Lecture 09 – Pose Estimation II

Nov 13nd, 2018

Danping Zou,

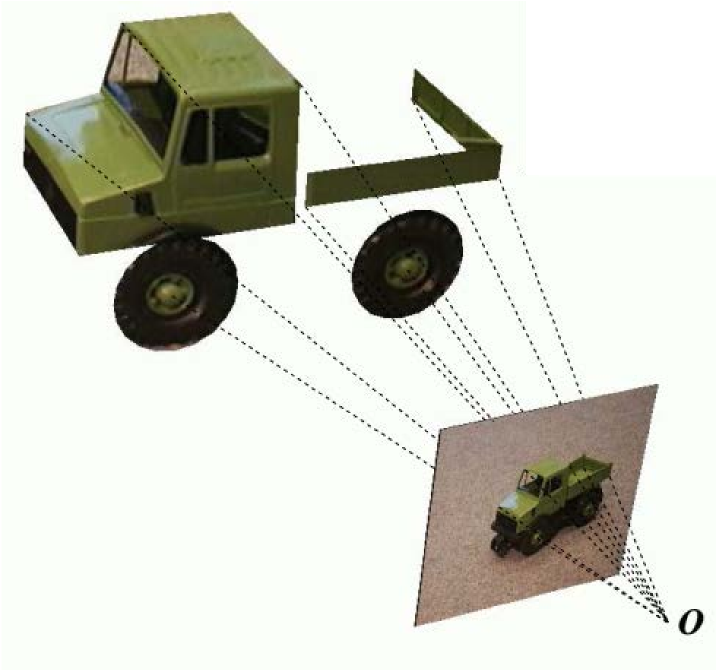
Associate Professor

Institute for Sensing and Navigation

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

3D-2D registration

- We first consider when the 3D object is a real 3D object (not a planar or a linear object)



$$\boldsymbol{X}_i \leftrightarrow \boldsymbol{x}_i \quad \Rightarrow \quad \mathbf{R}, \boldsymbol{t}$$

Camera pose estimation

- **Algebraic approach:**
  - P3P (Fischer & Bolles. 1981)
  - EPnP (Lepetit, et al. 2008)
- Positive aspects:
  - Fast
  - No initial guess
- Negative aspects:
  - Prone to noises
  - Numeric instabilities

Camera pose estimation

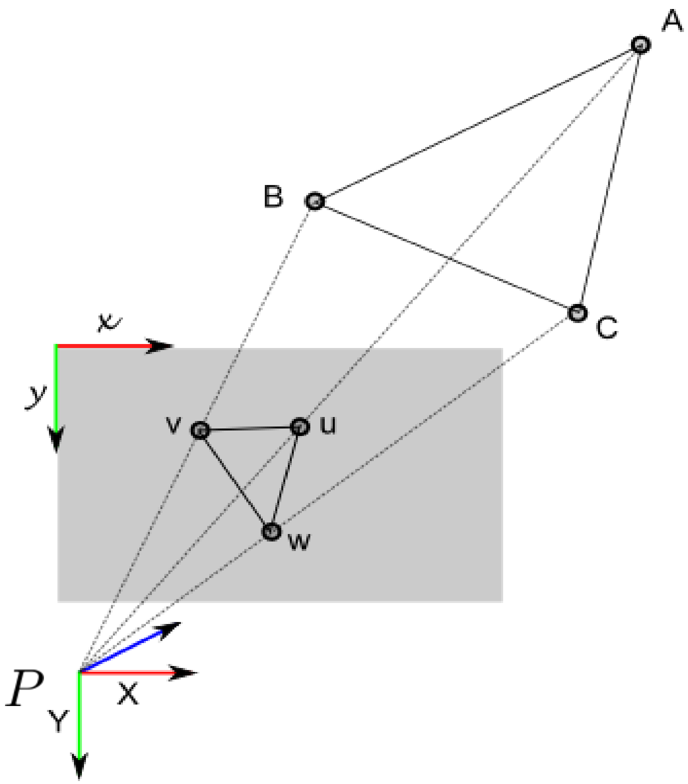
- **Iterative approach**
  - POSIT algorithm
  - Solve the nonlinear least squares problem

$$\min_{\mathbf{R},\boldsymbol{t}} \sum_{i=1}^{n} \|\boldsymbol{x}_i - \mathcal{P}(\boldsymbol{X}_i, \mathbf{R}, \boldsymbol{t})\|^2$$

  - Solved by Levenberg-Marquardt algorithm
- Positive aspects:
  - Numerically stable
- Negative aspects:
  - Need initial guess
  - Sometimes diverge

Camera pose estimation

- **P3P** algorithm
  - Fischer and Bolles : “**P**erspective **t**hree-**p**oint problem”



1. 3D-2D correspondences:

$$A \leftrightarrow u, B \leftrightarrow v, C \leftrightarrow w$$

2. Depths of A,B,C are solved (Law of cosines):

$$PA, PB, PC$$

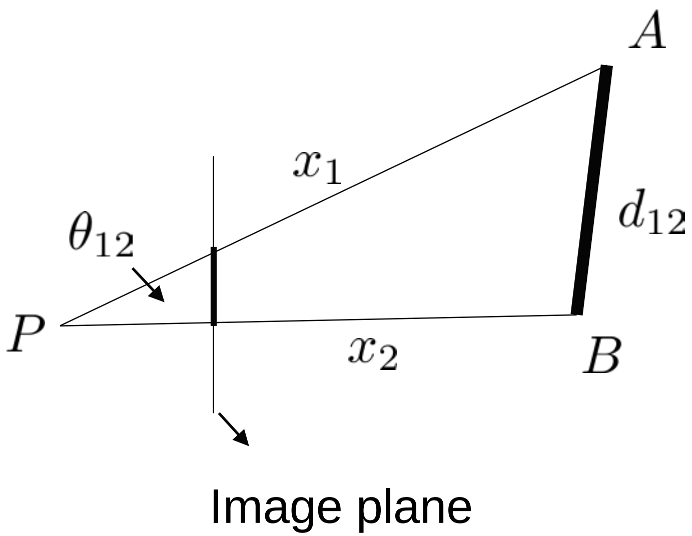
3. Distances converted into pose configurations

$$\mathbf{R}, \mathbf{t}$$

上海交通大学 SHANGHAI JIAO TONG UNIVERSITY

P3P

- Solving the depth - Law of cosines



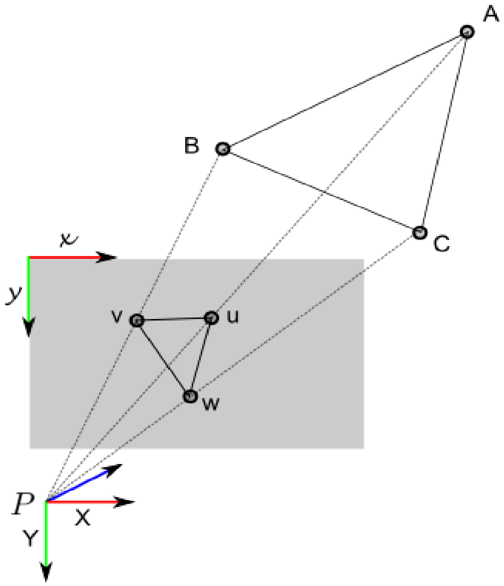
$$PA^2 + PB^2 - 2PA.PB.\cos(\theta) = AB^2$$

$$f_{1,2}(x_1, x_2) = x_1^2 + x_2^2 - 2x_1x_2\boxed{\cos\theta_{12}} - \boxed{d_{12}^2} = 0$$

Constant.

P3P

- We have three equations by considering three edges together



$$f_{1,2}(x_1, x_2) = x_1^2 + x_2^2 - 2x_1x_2\cos\theta_{12} - d_{12}^2 = 0$$
$$f_{1,3}(x_1, x_3) = x_1^2 + x_3^2 - 2x_1x_3\cos\theta_{13} - d_{13}^2 = 0$$
$$f_{2,3}(x_2, x_3) = x_2^2 + x_3^2 - 2x_2x_3\cos\theta_{23} - d_{23}^2 = 0$$

P3P

- We can solve those equations by elimination :

$$\begin{cases} f_{1,2}(x_1,x_2) = x_1^2 + x_2^2 - 2x_1x_2\cos\theta_{12} - d_{12}^2 = 0 \\ f_{1,3}(x_1,x_3) = x_1^2 + x_3^2 - 2x_1x_3\cos\theta_{13} - d_{13}^2 = 0 \\ f_{2,3}(x_2,x_3) = x_2^2 + x_3^2 - 2x_2x_3\cos\theta_{23} - d_{23}^2 = 0 \end{cases}$$

$$\left\{ \begin{array}{l} f_{1,2}(x_1,x_2) = 0 \\ \left.\begin{array}{l} f_{1,3}(x_1,x_3) = 0 \\ f_{2,3}(x_2,x_3) = 0 \end{array}\right\} \cancel{x_3} \quad h(x_1,x_2) = 0 \end{array} \right\} \cancel{x_2} \quad g(x_1) = 0$$

- But how do we eliminate the common variables for the two polynomial equations?

P3P

- We can solve those equations by elimination :

$$\begin{cases} f_{1,2}(x_1,x_2) = x_1^2 + x_2^2 - 2x_1x_2\cos\theta_{12} - d_{12}^2 = 0 \\ f_{1,3}(x_1,x_3) = x_1^2 + x_3^2 - 2x_1x_3\cos\theta_{13} - d_{13}^2 = 0 \\ f_{2,3}(x_2,x_3) = x_2^2 + x_3^2 - 2x_2x_3\cos\theta_{23} - d_{23}^2 = 0 \end{cases}$$

$$\left.\begin{array}{l} \left\{\begin{array}{l} f_{1,2}(x_1,x_2) = 0 \\ \left.\begin{array}{l} f_{1,3}(x_1,x_3) = 0 \\ f_{2,3}(x_2,x_3) = 0 \end{array}\right\} \xrightarrow{\cancel{x_3}} h(x_1,x_2) = 0 \end{array}\right. \end{array}\right\} \xrightarrow{\cancel{x_2}} g(x_1) = 0$$

- But how do we eliminate the common variables for the two polynomial equations?

P3P

- Consider we want eliminate x_1 from the first two equations

$$x_1^2 + x_2^2 - 2x_1x_2\cos\theta_{12} - d_{12}^2 = 0 \quad (1)$$

$$x_1^2 + x_3^2 - 2x_1x_3\cos\theta_{13} - d_{13}^2 = 0 \quad (2)$$

- A straightforward method is to solve (1) to get the solution of x_1 and then put into (2).
- But usually it is difficult to get the close-form solution of a high-ordered polynomial equation.

Polynomial resultant

- Given two polynomials $f(x)$ and $g(x)$, their resultant is equal to zero if and only if the two polynomials have a root in common.

$$f(x) = a_n x^n + \ldots + a_1 x + a_0$$

$$g(x) = b_m x^m + \ldots + b_1 x + b_0$$

- The resultant of $f(x)$ and $g(x)$ is defined as

$$\mathrm{Res}(f, g, x) = a_n^m b_m^n \Pi_{i,j} (\alpha_i - \beta_j)$$

where $\alpha_1, \ldots, \alpha_n$ and $\beta_1, \ldots, \beta_m$ are the solutions of $f(x) = 0$ $g(x) = 0$

Polynomial resultant

- The resultant can be computed through Sylvester matrix

$$
\mathrm{Syl}(f,g,x) = \begin{bmatrix}
a_n & & & & b_m & & & \\
a_{n-1} & a_n & & & b_{m-1} & b_m & & \\
a_{n-2} & a_{n-1} & \ddots & & b_{m-2} & b_{m-1} & \ddots & \\
\vdots & \vdots & \ddots & a_n & \vdots & \vdots & \ddots & b_m \\
\vdots & \vdots & & a_{n-1} & \vdots & \vdots & & b_{m-1} \\
a_0 & a_1 & & & b_0 & b_1 & & \\
& a_0 & \ddots & \vdots & & b_0 & \ddots & \vdots \\
& & \ddots & a_1 & & & \ddots & b_1 \\
& & & a_0 & & & & b_0
\end{bmatrix}
$$

$$
\mathrm{Res}(f,g,x) = \det(\mathrm{Syl}(f,g,x))
$$

Elimination of two polynomials

- Example I:
  - Consider two polynomials, we can use the resultant to check that if they have common roots.

$$f(x) = x^5 - 3x^4 - 2x^3 + 3x^2 + 7x + 6$$

$$g(x) = x^4 + x^2 + 1$$

$$\mathrm{Res}(f, g, x) = \begin{vmatrix} 1 & -3 & -2 & 3 & 7 & 6 & 0 & 0 & 0 \\ 0 & 1 & -3 & -2 & 3 & 7 & 6 & 0 & 0 \\ 0 & 0 & 1 & -3 & -2 & 3 & 7 & 6 & 0 \\ 0 & 0 & 0 & 1 & -3 & -2 & 3 & 7 & 6 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{vmatrix} = 0$$

Elimination of two polynomials

- Example II:
  - Consider two polynomial equations

$$f = x^2y - 3xy^2 + x^2 - 3xy = 0$$
$$g = x^3y + x^3 - 4y^2 - 3y + 1 = 0$$

  - We want to eliminate the variable x

Elimination of two polynomials

- Reorder the polynomials：

$$f = (y+1)x^2 - 3y(y+1)x$$

$$g = (y+1)x^3 + (y+1)(-4y+1)$$

$$\mathrm{Syl}(f,g,x) = \begin{bmatrix} (y+1) & -(3y^2+3y) & 0 & 0 & 0 \\ 0 & (y+1) & -(3y^2+3y) & 0 & 0 \\ 0 & 0 & (y+1) & -(3y^2+3y) & 0 \\ (y+1) & 0 & 0 & (-4y^2-3y+1) & 0 \\ 0 & (y+1) & 0 & 0 & (-4y^2-3y+1) \end{bmatrix}^T$$

$$\mathrm{Res}(f,g,x) = \det(\mathrm{Syl}(f,g,x)) = -(y+1)^5(4y-1)(27y^3-4y+1)$$

Elimination of two polynomials

- Example II:
  - Consider two polynomial equations

$$f = x^2y - 3xy^2 + x^2 - 3xy = 0$$
$$g = x^3y + x^3 - 4y^2 - 3y + 1 = 0$$

  - After eliminate the variable x , we have

$$(y+1)^5(4y-1)(27y^3-4y+1) = 0$$

P3P

- We can solve those equations by elimination :

$$\begin{cases} f_{1,2}(x_1,x_2) = x_1^2 + x_2^2 - 2x_1x_2\cos\theta_{12} - d_{12}^2 = 0 \\ f_{1,3}(x_1,x_3) = x_1^2 + x_3^2 - 2x_1x_3\cos\theta_{13} - d_{13}^2 = 0 \\ f_{2,3}(x_2,x_3) = x_2^2 + x_3^2 - 2x_2x_3\cos\theta_{23} - d_{23}^2 = 0 \end{cases}$$

$$\left.\begin{array}{l} \begin{cases} f_{1,2}(x_1,x_2) = 0 \\ \left.\begin{array}{l} f_{1,3}(x_1,x_3) = 0 \\ f_{2,3}(x_2,x_3) = 0 \end{array}\right\} \cancel{x_3} \quad h(x_1,x_2) = 0 \end{cases} \end{array}\right\} \cancel{x_2} \quad g(x_1) = 0$$

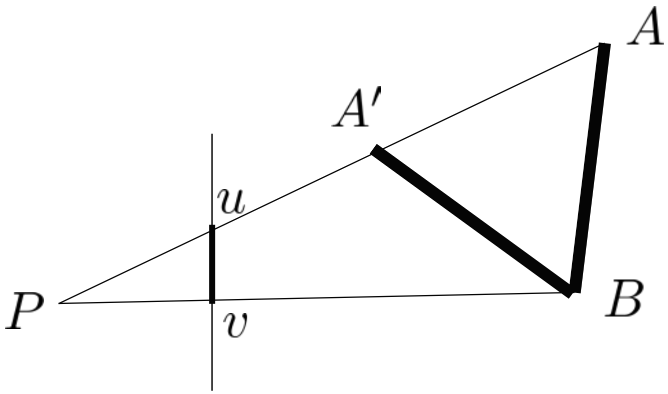
$$g(x) = a_8x_1^8 + a_6x_1^6 + a_4x_1^4 + a_2x_1^2 + a_0$$

Four possible solutions

$$(x_1 > 0)$$

P3P

- Depth ambiguity (multiple solutions)



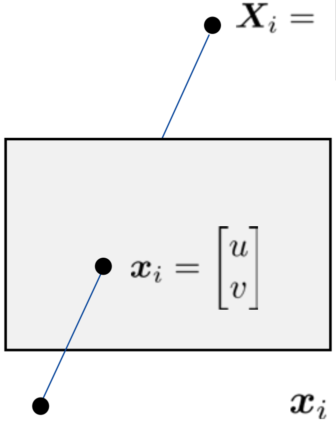
There are four solutions in maximum.

Need more points to eliminate the ambiguity.

PnP (Quan & Lan. 1999)

P3P

- After we estimate the depth value, we can compute the 3D coordinates of this point in the camera frame.
- Given their 3D coordinates in the world frame, we can solve the camera pose by 3D rigid registration (close-form solution)



$$\boldsymbol{X}_i = \begin{bmatrix} \lambda_i u \\ \lambda_i v \\ \lambda_i \end{bmatrix} \leftrightarrow \mathbf{Y}_i$$

$$\boldsymbol{x}_i = \begin{bmatrix} u \\ v \end{bmatrix}$$

$$\boldsymbol{x}_i = \mathbf{K}^{-1}\boldsymbol{m}_i$$

$$\begin{aligned} \boldsymbol{X}_1 &= \mathbf{R}\boldsymbol{Y}_1 + \boldsymbol{t} \\ \boldsymbol{X}_2 &= \mathbf{R}\boldsymbol{Y}_2 + \boldsymbol{t} \\ \boldsymbol{X}_3 &= \mathbf{R}\boldsymbol{Y}_3 + \boldsymbol{t} \end{aligned} \Rightarrow \mathbf{R}, \boldsymbol{t}$$

Other PnP algorithms

- PnP algorithm (> 3 point correspondences)

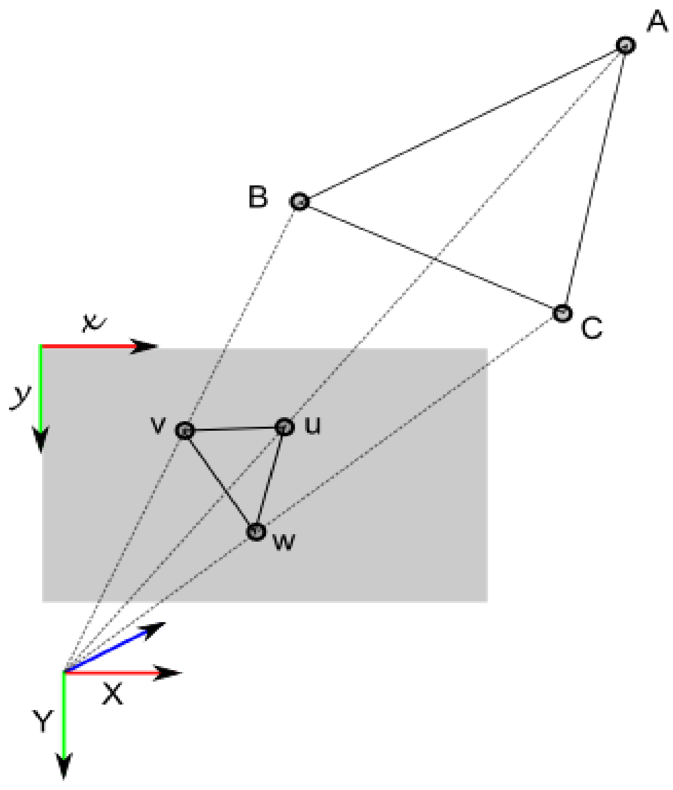
  Quan, Long, and Zhongdan Lan. "Linear n-point camera pose determination." *IEEE Transactions on pattern analysis and machine intelligence* 21.8 (1999): 774-780.

- ePnP algorithm (using virtual control points)

  Lepetit, Vincent, Francesc Moreno-Noguer, and Pascal Fua. "Epnp: An accurate o (n) solution to the pnp problem." *International journal of computer vision* 81.2 (2009): 155.

Summary

- P3P firstly estimate the depth of each point by using law of cosines for three edges.
- It involves solving a system of polynomial equations.
- Resultant can be used to eliminate the common variables between two polynomial equations.
- The depth values can be used to derive the 3D coordinates of those points in the camera frame.
- After that the camera pose can be computed from the corresponding coordinates represented in the camera frame and the world frame.



Outline

- ~~**About pose estimation**~~
- ~~**3D-to-3D registration**~~
  - ~~Rotation only~~
  - ~~Rotation plus Translation~~
  - ~~Unknown correspondences – Iterative Closest Point (ICP)~~
- ~~**3D-to-2D registration (Camera pose estimation)**~~
  - ~~3D objects~~
    - ~~Close-form algorithm - P3P~~
    - Iterative algorithm - POSIT
    - Iterative algorithm - Nonlinear least squares
  - Planar objects
    - Known patterns – Checkboard box, QR pattern
    - Planar Pictures

Camera pose estimation

- **Iterative approach**
  - POSIT algorithm
  - Solve the nonlinear least squares problem

$$\min_{\mathbf{R},\boldsymbol{t}} \sum_{i=1}^{n} \|\boldsymbol{x}_i - \mathcal{P}(\boldsymbol{X}_i, \mathbf{R}, \boldsymbol{t})\|^2$$

- Positive aspects:
  - Numerically stable
- Negative aspects:
  - Need initial guess
  - Sometimes diverge

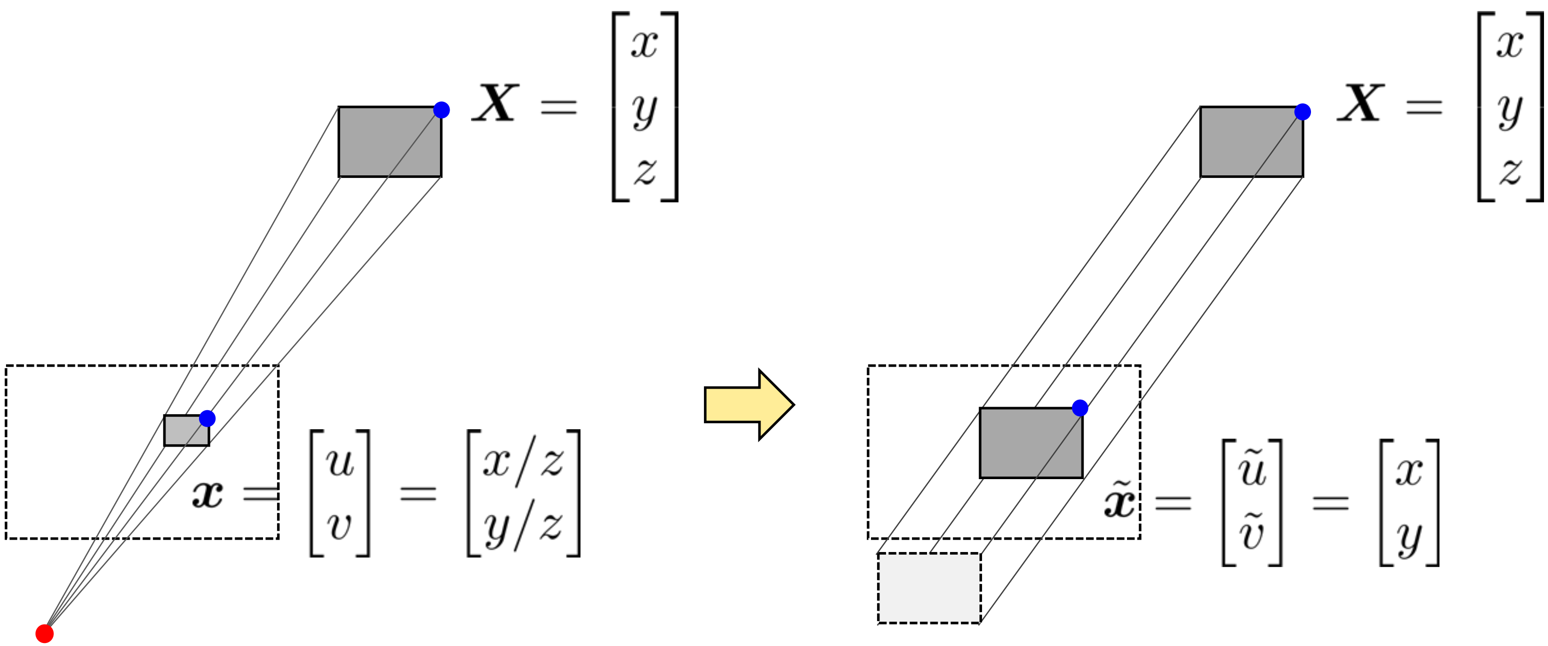
Camera pose estimation

- POSIT algorithm (**P**ose from **O**rthography and **S**caling with **It**erations)
  - **P**ose from Orthography and Scaling (**POS**)
    - Absolute orientation – (Estimate rotation only)
  - Iterations
    - Repeatedly compute the rotation until converge
  - Compute the translation

Dementhon, Daniel F., and Larry S. Davis. "Model-based object pose in 25 lines of code." *International journal of computer vision* 15.1-2 (1995): 123-141.

Orthographic projection

- Perspective projection vs orthographic projection



$$\boldsymbol{X} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad \boldsymbol{x} = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} x/z \\ y/z \end{bmatrix}$$

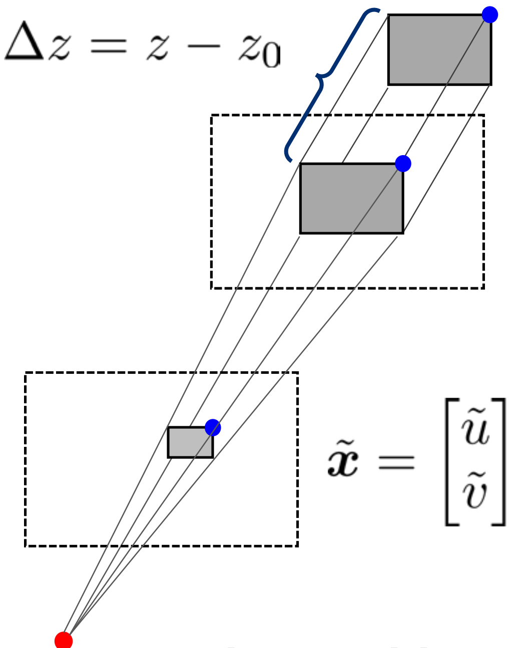
$$\boldsymbol{X} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad \tilde{\boldsymbol{x}} = \begin{bmatrix} \tilde{u} \\ \tilde{v} \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

Perspective projection **Orthographic projection**

POSIT

- Scaled Orthographic Projection

$$\Delta z = z - z_0$$



$$\boldsymbol{X} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\boldsymbol{X} = \begin{bmatrix} x \\ y \\ z_0 \end{bmatrix}$$

$$\tilde{\boldsymbol{x}} = \begin{bmatrix} \tilde{u} \\ \tilde{v} \end{bmatrix} = \begin{bmatrix} x/z_0 \\ y/z_0 \end{bmatrix}$$

Orthographic projection

$$\boldsymbol{x} = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} x/z \\ y/z \end{bmatrix}$$

$$\Updownarrow$$

$$\tilde{\boldsymbol{x}} = \begin{bmatrix} \tilde{u} \\ \tilde{v} \end{bmatrix} = \begin{bmatrix} x/z_0 \\ y/z_0 \end{bmatrix}$$

POSIT

- Consider the 3D point in the world frame is $\boldsymbol{Y}$,we have

$$\boldsymbol{X} = \mathbf{R}\boldsymbol{Y} + \boldsymbol{t}$$

- If subtract some reference point $\boldsymbol{X}_0$, the normalized 3D coordinates satisfy:

$$\boldsymbol{X}' = \mathbf{R}\boldsymbol{Y}'$$

- After scaled orthographic projection at some scale z_0 ,

$$\tilde{\boldsymbol{x}}' = \begin{bmatrix} \tilde{u}' \\ \tilde{v}' \end{bmatrix} = \begin{bmatrix} x'/z_0 \\ y'/z_0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{r}_1^{\mathrm{T}}\boldsymbol{Y}'/z_0 \\ \boldsymbol{r}_2^{\mathrm{T}}\boldsymbol{Y}'/z_0 \end{bmatrix} \qquad (\mathbf{R} = \begin{bmatrix} \boldsymbol{r}_1^{\mathrm{T}} \\ \boldsymbol{r}_2^{\mathrm{T}} \\ \boldsymbol{r}_3^{\mathrm{T}} \end{bmatrix})$$

POSIT

- We can write a linear system of equations of $\boldsymbol{r}_1, \boldsymbol{r}_2$ and z_0 .

$$\tilde{\boldsymbol{x}}' = \begin{bmatrix} \tilde{u}' \\ \tilde{v}' \end{bmatrix} = \begin{bmatrix} x'/z_0 \\ y'/z_0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{r}_1^{\mathrm{T}} \boldsymbol{Y}'/z_0 \\ \boldsymbol{r}_2^{\mathrm{T}} \boldsymbol{Y}'/z_0 \end{bmatrix} \Rightarrow \begin{cases} \tilde{u}' z_0 = \boldsymbol{r}_1^{\mathrm{T}} \boldsymbol{Y}' \\ \tilde{v}' z_0 = \boldsymbol{r}_2^{\mathrm{T}} \boldsymbol{Y}' \end{cases}$$

$$\Rightarrow \begin{bmatrix} \boldsymbol{Y}'^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}} & -\tilde{u}' \\ \mathbf{0}^{\mathrm{T}} & \boldsymbol{Y}'^{\mathrm{T}} & -\tilde{v}' \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \\ z_0 \end{bmatrix} = \mathbf{0}$$

- If we have more then four corresponding points,

$$\tilde{\boldsymbol{x}}'_i \leftrightarrow \boldsymbol{Y}'_i$$

we can solve $\boldsymbol{r}_1, \boldsymbol{r}_2, z_0$. Finally we can get $\mathbf{R}, \mathbf{t}$.

POSIT

- The key problem is that we do not really have the scaled orthogonal projections. $\cancel{\tilde{x}_i'}$
- Instead, we have only the perspective projections. $\boldsymbol{x}_i$
- Can we convert the perspective projections into the scaled orthogonal projections ?

POSIT

- Here we check two types of projections

$$\boldsymbol{x} = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} x/z \\ y/z \end{bmatrix} \quad \Leftrightarrow \quad \tilde{\boldsymbol{x}} = \begin{bmatrix} \tilde{u} \\ \tilde{v} \end{bmatrix} = \begin{bmatrix} x/z_0 \\ y/z_0 \end{bmatrix}$$

- The error is caused by $\Delta z = z - z_0$.

$$\boldsymbol{x}' = \begin{bmatrix} u' \\ v' \end{bmatrix} = \begin{bmatrix} x/(z_0 + \Delta z) \\ y/(z_0 + \Delta z) \end{bmatrix} = \begin{bmatrix} \boldsymbol{r}_1^{\mathrm{T}} \boldsymbol{Y}'/(z_0 + \Delta z) \\ \boldsymbol{r}_2^{\mathrm{T}} \boldsymbol{Y}'/(z_0 + \Delta z) \end{bmatrix}$$

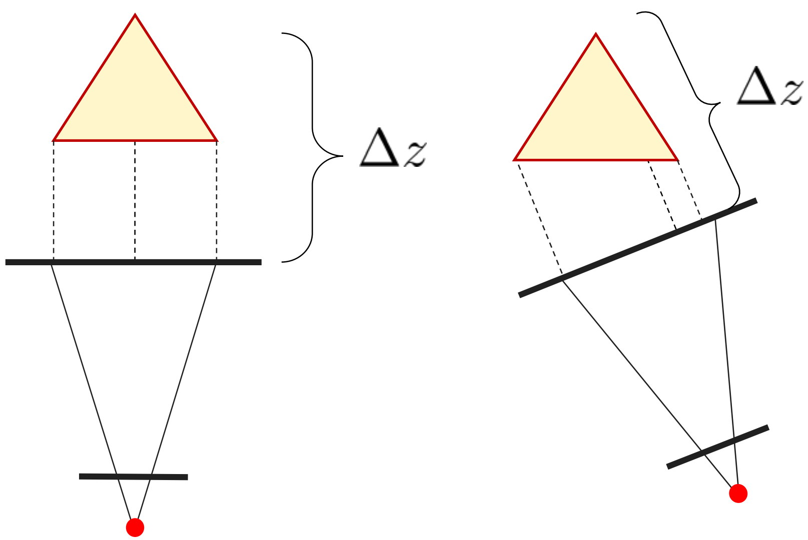
$$\Rightarrow \quad \begin{cases} u'(z_0 + \Delta z) = \boldsymbol{r}_1^{\mathrm{T}} \boldsymbol{Y}' \\ v'(z_0 + \Delta z) = \boldsymbol{r}_2^{\mathrm{T}} \boldsymbol{Y}' \end{cases}$$

POSIT

- If Δz is known, finally we have

$$\begin{bmatrix} \boldsymbol{Y}'^{\mathrm{T}} & \boldsymbol{0}^{\mathrm{T}} & -\tilde{u}' \\ \boldsymbol{0}^{\mathrm{T}} & \boldsymbol{Y}'^{\mathrm{T}} & -\tilde{v}' \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \\ z_0 \end{bmatrix} = \begin{bmatrix} u'\Delta z \\ v'\Delta z \end{bmatrix}$$

- and we can solve $\boldsymbol{r}_1, \boldsymbol{r}_2, z_0$.
- But Δz can be only decided if the orientation is known.



POSIT

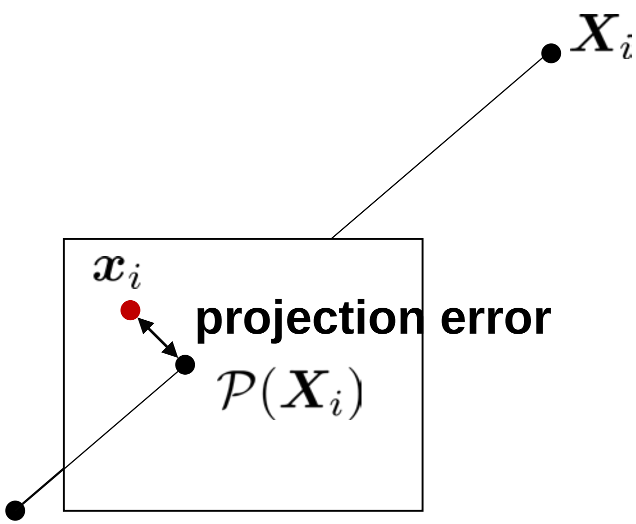
- Use a rough estimation of Δz and solve the rotation $(\boldsymbol{r}_1, \boldsymbol{r}_2)$ and the scale z_0 iteratively.
- As rotation and scale are improved, the estimation of Δz is also improved.
- Finally it converges and we can get the rotation.
- After the rotation is computed, the translation is computed as

$$\boldsymbol{t} = \boldsymbol{Y}_0 - \mathbf{R}\boldsymbol{X}_0$$

Iterative optimization

- The pose can be solved by minimizing the **projection error**

$$\min_{\mathbf{R},\boldsymbol{t}} \sum_{i=1}^{n} \|\underbrace{\boxed{\boldsymbol{x}_i}}_{\text{Image point}} - \underbrace{\mathcal{P}(\boldsymbol{X}_i,\mathbf{R},\boldsymbol{t})}_{\text{Projected point}}\|^2$$



Iterative optimization

- Gauss-Newton or Levenberg-Marquardt algorithm can be applied.

$$\Theta \leftarrow \Theta \boxplus \Delta\Theta$$

$$\begin{pmatrix} \mathbf{R} \\ \boldsymbol{t} \end{pmatrix} \leftarrow \begin{pmatrix} \mathbf{R} \exp(\Delta\theta^\wedge) \\ \boldsymbol{t} + \Delta\boldsymbol{t} \end{pmatrix}$$

- All we need is to solve the incremental step $\Delta\Theta = \begin{pmatrix} \Delta\theta \\ \Delta\boldsymbol{t} \end{pmatrix}$

Iterative optimization

- We rewrite the objective function as the following

$$\|\boldsymbol{x}_i - \mathcal{P}(\boldsymbol{X}_i, \Theta)\|^2$$

- By first-order approximation, we have

$$\approx \|\boldsymbol{x}_i - \mathcal{P}(\boldsymbol{X}_i, \Theta) - \mathbf{J}\Delta\Theta\|^2$$

$$= \|\boldsymbol{r}_i - \mathbf{J}\Delta\Theta\|^2$$

$$\Rightarrow \quad \Delta\Theta = (\mathbf{J}^{\mathrm{T}}\mathbf{J})^{-1}\mathbf{J}\boldsymbol{r}_i$$

- The Jacobian matrix is defined as $\mathbf{J} = \left[\frac{\partial \mathcal{P}}{\partial \Delta\theta} \quad \frac{\partial \mathcal{P}}{\partial \Delta \boldsymbol{t}}\right]$

Iterative optimization

- Compute the Jacobian matrix $\mathbf{J} = \frac{\partial \mathcal{P}}{\partial \Delta\theta \partial \Delta \boldsymbol{t}}$

$$\boldsymbol{x} \sim \mathbf{P}\boldsymbol{X} = \mathbf{K}[\mathbf{R}\ \boldsymbol{t}]\boldsymbol{X}$$

$$\boldsymbol{X}_c = \mathbf{R}\boldsymbol{X} + \mathbf{t} \qquad \frac{\partial \boldsymbol{X}_c}{\partial \Delta\theta \partial \Delta \boldsymbol{t}} = \left[\frac{\partial \boldsymbol{X}_c}{\partial \Delta\theta} =? \quad \mathbf{I}_{3\times 3}\right]$$

$$\begin{cases} u = x_c/z_c \\ v = y_c/z_c \end{cases} \qquad \frac{\partial u \partial v}{\partial \boldsymbol{X}_c} = \begin{bmatrix} 1/z_c & 0 & -x_c/z_c^2 \\ 0 & 1/z_c & -y_c/z_c^2 \end{bmatrix}$$

$$\begin{cases} x = f_x u + c_x \\ y = f_y v + c_y \end{cases} \qquad \frac{\partial \mathcal{P}}{\partial u \partial v} = [f_x\ f_y]$$

$$\boxed{\mathbf{J} = \frac{\partial \mathcal{P}}{\partial \Delta\theta \partial \Delta \boldsymbol{t}} = \frac{\partial \mathcal{P}}{\partial u \partial v} \frac{\partial u \partial v}{\partial \boldsymbol{X}_c} \frac{\partial \boldsymbol{X}_c}{\partial \Delta\theta \partial \Delta \boldsymbol{t}}}$$

Iterative optimization

- How about $\frac{\partial \boldsymbol{X}_c}{\partial \Delta\theta}$?

$$\boldsymbol{X}_c = \mathbf{R}\boldsymbol{X} + \mathbf{t} \qquad \frac{\partial \boldsymbol{X}_c}{\partial \Delta\theta \partial \Delta \boldsymbol{t}} = \left[\frac{\partial \boldsymbol{X}_c}{\partial \Delta\theta} =? \quad \mathbf{I}_{3\times 3}\right]$$

- First-order approximation

$$\frac{\partial \boldsymbol{X}_c}{\partial \Delta\theta} = \frac{\partial (\mathbf{R}\boldsymbol{X})}{\partial \Delta\theta}$$

$$\lim_{\Delta\theta \to 0} \frac{\mathbf{R}\exp(\Delta\theta^{\wedge})\boldsymbol{X} - \mathbf{R}\boldsymbol{X}}{\Delta\theta}$$

$$\approx \lim_{\Delta\theta \to 0} \frac{\mathbf{R}(\mathbf{I} + [\Delta\theta]_{\times})\boldsymbol{X} - \mathbf{R}\boldsymbol{X}}{\Delta\theta}$$

$$= \frac{\mathbf{R}[\Delta\theta]_{\times}\boldsymbol{X}}{\Delta\theta} = -[\mathbf{R}\boldsymbol{X}]_{\times} \quad (a \times b = -b \times a)$$

$$\Rightarrow \quad \frac{\partial \boldsymbol{X}_c}{\partial \Delta\theta} = -\boxed{[\mathbf{R}\boldsymbol{X}]_{\times}}$$

Iterative optimization

- We can also use numeric or automatic differentiation.
- Useful toolkits (c/c++) for solving nonlinear-least squares problem with numeric or automatic differentiation:
  - g2o
  - Ceres-solver

Summary

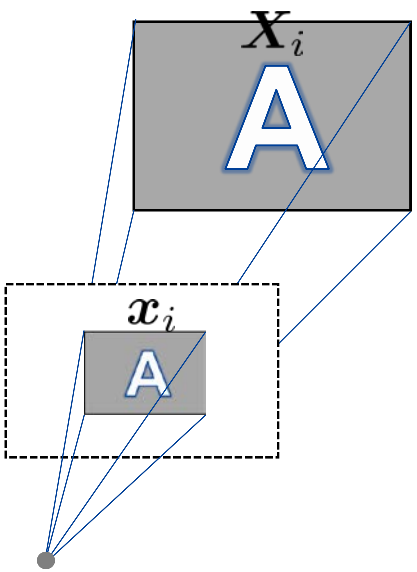
- POSIT
  - Scaled orthographic projection -> linear equation about rotation
  - Estimate Δz and rotation iteratively
- Optimization of projection error
  - Nonlinear least squares problem
  - Use chain rule / numeric differentiation to compute the Jacobian matrix

Outline

- ~~**About pose estimation**~~
- ~~**3D-to-3D registration**~~
  - ~~Rotation only~~
  - ~~Rotation plus Translation~~
  - ~~Unknown correspondences – Iterative Closest Point (ICP)~~
- ~~**3D-to-2D registration (Camera pose estimation)**~~
  - ~~3D objects~~
    - ~~Close-form algorithm - P3P~~
    - ~~Iterative algorithm - POSIT~~
    - ~~Iterative algorithm - Nonlinear least squares~~
  - Planar objects
    - Known patterns – Checkboard box, QR pattern
    - Planar Pictures -

Camera pose estimation

- Planar objects



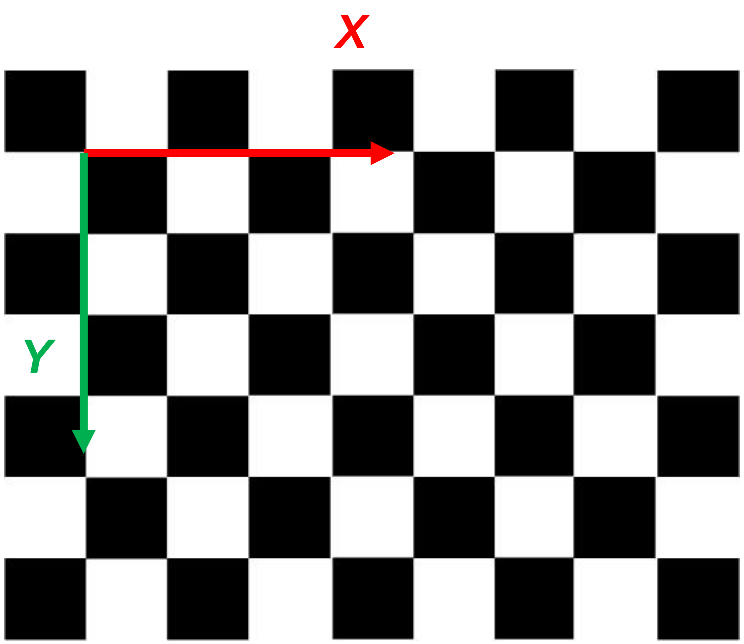
$$\boldsymbol{x}_i \leftrightarrow \boldsymbol{X}_i \quad (\boldsymbol{\pi}^{\mathrm{T}} \boldsymbol{X}_i) = 0$$

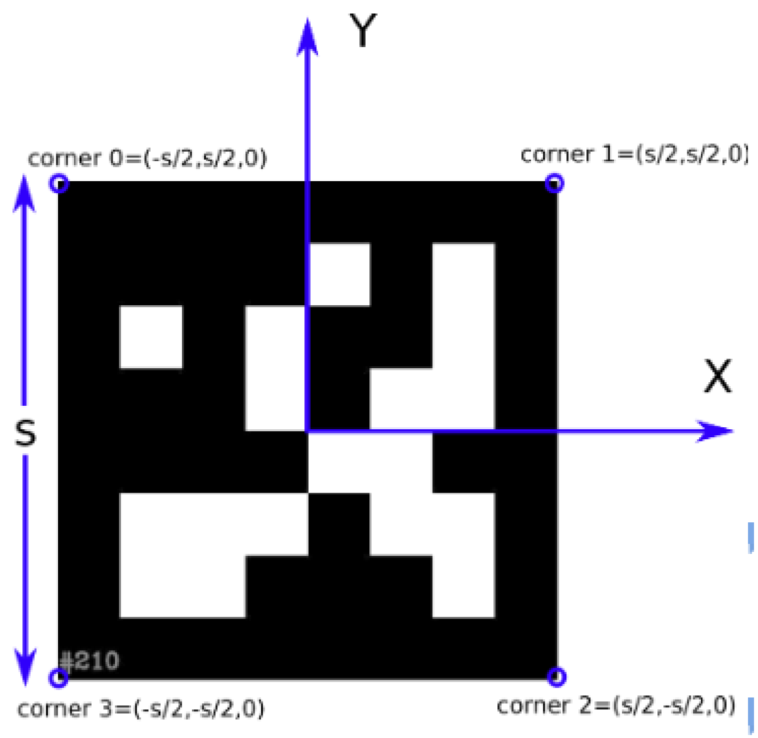
$$\Downarrow$$

$$\mathbf{R}, \boldsymbol{t}$$

Camera pose estimation

- Planar objects – Calibration board and QR codes

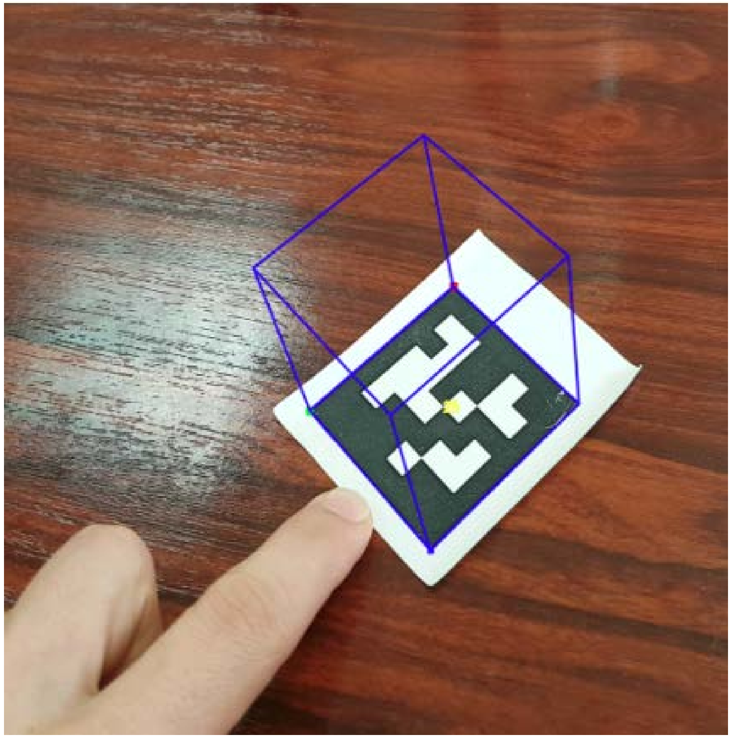




ArUco code

Camera pose estimation

- Planar objects – Calibration board and QR codes



$$\mathbf{x} \sim \mathbf{K}[\mathbf{R}\ \mathbf{t}]\mathbf{X}$$

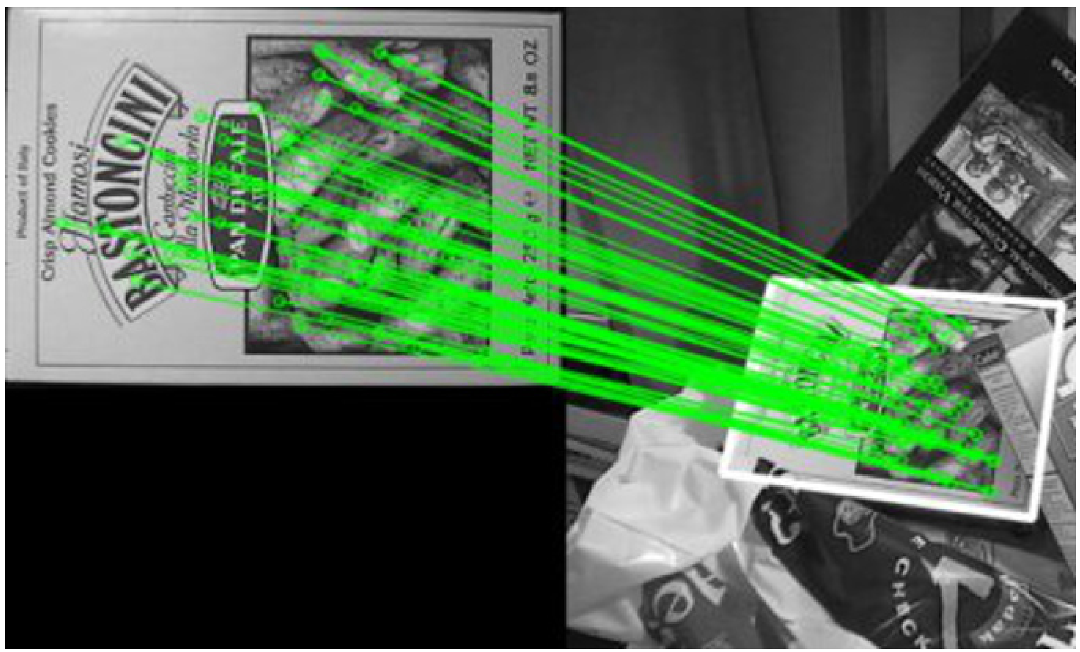
$$= \mathbf{K}[\mathbf{r}_1\ \mathbf{r}_2\ \mathbf{r}_3\ \mathbf{t}] \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \mathbf{K}[\mathbf{r}_1\ \mathbf{r}_2\ \mathbf{t}] \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

$$\boxed{\mathbf{x} \sim \mathbf{H}\tilde{\mathbf{X}}}$$

Camera pose estimation

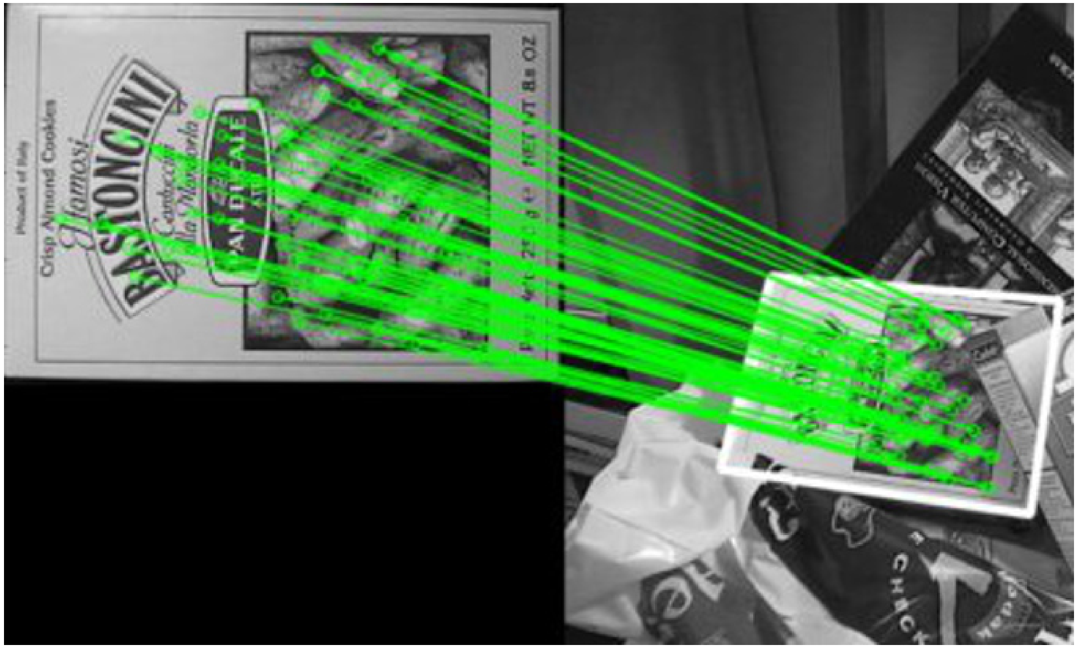
- Planar objects – Random patterns
  - The 2D coordinates of the points on the pattern are unknown.



Camera pose estimation

- Planar objects – Random patterns
  - The 2D coordinates of the points on the pattern are unknown.

$$\boldsymbol{X}_i = \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \ ?$$



$$\boldsymbol{x}_i = \mathbf{K}[\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{t}]\boldsymbol{X}_i \qquad \boldsymbol{x}'_i = \mathbf{K}[\boldsymbol{r}'_1, \boldsymbol{r}'_2, \boldsymbol{t}']\boldsymbol{X}_i$$

$$\boldsymbol{x}_i = \mathbf{H}\boldsymbol{X}_i \qquad \boldsymbol{x}'_i = \mathbf{H}'\boldsymbol{X}_i$$

Camera pose estimation

- Planar objects – Random patterns
  - If $\mathbf{R}, \boldsymbol{t}$ are known, usually $\mathbf{R} = \mathbf{I}, \boldsymbol{t} = [0, 0, s]^{\mathrm{T}}$

$$\boldsymbol{x}_i = \mathbf{K}[\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{t}]\boldsymbol{X}_i \qquad \boldsymbol{x}'_i = \mathbf{K}[\boldsymbol{r}'_1, \boldsymbol{r}'_2, \boldsymbol{t}']\boldsymbol{X}_i$$

$$\boldsymbol{x}_i = \mathbf{H}\boldsymbol{X}_i$$

$$\Downarrow$$

$$\boldsymbol{X}_i = \mathbf{H}^{-1}\boldsymbol{x}_i \quad \Rightarrow \quad \boxed{\boldsymbol{x}'_i} = \mathbf{H}'\boxed{\boldsymbol{X}_i}$$

$$\Rightarrow \quad \mathbf{H}'$$

$$\Rightarrow \quad \boldsymbol{r}'_1, \boldsymbol{r}'_2, \boldsymbol{t}' \rightarrow \mathbf{R}', \boldsymbol{t}'$$

Outline

- ~~**About pose estimation**~~
- ~~**3D-to-3D registration**~~
  - ~~Rotation only~~
  - ~~Rotation plus Translation~~
  - ~~Unknown correspondences – Iterative Closest Point (ICP)~~
- ~~**3D-to-2D registration (Camera pose estimation)**~~
  - ~~3D objects~~
    - ~~Close-form algorithm - P3P~~
    - ~~Iterative algorithm - POSIT~~
    - ~~Iterative algorithm - Nonlinear least squares~~
  - ~~Planar objects~~
    - ~~Known patterns – Checkboard box, QR pattern~~
    - ~~Planar Pictures – Random patterns~~

Pose estimation

- Open questions:
  - Texture-less objects
  - Symmetric objects