2D Image Transforms

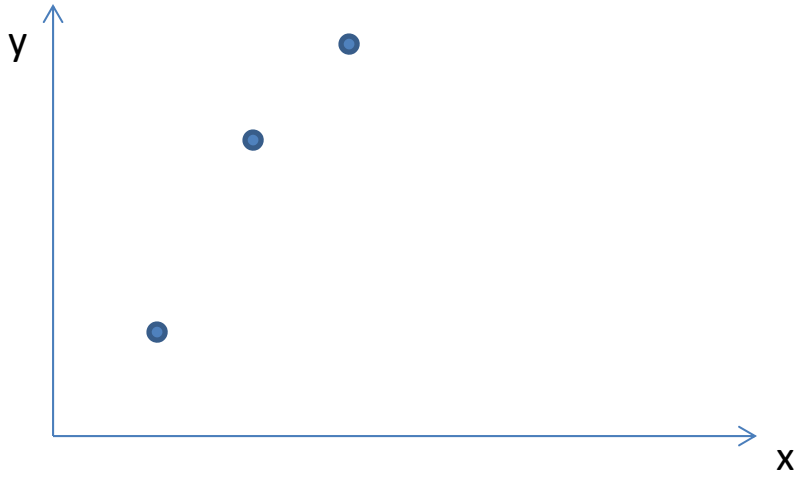
Outline

- 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

Least Squares Fitting to a Line

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

$$E = \sum_i \left| y_i - f\left(x_i\right) \right|^2$$



Example
(x1,y1) = (1,1)
(x2,y2) = (2,3)
(x3,y3) = (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 x** = **b**
  - The parameters to be fit are in the vector **x**
  - The input data is in **A,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

$$\mathbf{A} = \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \\ x_N & 1 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

Solving Linear Least Squares

- Want to minimize

$$E = \left|\mathbf{A}\mathbf{x} - \mathbf{b}\right|^2$$

- Expanding we get

$$E = \mathbf{x}^T\left(\mathbf{A}^T\mathbf{A}\right)\mathbf{x} - 2\mathbf{x}^T\left(\mathbf{A}^T\mathbf{b}\right) + \left|\mathbf{b}\right|^2$$

- To find the minimum, take derivative wrt x and set to zero, getting

$$\left(\mathbf{A}^T\mathbf{A}\right)\mathbf{x} = \mathbf{A}^T\mathbf{b}$$

Called the "normal equations"

- To solve, can do

$$\mathbf{x} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{b}$$

"pseudo inverse"

$$\mathbf{A}^+ = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T$$

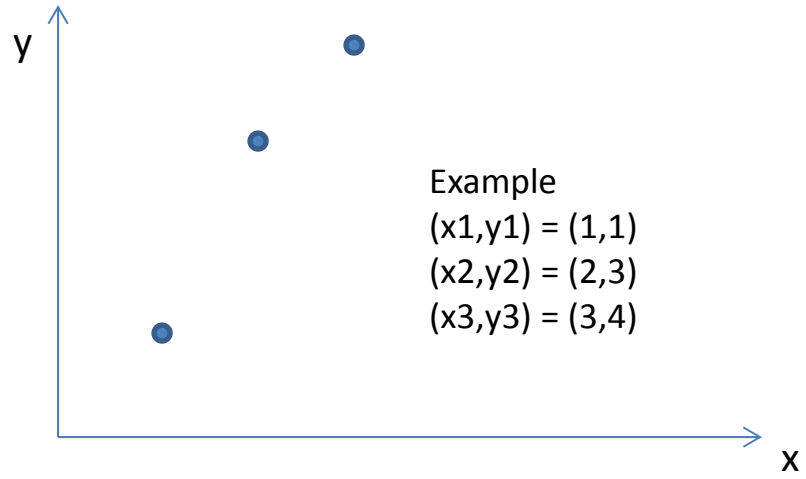
- In Matlab can do
  - x = pinv(A)*b;
  - or x = A\b;

- 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

$$\mathbf{A} = \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ 4 \end{pmatrix}$$



- Normal equations $\left(\mathbf{A}^T\mathbf{A}\right)\mathbf{x} = \mathbf{A}^T\mathbf{b}$

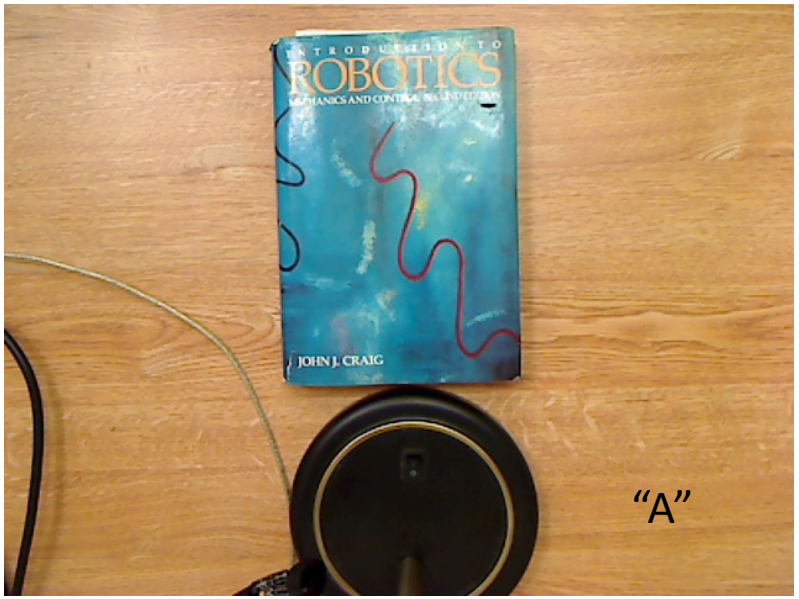
$$\mathbf{A}^T\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{pmatrix} = \begin{pmatrix} 14 & 6 \\ 6 & 3 \end{pmatrix}, \quad \mathbf{x} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{b} = \begin{pmatrix} 1.5 \\ -0.333 \end{pmatrix}$$

- So the best fit line is $y = 1.5x - 0.333$

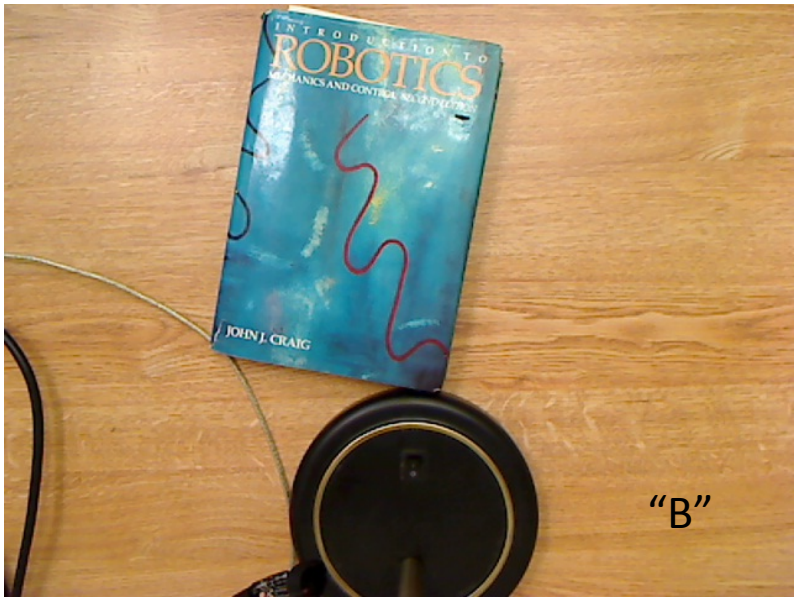
Finding an image transform

- 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 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];
```



"A"



"B"

Example (continued)

- A 2D rigid transform is

$$\begin{pmatrix} x_B \\ y_B \\ 1 \end{pmatrix} = \begin{pmatrix} c & -s & t_x \\ s & c & t_t \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_A \\ y_A \\ 1 \end{pmatrix}, \quad \text{where } c = \cos\theta,\ s = \sin\theta$$

- 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

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

Note: c and s are not really independent variables; however we treat them as independent so that we get a system of linear equations

Matlab

```
% 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));
tx = x(3);
ty = x(4);
```

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

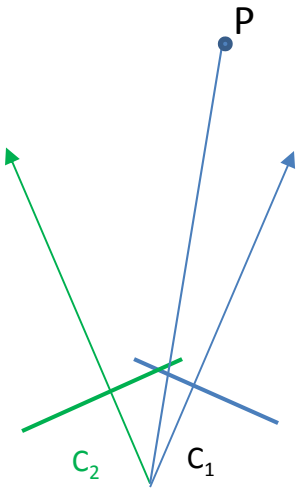
The case of a rotating camera

- Let point P be defined in camera #1's coordinate system. Then

$$\tilde{\mathbf{x}}_1 = \mathbf{K}\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\end{pmatrix}\begin{pmatrix}{}^{C_1}X\\ {}^{C_1}Y\\ {}^{C_1}Z\\ 1\end{pmatrix} = \mathbf{K}\begin{pmatrix}{}^{C_1}X\\ {}^{C_1}Y\\ {}^{C_1}Z\end{pmatrix}$$

- Also

$$\begin{pmatrix}{}^{C_1}X\\ {}^{C_1}Y\\ {}^{C_1}Z\end{pmatrix} = \mathbf{K}^{-1}\tilde{\mathbf{x}}_1$$



- If there is only rotation from camera #1 to camera #2, then

$$\begin{pmatrix}{}^{C_2}X\\ {}^{C_2}Y\\ {}^{C_2}Z\\ 1\end{pmatrix} = \begin{pmatrix}{}^{C_2}_{C_1}\mathbf{R} & \mathbf{0}\\ \mathbf{0} & 1\end{pmatrix}\begin{pmatrix}{}^{C_1}X\\ {}^{C_1}Y\\ {}^{C_1}Z\\ 1\end{pmatrix}, \quad \text{or} \quad \begin{pmatrix}{}^{C_2}X\\ {}^{C_2}Y\\ {}^{C_2}Z\end{pmatrix} = {}^{C_2}_{C_1}\mathbf{R}\begin{pmatrix}{}^{C_1}X\\ {}^{C_1}Y\\ {}^{C_1}Z\end{pmatrix}$$

- and

$$\tilde{\mathbf{x}}_2 = \mathbf{K}\begin{pmatrix}{}^{C_2}X\\ {}^{C_2}Y\\ {}^{C_2}Z\end{pmatrix} = \mathbf{K}\ {}^{C_2}_{C_1}\mathbf{R}\ \mathbf{K}^{-1}\ \tilde{\mathbf{x}}_1$$

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

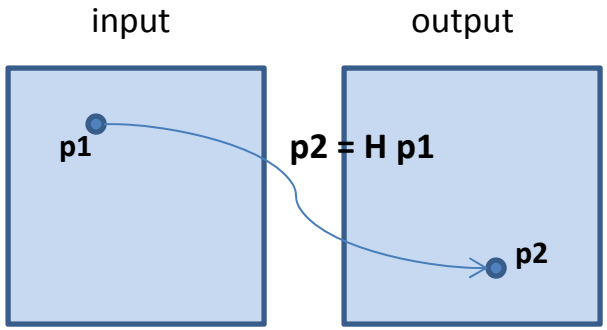
Example

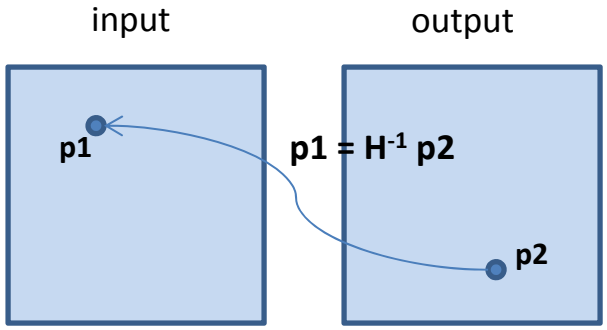
- Given an image of a scene, create another image as if it were taken with the camera at the same position, but rotated 30 degrees

```
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, **p2 = H p1**
- But instead we use the inverse, **p1 = H^{-1} p2**
- Then we scan through every point **p2** in the output image, and calculate the point **p1** 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 p1 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)





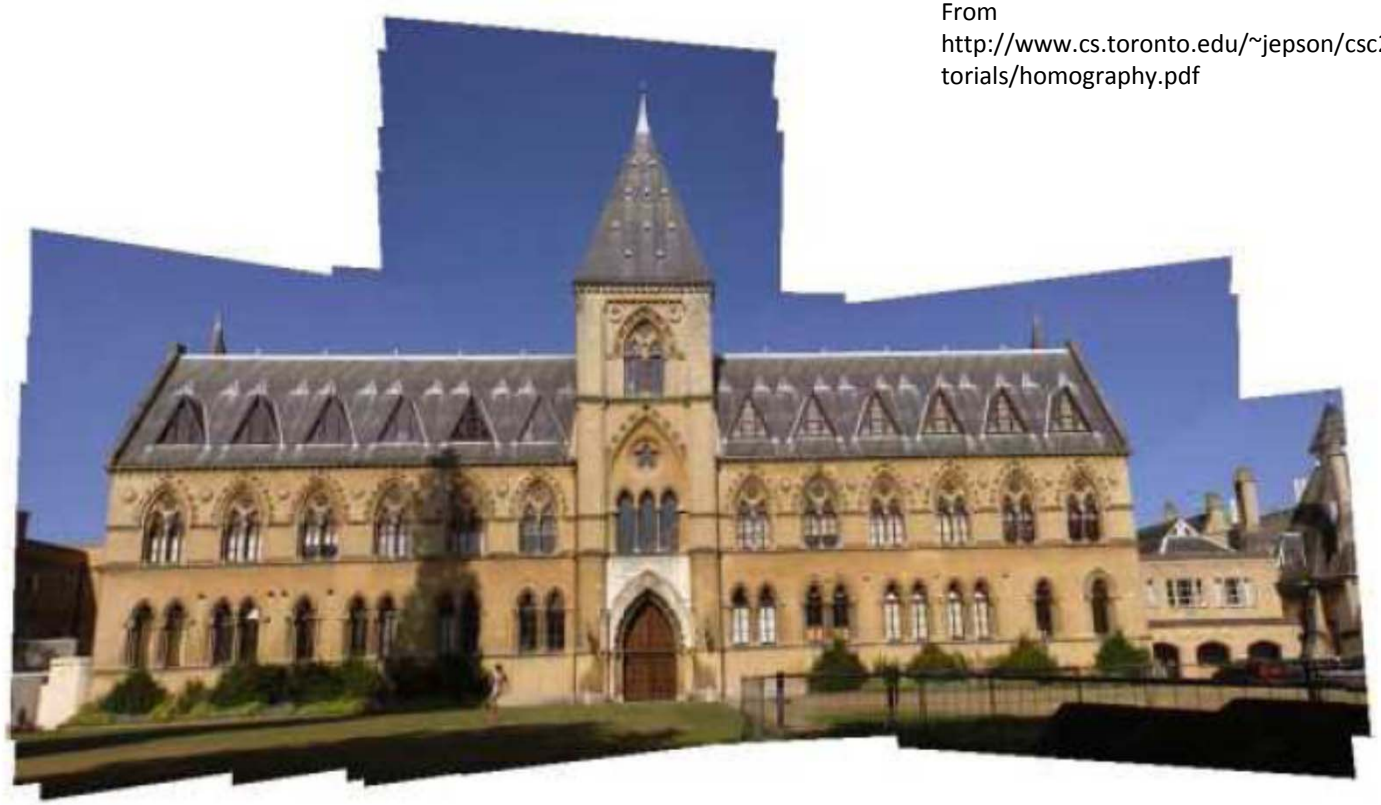
Matlab

```
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

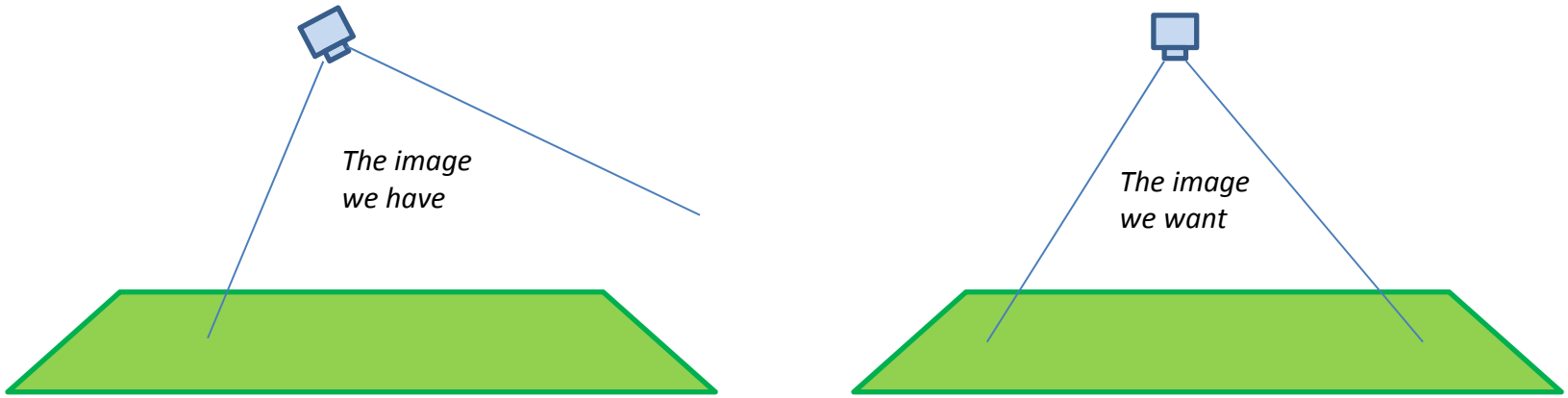
- Assume we have two images of the same scene from the same position but different camera angles
- The mapping between the two image planes is a homography
- We find a set of corresponding points between the left and the right image
  - Since the homography matrix has 8 degrees of freedom, we need at 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

From
http://www.cs.toronto.edu/~jepson/csc2503/tutorials/homography.pdf



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

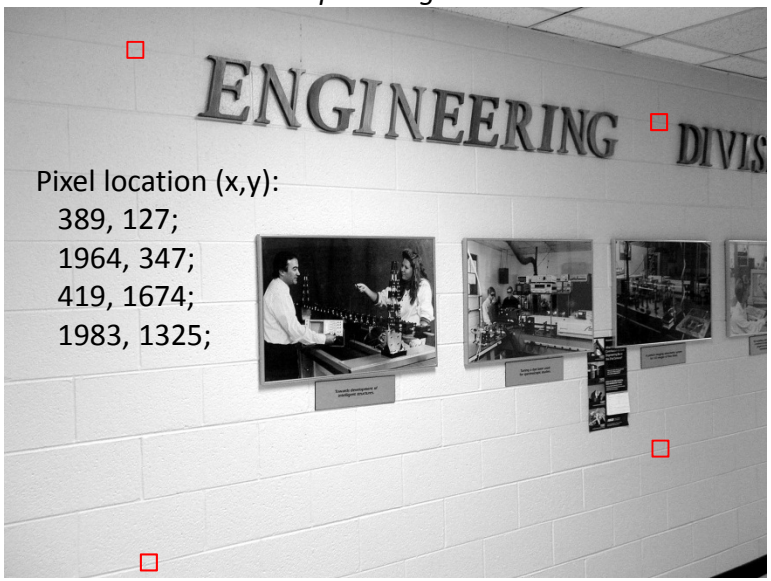


- One way to calculate the transform is to find some known “control points” in the input image, and specify where those points should appear in the output image
- Then calculate the transform using least squares fitting

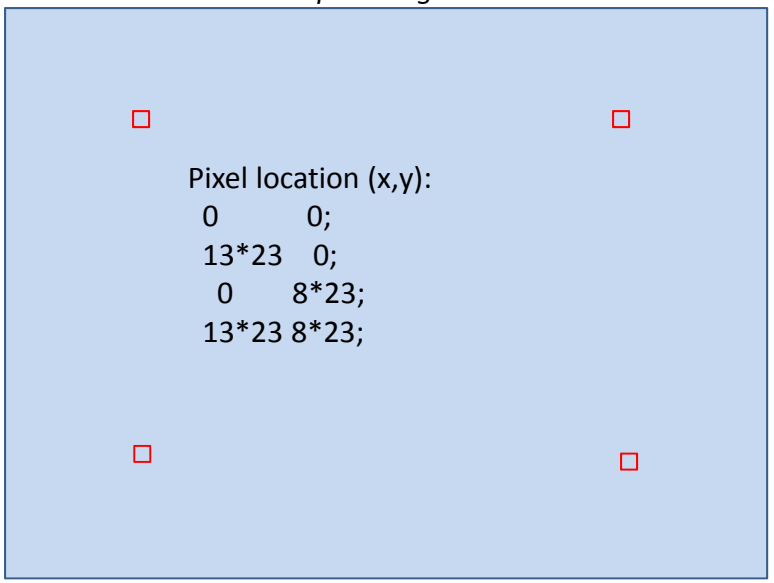
Example

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

Input image



Output image



- 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*23=184 cm high and 13*23=299 cm 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

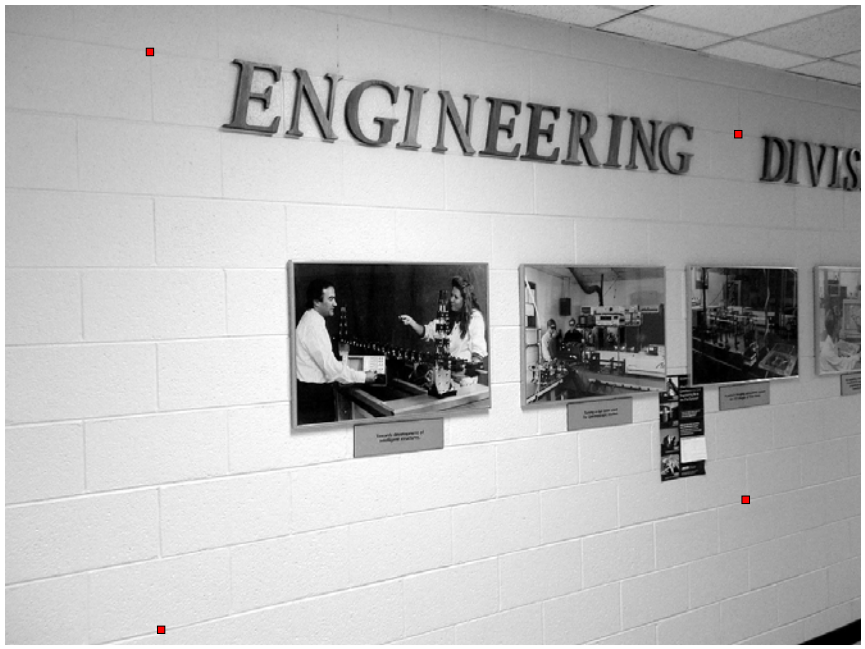
- `cp2tform`
  - Given two sets of control points, estimate the image transform using least squares fitting
  - Example:
    - `Tform = cp2tform(Pts1,Pts2,'projective');`

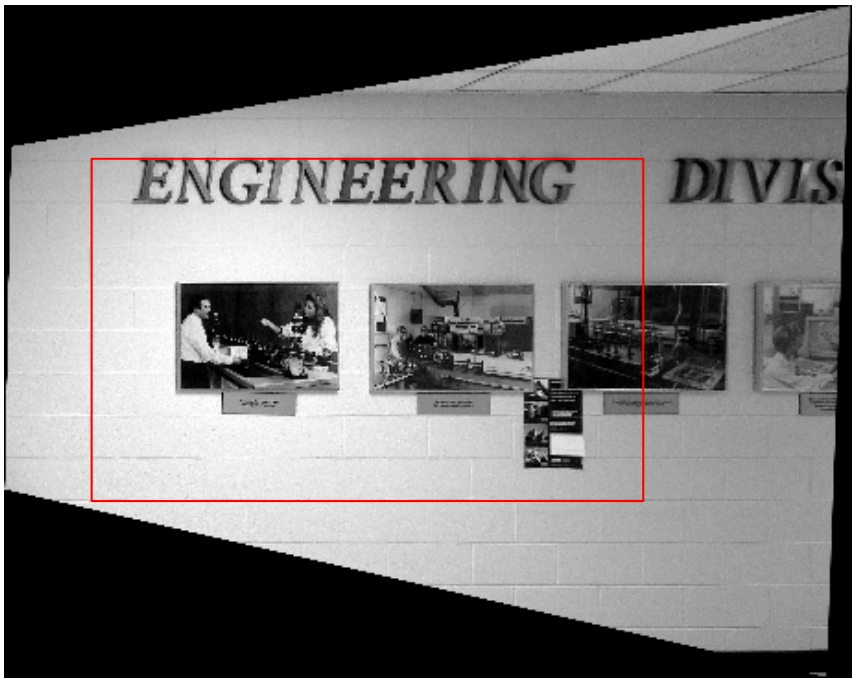
- `imtransform`
  - Transform an image using a pre-defined transform
  - Example:
    - `ITrans = imtransform(I1,Tform);`

Matlab

```
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.
Pin1 = [
    389, 127;
    1964, 347;
    419, 1674;
    1983, 1325;
];

% Mark control points on input image
for i=1:size(Pin1, 1)
    rectangle('Position', [Pin1(i,1)-10 Pin1(i,2)-10 20 20], 'FaceColor', 'r');
end

% Define location of control points in output image.  We'll make 1 pixel
% equal to 1 cm.  Also put the upper left control point at 0,0 pixels.
Pout1 = [
    0, 0;
    13*23, 0;
    0, 8*23;
    13*23, 8*23;
];

% Compute transform, from corresponding control points
Tform1 = cp2tform(Pin1,Pout1,'projective');

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

Results

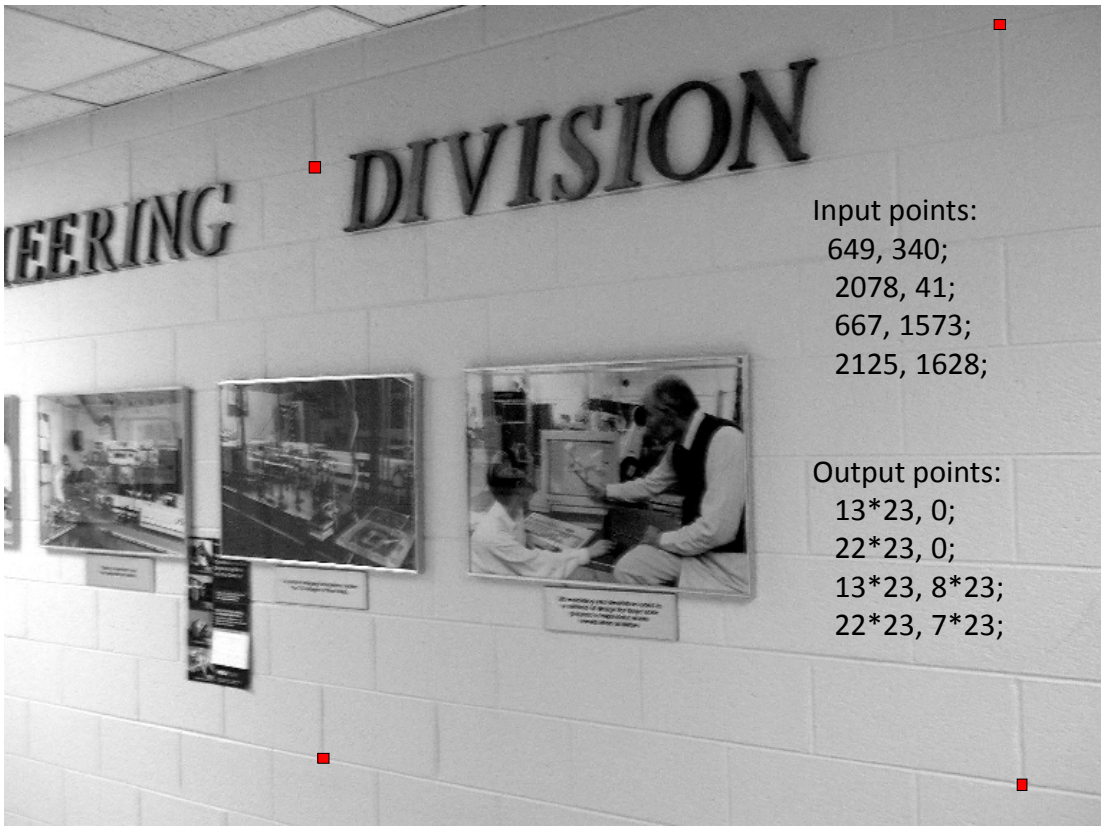




- Note – the upper left corner is not at (0,0) in the output image
- `imtransform` automatically enlarges the output image so that it contains the entire transformed image (you can override this)
- To see the location of the output image in the output XY space, use
  - `[ITrans, xdata, ydata] = imtransform(Iin1,Tform1);`

Mapping two images of the same scene

- We have a second image of the wall



Repeat process to transform image to an orthophoto

Then merge the two images

But to get them to merge properly, explicitly set output coordinates for both images

Specifying output coordinates

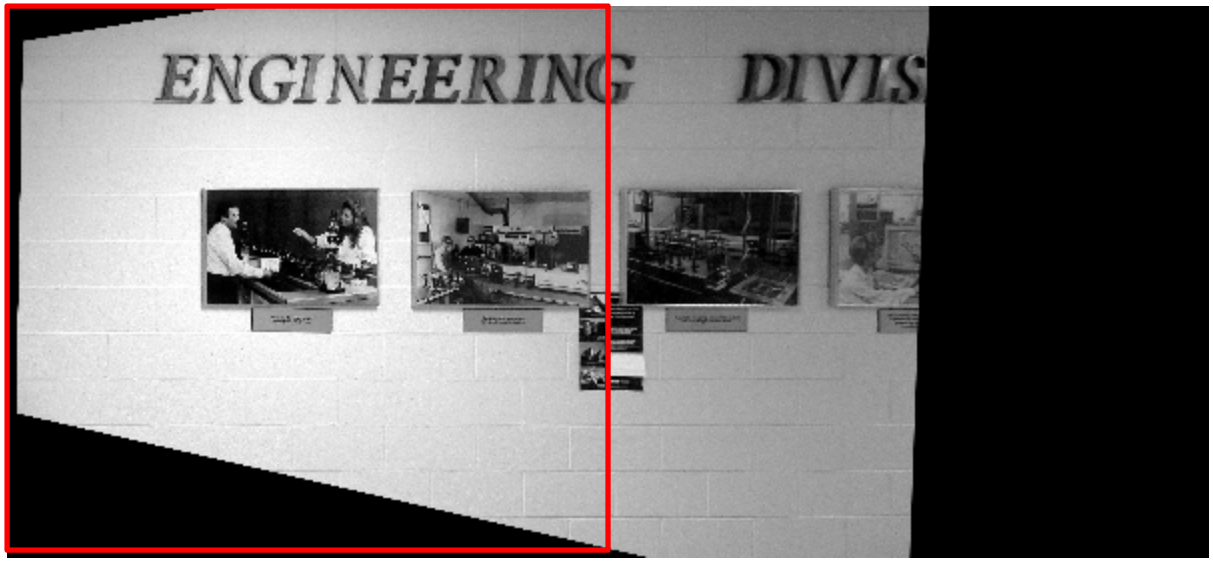
- In `imtransform`, we can specify output coordinates
  - `XData` specifies the x-coordinates of the first and last columns of the output image
  - `YData` specifies the y-coordinates of the first and last rows of the output image

```
Iout1 = imtransform(Iin1, Tform1, ...
     'XData',[-50 550], ...
     'YData',[-25 250]);

Iout2 = imtransform(Iin2, Tform2, ...
     'XData',[-50 550], ...
     'YData',[-25 250]);
```

Use the same output coordinates for both output images

- Finally, combine the images

```
[H,W] = size(Iout2);
Icombined = [Iout1(:,1:floor(W/2)) Iout2(:,floor(W/2)+1:end) ];
```



Iout1

Iout2

Final Result

Icombined