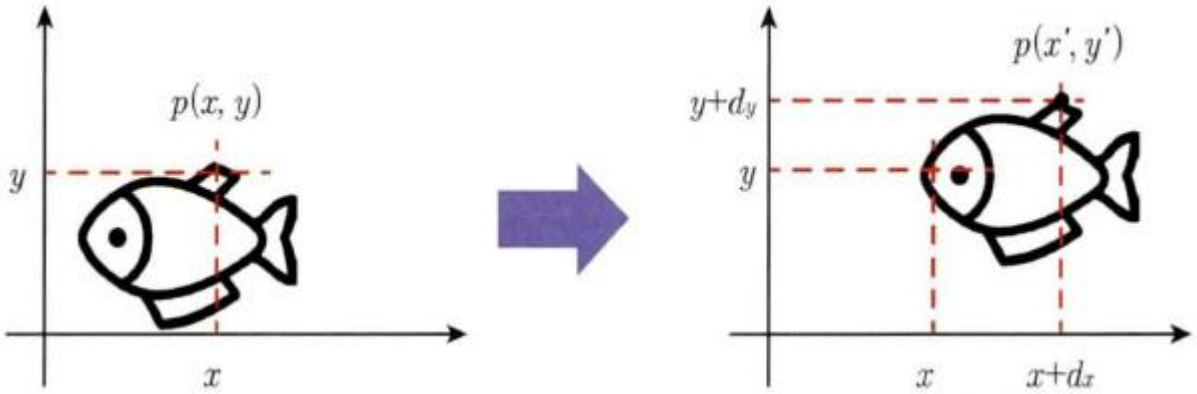
CH. 8. 기하학적 변환

Section 01 이동, 확대/축소, 회전

❑ 이동

- 2차원 공간에서 물체를 다른 곳으로 이동시키려면 원래 있던 좌표에 이동시키려는 거리만큼 더해서 이동할 새로운 좌표를 구하면 된다.



- 위 그림은 물고기 그림을 오른쪽 위로 이동하는 모습을 표현하고 있다.
- 이 그림에서 물고기의 어떤 점 p(x, y)를 dx와 dy만큼 옮기면 새로운 위치의 좌표 p(x', y')을 구할 수 있다. 이것을 수식으로 작성하면 아래와 같다.

$$x' = x + d_x$$

$$y' = y + d_y$$

Section 01 이동, 확대/축소, 회전

❑ 이동

- 위 방정식을 행렬식으로 바꾸어 표현하면 아래와 같다.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- 위의 행렬식을 아래와 같이 풀어서 표현하면 원래의 방정식과 같다는 것을 알 수 있다.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x + d_x \\ y + d_y \end{bmatrix} = \begin{bmatrix} 1x + 0y + 1d_x \\ 0x + 1y + 1d_y \end{bmatrix}$$

- 행렬식 중에서도 x, y는 이미 원본 이미지의 좌표 값으로 제공되므로 2 x 3 변환행렬만 전달하면 연산이 가능하다.

Section 01 이동, 확대/축소, 회전

❑ 이동

- OpenCV는 2 X3 **행렬로 영상의 좌표를 변환시켜 주는 함수를 다음과 같이 제공한다**.
- `dst = cv2.warpAffine(src, mtrx, dsize [, dst , flags , borderMode, borderValuel)`
  - src : **원본 영상**, NumPy **배열**
  - mtrx : 2 x 3 **변환행렬**, NumPy **배열**, dtype = float32
  - dsize : **결과 이미지 크기**, tuple(width, height)
  - flags : **보간법 알고리즘 선택 플래그**
    - cv2.INTER_LINEAR : **기본값, 인접한 4개 픽셀 값에 거리 가중치 사용**
    - cv2.INTER_NEAREST : **가장 가까운 픽셀 값 사용**
    - cv2.INTER_AREA : **픽셀 영역 관계를 이용한 재샘플링**
    - cv2.INTER_CUBIC : **인접한 16개 픽셀 값에 거리 가중치 사용**
    - cv2.INTER_LANCZOS4 : **인접한 8개 픽셀을 이용한 란초의 알고리즘**

Section 01 이동, 확대/축소, 회전

❑ 이동

- OpenCV는 2 X3 **행렬로 영상의 좌표를 변환시켜 주는 함수를 다음과 같이 제공한다**.
- `dst = cv2.warpAffine(src, mtrx, dsize [, dst , flags , borderMode, borderValuel)`
  - borderMode : **외곽 영역 보정 플래그**
    - cv2.BOROER_CONSTANT : **고정 색상 값**(999 | 12345 | 999)
    - cv2.BORDER_REPLICATE : **가장 자리 복제**(111 | 12345 | 555)
    - cv2.BORDER_WRAP : **반복**(345 | 12345 | 123)
    - cv2.BORDER_REFLECT: **반사**(321 | 12345 | 543)
  - borderValue : cv2.BORDER_CONSTANT**의 경우 사용할 색상 값**(**기본 값** = 0)
  - dst : **결과 이미지**, NumPy **배열**
- cv2.warpAffine() **함수는** src **영상을** mtrx **행렬에 따라 변환해서** dsize **크기로 만들어서 반환한다**.
- **그 뿐만 아니라 변환에 대부분 나타나는 픽셀 탈락 현상을 보정해주는 보간법 알고리즘과 경계 부분의 보정 방법도 선택할 수 있다**.

Section 01 이동, 확대/축소, 회전

❑ 이동

- **예제는** cv2.warpAffine( ) **함수와 변환행렬을 이용해서 영상을 이동 변환하는 예제이다.**
- **이동에 사용한 변환행렬은 앞서 설명한** $\begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \end{bmatrix}$ **이다.**

```
1   import cv2
2   import numpy as np
3
4   img = cv2.imread('./img/fish.jpg')
5   rows,cols = img.shape[0:2]  # 영상의 크기
6
7   dx, dy = 100, 50          # 이동할 픽셀 거리
8
9   mtrx = np.float32([[1, 0, dx], [0, 1, dy]])
10
11  dst = cv2.warpAffine(img, mtrx, (cols+dx, rows+dy))
12
```

Section 01 이동, 확대/축소, 회전

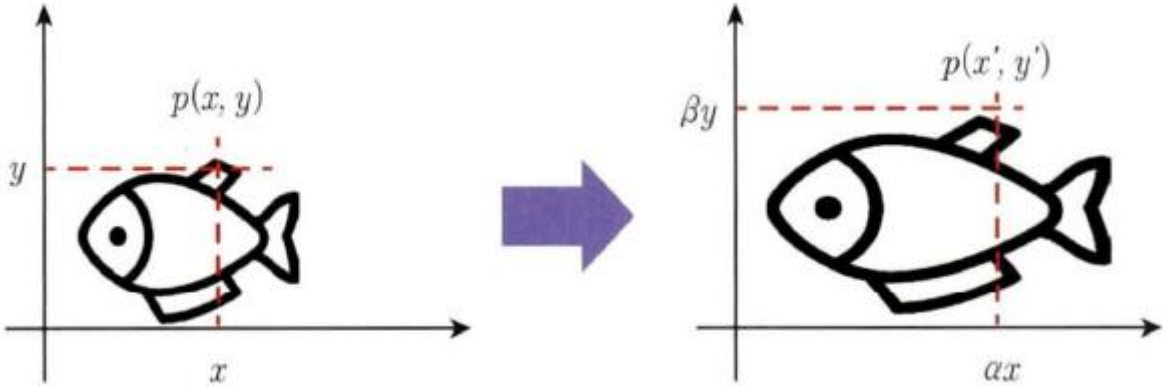
❑ 이동

- 예제는 cv2.warpAffine( ) 함수와 변환행렬을 이용해서 영상을 이동 변환하는 예제이다.
- 이동에 사용한 변환행렬은 앞서 설명한 $\begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \end{bmatrix}$ 이다.

```
13  dst2 = cv2.warpAffine(img, mtrx, (cols+dx, rows+dy), None, \
                    cv2.INTER_LINEAR, cv2.BORDER_CONSTANT, (255,0,0) )
14
15  dst3 = cv2.warpAffine(img, mtrx, (cols+dx, rows+dy), None, \
                        cv2.INTER_LINEAR, cv2.BORDER_REFLECT)
16
17  cv2.imshow('original', img)
18  cv2.imshow('trans',dst)
19  cv2.imshow('BORDER_CONSTATNT', dst2)
20  cv2.imshow('BORDER_FEFLECT', dst3)
21  cv2.waitKey(0)
22  cv2.destroyAllWindows()
```

Section 01 이동, 확대/축소, 회전

❑ 확대/축소



- 영상을 확대 또는 축소하려면 원래 있던 좌표에 원하는 비율만큼 곱해서 새로운 좌표를 구할 수 있다.
- 이 때 확대 /축소 비율을 가로와 세로 방향으로 각각 α와 β라고 하면 변환행렬은 아래와 같다.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Section 01 이동, 확대/축소, 회전

❑ 확대/축소

- 확대 혹은 축소를 하려면 2 x 2 행렬로도 충분히 표현이 가능한데 굳이 마지막 열에 0으로 채워진 열을 추가해서 2 x 3 행렬로 표현한 이유는 cv2.warpAffine( ) 함수와 이동 변환 때문이다.
- 앞서 다룬 이동을 위한 행렬식은 2 x 3 행렬로 표현해야 하므로 여러 가지 기하학적 변환을 지원해야 하는 cv2.warpAffine() 함수는 2 x 3 행렬이 아니면 오류를 발생한다.
- 행렬의 마지막 열에 dx, dy에 해당하는 값을 지정하면 확대와 축소 뿐만 아니라 이동도 가능하다.

Section 01 이동, 확대/축소, 회전

❑ 확대/축소

- 다음 예제는 변환행렬을 이용해서 0.5배 축소와 2배 확대를 하는 예제이다.

```
1   import cv2
2   import numpy as np
3
4   img = cv2.imread('./img/fish.jpg')
5   height, width = img.shape[:2]
6
7   m_small = np.float32([[0.5, 0, 0], [0, 0.5,0]])
8   m_big = np.float32([[2, 0, 0], [0, 2, 0]])
9
10  dst1 = cv2.warpAffine(img, m_small, (int(height*0.5), int(width*0.5)))
11  dst2 = cv2.warpAffine(img, m_big, (int(height*2), int(width*2)))
12
```

Section 01 이동, 확대/축소, 회전

❑ 확대/축소

- 다음 예제는 변환행렬을 이용해서 0.5배 축소와 2배 확대를 하는 예제이다.

```
13  dst3 = cv2.warpAffine(img, m_small, (int(height*0.5), int(width*0.5)), \
                    None, cv2.INTER_AREA)
14  dst4 = cv2.warpAffine(img, m_big, (int(height*2), int(width*2)), \
                    None, cv2.INTER_CUBIC)
15
16  cv2.imshow("original", img)
17  cv2.imshow("small", dst1)
18  cv2.imshow("big", dst2)
19  cv2.imshow("small INTER_AREA", dst3)
20  cv2.imshow("big INTER_CUBIC", dst4)
21  cv2.waitKey(0)
22  cv2.destroyAllWindows()
```

Section 01 이동, 확대/축소, 회전

❑ 확대/축소

- **보간법 알고리즘으로는 축소에는** cv2.INTER_AREA**가 효과적이고**, **확대에는** cv2.INTER_CUBIC**과** cv2.INTER_LINEAR**가 효과적인 것으로 알려져 있다**.
- OpenCV**는 변환행렬을 작성하지 않고도 확대와 축소 기능을 사용할 수 있게** cv2.resize() **함수를 별도로 제공한다**.
- `dst = cv2.warpAffine(src, mtrx, dsize [, dst , flags , borderMode, borderValuel)`
  - src : **입력 영상**, NumPy **배열**
  - dsize : **출력 영상 크기(확대 /축소 목표 크기)**, **생략하면** fx, fy**를 적용**
    - (width, height)
  - fx, fy : **크기 배율**, **생략하면** dsize**를 적용**
  - interpolation : **보간법 알고리즘 선택 플래그(** cv2.warpAffine( )**과 동일)**
  - dst : **결과 영상**, NumPy **배열**
- **만약** dsize**와** fx, fy **모두 값을 전달하면** dsize**만 적용한다**.

Section 01 이동, 확대/축소, 회전

❑ 확대/축소

- **다음 예제는** resize() **함수를 이용해서** 0.5**배 축소와** 2**배 확대를 하는 예제이다**.

```
1   import cv2
2   import numpy as np
3
4   img = cv2.imread('./img/fish.jpg')
5   height, width = img.shape[:2]
6
7   dst1 = cv2.resize(img, (int(width*0.5), int(height*0.5)), interpolation=cv2.INTER_AREA)
8
9   dst2 = cv2.resize(img, None,  None, 2, 2, cv2.INTER_CUBIC)
10
```

Section 01 이동, 확대/축소, 회전

❑ 확대/축소

- **다음 예제는** resize() **함수를 이용해서** 0.5**배 축소와** 2**배 확대를 하는 예제이다**.

```
11  cv2.imshow("original", img)
12  cv2.imshow("small", dst1)
13  cv2.imshow("big", dst2)
14  cv2.waitKey(0)
15  cv2.destroyAllWindows()
```

- cv2.resize() **함수가 변환행렬을 이용하는 코드보다 사용하기 쉽고 간결한 것을 알 수 있다**.

Section 01 이동, 확대/축소, 회전

❑ 회전

- **영상을 회전하려면 삼각함수를 써야 한다.**
- **회전을 위한 변환행렬 생성은 다소 까다로운 데다가 회전 축까지 반영하려면 일이 조금 복잡해진다.**
- OpenCV**는 개발자가 복잡한 계산을 하지 않고도 변환행렬을 생성할 수 있게 아래와 같은 함수를 제공한다.**
- `mtrx = cV2.getRotationMatrix2D(center, angle, scale)`
  - center : **회전 축 중심 좌표**, **튜플**(x, y)
  - angle : **회전 각도**, 60**진법**
  - scale : **확대/축소 배율**

- **이 함수를 쓰면 중심축 지정과 확대/축소까지 반영해서 손쉽게 변환행렬을 얻을 수 있다.**

Section 01 이동, 확대/축소, 회전

❑ 회전

- **다음 예제는 이미지를 45도, 90도로 회전한 예제이다.**

```
1   import cv2
2
3   img = cv2.imread('../img/fish.jpg')
4   rows,cols = img.shape[0:2]
5
6   m45 = cv2.getRotationMatrix2D((cols/2,rows/2),45,0.5)
7   m90 = cv2.getRotationMatrix2D((cols/2,rows/2),90,1.5)
8
9   img45 = cv2.warpAffine(img, m45,(cols, rows))
10  img90 = cv2.warpAffine(img, m90,(cols, rows))
```

Section 01 이동, 확대/축소, 회전

❑ 회전

- **다음 예제는 이미지를 45도, 90도로 회전한 예제이다.**

```
11

12  cv2.imshow("origin", img)
13  cv2.imshow("45", img45)
14  cv2.imshow("90", img90)
15  cv2.waitKey(0)
16  cv2.destroyAllWindows()
```

Section 02 뒤틀기

❑ 아핀변환

- **아핀 변환**(affine transform)**은 사실 이미 앞서 다룬 이동, 확대 /축소, 회전을 포함하는 변환으로 직선, 길이의 비율, 평행성을 보존하는 변환을 말한다**.
- **아핀 변환의 이런 성질 때문에 변환 전과 후의** 3**개의 점을 짝지어 매핑할 수 있다면 변환행렬을 거꾸로 계산할 수 있는데**, OpenCV**는** cv2.getAffineTransform() **함수로 이 기능을 제공한다**.
- `matrix = cv2.getAffineTransform(pts1, pts2)`
  - pts1 : **변환 전 영상의 좌표** 3**개**, 3 x 2 NumPy **배열** (float32)
  - pts2 : **변환 후 영상의 좌표** 3**개**, pts1**과 동일**
  - matrix : **변환행렬 반환**, 2 x 3 **행렬**

Section 02 뒤틀기

❑ 아핀변환

- 다음 예제는 정사각형의 영상이 마름모꼴의 모양으로 변형시키는 예제이다.

```
1   import cv2
2   import numpy as np
3
4   img = cv2.imread('./img/fish.jpg')
5   rows, cols = img.shape[:2]
6
7   pts1 = np.float32([[100, 50], [200, 50], [100, 200]])
8   pts2 = np.float32([[80, 70], [210, 60], [250, 120]])
9
10  cv2.circle(img, (100,50), 5, (255,0), -1)
11  cv2.circle(img, (200,50), 5, (0,255,0), -1)
12  cv2.circle(img, (100,200), 5, (0,0,255), -1)
```

Section 02 뒤틀기

❑ 아핀변환

- 다음 예제는 정사각형의 영상이 마름모꼴의 모양으로 변형시키는 예제이다.

```
13
14  mtrx = cv2.getAffineTransform(pts1, pts2)
15  dst = cv2.warpAffine(img, mtrx, (int(cols*1.5), rows))
16
17  cv2.imshow('origin',img)
18  cv2.imshow('affin', dst)
19  cv2.waitKey(0)
20  cv2.destroyAllWindows()
```

Section 02 뒤틀기

❑ 원근변환

- **원근 변환**(perspective transfolm)**은 보는 사람의 시각에 따라 같은 물체도 먼 것은 작게, 가까운 것은 크게 보이는 현상인 원근감을 주는 변환을 말한다.**
- **우리가 원근감을 느끼는 이유는 실제 세계가 3차원 좌표계이기 때문인데, 영상은 2차원 좌표계이다.**
- **그래서 차원 간의 차이를 보정해 줄 추가 연산과 시스템이 필요한데, 이때 사용하는 좌표계를 동차 좌표**(homogeneous coordinates)**라고 한다.**
- **이 때문에 원근 변환을 다른 말로 호모그래피** (homography)**라고도 한다.**
- **원근 변환을 하려면** (x, y, 1)**꼴의 좌표계가 필요하고, 아래와 같이** 3 X 3 **변환행렬식이 필요하다.**

$$\omega \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Section 02 뒤틀기

❑ 원근변환

- OpenCV는 **변환 전과 후를 짝짓는 4개의 매핑 좌표만 지정해 주면 원근 변환에 필요한** 3 x 3 **변환행렬을 계산해 주는** cV2.getPerspectiveTransform() **함수를 제공한다**.
- `mtrx = cv2.getPerspectiveTransform(pts1, pts2)`
  - pts1 : **변환 이전 영상의 좌표** 4**개**, 4 x 2 NumPy **배열** (float32)
  - pts2 : **변환 이전 영상의 좌표** 4**개**, pts1**과 동일**
  - mtrx : **변환행렬 반환**, 3 x 3 **행렬**
- OpenCV는 **원근 변환을 수행하는 함수로 지금까지 사용해온** cv2.warpAffine() **함수가 아닌 별도의 함수** cv2.warpPerspective() **함수를 제공한다**.
- **이 둘은 이름과 기능만 다를 뿐 사용 방법이 똑같다**.
- `dst = cv2.warpPerspective(src, mtrx, dsize [, dst, flags, borderMode, borderValue] )`
  - **모든 파라미터와 반환 값은** cv2.warpAffine() **과 동일**

Section 02 뒤틀기

❑ 원근변환

- 다음 예제는 원래 평면이었던 영상에 원근감을 표현하는 코드이다.

```
1   import cv2
2   import numpy as np
3
4   img = cv2.imread('./img/fish.jpg')
5   rows, cols = img.shape[:2]
6
7   pts1 = np.float32([[0,0], [0,rows], [cols, 0], [cols,rows]])
8   pts2 = np.float32([[100,50], [10,rows-50], [cols-100, 50], [cols-10,rows-50]])
9
10  cv2.circle(img, (0,0), 10, (255,0,0), -1)
11  cv2.circle(img, (0,rows), 10, (0,255,0), -1)
12  cv2.circle(img, (cols,0), 10, (0,0,255), -1)
13  cv2.circle(img, (cols,rows), 10, (0,255,255), -1)
```

Section 02 뒤틀기

❑ 원근변환

- 다음 예제는 원래 평면이었던 영상에 원근감을 표현하는 코드이다.

```
14
15  mtrx = cv2.getPerspectiveTransform(pts1, pts2)
16  dst = cv2.warpPerspective(img, mtrx, (cols, rows))
17
18  cv2.imshow('origin',img)
19  cv2.imshow('perspective', dst)
20  cv2.waitKey(0)
21  cv2.destroyAllWindows()
```

- 실행 결과를 보면 원래의 영상에서 좌표의 폭이 좁아진 부분은 작게 표현해서 마치 멀리 있는 것처럼 표현하는 것을 알 수 있다.
- 예제에서는 원래 평면이었던 영상에 원근감을 표현했지만, 실제로는 그 반대로 활용하는 경우가 더 많다.
- 카메라로 명함이나 문서 같은 것을 찍은 사진을 스캔한 것처럼 만들고 싶을 때는 반대로 원근감을 제거해야 한다.

Section 02 뒤틀기

❑ 원근변환

- 다음 예제는 사용자가 마우스로 영상 속 문서의 네 귀퉁이를 지정하면 원근 변환을 이용해서 스캔한 문서처럼 만들어 준다.

```
1   import cv2
2   import numpy as np
3
4   win_name = "scanning"
5   img = cv2.imread("../img/paper.jpg")
6   rows, cols = img.shape[:2]
7   draw = img.copy()
8   pts_cnt = 0
9   pts = np.zeros((4,2), dtype=np.float32)
10
11  def onMouse(event, x, y, flags, param):
12      global  pts_cnt
13      if event == cv2.EVENT_LBUTTONDOWN:
14          cv2.circle(draw, (x,y), 10, (0,255,0), -1)
15          cv2.imshow(win_name, draw)
```

Section 02 뒤틀기

❑ 원근변환

- 다음 예제는 사용자가 마우스로 영상 속 문서의 네 귀퉁이를 지정하면 원근 변환을 이용해서 스캔한 문서처럼 만들어 준다.

```
16
17          pts[pts_cnt] = [x,y]
18          pts_cnt+=1
19          if pts_cnt == 4:
20              sm = pts.sum(axis=1)
21              diff = np.diff(pts, axis = 1)
22
23              topLeft = pts[np.argmin(sm)]
24              bottomRight = pts[np.argmax(sm)]
25              topRight = pts[np.argmin(diff)]
26              bottomLeft = pts[np.argmax(diff)]
27
28              pts1 = np.float32([topLeft, topRight, bottomRight , bottomLeft])
29
```

Section 02 뒤틀기

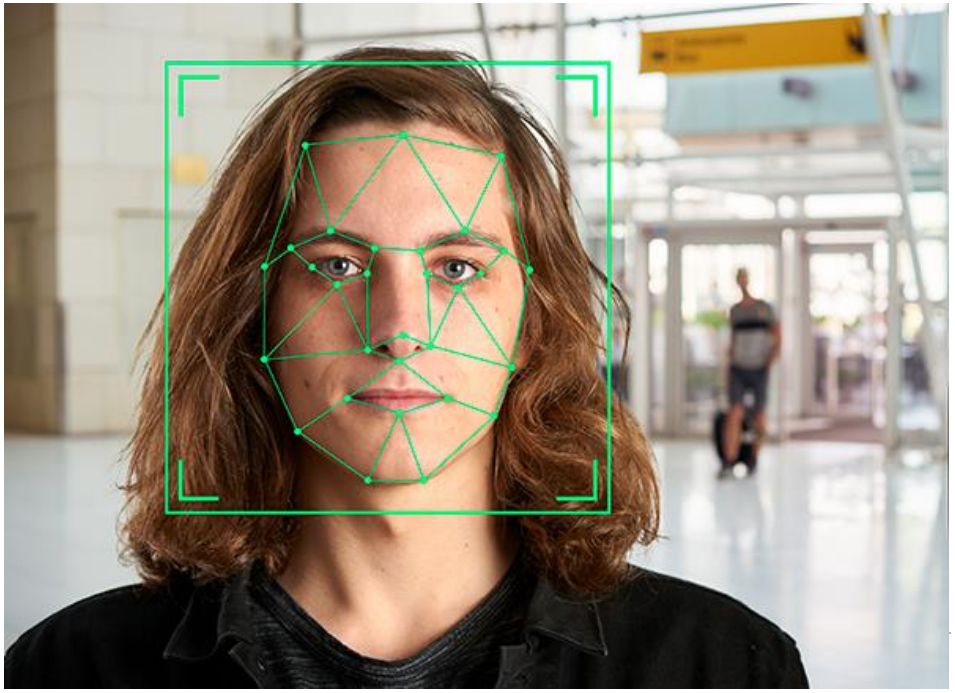
❑ 원근변환

```
30              w1 = abs(bottomRight[0] - bottomLeft[0])
31              w2 = abs(topRight[0] - topLeft[0])
32              h1 = abs(topRight[1] - bottomRight[1])
33              h2 = abs(topLeft[1] - bottomLeft[1])
34              width = max([w1, w2])
35              height = max([h1, h2])
36
37              pts2 = np.float32([[0,0], [width-1,0], [width-1,height-1], [0,height-1]])
38
39              mtrx = cv2.getPerspectiveTransform(pts1, pts2)
40              result = cv2.warpPerspective(img, mtrx, (width, height))
41              cv2.imshow('scanned', result)
42
43      cv2.imshow(win_name, img)
44      cv2.setMouseCallback(win_name, onMouse)
45      cv2.waitKey(0)
46      cv2.destroyAllWindows()
```

Section 02 뒤틀기

❑ 삼각형 아핀 변환

- 그림은 사람 얼굴에 대응점을 찾아 각 꼭짓점으로 여러 개의 삼각형을 그린 것이다.
- 이렇게 어떤 영역을 여러 개의 삼각형으로 나누는 기법을 들로네 삼각분할(Delaunay triangulation)이라고 하는데, 이렇게 나눈 삼각형들은 흔히 영상 분야에서는 입체적 표현이나 모핑 (morphing) 기술에 사용한다.



Section 02 뒤틀기

❑ 삼각형 아핀 변환

- 모핑 기술은 하나의 물체가 다른 물체로 자연스럽게 변하게 하는 것인데, 그림처럼 두 영상을 각각 여러 개의 삼각형으로 나누어 한 영상의 삼각형들의 크기와 모양이 나머지 영상에 대응하는 삼각형과 같아질 때까지 조금씩 바꿔서 전체적으로 하나의 영상이 다른 영상으로 자연스럽게 변하게 하는 것이다.
- 영상을 삼각 분할할 수 있다고 하더라도 삼각 분할된 영역을 변환하기는 쉽지 않다.
- OpenCV가 제공하는 기하학적 변환 기능은 영상을 대상으로 하므로 그 대상은 늘 사각형일 수밖에 없기 때문이다.

Section 02 뒤틀기

❑ 삼각형 아핀 변환

- 그래서 삼각형 모양의 변환을 하려면 다음과 같이 몇 가지 수고로운 과정을 거쳐야 한다.

1. 변환 전 삼각형 좌표 3쌍을 정한다.
2. 변환 후 삼각형 좌표 3쌍을 정한다.
3. 과정 1의 삼각형 좌표를 완전히 감싸는 외접 사각형 좌표를 구한다.
4. 과정 3의 사각형 영역을 관심영역으로 지정한다
5. 과정 4의 관심 영역을 대상으로 과정 1과 과정 2의 좌표로 변환행렬을 구하여 아핀 변환한다.
6. 과정 5의 변환된 관심 영 역에서 과정 2의 삼각형 좌표만 마스킹한다.
7. 과정 6의 마스크를 이 용해서 원본 또는 다른 영상에 합성한다.

Section 02 뒤틀기

❑ 삼각형 아핀 변환

- **위의 과정 3에서 삼각형 좌표를 완전히 감싸는 사각형의 좌표를 구하려면** cv2.boundingRect() **함수가 필요하다**.
- **이 함수는 삼각형뿐 아니라 다각형의 좌표를 전달하면 정확히 감싸는 외접 사각형의 좌표를 반환한다**.
- ```
  x, y, w, h = cv2.boundingRect(pts)
  ```
  - pts : **다각형 좌표**
  - x, y, w, h : **외접 사각형의 좌표와 폭과 높이**
  ```

Section 02 뒤틀기

❑ 삼각형 아핀 변환

- **과정 6에서 삼각형 마스크를 생성하기 위해** cV2.fillConvexPoly() **함수를 쓰면 편리하다.**
- **이 함수에 좌표를 전달하면 그 좌표 안을 원하는 색상 값으로 채워 주는데,** 255**나** 0**을 채우면 마스크를 쉽게 만들 수 있다.**
- `cv2.fillConvexPoly(img, points, color [, lineType])`
  - img : **입력 영상**
  - points : **다각형 꼭짓점 좌표**
  - color : **채우기에 사용할 색상**
  - lineType : **선 그리기 알고리즘 선택 플래그**

Section 02 뒤틀기

❑ 삼각형 아핀 변환

- 예제는 영상에 임의의 삼각형을 지정해서 그 삼각형만 어핀 변환하는 예제이다.

```
1   import cv2
2   import numpy as np
3
4   img = cv2.imread("../img/taekwonv1.jpg")
5   img2 = img.copy()
6   draw = img.copy()
7
8   pts1 = np.float32([[188,14], [85,202], [294,216]])
9   pts2 = np.float32([[128,40], [85,307], [306,167]])
10
11  x1,y1,w1,h1 = cv2.boundingRect(pts1)
12  x2,y2,w2,h2 = cv2.boundingRect(pts2)
13
14  roi1 = img[y1:y1+h1, x1:x1+w1]
15  roi2 = img2[y2:y2+h2, x2:x2+w2]
16
```

Section 02 뒤틀기

❑ 삼각형 아핀 변환

- 예제는 영상에 임의의 삼각형을 지정해서 그 삼각형만 어핀 변환하는 예제이다.

```
17  offset1 = np.zeros((3,2), dtype=np.float32)
18  offset2 = np.zeros((3,2), dtype=np.float32)
19  for i in range(3):
20      offset1[i][0], offset1[i][1] = pts1[i][0]-x1, pts1[i][1]-y1
21      offset2[i][0], offset2[i][1] = pts2[i][0]-x2, pts2[i][1]-y2
22
23  mtrx = cv2.getAffineTransform(offset1, offset2)
24  warped = cv2.warpAffine( roi1, mtrx, (w2, h2), None, \
                    cv2.INTER_LINEAR, cv2.BORDER_REFLECT_101)
25
26  mask = np.zeros((h2, w2), dtype = np.uint8)
27  cv2.fillConvexPoly(mask, np.int32(offset2), (255))
28
29  warped_masked = cv2.bitwise_and(warped, warped, mask=mask)
30  roi2_masked = cv2.bitwise_and(roi2, roi2, mask=cv2.bitwise_not(mask))
31  roi2_masked = roi2_masked + warped_masked
32  img2[y2:y2+h2, x2:x2+w2] = roi2_masked
```

Section 02 뒤틀기

❑ 삼각형 아핀 변환

- 예제는 영상에 임의의 삼각형을 지정해서 그 삼각형만 어핀 변환하는 예제이다.

```
33
34  cv2.rectangle(draw, (x1, y1), (x1+w1, y1+h1), (0,255,0), 1)
35  cv2.polylines(draw, [pts1.astype(np.int32)], True, (255,0,0), 1)
36  cv2.rectangle(img2, (x2, y2), (x2+w2, y2+h2), (0,255,0), 1)
37
38  cv2.imshow('origin', draw)
39  cv2.imshow('warped triangle', img2)
40  cv2.waitKey(0)
41  cv2.destroyAllWindows()
```

Section 03 렌즈 왜곡

❑ 리매핑

- OpenCV는 **규칙성 없이 마음대로 모양을 변환해 주는 함수로** cv2.remap()**을 제공한다**.
- **이 함수는 기존 픽셀의 위치를 원하는 위치로 재배치한다**.

```
dst = cv2.remap(src, mapx, mapy, interpolation [, dst, borderMode, borderValue) )
```

  - src : **입력 영상**
  - mapx, mapy : x**축과** y**축으로 이동할 좌표**(**인덱스**), src**와 동일한 크기**, dtype=float32
  - **나머지 인자는** cv2.warpAffine() **과 동일**
  - dst : **결과 영상**

- cv2.remap() **함수의** mapx, mapy**는 입력 영상인** src**와 크기가 같은 열로** float32**로 만들어야 하고**, **배열의 각 요소는** src**의 같은 인덱스의 픽셀이 각각 축과 축으로 옮겨 갈 새로운 인덱스를 갖게 해야 한다**.

Section 03 렌즈 왜곡

❑ 리매핑

- mapx와 mapy는 초기 값으로 0이나 1 같은 의미없는 값이 아니라 영상의 원래 좌표 값을 가지고 있는 것이 좋다.
- 왜냐하면 전체 픽셀에 옮기고 싶은 몇몇 픽셀에 대해서만 새로운 좌표 지정하거나 원래 있던 위치에서 얼마만큼 움직이라고 하는 것이 코딩하기 편하기 때문이다.
- mapx, mapy를 생성하는 코드는 아래와 같은 모양이 된다.

```
mapy, mapx = np.indices( (rows, cols), dtype=np.float32)
```

- cv2.remap() 함수는 배치할 좌표가 정수로 떨어지지 않는 등의 이유로 픽셀이 탈락하는 경우 자동으로 보정을 수행한다.

Section 03 렌즈 왜곡

❑ 리매핑

- **예제는 영상을 뒤집는 작업을 앞서 살펴본** cv2.warpAffin( ) **함수와** cv2.remap() **함수로 각각 구현해서 똑같은 결과를 보여준다.**

```
1   import cv2
2   import numpy as np
3   import time
4
5   img = cv2.imread("./img/lena.jpg")
6   rows, cols = img.shape[ :2]
7
8   st = time.time()
9   mflip = np.float32([[-1, 0, cols – 1], [0, -1, rows – 1]])
10  flipped1 = cv2.warpAffine(img, mflip, (cols, rows))
11  print('matrix:', time.time() – st)
12
```

Section 03 렌즈 왜곡

❑ 리매핑

- **예제는 영상을 뒤집는 작업을 앞서 살펴본** cv2.warpAffin( ) **함수와** cv2.remap() **함수로 각각 구현해서 똑같은 결과를 보여준다.**

```
13  st2 = time.time()
14  mapy, mapx = np.indices((rows, cols),dtype=np.float32) # 매핑 배열 초기화 생성
15  mapx = cols - mapx -1                                  # x축 좌표 뒤집기 연산
16  mapy = rows - mapy -1                                  # y축 좌표 뒤집기 연산
17  fliped2 = cv2.remap(img,mapx,mapy,cv2.INTER_LINEAR)  # remap 적용
18  print('remap:', time.time()-st2)
19
20  cv2.imshow('origin', img)
21  cv2.imshow('fliped1',fliped1)
22  cv2.imshow('fliped2',fliped2)
23  cv2.waitKey(0)
24  cv2.destroyAllWindows()
```

Section 03 렌즈 왜곡

❑ 리매핑

- 행렬로 표현할 수 있는 변환을 cv2.remap() 함수로 변환하는 것은 코드도 복잡하고 수행 속도도 느리다.
- 이 예제의 경우 일반 컴퓨터에서 약 4~5배 가량 속도가 느린 것으로 나타났다.
- 따라서 cv2.remap() 함수는 변환행렬로 표현할 수 없는 비선형 변환에만 사용하는 것이 좋다.
- 영상분야에서 비선형 변환은 대부분 렌즈 왜곡과 관련한 보정이나 효과에 사용한다

Section 03 렌즈 왜곡

❑ 리매핑

- **예제는 대표적인 비선형 변환이라 할 수 있는** sin, cos **파형을 리매핑 함수를 써서 변환한다**.

```
1   import cv2
2   import numpy as np
3
4   l = 20      # 파장(wave length)
5   amp = 15    # 진폭(amplitude)
6   rows, cols = img.shape[ :2]
7
8   img = cv2.imread('./img/taekwonv1.jpg')
9   rows, cols = img.shape[:2]
10
11  mapy, mapx = np.indices((rows, cols),dtype=np.float32)
12
```

Section 03 렌즈 왜곡

❑ 리매핑

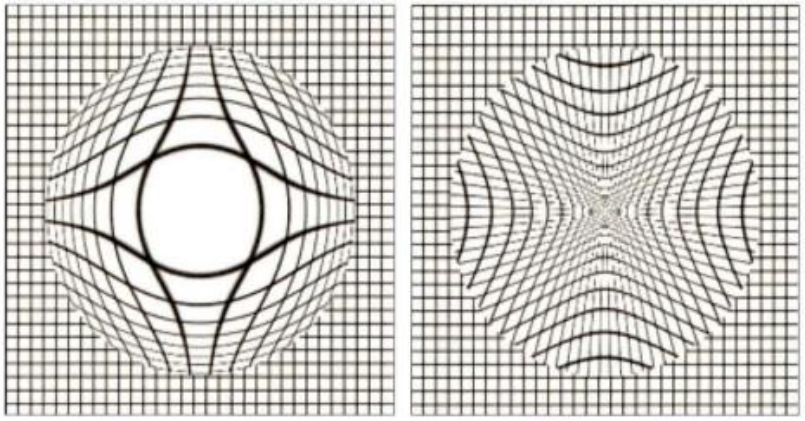
- **예제는 대표적인 비선형 변환이라 할 수 있는** sin, cos **파형을 리매핑 함수를 써서 변환한다**.

```
13  sinx = mapx + amp * np.sin(mapy/l)
14  cosy = mapy + amp * np.cos(mapx/l)
15
16  img_sinx=cv2.remap(img, sinx, mapy, cv2.INTER_LINEAR) # x축만 sin 곡선 적용
17  img_cosy=cv2.remap(img, mapx, cosy, cv2.INTER_LINEAR) # y축만 cos 곡선 적용
18  img_both=cv2.remap(img, sinx, cosy, cv2.INTER_LINEAR, None, cv2.BORDER_REPLICA
    TE)
19
20  cv2.imshow('origin', img)
21  cv2.imshow('sin x', img_sinx)
22  cv2.imshow('cos y', img_cosy)
23  cv2.imshow('sin cos', img_both)
24
25  cv2.waitKey(0)
26  cv2.destroyAllWindows()
```

Section 03 렌즈 왜곡

❑ 오목 렌즈와 볼록 렌즈 왜곡

- 리매핑 함수를 이용해서 영상에 볼록 렌즈와 오목 렌즈를 만드는 방법을 알아보자.



- 사각형 영상에서 동그란 원을 대상으로 작업을 하려면 좌표변환(직교좌표 ↔ 극좌표)이 필요하다.

Section 03 렌즈 왜곡

❑ 오목 렌즈와 볼록 렌즈 왜곡

- OpenCV는 **좌표 변환을 위한 함수를 다음과 같이 제공한다.**
- r, theta = cv2.cartToPolar(x, y) : 직교좌표 → 극좌표 변환
- x, y = cv2.polarToCart(r, theta) : 극좌표 → 직교좌표 변환
  - x, y : x, y **좌표 배열**
  - r : **원점과의 거리**
  - theta : **각도 값**
- **좌표의 변환뿐만 아니라 좌표의 기준점도 변경하는 것이 연산에 유리하다.**
- **영상의 경우 직교좌표를 사용할 때는 좌상단 끝을 (0, 0) 좌표로, 극좌표를 사용하는 경우에는 영상의 중앙을 기준점으로 사용하는 것이 당연하고 원점을 기준으로 좌측과 하단은 음수 좌표가 필요하므로 좌표의 값도 -1~1로 노멀라이즈해서 사용하는 것이 유리하다.**

Section 03 렌즈 왜곡

❑ 오목 렌즈와 볼록 렌즈 왜곡

- 다음 예제는 영상에 오목 렌즈와 볼록 렌즈 효과를 적용한 코드이다.

```
1   import cv2
2   import numpy as np
3
4   img = cv2.imread('./img/taekwonv1.jpg')
5   print(img.shape)
6   rows, cols = img.shape[ :2]
7
8   img = cv2.imread('./img/taekwonv1.jpg')
9   rows, cols = img.shape[:2]
10
11  exp = 2      # 볼록, 오목 지수 (오목 : 0.1 ~ 1, 볼록 : 1.1~)
12  scale = 1        # 변환 영역 크기 (0 ~ 1)
13
14  mapy, mapx = np.indices((rows, cols),dtype=np.float32)
15
```

Section 03 렌즈 왜곡

❑ 오목 렌즈와 볼록 렌즈 왜곡

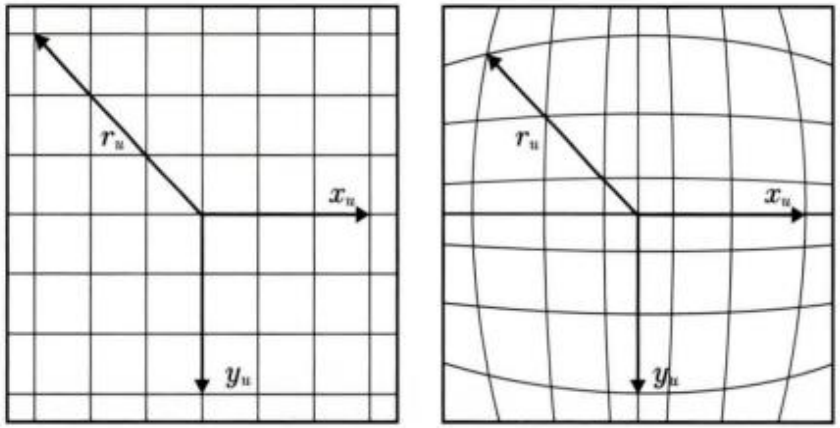
- 다음 예제는 영상에 오목 렌즈와 볼록 렌즈 효과를 적용한 코드이다.

```
16  mapx = 2*mapx/(cols-1)-1
17  mapy = 2*mapy/(rows-1)-1
18
19  r, theta = cv2.cartToPolar(mapx, mapy)
20
21  r[r< scale] = r[r<scale] **exp
22
23  mapx, mapy = cv2.polarToCart(r, theta)
24
25  mapx = ((mapx + 1)*cols-1)/2
26  mapy = ((mapy + 1)*rows-1)/2
27  distorted = cv2.remap(img,mapx,mapy,cv2.INTER_LINEAR)
28
29  cv2.imshow('origin', img)
30  cv2.imshow('distorted', distorted)
31  cv2.waitKey(0)
32  cv2.destroyAllWindows()
```

Section 03 렌즈 왜곡

❑ 방사 왜곡

- 우리는 대부분의 영상을 카메라로 촬영해서 얻는데, 카메라 렌즈는 동그랗고 영상은 사각형이다 보니 렌즈 가장 자리 영상에는 왜곡이 생기기 마련이고 이런 왜곡을 배럴 왜곡(barrel distortion)이라고 한다.



- 배럴 왜곡의 왜곡 계수의 값에 따라 밖으로 튀어 나오는 배럴 왜곡이 나타나기도 하고, 안으로 들어 가는 핀쿠션(pincushion distortion) 이 일어나기도 한다.

Section 03 렌즈 왜곡

❑ 방사 왜곡

- 다음 예제는 배럴 왜곡과 핀 쿠션을 발생시키는 코드이다.

```
1   import cv2
2   import numpy as np
3
4   k1, k2, k3 = 0.5, 0.2, 0.0 # 배럴 왜곡
5   #k1, k2, k3 = -0.3, 0, 0    # 핀쿠션 왜곡
6
7   img = cv2.imread('./img/lena.jpg')
8   rows, cols = img.shape[ :2]
9
10  mapy, mapx = np.indices((rows, cols),dtype=np.float32)
11
12  mapx = 2*mapx/(cols-1)-1
13  mapy = 2*mapy/(rows-1)-1
14  r, theta = cv2.cartToPolar(mapx, mapy)
```

Section 03 렌즈 왜곡

❑ 방사 왜곡

- 다음 예제는 배럴 왜곡과 핀 쿠션을 발생시키는 코드이다.

```
15
16  mapx, mapy = cv2.polarToCart(ru, theta)
17  mapx = ((mapx + 1)*cols-1)/2
18  mapy = ((mapy + 1)*rows-1)/2
19  distored = cv2.remap(img,mapx,mapy,cv2.INTER_LINEAR)
20
21  cv2.imshow('origin', img)
22  cv2.imshow('distorted', distorted)
23  cv2.waitKey(0)
24  cv2.destroyAllWindows()
```

Section 03 렌즈 왜곡

❑ 방사 왜곡

- OpenCV는 **배럴 왜곡 현상이 일어나는 렌즈의 왜곡을 제거하기 위한** cv2.undistort( ) **함수를 제공한다**.
- ```
  dst = cv2.undistort(src, cameraMtrix, distCoeffs)
  ```
  - src : **입력 원본 영상**
  - cameraMatrix : **카메라 매트릭스**
  - distCoeffs : **왜곡 계수**, **최소** 4**개 또는** 5, 8, 12, 14**개**
    - (k1, k2, p1, p2 [, k3])
- cameraMatrix**는 촬영할 카메라의 내부 파라미터인 중심점** cx, cy**와 초점 거리**, fx, fy**를 입력한다**.
- **이 값들은 카메라가 생산될 때 가지는 고유의 특성이므로 개발자가 단순히 값을 입력하거나 계산할 수는 없다**.
- OpenCV**는 정확히 그려진 체스보드 영상을 촬영해서 역으로 왜곡 현상을 없애는 과정인 카메라 영점 조정**(calibration)**에 사용할 추가적인 몇 가지 함수를 제공하는데, 이들을 이용하여 카메라 계수의 근사 값을 얻어낼 수 있다**.
  ```

Section 03 렌즈 왜곡

❑ 방사 왜곡

- **다음 예제는 방사 왜곡**(radial distortion) **현상을** cv2.undistort() **함수로 구현한 예제이다**.

```
1   import cv2
2   import numpy as np
3
4   img = np.full((300,400,3), 255, np.uint8)
5   img[::10, :, :] = 0
6   img[:, ::10, :] = 0
7   width  = img.shape[1]
8   height = img.shape[0]
9
10  k1, k2, p1, p2 = 0.001, 0, 0, 0     # 배럴 왜곡
11  #k1, k2, p1, p2 = -0.0005, 0, 0, 0  # 핀쿠션 왜곡
12  distCoeff = np.float64([k1, k2, p1, p2])
13
```

Section 03 렌즈 왜곡

❑ 방사 왜곡

- **다음 예제는 방사 왜곡**(radial distortion) **현상을** cv2.undistort() **함수로 구현한 예제이다**.

```
14  fx, fy = 10, 10
15  cx, cy = width/2, height/2
16  camMtx = np.float32([[fx,0, cx], [0, fy, cy], [0 ,0 ,1]])
17
18  dst = cv2.undistort(img,camMtx,distCoeff)
19
20  cv2.imshow('origin', img)
21  cv2.imshow('dst',dst)
22  cv2.waitKey(0)
23  cv2.destroyAllWindows()
```

Section 04 프로젝트

❑ 모자이크 처리

- 사진의 특정 영역을 마우스로 선택하면 그 영역을 모자이크 처리하게 해보자.

- 힌트
  - 특정 영역을 작게 축소했다가 다시 확대하면 원래의 픽셀과 비슷하긴 하지만, 보간법에 의해서 연산한 결과라서 선명도가 떨어져 뿌옇게 보인다.
  - 보간법 알고리즘으로는 cv2.INTER_AREA를 사용하면 예시에서처럼 저해상도 픽셀처럼 된다.

Section 04 프로젝트

❑ 모자이크 처리

```
1  import cv2
2  import numpy as np
3
4  rate = 15              # 모자이크에 사용할 축소 비율 (1/rate)
5  win_title = 'mosaic'   # 창 제목
6  img = cv2.imread('../img/taekwonv1.jpg')   # 이미지 읽기
7
8  while True:
9      x,y,w,h = cv2.selectROI(win_title, img, False) # 관심영역 선택
```

Section 04 프로젝트

❑ 모자이크 처리

```
10      if w and h:
11          roi = img[y:y+h, x:x+w]   # 관심영역 지정
12          roi = cv2.resize(roi, (w//rate, h//rate))       # 1/rate 비율로 축소
13          roi = cv2.resize(roi, (w,h), interpolation=cv2.INTER_AREA)
14          img[y:y+h, x:x+w] = roi   # 원본 이미지에 적용
15          cv2.imshow(win_title, img)
16      else:
17          break
18
19  cv2.destroyAllWindows()
```

Section 04 프로젝트

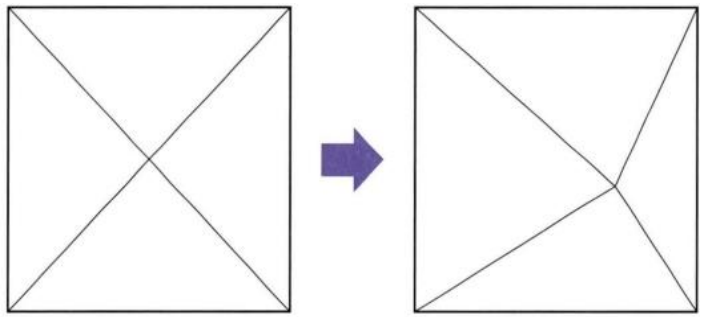
❑ 포토샵 리퀴파이 도구

- 포토샵에는 사진의 원하는 부분만 작게 하거나 크게 하는 리퀴파이(Liquify)라는 도구가 있어서 얼굴이나 몸매를 보정하는 데 많이 사용되고 있다.
- Liquify는 '액체로 만들다'라는 뜻으로 영상 분야에서는 영상의 일부분을 액체 괴물처럼 흐물거리게 해서 모양을 바꾸는 효과를 말한다.
- 영화 터미네이터의 액체 금속 로봇 T-1000은 몸을 자유자재로 바꾸는데, 이때 몸이 변하는 장면에 사용하는 기법도 리퀴파이이다.
- 그림처럼 마우스로 특정 부분만 변환하는 포토샵의 리퀴파이 기능을 만들어보자.

Section 04 프로젝트

❑ 포토샵 리퀴파이 도구

- 힌트
  - 아핀 변환 그림처럼 사각형 영역을 4개의 삼각형으로 나누어 마우스의 위치를 그 중 삼점으로 하게 한다.
  - 마우스를 드래그하면 드래그를 시작한 중심점에서 드래그가 끝난 좌표를 얻어서 새로운 삼각형 좌표 4개를 얻는다.
  - 새로운 좌표만큼 4개의 삼각형을 아핀 변환한다.



Section 04 프로젝트

❑ 포토샵 리퀴파이 도구

```
1   import cv2
2   import numpy as np
3
4   win_title = 'Liquify'   # 창 이름
5   half = 50               # 관심 영역 절반 크기
6   isDragging = False      # 드래그 여부 플래그
7
8   def liquify(img, cx1,cy1, cx2,cy2) :
9       x, y, w, h = cx1-half, cy1-half, half*2, half*2
10      roi = img[y:y+h, x:x+w].copy()
11     out = roi.copy()
12
13      offset_cx1,offset_cy1 = cx1-x, cy1-y
14      offset_cx2,offset_cy2 = cx2-x, cy2-y
15
```

Section 04 프로젝트

❑ 포토샵 리퀴파이 도구

```
16      tri1 = [[ (0,0), (w, 0), (offset_cx1, offset_cy1)], # 상,top
                [ [0,0], [0, h], [offset_cx1, offset_cy1]], # 좌,left
                [ [w, 0], [offset_cx1, offset_cy1], [w, h]], # 우, right
                [ [0, h], [offset_cx1, offset_cy1], [w, h]]] # 하, bottom
17
18      tri2 = [[ [0,0], [w,0], [offset_cx2, offset_cy2]], # 상, top
                [ [0,0], [0, h], [offset_cx2, offset_cy2]], # 좌, left
                [ [w,0], [offset_cx2, offset_cy2], [w, h]], # 우, right
                [ [0,h], [offset_cx2, offset_cy2], [w, h]]] # 하, bottom
19
20      for i in range(4):
21          matrix = cv2.getAffineTransform( np.float32(tri1[i]), np.float32(tri2[i]))
22          warped = cv2.warpAffine( roi.copy(), matrix, (w, h), \
                None, flags=cv2.INTER_LINEAR, borderMode=cv2.BORDER_REFLECT_101)
23          mask = np.zeros((h, w), dtype = np.uint8)
24          cv2.fillConvexPoly(mask, np.int32(tri2[i]), (255,255,255))
25
26          warped = cv2.bitwise_and(warped, warped, mask=mask)
27          out = cv2.bitwise_and(out, out, mask=cv2.bitwise_not(mask))
28          out = out + warped
29
```

Section 04 프로젝트

❑ 포토샵 리퀴파이 도구

```
30      img[y:y+h, x:x+w] = out
31      return img
32
33  def onMouse(event,x,y,flags,param):
34      global cx1, cy1, isDragging, img      # 전역변수 참조
35      if event == cv2.EVENT_MOUSEMOVE:
36          if not isDragging :
37              img_draw = img.copy()
38              cv2.rectangle(img_draw, (x-half, y-half), (x+half, y+half), (0,255,0))
39              cv2.imshow(win_title, img_draw) # 사각형 표시된 그림 화면 출력
40      elif event == cv2.EVENT_LBUTTONDOWN :
41          isDragging = True                 # 드래그 시작
42          cx1, cy1 = x, y                   # 드래그 시작된 원래의 위치 좌표 저장
43      elif event == cv2.EVENT_LBUTTONUP :
44          if isDragging:
45              isDragging = False            # 드래그 끝
46              liquify(img, cx1, cy1, x, y)
47              cv2.imshow(win_title, img)
48
```

Section 04 프로젝트

❑ 포토샵 리퀴파이 도구

```
49  if __name__ == '__main__' :
50      img = cv2.imread("../img/man_face.jpg")
51      h, w = img.shape[:2]
52
53      cv2.namedWindow(win_title)
54      cv2.setMouseCallback(win_title, onMouse)
55      cv2.imshow(win_title, img)
56      while True:
57          key = cv2.waitKey(1)
58          if key & 0xFF == 27:
59              break
60      cv2.destroyAllWindows()
```

Section 04 프로젝트

❑ 왜곡 거울 카메라

- 힌트
  - 렌즈 왜곡을 참고해서 하나의 영상에 원본, 좌우 거울 왜곡, 상하 거울 왜곡, 물결 왜곡, 볼록 렌즈 왜곡, 오목 렌즈 왜곡 효과를 모두 하나의 영상에 병합해서 출력한다.
  - 각각의 왜곡 효과를 위한 리매핑 좌표는 새로운 프레임마다 매번 구할 필요는 없고 한번 만들어진 리매핑 좌표에 새로운 프레임을 리매핑하기만 하면 된다.

Section 04 프로젝트

❑ 왜곡 거울 카메라

```
1   import cv2
2   import numpy as np
3
4   cap = cv2.VideoCapture(0)
5   cap.set(cv2.CAP_PROP_FRAME_WIDTH, 320)
6   cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 240)
7   rows, cols = 240, 320
8   map_y, map_x = np.indices((rows, cols), dtype=np.float32)
9
10  # 거울 왜곡 효과
11  map_mirrorh_x,map_mirrorh_y = map_x.copy(), map_y.copy()
12  map_mirrorv_x,map_mirrorv_y = map_x.copy(), map_y.copy()
13  map_mirrorh_x[: , cols//2:] = cols - map_mirrorh_x[:, cols//2:]-1
14  map_mirrorv_y[rows//2:, :] = rows - map_mirrorv_y[rows//2:, :]-1
15
```

Section 04 프로젝트

❑ 왜곡 거울 카메라

```
16  # 물결 효과
17  map_wave_x, map_wave_y = map_x.copy(), map_y.copy()
18  map_wave_x = map_wave_x + 15*np.sin(map_y/20)
19  map_wave_y = map_wave_y + 15*np.sin(map_x/20)
20
21  # 렌즈 효과
22  ## 렌즈 효과, 중심점 이동
23  map_lenz_x = 2*map_x/(cols-1)-1
24  map_lenz_y = 2*map_y/(rows-1)-1
25  ## 렌즈 효과, 극좌표 변환
26  r, theta = cv2.cartToPolar(map_lenz_x, map_lenz_y)
27  r_convex = r.copy()
28  r_concave = r
29  ## 볼록 렌즈 효과 매핑 좌표 연산
30  r_convex[r< 1] = r_convex[r<1] **2
31  print(r.shape, r_convex[r<1].shape)
32  ## 오목 렌즈 효과 매핑 좌표 연산
33  r_concave[r< 1] = r_concave[r<1] **0.5
```

Section 04 프로젝트

❑ 왜곡 거울 카메라

```
34  ## 렌즈 효과, 직교 좌표 복원
35  map_convex_x, map_convex_y = cv2.polarToCart(r_convex, theta)
36  map_concave_x, map_concave_y = cv2.polarToCart(r_concave, theta)
37  ## 렌즈 효과, 좌상단 좌표 복원
38  map_convex_x = ((map_convex_x + 1)*cols-1)/2
39  map_convex_y = ((map_convex_y + 1)*rows-1)/2
40  map_concave_x = ((map_concave_x + 1)*cols-1)/2
41  map_concave_y = ((map_concave_y + 1)*rows-1)/2
42
43  while True:
44      ret, frame = cap.read()
45      # 준비한 매핑 좌표로 영상 효과 적용
46      mirrorh=cv2.remap(frame,map_mirrorh_x,map_mirrorh_y,cv2.INTER_LINEAR)
47      mirrorv=cv2.remap(frame,map_mirrorv_x,map_mirrorv_y,cv2.INTER_LINEAR)
48      wave = cv2.remap(frame,map_wave_x,map_wave_y,cv2.INTER_LINEAR, \
                    None, cv2.BORDER_REPLICATE)
49      convex = cv2.remap(frame,map_convex_x,map_convex_y,cv2.INTER_LINEAR)
50      concave = cv2.remap(frame,map_concave_x,map_concave_y,cv2.INTER_LINEAR)
```

Section 04 프로젝트

❑ 왜곡 거울 카메라

- **힌트**

```
51    # 영상 합치기
52    r1 = np.hstack(( frame, mirrorh, mirrorv))
53    r2 = np.hstack(( wave, convex, concave))
54    merged = np.vstack((r1, r2))
55
56    cv2.imshow('distorted',merged)
57    if cv2.waitKey(1) & 0xFF== 27:
58        break
59
60  cap.release
61  cv2.destroyAllWindows()
```

Q&A