Epipolar Geometry and the Essential Matrix
Inferring 3D from 2D

- We have done model based pose estimation

- Single (calibrated) camera

- Known model

- Can determine the pose of the model wrt camera
Inferring 3D from 2D

- Another method is structure-from-motion
  - Might be better termed “structure-and-motion from a moving camera”

One (calibrated) moving camera

Arbitrary scene

Relative pose between camera positions is unknown

-> Can determine the positions of points in the scene, as well as the motion of the camera (R,t)

However, we will see that the positions of points and the translation of the camera have an unknown scale factor
Outline for today

• First we review the geometry and representation of epipolar lines
• Next we derive the essential matrix and show how it can predict the locations of epipolar lines
• Next we show how the essential matrix can be estimated from point correspondences

• Next week: getting rotation and translation from the essential matrix
Epipolar Geometry

- We have two views of a scene, taken from different viewpoints
- We see an image point \( p \) in one image, which is the projection of a 3D point

Given \( p_0 \) in the first image, where can the corresponding point \( p_1 \) in the second image be?
The optical centers of the two cameras, a point $P$, and the image points $p_0$ and $p_1$ of $P$ all lie in the same plane (epipolar plane).

These vectors are co-planar: $\overrightarrow{C_0p_0}$, $\overrightarrow{C_1p_1}$, $\overrightarrow{C_0C_1}$
Another way to write the fact they are co-planar is

\[ \overrightarrow{C_0p_0} \cdot \left( \overrightarrow{C_0C_1} \times \overrightarrow{C_1p_1} \right) = 0 \]
• Now, instead of treating $p_0$ as a point, treat it as a 3D direction vector*

$$p_0 = \begin{pmatrix} x_0 \\ y_0 \\ 1 \end{pmatrix}$$

This is defined with respect to the coordinate frame of camera 0

We assume “normalized” image coordinates; ie effective focal length=1

• $p_1$ is also a direction vector

$$p_1 = \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix}$$

This is defined with respect to the coordinate frame of camera 1

• The direction of $p_1$ in camera 0 coordinates is

$$C_0 \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix}$$

Namely, we apply the rotation matrix from the camera 1 to camera 0 pose

* A direction vector is a vector whose starting point (tail) doesn’t matter, just its direction
So we can write the coplanar constraint as
\[ \mathbf{p}_0 \cdot (\mathbf{t} \times \mathbf{Rp}_1) = 0 \]

Where \( \mathbf{R} \) is the rotation of camera 1 wrt camera 0
\[ \begin{pmatrix} C_0 \\ C_1 \end{pmatrix} \mathbf{R} \]

And \( \mathbf{t} \) is the translation of the camera 1 origin wrt camera 0
\[ \begin{pmatrix} C_0 \\ C_1 \end{pmatrix} \mathbf{t}_{C_{1\text{org}}} \]

Remember that the pose of camera 1 wrt camera 0 is
\[ \begin{pmatrix} C_0 \\ C_1 \end{pmatrix} \mathbf{H} = \begin{pmatrix} C_0 \mathbf{R} & C_0 \mathbf{t}_{C_{1\text{org}}} \\ 0 & 1 \end{pmatrix} \]
Cross Product as Matrix Multiplication

• The cross product of a vector \( \mathbf{a} \) with a vector \( \mathbf{b} \), \( \mathbf{a} \times \mathbf{b} \), can be represented as a 3x3 matrix times the vector \( \mathbf{b} \):
  – \( [\mathbf{a}]_x \mathbf{b} \), where \( [\mathbf{a}]_x \) is a skew symmetric matrix

• It is easy to show that

\[
[\mathbf{a}]_x = \begin{pmatrix}
0 & -a_3 & a_2 \\
-a_3 & 0 & -a_1 \\
-a_2 & a_1 & 0
\end{pmatrix}
\]

(Show this!)
Cross Product as Matrix Multiplication

- The cross product of a vector \( \mathbf{a} \) with a vector \( \mathbf{b} \), \( \mathbf{a} \times \mathbf{b} \), can be represented as a 3x3 matrix times the vector \( \mathbf{b} \):
  - \( \mathbf{[a]} \mathbf{x} \mathbf{b} \), where \( \mathbf{[a]} \mathbf{x} \) is a skew symmetric matrix

\[
\begin{bmatrix}
0 & -a_3 & a_2 \\
a_3 & 0 & -a_1 \\
-a_2 & a_1 & 0 \\
\end{bmatrix}
\]

(Show this!)

- It is easy to show that

\[
\mathbf{a} \times \mathbf{b} = \begin{bmatrix}
a_1 & a_2 & a_3 \\
a_3 & b_1 & b_2 \\
a_2 & b_3 & a_3 \\
\end{bmatrix} = \begin{bmatrix}a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{bmatrix}
\]
Matrix Form of Epipolar Constraint

• We have
  \[ \mathbf{p}_0 \cdot (\mathbf{t} \times \mathbf{Rp}_1) = 0 \]
  [Note: \( \times \) denotes the cross product.]

• Or
  \[ \mathbf{p}_0^T [\mathbf{t}]_x \mathbf{Rp}_1 = 0 \]
  [Note: \([\mathbf{t}]_x\) is the skew-symmetric matrix corresponding to \(\mathbf{t}\).]

• Let \( \mathbf{E} = [\mathbf{t}]_x \mathbf{R} \) (which is a 3x3 matrix)

• Then
  \[ \mathbf{p}_0^T \mathbf{E} \mathbf{p}_1 = 0 \]
  \[ \begin{pmatrix} x_0 & y_0 & 1 \end{pmatrix} \begin{pmatrix} E_{11} & E_{12} & E_{13} \\ E_{21} & E_{22} & E_{23} \\ E_{31} & E_{32} & E_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} = 0 \]

where \([\mathbf{t}]_x\) is the 3x3 skew symmetric matrix for \(\mathbf{t}\) corresponding to the cross product operator.
The Essential Matrix

- Is the matrix $E$, that relates the image of a point in one camera to its image in the other camera, given a translation and rotation

$$\mathbf{p}_0^T \mathbf{E} \mathbf{p}_1 = 0$$

- where

$$\mathbf{E} = [\mathbf{t}] \mathbf{R}$$

- Every point in one image is related to its corresponding point in the other image via the same matrix $\mathbf{E}$

- It’s a good way to check for correct correspondences
Example – Create a Scene

• Create some points on the face of cube
• Render image from two views

• Let pose of cube with respect to camera 1 be
  \( ax=120^\circ, ay=0^\circ, az=60^\circ, tx=3, ty=0, tz=0 \)

• Let pose of camera 2 with respect to camera 1 be
  \( ax=0^\circ, ay=-25^\circ, az=0^\circ, tx=3, ty=0, tz=1 \)

• Assume XYZ fixed angles
clear all

L = 300; % size of image in pixels
I1 = zeros(L,L);

f = L;
u0 = L/2;
v0 = L/2;

% Create the matrix of intrinsic camera parameters
K = [ f  0  u0;
     0  f  v0;
     0  0   1];

DEG_TO_RAD = pi/180;

% Create some points on the face of a cube
P_M = [
    0 0 0 0 0 0 0 0 0 1 2 1 2 1 2;
    2 1 0 2 1 0 2 1 0 0 0 0 0 0;
    0 0 0 -1 -1 -1 -2 -2 0 0 -1 -1 -2 -2;
    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ];
NPTS = length(P_M);

% Define pose of model with respect to camera
ax = 120 * DEG_TO_RAD;
ay = 0 * DEG_TO_RAD;
az = 60 * DEG_TO_RAD;

Rx = [ 1   0            0;
       0  cos(ax)  -sin(ax);
       0  sin(ax)   cos(ax) ];
Ry = [ cos(ay)    0   sin(ay);
       0           1   0;
       -sin(ay)   0    cos(ay)];
Rz = [ cos(az)  -sin(az)   0;
       sin(az)  cos(az)   0;
       0           0    1 ];
R_m_c1 = Rx * Ry * Rz;
Pmorg_c1 = [0; 0; 5]; % translation of model wrt camera

M = [ R_m_c1  Pmorg_c1 ]; % Extrinsic camera parameter matrix
Create points
(2 of 3)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Render image 1
p1 = M * P_M;

p1(1,:) = p1(1,:) ./ p1(3,:);
p1(2,:) = p1(2,:) ./ p1(3,:);
p1(3,:) = p1(3,:) ./ p1(3,:);

u1 = K * p1;  % Convert image points from normalized to unnormalized
for i=1:length(u1)
    x = round(u1(1,i));   y = round(u1(2,i));
    I1(y-2:y+2, x-2:x+2) = 255;
end
figure(1), imshow(I1, []), title('View 1');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set up second view.
% Define rotation of camera1 with respect to camera2
ax = 0 * DEG_TO_RAD;
ay = -25 * DEG_TO_RAD;
az = 0;

Rx = [ 1 0 0;
0 cos(ax) -sin(ax);
0 sin(ax) cos(ax) ];
Ry = [ cos(ay) 0 sin(ay);
0 1 0;
-sin(ay) 0 cos(ay) ];
Rz = [ cos(az) -sin(az) 0;
sin(az) cos(az) 0;
0 0 1 ];
R_c2_c1 = Rx * Ry * Rz;

% Define translation of camera2 with respect to camera1
Pc2org_c1 = [3; 0; 1];

% Figure out pose of model wrt camera 2.
H_m_c1 = [ R_m_c1 Pmorg_c1 ; 0 0 0 1];
H_c2_c1 = [ R_c2_c1 Pc2org_c1 ; 0 0 0 1];
H_cl_c2 = inv(H_c2_c1);
H_m_c2 = H_cl_c2 * H_m_c1;
R_m_c2 = H_m_c2(1:3,1:3);
Pmorg_c2 = H_m_c2(1:3,4);

% Extrinsic camera parameter matrix
M = [ R_m_c2 Pmorg_c2 ];
% Render image 2
I2 = zeros(L,L);
p2 = M * P_M;

p2(1,:) = p2(1,:) ./ p2(3,:);
p2(2,:) = p2(2,:) ./ p2(3,:);
p2(3,:) = p2(3,:) ./ p2(3,:);

% Convert image points from normalized to unnormalized
u2 = K * p2;
for i=1:length(u2)
    x = round(u2(1,i));   y = round(u2(2,i));
    I2(y-2:y+2, x-2:x+2) = 255;
end
figure(2), imshow(I2, []), title('View 2');

disp('Points in image 1:');
disp(u1);
disp('Points in image 2:');
disp(u2);
imwrite(I1, 'I1.tif');
imwrite(I2, 'I2.tif');

% This is the "true" essential matrix between the views
t = Pc2org_c1;
E = [ 0 -t(3) t(2); t(3) 0 -t(1); -t(2) t(1) 0 ] * R_c2_c1;
disp('True essential matrix:');
disp(E);

save('u1.mat', 'u1');   % Save points to files
save('u2.mat', 'u2');
save('E.mat', 'E');     % Save to file
\[
R_{m\_c1} = \\
\begin{bmatrix}
0.5000 & -0.8660 & 0 \\
-0.4330 & -0.2500 & -0.8660 \\
0.7500 & 0.4330 & -0.5000 \\
\end{bmatrix}
\]

\[
P_{morg\_c1} = \\
\begin{bmatrix}
0 \\
0 \\
5 \\
\end{bmatrix}
\]

\[
R_{c2\_c1} = \\
\begin{bmatrix}
0.9063 & 0 & -0.4226 \\
0 & 1.0000 & 0 \\
0.4226 & 0 & 0.9063 \\
\end{bmatrix}
\]

\[
P_{c2org\_c1} = \\
\begin{bmatrix}
3 \\
0 \\
1 \\
\end{bmatrix}
\]

\[
E = \\
\begin{bmatrix}
0 & -1.0000 & 0 \\
-0.3615 & 0 & -3.1415 \\
0 & 3.0000 & 0 \\
\end{bmatrix}
\]
Example 1

- A camera translates to its right (in the positive X direction) by 1 m, and down (in the positive Y direction) by 0.5 m.
  - Give the 3x3 essential matrix that relates these two views.

\[
\text{The essential matrix is } E = [t]_x R,
\]

where

- \([t]_x\) is the skew symmetric matrix corresponding to \(t\)
- \(t\) is the translation of camera 2 with respect to camera 1; i.e., \(c_1P_{c2\text{org}}\)
- \(R\) is rotation of camera 2 with respect to camera 1; i.e., \(c_1c_2R\)
Example 2

• An essential matrix $E$ relates two images, and has the values given below.

\[
E = \begin{bmatrix}
0 & 0 & 0 \\
0.5 & 0 & -0.8660 \\
0 & 1.0000 & 0
\end{bmatrix}
\]

– A point is observed in the second image, at (normalized) image coordinates $p_2 = (0.1, 0.1)$. Which one of the following points in the first image could possibly be the correct corresponding point?

(i) $p_1 = (0.1, 0.5)$  
(ii) $p_1 = (0.2, 0.1225)$  
(iii) $p_1 = (0.866, 0)$
Example 3

Which of the following camera motions are consistent with the essential matrix?

(i) \( t = [1; 0; 0] \), \( R = \begin{bmatrix} 1 & 0 & 0; & 0 & 1 & 0; & 0 & 0 & 1 \end{bmatrix} \)

(ii) \( t = [0; 1; 0] \), \( R = \begin{bmatrix} 0.8660 & 0 & -0.5; & 0 & 1.0 & 0; & 0.5 & 0 & 0.8660 \end{bmatrix} \)

(iii) \( t = [1; 0; 0] \), \( R = \begin{bmatrix} 0.8660 & 0.5; & 0 & 1.0 & 0; & -0.5 & 0 & 0.8660 \end{bmatrix} \)

\[
E = \begin{bmatrix} 0 & 0 & 0 \\ 0.5 & 0 & -0.8660 \\ 0 & 1.0000 & 0 \end{bmatrix}
\]
Representation of a line

• Equation of a line in the \((x,y)\) plane is \(ax + by + c = 0\)
  – There are three parameters \((a,b,c)^T\)

• Note - we get the same line with the equation
  \((ka)x + (kb)y + (kc) = 0\), for any non-zero constant \(k\)
  – So these are actually **homogeneous** coordinates, meaning they are only known up to a scale factor

• So a line is represented by the homogeneous coordinates
  \(l = (a,b,c)^T\)

• A point \(p\) lies on the line \(l\) if and only if \(p^T l = 0\)

\[
p^T l = \begin{pmatrix} x & y & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = ax + by + c = 0
\]
Epipolar Lines

• Recall $p_0^T E p_1 = 0$

• So $E p_1$ is the epipolar line corresponding to $p_1$ in the camera 0 image

• Or, writing another way,

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} x_0 & y_0 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = ax_0 + by_0 + c = 0$$

where $l = (a,b,c)^T = E p_1$ are the parameters of the line
Visualization

\[ p_0^T E p_1 = 0 \]

• Pick a point \( p_1 \) in the second image
• Calculate the corresponding epipolar line in the first image, \( l = E p_1 \)
  – where \( l = (a, b, c) \)
  – \( ax_0 + by_0 + c = 0 \) is the equation of the line
• A point \( p_0 \) lies on the line if \( p_0^T E p_1 = 0 \)
  – It is unlikely to be exactly zero
  – The residual error is \( |p_0^T E p_1| \)
• To draw the line on the first image,
  – Find two points \((x_a, y_a)\) and \((x_b, y_b)\) on the line, and draw a line between them
  – Let \( x_a = -1 \), solve for \( y_a = (-c - a*x_a)/b \)
  – Let \( x_b = +1 \), solve for \( y_b = (-c - a*x_b)/b \)
% Draw epipolar lines, from the essential matrix.
clear all
close all

I1 = imread('I1.tif');
I2 = imread('I2.tif');

% intrinsic camera parameters
K = [ 300  0  150;
     0  300  150;
     0  0   1];

load E % Read in essential matrix
load u1 % Read in unnormalized image points from image 1
load u2 % Read in unnormalized image points from image 2

% Display images
subplot(1,2,1), imshow(I1, []), title('View 1');
for i=1:length(u1)
    rectangle('Position', [u1(1,i)-4 u1(2,i)-4 8 8], 'EdgeColor', 'r');
    text(u1(1,i)+4, u1(2,i)+4, sprintf('%d', i), 'Color', 'r');
end
subplot(1,2,2), imshow(I2, []), title('View 2');
for i=1:length(u2)
    rectangle('Position', [u2(1,i)-4 u2(2,i)-4 8 8], 'EdgeColor', 'g');
    text(u2(1,i)+4, u2(2,i)+4, sprintf('%d', i), 'Color', 'g');
end

pause
input('Hit enter to continue', 's');

p1 = inv(K) * u1; % get normalized image coordinates
p2 = inv(K) * u2; % get normalized image coordinates
% Draw epipolar lines on image 1
for i=1:length(p2)
    subplot(1,2,1), imshow(I1,
    % The product l=E*p2 is the equation of the epipolar line corresponding
    % to p2, in the first image. Here, l=(a,b,c), and the equation of the
    % line is ax + by + c = 0.
    l = E * p2(:,i);

    % Calculate residual error. The product p1'*E*p2 should = 0. The
    % difference is the residual.
    res = p1(:,i)' * E * p2(:,i);
    fprintf('Residual is %f to point %d\n', res, i);

    % Let's find two points on this line. First set x=-1 and solve
    % for y, then set x=1 and solve for y.
    pLine0 = [-1; (-l(3)-l(1)*(-1))/l(2); 1];
    pLine1 = [1; (-l(3)-l(1))/l(2); 1];

    % Convert from normalized to unnormalized coords
    pLine0 = K * pLine0;
    pLine1 = K * pLine1;

    line([pLine0(1) pLine1(1)], [pLine0(2) pLine1(2)], 'Color', 'r');

    subplot(1,2,2), imshow(I2,
    rectangle('Position', [u2(1,i)-4 u2(2,i)-4 8 8], 'EdgeColor', 'g';
    text(u2(1,i)+4, u2(2,i)+4, sprintf('%d', i), 'Color', 'g');

    %pause;
    input('Hit enter to continue', 's');
end
Visualization (continued)

• To go the other way, take the transpose of

\[ \mathbf{p}_0^T \mathbf{E} \mathbf{p}_1 = 0 \]

\[ (\mathbf{p}_0^T \mathbf{E} \mathbf{p}_1)^T = \mathbf{p}_1^T \mathbf{E}^T \mathbf{p}_0 = 0 \]

• So, given a point \( \mathbf{p}_0 \) in the first image, \( \mathbf{l} = \mathbf{E}^T \mathbf{p}_0 \) is the corresponding epipolar line in the second image.

% Draw epipolar lines on image 2
for i=1:length(p1)
    subplot(1,2,2), imshow(I2, []);
    % The product l=E'*p1 is the equation of the epipolar line corresponding
    % to p1, in the second image. Here, l=(a,b,c), and the equation of the
    % line is ax + by + c = 0.
    l = E' * p1(:,i);
    % Let's find two points on this line. First set x=-1 and solve
    % for y, then set x=1 and solve for y.
    pLine0 = [-1; (-l(3)-l(1)*(-1))/l(2); 1];
    pLine1 = [1; (-l(3)-l(1))/l(2); 1];
    % Convert from normalized to unnormalized coords
    pLine0 = K * pLine0;
    pLine1 = K * pLine1;
    line([pLine0(1) pLine1(1)], [pLine0(2) pLine1(2)], 'Color', 'r');
    subplot(1,2,1), imshow(I1, []);
    rectangle('Position', [u1(1,i)-4 u1(2,i)-4 8 8], 'EdgeColor', 'r');
    text(u1(1,i)+4, u1(2,i)+4, sprintf('%d', i), 'Color', 'r');
    %pause;
    input('Hit enter to continue', 's');
end
Results

Epipolar lines corresponding to points in the second image, projected onto the first image

Epipolar lines corresponding to points in the first image, projected onto the second image
Calculating the Essential Matrix

• We have
  \[ p_0^T E p_1 = 0 \]
• We have a set of known point correspondences \( p_0 \) and \( p_1 \)
• We have one equation for each point correspondence
• We can solve for the unknowns in \( E \)

• Note that \( E \) is a 3x3 matrix with 9 unknowns
  – It’s only known up to a scale factor
  – We really only have 8 unknowns
• We need at least 8 equations – we can compute \( E \) from eight or more point correspondences
Calculating the Essential Matrix

- We have
  \[ p_0^T E p_1 = 0 \]
  \[ (x_0 \ y_0 \ 1) \begin{pmatrix} E_{11} & E_{12} & E_{13} \\ E_{21} & E_{22} & E_{23} \\ E_{31} & E_{32} & E_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} = 0 \]

- Write out as equation
  \[ \begin{pmatrix} E_{11}x_1 + E_{12}y_1 + E_{13} \\ E_{21}x_1 + E_{22}y_1 + E_{13} \\ E_{31}x_1 + E_{32}y_1 + E_{33} \end{pmatrix} = 0 \]

- Write as \( A \cdot x = 0 \), where \( x = (E_{11}, E_{12}, E_{13}, \ldots, E_{33}) \)
  \[ \begin{pmatrix} x_0 \ x_0 \ y_0 \ x_0 \ y_0 \ x_0 \ y_1 \ y_0 \ y_0 \ y_0 \ x_1 \ y_1 \ 1 \end{pmatrix} \begin{pmatrix} E_{11} \\ E_{12} \\ E_{13} \\ \vdots \\ E_{33} \end{pmatrix} = 0 \]

Actually, \( A \) will have one row for each point correspondence \((x_0, y_0) - (x_1, y_1)\)
Solving for E

• We have $A \mathbf{x} = \mathbf{0}$
• This is a system of homogeneous equations
• Ignoring the trivial solution $\mathbf{x} = 0$, you can find a unique solution for $\mathbf{x}$ that gives the least squares solution for $\mathbf{x}$; i.e., the solution that minimizes $\sum (p_0^T E p_1)^2$

• It is proportional to the only zero eigenvalue of $A^T A$

• You can use Singular Value Decomposition to find it:
  $A = U D V^T$
• The solution $\mathbf{x}$ is the column of $V$ corresponding to the only null singular value of $A$
  -- This is the rightmost column of $V$
Finding E using 8-point linear algorithm

- Solve $A \mathbf{x} = 0$ using Singular Value Decomposition (SVD):
  
  $$A = U D V^T$$

- The solution $\mathbf{x}$ is the rightmost column of $V$

```matlab
% Compute essential matrix E from point correspondences.
% We know that p1' E p2 = 0, where p1,p2 are the normalized image coords.
% We write out the equations in the unknowns E(i,j)
% A x = 0
A = [p1s(1,:)' .*p2s(1,:)'; p1s(1,:)'.*p2s(2,:)'; p1s(1,:)' ...
    p1s(2,:)'.*p2s(1,:)'; p1s(2,:)'.*p2s(2,:)'; p1s(2,:)' ...
    p2s(1,:)'; p2s(2,:)'; ones(length(p1s),1)];

% The solution to Ax=0 is the singular vector of A corresponding to the
% smallest singular value; that is, the last column of V in A=UDV'
[U,D,V] = svd(A);
x = V(:,size(V,2));                  % get last column of V

% Put unknowns into a 3x3 matrix. Transpose because Matlab's "reshape"
% uses the order E11 E21 E31 E12 ...
Escale = reshape(x,3,3)';
```
Observations

• Results can be unstable, due to poor numerical conditioning

• We can improve results by:
  – Preconditioning: We will first translate and scale the data points so they are centered at the origin and the average distance to the origin is $\approx 1$
  – Postconditioning: The values of $E$ are not independent. There are only five independent parameters. $E$ must have rank=2 ... we will enforce this
Preconditioning

% Scale and translate image points so that the centroid of
% the points is at the origin, and the average distance of the points to the
% origin is equal to sqrt(2).

% xn = p1(1:2,:);  % xn is a 2xN matrix
N = size(xn,2);
t = (1/N) * sum(xn,2);  % this is the (x,y) centroid of the points
xnc = xn - t*ones(1,N);  % center the points; xnc is a 2xN matrix
dc = sqrt(sum(xnc.^2));  % dist of each new point to 0,0; dc is 1xN vector
davg = (1/N)*sum(dc);  % average distance to the origin
s = sqrt(2)/davg;  % the scale factor, so that avg dist is sqrt(2)
T1 = [s*eye(2), -s*t ; 0 0 1];
p1s = T1 * p1;

% xn = p2(1:2,:);  % xn is a 2xN matrix
N = size(xn,2);
t = (1/N) * sum(xn,2);  % this is the (x,y) centroid of the points
xnc = xn - t*ones(1,N);  % center the points; xnc is a 2xN matrix
dc = sqrt(sum(xnc.^2));  % dist of each new point to 0,0; dc is 1xN vector
davg = (1/N)*sum(dc);  % average distance to the origin
s = sqrt(2)/davg;  % the scale factor, so that avg dist is sqrt(2)
T2 = [s*eye(2), -s*t ; 0 0 1];
p2s = T2 * p2;
Postconditioning

• Enforce the property that the essential matrix has only two non-zero eigenvalues, and that they are equal
• You can force this by taking the SVD of $E$
  \[ E = U S V^T \]
• then reconstruct $E$ with only its first two eigenvalues
  \[ E \leftarrow U \text{diag}(1,1,0) V^T \]
• In Matlab

\[
[U,D,V] = \text{svd}(E_{scale});
E_{scale} = U*\text{diag}([1 1 0])*V';
\]
Undoing Preconditioning

- After computing the essential matrix \textbf{Escale}, you then have to adjust the result to undo the effect of point scaling. This can be done by \( E = T_1^T \text{Escale} \ T_2 \).

\[
E = T_1' \ast \text{Escale} \ast T_2; \quad \% \text{ Undo scaling}
\]
% Calculate the essential matrix.
clear all
close all
K = [ 300 0 150; 0 300 150; 0 0 1];
% These are the points in image 1
u1 = [ 61.4195 102.1798 150.0000 68.3768 106.2098 150.0000 74.3208 109.6134 176.0870 196.1538 174.0000 192.8571 172.2222 190.0000;
      1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 ];
% These are the corresponding points in image 2
u2 = [ 45.5272 63.4568 86.9447 61.6620 80.2653 104.1468 75.7981 94.7507 118.6606 135.5451 176.3357 147.5739 184.5258 157.8633 191.6139;
      126.5989 136.7293 150.0000 165.9997 180.2731 198.5963 200.5196 217.7991 239.5982 125.7710 105.4355 172.3407 150.0000 212.1766 188.5687;
      1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 ];
I1 = imread('I1.tif');
I2 = imread('I2.tif');
% Display points on the images for visualization
imshow(I1, []);
for i=1:length(u1)
    x = round(u1(1,i));     y = round(u1(2,i));
    rectangle('Position', [x-4 y-4 8 8], 'EdgeColor', 'r');
    text(x+4, y+4, sprintf('%d', i), 'Color', 'r');
end
figure, imshow(I2, []);
for i=1:length(u2)
    x = round(u2(1,i));     y = round(u2(2,i));
    rectangle('Position', [x-4 y-4 8 8], 'EdgeColor', 'r');
    text(x+4, y+4, sprintf('%d', i), 'Color', 'r');
end

% Get normalized image points
p1 = inv(K)*u1;
p2 = inv(K)*u2;
% Scale and translate image points so that the centroid of 
% the points is at the origin, and the average distance of the points to the 
% origin is equal to sqrt(2).
% % xn = p1(:,1:2); % xn is a 2xN matrix
N = size(xn,2);
t = (1/N) * sum(xn,2); % this is the (x,y) centroid of the points
xnc = xn - t*ones(1,N); % center the points; xnc is a 2xN matrix
dc = sqrt(sum(xnc.^2)); % dist of each new point to 0,0; dc is 1xN vector
davg = (1/N)*sum(dc); % average distance to the origin
s = sqrt(2)/davg; % the scale factor, so that avg dist is sqrt(2)
T1 = [s*eye(2), -s*t ; 0 0 1];
p1s = T1 * p1;
xn = p2(:,1:2); % xn is a 2xN matrix
N = size(xn,2);
t = (1/N) * sum(xn,2); % this is the (x,y) centroid of the points
xnc = xn - t*ones(1,N); % center the points; xnc is a 2xN matrix
dc = sqrt(sum(xnc.^2)); % dist of each new point to 0,0; dc is 1xN vector
davg = (1/N)*sum(dc); % average distance to the origin
s = sqrt(2)/davg; % the scale factor, so that avg dist is sqrt(2)
T2 = [s*eye(2), -s*t ; 0 0 1];
p2s = T2 * p2;

% Compute essential matrix E from point correspondences.
% We know that p1s' E p2s = 0, where p1s,p2s are the scaled image coords.
% We write out the equations in the unknowns E(i,j)
% A x = 0
A = [p1s(:,1)'.p2s(:,1)' p1s(:,2)'.p2s(:,2)';
     p1s(:,1)'.p2s(:,2)' p1s(:,2)'.p2s(:,1)';
     ones(length(p1s),1)';
     p2s(:,1)';
     p2s(:,2)';
     ones(length(p2s),1)];

% The solution to Ax=0 is the singular vector of A corresponding to the
% smallest singular value; that is, the last column of V in A=UDV'
[U,D,V] = svd(A);
x = V(:,size(V,2)); % get last column of V

% Put unknowns into a 3x3 matrix. Transpose because Matlab's "reshape"
% uses the order E11 E21 E31 E12 ...
% Escale = reshape(x,3,3)';

% Force rank=2 and equal eigenvalues
[U,D,V] = svd(Escale);
Escale = U'diag([1 1 0])'*V';

% Undo scaling
E = T1' * Escale * T2;
disp('Calculated essential matrix:');
disp(E);
save E
Results

• Run program “essential.m”
  – This inputs the corresponding points, and calculates the essential matrix

• Verify that calculated essential matrix equals the “true” essential matrix (to within a scale factor)

  True essential matrix:
  
  0  -1.0000  0
  -0.3615  0  -3.1415
  0  3.0000  0

  Calculated essential matrix:
  
  0.0001  2.4639  0.0010
  0.8929  0.0834  7.7585
  -0.0004  -7.3917   -0.0031

• Run program “drawepipolar.m” again