 2D positions of point B

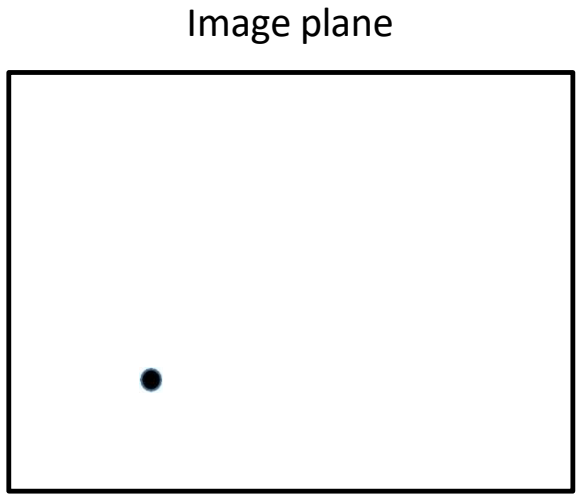
- **1 Point**:
  - infinite solutions
- **2 Points**:
  - infinitely many solutions, but bounded
- **3 Points** (non collinear):
  - up to 4 solution
- **4 Points**:
  - Unique solution

Zhang

1 Point

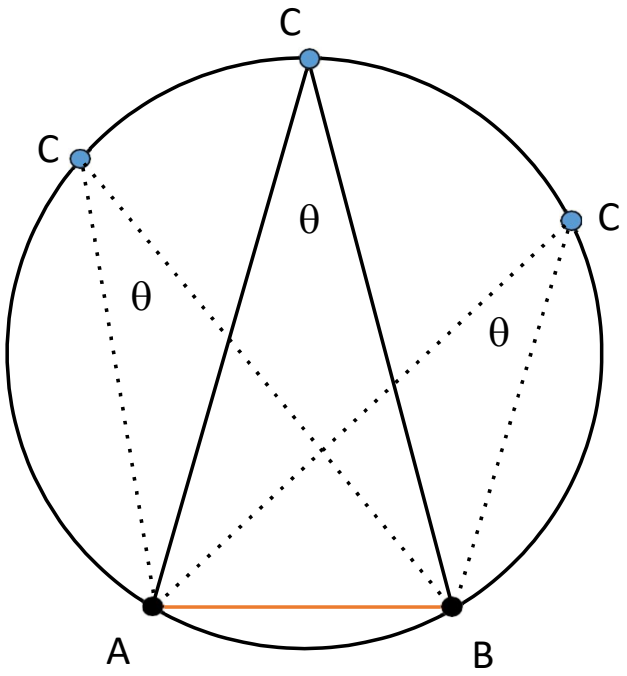
- **1 Point**:
  - infinite solutions





2 Points

- **2 Points**:
  - infinite solutions, but bounded



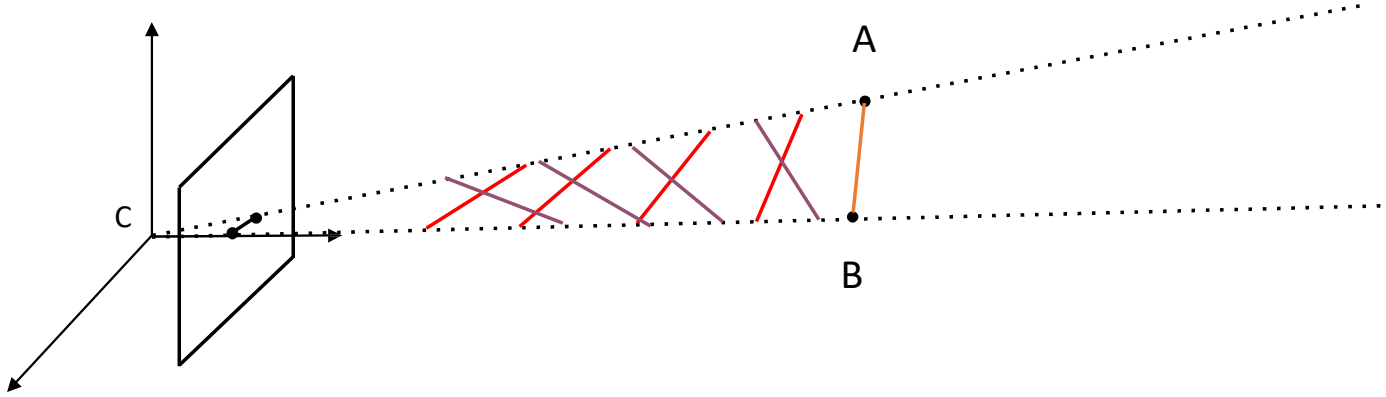
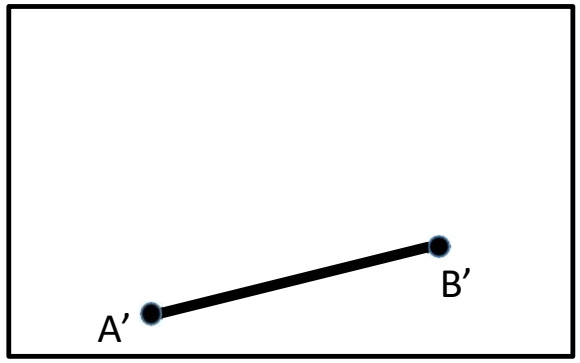


Image plane



3 Points (P3P problem)

- **3 Points** (non collinear):
  - up to 4 solution
  - $L_{A,}\, L_{B,}$ and L_C are unknown variables

From Carnot's Theorem:

$$|AB|^2 = s_1^2 = L_B^2 + L_A^2 - 2L_B L_A \cos\theta_{AB}$$

$$|AC|^2 = s_2^2 = L_A^2 + L_C^2 - 2L_A L_C \cos\theta_{AC}$$

$$|BC|^2 = s_3^2 = L_B^2 + L_C^2 - 2L_B L_C \cos\theta_{BC}$$

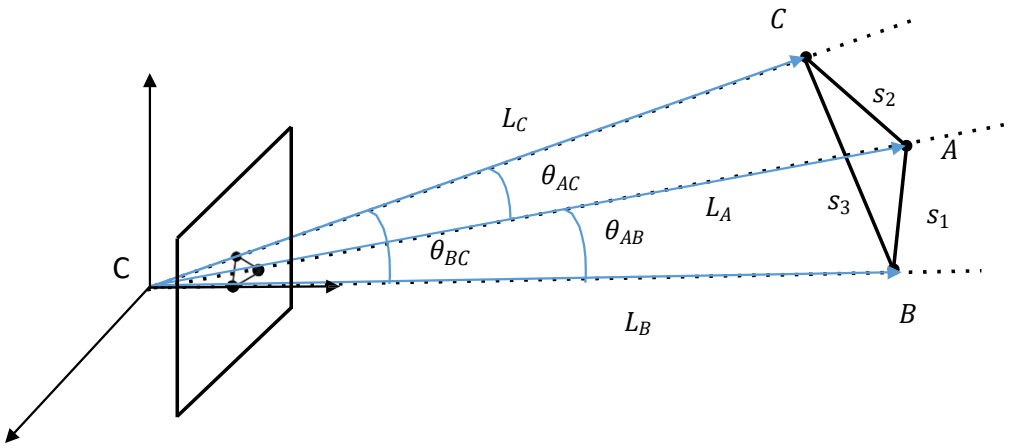
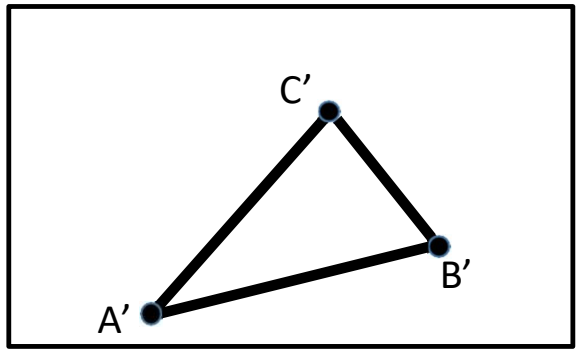


Image plane



Algebraic Approach: reduce to 4th order equation

$$||AB||^2 = s_1{}^2 = L_B{}^2 + L_A{}^2 - 2L_B L_A \cos\theta_{AB}$$
$$||AC||^2 = s_2{}^2 = L_A{}^2 + L_C{}^2 - 2L_A L_C \cos\theta_{AC}$$
$$||BC||^2 = s_3{}^2 = L_B{}^2 + L_C{}^2 - 2L_B L_C \cos\theta_{BC}$$

- It is known that $\boldsymbol{n}$ **independent polynomial equations**, in $\boldsymbol{n}$ **unknowns**, can have no more **solutions** than the **product of their respective degrees**. Thus, the system can have a maximum of **(2x2x2=8)** solutions. However, because every term in the system is either a constant or of second degree, for every real positive solution **there is a negative solution**.
- Thus, with 3 points, there are at most **(2x2x2/2=4) valid (positive) solutions**.

L_A L_B L_C ; positive.

M. A.Fischler and R. C.Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Graphics and Image Processing, 1981. PDF.

Algebraic Approach: reduce to 4th order equation

$$s_1^2 = L_B^2 + L_A^2 - 2L_B L_A \cos\theta_{AB}$$

$$s_2^2 = L_A^2 + L_C^2 - 2L_A L_C \cos\theta_{AC}$$

$$s_3^2 = L_B^2 + L_C^2 - 2L_B L_C \cos\theta_{BC}$$

- By defining $\boldsymbol{x = L_B/L_A}$, it can be shown that the system can be reduced to a 4th order equation:

$$G_0 + G_1 x + G_2 x^2 + G_3 x^3 + G_4 x^4 = 0$$

How can we disambiguate the 4 solutions? How do we determine R and T?

- A 4th point can be used to disambiguate the solutions. A classification of the four solutions and the determination of R and T from the point distances was given Gao's algorithm, implemented in OpenCV (*solvePnP_P3P*)
- In real worlds, we have more than three points, so use one of them to get only one (R,T) pair out of four pairs.

Gao, Hou, Tang, Cheng. Complete Solution Classification for the Perspective-Three-Point Problem. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2003. PDF.

Modern Solution to P3P

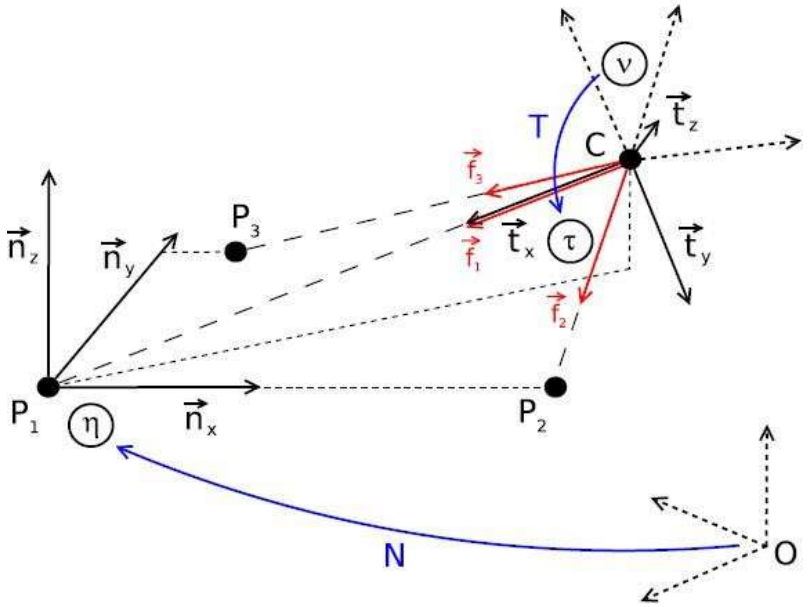
Details won't be asked at the exam ☺

modern은 다른 strategy 사용한다~.

A more **modern version of P3P** was developed by Kneip in 2011 and **directly solves for the camera's pose** (not distances from the points). This solution inspired the algorithm currently used in OpenCV (*solvePnP_AP3P*), by Ke'17, which consists of two steps:

1. Eliminate the camera's position and the features' distances to yield a system of 3 equations in *the camera's orientation alone*.
2. Successively eliminate two of the unknown 3-DOFs (angles) algebraically and arrive at a ***quartic polynomial equation***.

- Outperforms previous methods in terms of speed, accuracy, and robustness to close-to-singular cases.



Kneip, Scaramuzza, Siegwart. A Novel Parameterization of the Perspective-Three-Point Problem for a Direct Computation of Absolute Camera Position and Orientation. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2011. PDF.

Ke, Roumeliotis. An Efficient Algebraic Solution to the Perspective-Three-Point Problem. CVPR'17. PDF.

Solution to PnP for $n \geq 4$

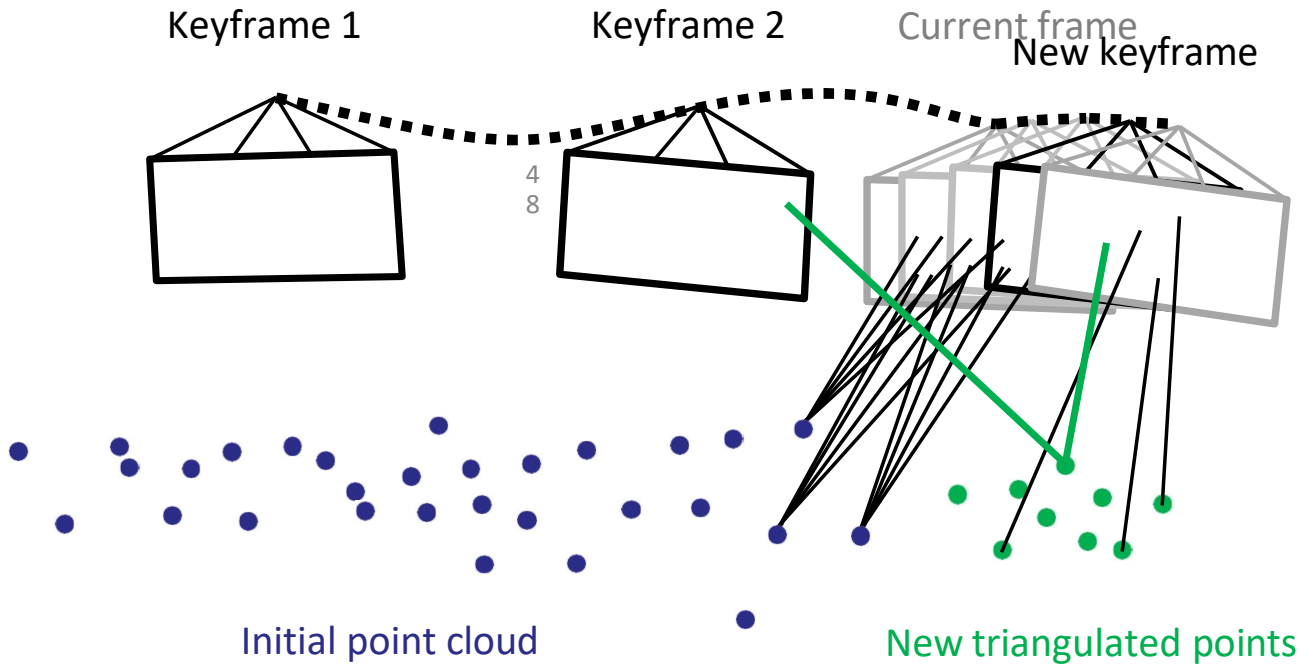
Details won't be asked at the exam ☺

An efficient algebraic solution to the PnP problem for $n \geq 4$ was developed by Lepetit in 2009 and coined **EPnP** (Efficient PnP) and can be found in OpenCV (solvePnP_EPnP)

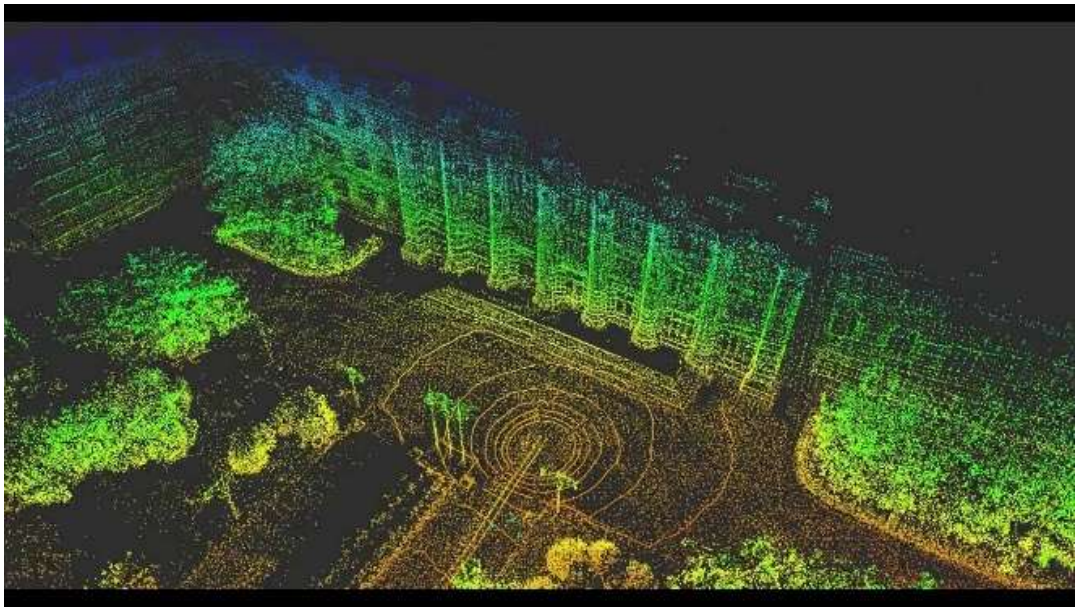
- EPnP expresses the *n* world's points as a weighted sum of **four virtual control points**
- The coordinates of these virtual control points become the **unknowns of the problem**, which can be solved in $O(n)$ time by solving a **constant number** of **quartic polynomial equations**
- The final pose of the camera is then solved from the control points

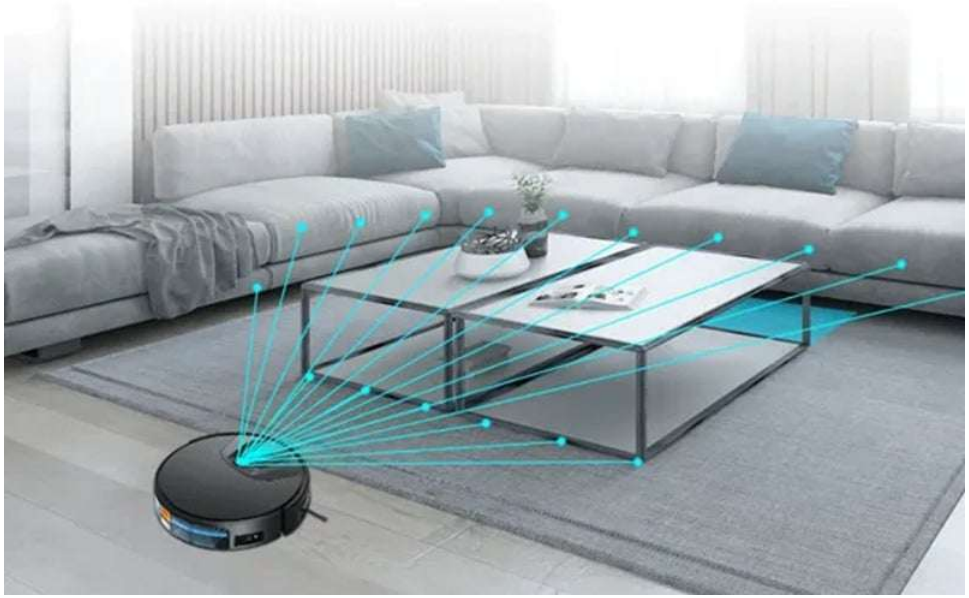
Lepetit, Moreno Noguer, Fua, EP*n*P: An Accurate *O(n)* Solution to the P*n*P Problem, International Journal of Computer Vision. PDF.

Application to Monocular Visual Odometry



SLAM
(Simultaneous Localization and Mapping)





Robust Estimation in Presence of Outliers

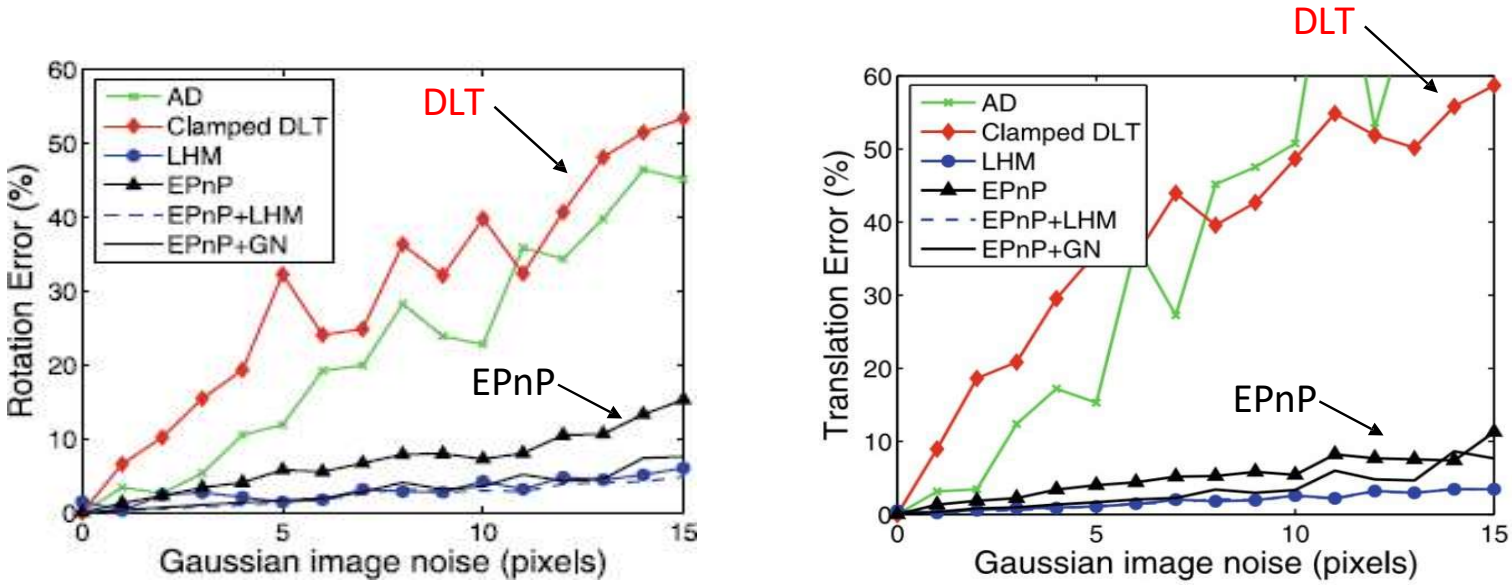
- **All P*n*P problems** (solved by DLT, EPnP, or P3P algorithms) are prone to errors if there are outliers in the set of 3D-2D point correspondences.
- The **RANSAC** algorithm can be used, in conjunction with the PnP algorithm, to **remove the outliers**.
- P*n*P with RANSAC can be found in OpenCV's (*solvePnPRansac*)

EPnP vs. DLT

If a camera is calibrated, only R and T need to be determined. In this case, **should we use DLT or EPnP**?

EPnP vs. DLT: Accuracy vs noise

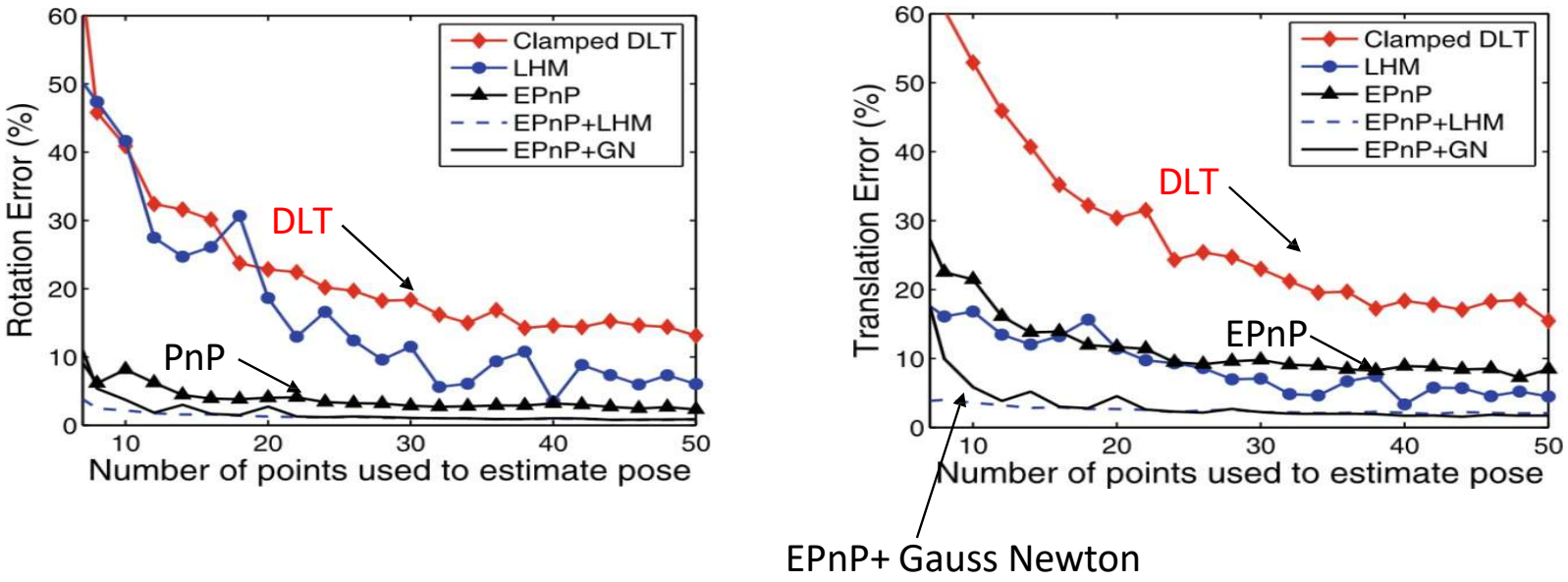
EPnP is more up to **10 × more accurate** and **more efficient** than DLT



Plots from
Lepetit, Moreno Noguer, Fua, EP*n*P: An Accurate *O(n)* Solution to the P*n*P Problem, International Journal of Computer Vision. PDF.

EPnP vs. DLT: Accuracy vs number of points

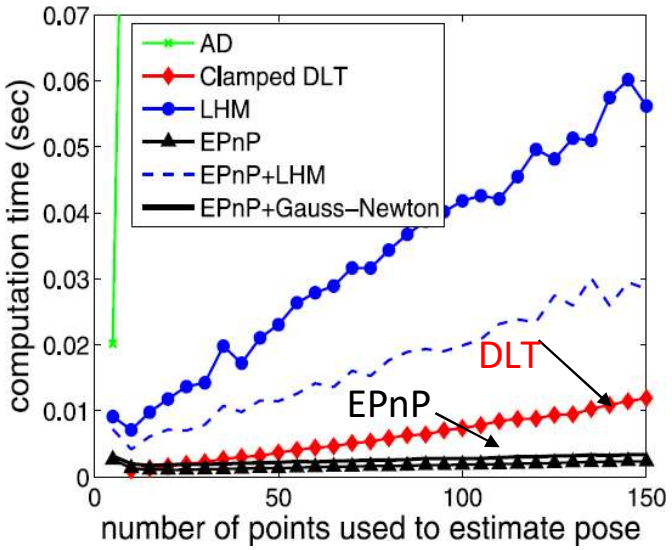
EPnP is more up to **10 × more accurate** and **more efficient** than DLT



Plots from
Lepetit, Moreno Noguer, Fua, EP*n*P: An Accurate *O(n)* Solution to the P*n*P Problem, International Journal of Computer Vision. PDF.

EPnP vs. DLT: Timing

EPnP is more up to $\mathbf{10\times}$ **more accurate** and **more efficient** than DLT



Plots from
Lepetit, Moreno Noguer, Fua, EP*n*P: An Accurate *O(n)* Solution to the P*n*P Problem, International Journal of Computer Vision. PDF.

PnP problem: Recap

Calibrated camera (i.e., instrinc parameters are known)	**Uncalibrated camera** (i.e., intrinsic parameters unknown)
Either DLT or EPnP can be used	**Only DLT** can be used

EPnP: minimum number of points: **3 (P3P) +1** for disambiguation
DLT: Minimum number of points: **4 if coplanar, 6 if non-coplanar**

The output of both DLT and EPnP can be refined via **non-linear optimization**
by minimizing the sum of squared reprojection errors

Real world cases

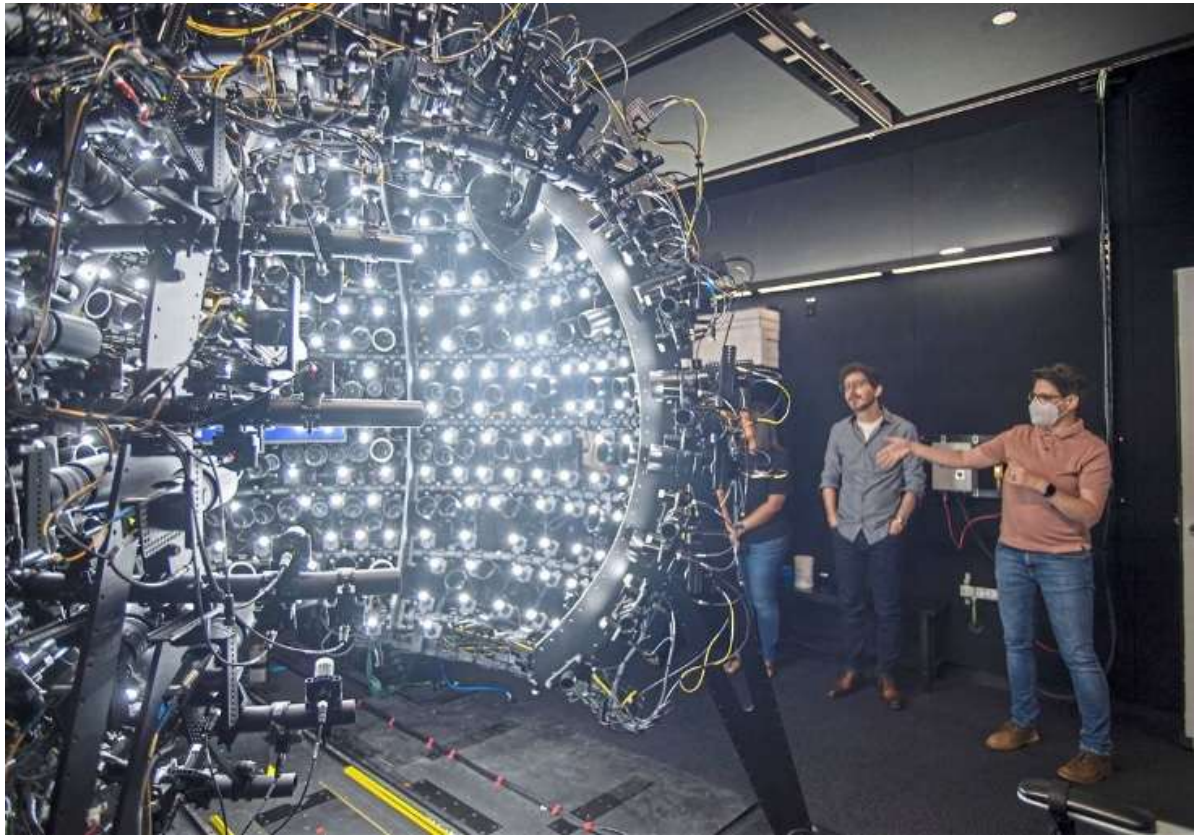
- Small capture setup with few Kinect cameras
- I and my colleagues made this to capture simple data quickly when I was in Meta

Real world cases

- Captured data with EMG wrist band

Real world cases

- Calibrating over 100 cameras…
- Calibrate cameras for every capture due to temperature/humidity changes



Mugsy studio in Meta

Real world cases

- We need to synchronize cameras in addition to calibrating them..!
- Tricky way: clap (!)
- Advanced way: use synchronized triggers