Two-View Geometry

Chul Min Yeum
Assistant Professor
Civil and Environmental Engineering
University of Waterloo, Canada

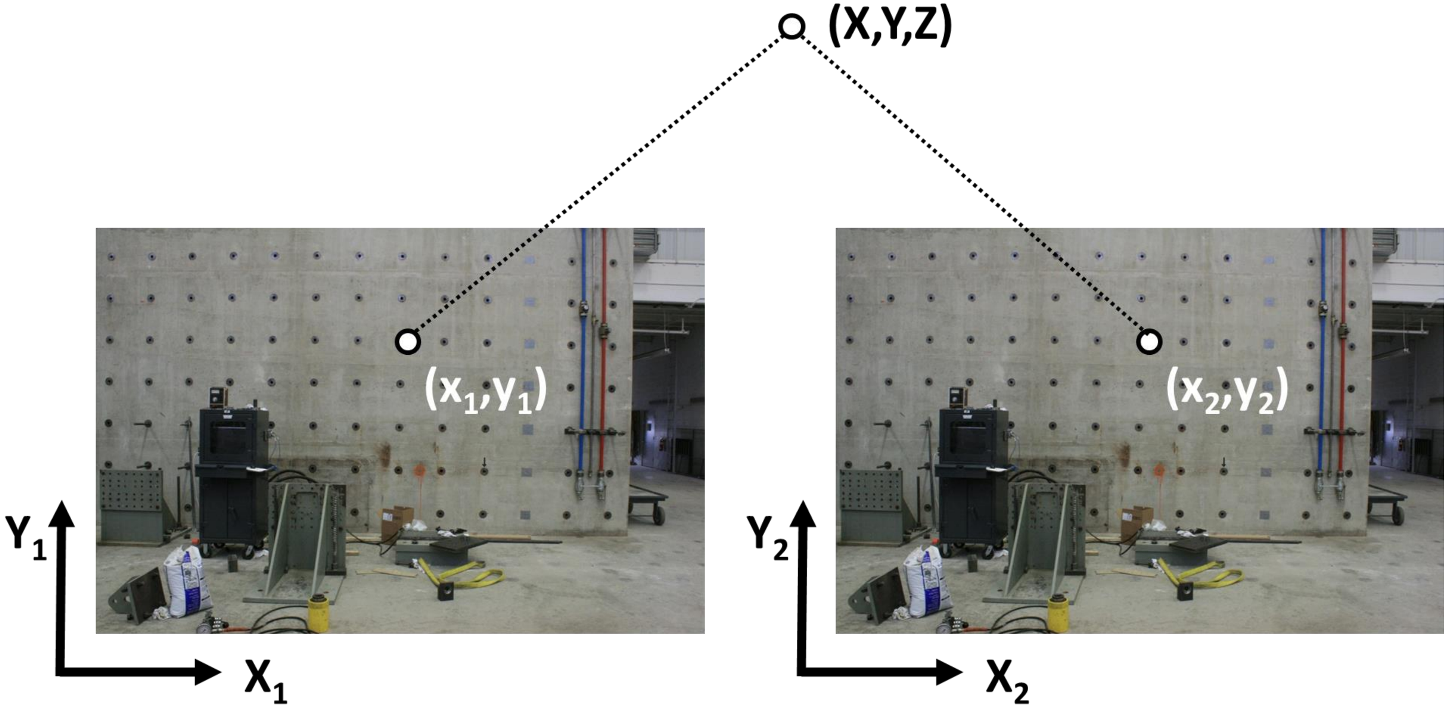
CIVE 497 – CIVE 700: Smart Structure Technology

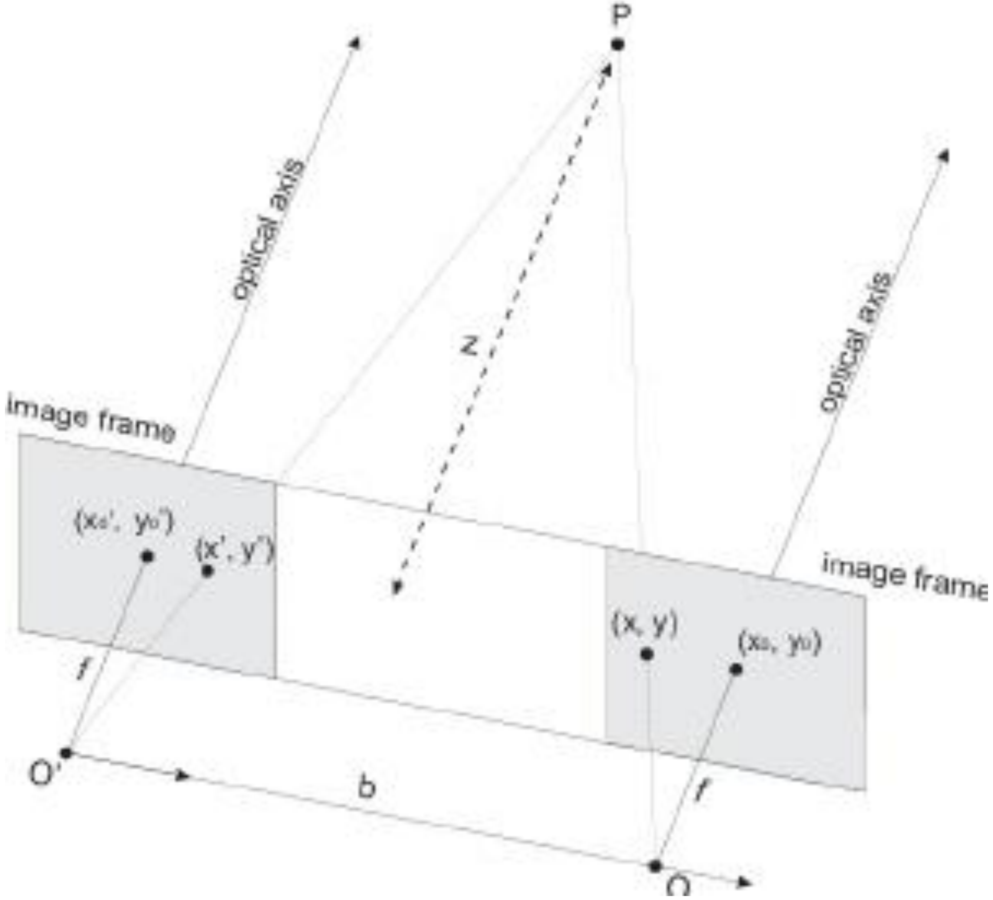
Last updated: 2021-02-09

UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING

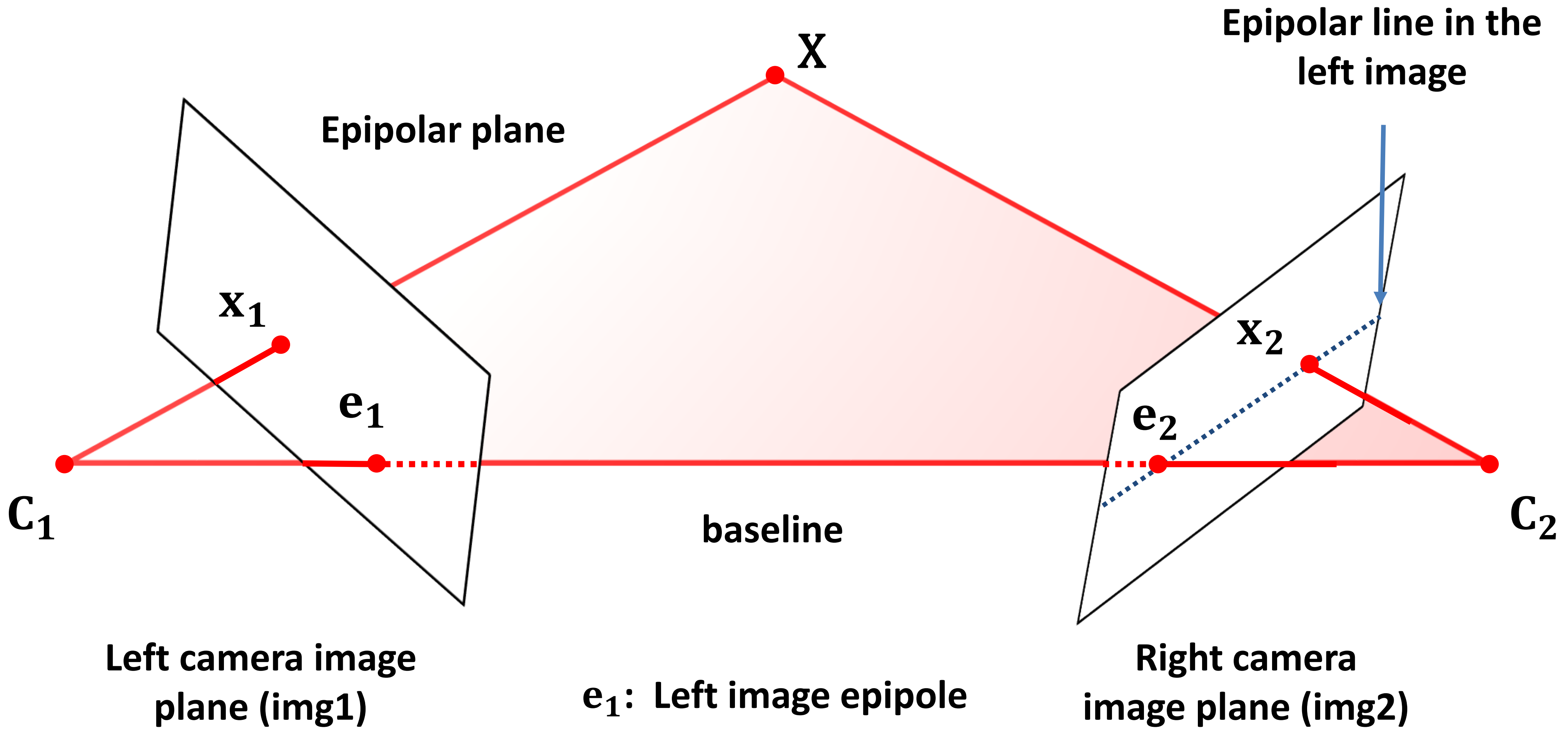
Fundamental Matrix Song

How do We Find 3D locations from Images?





Two View Geometry (Epipolar Geometry)



$\mathbf{e}_1$: Left image epipole

$\mathbf{e}_2$: Right image epipole

Two View Geometry

Cameras $\mathbf{P_1}$ and $\mathbf{P_2}$ such that

$$\mathbf{x_1} = \mathbf{P_1 X} \qquad \mathbf{x_2} = \mathbf{P_2 X}$$

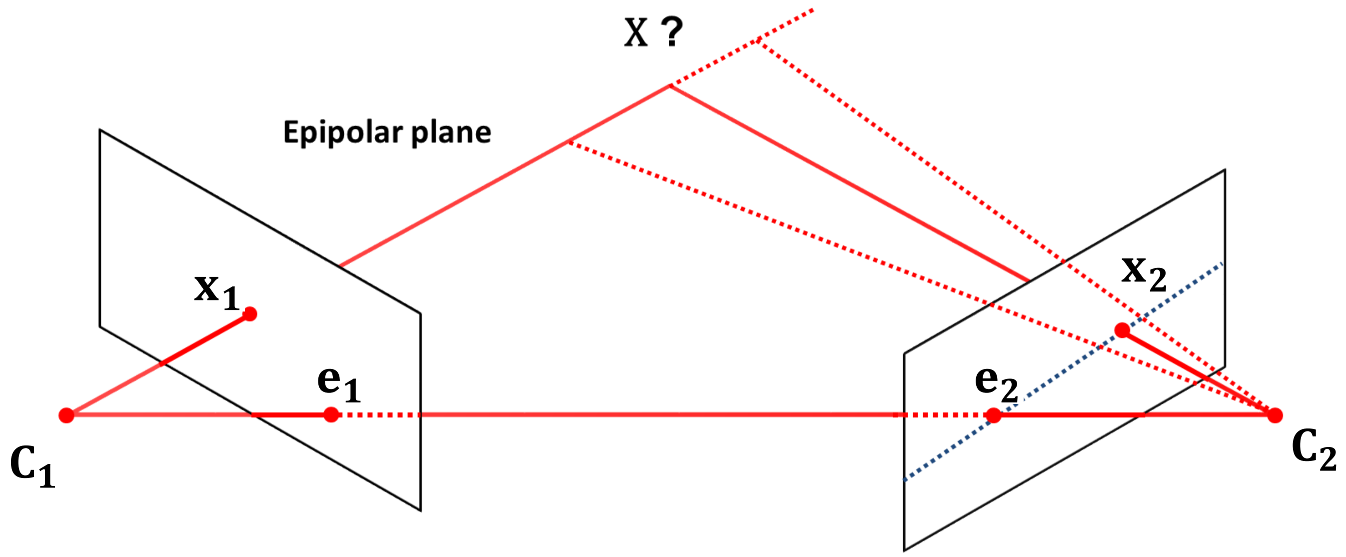
Baseline between the cameras is non-zero.

- Given an image point in the first view, where is the corresponding point in the second view?
- What is the relative position of the cameras?
- What is the 3D geometry of the scene?

Q. Can we use homoegraphy to find out the corresponding point?

Correspondence Geometry

Given the image of a point in one view, what can we say about its position in another?



A point in one image “generates” a line in the other image. This line is known as an epipolar line, and the geometry which gives rise to it is known as the epipolar geometry.

Fundamental Matrix

Given two camera looking at the same scenes, there exists a 3 x 3 matrix **F**, of rank 2 that captures the fundamental relationship between the pixel $\mathbf{x}_1$ and $\mathbf{x}_2$ in the two cameras for the same scene point **X**

$$\mathbf{x}_2^T F \mathbf{x}_1 = \mathbf{0}$$

We call **F** a <u>**fundamental matrix**</u>.

Matrix Multiplication of Cross Product

The vector cross product also can be expressed as the product of a skew-symmetric matrix and a vector:[11]

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

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

where superscript $^{\mathrm{T}}$ refers to the transpose operation, and $[\mathbf{a}]_{\times}$ is defined by:

$$[\mathbf{a}]_{\times} \stackrel{\text{def}}{=} \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}.$$

$[\mathbf{a}]_x$ is a 3x3 skew-symmetric matrix of rank 2.

a is the null-vector of $[a]_x$.

Skew-symmetric Matrix

```
a = rands(3,1); b = rands(3,1);

a_sk = zeros(3,3);
a_sk(1,2) = -a(3);  a_sk(2,1) =  a(3);
a_sk(1,3) =  a(2);  a_sk(3,1) = -a(2);
a_sk(2,3) = -a(1);  a_sk(3,2) =  a(1);

rank(a_sk)
```

```
ans = 2
```

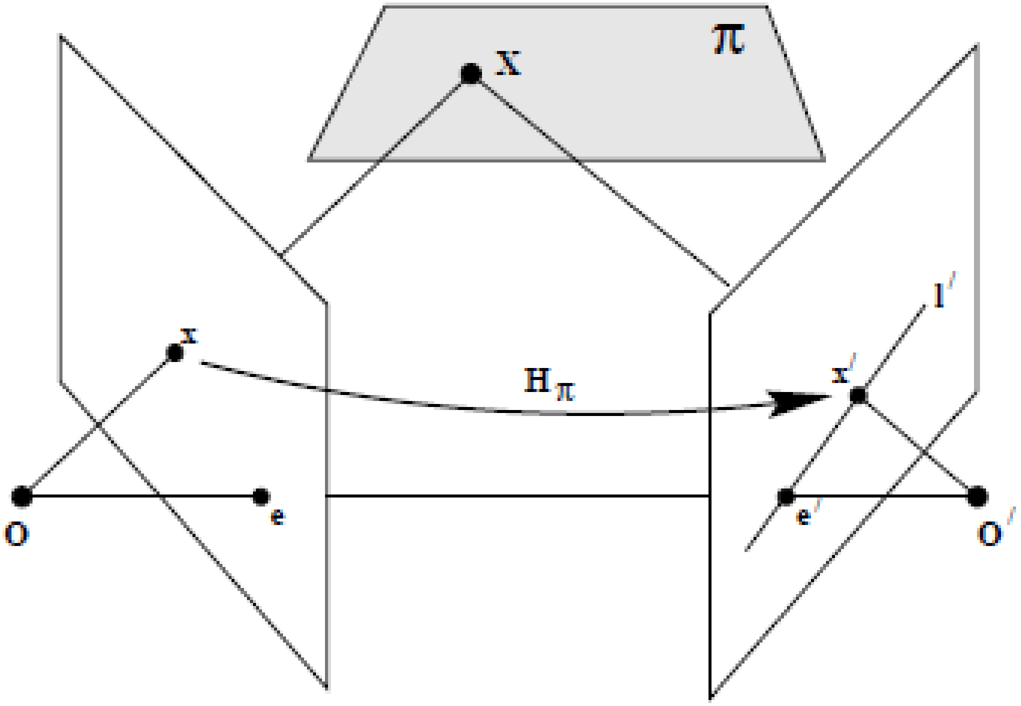
```
m1 = cross(a,b)
```

```
m1 = 3×1
   -1.1819
   -0.1121
    0.7808
```

```
m2 = a_sk*b
```

```
m2 = 3×1
   -1.1819
   -0.1121
    0.7808
```

Fundamental Matrix (Derivation)



Step 1. Point transfer via a plane

$$\mathbf{x}' = \mathrm{H}_{\pi}\mathbf{x}$$

Step 2. Construct the epipolar line

$$\mathbf{l}' = \mathbf{e}' \times \mathbf{x}' = [\mathbf{e}']_{\times}\mathbf{x}'$$

$$\mathbf{l}' = [\mathbf{e}']_{\times}\mathrm{H}_{\pi}\mathbf{x} = \mathbf{F}\mathbf{x}$$

$$\mathbf{F} = [\mathbf{e}']_{\times}\mathrm{H}_{\pi}$$

This shows that **F** is a 3 x 3 rank 2 matrix

rank(AB)≤min(rank(A),rank(B)).

https://math.stackexchange.com/questions/48989/how-to-prove-textrankab-leq-min-textranka-textrankb

Properties of F

1. F can be of great help in solving the stereo correspondence problem. This is the problem of finding the $\mathbf{x_2}$ in the second image that corresponds to a given $\mathbf{x_1}$ in the first image. If we know **F**, **we can confine our search to the line $\mathbf{l_2} = \mathbf{F}\mathbf{x_1}$ in the second image**, the corresponding pixel in the first image is on the epipolar line $\mathbf{l_1} = \mathbf{F}^\mathbf{T}\mathbf{x_2}$.

2. The determinant of **F** is always zero: det(**F**)=0. This follows from the fact that for all n x n matrices because it's a rank 2 matrix.

3. If **F** is the fundamental matrix for a given ordered pair of cameras, the fundamental matrix becomes $\mathbf{F}^\mathbf{T}$ if you reverse the order of the cameras.

- **Transpose of a product:** The transpose of the product of two matrices is equivalent to the product of their transposes in reversed order: $(AB)^T = B^T A^T$
- The same is true for the product of multiple matrices: $(ABC)^T = C^T B^T A^T$.

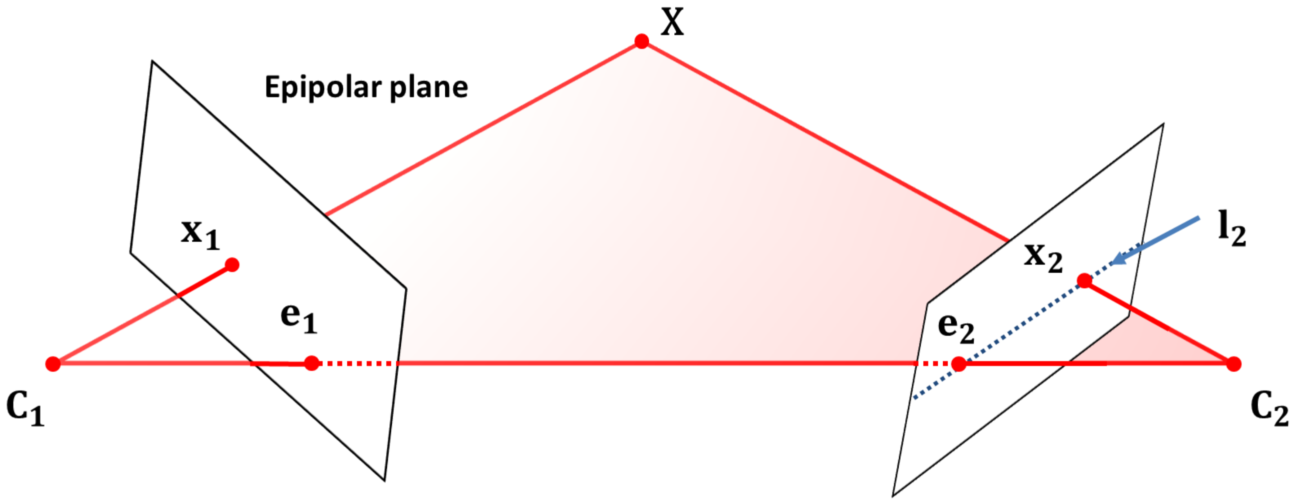
Properties of F (Continue)

4. The second-image epipole $\mathbf{e_2}$ is the left null-vector of **F** and the first-image epipole $\mathbf{e_1}$ is its right null vector:

$$\mathbf{e_2}^{\mathrm{T}}\mathbf{F} = 0 \text{ and } \mathbf{Fe_1} = 0.$$

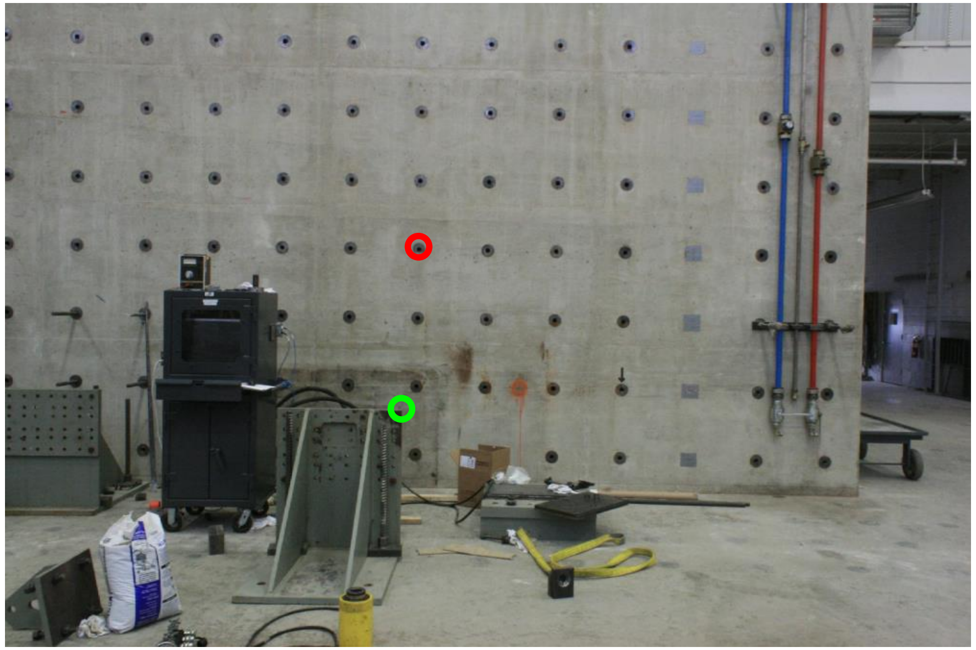
To prove $\mathbf{e_2}$ is the left null-vector, we note that $\mathbf{x_2}$ for a given $\mathbf{x_1}$ is on the right image line $\mathbf{l_2} = \mathbf{Fx_1}$. Since $\mathbf{e_2}$ is also on this line, we have $\mathbf{e}_2^T\mathbf{l_2} = \mathbf{e}_2^T\mathbf{Fx_1} = \mathbf{0}$. Since $\mathbf{e}_2^T\mathbf{Fx_1} = \mathbf{0}$ must be true for every pixel x in the first image, it must be the case that $\mathbf{e}_2^T\mathbf{F} = \mathbf{0}$.

$$\mathbf{x}_2^T \boldsymbol{F} \mathbf{x}_1 = \mathbf{0}$$

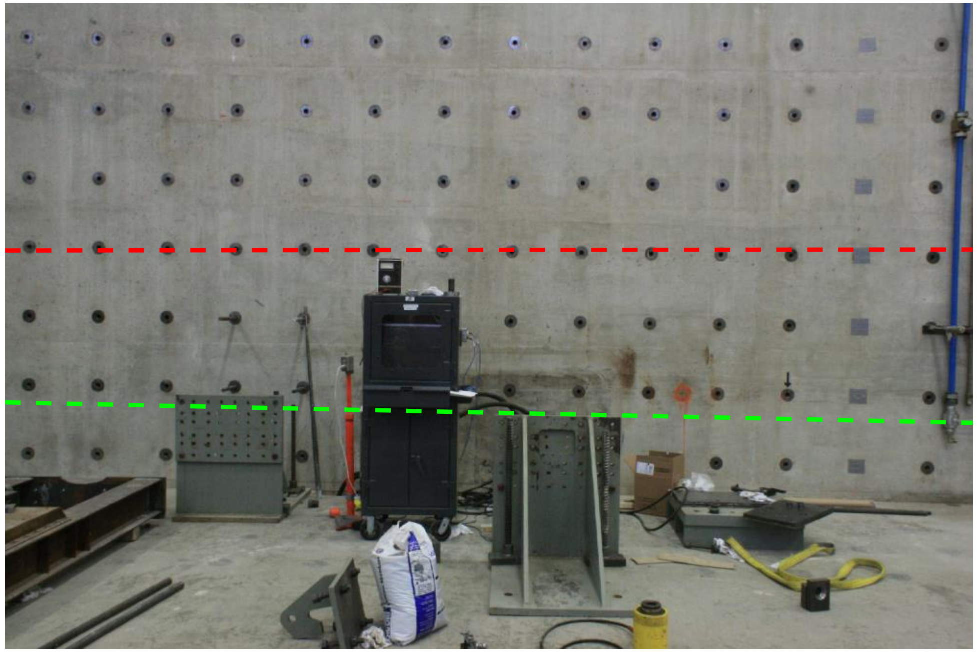


Example: Fundamental Matrix

Img1



Img2



$$\mathbf{x}_2^T F \mathbf{x}_1 = 0$$

Tutorial

Back-projecting an Image Pixel into World Frame

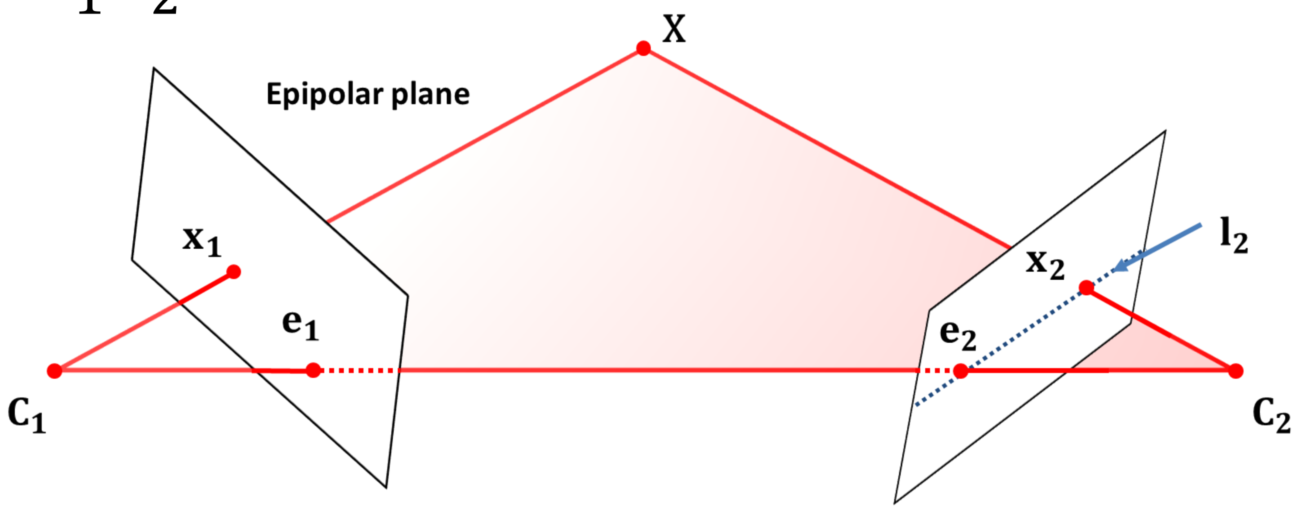
For a given pixel x, there exists a world point P^{+}x on the corresponding ray where $P^{+} = P^{T}\left(PP^{T}\right)^{-1}$ (Pseudoinverse of P). This claim in the based on the observation that the location of the image of this world point is the same as x:

$$PX = P(P^{+}x) = P\left(P^{T}\left(PP^{T}\right)^{-1}x\right) = PP^{T}\left(PP^{T}\right)^{-1}x=x$$

Since P is of rank 3, the 3 x 3 matrix PP^{T} is of full rank. The inverse is $\left(PP^{T}\right)^{-1}$therefore guaranteed to exist.

Relationship Between Fundamental Matrix and Projection Matrix

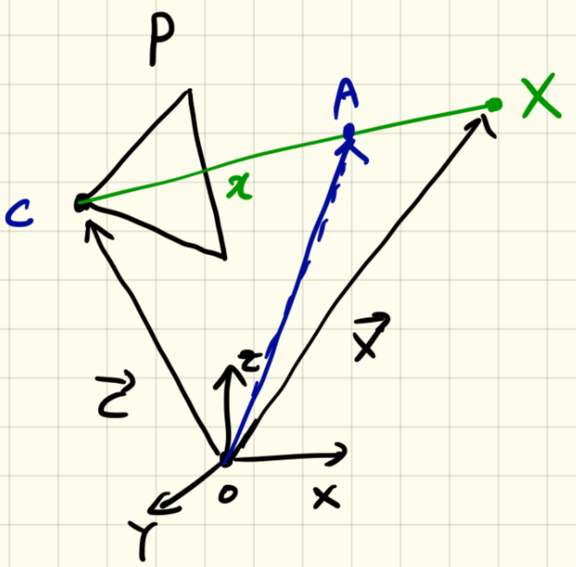
By construction, $e_2 = P_2 C_1$ and $e_1 = P_1 C_2$



Two points on l_2: the epipole e_2 and the pixel at $P_2(P_1^+ x_1)$. Therefore, $l_2 = e_2 \times P_2(P_1^+ x_1) = [e_2]_\times P_2 P_1^+ x_1$. Therefore, $l_2 = [e_2]_\times P_2 P_1^+ x_1 = \mathbf{F} x_1$ where $\mathbf{F} = [e_2]_\times P_2 P_1^+$.

$$\mathbf{F} = [e_2]_\times P_2 P_1^+$$

Supplement



P : a 3×4 matrix.

$\therefore$ there must exist at least one null vector.

$$\vec{CX} = X - C$$

$$A = \lambda(X - C) + C$$

$$= \lambda_1 X + \lambda_2 C$$

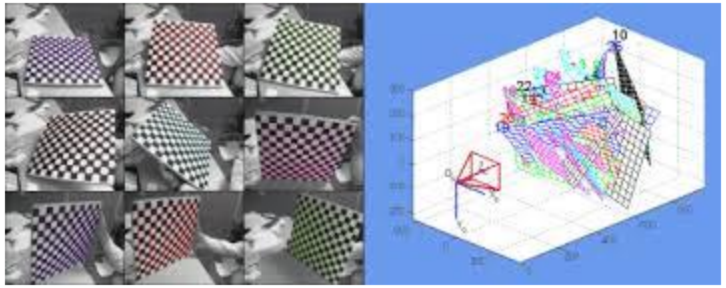
$$PX = PA = x$$

$$PA = P(\lambda_1 X + \lambda_2 C) = \underbrace{\lambda_1 PX}_{x} + \lambda_2 PC = x$$

$$\therefore PC = 0$$

a null vector of P

Topics will not be Covered in This Course.



Camera Calibration: Estimate interior camera parameters in K including focal length, principal points, and lens distortion.

$$\mathbf{x} = \mathbf{K}[\mathbf{R} \quad \mathbf{t}]\mathbf{X} \qquad \mathbf{K} = \begin{bmatrix} f & 0 & p_y \\ 0 & f & p_x \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{x}_2^T \boldsymbol{K}^{-T} \boldsymbol{F} \boldsymbol{K} \mathbf{x}_1 = \mathbf{0}$$

Essential matrix and projection matrix estimation for essential matrix: A projection matrix from two views can be extracted from essential matrix.

It has been seen that a pair of camera matrices determines a unique fundamental matrix. This mapping is not injective (one-to-one) however, since pairs of camera matrices that differ by a projective transformation give rise to the same fundamental matrix. …. (H&Z, 254p)

Result 9.8. *If* H *is a* 4×4 *matrix representing a projective transformation of 3-space, then the fundamental matrices corresponding to the pairs of camera matrices* $(\mathrm{P}, \mathrm{P}')$ *and* $(\mathrm{PH}, \mathrm{P}'\mathrm{H})$ *are the same.*

Once the essential matrix is known, the camera matrices may be retrieved from E. In contrast with the fundamental matrix case, where there is a projective ambiguity, the camera matrices may be retrieved from the essential matrix up to scale and a four-fold ambiguity. That is there are four possible solutions, except for overall scale, which cannot be determined. (H&Z, 258p)

Estimating F

- If we don't know $\mathbf{K}_1$, $\mathbf{K}_2$, $\mathbf{R}$, or $\mathbf{t}$, can we estimate $\mathbf{F}$ for two images?

- Yes, given enough correspondences

Estimating F: 8 Point Algorithm

The fundamental matrix F is defined by

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

for any pair of matches x and x' in two images.

$$\mathbf{F} = \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}$$

Let x=$(u,v,1)^T$ and x'=$(u',v',1)^T$,

each match gives a linear equation

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

Tutorial

8 Point Algorithm

$$\begin{bmatrix} u_1u_1' & v_1u_1' & u_1' & u_1v_1' & v_1v_1' & v_1' & u_1 & v_1 & 1 \\ u_2u_2' & v_2u_2' & u_2' & u_2v_2' & v_2v_2' & v_2' & u_2 & v_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_nu_n' & v_nu_n' & u_n' & u_nv_n' & v_nv_n' & v_n' & u_n & v_n & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = 0$$

The solution will be a null vector of A

8-point Algorithm (Continue)

- **F** should have rank 2
- To enforce that **F** is of rank 2, **F** is replaced by **F'** that minimizes $\|\mathbf{F} - \mathbf{F}'\|$ subject to the rank constraint.
- This is achieved by SVD. Let $\mathbf{F} = \mathbf{U\Sigma V^T}$, where

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix} \qquad \Sigma' = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

then $\mathbf{F} = \mathbf{U\Sigma' V^T}$ is the solution.

F-matrix (Enforcing Rank 2)

```
FT = rands(3,3);
[U,D,V] = svd(FT);
rank(FT)
```

```
ans =
     3
```

```
F = U*diag([D(1,1) D(2,2) 0])*V';
[U,D,V] = svd(F);
rank(F)
```

```
ans =
     2
```

Tutorial

Singular Value Decomposition

$$\underset{m\times m}{\begin{bmatrix} U_1 & U_2 & \cdots & U_m \end{bmatrix}}_{U} \underset{m\times n}{\begin{bmatrix} \sigma_1 & & & | \\ & \sigma_2 & & 0 \\ & & \ddots & \\ & & \sigma_m & | \end{bmatrix}} \underset{n\times n}{\begin{bmatrix} V_1 \\ V_2 \\ \vdots \\ V_n \end{bmatrix}}_{V} = M.$$

Ex.

$$\begin{bmatrix} U_{11} & U_{12} & U_{13} \\ U_{21} & U_{22} & U_{23} \\ U_{31} & U_{32} & U_{33} \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} U_1 & U_2 & U_3 \end{bmatrix} \begin{bmatrix} \sigma_1 & \\ & \sigma_2 \\ -0 & - \end{bmatrix}$$

$$= \begin{bmatrix} \sigma_1 U_{11} & \sigma_2 U_{12} \\ \sigma_1 U_{21} & \sigma_2 U_{22} \\ \sigma_1 U_{31} & \sigma_2 U_{32} \end{bmatrix} = \begin{bmatrix} \sigma_1 U_1 & \sigma_2 U_2 \end{bmatrix}$$

$$M = \begin{bmatrix} \sigma_1 U_1, & \sigma_2 U_2 & \cdots & \sigma_m U_m, & -0- \end{bmatrix} \begin{bmatrix} V_1 \\ \vdots \\ V_n \end{bmatrix}$$

$$= \sigma_1 U_1 V_1 + \sigma_2 U_2 V_2 \cdots + \sigma_m U_m V_m$$

Here, U_i : $m \times 1$

V_i : $1 \times n$

$m < n$.

Let's pick one vector from $V_{m+1} \cdots V_n$ (V_i)

$$MV_i = \sigma_1 U_1 \underbrace{V_1 \cdot V_i}_{0} + \sigma_2 U_2 \underbrace{V_2 \cdot V_i}_{0} + \cdots + \sigma_m U_m \underbrace{V_m \cdot V_i}_{0}$$

(because there are orthogonal basis.)

$= 0$

$\therefore$ $V_{m+1} \cdots V_n$ is null vectors of M

Rank 2 and Rank 3 Matrix

```
M1 = cat(2,[1;0;0], [0;1;0], [1;1;0])
M2 = cat(2,[1;0;0], [0;1;0], [1;1;0.001])

[~,D1,~] = svd(M1)
rank(M1)

[~,D2,~] = svd(M2)
rank(M2)
```

```
M1 = 3×3
     1     0     1
     0     1     1
     0     0     0

M2 = 3×3
    1.0000         0    1.0000
         0    1.0000    1.0000
         0         0    0.0010

D1 = 3×3
    1.7321         0         0
         0    1.0000         0
         0         0         0

ans = 2
D2 = 3×3
    1.7321         0         0
         0    1.0000         0
         0         0    0.0006

ans = 3
```

Tutorial

Example: 8 Point Algorithm (Experiment)

```
% synthetic projection matrix creation
P1 = [eye(3,3) zeros(3,1)];
P2 = eye(3,3)*[rotx(10)*roty(20)*rotz(30) [5;5;1]];

% synthetic 100 numbers of 3D points (X)
nPt = 100;
X = rand(4, nPt);

% images points corresponding to each X
x1 = P1*X; x1 = bsxfun(@rdivide, x1(1:2,:), x1(3,:));
x2 = P2*X; x2 = bsxfun(@rdivide, x2(1:2,:), x2(3,:));

% 8point algorithm
funRow = @(u,v,up,vp) [u*up v*up up u*vp v*vp vp u v 1];

% pick 8 points
A = zeros(8,9);
id = randperm(nPt,8);
for ii=1:8
   A(ii,:) = funRow(x1(1,id(ii)), x1(2,id(ii)), x2(1,id(ii)), x2(2,id(ii)));
end

[~, ~, V] = svd(A);
F = reshape(V(:,9), 3, 3)';

% enforcing rank 2
[U, D, V] = svd(F);
F = U*diag([D(1,1) D(2,2) 0])*V';
```

```
% test x'*F*x = 0
xFx = zeros(100,1);
for ii=1:100
    l = [x2(:,ii);1]'*F;
    dist = abs(l(1)*x1(1,ii) + l(2)*x1(2,ii) + l(3))/norm(l(1:2));
    xFx(ii) = dist;
end
mean(xFx)

% Compute Fundamental Matrix from projection matrices
C1 = null(P1);
e2 = P2*C1;
e2x = [0 -e2(3) e2(2);e2(3) 0 -e2(1);-e2(2) e2(1) 0];
FNew = e2x * P2 * pinv(P1); % a fundamental matrix from projection matrieces

disp(F./F(3,3))
disp(FNew./FNew(3,3))
```

```
ans =

    9.102e-14
```

```
  0.62083    -0.30515    -1.8964
  -0.7277     0.81704     1.6964
  0.53437     -2.5594          1
```

```
  0.62083    -0.30515    -1.8964
  -0.7277     0.81704     1.6964
  0.53437     -2.5594          1
```

$$\mathbf{F} = [\mathrm{e}_2]_{\times} \mathrm{P}_2 \mathrm{P}_1^{+}$$

Problem with 8-point algorithm

$$\begin{bmatrix} u_1u_1' & v_1u_1' & u_1' & u_1v_1' & v_1v_1' & v_1' & u_1 & v_1 & 1 \\ u_2u_2' & v_2u_2' & u_2' & u_2v_2' & v_2v_2' & v_2' & u_2 & v_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_nu_n' & v_nu_n' & u_n' & u_nv_n' & v_nv_n' & v_n' & u_n & v_n & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = 0$$

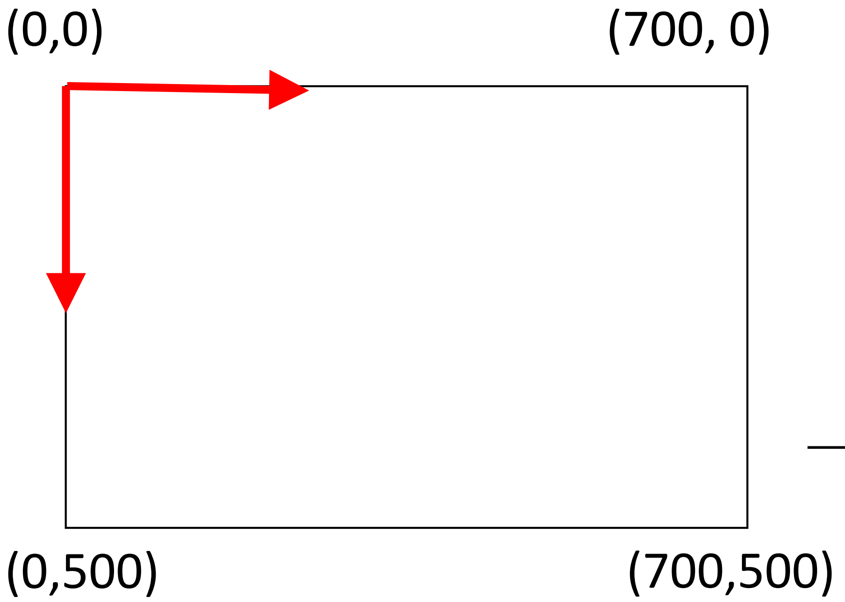
$$\begin{array}{ccccccccc} \sim 1000k & \sim 1000k & \sim 1k & \sim 1k & \sim 1000k & \sim 1k & \sim 1k & \sim 1k & 1 \end{array}$$

Orders of magnitude difference
between column of data matrix
Instability

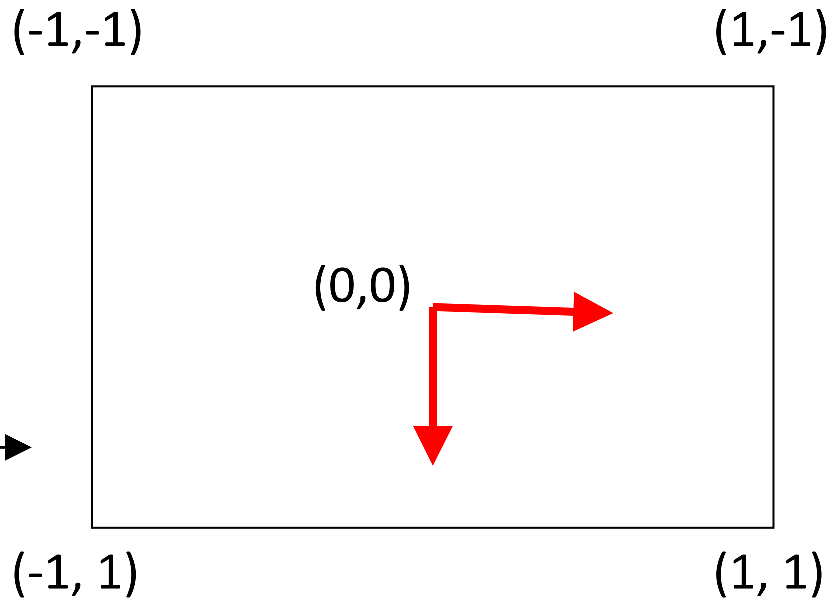
Normalized 8-point Algorithm

Normalized least squares yields good results

Transform image to become [-1,1] x [-1,1]



$$\begin{bmatrix} \frac{2}{700} & 0 & -1 \\ & \frac{2}{500} & -1 \\ & & 1 \end{bmatrix}$$



1. Transform input by $\hat{\mathbf{x}}_i = \mathbf{T}\mathbf{x}_i$ and $\hat{\mathbf{x}}_i{}' = \mathbf{T}'\mathbf{x}_i'$ (T and T' are different when the size of the images are different)

2. Call 8-point on to obtain $\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_i{}'$ to obtain $\hat{\mathbf{F}}$

3. $\mathbf{F} = \mathbf{T}'^{\mathrm{T}}\hat{\mathbf{F}}\mathbf{T}$

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

$$\underbrace{\boxed{\mathbf{x}_i'^{\mathrm{T}}\mathbf{T}^{\mathrm{T}}}\hat{\mathbf{F}}\boxed{\mathbf{T}\mathbf{x}_i}}_{\mathbf{F}} = 0$$

Normalized 8-point Algorithm (MATLAB)

```
clear; close all; clc;
rng(100);

% synthetic projection matrix creation
K = [4000 0 2500; 0 4000 2500; 0 0 1]; % interior matrix (5000 x 5000) and focal length 4000
P1 = [K zeros(3,1)];
P2 = K*[rotx(10)*roty(20)*rotz(30) [-1;-1;1]];

% synthetic 100 numbers of 3D points (X)
nPt = 100;
X = rand(4, nPt);

% images points corresponding to each X
x1 = P1*X; x1 = bsxfun(@rdivide, x1(1:2,:), x1(3,:));
x2 = P2*X; x2 = bsxfun(@rdivide, x2(1:2,:), x2(3,:));

% randomly pick 8 points
id = randperm(nPt,8);

% add perturbation
x1(1, id(1)) = x1(1, id(1)) + 10; % 10 pixel error
x1(2, id(1)) = x1(2, id(1)) + 10; % 10 pixel error

% x1 = x1 + 5*randn(1, nPt); % random error perturbation
% x2 = x2 + 5*randn(1, nPt); % random error perturbation

funRow = @(u,v,up,vp) [u*up v*up up u*vp v*vp vp u v 1];

% 8 point algorithm
A = zeros(8,9);
for ii=1:8
   A(ii,:) = funRow(x1(1,id(ii)), x1(2,id(ii)), x2(1,id(ii)), x2(2,id(ii)));
end

[~, ~, V] = svd(A);
F = reshape(V(:,9), 3, 3)';
[U, D, V] = svd(F);
F = U*diag([D(1,1) D(2,2) 0])*V';
```

Perturbation

```
% test transpose(x')*F*x = 0
xFx = zeros(100,1);
for ii=1:100
    l = [x2(:,ii);1]'*F;
    dist = abs(l(1)*x1(1,ii) + l(2)*x1(2,ii) + l(3))/norm(l(1:2));
    xFx(ii) = dist;
end

% normalize 8 point algorithm
T1 = [2/5000 0 -1; 0 2/5000 -1; 0 0 1];
T2 = [2/5000 0 -1; 0 2/5000 -1; 0 0 1];

x1T = T1*[x1;ones(1,nPt)]; x1T = bsxfun(@rdivide, x1T(1:2,:), x1T(3,:));
x2T = T2*[x2;ones(1,nPt)]; x2T = bsxfun(@rdivide, x2T(1:2,:), x2T(3,:));

A = zeros(8,9);
for ii=1:8
   A(ii,:) = funRow(x1T(1,id(ii)), x1T(2,id(ii)), x2T(1,id(ii)), x2T(2,id(ii)));
end

[~, ~, V] = svd(A);
FT = reshape(V(:,9), 3, 3)';
[U, D, V] = svd(FT);
FT = U*diag([D(1,1) D(2,2) 0])*V';

F = T2'*FT*T1;

% test transpose(x')*F*x = 0
xFx_norm = zeros(100,1);
for ii=1:100
    l = [x2(:,ii);1]'*F;
    dist = abs(l(1)*x1(1,ii) + l(2)*x1(2,ii) + l(3))/norm(l(1:2));
    xFx_norm(ii) = dist;
end

% Averaging errors from xFx_norm is much smaller than ones from xFx
mean(xFx)
mean(xFx_norm)
```

```
ans = 0.3192

ans = 0.0899
```

Slide Credits and References

- Lecture notes: Robert Collins
- Lecture notes: Avinash Kak
- Lecture notes: Noah Snavely
- Lecture notes: Richard Hartely and Andrew Zisserman
- Hartley, Richard, and Andrew Zisserman. Multiple view geometry in computer vision. Cambridge university press, 2003.