

BLG453E COMPUTER VISION

Fall 2021 Term

Week 3



Istanbul Technical University
Computer Engineering Department

Instructor: Prof. Güzde ÜNAL

Teaching Assistant: Yusuf H. ŞAHİN

1

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

2



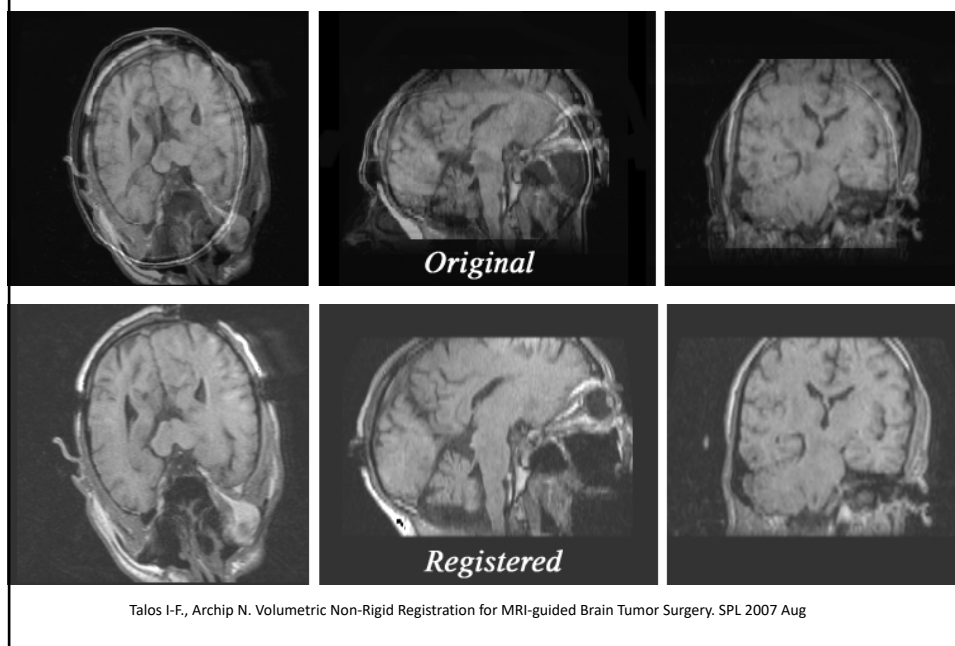
Week 3: LOs: Geometric Image transforms

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

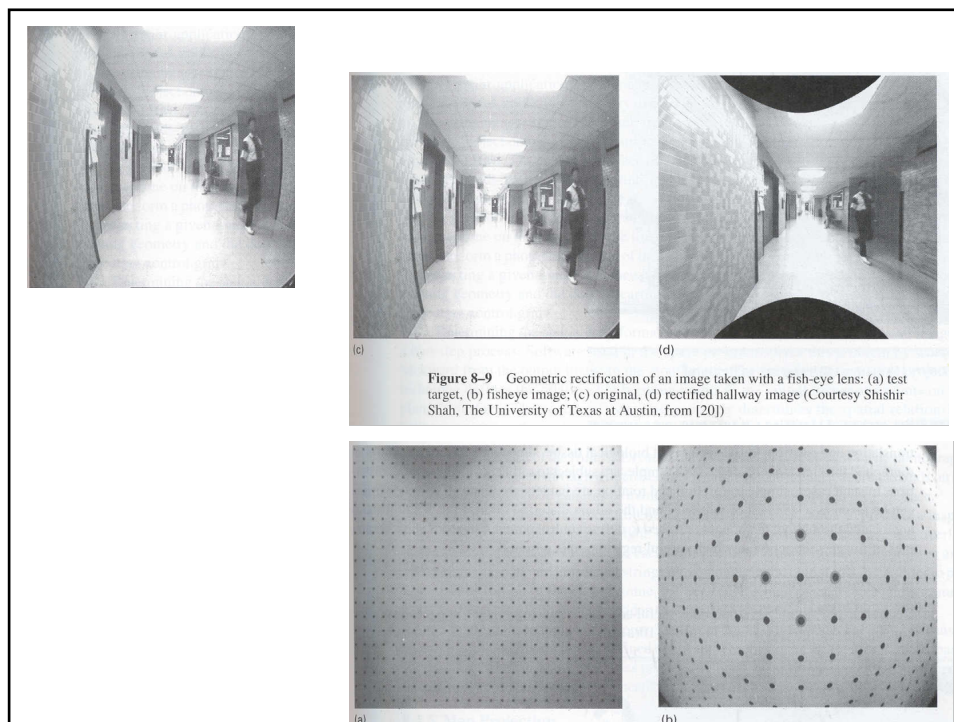
2. Design and implement various image transforms:
 - geometric transforms over images, point-wise transforms, neighborhood operation-based spatial filters

3

Image Registration Problem



4



5

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

- Bringing multiple images into the same coordinate system, e.g. Registration, Homography Estimation (related to 3D Vision)
- Removing distortion
- Simplifying further processing, eg. In stereo matching
- Image morphing, warping, etc.

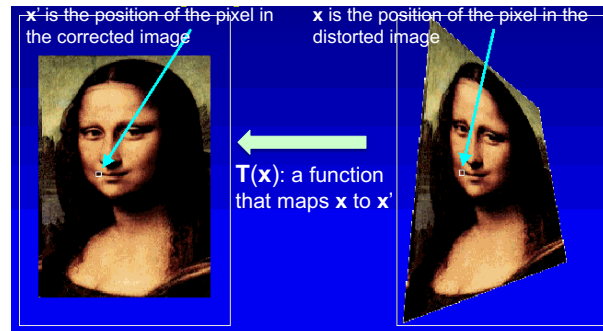


Distorted image

Corrected image

6

Geometric transformations



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

Map your coordinates first $\mathbf{x}' = \mathbf{T}(\mathbf{x})$

then map the images $J(\mathbf{x}') = I(\mathbf{x})$

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

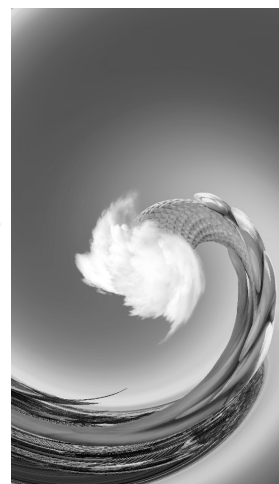
7

Image Warping (Geometric Transformation)

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



Source Image

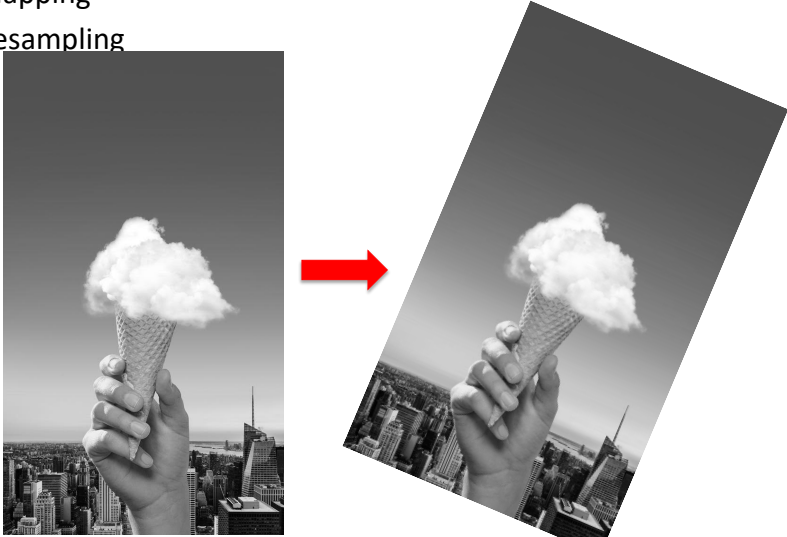


Destination Image

8

Image Warping (Geometric Transformation)

- Move pixels of image by
 - Mapping
 - Resampling




Source Image

Destination Image

9

Coordinate Mapping

- 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

Im = cv2.imread("sample1.jpg", -1)
[h, w, dim] = Im.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
  
```

Question/Exercise:
Which transform effect is given here?
Code and observe

10

Computing an Image Warp

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)
  
```

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

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

11

Example Coordinate Mapping: Scaling

- Scale by factor

$$x = factor * u$$

$$y = factor * v$$

```

lm = cv2.imread("sample1.jpg", -1)
[h, w, dim] = lm.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
  
```

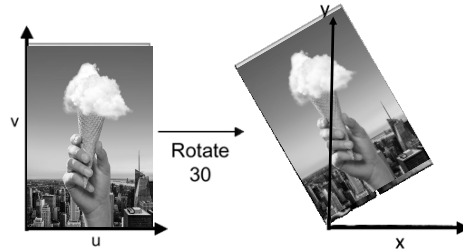
12

Example Coordinate Mapping

- Rotate by θ degrees

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

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



```

lm = cv2.imread("sample1.jpg", -1)
[h, w, dim] = lm.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
  
```

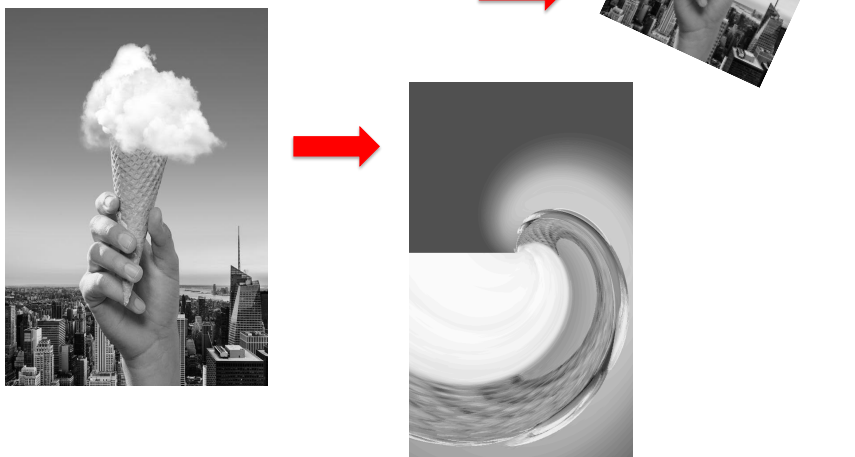
13

A general Coordinate Mapping

- Any function of u and v

$$x = f_x(u, v)$$

$$y = f_y(u, v)$$



14

Geometric Transform: Forward Mapping

- Iterate over source image

Rotate
-30

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

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

15

Backward Mapping

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

- We must resample source image

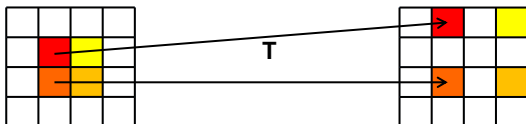
Rotate by 30°

Forward mapping was rotation by: -30°
This is the inverse mapping for that

16

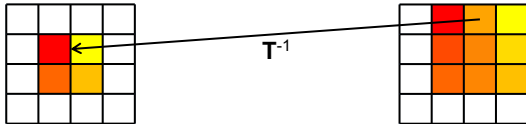
Geometric transformations

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} .



T

Forward mapping may result in gaps



T^{-1}

Inverse mapping ensures no gaps

⇒ 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.

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

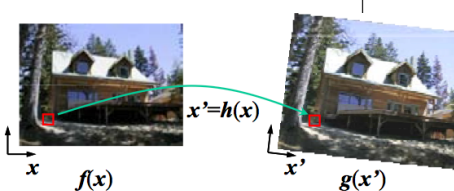
17

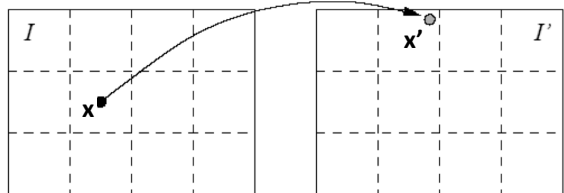
Forward Mapping / Warp (NOT USED much in practice)

procedure *forwardWarp*($f, h, \text{out } g$):

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 $g(x')$.





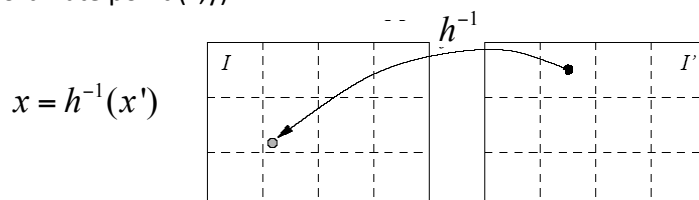
$x' = f(x)$ is real valued, there will be round-off errors, and missing grid points.

R. Szeliski, Computer Vision Book, 2010

18

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

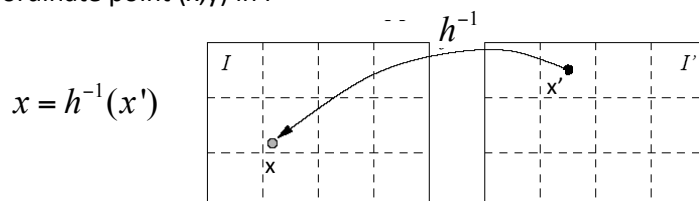
Equivalently: $I'(x',y') = I(h^{-1}(x',y')) = I(x,y)$

Advantage: No round-off error

19

Backward Mapping

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



```

lm = cv2.imread("sample1.jpg", -1)
[h, w, dim] = lm.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

```

20

Geometric Image Transform Implementation

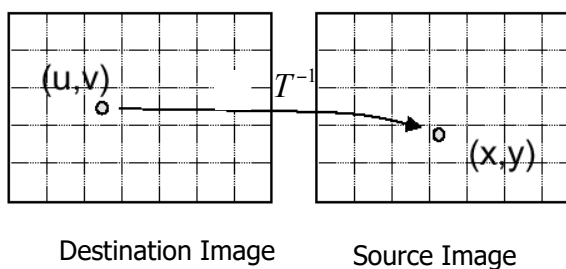
- backward mapping

```

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) = resample_I(x, y); // interpolation over source image intensity values
  }
}

```

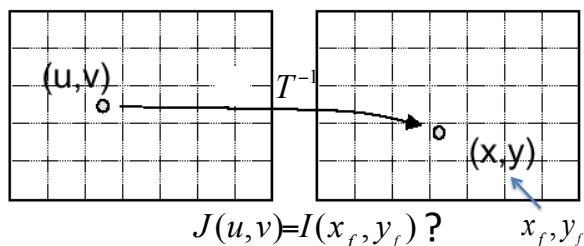
Note that backward mapping T^{-1} is used



21

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:

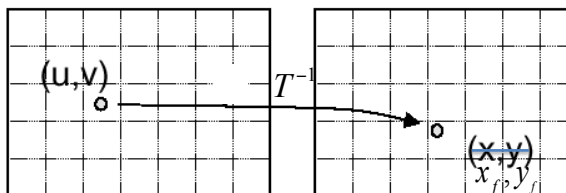
Nearest Neighbor interpolation: Gray level of the pixel (u, v) is taken to be that of the nearest pixel location to (x_f, y_f)

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

22

Gray level interpolation

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



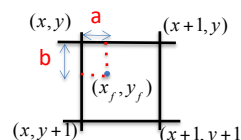
T maps an integer coordinate point (u,v) in J to a real-coordinate point (x_f, y_f) in I.

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

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

23

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)

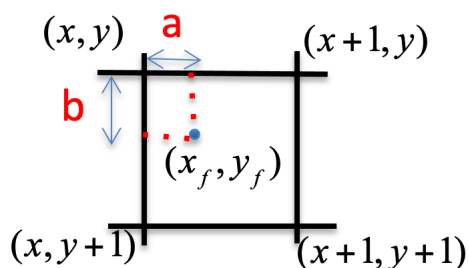
2. Bilinear Interpolation:

Advantage: round-off error avoided

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

24

GRAY LEVEL INTERPOLATION



2. Bilinear Interpolation:

Advantage: round-off error avoided

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

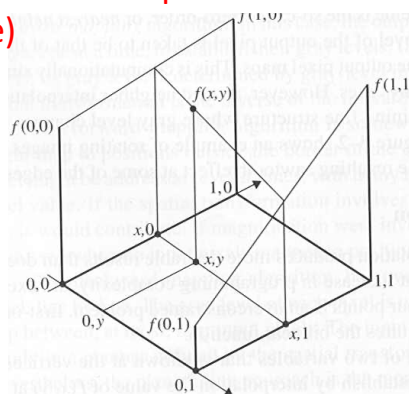
equation (*)

25

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: is linear in both of its arguments, i.e. x and y above; 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

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

26

3. Linearly interpolate vertically to determine the value of:

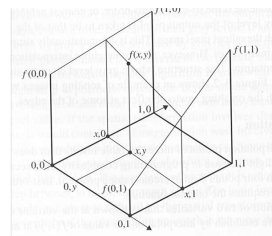
$$I(x,y) = I(x,0) + y [I(x,1) - I(x,0)]$$

4. Substituting all,

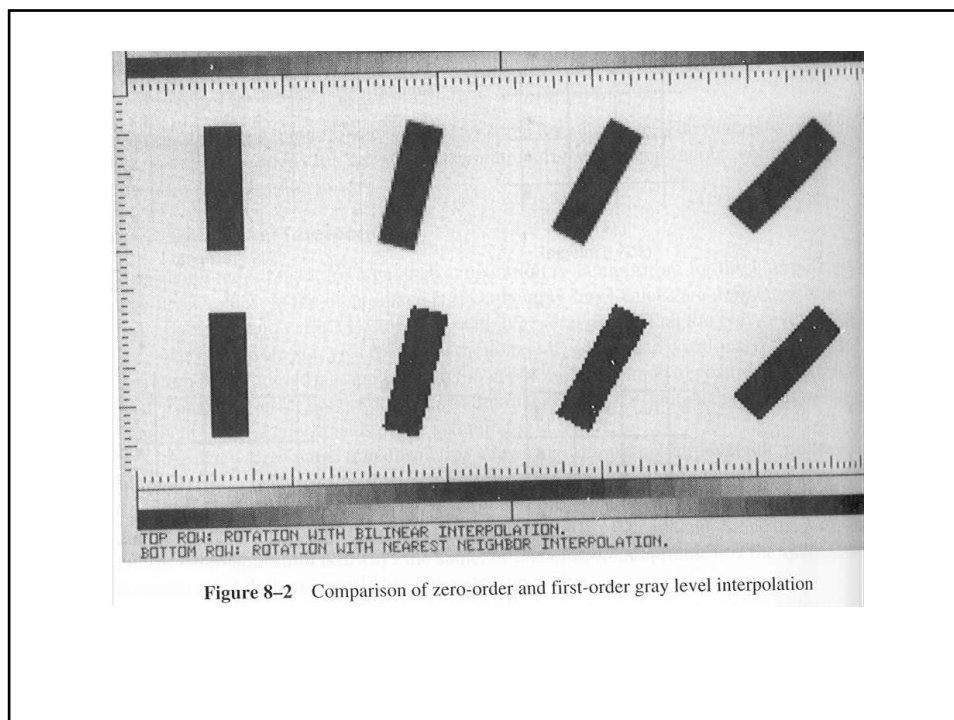
$$I(x,y) = [I(1,0) - I(0,0)]x + [I(0,1) - I(0,0)]y + [I(1,1) + I(0,0) - I(0,1) - I(1,0)]xy + I(0,0)$$

which is a bilinear equation.

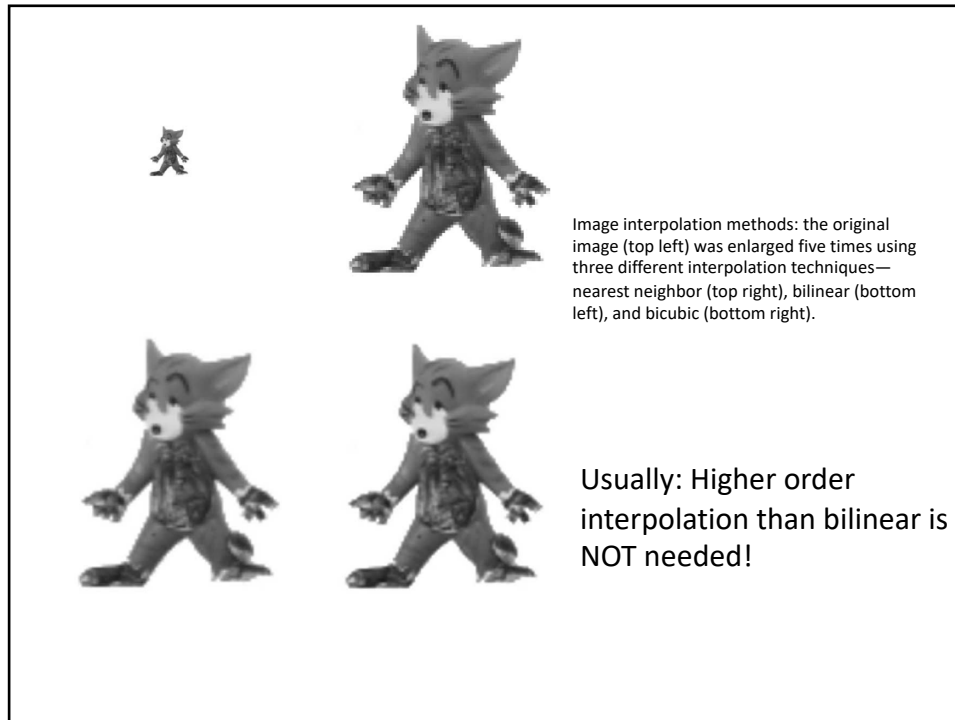
Note: This is equal to the equation (*) two slides ago.



27



28



29

Interpolation: What to do at Image Grid Boundaries?

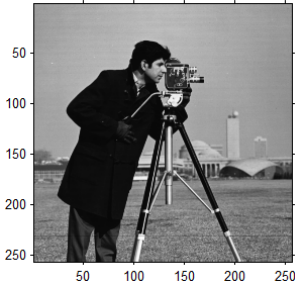
Specify value of your function at the boundaries, two ways:

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

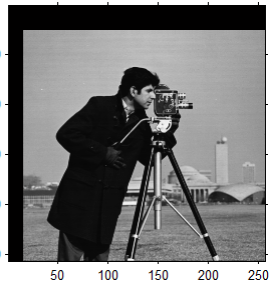
30

THQ

U,V



X,Y



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

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

Next: PARAMETRIC GEOMETRIC TRANSFORMS

31

TRANSLATION

$$U = X + X_0$$

$$V = Y + Y_0$$

Homogeneous coordinates

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

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

32

TRANSLATION

In Homogeneous coordinates:

$$x = \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \quad u = \begin{bmatrix} U \\ V \\ 1 \end{bmatrix}$$

$$u = Tx$$

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

```

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)

```

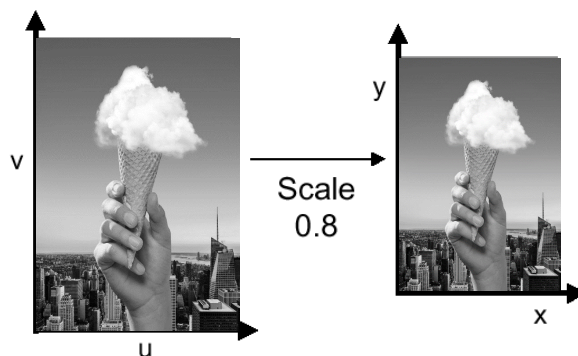
$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

33

SCALING

$$U = 1/s * X$$

$$V = 1/s * Y$$

e.g. $s = 0.8$

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

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

non-uniform scaling
This is more general

```

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

```

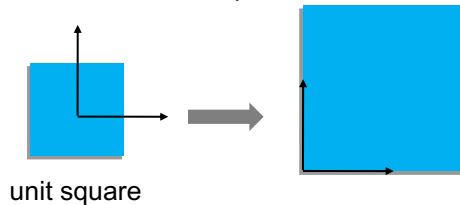
34

Question

Consider the transformation

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

How does it transform the unit square?



35

ROTATION

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

Q: How many parameters ?

$$\theta = \pi / 4$$

Important: Need to specify the Center of Rotation.



Above example Q: **Center of rotation?** =Image Center

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

36

Let $u=U(x,y) = x \cos(\theta) - y \sin(\theta)$

$v=V(x,y) = x \sin(\theta) + y \cos(\theta)$

produces a clockwise rotation through an angle θ 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)
```

Or Rotate around a given center of rotation (x_0, y_0) :

$x' = x - x_0$; $y' = y - y_0$;

$u' = x' \cos(\theta) - y' \sin(\theta)$

$v' = x' \sin(\theta) + y' \cos(\theta)$

Add back the center of rotation \rightarrow $u = u' + x_0$
 $v = v' + y_0$

37

How to compute INVERSE TRANSFORMATION?

Compute inverse of the transformation matrix:

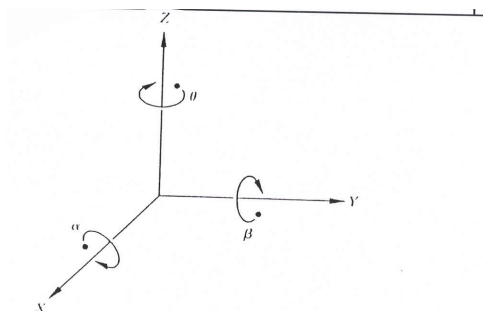
$$T^{-1} = \begin{bmatrix} 1 & 0 & -X_0 \\ 0 & 1 & -Y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

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

38

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.

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

39

THQ

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} c & 0 & 0 \\ 0 & d & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

What happens to the image by the above transform?

Assume $c, d > 1$

Answer: The image will be shrunk by the factors c in the x -direction and d in the y -direction

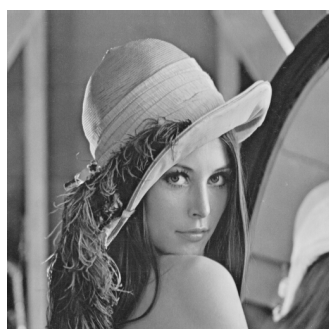
The origin (typically the upper left-hand corner) remains stationary.

Q: What about $c, d < 0$? What is the effect?

40

AFFINE TRANSFORM

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



41

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`

42

Special cases

There are several common special cases, including

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

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

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

43

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

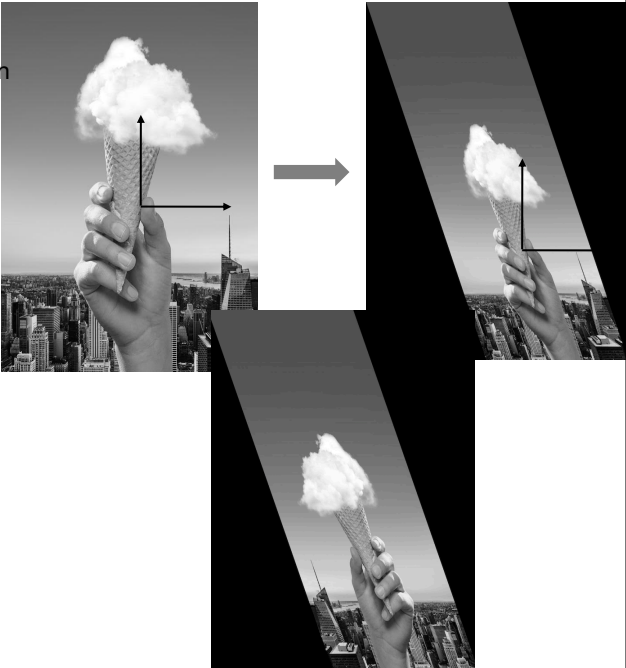
44

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 ?

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

45

Example

```

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


height,width=I.shape[:2]

# Rotate clockwise 15 degrees to align base
M= cv2.getRotationMatrix2D((width/2,height/2),-15,1)
J1 = cv2.warpAffine(I,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)

```



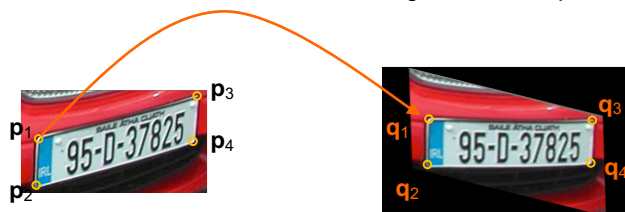
Slide: Prof. G. Slabaugh, City U. London

46

Estimation of Affine Transform through correspondences

When the affine transformation is not known in advance, we can estimate it.

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.



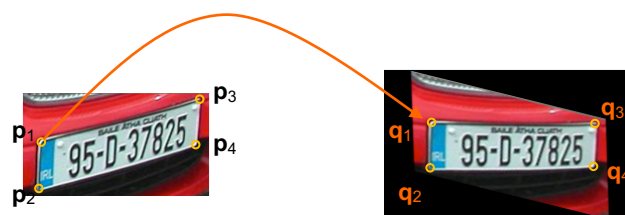
Estimation here means: to determine the six coefficients in the A matrix.

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

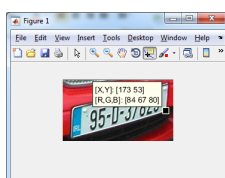
This can be achieved by finding at least **three** correspondences, or matching points.

47

Estimation of Affine Transform through correspondences



Say you pick 4 point correspondences:



$$\begin{aligned} \mathbf{p}_1 &= [18, 47]^T & \mathbf{q}_1 &= [48, 50]^T \\ \mathbf{p}_2 &= [15, 100]^T & \mathbf{q}_2 &= [48, 100]^T \\ \mathbf{p}_3 &= [178, 6]^T & \mathbf{q}_3 &= [212, 50]^T \\ \mathbf{p}_4 &= [173, 53]^T & \mathbf{q}_4 &= [212, 100]^T \end{aligned}$$

Next, set up the equations

48

Affine Transform

$$p = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \quad q = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (4)$$

- p and q are **homogeneous** coordinates.
- Affine transformation is a linear transformation.



- How many corresponding pairs needed to solve for the parameters?

49

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 $q = Ma$ which can be solved as $a = M^{-1}q$

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!

50

Estimation through correspondences

51

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:

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

52

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.

53

Least Squares Estimation

54

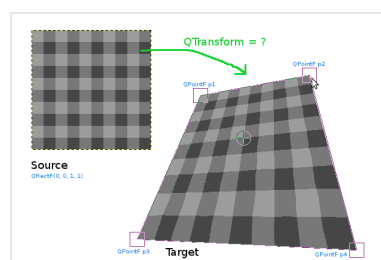
Projective transformation

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).



58

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

59

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?

60

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	$\begin{bmatrix} sr_{11} & sr_{12} & t_x \\ sr_{21} & sr_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		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

61

HIERARCHY OF TRANSFORMATIONS

- Euclidean
- Similarity
- Affine
- Projective

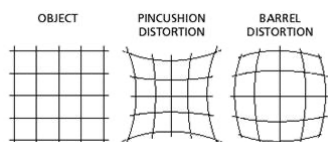
62

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

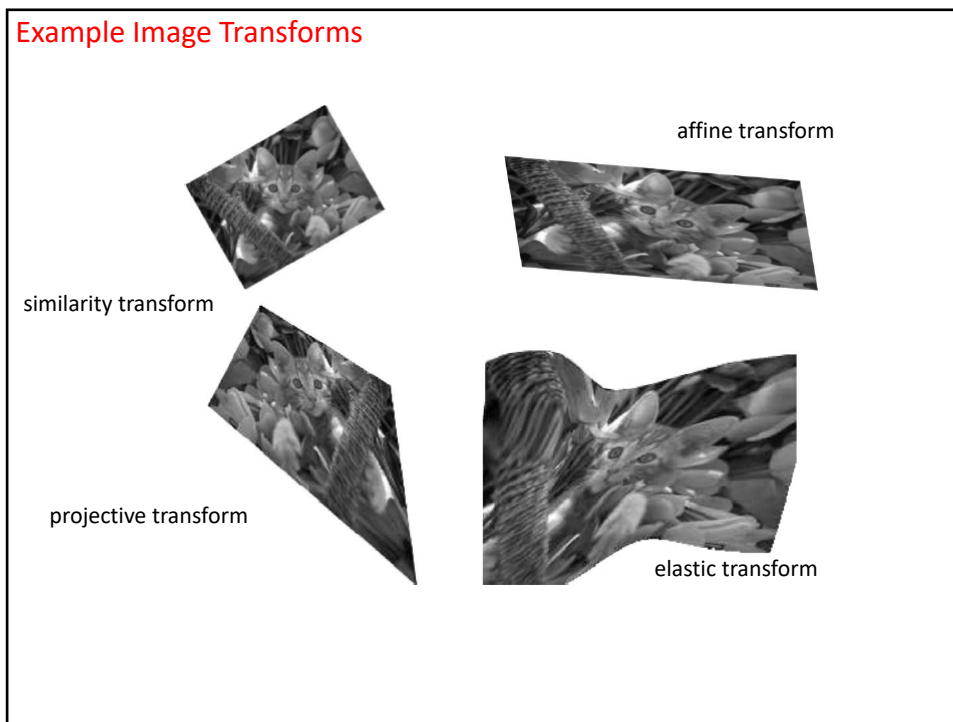


Radial lens distortion

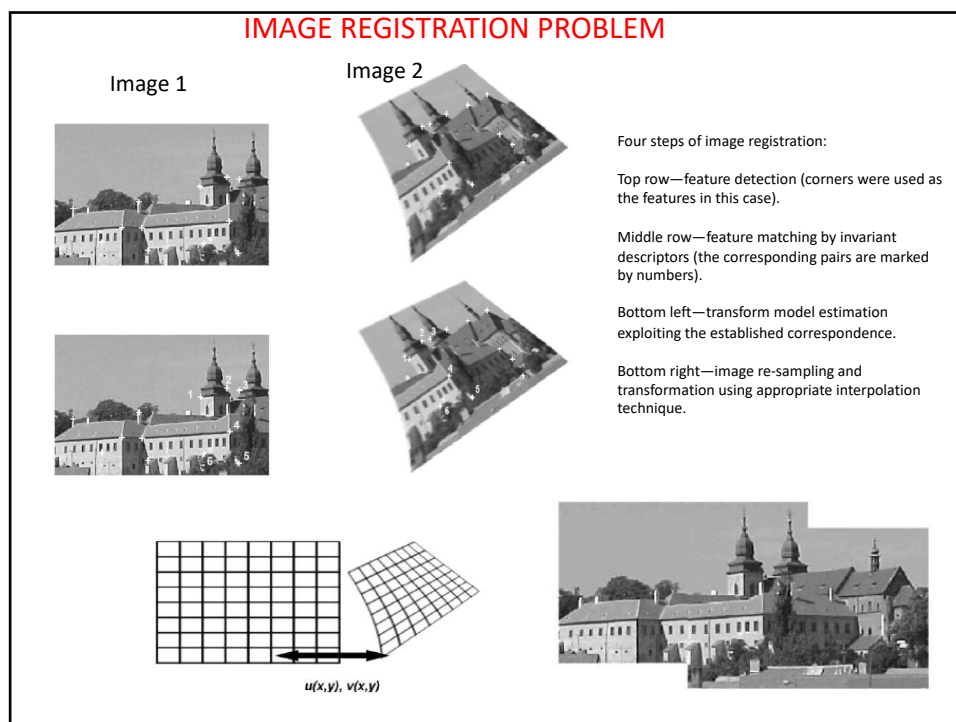


Non-rigid transformation

63



64



65

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

66

GEOMETRIC Transformations - Summary

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

1. Define the **spatial (geometric) 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

69

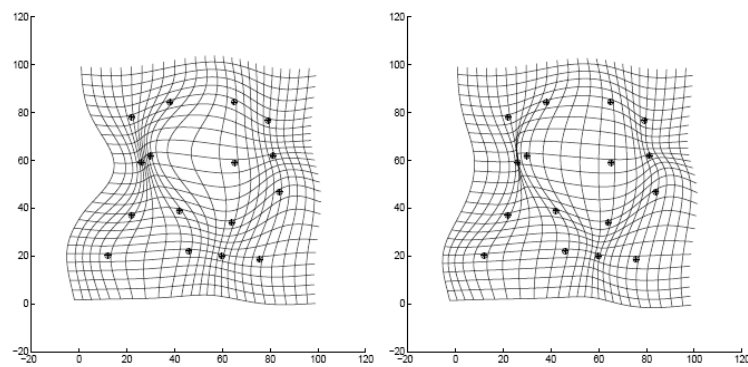
Towards more General Spatial Transformations



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

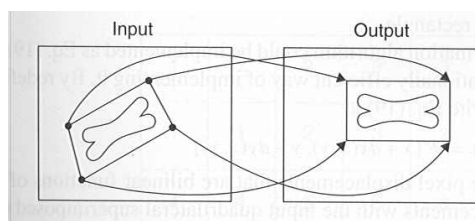
71

Towards more General Transformations



72

Specification by Control Points



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

Figure 8-5 Spatial mapping of control points

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)

73

Polynomial Transformation

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

$$\begin{aligned} u &= \sum_{k=0}^n \sum_{l=0}^{n-k} a_{kl} x^k y^l && \text{e.g. write 3rd order, } n=3 \\ v &= \sum_{k=0}^n \sum_{l=0}^{n-k} b_{kl} x^k y^l \end{aligned} \quad (14)$$

Example: 2nd-order polynomial transformation.

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

74

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)

Q: how many points correspondences are needed to estimate the above transform?

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

75

Some Applications of Geometric Operations

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

76

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

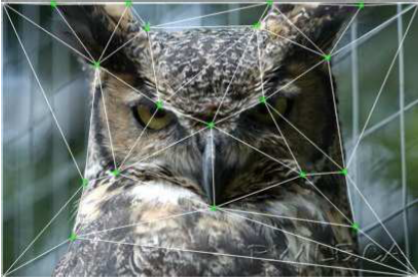
$$J(q_i) = I(p_i) \quad \forall i$$

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

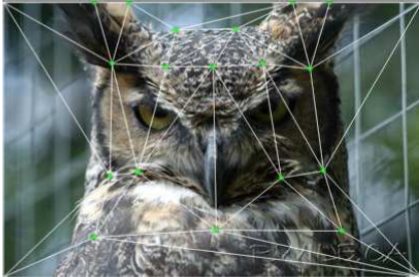
77

Image Warping
Local Transformation

Sample triangulation of an image: ↑



(a) Initial control points.



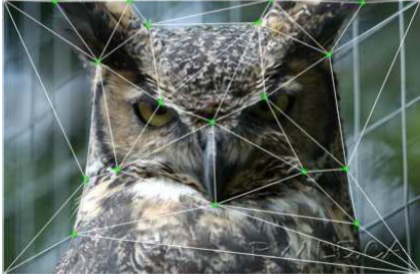
(b) Displaced control points.

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


78

Image Warping Local Transformation

Image warping example:



(c) Initial image.



(d) Warped image.

↑

79

Warping and Morphing

Warping

- Single object
- Specification of original and deformed states

Morphing

- Two objects
- Specification of initial and final states

80

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 steps:

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

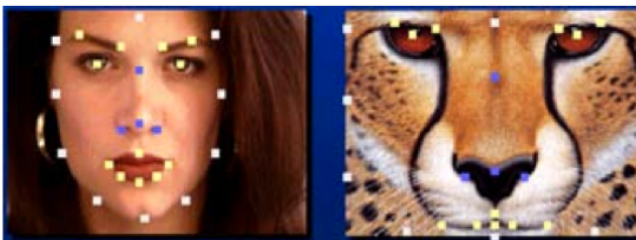
$$r_i(t) = (1-t)p_i + t q_i$$

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

81

Image Morphing

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



$$0 \leq t \leq 1$$

82

Image Morphing

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

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

6. 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

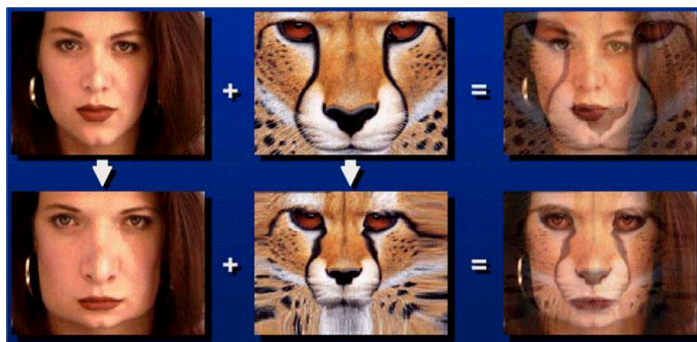


Figure 3.53 Image morphing (Gomes, Darsa, Costa *et al.* 1999) © 1999 Morgan Kaufmann.
 Top row: if the two images are just blended, visible ghosting results. Bottom row: both images are first warped to the same intermediate location (e.g., halfway towards the other image) and the resulting warped images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

Szeliski Computer Vision Book 2010

83

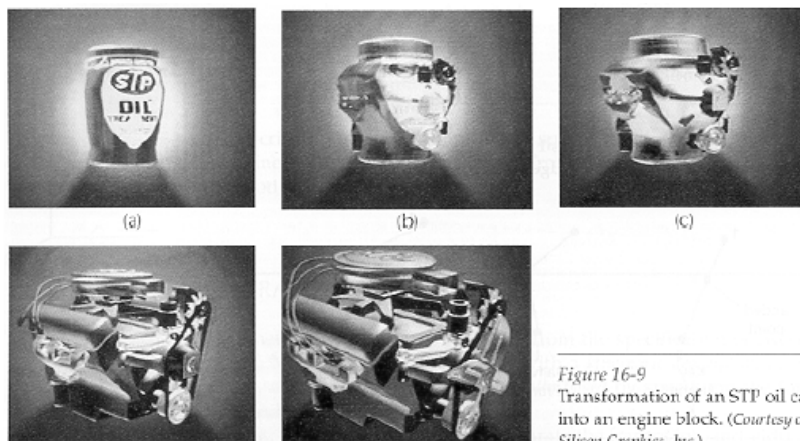
Image Morphing



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

84

Image Morphing involves Warping as a step



- Warping: You need to align features in images

85

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>

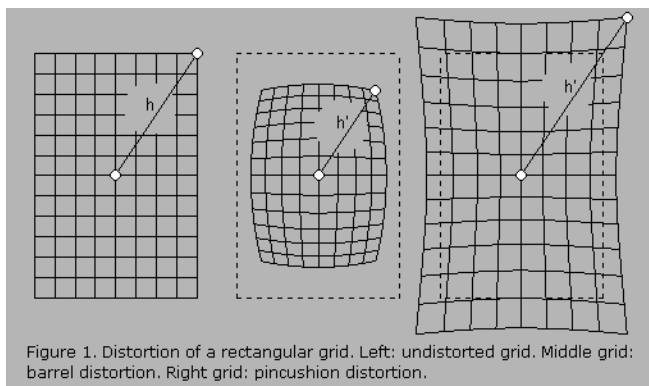
Commercial software such as : <http://www.fantamorph.com/>

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.

87

GEOMETRIC CORRECTION



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

88

GEOMETRIC CORRECTION / Camera Calibration

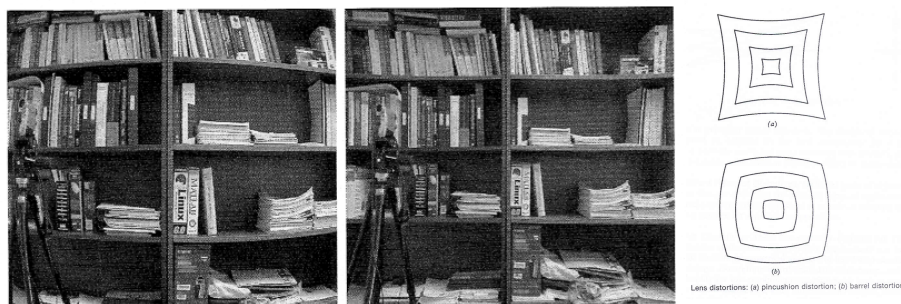


Figure 3.9. Left: image taken by a camera with a short focal length; note that the straight lines in the scene become curved on the image. Right: image with radial distortion compensated for.

89

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

90

- Lens Distortion: A polynomial warp example

Simple Radial distortion model:

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

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

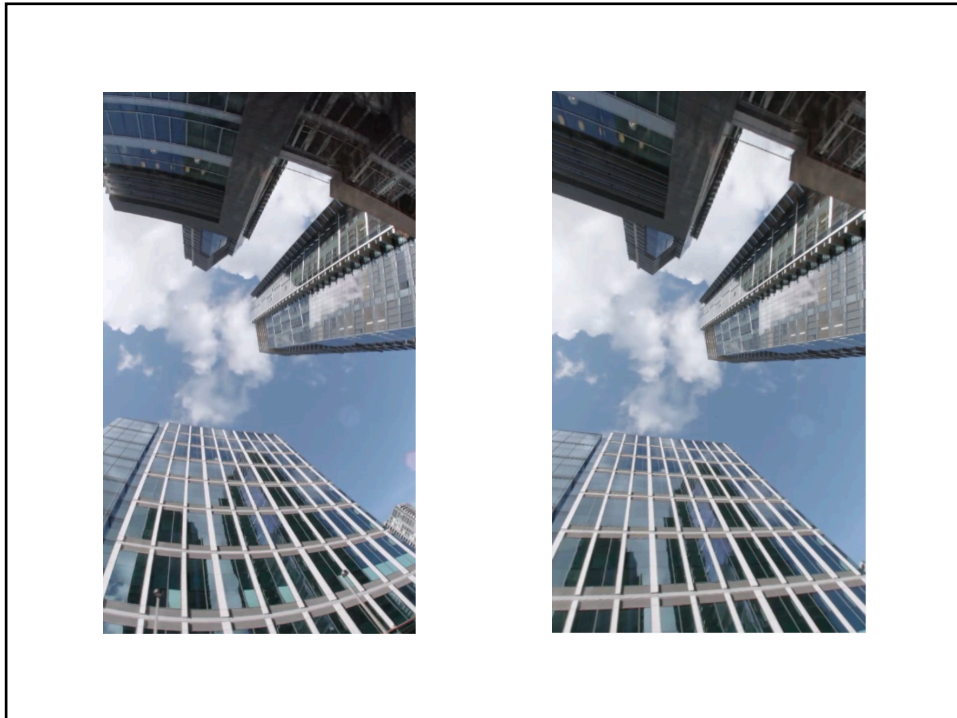
where a_1 and a_2 are the parameters to be estimated, and:

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

: r: distance from center of radial distortion = image center usually

(x_c, y_c) : Center of radial distortion

91



92

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

93