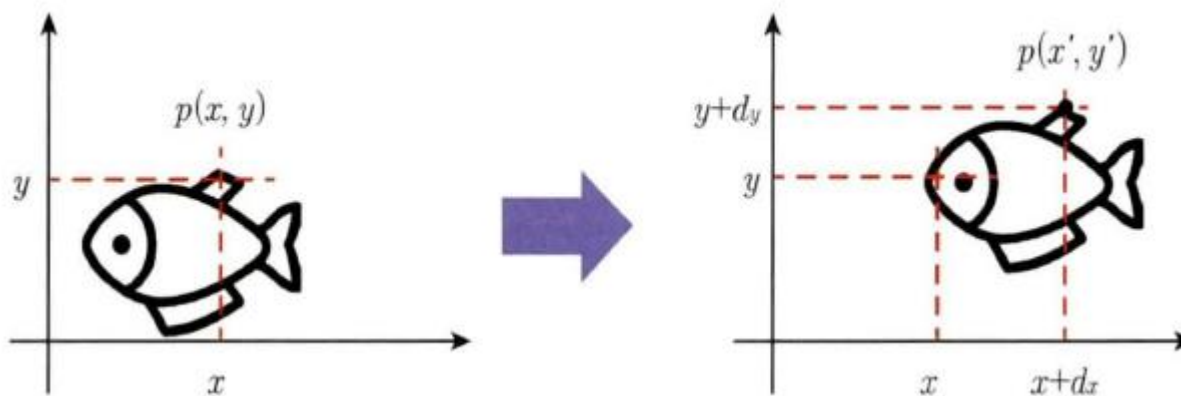


CH. 8. 기하학적 변환

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 dx 와 dy 만큼 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, y 는 이미 원본 이미지의 좌표 값으로 제공되므로 2×3 변환행렬만 전달하면 연산이 가능하다.

Section 01 이동, 확대/축소, 회전

□ 이동

- OpenCV는 2 X 3 행렬로 영상의 좌표를 변환시켜 주는 함수를 다음과 같이 제공한다.

```
dst = cv2.warpAffine(src, mtrx, dsize [, dst , flags , borderMode, borderValue])
```

- 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 X 3 행렬로 영상의 좌표를 변환시켜 주는 함수를 다음과 같이 제공한다.

```
dst = cv2.warpAffine(src, mtrx, dsize [, dst , flags , borderMode, borderValue])
```

- borderMode : 외곽 영역 보정 플래그
 - cv2.BORDER_CONSTANT : 고정 색상 값(999 | 12345 | 999)
 - cv2.BORDER_REPLICATE : 가장자리 복제(111 | 12345 | 555)
 - cv2.BORDER_WRAP : 반복(345 | 12345 | 123)
 - cv2.BORDER_REFLECT: 반사(321 | 12345 | 543)
- borderValue : cv2.BORDER_CONSTANT의 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

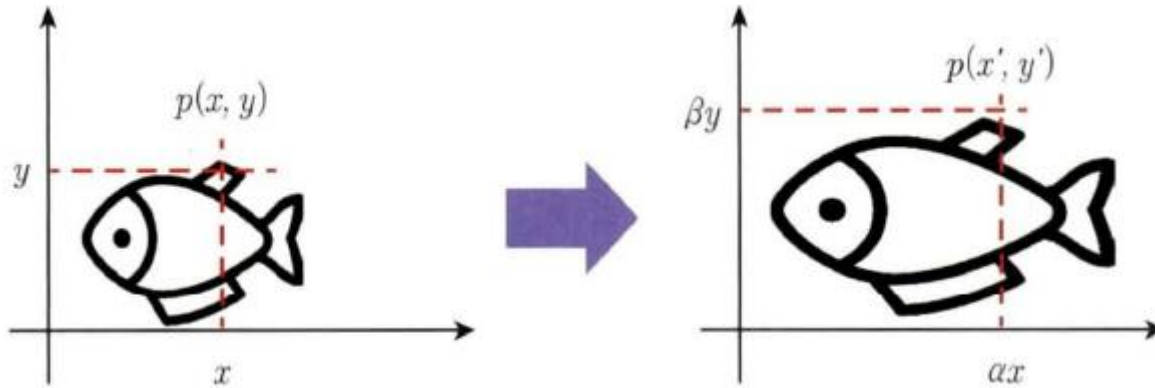
□ 이동

- 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_CONSTANT', dst2)
20 cv2.imshow('BORDER_REFLECT', 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 α 와 β 라고 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 2×2 행렬로도 충분히 표현이 가능한데 굳이 마지막 열에 0으로 채워진 열을 추가해서 2×3 행렬로 표현한 이유는 `cv2.warpAffine()` 함수와 이동 변환 때문이다.
- 앞서 다룬 이동을 위한 행렬식은 2×3 행렬로 표현해야 하므로 여러 가지 기하학적 변환을 지원해야 하는 `cv2.warpAffine()` 함수는 2×3 행렬이 아니면 오류를 발생한다.
- 행렬의 마지막 열에 `dx`, `dy`에 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.
- OpenCV는 변환행렬을 작성하지 않고도 확대와 축소 기능을 사용할 수 있게 `cv2.resize()` 함수를 별도로 제공한다.
- ```
dst = cv2.warpAffine(src, mtrx, dsize [, dst , flags , borderMode, borderValue])
```

  - `src` : 입력 영상, NumPy 배열
  - `dsize` : 출력 영상 크기(확대 /축소 목표 크기), 생략하면 `fx, fy`를 적용
    - (width, height)
  - `fx, fy` : 크기 비율, 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 0.5배 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 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 0.5배 축소와 2배 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.
- OpenCV는 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, 60진법
 - 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 3개의 점을 짝지어 매핑할 수 있다면 변환행렬을 거꾸로 계산할 수 있는데, OpenCV는 `cv2.getAffineTransform()` 함수로 이 기능을 제공한다.
- `matrix = cv2.getAffineTransform(pts1, pts2)`
 - pts1 : 변환 전 영상의 좌표 3개, 3 x 2 NumPy 배열 (float32)
 - pts2 : 변환 후 영상의 좌표 3개, pts1과 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 transform)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 3×3 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 3×3 변환행렬을 계산해 주는 `cv2.getPerspectiveTransform()` 함수를 제공한다.

- `mtrx = cv2.getPerspectiveTransform(pts1, pts2)`

- pts1 : 변환 이전 영상의 좌표 4개, 4×2 NumPy 배열 (float32)
- pts2 : 변환 이전 영상의 좌표 4개, pts1과 동일
- mtrx : 변환행렬 반환, 3×3 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])`

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

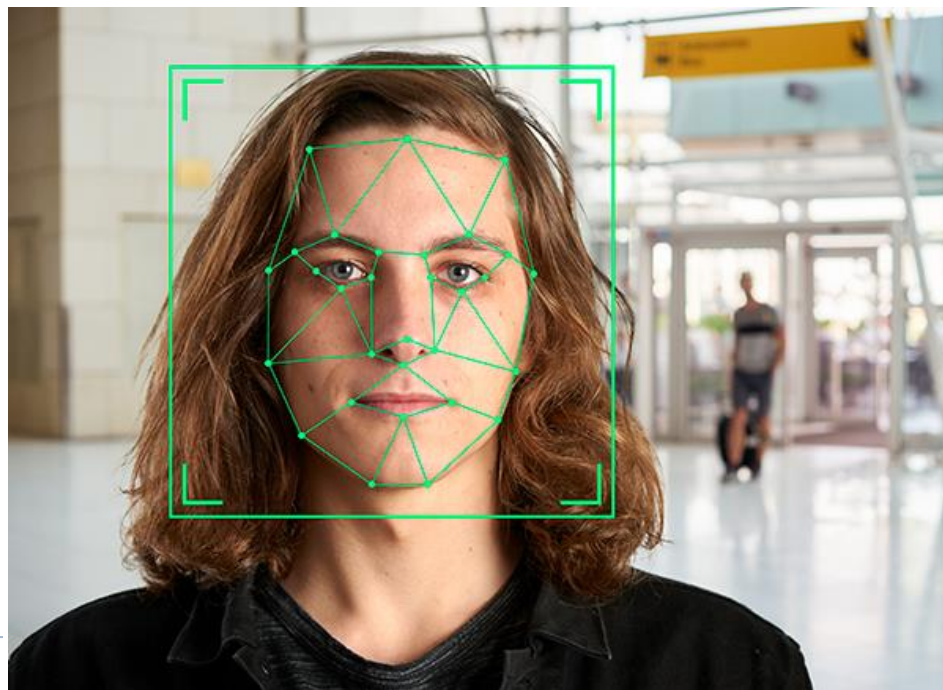
□ 원근변환

```
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 영역에서 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, 255나 0을 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 아핀 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 아핀 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 아핀 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` : x축과 y축으로 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 x축과 y축으로 옮겨 갈 새로운 인덱스를 갖게 해야 한다.

Section 03 렌즈 왜곡

□ 리매핑

- `mapx`와 `mapy`는 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`, `mapy`를 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  flipped2 = 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', flipped1)
22  cv2.imshow('fliped2', flipped2)
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

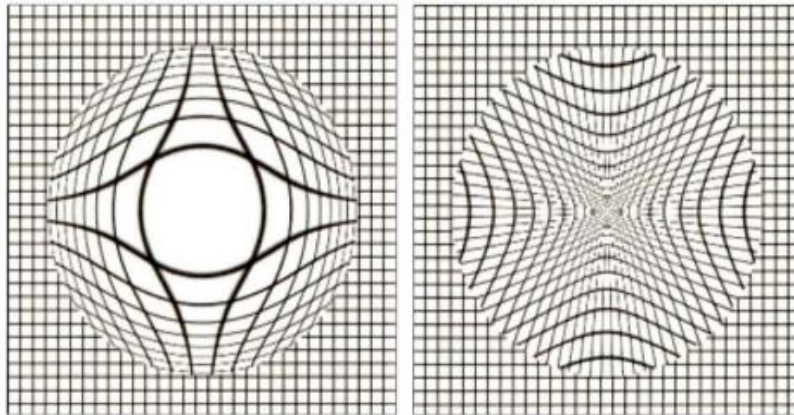
- 예제는 대표적인 비선형 변환이라 할 수 있는 sin, cos 파형을 리매핑 함수를 써서 변환한다.

```
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 기준으로 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

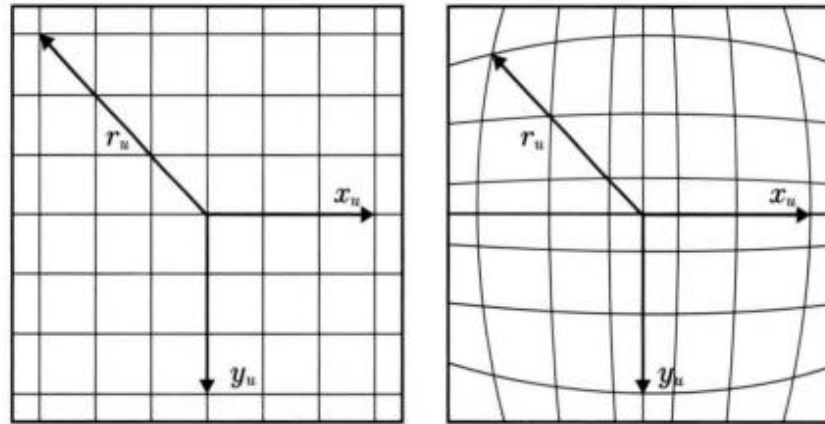
- 다음 예제는 영상에 오목 렌즈와 볼록 렌즈 효과를 적용한 코드이다.

```
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 가장자리 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 4개 또는 5, 8, 12, 14개
    - $(k_1, k_2, p_1, p_2 [, k_3])$
- `cameraMatrix`는 촬영할 카메라의 내부 파라미터인 중심점  $c_x, c_y$ 와 초점 거리,  $f_x, f_y$ 를 입력한다.
- 이 값들은 카메라가 생산될 때 가지는 고유의 특성이므로 개발자가 단순히 값을 입력하거나 계산할 수는 없다.
- OpenCV는 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_AREA`를 사용하면 예시에서처럼 저해상도 픽셀처럼 된다.

# Section 04 프로젝트

## □ 모자이크 처리

```
1 import cv2
2
3 import numpy as np
4
5 rate = 15 # 모자이크에 사용할 축소 비율 (1/rate)
6
7 win_title = 'mosaic' # 창 제목
8
9 img = cv2.imread('../img/taekwonv1.jpg') # 이미지 읽기
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
26
```

# 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.



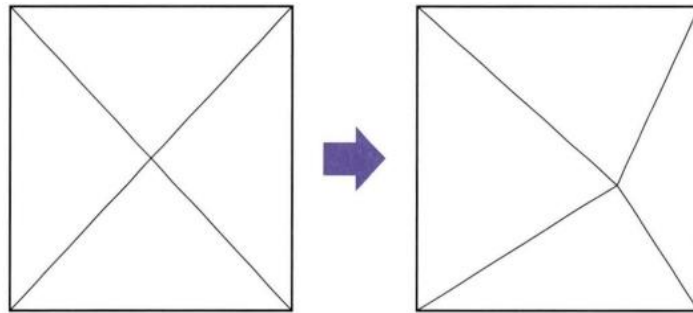


## Section 04 프로젝트

### □ 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)
```

# 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

