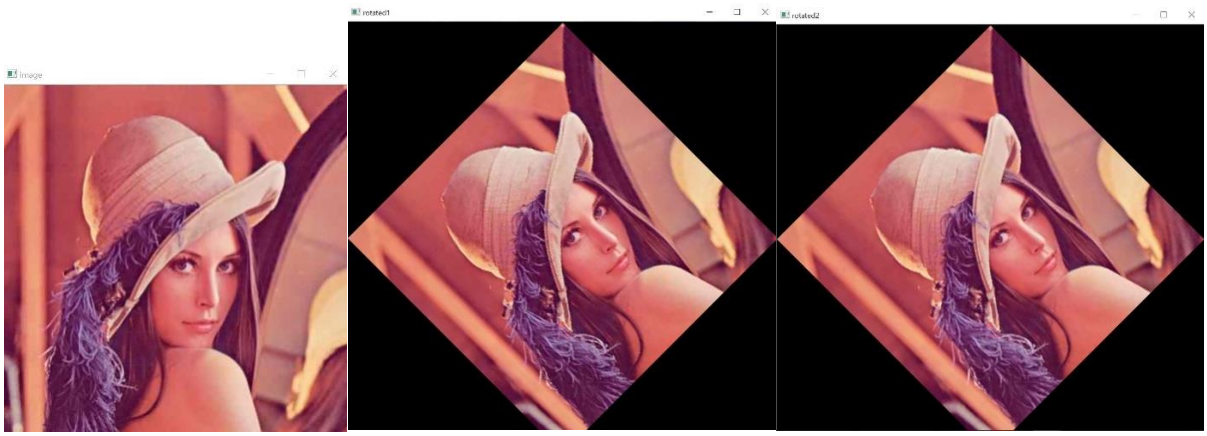
Technical Report

1829008 김민영

<1> Practice1

Implement the Image rotation

1. result image



Input - Image(기본이미지) **output- rotated1:nearest 이용** **output -rotated2: bilinear 이용**

Image 가 기본이미지로 들어왔을 때, 각각 nearest, bilinear interpolation 을 이용하여 입력한 회전각도로 회전시킨다.

2. Explanation of code

- main 함수 : input 이미지를 읽어오고, 회전한 이미지(output)를 show 한다.

- Mat myRotate : nearest interpolation, bilinear interpolation 을 이용하여 input 이미지를 회전시킨다.

1) nearest interpolation

```
if (!strcmp(opt, "nearest")) {
    float lammda1 = x - x1;
    float lammda2 = y - y1;

    // x가 x1에 더 가까우면
    if (lammda1 <= 0.5) {
        //y가 y1에 더 가까우면
        if (lammda2 <= 0.5) {
            output.at<Vec3b>(i, j) = input.at<Vec3b>(y1, x1);
        }
        // y가 y2에 더 가까우면
        else {
            output.at<Vec3b>(i, j) = input.at<Vec3b>(y2, x1);
        }
    }

    // x가 x2에 더 가까우면
    else {
        //y가 y1에 더 가까우면
        if (lammda2 <= 0.5) {
            output.at<Vec3b>(i, j) = input.at<Vec3b>(y1, x2);
        }
        // y가 y2에 더 가까우면
        else {
            output.at<Vec3b>(i, j) = input.at<Vec3b>(y2, x2);
        }
    }
```

Output 에 (x1,y1) (x1,y2) (x2,y1) (x2,y2) 중에서 더 가까운 것을 붙여쓴다.

2) bilinear interpolation

```
else if (!strcmp(opt, "bilinear")) {
    // 각 네점(point)
    Vec3b f1 = input.at<Vec3b>(y1, x1);
    Vec3b f2 = input.at<Vec3b>(y1, x2);
    Vec3b f3 = input.at<Vec3b>(y2, x1);
    Vec3b f4 = input.at<Vec3b>(y2, x2);

    // mu, lambda
    float mu = y - y1;
    float lambda = x - x1;

    // 1. 2. 3. 순서로 점을 구하는 과정
    Vec3b ff1 = mu*f3 + (1-mu)*f1;
    Vec3b ff2 = mu*f4 + (1 - mu)*f2;
    Vec3b ff3 = lambda*ff2 + (1-lambda)*ff1;

    // 예측한 ff3 값을 output Mat에 넣는다
    output.at<Vec3b>(i, j) = ff3;
}
```

먼저 floor, ceil 로 구한 x1, x2, y1, y2 를 이용하여 각 네 점 f1, f2, f3, f4 를 구한다. 이후 mu 와 lambda 값을 구한다. 1.2.3. 순서로 점을 구한다. 최종적으로 구한 ff3 을 output 에 넣는다.

3. Anaysis (전체적인 과정)

1. Define an enlarged, rotated image output(original image 보다 크게)

2. Assume(x,y) then transform

3. bilinear, nearest interpolation 수행

+ Bilinear interpolation 이 Nearest neighbor interpolation 보다 좀더 퀄리티가 좋은 것을 확인할 수 있는데 이런 이유 때문에 Bilinear interpolation 을 Nearest neighbor interpolation 보다 더 자주 사용한다고 한다.

<2>Practice2

Implement the Image stitching using Affine Transformation

1. result image

<I1 : StitchingL >

<I2 : stitchingR >

<I_f : output image>

Image1, Image2 를 입력했을 때 위와 같이 image stitching 한다.

2. Explanation Code

- main 함수 : Img1 과 Img2 를 읽어온다. affine transform 을 한 뒤에, bilinear interpolation 으로 inverse warping 을 한다. 이후 blend_stitching 함수를 호출시키고 I_f 이미지를 show 한다.

- Mat cal_affine : 행렬 연산을 사용하여 affineM 을 구한다.

- void blend_stitching : Blend two Images

1) Inverse warping with bilinear interpolation

```
// inverse warping with bilinear interplolation
for (int i = -diff_x; i < I_f.rows-diff_x_; i++) {
    for (int j = -diff_y; j < I_f.cols-diff_y_; j++) {
        float x = A12.at<float>(0) * i + A12.at<float>(1) * j + A12.at<float>(2) + diff_x_;
        float y = A12.at<float>(3) * i + A12.at<float>(4) * j + A12.at<float>(5) + diff_y_;

        float y1 = floor(y);
        float y2 = ceil(y);
        float x1 = floor(x);
        float x2 = ceil(x);

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

        if (x1 >= 0 && x2 < I2_row && y1 >= 0 && y2 < I2_col) {
            Vec3f f1 = I2.at<Vec3f>(x1, y1);
            Vec3f f2 = I2.at<Vec3f>(x2, y1);
            Vec3f f3 = I2.at<Vec3f>(x1, y2);
            Vec3f f4 = I2.at<Vec3f>(x2, y2);

            Vec3f ff1 = mu*f3 + (1 - mu)*f1;
            Vec3f ff2 = mu*f4 + (1 - mu)*f2;
            Vec3f ff3 = lambda*ff2 + (1 - lambda)*ff1;

            I_f.at<Vec3f>(i + diff_x_, j + diff_y_) = ff3;
        }
    }
}
```

Practice1 에서와 마찬가지로 f1, f2, f3, f4 점을 I2 에서 가져온 뒤, mu 를 이용하여 ff1, ff2 점을 구하고 lambda 를 이용하여 ff3 점을 구한다. 이후 I_f(i+diff_x, j+diff_y_)에 앞에서 구한 ff3 점을 넣는다.

2) cal_affine

```
Mat cal_affine(int ptl_x[], int ptl_y[], int ptr_x[], int ptr_y[], int number_of_points) {

    Mat M(2 * number_of_points, 6, CV_32F, Scalar(0));
    Mat b(2 * number_of_points, 1, CV_32F);

    Mat M_trans, temp, affineM;

    // initialize matrix
    for (int i = 0; i < number_of_points; i++) {

        int ptl1[6] = { ptl_x[i],ptl_y[i],1,0,0,0 };
        int ptl2[6] = { 0,0,0,ptl_x[i],ptl_y[i],1 };

        for (int j = 0; j < 6; j++) {
            M.at<float>(2 * i, j) = ptl1[j];
            M.at<float>(2 * i + 1, j) = ptl2[j];
        }

        b.at<float>(2 * i, 0) = ptr_x[i];
        b.at<float>(2 * i + 1, 0) = ptr_y[i];
    }

    M_trans = M.t();
    temp = (M_trans * M).inv();
    affineM = temp * M_trans*b;

    return affineM;
}
```

우선 M, b 를 다음과 같이 initialize 를 하였다.

$$\underset{\boldsymbol{M}}{\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ \vdots & & & & & \\ x_N & y_N & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_N & y_N & 1 \end{pmatrix}} \underset{\boldsymbol{x}}{\begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix}} = \underset{\boldsymbol{b}}{\begin{pmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ \vdots \\ x_N' \\ y_N' \end{pmatrix}}$$

이후 AffineM 을 구하기 위해 아래와 같은 행렬연산을 사용했다.

$\boldsymbol{Mx} = \boldsymbol{b} \rightarrow \boldsymbol{x} = (\boldsymbol{M}^{\mathrm{T}}\boldsymbol{M})^{-1}\boldsymbol{M}^{\mathrm{T}}\boldsymbol{b}$

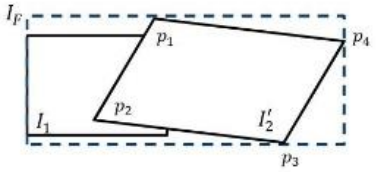
Temp 에 $(\boldsymbol{M}^{\mathrm{T}}\boldsymbol{M})^{-1}$ 를 저장하고, affine 에 최종 연산을 한 x 값을 넣어 이를 반환한다.

3. Analysis (전체적인 과정)

1. Estimate affine transform

2. Merge two images(I2 to I1)

- A21 을 사용하여 p1,p2,p3,p4 추정.



- A12 를 사용하여 위에서 구한 corners [p1,p2,p3,p4] Inverse warping

- Blend two Images (여기서 if only I2′ is valid 부분은 inverse warping 으로 구하고 생략한다.)

$$I_F = \begin{cases} \alpha I_1 + (1-\alpha) I_2' & if\ both\ I_1\ and\ I_2\ are\ valid \\ I_1 & if\ only\ I_1\ is\ valid \\ I_2' & if\ only\ I_2'\ is\ valid \\ 0 & otherwise \end{cases}$$