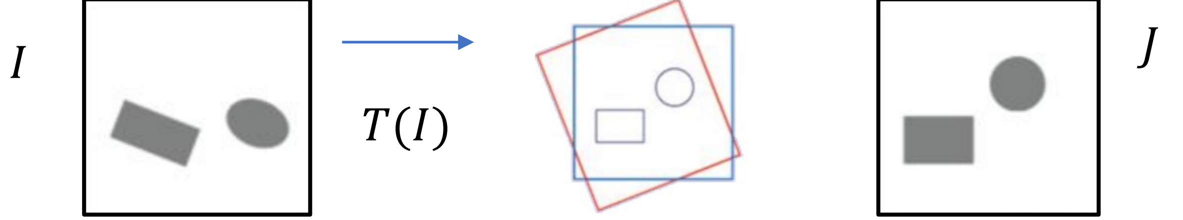
CS463/516

Lecture 9

Linear transforms



- Reminder: registration problem: $argmin_{T \in F}\{\, Similarity(\mathrm{T(I),J})\,\} = ?$
- We just covered some different $Similarity$ measures:
  - Sum-squared difference, correlation, mutual information
- Need a way of transforming the image, so we can apply $Similarity$
- let us examine T, the spatial transform that will map I onto J

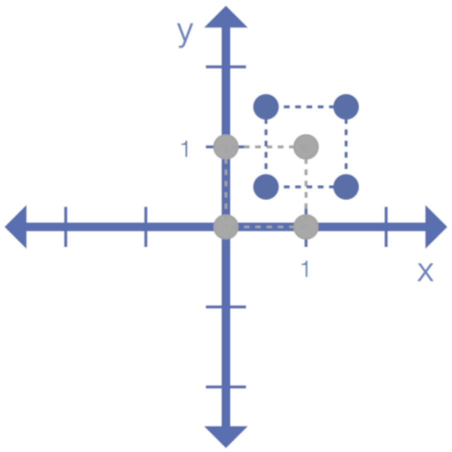
- To align two images, need to establish the *mathematical relationship* between them
- Consider two images I and J in their own separate coordinate systems.
- We have $p(x, y, 1)$ and $q(x', y', 1)$ as the *homogenous coordinates* of the pixels in the image
- The mathematical model allows us to associate a point p in I with corresponding point q in J

What transforms are we considering?

- For now we'll stick to *affine* transforms
- Translation
- Rotation
- Scaling
- Shear

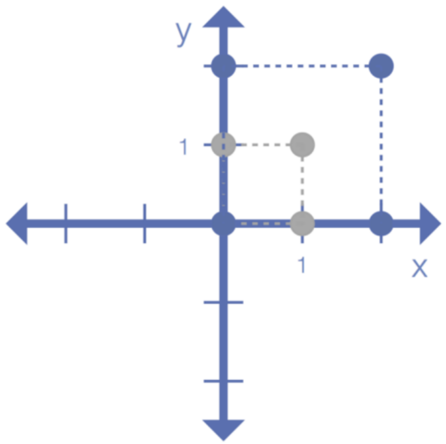
Translate

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



Scale

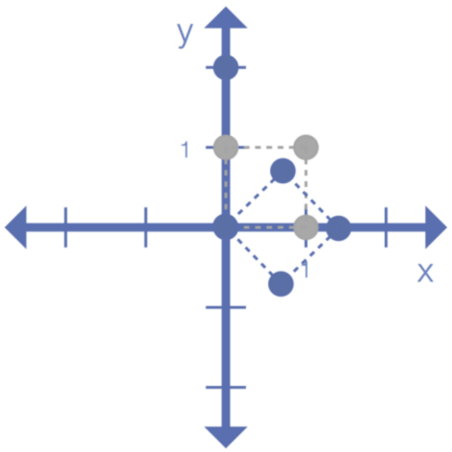
$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Rotate

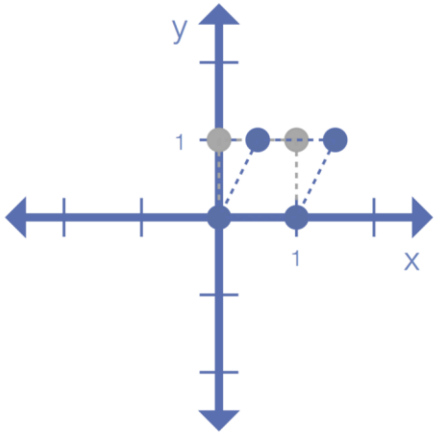
$$\begin{bmatrix} c & s & 0 \\ -s & c & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

c = s = sin(45°)



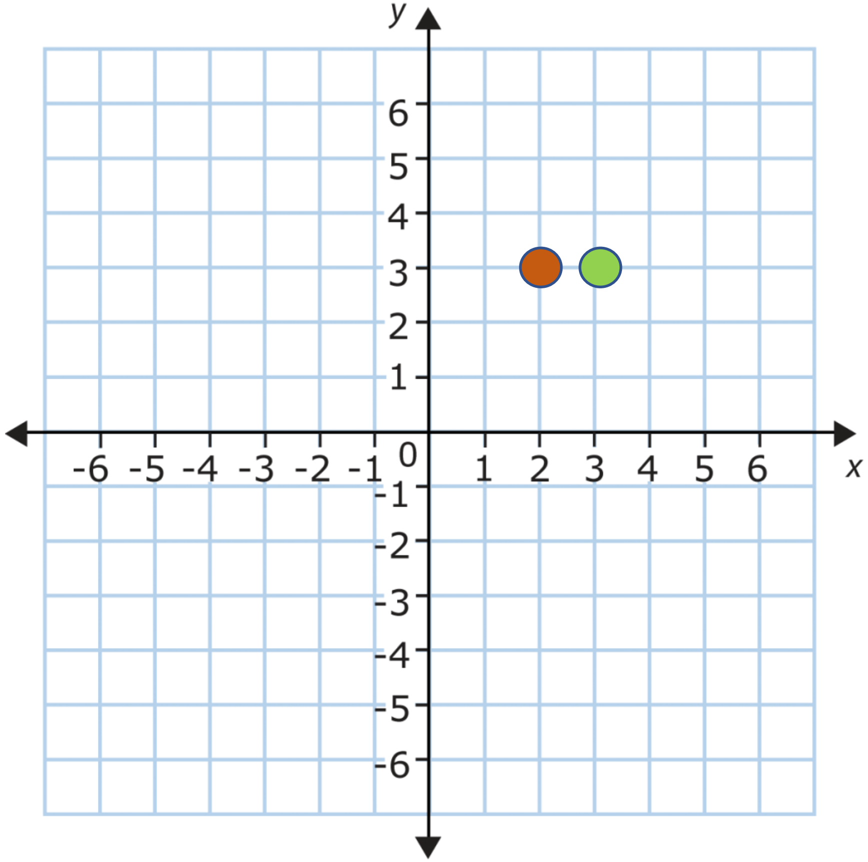
Shear

$$\begin{bmatrix} 1 & 0.5 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$



Mathematical models for transformation

- We generally assume there exists a global transformation relating each pixel p in image I to its counterpart q in image J:
- $Hp = q$
- For 2d images, H is a 3x3 matrix
- p and q are written in *homogenous coordinates:* $p = (x, y, 1)$ (for 2d case)
- Example: let $p = [2,3,1]^T$ and $H = \begin{bmatrix} 1 & 0 & u_x \\ 0 & 1 & u_y \\ 0 & 0 & 1 \end{bmatrix}$
  - Where u_x and u_y are the amounts we want to translate p in x and y dimensions
- Suppose we want to translate p by 1 in x dimension.
- Set $u_x = 1, u_y = 0$
- dot product $Hp = [2 + 0 + 1,\ 0 + 3 + 0,\ 0 + 0 + 1] = [3, 3, 1]$

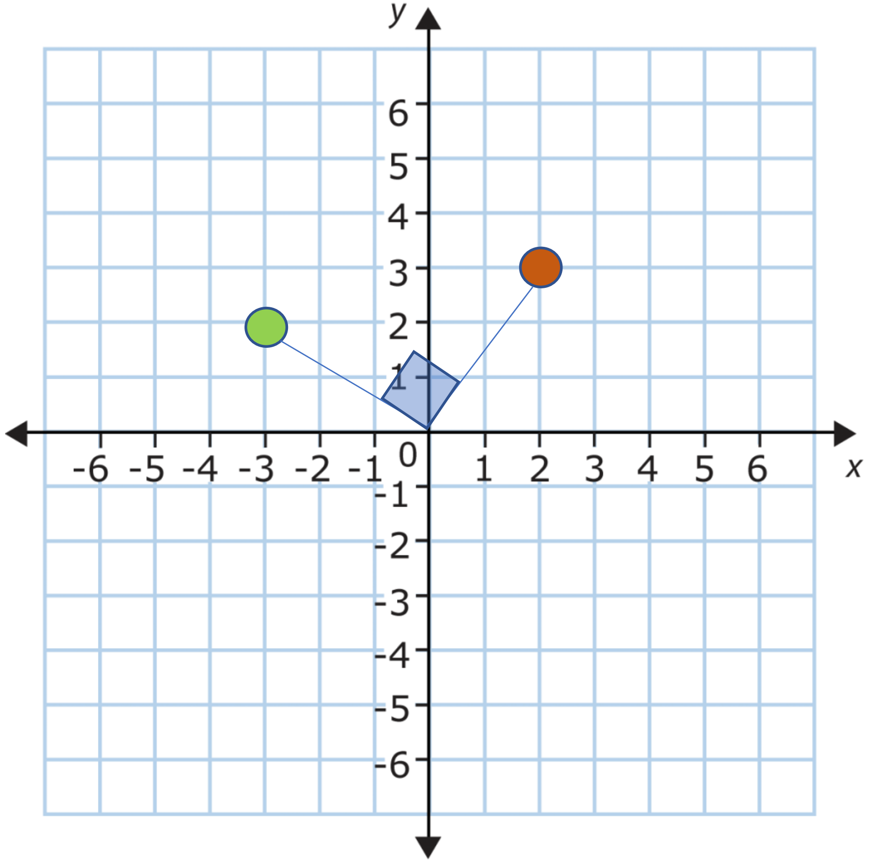


Rotation matrix

- Suppose we want to rotate $p = [2,3]$ by angle θ
- Use *rotation matrix:*
- $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$
- Again let $p = [2,3]^T$, and let $\theta = 90^\circ$ or $\pi/2$
- Then, $q = \begin{bmatrix} cos\pi/2 & -sin\pi/2 \\ sin\pi/2 & cos\pi/2 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = [-3, 2]$

```
orig_point = np.zeros([2,1])
orig_point[0] = 2;
orig_point[1] = 3
rotmat = np.zeros([2,2])
theta = np.pi/2
rotmat[0,0] = np.cos(theta)
rotmat[0,1] = -np.sin(theta)
rotmat[1,0] = np.sin(theta)
rotmat[1,1] = np.cos(theta)
new_point = np.matmul(rotmat,orig_point)
```

```
In [423]: new_point
    ...:
Out[423]:
array([[-3.],
       [ 2.]])
```



Homogenous coordinates for 2d rigid transform

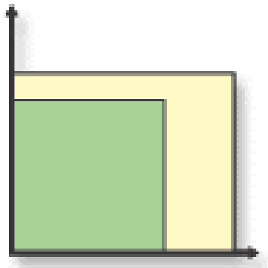
$$\begin{bmatrix} cos\theta & -sin\theta & u_x \\ sin\theta & cos\theta & u_y \\ 0 & 0 & 1 \end{bmatrix} p = q$$

3 degrees of freedom: θ, u_x, u_y - rotation, x translation, y translation

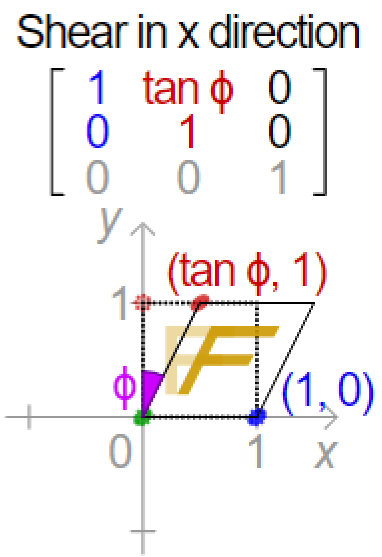
Encodes the rotation + translation in a single matrix multiplication

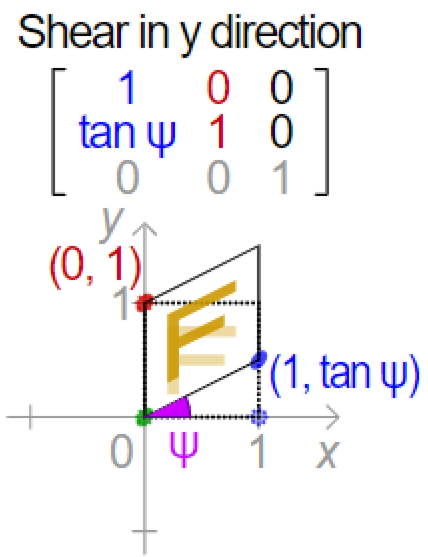
2d scaling and projective transforms

- can add a *scaling factor* d along the diagonal to scale the image
- $\begin{bmatrix} d\cos\theta & -\sin\theta & u_x \\ \sin\theta & d\cos\theta & u_y \\ 0 & 0 & 1 \end{bmatrix} p = q$
- Now, $T(\boldsymbol{x}) = dR\boldsymbol{x} + \boldsymbol{t}$
  - Where d is scale factor, R is rotation matrix, and $\boldsymbol{t}$ is translation



- Shear in x and y directions:

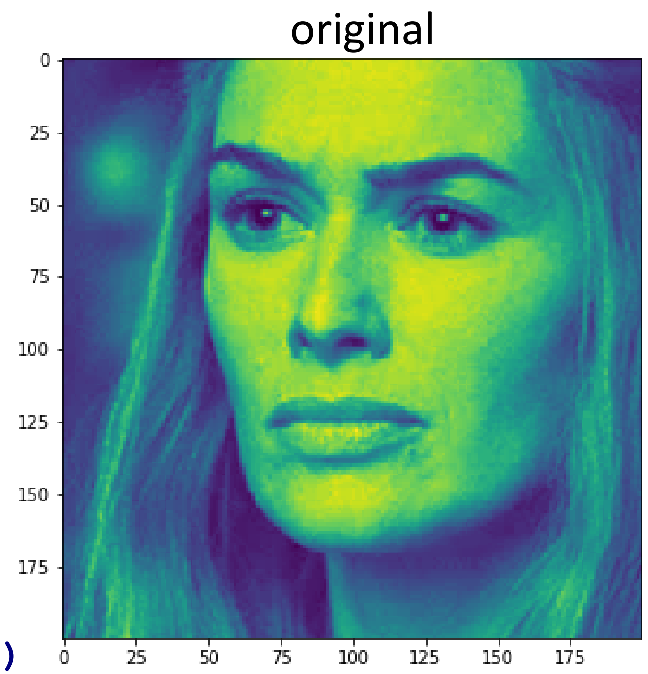


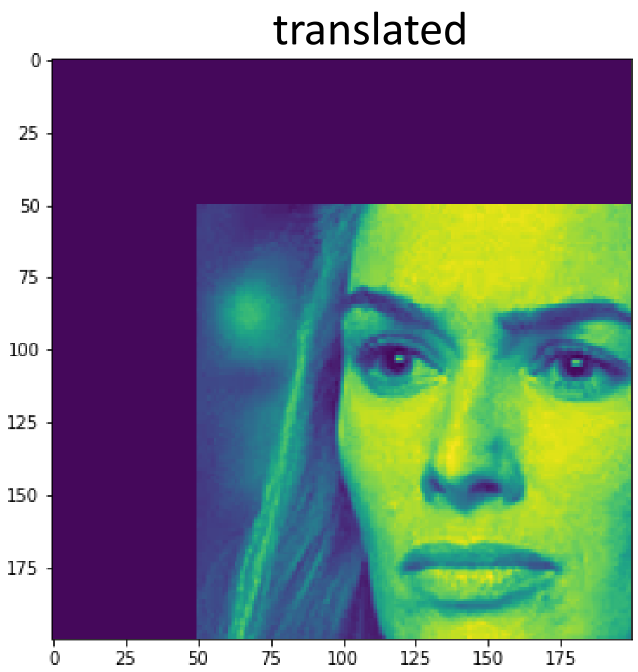


- Can *compose* an affine transform through matrix multiplication of simpler transforms
- Example, let:
  - T be a translation matrix
  - D be a scaling matrix
  - R be a rotation matrix
  - S be a shearing matrix
  - (all in homogenous coordinates)
- Then, TDRS yields an affine matrix that can be applied to a 2d point

Example: translating an image in numpy (assignment 2)

```
import numpy as np
import matplotlib.pyplot as plt
from skimage.io import imread
from scipy.interpolate import interp2d

cers =
imread('C:/shared/courses/cs516/figures/cersei.png')
cers = cers[100:300,100:300,0]

sz_x = 200
sz_y = 200

x = np.linspace(0,200,200)
y = np.linspace(0,200,200)

f = interp2d(x+50,y+50,cers,kind='cubic',fill_value=0)
znew = f(x,y)

plt.subplot(1,2,1); plt.imshow(cers)
plt.subplot(1,2,2); plt.imshow(znew)
```

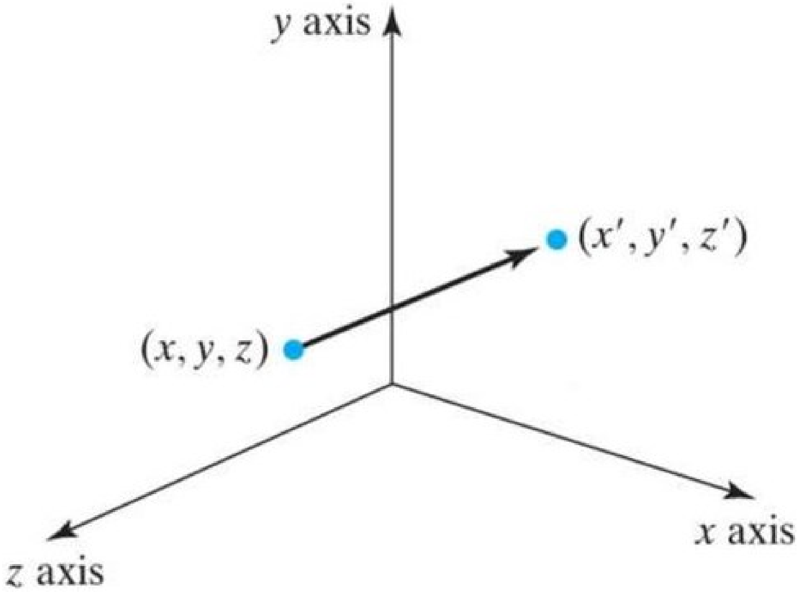




You may use interpolation functions (interp2d, or others) but *do not* use things like scipy.ndimage.shift, or scipy.ndimage.rotate
The goal of the assignment is to use matrices to transform your grid points, and then interpolate over the new grid

Homogenous coordinates for 3d transforms

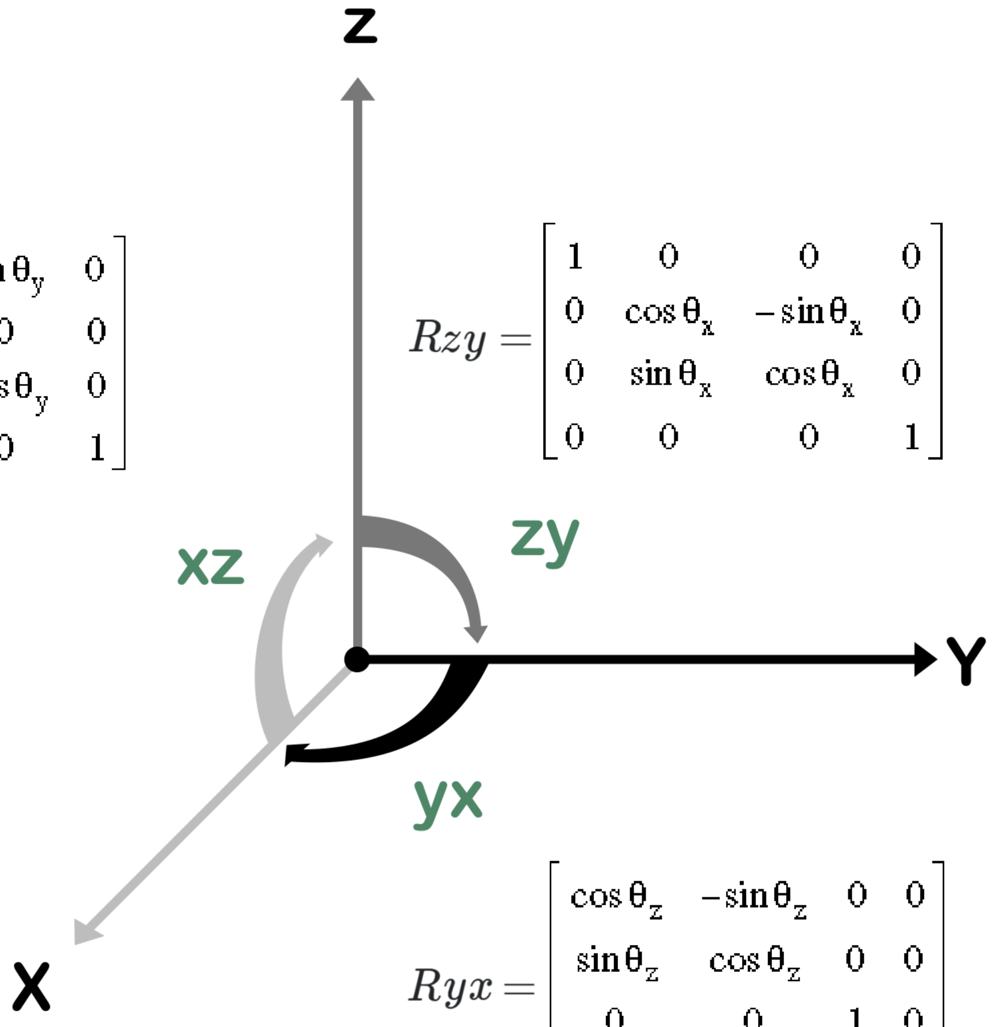
- Most medical images are 3d. Can extend the affine transform to 3d case
- Translation (a) and rotation (b) matrices



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

$$Rxz = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rzy = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x & 0 \\ 0 & \sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



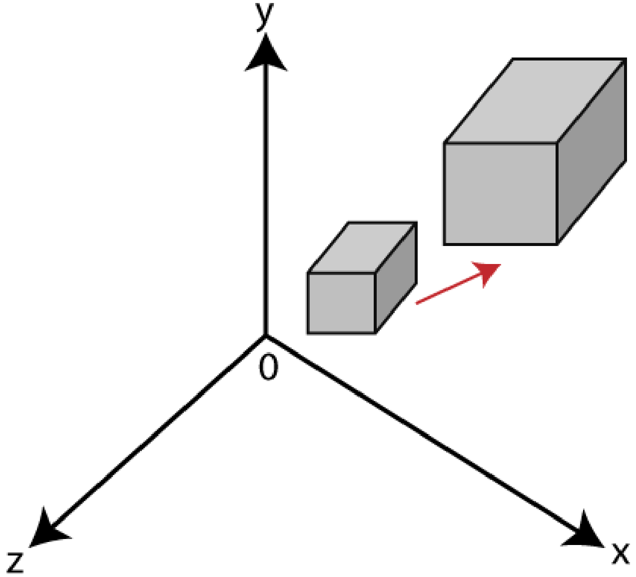
$$Ryx = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 & 0 \\ \sin\theta_z & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Homogenous coordinates for 3d transforms

- Scale (a) and shear (b) matrices
- As with 2d case, can compose multiple transforms into single matrix by multiplying the matrices together

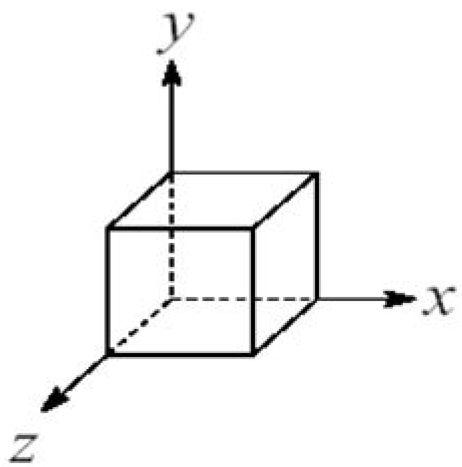
a)

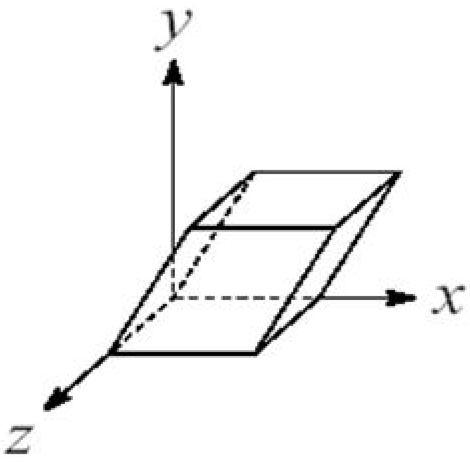
$$\begin{pmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x * sx \\ y * sy \\ z * sz \\ w \end{bmatrix}$$



b)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & b & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

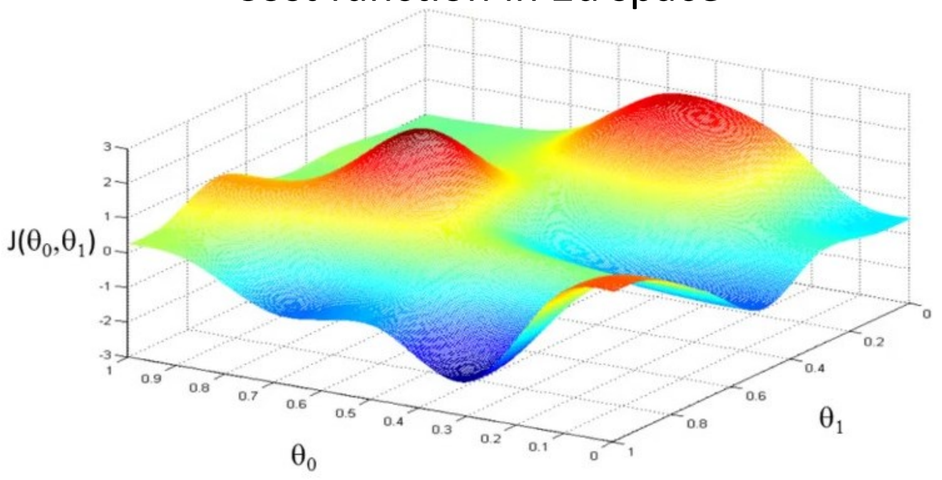




Techniques of alignment (optimization techniques)

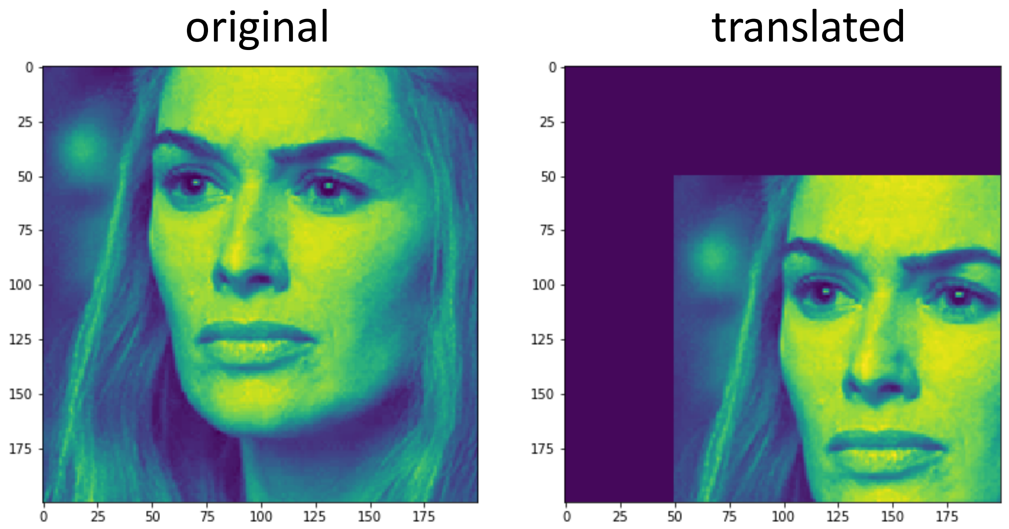
- Given a similarity criteria (mutual information, correlation, ssd) and a family of transforms (rigid or affine), how to find the transformation T such that $T(I)$ and J are aligned?
- Recall: 12 degrees of freedom (12 parameters) in 3d affine transform
  - Translation, rotation, scale, shear (all in x,y,z directions)
  - Need to find point in this 12-dimensional space that gives highest similarity between $T(I)$ and J
- Two basic approaches to alignment:
  - 1) direct alignment – base the cost function solely on the intensity values of image we are aligning
  - 2) geometric approach – segment the images, extract some primitive geometric features, match the features across images we are aligning

Cost function in 2d space



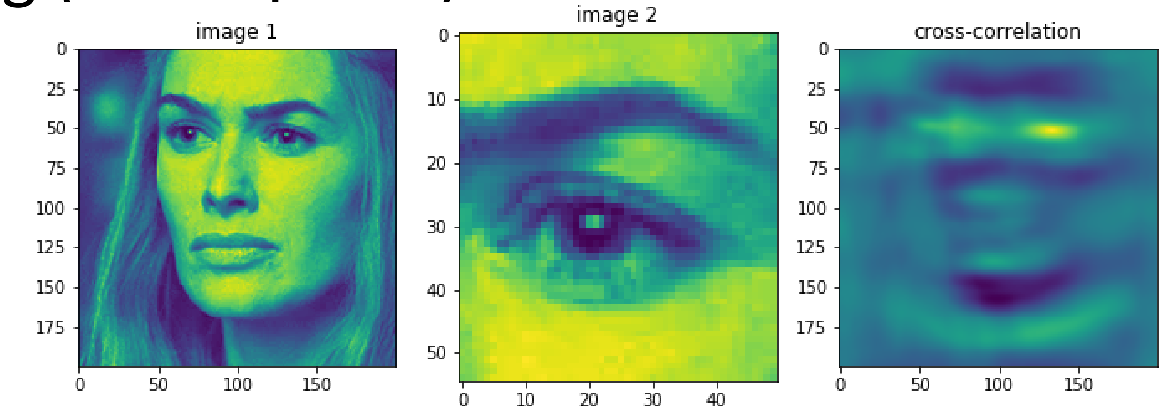
Direct alignment: translation

- Assume the only difference between I and J is a translation $\boldsymbol{u}$
- 3 basic ways to find the optimal $\boldsymbol{u}$
- 1) exhaustive search
  - Slow, precise to single pixel
- 2) FFT
  - Fast and precise to single pixel, but valid only for small $|\boldsymbol{u}|$
- 3) Lucas-Kanade
  - Moderately fast, requires that $|\boldsymbol{u}|$ is small, precise to sub-pixel



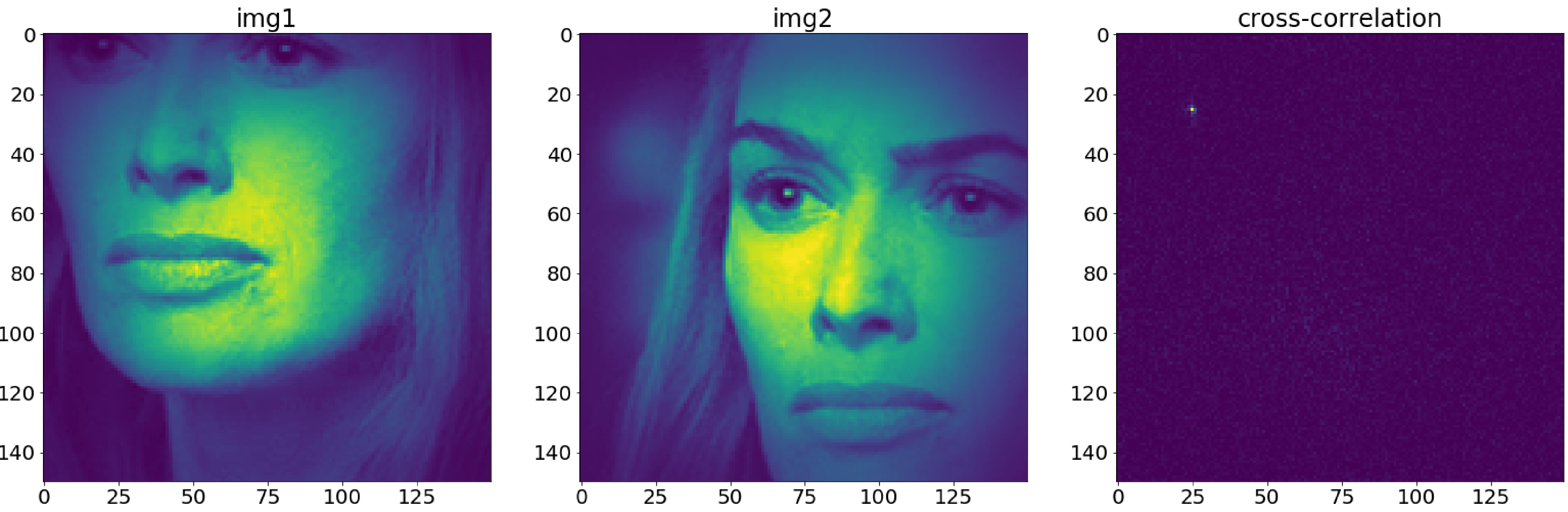
Direct alignment: translation (by FFT)

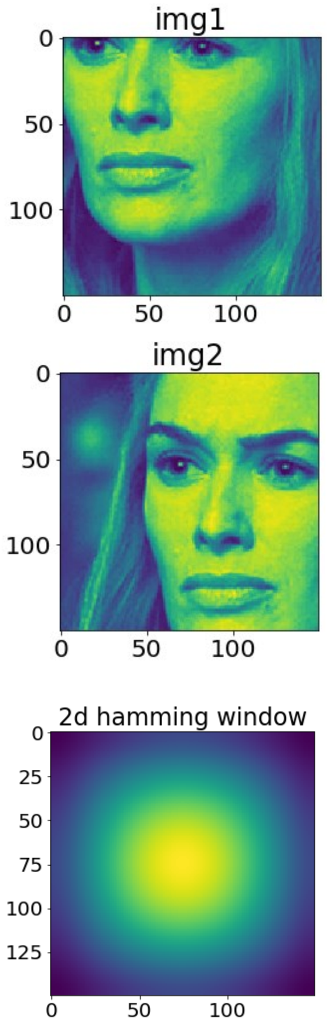
- Exhaustive search is too slow. Involves shifting the image one pixel at a time, computing similarity measure, and repeating (for all pixels)
- We can speed up the operation by taking advantage of properties of the Fourier Transform (FFT):
  - This method is known as 'phase correlation'



- Steps: given two input images g_a and g_b :
  - 1) calculate the 2d FFT of both images: $G_a = F\{g_a\}, G_b = F\{g_b\}$
  - 2) calculate cross-power spectrum by taking complex conjugate of G_b, multiplying Fourier transforms together elementwise, and normalizing this product elementwise:
    - $R = \frac{G_a \circ G_b^*}{|G_a \circ G_b^*|}$, where $\circ$ is the Hadamard (elementwise) product and G_b^* is complex conjugate of G_b
  - 3) obtain normalized cross-correlation by applying inverse Fourier: $r = F^{-1}\{R\}$
  - 4) determine location of peak in r : $(\Delta x, \Delta y) = argmax_{x,y}\{r\}$
    - Offset of peak from center gives the translation

Example: finding translation using FFT






```python
h = np.hamming(150)
ham2d = np.sqrt(np.outer(h,h))

fimg1 = fft2(img1 * ham2d)
fimg2 = fft2(img2 * ham2d)

conj2 = np.conj(fimg2)
r = (fimg1*conj2)/np.abs(fimg1*conj2)
xcorr = fftshift(np.abs(ifft2(r)))

max_pos = np.unravel_index(xcorr.argmax(), xcorr.shape)
x_translation = max_pos[0] - xcorr.shape[0]//2
y_translation = max_pos[1] - xcorr.shape[1]//2
```

```
In [81]: x_translation
Out[81]: -50

In [82]: y_translation
Out[82]: -50
```

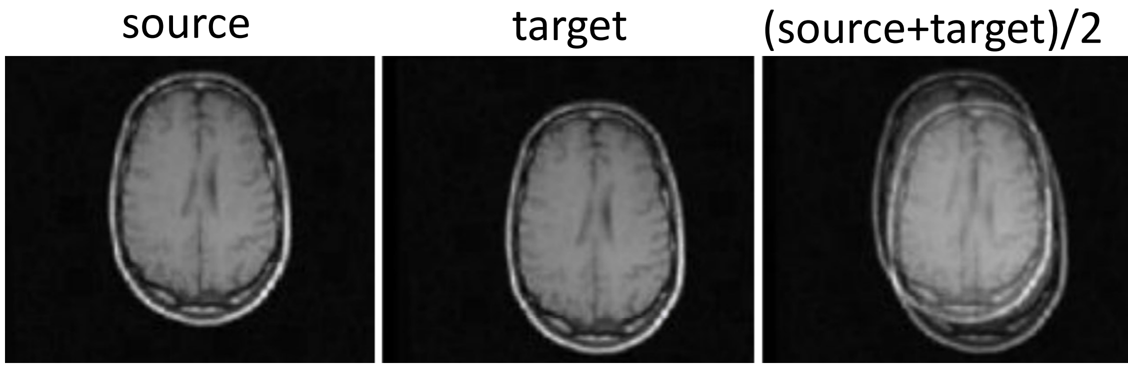
Alignment using sum-squared difference (SSD)

- Want to find translation $\boldsymbol{u} = (u_x, u_y)$ using information from all pixels in image
- For any give $\boldsymbol{u} = \boldsymbol{t} = (p, q)$, then:
- $SSD(\boldsymbol{t}) = \sum_{x,y}(I(x+p, y+q) - J(x,y))^2$. How to find $\boldsymbol{t}$ that minimizes this?
- Calculate derivatives:
- $\frac{\partial SSD}{\partial p} = 2\sum_{x,y}[(I(x+p, y+q) - J(x,y)) * \frac{\partial I}{\partial x}(x+p, y+q)]$
- $\frac{\partial SSD}{\partial q} = 2\sum_{x,y}[(I(x+p, y+q) - J(x,y)) * \frac{\partial I}{\partial y}(x+p, y+q)]$
- Fixed-step gradient descent:

$$p_{i+1} = p_i - \varepsilon\frac{\partial SSD}{\partial p}, q_{i+1} = q_i - \varepsilon\frac{\partial SSD}{\partial q}$$

Direct alignment with Lucas-Kanade

- We seek the vector $\boldsymbol{u}$ such that:
- $E_{SSD}(\boldsymbol{u}) = \sum_s (I_2(s+\boldsymbol{u}) - I_1(s))^2$ is minimized
- Can be shown that the optimal solution is:
- $\boldsymbol{u} = M^{-1}\boldsymbol{b}$
- Where:
- $M = \begin{bmatrix} \sum_s I_x^2 & \sum_s I_y I_x \\ \sum_s I_y I_x & \sum_x I_y^2 \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} -\sum_s I_x I_t \\ -\sum_s I_y I_t \end{bmatrix},$



where $I_x = \partial_x I_2(s), I_y = \partial_y I_2(s), I_t = I_2(p) - I_1(s)$

Lucas-Kanade method (analytical solution):

- Take two images I_1 and I_2
- Let I_x be the derivative of I_2 in x direction
- Let I_y be the derivative of I_2 in y direction
- Let $I_t = I_2 - I_1$

- $M = \begin{bmatrix} \sum_s I_x^2 & \sum_s I_y I_x \\ \sum_s I_y I_x & \sum_x I_y^2 \end{bmatrix}$, $\boldsymbol{b} = \begin{bmatrix} -\sum_s I_x I_t \\ -\sum_s I_y I_t \end{bmatrix}$,

- Then, $\boldsymbol{u} = -M^{-1}\boldsymbol{b}$
- Finally, translate I_2 by $\boldsymbol{u}$

Lucas-Kanade method (iterative solution)

- Can improve Lucas-Kanade using an iterative implementation:
- Let I_2 and I_1 be two images

Set $\boldsymbol{u} = (0,0)$;

$for\ i = 0\ to\ ITER\_MAX$:

$\hat{I}_2 = translate\ I_2\ by\ \boldsymbol{u}$

$I_x = derivative\ of\ \hat{I}_2\ in\ x$

$I_y = derivative\ of\ \hat{I}_2\ in\ y$

$I_t = \hat{I}_2 - I_1$

recompute M and $\boldsymbol{b}$

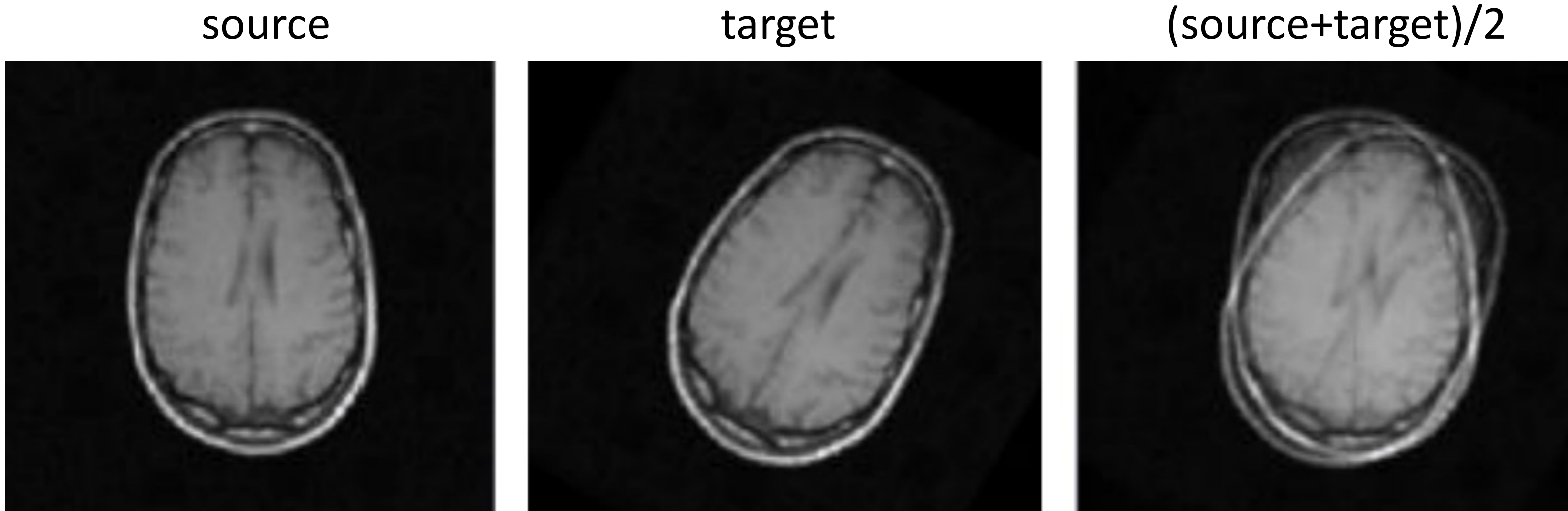
update $\boldsymbol{u} = \boldsymbol{u} - M^{-1}\boldsymbol{b}$

$$M = \begin{bmatrix} \sum_s I_x^2 & \sum_s I_y I_x \\ \sum_s I_y I_x & \sum_x I_y^2 \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} -\sum_s I_x I_t \\ -\sum_s I_y I_t \end{bmatrix},$$

Finally, translate I_2 by $\boldsymbol{u}$

Adding rotation to SSD method

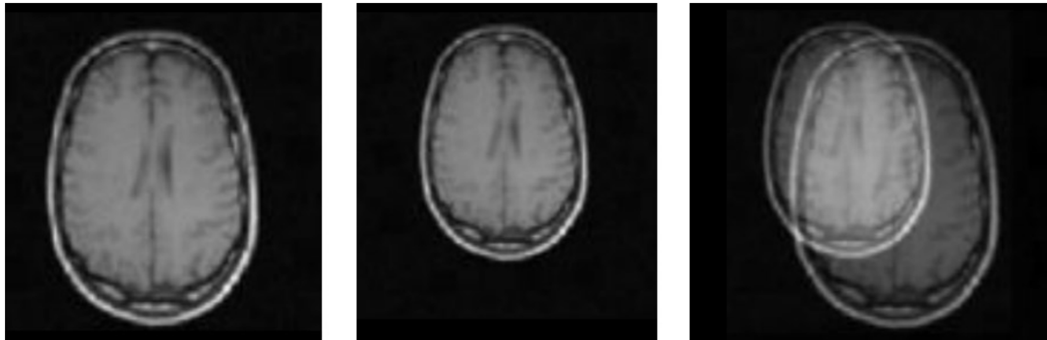
- Up to now, we only focused on solving the translation (find vector $\boldsymbol{u}$)
- To add rotation, we search the theta that minimizes:
- $SSD(\theta) = \sum_{x,y}(I(xcos\theta - ysin\theta, xsin\theta + ycos\theta) - J(x,y))^2$
- How to minimize? Calculate derivatives:
- $\frac{\partial SSD}{\partial \theta} = 2\sum_{x,y}\big(I(xcos\theta - ysin\theta, xsin\theta + ycos\theta) - J(x,y)\big) * (\frac{\partial I}{\partial x} * (-xsin\theta - ycos\theta) + \frac{\partial I}{\partial y} * (xcos\theta - ysin\theta))$
- Fixed step gradient descent:
- $\theta_{i+1} = \theta_i - \varepsilon \frac{\partial SSD}{\partial \theta}$



2d registration using rigid transform

- Similarity criteria:
- $SSD(\theta, p, q) = \sum_{x,y}(I(x cos\theta - y sin\theta + p,\ x sin\theta + y cos\theta + q)\ - J(x, y))^2$
- Fixed step gradient descent:
- $\theta_{i+1} = \theta_i - \varepsilon \frac{\partial SSD}{\partial \theta}, p_{i+1} = p_i - \varepsilon \frac{\partial SSD}{\partial p}, q_{i+1} = q_i - \varepsilon \frac{\partial SSD}{\partial q}$
- **Add scaling as well:**
- $SSD(\theta, p, q) = \sum_{x,y}(I(sx, sy) - J(x, y))^2$
- Derivative:
- $\frac{\partial SSD}{\partial s} = 2 \sum_{x,y}(I(sx, sy) - J(x, y)(x \frac{\partial I}{\partial x} + y \frac{\partial I}{\partial y})$
- Gradient descent: $s_{i+1} = s_i - \varepsilon \frac{\partial SSD}{\partial s}$

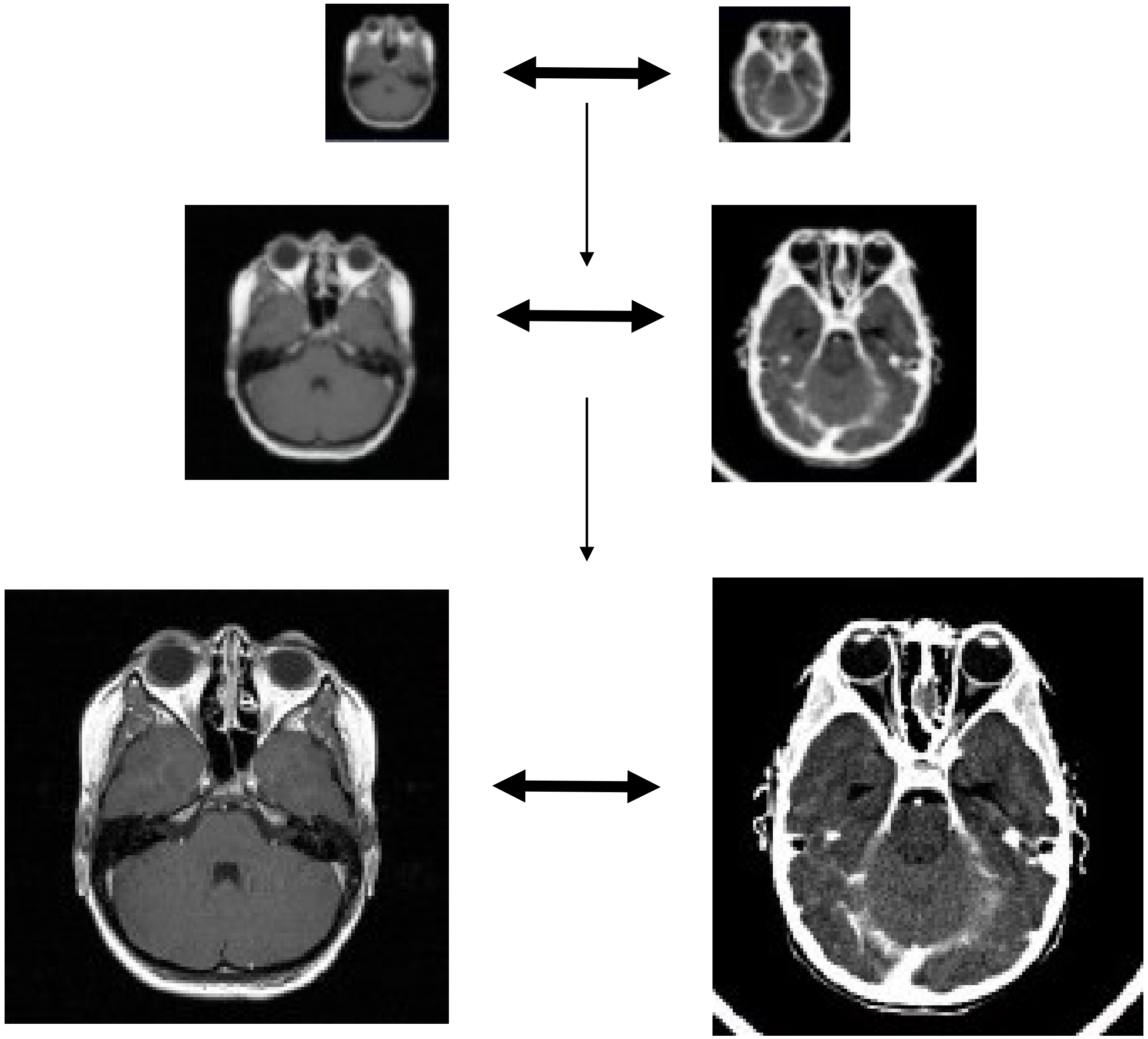
source target (source+target)/2



Traditional approach summary

- Use gradient descent algorithms to minimize similarity criteria based on image intensity
- In practice, simple gradient descent seen here is rarely used
- Cannot always calculate gradient (depending on what similarity criteria we use)
  - Cannot calculate gradient of mutual information based on joint histogram, for example
- Approaches without using gradient:
  - Simplex method, Powell method
  - 0 order methods
- These methods ideal if we have more degrees of freedom (12, for example)
  - They also converge more easily to a global minimum

Multiresolution methods



Use transformation derived from down-sampled datasets to initialize registration algorithm at higher resolutions (saves time)