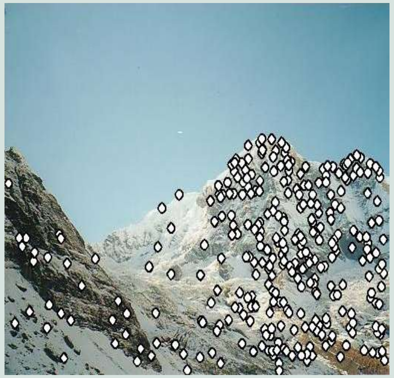
BMB5113
COMPUTER VISION

TRANSFORMATIONS & ALIGNMENT

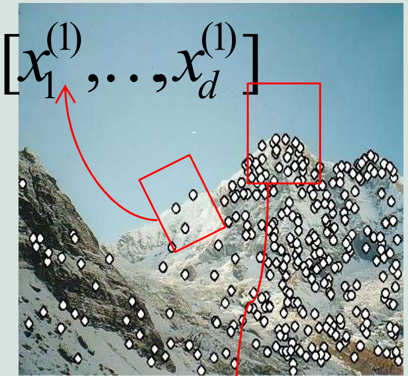
Stitching Problem

Matching Image Features

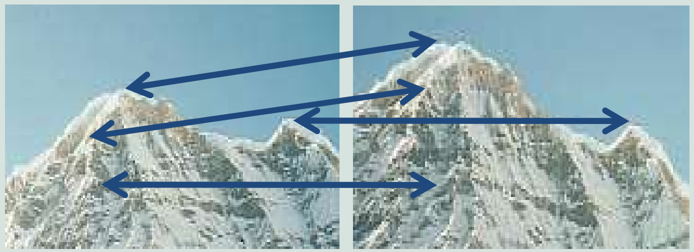
1) **Feature Detection:**
   Identify image features
2) **Feature Description:**
   Extract feature descriptor for each feature
3) **Feature Matching:**
   Find candidate matches between features
4) **Feature Correspondence:**
   Find consistent set of (inlier) correspondences between features



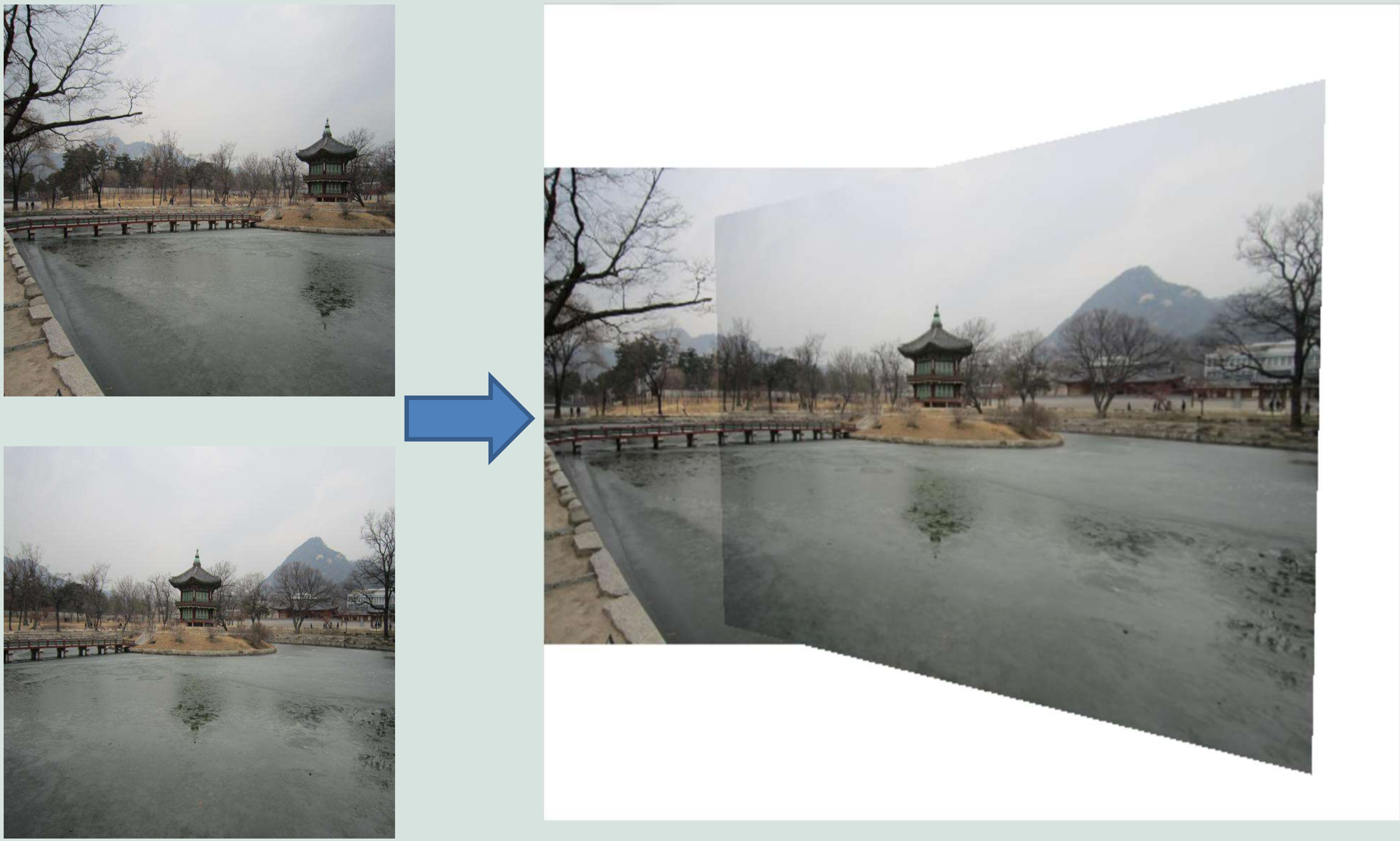
$$\mathbf{x}_1 = [x_1^{(1)}, \ldots, x_d^{(1)}]$$



$$\mathbf{x}_2 = [x_1^{(2)}, \ldots, x_d^{(2)}]$$



An Example



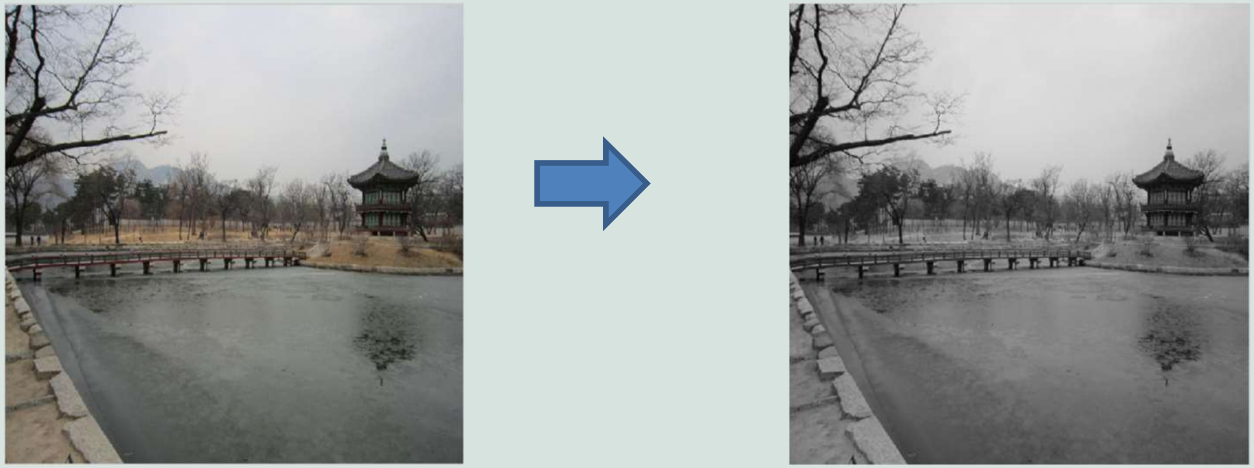
Problem 1: Preprocessing

- Most feature descriptors only work with grayscale images

  RGB image: 584 x 778 x 3

  Grayscale image: 584 x 778

- Convert color images to grayscale



Problem 2: Detecting Keypoints

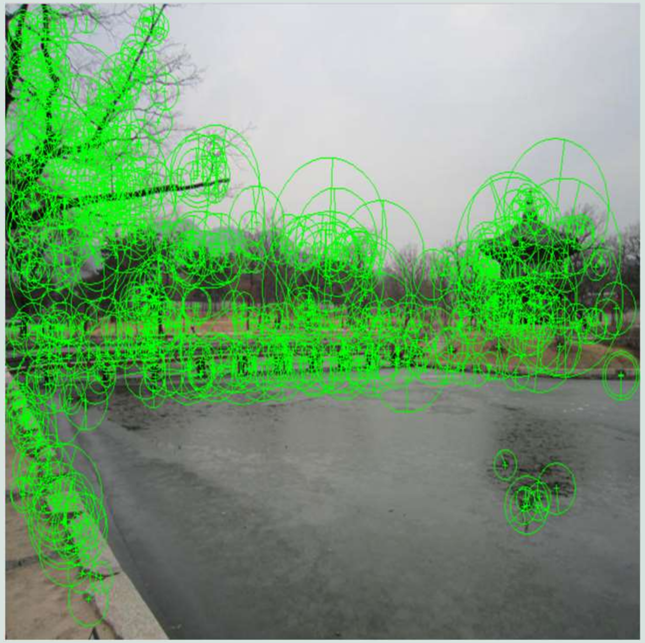
- Identify features in an image

  Detector: Hessian, Harris, FAST, etc…

Independently detect keypoints in both images

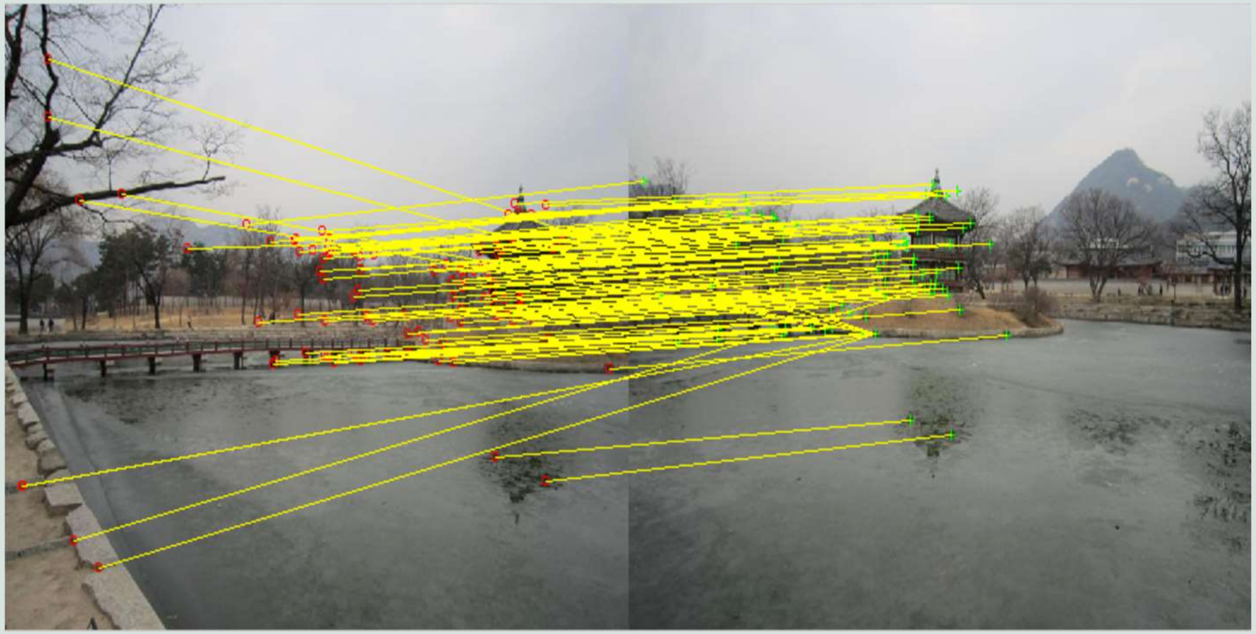
Problem 3: Extracting Descriptors

- Extract feature descriptors for each keypoint
  - Derived from pixels surrounding an interesting point.
  - HoG, LBP, Haar-wavelets,etc…
  - Remember SIFT, SURF, ORB



Problem 4: Matching Features

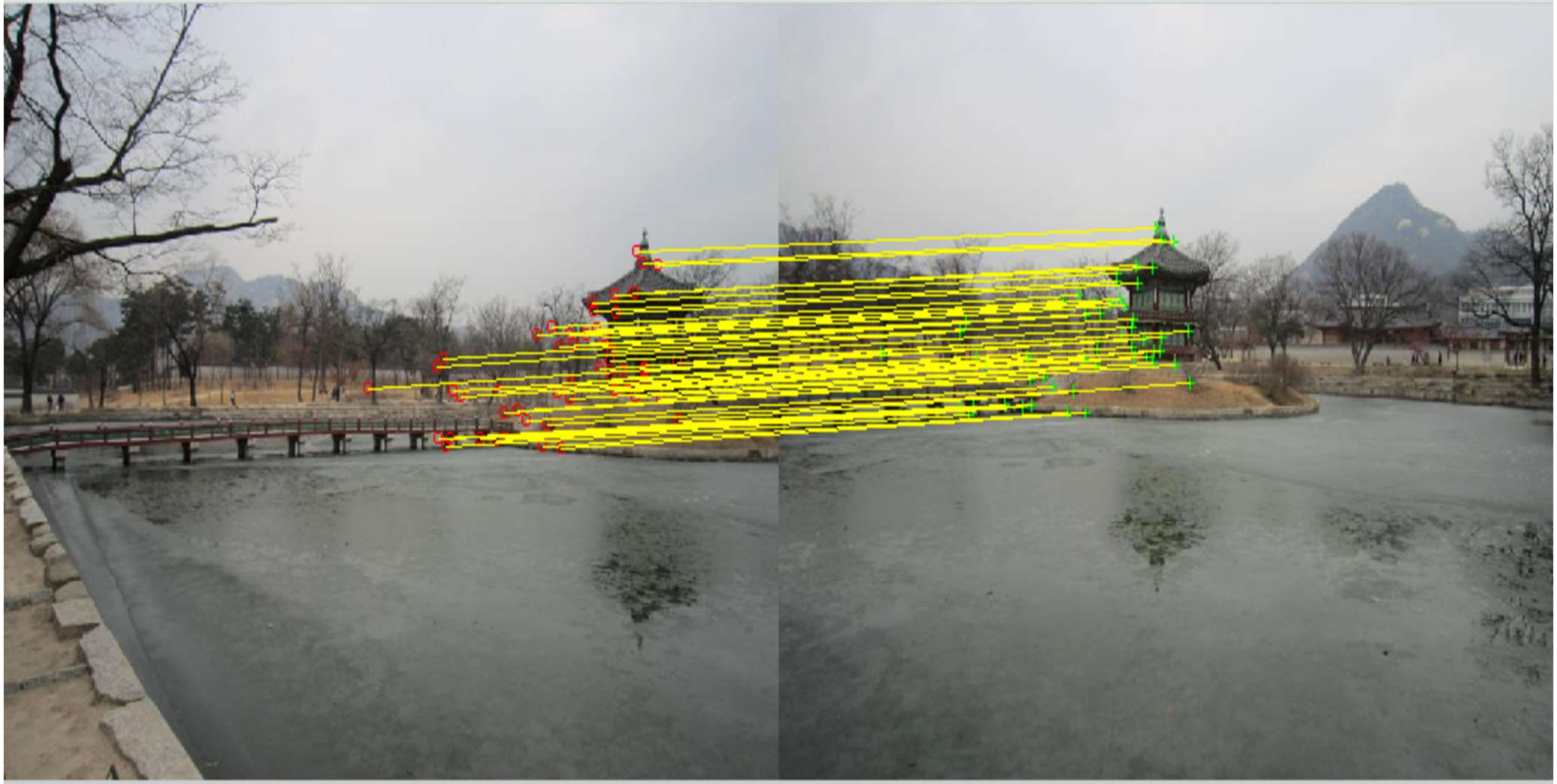
- Find pairs of features in the two images that match
  - Possible strategies: Match Threshold, Nearest Neighbor Symmetric, and Nearest Neighbor Ratio
- Find pairs of indices in feature list of image 1 and feature list of image 2 that match



Problem 5: RANSAC to Estimate Homography

- Exclude outlier matches
- Compute a homography to map one image plane to the other
  - Solution is RANSAC (Random Sample Consensus)
- RANSAC
  - Finds inlier points and compute a transformation from image 2 to image 1
  - Returns the transformation from inlier_points2 to inlier_points1
  - Transform_type?
    - Similarity, Affine, Projective

Problem 5: RANSAC to Estimate Homography



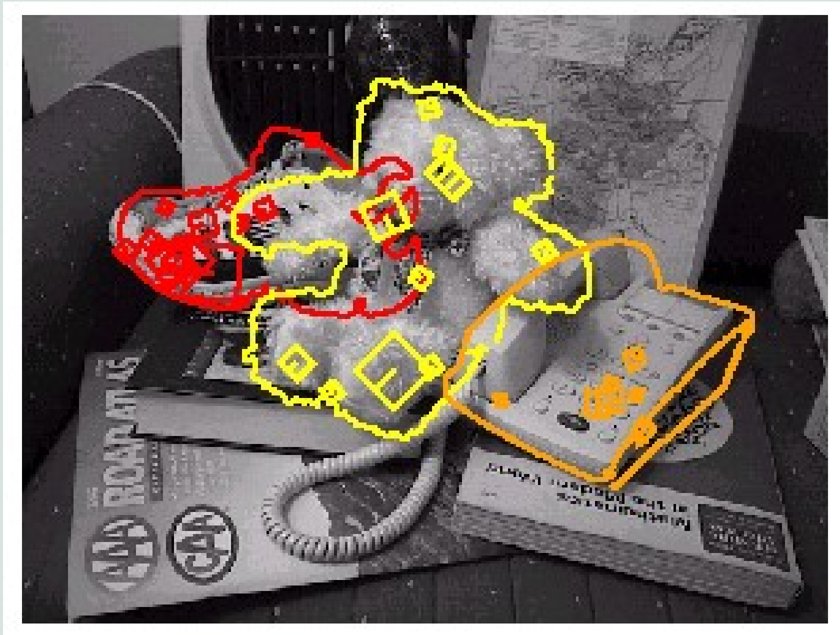
Problem 6: Stitching Panorama

- Warp the two images to make a final panoramic image

Alignment

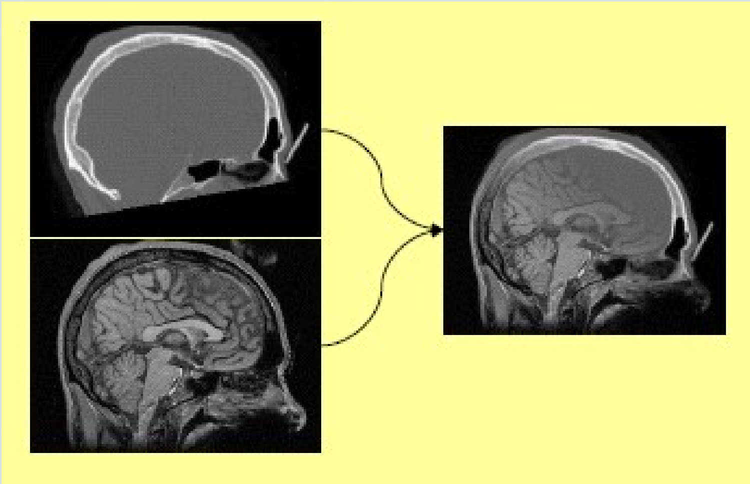
- Homographies
- Rotational Panoramas
- RANSAC
- Global alignment
- Warping
- Blending

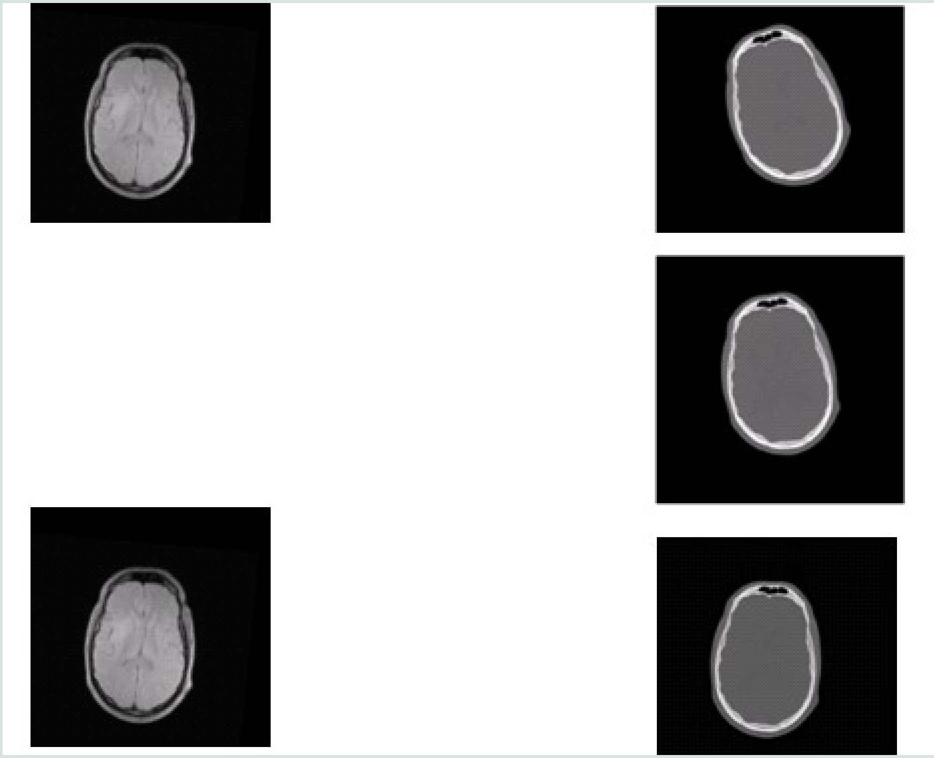
Motivation: Recognition



Figures from David Lowe

Motivation: Medical Image Registration



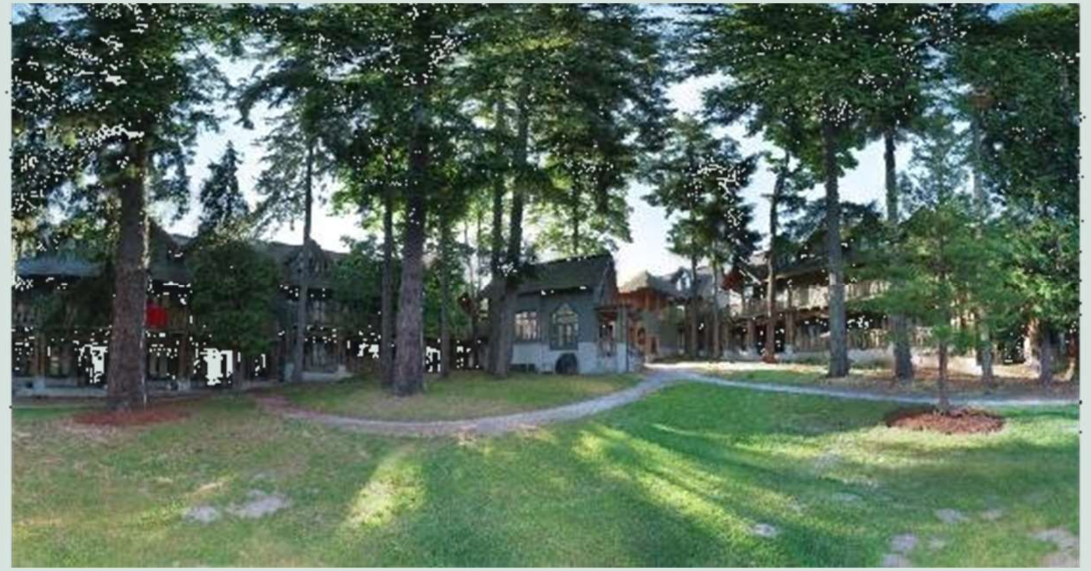


Motivation: Mosaics

- Getting the whole picture
  - Consumer camera: 50˚ x 35˚

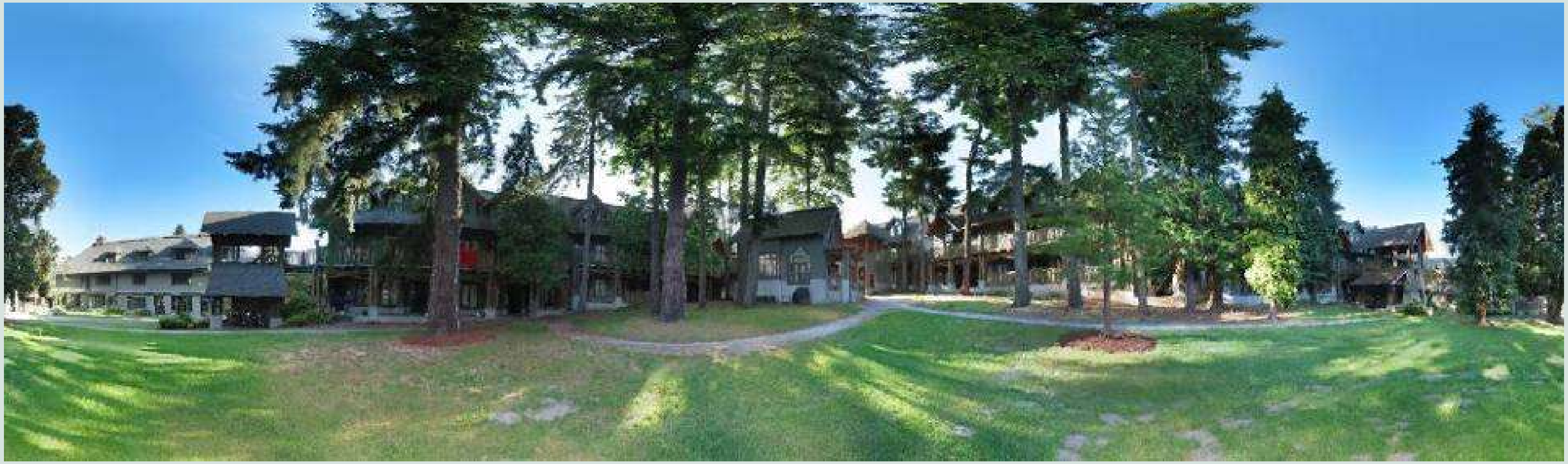
Motivation: Mosaics

- Getting the whole picture
  - Consumer camera: 50° x 35°
  - Human Vision: 176° x 135°



Motivation: Mosaics

- Getting the whole picture
  - Consumer camera: 50˚ x 35˚
  - Human Vision: 176˚ x 135˚



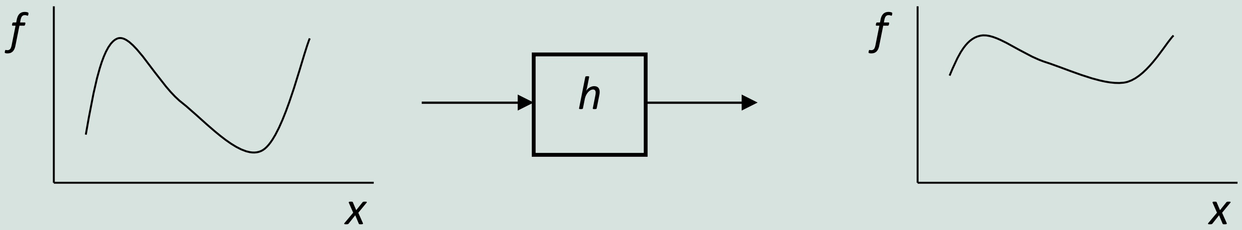
- Panoramic Mosaic = up to 360 ˚ x 180°

Motion models

- What happens when we take two images with a camera and try to align them?
- translation?
- rotation?
- scale?
- affine?
- perspective?

Image Warping

- image filtering: change *range* of image
    - $g(x) = h(f(x))$



- image warping: change *domain* of image
    - $g(x) = f(h(x))$

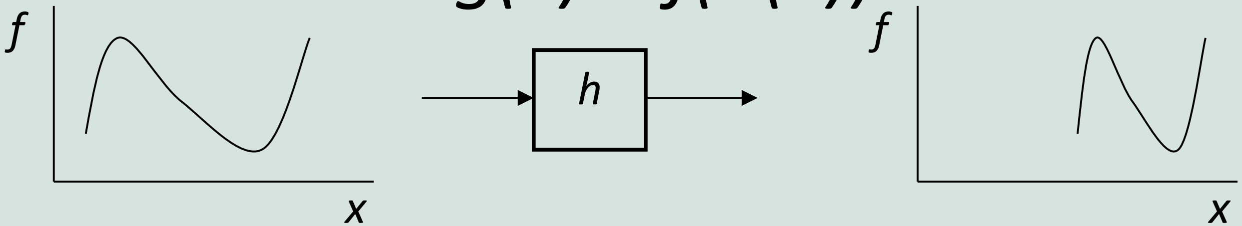
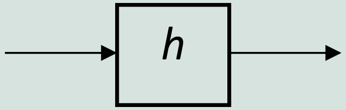


Image Warping

- image filtering: change *range* of image
    - *g(x) = h(f(x))*

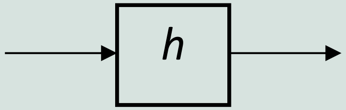
f



g

- image warping: change *domain* of image
    - *g(x) = f(h(x))*

f



g

Parametric (Global) Warping

- Examples of parametric warps:

translation

rotation

aspect

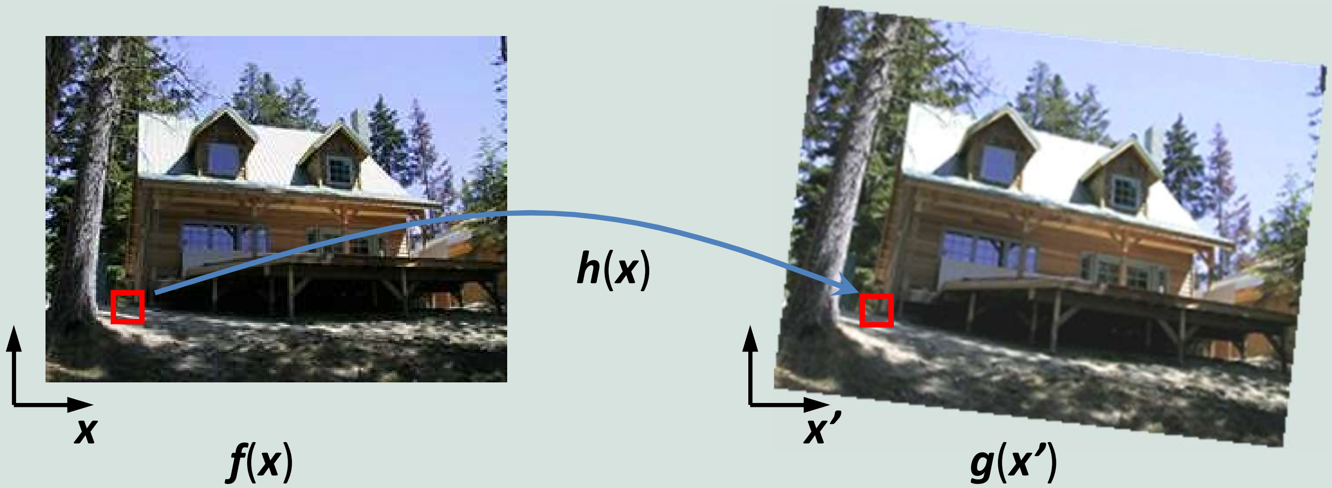
affine

perspective

cylindrical

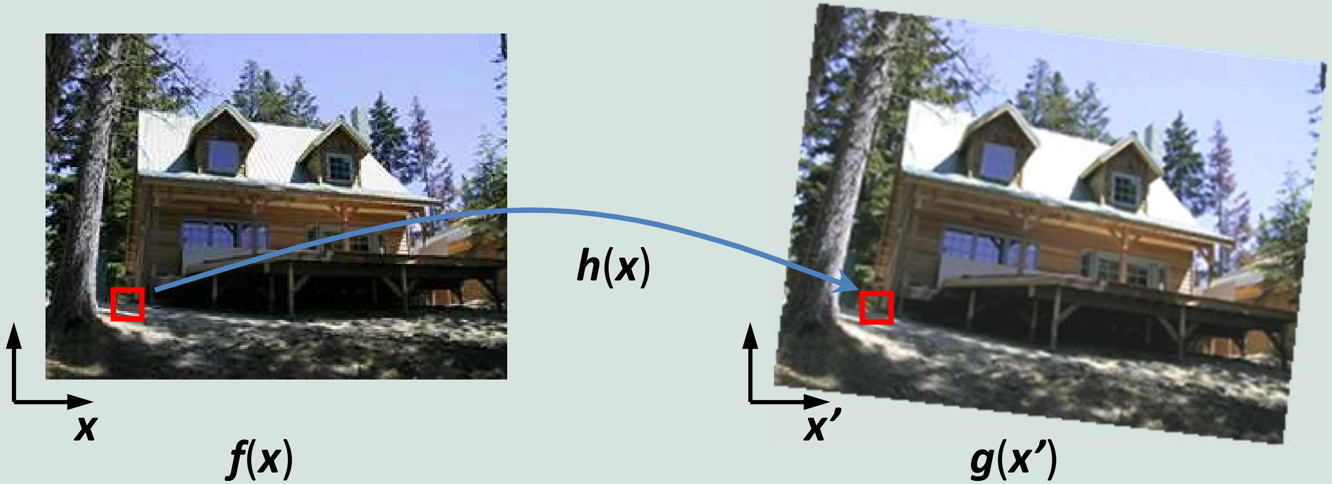
Image Warping

- Given a coordinate transform $\boldsymbol{x'} = \boldsymbol{h}(\boldsymbol{x})$ and a source image $\boldsymbol{f}(\boldsymbol{x})$, how do we compute a transformed image $\boldsymbol{g}(\boldsymbol{x'}) = \boldsymbol{f}(\boldsymbol{h}(\boldsymbol{x}))$?



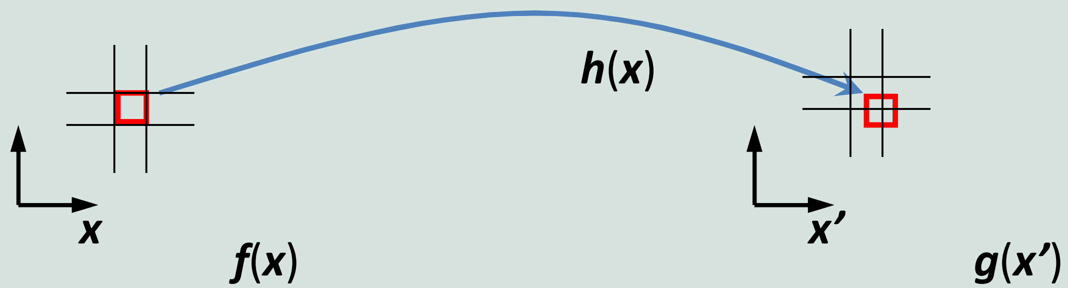
Forward Warping

- Send each pixel ***f***(***x***) to its corresponding location ***x'*** = ***h***(***x***) in ***g***(***x'***)
  - What if pixel lands "between" two pixels?



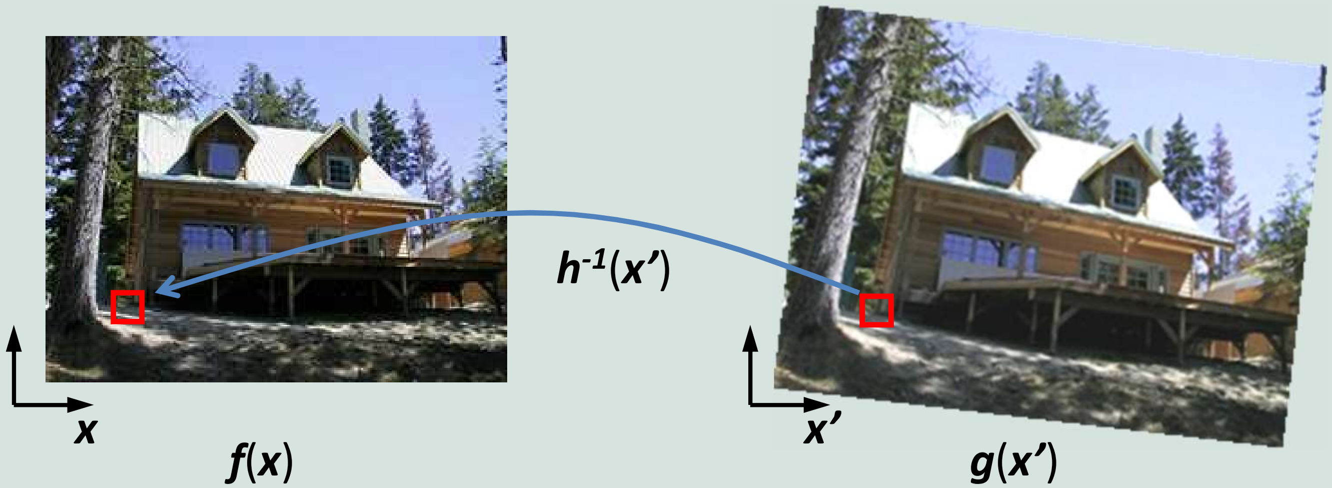
Forward Warping

- Send each pixel $\boldsymbol{f}(\boldsymbol{x})$ to its corresponding location $\boldsymbol{x}' = \boldsymbol{h}(\boldsymbol{x})$ in $\boldsymbol{g}(\boldsymbol{x}')$
  - What if pixel lands “between” two pixels?
  - Solution: add “contribution” to several pixels, normalize later (splatting)

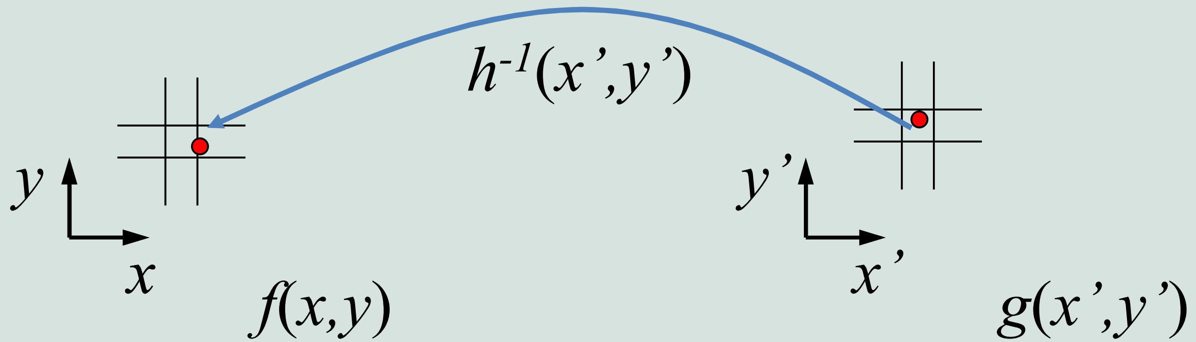


Inverse Warping

- Get each pixel $\boldsymbol{g}(\boldsymbol{x'})$ from its corresponding location $\boldsymbol{x'} = \boldsymbol{h}(\boldsymbol{x})$ in $\boldsymbol{f}(\boldsymbol{x})$
  - What if pixel comes from "between" two pixels?



Inverse Warping



Get each pixel $g(x',y')$ from its corresponding location

$(x,y) = h^{-1}(x',y')$ in the first image

Q: what if pixel comes from "between" two pixels?

A: *Interpolate* color value from neighbors

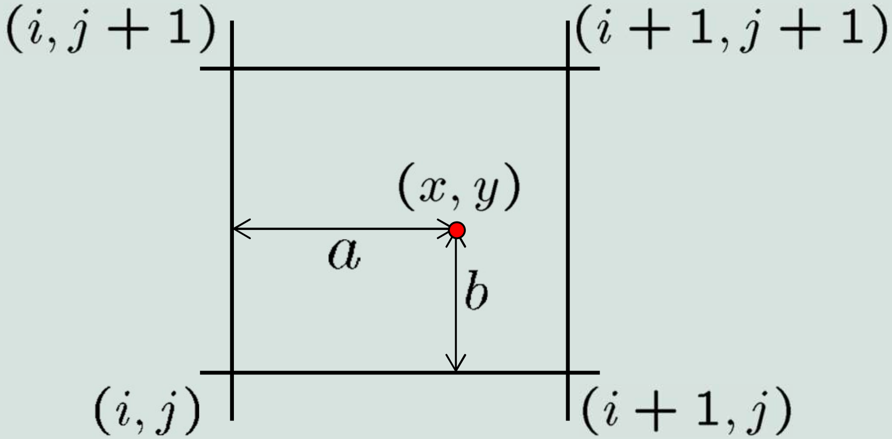
– nearest neighbor, bilinear…

Interpolation

- Possible interpolation filters:
  - nearest neighbor
  - bilinear
  - bicubic (interpolating)
  - sinc / FIR
- Needed to prevent “jaggies” and “texture crawl”

Bilinear interpolation

- Sampling at *f(x,y):*



$$
\begin{aligned}
f(x,y) = \quad & (1-a)(1-b) && f[i,j] \\
& +a(1-b) && f[i+1,j] \\
& +ab && f[i+1,j+1] \\
& +(1-a)b && f[i,j+1]
\end{aligned}
$$

Python Warp Function

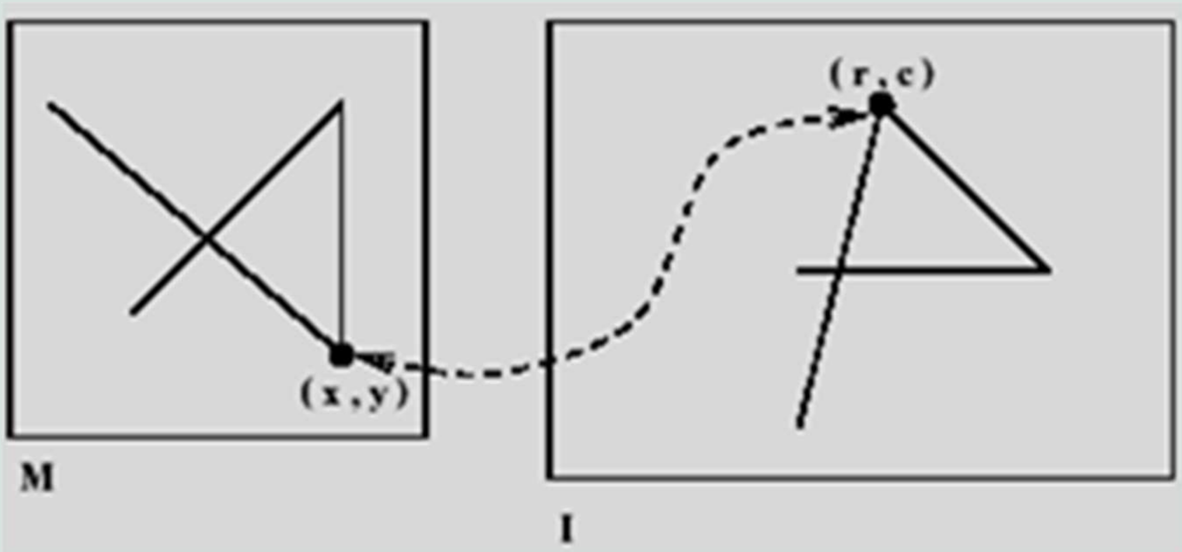
```
import cv2

translation_matrix = np.float32([[1,0,160],[0,1,40]])

transformed = cv2.warpAffine(img, translation_matrix,
(img.shape[1], img.shape[0]))

affine_tr = cv2.getAffineTransform(pts1,pts2)
transformed = cv2.warpAffine(img, affine_tr, (shape1,
shape2))

from skimage import transform as tf

img2 = tf.rotate(img1, 180)

tform = tf.AffineTransform(scale=(1.3, 1.1),
rotation=0.5,translation=(0, -200))

img3 = tf.warp(img1, tform)
```

Transformations

- How to make and use correspondences between
  - images and maps,
  - images and models,
  - images and other images
- Transformation may be in two or three dimensions

Image Registration

- Points of two images with similar viewpoints of essentially the same scene are geometrically transformed so that
  - corresponding feature points of the two images have the same coordinates after transformation



$$M[x, y] \cong I[g(x, y), h(x, y)]$$

$$I[r, c] \cong M[g^{-1}(r, c), h^{-1}(r, c)]$$

Representation of Points

- A 2D point has two coordinates and is conveniently represented as either
  - a row vector $P=[x,y]$
  - column vector $P=[x,y]^t$
- Homogeneous Coordinates:
  - A convenient notation for computer processing of points
    - especially when affine transformations are used.
- The homogeneous coordinates of a 2D point $P=[x,y]^t$ are $[s.x,s.y,s]^t$, where s is a scale factor and commonly 1.0

2D Coordinate Transformations

- translation: $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{t}$ $\quad \boldsymbol{x} = (x,y)$
- rotation: $\boldsymbol{x}' = \boldsymbol{R}\,\boldsymbol{x} + \boldsymbol{t}$
- similarity: $\boldsymbol{x}' = s\,\boldsymbol{R}\,\boldsymbol{x} + \boldsymbol{t}$
- affine: $\boldsymbol{x}' = \boldsymbol{A}\,\boldsymbol{x} + \boldsymbol{t}$
- perspective: $\underline{\boldsymbol{x}}' \cong \boldsymbol{H}\,\underline{\boldsymbol{x}}$ $\quad \underline{\boldsymbol{x}} = (x,y,1)$

  ($\underline{\boldsymbol{x}}$ is a *homogeneous* coordinate)

  – These all form a nested *group* (closed with inverse)

Basic 2D Transformations

- Basic 2D transformations as 3x3 matrices

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translate

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{s}_{\boldsymbol{x}} & 0 & 0 \\ 0 & \boldsymbol{s}_{\boldsymbol{y}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

Scale

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotate

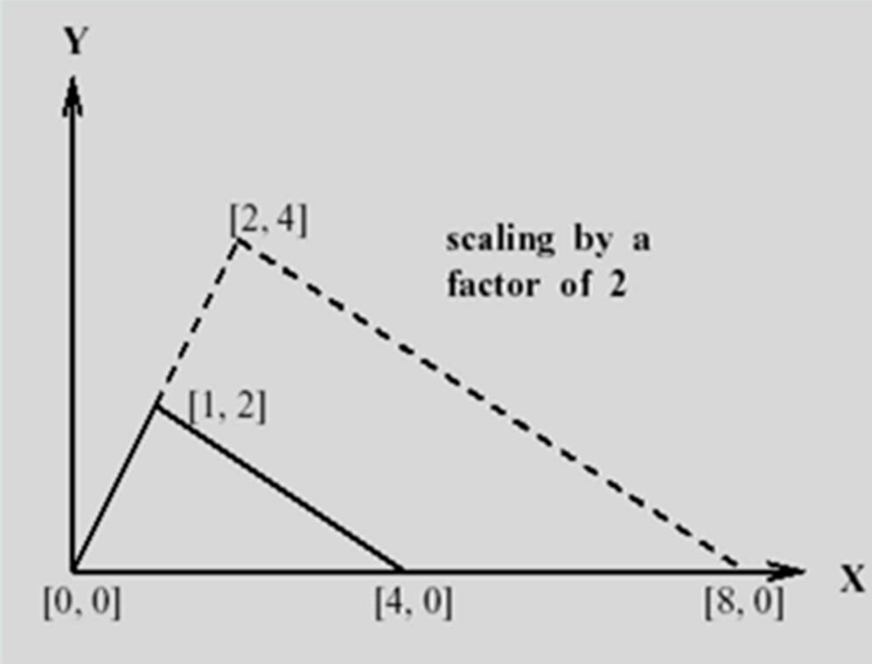
$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \boldsymbol{sh}_{\boldsymbol{x}} & 0 \\ \boldsymbol{sh}_{\boldsymbol{y}} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

Shear

Scaling

- Uniform scaling
  - changes all coordinates in the same way
  - equivalently changes the size of all objects in the same way.

- A 2D point P=[1,2] scaled by a factor of 2 to obtain the new point P'=[2,4].



Scaling

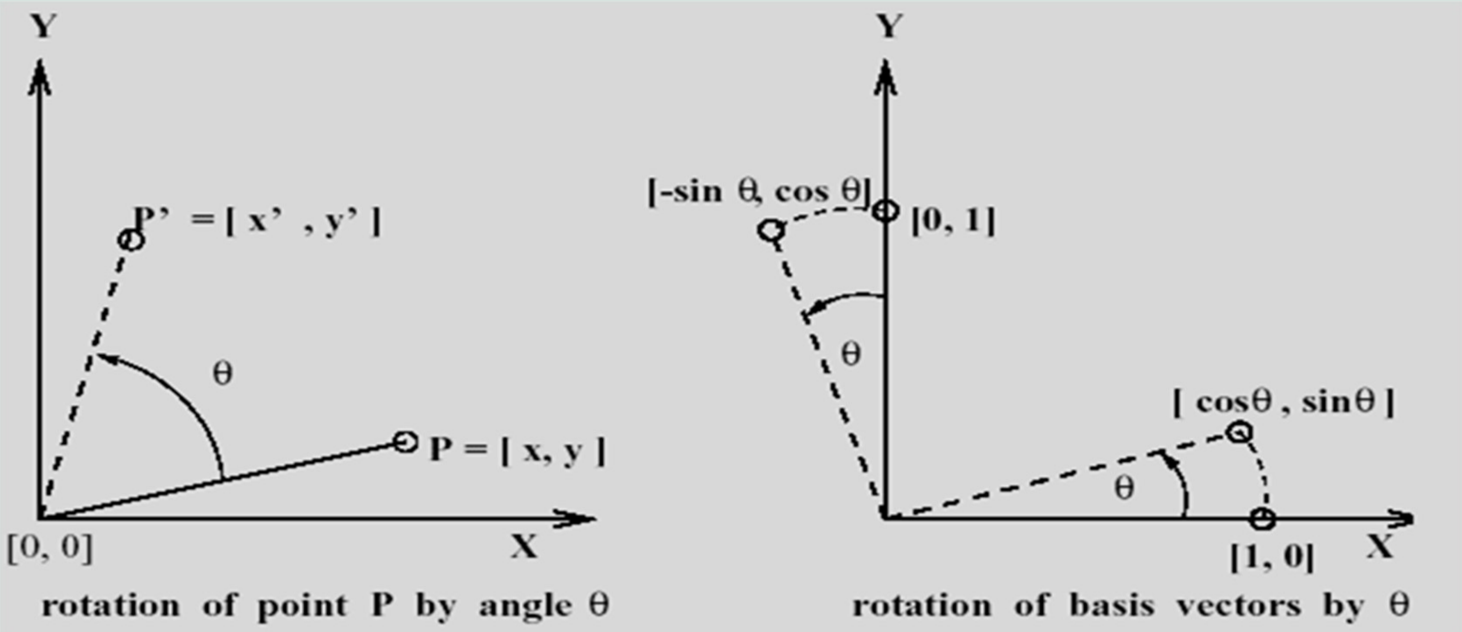
- Scaling is a linear transformation
  – can be easily represented in terms of the scale factor applied to the two basis vectors for 2D Euclidean space.
- For example,

$$[1,2]=1[1,0]+2[0,1] \text{ and}$$

$$2[1,2]=2(1[1,0]+2[0,1])=[2,4]$$

Rotation

- A second common operation is rotation about a point in 2D space.
- A 2D point P=[x,y] rotated by angle θ counterclockwise about the origin to obtain the new point P'=[x',y'].



Rotation

- Rotation is also a linear transformation,
  - the columns of the matrix are result of the transformation applied to the basis vectors
  - transformation of any other vectors can be expressed as a linear combination of the basis vectors.

$$R_\theta(x.[1,0] + y.[0,1]) = (x.R_\theta.[1,0]) + (y.R_\theta.[0,1])$$

$$= x[\cos(\theta)\,, \sin(\theta)] + y[-\sin(\theta), \cos(\theta)]$$

$$[x', y'] = [x\cos(\theta) - y\sin(\theta)\,, x\sin(\theta) + y\cos(\theta)]$$

$$[x', y'] = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos(\theta) - y\sin(\theta) \\ x\sin(\theta) + y\cos(\theta) \end{bmatrix}$$

Orthogonal and Orthonormal Transformations

- **Orthogonal** A set of vectors is orthogonal if all pairs of vectors in the set are perpendicular (have scalar product of zero).
- **Orthonormal** A set of vectors is orthonormal if it is an orthogonal set and all vectors have unit length.

- *A rotation preserves both the length of the basis vectors and their orthogonality.*

Translation

- Point coordinates need to be shifted by some constant amount.
  - equivalent to changing the origin of the coordinate system.
- Since translation does not map the origin [0,0] to itself, we cannot model it using a simple 2x2 matrix as has been done for scaling and rotation.
  - In other words, **translation is not a linear operation.**

Translation

- Translation can be represented as the matrix multiplication in homogeneous coordinates.
- A translation T of point [x,y] so that [x',y']=T([x,y])=[x+t_x,y+t_x]

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

2D Affine Transformations

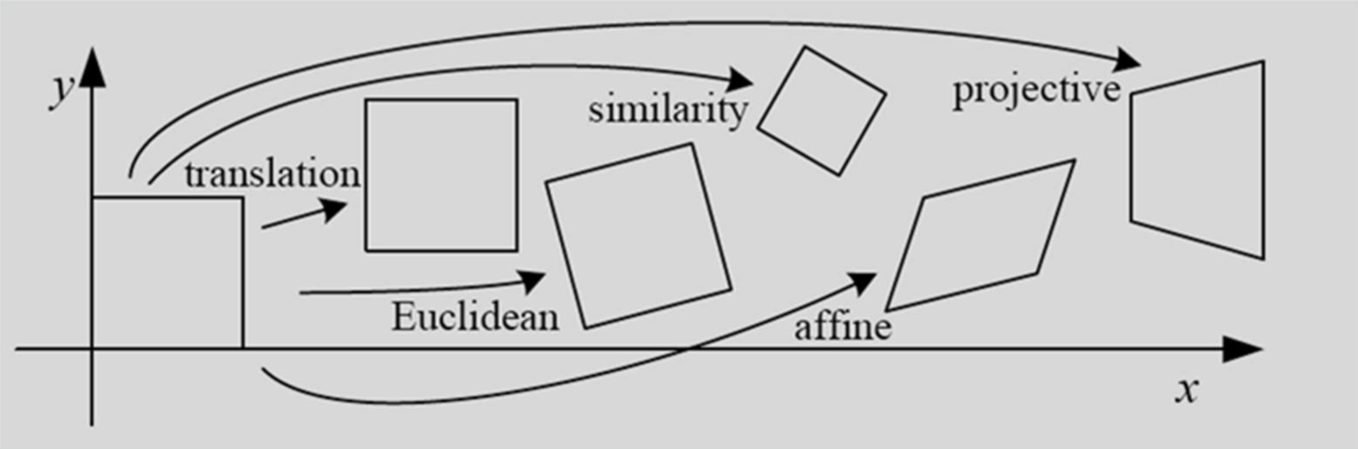
$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- Affine transformations are combinations of …
  - Linear transformations (rotation + scale), and
  - Translations
- Properties of affine transformations:
  - Origin does not necessarily map to origin
  - Lines map to lines
  - **<u>Parallel lines remain parallel</u>**
  - Ratios are preserved
  - Closed under composition
  - Models change of basis

Projective Transformations

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- Projective transformations:
  - Affine transformations, and
  - Projective warps
- Parallel lines do not necessarily remain parallel



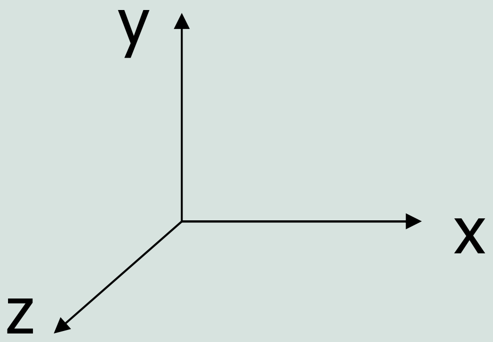
Fitting an Affine Transformation

Affine model approximates perspective projection of planar objects.

David Lowe, ICCV 1999

Points in 3D Space

- A 3D point (x,y,z) – x, y, and z coordinates
- Column vectors are used to represent points
  - Homogeneous coordinates of a 3D point $(x,y,z,1)^T$
- In general homogeneous coordinates in 3D
  - $[x,y,z,1]^T \equiv (x,y,z,w)$
- Transformation will be performed using 4x4 matrix



$$\begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} n_x.x + o_x y + a_x z + p_x \\ n_y.x + o_y y + a_y z + p_y \\ n_z.x + o_z y + a_z z + p_z \\ 1 \end{bmatrix}$$

Affine Transformations in 3-D

- Translation

$$T(d,h,l)=\begin{bmatrix}1&0&0&d\\0&1&0&h\\0&0&1&l\\0&0&0&1\end{bmatrix}$$

$$\begin{bmatrix}1&0&0&d\\0&1&0&h\\0&0&1&l\\0&0&0&1\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}=\begin{bmatrix}x+d\\y+h\\z+l\\1\end{bmatrix}$$

- Scaling

$$S(a,f,k)=\begin{bmatrix}a&0&0&0\\0&f&0&0\\0&0&k&0\\0&0&0&1\end{bmatrix}$$

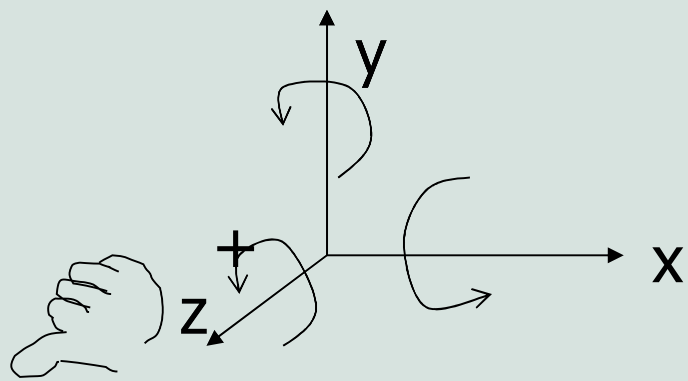
$$\begin{bmatrix}a&0&0&0\\0&f&0&0\\0&0&k&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}=\begin{bmatrix}ax\\fy\\kz\\1\end{bmatrix}$$

- Rotation?

$$\begin{bmatrix}\cos\theta&-\sin\theta&0\\\sin\theta&\cos\theta&0\\0&0&1\end{bmatrix}\begin{bmatrix}x\\y\\1\end{bmatrix}=\begin{bmatrix}x\cos\theta-y\sin\theta\\x\sin\theta+y\cos\theta\\1\end{bmatrix}$$

3D Rotation

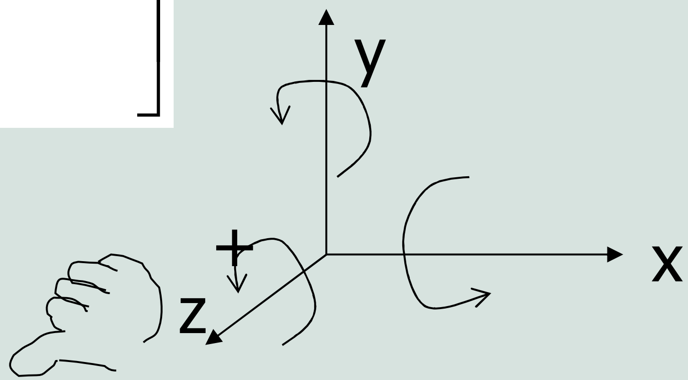
- 3D rotation is done around an axis
- Fundamental rotations – rotate about x, y, or z axes
- Counter-clockwise rotation is referred to as positive rotation
  - when you look from positive down to negative axis



Rotation about Z axis

- Same with 2-D version except the added additional row and column for the z axis.

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{z} \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}\cos\theta - \boldsymbol{y}\sin\theta \\ \boldsymbol{x}\sin\theta + \boldsymbol{y}\cos\theta \\ \boldsymbol{z} \\ 1 \end{bmatrix}$$



3D Transformation

- Rotation about y

$(z \rightarrow y,\ y \rightarrow x,\ x \rightarrow z)$

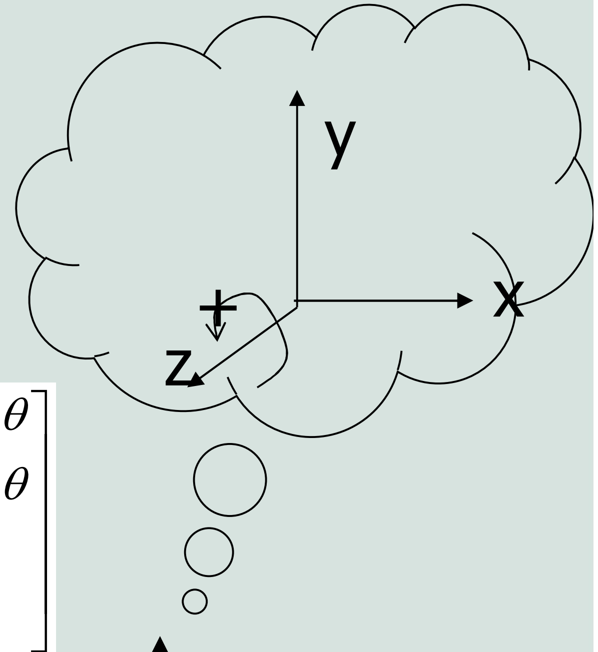
$z' = -x \sin(\theta) + z \cos(\theta)$

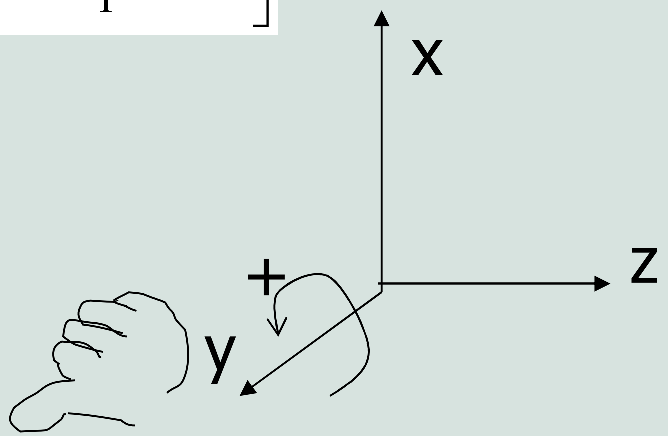
$x' = x \cos(\theta) + z \sin(\theta)$

$y' = y$

$$\begin{vmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$\begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \\ z \\ 1 \end{bmatrix}$$





3D Transformation

- Rotation about x

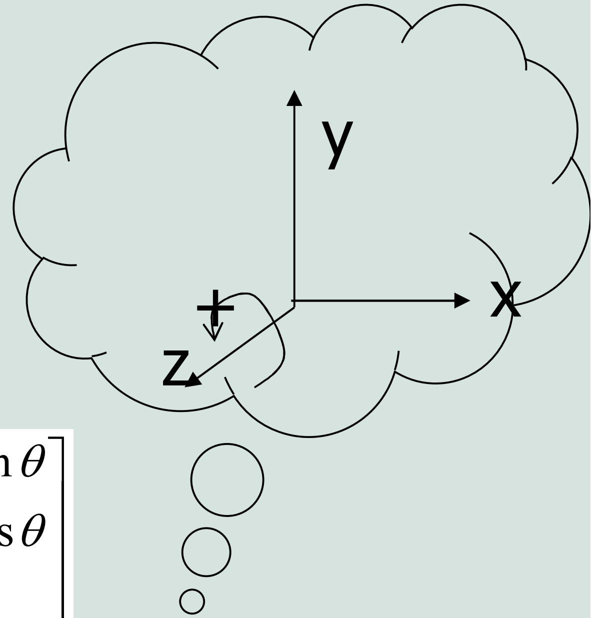
  $(z \rightarrow x,\ y \rightarrow z,\ x \rightarrow y)$

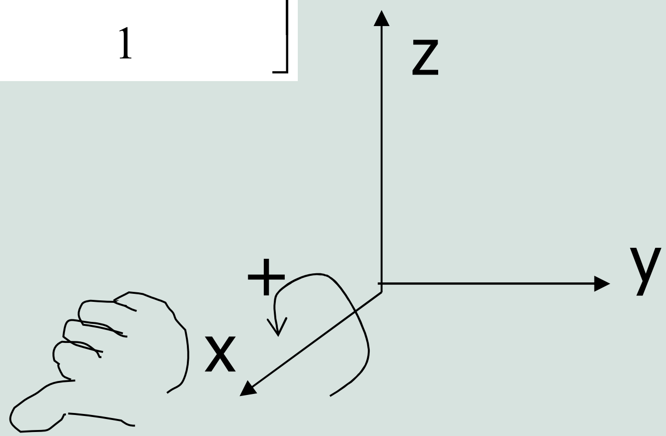
  $y' = y\cos(\theta) - z\sin(\theta)$

  $z' = y\sin(\theta) + z\cos(\theta)$

  $x' = x$

$$\begin{bmatrix} \boldsymbol{x}\cos\theta - \boldsymbol{y}\sin\theta \\ \boldsymbol{x}\sin\theta + \boldsymbol{y}\cos\theta \\ \boldsymbol{z} \\ 1 \end{bmatrix}$$





$$\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

The 3 Rotation Matrices

- Rotation about an arbitrary axis can be represented by only these three main matrices.

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Skew or Shear in 3-D

- In shearing x, y or z coordinates can be updated depending on the values of other point coordinates.

$$H_x(b) = \begin{bmatrix} 1 & b & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matrix for Perspective Projection?

- Division is needed to do projection!
  - But, matrix multiplication only does multiplication and addition
- What about:

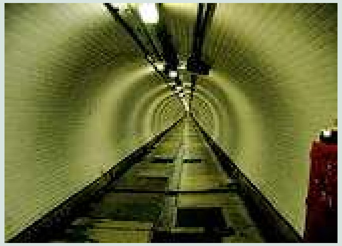
$$M_{per} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

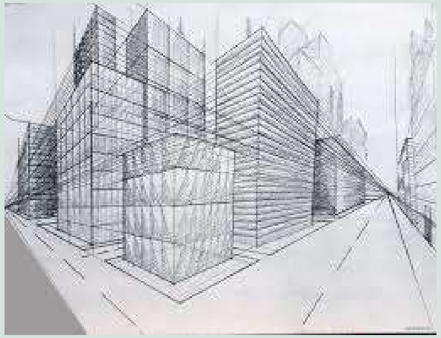
$$M_{per} \cdot P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

i-Point Perspective Transformation

1 - point

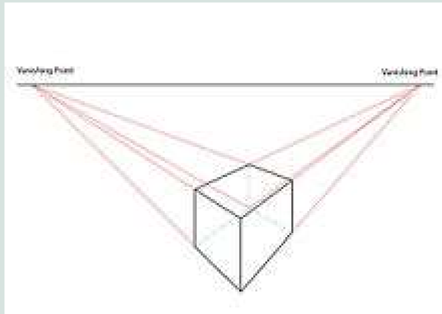
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix}$$

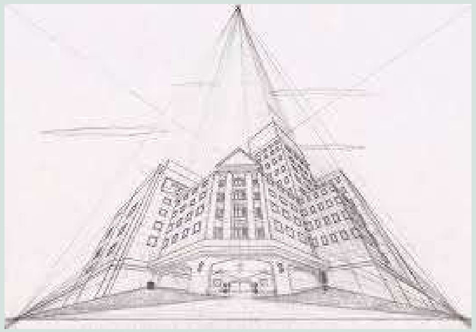




2 - point

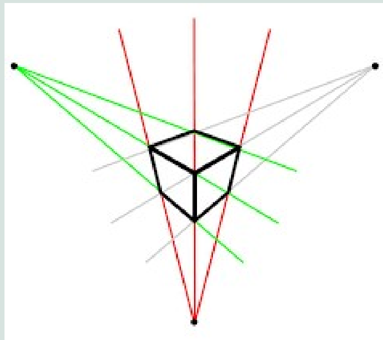
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & p & q & 1 \end{bmatrix}$$





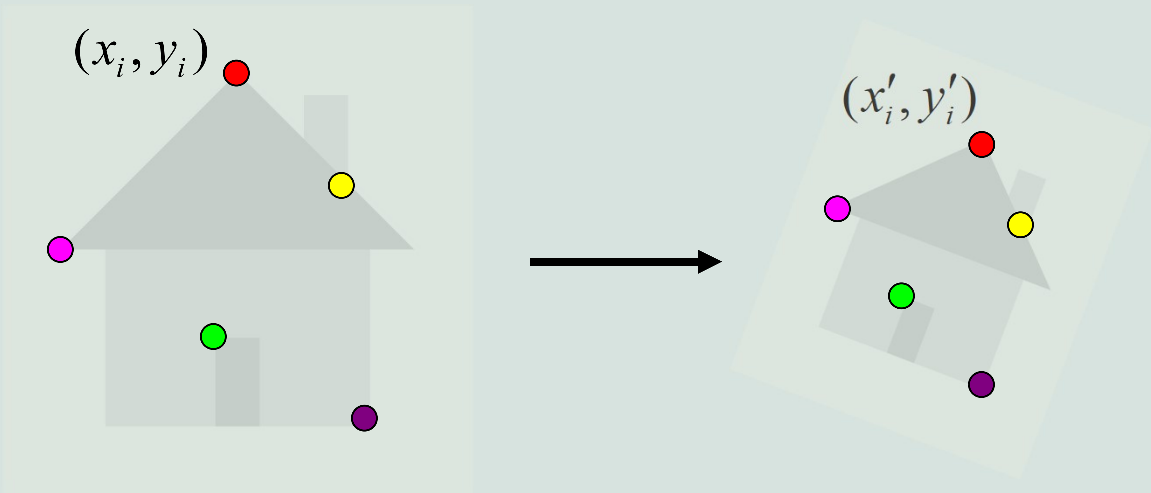
3 - point

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ p & q & r & 1 \end{bmatrix}$$



Fitting an Affine Transformation

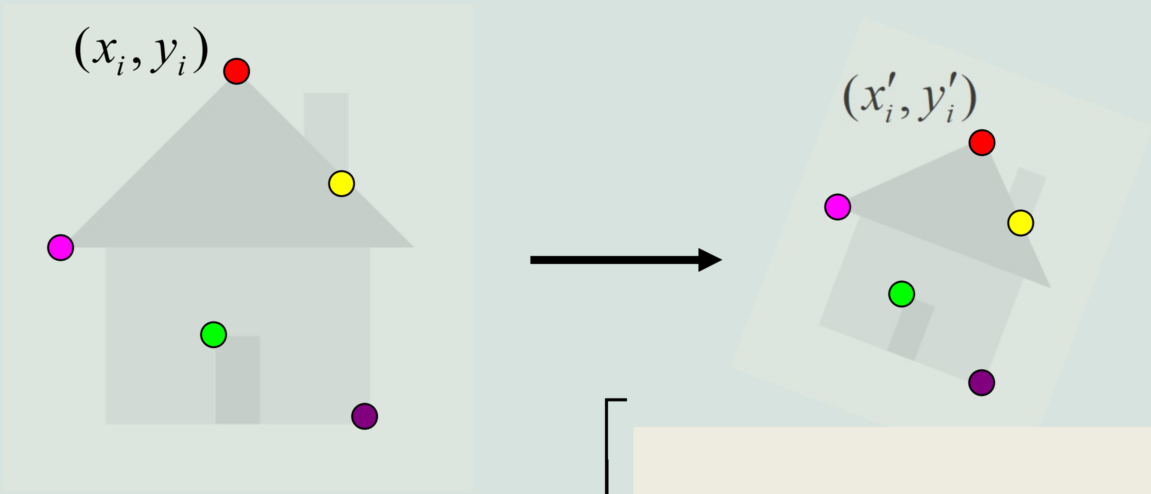
- Assuming we know the correspondences, how do we get the transformation?



$$\begin{bmatrix} x'_i \\ y'_i \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

Fitting an Affine Transformation

- Assuming we know the correspondences, how do we get the transformation?



$$\begin{bmatrix} x'_i \\ y'_i \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

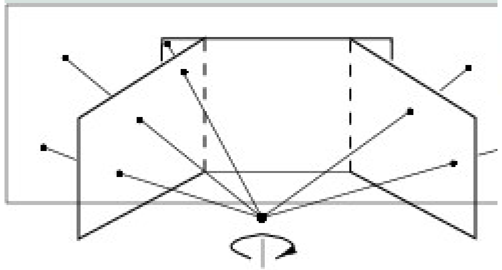
$$\begin{bmatrix} \\ \\ \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} \\ \\ \end{bmatrix}$$

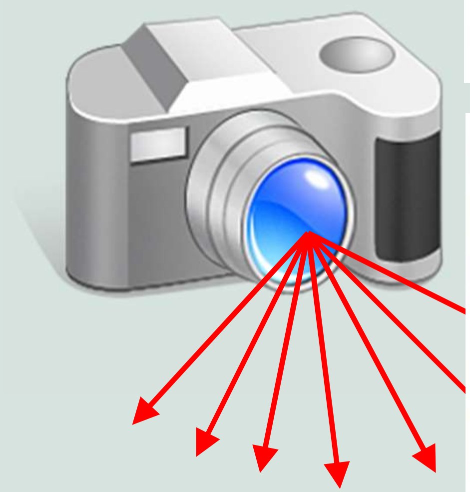
Fitting an Affine Transformation

$$\begin{bmatrix} & & \cdots & & & \\ x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \\ & & \cdots & & & \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} \cdots \\ x_i' \\ y_i' \\ \cdots \end{bmatrix}$$

- How many matches (correspondence pairs) do we need to solve for the transformation parameters?
- Once we have solved for the parameters, how do we compute the coordinates of the corresponding point for (x_{new}, y_{new})?

Panoramas





. . .

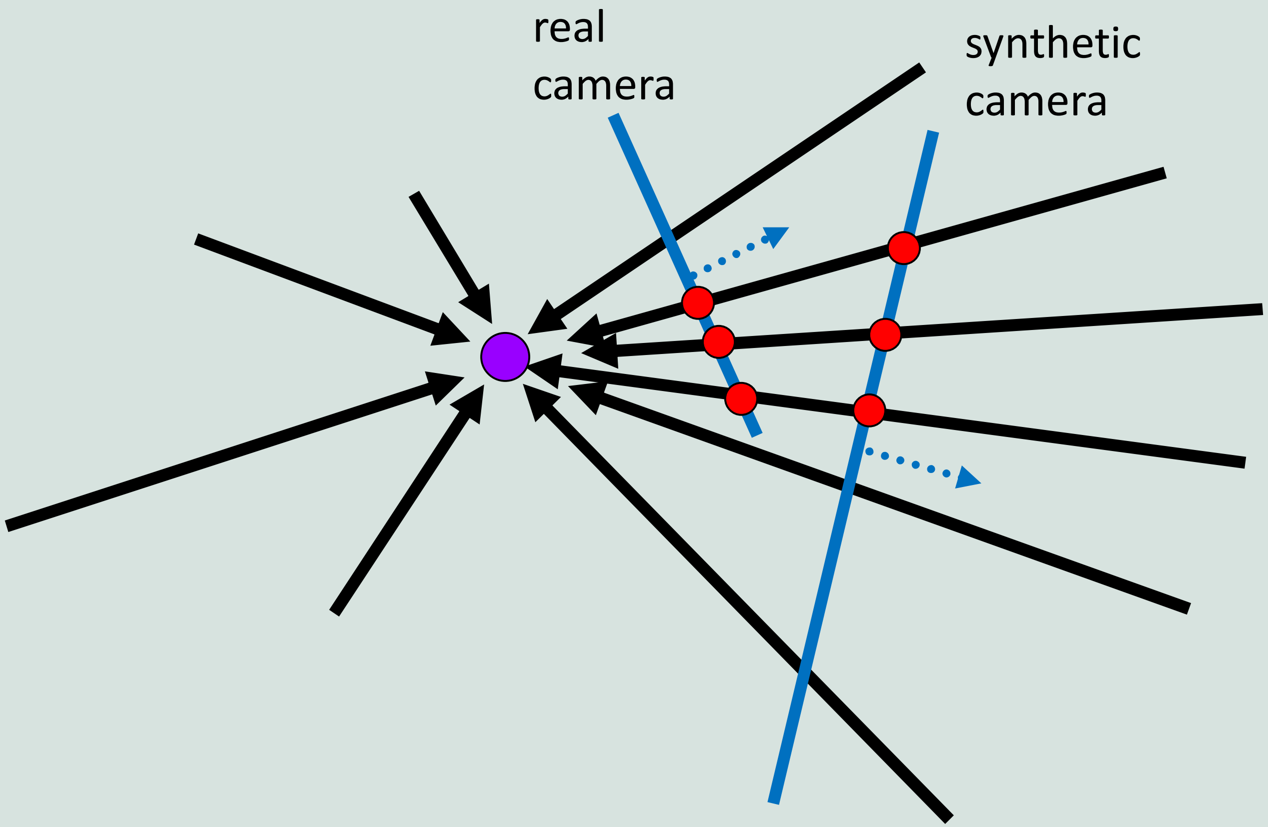
image from S. Seitz

Obtain a wider angle view by combining multiple images.

How to Stitch Together a Panorama?

- Basic Procedure
  - Take a sequence of images from the same position
    - Rotate the camera about its optical center
  - Compute transformation between second image and first
  - Transform the second image to overlap with the first
  - Blend the two together to form a mosaic
  - If there are more images, repeat
- …but **wait**, why should this work at all?
  - What about the 3D geometry of the scene?
  - Why aren't we using it?

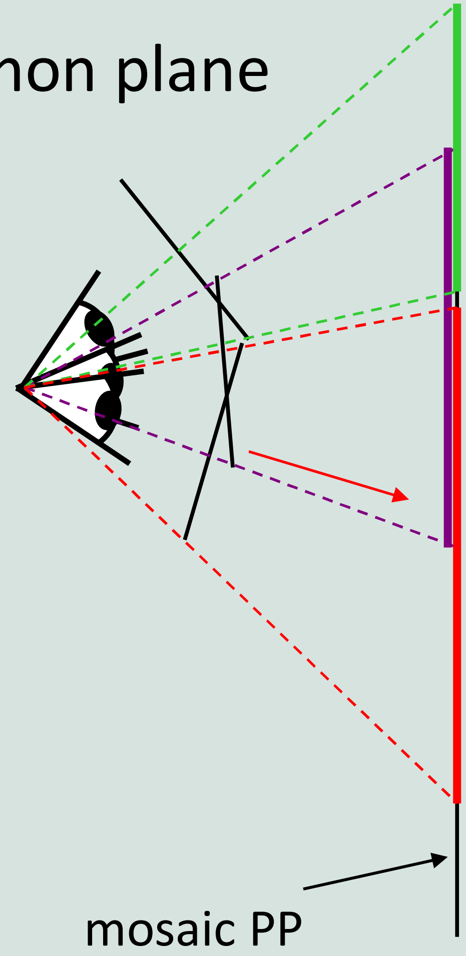
Panoramas: Generating Synthetic Views



Can generate any synthetic camera view
as long as it has **the same center of projection**!

Image reprojection

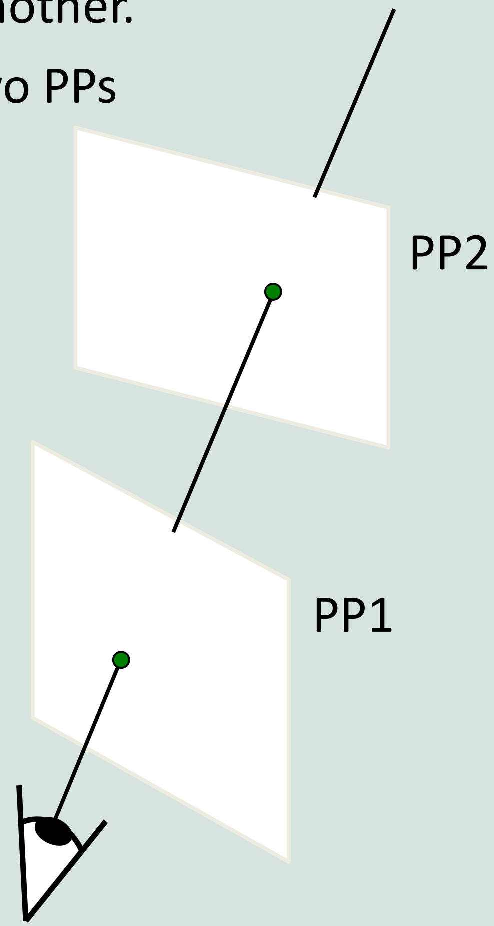
- The mosaic has a natural interpretation in 3D
  - The images are reprojected onto a common plane
  - The mosaic is formed on this plane
  - Mosaic is a *synthetic wide-angle camera*



Homography

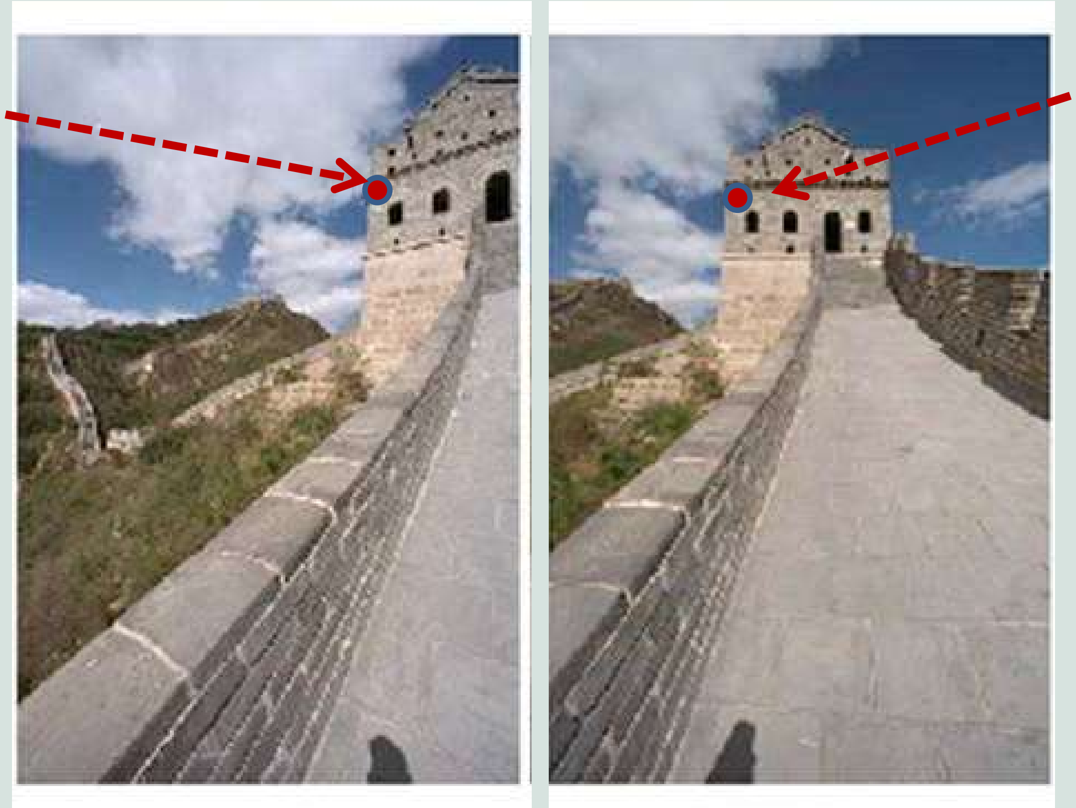
- How to relate two images from the same camera center?
  - how to map a pixel from PP1 to PP2?
- Think of it as a 2D **image warp** from one image to another.
- A projective transform is a mapping between any two PPs with the same center of projection
  - rectangle should map to arbitrary quadrilateral
  - parallel lines aren't parallel
  - but must preserve straight lines
  - called **Homography**

$$\underbrace{\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix}}_{\mathbf{p'}} = \underbrace{\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}}_{\mathbf{H}} \underbrace{\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}}_{\mathbf{p}}$$



Homography

$$\left(x, y\right)$$



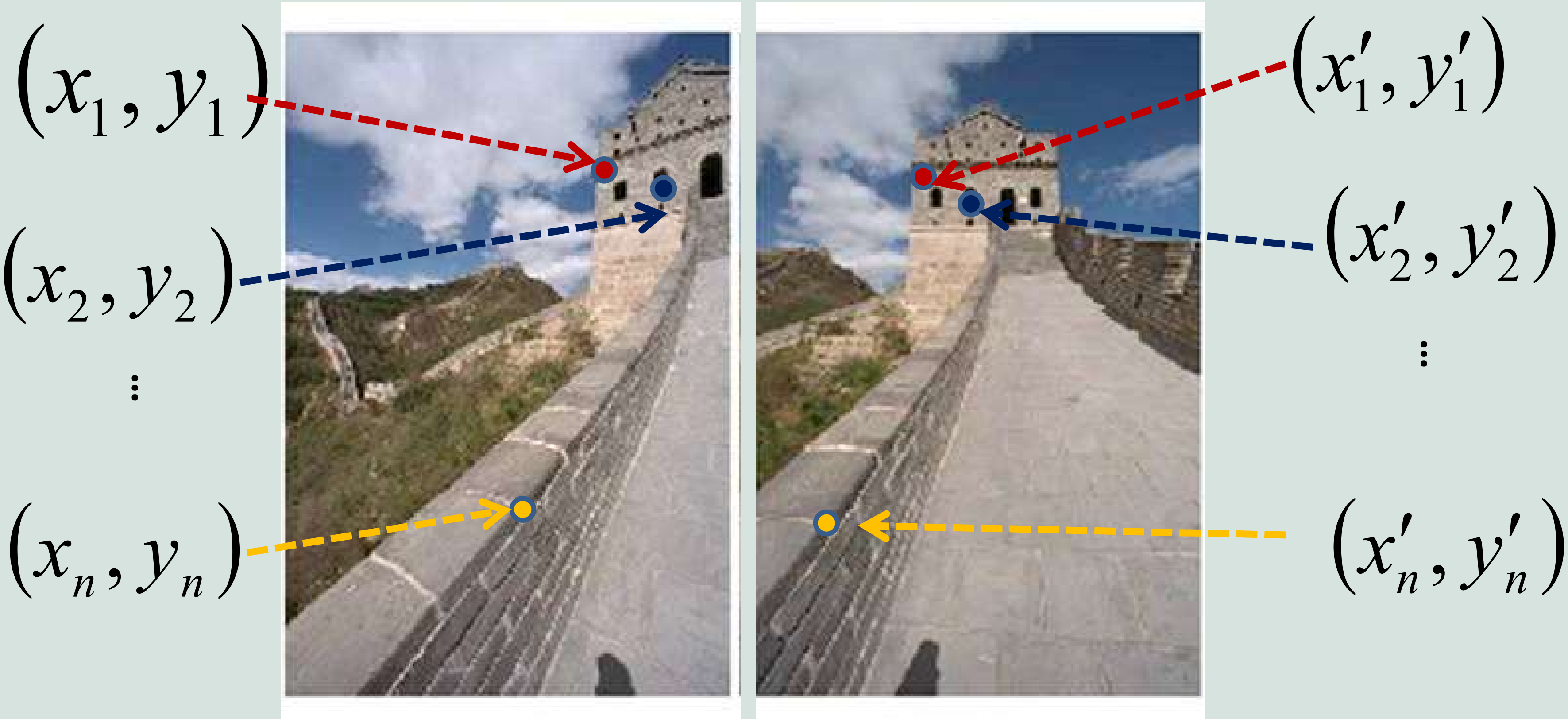
$$\left({}^{wx'}\!/_{w}, {}^{wy'}\!/_{w}\right) = \left(x', y'\right)$$

To **apply** a given homography **H**

- Compute **p'** = **Hp** (regular matrix multiply)
- Convert **p'** from homogeneous to image coordinates

$$\underset{\mathbf{p'}}{\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix}} = \underset{\mathbf{H}}{\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}} \underset{\mathbf{p}}{\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}}$$

Homography



To **compute** the homography given pairs of corresponding points in the images, we need to set up an equation where the parameters of **H** are the unknowns…

Solving for Homographies

p' = Hp

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Can set scale factor *i*=1. So, there are 8 unknowns.
- Set up a system of linear equations:

Ah = b

where vector of unknowns $h = [a,b,c,d,e,f,g,h]^T$

- Need at least 8 eqs, but the more the better…
- Solve for h. If overconstrained, solve using least-squares:

$$\min \|Ah - b\|^2$$

- Practical solutions to equation are through functions such as
  - `lmdivide`, `svd`

Image Warping with Homographies

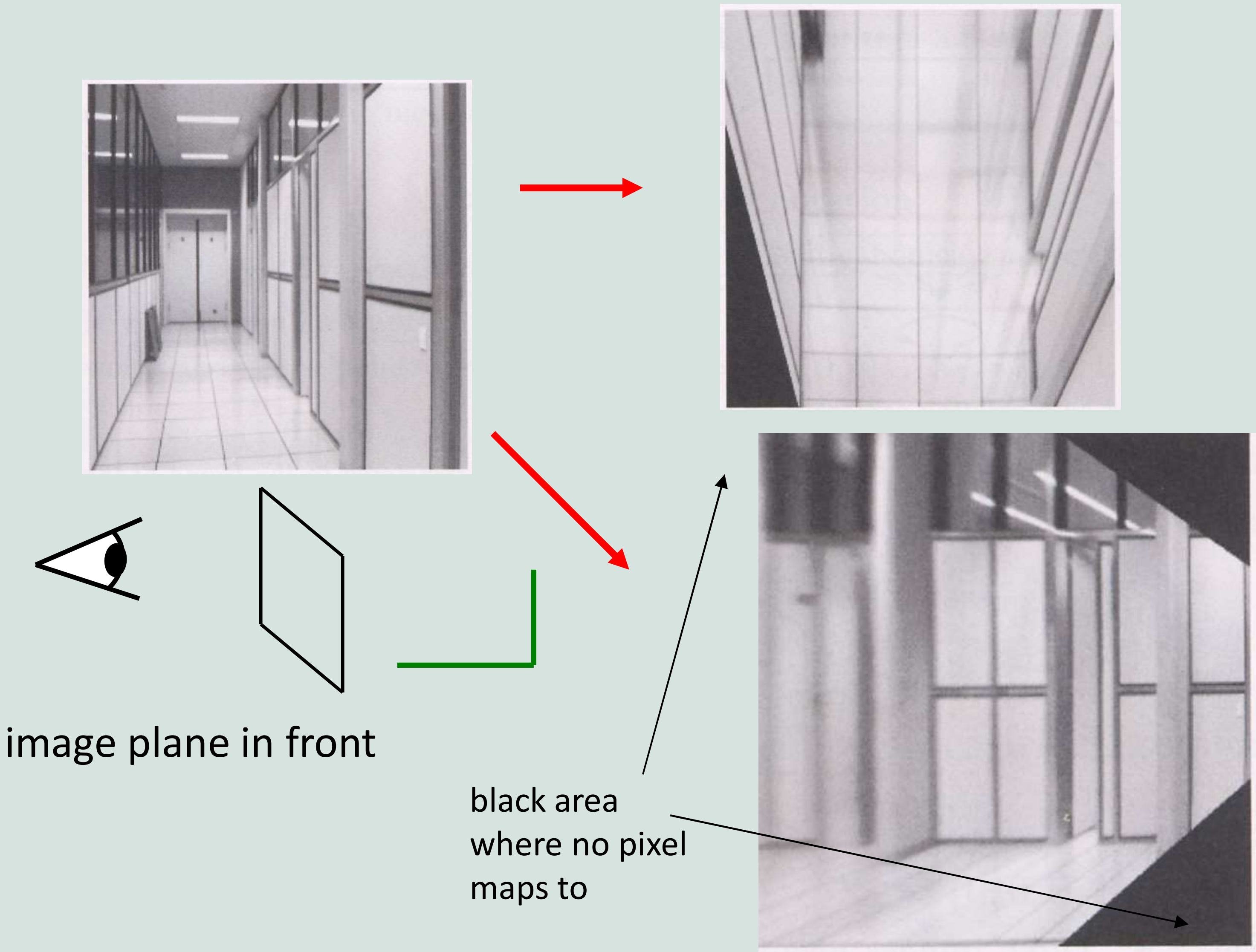
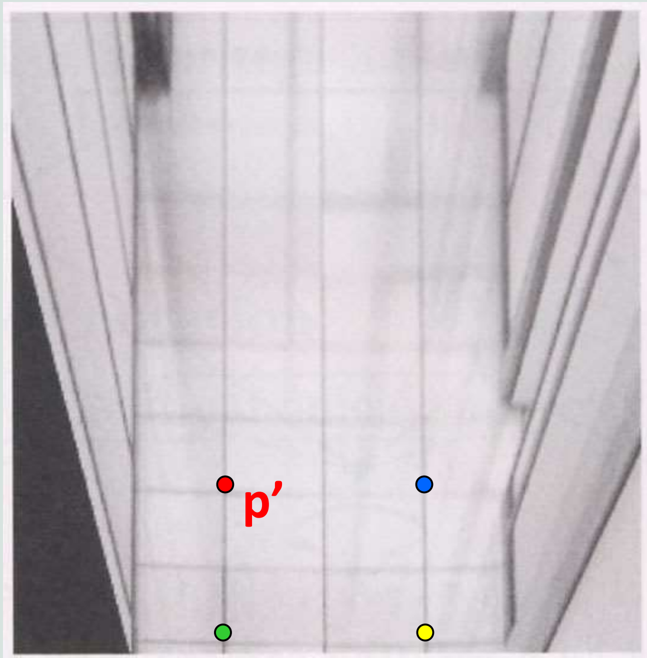
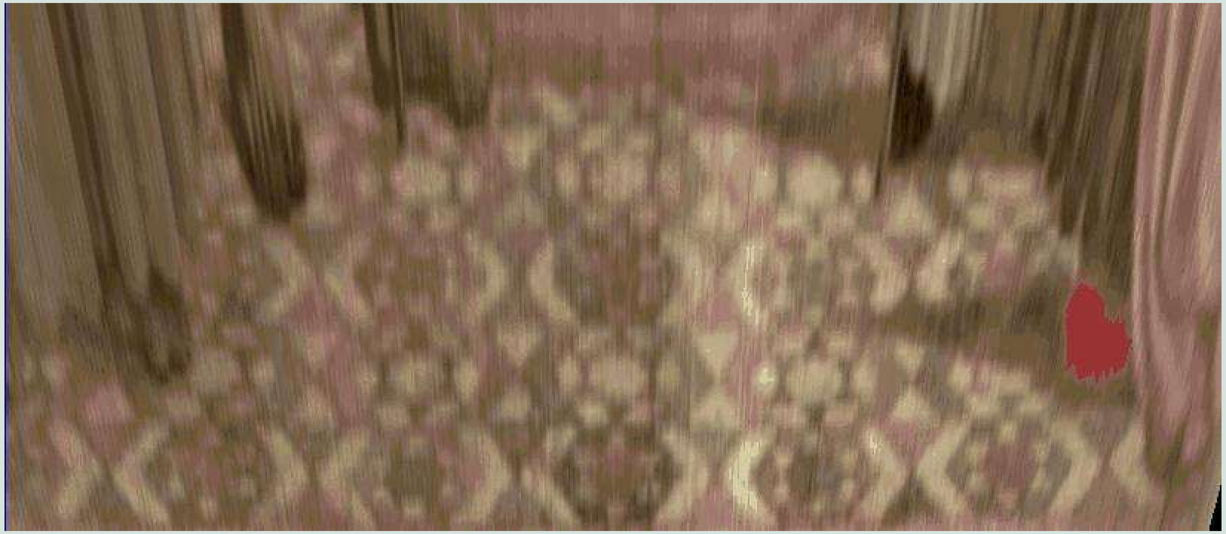


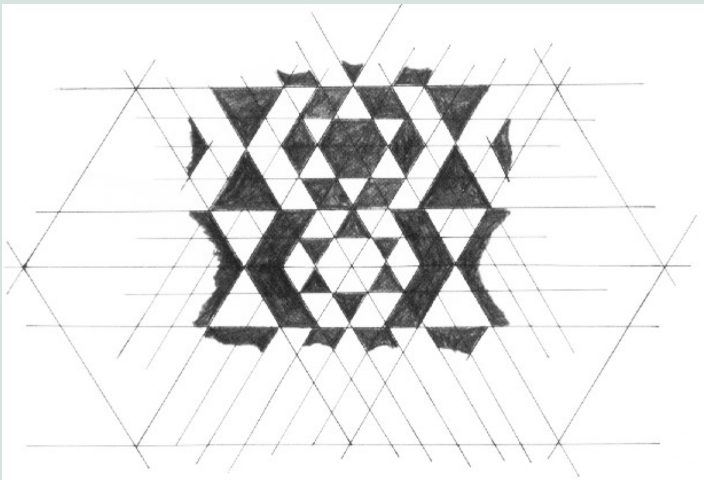
Image Rectification



Analysing Patterns and Shapes



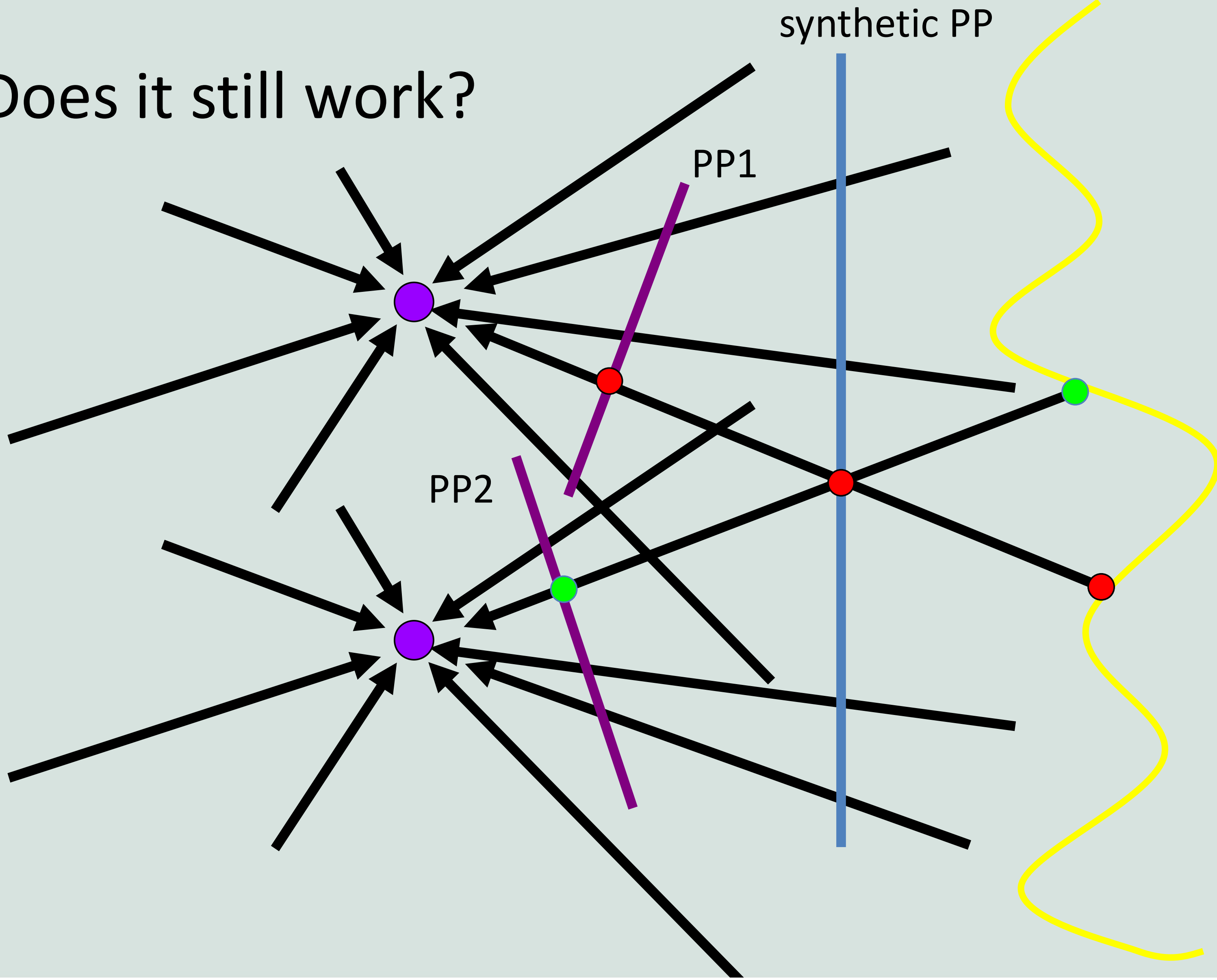
Automatic rectification



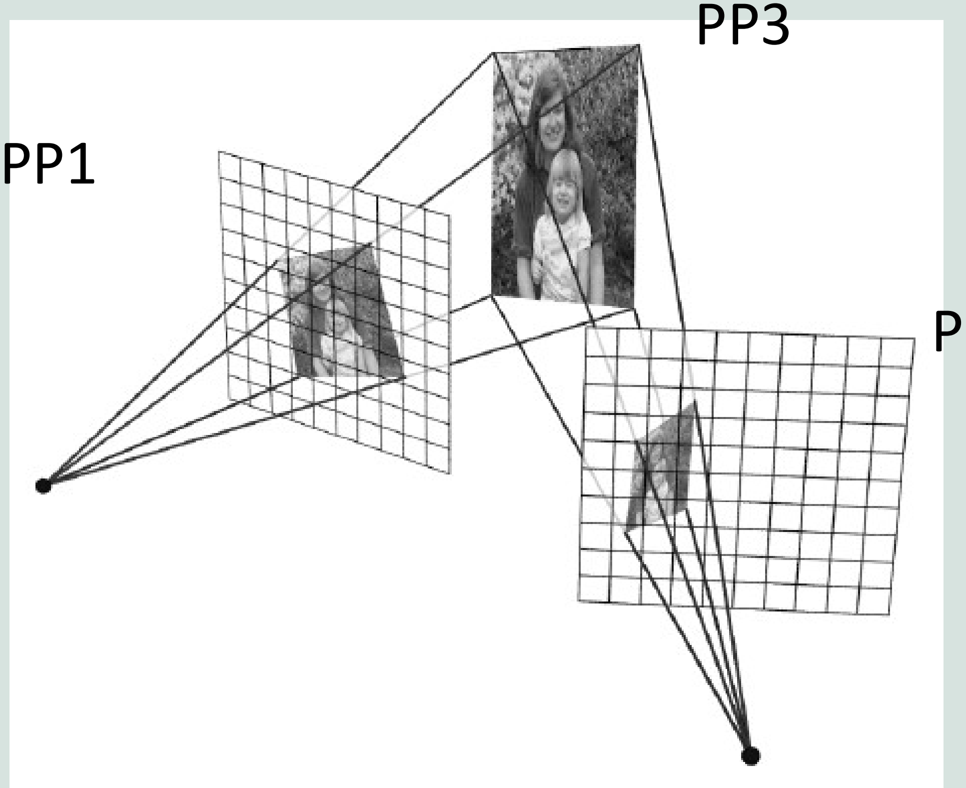
From Martin Kemp, *The Science of Art* *(manual reconstruction)*

Changing Camera Center

- Does it still work?



Planar Scene (Far-away Scene)

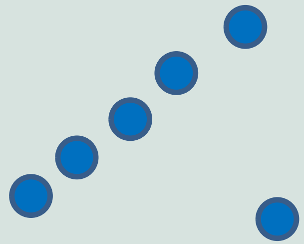


- PP3 is a projection plane of both centers of projection, so we are OK!
- This is how big aerial photographs are made.

Outliers

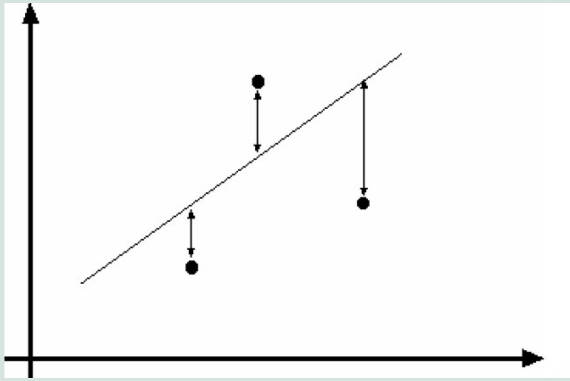
- **Outliers** can hurt the quality of our parameter estimates, e.g.,
  - an erroneous pair of matching points from two images
  - an edge point that is noise, or doesn't belong to the line we are fitting.

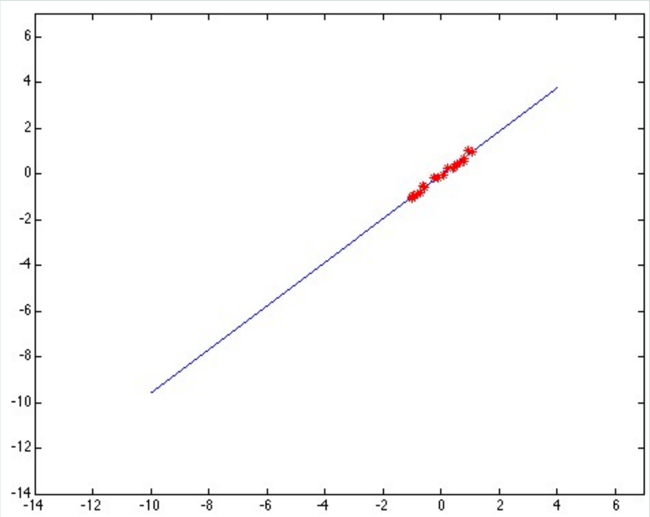


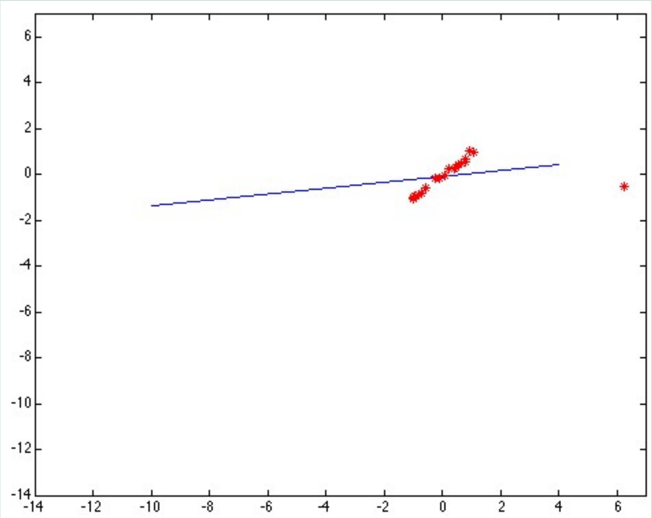


Least Squares Line Fitting

- Assuming all the points that belong to a particular line are known
- Outliers affect least squares fit







Random Sample Consensus (RANSAC)

- Approach:
  - avoid the impact of outliers
  - look for inliers and use those only
- Intuition:
  - if an outlier is chosen to compute the current fit, then the resulting line won't have much support from rest of the points.

RANSAC

- RANSAC algorithm:
  1. Randomly select a *seed group* of points on which to base transformation estimate (e.g., a group of matches)
  2. Compute transformation **H** from seed group
  3. Find *inliers* to this transformation

$$p_i' = H.p_i\;, p_i \rightarrow {}^m p_i,$$

$$if\;\; \|{}^m p_i - p_i'\| < thresh \;\text{ then }\; (p_i, {}^m p_i) \;\text{ is inlier to } \mathbf{H}$$

$$\text{even if } (p_i, {}^m p_i) \text{ matching pair is incorrect.}$$

  4. If the number of inliers is sufficiently large, re-compute least-squares estimate of transformation on all of the inliers

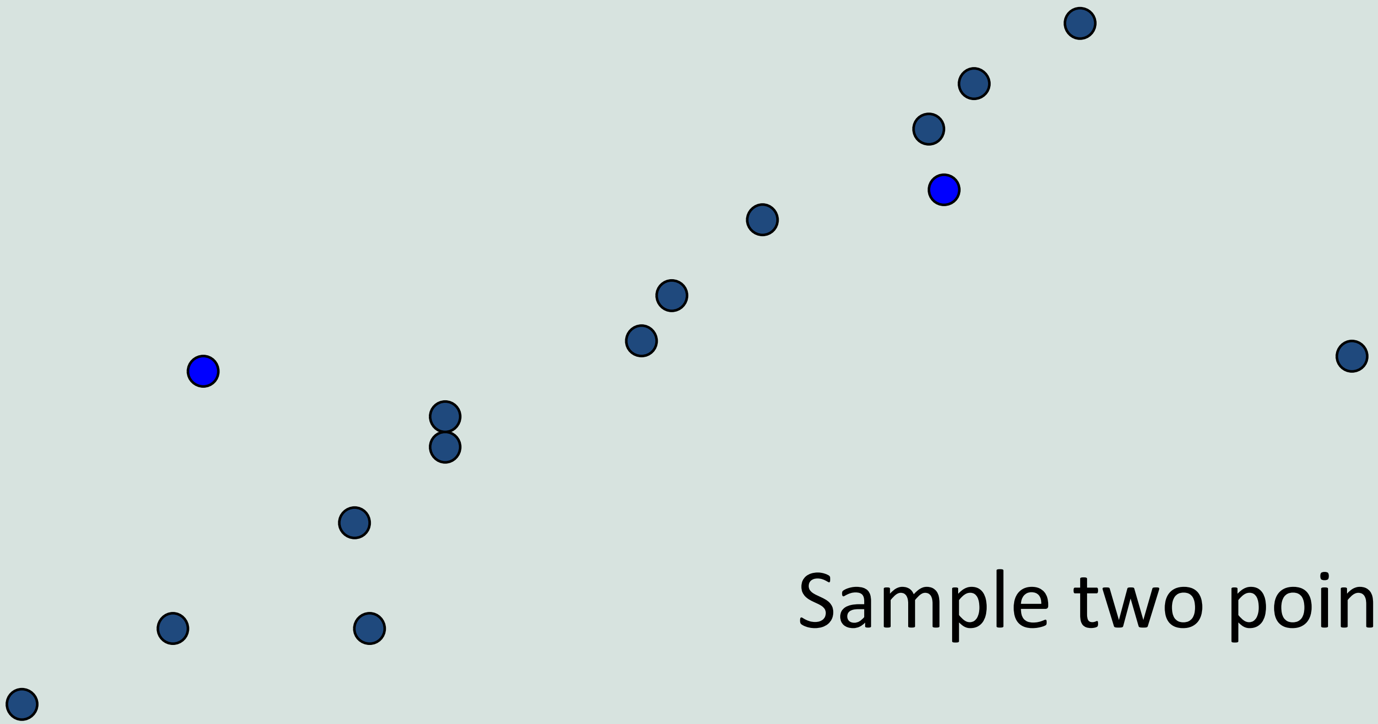
  – Keep the transformation with the largest number of inliers

RANSAC Line Fitting Example

Task:

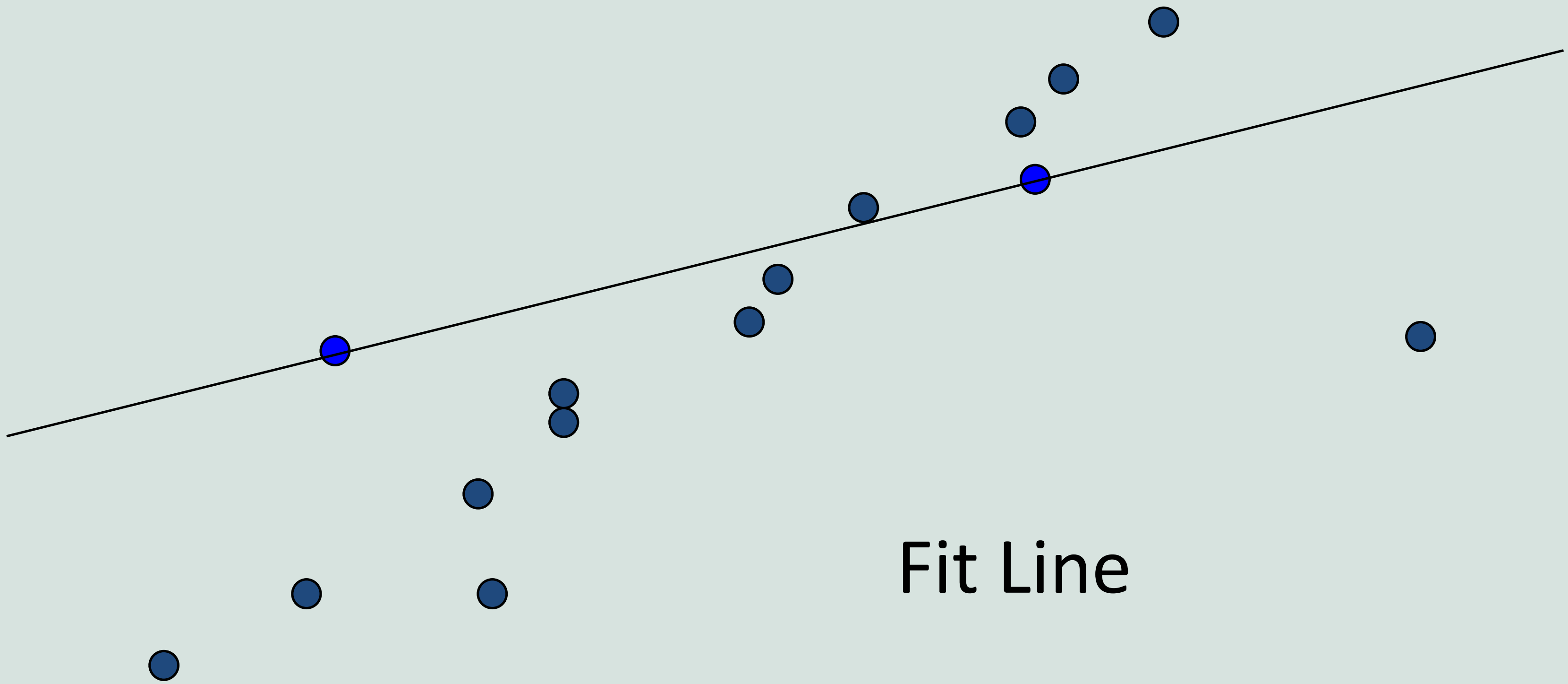
Estimate best line

RANSAC Line Fitting Example

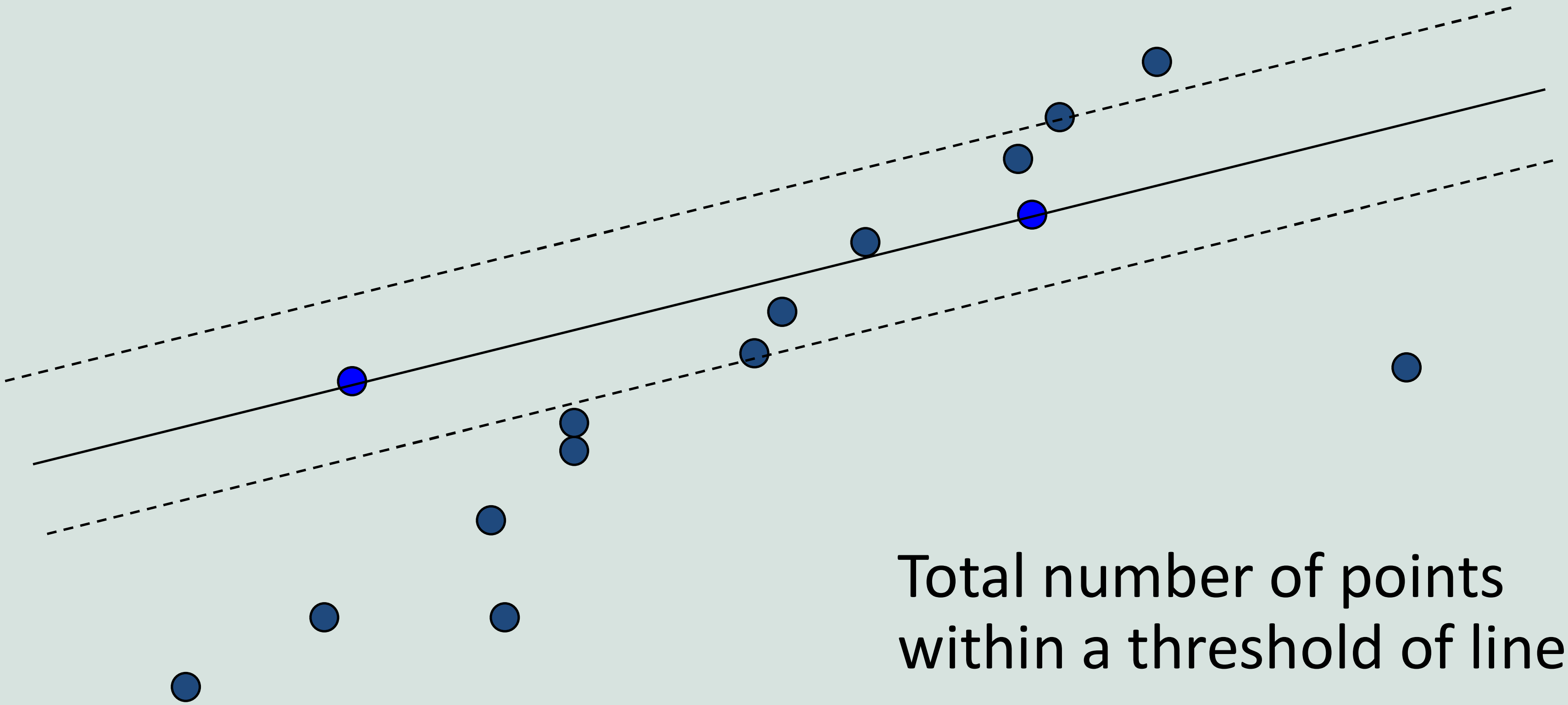


Sample two points

RANSAC Line Fitting Example



RANSAC Line Fitting Example

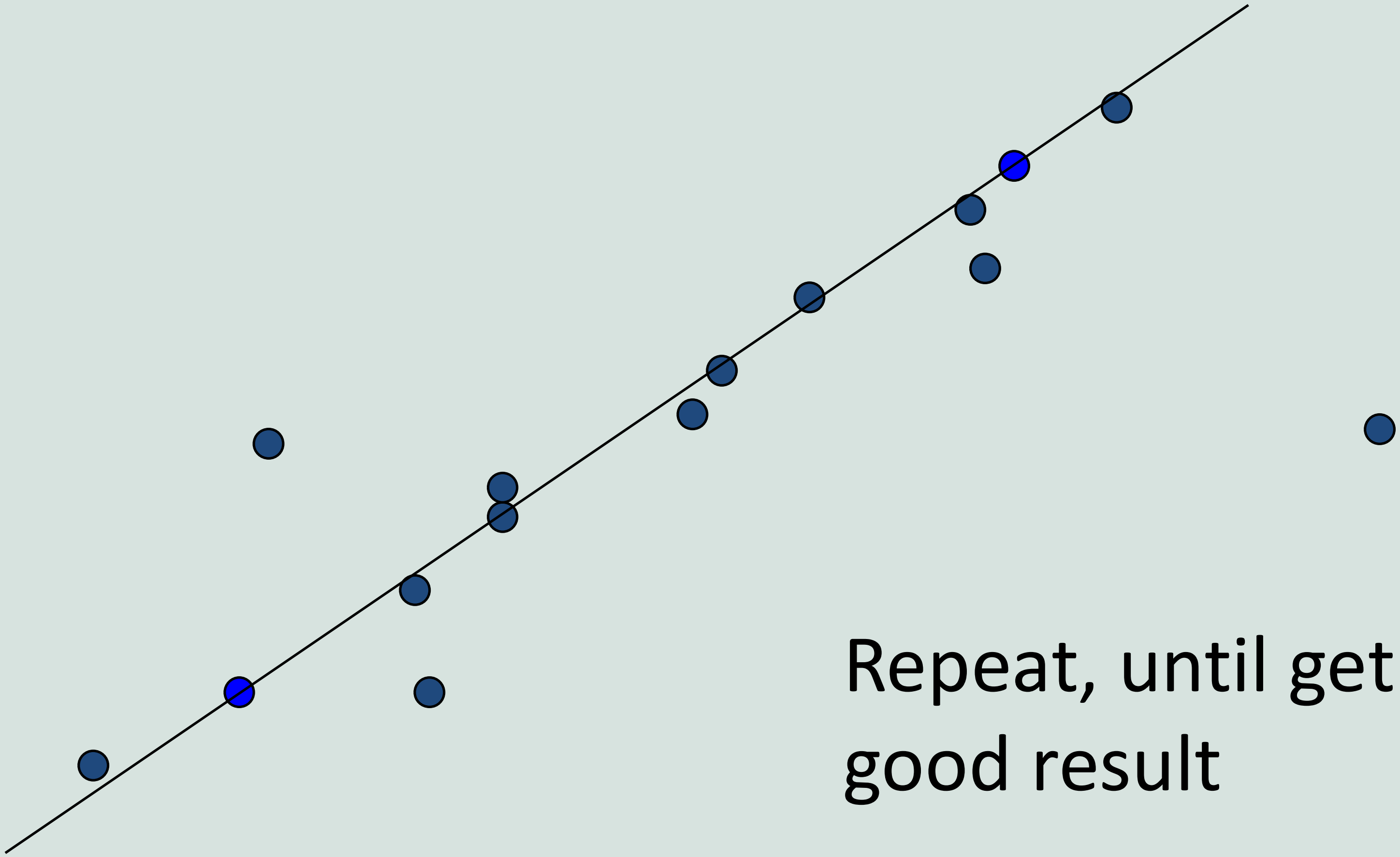


Total number of points within a threshold of line.

RANSAC Line Fitting Example

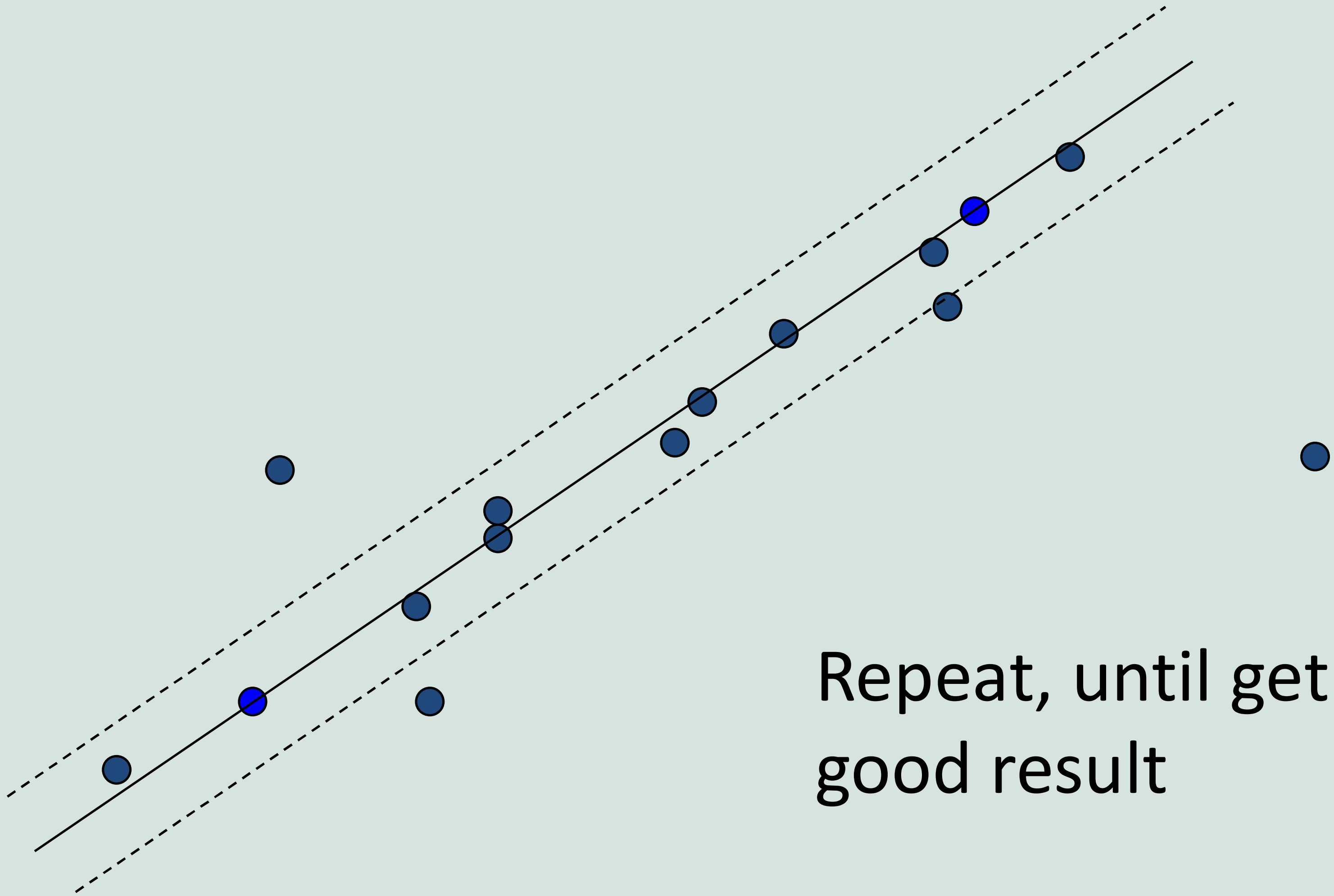
Repeat, until get a good result

RANSAC Line Fitting Example



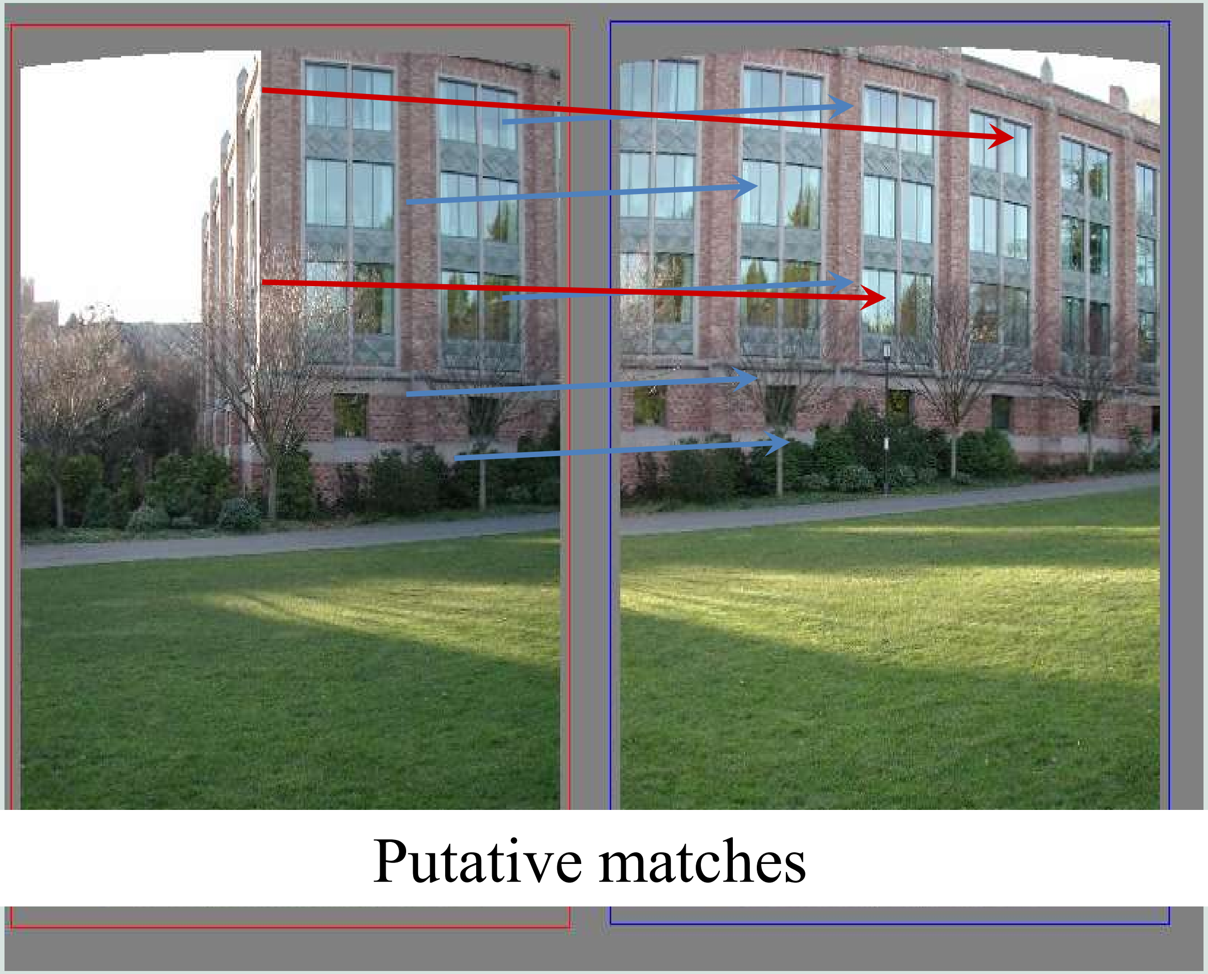
Repeat, until get a good result

RANSAC Line Fitting Example



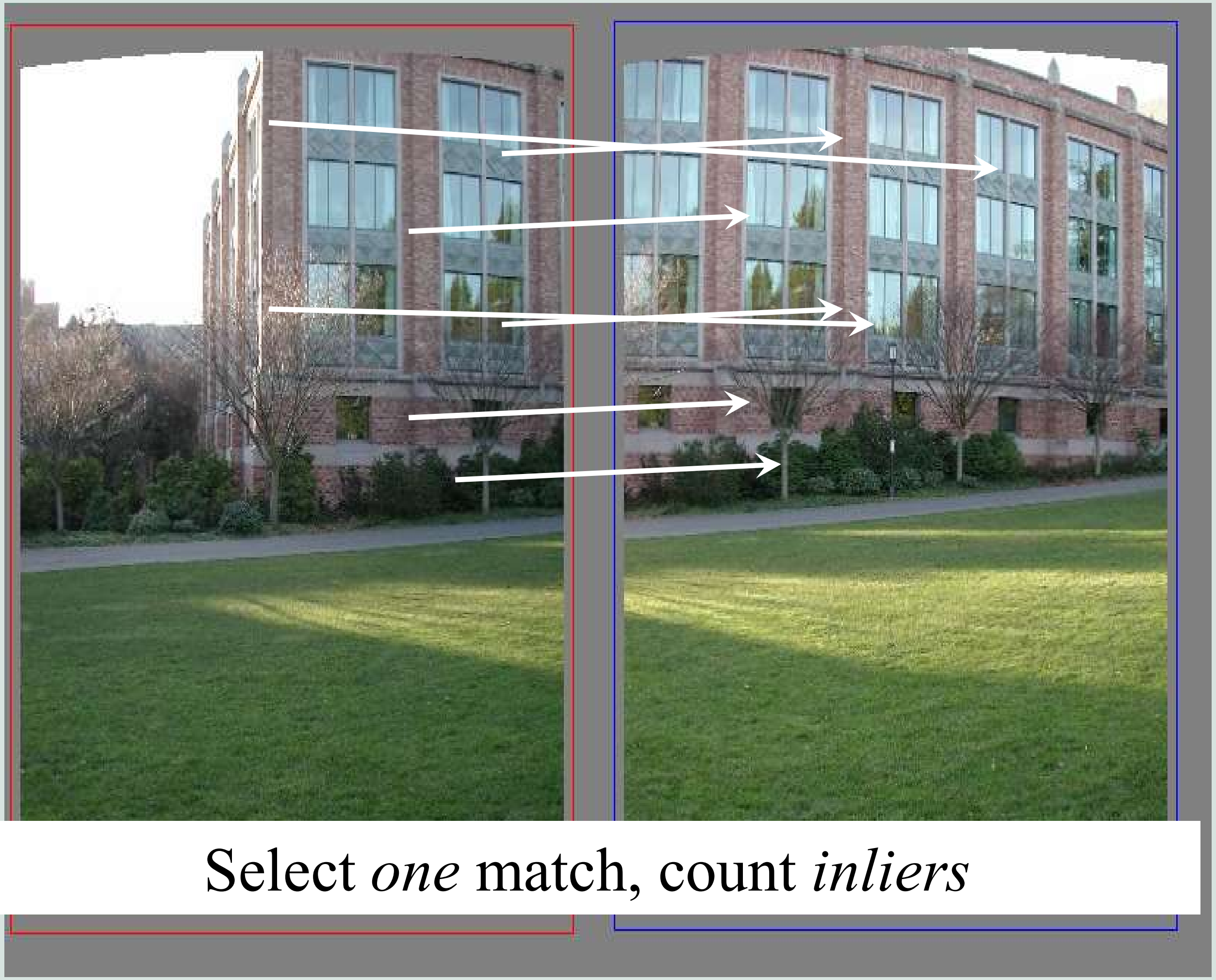
Repeat, until get a good result

RANSAC Example: Translation



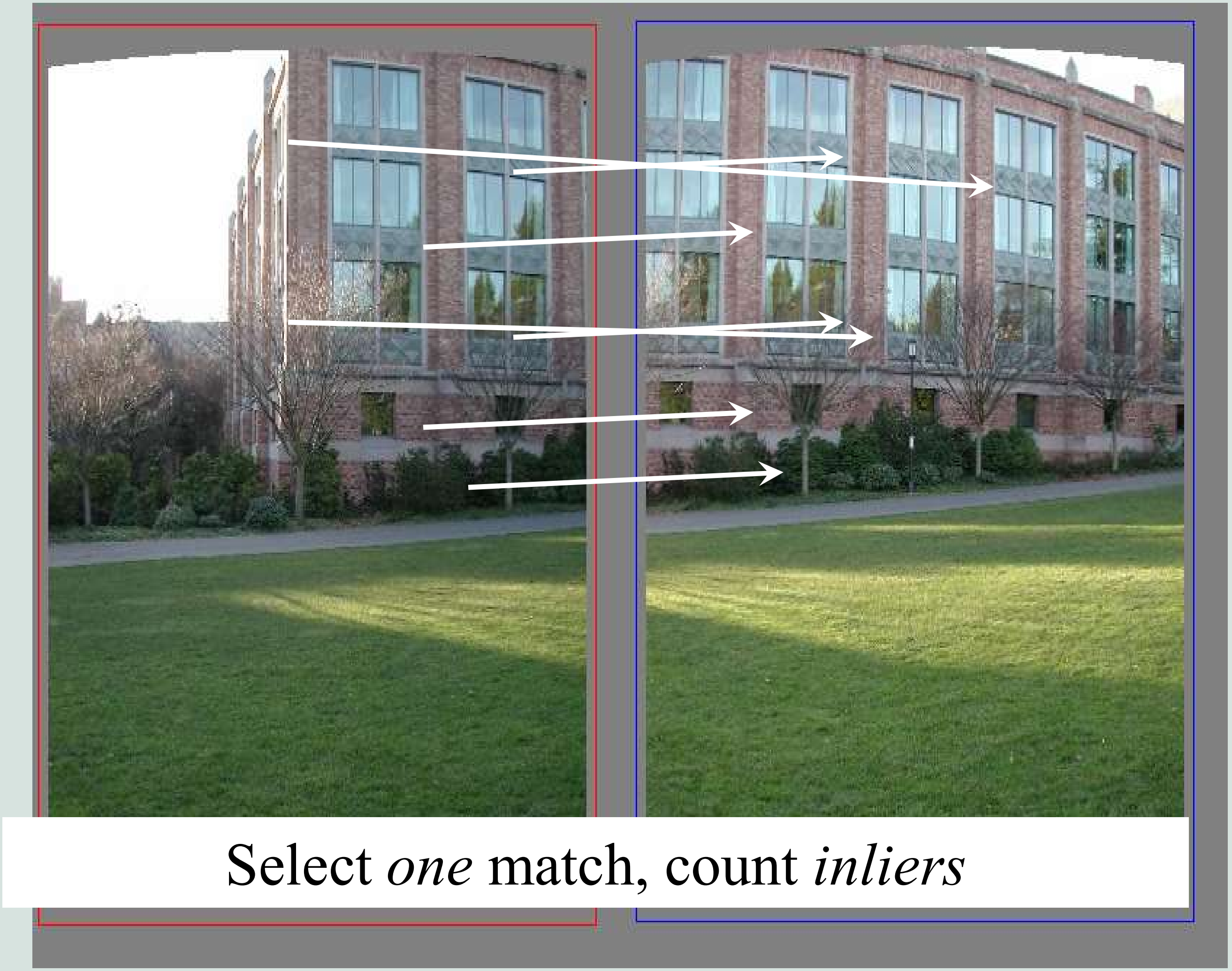
Putative matches

RANSAC Example: Translation

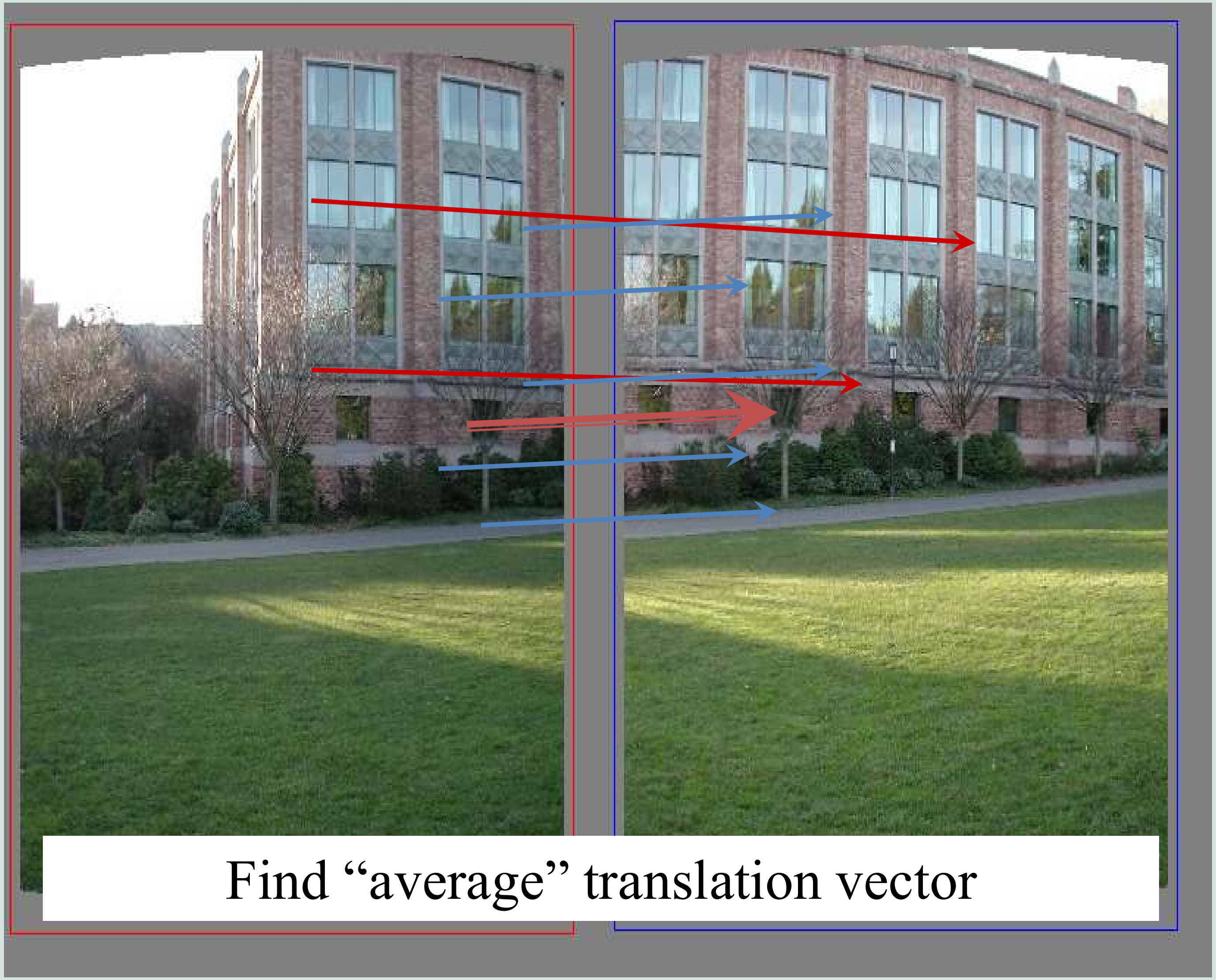


Select *one* match, count *inliers*

RANSAC Example: Translation



RANSAC example: Translation



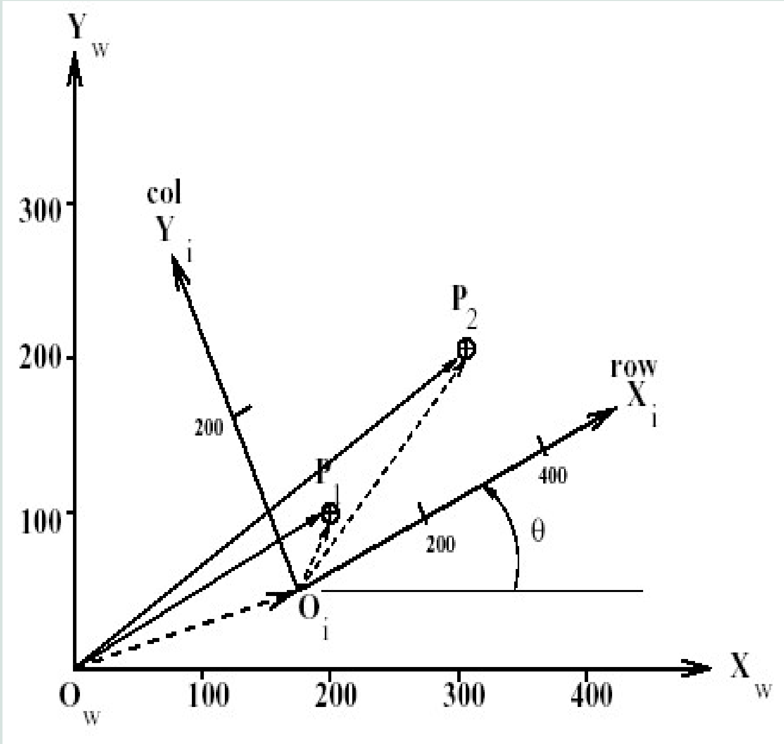
Find "average" translation vector

Exercise 1: Rotation About A Point

- Give the 3x3 matrix that represents a $\pi/2$ rotation of the plane about the point [5,8].
- Hint:
  - First derive the matrix $D_{-5,-8}$ that translates the point [5,8] to the origin of a new coordinate frame.
  - The matrix which we want will be the combination $\mathbf{D}_{5,8}\mathbf{R}_{\pi/2}\,\mathbf{D}_{-5,-8}$
- Check that your matrix correctly transforms points [5,8], [6,8] and [5,9]

Exercise 2:
Rotation, Scaling and Translation

- Given a planar workspace W[x,y]
  - An image I[r,c] is taken by a square-pixel camera looking perpendicularly down on a planar workspace W[x,y].
  - This conversion can be done by composing a rotation **R**, a scaling **S** and a translation **D** as given and denoted by ${}^{w}\mathbf{P}_j = \mathbf{D}_{x0,y0}\mathbf{S}_s\mathbf{R}_\theta {}^{i}\mathbf{P}_j$

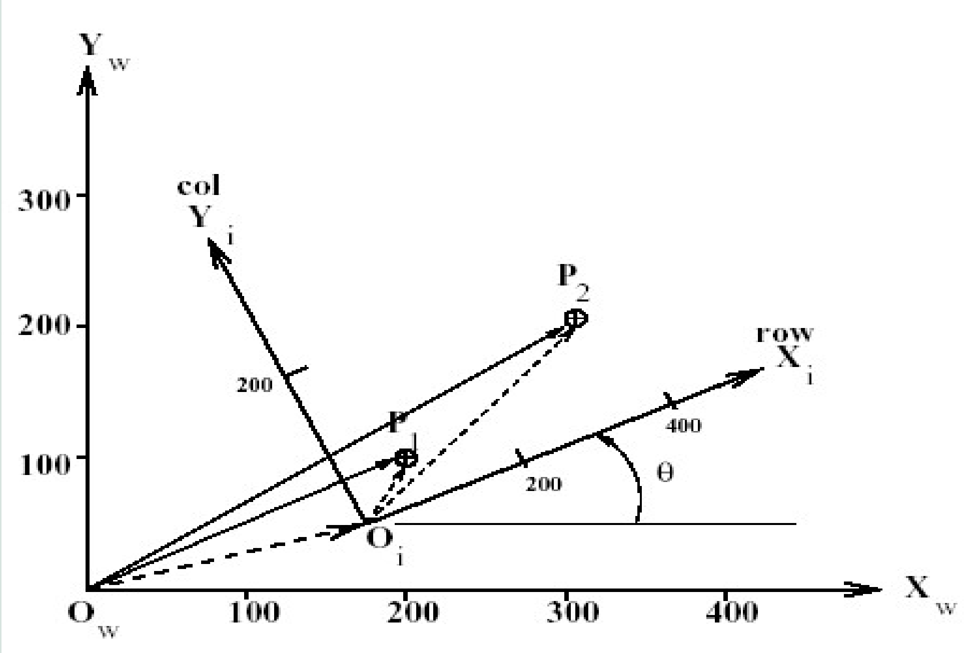


Exercise 2: Rotation, Scaling and Translation

- ${}^{w}\mathbf{P}_j = \mathbf{D}_{x0,y0}\mathbf{S}_s\mathbf{R}_\theta{}^{i}\mathbf{P}_j$

$$\begin{bmatrix} x_w \\ y_w \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$x_w = x_i s\cos\theta - y_i s\sin\theta + x_0$$
$$y_w = x_i s\sin\theta + y_i s\cos\theta + y_0$$



Exercise 2: Rotation, Scaling and Translation

- Control points are clearly distinguishable and easily measured points to establish known correspondences between different coordinate spaces.
- Given the following two matches between the world and image planes

$$^{i}P_1 = [100{,}60]\ and\ ^{w}P_1 = [200{,}100]$$

$$^{i}P_2 = [380{,}120]\ and\ ^{w}P_2 = [300{,}200]$$

- θ is easily determined independent of the other parameters as follows:

$$\theta_i = \arctan((^{i}y_2 - {}^{i}y_1)/(^{i}x_2 - {}^{i}x_1))\ \ \theta_i = \arctan(60/280) = 12.09^o$$

$$\theta_w = \arctan((^{w}y_2 - {}^{w}y_1)/(^{w}x_2 - {}^{w}x_1))\ \theta_w = \arctan(100/100) = 45^o$$

$$\theta = \ \theta_w - \ \theta_i\ \ \theta = 32.91^o$$

- Once θ is determined, all *sin* and *cos* elements are known
- There are 4 equations and 3 unknowns which can be solved for s and x_0 ,y_0.

$$200\ =\ 100.s.0.84 - 60.s.0.54 + x_0\ =\ 51.6.s + x_0$$

$$100\ =\ 100.s.0.54 - 60.s.0.84 + y_0\ =\ 3.6.s + y_0$$

$$300\ =\ 380.s.0.84 - 120.s.0.54 + x_0\ =\ 254.4.s + x_0$$

$$200\ =\ 380.s.0.54 - 120.s.0.84 + y_0\ =\ 104.4.s + y_0$$