rotation_technical_report

introduction

인풋 이미지를 rotation하는 과정에서 bilinear interpolation 혹은 nearest-neighbor interpolation을 선택하여 새로운 이미지를 생성한다.

코드 구현

main 함수

```
Mat input, rotated;

// Read each image
input = imread("lena.jpg");

// Check for invalid input
if (!input.data) {
    std::cout << "Could not open or find the image" << std::endl;
    return -1;
}

// original image
namedWindow("image");
imshow("image", input);
```

- imread 함수를 이용하여 lena.jpg 이미지를 로드한다
- 해당 이미지가 없을 시, “Could not open or find the image”를 출력한다
- 보간법 적용 전 오리지널 이미지를 imshow 함수를 이용해 이미지 출력

myrotate 함수

```
int row = input.rows;
int col = input.cols;

float radian = angle * CV_PI / 180;

float sq_row = ceil(row * sin(radian) + col * cos(radian));
float sq_col = ceil(col * sin(radian) + row * cos(radian));

Mat output = Mat::zeros(sq_row, sq_col, input.type());
```

- row, col → 인풋 이미지의 row, col 값
- radian → angle을 라디안으로 변형
- sq_row, sq_col → 회전 이후의 이미지의 row 값, col 값 / rotation matrix * row, col

```
for (int i = 0; i < sq_row; i++) {
    for (int j = 0; j < sq_col; j++) {
        float x = (j - sq_col / 2) * cos(radian) - (i - sq_row / 2) * sin(radian) + col /
        float y = (j - sq_col / 2) * sin(radian) + (i - sq_row / 2) * cos(radian) + row
        }
    }
}
```

Inverse warping

- float x, float y → 회전 이후의 좌표* 회전 행렬 => float으로 표현된 원본 좌표 구하기
    - (j - sq_col / 2) , (i - sq_row / 2) → 회전 중심을 (0,0)으로 이동하기 위해 픽셀 좌표 변환
    - 회전 변환 적용

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

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

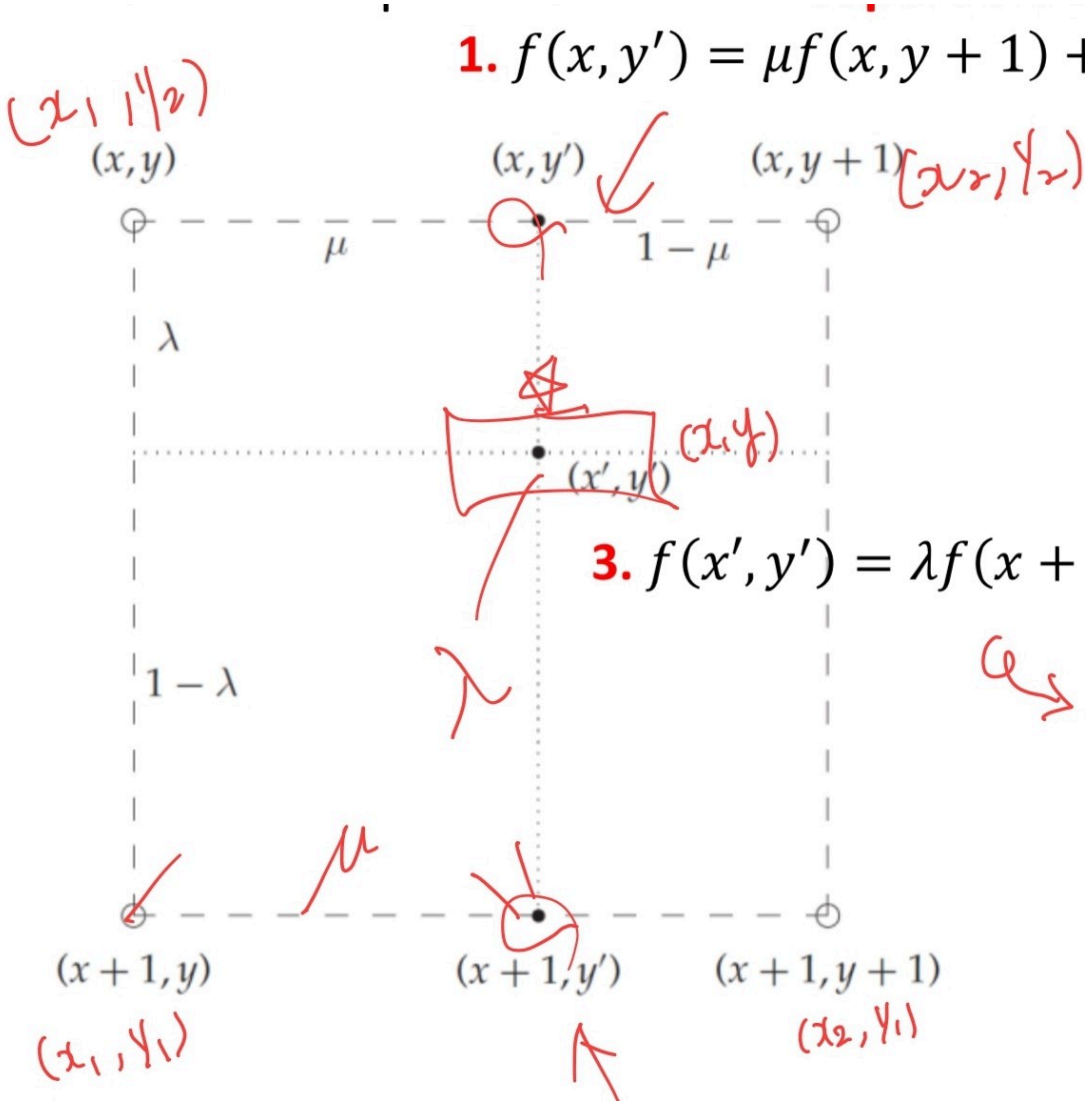
- + col / 2 → 다시 원래 좌표계로 되돌리기 위해 보정함

```
if ((y >= 0) && (y <= (row - 1)) && (x >= 0) && (x <= (col - 1))) { //float으로 바꾼
        if (!strcmp(opt, "nearest")) { //주변 값 복사
            int nearX = round(x);
            int nearY = round(y);

            output.at<T>(i, j) = input.at<T>(nearY, nearX); //opencv 좌표계!!

        }
        else if (!strcmp(opt, "bilinear")) { //보간법
            int x1 = floor(x);
            int x2 = ceil(x);
            int y1 = floor(y);
            int y2 = ceil(y);

            float mu = x - x1;
            float lam = y - y1;

            T f_x_y1 = mu * input.at<T>(y2, x2) + (1 - mu) * input.at<T>(y2, x1);
            T f_x1_y = mu * input.at<T>(y1, x2) + (1 - mu) * input.at<T>(y1, x1);
            T f_x1_y1 = lam * f_x_y1 + (1 - lam) * f_x1_y;

            output.at<T>(i, j) = f_x1_y1;

        }
```

Interpolation 방법

- nearest - neighbor
  - blocky한 곳을 채우기 위해, 주변 정수 값 픽셀 복사
- bilinear
  - x1, y1, x2, y2의 좌표를 target 좌표인 (x, y) 주변의 정수 좌표로 두고,

    mu와 lamda를 계산한다



- 해당 보간법을 이용해서 target 좌표를 구할 수 있다

Result

bilinear interpolation과 nearest-neighbor interpolation을 각각 적용한 이미지에는 차이가 있었다.

nn의 경우, 주변의 점을 복사하는 형식이라 이미지가 상대적으로 부자연스럽게 출력된다.

bilinear의 경우, 선형적으로 계산하기 때문에 nn보다 자연스럽게 출력됨을 알 수 있다.