Colorado School of Mines

Computer Vision

Professor William Hoff
Dept of Electrical Engineering & Computer Science
http://inside.mines.edu/~whoff/
2D Image Transforms
Outline

• First, a review of least squares fitting
• Finding an image transform using least squares
• The transform caused by a rotating camera
• Applying a transform to an input image to get an output image
  – Example: generating an orthophoto
  – Example: stitching two images together to make a “panorama”
Least Squares Fitting to a Line

- We have some measurement data \((x_i, y_i)\)
- We want to fit the data to \(y = f(x) = mx + b\)
- We will find the parameters \((m, b)\) that minimize the objective function

\[
E = \sum_i |y_i - f(x_i)|^2
\]

Example:
- \((x_1, y_1) = (1, 1)\)
- \((x_2, y_2) = (2, 3)\)
- \((x_3, y_3) = (3, 4)\)
Linear Least Squares

• In general
  – The input data can be vectors
  – The function can be a linear combination of the input data

• We write $A \mathbf{x} = \mathbf{b}$
  – The parameters to be fit are in the vector $\mathbf{x}$
  – The input data is in $A, \mathbf{b}$

• Example of a line
  – Parameter vector
    $\mathbf{x} = \begin{pmatrix} m \\ b \end{pmatrix}$
  – Linear equations
    $\begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \end{pmatrix} \begin{pmatrix} m \\ b \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$

• So for a line
  $A = \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots \\ x_N & 1 \end{pmatrix}$, $\mathbf{b} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$
Solving Linear Least Squares

• Want to minimize
  \[ E = |Ax - b|^2 \]

• Expanding we get
  \[ E = x^T (A^T A)x - 2x^T (A^T b) + |b|^2 \]

• To find the minimum, take derivative wrt \( x \) and set to zero, getting
  \[ (A^T A)x = A^T b \]
  Called the “normal equations”

• To solve, can do
  \[ x = (A^T A)^{-1} A^T b \]
  “pseudo inverse”
  \[ A^+ = (A^T A)^{-1} A^T \]

• In Matlab can do
  – \( x = \text{pinv}(A)*b; \)
  – or \( x = A\backslash b; \)

• Note – it is preferable to solve the normal equations using Cholesky decomposition
Example

• The linear system for the line example earlier is $Ax = b$, where
  
  \[
  A = \begin{pmatrix}
  x_1 & 1 \\
  x_2 & 1 \\
  x_3 & 1
  \end{pmatrix} = \begin{pmatrix}
  1 & 1 \\
  2 & 1 \\
  3 & 1
  \end{pmatrix}, \quad
  b = \begin{pmatrix}
  y_1 \\
  y_2 \\
  y_3
  \end{pmatrix} = \begin{pmatrix}
  1 \\
  3 \\
  4
  \end{pmatrix}
  \]

• Normal equations \( (A^T A)x = A^T b \)

  \[
  A^T A = \begin{pmatrix}
  1 & 2 & 3 \\
  1 & 2 & 1 \\
  1 & 3 & 1
  \end{pmatrix} \begin{pmatrix}
  1 & 1 \\
  2 & 1 \\
  3 & 1
  \end{pmatrix} = \begin{pmatrix}
  14 & 6 \\
  6 & 3
  \end{pmatrix}, \quad
  x = (A^T A)^{-1} A^T b = \begin{pmatrix}
  1.5 \\
  -0.333
  \end{pmatrix}
  \]

• So the best fit line is \( y = 1.5x - 0.333 \)
Finding an image transform

- If you know a set of point correspondences, you can estimate the parameters of the transform.
- Example – find the rotation and translation of the book in the images below.

% Using imtool, we manually find % corresponding points (x;y), which are % the four corners of the book
pA = [221 413 416 228; 31 20 304 308];
pB = [214 404 352 169; 7 34 314 280];
Example (continued)

- A 2D rigid transform is

\[
\begin{bmatrix}
    x_B \\
y_B \\
1
\end{bmatrix} = \begin{bmatrix}
    c & -s & t_x \\
s & c & t_y \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
    x_A \\
y_A \\
1
\end{bmatrix}, \quad \text{where } c = \cos \theta, \ s = \sin \theta
\]

- Or

\[
x_B = cx_A - sy_A + t_x \\
y_B = sx_A + cy_A + t_y
\]

- We put into the form \( Ax = b \), where

\[
A = \begin{pmatrix}
    x_A^{(1)} & -y_A^{(1)} & 1 & 0 \\
y_A^{(1)} & x_A^{(1)} & 0 & 1 \\
\vdots & & & \\
y_A^{(N)} & x_A^{(N)} & 0 & 1
\end{pmatrix}, \quad x = \begin{pmatrix}
c \\
s \\
t_x \\
t_y
\end{pmatrix}, \quad b = \begin{pmatrix}
x_B^{(1)} \\
y_B^{(1)} \\
\vdots \\
y_B^{(N)}
\end{pmatrix}
\]

Note: \( c \) and \( s \) are not really independent variables; however we treat them as independent so that we get a system of linear equations.
% Here are corresponding points (x;y)
pA = [ 
    221 413 416 228; 
    31 20 304 308];
pB = [ 
    214 404 352 169; 
    7 34 314 280];
N = size(pA,2);

A = zeros(2*N,4);
for i=1:N
    A( 2*(i-1)+1, :) = [ pA(1,i) -pA(2,i) 1 0];
    A( 2*(i-1)+2, :) = [ pA(2,i)  pA(1,i) 0 1];
end
b = reshape(pB, [], 1);

x = A\b;

theta = acos(x(1));
rx = x(3);
ry = x(4);

Note: you might get slightly different values of theta, from c and s. You could average them to get a better estimate.
The case of a rotating camera

- Let point \( P \) be defined in camera #1’s coordinate system. Then
  \[
  \tilde{x}_1 = \begin{pmatrix}
  1 & 0 & 0 & 0
  \\
  0 & 1 & 0 & 0
  \\
  0 & 0 & 1 & 0
  \\
  0 & 0 & 0 & 1
  \end{pmatrix}
  \begin{pmatrix}
  c_1 X \\
  c_1 Y \\
  c_1 Z \\
  1
  \end{pmatrix}
  = \begin{pmatrix}
  c_1 X \\
  c_1 Y \\
  c_1 Z
  \end{pmatrix}
  \]

- Also
  \[
  \begin{pmatrix}
  c_1 X \\
  c_1 Y \\
  c_1 Z
  \end{pmatrix} = K^{-1} \tilde{x}_1
  \]

- If there is only rotation from camera #1 to camera #2, then
  \[
  \begin{pmatrix}
  c_2 X \\
  c_2 Y \\
  c_2 Z
  \end{pmatrix} = \begin{pmatrix}
  c_2 R & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1
  \end{pmatrix}
  \begin{pmatrix}
  c_1 X \\
  c_1 Y \\
  c_1 Z
  \end{pmatrix},
  \quad or
  \begin{pmatrix}
  c_2 X \\
  c_2 Y \\
  c_2 Z
  \end{pmatrix} = \begin{pmatrix}
  c_2 & 0 & 0 \\
  0 & c_1 & 0 \\
  0 & 0 & c_1
  \end{pmatrix}
  \begin{pmatrix}
  c_2 R & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1
  \end{pmatrix}
  \begin{pmatrix}
  c_1 X \\
  c_1 Y \\
  c_1 Z
  \end{pmatrix}
  \]

- and
  \[
  \tilde{x}_2 = \begin{pmatrix}
  c_2 X \\
  c_2 Y \\
  c_2 Z
  \end{pmatrix} = \begin{pmatrix}
  c_2 & 0 & 0 \\
  0 & c_1 & 0 \\
  0 & 0 & c_1
  \end{pmatrix}
  \begin{pmatrix}
  c_2 R & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1
  \end{pmatrix}
  \begin{pmatrix}
  c_1 X \\
  c_1 Y \\
  c_1 Z
  \end{pmatrix}
  \]

The 3x3 matrix \( KRK^{-1} \) is a projective transform (homography) from image 1 to image 2.
Example

- Given an image of a scene, create another image as if it were taken with the camera at the same position, but rotated 30 degrees

```matlab
clear all
close all

I1 = imread('cameraman.tif');
I2 = zeros(size(I1));

% Say we have a rotation about the camera's y axis
th = 30.0 * pi/180;
Ry = [ cos(th)  0    sin(th);
    0        1       0;
   -sin(th) 0    cos(th)];
R_1_2 = Ry;

K = [ 128   0       128;
    0     128     128;
    0     0       1];
Kinv = inv(K);

% This is the projective transform (homography) from image 1 to image 2
H = K*R_1_2*Kinv
```
Generating another image using a transform

- We know the transformation from input image to output image, \( p_2 = H p_1 \)

- But instead we use the inverse, \( p_1 = H^{-1} p_2 \)

- Then we scan through every point \( p_2 \) in the output image, and calculate the point \( p_1 \) in the input image where we should get the intensity value to use

  - This makes sure that we don’t miss assigning any pixels in the output image

  - If \( p_1 \) falls at a non-integer location, we just take the value at the nearest integer point (a better way to do it is to interpolate among the neighbors)
Hinv = inv(H);
for x2=1:size(I2,2)
    for y2=1:size(I2,1)
        p2 = [x2; y2; 1];
        p1 = Hinv * p2;
        p1 = p1/p1(3);

        % We'll just pick the nearest point to p1 (better way is to 
        % interpolate).
        x1 = round(p1(1));
        y1 = round(p1(2));

        if x1>0 && x1<=size(I1,2) && y1>0 && y1<=size(I1,1)
            I2(y2,x2) = I1(y1,x1);
        end
    end
end
Example Application - Building Mosaics

• Assume we have two images of the same scene from the same position but different camera angles

• The mapping between the two image planes is a homography

• We find a set of corresponding points between the left and the right image
  – Since the homography matrix has 8 degrees of freedom, we need at least 4 corresponding point pairs
  – We solve for the homography matrix using least squares fitting

• We then apply the homography transform to one image, to map it into the plane of the other image
Examples

• from final project by CSM student Ryan Crawford (2012)
Matching points

Registered images

Note – blending should be done so that “seams” are not visible where the images are joined.
Example Application: Generating an Orthophoto

- An “orthophoto” is an aerial photograph geometrically corrected such that the scale is uniform
  - Like a map, an orthophotograph can be used to measure true distances
- Essentially, we want to take the image taken by a camera at some off-axis angle, and transform it as if it were taken looking straight down

One way to calculate the transform is to find some known “control points” in the input image, and specify where those points should appear in the output image

Then calculate the transform using least squares fitting
Example

• Transform the image as if it were taken from a camera perpendicular to the wall

  \textit{Input image}

  \textit{Output image (orthophoto)}

Pixel location (x,y):
389, 127;
1964, 347;
419, 1674;
1983, 1325;

Pixel location (x,y):
0 0;
299 0;
0 184;
299 184;

• For control points, we use four brick corners that define a rectangle of known size
  – The rectangle is 8 bricks high and 13 bricks wide
  – Each brick is about 23 cm, so rectangle is 8\times23=184 \text{ cm} \text{ high and } 13\times23=299 \text{ cm} \text{ wide}

• We’ll specify the corresponding rectangle in the output image
  – Use scale of 1 cm = 1 pixel
  – Put upper left corner at 0,0
Useful Matlab functions

• **fitgeotrans**
  – Given two sets of control points, estimate the image transform using least squares fitting
  – Example:
    • \(T\text{form1} = \text{fitgeotrans}(\text{Pimg1}, \text{Pworld1}, '\text{projective}');\)

• **imwarp**
  – Transform an image using a transform created by “fitgeotrans”
  – Example:
    • \(I\text{out1} = \text{imwarp}(I\text{in1}, T\text{form1});\)

**Note:** prior to Matlab release R2013, these functions were `cp2tform` and `imtransform`
clear all
close all

Iin1 = imread('wall1.jpg');
imshow(Iin1,[]), impixelinfo;

% Location of control points in (x,y) input image coords (pixels)
% These are the corners of a rectangle that is 8 bricks high by 13 bricks
% wide. Each brick is about 23 cm.
Pimg1 = [
    389, 127;
    1964, 347;
    419, 1674;
    1983, 1325;
];

% Mark control points on input image, to verify that we have correct locations.
for i=1:size(Pimg1, 1)
    rectangle('Position', [Pimg1(i,1)-10 Pimg1(i,2)-10 20 20], 'FaceColor', 'r');
end

% Define location of control points in the world. The control points are
% the corners of a rectangle that is 8 bricks high by 13 bricks wide. Each
% brick is about 23 cm. We'll define the upper left control point to be at
% (X,Y)=(0,0), with the +X axis to the right and the +Y axis down.
Pworld1 = [
    0, 0;       % Units in cm
    299, 0;
    0, 184;
    299, 184;
];

% Compute transform, from corresponding control points
Tform1 = fitgeotrans(Pimg1,Pworld1,'projective');

% Transform input image to output image
Iout1 = imwarp(Iin1,Tform1);
figure, imshow(Iout1,[]);
Results

• Note – the upper left corner is not at (0,0) in the output image
• \texttt{imwarp} automatically enlarges the output image so that it contains the entire transformed image (you can override this)
• To see the location of the output image in the output XY space, use
  \[
  \texttt{[Iout1, ref2Dout] = imwarp(Iin1,Tform1);} \\
  \texttt{disp(ref2Dout)}
  \]
• Fields of the “\texttt{imref2D}” structure:
  – \texttt{XWorldLimits, YWorldLimits}: x,y borders of output image
  – \texttt{PixelExtentInWorldX, PixelExtentInWorldY}: scale of output image
• We specify the control points using the same world coordinate system
% Second image
Iin2 = imread('wall2.jpg');
figure, imshow(Iin2,[]), impixelinfo;

% Location of control points in (x,y) input image
Pimg2 = [       
    649, 340;  
    2078, 41;  
    667, 1573;  
    2125, 1628;
];

% Mark control points on input image, just to verify that we have the % correct locations.
for i=1:size(Pimg2, 1)
    rectangle('Position', [Pimg2(i,1)-10 Pimg2(i,2)-10 20 20], 'FaceColor', 'r');
end

% Define location of control points in the world (in cm).
Pworld2 = [
    299, 0;  % X,Y units in cm
    506, 0;
    299, 184;
    506, 161;
];

% Compute transform, from corresponding control points
Tform2 = fitgeotrans(Pimg2,Pworld2,'projective');

% Transform input image to output image
Iout2 = imwarp(Iin2,Tform2);
figure, imshow(Iout2,[]);
Stitching images together

- We want to stitch the two images together to make a panorama
- We can merge them once they are registered to the same coordinate system (i.e., they are both orthophotos)
- But to get them to merge properly, we need to explicitly set output coordinates for both images
Specifying output coordinates

- When calling `imwarp`, we pass in an “imref2d” structure. This specifies the location and size of the output images.
  - Both output images are mapped to the same coordinate system
  - We make the output size big enough to hold both of input images

% Use the same ref2D structure to map both of them.
ref2Doutput = imref2d( ... 
    [185, 506], ... % Size of output image (rows, cols)
    [0, 506], ... % xWorldLimits
    [0, 185]); % yWorldLimits

Iout1 = imwarp(Iin1, Tform1, 'OutputView', ref2Doutput );
Iout2 = imwarp(Iin2, Tform2, 'OutputView', ref2Doutput );

- Finally, combine the images

  Imerge = [Iout1(:,1:300) Iout2(:,301:end)];
  figure, imshow(Imerge, []);
Final Result

Imerge