

BLG453E COMPUTER VISION

Fall 2018 Term

Week 3



Istanbul Technical University
Computer Engineering Department

Instructor: Prof. Gözde ÜNAL

Teaching Assistant: Enes ALBAY

Learning Outcomes of the Course

Students will be able to:

1. Discuss the main problems of computer (artificial) vision, its uses and applications
2. Design and implement various image transforms: point-wise transforms, neighborhood operation-based spatial filters, and geometric transforms over images
3. Define and construct segmentation, feature extraction, and visual motion estimation algorithms to extract relevant information from images
4. Construct least squares solutions to problems in computer vision
5. Describe the idea behind dimensionality reduction and how it is used in data processing
6. Apply object and shape recognition approaches to problems in computer vision



Week 3: LOs: Geometric Image transforms

At the end of Week 3: Students will be able to:

2. Design and implement various image transforms:
 geometric transforms over images, point-wise transforms, neighborhood
 operation-based spatial filters

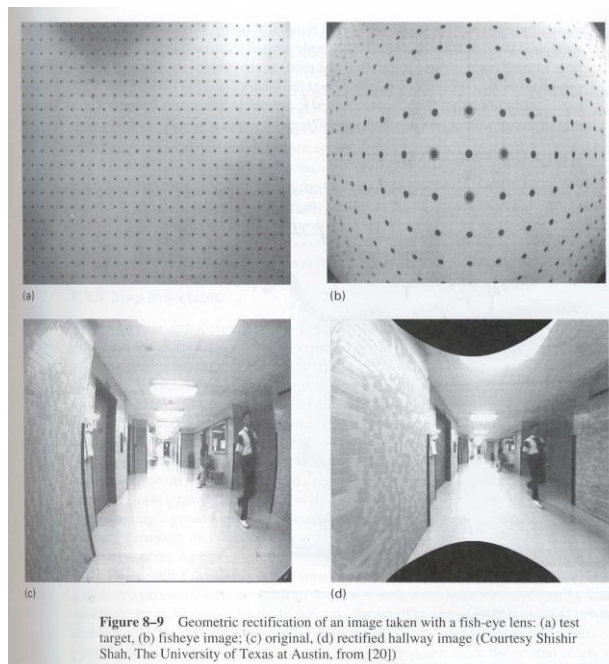
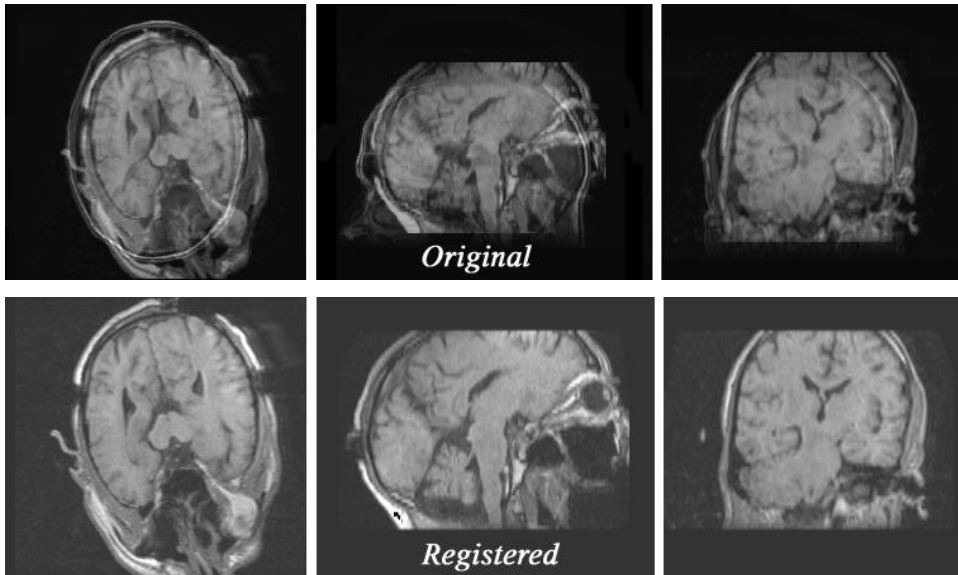
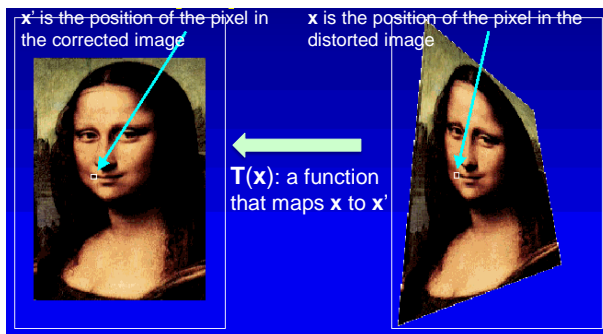


Image Registration Problem



Talos I-F., Archip N. Volumetric Non-Rigid Registration for MRI-guided Brain Tumor Surgery. SPL 2007 Aug

Geometric transformations



In a geometric transformation, the positions of pixels in the image is transformed. Mathematically, this is expressed (in a general form) as

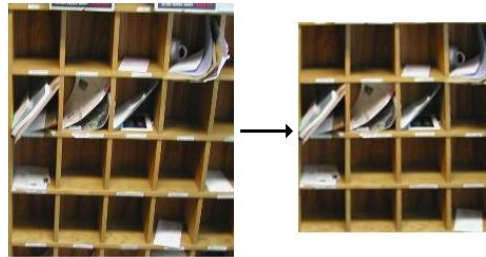
$$\mathbf{x}' = \mathbf{T}(\mathbf{x})$$

Picture: Courtesy Prof. M. Milanova, University of Arkansas at Little Rock

Geometric transformations

Geometric transformations change the spatial position of pixels in the image. They are also known as *image warps*. Geometric transformations have a variety of practical uses, including

- Registration: bringing multiple images into the same coordinate system
- Removing distortion
- Simplifying further processing



Distorted image

Corrected image

Slide: Prof. G. Slabaugh, City U. London

Computing an Image Warp

An image warp is normally implemented as follows: for every pixel position x,y in the destination image:

Using T^{-1} , determine (u,v) , where (x,y) came from in the source image

Interpolate a value from source image $I(u,v)$ to produce destination image $J(x,y)$

end

Mapping definition:

$$u = U(x,y)$$

$$v = V(x,y)$$

Warping algorithm:

```

for y = ymin to ymax
  for x = xmin to xmax
    u = U(x,y)
    v = V(x,y)
    copy pixel at source (u,v)
    to destination (x,y)
  
```

Slide: Prof. M. Milanova, University of Arkansas at Little Rock

Image Warping (Geometric Transformation)

- Move pixels of image by
 1. Mapping
 2. Resampling



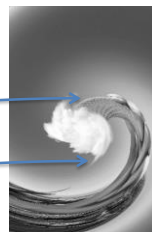
Source Image



Destination Image

Coordinate Mapping

1. Define transformation
 - Describe the destination (x, y) for every location (u, v) in the source image (or vice-versa, if invertible transformation)



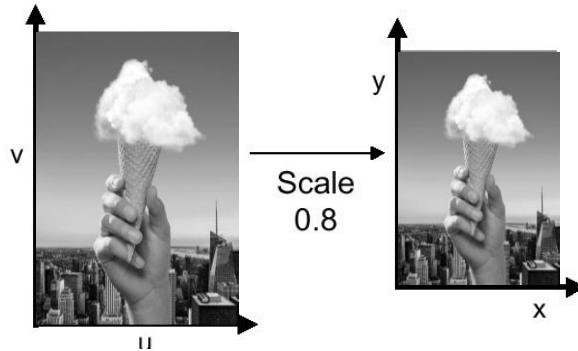
```
import cv2 # OpenCV library is imported to program
lm = cv2.imread("sample1.jpg", -1)
[h, w, dim] = lm.shape
centerx = w // 2
centery = h // 2
for j in range(0, h):
    for k in range(0, w):
        angle = rotation_amount * np.exp((-1 * ((k - centerx)**2 + (j - centery)**2)) / effect**2)
        rot_mat = np.asarray([[np.cos(angle), np.sin(angle)], [-1 * np.sin(angle), np.cos(angle)]])
        coord = np.matmul(rot_mat, np.asarray([k - centerx, j - centery]))
        coord += center_of_image
```

Example Coordinate Mapping: Scaling

- Scale by factor

$$x = \text{factor} * u$$

$$y = \text{factor} * v$$



```

Im = cv2.imread("sample1.jpg", -1)
[h, w, dim] = Im.shape
angle = np.pi/6
for j in range(0, h):
    for k in range(0, w):
        scale_mat = np.eye(2) * 0.8
        scale_mat = np.linalg.inv(scale_mat)
        coord = np.matmul(scale_mat, np.asarray([k - centerx, j - centery]))
        coord += center_of_image

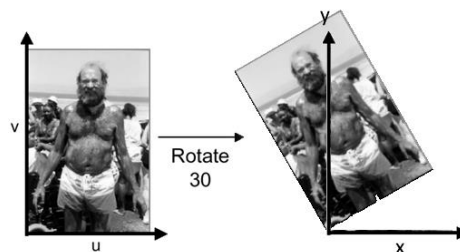
```

Example Coordinate Mapping

- Rotate by θ degrees

$$x = u \cos \theta - v \sin \theta$$

$$y = u \sin \theta + v \cos \theta$$



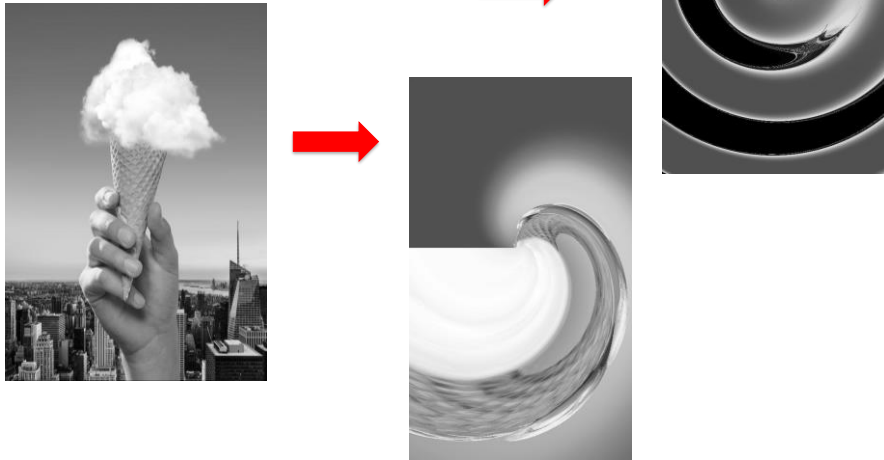
```

Im = cv2.imread("sample1.jpg", -1)
[h, w, dim] = Im.shape
angle = np.pi/6
# center of rotation
centerx = w // 2
centery = h // 2
for j in range(0, h):
    for k in range(0, w):
        rot_mat = np.asarray([[np.cos(angle), np.sin(angle)], [-1 * np.sin(angle), np.cos(angle)]])
        coord = np.matmul(rot_mat, np.asarray([k - centerx, j - centery]))
        coord += center_of_image

```

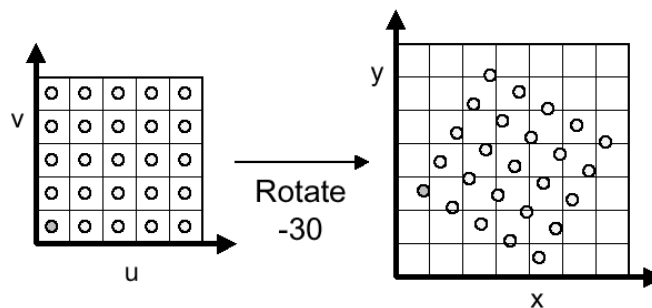
A general Coordinate Mapping

- Any function of u and v
 $x = f_x(u, v)$
 $y = f_y(u, v)$



Geometric Transform: Forward Mapping

- Iterate over source image

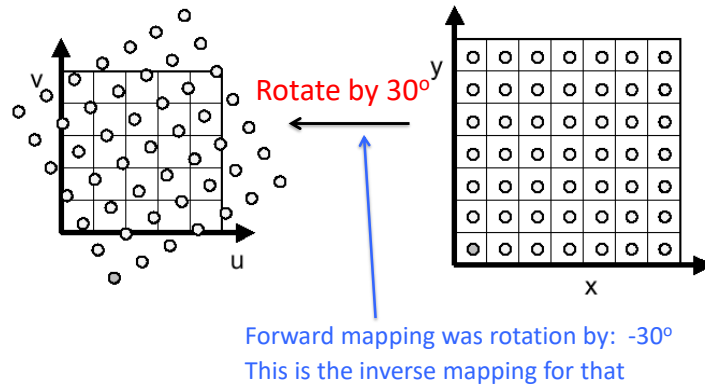


Some destination pixels may not be covered
 Many source pixels can map to the same destination pixel

Backward Mapping

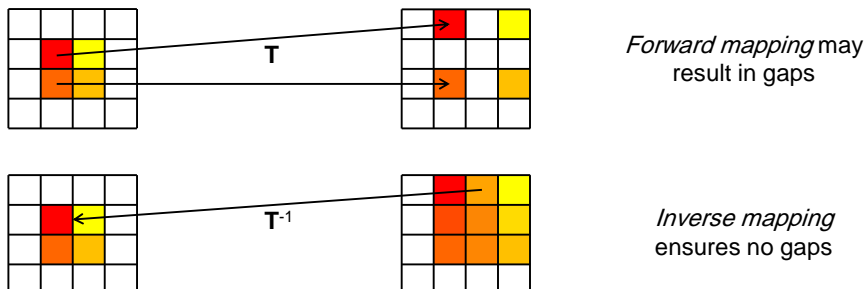
Iterate over destination image to map it by the inverse (backward) transformation

- We must resample source image



Geometric transformations

We will prefer *backwards*, rather than using a (forward) mapping T to transform pixels from the distorted image to the corrected image, we use an (inverse) transform T^{-1} .



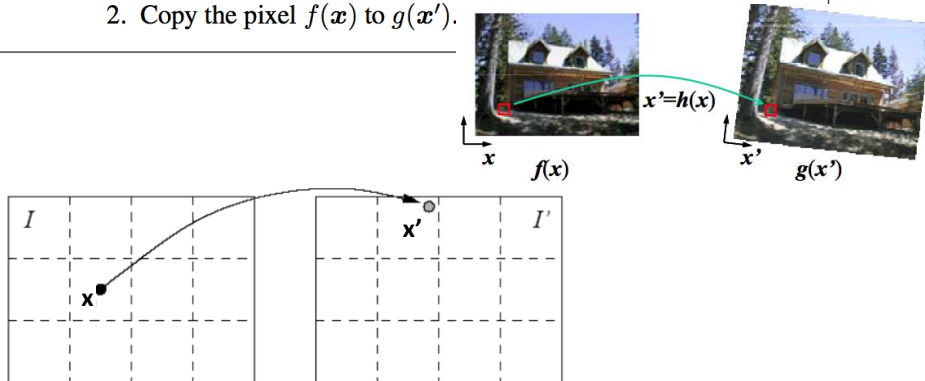
⇒ Using an inverse mapping ensures all the pixels in the corrected image will be filled. However, it's necessary to *interpolate* pixels from the distorted image.

Forward Warp

procedure *forwardWarp*(f, h , out g):

For every pixel x in $f(x)$ (traverse source image pixels)

1. Compute the destination location $x' = h(x)$.
2. Copy the pixel $f(x)$ to $g(x')$.



R. Szeliski, Computer Vision Book, 2010

Backward Warp

procedure *inverseWarp*(f, h , out g):

For every pixel x' in $g(x')$ (traverse output pixels)

1. Compute the source location $x = \hat{h}(x')$
2. Resample $f(x)$ at location x and copy to $g(x')$

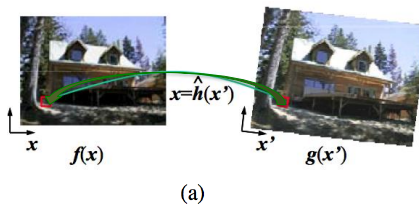
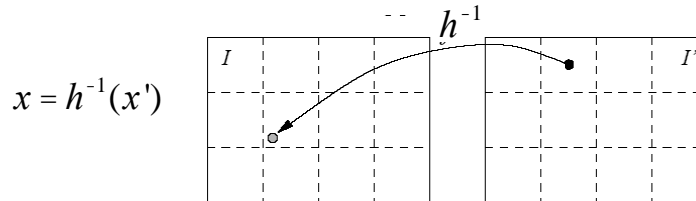


Figure 3.47 Inverse warping algorithm: (a) a pixel $g(x')$ is sampled from its corresponding location $x = \hat{h}(x')$ in image $f(x)$; (b) detail of the source and destination pixel locations.

R. Szeliski, Computer Vision Book, 2010

Backward Mapping

The inverse transform maps an integer-coordinate point (x', y') in I' into a real coordinate point (x, y) in I



Use the colors of neighboring integer coordinate points in I to estimate $I(p)$ (e.g. use bilinear interpolation: we will learn in the coming slides)

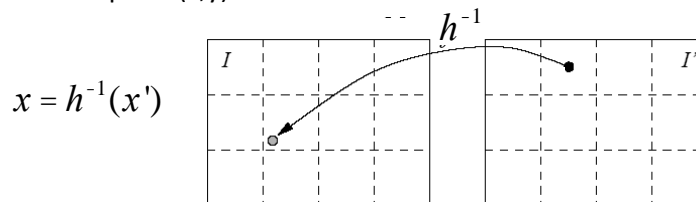
Then: $I'(x', y') = I(x, y)$

Equivalently: $I'(x', y') = I(h^{-1}(x', y')) = I(x, y)$

Advantage: No round-off error

Backward Mapping

The inverse transform maps an integer-coordinate point (x', y') in I' into a real coordinate point (x, y) in I

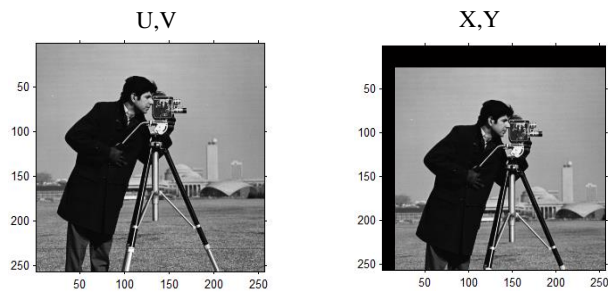


```

Im = cv2.imread("sample1.jpg", -1)
[h, w, dim] = Im.shape
angle = np.pi/6
for j in range(0, h):
    for k in range(0, w):
        rot_mat = np.asarray([[np.cos(angle), np.sin(angle)], [-1 * np.sin(angle), np.cos(angle)]])
        inv_rot_mat = np.linalg.inv(rot_mat)
        coord = np.matmul(inv_rot_mat, np.asarray([k - centerx, j - centery]))
        coord += center_of_image

```

THQ



$$U = X - 20; \quad V = Y - 30$$

$$X = U + 20; \quad Y = V + 30$$

Next: PARAMETRIC GEOMETRIC TRANSFORMS

TRANSLATION

$$U = X + X_0$$

$$V = Y + Y_0$$

Homogeneous coordinates

$$\left\{ \begin{bmatrix} U \\ V \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & X_0 \\ 0 & 1 & Y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \right.$$

Using Homogeneous coordinates makes it possible for these geometric transforms (e.g. translation or affine) to be represented as matrix vector multiplication, i.e. a linear transformation

TRANSLATION

In Homogeneous coordinates: $x = \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$ $u = \begin{bmatrix} U \\ V \\ 1 \end{bmatrix}$

$$u = Tx$$

```
height, width=l.shape
i, j = np.meshgrid(range(height), range(width), indexing='ij')
i = i.reshape((1, -1))
j = j.reshape((1, -1))
onesCol = np.ones((1, i.shape[1]))
coords = np.concatenate((i, j, onesCol), axis=0)
X0 = 10
Y0 = 20
new_coords = np.matmul(np.asarray([[1, 0, X0],
                                   [0, 1, Y0],
                                   [0, 0, 1]]), coords)
```

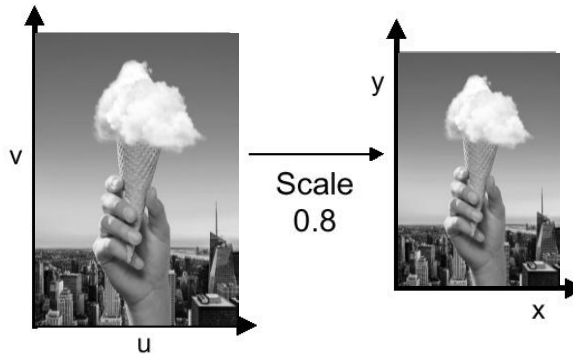
$$T = \begin{bmatrix} 1 & 0 & X_0 \\ 0 & 1 & Y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

SCALING

$$U = 1/s * X$$

$$V = 1/s * Y$$



e.g. $s = 0.8$

$$S = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

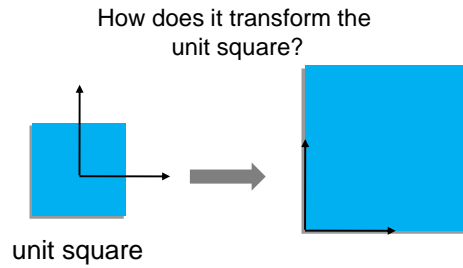
```
new_coords = np.matmul(np.asarray([[0.8, 0, 0],
                                   [0, 0.8, 0],
                                   [0, 0, 1]]), coords)
```

more general

Question

Consider the transformation

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



ROTATION

$$R_\theta = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Q: How many parameters ?

$$q = p / 4$$



Need to specify the Center of Rotation. Above example: Center of rotation=Image Center

2D Rotation requires 1 rotation parameter since it is only in the image plane

Let

$$u = a(x, y) = x \cos(q) - y \sin(q)$$

$$v = b(x, y) = x \sin(q) + y \cos(q)$$

produces a clockwise rotation through an angle q about the origin.

In homogeneous coordinates:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

```
new_coords = np.matmul(np.asarray([[np.cos(angle), -1 * np.sin(angle), 0],
                                   [np.sin(angle), np.cos(angle), 0],
                                   [0, 0, 1]]), coords)
```

How to compute INVERSE TRANSFORMATION?

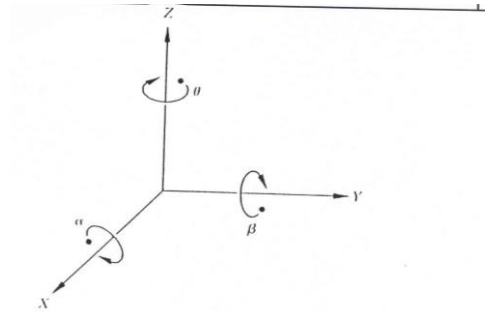
Compute inverse of the transformation matrix:

$$T^{-1} = \begin{bmatrix} 1 & 0 & -X_0 \\ 0 & 1 & -Y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

$$S^{-1} = \begin{bmatrix} 1/S_x & 0 & 0 \\ 0 & 1/S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3D ROTATION



Rotation of a 3D point about each of the coordinate axes. Angles are measured clockwise when looking along the rotation axis toward the origin.

$$R_\alpha = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & \sin \alpha & 0 \\ 0 & -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} R_\beta = \begin{bmatrix} \cos \beta & 0 & -\sin \beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin \beta & 0 & \cos \beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} R_\theta = \begin{bmatrix} \cos \theta & \sin \theta & 0 & 0 \\ -\sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

AFFINE TRANSFORM

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ 0 & 0 & 1 \end{bmatrix} \left. \vphantom{\begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ 0 & 0 & 1 \end{bmatrix}} \right\} \leftarrow \text{with zero translation}$$



Affine transformation

An affine transformation takes the form $\mathbf{x}' = A\mathbf{x}$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y + a_{13} \\ a_{21}x + a_{22}y + a_{23} \\ 1 \end{bmatrix}$$

In OpenCV, you apply the transformation to an image using `cv2.warpAffine`

Slide: Prof. G. Slabaugh, City U. London

Special cases

There are several common special cases, including

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

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

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \\ 1 \end{bmatrix} \quad \text{Scaling}$$

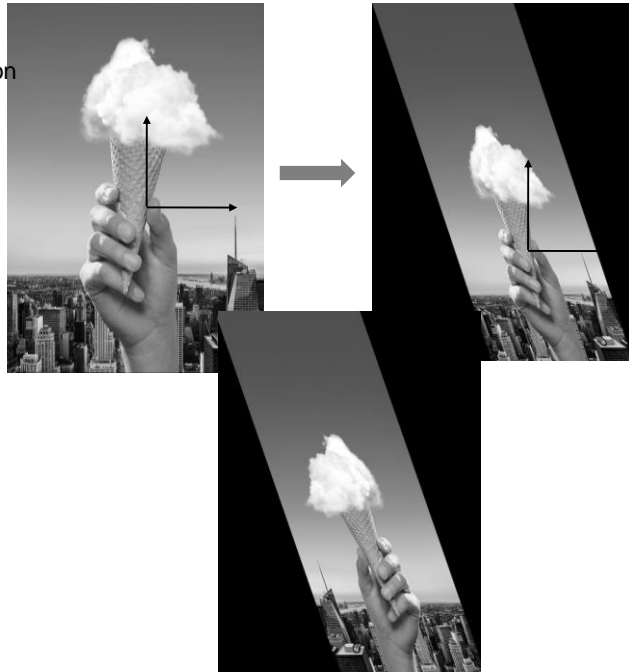
Slide: Prof. G. Slabaugh, City U. London

Question

Consider the transformation

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

How does it transform the image?



What about ?

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

Affine transformation

Another special case includes skew

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & s & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + sy \\ y \\ 1 \end{bmatrix}$$



s=0.5

Example

```

l=cv2.imread("LicensePlate.png") #read image
cv2.imshow("License Plate Original", l)

height,width=l.shape[:2]

# Rotate clockwise 15 degrees to align base
M= cv2.getRotationMatrix2D((width/2,height/2),-15,1)
J1 = cv2.warpAffine(l,M,(width,height))
cv2.imshow("License Plate Rotated", J1)

# Now apply a skew
tform=np.float32([[ 1,0.3,0],[0,1,0]])
height,width=J1.shape[:2]
J2=np.zeros([h,w], dtype=np.uint8)
J2 = cv2.warpAffine(J1,tform,(width,height))
cv2.imshow("License Plate Skewed", J2)

cv2.waitKey(0)

```

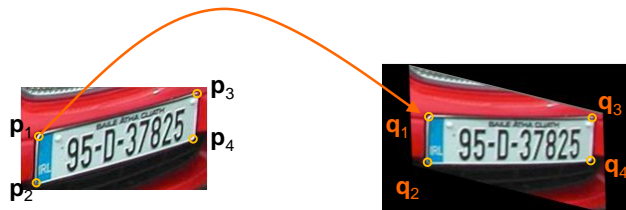


Slide: Prof. G. Slabaugh, City U. London

Estimation of Affine Transform through correspondences

If the affine transformation is not known in advance, it can be estimated.

For example, we could say \mathbf{p}_1 corresponds to \mathbf{q}_1 . What we would like to do is estimate the affine transformation that best aligns the correspondences.

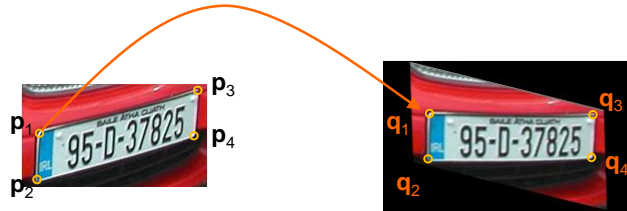


Estimation here means: to determine the six coefficients in the A matrix.

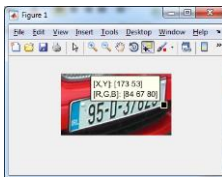
Q: At least how many point correspondences do you need ?

This can be achieved by finding at least **three correspondences**, or matching points.

Estimation of Affine Transform through correspondences



Say you pick 4 point correspondences:



$$\begin{aligned} \mathbf{p}_1 &= [18, 47]^T \\ \mathbf{p}_2 &= [15, 100]^T \\ \mathbf{p}_3 &= [178, 6]^T \\ \mathbf{p}_4 &= [173, 53]^T \end{aligned}$$

$$\begin{aligned} \mathbf{q}_1 &= [48, 50]^T \\ \mathbf{q}_2 &= [48, 100]^T \\ \mathbf{q}_3 &= [212, 50]^T \\ \mathbf{q}_4 &= [212, 100]^T \end{aligned}$$

Next, set up the equations

Slide: Prof. G. Slabaugh, City U. London

Estimation through correspondences

Noting that

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y + a_{13} \\ a_{21}x + a_{22}y + a_{23} \\ 1 \end{bmatrix}$$

If we have three correspondences we can write

$$\begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ x_3 & y_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \end{bmatrix}$$

Or $\mathbf{q} = \mathbf{M}\mathbf{a}$ which can be solved as $\mathbf{a} = \mathbf{M}^{-1}\mathbf{q}$

This gives the coefficients needed for the affine transformation. What can you do if you have more than three correspondences?

Use the pseudo-inverse instead of the inverse!

Slide: Prof. G. Slabaugh, City U. London

Affine transform estimation

Method 1

From Eq. 4,

Solve by constructing a linear system of equations

$$\begin{aligned} u_i &= a_{11} x_i + a_{12} y_i + a_{13} \\ v_i &= a_{21} x_i + a_{22} y_i + a_{23} \end{aligned} \quad (5)$$

for $i = 1, \dots, n$.

Now, we have two sets of linear equations of the form

$$\mathbf{M} \mathbf{a} = \mathbf{b} \quad (6)$$

First set:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \end{bmatrix} = \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} \quad (7)$$

Slide: L.W. Kheng, National University of Singapore

Second set:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a_{21} \\ a_{22} \\ a_{23} \end{bmatrix} = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \quad (8)$$

- Can compute best fitting a_{kl} for each set independently using standard methods.

Slide: L.W. Kheng, National University of Singapore

Least Squares Estimation

Method 2

Compute the sum-squared error E as

$$E = \sum_{i=1}^n \|\mathbf{p}_i - \mathbf{T} \mathbf{q}_i\|^2 \quad (9)$$

If E is small, then the fit is good.

So, we want to find the best fitting \mathbf{T} that minimizes E .

So, do the usual thing: $\partial E / \partial \mathbf{T} = 0$

$$\begin{aligned} \frac{\partial E}{\partial \mathbf{T}} &= -2 \sum_i (\mathbf{p}_i - \mathbf{T} \mathbf{q}_i) \mathbf{q}_i^T = 0 \\ \sum_i \mathbf{T} \mathbf{q}_i \mathbf{q}_i^T &= \sum_i \mathbf{p}_i \mathbf{q}_i^T \end{aligned} \quad (10)$$

Slide: L.W. Kheng, National University of Singapore

That is,

$$\sum_i \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \begin{bmatrix} x_i & y_i & 1 \end{bmatrix} = \sum_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \begin{bmatrix} x_i & y_i & 1 \end{bmatrix} \quad (11)$$

Rearranging terms yield

$$\begin{bmatrix} \mathbf{M} & \mathbf{0} \\ \mathbf{0} & \mathbf{M} \end{bmatrix} \mathbf{a} = \mathbf{b} \quad (12)$$

where

$$\mathbf{M} = \begin{bmatrix} \sum_i x_i^2 & \sum_i x_i y_i & \sum_i x_i \\ \sum_i x_i y_i & \sum_i y_i^2 & \sum_i y_i \\ \sum_i x_i & \sum_i y_i & \sum_i 1 \end{bmatrix}$$

$$\mathbf{0} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{a} = [a_{11} \ a_{12} \ a_{13} \ a_{21} \ a_{22} \ a_{23}]^T$$

$$\mathbf{b} = \left[\sum_i u_i x_i \ \sum_i u_i y_i \ \sum_i u_i \ \sum_i v_i x_i \ \sum_i v_i y_i \ \sum_i v_i \right]^T$$

Now, we again have a linear system of equations to solve for a_{ij} .

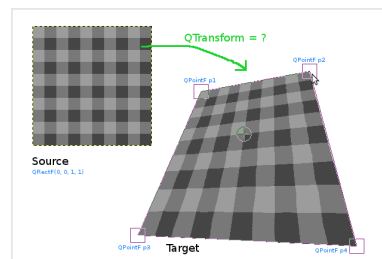
Projective transformation

Images normally acquired by photographic cameras are formed by perspective projection. If we view a planar surface not parallel to the image plane, then an affine transformation will not map the shape to a rectangle.

Instead, we must use a *projective* transformation of the form

$$\begin{bmatrix} x'w \\ y'w \\ w \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11}x + p_{12}y + p_{13} \\ p_{21}x + p_{22}y + p_{23} \\ p_{31}x + p_{32}y + 1 \end{bmatrix}$$

To estimate a projective transformation, at least **four** 2D correspondences are needed (due to the eight unknowns).



Projective transformation

Example: Find the best fitting warp to transform the game area to a given rectangle



original



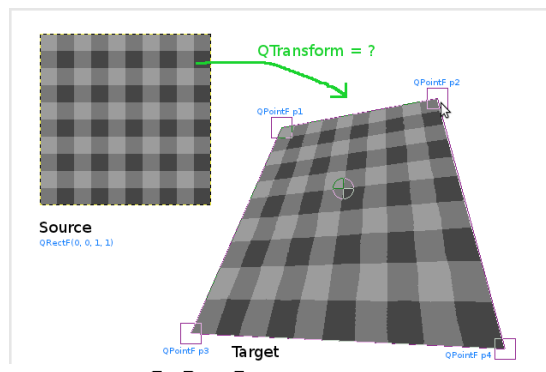
affine



projective

Slide: Prof. G. Slabaugh, City U. London

Projective (Perspective) Transformation

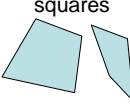
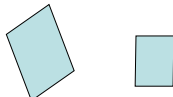
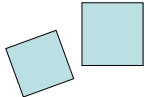
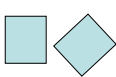


$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad (13)$$

- Eq. 13 is a set of linear equations.
- But, perspective transformation is a nonlinear transformation.
- Linear equations describe nonlinear transformation.
Seems paradoxical, but there is nothing wrong. Why?

Slide: L.W. Kheng, National University of Singapore

Hierarchy of 2D transformations

Projective 8 DOF	$\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & 1 \end{bmatrix}$	Transformed squares 	Preserves Collinearity
Affine 6 DOF	$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Parallellism of lines
Similarity 4 DOF	$\begin{bmatrix} s r_{11} & s r_{12} & t_x \\ s r_{21} & s r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Ratios of lengths, angles
Rigid-body 3 DOF	$\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Lengths, areas

Slide source: Marc Pollefeys

HIERARCHY OF TRANSFORMATIONS

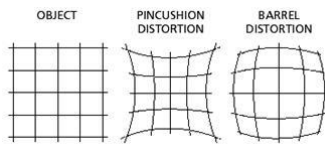
- Euclidean
- Similarity
- Affine
- Projective

Other transformations

Other transformations are possible, including those that do not involve a matrix, and instead a more general function that transforms pixel locations. What's needed is a way to describe the transformation

$$\mathbf{x}' = \mathbf{T}(\mathbf{x})$$

Examples of other common transformations include



Radial lens distortion

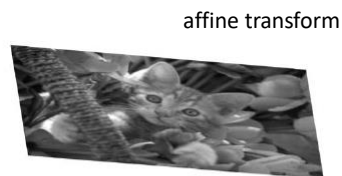


Non-rigid transformation

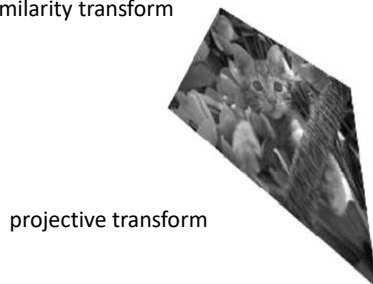
Example Image Transforms



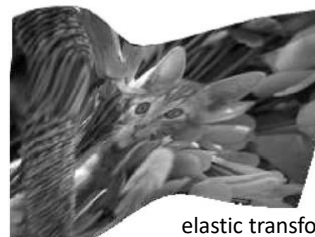
similarity transform



affine transform

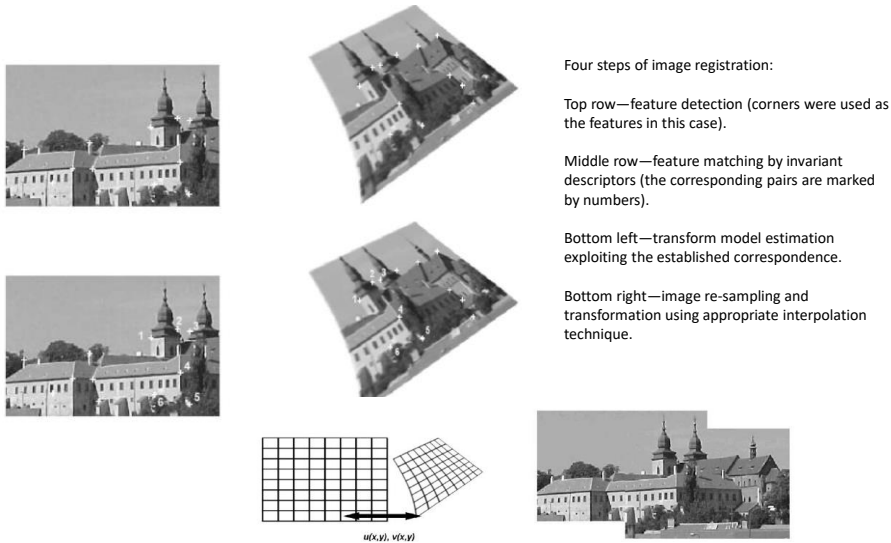


projective transform



elastic transform

IMAGE REGISTRATION PROBLEM



Correspondence Problem in Image Registration

For each point in one image, find the corresponding point in the other image.

Quite a challenging problem!

In this course: we will assume that we have the correspondences between control points

in today's lecture, we are not covering these

Intensity Based Correspondence Matching

- Block Matching
- Template Matching, Correlation Analysis
- For video: optical-flow based

in today's lecture, we are not covering these

Feature Based Correspondence Matching

- Edge matching,
- Identifying interesting points in both images and match these points using region correlation methods

ADVANTAGES:

- Ambiguities are reduced
- Less sensitive to photometric variations
- More accurate (sub-pixel accuracy)

GEOMETRIC Transformations

Geometric operations change the spatial relationships among the objects in an image.

1. Define the **spatial transformation** – moving each pixel from its initial to final position in the image
2. **Gray-level interpolation** (in general for a backward mapping) integer u, v positions on the output image map to fractional (noninteger) positions in the input image

GEOMETRIC OPERATIONS

The general definition for a geometric operation is

$$g(x,y) = f(x',y') = f[a(x,y),b(x,y)]$$

where $f(x,y)$ is the input image and $g(x,y)$ is the output image.

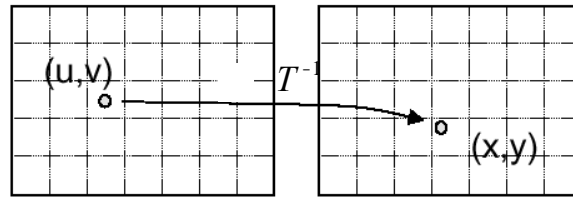
If a and b are continuous, connectivity will be preserved in the image.

Geometric Image Transform Implementation

- backward mapping

```
for (int u = 0; u < u_max; u++) {
  for (int v = 0; v < v_max; v++) {
    float x =  $T_x^{-1}(u, v)$ ;
    float y =  $T_y^{-1}(u, v)$ ;
     $J(u, v) = \text{resample\_I}(x, y)$ ; // interpolation over source image intensity values
  }
}
```

Note that
backward
mapping
is used

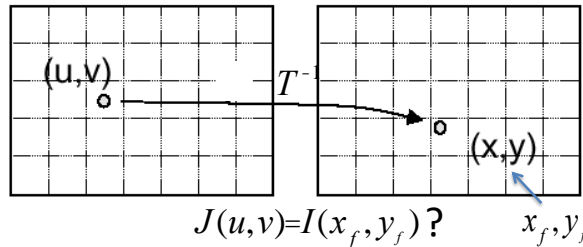
 T^{-1}


Destination Image

Source Image

Gray level interpolation

Through a geometric mapping, pixels in image f can map to positions between pixels in image g



T maps an integer coordinate point (u, v) in J to a real-coordinate point (x_f, y_f) in I .

We study 2 types of interpolation techniques:

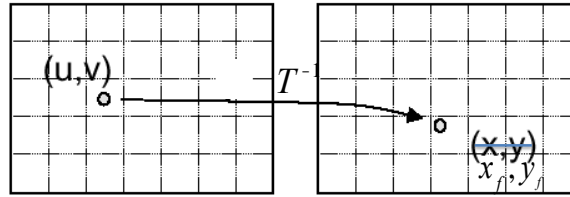
Nearest Neighbor interpolation: Gray level of the pixel (u, v) is taken to be that of the nearest pixel location to (x_f, y_f)

Bilinear Interpolation: an extension of linear interpolation for interpolating functions of two variables on a regular grid.

The key idea is to perform linear interpolation first in one direction, and then again in the other direction. Next page

Gray level interpolation

Through a geometric mapping, pixels in image f can map to positions between pixels in image g

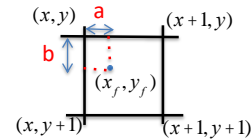


T maps an integer coordinate point (u, v) in J to a real-coordinate point (x_f, y_f) in I .

Use the colors (or gray values) of neighboring integer-coordinate points in I to estimate $I(x_f, y_f)$

Then: $J(u, v) = I(x_f, y_f)$

GRAY LEVEL INTERPOLATION



1. Nearest Neighbor interpolation:

$$J(u, v) = I(x_f, y_f) = I(x, y)$$

(x_f, y_f)

Floating point coordinate values

Nearest Integer coordinate values (rounded)

2. Bilinear Interpolation:

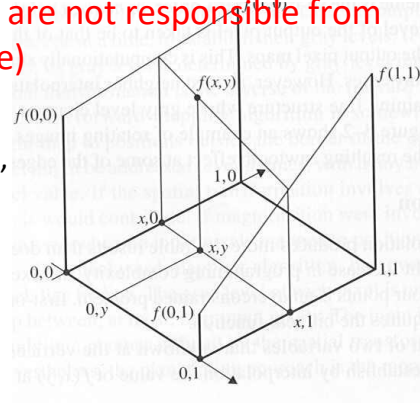
Advantage: round-off error avoided

$$I(x_f, y_f) = (1 - a)(1 - b)I(x, y) + a(1 - b)I(x + 1, y) \\ + (1 - a)b I(x, y + 1) + a b I(x + 1, y + 1)$$

Bilinear Interpolation Details: (You are not responsible from the derivation in this and next slide)

$$I(x,y) = ax + by + cxy + d$$

A bilinear function: its four coefficients, a through d , are to be chosen so that $I(x,y)$ fits the known values at the four corners.



1. Linearly interpolate between the upper two points to establish the value of:

$$I(x,0) = I(0,0) + x [I(1,0) - I(0,0)]$$

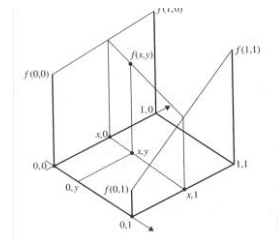
2. Similarly, for the lower two points

$$I(x,1) = I(0,1) + x [I(1,1) - I(0,1)]$$

3. Linearly interpolate vertically to determine the value of:

$$I(x,y) = I(x,0) + y [I(x,1) - I(x,0)]$$

4. Substituting all,



$$I(x,y) = [I(1,0) - I(0,0)]x + [I(0,1) - I(0,0)]y + [I(1,1) + I(0,0) - I(0,1) - I(1,0)]xy + f(0,0)$$

which is a bilinear equation.

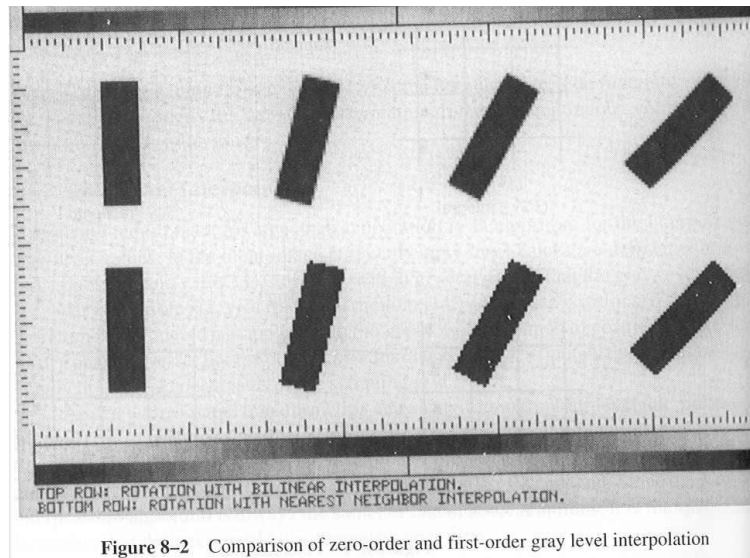


Figure 8-2 Comparison of zero-order and first-order gray level interpolation

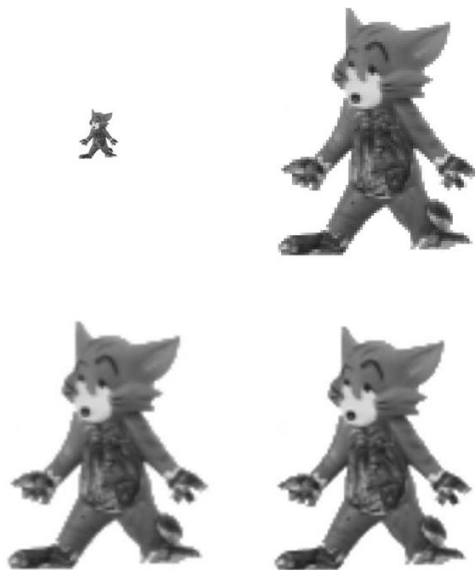


Image interpolation methods: the original image (top left) was enlarged five times using three different interpolation techniques—nearest neighbor (top right), bilinear (bottom left), and bicubic (bottom right).

Usually: Higher order interpolation than bilinear is NOT needed!

Interpolation: What to do at Image Grid Boundaries?

Specify value of your function at the boundaries, two ways:

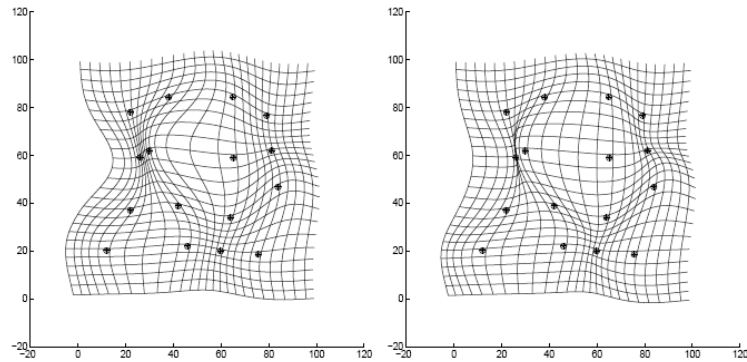
1. Pad zeros or a constant value at boundaries
2. Wrap your image around, i.e. Periodically replicate

Towards more General Spatial Transformations



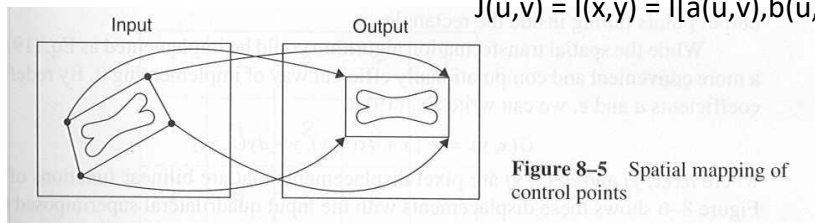
$$J(u,v) = I(x,y) \text{ where } x=a(u,v); y=b(u,v)$$

Towards more General Transformations



Specification by Control Points

$$J(u,v) = I(x,y) = I[a(u,v), b(u,v)]$$



Specify the spatial transformation as displacement values for selected *control points* in the image: $x=a(u,v)$; $y=b(u,v)$

1. Determine a set of suitable control points, from which the transform parameters (e.g. here polynomial) are determined.
2. Apply the actual correction to the image data using this transform (by finding all corresponding pixel locations in the two images)
3. Remap intensity data (i.e. interpolate)

Polynomial Transformation

In general, any polynomial transformation can be expressed as follows:

$$\begin{aligned} u &= \sum_k \sum_l a_{kl} x^k y^l \\ v &= \sum_k \sum_l b_{kl} x^k y^l \end{aligned} \quad (14)$$

Example: 2nd-order polynomial transformation.

$$\begin{aligned} u &= a_{20}x^2 + a_{02}y^2 + a_{11}xy + a_{10}x + a_{01}y + a_{00} \\ v &= b_{20}x^2 + b_{02}y^2 + b_{11}xy + b_{10}x + b_{01}y + b_{00} \end{aligned} \quad (15)$$

Slide: L.W. Kheng, National University of Singapore

In matrix form, we have

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a_{20} & a_{02} & a_{11} & a_{10} & a_{01} & a_{00} \\ b_{20} & b_{02} & b_{11} & b_{10} & b_{01} & b_{00} \end{bmatrix} \begin{bmatrix} x^2 \\ y^2 \\ xy \\ x \\ y \\ 1 \end{bmatrix} \quad (16)$$

If $a_{20} = a_{02} = a_{11} = b_{20} = b_{02} = b_{11} = 0$, then it becomes an affine transformation.

Again, given a set of corresponding points \mathbf{p}_i and \mathbf{q}_i , can form a system of linear equations to solve for the a_{kl} and b_{kl} . (Exercise)

12 coefficients need to be estimated, therefore at least six points are needed.

Slide: L.W. Kheng, National University of Singapore

Some Applications of Geometric Operations

- Image Warping
- Image morphing
- Geometric calibration
- Image Rectification

Image Warping

Goal: Warp a given image to a new purposefully distorted image

Given a source image I , and the correspondences between original control points p_i in I and desired destination points q_i $i=1,\dots,n$

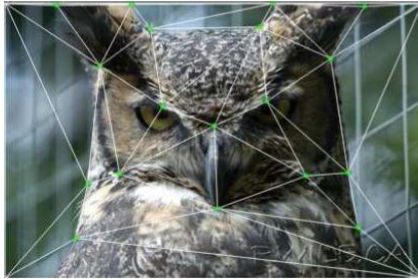
Generate a Warped image J such that

$$J(q_i) = I(p_i) \quad \forall i$$

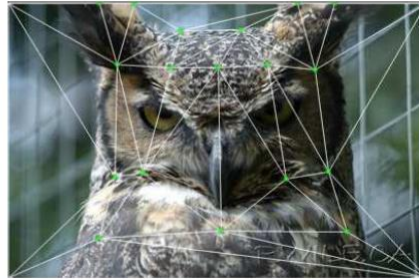
The idea of a correspondence can be given either by a mapping function f , or manually selected control point pairs.

Image Warping Local Transformation

Sample triangulation of an image:



(a) Initial control points.

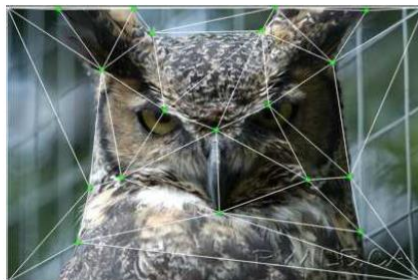


(b) Displaced control points.

Slide: L.W. Kheng, National University of Singapore

Image Warping Local Transformation

Image warping example:



(c) Initial image.



(d) Warped image.

Slide: L.W. Kheng, National University of Singapore

Warping and Morphing

Warping

- Single object
- Specification of original and deformed states

Morphing

- Two objects
- Specification of initial and final states

Image Morphing

Given two images I and J , generate a sequence of images $M(t)$, that changes smoothly from I to J .

$$0 \leq t \leq 1$$



Basic ideas:

- Select the corresponding points p_i in I and q_i in J .
- The corresponding point $r_i(t)$ in $M(t)$ lies in between p_i and q_i , e.g.

$$r_i(t) = (1 - t)p_i + t q_i$$
- Compute mapping function between I and $M(t)$ and between J and $M(t)$.
- Use the mapping functions to warp I to $I(t)$ and J to $J(t)$
- Blend $I(t)$ and $J(t)$:

$$M(t) = (1 - t)I(t) + t J(t)$$

Image Morphing

$$r_i(t) = (1 - t)p_i + t q_i$$

$$M(t) = (1 - t)I(t) + t J(t)$$

- Repeat for different values of t from 0 to 1.
- When the sequence is played, $r_i(t)$ moves from p_i to q_i , and $M(t)$ changes from I to J

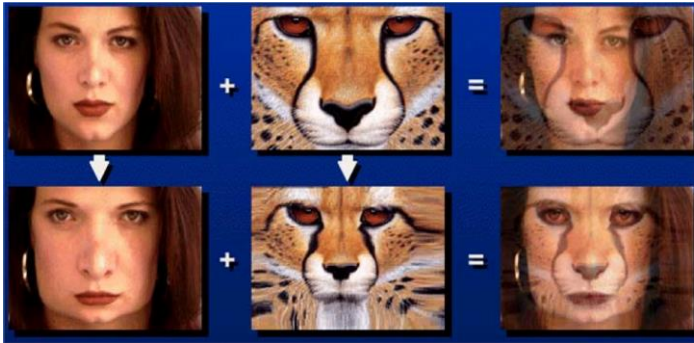


Figure 3.53 Image morphing (Gomes, Darsa, Costa *et al.* 1999) © 1999 Morgan Kaufmann.
 Top row: if the two images are just blended, visible ghosting results. Bottom row: both images are first warped to the same intermediate location (e.g., halfway towards the other image) and the resulting warped images are then blended resulting in a seamless morph.

Szeliski Computer Vision Book 2010

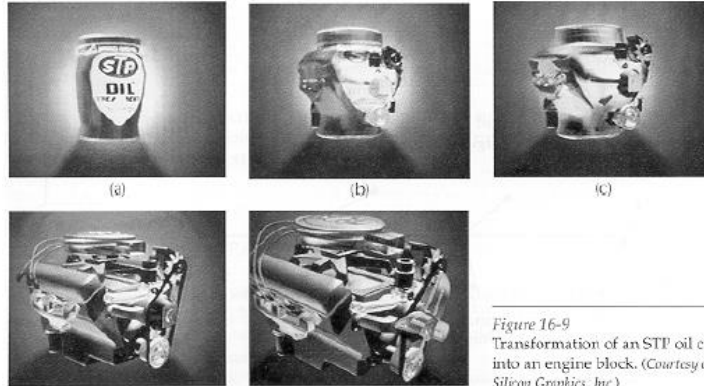
Image Morphing



- <https://paulbakaus.com/wp-content/uploads/2009/10/bush-obama-morphing.jpg>

Image Morphing involves Warping as a step

- Warping step is the hard one
 - Aligns features in images



Slide: Prof. M. Milanova, University of Arkansas at Little Rock

Warping

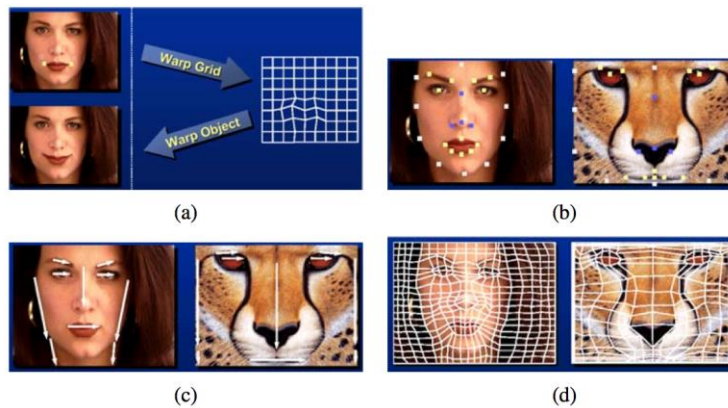


Figure 3.51 Image warping alternatives (Gomes, Darsa, Costa *et al.* 1999) © 1999 Morgan Kaufmann: (a) sparse control points \rightarrow deformation grid; (b) denser set of control point correspondences; (c) oriented line correspondences; (d) uniform quadrilateral grid.

Warping and Morphing

<http://www-2.cs.cmu.edu/~seitz/vmorph/vmorph.html>

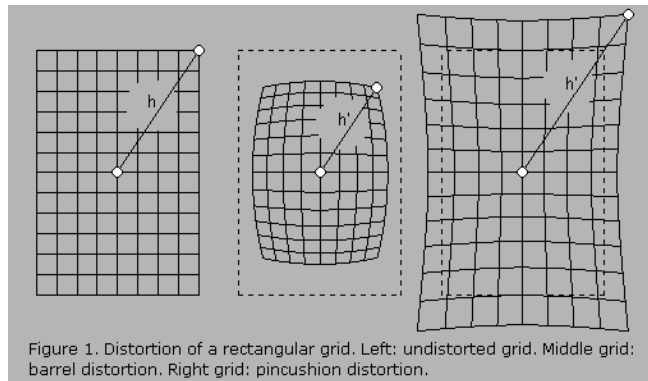
<http://www.css.tayloru.edu/~btoll/s99/424/res/model/morph/morph.html>

Commercial software such as : <http://www.fantamorph.com/>

More references on Image morphing:

- * G. Wolberg, "Digital Image Warping", IEEE Computer Soc. Press 1990.
- * S-Y.Lee and S.Y. Shin, "Warp generation and transition control in image morphing", In Interactive Computer Animation, Prentice Hall, 1996
- * Line Segment based morphing in the paper:
Beier, T. and Neely, S. (1992). "Feature-based image metamorphosis", Computer Graphics (SIGGRAPH '92), 26(2):35-42.

GEOMETRIC CORRECTION



The desired spatial transformation is that which makes the grid pattern rectangular again, thereby correcting the distortion introduced by the camera.

GEOMETRIC CORRECTION / Camera Calibration

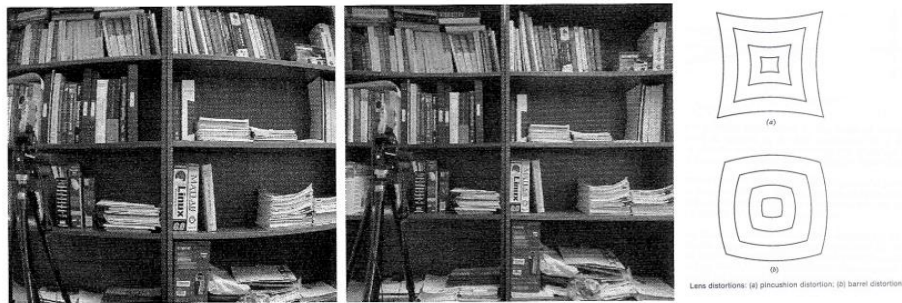


Figure 3.9. Left: image taken by a camera with a short focal length; note that the straight lines in the scene become curved on the image. Right: image with radial distortion compensated for.

To achieve geometric distortion correction, two entities are required:

1. A mathematical model that describes the distortion
2. A set of corresponding image points of the form $(x,y)(x_d,y_d)$ where
 - the 2×1 vector (x,y) represents location of the undistorted image plane point
 - The (x_d,y_d) represents the vector location of the distorted point

- Lens Distortion: A polynomial warp example

Simple Radial distortion model:

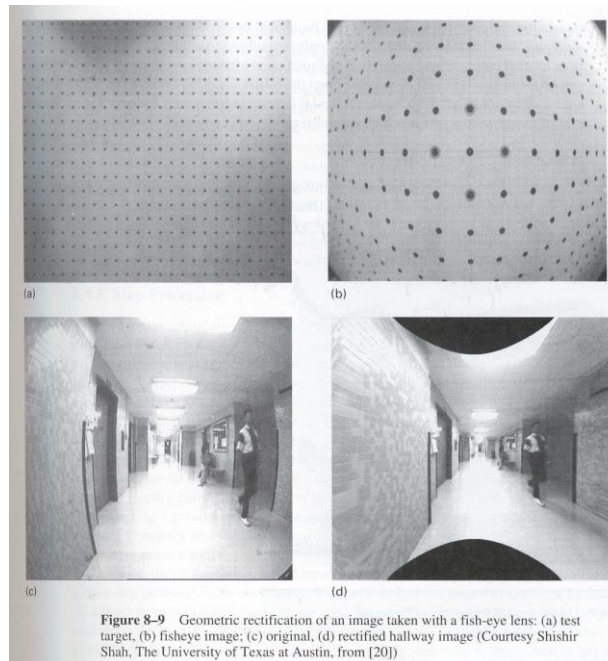
$$x = x_d(1 + a_1r^2 + a_2r^4)$$

$$y = y_d(1 + a_1r^2 + a_2r^4)$$

where

$$r^2 = (x_d - x_c)^2 + (y_d - y_c)^2$$

(x_c, y_c) : Center of radial distortion



END OF LECTURE

Recall Learning Objective (LO) for Week 3: Students will be able to:

LO2. Design and implement various image transforms: geometric image transforms

Next assignment: work on geometric xforms

Reading Assignments:

Study this week's topics from your lecture notes

NEXT TIME: We will study Neighborhood Image Processing, Spatial Filtering

Python code

```
import cv2
import numpy as np

l=cv2.imread("LicensePlate.png") #read image
height,width=l.shape[:2]
p1 = [18, 47]; p2 = [15, 100]; p3 = [178, 6]
q1 = [48, 50]; q2 = [48, 100]; q3 = [212, 50]
q = np.asarray([q1[0], q1[1], q2[0], q2[1], q3[0], q3[1]]).T
M = [[p1[0], p1[1], 1, 0, 0, 0],
      [0, 0, 0, p1[0], p1[1], 1],
      [p2[0], p2[1], 1, 0, 0, 0],
      [0, 0, 0, p2[0], p2[1], 1],
      [p3[0], p3[1], 1, 0, 0, 0],
      [0, 0, 0, p3[0], p3[1], 1]]
a = np.matmul(np.linalg.inv(M), q);
maxw, maxh = np.matmul(np.asarray([[a[0], a[1], a[2]],
                                   [a[3], a[4], a[5]]]), np.asarray([width, height, 1])).astype(int)

tform_l = cv2.warpAffine(l, np.asarray([[a[0], a[1], a[2]],
                                       [a[3], a[4], a[5]]]), (maxw, maxh))

cv2.imshow("License Plate Skewed", tform_l)
cv2.waitKey(0)
```



Slide: Prof. G. Slabaugh, City U. London

estimateGeometricTransform

OpenCV has a function, **estimateRigidTransform** that simplifies estimation of (similarity, affine, and projective) geometric transformations.

```
tform = cv2.estimateRigidTransform(matchedPoints1,matchedPoints2,transformType)

import cv2
import numpy as np

l=cv2.imread("LicensePlate.png") #read image
height,width=l.shape[:2]

p1 = [18, 47]; p2 = [15, 100]; p3 = [178, 6]
q1 = [48, 50]; q2 = [48, 100]; q3 = [212, 50]

P = np.asarray([p1, p2, p3])
Q = np.asarray([q1, q2, q3])

tform = cv2.estimateRigidTransform(P, Q, True)

maxw, maxh = np.matmul(tform, np.asarray([width, height, 1])).astype(int)
tform_l = cv2.warpAffine(l, tform, (maxw, maxh))
cv2.imshow("License Plate Skewed", tform_l)
cv2.waitKey(0)
```



Source: Prof. G. Slabaugh, City U. London

Estimation through correspondences

In a similar way as in the affine case, we can write

$$\begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x'_3 & -y_3x'_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y'_3 & -y_3y'_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x'_4 & -y_4x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y'_4 & -y_4y'_4 \end{bmatrix} \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{31} \\ p_{32} \end{bmatrix}$$

Or $q = Mp$ which can be solved as $p = M^{-1}q$

This gives the coefficients needed for the transformation. In OpenCV, one can use the **estimateRigidTransform** function with 'fullAffine=True' as the third argument to solve the equation shown above.

Polynomial Warping (Extra Material with the general form)

Use a polynomial as the general form of the transformation expression. Select the parameters to make it fit the control points and their specified displacements.

$$x' = \sum_{i=0}^N \sum_{j=0}^N k_{ij}^1 x^i y^j \quad y' = \sum_{i=0}^N \sum_{j=0}^N k_{ij}^2 x^i y^j$$

Image $f_1(x)$ is recorded in the (x_1, x_2) coordinate system, and image f_2 is recorded in the (w_1, w_2) coordinate system.

The polynomial warp relationship is:

$$x_1 = \sum_{i=0}^N \sum_{j=0}^N k_{ij}^1 w_1^i w_2^j \quad x_2 = \sum_{i=0}^N \sum_{j=0}^N k_{ij}^2 w_1^i w_2^j$$

$$x_1 = g_1(w_1, w_2) \quad x_2 = g_2(w_1, w_2)$$

Given a set of m control points in the x - and w - based coordinate systems of the form: (x_i, w_i) $i = 1, 2, 3, \dots, m$,
that is, point pairs of the form:

$$(x_{1i}, x_{2i}, w_{1i}, w_{2i})$$

expand the equation and solve for k_{ij}^1 and k_{ij}^2 variables.

For $N = 2$, general case:

$$\begin{aligned} x_{1k} = & k_{00}^1 + k_{10}^1 w_{1k} + k_{01}^1 w_{2k} + k_{11}^1 w_{1k} w_{2k} + k_{20}^1 (w_{1k})^2 + k_{02}^1 (w_{2k})^2 \\ & + k_{21}^1 (w_{1k})^2 w_{2k} + k_{12}^1 w_{1k} (w_{2k})^2 + k_{22}^1 (w_{1k})^2 (w_{2k})^2 \end{aligned}$$

$$\begin{aligned} x_{2k} = & k_{00}^2 + k_{10}^2 w_{1k} + k_{01}^2 w_{2k} + k_{11}^2 w_{1k} w_{2k} + k_{20}^2 (w_{1k})^2 + k_{02}^2 (w_{2k})^2 \\ & + k_{21}^2 (w_{1k})^2 w_{2k} + k_{12}^2 w_{1k} (w_{2k})^2 + k_{22}^2 (w_{1k})^2 (w_{2k})^2 \end{aligned}$$

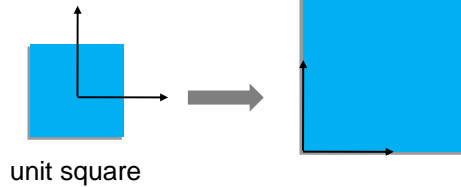
18 coefficients need to be estimated, therefore at least nine points are needed.

In-class exercise

Consider the transformation

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

How does it transform the unit square?



unit square

What about

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



unit square

Slide: Prof. G. Slabaugh, City U. London

2D Transformation Groups and Invariants

Group	Matrix	Distortion	Invariant properties
Projective 8 dof	$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$		Concurrency, collinearity, order of contact : intersection (1 pt contact); tangency (2 pt contact); inflections (3 pt contact with line); tangent discontinuities and cusps. cross ratio (ratio of ratio of lengths).
Affine 6 dof	$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Parallelism, ratio of areas, ratio of lengths on collinear or parallel lines (e.g. midpoints), linear combinations of vectors (e.g. centroids).
Similarity 4 dof	$\begin{bmatrix} sr_{11} & sr_{12} & t_x \\ sr_{21} & sr_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Ratio of lengths, angle.
Euclidean 3 dof	$\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Length, area

Table 1.1. Geometric properties invariant to commonly occurring *planar* transformations. The matrix $A = [a_{ij}]$ is an invertible 2×2 matrix, $R = [r_{ij}]$ is a 2D rotation matrix, and (t_x, t_y) a 2D translation. The distortion column shows typical effects of the transformations on a square. Transformations higher in the table can produce all the actions of the ones below. These range from Euclidean, where only translations and rotations occur, to projective where the square can be transformed to any arbitrary quadrilateral (provided no three points are collinear).

Table from [Hartley,Zisserman] book