[Computer Vision Programming] Lab2. Lens distortion

전자공학과 21611591 김난희

1. 첨부 영상(input.jpg)에 대해 왜곡을 제거하기

exerciseCV (전역 범위)

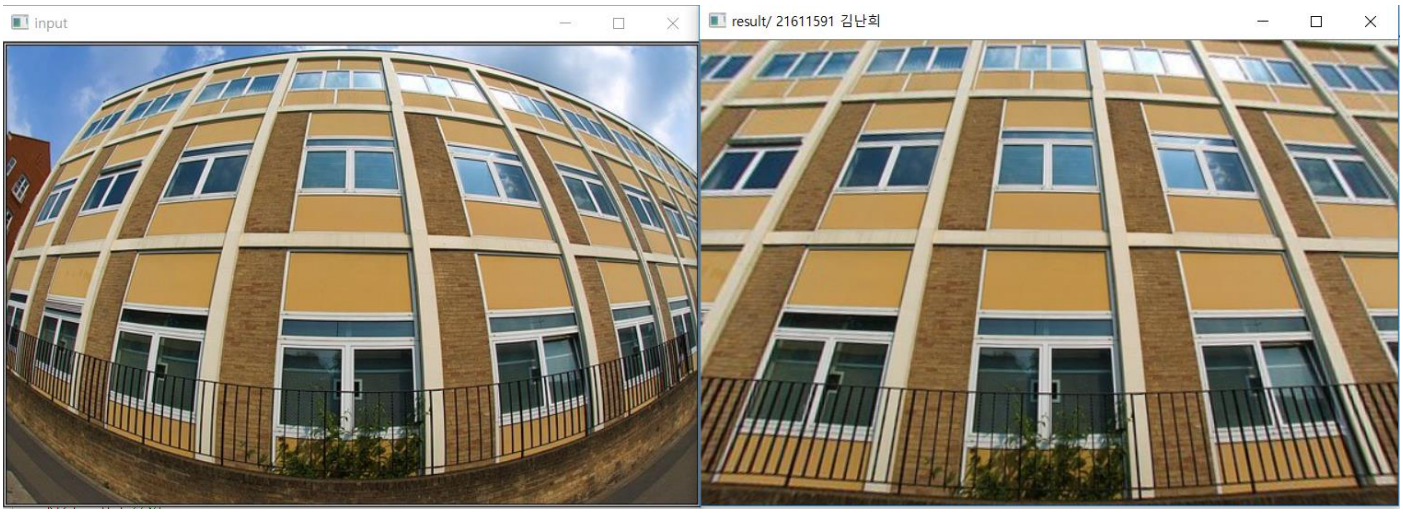
```
#include "opencv2/opencv.hpp"//openCV lib
#include "opencv2/imgproc/imgproc.hpp" //image proc lib
using namespace cv;
//21611591 김난희
int main(void)
{
    //load an image
    Mat in = imread("C:₩₩Users₩₩난희₩₩Documents₩₩Visual Studio 2017₩₩Projects₩₩exerciseCV₩₩exerciseCV₩₩input.jpg");
    imshow("input", in);

    //set camera parameters
    float x0 = 606.0/2.0; //Cx 영상 중심의 x좌표
    float y0 = 404.0/2.0; //Cy 영상 중심의 y좌표
    float fx = 1108.0; //[pixel] //카메라 focal length
    float fy = 1108.0; //[pixel] //카메라

    //객체 생성 후 insertion operator 써서 행렬 채워넣음
    Mat cameraMatrix = (Mat_<float>(3, 3) << fx, 0, x0, 0, fy, y0, 0, 0, 1);

    //set distortion parameters
    float k[5];
    k[0] = -3.4; //k1 //양수쪽으로 갈 수록 볼록해짐 // 음수로 가면 오목해짐
    k[1] = 7; //k2 //음수쪽으로 갈수록 가장자리가 볼록해짐 //양수쪽으로 갈수록 볼록+ 작아짐(쏠림)
    k[2] = 0.; //p1
    k[3] = 0.; //p2
    k[4] = 0; //k3

    //distortion 객체 생성
    Mat distCoeff = (Mat1d(1, 5) << k[0], k[1], k[2], k[3], k[4]);

    // 결과 객체 생성
    Mat result;
    undistort(in, result, cameraMatrix, distCoeff);
    imshow("result/ 21611591 김난희", result);
    waitKey();

    return 0;
}
```

[Visual studio]

```
#include "opencv2/opencv.hpp"//openCV lib
#include "opencv2/imgproc/imgproc.hpp" //image proc lib
using namespace cv;
//21611591 김난희
int main(void)
{
        //load an image
        Mat in = imread("C:₩₩Users₩₩난희₩₩Documents₩₩Visual Studio
2017₩₩Projects₩₩exerciseCV₩₩exerciseCV₩₩input.jpg");
        imshow("input", in);

        //set camera parameters
        float x0 = 606.0/2.0; //Cx 영상 중심의 x좌표
        float y0 = 404.0/2.0; //Cy 영상 중심의 y좌표
        float fx = 1108.0; //[pixel] //카메라 focal length
        float fy = 1108.0; //[pixel] //카메라

        //객체 생성 후 insertion operator 써서 행렬 채워넣음
```

```
        Mat cameraMatrix = (Mat_<float>(3, 3) << fx, 0, x0, 0, fy, y0, 0, 0, 1);

        //set distortion parameters
        float k[5];
        k[0] = -3.4; //k1 //양수쪽으로 갈 수록 볼록해짐 // 음수로 가면 오목해짐
        k[1] = 7; //k2 //음수쪽으로 갈수록 가장자리가 볼록해짐 //양수쪽으로 갈수록 볼록+ 작아짐(쏠림)
        k[2] = 0.; //p1
        k[3] = 0.; //p2
        k[4] = 0; //k3

        //distortion 객체 생성
        Mat distCoeff = (Mat1d(1, 5) << k[0], k[1], k[2], k[3], k[4]);

        // 결과 객체 생성
        Mat result;
        undistort(in, result, cameraMatrix, distCoeff);
        imshow("result/ 21611591 김난희", result);
        waitKey();

        return 0;
}
```

[Visual studio source code]



[Image result]

2. k1, k2, k3, p1, p2 추정 방법

참고 사이트: https://darkpgmr.tistory.com/31

왜곡된 영상은 Id, 보정된 영상을 Iu라고 하면,
우리가 넣은 input은 Id, output은 Iu가 된다.
각 픽셀 값을 해당 픽셀 좌표로 왜곡시켰을 때 대응되는 픽셀 값으로 채우는 것을 기본 아이디어로 한다.

$$\begin{bmatrix} x_{n\_u} \\ y_{n\_u} \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \text{skew\_c}f_x & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} x_{p\_u} \\ y_{p\_u} \\ 1 \end{bmatrix}$$

보정된 영상의 좌표 Xp_u, Yp_u에서 카메라 역파라미터를 통해 nomalize한 좌표 Xn_u, Yn_u를 구한다.

$$x_{n\_u} = (x_{p\_u} - c_x)/f_x - \text{skew\_c}y_{n\_u}$$

$$y_{n\_u} = (y_{p\_u} - c_y)/f_y$$

행렬식을 풀면 위와 같은 식이 된다. skew는 최근 카메라에서는 잘 나타나지 않는 오류이기에 보통 0으로 둔다. Cx와 Cy는 image의 width, height의 1/2를 넣곤 하는데 정확하게는 다르다. 정확한 수를 넣어줄수록 좋다.

왜곡에는 방사 왜곡(radial distortion)과 접선 왜곡(tangential distortion)이 있는데
아래의 식에서 =의 오른쪽 항에서 왼쪽식이 방사왜곡과 관련되었고, 왼쪽식이 접선 왜곡과 관련이 있다.
접선 왜곡은 카메라 제조 과정에서 렌즈와 영상 평면이 완벽히 평행이지 않아 생기는 문제이므로 주로 0으로 두고 풀기도 한다. 즉, p1=0, p2=0이다. 그리고 왜곡이 많이 심하다면 k3도 들어가지만 보통은 k3도 0을 넣는다.

$$\begin{bmatrix} x_{n\_d} \\ y_{n\_d} \end{bmatrix} = (1 + k_1 r_u^2 + k_2 r_u^4 + k_3 r_u^6) \begin{bmatrix} x_{n\_u} \\ y_{n\_u} \end{bmatrix} + \begin{bmatrix} 2p_1 x_{n\_u} y_{n\_u} + p_2 (r_u^2 + 2x_{n\_u}^2) \\ p_1 (r_u^2 + 2y_{n\_u}^2) + 2p_2 x_{n\_u} y_{n\_u} \end{bmatrix}$$

마지막으로 픽셀 좌표계로 변환하여 왜곡된 영상에서의 좌표를 구해주면 된다.

$$\begin{bmatrix} x_{p\_d} \\ y_{p\_d} \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \text{skew\_c}f_x & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{n\_d} \\ y_{n\_d} \\ 1 \end{bmatrix}$$

우리는 어느정도 왜곡이 되어있는지 좌표 이동을 생각할 수 없으므로 어떻게 이러한 식을 가지고 풀 수 있을 지 생각해보았다. 아래 3번과 같은 방법으로 풀게 되었다.

3. 결과: 자신이 사용한 방법 및 느낀점

왜곡된 영상 보정은 카메라 캘리브레이션 과정을 통해 내부 파라미터를 구해야 정확한 결과를 만들어낼 수 있다. 하지만, 주어진 input 사진은 단 1개 이므로, 캘리브레이션 툴을 통해 파라미터를 알아낼 수 없었다.

그리고 위 2번의 식들을 반대로 푸는 과정은 아주 복잡한 7차 방정식을 푸는 과정이 될 것이다. 만약 처음 식에서 보정된 좌표(Xp_u, Yp_u)를 왜곡된 좌표(Xn_d, Yn_d)로 두고 조금씩 수정해보면 풀면 어느정도 값이 나올 것이라 예상한다. 이는 시간이 오래 걸리는 작업이라 생각하여, 결론적으로 임의로 k1, k2, k3, p1, p2를 조절하며 lab2를 풀었다.

우선 p1, p2는 접선 왜곡에 의해 생기는 문제이므로 0으로 두고 풀었으며, k3도 아주 심한 왜곡이 아니라면 보통 사용하지 않으므로 0으로 두었다. k1과 k2를 조절하며 왜곡이 덜 생기는 방향으로 output 사진이 나오게 하였다. k1을 양수 쪽으로 크게 갈수록 볼록해지고, 음수 쪽으로 가면 오목해지는 것과, k2를 음수 쪽으로 갈수록 가장자리가 볼록해지고, 양수 쪽으로 갈수록 볼록 + 작아짐을 실험적으로 알게 되었다. 가장 왜곡이 보정된 형태의 사진을 k1=-3.4, k2=7로 두어 구할 수 있었다.

자세히 보지 않으면 결과 사진은 어느정도 반듯한 건물 형태처럼 보인다. input 사진이 여러 장이라면 카메라 캘리브레이션 툴을 통해 보정을 더 쉽고 정확하게 할 수 있었을 것이다. 왜곡된 영상을 보정하는 일은 쉽지 않은 일인 것 같다.