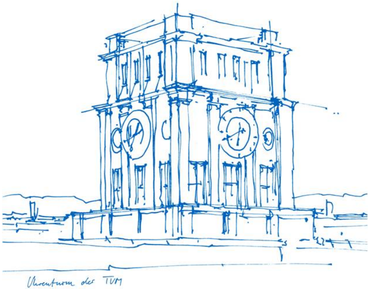
Computer Vision II: Multiple View Geometry (IN2228)

Chapter 07 3D-2D Geometry

Dr. Haoang Li

21 June 2023 12:00-13:30



Announcements before Class

- Reminder

✓ For 2D-2D geometry, due to time limit, we skip the case of **multiple views** (last year, Prof. Cremers also skipped this part).

✓ **Thus, we cancel the Exercise 7**.

Wed 14.06.2023 Exercise 6: Reconstruction from two views

Wed 21.06.2023 Exercise 7: Reconstruction from multiple views

Wed 05.07.2023 Exercise 8: Direct Image Alignment

Wed 12.07.2023 Exercise 9: Direct Image Alignment

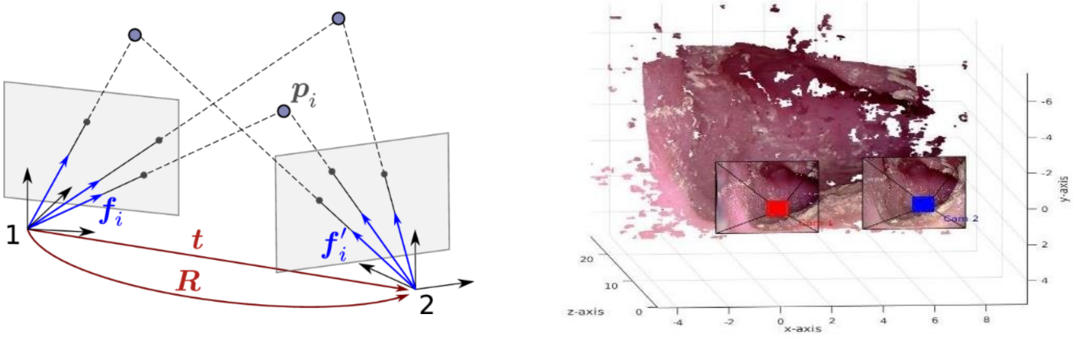
Today's Outline

- Overview of 3D-2D Geometry
- Definition of Perspective-n-Points (PnP)
- Classical Algorithms
- Advanced Algorithms
- Brief Introduction to Perspective-n-Lines (PnL)

Overview of 3D-2D Geometry

➢ Recap on Coordinate System

✓ Relative pose from the **right camera frame** to the **left camera frame**

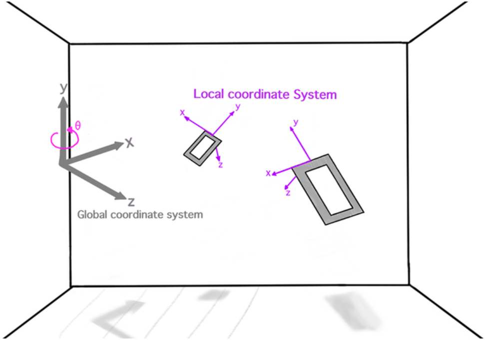


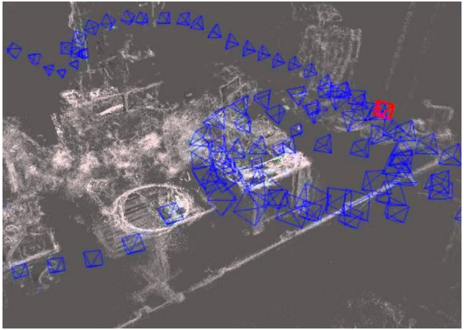
Left and right camera frames in VO/SLAM/SFM

Overview of 3D-2D Geometry

➢ Recap on Coordinate System

✓ Absolute pose from the **camera frame** to the **world frame**





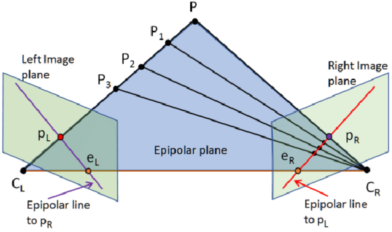
World frame and camera frames in VO/SLAM/SFM

Overview of 3D-2D Geometry

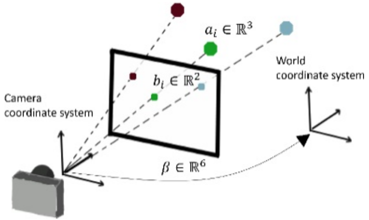
➢ Comparison Between 2D-2D Geometry and 3D-2D Geometry

✓ Different types of correspondences

- 2D-2D geometry: 2D-2D correspondences for **relative** camera pose estimation. It is NOT suitable to compute the absolute poses of sequential images since 1) it is time-consuming, and 2) the estimated translation is up-to-scale.
- 3D-2D geometry: 3D-2D correspondences for **absolute** camera pose estimation.



Relative camera pose

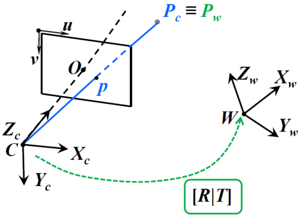


Absolute camera pose

Overview of 3D-2D Geometry

➢ Recap on Perspective Projection

✓ Perspective projection model and practical configuration



Extrinsic Parameters

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \underbrace{\boxed{K[R \mid T]}}_{\text{Projection Matrix (M)}} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

World frame

3D Point Cloud

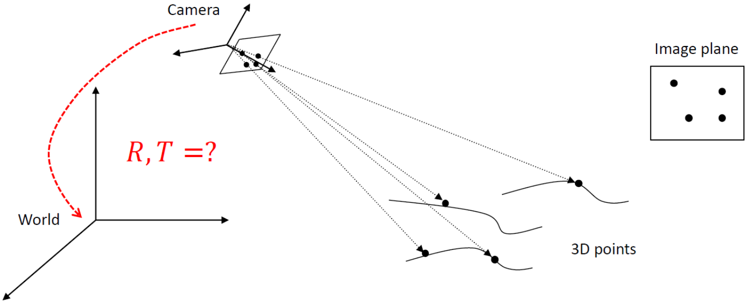
Two practical configurations:

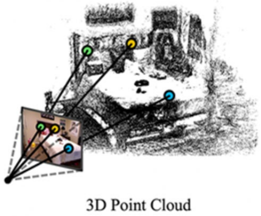
1. R,T is known. We use them to obtain 2D projections.
2. 2D projections (associated with 3D points) is known. We use them to **compute R, T** -> Our today's content

Definition of Perspective-n-Points

➢ Input and Output

- ✓ Perspective-n-Points (PnP) is to determine the **6-DoF absolute pose** of a camera (extrinsic parameters) with respect to the world frame, given a set of 3D-2D point correspondences.
- ✓ It assumes that the camera is already calibrated (i.e., we know its intrinsic parameters).

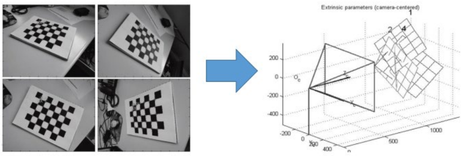




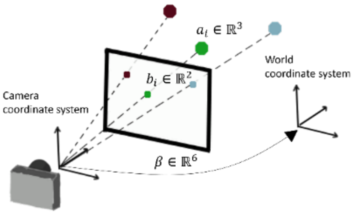
Definition of Perspective-n-Points

➢ Relationship with Camera Calibration

- ✓ Camera calibration focuses on "**simultaneous**" calibration of **extrinsic** and **intrinsic** parameters.
- ✓ PnP aims to only estimate the **extrinsic** parameters (with "known" intrinsic parameters), i.e., a camera localization problem.



Camera calibration
(multiple images)

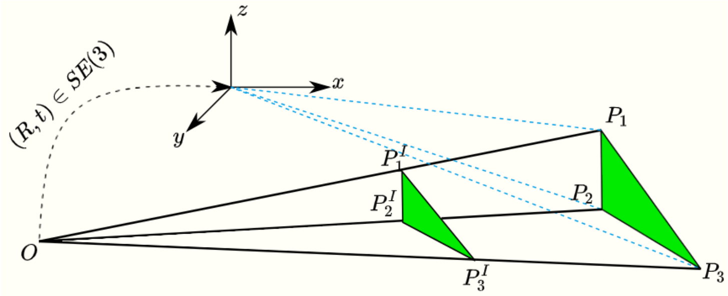


Camera localization
(a single image)

Definition of Perspective-n-Points

- Minimal Case

- ✓ 2 Points: a infinite number of solutions, but bounded
- ✓ **3 Points: minimal case**
- ✓ 4 Points: more reliable

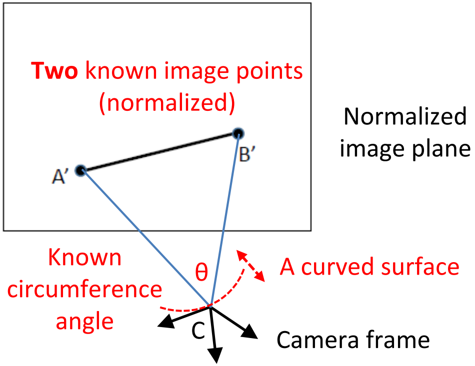


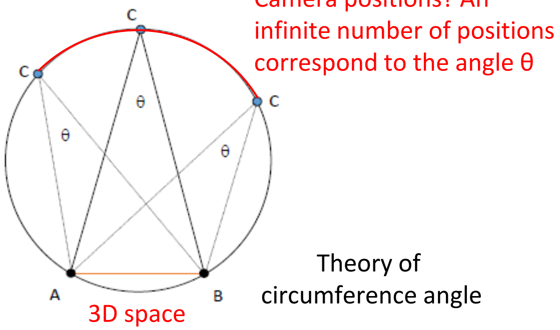
Definition of Perspective-n-Points

➢ Minimal Case

✓ Geometric illustration of 2-point case

Camera position has a infinite number of solutions.





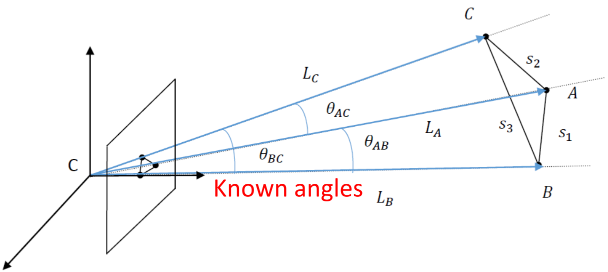
Definition of Perspective-n-Points

➢ Minimal Case

✓ Geometric illustration of 3-point case

Camera position can be determined (minimal case).

- The first and second curved surfaces intersect, forming a 3D curve.
- The 3D curve and the third curved surface intersect, forming a 3D point (camera center)



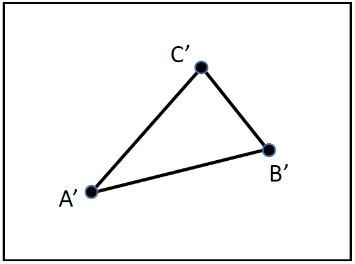


Image plane

Definition of Perspective-n-Points

$$\mathbf{a} \times \mathbf{b} = [\mathbf{a}]_\times \mathbf{b} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

- ➢ Minimal Case

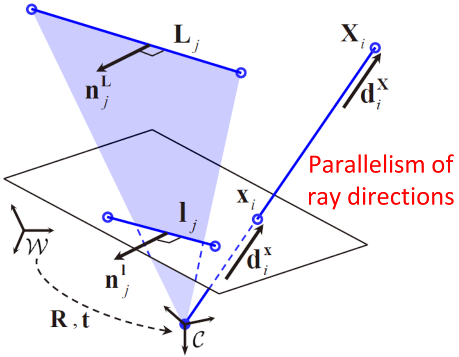
✓ Algebraic illustration

$$\mathbf{d}_i^{\mathbf{x}} \propto \mathbf{d}_i^{\mathbf{X}} \Rightarrow \underbrace{\mathbf{K}^{-1}\mathbf{x}_i}_{\text{3D vector}} \propto \underbrace{[\mathbf{R} \quad \mathbf{t}]\mathbf{X}_i}_{\text{3D vector}}$$

" $\propto$ " represents equality regardless of scale, i.e., two vectors are parallel, which leads to the cross product of 0.

A 3*3 skew-symmetric matrix has the rank of 2, so each 3D-2D point correspondence provide two constraints.

Camera pose has 6 DOF, so we need at least three point correspondences.



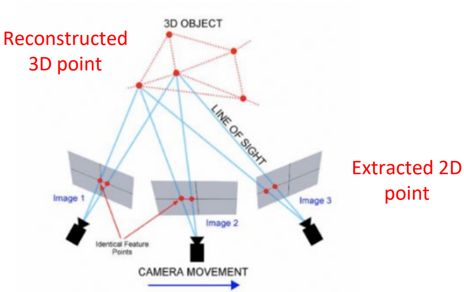
Camera frame

Definition of Perspective-n-Points

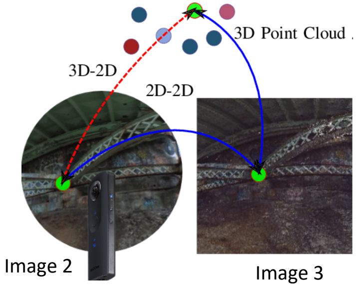
➢ 3D-2D Correspondence Establishment

✓ Generating 3D-2D correspondence based on 2D descriptor

- Mapping descriptor of 2D point to reconstructed 3D point
- Matching 3D point to 2D extracted point based on descriptor similarity
- We can also use prior camera pose to establish correspondences geometrically (introduced in the future)



3D point reconstruction



Correspondence establishment

Classical Algorithms

➢ Direct Linear Transformation (DLT)

✓ Recap on rewriting perspective projection

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

Calibration problem

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

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

Classical Algorithms

➢ Direct Linear Transformation (DLT)

Point normalization

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

✓ Linear constraint derivation

Express *s* based on the last row, and rewrite the first and second rows.

Normalized point

$$\boxed{s} \begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} t_1 & t_2 & t_3 & t_4 \\ t_5 & t_6 & t_7 & t_8 \\ t_9 & t_{10} & t_{11} & t_{12} \end{pmatrix}}_{[\mathbf{R}|\mathbf{t}]} \boxed{\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}}$$

⇨

$$\begin{cases} u_1 = \dfrac{t_1X + t_2Y + t_3Z + t_4}{\boxed{t_9X + t_{10}Y + t_{11}Z + t_{12}}} \\ v_1 = \dfrac{t_5X + t_6Y + t_7Z + t_8}{\boxed{t_9X + t_{10}Y + t_{11}Z + t_{12}}} \end{cases}$$

Classical Algorithms

- ➢ Direct Linear Transformation (DLT)

$$s\begin{pmatrix}u_1\\v_1\\1\end{pmatrix}=\underbrace{\begin{pmatrix}t_1 & t_2 & t_3 & t_4\\ t_5 & t_6 & t_7 & t_8\\ t_9 & t_{10} & t_{11} & t_{12}\end{pmatrix}}_{[\mathbf{R}|\mathbf{t}]}\begin{pmatrix}X\\Y\\Z\\1\end{pmatrix}$$

✓ Rewrite transformation matrix by row vectors

New vector definition $\mathbf{t}_1=(t_1,t_2,t_3,t_4)^T,\mathbf{t}_2=(t_5,t_6,t_7,t_8)^T,\mathbf{t}_3=(t_9,t_{10},t_{11},t_{12})^T.$

$$\begin{cases} u_1=\dfrac{t_1\boxed{X}+t_2\boxed{Y}+t_3\boxed{Z}+t_4}{t_9X+t_{10}Y+t_{11}Z+t_{12}}\\[2ex] v_1=\dfrac{t_5X+t_6Y+t_7Z+t_8}{t_9X+t_{10}Y+t_{11}Z+t_{12}}\end{cases}$$

Previous derivation result

$$\mathbf{t}_1^T\boxed{\mathbf{P}}-\mathbf{t}_3^T\boxed{\mathbf{P}}u_1=0,$$

P is in homogeneous coordinates

$$\mathbf{t}_2^T\mathbf{P}-\mathbf{t}_3^T\mathbf{P}v_1=0.$$

Classical Algorithms

➢ Direct Linear Transformation (DLT)

$$s\begin{pmatrix}u_1\\v_1\\1\end{pmatrix}=\underbrace{\boxed{\begin{pmatrix}t_1 & t_2 & t_3 & t_4\\ t_5 & t_6 & t_7 & t_8\\ t_9 & t_{10} & t_{11} & t_{12}\end{pmatrix}}}_{[\mathbf{R}|\mathbf{t}]}\begin{pmatrix}X\\Y\\Z\\1\end{pmatrix}$$

✓ Generate a linear system

$$\mathbf{t}_1^T\mathbf{P}-\mathbf{t}_3^T\mathbf{P}u_1=0,$$

$$\mathbf{t}_2^T\mathbf{P}-\mathbf{t}_3^T\mathbf{P}v_1=0.$$

Constraint of one correspondence

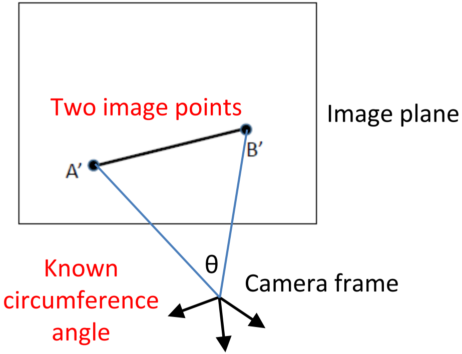
$$\Rightarrow\quad\begin{pmatrix}\mathbf{P}_1^T & 0 & -u_1\mathbf{P}_1^T\\ 0 & \mathbf{P}_1^T & -v_1\mathbf{P}_1^T\\ \vdots & \vdots & \vdots\\ \mathbf{P}_N^T & 0 & -u_N\mathbf{P}_N^T\\ 0 & \mathbf{P}_N^T & -v_N\mathbf{P}_N^T\end{pmatrix}\boxed{\begin{pmatrix}\mathbf{t}_1\\ \mathbf{t}_2\\ \mathbf{t}_3\end{pmatrix}}=0$$

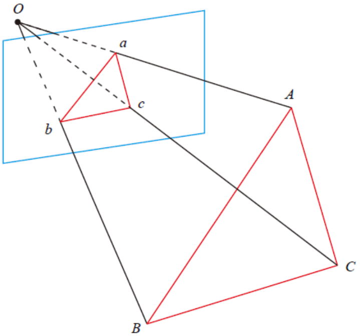
Since **t** has a total dimension of 12, the linear solution of the transformation matrix **T** can be achieved by at least six pairs of matching points.

Classical Algorithms

➢ Perspective-3-Points (P3P)

✓ Configuration of P3P





Configuration of P3P problem

$$\left\{\begin{array}{l}\Delta Oab - \Delta OAB \\ \Delta Obc - \Delta OBC \\ \Delta Oac - \Delta OAC\end{array}\right.$$

Three pairs of triangles

Classical Algorithms

➢ Perspective-3-Points (P3P)

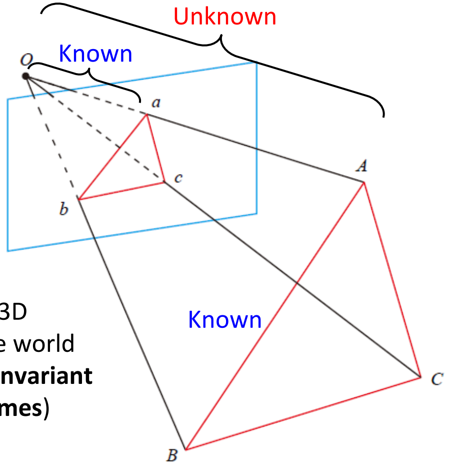
✓ The law of cosines

Known from the normalized image points

$$\overline{OA}^2 + \overline{OB}^2 - 2.\overline{OA}.\overline{OB}.\cos\langle a, b\rangle = \overline{AB}^2$$
$$\overline{OB}^2 + \overline{OC}^2 - 2.\overline{OB}.\overline{OC}.\cos\langle b, c\rangle = \overline{BC}^2$$
$$\overline{OA}^2 + \overline{OC}^2 - 2.\overline{OA}.\overline{OC}.\cos\langle a, c\rangle = \overline{AC}^2$$

Variable in the camera frame w.r.t. the unknown camera pose

Known from 3D coordinates in the world frame (**distance is invariant in different frames**)



Configuration of P3P problem

Classical Algorithms

➢ Perspective-3-Points (P3P)

✓ Rewrite the law of cosines

Unknown

Divide each equation by $\overline{OC}^2$ on both sides, and denote $x = \overline{OA}/\overline{OC},\ y = \overline{OB}/\overline{OC}$

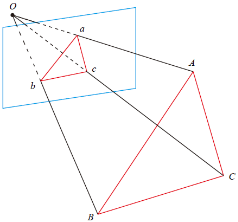
$$\begin{aligned}\overline{OA}^2 + \overline{OB}^2 - 2.\overline{OA}.\overline{OB}\cos\langle a,b\rangle &= \overline{AB}^2\\ \overline{OB}^2 + \overline{OC}^2 - 2.\overline{OB}.\overline{OC}.\cos\langle b,c\rangle &= \overline{BC}^2\\ \overline{OA}^2 + \overline{OC}^2 - 2.\overline{OA}.\overline{OC}.\cos\langle a,c\rangle &= \overline{AC}^2\end{aligned} \quad\Rightarrow\quad \begin{aligned}x^2 + y^2 - 2.x.y\cos\langle a,b\rangle &= \overline{AB}^2/\overline{OC}^2\\ y^2 + 1^2 - 2.y.\cos\langle b,c\rangle &= \overline{BC}^2/\overline{OC}^2\\ x^2 + 1^2 - 2.x.\cos\langle a,c\rangle &= \overline{AC}^2/\overline{OC}^2.\end{aligned}$$

We further denote $v = \overline{AB}^2/\overline{OC}^2$ $u = \overline{BC}^2/\overline{AB}^2, w = \overline{AC}^2/\overline{AB}^2$

Unknown Known variable for derivation

Accordingly, we have $u.v = \overline{BC}^2/\overline{OC}^2,\ w.v = \overline{AC}^2/\overline{OC}^2$

Unknown



Classical Algorithms

- Perspective-3-Points (P3P)

✓ Rewrite the law of cosines

Generate polynomial system w.r.t. unknown v, x, and y

$$
\begin{aligned}
x^2 + y^2 - 2.x.y.\cos\langle a, b\rangle &= \boxed{\overline{AB}^2/\overline{OC}^2} \\
y^2 + 1^2 - 2.y.\cos\langle b, c\rangle &= \boxed{\overline{BC}^2/\overline{OC}^2} \\
x^2 + 1^2 - 2.x.\cos\langle a, c\rangle &= \boxed{\overline{AC}^2/\overline{OC}^2}.
\end{aligned}
\quad \Rightarrow \quad
\begin{aligned}
x^2 + y^2 - 2.x.y.\cos\langle a, b\rangle - v &= 0 \\
y^2 + 1^2 - 2.y.\cos\langle b, c\rangle - u.v &= 0 \\
x^2 + 1^2 - 2.x.\cos\langle a, c\rangle - w.v &= 0.
\end{aligned}
$$

Unknown known Unknown

known

We substitute the first equation to the second and third **to *eliminate v***, obtaining

$$
\begin{aligned}
(1-u)\,y^2 - ux^2 - 2\cos\langle b, c\rangle\, y + 2uxy\cos\langle a, b\rangle + 1 &= 0 \\
(1-w)\,x^2 - wy^2 - 2\cos\langle a, c\rangle\, x + 2wxy\cos\langle a, b\rangle + 1 &= 0.
\end{aligned}
$$

Classical Algorithms

➢ Perspective-3-Points (P3P)

$$x = \overline{OA}/\overline{OC}, \; y = \overline{OB}/\overline{OC}$$

$$\boxed{x^2 + y^2 - 2.x.y}\boxed{\cos\langle a, b\rangle} = \boxed{\boxed{\overline{AB}^2}/\overline{OC}^2}$$

✓ Solving the multivariate quadratic polynomial w.r.t. unknown x and y

$$\begin{aligned}(1-\boxed{u})\,y^2 - ux^2 - 2\cos\langle b, c\rangle\,y + 2uxy\cos\langle a, b\rangle + 1 = 0\\(1-\boxed{w})\,x^2 - wy^2 - 2\cos\langle a, c\rangle\,x + 2wxy\cos\langle a, b\rangle + 1 = 0.\end{aligned}$$

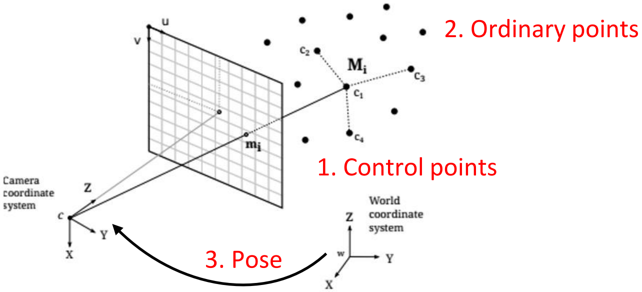
- Use Wu's elimination method to obtain x and y.
- Use x and y to compute OC
- Use x, y and OC to obtain OA and OB
- We can thus obtain the coordinates of A, B, C in the camera frame
- Based on the 3D−3D point pair, the closed-form solution of camera movement R, t can be calculated. (introduced tommorw)

Advanced Algorithms

➢ Efficient Perspective-n-Points (EPnP)

Express each 3D point by a linear combination of **four control points**.

- We first determine the coordinates of these **four control points** in both **camera** and **world** frames.
- Then we use control points to obtain coordinates of **each 3D point** in both camera and world frames (3D-3D point correspondences).
- Finally, we use 3D-3D point correspondences to compute closed-form solution of rotation and translation.

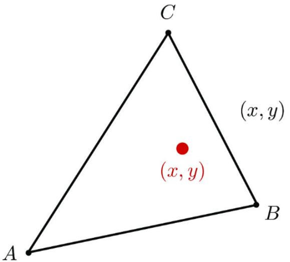


Advanced Algorithms

➢ Efficient Perspective-n-Points (EPnP)

Express each 3D point by a linear combination of **four** control points in the **world** frame.

A coordinate system for triangles (α, β, γ)



$$(x, y) = \alpha A + \beta B + \gamma C$$
$$\alpha + \beta + \gamma = 1$$

Barycentric coordinates in 2D

Non-homogeneous

$$\mathbf{p}_i^w = \sum_{j=1}^{4} \alpha_{ij} \mathbf{c}_j^w$$
$$\sum_{j=1}^{4} \alpha_{ij} = 1$$

Coefficients are called homogeneous barycentric coordinates

Advanced Algorithms

- Efficient Perspective-n-Points (EPnP)

$$\mathbf{p}_i^w = \sum_{j=1}^4 \alpha_{ij}\mathbf{c}_j^w$$
$$\sum_{j=1}^4 \alpha_{ij} = 1$$

$$\mathbf{c}_j^c = [\mathbf{R}|\mathbf{t}]\begin{bmatrix}\mathbf{c}_j^w \\ 1\end{bmatrix}$$

We denote the pose from the world frame to the camera frame by $[\mathbf{R}|\mathbf{t}]$

An **ordinary 3D point** in the camera frame can be expressed by

$$\mathbf{p}_i^c = [\mathbf{R}|\mathbf{t}]\begin{bmatrix}\boxed{\mathbf{p}_i^w} \\ 1\end{bmatrix} = [\mathbf{R}|\mathbf{t}]\begin{bmatrix}\sum_{j=1}^4 \boxed{\alpha_{ij}}\mathbf{c}_j^w \\ \sum_{j=1}^4 \boxed{\alpha_{ij}}\end{bmatrix} = \sum_{j=1}^4 \boxed{\alpha_{ij}}\boxed{\mathbf{c}_j^c}$$

Euclidian transformation

Control point-based expression in the world frame

Ordinary points can be expressed by **unknown** control points in the camera frame

Advanced Algorithms

- Efficient Perspective-n-Points (EPnP)

✓ Linear constraint to solve **control points in the camera**

Perspective projection of **ordinary points in the camera frame**

Ordinary point

$$\omega_i \underbrace{\begin{bmatrix} \mathbf{u}_i \\ 1 \end{bmatrix}}_{\text{Known}} = \mathbf{K} \underbrace{\boxed{\mathbf{p}_i^c}}_{\text{Unknown}} = \mathbf{K} \sum_{j=1}^{4} \alpha_{ij} \underbrace{\boxed{\mathbf{c}_j^c}}_{\text{Unknown}} = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Known}} \sum_{j=1}^{4} \underbrace{\alpha_{ij}}_{\text{Known}} \underbrace{\boxed{\begin{bmatrix} x_j^c \\ y_j^c \\ z_j^c \end{bmatrix}}}_{\text{Unknown}}$$

Constraint w.r.t. unknown control points in the camera frame

$$\begin{cases} \sum_{j=1}^{4} \left(\alpha_{ij} f_x \boxed{x_j^c} + \alpha_{ij} \left(c_x - u_i \right) \boxed{z_j^c} \right) = 0 \\ \sum_{j=1}^{4} \left(\alpha_{ij} f_y \boxed{y_j^c} + \alpha_{ij} \left(c_y - v_i \right) \boxed{z_j^c} \right) = 0 \end{cases}$$

Advanced Algorithms

- Efficient Perspective-n-Points (EPnP)

Each 3D-2D correspondence can provide two linear constraints. We use at least 6 correspondences to generate 12 equations, solving 12 unknown parameters of control points **in the camera frame**.

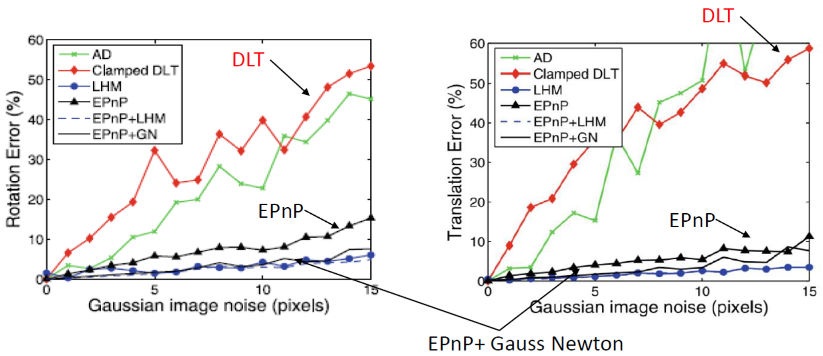
$$\begin{cases} \sum_{j=1}^{4} \left(\alpha_{ij} f_x x_j^c + \alpha_{ij} \left(c_x - u_i \right) z_j^c \right) = 0 \\ \sum_{j=1}^{4} \left(\alpha_{ij} f_y y_j^c + \alpha_{ij} \left(c_y - v_i \right) z_j^c \right) = 0 \end{cases} \Rightarrow \overset{\overset{2n \times 12}{\downarrow}}{\mathbf{M}} \underset{\underset{\begin{bmatrix} x_j^c \\ y_j^c \\ z_j^c \end{bmatrix} \mathbf{c}_j, \quad j = 1, \cdots, 4}{\uparrow}}{\mathbf{x}} = 0$$

Advanced Algorithms

- Comparison between EPnP and DLT

✓ Accuracy test w.r.t. noise

EPnP is up to **10 times** more robust to noise than DLT.

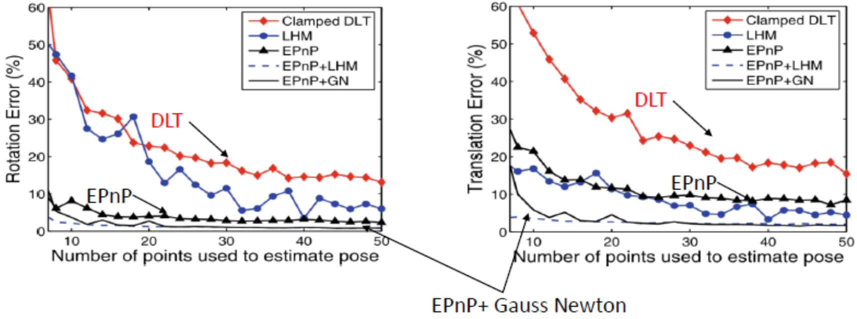


Advanced Algorithms

- Comparison between EPnP and DLT

✓ Accuracy test w.r.t. number of correspondences

EPnP is up to **10 times** more accurate than DLT

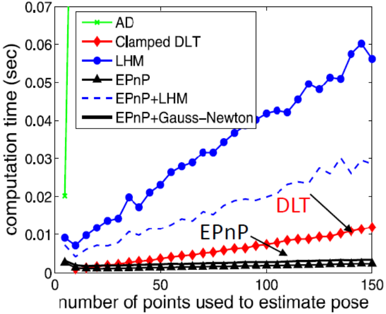


Advanced Algorithms

- ➢ Comparison between EPnP and DLT

✓ Efficiency test w.r.t. number of correspondences

EPnP is up to **10 times** more efficient than DLT



Advanced Algorithms

- Iterative Method

In addition to the linear method, we can also formulate the PnP problem as a nonlinear least-square problem about re-projection errors.

$$s_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \mathbf{KT} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} \Rightarrow \quad \mathbf{T}^* = \arg\min_{\mathbf{T}} \frac{1}{2} \sum_{i=1}^{n} \left\| \mathbf{u}_i - \frac{1}{s_i} \mathbf{K} \boxed{\mathbf{T}} \mathbf{P}_i \right\|_2^2 .$$

Known intrinsic matrix

$$s_i \mathbf{u}_i = \mathbf{KTP}_i$$

Unknown extrinsic matrix

Initial value provided by DLT/EPnP

Brief Introduction to Perspective-n-Lines (PnL)

- Definition of PnL

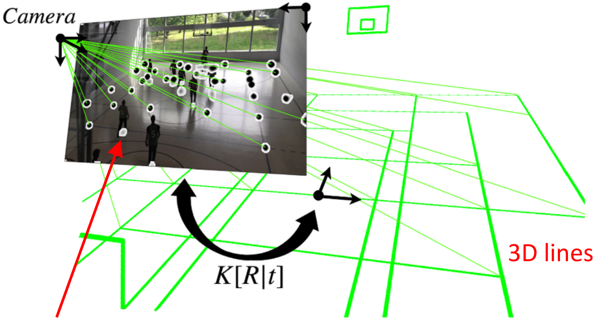
✓ Input

A set of 3D-2D line correspondences

3D lines is in the world frame

✓ Output

3-DOF rotation and 3-DOF translation aligning the world frame to the camera frame.



Details of PnL will not be asked in the exam.

Brief Introduction to Perspective-n-Lines (PnL)

➢ Definition of PnL

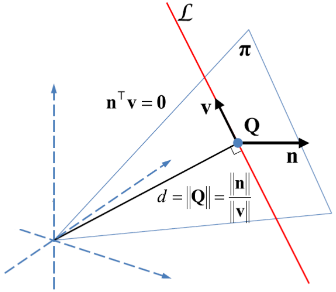
✓ Basic geometric constraints

The **transformation from the world frame to the camera frame** for the Plücker line coordinates (page 22/57 of Chapter 02 Part 1)

n is with respect to both rotation and translation.

$$\begin{bmatrix} \boxed{\mathbf{n}_j} \\ \mathbf{v}_j \end{bmatrix} = \underbrace{\begin{bmatrix} \mathrm{R}_{ji} & [\mathbf{t}_{ji}]_\times \mathrm{R}_{ji} \\ \mathbf{0} & \mathrm{R}_{ji} \end{bmatrix}}_{\text{Unknown}} \begin{bmatrix} \mathbf{n}_i \\ \mathbf{v}_i \end{bmatrix} \quad \text{Known}$$

v is with respect to only rotation.



Brief Introduction to Perspective-n-Lines (PnL)

➢ Definition of PnL

✓ Basic geometric constraints

Perspective projection of Plücker line coordinates (page 32/51 of Chapter 03 Part 1) in the camera frame

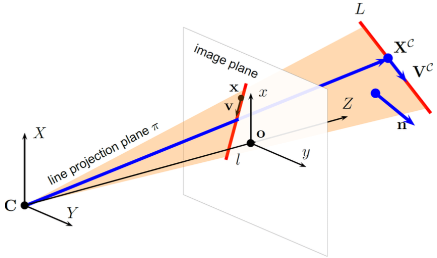
n is with respect to both rotation and translation.

$$\mathbf{l} = \mathcal{K}\boxed{\mathbf{n}}$$

Known image line (3D vector)

$$\mathcal{K} = \begin{bmatrix} f_y & 0 & 0 \\ 0 & f_x & 0 \\ -f_y x_0 & -f_x y_0 & f_x f_y \end{bmatrix}$$

Known Intrinsic matrix for line projection



Brief Introduction to Perspective-n-Lines (PnL)

➢ Methods to Solve PnL

✓ Direct Linear Transform (one-step method)

We can jointly estimate the rotation and translation.

$$\begin{bmatrix} \boxed{\mathbf{n}_j} \\ \mathbf{v}_j \end{bmatrix} = \begin{bmatrix} \mathrm{R}_{ji} & [\mathbf{t}_{ji}]_\times \mathrm{R}_{ji} \\ \mathbf{0} & \mathrm{R}_{ji} \end{bmatrix} \begin{bmatrix} \boxed{\mathbf{n}_i} \\ \mathbf{v}_i \end{bmatrix} \qquad \boxed{\mathbf{l}} = \boxed{\mathcal{K}}\boxed{\mathbf{n}}$$

Each line correspondence can provide two linear constraints. We need at least 9 correspondences to estimate 18 elements.

w.r.t. rotation and translation (3*6=18 elements)

Known image line

$$\Rightarrow \quad \mathbf{Mp} = \mathbf{0}$$

Known coefficient matrix

3*6=18 elements of the above matrix

Bronislav Priby et al., "Camera Pose Estimation from Lines using Plücker Coordinates", in BMVC, 2015

Summary

- Overview of 3D-2D Geometry
- Definition of Perspective-n-Points (PnP)
- Classical Algorithms
- Advanced Algorithms
- Brief Introduction to Perspective-n-Lines (PnL)

Thank you for your listening!

If you have any questions, please come to me :-)