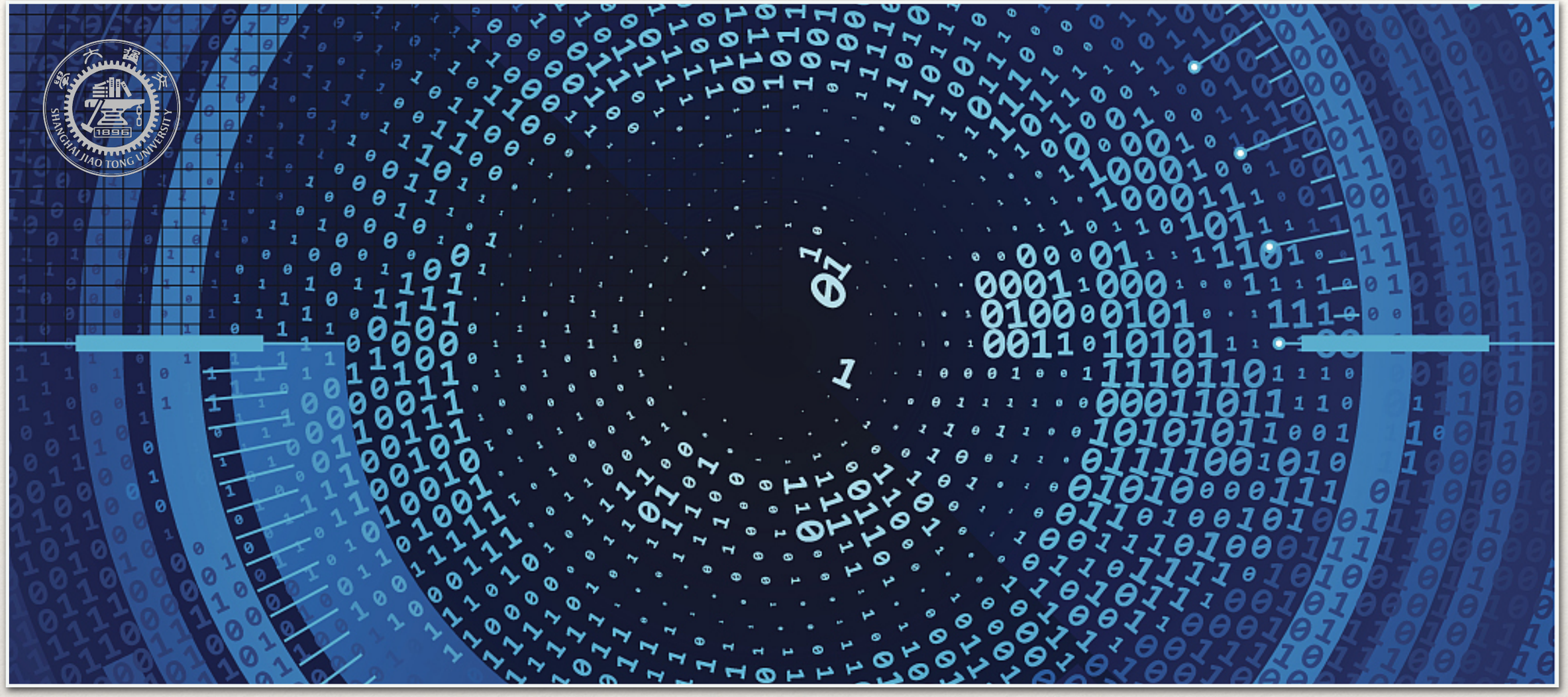


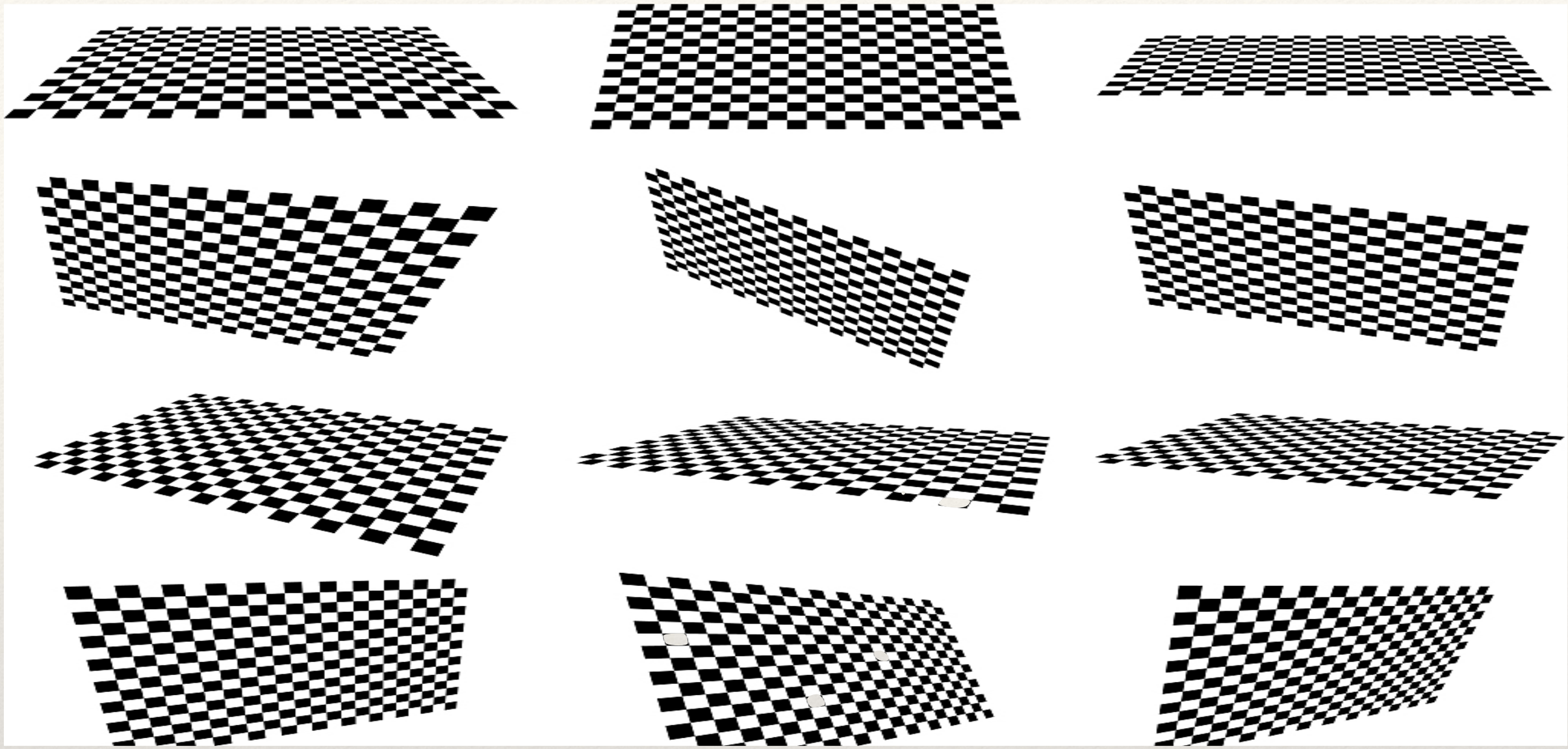
Shanghai Jiao Tong University

Computer Vision

Instructor: Xu Zhao

Class No.: C032703 F032528

Spring 2020



Xu Zhao @ Shanghai Jiao Tong university

Lecture 5-1: Calibration

Contents

- Geometric transformation
- Camera matrix
- Camera calibration

Transformation

- Parametric (global) transformations
  - Transformation T is a coordinate-changing machin: p’ = *T*(p)
  - What does it mean that *T* is global?
    - *T* is the same for any point p
    - *T* can be described by just a few numbers (parameters)
  - For linear transformations, we can represent T as a matrix

p’ = **T**p

p = (x,y)

p’ = (x’,y’)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{T} \begin{bmatrix} x \\ y \end{bmatrix}$$

Common transformations

Original

Transformed

Translation

Rotation

Scaling

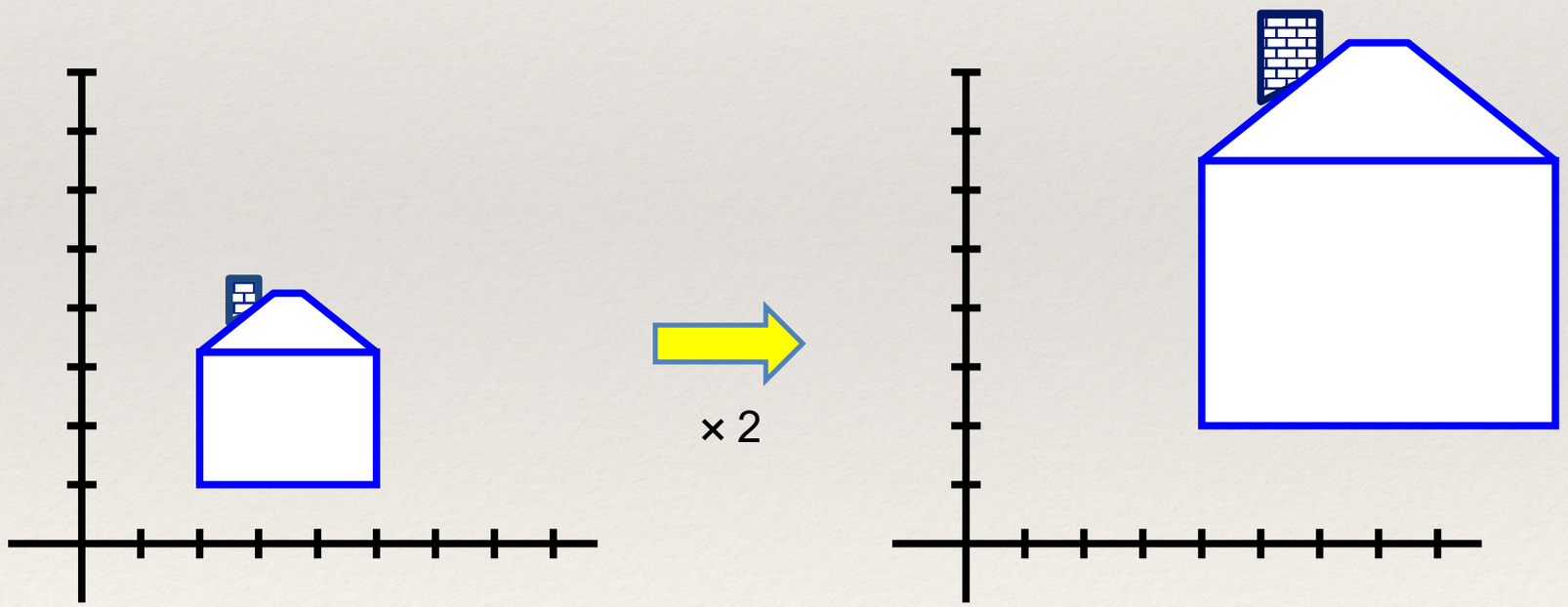
Affine

Perspective

Slide credit (next few slides):
A. Efros and/or S. Seitz

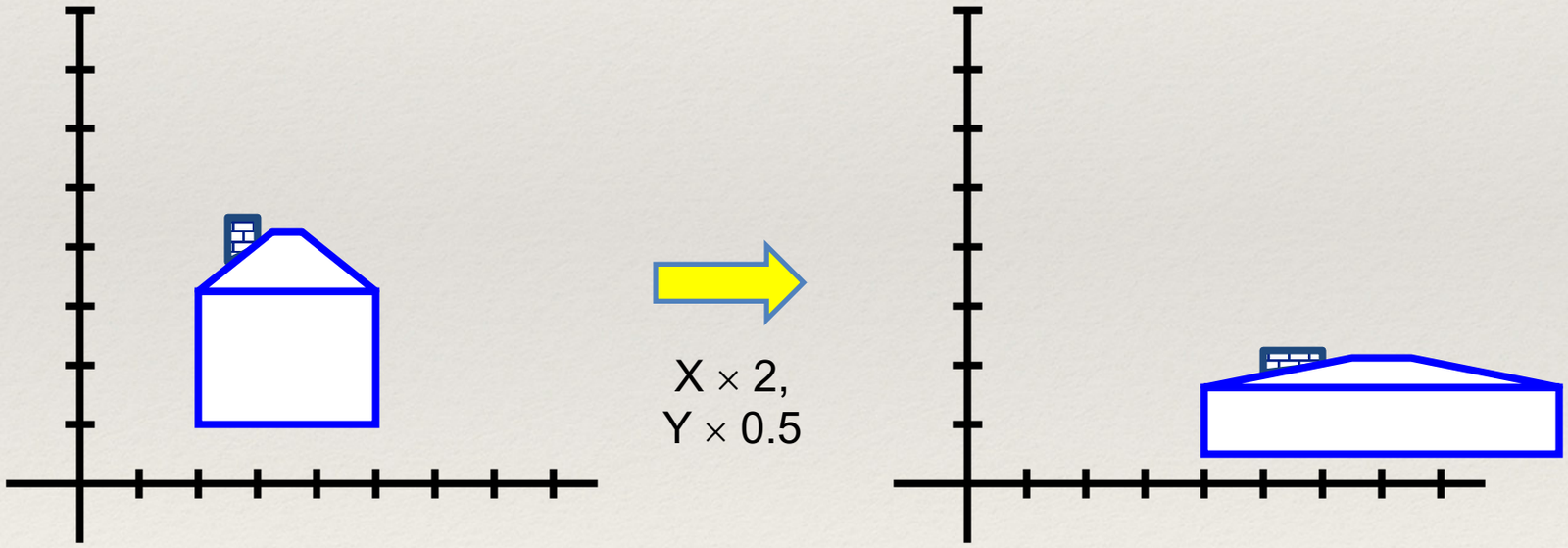
Scaling

- *Scaling* a coordinate means multiplying each of its components by a scalar
- *Uniform scaling* means this scalar is the same for all components:



Scaling

- *Non-uniform scaling*: different scalars per component



Scaling

- Scaling operation:

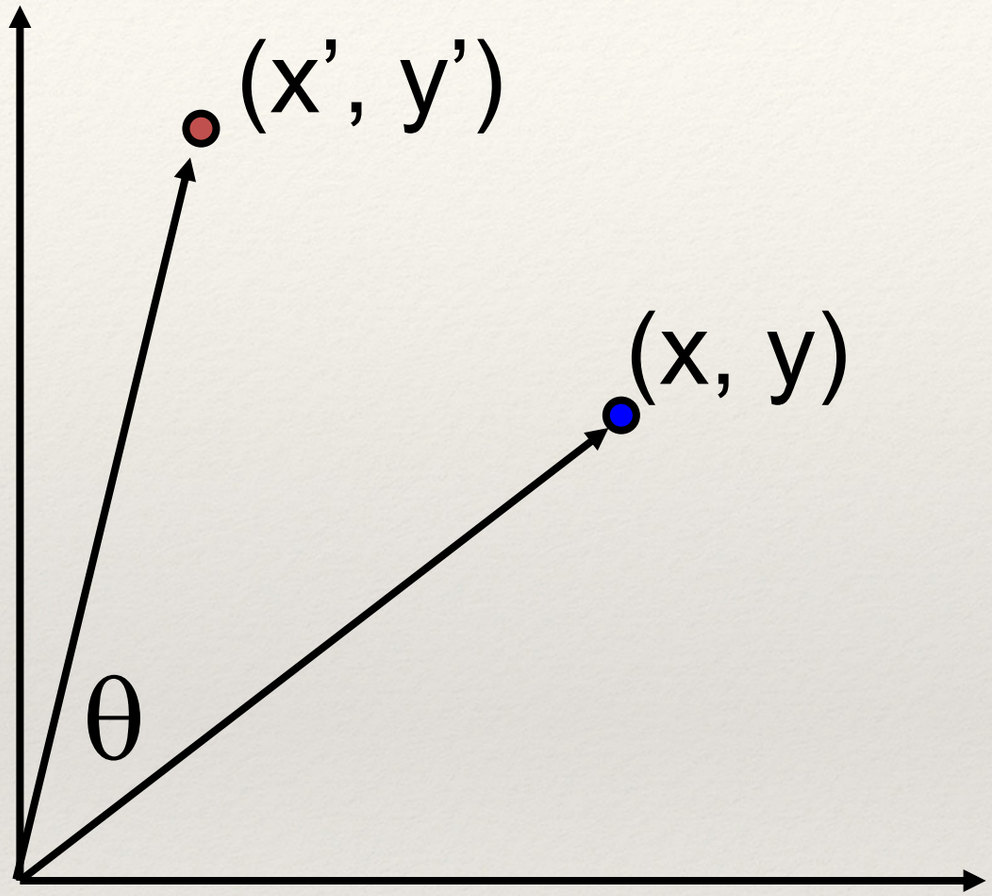
$$x' = ax$$

$$y' = by$$

- Or, in matrix form:

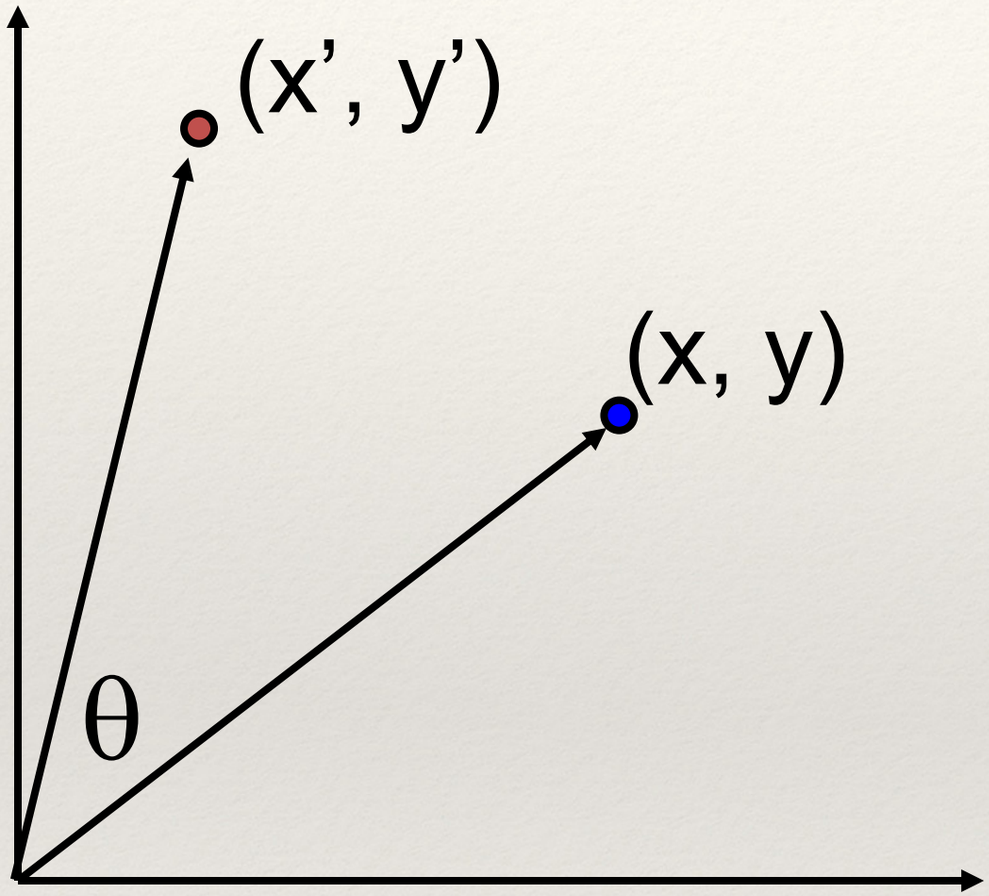
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}}_{\text{scaling matrix } S} \begin{bmatrix} x \\ y \end{bmatrix}$$

2-D Rotation



$$x' = x\ \mathbf{cos}(\theta) - y\ \mathbf{sin}(\theta)$$
$$y' = x\ \mathbf{sin}(\theta) + y\ \mathbf{cos}(\theta)$$

2-D Rotation



$x' = x \cos(\theta) - y \sin(\theta)$
$y' = x \sin(\theta) + y \cos(\theta)$

Polar coordinates…

$x = r \cos(\phi)$
$y = r \sin(\phi)$
$x' = r \cos(\phi + \theta)$
$y' = r \sin(\phi + \theta)$

Trig Identity…

$x' = r \cos(\phi) \cos(\theta) - r \sin(\phi) \sin(\theta)$
$y' = r \sin(\phi) \cos(\theta) + r \cos(\phi) \sin(\theta)$

Substitute…

$x' = x \cos(\theta) - y \sin(\theta)$
$y' = x \sin(\theta) + y \cos(\theta)$

2-D Rotation

- This is easy to capture in matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}}_{\mathbf{R}} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Even though $sin\theta$ and $cos\theta$ are nonlinear functions of θ
  - x′ is a linear combination of x and y
  - y′ is a linear combination of x and y
- What is the inverse transformation?
  - Rotation by -θ
  - For rotation matrices $\mathbf{R}^{-1} = \mathbf{R}^{T}$

Basic 2D transformations

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Scale

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & \alpha_x \\ \alpha_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Shear

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Rotate

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translate

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

Affine

Affine is any combination of translation, scale, rotation, and shear

Affine Transformations

- Affine transformations are combinations of
  - Linear transformations, and
  - Translations

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

- Properties of affine transformations:
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition

or

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

Projective transformations

- Projective transformations are combos of
  - Affine transformations, and
  - Projective warps
- Properties of projective transformations:
  - Lines map to lines
  - Parallel lines do not necessarily remain parallel
  - Ratios are not preserved
  - Closed under composition
  - Models change of basis
  - Projective matrix is defined up to a scale (8 DOF)

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

2D transformations

Isometries

- Isometries are transformations that preserve Euclidean distance
- Invariants: length (the distance between two points), angle (the angle between two lines), and area

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} \epsilon\cos\theta & -\sin\theta & t_x \\ \epsilon\sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

where $\epsilon = \pm 1$.

$$\mathbf{x}' = \mathrm{H}_{\mathrm{E}}\mathbf{x} = \begin{bmatrix} \mathrm{R} & \mathbf{t} \\ \mathbf{0}^{\mathsf{T}} & 1 \end{bmatrix} \mathbf{x}$$

2D transformations

Similarity

- Is an isometry composed with an isotropic scaling

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} s\cos\theta & -s\sin\theta & t_x \\ s\sin\theta & s\cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- Invariants: angle, ratio of length, ratio of area

$$\mathbf{x}' = \mathrm{H_S}\mathbf{x} = \begin{bmatrix} s\mathrm{R} & \mathbf{t} \\ \mathbf{0}^\mathsf{T} & 1 \end{bmatrix} \mathbf{x}$$

2D transformations

Affine

- Is a non-singular linear transformation followed by a translation

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- Invariants: parallel lines; ratio of lengths of parallel line segments; ratio of area

$$\mathbf{x}' = \mathrm{H}_\mathrm{A}\mathbf{x} = \begin{bmatrix} \mathrm{A} & \mathbf{t} \\ \mathbf{0}^\mathsf{T} & 1 \end{bmatrix} \mathbf{x}$$

$$\mathrm{A} = \mathrm{R}(\theta)\,\mathrm{R}(-\phi)\,\mathrm{D}\,\mathrm{R}(\phi)$$

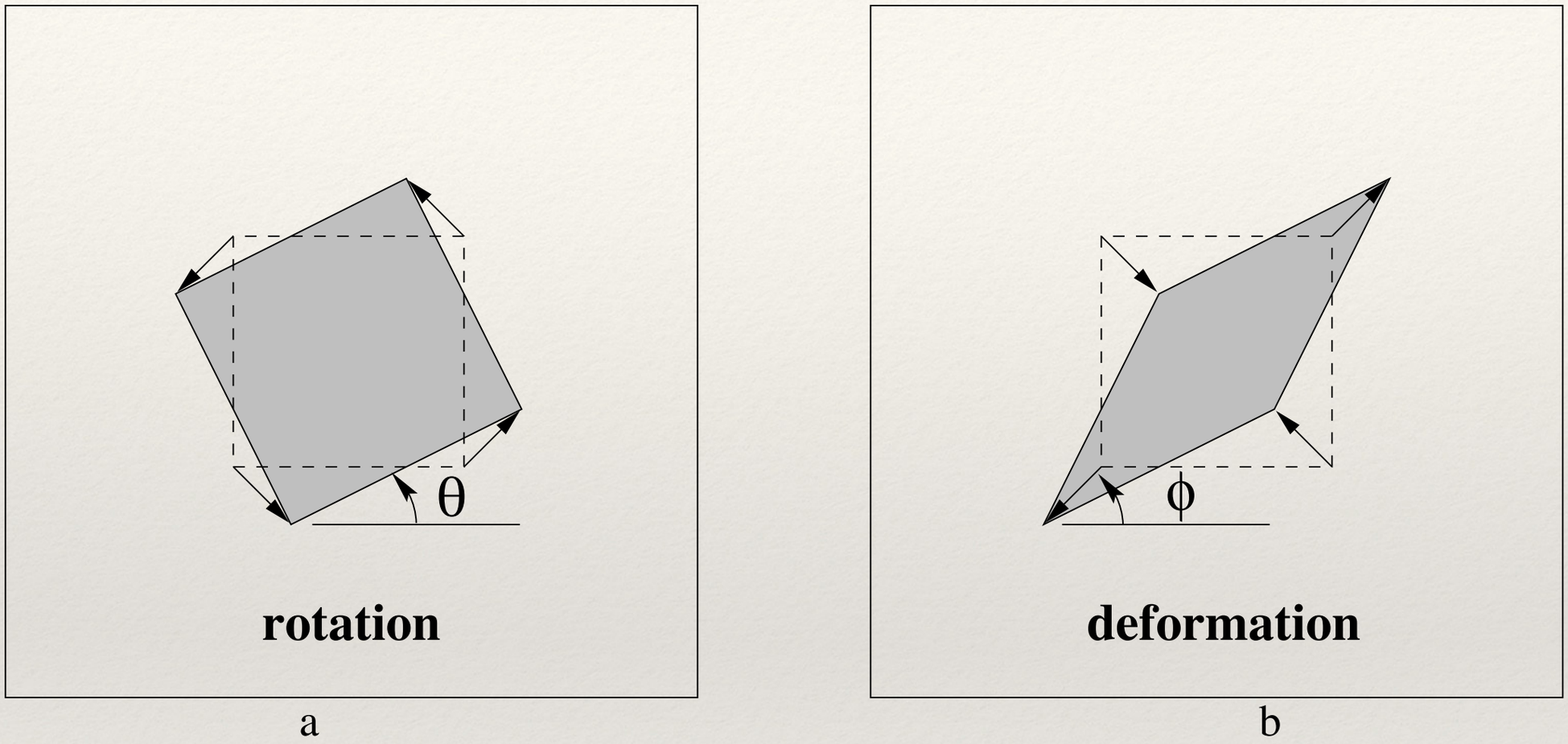


Fig. 2.7. **Distortions arising from a planar affine transformation.** *(a) Rotation by* $\mathtt{R}(\theta)$. *(b) A deformation* $\mathtt{R}(-\phi)\,\mathtt{D}\,\mathtt{R}(\phi)$. *Note, the scaling directions in the deformation are orthogonal.*

2D transformations

Projective (Homogragy)

- A general non-singular linear transformation of *homogeneous* coordinates
- Invariants: cross ratio of four collinear points

$$\mathbf{x}' = \mathtt{H}_{\mathrm{P}}\mathbf{x} = \begin{bmatrix} \mathtt{A} & \mathbf{t} \\ \mathbf{v}^{\mathsf{T}} & v \end{bmatrix} \mathbf{x}$$

$$\mathbf{v} = (v_1, v_2)^{\mathsf{T}}$$

2D transformations

For ideal point (x_1, x_2, 0)

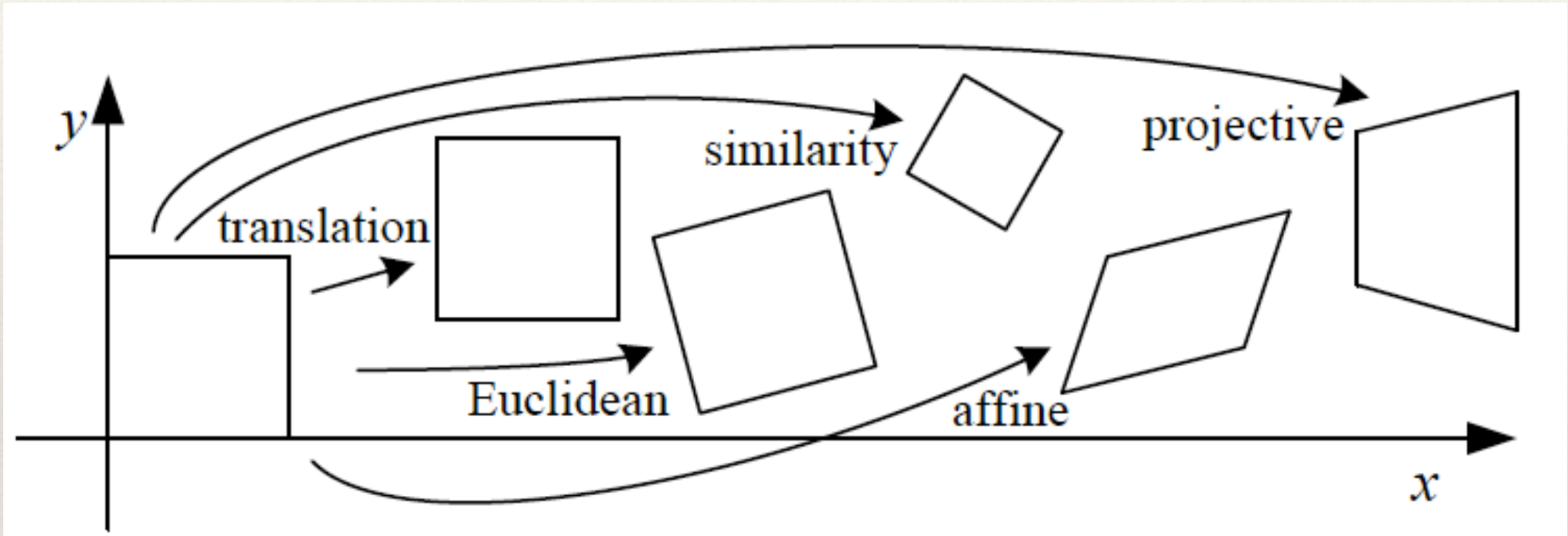
$$\begin{bmatrix} \mathtt{A} & \mathbf{t} \\ \mathbf{0}^\mathsf{T} & 1 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 0 \end{pmatrix} = \begin{pmatrix} \mathtt{A} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\ 0 \end{pmatrix}$$

Affine

$$\begin{bmatrix} \mathtt{A} & \mathbf{t} \\ \mathbf{v}^\mathsf{T} & v \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 0 \end{pmatrix} = \begin{pmatrix} \mathtt{A} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\ v_1 x_1 + v_2 x_2 \end{pmatrix}$$

Projective

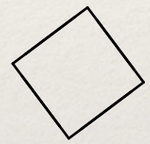
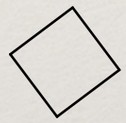
2D transformations



2D transformations

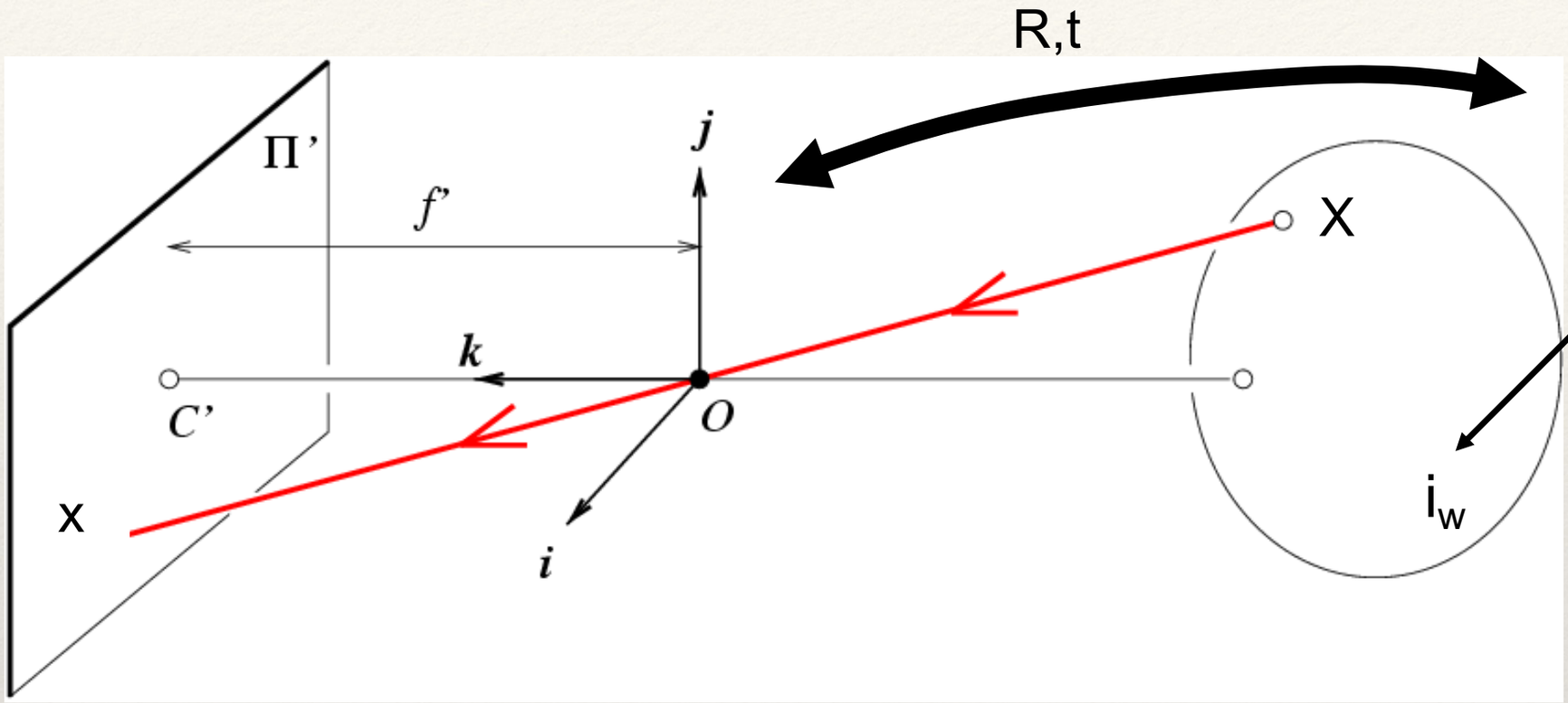
Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	3	lengths	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	4	angles	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times3}$	8	straight lines	

3D transformations

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{3\times4}$	3	orientation	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{3\times4}$	6	lengths	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{3\times4}$	7	angles	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{3\times4}$	12	parallelism	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{4\times4}$	15	straight lines	

Szeliski.

Camera (projection) matrix



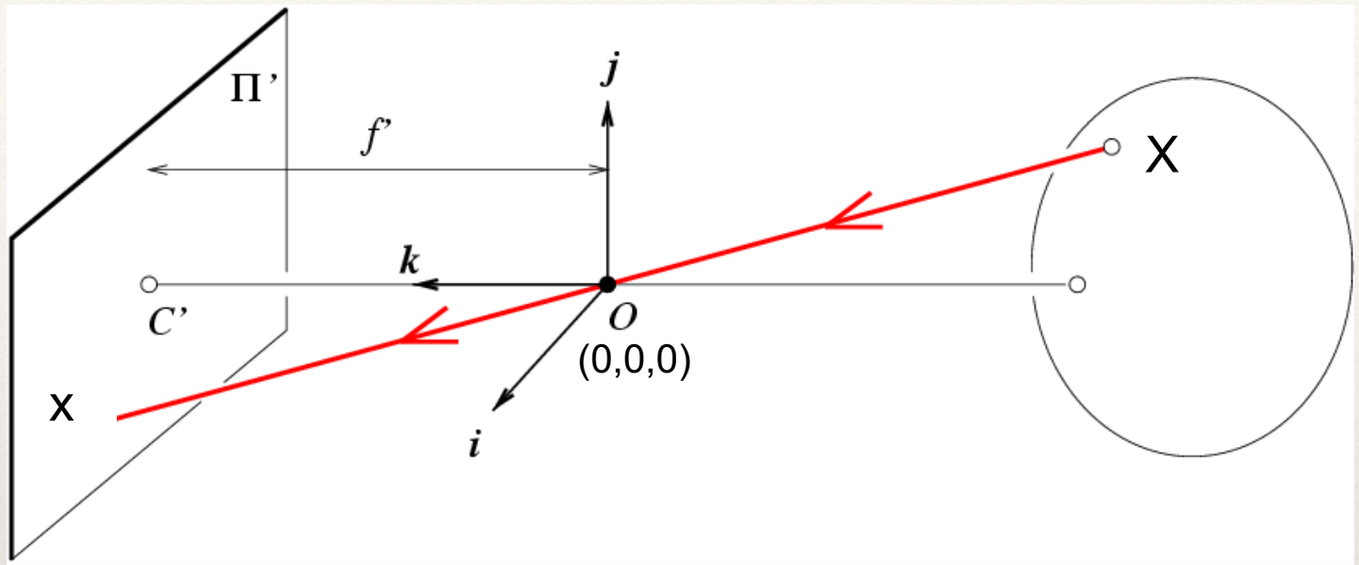
$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}\mathbf{X}$$

Extrinsic Matrix

x: Image Coordinates: (u,v,1)
K: Intrinsic Matrix (3x3)
R: Rotation (3x3)
t: Translation (3x1)
X: World Coordinates: (X,Y,Z,1)

Slide Credit: Savarese

Projection matrix



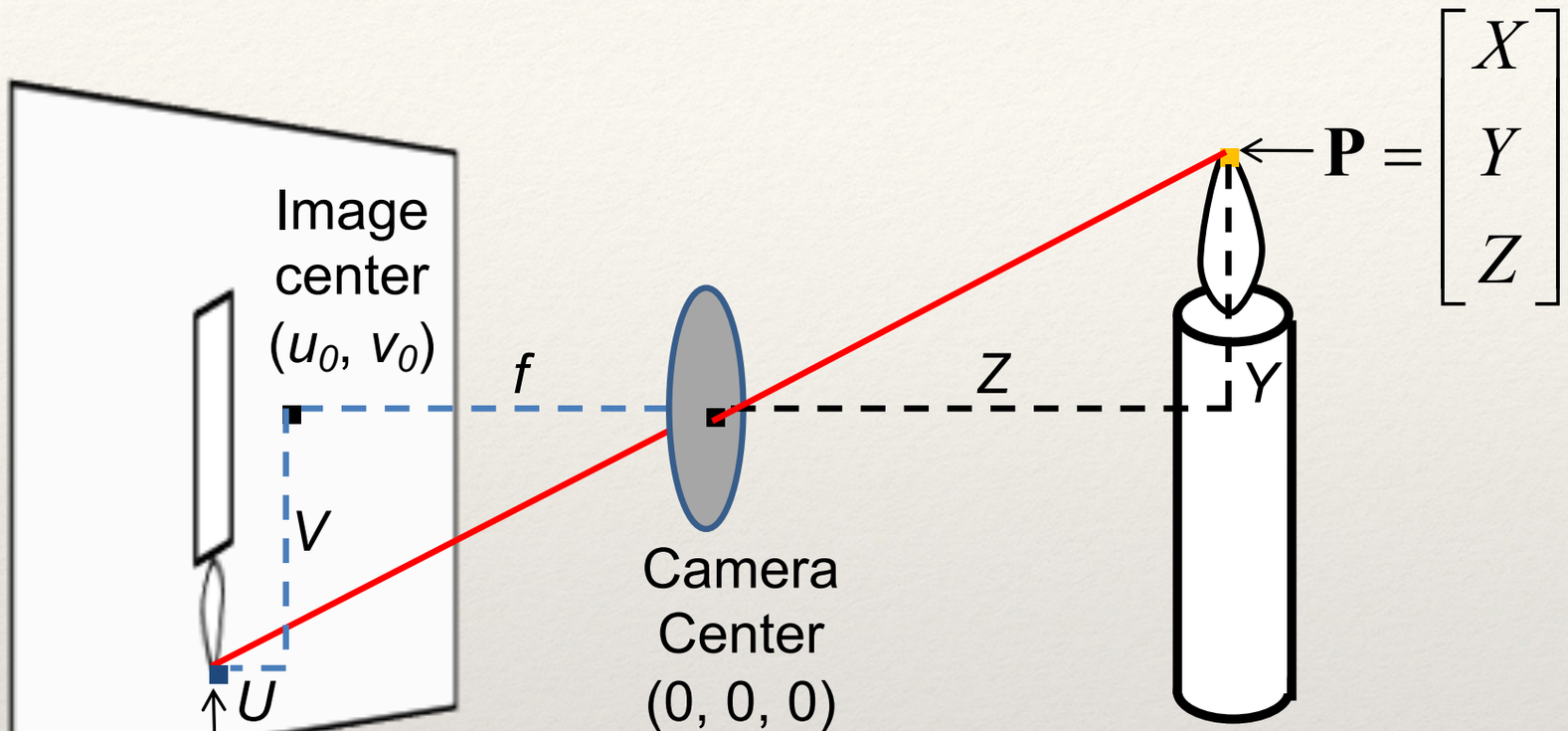
Intrinsic Assumptions
- Unit aspect ratio
- Optical center at (0,0)
- No skew

Extrinsic Assumptions
- No rotation
- Camera at (0,0,0)

$$\mathbf{x} = \mathbf{K}[\mathbf{I} \quad \mathbf{0}]\mathbf{X} \Rightarrow w\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \overbrace{\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}^{\mathbf{K}} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Slide Credit: Savarese

Projection: world coordinates → image coordinates



p = distance from image center

$$U = -X * \frac{f}{Z} \qquad V = -Y * \frac{f}{Z}$$

Slide Credit: Savarese

Remove assumption: known optical center

Intrinsic Assumptions
- Unit aspect ratio
- No skew

Extrinsic Assumptions
- No rotation
- Camera at (0,0,0)

$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{I} & \mathbf{0}\end{bmatrix}\mathbf{X} \Rightarrow w\begin{bmatrix}u\\v\\1\end{bmatrix} = \begin{bmatrix}f & 0 & u_0 & 0\\0 & f & v_0 & 0\\0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}$$

K

Principal point offset

Remove assumption: equal aspect ratio

Intrinsic Assumptions
- No skew

Extrinsic Assumptions
- No rotation
- Camera at (0,0,0)

$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{I} & \mathbf{0}\end{bmatrix}\mathbf{X} \Rightarrow w\begin{bmatrix}u\\ v\\ 1\end{bmatrix} = \begin{bmatrix}f_x & 0 & u_0 & 0\\ 0 & f_y & v_0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}x\\ y\\ z\\ 1\end{bmatrix}$$

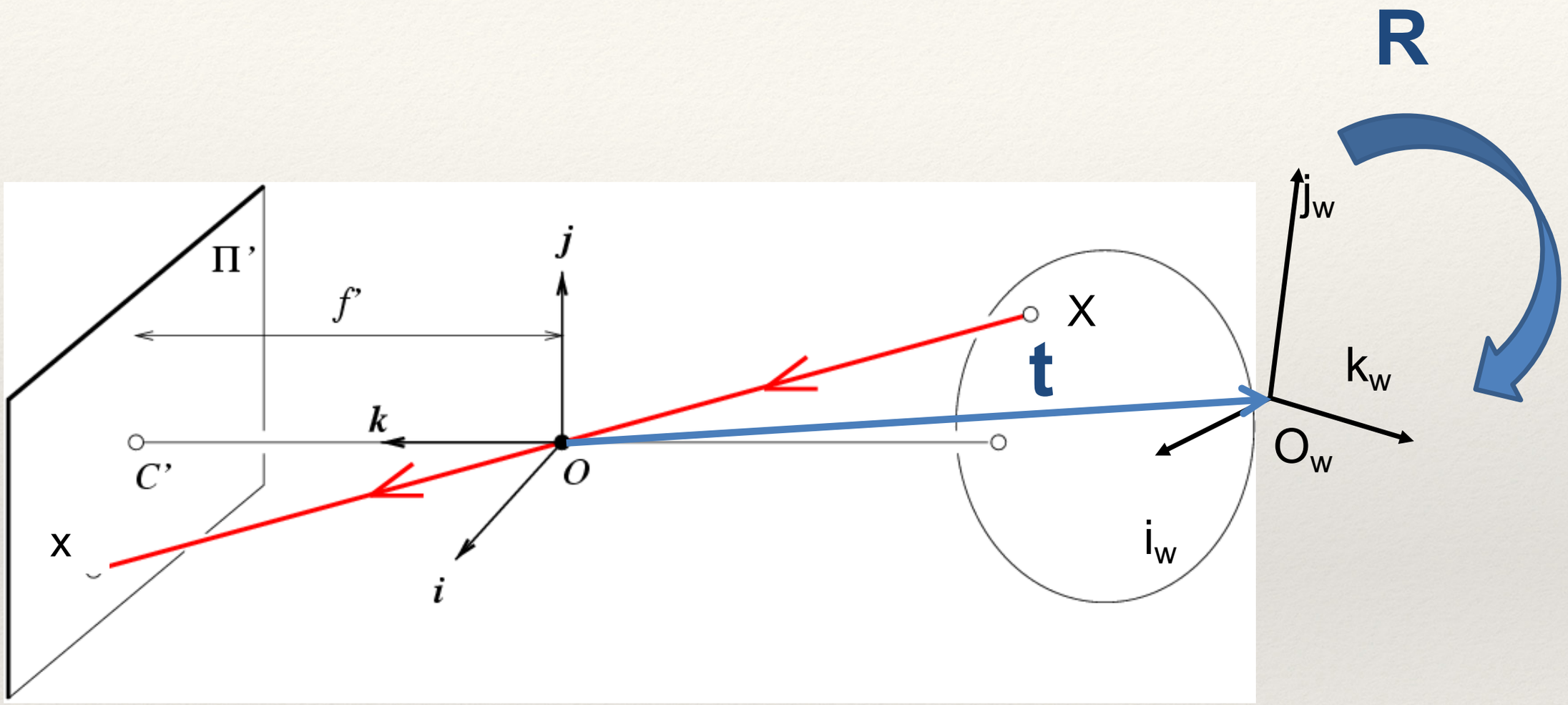
Remove assumption: non-skewed pixels

Intrinsic Assumptions

Extrinsic Assumptions
- No rotation
- Camera at (0,0,0)

$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{I} & \mathbf{0}\end{bmatrix}\mathbf{X} \Rightarrow w\begin{bmatrix}u\\v\\1\end{bmatrix} = \begin{bmatrix}f_x & s & u_0 & 0\\0 & f_y & v_0 & 0\\0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}$$

Oriented and translated camera



Allow camera translation

Intrinsic Assumptions

Extrinsic Assumptions
- No rotation

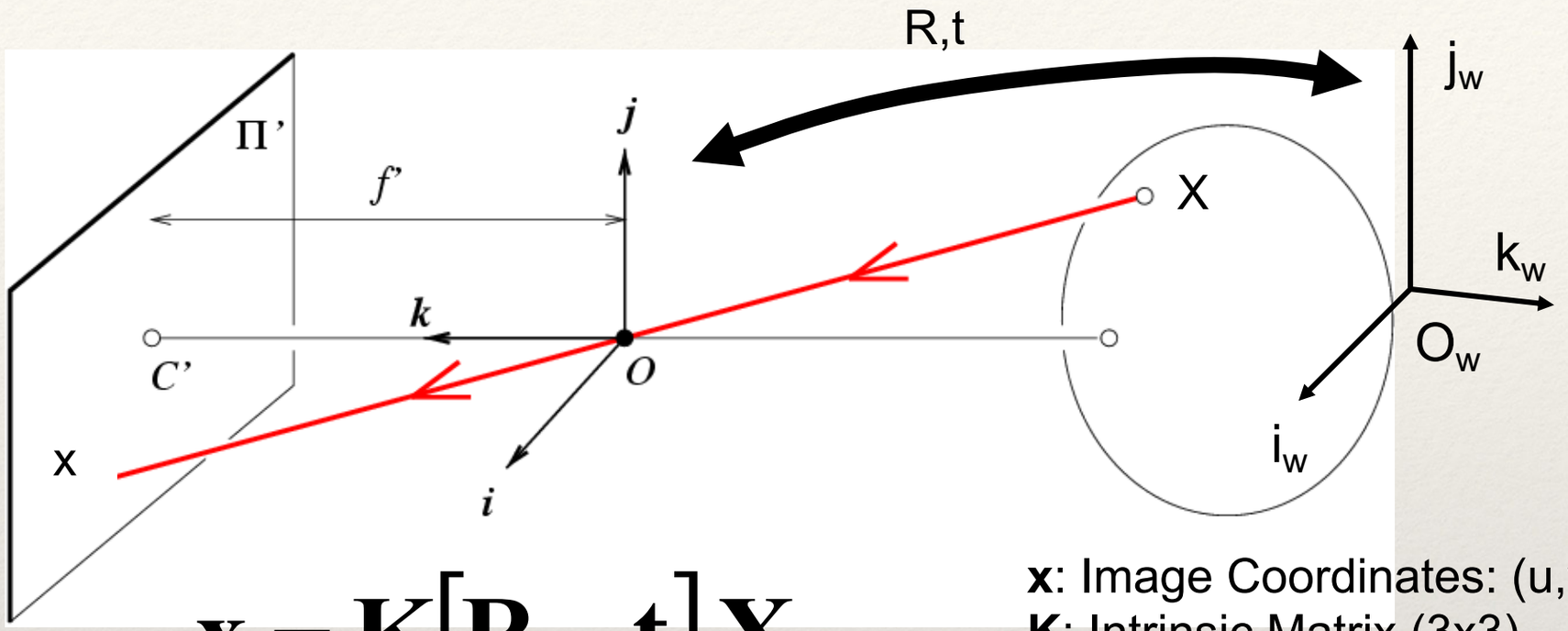
$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{I} & \mathbf{t}\end{bmatrix}\mathbf{X} \Rightarrow w\begin{bmatrix}u\\v\\1\end{bmatrix} = \begin{bmatrix}f_x & s & u_0\\0 & f_y & v_0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0 & t_x\\0 & 1 & 0 & t_y\\0 & 0 & 1 & t_z\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}$$

Allow rotation

$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}\mathbf{X}$$

$$w\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Camera (projection) matrix



$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}\mathbf{X}$$

Extrinsic Matrix

x: Image Coordinates: (u,v,1)
K: Intrinsic Matrix (3x3)
R: Rotation (3x3)
t: Translation (3x1)
X: World Coordinates: (X,Y,Z,1)

$$w\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Degrees of freedom

$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}\mathbf{X}$$

5 6

$$w\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Vanishing Point = Projection from Infinity

$$\mathbf{p} = \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix} \Rightarrow \mathbf{p} = \mathbf{KR}\begin{bmatrix} x \\ y \\ z \end{bmatrix} \Rightarrow \mathbf{p} = \mathbf{K}\begin{bmatrix} x_R \\ y_R \\ z_R \end{bmatrix}$$

$$w\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x_R \\ y_R \\ z_R \end{bmatrix} \Rightarrow \quad u = \frac{fx_R}{z_R} + u_0 \qquad v = \frac{fy_R}{z_R} + v_0$$

How to calibrate the camera?

- Also called "camera resectioning"

$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}\mathbf{X}$$

$$\mathbf{x} = \mathbf{M}\mathbf{X}$$

$$\begin{bmatrix} wu \\ wv \\ w \end{bmatrix} = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

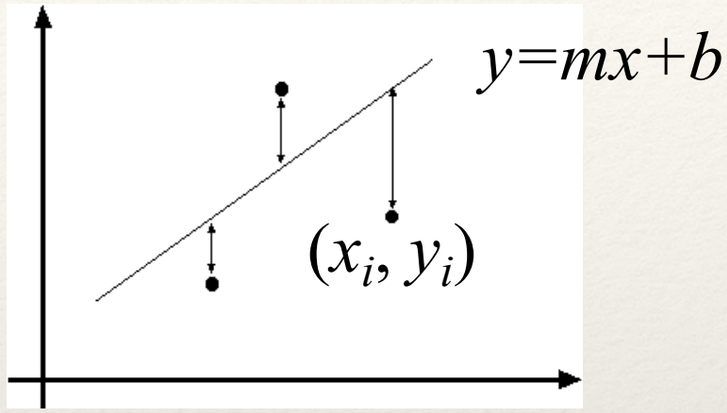
Linear least-squares regression!

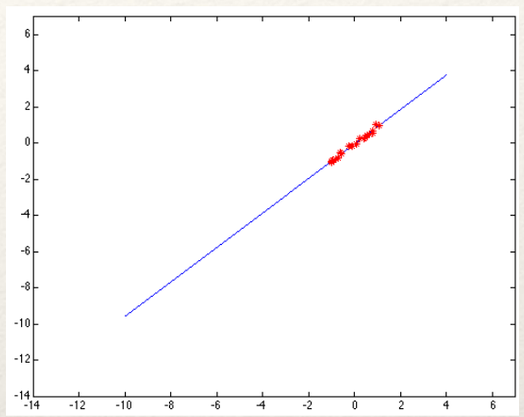
James Hays

Least squares line fitting

- Data: $(x_1, y_1), \ldots, (x_n, y_n)$
- Line equation: $y_i = m x_i + b$
- Find (m, b) to minimize

$$E = \sum_{i=1}^{n} (y_i - mx_i - b)^2$$





$$E = \sum_{i=1}^{n} \left(\begin{bmatrix} x_i & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - y_i \right)^2 = \left\| \begin{bmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \right\|^2 = \|\mathbf{Ap} - \mathbf{y}\|^2$$

$$= \mathbf{y}^T\mathbf{y} - 2(\mathbf{Ap})^T\mathbf{y} + (\mathbf{Ap})^T(\mathbf{Ap})$$

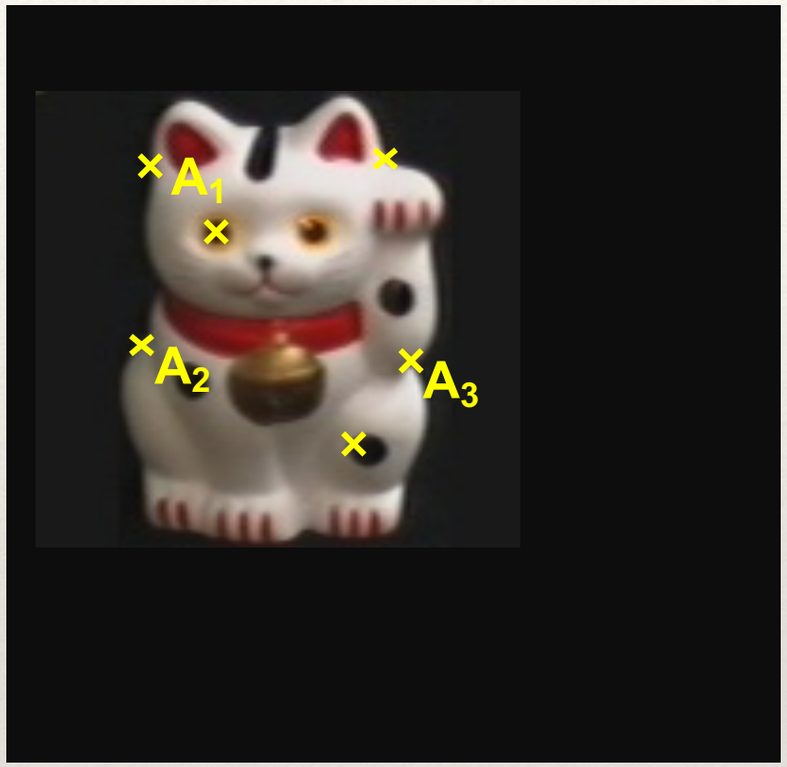
$$\frac{dE}{dp} = 2\mathbf{A}^T\mathbf{Ap} - 2\mathbf{A}^T\mathbf{y} = 0$$

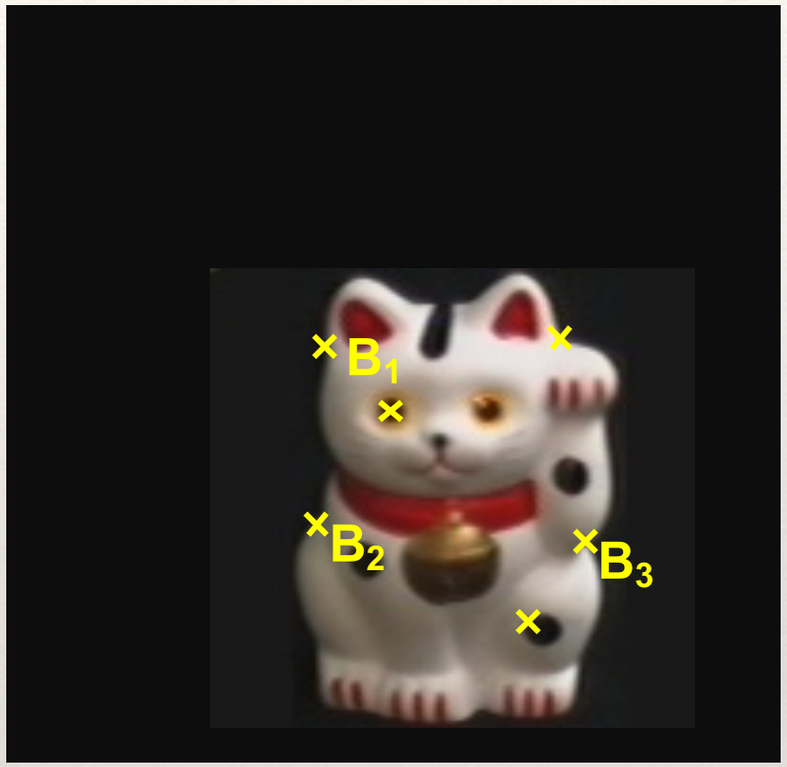
```
Matlab: p = A \ y;
Python:
p = np.linalg.lstsq(A,y)[0]
```

$$\mathbf{A}^T\mathbf{Ap} = \mathbf{A}^T\mathbf{y} \Rightarrow \mathbf{p} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{y}$$ (Closed form solution)

Modified from S. Lazebnik

Example: solving for translation

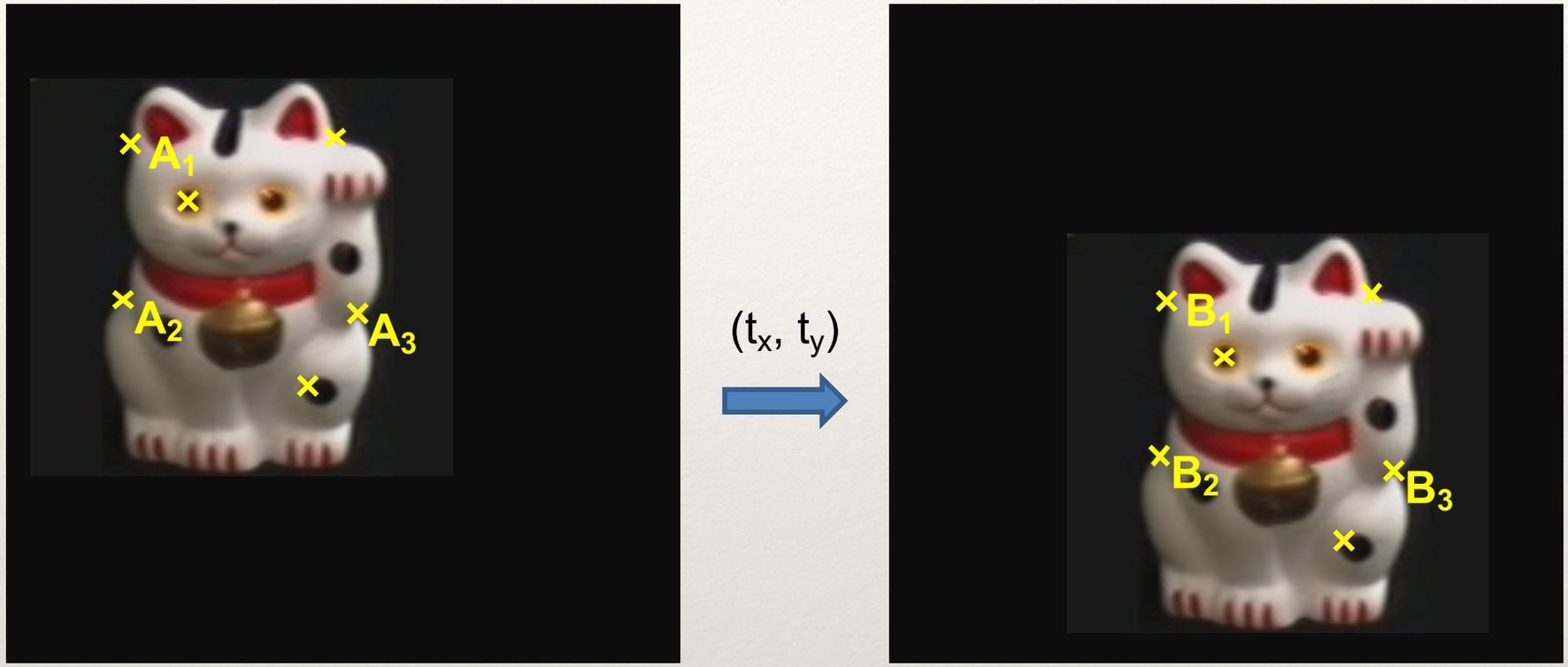




Given matched points in {A} and {B}, estimate the translation of the object

$$\begin{bmatrix} x_i^B \\ y_i^B \end{bmatrix} = \begin{bmatrix} x_i^A \\ y_i^A \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Example: solving for translation

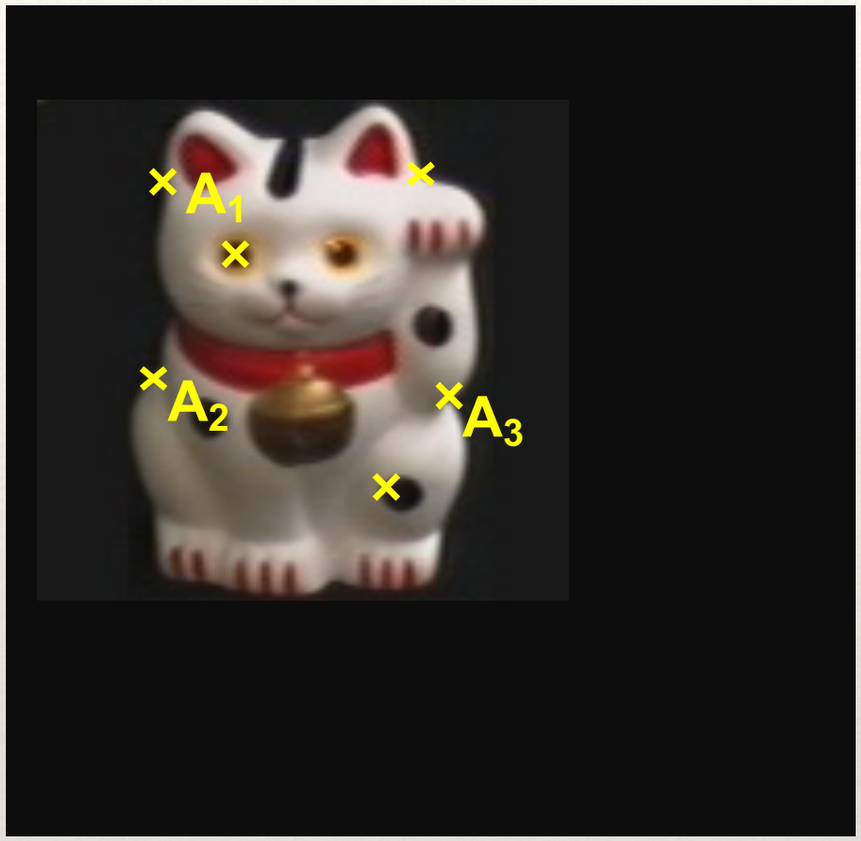


$$\begin{bmatrix} x_i^B \\ y_i^B \end{bmatrix} = \begin{bmatrix} x_i^A \\ y_i^A \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Least squares setup

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ \vdots & \vdots \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x_1^B - x_1^A \\ y_1^B - y_1^A \\ \vdots \\ x_n^B - x_n^A \\ y_n^B - y_n^A \end{bmatrix}$$

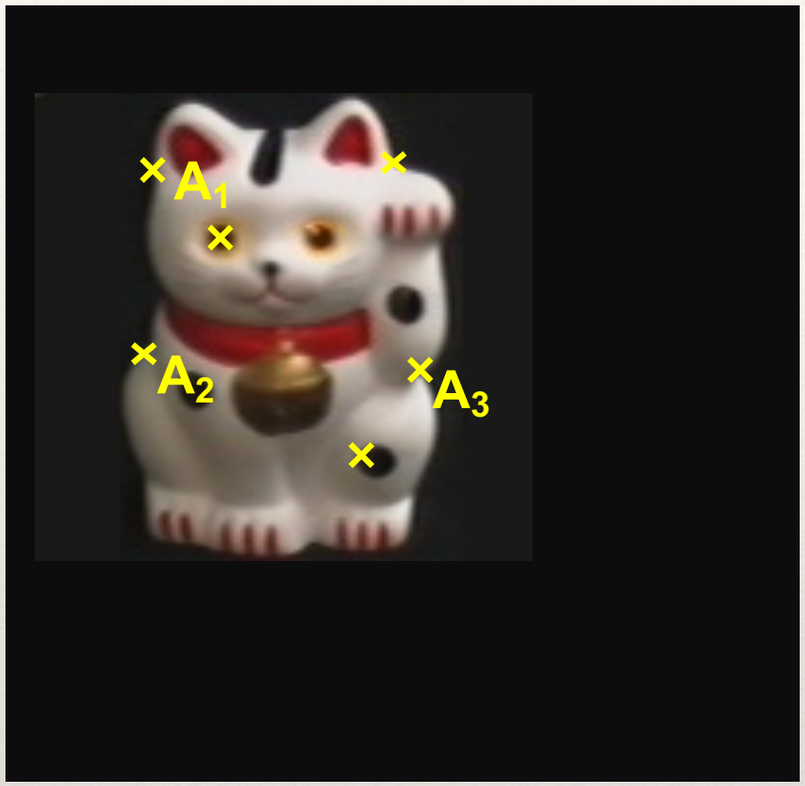
Example: solving for translation



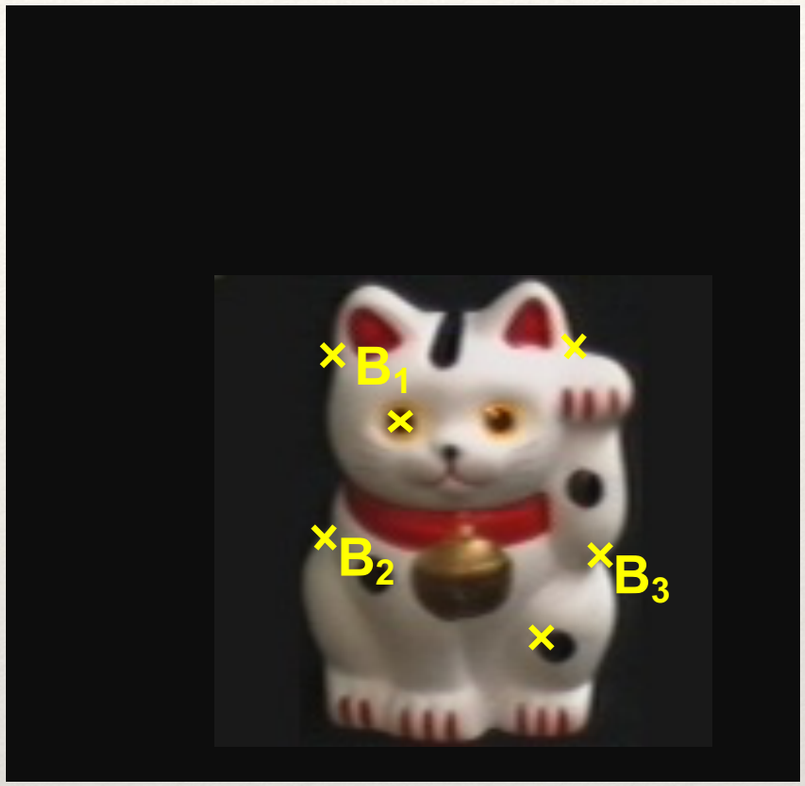
Given matched points in {A} and {B}, estimate the transformation matrix

$$\begin{bmatrix} x_i^B \\ y_i^B \end{bmatrix} = \mathrm{T}\begin{bmatrix} x_i^A \\ y_i^A \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \qquad \mathrm{T} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Example: solving for translation



(t_x, t_y)



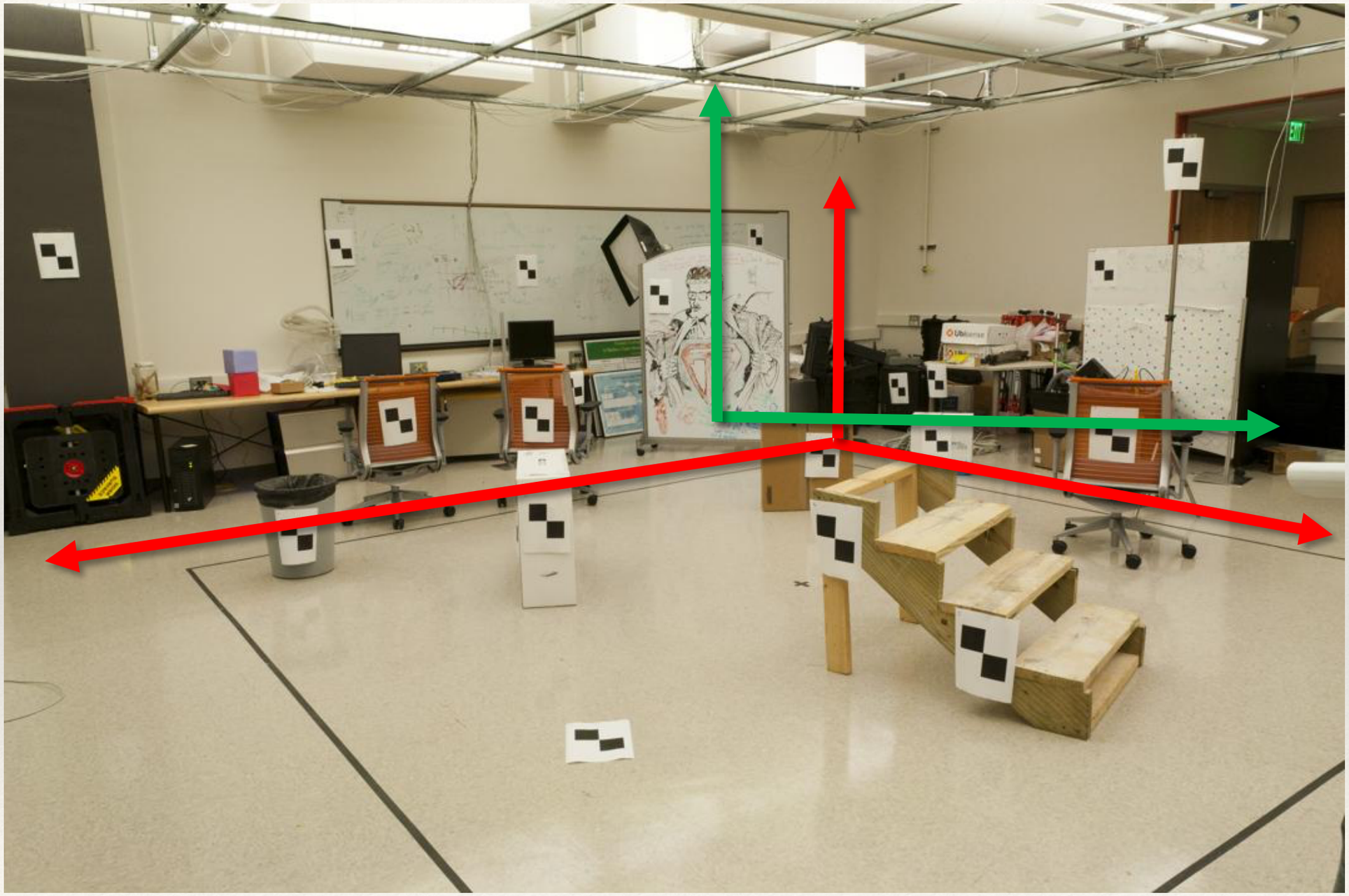
Least squares solution

1. Write down objective function
2. Derived solution
   a) Compute derivative
   b) Compute solution
3. Computational solution
   a) Write in form Ax=p
   b) Solve using closed-form solution

$$\begin{bmatrix} x_i^B \\ y_i^B \end{bmatrix} = \begin{bmatrix} x_i^A \\ y_i^A \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ \vdots & \vdots \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x_1^B - x_1^A \\ y_1^B - y_1^A \\ \vdots \\ x_n^B - x_n^A \\ y_n^B - y_n^A \end{bmatrix}$$

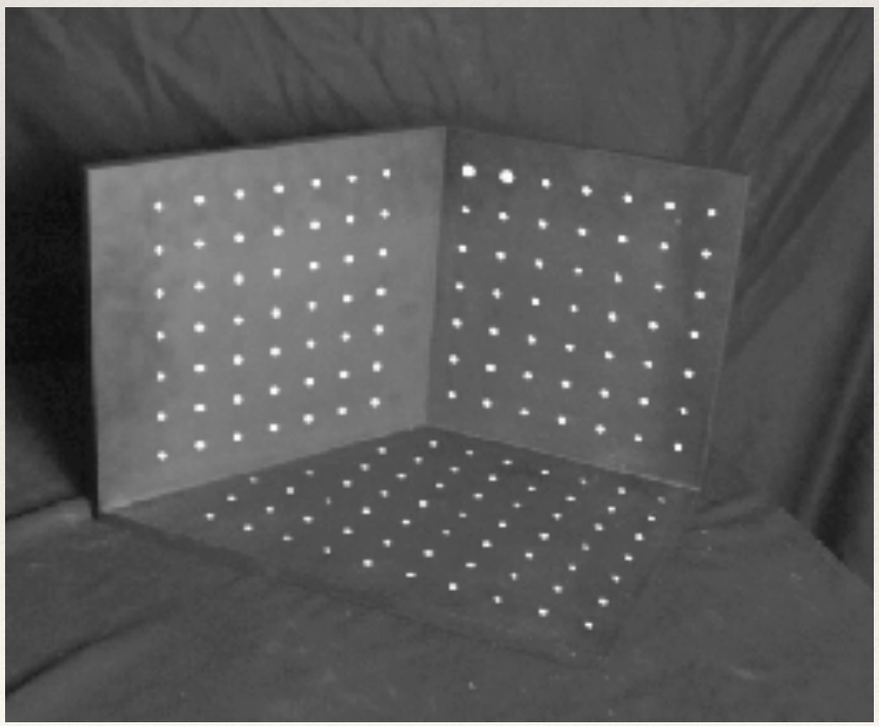
World vs Camera coordinates



Calibrating the Camera

Use an scene with **known** geometry

- Correspond image points to 3d points
- Get least squares solution (or non-linear solution)



Known 2d — image coords

Known 3d — world locations

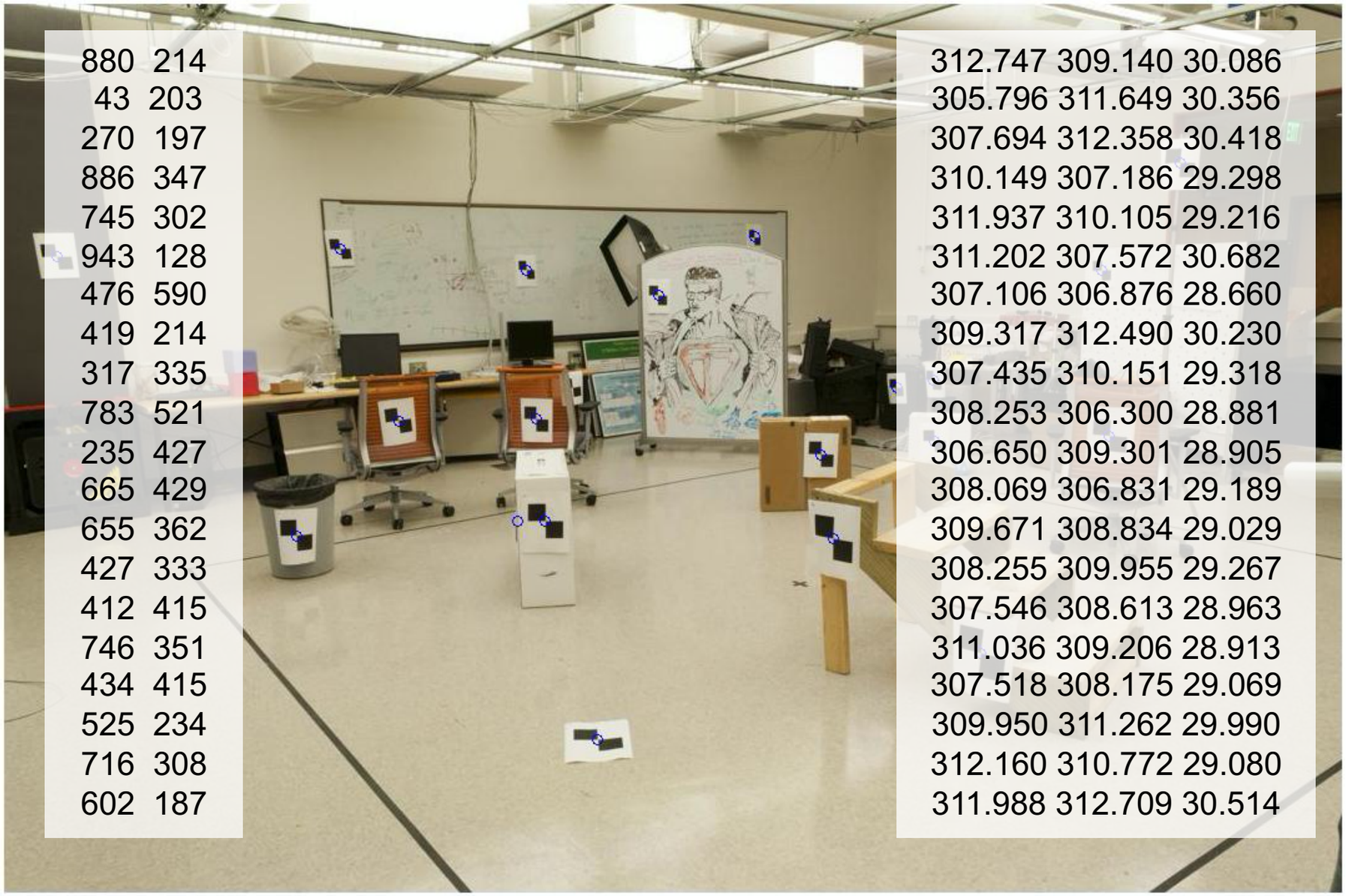
$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \underbrace{\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix}}_{\mathbf{M}} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Unknown Camera Parameters

How do we calibrate a camera?

Known 2d image coords

Known 3d world locations

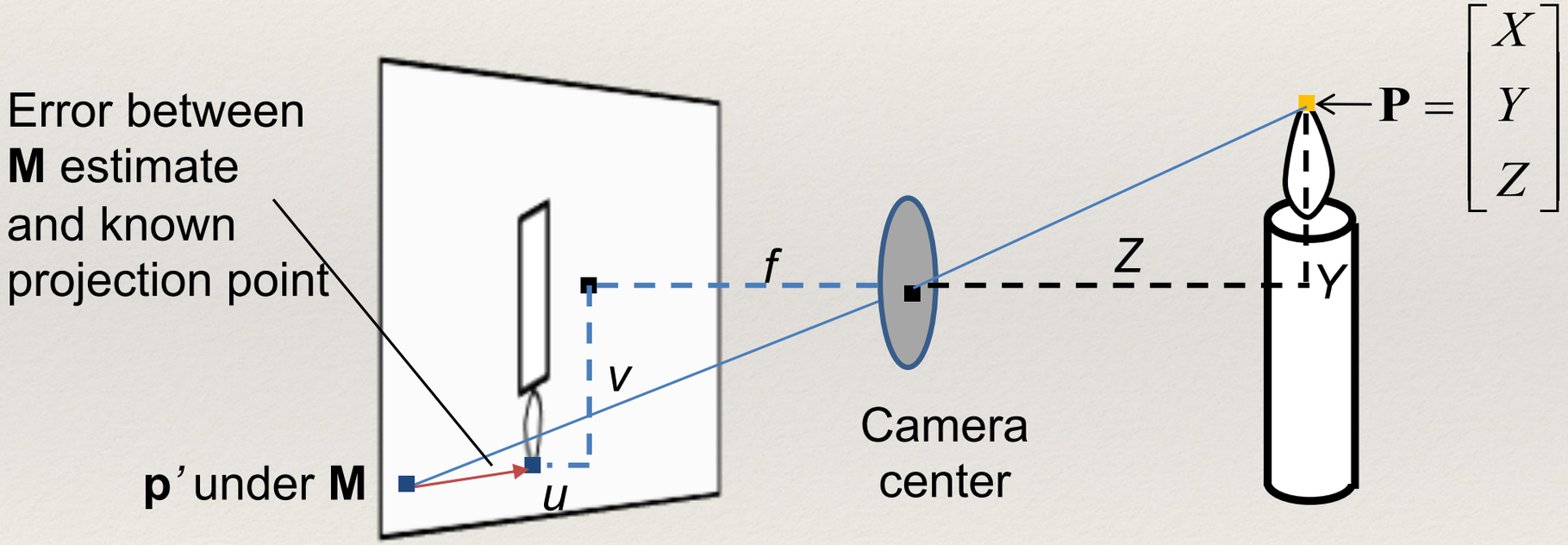


880	214
43	203
270	197
886	347
745	302
943	128
476	590
419	214
317	335
783	521
235	427
665	429
655	362
427	333
412	415
746	351
434	415
525	234
716	308
602	187

312.747	309.140	30.086
305.796	311.649	30.356
307.694	312.358	30.418
310.149	307.186	29.298
311.937	310.105	29.216
311.202	307.572	30.682
307.106	306.876	28.660
309.317	312.490	30.230
307.435	310.151	29.318
308.253	306.300	28.881
306.650	309.301	28.905
308.069	306.831	29.189
309.671	308.834	29.029
308.255	309.955	29.267
307.546	308.613	28.963
311.036	309.206	28.913
307.518	308.175	29.069
309.950	311.262	29.990
312.160	310.772	29.080
311.988	312.709	30.514

What is least squares doing?

- Given 3D point evidence, find best **M** which minimizes error between estimate (**p**’) and known corresponding 2D points (**p**).

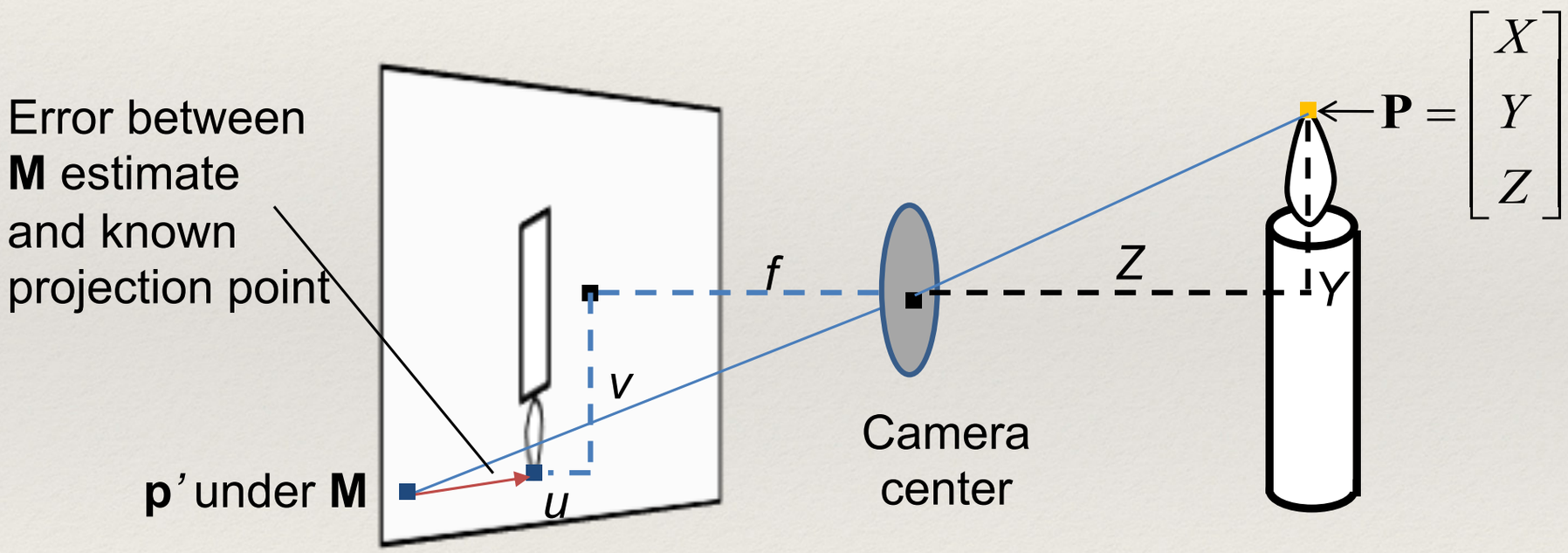


$$\mathbf{p} = \begin{bmatrix} u \\ v \end{bmatrix}$$

p = distance from image center

What is least squares doing?

- Best **M** occurs when **p’** = **p**, or when **p’** – **p** = 0
- Form these equations from all point evidence
- Solve for model via closed-form regression



$$\mathbf{p} = \begin{bmatrix} u \\ v \end{bmatrix}$$

p = distance from image center

Unknown Camera Parameters

Known 2d image coords

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Known 3d locations

First, work out where X,Y,Z projects to under candidate **M.**

$$su = m_{11}X + m_{12}Y + m_{13}Z + m_{14}$$

$$sv = m_{21}X + m_{22}Y + m_{23}Z + m_{24}$$

$$s = m_{31}X + m_{32}Y + m_{33}Z + m_{34}$$

Two equations per 3D point correspondence

$$u = \frac{m_{11}X + m_{12}Y + m_{13}Z + m_{14}}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}}$$

$$v = \frac{m_{21}X + m_{22}Y + m_{23}Z + m_{24}}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}}$$

Unknown Camera Parameters

Known 2d image coords

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Known 3d locations

Next, rearrange into form where all **M** coefficients are individually stated in terms of X,Y,Z,u,v.
-> Allows us to form lsq matrix.

$$u = \frac{m_{11}X + m_{12}Y + m_{13}Z + m_{14}}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}}$$

$$v = \frac{m_{21}X + m_{22}Y + m_{23}Z + m_{24}}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}}$$

$$(m_{31}X + m_{32}Y + m_{33}Z + m_{34})u = m_{11}X + m_{12}Y + m_{13}Z + m_{14}$$

$$(m_{31}X + m_{32}Y + m_{33}Z + m_{34})v = m_{21}X + m_{22}Y + m_{23}Z + m_{24}$$

$$m_{31}uX + m_{32}uY + m_{33}uZ + m_{34}u = m_{11}X + m_{12}Y + m_{13}Z + m_{14}$$

$$m_{31}vX + m_{32}vY + m_{33}vZ + m_{34}v = m_{21}X + m_{22}Y + m_{23}Z + m_{24}$$

Unknown Camera Parameters

Known 2d image coords

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Known 3d locations

Next, rearrange into form where all **M** coefficients are individually stated in terms of X,Y,Z,u,v.
-> Allows us to form lsq matrix.

$$m_{31}uX + m_{32}uY + m_{33}uZ + m_{34}u = m_{11}X + m_{12}Y + m_{13}Z + m_{14}$$
$$m_{31}vX + m_{32}vY + m_{33}vZ + m_{34}v = m_{21}X + m_{22}Y + m_{23}Z + m_{24}$$

$$0 = m_{11}X + m_{12}Y + m_{13}Z + m_{14} - m_{31}uX - m_{32}uY - m_{33}uZ - m_{34}u$$
$$0 = m_{21}X + m_{22}Y + m_{23}Z + m_{24} - m_{31}vX - m_{32}vY - m_{33}vZ - m_{34}v$$

Unknown Camera Parameters

Known 2d image coords

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Known 3d locations

- Finally, solve for m's entries using linear least squares
- Method 1 – **Ax**=**b** form

$$\underbrace{\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -u_1X_1 & -u_1Y_1 & -u_1Z_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1X_1 & -v_1Y_1 & -v_1Z_1 \\ & & & & & & \vdots & & & & \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -u_nX_n & -u_nY_n & -u_nZ_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_nX_n & -v_nY_n & -v_nZ_n \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} u_1 \\ v_1 \\ \vdots \\ u_n \\ v_n \end{bmatrix}}_{\mathbf{b}}$$

MATLAB:

```
M = A\b;
M = [M;1];
M = reshape(M,[],3)';
```

Python Numpy:

```
M = np.linalg.lstsq(A,b)[0];
M = np.append(M,1)
M = np.reshape(M, (3,4))
```

Note: Must reshape M afterwards!

Unknown Camera Parameters

Known 2d image coords

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Known 3d locations

- Or, solve for m's entries using total linear least-squares
- Method 2 – **Ax=0** form
  - Find non-trivial solution (not A=0)

A

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -u_1X_1 & -u_1Y_1 & -u_1Z_1 & -u_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1X_1 & -v_1Y_1 & -v_1Z_1 & -v_1 \\ & & & & & & \vdots & & & & & \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -u_nX_n & -u_nY_n & -u_nZ_n & -u_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_nX_n & -v_nY_n & -v_nZ_n & -v_n \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

x

MATLAB:

```
[U, S, V] = svd(A);
M = V(:,end);
M = reshape(M,[],3)';
```

Python Numpy:

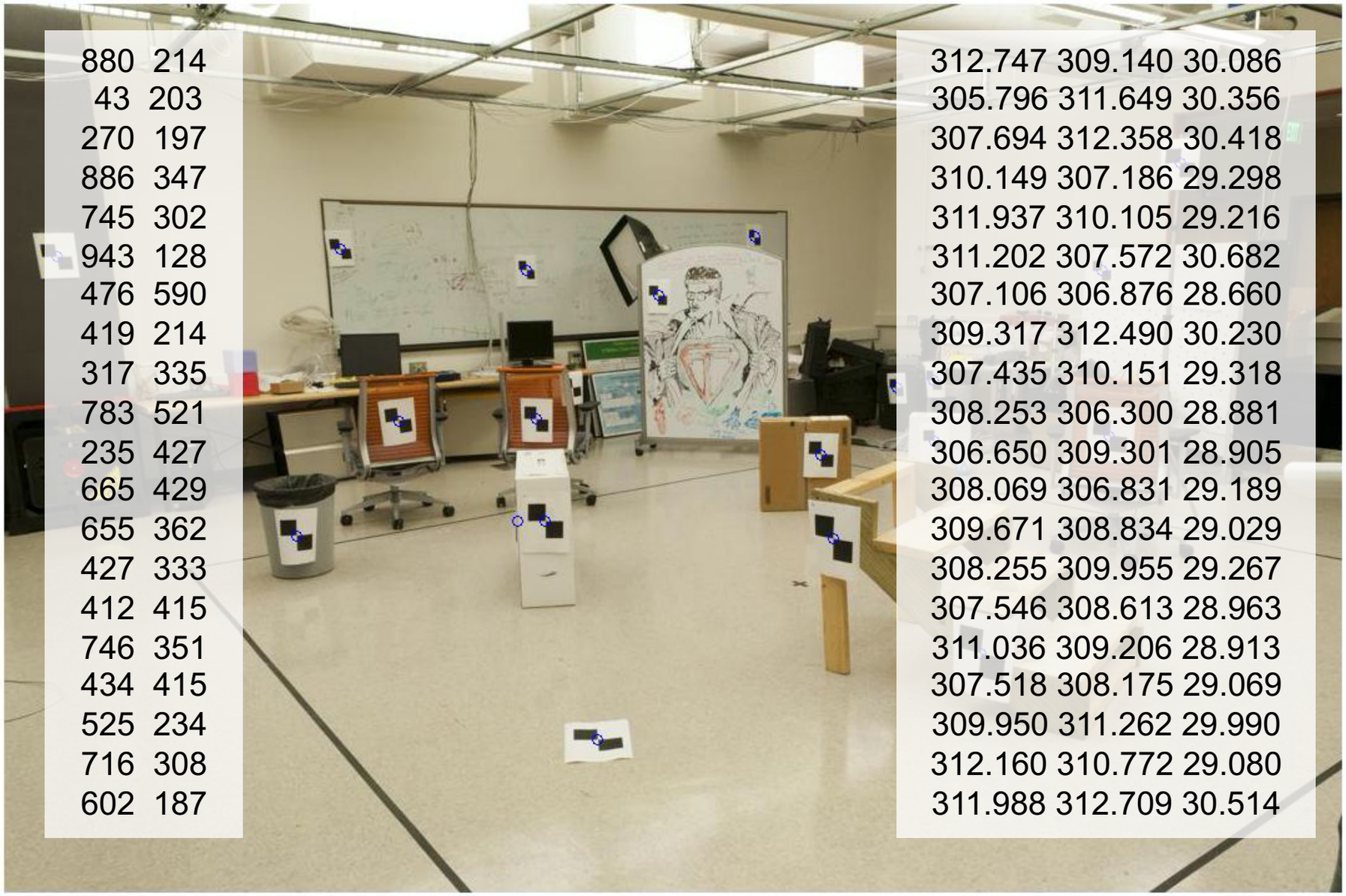
```
U, S, Vh = np.linalg.svd(a)
# V = Vh.T
M = Vh[-1,:]
M = np.reshape(M, (3,4))
```

James Hays

How do we calibrate a camera?

Known 2d image coords

Known 3d world locations



880	214
43	203
270	197
886	347
745	302
943	128
476	590
419	214
317	335
783	521
235	427
665	429
655	362
427	333
412	415
746	351
434	415
525	234
716	308
602	187

312.747	309.140	30.086
305.796	311.649	30.356
307.694	312.358	30.418
310.149	307.186	29.298
311.937	310.105	29.216
311.202	307.572	30.682
307.106	306.876	28.660
309.317	312.490	30.230
307.435	310.151	29.318
308.253	306.300	28.881
306.650	309.301	28.905
308.069	306.831	29.189
309.671	308.834	29.029
308.255	309.955	29.267
307.546	308.613	28.963
311.036	309.206	28.913
307.518	308.175	29.069
309.950	311.262	29.990
312.160	310.772	29.080
311.988	312.709	30.514

Known 2d image coords

Known 3d world locations

1st point

880 214	(u_1, v_1)	312.747 309.140 30.086 (X_1, Y_1, Z_1)
43 203		305.796 311.649 30.356
270 197		307.694 312.358 30.418
886 347		310.149 307.186 29.298
745 302		311.937 310.105 29.216
943 128		311.202 307.572 30.682
476 590		307.106 306.876 28.660
419 214		309.317 312.490 30.230
317 335		307.435 310.151 29.318
…		…..

Projection error defined by two equations – one for *u* and one for *v*

$$\begin{bmatrix} 312.747 & 309.140 & 30.086 & 1 & 0 & 0 & 0 & 0 & -880\times312.747 & -880\times309.140 & -880\times30.086 & -880 \\ 0 & 0 & 0 & 0 & 312.747 & 309.140 & 30.086 & 1 & -214\times312.747 & -214\times309.140 & -214\times30.086 & -214 \\ & & & & & & \vdots & & & & & \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -u_nX_n & -u_nY_n & -u_nZ_n & -u_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_nX_n & -v_nY_n & -v_nZ_n & -v_n \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

Known 2d image coords

Known 3d world locations

2nd point

880 214	312.747 309.140 30.086
43 203 (u_2, v_2)	305.796 311.649 30.356 (X_2, Y_2, Z_2)
270 197	307.694 312.358 30.418
886 347	310.149 307.186 29.298
745 302	311.937 310.105 29.216
943 128	311.202 307.572 30.682
476 590	307.106 306.876 28.660
419 214	309.317 312.490 30.230
317 335	307.435 310.151 29.318
…	…..

Projection error defined by two equations – one for *u* and one for *v*

$$\begin{bmatrix} 312.747 & 309.140 & 30.086 & 1 & 0 & 0 & 0 & 0 & -880\times312.747 & -880\times309.140 & -880\times30.086 & -880 \\ 0 & 0 & 0 & 0 & 312.747 & 309.140 & 30.086 & 1 & -214\times312.747 & -214\times309.140 & -214\times30.086 & -214 \\ 305.796 & 311.649 & 30.356 & 1 & 0 & 0 & 0 & 0 & -43\times305.796 & -43\times311.649 & -43\times30.356 & -43 \\ 0 & 0 & 0 & 0 & 305.796 & 311.649 & 30.356 & 1 & -203\times305.796 & -203\times311.649 & -43\times30.356 & -203 \\ & & & & & & \vdots & & & & & \\ X_n & Y_n & Z_n & 1_n & 0 & 0 & 0 & 0 & -u_nX_n & -u_nY_n & -u_nZ_n & -u_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_nX_n & -v_nY_n & -v_nZ_n & -v_n \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

How many points do I need to fit the model?

$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}\mathbf{X}$$

Degrees of freedom? 5 6

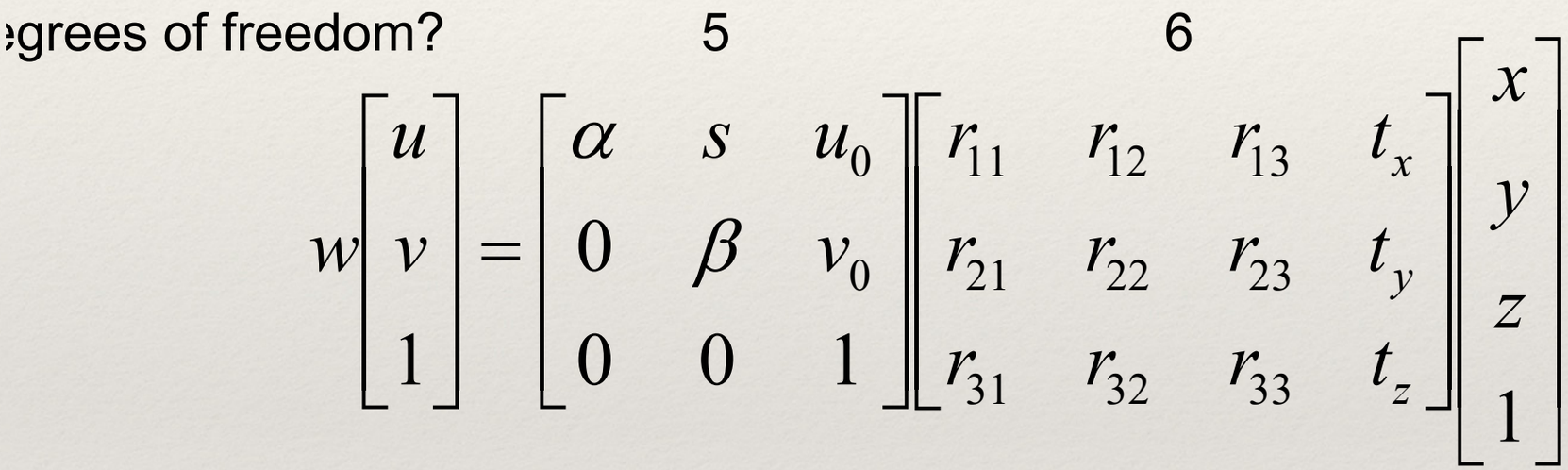
$$w\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & s & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Think 3:

- Rotation around x
- Rotation around y
- Rotation around z

How many points do I need to fit the model?

$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}\mathbf{X}$$



M is 3x4, so 12 unknowns, but projective scale ambiguity – 11 deg. freedom.
One equation per unknown -> 5 1/2 point correspondences determines a solution (e.g., either u or v).

More than 5 1/2 point correspondences -> overdetermined, many solutions to **M**.
Least squares is finding the solution that best satisfies the overdetermined system.

Why use more than 6? Robustness to error in feature points.

Calibration with linear method

- Advantages
  - Easy to formulate and solve
  - Provides initialization for non-linear methods
- Disadvantages
  - Doesn’t directly give you camera parameters
  - Doesn’t model radial distortion
  - Can’t impose constraints, such as known focal length
- Non-linear methods are preferred
  - Define error as difference between projected points and measured points
  - Minimize error using Newton’s method or other non-linear optimization

Can we factorize M back to K [R | T]?

- Yes
- We can directly solve for the individual entries of K [R | T].

$\mathbf{a}_n$ = nth column of A

Extracting camera parameters

$$\frac{M}{\rho} = \begin{pmatrix} \alpha \boldsymbol{r}_1^T - \alpha \cot\theta \boldsymbol{r}_2^T + u_0 \boldsymbol{r}_3^T & \alpha t_x - \alpha \cot\theta t_y + u_0 t_z \\ \frac{\beta}{\sin\theta} \boldsymbol{r}_2^T + v_0 \boldsymbol{r}_3^T & \frac{\beta}{\sin\theta} t_y + v_0 t_z \\ \boldsymbol{r}_3^T & t_z \end{pmatrix} = K[R \quad T]$$

A $\quad$ $\mathbf{b}$

$$K = \begin{bmatrix} \alpha & -\alpha\cot\theta & u_o \\ 0 & \frac{\beta}{\sin\theta} & v_o \\ 0 & 0 & 1 \end{bmatrix}$$

Box 1

$$A = \begin{bmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \mathbf{a}_3^T \end{bmatrix} \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Estimated values

Intrinsic

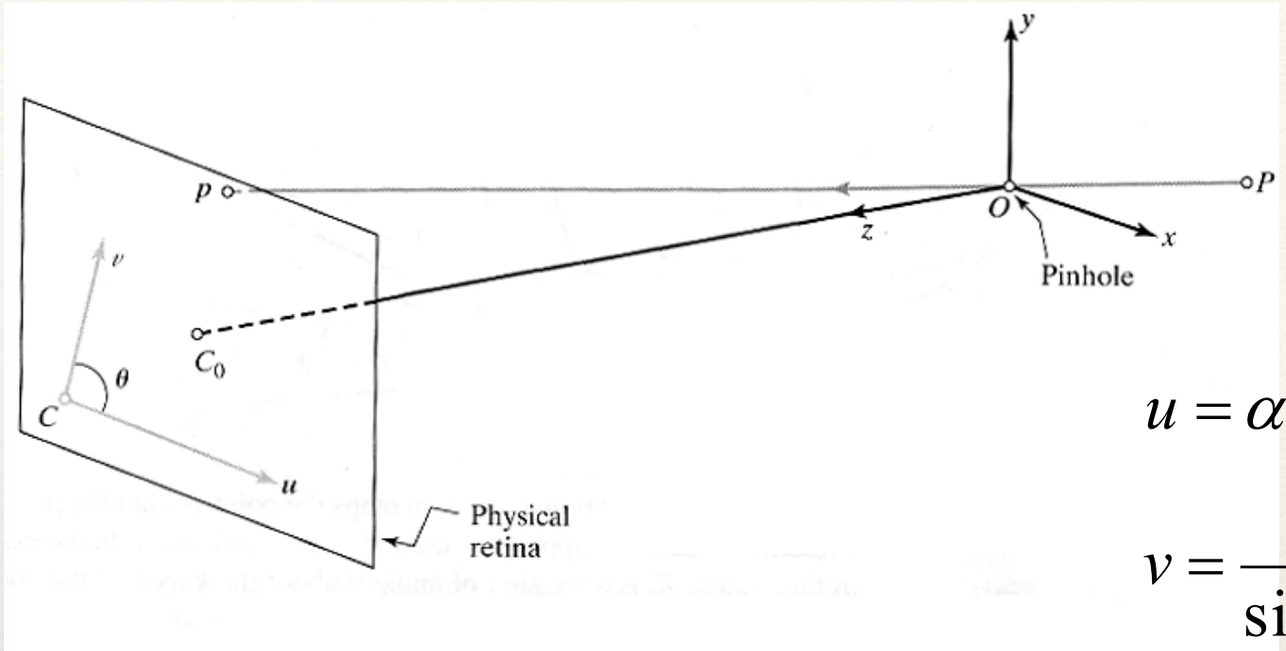
$$\rho = \frac{\pm 1}{|\mathbf{a}_3|} \qquad u_o = \rho^2(\mathbf{a}_1 \cdot \mathbf{a}_3)$$

$$v_o = \rho^2(\mathbf{a}_2 \cdot \mathbf{a}_3)$$

$$\cos\theta = \frac{(\mathbf{a}_1 \times \mathbf{a}_3) \cdot (\mathbf{a}_2 \times \mathbf{a}_3)}{|\mathbf{a}_1 \times \mathbf{a}_3| \cdot |\mathbf{a}_2 \times \mathbf{a}_3|}$$

James Hays

Intrinsic parameters, homogeneous coordinates



$$u = \alpha \frac{x}{z} - \alpha \cot(\theta) \frac{y}{z} + u_0$$

$$v = \frac{\beta}{\sin(\theta)} \frac{y}{z} + v_0$$

Using homogenous coordinates, we can write this as:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha & -\alpha\cot(\theta) & u_0 & 0 \\ 0 & \frac{\beta}{\sin(\theta)} & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

or:

In pixels $\longrightarrow$ $\vec{p} = \mathrm{K}\, {}^{c}\vec{p}$

In camera-based coords

Extracting camera parameters

$$\frac{\mathcal{M}}{\rho} = \begin{pmatrix} \alpha \boldsymbol{r}_1^T - \alpha \cot\theta \boldsymbol{r}_2^T + u_0 \boldsymbol{r}_3^T & \alpha t_x - \alpha \cot\theta t_y + u_0 t_z \\ \frac{\beta}{\sin\theta} \boldsymbol{r}_2^T + v_0 \boldsymbol{r}_3^T & \frac{\beta}{\sin\theta} t_y + v_0 t_z \\ \boldsymbol{r}_3^T & t_z \end{pmatrix} = \mathrm{K}\begin{bmatrix}\mathrm{R} & \mathrm{T}\end{bmatrix}$$

A $\quad$ $\mathbf{b}$

$$\mathrm{A} = \begin{bmatrix} \mathbf{a}_1^{\mathrm{T}} \\ \mathbf{a}_2^{\mathrm{T}} \\ \mathbf{a}_3^{\mathrm{T}} \end{bmatrix} \qquad \mathrm{b} = \begin{bmatrix} \mathrm{b}_1 \\ \mathrm{b}_2 \\ \mathrm{b}_3 \end{bmatrix}$$

Estimated values

Intrinsic

$$\alpha = \rho^2 |\mathbf{a}_1 \times \mathbf{a}_3| \sin\theta$$

$$\beta = \rho^2 |\mathbf{a}_2 \times \mathbf{a}_3| \sin\theta$$

Extracting camera parameters

$$\frac{\mathcal{M}}{\rho} = \begin{pmatrix} \alpha \boldsymbol{r}_1^T - \alpha \cot\theta \boldsymbol{r}_2^T + u_0 \boldsymbol{r}_3^T & \alpha t_x - \alpha \cot\theta t_y + u_0 t_z \\ \frac{\beta}{\sin\theta} \boldsymbol{r}_2^T + v_0 \boldsymbol{r}_3^T & \frac{\beta}{\sin\theta} t_y + v_0 t_z \\ \boldsymbol{r}_3^T & t_z \end{pmatrix} = \mathrm{K}\begin{bmatrix}\mathrm{R} & \mathrm{T}\end{bmatrix}$$

A $\quad$ $\mathbf{b}$

$$\mathrm{A} = \begin{bmatrix} \mathbf{a}_1^{\mathrm{T}} \\ \mathbf{a}_2^{\mathrm{T}} \\ \mathbf{a}_3^{\mathrm{T}} \end{bmatrix} \qquad \mathrm{b} = \begin{bmatrix} \mathrm{b}_1 \\ \mathrm{b}_2 \\ \mathrm{b}_3 \end{bmatrix}$$

Estimated values

Extrinsic

$$\mathbf{r}_1 = \frac{(\mathbf{a}_2 \times \mathbf{a}_3)}{|\mathbf{a}_2 \times \mathbf{a}_3|} \qquad \mathbf{r}_3 = \frac{\pm \mathbf{a}_3}{|\mathbf{a}_3|}$$

$$\mathbf{r}_2 = \mathbf{r}_3 \times \mathbf{r}_1 \qquad \mathrm{T} = \rho\, \mathrm{K}^{-1}\mathbf{b}$$

James Hays

Can we factorize M back to K [R | T]?

- Yes
- We can also use *RQ* factorization (not QR)
  - R in RQ is not rotation matrix R; crossed names!
  - *R* (right diagonal) is K
  - *Q* (orthogonal basis) is R.
  - T, the last column of [R | T], is inv(K) * last column of M.
  - But you need to do a bit of post-processing to make sure that the matrices are valid. See http://ksimek.github.io/2012/08/14/decompose/

Recovering the camera center

$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}\mathbf{X}$$

$$w\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & s & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

t

This is not the camera center C.
It is –RC, as the point is rotated before t_x, t_y, and t_z are added

So we need
$-R^{-1} K^{-1} m_4$ to get C.

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix}\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

m_4

Q

This is **t** × K

So $K^{-1} m_4$ is **t**

Q is K × R.
So we just need $-Q^{-1} m_4$