CHAPTER 08
기하학 처리

8.1 사상
8.2 크기 변경(확대/축소)
8.3 보간
8.4 평행이동
8.5 회전

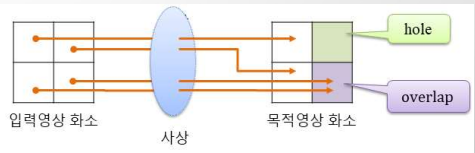
1

8.1 사상

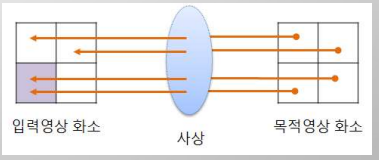
◆기하학적 처리의 기본
- ❖ 화소들의 배치 변경 →사상의 의미 이해

◆사상(mapping)
- ❖ 화소들의 배치를 변경할 때, 입력영상의 좌표에 해당하는 해당 목적영상의 좌표를 찾아서 화소 값을 옮기는 과정

◆**순방향 사상(Forward mapping)**
- ❖ 원본영상의 좌표를 중심으로 목적영상의 좌표를 계산하여 화소의 위치를 변환하는 방식
- ❖ 홀(hole)이나 오버랩(overlap)의 문제가 발생
  - ●입력 영상과 출력 영상의 크기가 다른 경우
  - ●입력 영상 화소와 출력 영상 화소가 1:1 사상이 되지 않는 경우



◆**역방향 사상(Backward mapping)**
- ❖ 목적영상의 좌표를 중심으로 역변환을 계산하여 해당하는 입력 영상의 좌표를 찾아서 화소값을 가져오는 방식
- ❖ 홀이나 오버랩은 발생하지 않음
- ❖ 입력영상의 한 화소를 목적영상의 여러 화소에서 사용하게 되면 결과 영상의 품질 저하



2

8.2 크기 변경 (확대/축소)

◆크기 변경(scaling)

❖ 입력영상의 가로와 세로로 크기를 변경해서 목적영상을 만드는 방법

❖ 입력영상보다 변경하고자 하는 영상의 크기가 커지면 확대, 작아지면 축소

◆크기 변경 수식

❖ 변경 비율 및 변경 크기 이용

$$x' = x \cdot ratio\ X$$
$$y' = y \cdot ratio\ Y$$

$$ratio\ X = \frac{dst_{width}}{org_{width}}, \quad ratio\ Y = \frac{dst_{height}}{org_{height}}$$

3

8.2 크기 변경 (확대/축소)

예제 8.2.1 영상 크기 변경 – 01.scaling.cpp

```
01  import numpy as np, cv2
02
03  def scaling(img, size):                                  # 크기 변경 함수1
04      dst = np.zeros(size[::-1], img.dtype)                # size와 shape는 원소 역순
05      ratioY, ratioX = np.divide(size[::-1], img.shape[:2])          # 비율 계산
06      y = np.arange(0, img.shape[0], 1)                    # 입력 영상 세로(y) 좌표 생성
07      x = np.arange(0, img.shape[1], 1)                    # 입력 영상 가로(x) 좌표 생성
08      y, x = np.meshgrid(y, x)                             # i, j 좌표에 대한 정방행렬 생성
09      i, j = np.int32(y * ratioY), np.int32(x * ratioX)    # 목적 영상 좌표
10      dst[i, j] = img[y, x]                                # 정방향 사상→목적 영상 좌표 계산
11      return dst
12
13  def scaling2(img, size):                                 # 크기 변경 함수2
14      dst = np.zeros(size[::-1], img.dtype)
15      ratioY, ratioX = np.divide(size[::-1], img.shape[:2])
16      for y in range(img.shape[0]):                        # 입력 영상 순회– 순방향 사상
17          for x in range(img.shape[1]):
18              i, j = int(y * ratioY), int(x * ratioX)      # 목적 영상의 y, x 좌표
19              dst[i, j] = img[y, x]
20      return dst
21
```

변경될 목적영상의 크기를 이용해서 크기 변경을 수행하는 함수

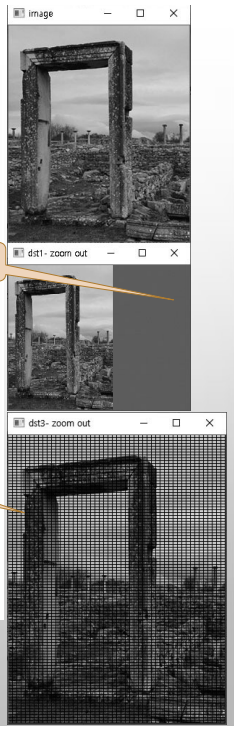
목적영상 좌표

원본영상 좌표

4

8.2 크기 변경 (확대/축소)

```
22  def time_check(func, image, size, title):              # 수행시간 체크 함수
23      start_time = time.perf_counter()
24      ret_img = func(image, size)
25      elapsed = (time.perf_counter() - start_time) * 1000
26      print(title, "수행시간 = %0.2f ms" % elapsed)
27      return ret_img
28
29  image = cv2.imread("images/scaling.jpg", cv2.IMREAD_GRAYSCALE)
30  if image is None: raise Exception("영상파일 읽기 에러")
31
32  dst1 = scaling(image, (150, 200))                      # 크기 변경- 축소
33  dst2 = scaling2(image, (150, 200))                     # 크기 변경- 축소
34  dst3 = time_check(scaling, image, (300,400), "[방법1]: 정방행렬 방식>")    # 확대
35  dst4 = time_check(scaling2, image, (300,400), "[방법2]: 반복문 방식>")     # 확대
36
37  cv2.imshow("image", image)
38  cv2.imshow("dst1- zoom out", dst1)
39  cv2.imshow("dst3- zoom out" , dst3)
40  cv2.resizeWindow("dst1- zoom out", 260, 200)           # 윈도우 크기 확장
41  cv2.waitKey(0)
```

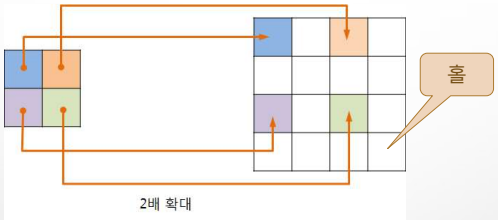


❖ 크기변경 in OpenCV
- `cv2.resize`(src, 원하는 크기, 원하는 크기 비율, 보간법)

5

8.3 보간

◆순방향 사상으로 영상 확대 - 홀이 많이 발생



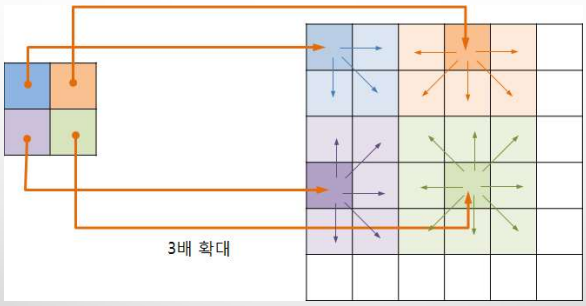
◆역방향 사상을 통해서 홀의 화소들을 입력영상에서 찾음
- ❖ 영상을 축소할 때에는 오버랩의 문제가 발생

◆보간법(Interpolation) 필요
- ❖ 목적영상에서 홀의 화소들을 채우고, 오버랩이 되지 않게 화소들을 배치하여 목적영상을 만드는 기법

6

8.3.1 최근접 이웃 보간법

◆목적영상을 만드는 과정에서 홀이 되어 할당 받지 못하는 화소들의 값을 찾을 때, 목적영상의 화소에 가장 가깝게 이웃한 입력영상의 화소값을 가져오는 방법



$$\begin{aligned} y' &= y \cdot ratio\ Y \\ x' &= x \cdot ratio\ X \end{aligned} \quad \Rightarrow \quad y = \frac{y'}{ratio\ Y},\ x = \frac{x'}{ratio\ X}$$

7

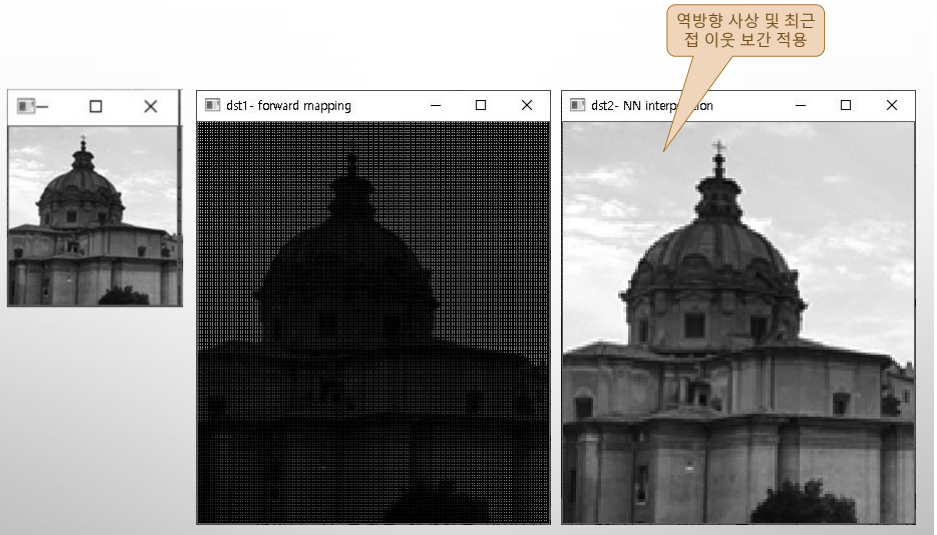
8.3.1 최근접 이웃 보간법

예제 8.3.1 크기변경 & 최근접 이웃 보간- 02.scaling_nearset.cpp

```
01  import numpy as np, cv2
02  from Common.interpolation import scaling     # interpolation 모듈의 scaling() 함수 임포트
03
04  def scaling_nearest(img, size):                          # 크기 변경 함수3
05      dst = np.zeros(size[::-1], img.dtype)                # 행렬과 크기는 원소가 역순
06      ratioY, ratioX = np.divide(size[::-1], img.shape[:2]) # 변경 크기 비율
07      i = np.arange(0, size[1], 1)                         # 목적 영상 세로(i) 좌표 생성
08      j = np.arange(0, size[0], 1)                         # 목적 영상 가로(j) 좌표 생성
09      i, j = np.meshgrid(i, j)
10      y, x = np.int32(i / ratioY), np.int32(j / ratioX)    # 입력 영상 좌표
11      dst[i, j] = img[y, x]                                # 역방향 사상 → 입력 영상 좌표 계산
12
13      return dst
14
15  image = cv2.imread("images/interpolation.jpg", cv2.IMREAD_GRAYSCALE)
16  if image is None: raise Exception("영상파일 읽기 에러")
17
18  dst1 = scaling(image, (350, 400))                        # 크기 변경- 기본
19  dst2 = scaling_nearest(image, (350, 400))                # 크기 변경- 최근접 이웃 보간
20
21  cv2.imshow("image", image)
22  cv2.imshow("dst1- forward mapping", dst1)                # 순방향 사상
23  cv2.imshow("dst2- NN interpolation", dst2)               # 최근접 이웃 보간
24  cv2.waitKey(0)
```

8

8.3.1 최근접 이웃 보간법



9

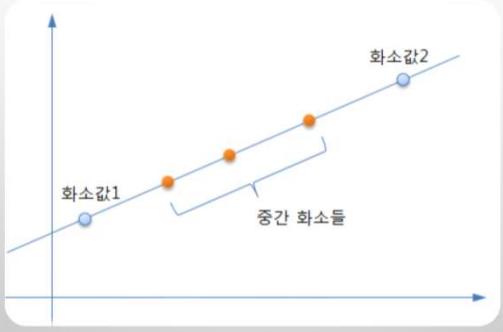
8.3.2 양선형 보간법

◆최근접 이웃 보간법

❖ 확대비율이 커지면, 모자이크 현상 혹은 경계부분에서 계단현상 발생

◆선형 보간으로 해결

❖ 직선의 선상에 위치한 중간 화소들의 값은 직선의 수식을 이용해서 쉽게 계산



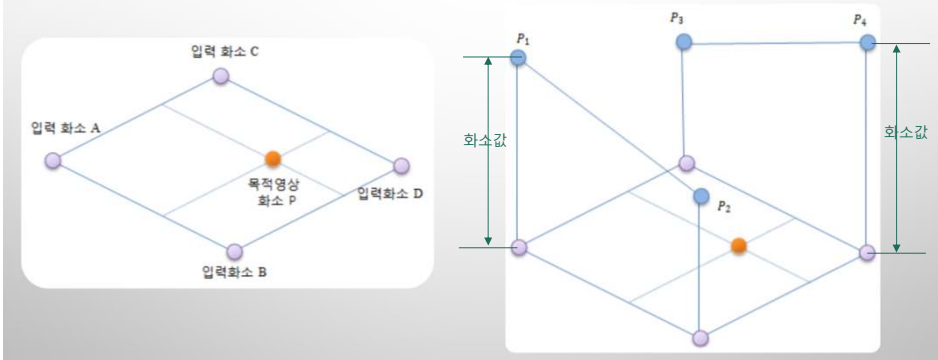
10

8.3.2 양선형 보간법

◆양선형 보간법 - 선형 보간을 두 번에 걸쳐서 수행하기에 붙여진 이름

❖ 목적영상 화소(P)를 역변환으로 계산하여 가장 가까운 위치에 있는 입력영상의 4개 화소(A, B, C, D) 값 가져옴

❖ 4개 화소를 두 개씩(AB, CD) 묶어서 두 화소를 화소값(P_1, P_2, P_3, P_4)으로 잇는 직선 구성



11

8.3.2 양선형 보간법

❖ 직선상에서 목적영상 화소의 좌표로 중간 위치 찾음

●중간 위치는 거리 비율(α , 1- α)로 찾음

❖ 중간 위치 화소값(M_1 , M_2) 계산

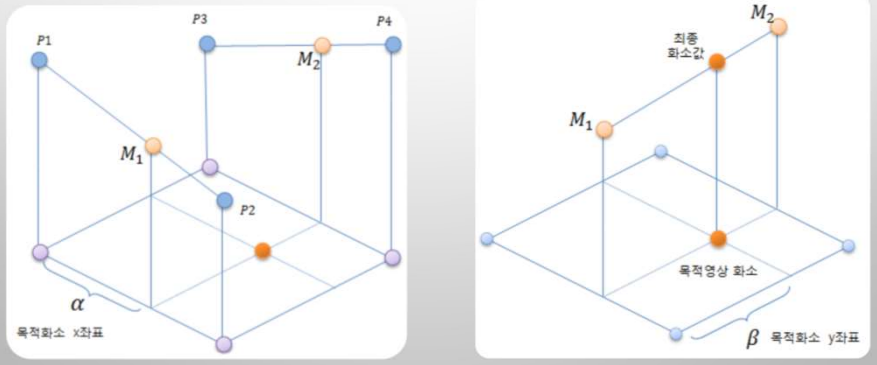
●입력 화소값과 거리 비율(α , 1- α)로 직선 수식 이용해 계산

$$M_1 = \alpha \cdot B + (1-\alpha) \cdot A = A + \alpha \cdot (B-A)$$
$$M_2 = \alpha \cdot D + (1-\alpha) \cdot C = C + \alpha \cdot (D-C)$$

❖ 두 개의 중간 화소값(M_1 , M_2)을 잇는 직선 다시 구성

❖ 두 개의 중간 화소값과 거리 비율(β , 1- β)로 직선 수식을 이용해서 최종 화소값 계산

$$P = \beta \cdot M_2 + (1-\beta) \cdot M_1 = M_1 + \beta \cdot (M_2 - M_1)$$



12

8.3.2 양선형 보간법

예제 8.3.2 크기변경 & 양선형 보간 – 03.scaling_bilinear.cpp

```
01  import numpy as np, cv2
02  from Common.interpolation import scaling_nearest     # 최근접 이웃 보간 함수 임포트
03
04  def bilinear_value(img, pt):                         # 단일 화소 양선형 보간 수행 함수
05      x, y = np.int32(pt)
06      if x >= img.shape[1]-1: x = x - 1                # 영상 범위 벗어남 처리
07      if y >= img.shape[0]-1: y = y - 1
08
09      P1, P2, P3, P4 = np.float32(img[y:y+2,x:x+2].flatten())  # 4개 화소-관심 영역으로 접근
10      ## 4개 화소 - 화소 직접 접근
11      # P1 = float(img[y, x] )                         # 좌상단 화소
12      # P2 = float(img[y + 0, x + 1])                  # 우상단 화소
13      # P3 = float(img[y + 1, x + 0])                  # 좌하단 화소
14      # P4 = float(img[y + 1, x + 1])                  # 우하단 화소
15
16      alpha, beta = pt[1] - y, pt[0] - x               # 거리 비율
17      M1 = P1 + alpha * (P3 - P1)                      # 1차 보간
18      M2 = P2 + alpha * (P4 - P2)
19      P  = M1 + beta * (M2 - M1)                       # 2차 보간
20      return np.clip(P, 0, 255)                        # 화소값 saturation 후 반환
21
22  def scaling_bilinear(img, size):                     # 양선형 보간
23      ratioY, ratioX = np.divide(size[::-1], img.shape[:2]) # 변경 크기 비율-행태와 크기는 역순
24
25      dst = [[ bilinear_value(img, (j/ratioX, i/ratioY))  # 리스트 생성
26               for j in range(size[0])]
27             for i in range(size[1])]
28      return np.array(dst, img.dtype)
```

단일 화소 양선형 보간

13

8.3.2 양선형 보간법

```
30  image = cv2.imread("images/interpolation.jpg", cv2.IMREAD_GRAYSCALE)
31  if image is None: raise Exception("영상파일 읽기 에러")
32
33  size = (350, 400)
34  dst1 = scaling_bilinear(image, size)                 # 크기 변경- 양선형 보간
35  dst2 = scaling_nearest(image, size)                  # 크기 변경- 최근접 이웃 보간
36  dst3 = cv2.resize(image, size, 0, 0, cv2.INTER_LINEAR)  # OpenCV 함수 – 양선형
37  dst4 = cv2.resize(image, size, 0, 0, cv2.INTER_NEAREST) # OpenCV 함수 – 최근접
38
39  cv2.imshow("image", image)
40  cv2.imshow("User_bilinear", dst1);
41  cv2.imshow("User_Nearest", dst2)
42  cv2.imshow("OpenCV_bilinear", dst3);
43  cv2.imshow("OpenCV_Nearest", dst4)
44  cv2.waitKey(0)
```

❖ OpenCV 보간 옵션

〈표 8.3.1〉 보간 방법에 대한 flag 옵션

옵션 상수	값	설명
INTER_NEAREST	0	최근접 이웃 보간
INTER_LINEAR	1	양선형 보간 (기본값)
INTER_CUBIC	2	바이큐빅 보간 – 4x4 이웃 화소 이용
INTER_AREA	3	픽셀 영역의 관계로 리샘플링
INTER_LANCZOS4	4	Lanczos 보간 – 8x8 이웃 화소 이용

14

8.3.2 양선형 보간법

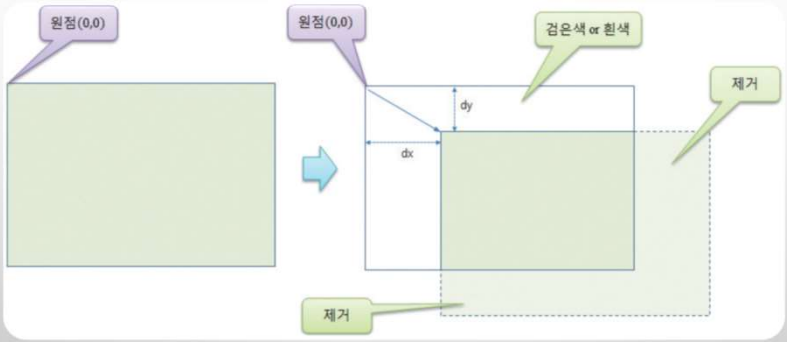
◆실행결과

15

8.4 평행이동

❖ 영상의 원점을 기준으로 모든 화소를 동일하게 가로방향과 세로 방향으로 옮기는 것

❖ 가로 방향으로 dx만큼, 세로 방향으로 dy만큼 전체 영상의 모든 화소 이동한 예



순방향 사상

$$x' = x + dx$$
$$y' = y + dy$$

역방향사상

$$x = x' - dx$$
$$y = y' - dy$$

16

8.4 평행이동

예제 8.4.1 영상 평행이동 – 04.translation.cpp

```
01  import numpy as np, cv2
02
03  def contain(p, shape):                          # 좌표(y,x)가 범위내 인지 검사
04      return 0<= p[0] < shape[0] and 0<= p[1] < shape[1]
05
06  def translate(img, pt):
07      dst = np.zeros(img.shape, img.dtype)        # 목적 영상 생성
08      for i in range(img.shape[0]):               # 목적 영상 순회– 역방향 사상
09          for j in range(img.shape[1]):
10              x, y = np.subtract((j, i) , pt)     # 좌표는 가로, 세로 순서
11              if contain((y, x), img.shape):      # 영상 범위 확인
12                  dst[i, j] = img[y, x]           # 행렬은 행, 열 순서
13      return dst
14
15  image = cv2.imread("images/translate.jpg", cv2.IMREAD_GRAYSCALE)
16  if image is None: raise Exception("영상파일 읽기 에러")
17
18  dst1 = translate(image, (30, 80))               # x=30, y=80 으로 평행이동
19  dst2 = translate(image, (-70, -50))
20
21  cv2.imshow("image", image)
22  cv2.imshow("dst1: transted to (30, 80)", dst1);
23  cv2.imshow("dst2: transted to (-70, -50)", dst2);
24  cv2.waitKey(0)
```

입력영상 좌표 계산

계산 화소가 입력영상 범위내에 있어야 됨

17

8.4 평행이동

◆실행결과

입력영상 범위 벗어나면 검은색 처리

18

8.5 회전

❖ 입력영상의 모든 화소를 영상의 원점을 기준으로 원하는 각도만큼 모든 화소에 대해서 회전 변환을 시키는 것

❖ 회전 변환 수식

순방향 사상

$$x' = x \cdot \cos\theta - y \cdot \sin\theta$$
$$y' = x \cdot \sin\theta + y \cdot \cos\theta$$

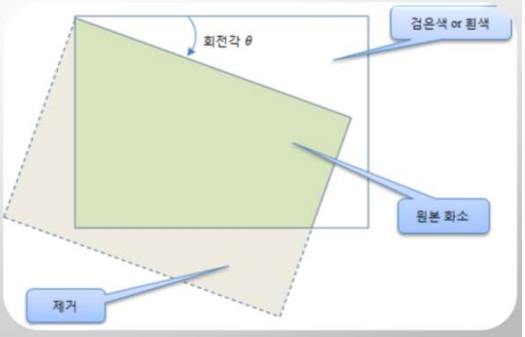
역방향사상

$$x = x' \cdot \cos\theta + y' \cdot \sin\theta$$
$$y = -x' \cdot \sin\theta + y' \cdot \cos\theta$$

❖ 원점으로부터 시계 방향으로 정해진 각도만큼 회전된 영상 생성

✓직교 좌표계에서 회전 변환은 반시계 방향으로 적용

✓영상 좌표계에서는 y 좌표가 하단으로 내려갈수록 증가하기 때문에 시계방향 회전



19

8.5 회전

◆특정 좌표에서(center X, center Y) 회전하는 경우

❖ 회전의 기준점으로 영상을 이동시킨 후, 회전 수행

❖ 다시 원점 좌표로 이동

$$x = (x' - center\ X)\cos\theta + (y' - center\ Y)\sin\theta + center\ X$$
$$y = -(x' - center\ X)\sin\theta + (y' - center\ Y)\cos\theta + center\ Y$$

기준점 이동

원점 재이동

20

8.5 회전

예제 8.5.1 영상 회전 – 05.rotation.py

```
01  import numpy as np, math, cv2
02  from Common.interpolation import bilinear_value      # 양선형 보간 함수 임포트
03  from Common.functions import contain                 # 사각형으로 범위 확인 함수
04
05  def rotate(img, degree):                             # 원점 기준 회전 변환 함수
06      dst = np.zeros(img.shape[:2], img.dtype)         # 목적 영상 생성
07      radian = (degree/180) * np.pi                    # 회전 각도– 라디언
        sin, cos = np.sin(radian), np.cos(radian)        # 사인, 코사인 값 미리 계산
10      for i in range(img.shape[0]):                    # 목적 영상 순회– 역방향 사상
11          for j in range(img.shape[1]):
                y = -j * sin + i * cos
                x = j * cos + i * sin                    # 회선 변환 수식
14              if contain((y, x), img.shape):           # 입력 영상의 범위 확인
15                  dst[i, j] = bilinear_value(img, [x, y])      # 화소값 양선형 보간
16      return dst
17
```

반복문 내에서 계산시 속도 저하

역방향 사상 수식

21

8.5 회전

회전 기준점

```
18  def rotate_pt(img, degree, pt):                      # pt 기준 회전 변환 함수
19      dst = np.zeros(img.shape[:2], img.dtype)         # 목적 영상 생성
20      radian = (degree/180) * np.pi                    # 회전 각도– 라디언
21      sin, cos = math.sin(radian), math.cos(radian)    # 사인, 코사인 값 미리 계산
22
23      for i in range(img.shape[0]):                    # 목적 영상 순회– 역방향 사상
24          for j in range(img.shape[1]):
                jj, ii = np.subtract((j, i), pt)         # 중심 좌표로 평행이동
                y = -jj * sin + ii * cos                 # 회선 변환 수식
27              x = jj * cos + ii * sin
                x, y = np.add((x, y), pt)                # 중심 좌표로 평행이동
29              if contain((y, x), img.shape):           # 입력 영상의 범위 확인
30                  dst[i, j] = bilinear_value(img, (x, y))      # 화소값 양선형 보간
31      return dst
32
33  image = cv2.imread("images/rotate.jpg", cv2.IMREAD_GRAYSCALE)
34  if image is None: raise Exception("영상파일 읽기 에러")
35
36  center = np.divmod(image.shape[::-1], 2)[0]          # 영상 크기로 중심 좌표 계산
37  dst1 = rotate(image, 20)                             # 원점 기준 회전 변환
38  dst2 = rotate_pt(image, 20, center)                  # center 기준 회전 변환
39
40  cv2.imshow("image", image)
41  cv2.imshow("dst1 : rotated on (0, 0)", dst1);
42  cv2.imshow("dst2 : rotated on center point", dst2);
43  cv2.waitKey(0)
```

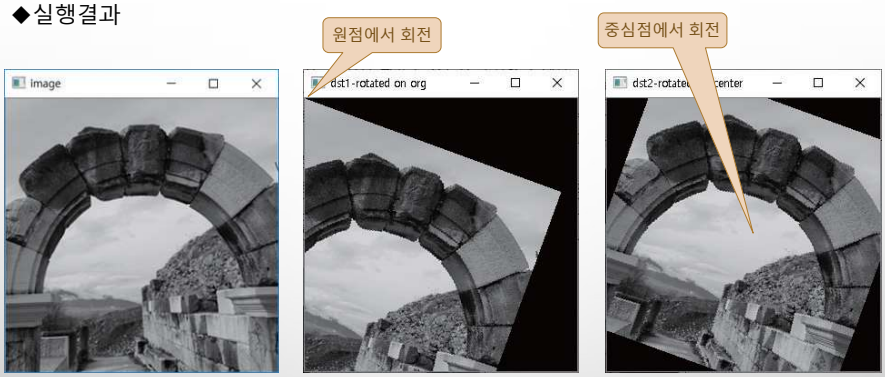
회전 기준점으로 평행이동

회전후 역 평행이동

22

8.5 회전

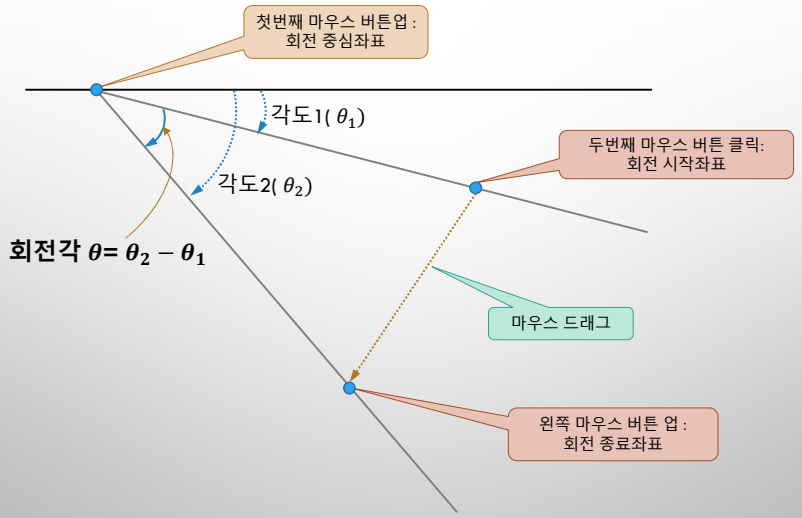
◆실행결과



23

8.5 회전 – 심화예제

◆심화예제 – 기준점에서 마우스 드래그로 회전 수행하기



24

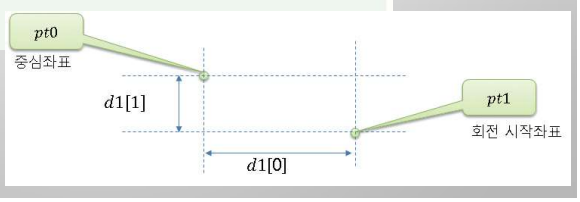
8.5 회전 – 심화예제

심화예제 8.5.2 마우스 드래그로 영상 회전하기 – 06.rotation2.py

```
01  import numpy as np, cv2
02  from Common.interpolation import rotate_pt          # 좌표 기준 영상 회전 함수 임포트
03
04  def calc_angle(pt):                                 # 3개 좌표로 각도 계산 함수
05      d1 = np.subtract(pts[1], pts[0]).astype(float)  # 두 좌표간 차분 계산
06      d2 = np.subtract(pts[2], pts[0]).astype(float)
07      angle1 = cv2.fastAtan2(d1[1], d1[0])            # 차분으로 각도 계산
08      angle2 = cv2.fastAtan2(d2[1], d2[0])
09      return (angle2 - angle1)                        # 두 각도 간의 차분
10
11  def draw_point(x, y):                               # 좌표 저장 및 그리기
12      pts.append([x,y])
13      print("좌표:", len(pts), [x,y])                 # 클릭 좌표 표시
14      cv2.circle(tmp, (x, y), 2, 255, 2)              # 중심 좌표 표시
15      cv2.imshow("image", tmp)
16
```

fastAtan2() 함수의 인수로 실수 자료형 필요

복사된 영상에 점(작은원) 그리기



25

8.5 회전 – 심화예제

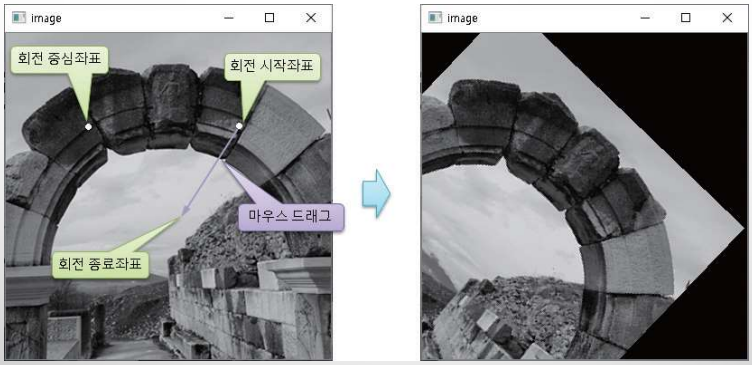
함수내부에서 벼수값 유지 위해 전역 변수 선수

```
17  def onMouse(event, x, y, flags, param):             # 마우스 콜백 함수
18      global tmp, pts
19      if (event == cv2.EVENT_LBUTTONUP and len(pts) == 0): draw_point(x, y)
20      if (event == cv2.EVENT_LBUTTONDOWN and len(pts) == 1): draw_point(x, y)
21      if (event == cv2.EVENT_LBUTTONUP and len(pts) == 2): draw_point(x, y)
22
23      if len(pts) == 3:
24          angle = calc_angle(pts)                     # 회전각 계산
25          print("회전각: %3.2f" % angle)
26          dst = rotate_pt(image, angle, pts[0])       # 저자 구현 회전 수행
27          cv2.imshow("image", dst)
28          tmp = np.copy(image)                        # 임시 행렬 초기화
29          pts = []                                    # 클릭 좌표 초기화
30
31  image = cv2.imread("images/rotate.jpg", cv2.IMREAD_GRAYSCALE)
32  if image is None: raise Exception("영상파일 읽기 에러")
33  tmp = np.copy(image)
34  pts = []
35
36  cv2.imshow("image", image)
37  cv2.setMouseCallback("image", onMouse, 0)
38  cv2.waitKey(0)
```

26

8.5 회전 – 심화예제

◆실행결과




```
Run: 06.rotate2
C:\Python\python.exe D:/source/chap08/06.rotate2.py
좌표: 1 [101, 103]
좌표: 2 [253, 105]
좌표: 3 [195, 207]
회전각 : 47.13
```

27

단원 요약

◆사상(mapping)은 화소들의 배치를 변경할 때, 입력영상의 좌표가 새롭게 배치될 해당 목적영상의 좌표를 찾아서 화소값을 옮기는 과정을 말한다. 순방향 사상(forward mapping)과 역방향 사상(reverse mapping)의 두 가지 방식이 있다.

◆사상의 과정에서 홀(hole)과 오버랩(overlap)이 발생할 수 있다. 홀은 입력영상의 좌표들로 목적영상의 좌표를 만드는 과정에서 사상되지 않은 화소이다. 오버랩은 원본 영상의 여러 화소가 목적영상의 한 화소로 사상되는 것을 말한다.

◆목적영상에서 홀의 화소들을 채우고, 오버랩이 되지 않게 화소들을 배치하여 목적영상을 만드는 기법을 보간법(interpolation)이라 하며, 그 종류에는 최근접 이웃 보간법, 양선형 보간법, 3차 회선 보간법 등 다양한 방법이 있다.

◆최근접 이웃 보간법은 목적영상을 만드는 과정에서 홀이 되어 할당 받지 못하는 화소들의 값을 찾을 때, 목적영상의 화소에 가장 가깝게 이웃한 입력영상의 화소값을 가져오는 방법이다. 양선형 보간법은 선형 보간을 두 번에 걸쳐서 수행하기에 붙여진 이름이다.

◆OpneCV에서는 cv::resize(), cv::remap(), cv::warpAffine(), cv::warpPerspective() 등과 같이 영상을 변환하는 함수에서 보간을 위한 flag 옵션을 제공한다. 대표적으로 'INTER_NEAREST'는 최근접 이웃 보간이며, 'INTER_LINEAR'는 양선형 보간이며, 'INTER_CUBIC'는 바이큐빅 보간이다.

28