10nd Week Lab Assignment

12201928 이상혁

Task 1

❑ getRotationMatrix()과 동일한 getMyRotationMatrix()함수를 직접 구현하고 두 결과가 동일한지 검증하라

❑ Scale 변화는 구현하지 않아도 됨

❑ 45 도 변화 결과가 동일한지 비교하면 됨

```
Mat getMyRotationMatrix(Point center, double angle) {
    double radians = angle * CV_PI / 180;
    double alpha = cos(radians);
    double beta = sin(radians);

    Mat matrix = (Mat_<double>(2, 3) <<
        alpha, beta, (1 - alpha) * center.x - beta * center.y,
        -beta, alpha, beta * center.x + (1 - alpha) * center.y);

    return matrix;
}

void Task1() {
    Mat getMyRotationMatrix(Point center, double angle);
    Mat src = imread("10/Lenna.png");
    if (src.empty()) {
        cerr << "Error loading the image!" << endl;
        return;
    }

    Point center = Point(src.cols / 2, src.rows / 2);
    Mat matrix = getRotationMatrix2D(center, 45.0, 1.0);
    Mat mymatrix = getMyRotationMatrix(center, 45.0);
    Mat dst, mydst;

    warpAffine(src, dst, matrix, src.size());
    warpAffine(src, mydst, mymatrix, src.size());

    imwrite("rot.jpg", dst);
    imwrite("myrot.jpg", mydst);


    imshow("rot.jpg", dst);
    imshow("myrot.jpg", mydst);
    waitKey(0);
    destroyAllWindows();
}
```

코드의 전반적인 부분은 이미 구현되어 있었으므로 코드 분석에 중점을 두어 Task 를 진행해 보았다.

먼저 center 에 이미지의 중심을 구한 후 저장을 한다. 그다음 구현되어있는 get 을 이용하여 이미지를 돌린 후 저정 한다. 그다음 myrataion 을 사용하여 동일하게 45 를 돌린 후 실행한다.

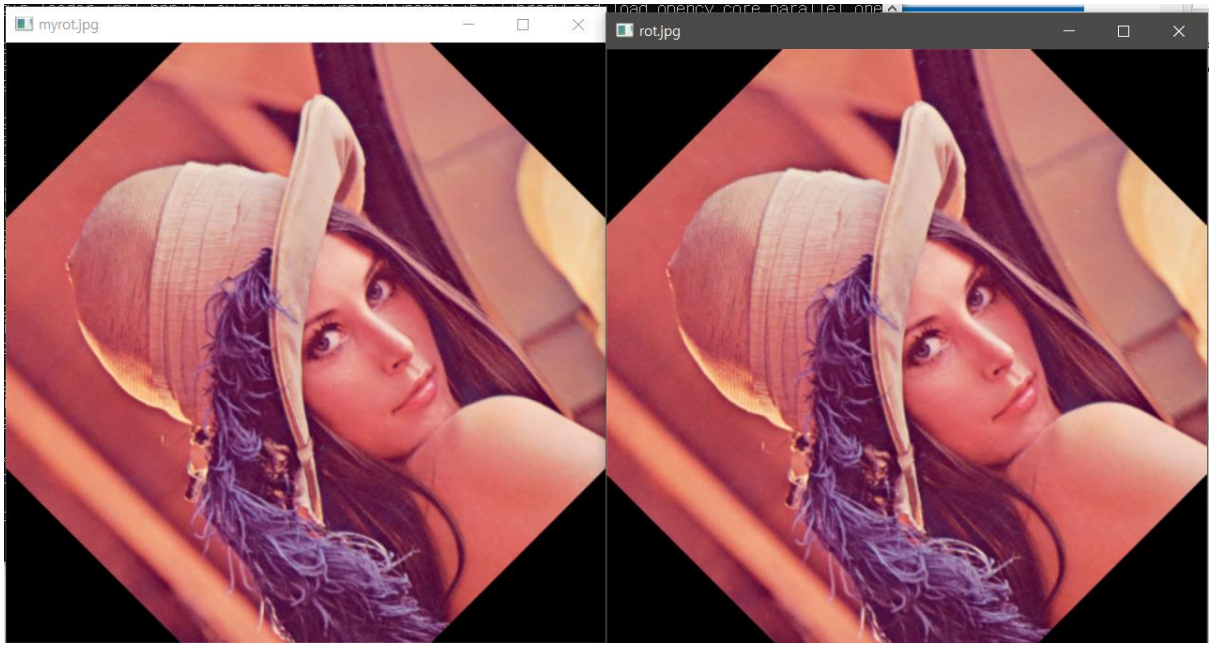
Rotation code 를 봐보자

```
Mat getMyRotationMatrix(Point center, double angle) {
    double radians = angle * CV_PI / 180;
    double alpha = cos(radians);
    double beta = sin(radians);

    Mat matrix = (Mat_<double>(2, 3) <<
        alpha, beta, (1 - alpha) * center.x - beta * center.y,
        -beta, alpha, beta * center.x + (1 - alpha) * center.y);

    return matrix;
}
```

우선 입력받은 각도를 라디안단위로 변환한다.

그 다음 라디안 값으로 cos, sin 을 계산해 2x3 의 행렬을 생성하게 된다. 이는 회전행렬로 affine transformation 을 나타낸다. 행렬을 간단하게 살펴보면 첫번째 행은 회전과 x 축 이동에 대한 정보, 두번째 행은 회전과 y 축 이동에 대한 정보를 가지고 있다. 마지막 열은 이미지의 중심과 회전한 이미지의 중심을 일치하기 위해 아까 구한 center 를 사용하여 유지시킨다.

이와 같은 방법으로 수행한 결과는 아래와 같다.



Task 2

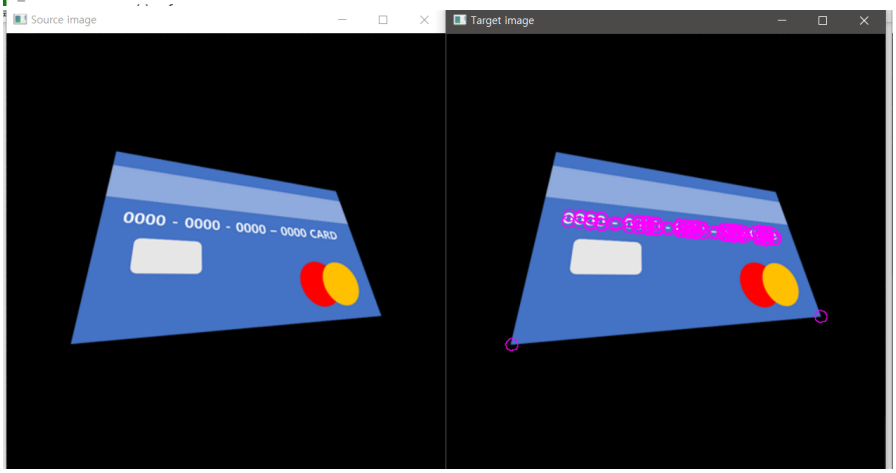
❑card_per.png 영상을 getPerspectiveTransform() 함수를이용해카드의면이시선정면을향하도록정렬시켜라.

❑ 입력매개변수를구하기위해네꼭지점을직접찾으면부분점수

❑ 지난실습및과제를참고해네꼭지점을자동으로탐색하도록만들면만점

❑ 둘중어떠한방법으로구현했는지반드시잘보이게명시할것!

```
Mat cvHarrisCorner() {
    Mat img = imread("10/Card_per.png");
    resize(img, img, Size(500, 500), 0, 0, INTER_CUBIC);

    Mat gray;
    cvtColor(img, gray, COLOR_BGR2GRAY);
    Mat harr;
    cornerHarris(gray, harr, 2, 3, 0.05, BORDER_DEFAULT);
    normalize(harr, harr, 0, 255, NORM_MINMAX, CV_32FC1, Mat());
    Mat harr_abs;
    convertScaleAbs(harr, harr_abs);

    int thresh = 125;
    Mat result = img.clone();
    for (int y = 0; y < harr.rows; y += 1) {
        for (int x = 0; x < harr.cols; x += 1) {
            if ((int)harr.at<float>(y, x) > thresh)
                circle(result, Point(x, y), 7, Scalar(255, 0, 255), 0, 4, 0);
        }
    }

    imshow("Source image", img);
    imshow("Harris image", harr_abs);
    imshow("Target image", result);
    waitKey(0);
    destroyAllWindows();
    return result;
}
```

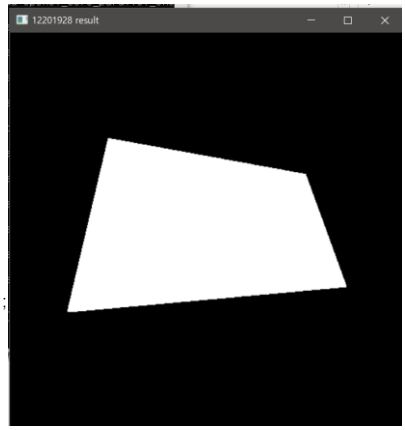


우선 저번 실습시간에 진행하였던 Harris Corner 를 이용해 카드의 코너부분을 찾으려 하였다.
그러나 위와 같이 카드의 아래쪽의 두 코너 부분은 잘 감지하였으나, 카드 내부의 번호 부분들을 모두 코너로

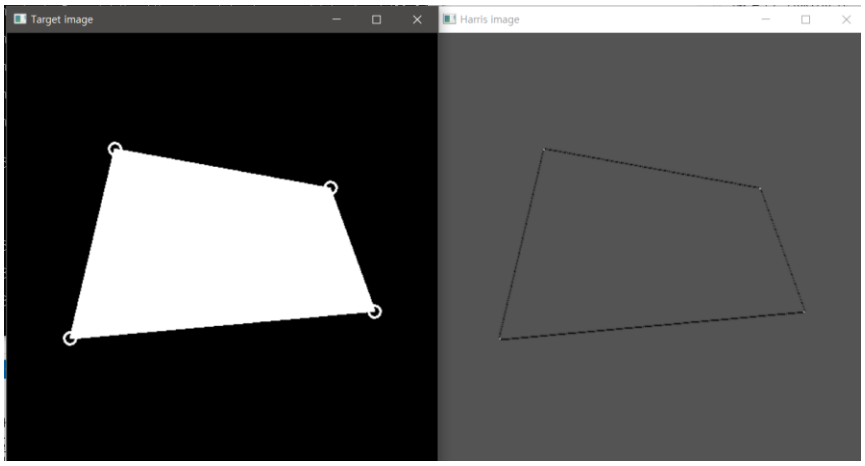
인식하는 오류가 발생하였다. 따라서 내부의 코너를 신경쓰지않고 물체 바깥쪽만 디텍하기 위하여 boundary 만을 검출하여야 했다.

따라서 8 주차 실습에서 진행하였던 grabcut 부분에서 mask 부분을 이용하여 코드를 작성해 보기로 하였다.

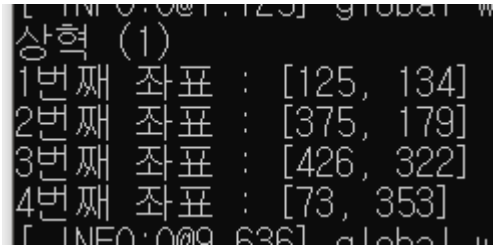
```
Mat img = imread("10/card_per.png", 1);
resize(img, img, Size(500, 500), 0, 0, INTER_CUBIC);
Rect rect(10, 10, 480, 480);
Mat result, bg_model, fg_model;
grabCut(img, result, rect, bg_model, fg_model, 5, GC_INIT_WITH_RECT);
compare(result, GC_PR_FGD, result, CMP_EQ);
imshow("이상혁",result);
waitKey(0);
cvHarrisCorner(result);
waitKey(0);
```



위와 같은 결과를 얻을 수 있었다. 위 사진을 이용하여 harrsonCorner 를 진행한 결과는 아래와 같다.



이 과정에서 생기는 원을 모두 vector 에 저장하였더니 비슷하게 겹치는 부분이 3 개씩 생성되었다. 따라서 vector 에 저장하는 과정에서 전에 들어온 값에서 5 정도 큰값이 들어오는 순간에만 저장이 되고 원이 생기도록 하였다.



왼쪽위가 1 번째 오른쪽위가 2 번째 오른쪽 하단이 3 번째 왼쪽 하단이 4 번째 좌표로 잘 표시됨을 확인할 수 있다.

코드는 아래와 같다.

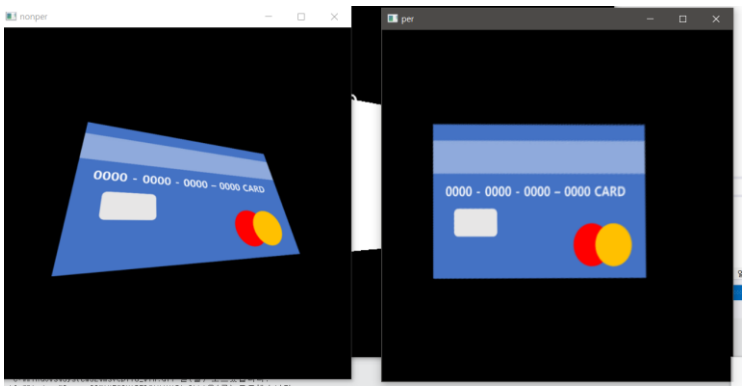
```
Mat cvHarrisCorner(Mat img) {
    Mat src = imread("10/card_per.png", 1);
    Mat harr;
    cornerHarris(img, harr, 2, 3, 0.05, BORDER_DEFAULT);
    normalize(harr, harr, 0, 255, NORM_MINMAX, CV_32FC1, Mat());
    Mat harr_abs;
    convertScaleAbs(harr, harr_abs);

    int thresh = 125;
    Mat result = img.clone();
    vector<Point> corner_points;
    int minDist = 5;
    for (int y = 0; y < harr.rows; y += 1) {
        for (int x = 0; x < harr.cols; x += 1) {
            if ((int)harr.at<float>(y, x) > thresh) {
                Point candidateCorner = Point(x, y);
                bool tooClose = false;
                for (auto storedCorner : corner_points) {
                    double dist = norm(candidateCorner - storedCorner);
                    if (dist < minDist) {
                        tooClose = true;
                        break;
                    }
                }
                if (!tooClose) {
                    corner_points.push_back(candidateCorner);
                    circle(result, candidateCorner, 7, Scalar(255, 0, 255), 2, 8, 0);
                }
            }
        }
    }
    cout << "1번째 좌표 : " << corner_points[0] << endl;
    cout << "2번째 좌표 : " << corner_points[1] << endl;
    cout << "3번째 좌표 : " << corner_points[2] << endl;
    cout << "4번째 좌표 : " << corner_points[3] << endl;
    imshow("Source image", img);
    imshow("Harris image", harr_abs);
    imshow("Target image", result);
    waitKey(0);
```

이제 이 좌표를 각각 입력한후에 perpectiveform 을 이용하여 warp 시켜야한다.

```
Mat dst, matrix;
Point2f srcQuad[4];
srcQuad[0] = corner_points[0];
srcQuad[1] = corner_points[1];
srcQuad[2] = corner_points[2];
srcQuad[3] = corner_points[3];
```

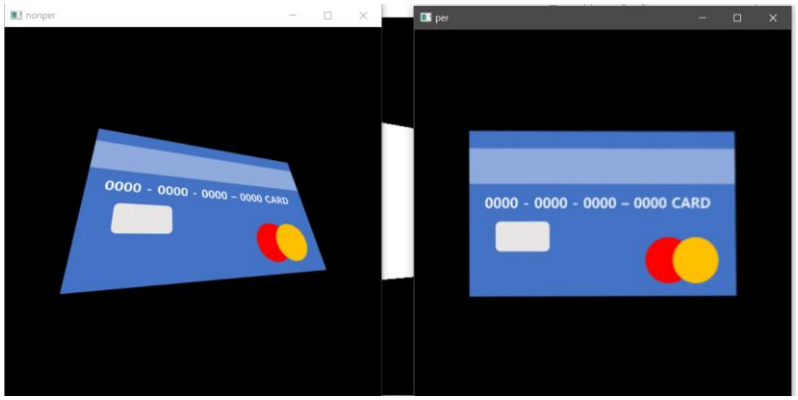
이렇게 src 좌표값들을 아까 구한값으로 정의를 해준다면 준비는 끝이다 dst 에 src 값을 각각 매칭만 해주면 끝이난다.

```
Point2f dstQuad[4];
dstQuad[0] = Point2f(corner_points[3].x, corner_points[0].y);
dstQuad[1] = Point2f(corner_points[1].x, corner_points[0].y);
dstQuad[2] = Point2f(corner_points[1].x, corner_points[3].y);
dstQuad[3] = corner_points[3];
```



이런 방식으로 매칭을 해주니 약간 완성은 얼추되었지만 뭔가 가로폭이 줄어든 느낌이 난다. 따라서 아래처럼 오른쪽 x 값을 2 로 바꿔주니 원하는 모양이 출력되었다.

```
Point2f dstQuad[4];
dstQuad[0] = Point2f(corner_points[3].x, corner_points[0].y);
dstQuad[1] = Point2f(corner_points[2].x, corner_points[0].y);
dstQuad[2] = Point2f(corner_points[2].x, corner_points[3].y);
dstQuad[3] = corner_points[3];
```



따라서 harrsion corner 를 이용하고 각 좌표값을 저장함 뒤에 이미지를 원래대로 warp 시키는 모든 과정을 수행하였다. 다른 비슷한 형태의 이미지를 가지고와도 잘 실행될것이다.