Practice: Image Processing III

COMPUTER VISION (COURSE-HY24011)

Q YOUN HONG

Content

- **Morphological Operation**
- 2D Geometric Transformation

Morphological Operation

- Change the shape of the underlying (binary/grayscale) object
- Do convolution of a structuring element with the image
- Dilation, erosion, opening, closing, etc.

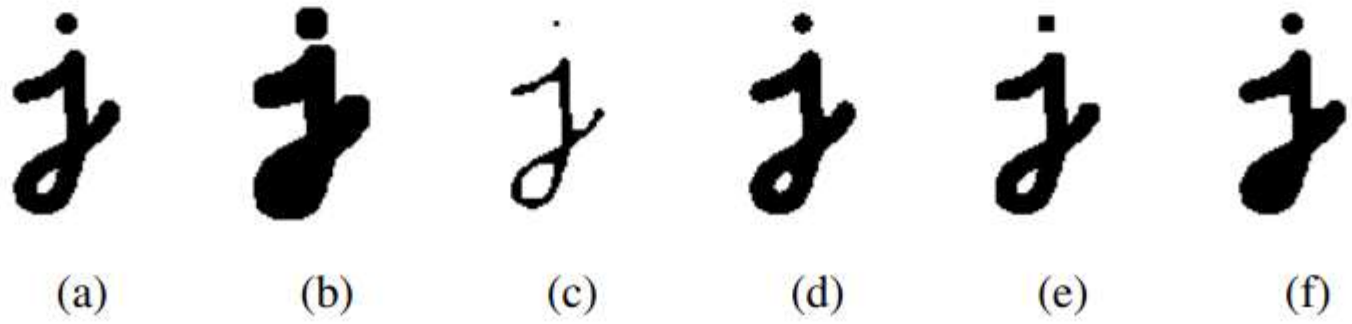
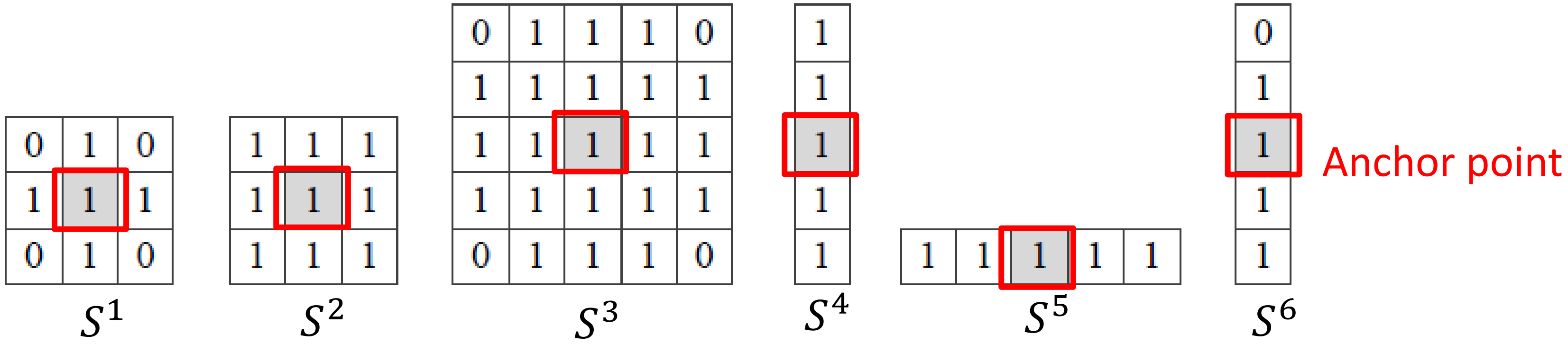


Figure 3.22 *Binary image morphology: (a) original image; (b) dilation; (c) erosion; (d) majority; (e) opening; (f) closing. The structuring element for all examples is a 5×5 square. The effects of majority are a subtle rounding of sharp corners. Opening fails to eliminate the dot, as it is not wide enough.*

Morphological Operation

- Structuring elements
  - Can be of any shape
  - Represented as a set of non-zero pixel elements



$S^1 = \{(0,0), (0,1), (-1,0), (0,-1),(1,0)\}$

$S^2 = \{(0,0), (0,1)(-1,1), (-1,0), (-1,-1), (0,-1), (1,-1), (1,0), (1,1)\}$

Morphological Operation

1. Dilation
   - For every non-zero pixel of f, change $f(j+k, i+l)$ to 1 if $(k,l) \in S \Rightarrow f \oplus S = \bigcup_{\boldsymbol{x} \in f} S_{\boldsymbol{x}}$
   - Thickens the shape by the structuring element
2. Erosion
   - Set $f(j,i) = 1$ only if $f(\mathrm{j}+\mathrm{k}, \mathrm{i}+\mathrm{l}) = 1$ for $\forall(k,l) \in S \Rightarrow f \ominus S = \{\boldsymbol{x} | \boldsymbol{x} + \boldsymbol{s} \in f, \forall \boldsymbol{s} \in \mathrm{S}\}$
   - Shrinks the shape by the structuring element
3. Opening
   - $f \circ S = (f \ominus S) \oplus S$
   - Removes the shapes smaller than the structuring element ⇒ remove small details in non-zero pixels
4. Closing
   - $f \cdot S = (f \oplus S) \ominus S$
   - Fills the holes smaller than structuring element ⇒ remove small details in zero pixels

Example1. Morphological Operations

```python
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

img=cv.imread('JohnHancocksSignature.png',cv.IMREAD_UNCHANGED)
t,bin_img=cv.threshold(img[:,:,3],0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
plt.imshow(bin_img,cmap='gray'), plt.xticks([]), plt.yticks([])
plt.show()

b=bin_img[235:420, 770:1040]
plt.imshow(b, cmap='gray'), plt.xticks([]), plt.yticks([])
plt.show()

# structuring element
se=np.uint8([[0,0,1,0,0],
             [0,1,1,1,0],
             [1,1,1,1,1],
             [0,1,1,1,0],
             [0,0,1,0,0]])

# dilation operation
b_dilation=cv.dilate(b, se, iterations=1)
plt.imshow(b_dilation, cmap='gray'), plt.xticks([]), plt.yticks([])
plt.show()

# erosion operation
b_erosion=cv.erode(b, se, iterations=1)
plt.imshow(b_erosion, cmap='gray'), plt.xticks([]), plt.yticks([])
plt.show()
```

Original Image

Original image cropped (Top)
Dilation result (Bottom left)
Erosion result (Bottom right)

Example1. Morphological Operations

```
1  import cv2 as cv
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  img=cv.imread('JohnHancocksSignature.png',cv.IMREAD_UNCHANGED)
6  t,bin_img=cv.threshold(img[:,:,3],0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
7  plt.imshow(bin_img,cmap='gray'), plt.xticks([]), plt.yticks([])
8  plt.show()
9
10 b=bin_img[235:420, 770:1040]
11 plt.imshow(b, cmap='gray'), plt.xticks([]), plt.yticks([])
12 plt.show()
13
14 # structuring element
15 se=np.uint8([[0,0,1,0,0],
16              [0,1,1,1,0],
17              [1,1,1,1,1],
18              [0,1,1,1,0],
19              [0,0,1,0,0]])
20
21 # dilation operation
22 b_dilation=cv.dilate(b, se, iterations=1)
23 plt.imshow(b_dilation, cmap='gray'), plt.xticks([]), plt.yticks([])
24 plt.show()
25
26 # erosion operation
27 b_erosion=cv.erode(b, se, iterations=1)
28 plt.imshow(b_erosion, cmap='gray'), plt.xticks([]), plt.yticks([])
29 plt.show()
```

- (L5) JohnHancocksSignature.png has 4 channels and signature is stored in channel 3
- (L6) Binarizes the signature image using Otsu binarization

- (L15) Defines the structuring element SE
  ⇒ Define the structuring element using a 2D array of 0's and 1's

0	0	1	0	0
0	1	1	1	0
1	1	1	1	1
0	1	1	1	0
0	0	1	0	0

se
(structuring element)

Example1. Morphological Operations

```
1  import cv2 as cv
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  img=cv.imread('JohnHancocksSignature.png',cv.IMREAD_UNCHANGED)
6  t,bin_img=cv.threshold(img[:,:,3],0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
7  plt.imshow(bin_img,cmap='gray'), plt.xticks([]), plt.yticks([])
8  plt.show()
9
10 b=bin_img[235:420, 770:1040]
11 plt.imshow(b, cmap='gray'), plt.xticks([]), plt.yticks([])
12 plt.show()
13
14 # structuring element
15 se=np.uint8([[0,0,1,0,0],
16              [0,1,1,1,0],
17              [1,1,1,1,1],
18              [0,1,1,1,0],
19              [0,0,1,0,0]])
20
21 # dilation operation
22 b_dilation=cv.dilate(b, se, iterations=1)
23 plt.imshow(b_dilation, cmap='gray'), plt.xticks([]), plt.yticks([])
24 plt.show()
25
26 # erosion operation
27 b_erosion=cv.erode(b, se, iterations=1)
28 plt.imshow(b_erosion, cmap='gray'), plt.xticks([]), plt.yticks([])
29 plt.show()
```

- (L22) cv.dilate(): dilates the binary image by se

◆ dilate()

```
void cv::dilate ( InputArray          src,
                  OutputArray         dst,
                  InputArray          kernel,
                  Point               anchor = Point(-1,-1),
                  int                 iterations = 1,
                  int                 borderType = BORDER_CONSTANT,
                  const Scalar &      borderValue = morphologyDefaultBorderValue()
                )
```

Python:

cv.dilate(src, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]]) -> dst

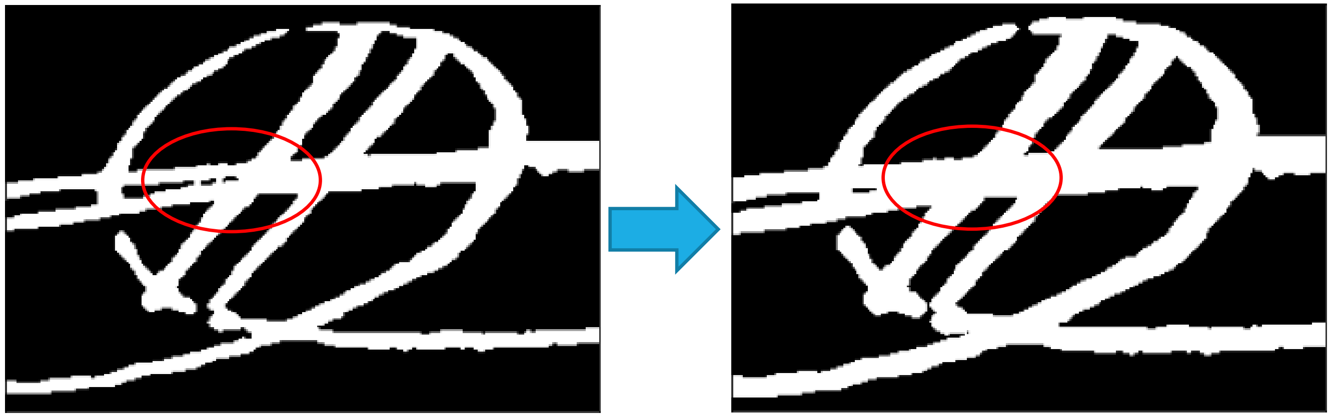
- kernel: structuring element. Kernels can be created using cv.getStructuringElement(morphShape, ksize[, anchor]) as well (morphShape: MORPH_RECT, MORPH_CROSS, MORPH_ELLIPSE)

- anchor: the position of the anchor point, If not specified, use the center point of kernel
- iterations: Number of times to apply dilation

Example1. Morphological Operations

```
1  import cv2 as cv
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  img=cv.imread('JohnHancocksSignature.png',cv.IMREAD_UNCHANGED)
6  t,bin_img=cv.threshold(img[:,:,3],0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
7  plt.imshow(bin_img,cmap='gray'), plt.xticks([]), plt.yticks([])
8  plt.show()
9
10 b=bin_img[235:420, 770:1040]
11 plt.imshow(b, cmap='gray'), plt.xticks([]), plt.yticks([])
12 plt.show()
13
14 # structuring element
15 se=np.uint8([[0,0,1,0,0],
16              [0,1,1,1,0],
17              [1,1,1,1,1],
18              [0,1,1,1,0],
19              [0,0,1,0,0]])
20
21 # dilation operation
22 b_dilation=cv.dilate(b, se, iterations=1)
23 plt.imshow(b_dilation, cmap='gray'), plt.xticks([]), plt.yticks([])
24 plt.show()
25
26 # erosion operation
27 b_erosion=cv.erode(b, se, iterations=1)
28 plt.imshow(b_erosion, cmap='gray'), plt.xticks([]), plt.yticks([])
29 plt.show()
```

- (L22) cv.dilate(): dilates the binary image by se



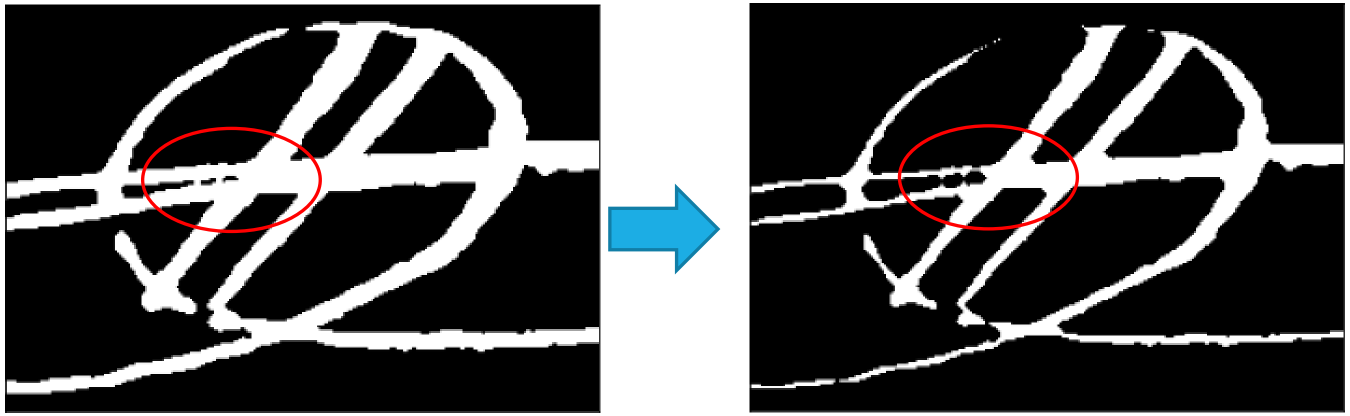
	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
78	0	0	0	0	0	0	0	0	0	255	255	255	255	255	255
79	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
80	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
81	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
82	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
83	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
84	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
85	255	0	0	0	0	0	0	0	0	0	0	0	0	0	0
86	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
87	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
88	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
89	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
78	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
79	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
80	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
81	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
82	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
83	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
84	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
85	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
86	255	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
87	255	255	255	0	0	0	0	0	0	0	0	0	0	0	0	0	0
88	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
89	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Example1. Morphological Operations

```
1   import cv2 as cv
2   import numpy as np
3   import matplotlib.pyplot as plt
4
5   img=cv.imread('JohnHancocksSignature.png',cv.IMREAD_UNCHANGED)
6   t,bin_img=cv.threshold(img[:,:,3],0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
7   plt.imshow(bin_img,cmap='gray'), plt.xticks([]), plt.yticks([])
8   plt.show()
9
10  b=bin_img[235:420, 770:1040]
11  plt.imshow(b, cmap='gray'), plt.xticks([]), plt.yticks([])
12  plt.show()
13
14  # structuring element
15  se=np.uint8([[0,0,1,0,0],
16               [0,1,1,1,0],
17               [1,1,1,1,1],
18               [0,1,1,1,0],
19               [0,0,1,0,0]])
20
21  # dilation operation
22  b_dilation=cv.dilate(b, se, iterations=1)
23  plt.imshow(b_dilation, cmap='gray'), plt.xticks([]), plt.yticks([])
24  plt.show()
25
26  # erosion operation
27  b_erosion=cv.erode(b, se, iterations=1)
28  plt.imshow(b_erosion, cmap='gray'), plt.xticks([]), plt.yticks([])
29  plt.show()
```

- (L27) cv.erode(): erodes the binary image by se



	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
78	0	0	0	0	0	0	0	0	0	255	255	255	255	255	255
79	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
80	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
81	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
82	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
83	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
84	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
85	255	0	0	0	0	0	0	0	0	0	0	0	0	0	0
86	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
87	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
88	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
89	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

2	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	255	255	255	255	255	255
0	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
55	255	255	255	255	255	255	255	255	255	255	255	255	0	0	0	0	0
55	255	255	255	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Example1. Morphological Operations

- (Question) Add the codes to do closing and opening operations (iterations=1)

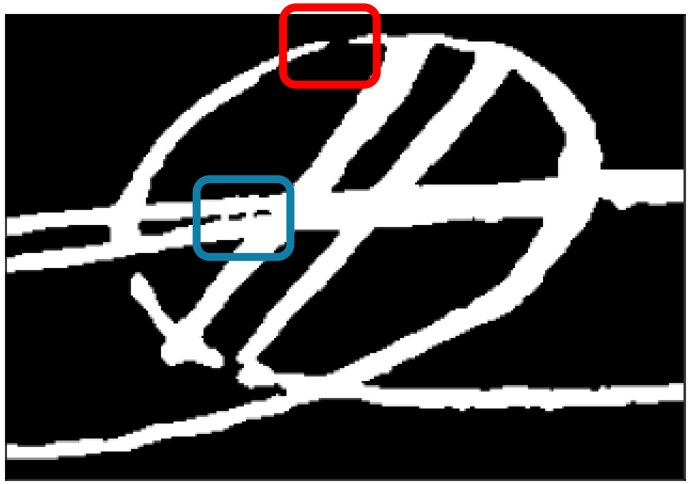
```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

img=cv.imread('JohnHancocksSignature.png',cv.IMREAD_UNCHANGED)
t,bin_img=cv.threshold(img[:,:,3],0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
plt.imshow(bin_img,cmap='gray'), plt.xticks([]), plt.yticks([])
plt.show()

b=bin_img[235:420, 770:1040]
plt.imshow(b, cmap='gray'), plt.xticks([]), plt.yticks([])
plt.show()

# structuring element
se=np.uint8([[0,0,1,0,0],
             [0,1,1,1,0],
             [1,1,1,1,1],
             [0,1,1,1,0],
             [0,0,1,0,0]])

# dilation operation
b_dilation=cv.dilate(b, se, iterations=1)
plt.imshow(b_dilation, cmap='gray'), plt.xticks([]), plt.yticks([])
plt.show()

# erosion operation
b_erosion=cv.erode(b, se, iterations=1)
plt.imshow(b_erosion, cmap='gray'), plt.xticks([]), plt.yticks([])
plt.show()
```

Example1. Morphological Operations

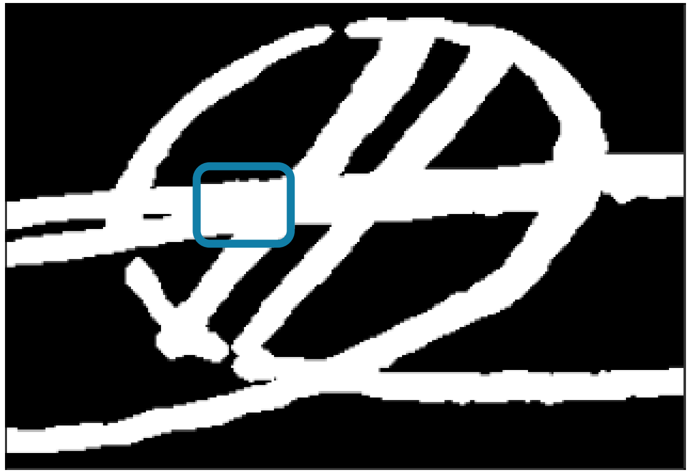
(Answer) Add the codes to do closing and opening operations (iterations=1)

```
# dilation operation
b_dilation=cv.dilate(b, se, iterations=1)
plt.imshow(b_dilation, cmap='gray'), plt.xticks([]), plt.yticks([])
plt.show()

# erosion operation
b_erosion=cv.erode(b, se, iterations=1)
plt.imshow(b_erosion, cmap='gray'), plt.xticks([]), plt.yticks([])
plt.show()

# Write closing and opening operations #
b_closing=cv.erode(cv.dilate(b, se, iterations=1), se, iterations=1)
plt.imshow(b_closing, cmap='gray'), plt.xticks([]), plt.yticks([])
plt.show()

b_opening=cv.dilate(cv.erode(b, se, iterations=1), se, iterations=1)
plt.imshow(b_opening, cmap='gray'), plt.xticks([]), plt.yticks([])
plt.show()
```

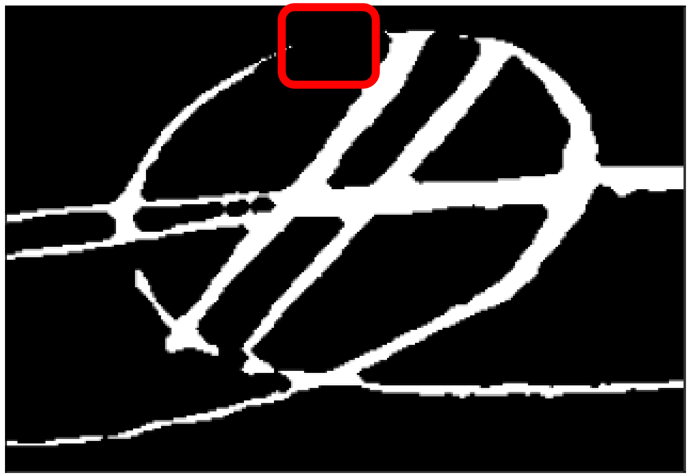
Example1. Morphological Operations



Original Image



Dilation result



Erosion result

- Closing and opening operations remove small noises from the input image, while leaving large areas
- Closing: dilation first fills small holes (having 0's) in the image. Then erosion shrinks the expanded area
- Opening: erosion first trims small extrusions (and isolated areas) in the input image. Then dilation expands the thinned area

Closing result

Opening result

Content

- Morphological Operation
- **2D Geometric Transformation**

2D Geometric Transformation

- Change the location of pixels while keeping other properties (e.g. intensity, filtering) of pixels
- A pixel at (y, x) transforms to (y', x') with the following functions:

$$y' = f_1(y, x), x' = f_2(y, x)$$

- Affine transformation: write f_1 and f_2 as

$$\begin{aligned} y' &= f_1(y, x) = ay + bx + c \\ x' &= f_2(y, x) = dy + ex + f \end{aligned} \quad \text{or} \quad [y' \quad x'] = [y \quad x] \begin{bmatrix} a & d \\ b & e \end{bmatrix} + [c \quad f]$$

$\Rightarrow$ When using homogeneous coordinates,

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

Have 6 DOF(Degrees of Freedom)!

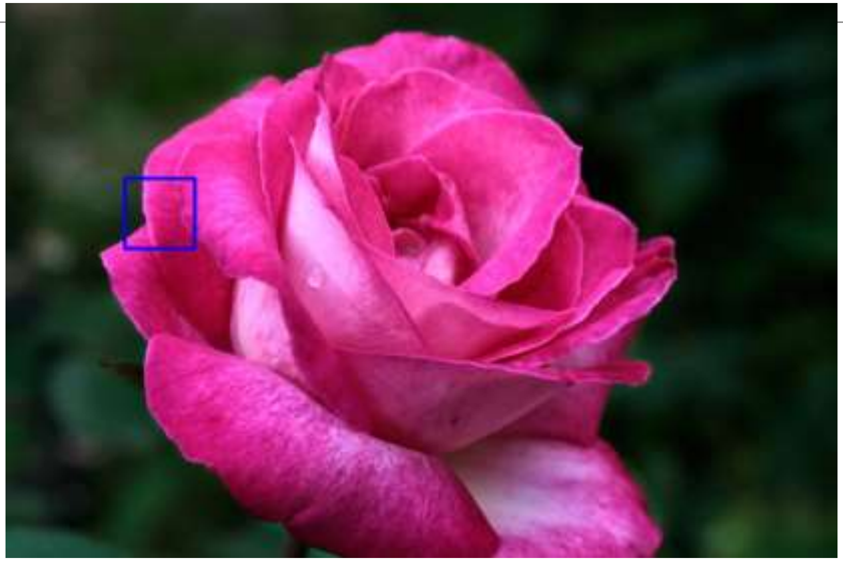
2D Geometric Transformation

- Rotation, scaling, translation, and shearing are also affine transformation as well

Transformation	Homogeneous Matrix $\dot{H}$	Geometric Meaning
Translation	$T(t_y, t_x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_y & t_x & 1 \end{bmatrix}$	Translate by t_y in y direction and t_x in x direction
Rotation	$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Rotation clockwise by θ
Scale	$S(s_y, s_x) = \begin{bmatrix} s_y & 0 & 0 \\ 0 & s_x & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Scale by s_y in y direction and s_x in x direction
Shear	$Sh_y(h_y) = \begin{bmatrix} 1 & 0 & 0 \\ h_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, Sh_x(h_x) = \begin{bmatrix} 1 & h_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Sh_y : Shear by h_y in y direction Sh_x : Shear by h_x in x direction

Example2-1. Image Scaling

```python
import cv2 as cv

img=cv.imread('rose.png')
patch=img[250:350,170:270,:]

img=cv.rectangle(img,(170,250),(270,350),(255,0,0),3)
patch1=cv.resize(patch,dsize=(0,0),fx=5,fy=5,
                 interpolation=cv.INTER_NEAREST)
patch2=cv.resize(patch,dsize=(0,0),fx=5,fy=5,
                 interpolation=cv.INTER_LINEAR)
patch3=cv.resize(patch,dsize=(0,0),fx=5,fy=5,
                 interpolation=cv.INTER_CUBIC)

cv.imshow('Original', img)
cv.imshow('Resize nearest', patch1)
cv.imshow('Resize bilinear', patch2)
cv.imshow('Resize bicubic', patch3)

cv.waitKey()
cv.destroyAllWindows()
```



<Original>

<Resize nearest>

<Resize bilinear>

<Resize bicubic>

Example2-1. Image Scaling

```
1  import cv2 as cv
2
3  img=cv.imread('rose.png')
4  patch=img[250:350,170:270,:]
5
6  img=cv.rectangle(img,(170,250),(270,350),(255,0,0),3)
7  patch1=cv.resize(patch,dsize=(0,0),fx=5,fy=5,
8                   interpolation=cv.INTER_NEAREST)
9  patch2=cv.resize(patch,dsize=(0,0),fx=5,fy=5,
10                  interpolation=cv.INTER_LINEAR)
11 patch3=cv.resize(patch,dsize=(0,0),fx=5,fy=5,
12                  interpolation=cv.INTER_CUBIC)
13
14 cv.imshow('Original', img)
15 cv.imshow('Resize nearest', patch1)
16 cv.imshow('Resize bilinear', patch2)
17 cv.imshow('Resize bicubic', patch3)
18
19 cv.waitKey()
20 cv.destroyAllWindows()
```

- (L7-L12) cv.resize(): scaling the input image
- Can set the size of the scaled image or the scaling factor
- Can set different interpolation method by specifying ‘interpolation’ parameter:
  cv.INTER_NEAREST: pick colors from nearest pixel
  cv.INTER_LINEAR: use bilinear interpolation
  cv.INTER_CUBIC: use bicubic interpolation
  → Aliasing artifacts are found in <Resize nearest>

<Resize nearest>

<Resize bilinear>

<Resize bicubic>

Affine Transformation in OpenCV

- Any affine transformation is represented by 6 parameters

$$y' = f_1(y,x) = ay + bx + c \qquad \text{or} \quad [y' \quad x'] = [y \quad x]\begin{bmatrix} a & d \\ b & e \end{bmatrix} + [c \quad f]$$
$$x' = f_2(y,x) = dy + ex + f$$

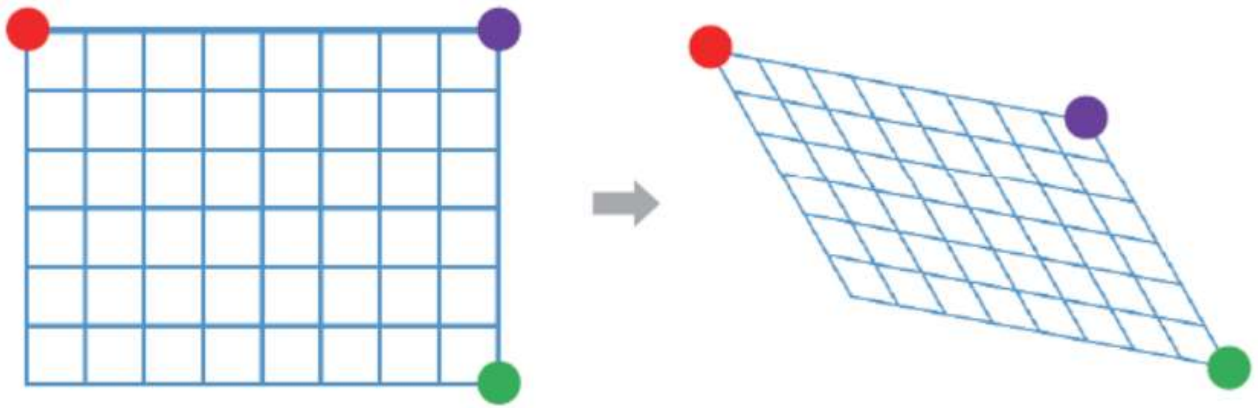
$\Rightarrow$ a, b, c, d, e, f defines one affine transformation

- How many points do we need to define an affine transformation?

$\Rightarrow$ We need 3 points to formulate 6 linear equations with 6 unknowns

- As affine transformation preserves parallel lines, it is sufficient to provide

3 corner points of a rectangle to get the unique transformation



Affine Transformation in OpenCV

- OpenCV provides a function to do affine transformation for the input image

1. Set up an affine transformation matrix M:
   - (Method 1) Write the matrix M as a 2x3 array

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} A & B & C \\ D & E & F \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$ → M=np.float32[[A,B,C], [D,E,F]]

(*Be careful for the order of parameters!*)

   - (Method 2) cv.getAffineTransform(): Compute a M from three points in the input image and their transformed points in the output image

2. Apply M to the input image using cv.warpAffine():

Affine Transformation in OpenCV

◆ warpAffine()

```
void cv::warpAffine ( InputArray        src,
                      OutputArray       dst,
                      InputArray        M,
                      Size              dsize,
                      int               flags = INTER_LINEAR ,
                      int               borderMode = BORDER_CONSTANT ,
                      const Scalar &    borderValue = Scalar()
                    )
```

Python:

```
cv.warpAffine( src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]] ) -> dst
```

- M: 2x3 affine transformation matrix computed using 1.
- dsize: the size of the output image (warning: the size should be given as (*width, height*), where width represents the number of columns, and height represents the number of rows of the image

`#include <opencv2/imgproc.hpp>`

Applies an affine transformation to an image.

The function warpAffine transforms the source image using the specified matrix:

$$\texttt{dst}(x, y) = \texttt{src}(M_{11}x + M_{12}y + M_{13}, M_{21}x + M_{22}y + M_{23})$$

when the flag **WARP_INVERSE_MAP** is set. Otherwise, the transformation is first inverted with **invertAffineTransform** and then put in the formula above instead of M. The function cannot operate in-place.

Example2-2. Translation

```
1  import cv2 as cv
2  import numpy as np
3
4  img=cv.imread('tekapo.bmp')
5  rows, cols,channels = img.shape
6
7  M = np.float32([[1,0,100], [0,1,50]])
8  dst = cv.warpAffine(img, M, (cols, rows))
9
10 cv.imshow('Original', img)
11 cv.imshow('Translation', dst)
12
13 cv.waitKey()
14 cv.destroyAllWindows()
```

<Original>

<Translation>

- (L7): define a translation matrix $M = \begin{bmatrix} 1 & 0 & 100 \\ 0 & 1 & 50 \end{bmatrix}$
- (L8): translation the input image by 100 along x-axis, and by 50 along y-axis
- (L8): if the pixel in the output image corresponding to outliers in the input image are colored as black (can be changed by using different borderMode and borderValue parameters)

Example2-3. Rotation

```python
import cv2 as cv

img=cv.imread('tekapo.bmp')
rows, cols,channels = img.shape

M=cv.getRotationMatrix2D(((cols-1)/2.0, (rows-1)/2.0),
                          30, 1)
dst = cv.warpAffine(img, M, (cols, rows))

cv.imshow('Original', img)
cv.imshow('Rotation', dst)

cv.waitKey()
cv.destroyAllWindows()
```

<Original>

<Rotation>

Example2-3. Rotation

```
1  import cv2 as cv
2
3  img=cv.imread('tekapo.bmp')
4  rows, cols,channels = img.shape
5
6  M=cv.getRotationMatrix2D(((cols-1)/2.0, (rows-1)/2.0),
7                               30, 1)
8  dst = cv.warpAffine(img, M, (cols, rows))
9
10 cv.imshow('Original', img)
11 cv.imshow('Rotation', dst)
12
13 cv.waitKey()
14 cv.destroyAllWindows()
```

- In OpenCV, a Matrix M for rotating by 30 degrees ccw,

$$M = \begin{bmatrix} \cos 30° & -\sin 30° \\ \sin 30° & \cos 30° \end{bmatrix}$$

- (L6) cv.getRotationMatrix2D(center, angle, scale): create a rotation matrix M with the adjustable center. The function calculates the following matrix:

$$\begin{bmatrix} \alpha & \beta & (1-\alpha)\cdot center.x - \beta \cdot center.y \\ -\beta & \alpha & \beta \cdot center.x + (1-\alpha)\cdot center.y \end{bmatrix}$$

$$\alpha = scale \cdot \cos\theta,$$
$$\beta = scale \cdot \sin\theta$$

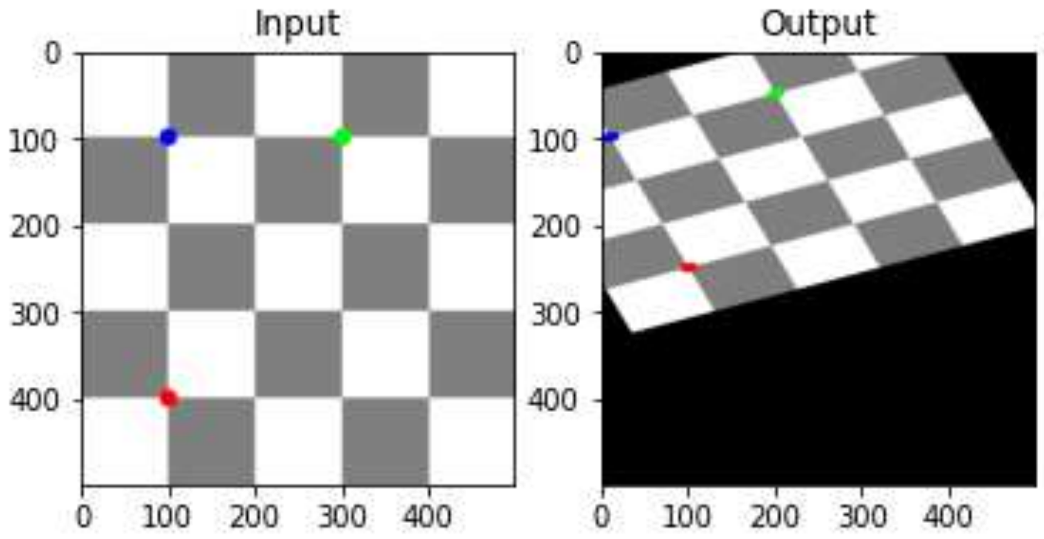
- (L6) M is an affine transformation matrix to represent a rotation of 30 degrees pivoted at the center point of the image

Example2-4. Affine Transformation

- In affine transformation, parallel lines goes to parallel lines

⇒ Rectangles can be transformed into parallelograms

```python
1  import cv2 as cv
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  img=cv.imread('checkerboardWithDots.png')
6  rows,cols,ch=img.shape
7
8  pts1=np.float32([[100,100],[300,100],[100,400]])
9  pts2=np.float32([[10,100],[200,50],[100,250]])
10
11 M = cv.getAffineTransform(pts1,pts2)
12 dst=cv.warpAffine(img,M,(cols,rows))
13
14 plt.subplot(121),plt.imshow(img),plt.title('Input')
15 plt.subplot(122),plt.imshow(dst),plt.title('Output')
16 plt.show()
```



- (L8-L9) Pick three points in the input image and their corresponding points in the output image to be used in computing affine transformation
- (L11) cv.getAffineTransform(): compute an 2x3 affine transformation matrix M which transform pts1 to pts2

Example2-5. Perspective Transformation

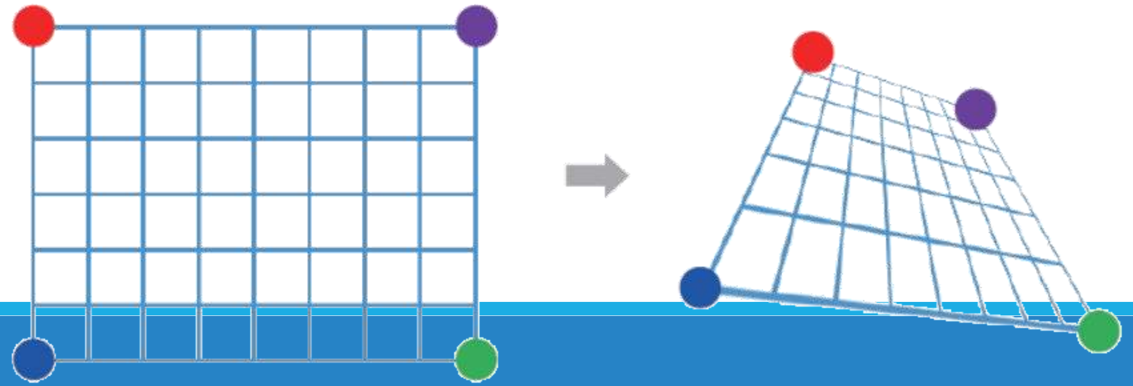
- Perspective transformation: determined by 8 parameters

$$[y' \quad x' \quad 1] = [y \quad x \quad 1]\begin{bmatrix} a & d & g \\ b & e & h \\ c & f & 1 \end{bmatrix}$$

$\Rightarrow$ We need 4 points to solve 8 linear equations of 8 unknowns (3 points must not be colinear)

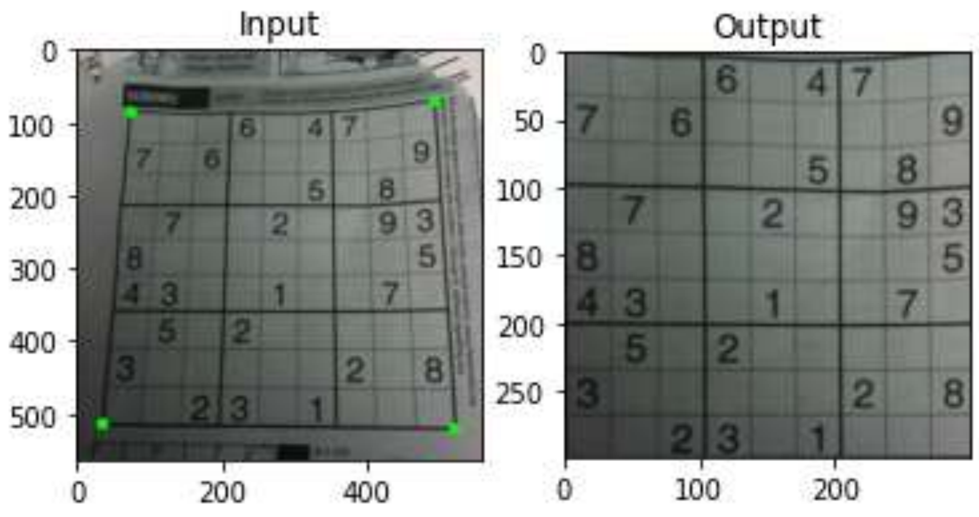
$\Rightarrow$ 4 points determine the unique perspective transformation

- In perspective transformation, parallel lines are no longer preserved
- In perspective transformation, straight lines are preserved



Example2-5. Perspective Transform

```
1  import cv2 as cv
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  img=cv.imread('sudoku.jpg')
6  rows,cols,ch=img.shape
7
8  pts1=np.float32([[75,86],[490,71],[36,513],[517,520]])
9  pts2=np.float32([[0,0],[300,0],[0,300],[300,300]])
10
11 M = cv.getPerspectiveTransform(pts1,pts2)
12 dst = cv.warpPerspective(img,M,(300,300))
13
14 upts1 = np.uint16(pts1)
15 for i in range(4):
16     cv.circle(img,(upts1[i,0],upts1[i,1]),8,(0,255,0),-1)
17
18
19 plt.subplot(121),plt.imshow(img),plt.title('Input')
20 plt.subplot(122),plt.imshow(dst),plt.title('Output')
21 plt.show()
```



- (L8-L9) Pick 4 points in the input image and their corresponding points in the output image to compute the perspective transformation matrix M
- (L11) cv.getPerspectiveTransform(): compute the perspective transformation matrix M to transform pts1 to pts2
- (L12) cv.warpPerspective(): apply perspective transformation