Image Rotation

20191467 이재형

국민대학교 전자공학부

harry8121@kookmin.ac.kr

요 약

입력이미지와 각도를 입력하여 입력이미지를 해당 각도만큼 회전하여 출력하는 코드입니다. 회전된 이미지는 billinear interpolation을 사용하여 원본이미지로부터 각 픽셀 값을 조회해오도록 했습니다.

I. 서론

이미지 회전을 위한 회전 행렬에 대한 탐구를 시작으로, 원본 이미지의 4개의 모서리 좌표를 회전 행렬과의 곱을 통해 회전된 이미지의 모서리 좌표를 구했습니다. 그렇게 얻은 좌표의 x, y각각의 최솟값을 이용하여 offset값을 설정하였고, 회전이미지의 크기를 구했습니다.

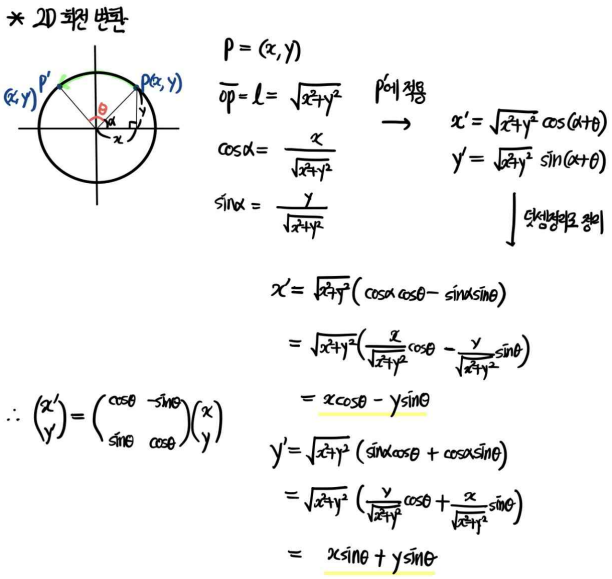
역연산을 통해 구했던 offset을 회전된 이미지의 좌표에서 빼주고 기존의 회전행렬에서 '-각도'를 적용한 행렬곱을 통해 회전된 이미지의 픽셀값을 양선형보간법을 통해 원본 이미지로부터 도출했습니다.

II. 과제 수행 내용

'회전 행렬 탐구 - 회전 이미지의 크기 분석 - 회전된 이미지 픽셀 값 원본 이미지로부터 도출 + 양선형보간법'

위와 같은 순서로 과제 수행 내용에 대하여 기술하겠습니다. (패드를 통해 수기로 작성된 내용에 대한 사진 첨부와 간단한 설명의 방식으로 기술하겠습니다)

2.1 회전 행렬 탐구



회전행렬 도출

2D 회전 변환에서 회전 행렬이 어떻게 도출되는지 먼저 살펴보았습니다. 단위 원상에서 특정 각도 θ로 회전할 때 삼각함수의 덧셈정리를 이용하여 정리한 결과 회전 행렬을 도출할 수 있었습니다.

```
#1. compute rotation matrix
r_matrix = np.array([[np.cos(ang), -np.sin(ang)],
                     [np.sin(ang), np.cos(ang)]])
```

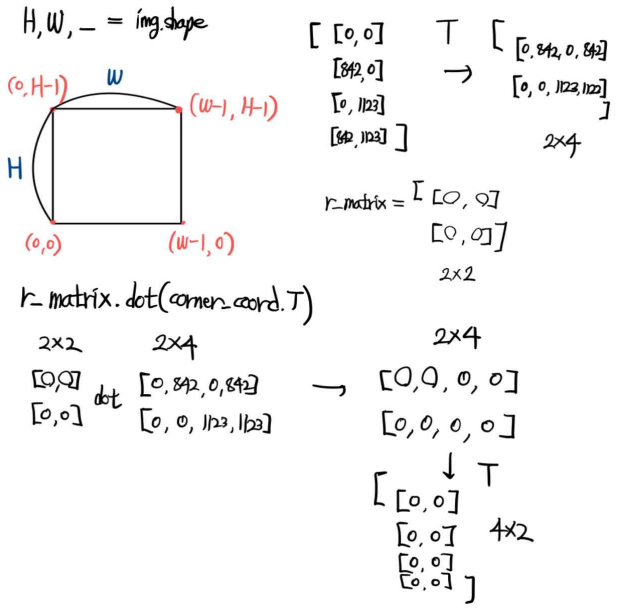
회전행렬 코드

또한, 추후에 회전된 이미지의 픽셀값을 원본 이미지로 부터 도출할 때 역행렬의 연산이 필요한데, 그때의 역행렬은 처음 회전한 각도에 (-)부호를 취함으로써 얻을 수 있음을 파악했습니다. cos, sin의 우, 기함수 특성에 의해 2X2행렬에서 sin의 부호만 바뀐다는 것을 알 수 있습니다.

2.2 회전 이미지 크기 분석

원본 이미지의 4개의 모서리 좌표를 회전 행렬과 곱함으로써 회전된 이미지의 4개의 모서리 좌표를 구할 수 있습니다. 이를 통해 회전된 이미지의 크기를 구했고, 그렇게 구해진 좌표의 x, y각각의 최솟값을 통해서 x_offset과 y_offset을 구했습니다.

밑의 사진과 코드는 전체 과정에 있어서 행렬의 shape를 조정하여 궁극적으로 4X2행렬의 형태에서 세로축을 기준으로 x, y각각의 최솟값을 구하는 과정에 대한 출력과 행렬 그림입니다.



행렬 shape분석

```python
# compute rotation matrix
r_matrix = np.array([[np.cos(ang), -np.sin(ang)],
                     [np.sin(ang), np.cos(ang)]])
# compute output coordinates for image corners
H, W, _ = img.shape
corner_coord = np.array([[0, 0], [W-1, 0],
                         [0, H-1], [W-1, H-1]])
print(corner_coord)
print()

# 회전된 이미지의 모서리 좌표
r_corner = r_matrix.dot(corner_coord.T).T
print(r_corner)

# output image size 구하기
x_min, y_min = np.min(r_corner, axis=0)
x_max, y_max = np.max(r_corner, axis=0)
H_out = int(np.ceil(y_max - y_min))
W_out = int(np.ceil(x_max - x_min))
x_offset = -x_min
y_offset = -y_min

print(H_out, W_out)
```

```
[[   0    0]
 [ 842    0]
 [   0 1123]
 [ 842 1123]]

[[   0.            0.        ]
 [ 421.          729.19338999]
 [-972.54652845  561.5       ]
 [-551.54652845 1290.69338999]]
1291 1394
```

행렬 shpae 출력 + 회전된 이미지 크기 출력

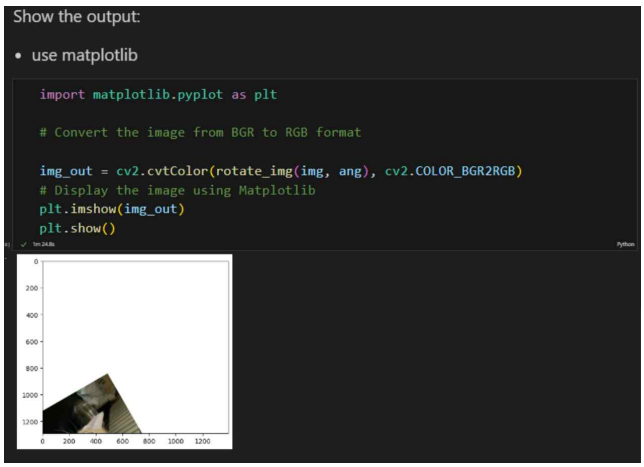
2.3 회전 이미지 픽셀 원본 이미지로부터 도출 & bilinear interpolation

모서리 좌표를 이용하여 회전된 이미지의 크기를 구했고, 회전된 이미지의 픽셀값들을 원본 이미지로부터 도출하는 과정이 필요합니다.

처음엔 오른쪽 코드와 같이 역행렬을 구하고 이를 회전된 이미지 픽셀과 곱하고 offset을 빼주는 형태로 구현을 했습니다. 그렇게 구한 원본 이미지의 픽셀 값으로 양선형 보간법을 진행했습니다.

```
for y_out in range(H_out) :
  for x_out in range(W_out) :
    x, y = np.linalg.inv(r_matrix).dot(np.array([x_out, y_out])) - offset
    x1, y1 = int(np.floor(x)), int(np.floor(y))
    x2, y2 = x1 + 1, y1 + 1
```

하지만, 사진이 도출되는 시간까지 1분이 넘게 소요되었고 도출된 회전된 이미지 또한 오른쪽 사진과 같이 많은 부분이 잘려나오는 것을 확인했습니다. 원인 분석을 해보았습니다.



첫 번째. 연산 수행 속도가 느리게 나온점은 바로 2중 for문안에 np.linalg.inv()를 사용했기 때문이었습니다. 그리고, 여기서 역연산을 위한 역행렬은 처음 회전행렬의 각도만 (-)를 붙이면 되는 것이기에 r_matrix2 행렬을 만들어서 연산을 처리했습니다.

두 번째, 출력 이미지가 잘려나오는 것은 회전 이미지의 픽셀로부터 원본 이미지의 픽셀로 돌아가는 역연산의 수식에 문제가 있었기 때문입니다.
순차적으로 생각해보면 원본 이미지 픽셀에서 회전 이미지로 변환할 때 회전 행렬을 곱하고, offset만큼 평행이동 해준 것이라고 생각할 수 있습니다. 그렇다면, 그것의 역연산이 되려면 평행이동 하고 회전 행렬을 곱해야하는 것입니다. 즉, offset을 회전 이미지 좌표에서 빼준 후에 역회전행렬을 곱해야 하는 것임을 깨달았습니다.

```
# 회전행렬의 역행렬
r_matrix2 = np.array([[np.cos(ang), np.sin(ang)],
                      [-np.sin(ang), np.cos(ang)]])
```

위의 2가지 부분을 보완하여 작성한 코드는 오른쪽과 같습니다.

```
for y_out in range(H_out) :
  for x_out in range(W_out) :
    x, y = np.dot(r_matrix2, (np.array([x_out, y_out]) - offset))
    x1, y1 = int(np.floor(x)), int(np.floor(y))
    x2, y2 = x1 + 1, y1 + 1
```

양선형보간법은 교수님께서 수업시간에 알려주신 것과 앞선 과정에서 배운 것들을 종합해서 손쉽게 구현할 수 있었습니다.
가중 평균의 의미를 생각하여 픽셀 값이 조건에 부합할 때 출력 이미지의 픽셀값을 구했고, 그렇지 않다면 255를 부여하여 흰색으로 표현했습니다.

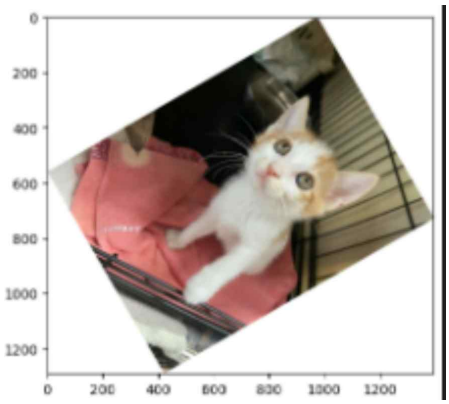
```
x, y = np.dot(r_matrix2, (np.array([x_out, y_out]) - offset))
x1, y1 = int(np.floor(x)), int(np.floor(y))
x2, y2 = x1 + 1, y1 + 1

if x1 >= 0 and x2 < img.shape[1] and y1 >= 0 and y2 < img.shape[0] and x >= 0 and y >= 0 :
  img_out[y_out, x_out] = (
    (x2 - x)*(y2 - y)*img[y1, x1] + (x - x1)*(y2 - y)*img[y1, x2] +
    (x2 - x)*(y - y1)*img[y2, x1] + (x - x1)*(y - y1)*img[y2, x2]
  )

else :
  img_out[y_out, x_out] = 255
```

3. 실험 결과 및 분석

이와 같이 rotate_img()함수를 구현하여 이미지를 출력해보면, 각도에 따라 회전된 이미지가 잘 출력되었습니다.
추가적으로 openCV의 함수를 이용하여 이미 회전을 구현하는 코드도 작성해보았습니다.

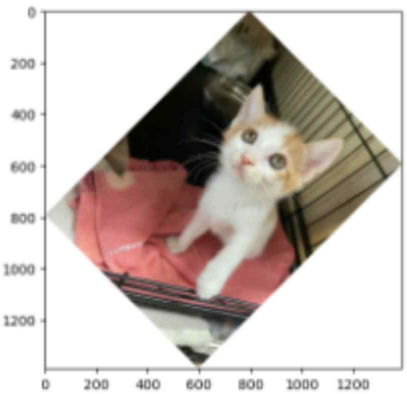
openCV의 getRotationMatrix2D()함수를 통해 회전 행렬을 구하고, warpAffine()에 이미지와, 회전 행렬, 이미지의 너비와 폭을 인자로 집어넣어서 출력하면 각도에 회전행렬에 지정된 각도만큼 이미지가 회전되어 훌력됨을 확인할 수 있습니다.

warpAffine() 함수 내에 양선형 보간법이 적용되어 출력되는데, 제가 직접 구현한 함수를 이용했을 때보다 출력 이미지가 도출되는 속도의 차이가 컸습니다. 추후에 openCV에서 warpAffine의 함수가 어떻게 구현되어있는지를 통해서 성능 차이를 분석해볼 수 있음을 생각해보았습니다.



60도 회전

90도 회전



45도 회전

```python
# openCV함수 활용해서 회전시키기
h, w, _ = img.shape
img2 = img.copy()

matrix = cv2.getRotationMatrix2D((w/2, h/2), 90, 1)
rotate_img = cv2.warpAffine(img2, matrix, (w, h))

cv2.imshow('rotation_img', rotate_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

I. 결론

openCV의 함수를 사용하면 몇줄 되지 않는 크게 어렵지 않은 이미지의 회전을 numpy를 이용하여 행렬의 연산을 통해 생각을해보고, 회전된 이미지의 픽셀을 원본 이미지로부터 가져오는 방법에 대한 고찰과, 양선형 보간법을 직접적으로 적용하여 회전된 이미지를 출력해봄으로써 2D기반 회전의 기초적인 지식과 로직에 대해 알 수 있었습니다.

참고문헌

[1] ChatGPT

재형 이미지와 각도가 매개변수로 주어졌을 때 각도만큼 회전된 이미지를 양선형보간법을 이용하여 구하는 함수를 만들어 봤는데 코드좀 체크해줘

```
def rotate_img(img, ang):
 #1. compute rotation matrix
 r_matrix = np.array([[np.cos(ang), -np.sin(ang)],
                      [np.sin(ang), np.cos(ang)]])
 #2. compute output coordinates for image corners
 H, W, _ = img.shape
 corner_coord = np.array([[0, 0], [W-1, 0],
                          [0, H-1], [W-1, H-1]])
 # 회전된 이미지의 모서리 좌표
 r_corner = r_matrix.dot(corner_coord.T).T
 #print(r_corner)

 #3. compute output image size and offset
 # output image size 구하기 -> height, width길이 정수값으로 구하기
 x_min, y_min = np.min(r_corner, axis=0)
 x_max, y_max = np.max(r_corner, axis=0)
 H_out = int(np.ceil(y_max - y_min))
 W_out = int(np.ceil(x_max - x_min))

 # offset 구하기 -> 음수값으로 나온 좌표를 offset만큼 이동시켜 보정한다
 x_offset = -x_min
 y_offset = -y_min
 offset = np.array([x_offset, y_offset])
```

코드는 대체로 정확해 보입니다. 하지만 몇 가지 수정 사항이 있습니다.

첫째, x1, y1, x2, y2 변수를 계산할 때, 입력 이미지 크기를 사용해야 합니다. 따라서 `W`와 `H` 대신 `img.shape[1]`과 `img.shape[0]`을 사용해야합니다.

둘째, 보간법에서 `x2`와 `y2`가 입력 이미지의 크기보다 크거나 같을 수 있으므로 if 문의 조건에 `x2 < W`와 `y2 < H`를 추가해야 합니다.

셋째, `img_out`의 모든 픽셀 값을 255로 설정하지 않아도됩니다. 대신 `img_out`에 대한 좌표가 입력 이미지 범위를 벗어나는 경우 보간 대신 `img_out`의 해당 픽셀을 무시하면됩니다.

아래는 수정된 코드입니다.


```
def rotate_img(img, ang):
  #1. compute rotation matrix
  r_matrix = np.array([[np.cos(ang), -np.sin(ang)],
                       [np.sin(ang), np.cos(ang)]])
  #2. compute output coordinates for image corners
  H, W, _ = img.shape
  corner_coord = np.array([[0, 0], [W-1, 0],
                           [0, H-1], [W-1, H-1]])
  # 회전된 이미지의 모서리 좌표
  r_corner = r_matrix.dot(corner_coord.T).T
  #print(r_corner)

  #3. compute output image size and offset
  # output image size 구하기 -> height, width길이 정수
  x_min, y_min = np.min(r_corner, axis=0)
  x_max, y_max = np.max(r_corner, axis=0)
  H_out = int(np.ceil(y_max - y_min))
  W_out = int(np.ceil(x_max - x_min))

  # offset 구하기 -> 음수값으로 나온 좌표를 offset만큼
  x_offset = -x_min
  y_offset = -y_min
  offset = np.array([x_offset, y_offset])

  # define output image object
  img_out = np.zeros((H_out, W_out, 3), dtype=img.dty

  for y_out in range(H_out) :
    for x_out in range(W_out) :
      x, y = np.linalg.inv(r_matrix).dot(np.array([x_
      x1, y1 = int(np.floor(x)), int(np.floor(y))
      x2, y2 = x1 + 1, y1 + 1

      if x1 >= 0 and x2 < img.shape[1] and y1 >= 0 an
        img_out[y_out, x_out] = (
          (x2 - x)*(y2 - y)*img[y1, x1] + (x - x1)*(y
          (x2 - x)*(y - y1)*img[y2, x1] + (x
```

- rotate_img()함수를 구현하고, 처음에 사진이 잘려서 출력된 결과로 나와서 GPT에게 코드에 대한 피드백을 받았습니다.

[2] 선형보간법 개념

- https://ballentain.tistory.com/55