과제 1 영상 회전 및 보간

황 순 규
국민대학교 전자공학부
emfprh123@kookmin.ac.kr

요 약

영상 회전을 진행 한 후에 보간 과정을 통해서 영상의 잘린 부분 없이 전체 화면에 영상이 출력되게끔 만드는 과제를 수행한 코드이다.

1. 서론

이 과제를 수행하기 위해서 회전행렬, 평행이동, 역행렬, 쌍선형 보간 개념을 이용하였다. 영상을 회전하고 보간하는 과정은 간단하게 다음과 같다. 먼저 원본 영상의 꼭짓점 좌표들을 회전 행렬을 통해서 회전시켜주고 이를 통해 출력 영상의 크기를 구한 다음 출력 영상의 크기 내부를 보간 과정을 통해서 영상을 채운다.

2. 수행 내용

2.1 영상의 회전 및 입력 영상 크기 추출

영상을 회전시켜주기 위해서 회전행렬 $R = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$을 사용하였다. 위 행렬을 통해서 원본 이미지의 height, width를 이용해 구한 네 개의 꼭짓점 좌표를 회전시켜 주었다

```
def rotate_img(img, ang):
  # compute rotation matrix
  rotate_matrix = np.array([[cos, -sin], [sin, cos]])
  # compute output coordinates for image corners
  rotate_matrix = np.array([[cos, -sin], [sin, cos]])
  height, width, channel = img.shape
  height, width, channel = img.shape
  corners = np.array([[0, 0], [width, 0], [width, height], [0, height]])
  # compute output image size and offset
  output_corners = np.dot(corners, rotate_matrix)
```

그림 1 영상 회전 과정 코드

np.dot 내적 계산을 통해 output_corners를 구해주었다. 이때 output corners는 원본 이미지의 네 개 꼭짓점 좌표를 회전행렬을 통해 회전 시킨 값을 저장한 배열이다.

위 과정을 통해 회전한 새로운 이미지의 네 개의 꼭짓점 좌표가 생성되었을 것이다. 이대로 출력하게 된다면 영상이 잘려서 출력될 것이므로 새로운 출력 영상의 크기를 구해주어야 한다.

2.2 출력 영상의 크기 추출

```
x_min, y_min = np.min(output_corners, axis=0)
x_max, y_max = np.max(output_corners, axis=0)

output_width = int(np.ceil(x_max - x_min))
output_height = int(np.ceil(y_max - y_min))
# define output image object
img_out = np.zeros((output_height, output_width,3), dtype=np.uint8)
```

그림 2 출력 영상 크기 추출

2.1번 과정에서 출력 영상의 꼭짓점 좌표 4개를 얻었을 것이다. 꼭짓점의 좌표 중에서 가장 큰 값과 가장 작은 값의 차를 올림해주는 과정을 통해 width와 height를 구한다. 이때 올림을 해주는 이유는 이후의 보간을 하는 과정에서 좌표 값을 조회할 때 인덱스 오류가 날 수 있기 때문에 ceil을 통해 올림을 해준다. 위 과정으로 출력 영상의 height, width를 구했으면 np.zeros를 통해서 검은색의 빈 영상을 출력영상의 크기로 생성해준다.

2.3 보간 과정

이제 비어있는 출력 영상을 채워주어야 할 차례이다. 이중 for 문을 통해서 출력 영상의 모든 좌표들을 조회한다. 임의의 (a, b) 좌표를 예시로 들어보겠다. 출력 영상에서 (a, b)의 좌표에 해당하는 픽

셀 값을 채워야하는데 이를 좌표 값의 평행이동, 역행렬 계산 후 계산된 좌표 값이 원래 이미지 내부에 있으면 보간 아니면 0으로 처리하는 방식을 사용할 것이다.

```
for y_out in range(output_height):
    for x_out in range(output_width):
        origin_coord = [x_out + offset_x, y_out + offset_y]
        final_coord = np.dot(origin_coord, rot_inv)
        #원래대로 돌렸을 때 내 처음의 이미지의 Height, width 내에 있는지 판단하여 안에 있으면 보간 해주고 아니면 그냥 [0,0,0] 값을 유지
        if np.floor(final_coord[0]) > 0 and np.ceil(final_coord[0]) < width and np.floor(final_coord[1]) > 0  and np.ceil(final_coord[1]) < height:
```

그림 3 보간 과정 1

이제 본격적인 쌍선형 보간 과정을 이행할 것이다. 쌍선형 보간 과정은 다음 그림을 통해 진행하였다.

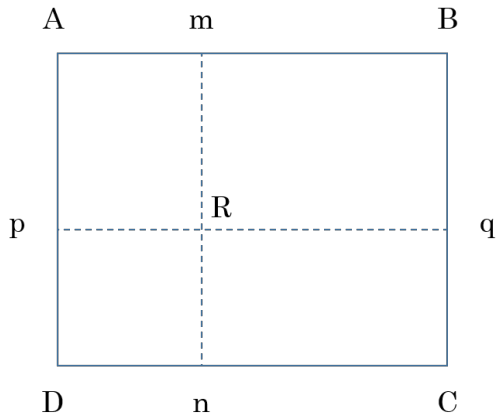

그림 4 쌍선형 보간

A, B의 선형보간을 통해 m을 구하고 D, C를 선형보간해 n을 구하고 m, n을 선형보간해 R을 구하는 과정으로 진행한다. 하지만 여기서 문제가 생기는데 R 좌표가 정수로 안떨어지는 경우이다. 좌표 값은 정수로 이루어져있는데 정수가 아닌 경우에는 값이 없기 때문이다. 하지만 올림, 내림 과정을 통해서 정수 A, B, C, D 좌표를 구해서 쌍선형 보간을 진행하면 되기 때문에 문제될 것은 없다. 이 과정을 진행한 코드는 다음과 같다.

```
x_min, x_ (variable) final_coord: Any ior(final_coord[0])) , int(np.ceil(final_coord[0])), int(np.floor(final_coord[1])), int(np.ceil(final_coord[1]))
d1 = abs(final_coord[0] - x_min)
d2 = abs(x_max - final_coord[0])
d3 = abs(y_max - final_coord[1])
d4 = abs(final_coord[1] - y_min)

pixel1 = ((d1*img[y_max, x_max]+d2*img[y_max, x_min])/(d1+d2))
pixel2 = ((d1*img[y_min, x_max]+d2*img[y_min, x_min])/(d1+d2))

img_out[y_out, x_out] = (d3*pixel2 + d4*pixel1)/(d3+d4)
```

그림 5 보간 과정 2

마지막으로 한 가지를 더 생각해 주어야 하는데 각도가 0도, 90도 와 같이 pi/2의 배수로 떨어지는 부분을 따로 처리해주어야 한다. 이때는 위의 쌍선형 보간 과정에서 R 값이 정수로 무조건 주어지기 때문에 따로 쌍선형 보간 과정을 거칠 필요 없이 출력 영상의 좌표를 쭉 돌면서 offset 만큼 빼준 좌표의 역행렬 값이 원본 이미지 내에 있는지만 판별해 있으면 값을 바로 가져오고 아니면 0으로 처리하는 방법을 통해 픽셀 값을 가져오면 된다.

```
# 0도, 90도, 180도, 270도, 360도인 경우에는 쌍선형 보간 필요X
if ang == 0 or ang == np.pi/2 or ang == np.pi or ang == 3*np.pi/2 or ang == 2 * np.pi:
    for y_out in range(output_height):
        for x_out in range(output_width):
            origin_coord = [x_out + offset_x, y_out + offset_y]
            final_coord  = np.dot(origin_coord, rot_inv)
            if np.floor(final_coord[0]) > 0 and np.ceil(final_coord[0]) < width and np.floor(final_coord[1]) > 0  and np.ceil(final_coord[1]) < height:
                img_out[y_out, x_out] = img[int(final_coord[1]), int(final_coord[0])]
```

그림 6 보간 과정 3

3. 실험 결과 및 분석

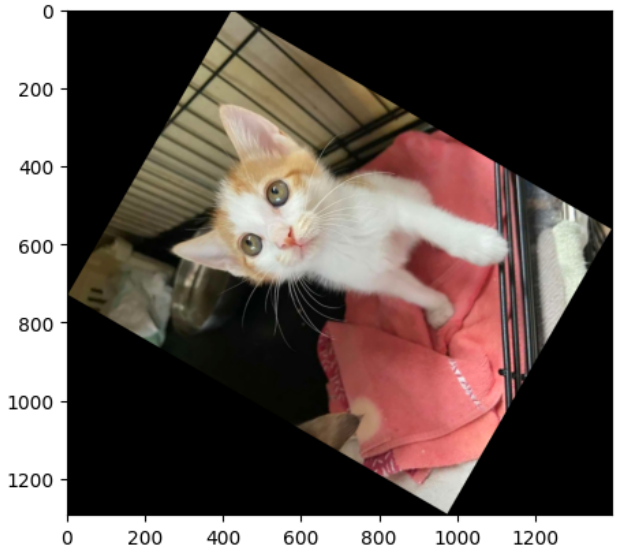

그림 7 예각의 경우 (60도)

그림 8 90도의 경우

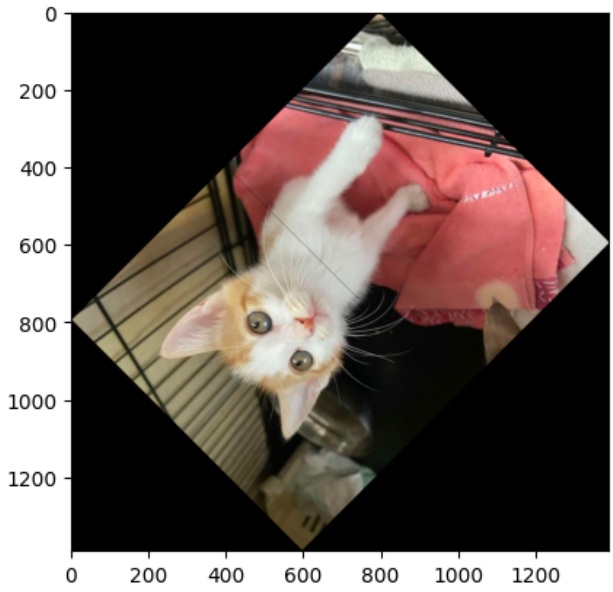

그림 9 둔각의 경우 (135도)

4. 결론

예각, 둔각, 직각의 배수 배 만큼 영상의 회전이 잘 이루어지는 것을 확인할 수 있다. cv2의 warpaffine과 같은 툴이 존재하지만 numpy만을 이용해서 같은 기능을 구현할 수 있음을 보였다.

참고문헌

[1] “선형 보간법과 쌍선형 보간법”, 딥러닝, 패기있게, 2021년 8월 31일 수정, 2023년 5월 8일 접속, https://ballentain.tistory.com/55