Robotic Mapping & Localization

Kaveh Fathian
Assistant Professor
Computer Science Department
Colorado School of Mines

Lec09: Two-View Geometry

Our Simple SLAM System

robotic-mapping / code / visual-odometry / vo_epipolar.cpp

KavehFathian first commit

Code Blame 97 lines (88 loc) · 3.52 KB

```cpp
#include <iostream>
#include <vector>
#include <string>
#include "opencv2/opencv.hpp"

int main()
{
    const char* video_file = "../data/KITTI07/image_0/%06d.png";
    double f = 707.0912;
    cv::Point2d c(601.8873, 183.1104);
    bool use_5pt = true;
    int min_inlier_num = 100;
    double min_inlier_ratio = 0.2;
    const char* traj_file = "vo_epipolar.xyz";
```

Our Simple SLAM System


```cpp
#include <iostream>
#include <vector>
#include <string>
#include "opencv2/opencv.hpp"

int main()
{
    const char* video_file = "../data/KITTI07/image_0/%06d.png";
    double f = 707.0912;
    cv::Point2d c(601.8873, 183.1104);
    bool use_5pt = true;
    int min_inlier_num = 100;
    double min_inlier_ratio = 0.2;
    const char* traj_file = "vo_epipolar.xyz";

    // Open a video and get an initial image
    cv::VideoCapture video;
    if (!video.open(video_file)) return -1;

    cv::Mat gray_prev;
    video >> gray_prev;
    if (gray_prev.empty())
    {
        video.release();
        return -1;
    }
    if (gray_prev.channels() > 1) cv::cvtColor(gray_prev, gray_prev,
    cv::COLOR_RGB2GRAY);

    // Run the monocular visual odometry
    cv::Mat K = (cv::Mat_<double>(3, 3) << f, 0, c.x, 0, f, c.y, 0, 0, 1);
    cv::Mat camera_pose = cv::Mat::eye(4, 4, CV_64F);
    FILE* camera_traj = fopen(traj_file, "wt");
    if (camera_traj == NULL) return -1;
```

OpenCV library

Camera parameters

SLAM system parameters

Convert RGB images to gray

Camera calibration matrix

Pose (position & orientation) estimates

Our Simple SLAM System

```
34      while (true)
35      {
36          // Grab an image from the video
37          cv::Mat img, gray;
38          video >> img;
39          if (img.empty()) break;
40          if (img.channels() > 1) cv::cvtColor(img, gray, cv::COLOR_RGB2GRAY);
41          else                    gray = img.clone();
42
43          // Extract optical flow
44          std::vector<cv::Point2f> pts_prev, pts;
45          cv::goodFeaturesToTrack(gray_prev, pts_prev, 2000, 0.01, 10);
46          std::vector<uchar> status;
47          cv::Mat err;
48          cv::calcOpticalFlowPyrLK(gray_prev, gray, pts_prev, pts, status, err);
49          gray_prev = gray;
50
51          // Calculate relative pose
52          cv::Mat E, inlier_mask;
53          if (use_5pt)
54          {
55              E = cv::findEssentialMat(pts_prev, pts, f, c, cv::RANSAC, 0.999, 1,
                inlier_mask);
56          }
57          else
58          {
59              cv::Mat F = cv::findFundamentalMat(pts_prev, pts, cv::FM_RANSAC, 1, 0.
                99, inlier_mask);
60              E = K.t() * F * K;
61          }
62          cv::Mat R, t;
63          int inlier_num = cv::recoverPose(E, pts_prev, pts, R, t, f, c, inlier_mask);
64          double inlier_ratio = static_cast<double>(inlier_num) / static_cast<double>
            (pts.size());
65
66          // Accumulate relative pose if result is reliable
67          cv::Vec3b info_color(0, 255, 0);
68          if ((inlier_num > min_inlier_num) && (inlier_ratio > min_inlier_ratio))
69          {
70              cv::Mat T = cv::Mat::eye(4, 4, R.type());
71              T(cv::Rect(0, 0, 3, 3)) = R * 1.0;
72              T.col(3).rowRange(0, 3) = t * 1.0;
73              camera_pose = camera_pose * T.inv();
74              info_color = cv::Vec3b(0, 0, 255);
75          }
```

Iterate over images

Feature detection & tracking using optical flow

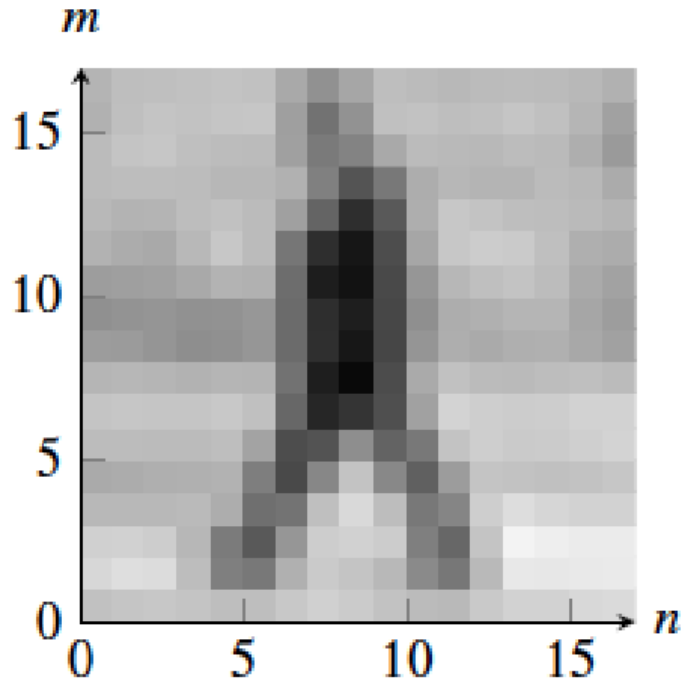
Estimate camera pose (in the form of essential matrix) from consecutive images

RANSAC outlier rejection

Extract camera pose from essential matrix

Camera pose (and trajectory) over time

Image as a 2D discrete signal

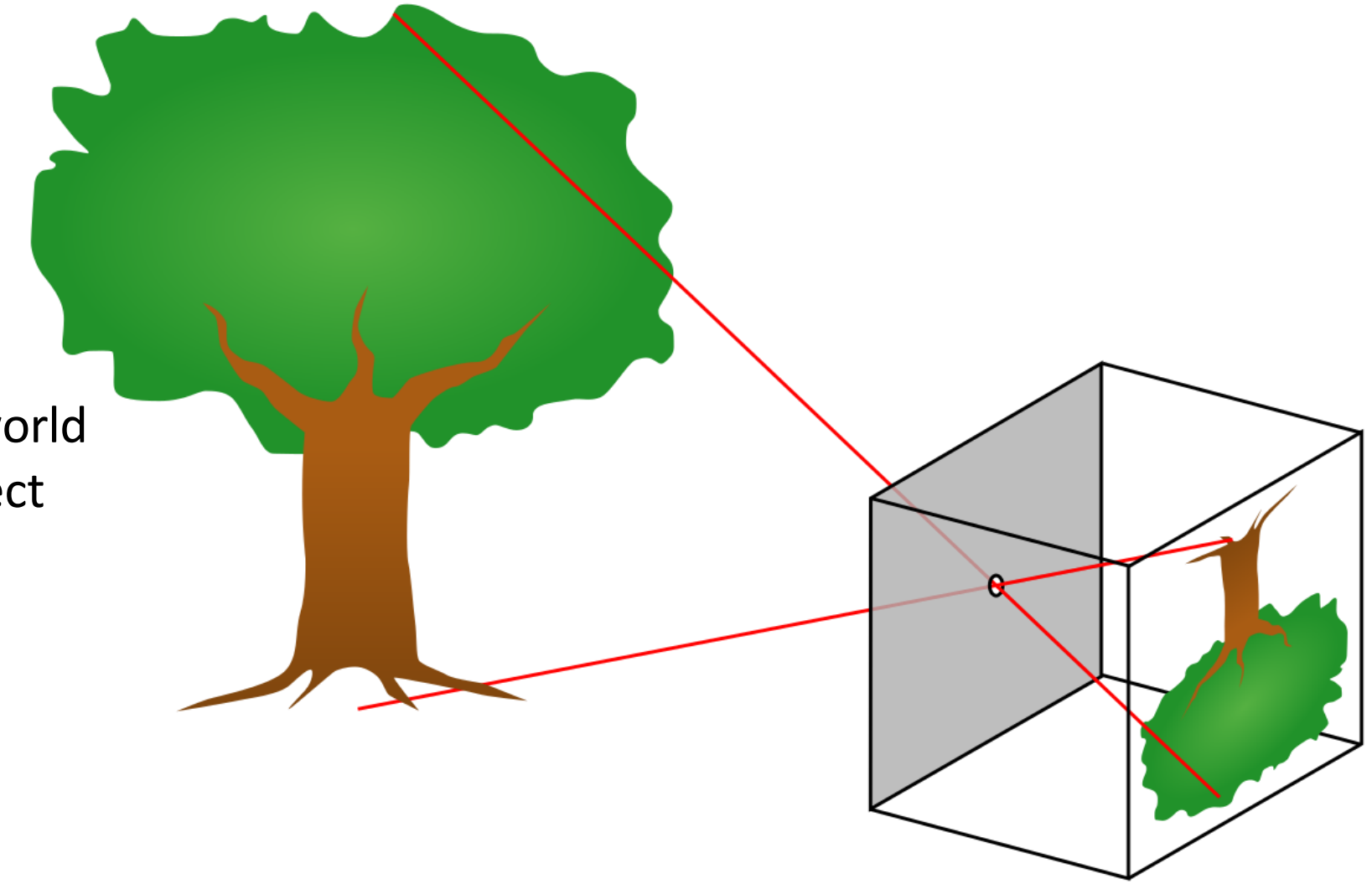


A tiny person of 18 x 18 pixels

$$\mathbf{I} = \begin{bmatrix}
160 & 175 & 171 & 168 & 168 & 172 & 164 & 158 & 167 & 173 & 167 & 163 & 162 & 164 & 160 & 159 & 163 & 162 \\
149 & 164 & 172 & 175 & 178 & 179 & 176 & 118 & 97 & 168 & 175 & 171 & 169 & 175 & 176 & 177 & 165 & 152 \\
161 & 166 & 182 & 171 & 170 & 177 & 175 & 116 & 109 & 169 & 177 & 173 & 168 & 175 & 175 & 159 & 153 & 123 \\
171 & 174 & 177 & 175 & 167 & 161 & 157 & 138 & 103 & 112 & 157 & 164 & 159 & 160 & 165 & 169 & 148 & 144 \\
163 & 163 & 162 & 165 & 167 & 164 & 178 & 167 & 77 & 55 & 134 & 170 & 167 & 162 & 164 & 175 & 168 & 160 \\
173 & 164 & 158 & 165 & 180 & 180 & 150 & 89 & 61 & 34 & 137 & 186 & 186 & 182 & 175 & 165 & 160 & 164 \\
152 & 155 & 146 & 147 & 169 & 180 & 163 & 51 & 24 & 32 & 119 & 163 & 175 & 182 & 181 & 162 & 148 & 153 \\
134 & 135 & 147 & 149 & 150 & 147 & 148 & 62 & 36 & 46 & 114 & 157 & 163 & 167 & 169 & 163 & 146 & 147 \\
135 & 132 & 131 & 125 & 115 & 129 & 132 & 74 & 54 & 41 & 104 & 156 & 152 & 156 & 164 & 156 & 141 & 144 \\
151 & 155 & 151 & 145 & 144 & 149 & 143 & 71 & 31 & 29 & 129 & 164 & 157 & 155 & 159 & 158 & 156 & 148 \\
172 & 174 & 178 & 177 & 177 & 181 & 174 & 54 & 21 & 29 & 136 & 190 & 180 & 179 & 176 & 184 & 187 & 182 \\
177 & 178 & 176 & 173 & 174 & 180 & 150 & 27 & 101 & 94 & 74 & 189 & 188 & 186 & 183 & 186 & 188 & 187 \\
160 & 160 & 163 & 163 & 161 & 167 & 100 & 45 & 169 & 166 & 59 & 136 & 184 & 176 & 175 & 177 & 185 & 186 \\
147 & 150 & 153 & 155 & 160 & 155 & 56 & 111 & 182 & 180 & 104 & 84 & 168 & 172 & 171 & 164 & 168 & 167 \\
184 & 182 & 178 & 175 & 179 & 133 & 86 & 191 & 201 & 204 & 191 & 79 & 172 & 220 & 217 & 205 & 209 & 200 \\
184 & 187 & 192 & 182 & 124 & 32 & 109 & 168 & 171 & 167 & 163 & 51 & 105 & 203 & 209 & 203 & 210 & 205 \\
191 & 198 & 203 & 197 & 175 & 149 & 169 & 189 & 190 & 173 & 160 & 145 & 156 & 202 & 199 & 201 & 205 & 202 \\
153 & 149 & 153 & 155 & 173 & 182 & 179 & 177 & 182 & 177 & 182 & 185 & 179 & 177 & 167 & 176 & 182 & 180
\end{bmatrix}$$

Pinhole imaging

real-world object



copy of real-world object (inverted & scaled)

The camera as a coordinate transformation

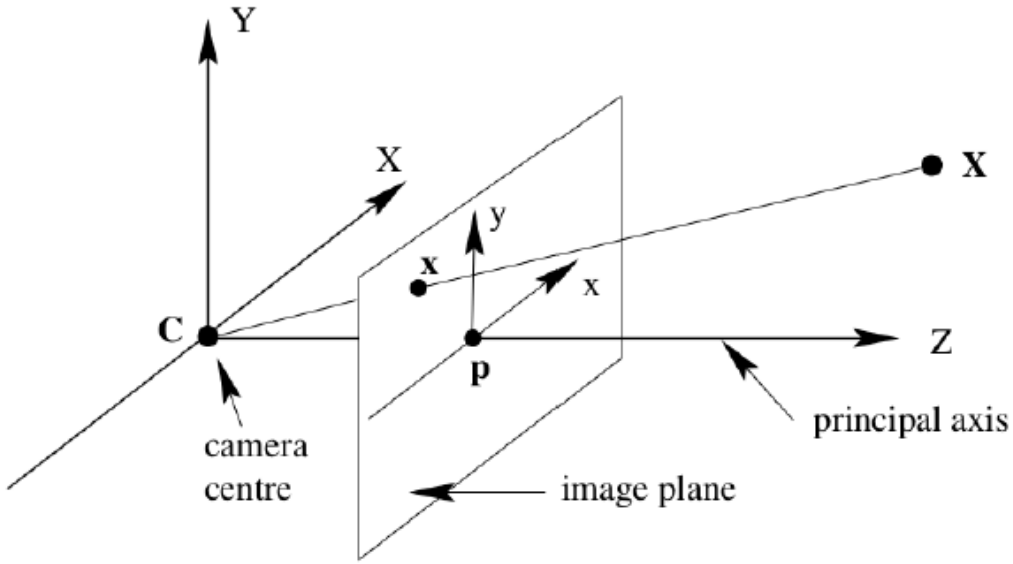
homogeneous coordinates

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

2D image point

camera matrix

3D world point



The camera as a coordinate transformation

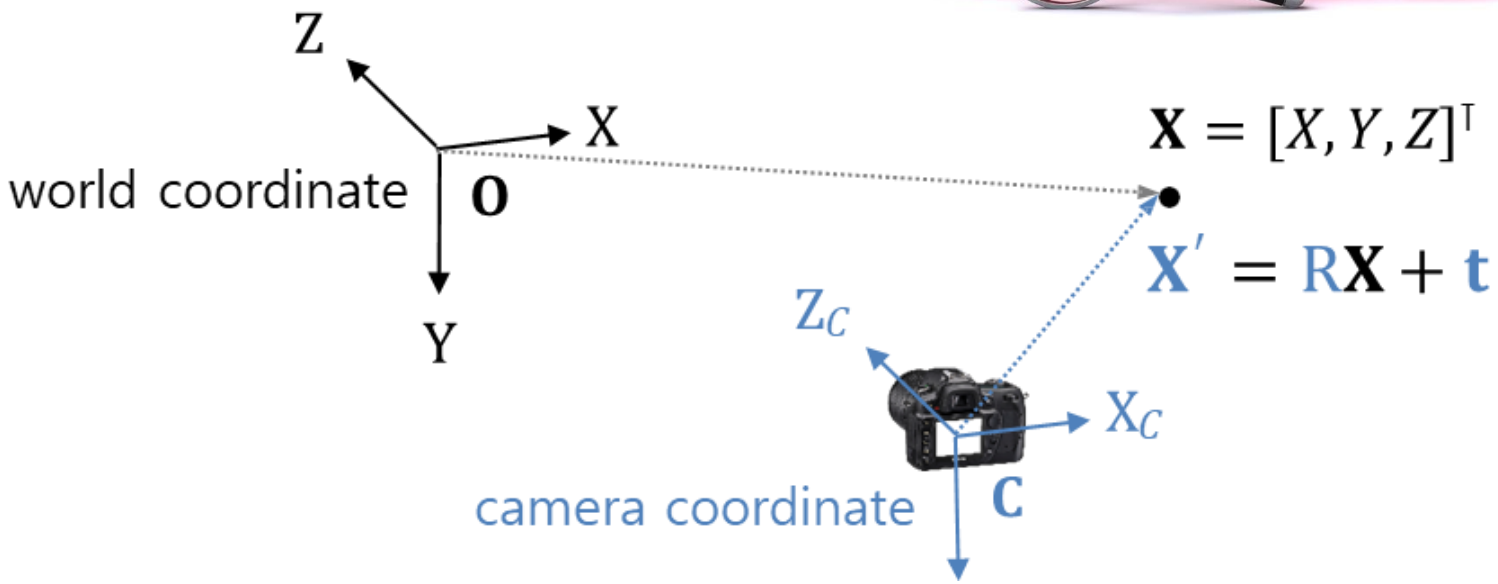
$$\mathbf{x} = \mathbf{P}\mathbf{X}$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

homogeneous image coordinates	camera matrix	homogeneous world coordinates
3 x 1	3 x 4	4 x 1

General pinhole camera matrix

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$$



$$\mathbf{P} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \left[\begin{array}{ccc|c} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \end{array}\right]$$

intrinsic parameters

extrinsic parameters

$$\mathbf{R} = \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix} \quad \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$$

3D rotation

3D translation

Simulate a Camera

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

Unknown Known Known

Camera Calibration

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

Known Unknown Known

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

Intrinsics Calibration

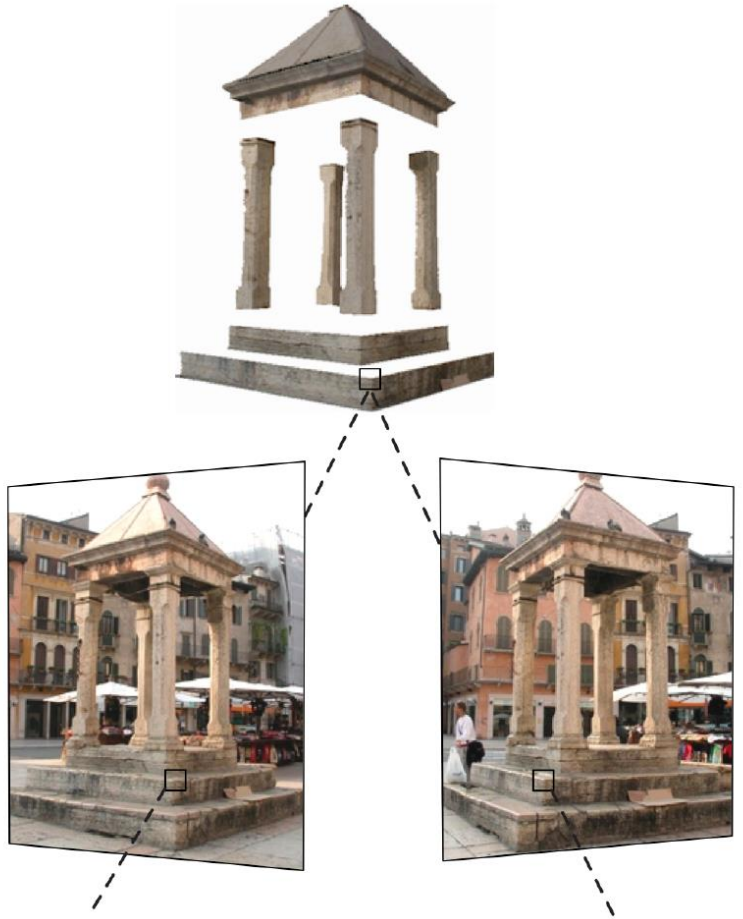
Extrinsics Calibration / Pose Estimation

3D Reconstruction

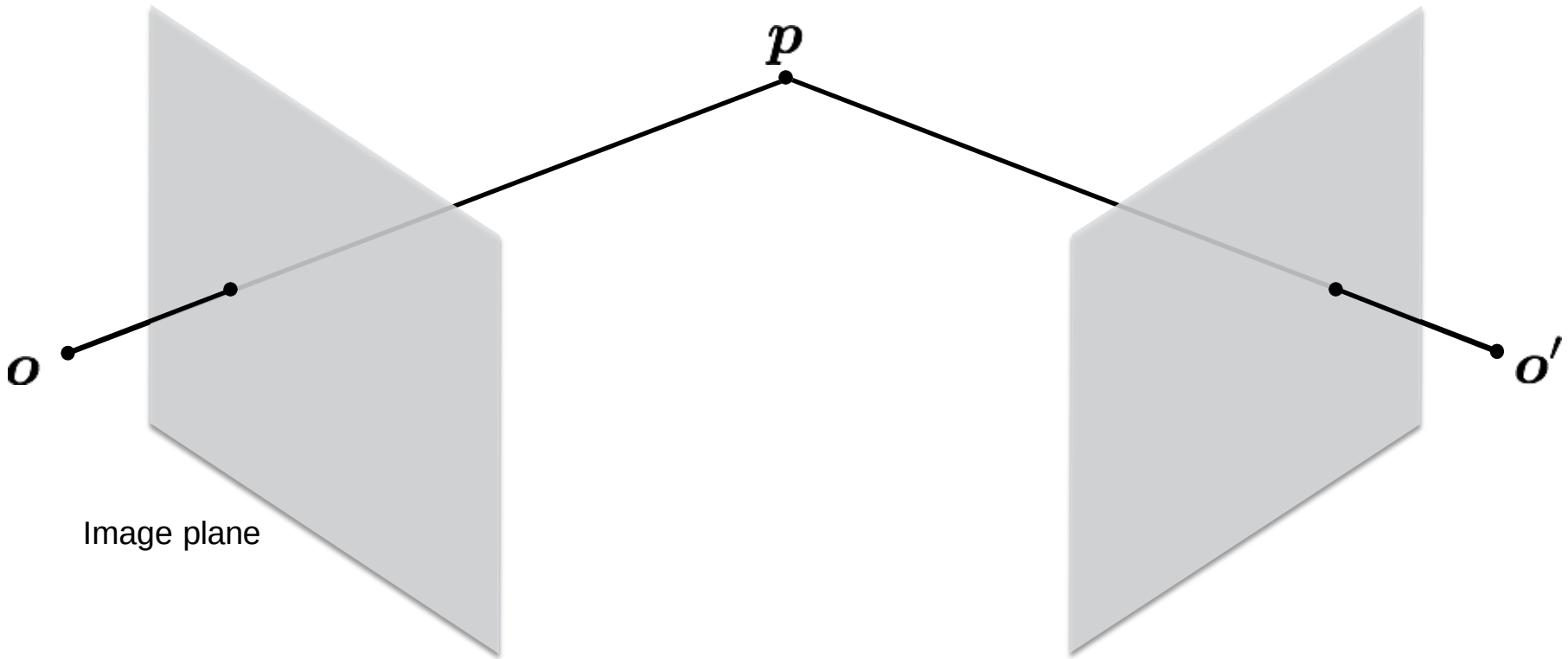
$$\mathbf{x} = \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}\mathbf{X}$$

Known Known Unknown

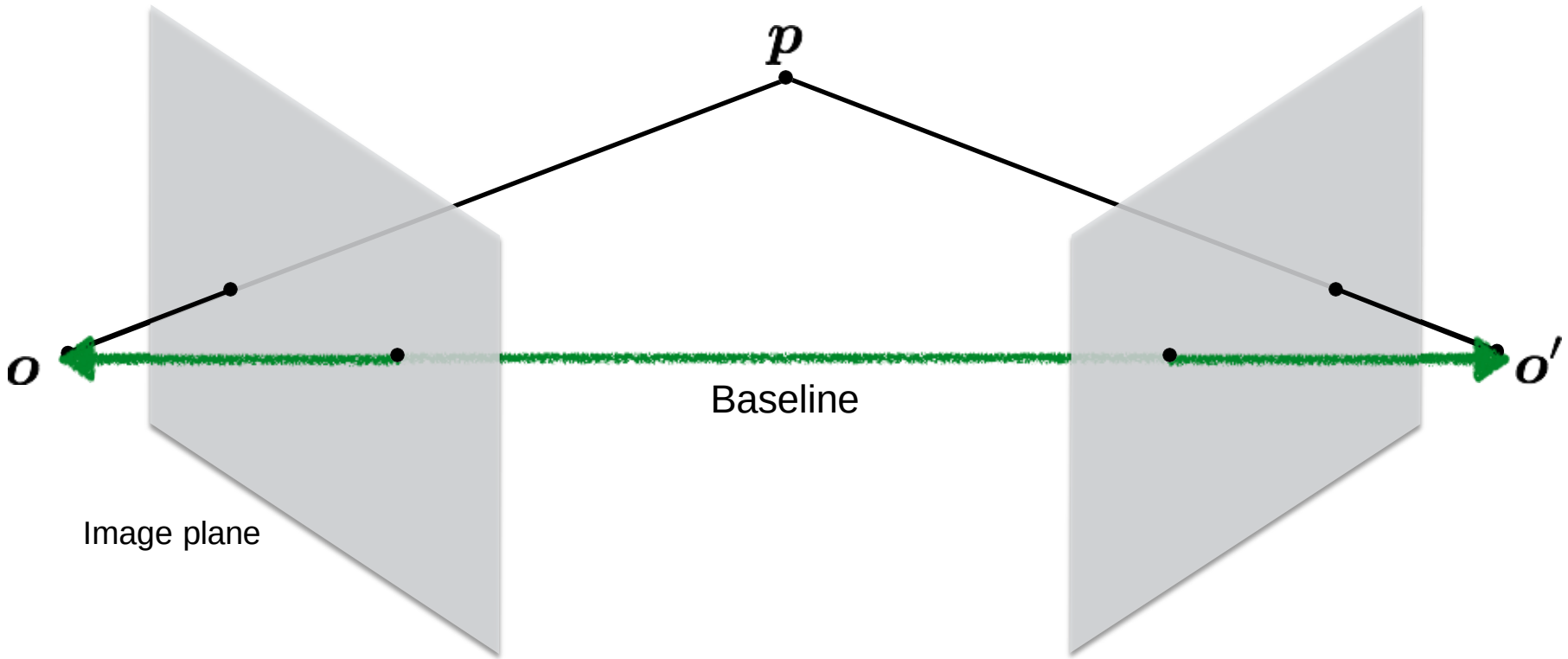
Two-View Geometry



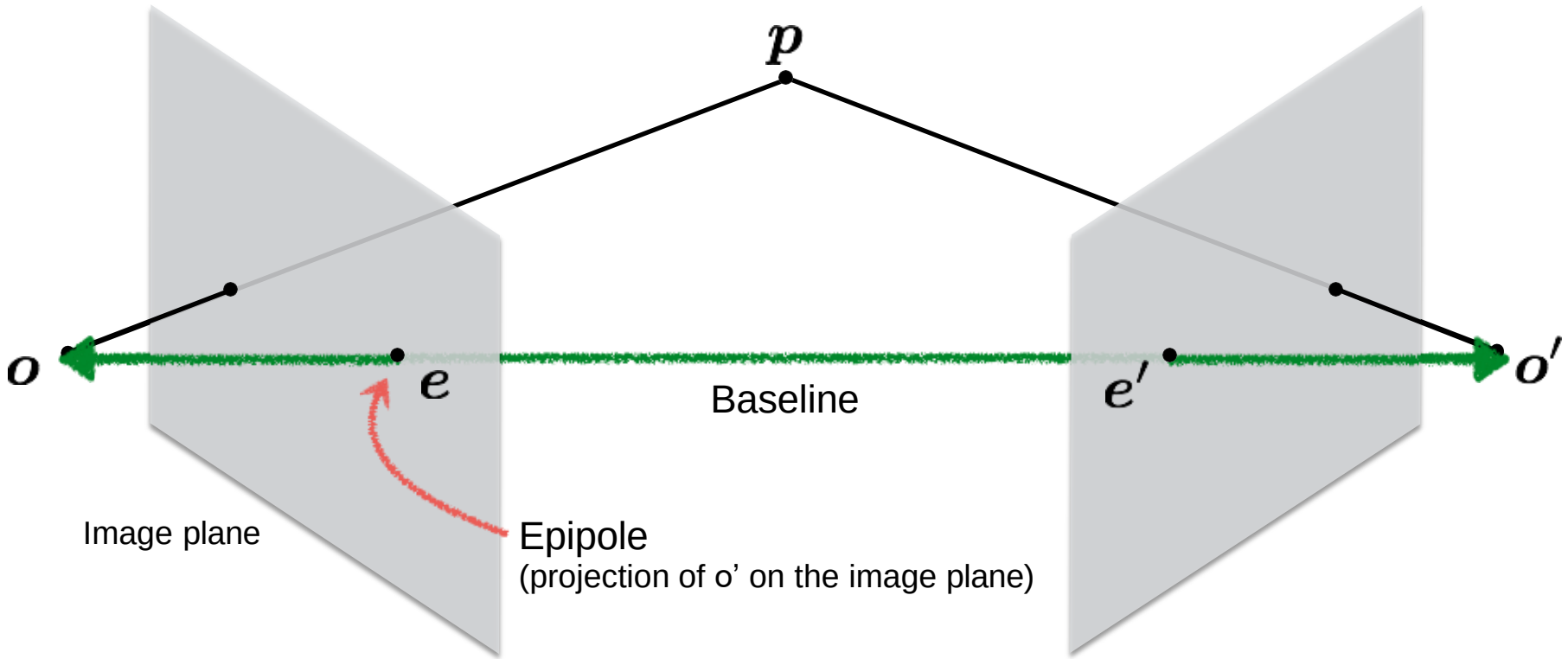
Notation



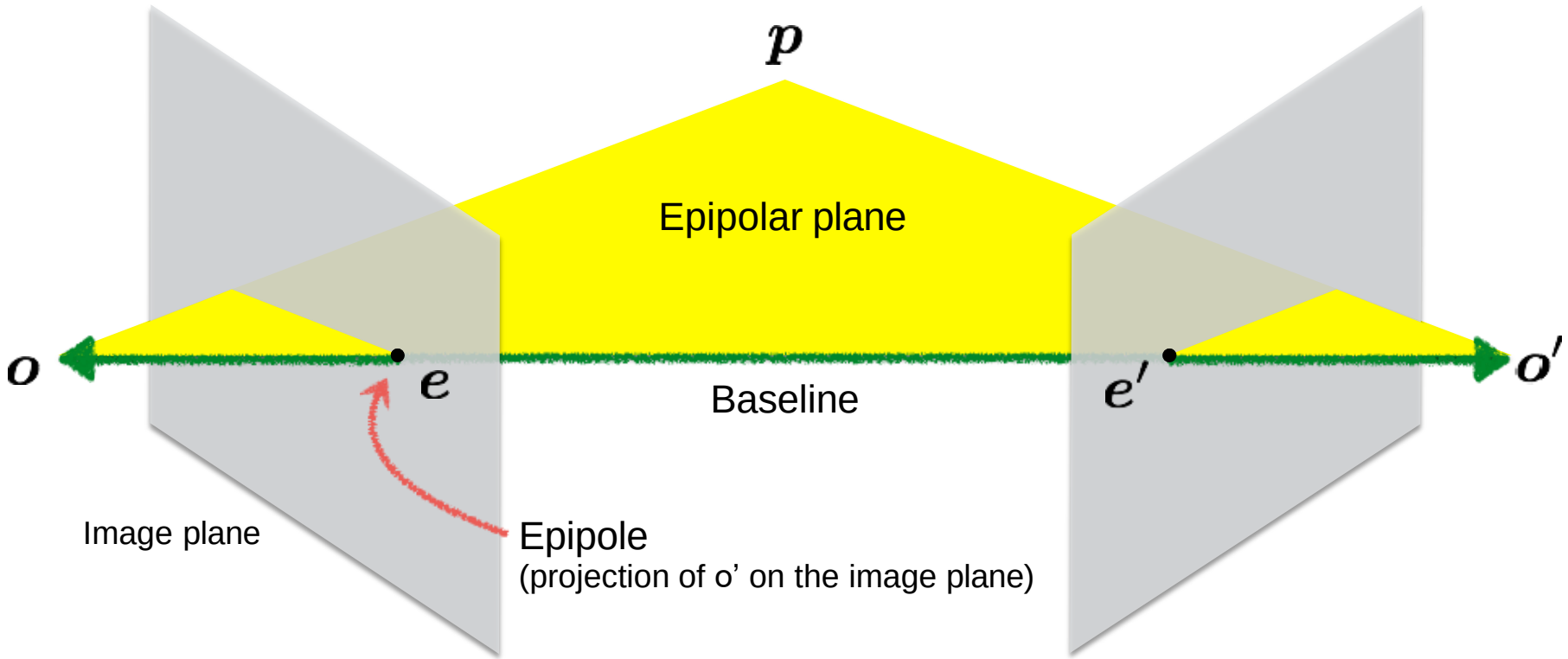
Notation



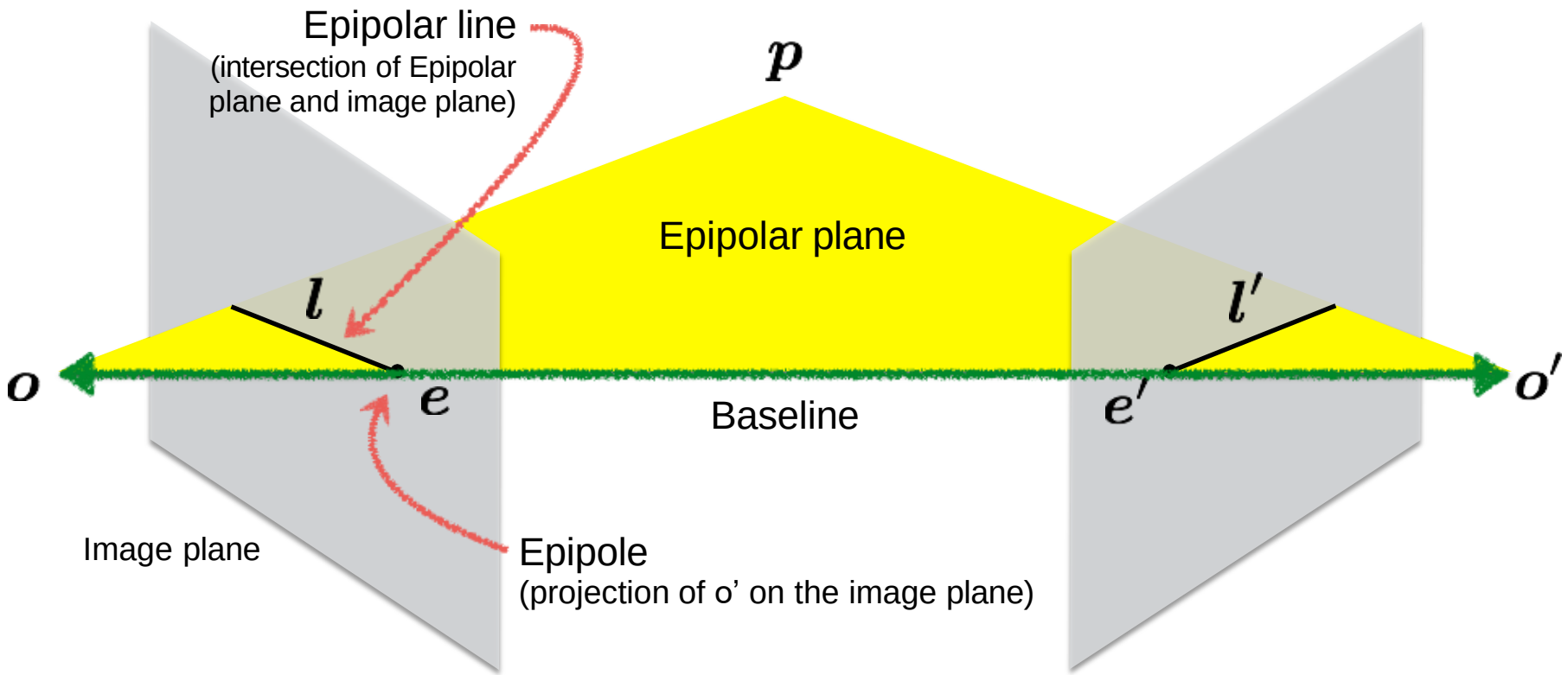
Notation


p
o
e
Baseline
e′
o′
Image plane
Epipole
(projection of o’ on the image plane)

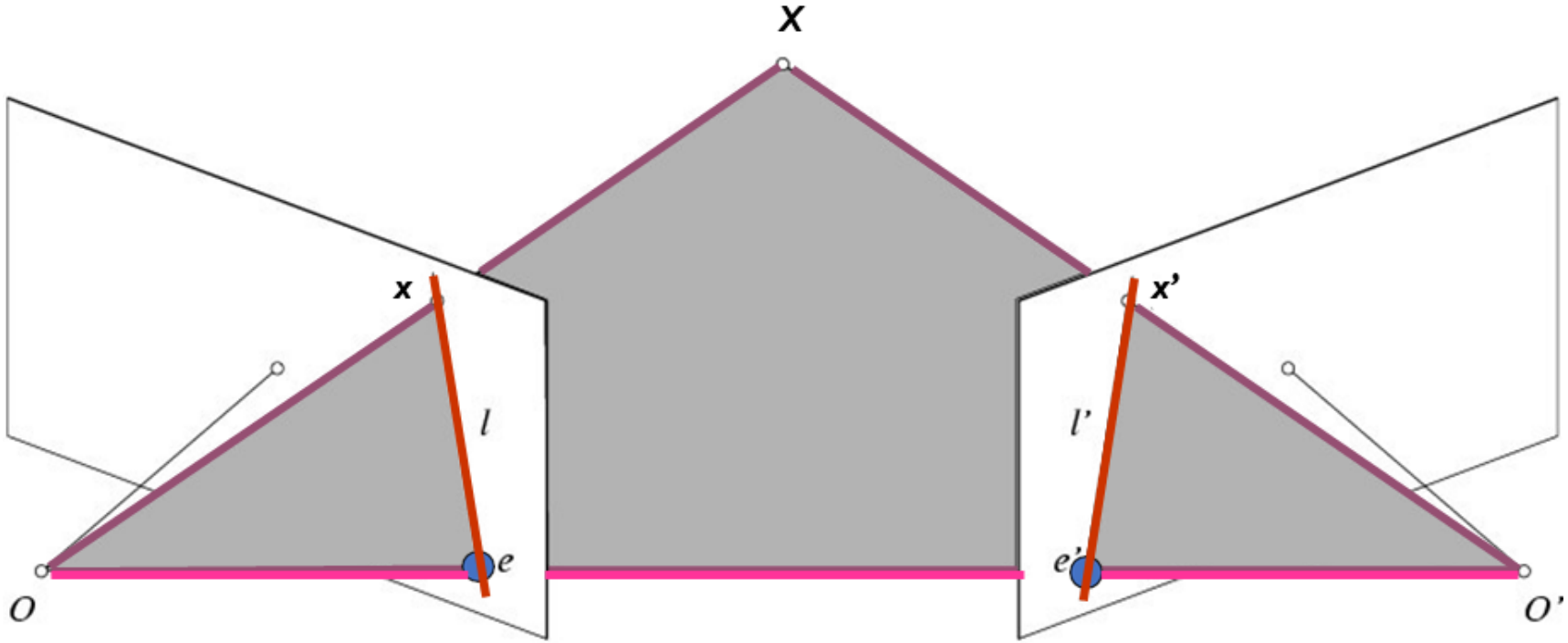
Notation


p
Epipolar plane
o
e
Baseline
e′
o′
Image plane
Epipole
(projection of o’ on the image plane)

Notation


Epipolar line
(intersection of Epipolar plane and image plane)
p
Epipolar plane
l
l′
o
e
e′
o′
Baseline
Image plane
Epipole
(projection of o’ on the image plane)

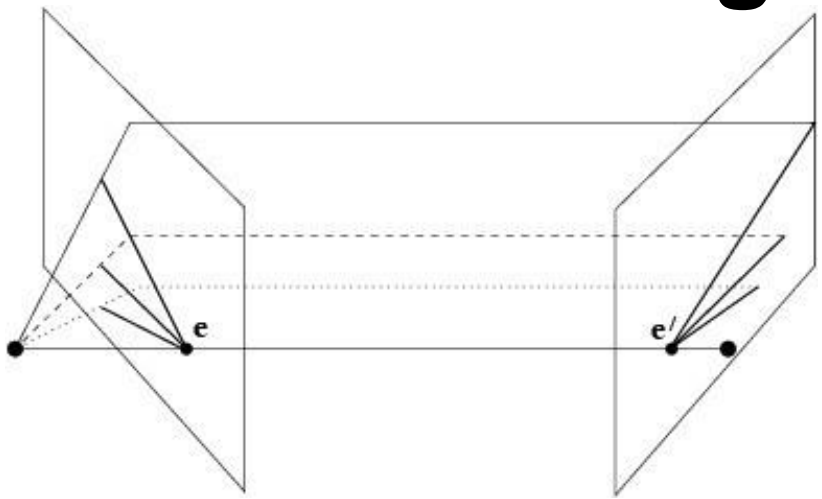
Notation

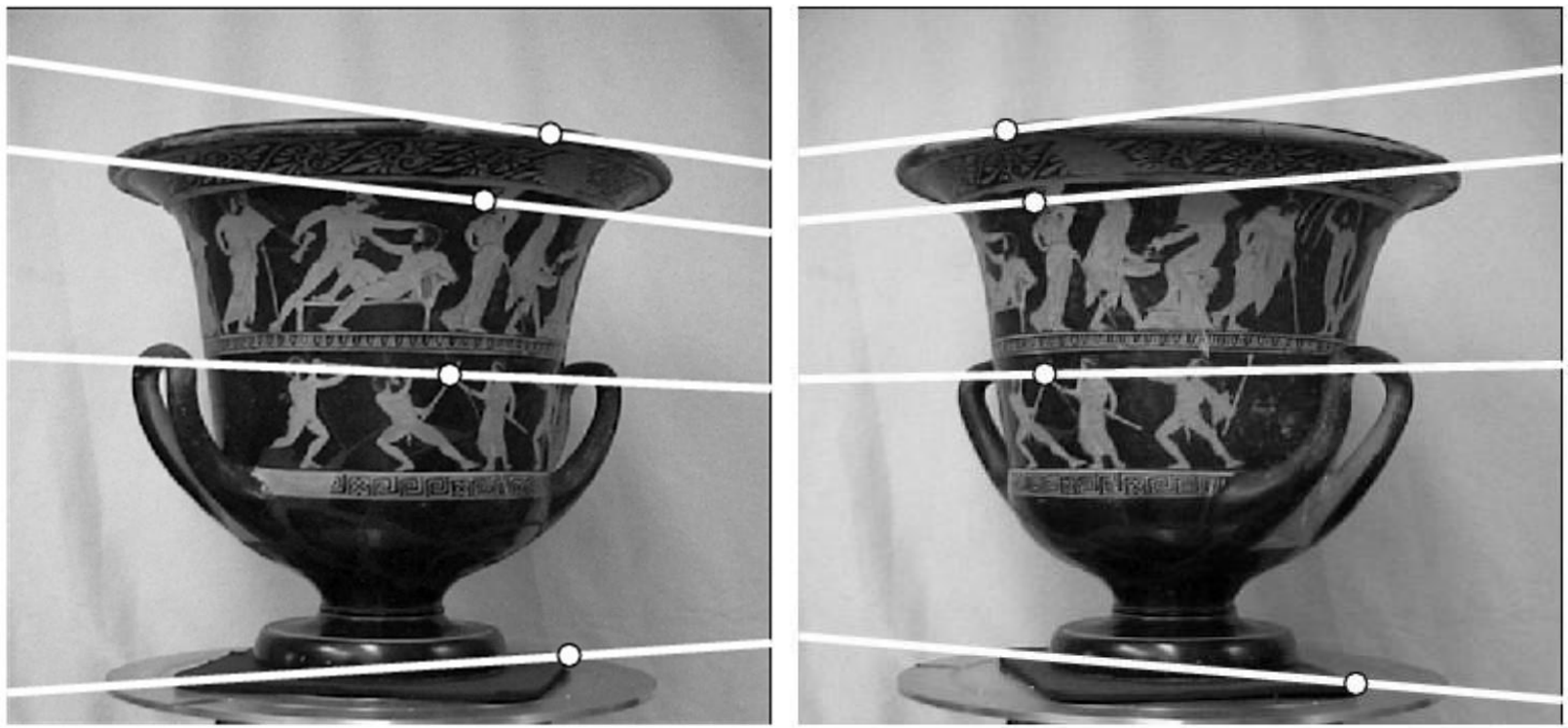


- **Baseline**: Line connecting the two camera centers
- **Epipoles**:
  - Intersections of baseline with image planes, or
  - Projections of the other camera center
- **Epipolar Plane**: Plane containing baseline
- **Epipolar Lines**: Intersections of epipolar plane with image planes (always come in corresponding pairs)

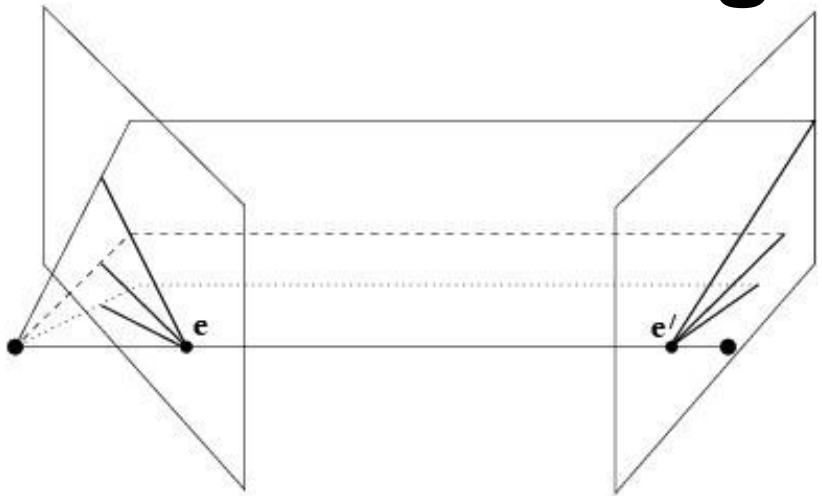
Example: Inward Facing Cameras

Where is the epipole in this image?

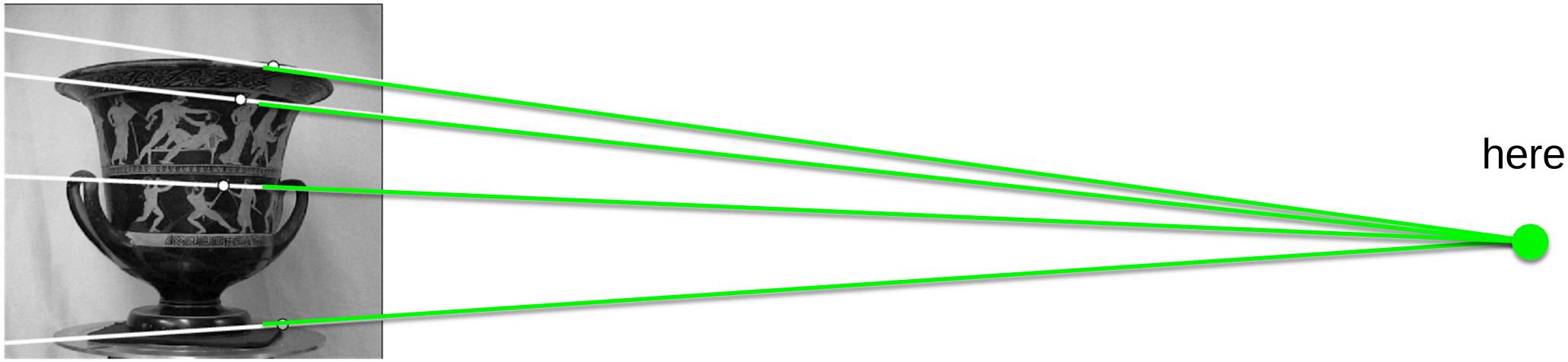




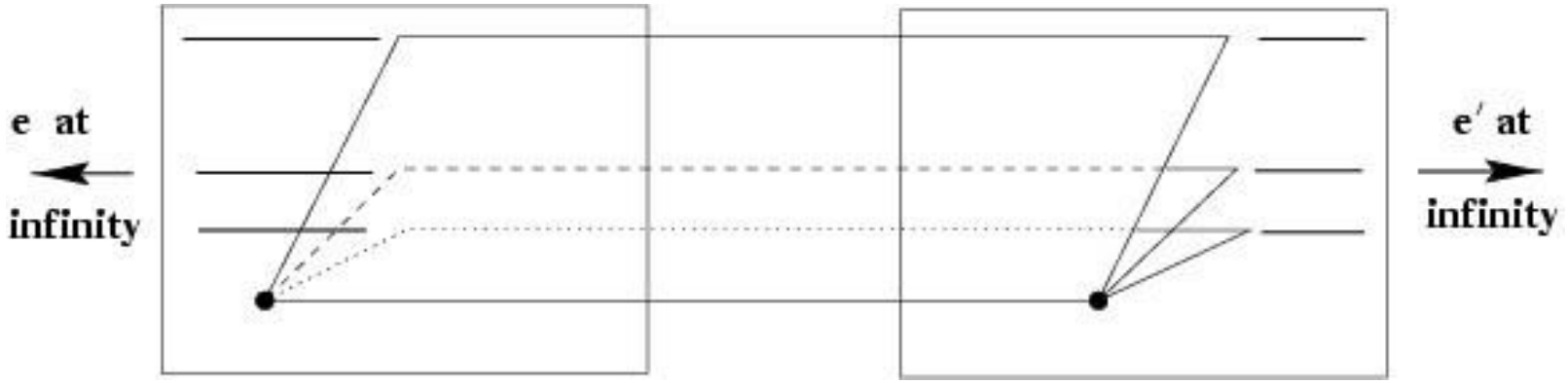
Example: Inward Facing Cameras



Where is the epipole in this image?

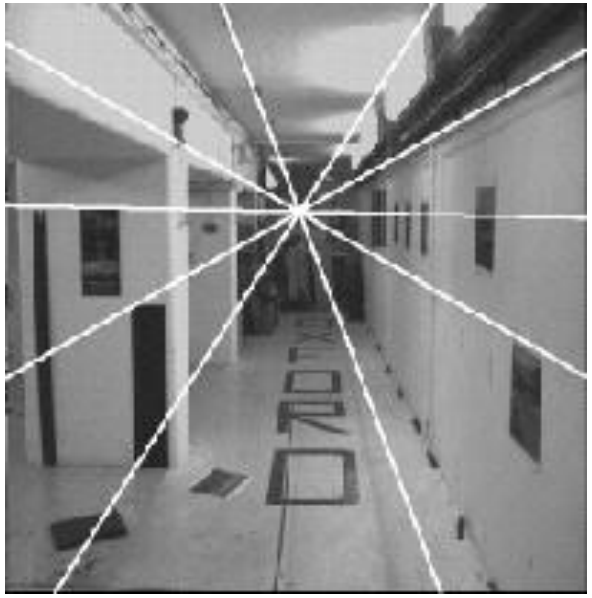


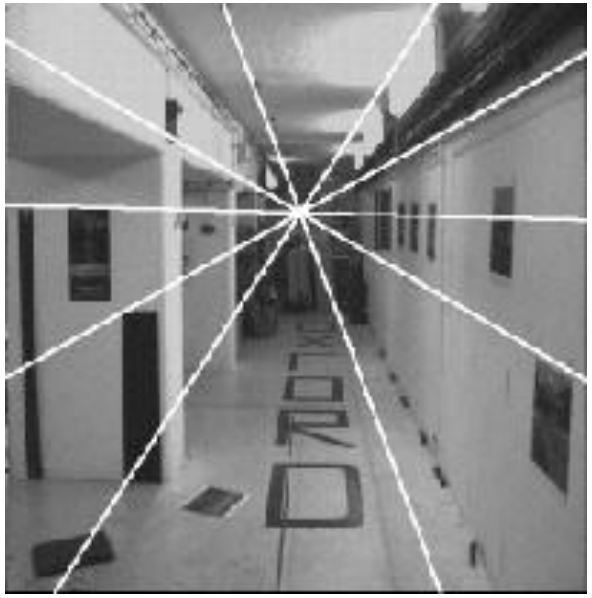
Example: Motion Parallel to Image Plane

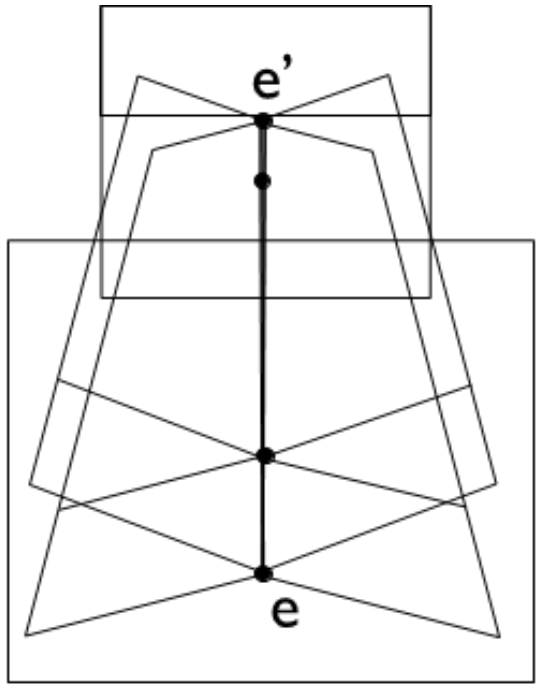


Where is the epipole?

Example: Forward Motion

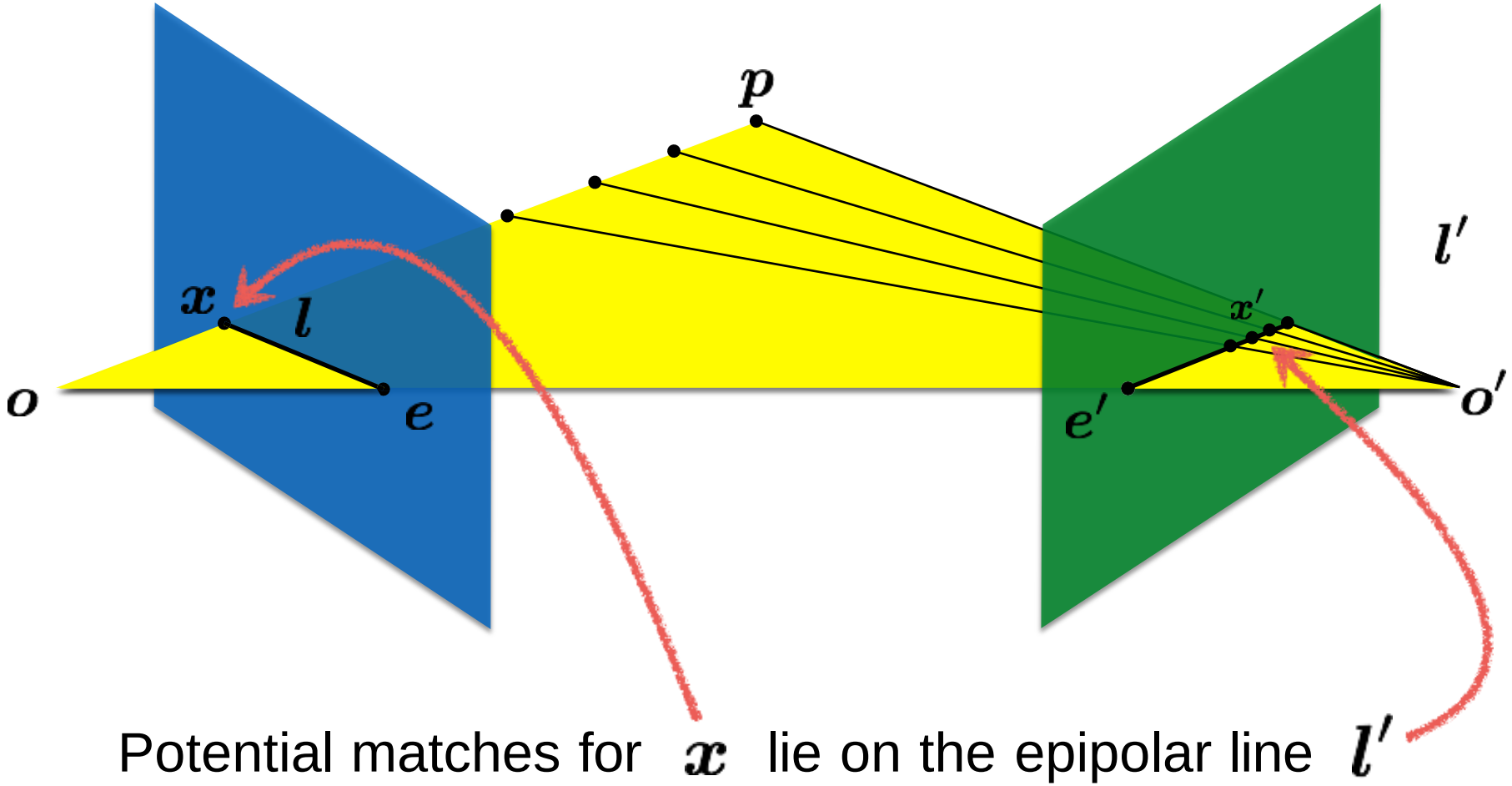






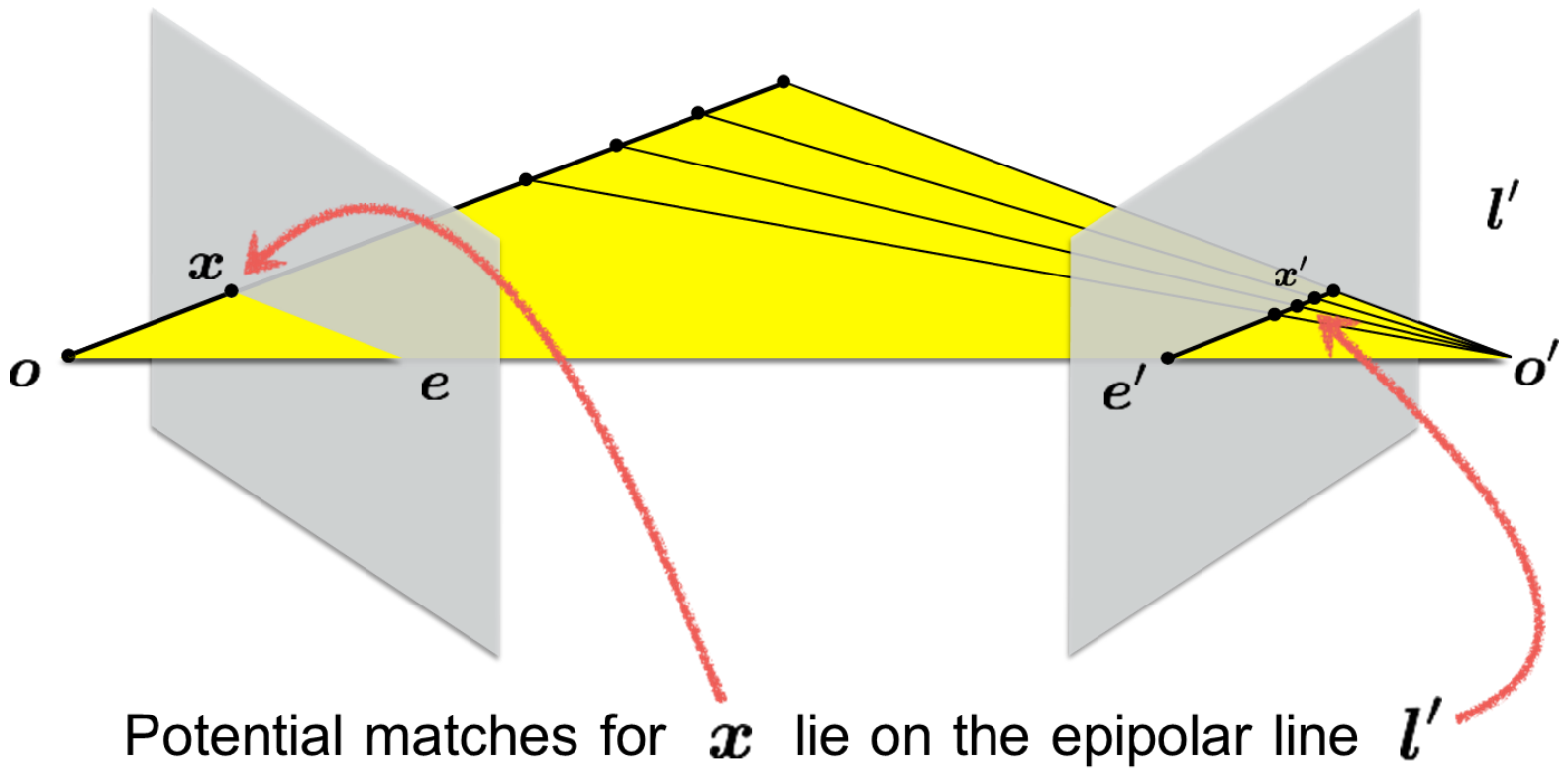
Epipole has same coordinates in both images. Points move along lines radiating from e:“Focus of expansion”

Epipolar Constraint



Potential matches for x lie on the epipolar line l'

What is this useful for?

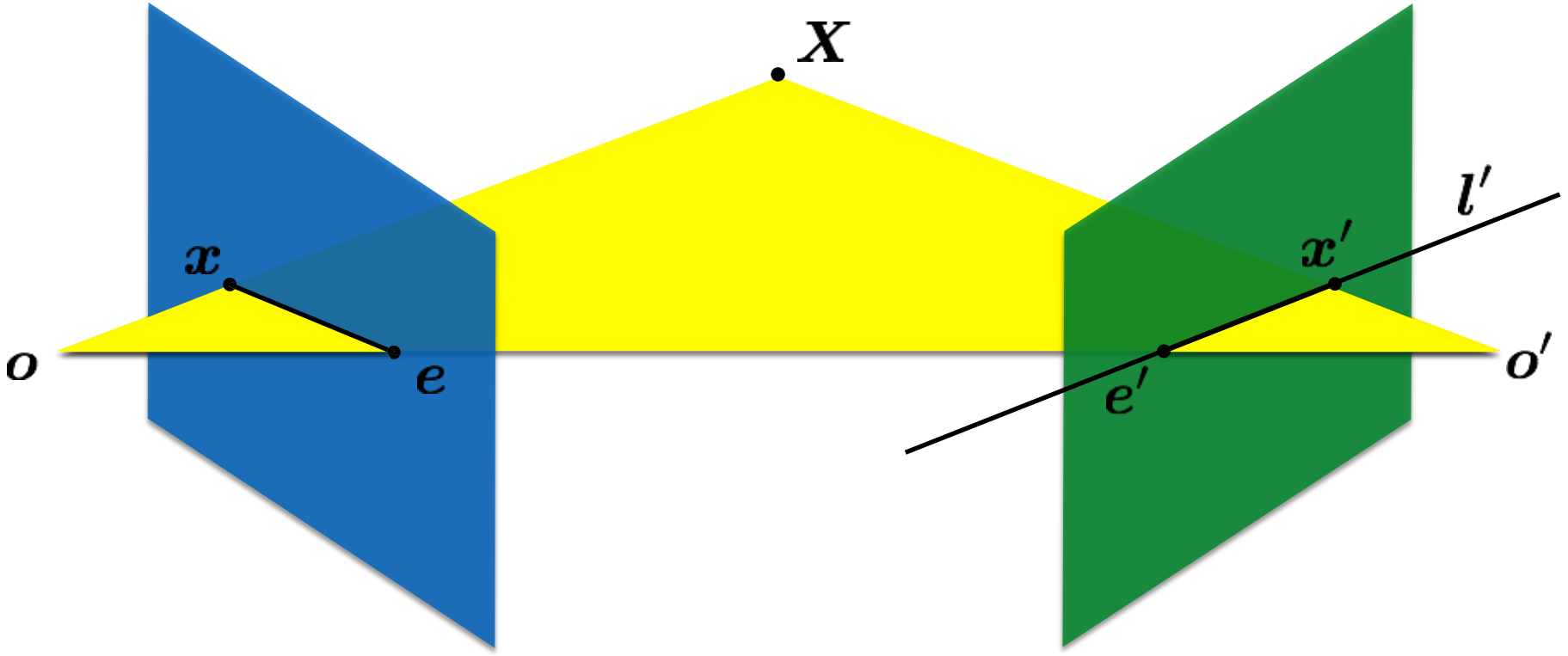


- If I know x, and have calibrated cameras (known intrinsics K, K' and extrinsic matrix), I can restrict x' to be along l'
- If we have enough x, x' correspondences,

Essential Matrix

Given a point in one image, multiplying by the **essential matrix** will tell us the **epipolar line** in the second view.

$$\mathbf{E}x = l'$$



Motivation

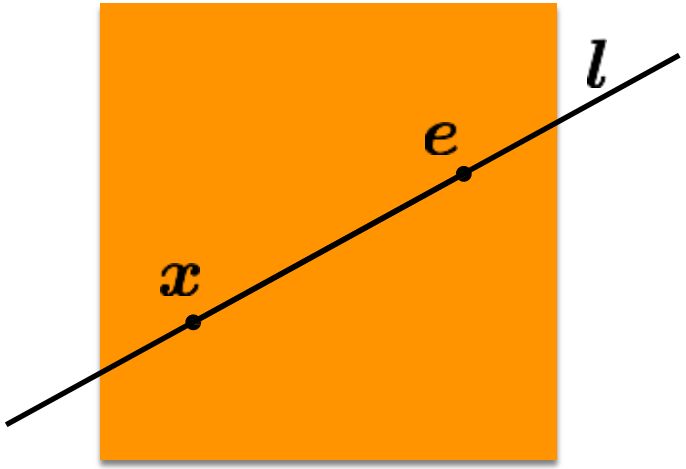
The Essential Matrix is a 3 x 3 matrix that encodes **epipolar geometry**

Given a point in one image, multiplying by the **essential matrix** will tell us the **epipolar line** in the second image.

We can estimate relative position & orientation between the cameras (and the 3D position of corresponding image points) using **essential matrix**

Epipolar Line

$$ax + by + c = 0$$ in vector form $$\boldsymbol{l} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

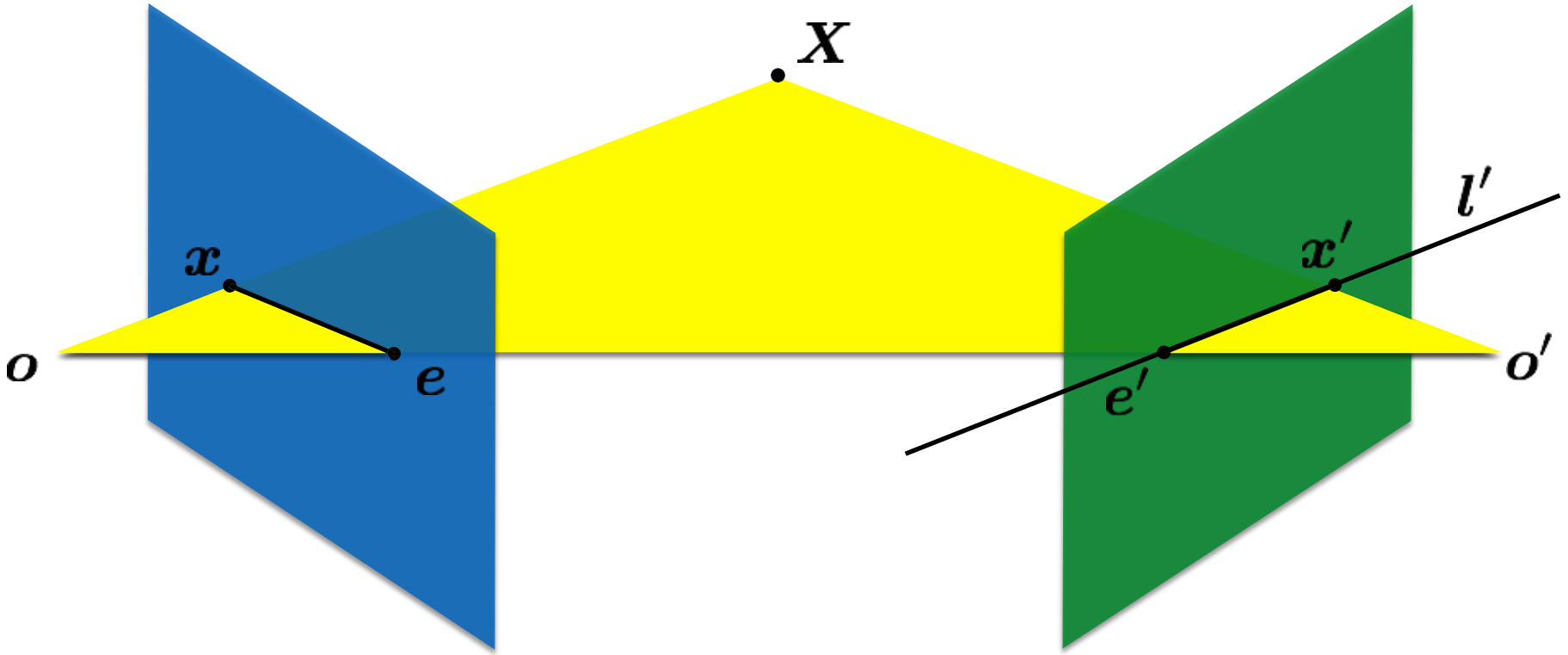


If the point $\boldsymbol{x}$ is on the epipolar line $\boldsymbol{l}$ then

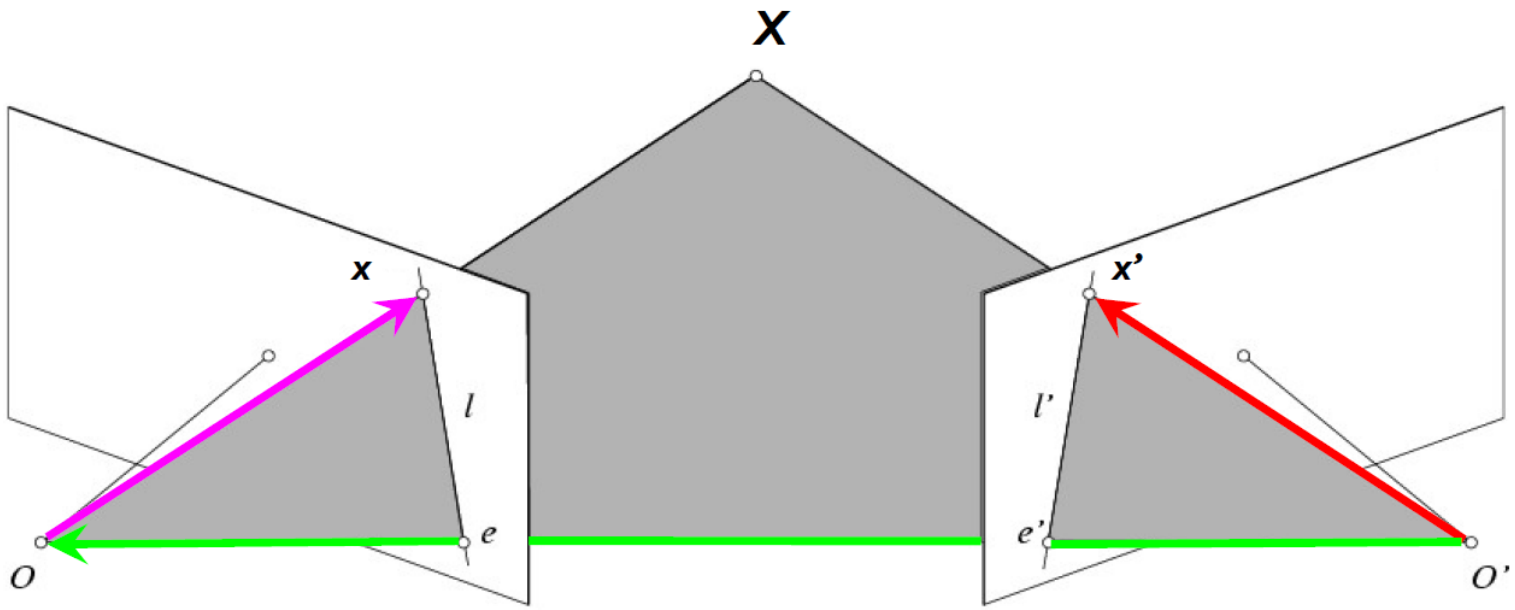
$$\boldsymbol{x}^\top \boldsymbol{l} = 0$$

So if $\boldsymbol{x'}^\top \boldsymbol{l'} = 0$ and $\mathbf{E}\boldsymbol{x} = \boldsymbol{l'}$ then

$$\boxed{\boldsymbol{x'}^\top \mathbf{E}\boldsymbol{x} = 0}$$



Essential Matrix



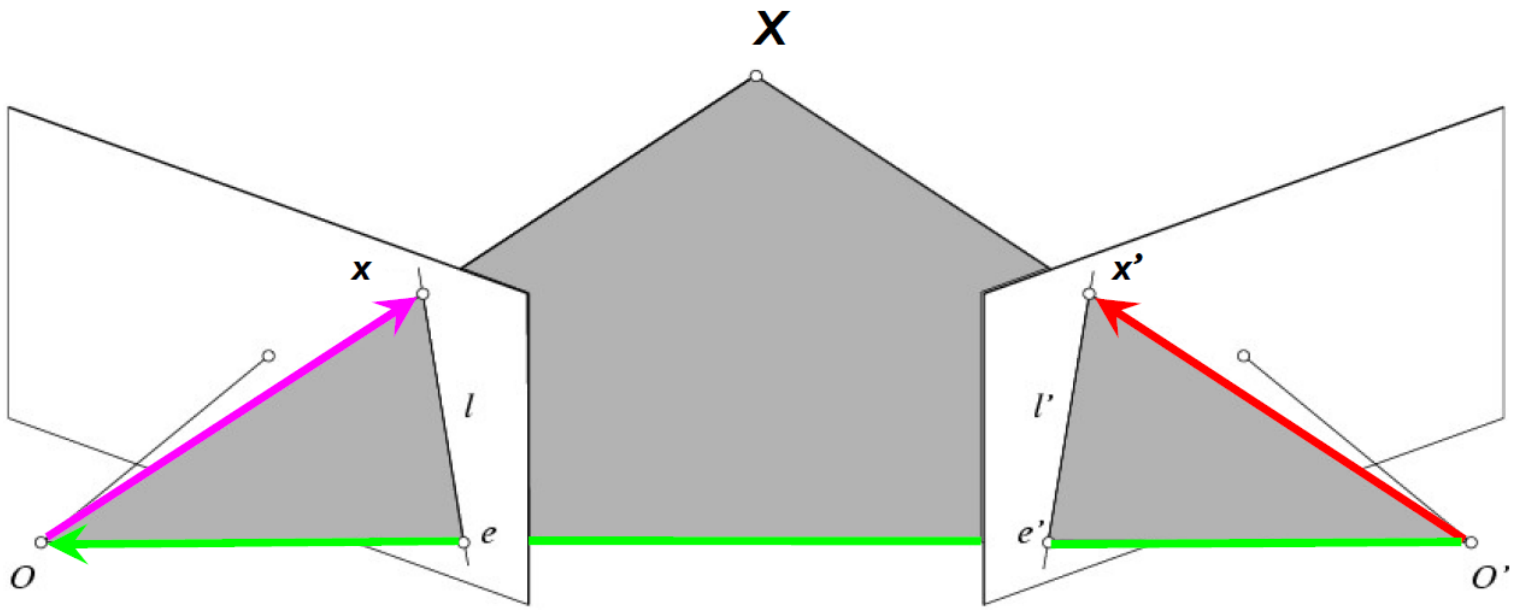
$$\hat{x} \cdot [t \times (R\hat{x}')] = 0 \quad \Rightarrow \quad \hat{x}^T E \hat{x}' = 0 \quad \text{with} \quad E = [t]_\times R$$

Essential Matrix
(Longuet-Higgins, 1981)

Note: $[t]_\times$ is matrix representation of cross product

- E is a 3x3 matrix which relates corresponding pairs of (homogeneous) image points across pairs of images (for calibrated cameras)
- Is used to estimates relative position/orientation

Additional Slide: Properties of the Essential matrix



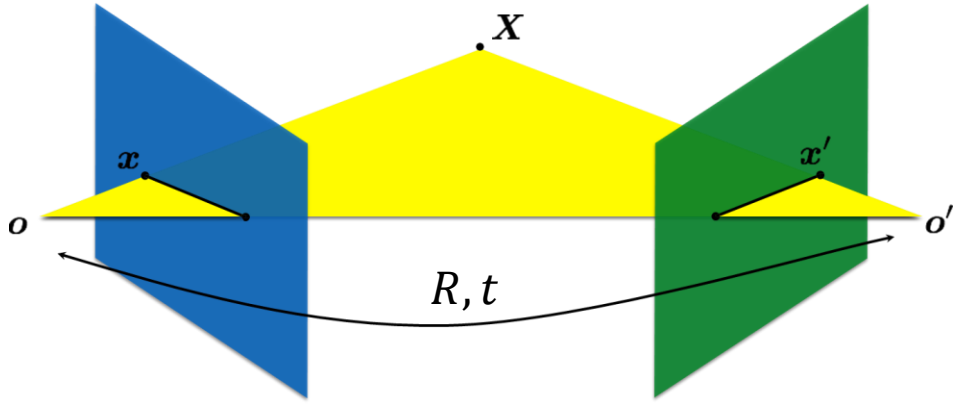
$$\hat{x} \cdot [t \times (R\hat{x}')] = 0 \quad \Rightarrow \quad \hat{x}^T E \hat{x}' = 0 \quad \text{with} \quad E = [t]_\times R$$

- $E\ x'$ 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)

Essential Matrix

Essential matrix is related to camera pose as

$$E = [t_\times]\, R$$



where (R, t) is the relative rotation & translation between camera frames, and the skew-symmetric matrix $[t_\times]$ is defined using the elements of translation vector $t = [t_x, t_y, t_z]^T$ as

$$[t_\times] = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$$

- Specifically, given the pose of camera 2 in camera 1 denoted by $({}^1R_2, {}^1t_2)$, the essential matrix constructed can be denoted as ${}^1E_2$, and the epipolar constraint is

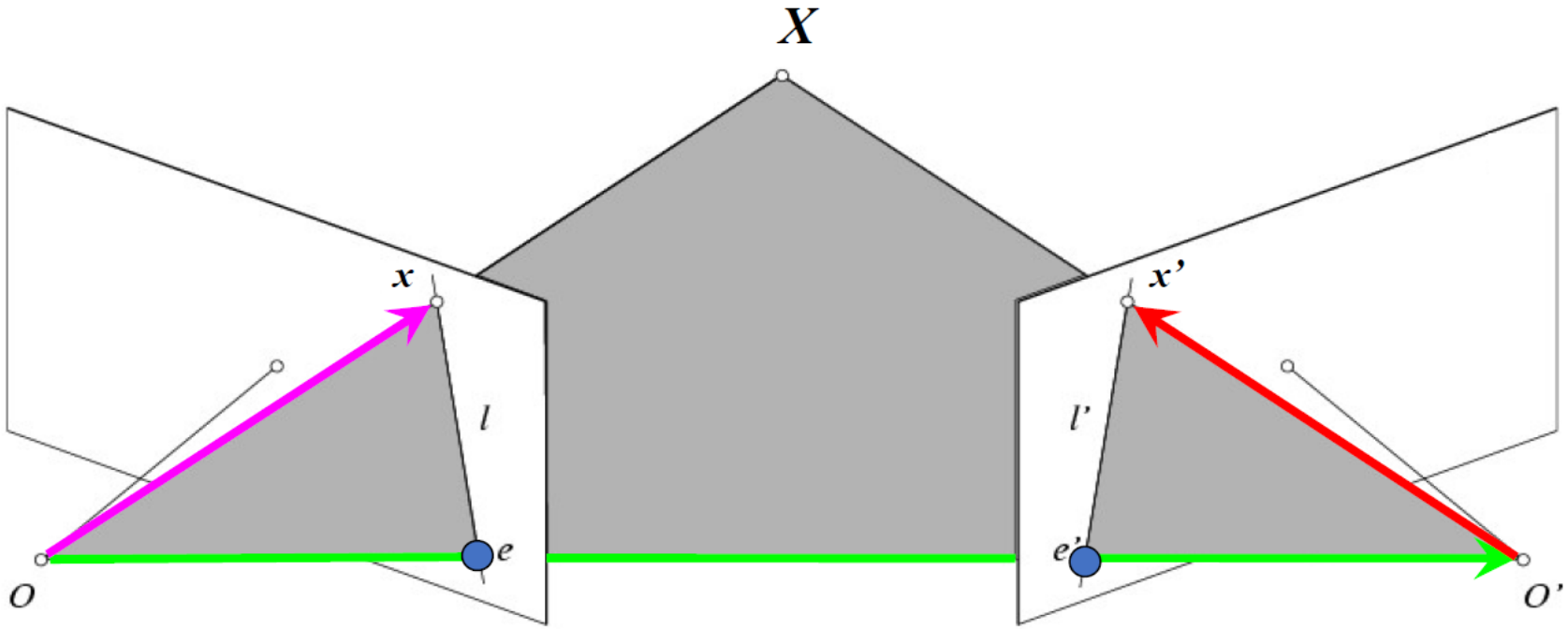
  $${}^1x^T\ {}^1E_2\ {}^2x\ = 0$$

  where ${}^1x$ and ${}^2x$ are **Cartesian** coordinates (*not* pixel coordinates) of image points in camera frames 1 and 2, respectively.

- On the other hand, using $({}^2R_1, {}^2t_1)$, we get ${}^2E_1$ and have ${}^2x^T\ {}^2E_1\ {}^1x\ = 0$.

- It holds that ${}^1E_2\ =\ {}^2E_1^T$

Epipolar Constraint: Uncalibrated Setting



If we don't know intrinsics K and K', then we can write the epipolar constraint in terms of unknown normalized coordinates:

$$\hat{x}^T E \hat{x}' = 0 \qquad x = K\hat{x}, \quad x' = K'\hat{x}'$$

The Fundamental Matrix

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

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

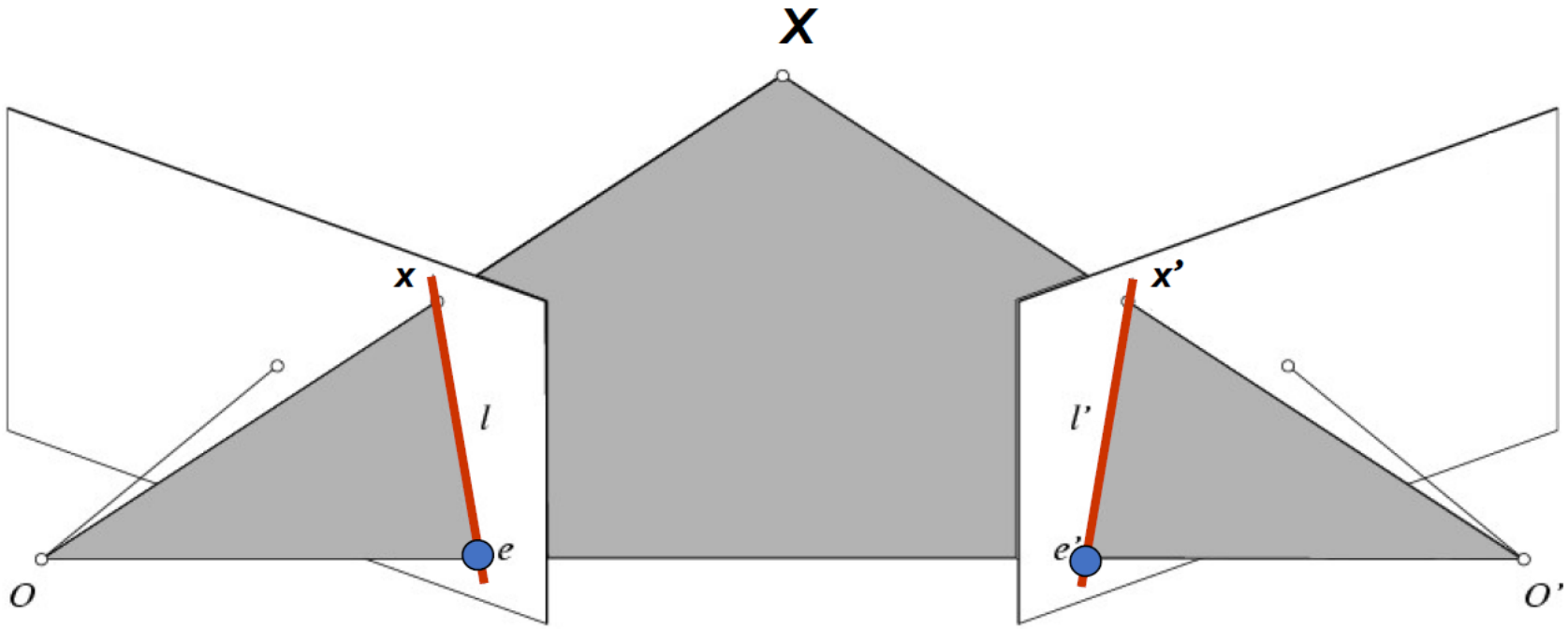
$$\hat{x} = K^{-1} x$$

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

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

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

F in more detail

- F is a 3x3 matrix
- F is Rank 2: one column is a linear combination of the other two

$$\begin{bmatrix} a & b & \alpha a + \beta b \\ c & d & \alpha c + \beta d \\ e & f & \alpha e + \beta f \end{bmatrix}$$

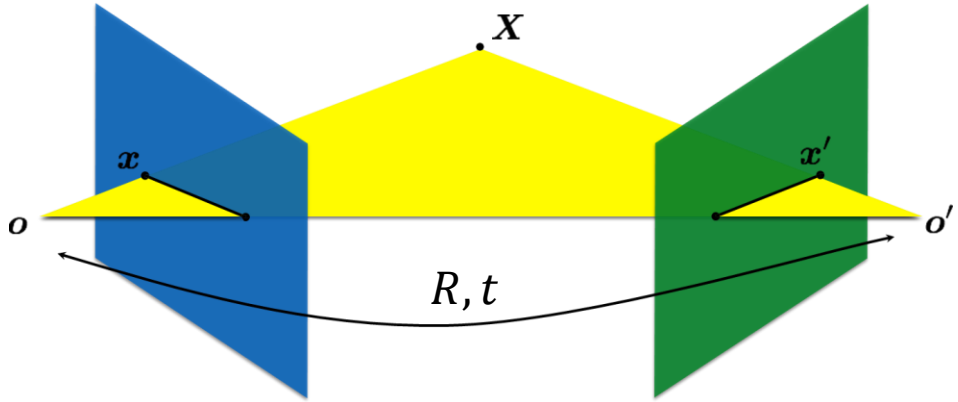
- Determined up to scale
- 7 degrees of freedom
- Given x projected from X into image 1, F constrains the projection of x’ into image 2 to an epipolar line

Fundamental Matrix

Fundamental matrix is related to camera pose as

$$F = K^{-1}[t_\times]\, R\, K^{-1}$$

where (R, t) is the relative rotation & translation between camera frames, and K is the camera calibration matrix.



- Similar to the essential matrix, given $({}^1R_2, {}^1t_2)$, the fundamental matrix constructed can be denoted as ${}^1F_2$, and the epipolar constraint is

$$ {}^1x^T\ {}^1F_2\ {}^2x\ = 0$$

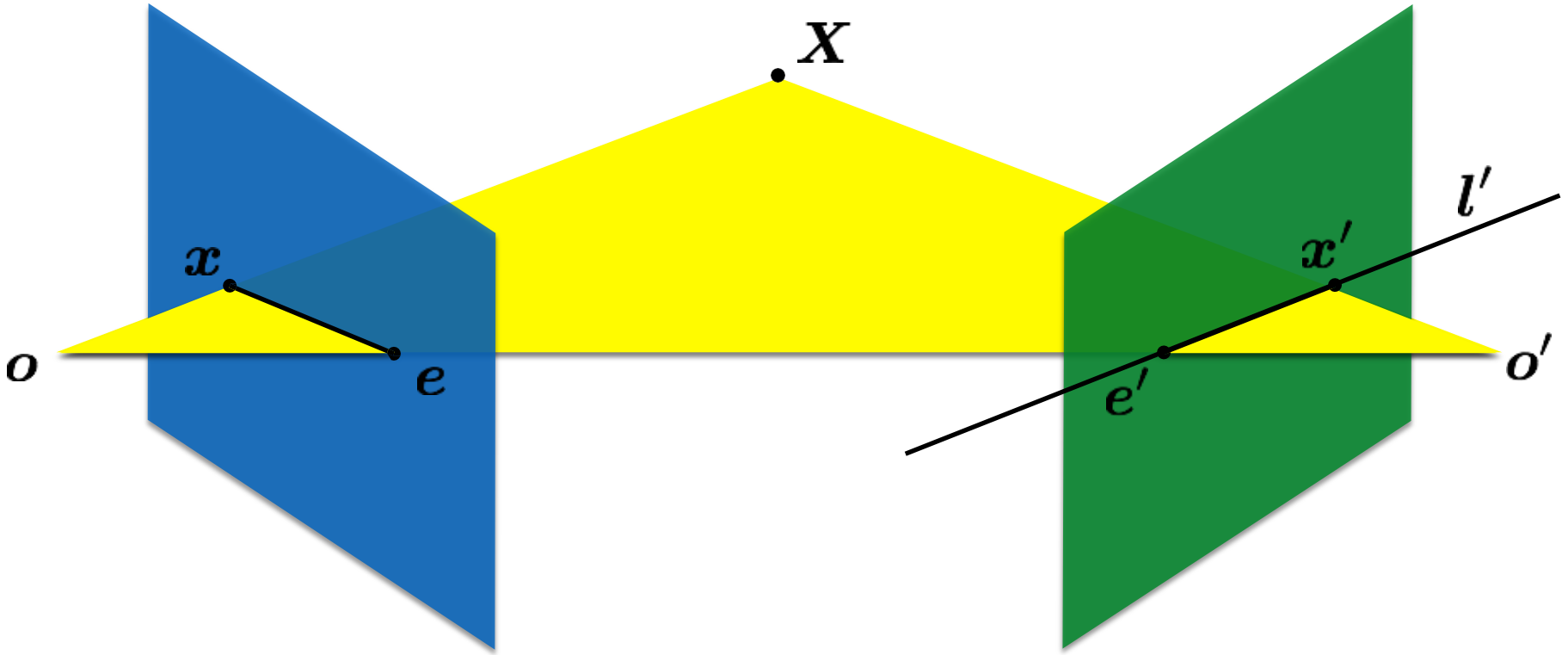
  where ${}^1x$ and ${}^2x$ are **pixel** coordinates (*not Cartesian* coordinates) of points in image 1 and 2, respectively.

- Using $({}^2R_1, {}^2t_1)$, we get ${}^2F_1$ and have ${}^2x^T\ {}^2F_1\ {}^1x\ = 0$.

- It holds that ${}^1F_2 = {}^2F_1^T$

Recap: Essential Matrix

$$\boldsymbol{x}'^{\top}\mathbf{E}\boldsymbol{x} = 0$$



Recap: Properties of Essential Matrix

Longuet-Higgins equation $\boldsymbol{x}'^{\top}\mathbf{E}\boldsymbol{x} = 0$

Epipolar lines $\boldsymbol{x}^{\top}\boldsymbol{l} = 0$ $\quad \boldsymbol{x}'^{\top}\boldsymbol{l}' = 0$

$\boldsymbol{l}' = \mathbf{E}\boldsymbol{x}$ $\quad \boldsymbol{l} = \mathbf{E}^{T}\boldsymbol{x}'$

Epipoles $\boldsymbol{e}'^{\top}\mathbf{E} = \mathbf{0}$ $\quad \mathbf{E}\boldsymbol{e} = \mathbf{0}$

2D points are expressed in camera coordinate system

Recap: Fundamental Matrix

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

$$\hat{x}'^{\top} \mathbf{E} \hat{x} = 0$$

The essential matrix operates on image points expressed in **2D coordinates** expressed in the camera coordinate system

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

$\hat{x}'$: camera point; x': image point

Writing out the epipolar constraint in terms of image coordinates

$$\mathbf{K}'^{-\top} \mathbf{E} \mathbf{K}^{-1} x = 0$$

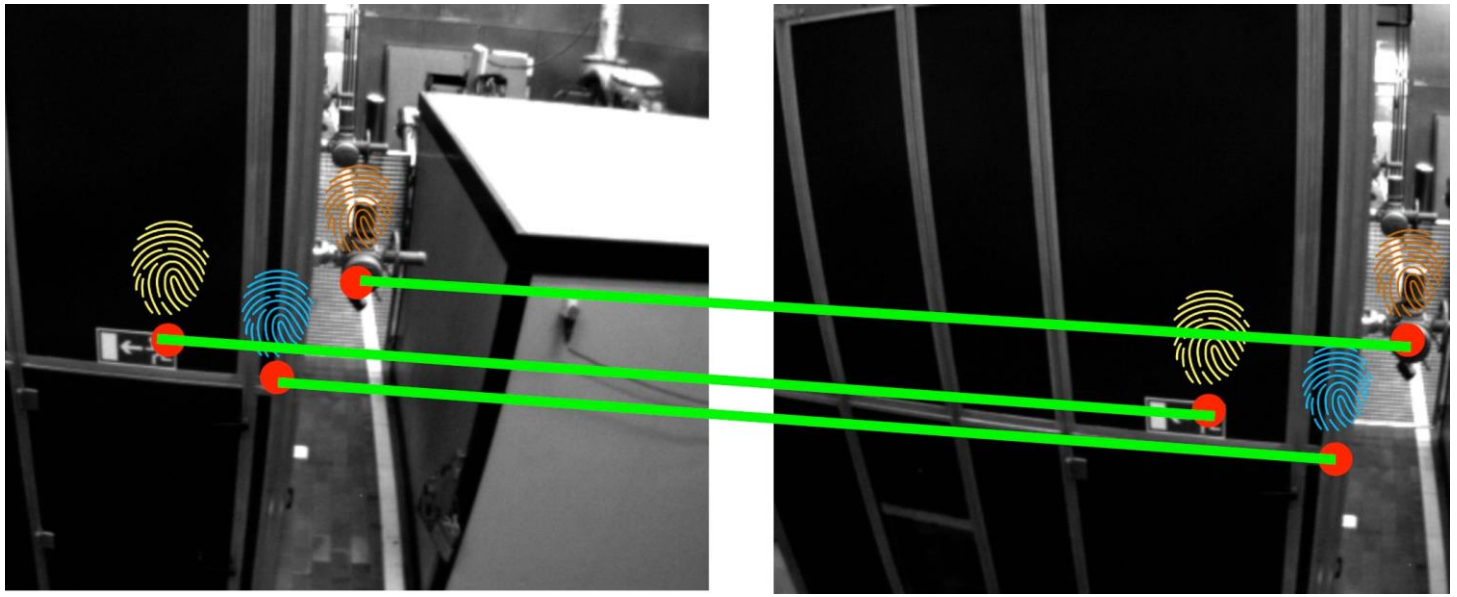
$$x'^{\top} (\mathbf{K}'^{-\top} \mathbf{E} \mathbf{K}^{-1}) x = 0$$

$$x'^{\top} \mathbf{F} x = 0$$

Fundamental Matrix

$$\boxed{\boldsymbol{x}_m'^{\top} \mathbf{F} \boldsymbol{x}_m = 0}$$

$$\begin{bmatrix} x'_m & y'_m & 1 \end{bmatrix} \begin{bmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{bmatrix} \begin{bmatrix} x_m \\ y_m \\ 1 \end{bmatrix} = 0$$



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

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

8-Point Algorithm

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

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

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

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

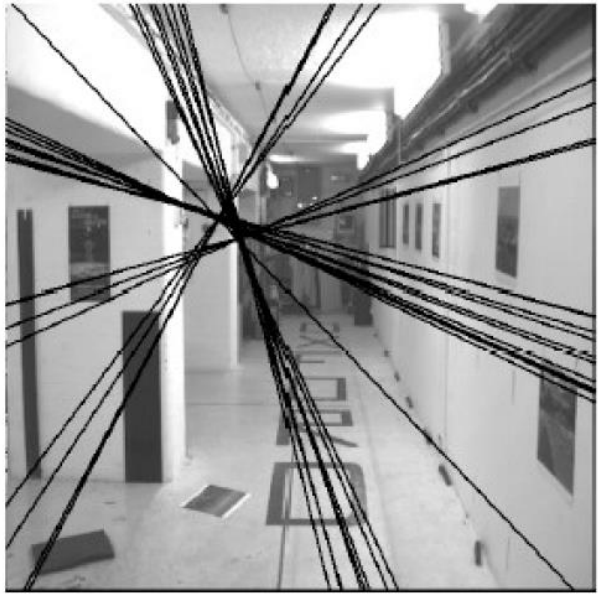
8-Point Algorithm

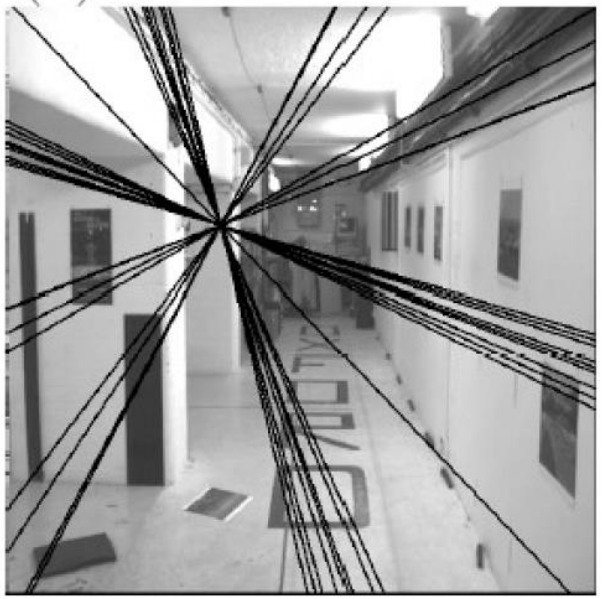
- Solve a system of homogeneous linear equations
  - Write down the system of equations
  - Solve f from $Af = 0$ using SVD

```
Python Numpy:
U, S, Vh = np.linalg.svd(A)
F = Vh[-1,:]
F = np.reshape(F, (3,3))
```

8-Point Algorithm

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





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

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

8-Point Algorithm

- Solve a system of homogeneous linear equations
  - Write down the system of equations
  - Solve f from $Af = 0$ using SVD

```
Python Numpy:
U, S, Vh = np.linalg.svd(A)
F = Vh[-1,:]
F = np.reshape(F, (3,3))
```

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

```
Python Numpy:
U, S, Vh = np.linalg.svd(F)
S[-1] = 0
F = U @ np.diagflat(S) @ Vh
```

@ operator = matrix multiplication

Decompose E to get R, t

SVD: $\mathbf{E} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$

Let $\mathbf{W} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

We get FOUR solutions:

$\mathbf{R}_1 = \mathbf{U}\mathbf{W}\mathbf{V}^\top \quad \mathbf{R}_2 = \mathbf{U}\mathbf{W}^\top\mathbf{V}^\top$

two possible rotations

$\mathbf{T}_1 = U_3 \quad \mathbf{T}_2 = -U_3$

two possible translations

We get FOUR solutions

$$\mathbf{R}_1 = \mathbf{U}\mathbf{W}\mathbf{V}^\top \quad \mathbf{T}_1 = U_3$$

$$\mathbf{R}_1 = \mathbf{U}\mathbf{W}\mathbf{V}^\top \quad \mathbf{T}_2 = -U_3$$

$$\mathbf{R}_2 = \mathbf{U}\mathbf{W}^\top\mathbf{V}^\top \quad \mathbf{T}_2 = -U_3$$

$$\mathbf{R}_2 = \mathbf{U}\mathbf{W}^\top\mathbf{V}^\top \quad \mathbf{T}_1 = U_3$$

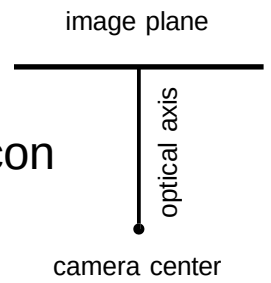
Which one do we choose? Chirality constraints:

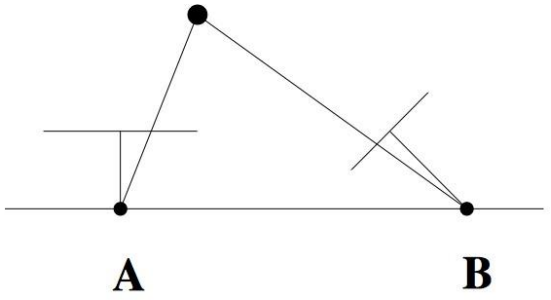
- Compute determinant of R, valid solution must be equal to 1
  (note: det(R) = -1 means rotation and reflection)
- Compute 3D point using triangulation, valid solution has positive depth value
  (Note: negative depth means point is behind the camera)

Correct Solution

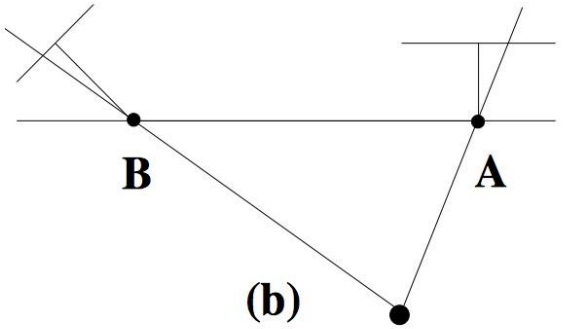
The configuration where Points are in front of cameras

Camera Icon

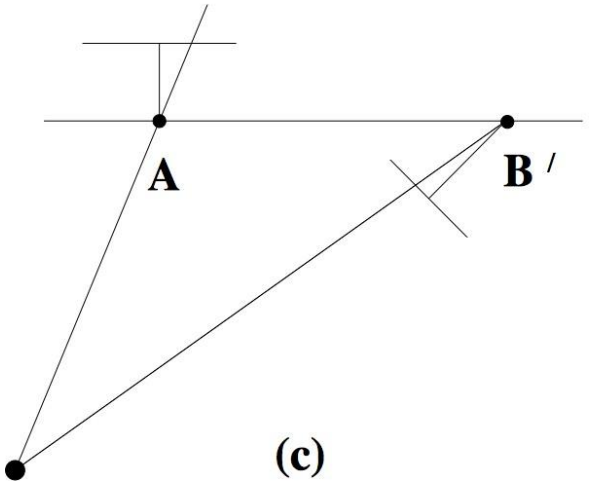




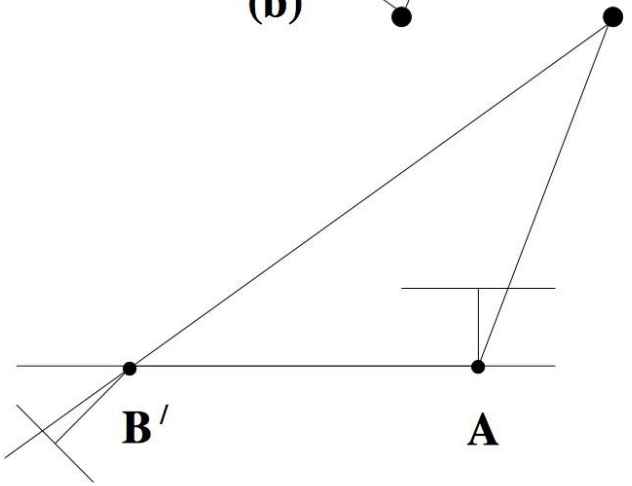
(a)



(b)



(c)

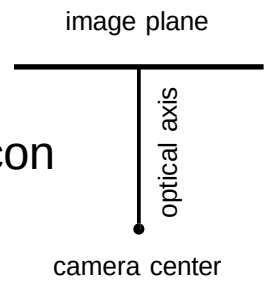


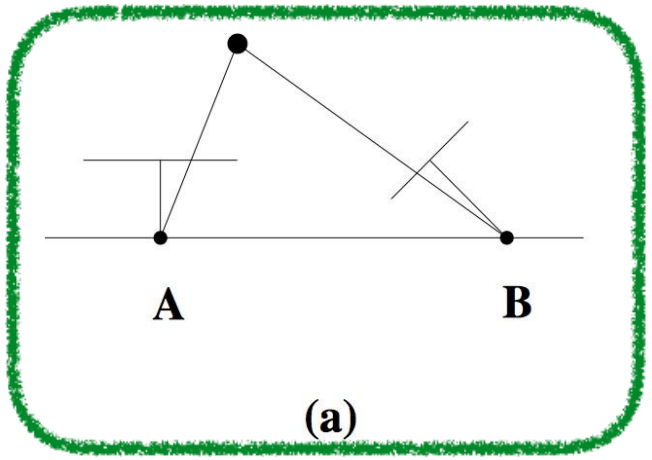
(d)

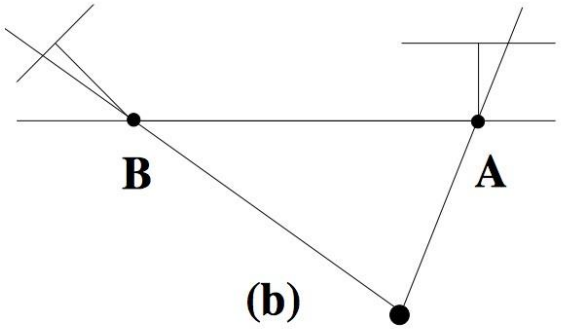
Correct Solution

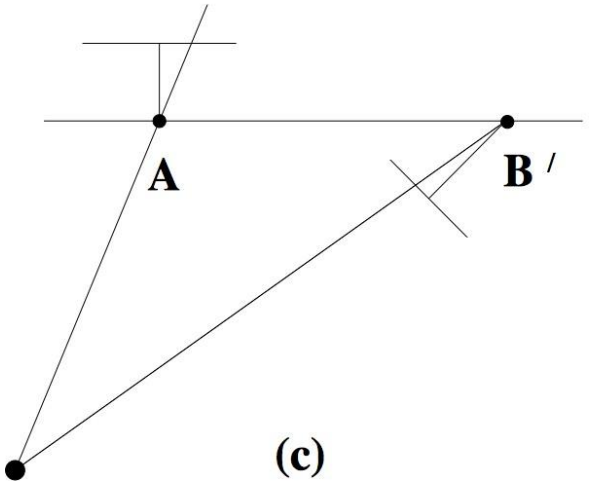
The configuration where Points are in front of cameras

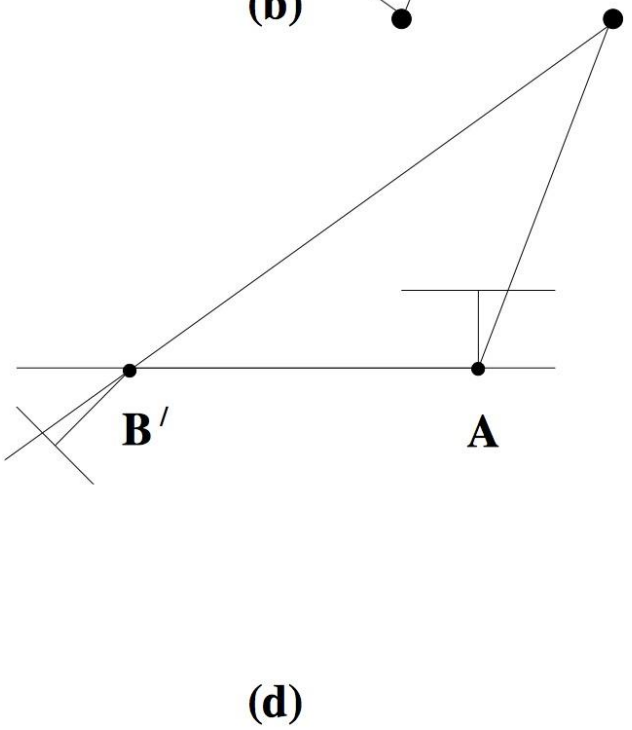
Camera Icon





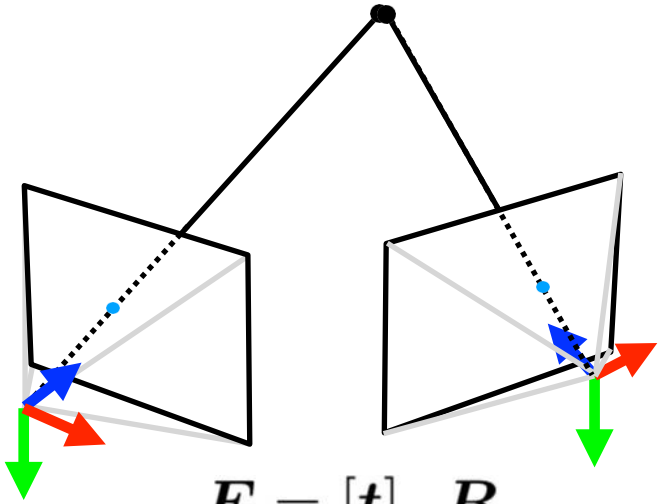




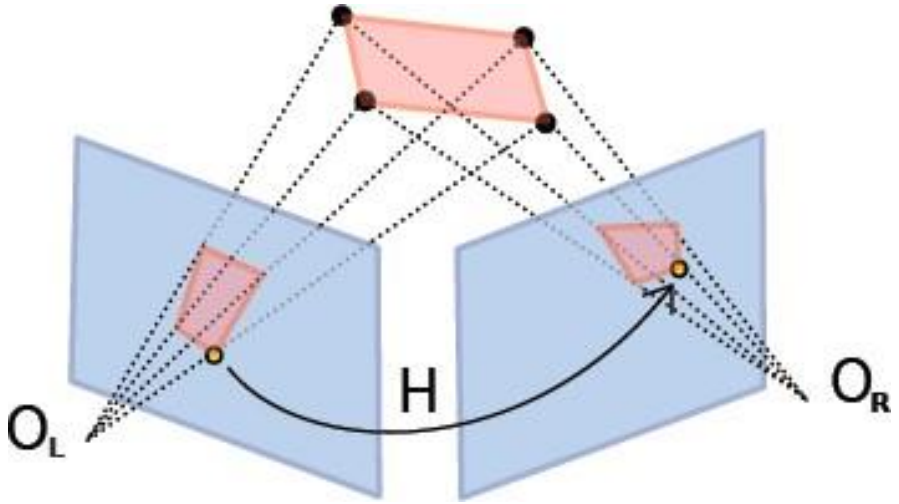


8-point Method: Limitations

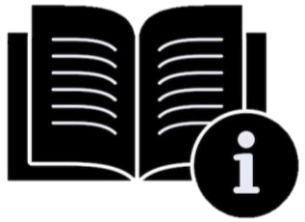
Number of correspondences:
do we really need 8 points?



Scene structures:
There are certain configurations of
3D points that make the algorithm fail

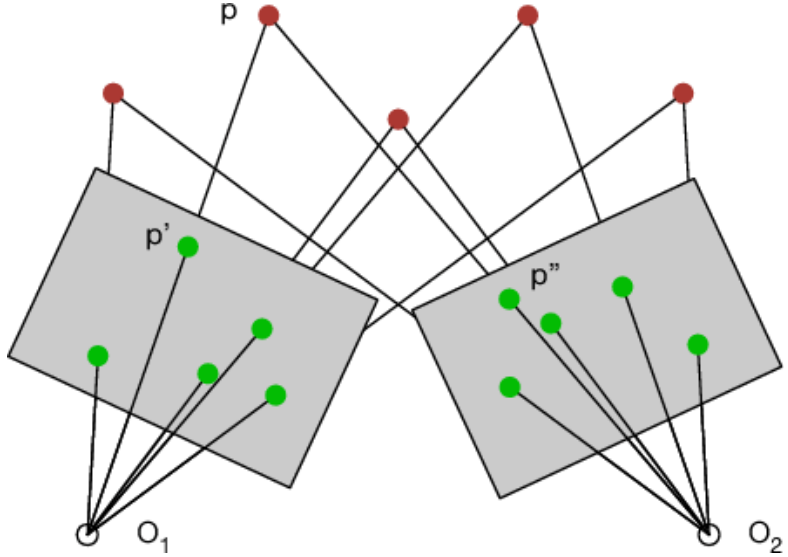


Parallax: what if ***t* = 0**?



5-Point Algorithm

- Proposed by Nistér in 2003/2004
- Standard solution today to obtaining the direct solution
- Solving a polynomial of degree 10
- 10 possible solutions
- Often used together RANSAC
  - RANSAC proposes correspondences
  - Evaluate all 5-point solutions based on the other corresponding points



- Stewenius, Engels, Nistér: "Recent Developments on Direct Relative Orientation", ISPRS 2006
- Li and Hartley: "Five-Point Motion Estimation Made Easy"

Homography

We can use homographies when…

- The scene is planar

- The scene is very far or has small (relative) depth variation → scene is approximately planar

- The scene is captured under camera rotation only (no translation)

Panorama: stitching images from different viewpoints

Use image homographies:

Homography Matrix

Given matched image points (P, P'), the homography matrix H relates

$$P' = H \cdot P$$

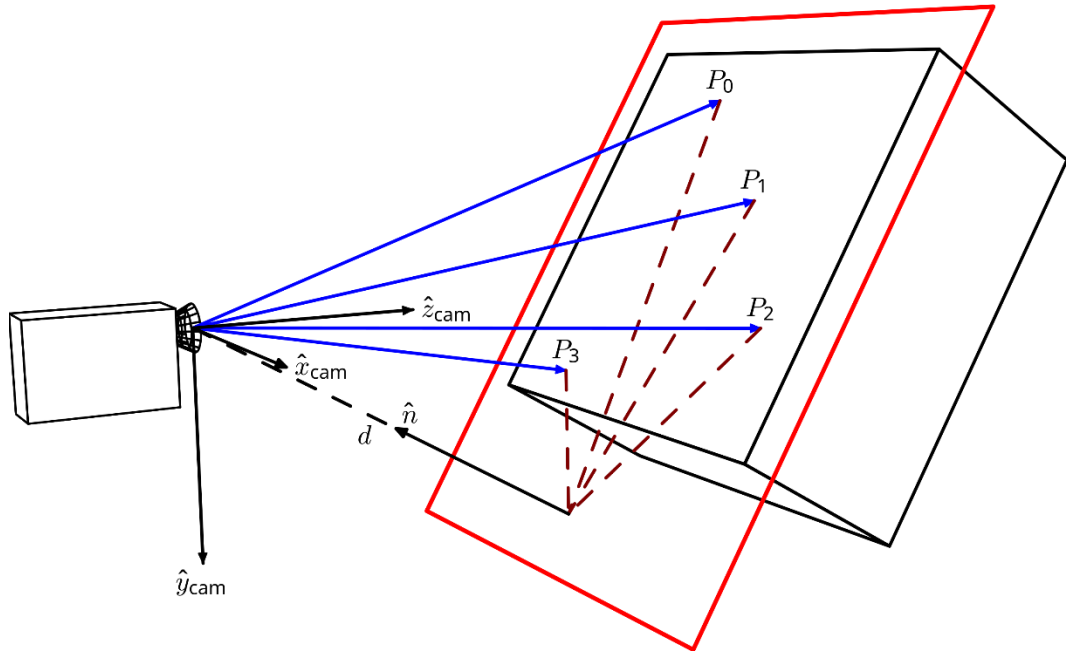
Note: Homography holds only for co-planar points or rotation-only motion

Homography Matrix

Homography matrix is related to camera relative pose via

$$^1H_2 = {}^1R_2 - \frac{^1t_2 \,.\, ^2n^T}{^2d}$$

Where (R, t) is the relative pose, n is the normal vector of the plane, and d is the distance from the origin of camera frame to the plane.



Using $(^1R_2, ^1t_2)$, the homography matrix can be denoted as $^1H_2$, and the homography constraint is

$$^1x = {}^1H_2 \, ^2x$$

Note that in this case n and d must be computed in camera 1 frame. Point ^1x and ^2x are **Cartesian** coordinates (not pixels) in camera frames.

On the other hand, using $(^2R_1, ^2t_1), ^2n, ^2d$ will yield $^2H_1$, and therefore we have $^2x = {}^2H_1 \, ^1x$
It holds that $^1H_2 = {}^2H_1^{-1}$

Estimating the homography matrix

Write out linear equation for each correspondence:

$$P' = H \cdot P \quad \text{or} \quad \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \alpha \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

(Homogeneous coordinates, so up to a scale factor)

Estimating the homography matrix

Write out linear equation for each correspondence:

$$P' = H \cdot P \quad \text{or} \quad \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \alpha \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Expand matrix multiplication:

$$\begin{aligned} x' &= \alpha(h_1 x + h_2 y + h_3) \\ y' &= \alpha(h_4 x + h_5 y + h_6) \\ 1 &= \alpha(h_7 x + h_8 y + h_9) \end{aligned}$$

Estimating the homography matrix

Write out linear equation for each correspondence:

$$P' = H \cdot P \quad \text{or} \quad \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \alpha \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Expand matrix multiplication:

$$\begin{aligned} x' &= \alpha(h_1 x + h_2 y + h_3) \\ y' &= \alpha(h_4 x + h_5 y + h_6) \\ 1 &= \alpha(h_7 x + h_8 y + h_9) \end{aligned}$$

Divide out unknown scale factor:

$$\begin{aligned} x'(h_7 x + h_8 y + h_9) &= (h_1 x + h_2 y + h_3) \\ y'(h_7 x + h_8 y + h_9) &= (h_4 x + h_5 y + h_6) \end{aligned}$$

How do you rearrange terms to make it a linear system?

Estimating the homography matrix

$$x'(h_7x + h_8y + h_9) = (h_1x + h_2y + h_3)$$
$$y'(h_7x + h_8y + h_9) = (h_4x + h_5y + h_6)$$

Rearrange the terms

$$h_7xx' + h_8yx' + h_9x' - h_1x - h_2y - h_3 = 0$$
$$h_7xy' + h_8yy' + h_9y' - h_4x - h_5y - h_6 = 0$$

Estimating the homography matrix

Re-arrange terms:

$$h_7 x x' + h_8 y x' + h_9 x' - h_1 x - h_2 y - h_3 = 0$$
$$h_7 x y' + h_8 y y' + h_9 y' - h_4 x - h_5 y - h_6 = 0$$

Re-write in matrix form:

$$\mathbf{A}_i \boldsymbol{h} = \mathbf{0}$$

$$\mathbf{A}_i = \begin{bmatrix} -x & -y & -1 & 0 & 0 & 0 & xx' & yx' & x' \\ 0 & 0 & 0 & -x & -y & -1 & xy' & yy' & y' \end{bmatrix}$$

$$\boldsymbol{h} = \begin{bmatrix} h_1 & h_2 & h_3 & h_4 & h_5 & h_6 & h_7 & h_8 & h_9 \end{bmatrix}^\top$$

How many equations from one point correspondence?

Estimating the Homography Matrix

Stack together constraints from multiple point correspondences

$$\mathbf{A}h = 0$$

$$\begin{bmatrix} -x & -y & -1 & 0 & 0 & 0 & xx' & yx' & x' \\ 0 & 0 & 0 & -x & -y & -1 & xy' & yy' & y' \end{bmatrix}$$

$$\begin{bmatrix} -x & -y & -1 & 0 & 0 & 0 & xx' & yx' & x' \\ 0 & 0 & 0 & -x & -y & -1 & xy' & yy' & y' \end{bmatrix}$$

$$\vdots$$

$$\begin{bmatrix} -x & -y & -1 & 0 & 0 & 0 & xx' & yx' & x' \\ 0 & 0 & 0 & -x & -y & -1 & xy' & yy' & y' \end{bmatrix}$$

$$\begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Estimating the Homography Matrix

- **Direct Linear Transform (DLT)** algorithm
  - Create matrix $\boldsymbol{A}$ from point correspondences
  - Compute SVD: $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathbf{T}}$

```
Python Numpy:
U, S, Vh = np.linalg.svd(A)
```

  - Store singular vector of the smallest singular value $\boldsymbol{h} = \boldsymbol{v_i}$
  - Reshape to get $\boldsymbol{H}$

```
h = Vh[-1,:]
H = np.reshape(h, (3,3))
```

Decomposing Homography Matrix

The decomposition of $H = R - \frac{t\, n^T}{d}$ into R, t, n:

- SVD: $H = U\, \Sigma\, V^T$
- $R = U\, V^T$
- Let $M = H\ - R$
- SVD: $M = U_M\, \Sigma_M\, V_M^T$
- Let u, v be the left and right singular vectors, and σ be the corresponding singular value in SVD of M
- $t = \sigma d u,\ \ n = v$

Decomposing Homography Matrix

Example:

$$H = \begin{bmatrix} 1.1 & -0.2 & 0.3 \\ 0.1 & 0.9 & -0.1 \\ 0.05 & 0.1 & 1.0 \end{bmatrix}$$

$$H = U\Sigma V^T \Rightarrow U = \begin{bmatrix} -0.89 & 0.45 & -0.06 \\ -0.35 & -0.76 & -0.55 \\ -0.30 & -0.47 & 0.83 \end{bmatrix}, \quad \Sigma = \begin{bmatrix} 1.23 & 0 & 0 \\ 0 & 0.91 & 0 \\ 0 & 0 & 0.78 \end{bmatrix}, \quad V^T = \begin{bmatrix} -0.92 & 0.38 & -0.06 \\ -0.38 & -0.91 & -0.15 \\ -0.06 & -0.15 & 0.99 \end{bmatrix}$$

$$R = UV^T \Rightarrow R = \begin{bmatrix} 0.99 & -0.10 & 0.09 \\ 0.10 & 0.99 & -0.03 \\ -0.09 & 0.04 & 0.99 \end{bmatrix}$$

$$M = H - R = \begin{bmatrix} 0.11 & -0.10 & 0.21 \\ 0.00 & -0.09 & -0.07 \\ 0.14 & 0.06 & 0.01 \end{bmatrix}$$

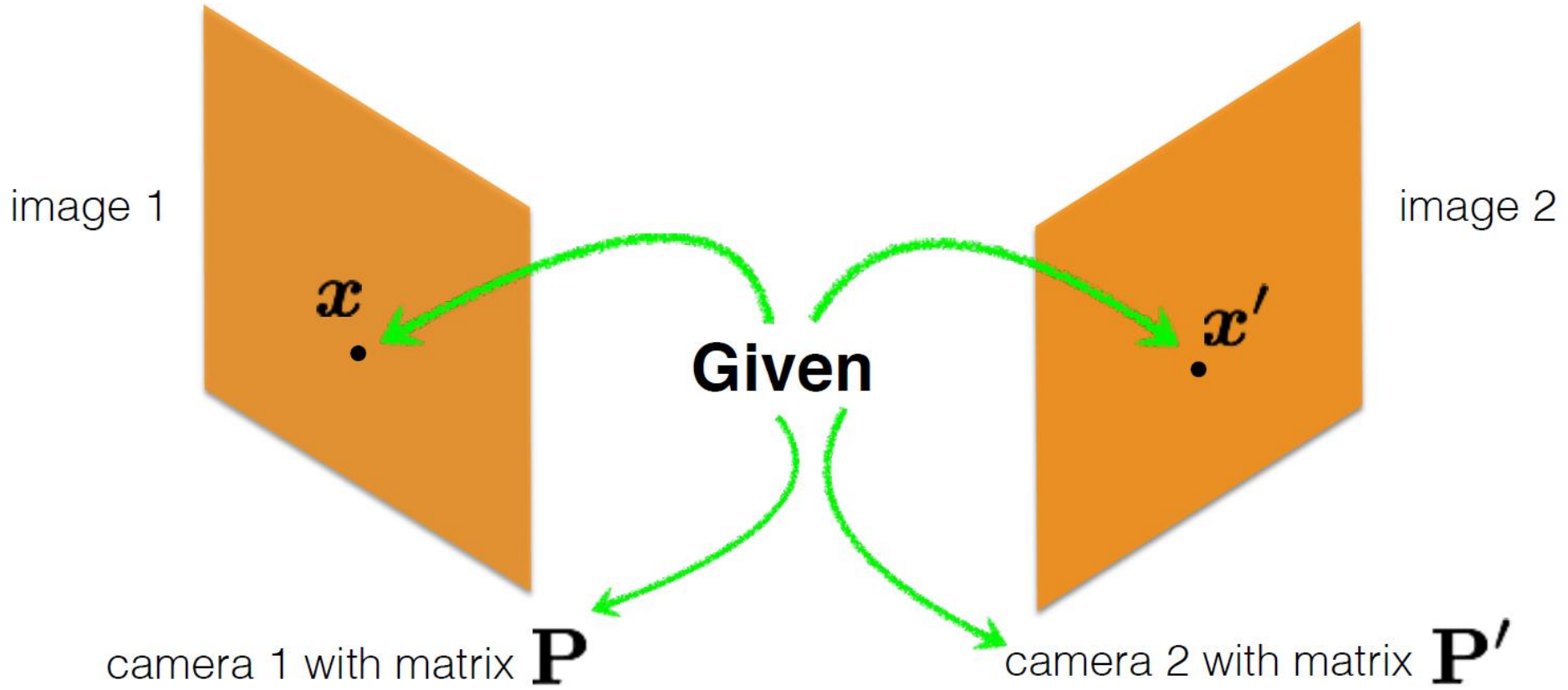
$$M = U_M \Sigma_M V_M^T \Rightarrow U_M = \begin{bmatrix} -0.82 & -0.57 & -0.07 \\ 0.30 & -0.52 & 0.80 \\ -0.49 & 0.64 & 0.59 \end{bmatrix}, \quad \Sigma_M = \begin{bmatrix} 0.29 & 0 & 0 \\ 0 & 0.12 & 0 \\ 0 & 0 & 0.01 \end{bmatrix}, \quad V_M^T = \begin{bmatrix} -0.47 & 0.87 & -0.16 \\ -0.88 & -0.46 & 0.12 \\ 0.05 & 0.19 & 0.98 \end{bmatrix}$$

$$\mathbf{t} = \sigma_1 \mathbf{u}_1 = 0.29 \begin{bmatrix} -0.82 \\ 0.30 \\ -0.49 \end{bmatrix} = \begin{bmatrix} -0.24 \\ 0.09 \\ -0.14 \end{bmatrix}$$

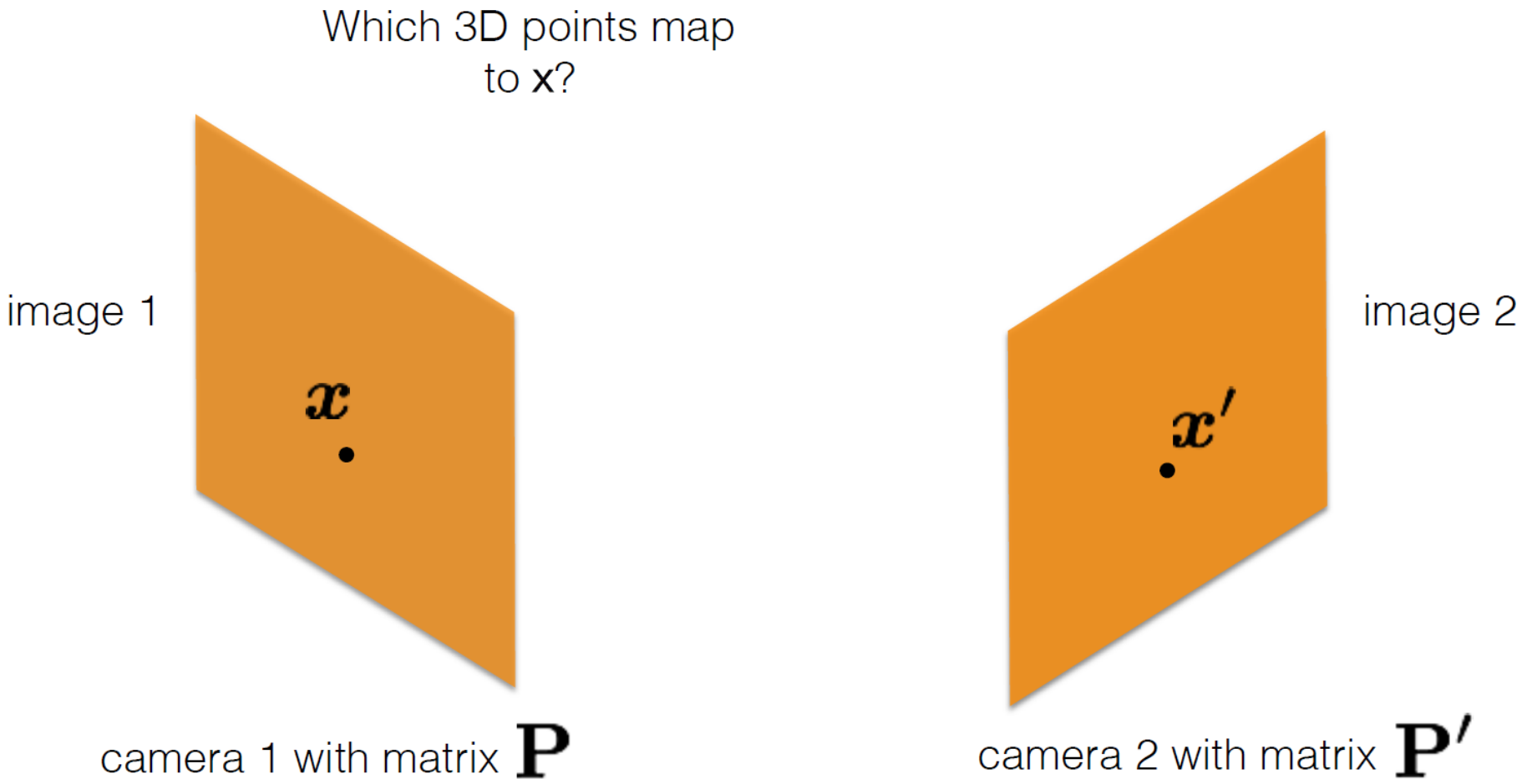
$$\mathbf{n} = \mathbf{v}_1 = \begin{bmatrix} -0.47 \\ -0.88 \\ 0.05 \end{bmatrix}$$

3D Reconstruction

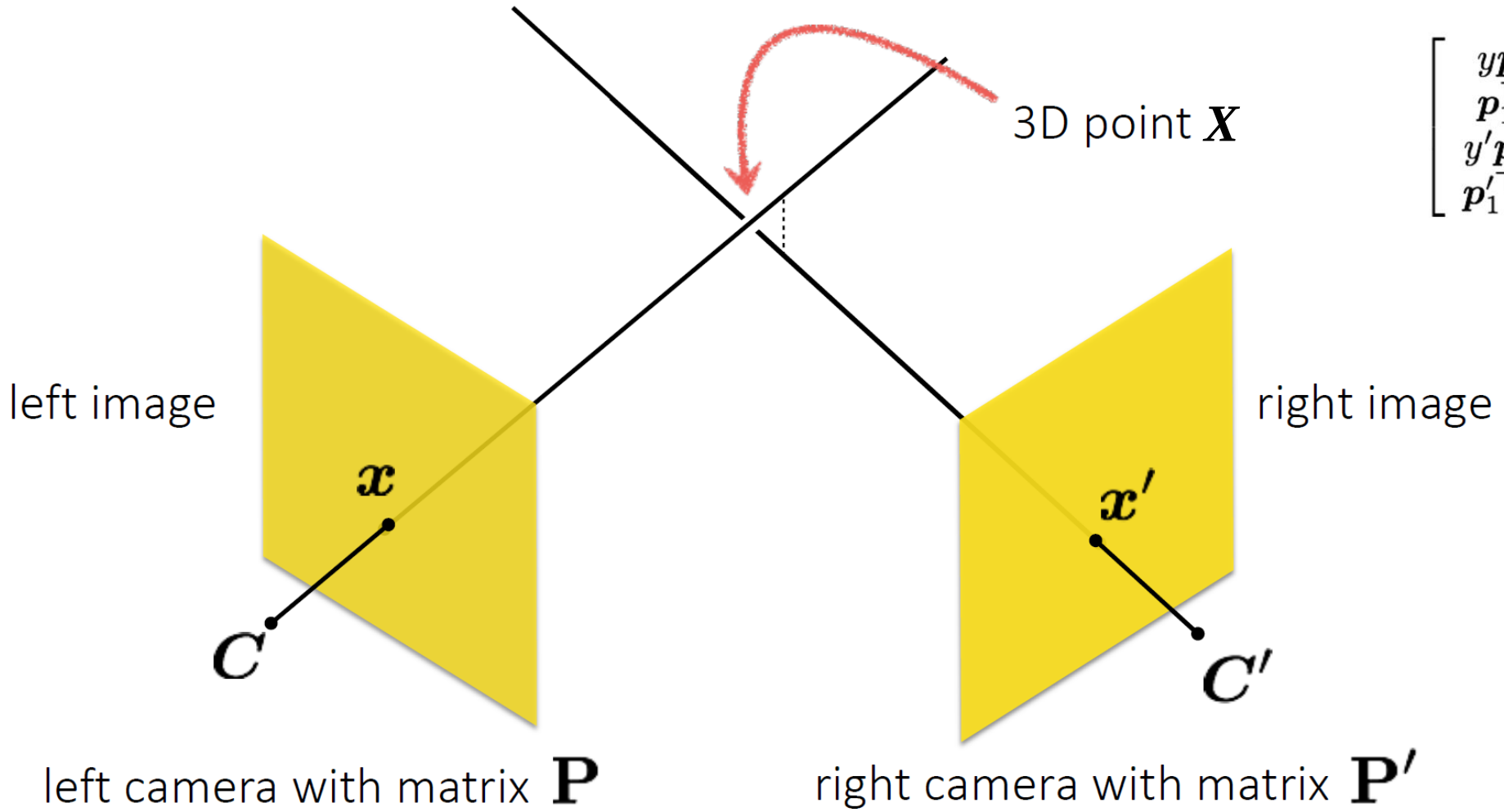
Triangulation



Triangulation



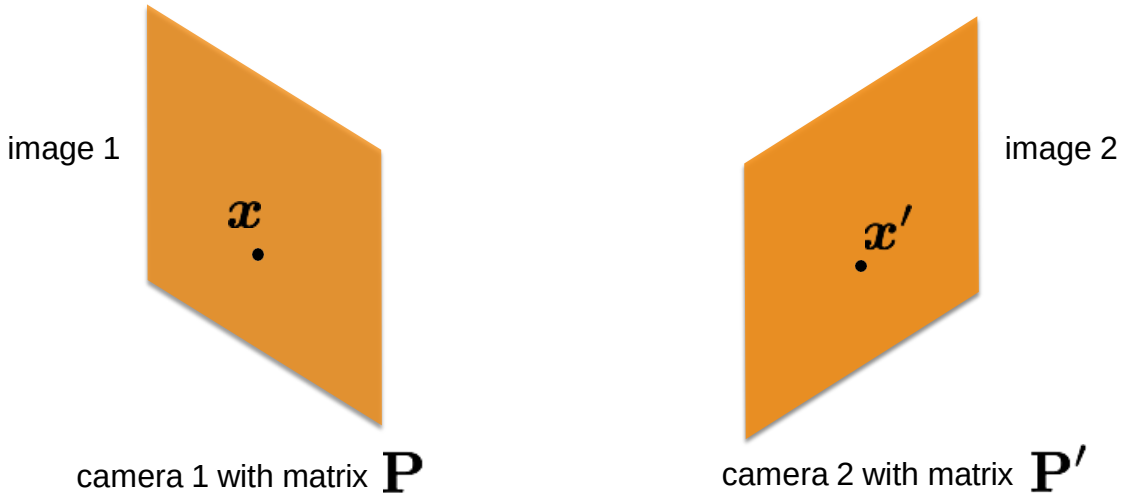
Triangulation



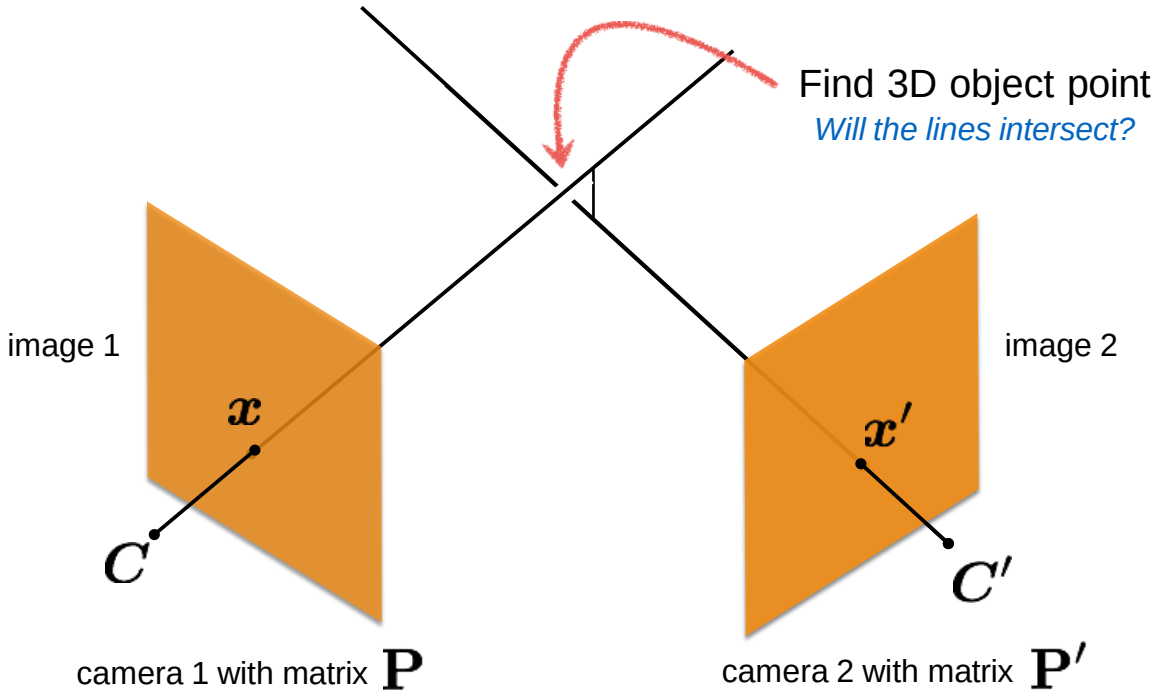
$$\begin{bmatrix} y\boldsymbol{p}_3^\top - \boldsymbol{p}_2^\top \\ \boldsymbol{p}_1^\top - x\boldsymbol{p}_3^\top \\ y'\boldsymbol{p}_3'^\top - \boldsymbol{p}_2'^\top \\ \boldsymbol{p}_1'^\top - x'\boldsymbol{p}_3'^\top \end{bmatrix} \boldsymbol{X} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Triangulation

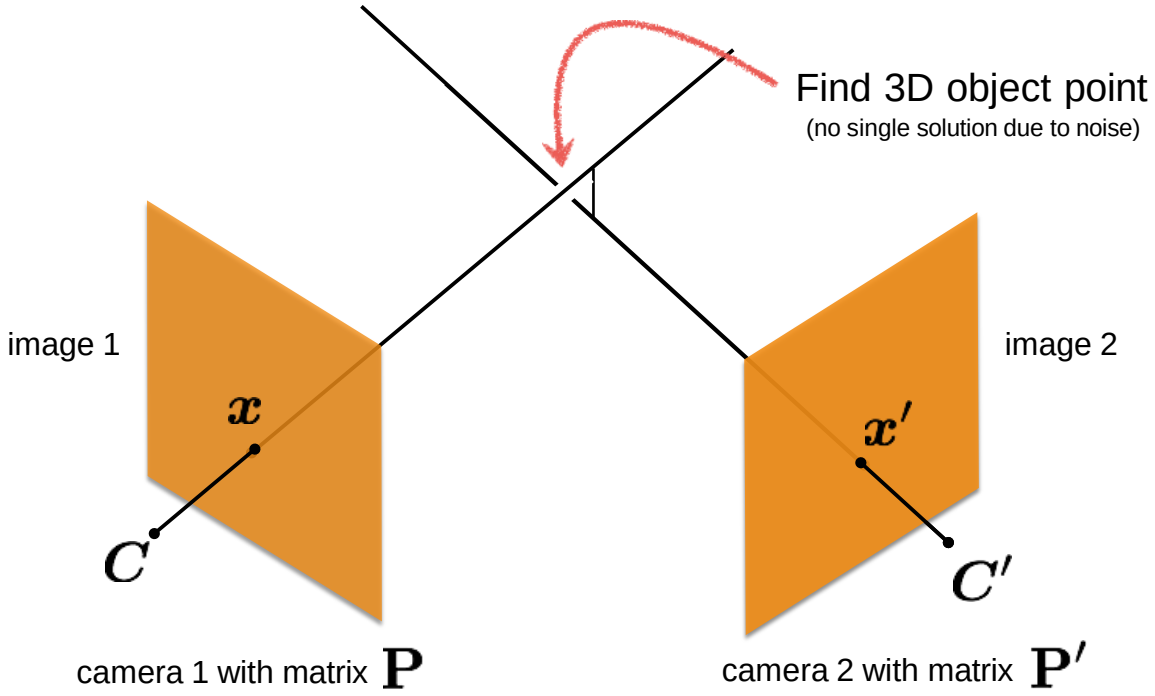
Which 3D points map to x?



Triangulation



Triangulation



Triangulation

What is triangulation?

Given a set of (noisy) matched points

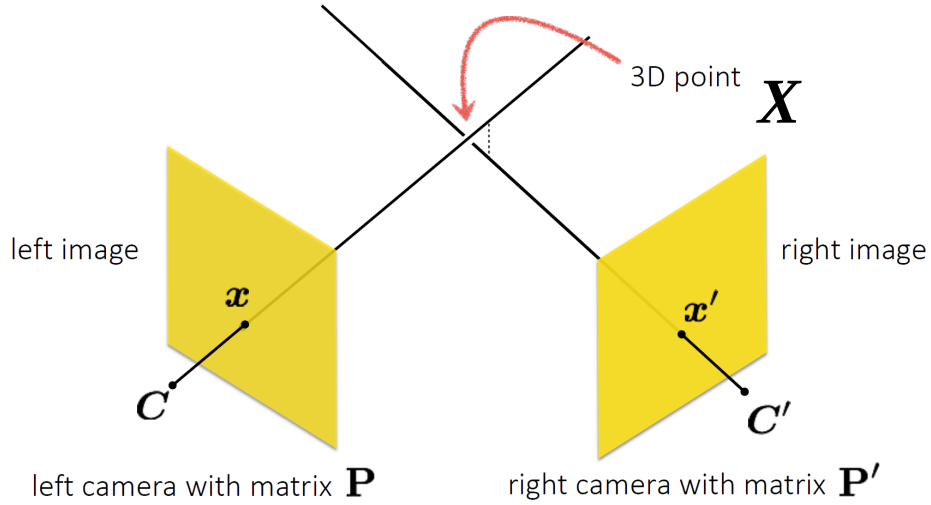
$$\{\boldsymbol{x}_i, \boldsymbol{x}'_i\}$$

and camera matrices

$$\mathbf{P}, \mathbf{P}'$$

Estimate the 3D point

$$\mathbf{X}$$



Can we compute X from a single correspondence x?

$$\mathbf{x} = \mathbf{P}\boldsymbol{X}$$

known known

No!

Triangulation

$$\mathbf{x} = \mathbf{P}\boldsymbol{X}$$

(homogeneous coordinate)

This is a *similarity relation* because it involves *homogeneous* coordinates

$$\mathbf{x} = \alpha\mathbf{P}\boldsymbol{X}$$

(heterogeneous coordinate)

Same ray direction but differs by a scale factor

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \alpha \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

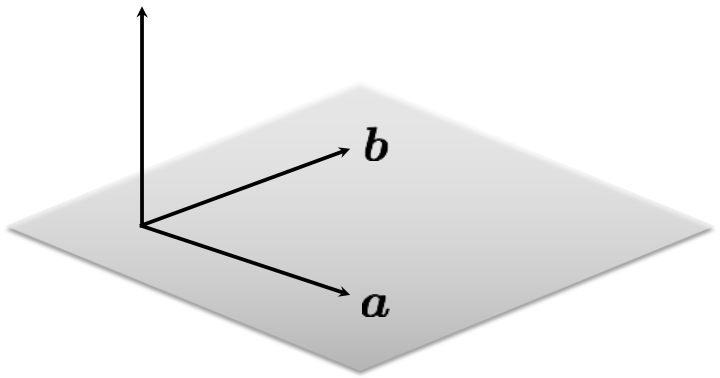
How do we solve for unknowns in a similarity relation?

Linear algebra reminder

Vector cross product

takes two vectors and returns a vector <u>perpendicular</u> to both

$$\boldsymbol{c} = \boldsymbol{a} \times \boldsymbol{b}$$



$$\boldsymbol{c} \cdot \boldsymbol{a} = 0 \qquad \boldsymbol{c} \cdot \boldsymbol{b} = 0$$

$$\boldsymbol{a} \times \boldsymbol{b} = \begin{bmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{bmatrix}$$

cross product of two vectors in the same direction is zero vector

$$\boldsymbol{a} \times \boldsymbol{a} = 0$$

remember this!!!

Linear algebra reminder: cross product

Cross product

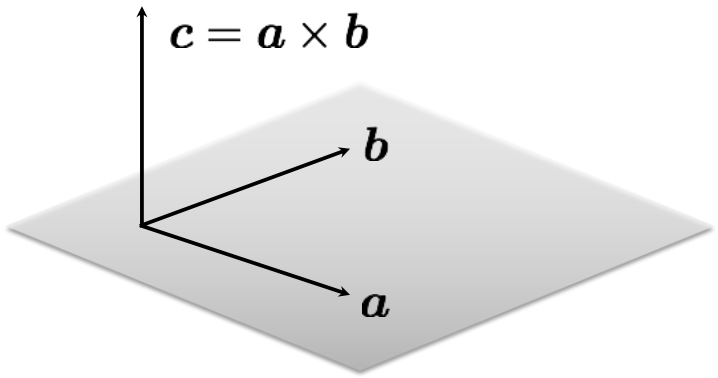
$$\boldsymbol{a} \times \boldsymbol{b} = \begin{bmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{bmatrix}$$

Can also be written as a matrix multiplication

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

Skew symmetric

Compare with: dot product



$\boldsymbol{c} \cdot \boldsymbol{a} = 0$ $\boldsymbol{c} \cdot \boldsymbol{b} = 0$

dot product of two orthogonal vectors is (scalar) zero

Back to triangulation

$$\mathbf{x} = \alpha \mathbf{P} \boldsymbol{X}$$

Same direction but differs by a scale factor

How can we rewrite this using vector products?

$$\mathbf{x} = \alpha \mathbf{P} \boldsymbol{X}$$

Same direction but differs by a scale factor

$$\mathbf{x} \times \mathbf{P} \boldsymbol{X} = \mathbf{0}$$

Cross product of two vectors of same direction is zero
(this equality removes the scale factor)

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \alpha \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Do the same after first expanding out the camera matrix and points

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \alpha \begin{bmatrix} — & \boldsymbol{p}_1^\top & — \\ — & \boldsymbol{p}_2^\top & — \\ — & \boldsymbol{p}_3^\top & — \end{bmatrix} \begin{bmatrix} | \\ \boldsymbol{X} \\ | \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \alpha \begin{bmatrix} \boldsymbol{p}_1^\top \boldsymbol{X} \\ \boldsymbol{p}_2^\top \boldsymbol{X} \\ \boldsymbol{p}_3^\top \boldsymbol{X} \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \times \begin{bmatrix} \boldsymbol{p}_1^\top \boldsymbol{X} \\ \boldsymbol{p}_2^\top \boldsymbol{X} \\ \boldsymbol{p}_3^\top \boldsymbol{X} \end{bmatrix} = \begin{bmatrix} y\boldsymbol{p}_3^\top \boldsymbol{X} - \boldsymbol{p}_2^\top \boldsymbol{X} \\ \boldsymbol{p}_1^\top \boldsymbol{X} - x\boldsymbol{p}_3^\top \boldsymbol{X} \\ x\boldsymbol{p}_2^\top \boldsymbol{X} - y\boldsymbol{p}_1^\top \boldsymbol{X} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Using the fact that the cross product should be zero

$$\mathbf{x} \times \mathbf{P}\boldsymbol{X} = \mathbf{0}$$

$$\begin{bmatrix} y\boldsymbol{p}_3^\top \boldsymbol{X} - \boldsymbol{p}_2^\top \boldsymbol{X} \\ \boldsymbol{p}_1^\top \boldsymbol{X} - x\boldsymbol{p}_3^\top \boldsymbol{X} \\ x\boldsymbol{p}_2^\top \boldsymbol{X} - y\boldsymbol{p}_1^\top \boldsymbol{X} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Third line is a linear combination of the first and second lines.
(x times the first line plus y times the second line)

One 2D to 3D point correspondence give you 2 equations

$$\begin{bmatrix} y\boldsymbol{p}_3^\top \boldsymbol{X} - \boldsymbol{p}_2^\top \boldsymbol{X} \\ \boldsymbol{p}_1^\top \boldsymbol{X} - x\boldsymbol{p}_3^\top \boldsymbol{X} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Remove third row, and rearrange as system on unknowns

$$\begin{bmatrix} y\boldsymbol{p}_3^\top - \boldsymbol{p}_2^\top \\ \boldsymbol{p}_1^\top - x\boldsymbol{p}_3^\top \end{bmatrix} \boldsymbol{X} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\mathbf{A}_i \boldsymbol{X} = \mathbf{0}$$

Now we can make a system of linear equations
(two lines for each 2D point correspondence)

Concatenate the 2D points from both images

Two rows from camera one

Two rows from camera two

$$\begin{bmatrix} y\boldsymbol{p}_3^\top - \boldsymbol{p}_2^\top \\ \boldsymbol{p}_1^\top - x\boldsymbol{p}_3^\top \\ y'\boldsymbol{p}_3'^\top - \boldsymbol{p}_2'^\top \\ \boldsymbol{p}_1'^\top - x'\boldsymbol{p}_3'^\top \end{bmatrix} \boldsymbol{X} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

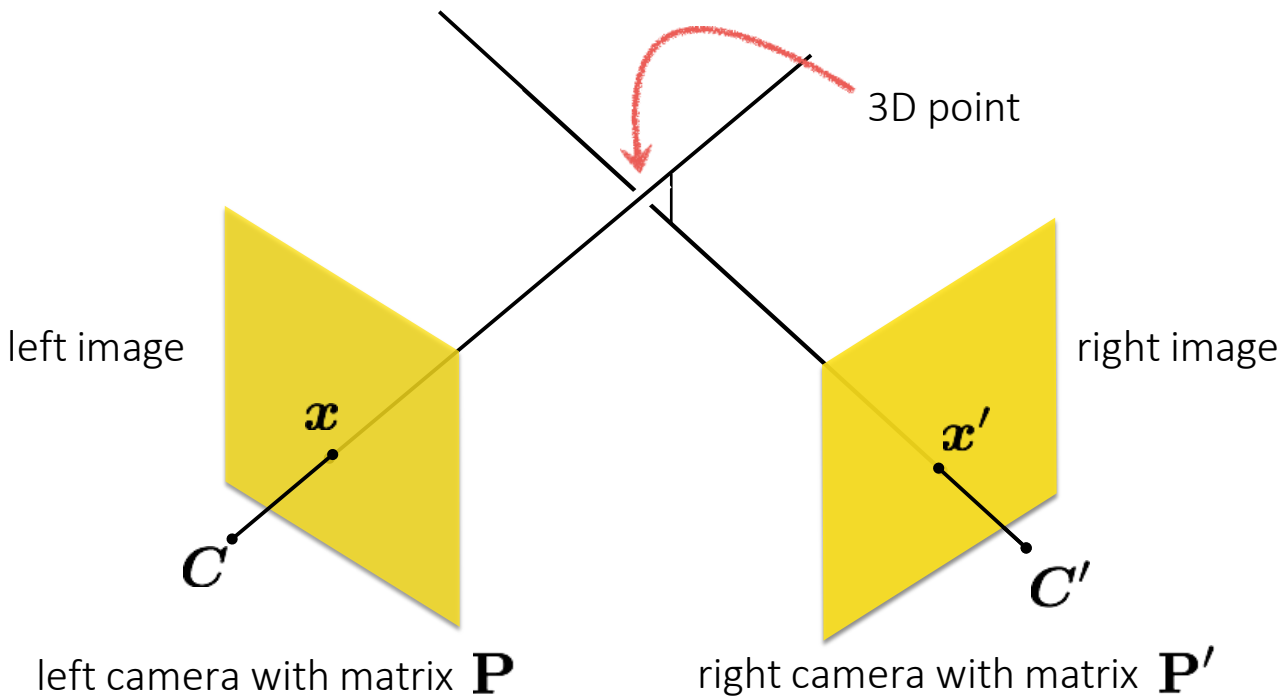
sanity check! dimensions?

$$\mathbf{A}\boldsymbol{X} = \mathbf{0}$$

How do we solve homogeneous linear system?

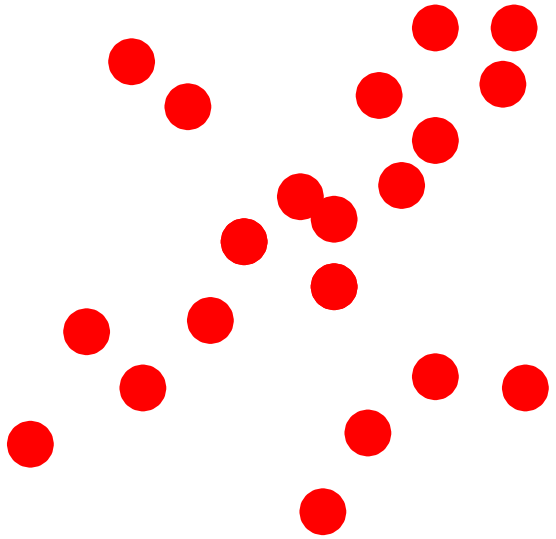
SVD!

Triangulation



Random Sample Consensus (RANSAC)

Fitting lines (with outliers)

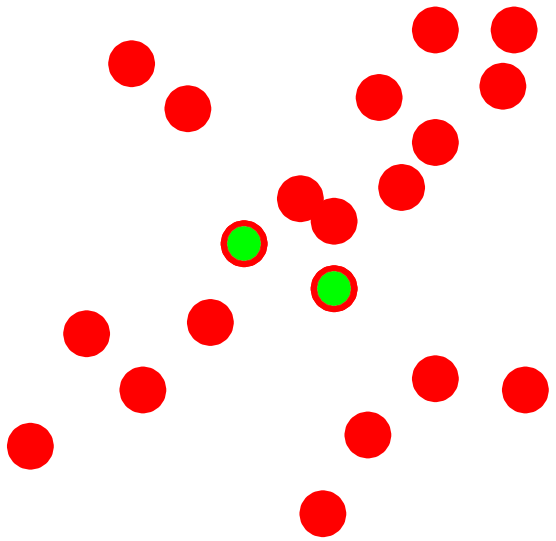


Algorithm:

1. Sample (randomly) the number of points required to fit the model
2. Solve for model parameters using samples
3. Score by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

Fitting lines (with outliers)

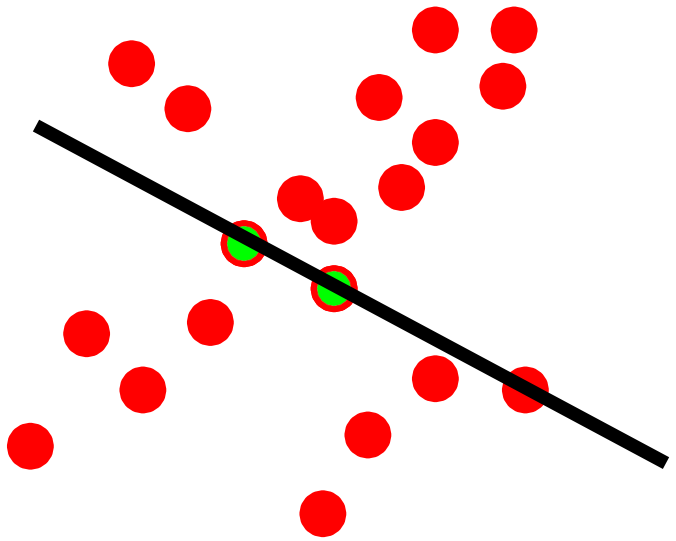


Algorithm:

1. **Sample (randomly) the number of points required to fit the model**
2. Solve for model parameters using samples
3. Score by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

Fitting lines (with outliers)

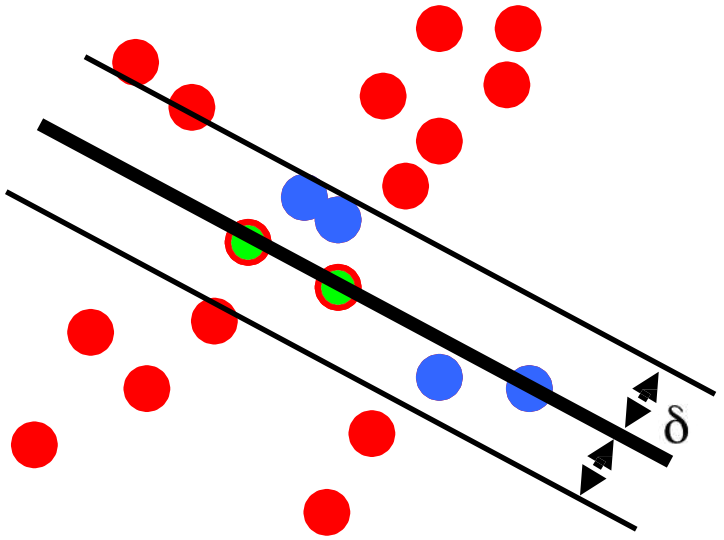


Algorithm:

1. Sample (randomly) the number of points required to fit the model
2. **Solve for model parameters using samples**
3. Score by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

Fitting lines (with outliers)



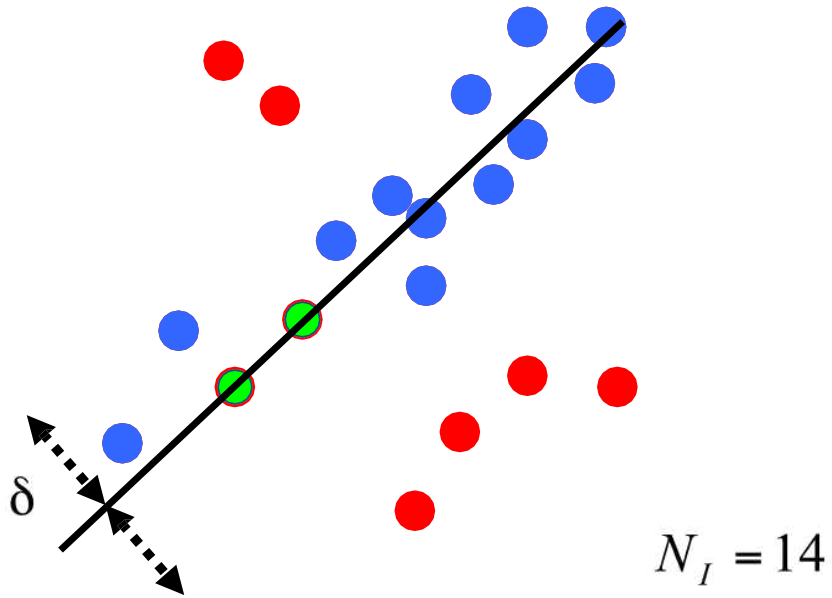
$N_I = 6$

Algorithm:

1. Sample (randomly) the number of points required to fit the model
2. Solve for model parameters using samples
3. **Score by the fraction of inliers within a preset threshold of the model**

Repeat 1-3 until the best model is found with high confidence

Fitting lines (with outliers)



Algorithm:

1. Sample (randomly) the number of points required to fit the model
2. Solve for model parameters using samples
3. Score by the fraction of inliers within a preset threshold of the model

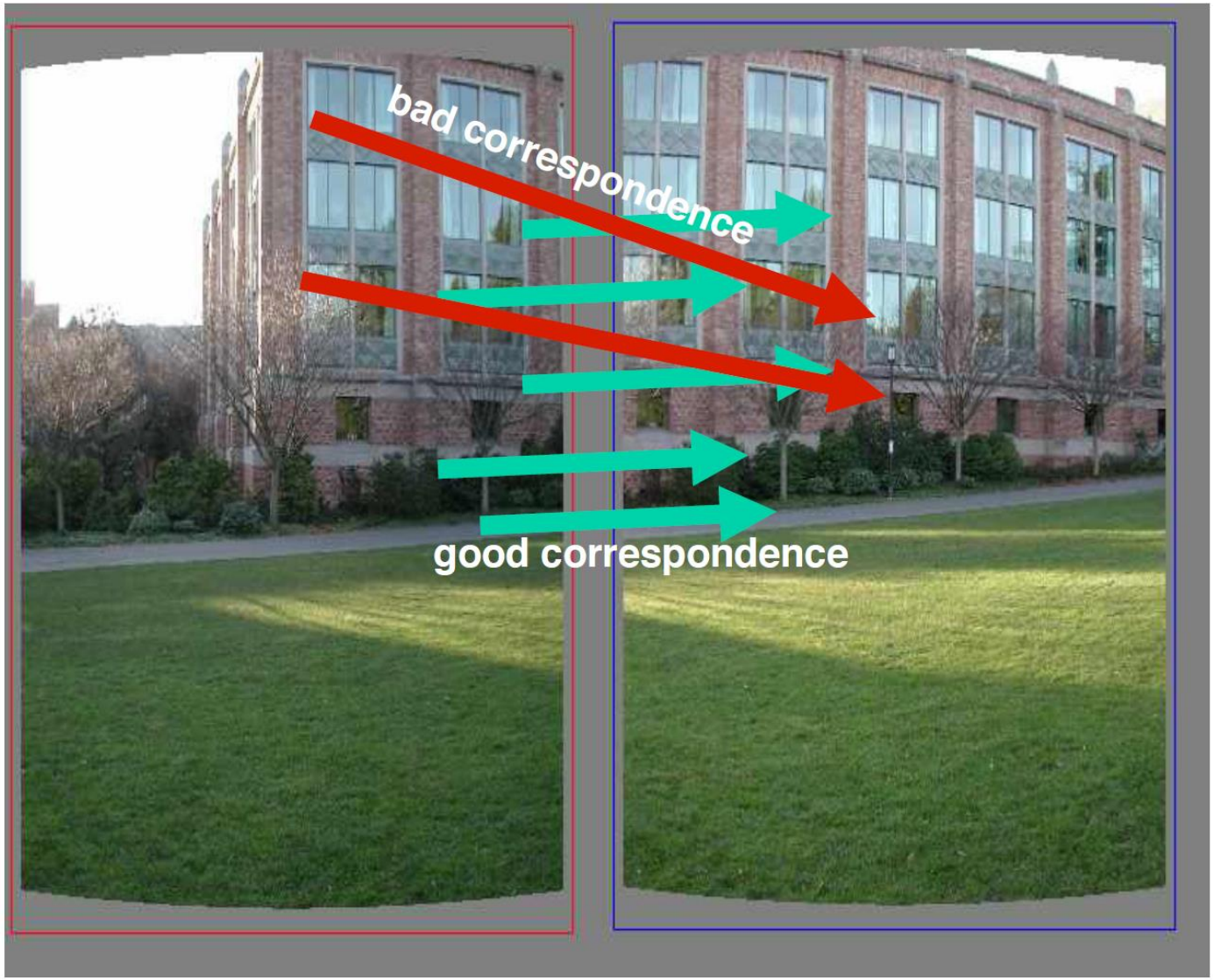
Repeat 1-3 until the best model is found with high confidence

Example

Given two images, find matching features (e.g., SIFT) and a translation transform

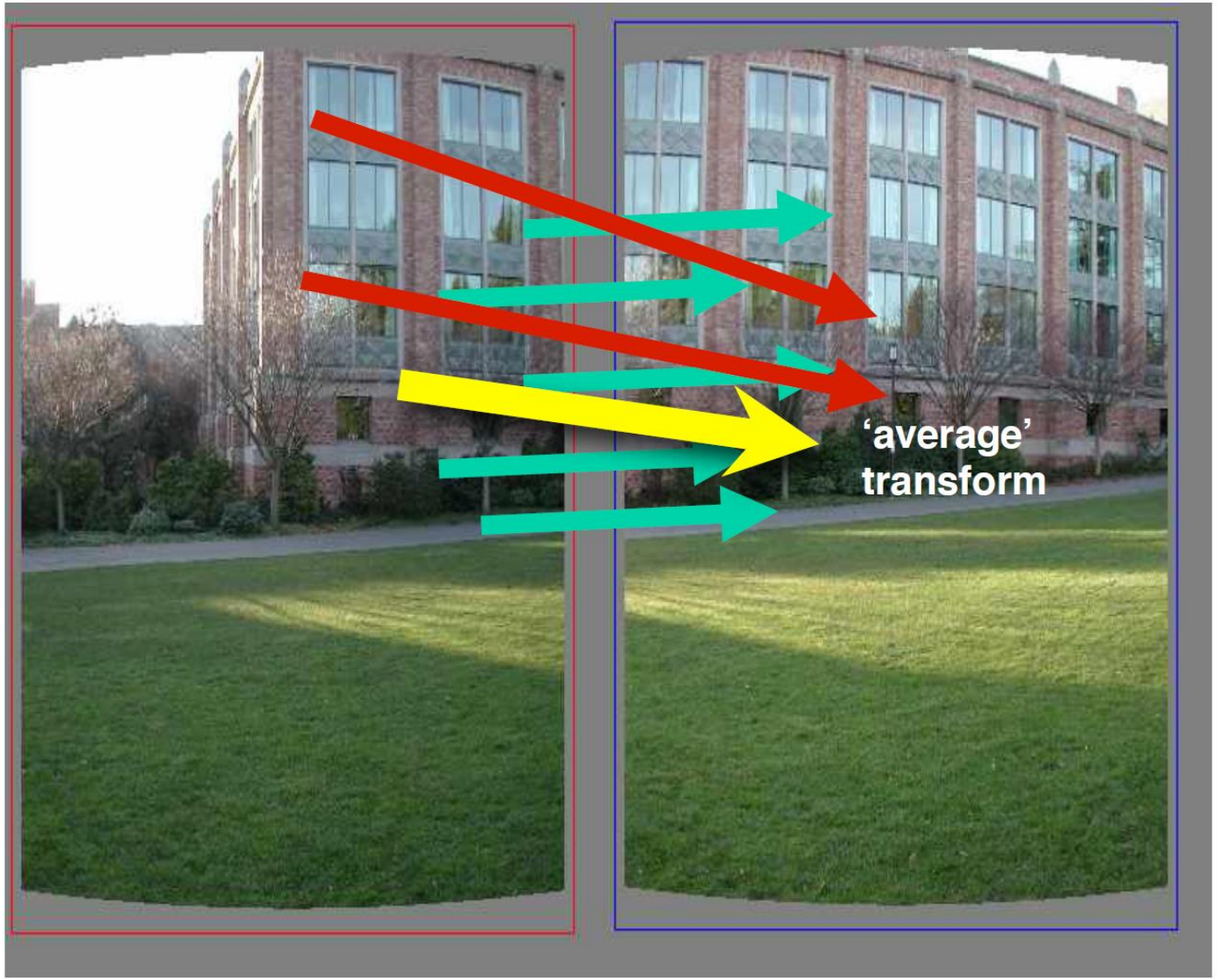
Example

Matched points will usually contain bad correspondences



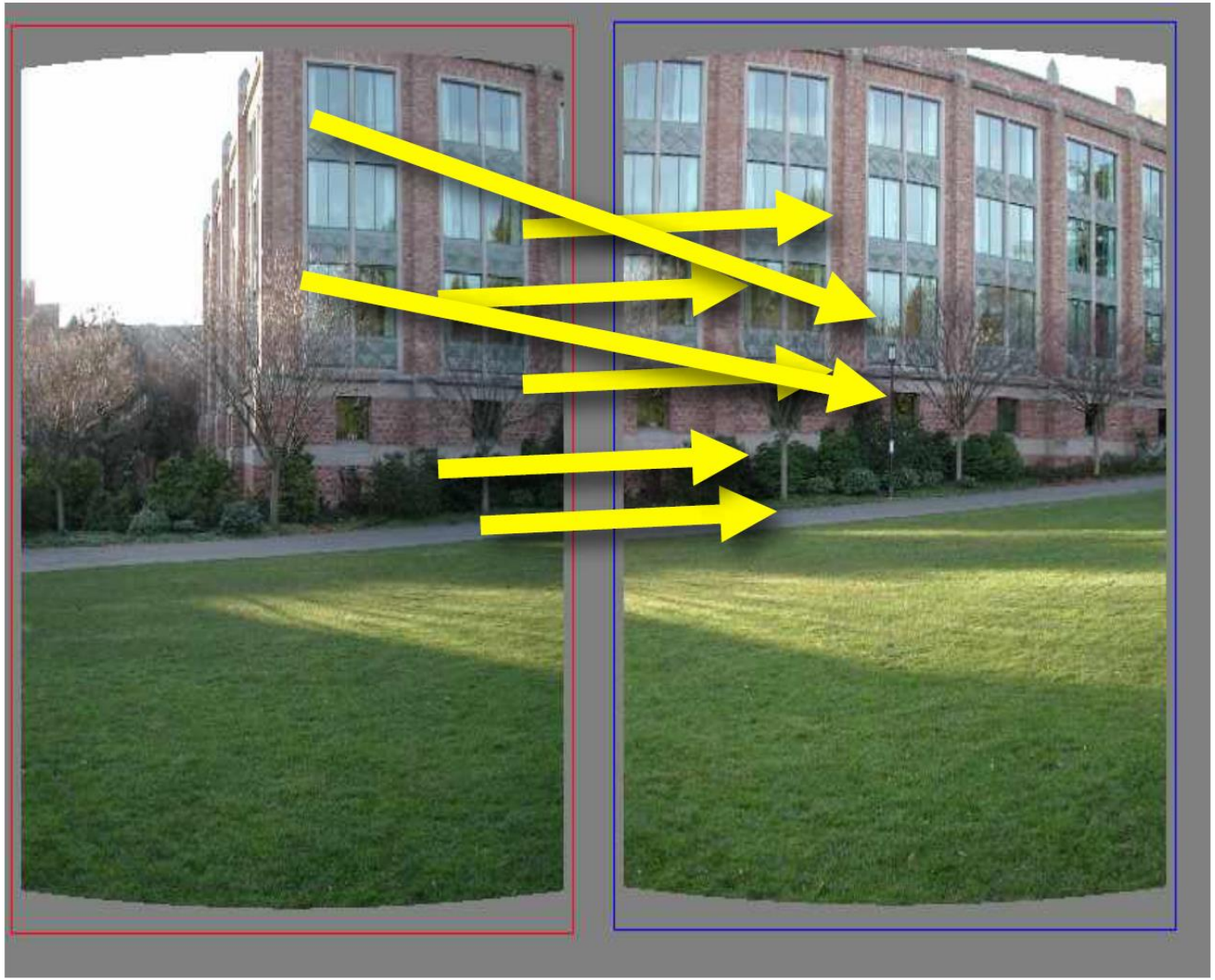
Example

Least squares solutions will find the "average" transform



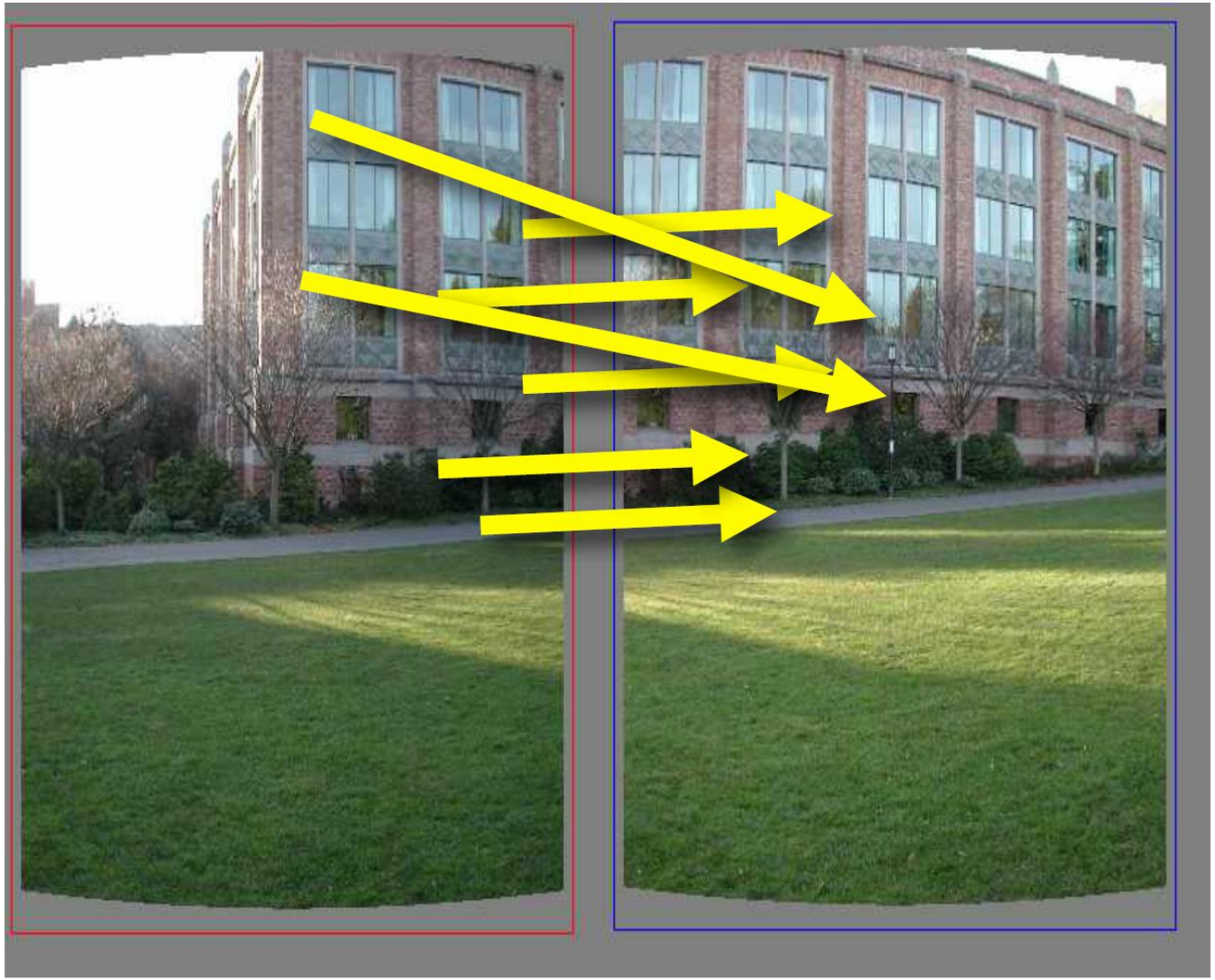
Example

Use RANSAC!



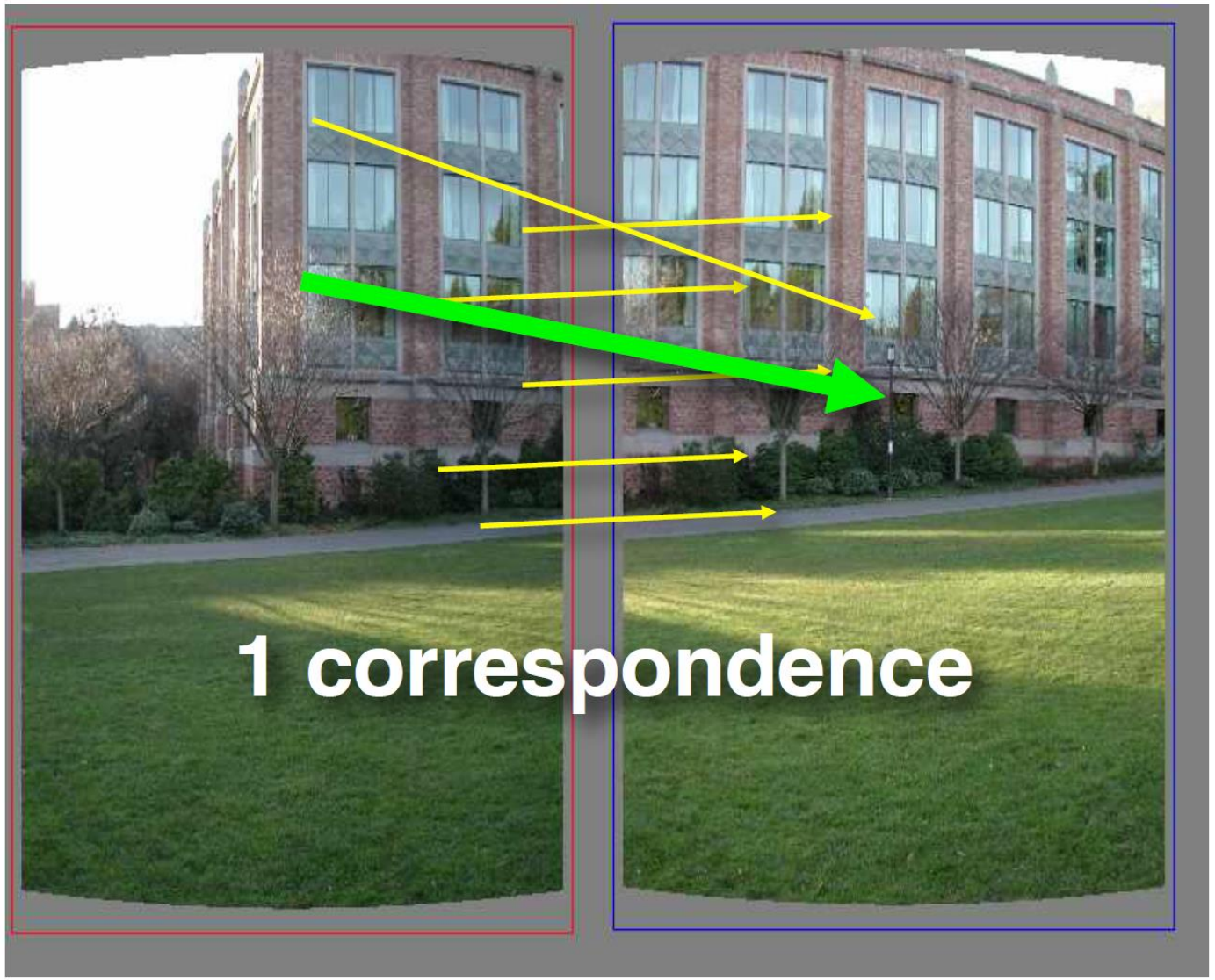
Example

Need only one correspondence, to find translation model



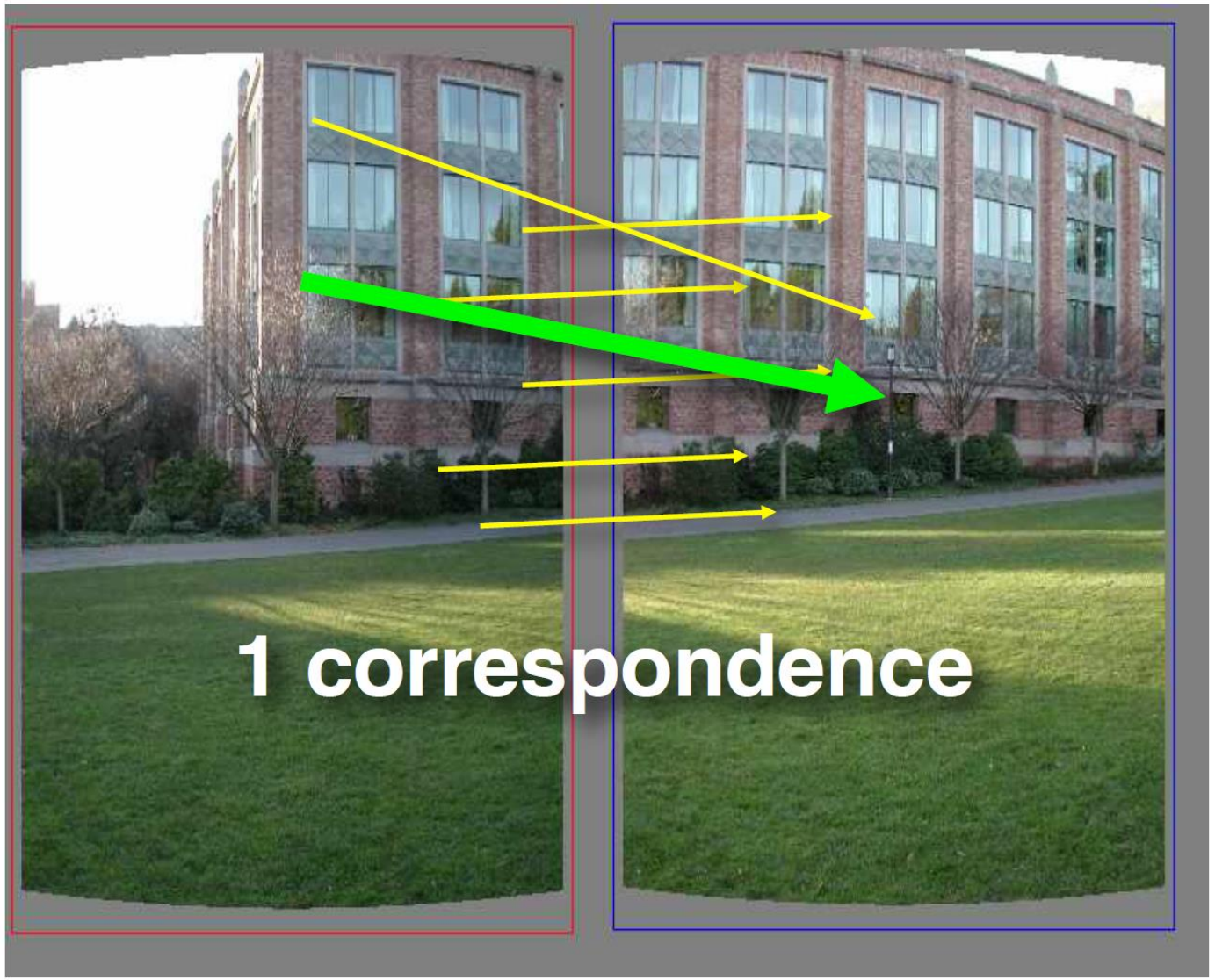
Example

Pick one correspondence, count inliers



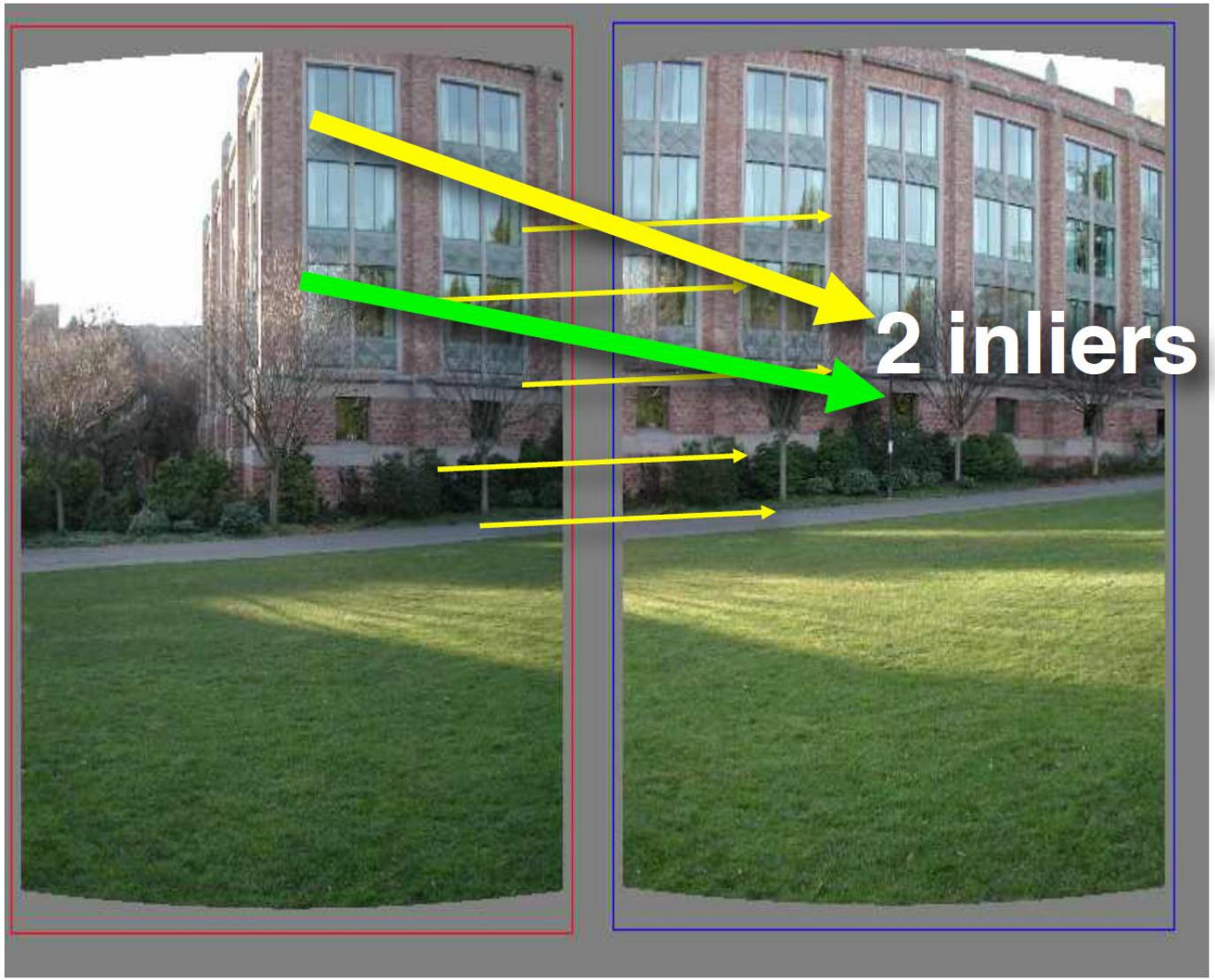
Example

Pick one correspondence, count inliers



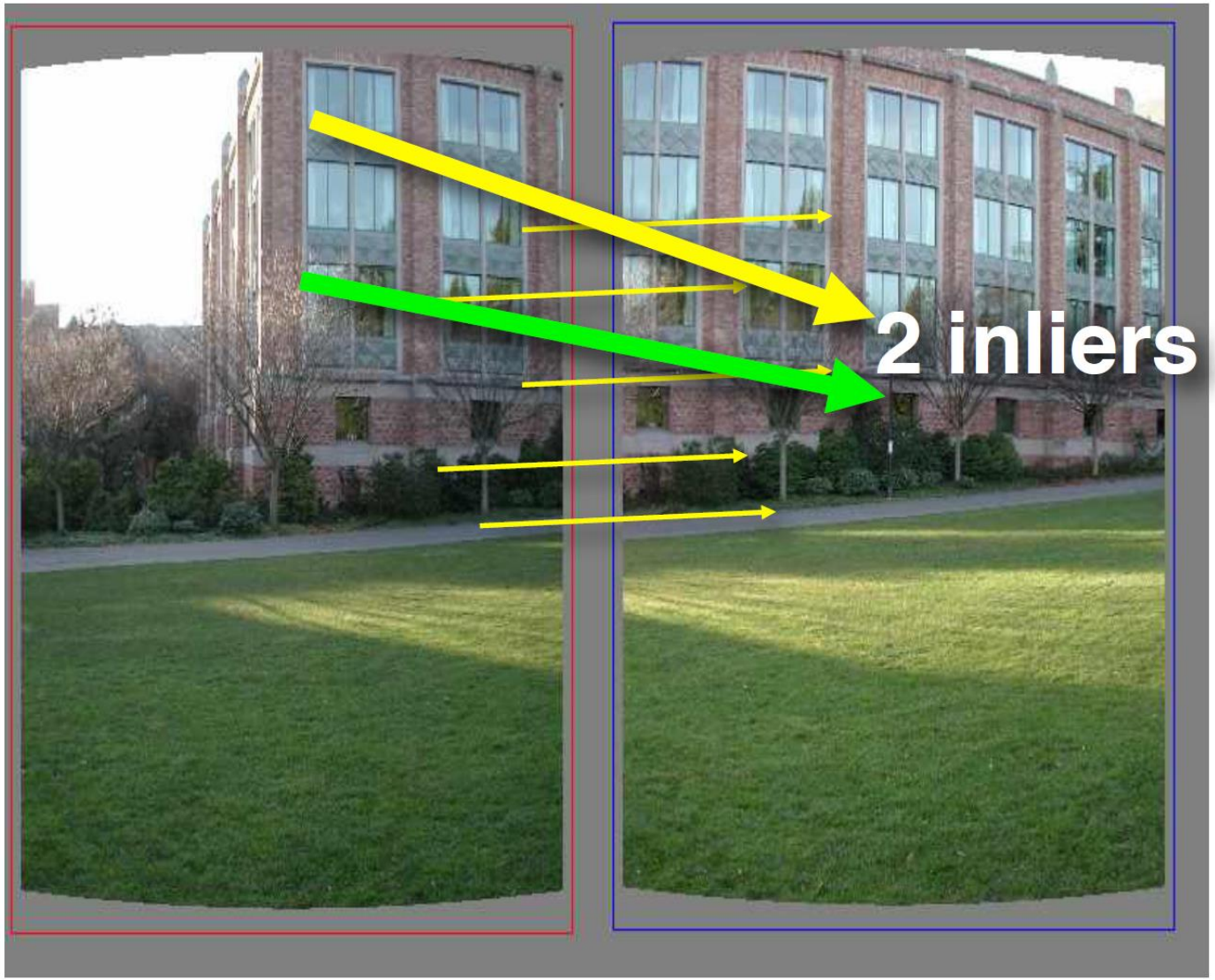
Example

Pick one correspondence, count inliers



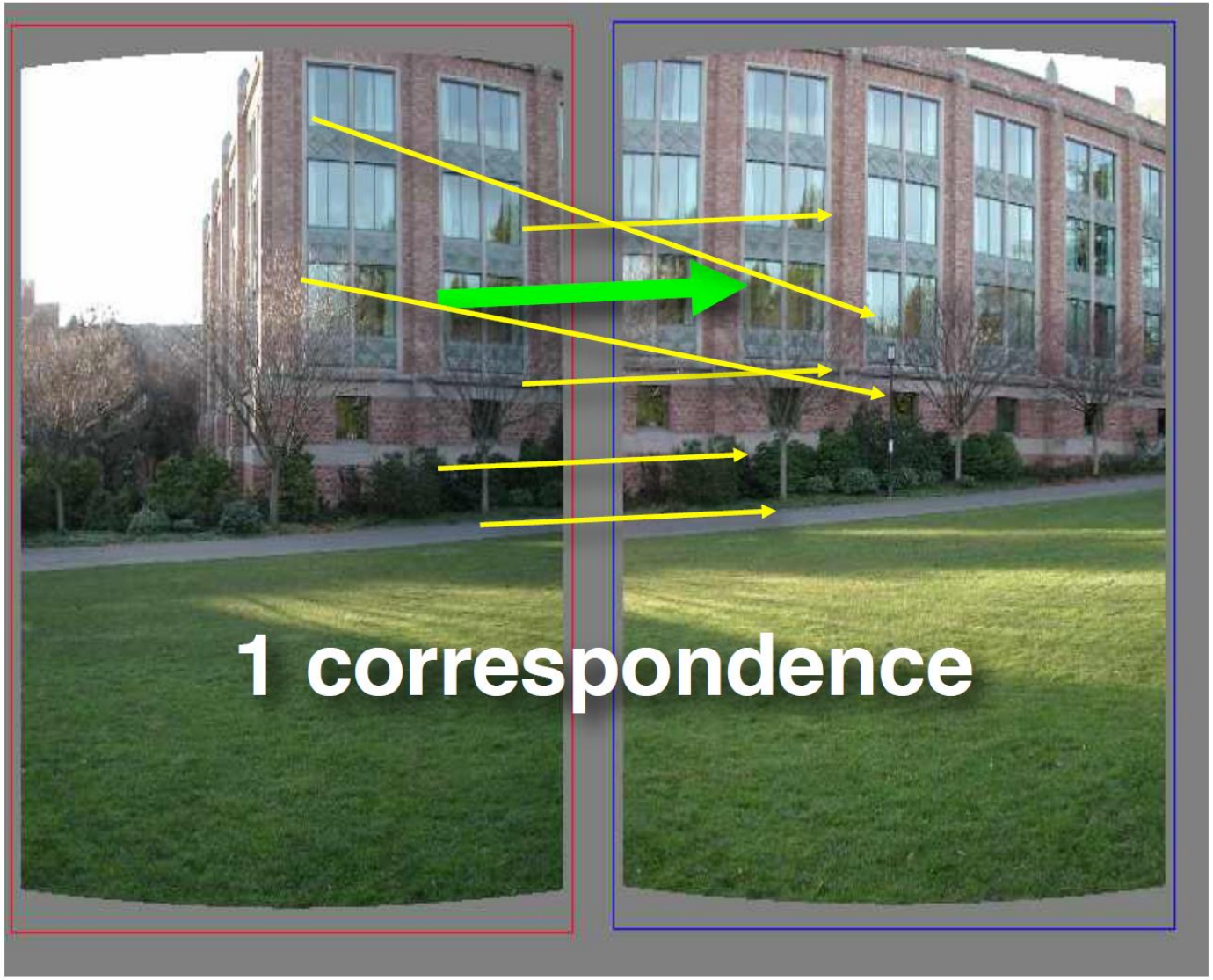
Example

Pick one correspondence, count inliers



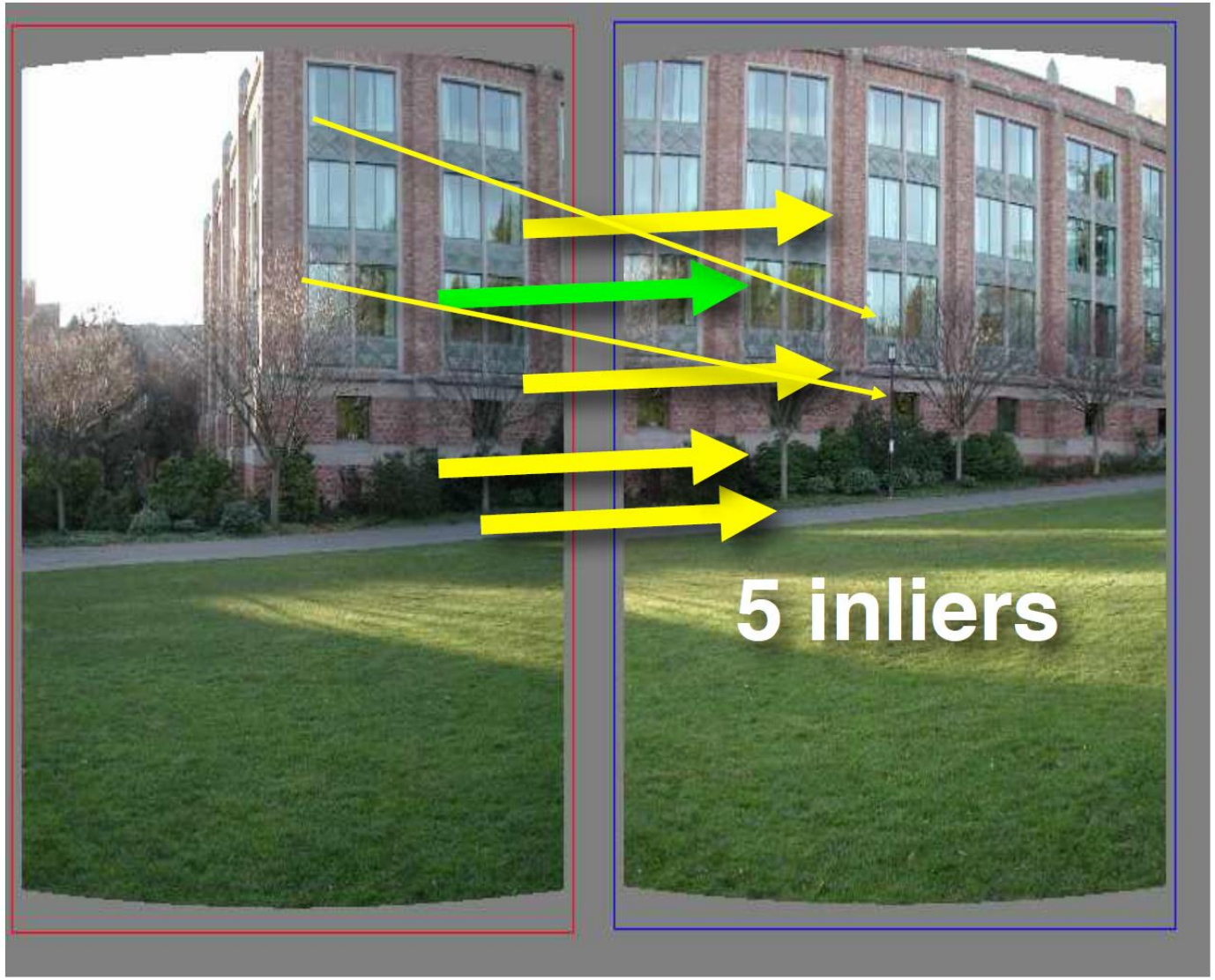
Example

Pick one correspondence, count inliers



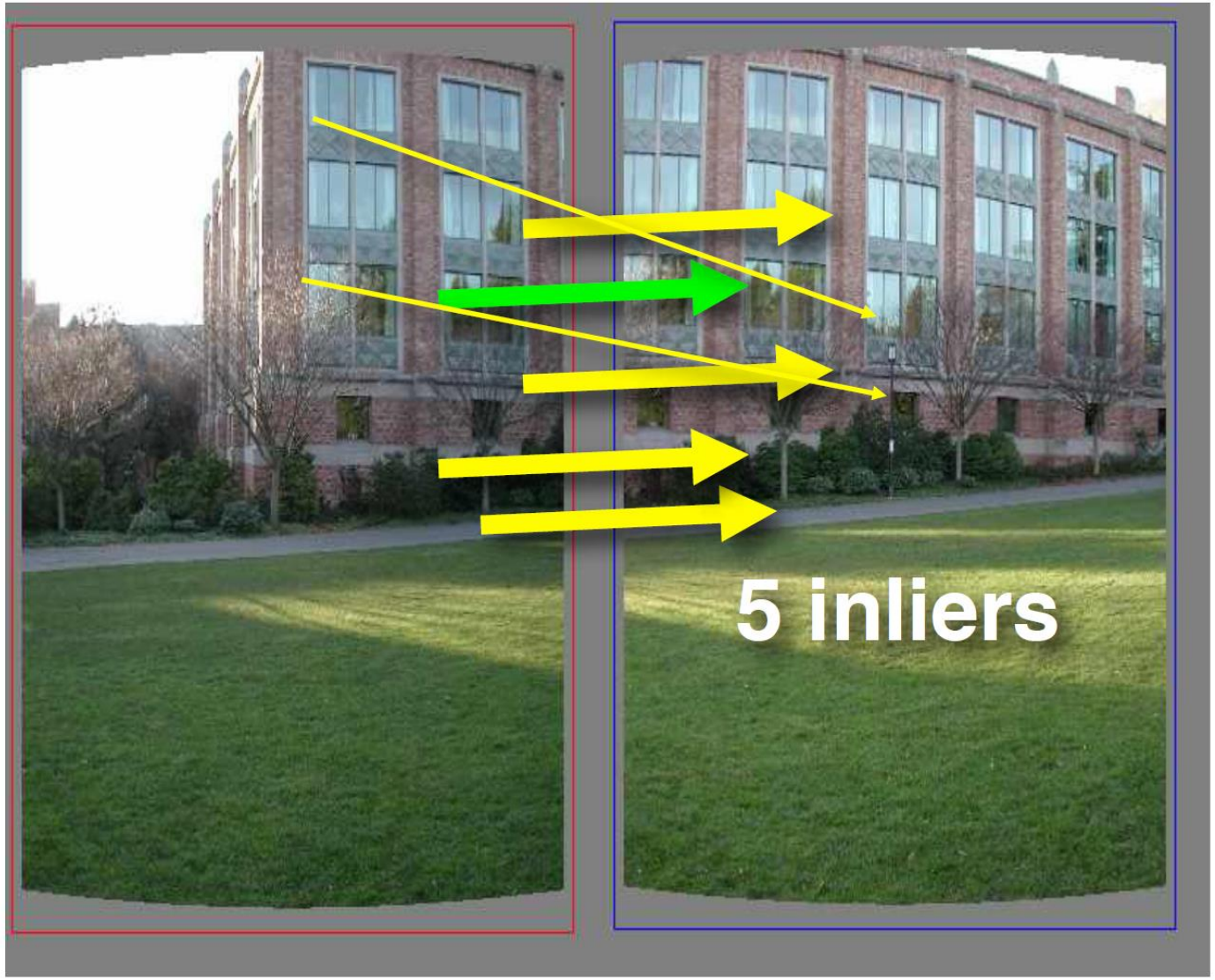
Example

Pick one correspondence, count inliers



Example

Pick the model with the **highest** number of **inliers**!



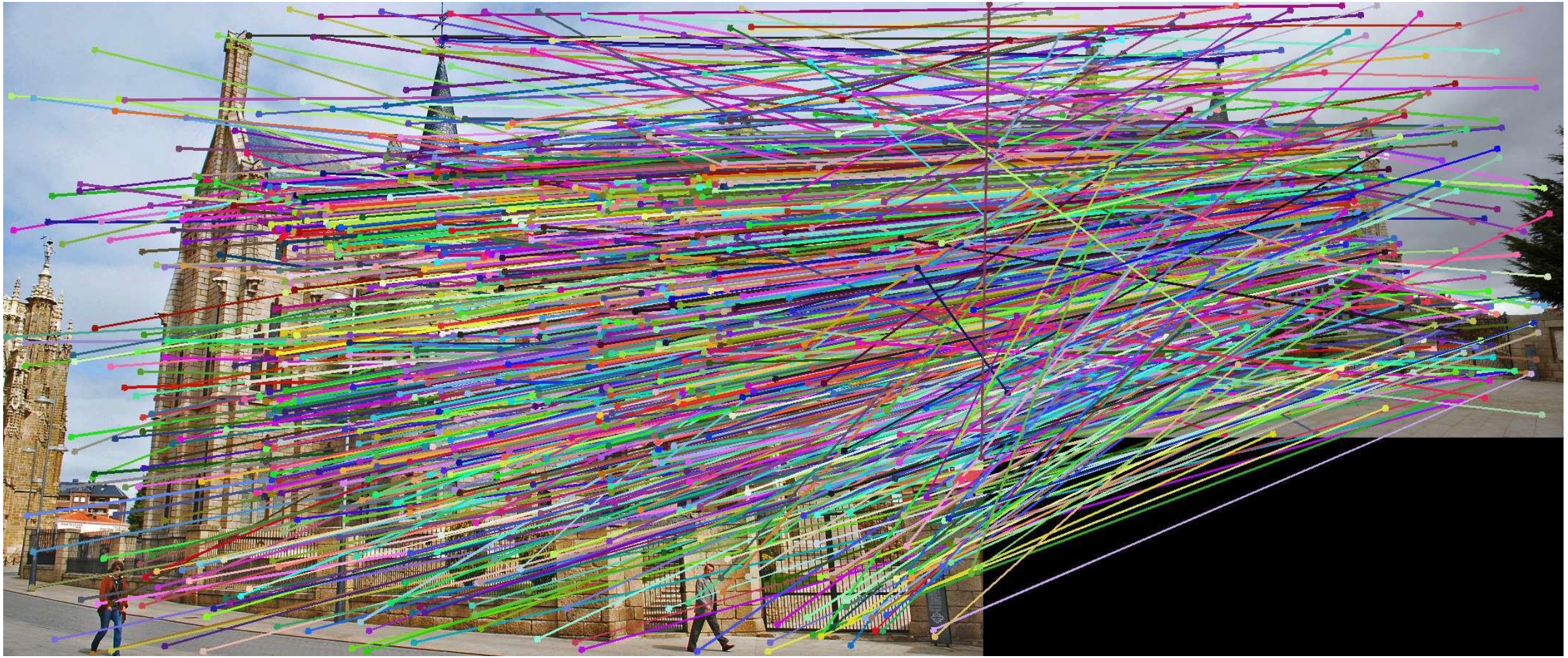
Estimating homography using RANSAC

- RANSAC loop
  1. Get 4 point correspondences (randomly)
  2. Compute H using DLT
  3. Count inliers ($Hp = p'$)
  4. Keep H if largest number of inliers
- Recompute H using all inliers

Estimating essential matrix using RANSAC

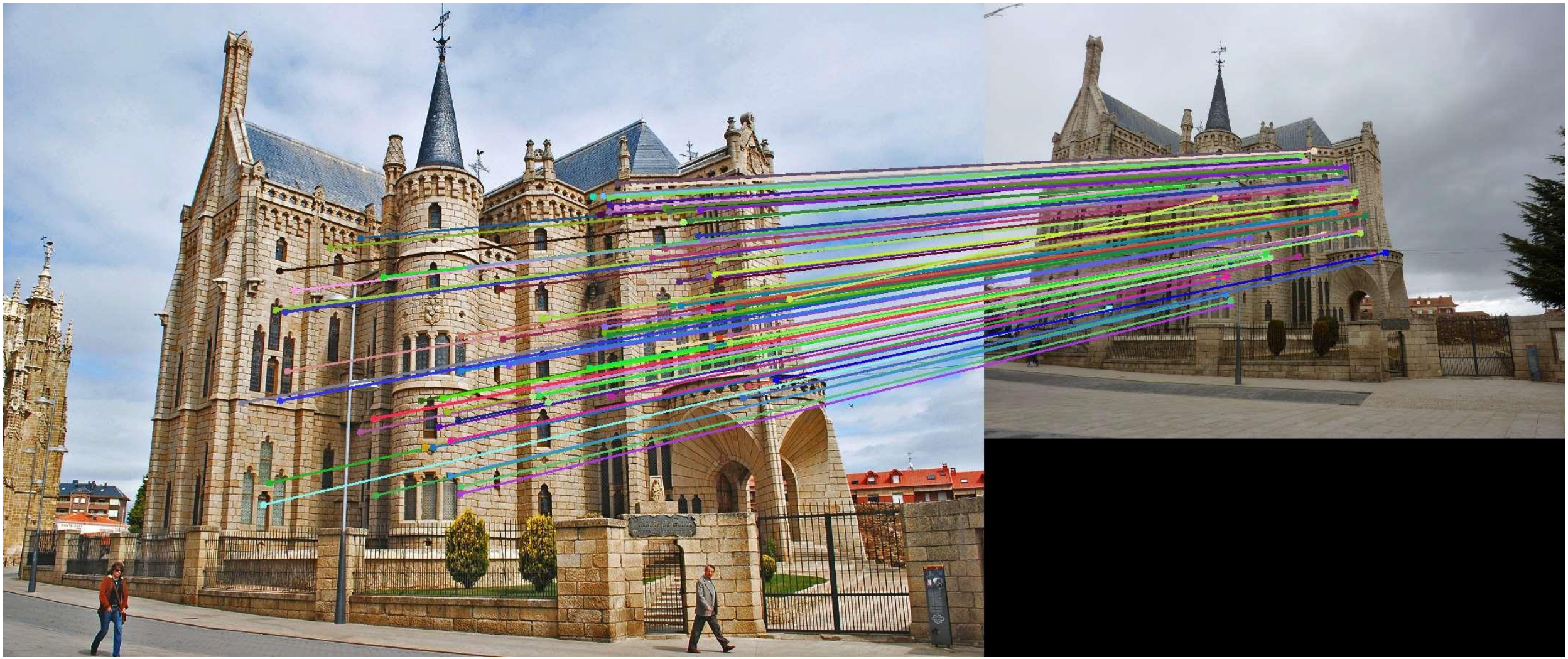
- RANSAC loop
  1. Get 8 point correspondences (randomly)
  2. Compute E using 8-point algorithm
  3. Count inliers ($p^T E p' = 0$)
  4. Keep E if largest number of inliers
- Recompute E using all inliers

SIFT



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

RANSAC



RANSAC conclusions

Good

- Robust to outliers
- Applicable to large number of models (or objective functions)
- Parameters are easy to choose

Bad

- Computational time grows quickly (exponentially) with outlier percentage and number of parameters in the model
- Not good for getting multiple fits

Common applications

- Estimating fundamental matrix (relating two views)
- Computing a homography (e.g., image stitching)