



# From last lecture

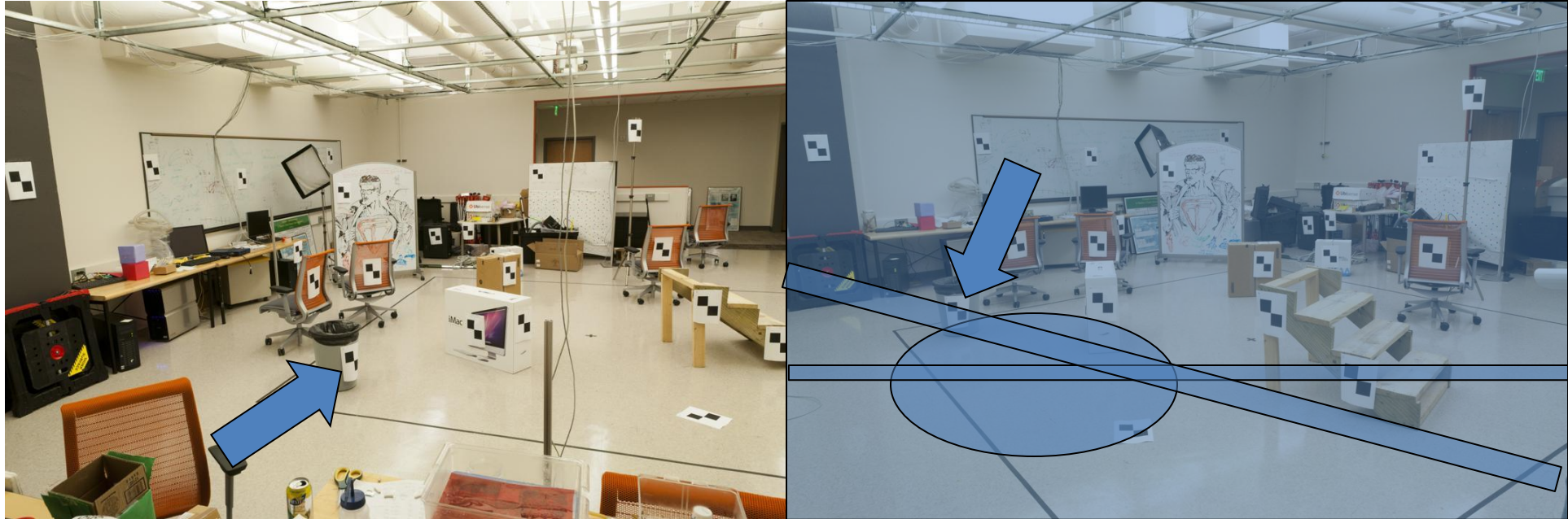
- Stereo and lidar can fall victim to mirrors?
- Yes, there's no easy way around that
- <https://youtu.be/pBzU8TD1iks>



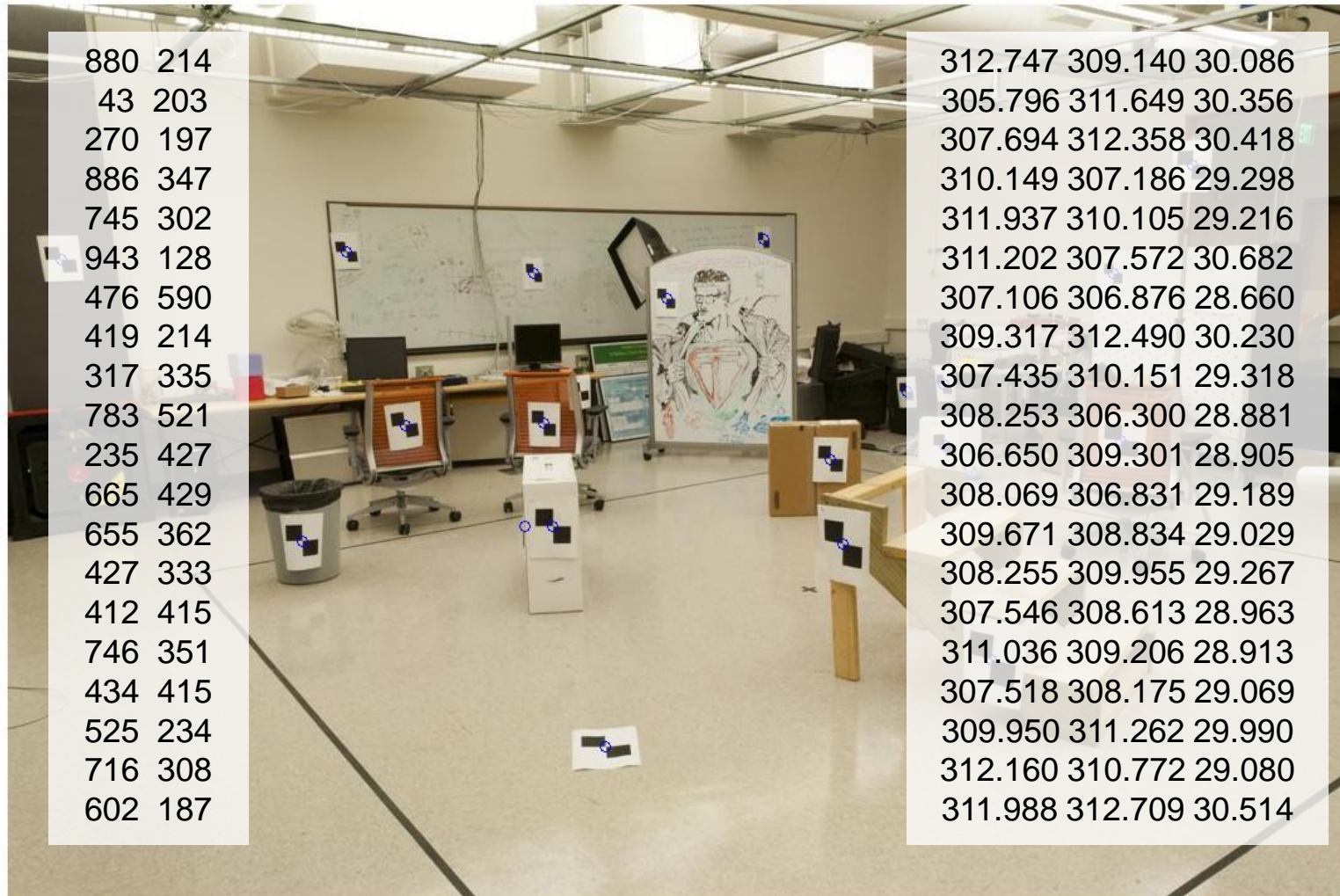
# Outline

- Camera calibration
- Epipolar Geometry

# Where do we need to search?



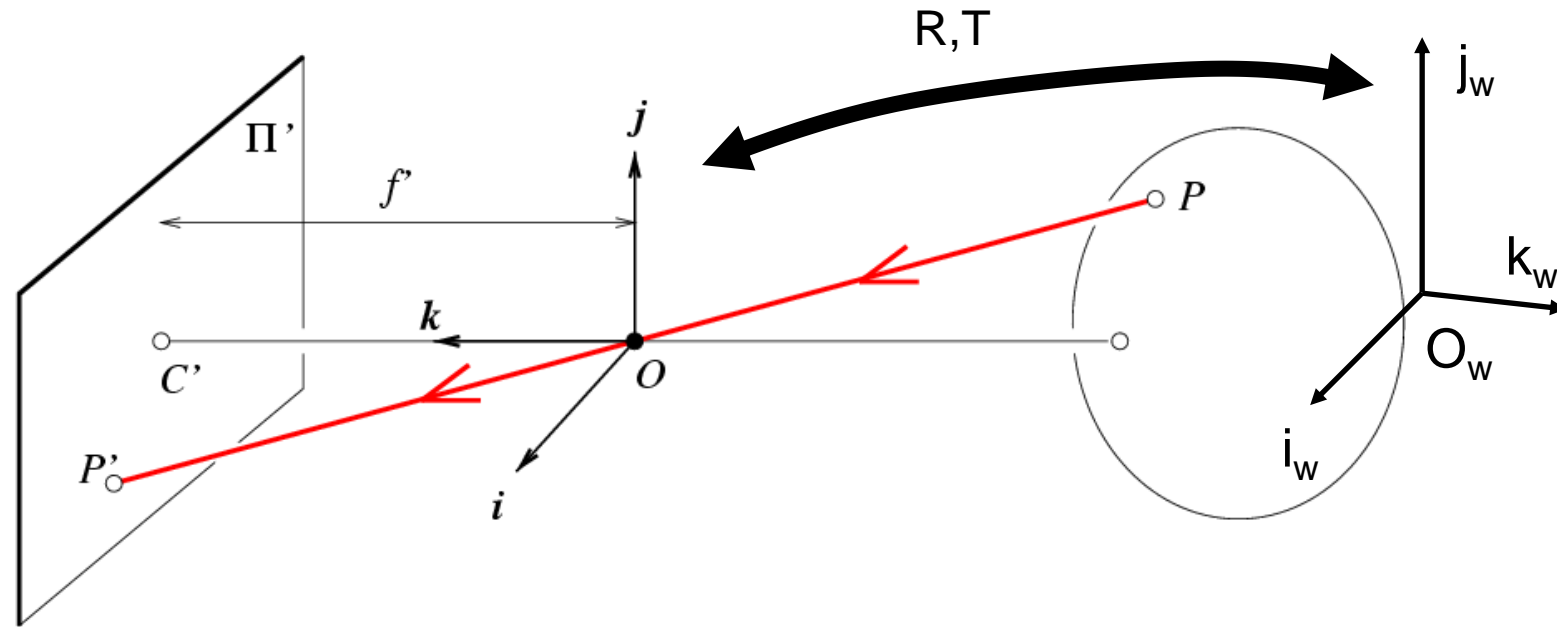
# How do we calibrate a camera?



# World vs Camera coordinates



# Projection matrix



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

$\mathbf{x}$ : Image Coordinates:  $(u, v, 1)$

$\mathbf{K}$ : Intrinsic Matrix  $(3 \times 3)$

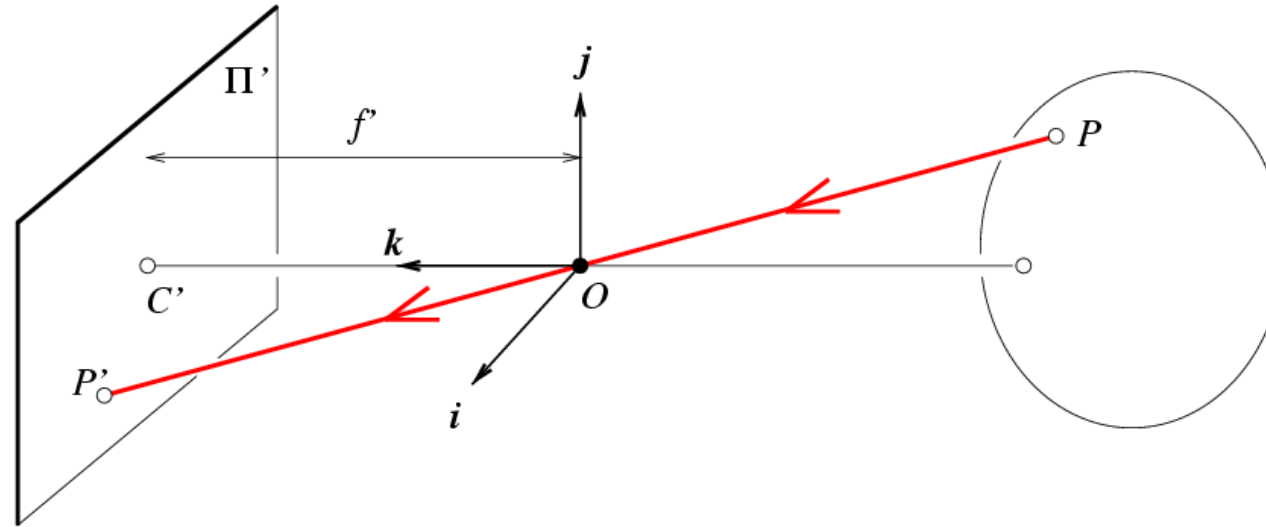
$\mathbf{R}$ : Rotation  $(3 \times 3)$

$\mathbf{t}$ : Translation  $(3 \times 1)$

$\mathbf{X}$ : World Coordinates:  $(X, Y, Z, 1)$



# Projection matrix



## Intrinsic Assumptions

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

## Extrinsic Assumptions

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

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

**K**

# Remove assumption: known optical center

## Intrinsic Assumptions

- Unit aspect ratio
- No skew

## Extrinsic Assumptions

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

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

# Remove assumption: square pixels

Intrinsic Assumptions

- No skew

Extrinsic Assumptions

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

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

# Remove assumption: non-skewed pixels

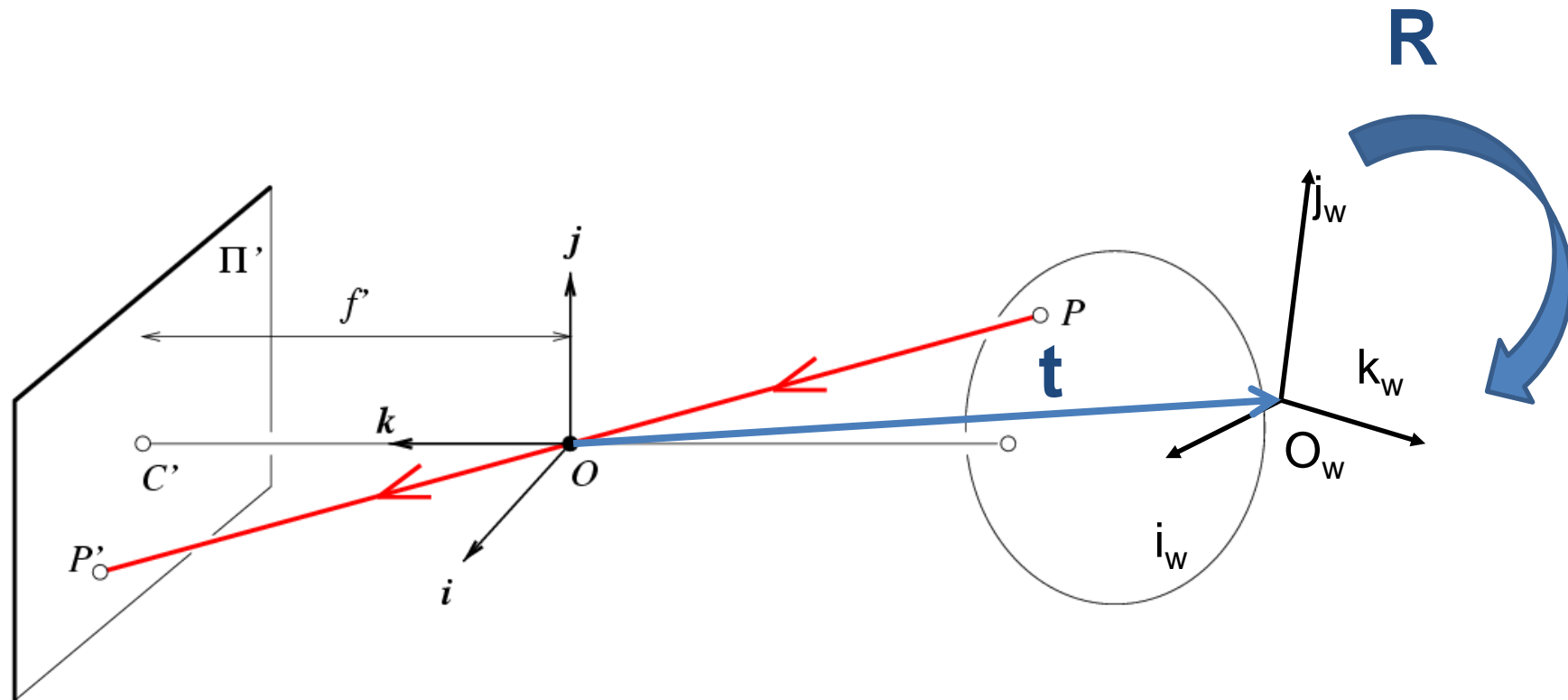
Intrinsic Assumptions    Extrinsic Assumptions

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

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

Note: different books use different notation for parameters

# Oriented and Translated Camera



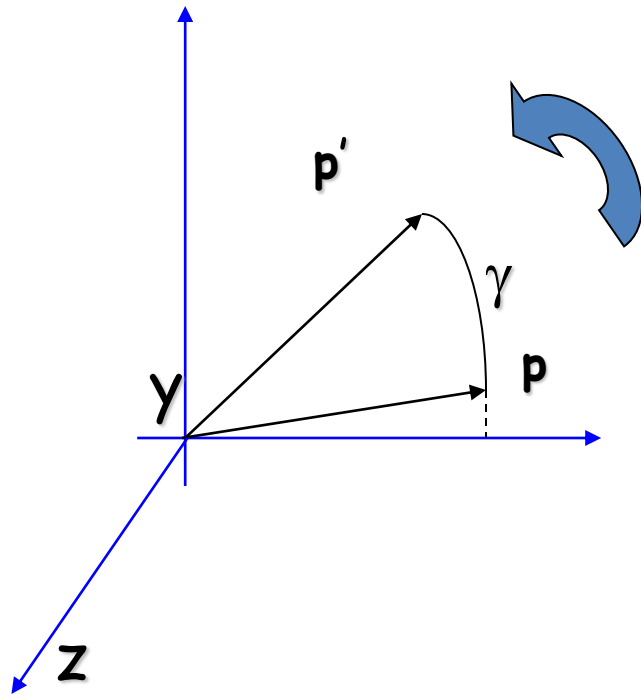
# Allow camera translation

Intrinsic Assumptions    Extrinsic Assumptions  
• No rotation

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

# 3D Rotation of Points

Rotation around the coordinate axes, **counter-clockwise**:



$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix}$$

$$R_y(\beta) = \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix}$$

$$R_z(\gamma) = \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

# Allow camera rotation

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



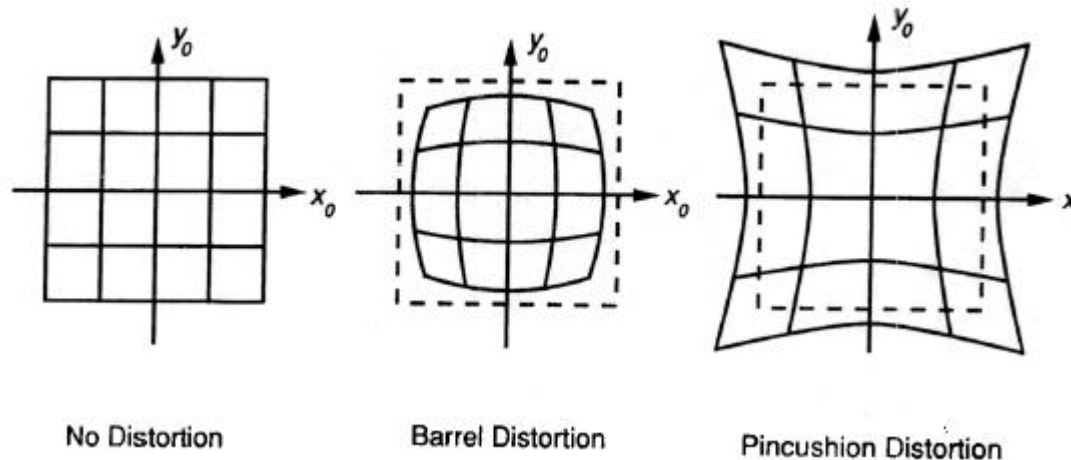
$$w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & s & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} t_x \\ t_y \\ t_z \\ 1 \end{bmatrix} \begin{bmatrix}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





# Beyond Pinholes: Radial Distortion

- Common in wide-angle lenses or for special applications (e.g., security)
- Creates non-linear terms in projection
- Usually handled by through solving for non-linear terms and then correcting image



Corrected Barrel Distortion

How to calibrate the camera?  
(also called “camera resectioning”)

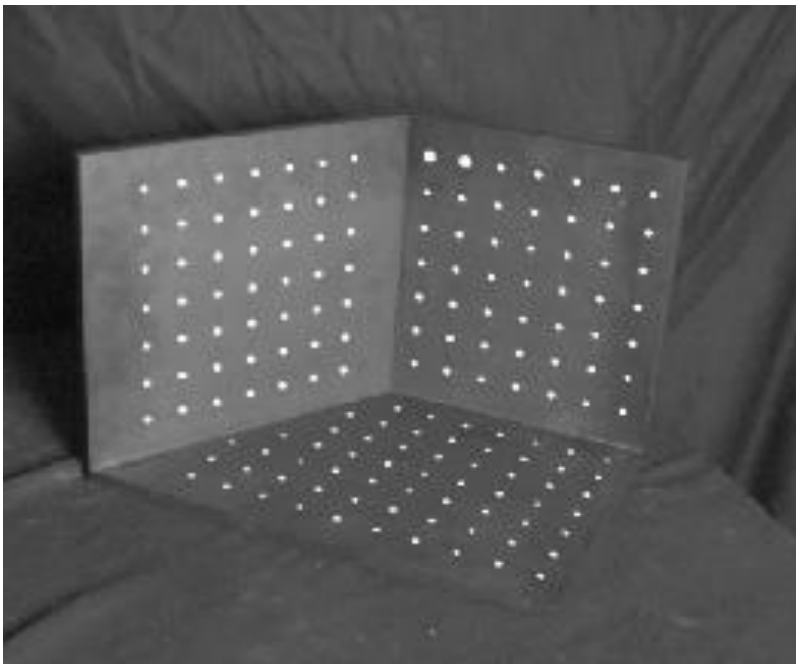
$$\mathbf{x} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \mathbf{X}$$

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

# 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



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

Known 3d  
locations

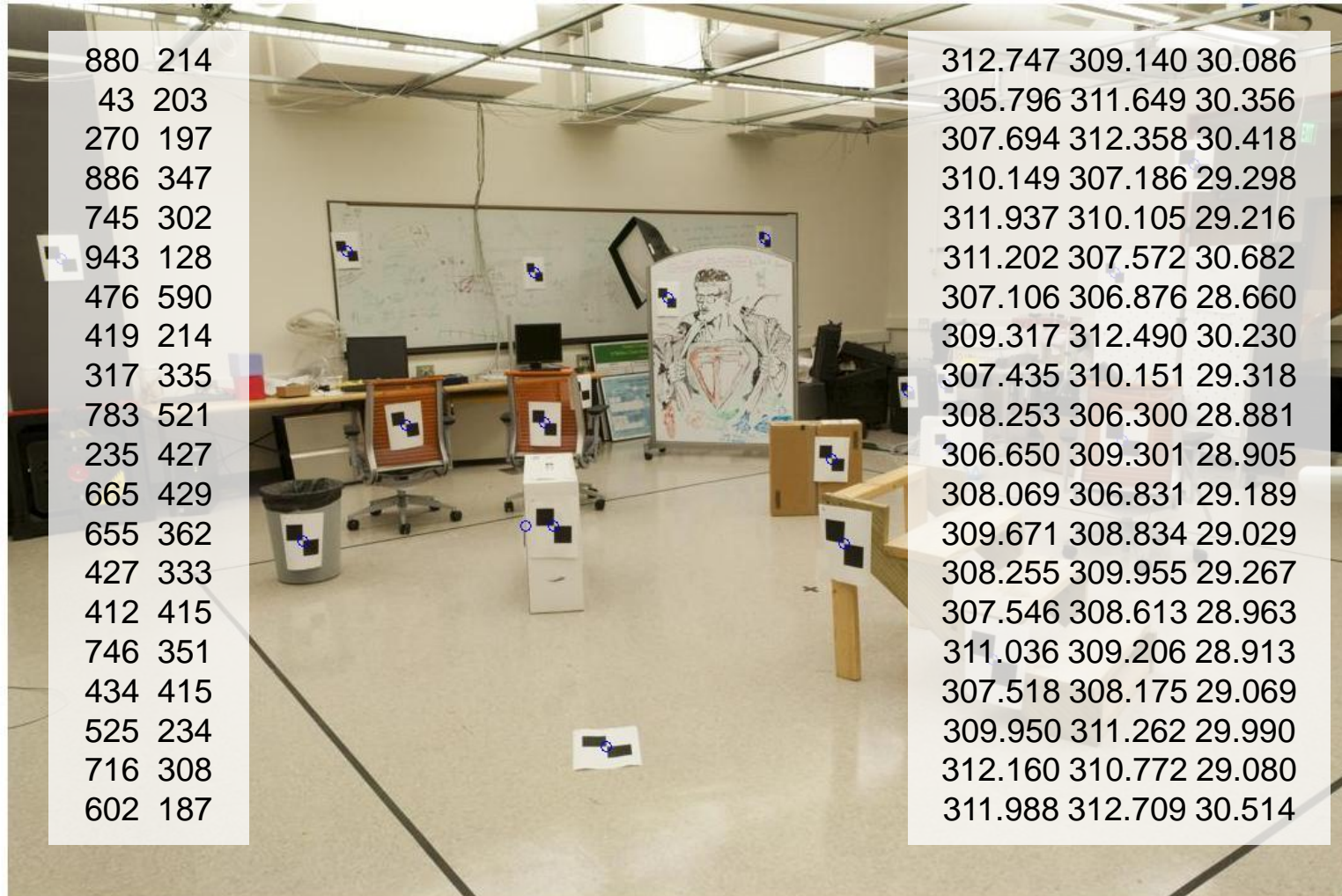


Unknown Camera Parameters

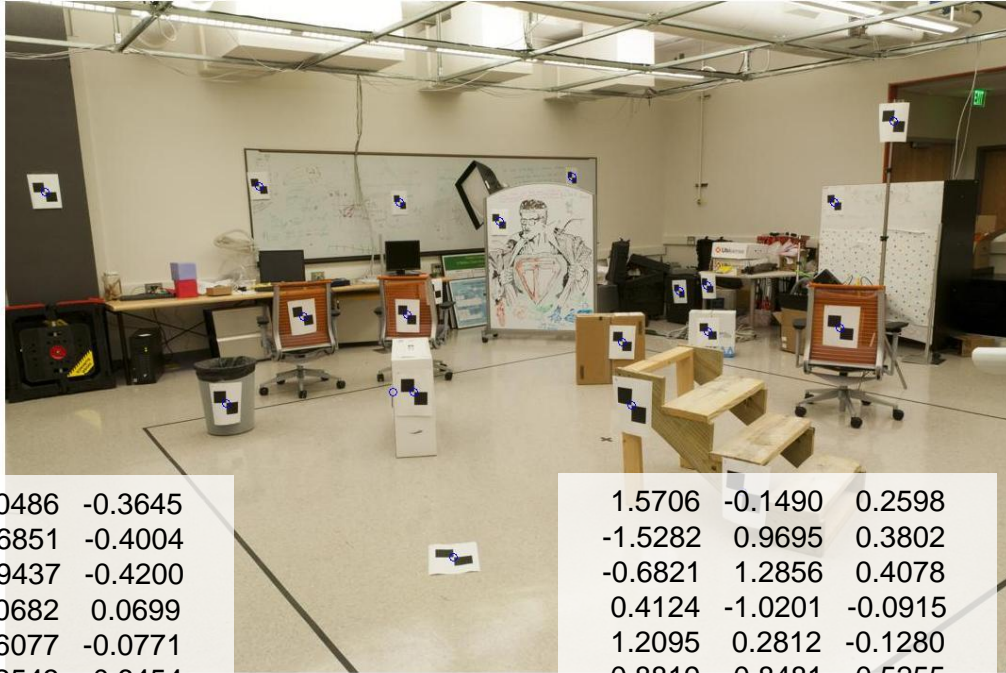
# How do we calibrate a camera?

Known 2d  
image coords

Known 3d  
locations

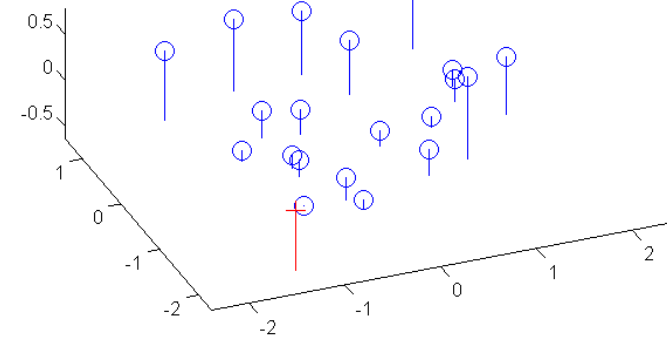


# Estimate of camera center



1.0486	-0.3645
-1.6851	-0.4004
-0.9437	-0.4200
1.0682	0.0699
0.6077	-0.0771
1.2543	-0.6454
-0.2709	0.8635
-0.4571	-0.3645
-0.7902	0.0307
0.7318	0.6382
-1.0580	0.3312
0.3464	0.3377
0.3137	0.1189
-0.4310	0.0242
-0.4799	0.2920
0.6109	0.0830
-0.4081	0.2920
-0.1109	-0.2992
0.5129	-0.0575
0.1406	-0.4527

1.5706	-0.1490	0.2598
-1.5282	0.9695	0.3802
-0.6821	1.2856	0.4078
0.4124	-1.0201	-0.0915
1.2095	0.2812	-0.1280
0.8819	-0.8481	0.5255
-0.9442	-1.1583	-0.3759
0.0415	1.3445	0.3240
-0.7975	0.3017	-0.0826
-0.4329	-1.4151	-0.2774
-1.1475	-0.0772	-0.2667
-0.5149	-1.1784	-0.1401
0.1993	-0.2854	-0.2114
-0.4320	0.2143	-0.1053
-0.7481	-0.3840	-0.2408
0.8078	-0.1196	-0.2631
-0.7605	-0.5792	-0.1936
0.3237	0.7970	0.2170
1.3089	0.5786	-0.1887
1.2323	1.4421	0.4506



# 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

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

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

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

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

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

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

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

# 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

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

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

$$0 = m_{11}X + m_{12}Y + m_{13}Z + m_{14} - m_{31}uX - m_{32}uY - m_{33}uZ - m_{34}u$$

$$0 = m_{21}X + m_{22}Y + m_{23}Z + m_{24} - m_{31}vX - m_{32}vY - m_{33}vZ - m_{34}v$$



# 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

$$0 = m_{11}X + m_{12}Y + m_{13}Z + m_{14} - m_{31}uX - m_{32}uY - m_{33}uZ - m_{34}u$$

$$0 = m_{21}X + m_{22}Y + m_{23}Z + m_{24} - m_{31}vX - m_{32}vY - m_{33}vZ - m_{34}v$$

- Method 1 – homogeneous linear system. Solve for m's entries using linear least squares

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

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

**For python, see  
numpy.linalg.svd**

# 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

- Method 2 – nonhomogeneous linear system. Solve for m's entries using linear least squares

**Ax=b** form

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -u_1 X_1 & -u_1 Y_1 & -u_1 Z_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1 X_1 & -v_1 Y_1 & -v_1 Z_1 \\ & & & & & & \vdots & & & & \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -u_n X_n & -u_n Y_n & -u_n Z_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_n X_n & -v_n Y_n & -v_n Z_n \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \end{bmatrix} = \begin{bmatrix} u_1 \\ v_1 \\ \vdots \\ u_n \\ v_n \end{bmatrix}$$

$$M = A \setminus Y;$$

$$M = [M; 1];$$

$$M = \text{reshape}(M, [], 3)';$$

**For python, see  
numpy.linalg.lstsq**

# 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 $K [R \mid T]$ ?

- Yes!
- You can use  $RQ$  factorization (note – not the more familiar  $QR$  factorization).  $R$  (right diagonal) is  $K$ , and  $Q$  (orthogonal basis) is  $R$ .  $T$ , the last column of  $[R \mid T]$ , is  $\text{inv}(K) * \text{last column of } M$ .
  - 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/>

# Can we factorize $M$ back to $K [R \mid T]$ ?

- Yes!
- Alternatively, you can more directly solve for the individual entries of  $K [R \mid T]$ .

# Extracting camera parameters

$$\frac{M}{\rho} = \left( \begin{array}{c|c} \alpha \mathbf{r}_1^T - \alpha \cot \theta \mathbf{r}_2^T + u_0 \mathbf{r}_3^T & \alpha t_x - \alpha \cot \theta t_y + u_0 t_z \\ \hline \frac{\beta}{\sin \theta} \mathbf{r}_2^T + v_0 \mathbf{r}_3^T & \frac{\beta}{\sin \theta} t_y + v_0 t_z \\ \mathbf{r}_3^T & t_z \end{array} \right) = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{T} \end{bmatrix}$$

$$\mathbf{K} = \begin{bmatrix} \alpha & -\alpha \cot \theta & u_0 \\ 0 & \frac{\beta}{\sin \theta} & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

$A$   $\mathbf{b}$

Box 1

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

Estimated values

## Intrinsic

$$\rho = \frac{\pm 1}{|\mathbf{a}_3|} \quad \begin{array}{l} u_0 = \rho^2 (\mathbf{a}_1 \cdot \mathbf{a}_3) \\ v_0 = \rho^2 (\mathbf{a}_2 \cdot \mathbf{a}_3) \end{array}$$

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

# Extracting camera parameters

$$\frac{\mathcal{M}}{\rho} = \left( \begin{array}{c|c} \alpha \mathbf{r}_1^T - \alpha \cot \theta \mathbf{r}_2^T + u_0 \mathbf{r}_3^T & \alpha t_x - \alpha \cot \theta t_y + u_0 t_z \\ \frac{\beta}{\sin \theta} \mathbf{r}_2^T + v_0 \mathbf{r}_3^T & \frac{\beta}{\sin \theta} t_y + v_0 t_z \\ \mathbf{r}_3^T & t_z \end{array} \right) = \mathbf{K} \begin{bmatrix} \mathbf{R} \\ \mathbf{T} \end{bmatrix}$$

**A** **b**

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

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

$$\frac{\mathcal{M}}{\rho} = \left( \begin{array}{c|c} \alpha \mathbf{r}_1^T - \alpha \cot \theta \mathbf{r}_2^T + u_0 \mathbf{r}_3^T & \alpha t_x - \alpha \cot \theta t_y + u_0 t_z \\ \hline \frac{\beta}{\sin \theta} \mathbf{r}_2^T + v_0 \mathbf{r}_3^T & \frac{\beta}{\sin \theta} t_y + v_0 t_z \\ \mathbf{r}_3^T & t_z \end{array} \right) = \mathbf{K} \begin{bmatrix} \mathbf{R} \\ \mathbf{T} \end{bmatrix}$$

**A** **b**

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

Estimated values

## Extrinsic

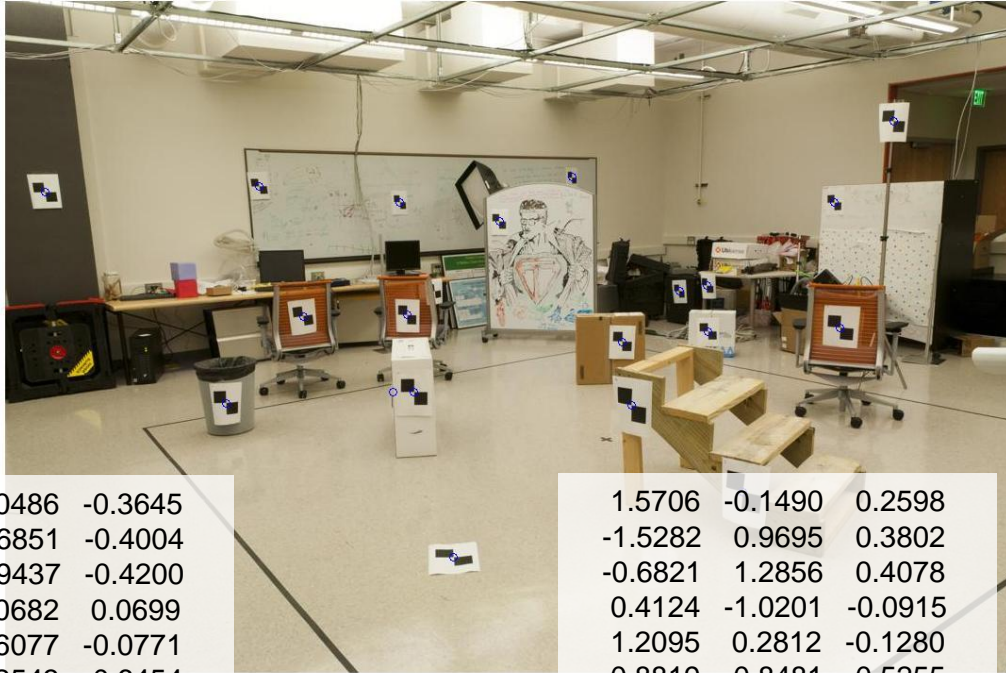
$$\mathbf{r}_1 = \frac{(\mathbf{a}_2 \times \mathbf{a}_3)}{|\mathbf{a}_2 \times \mathbf{a}_3|} \quad \mathbf{r}_3 = \frac{\pm \mathbf{a}_3}{|\mathbf{a}_3|}$$

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



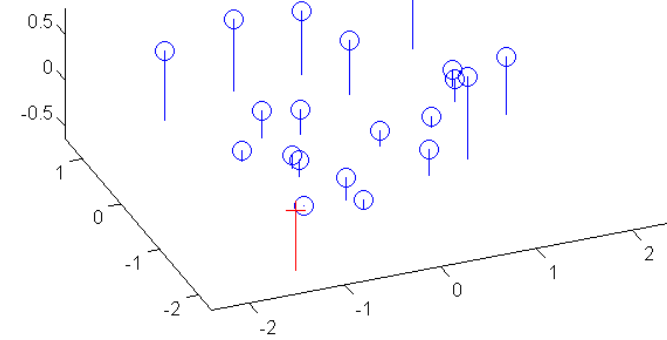
For project 3, we want the camera center

# Estimate of camera center

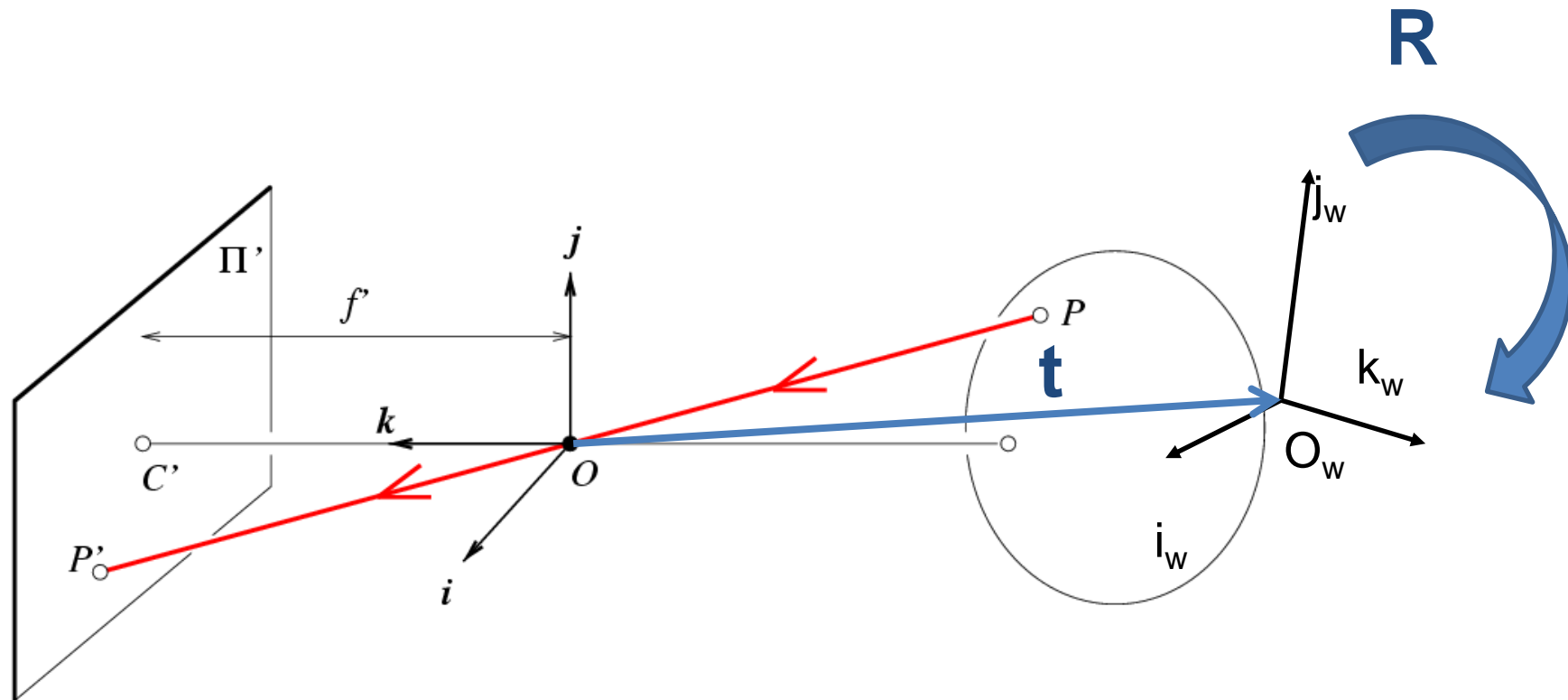


1.0486	-0.3645
-1.6851	-0.4004
-0.9437	-0.4200
1.0682	0.0699
0.6077	-0.0771
1.2543	-0.6454
-0.2709	0.8635
-0.4571	-0.3645
-0.7902	0.0307
0.7318	0.6382
-1.0580	0.3312
0.3464	0.3377
0.3137	0.1189
-0.4310	0.0242
-0.4799	0.2920
0.6109	0.0830
-0.4081	0.2920
-0.1109	-0.2992
0.5129	-0.0575
0.1406	-0.4527

1.5706	-0.1490	0.2598
-1.5282	0.9695	0.3802
-0.6821	1.2856	0.4078
0.4124	-1.0201	-0.0915
1.2095	0.2812	-0.1280
0.8819	-0.8481	0.5255
-0.9442	-1.1583	-0.3759
0.0415	1.3445	0.3240
-0.7975	0.3017	-0.0826
-0.4329	-1.4151	-0.2774
-1.1475	-0.0772	-0.2667
-0.5149	-1.1784	-0.1401
0.1993	-0.2854	-0.2114
-0.4320	0.2143	-0.1053
-0.7481	-0.3840	-0.2408
0.8078	-0.1196	-0.2631
-0.7605	-0.5792	-0.1936
0.3237	0.7970	0.2170
1.3089	0.5786	-0.1887
1.2323	1.4421	0.4506



# Oriented and Translated Camera



# Recovering the camera center

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

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

This is not the camera center  $-C$ . It is  $-RC$  (because a point will be rotated before  $t_x$ ,  $t_y$ , and  $t_z$  are added)

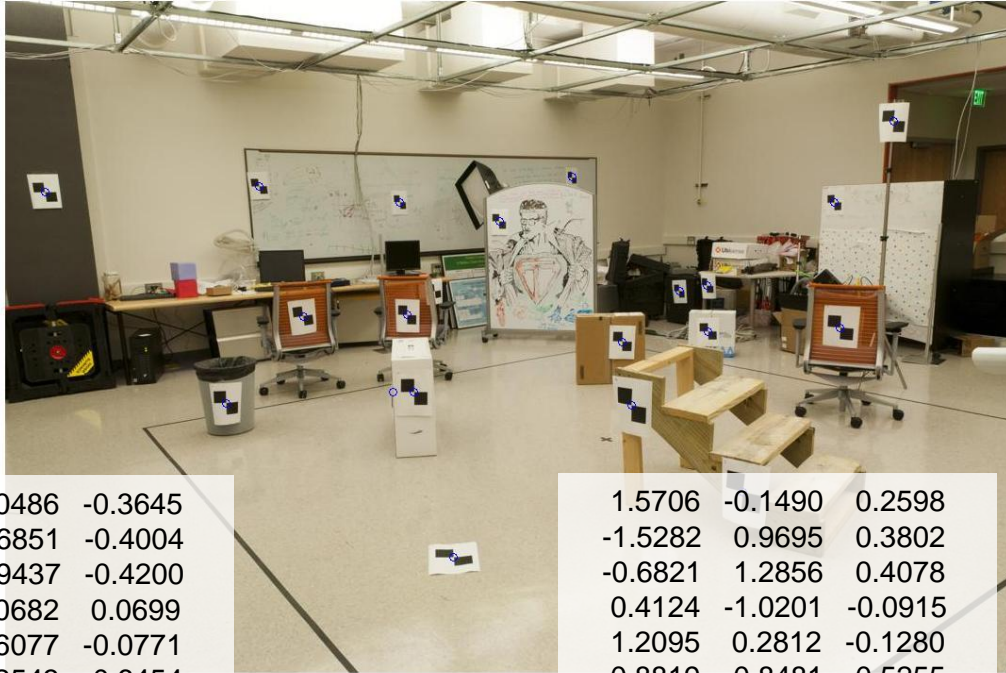
This,  $m_4$ , is  $K * t$   
So  $K^{-1} m_4$  is  $t$

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

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

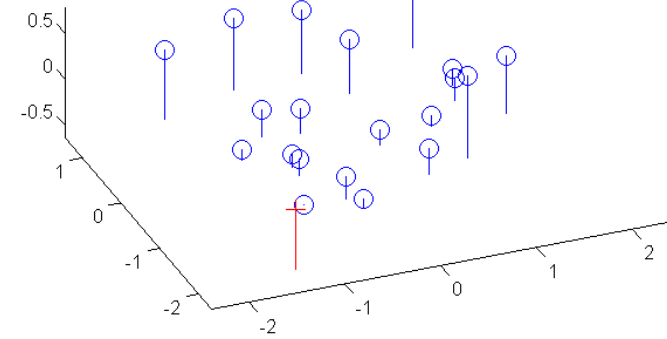
$Q$  is  $K * R$ . So we just need  $-Q^{-1} m_4$

# Estimate of camera center



1.0486	-0.3645
-1.6851	-0.4004
-0.9437	-0.4200
1.0682	0.0699
0.6077	-0.0771
1.2543	-0.6454
-0.2709	0.8635
-0.4571	-0.3645
-0.7902	0.0307
0.7318	0.6382
-1.0580	0.3312
0.3464	0.3377
0.3137	0.1189
-0.4310	0.0242
-0.4799	0.2920
0.6109	0.0830
-0.4081	0.2920
-0.1109	-0.2992
0.5129	-0.0575
0.1406	-0.4527

1.5706	-0.1490	0.2598
-1.5282	0.9695	0.3802
-0.6821	1.2856	0.4078
0.4124	-1.0201	-0.0915
1.2095	0.2812	-0.1280
0.8819	-0.8481	0.5255
-0.9442	-1.1583	-0.3759
0.0415	1.3445	0.3240
-0.7975	0.3017	-0.0826
-0.4329	-1.4151	-0.2774
-1.1475	-0.0772	-0.2667
-0.5149	-1.1784	-0.1401
0.1993	-0.2854	-0.2114
-0.4320	0.2143	-0.1053
-0.7481	-0.3840	-0.2408
0.8078	-0.1196	-0.2631
-0.7605	-0.5792	-0.1936
0.3237	0.7970	0.2170
1.3089	0.5786	-0.1887
1.2323	1.4421	0.4506



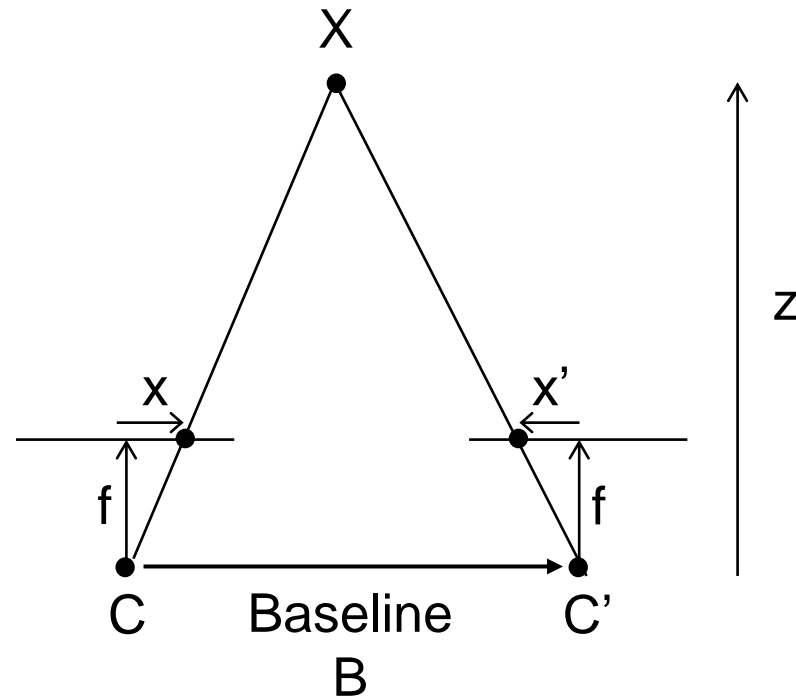
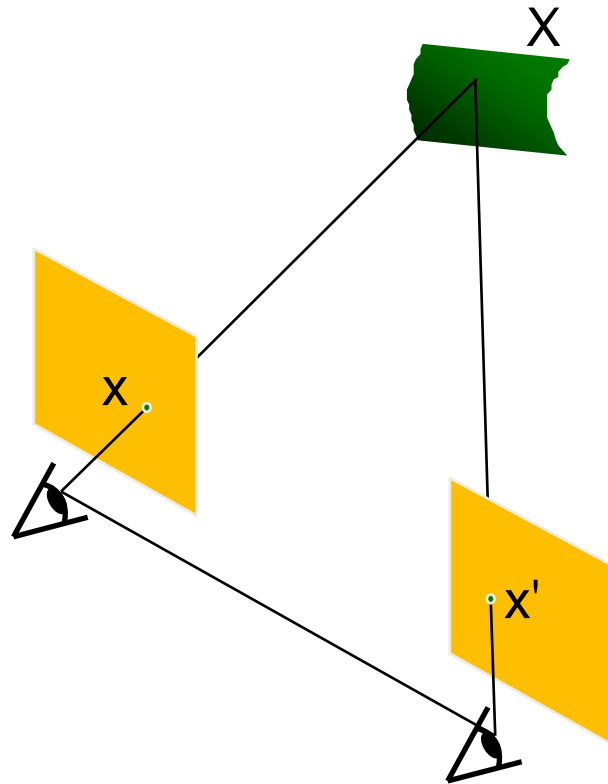
# Epipolar Geometry and Stereo Vision

Chapter 11.2 in Szeliski

- Epipolar geometry
  - Relates cameras from two positions

# Depth from Stereo

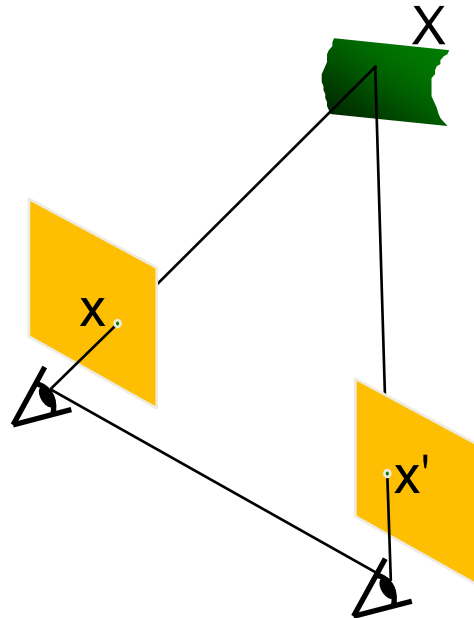
- Goal: recover depth by finding image coordinate  $x'$  that corresponds to  $x$





# Depth from Stereo

- Goal: recover depth by finding image coordinate  $x'$  that corresponds to  $x$
- Sub-Problems
  1. Calibration: How do we recover the relation of the cameras (if not already known)?
  2. Correspondence: How do we search for the matching point  $x'$ ?

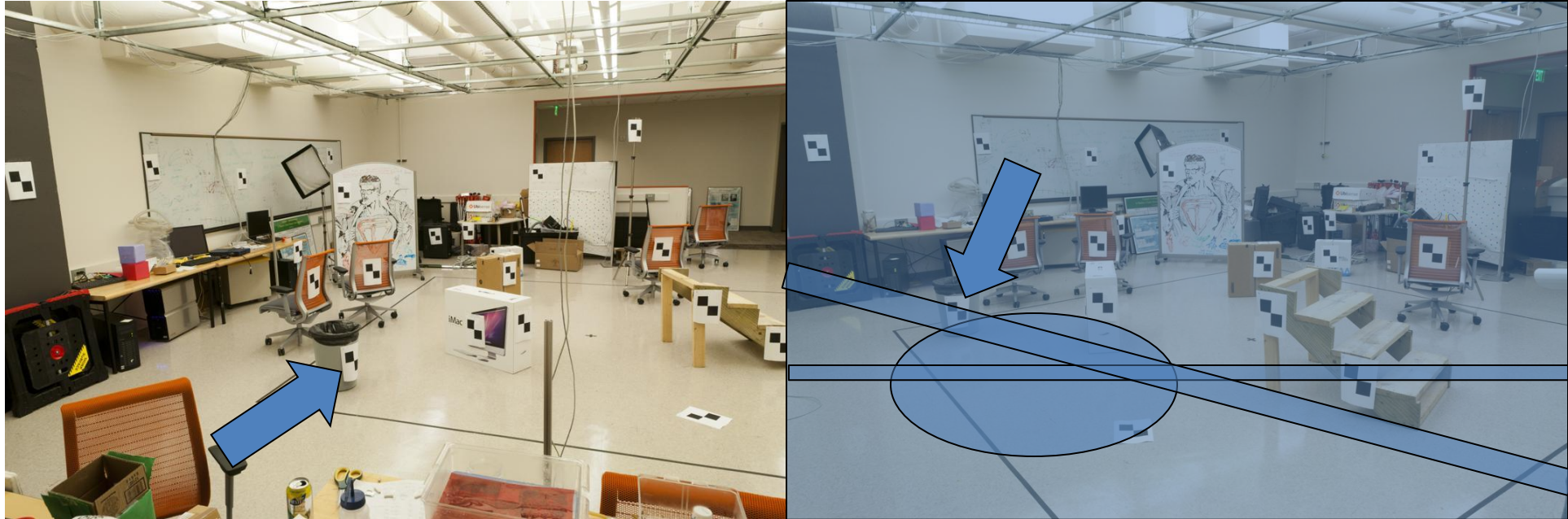


# Correspondence Problem



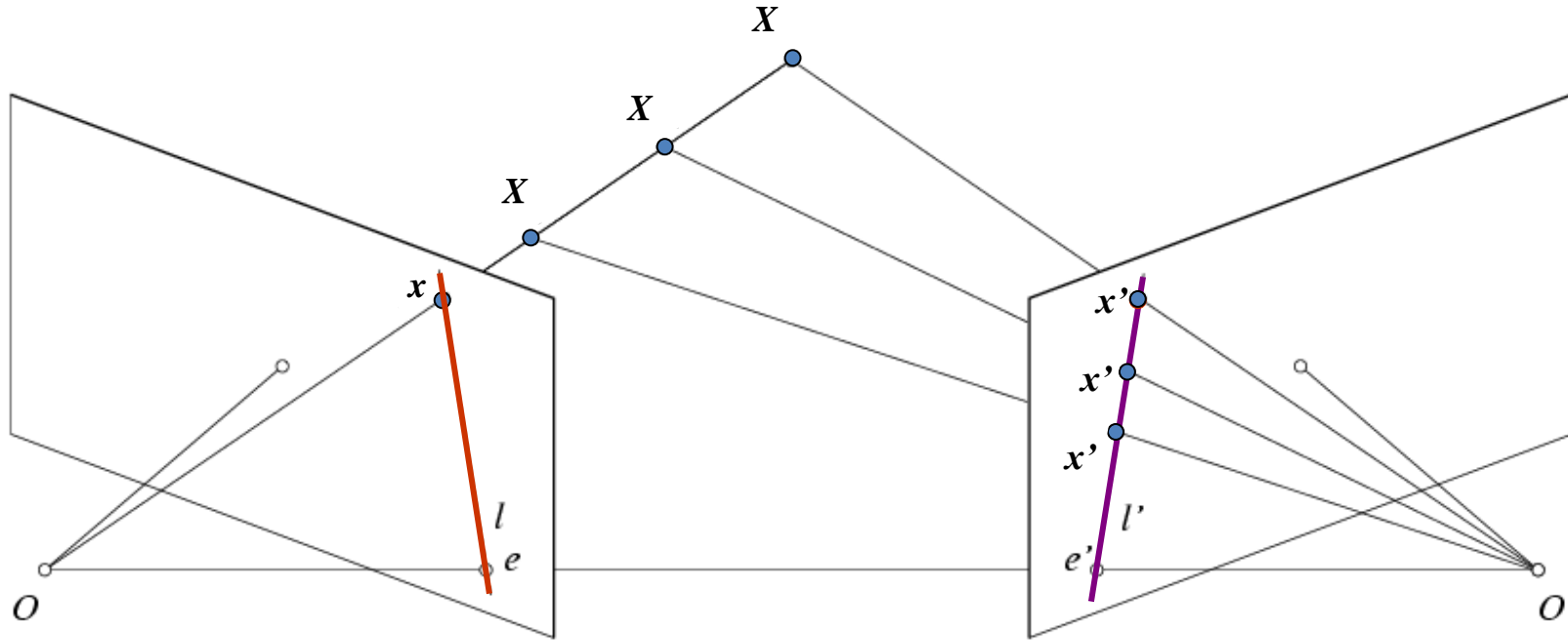
- We have two images taken from cameras with different intrinsic and extrinsic parameters
- How do we match a point in the first image to a point in the second? How can we constrain our search?

# Where do we need to search?



Key idea: Epipolar constraint

# Key idea: Epipolar constraint

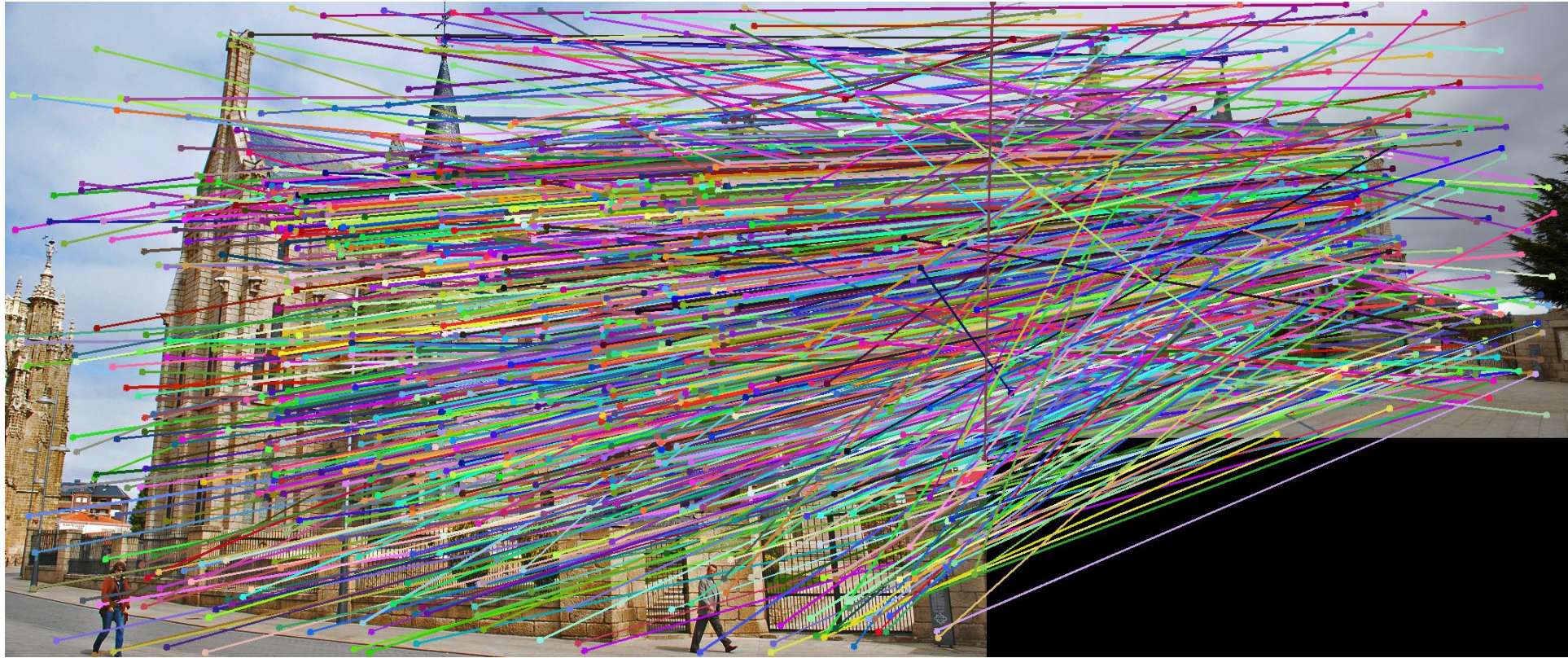


Potential matches for  $x$  have to lie on the corresponding line  $l'$ .

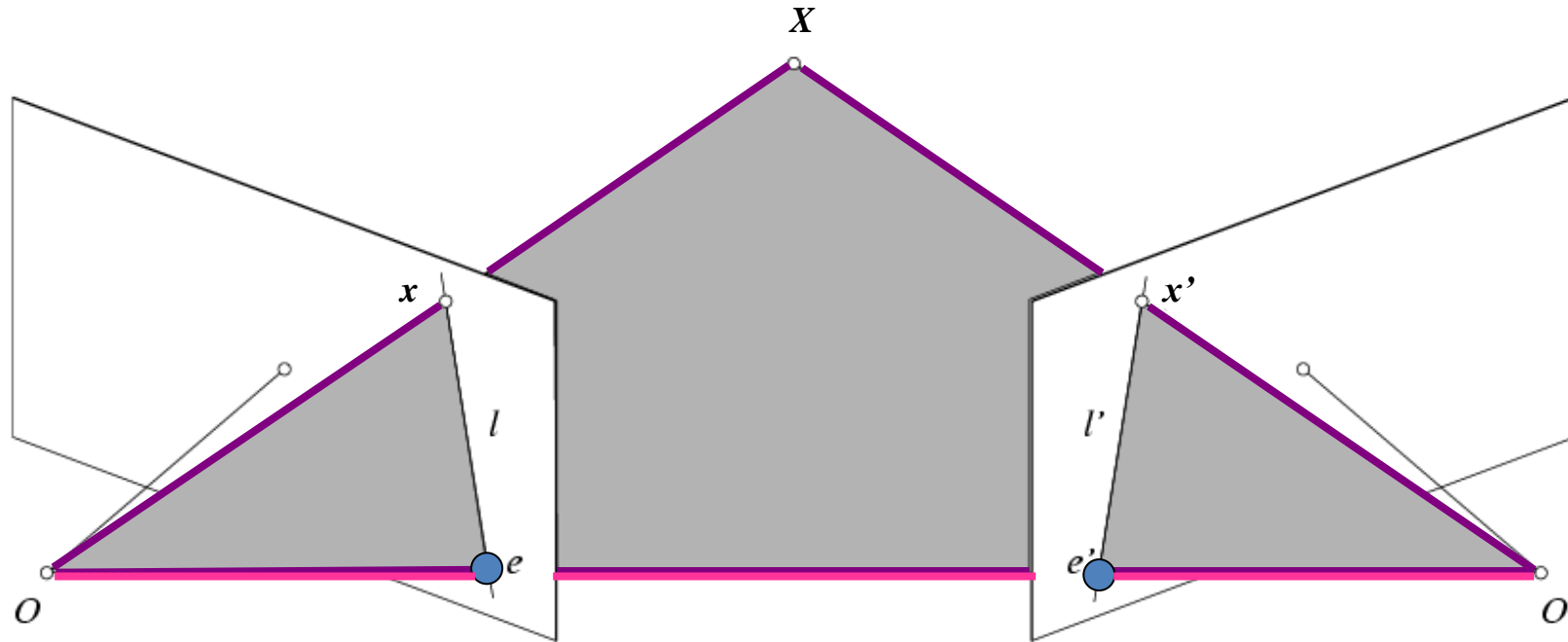
Potential matches for  $x'$  have to lie on the corresponding line  $l$ .

Wouldn't it be nice to know where matches can live? To constrain our 2d search to 1d.

VLFeat's 800 most confident matches  
among 10,000+ local features.



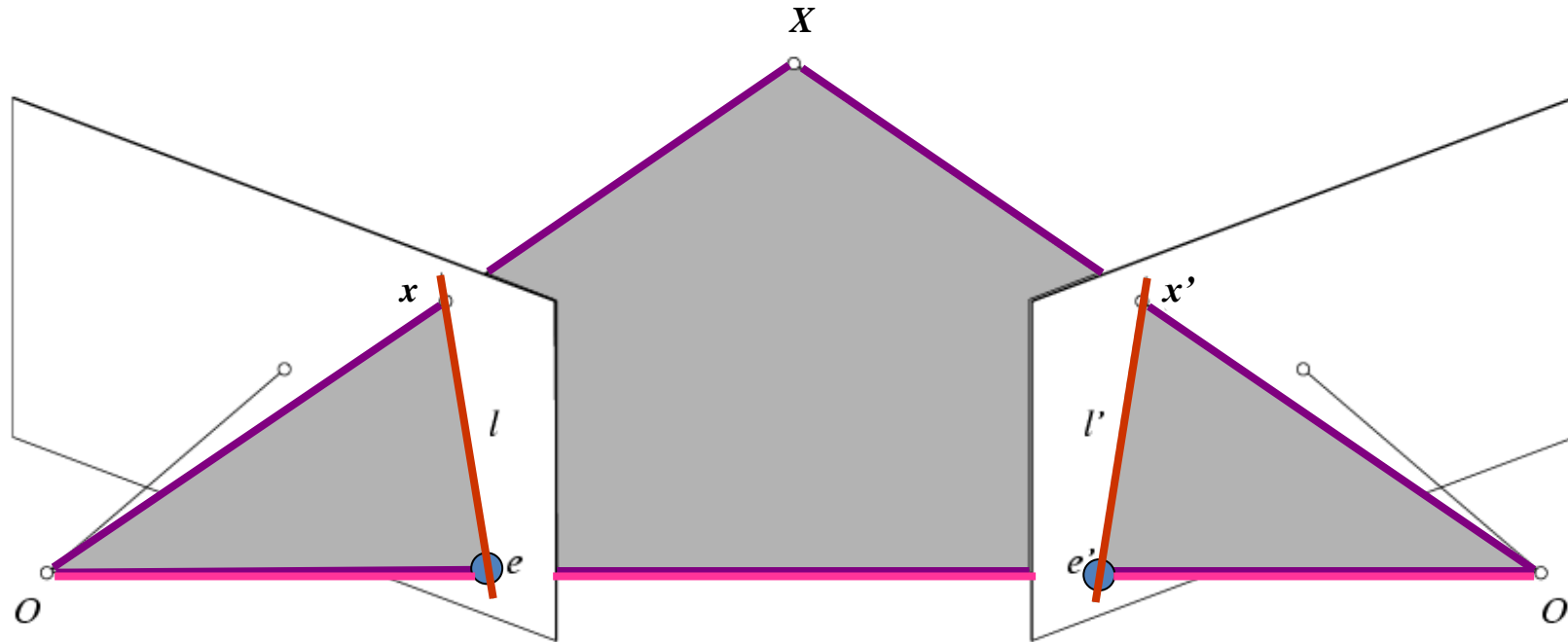
# Epipolar geometry: notation



- **Baseline** – line connecting the two camera centers
- **Epipoles**  
= intersections of baseline with image planes  
= projections of the other camera center
- **Epipolar Plane** – plane containing baseline (1D family)

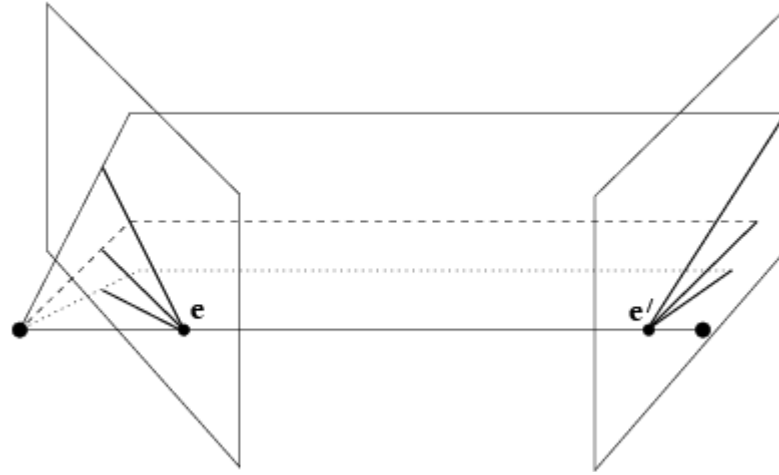


# Epipolar geometry: notation

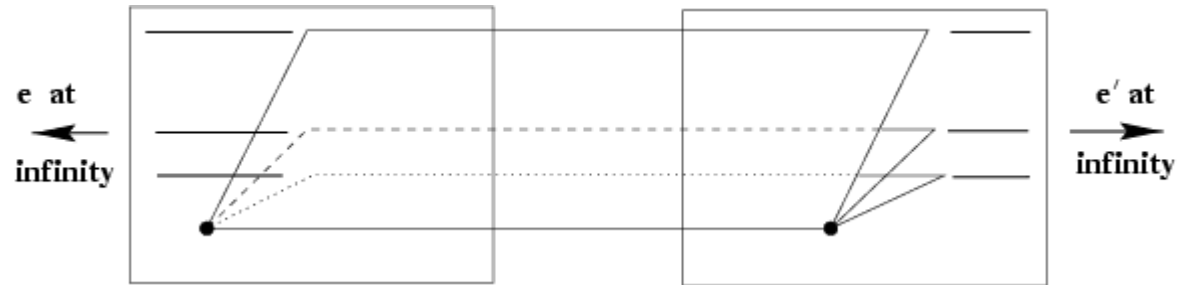


- **Baseline** – line connecting the two camera centers
- **Epipoles**  
= intersections of baseline with image planes  
= projections of the other camera center
- **Epipolar Plane** – plane containing baseline (1D family)
- **Epipolar Lines** - intersections of epipolar plane with image planes (always come in corresponding pairs)

# Example: Converging cameras



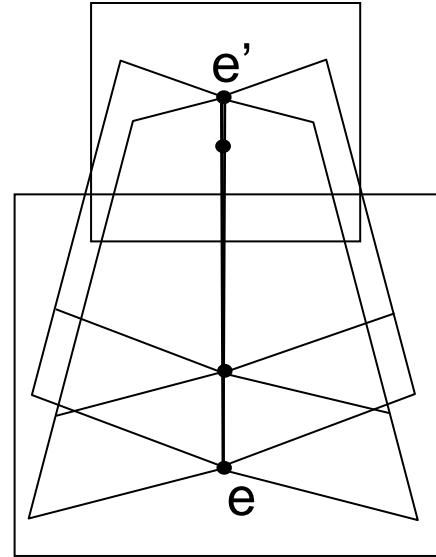
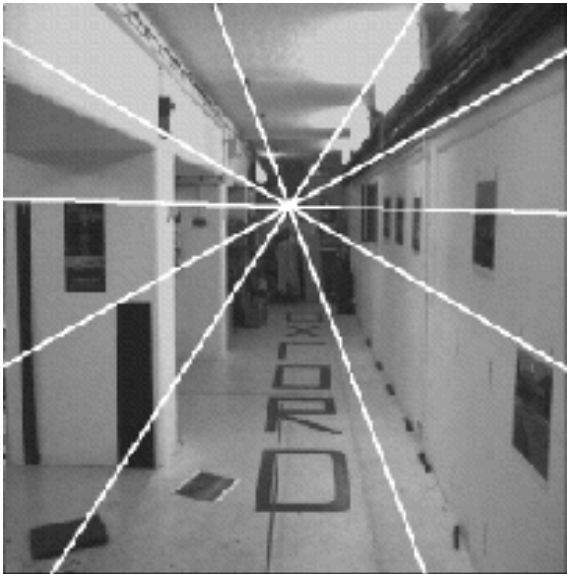
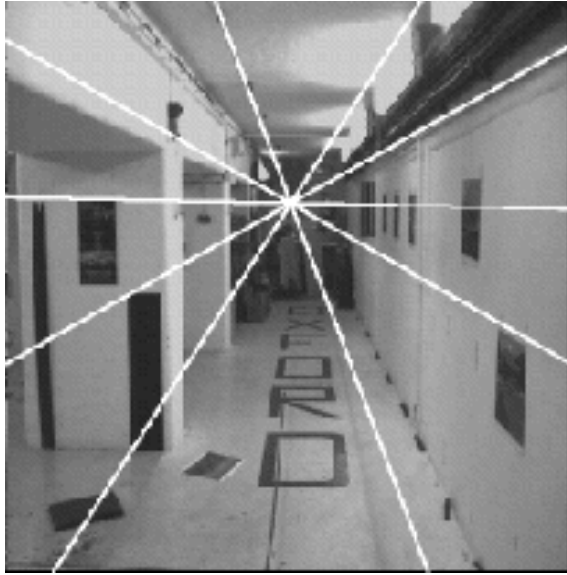
# Example: Motion parallel to image plane



# Example: Forward motion

What would the epipolar lines look like if the camera moves directly forward?

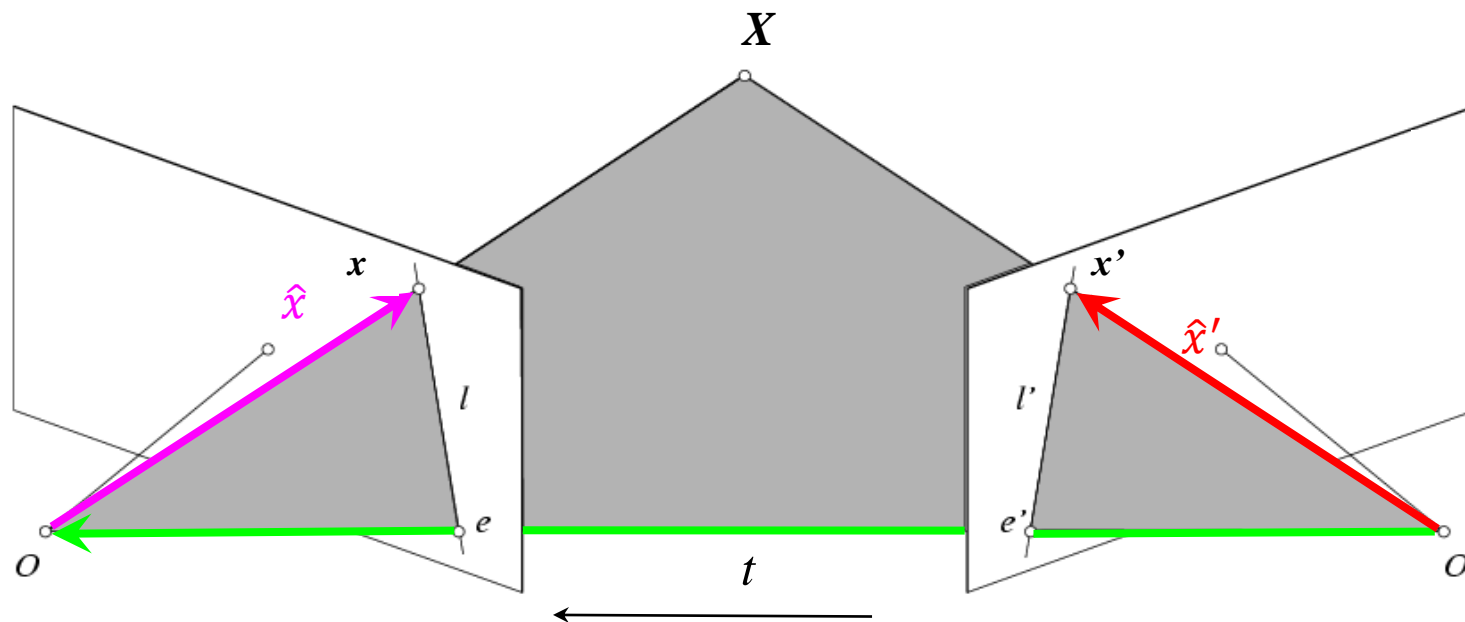
# Example: Forward motion



Epipole has same coordinates in both images.

Points move along lines radiating from  $e$ :  
“Focus of expansion”

# Epipolar constraint: Calibrated case



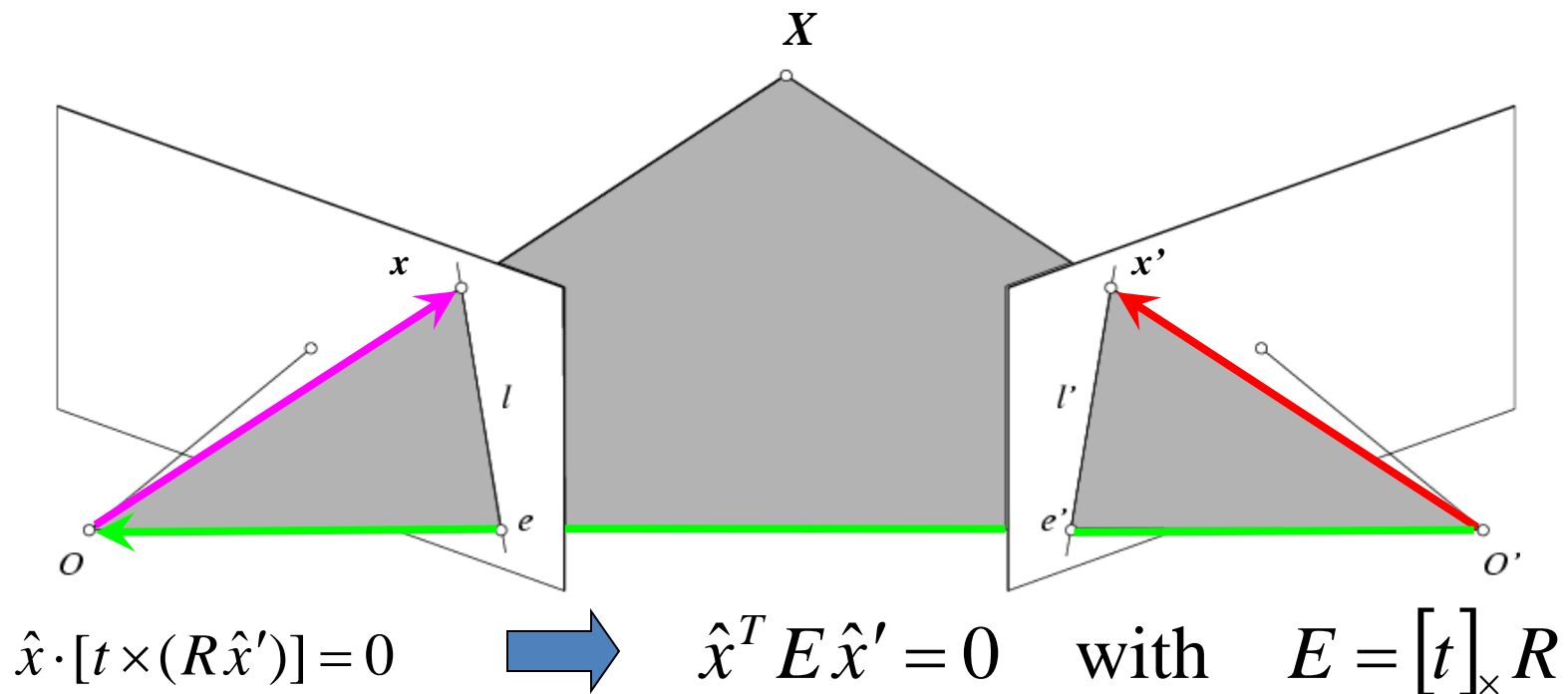
$$\hat{x} = K^{-1} x = X$$

$$\hat{x}' = K'^{-1} x' = X'$$

$$\hat{x} \cdot [t \times (R\hat{x}')] = 0$$

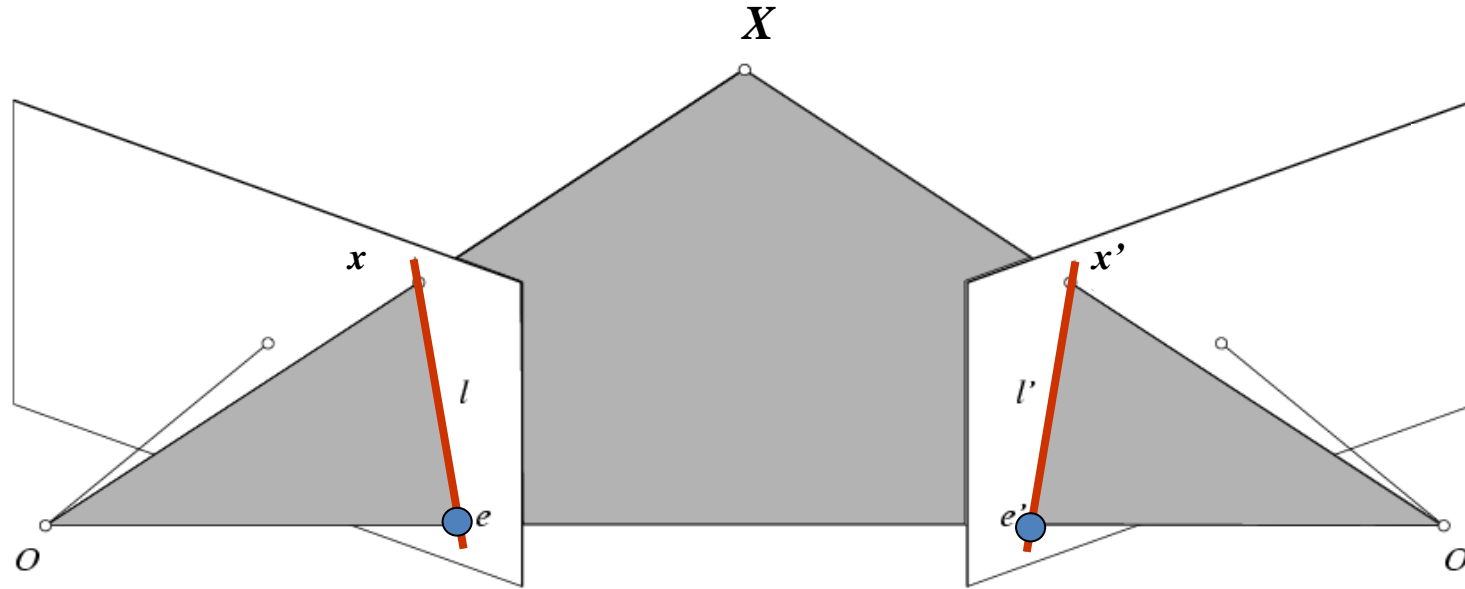
(because  $\hat{x}$ ,  $R\hat{x}'$ , and  $t$  are co-planar)

# Essential matrix



**Essential Matrix**  
(Longuet-Higgins, 1981)

# Properties of the Essential matrix



$$\hat{x} \cdot [t \times (R \hat{x}')] = 0 \quad \Rightarrow \quad \hat{x}^T E \hat{x}' = 0 \quad \text{with} \quad E = [t]_{\times} R$$

Drop ^ below to simplify notation

- $E x'$  is the epipolar line associated with  $x'$  ( $l = E x'$ )
- $E^T x$  is the epipolar line associated with  $x$  ( $l' = E^T x$ )
- $E e' = 0$  and  $E^T e = 0$
- $E$  is singular (rank two)
- $E$  has five degrees of freedom
  - (3 for  $R$ , 2 for  $t$  because it's up to a scale)

Skew-symmetric matrix



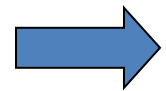
# The Fundamental Matrix

Without knowing  $K$  and  $K'$ , we can define a similar relation using *unknown* normalized coordinates

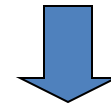
$$\hat{x}^T E \hat{x}' = 0$$

$$\hat{x} = K^{-1} x$$

$$\hat{x}' = K'^{-1} x'$$

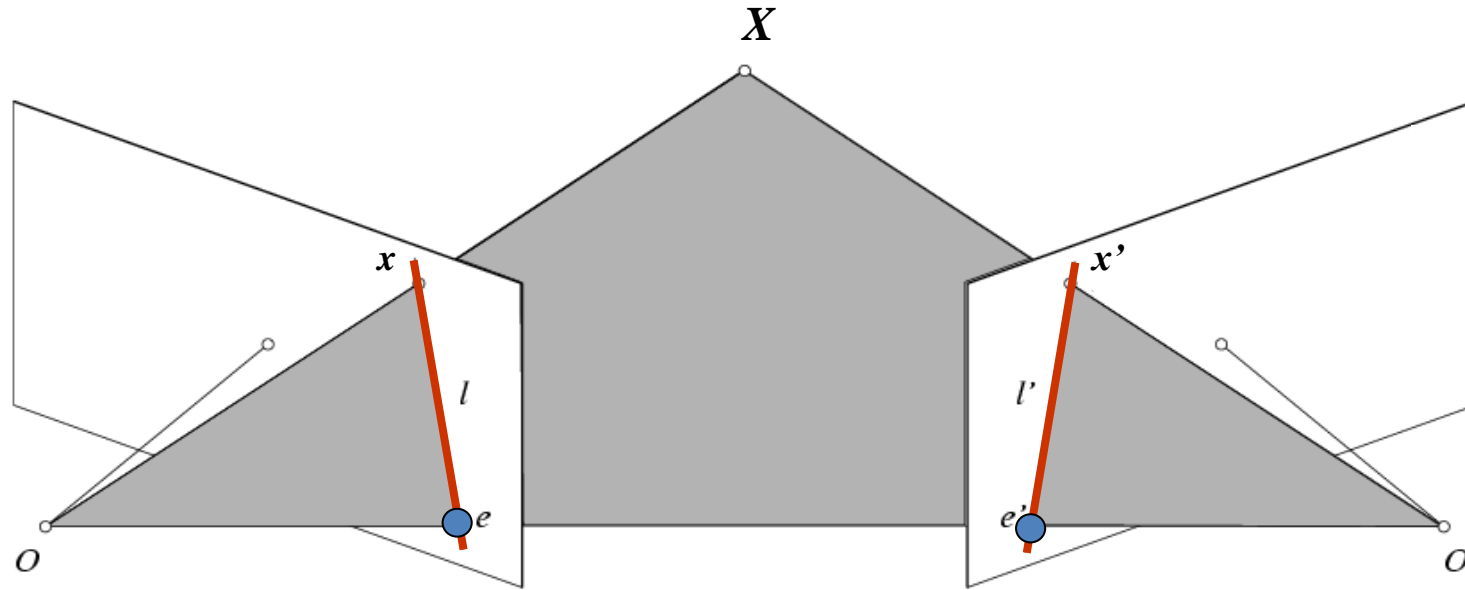


$$x^T F x' = 0 \quad \text{with} \quad F = K^{-T} E K'^{-1}$$



**Fundamental Matrix**  
(Faugeras and Luong, 1992)

# Properties of the Fundamental matrix



$$x^T F x' = 0 \quad \text{with} \quad F = K^{-T} E K'^{-1}$$

- $F x' = 0$  is the epipolar line associated with  $x'$
- $F^T x = 0$  is the epipolar line associated with  $x$
- $F e' = 0$  and  $F^T e = 0$
- $F$  is singular (rank two):  $\det(F)=0$
- $F$  has seven degrees of freedom: 9 entries but defined up to scale,  $\det(F)=0$

# Estimating the Fundamental Matrix

- 8-point algorithm
  - Least squares solution using SVD on equations from 8 pairs of correspondences
  - Enforce  $\det(F)=0$  constraint using SVD on F
- 7-point algorithm
  - Use least squares to solve for null space (two vectors) using SVD and 7 pairs of correspondences
  - Solve for linear combination of null space vectors that satisfies  $\det(F)=0$
- Minimize reprojection error
  - Non-linear least squares

Note: estimation of F (or E) is degenerate for a planar scene.

# 8-point algorithm

1. Solve a system of homogeneous linear equations
  - a. Write down the system of equations

$$\mathbf{x}^T F \mathbf{x}' = 0$$

$$uu'f_{11} + uv'f_{12} + uf_{13} + vu'f_{21} + vv'f_{22} + vf_{23} + u'f_{31} + v'f_{32} + f_{33} = 0$$

$$A\mathbf{f} = \begin{bmatrix} u_1u_1' & u_1v_1' & u_1 & v_1u_1' & v_1v_1' & v_1 & u_1' & v_1' & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_nu_n' & u_nv_n' & u_n & v_nu_n' & v_nv_n' & v_n & u_n' & v_n' & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ \vdots \\ f_{33} \end{bmatrix} = \mathbf{0}$$

# 8-point algorithm

1. Solve a system of homogeneous linear equations
  - a. Write down the system of equations
  - b. Solve  $\mathbf{f}$  from  $\mathbf{A}\mathbf{f}=\mathbf{0}$  using SVD

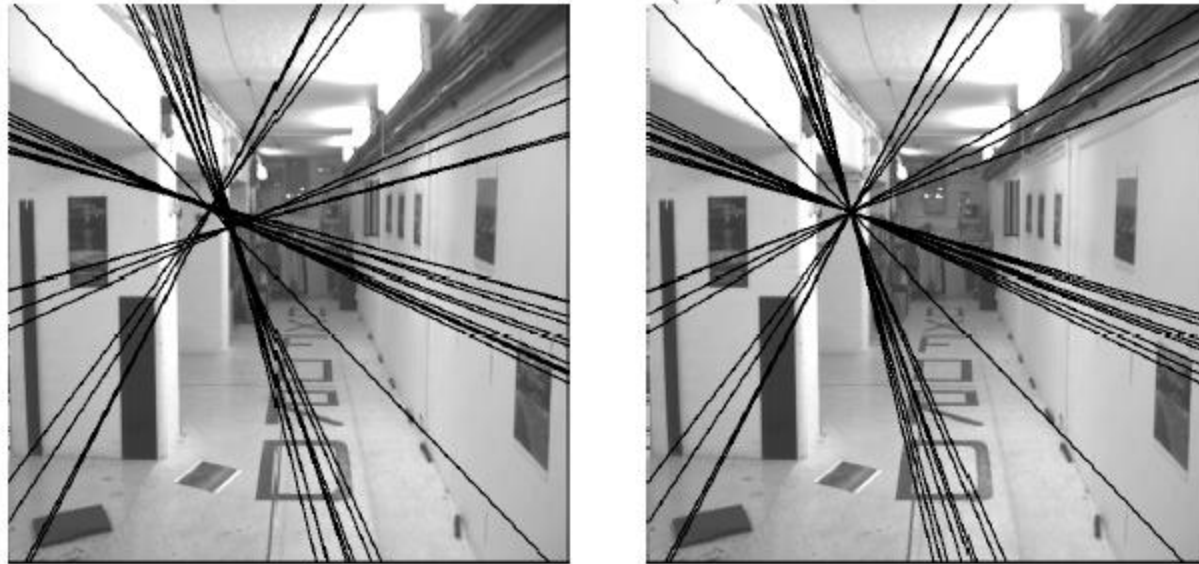
Matlab:

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

For python, see  
`numpy.linalg.svd`

# Need to enforce singularity constraint

Fundamental matrix has rank 2 :  $\det(\mathbf{F}) = 0$ .



**Left :** Uncorrected  $\mathbf{F}$  – epipolar lines are not coincident.

**Right :** Epipolar lines from corrected  $\mathbf{F}$ .

# 8-point algorithm

1. Solve a system of homogeneous linear equations
  - a. Write down the system of equations
  - b. Solve  $\mathbf{f}$  from  $\mathbf{A}\mathbf{f}=\mathbf{0}$  using SVD

Matlab:

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

2. Resolve  $\det(F) = 0$  constraint using SVD

Matlab:

```
[U, S, V] = svd(F);  
S(3,3) = 0;  
F = U*S*V';
```

For python, see  
[numpy.linalg.svd](#)

# 8-point algorithm

1. Solve a system of homogeneous linear equations
  - a. Write down the system of equations
  - b. Solve  $\mathbf{f}$  from  $A\mathbf{f}=\mathbf{0}$  using SVD
2. Resolve  $\det(F) = 0$  constraint by SVD

## Notes:

- Use RANSAC to deal with outliers (sample 8 points)
  - How to test for outliers?



# Problem with eight-point algorithm

---

$$\begin{bmatrix} u'u & u'v & u' & v'u & v'v & v' & u & v \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \end{bmatrix} = -1$$

# Problem with eight-point algorithm

---

250906.36	183269.57	921.81	200931.10	146766.13	738.21	272.19	198.81
2692.28	131633.03	176.27	6196.73	302975.59	405.71	15.27	746.79
416374.23	871684.30	935.47	408110.89	854384.92	916.90	445.10	931.81
191183.60	171759.40	410.27	416435.62	374125.90	893.65	465.99	418.65
48988.86	30401.76	57.89	298604.57	185309.58	352.87	846.22	525.15
164786.04	546559.67	813.17	1998.37	6628.15	9.86	202.65	672.14
116407.01	2727.75	138.89	169941.27	3982.21	202.77	838.12	19.64
135384.58	75411.13	198.72	411350.03	229127.78	603.79	681.28	379.48

$$\begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \end{bmatrix} = -\mathbf{1}$$

Poor numerical conditioning

Can be fixed by rescaling the data

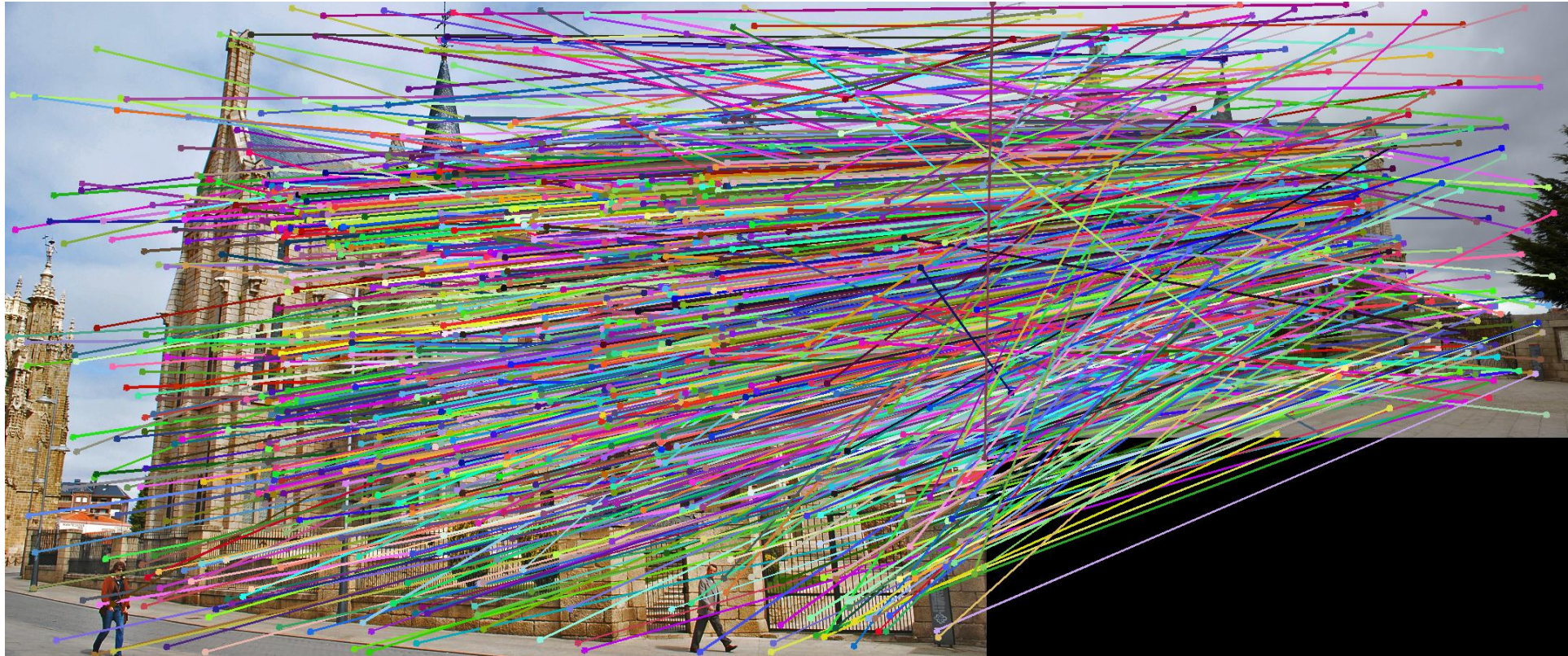
# The normalized eight-point algorithm

---

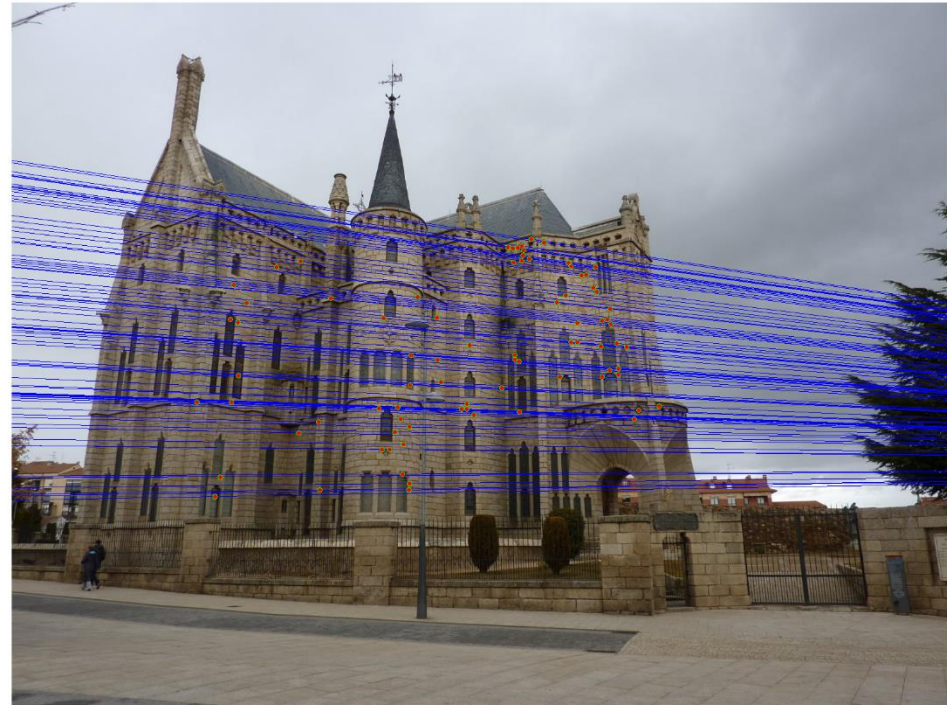
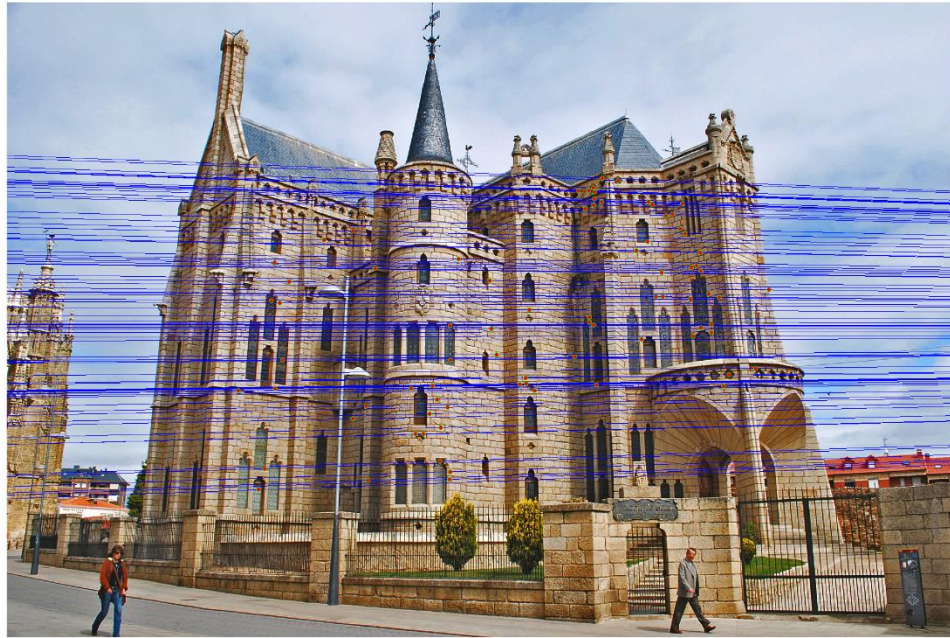
(Hartley, 1995)

- Center the image data at the origin, and scale it so the mean squared distance between the origin and the data points is 2 pixels
- Use the eight-point algorithm to compute  $\mathbf{F}$  from the normalized points
- Enforce the rank-2 constraint (for example, take SVD of  $\mathbf{F}$  and throw out the smallest singular value)
- Transform fundamental matrix back to original units: if  $\mathbf{T}$  and  $\mathbf{T}'$  are the normalizing transformations in the two images, then the fundamental matrix in original coordinates is  $\mathbf{T}'^T \mathbf{F} \mathbf{T}$

VLFeat's 800 most confident matches  
among 10,000+ local features.



# Epipolar lines



Keep only the matches that are “inliers” with respect to the “best” fundamental matrix

