山东大学＿＿＿计算机科学与技术＿＿＿学院

＿计算机视觉＿课程实验报告

学号：201900161140	姓名： 张文浩	2021.9.24
实验题目：几何变换与变形		
实验过程中遇到和解决的问题：		

实验 2.1：图像变形

根据给定的公式对图像的坐标进行映射，实现图像的变形。

$$[x,y]=f^{-1}([x',y'])=\begin{cases}[x',y'] & \text{if } \ r\geq 1\\ [\cos(\theta)x'-\sin(\theta)y',\sin(\theta)x'+\cos(\theta)y'] & \text{otherwise}\end{cases}$$

其中：$r=\sqrt{x'^2+y'^2}$ $\quad \theta=(1-r)^2$

实现方法：

非插值算法：

1. 读入图像，处理图像

```
Mat imagebig = imread("E:/pix/cUNJkLuG.jpg");
Mat image_before;
//我找的这个图片太大了，先缩小一点
resize(imagebig, image_before, Size(512, 256), (0, 0), (0, 0), INTER_LINEAR);
Mat image_after(image_before.rows, image_before.cols, CV_8UC3);
```

2. 两个 for 循环遍历图片每一个坐标，

先对 x，y 坐标进行中心归一化处理，根据公式

$$x'=\frac{x-0.5W}{0.5W} \qquad y'=\frac{y-0.5H}{0.5H}$$

```
for (int i = 0; i < row; i++)
{
    for (int j = 0; j < col; j++)
    {
        //中心归一化坐标
        double x_normal = i / (0.5 * row) - 1;
        double y_normal = j / (0.5 * col) - 1;
```

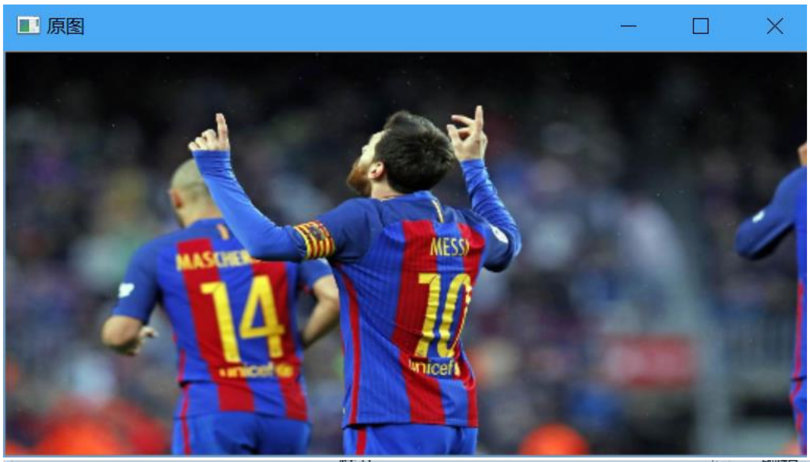
在根据映射函数进行坐标变换

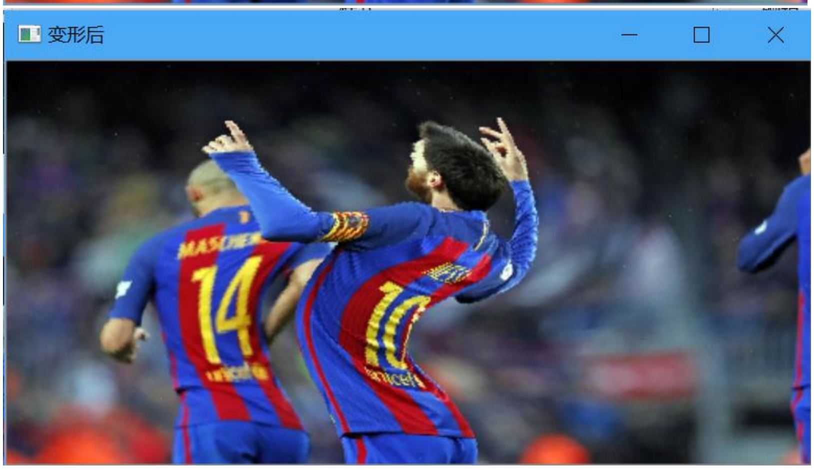
```
double r = sqrt(pow(x_normal, 2.0) + pow(y_normal, 2.0));
double angle = pow(1 - r, 2.0);
double x_normal_after;
double y_normal_after;
if (r >= 1)
{
    x_normal_after = x_normal;
    y_normal_after = y_normal;
}
else {
    x_normal_after = cos(angle) * x_normal - sin(angle) * y_normal;
    y_normal_after = sin(angle) * x_normal + cos(angle) * y_normal;
}
```

因为之前进行了坐标中心归一化处理，所以在坐标变换之后要进行去中心归一化的坐标还原操作。

```
//将中心归一化的坐标还原
int x_after = (x_normal_after + 1) * 0.5 * row;
int y_after = (y_normal_after + 1) * 0.5 * col;
image_after.ptr(i, j)[0] = image_before.ptr(x_after, y_after)[0];
image_after.ptr(i, j)[1] = image_before.ptr(x_after, y_after)[1];
image_after.ptr(i, j)[2] = image_before.ptr(x_after, y_after)[2];
```

最后效果如下：





双线性插值：

```cpp
Mat bilinearpolation(Mat& src) {
    int row = src.rows;
    int col = src.cols;
    Mat dst = Mat::zeros(src.size(), src.type());
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            double x_normal = i / (0.5 * row) - 1;
            double y_normal = j / (0.5 * col) - 1;
            double r = sqrt(pow(x_normal, 2.0) + pow(y_normal, 2.0));
            double angle = pow(1 - r, 2.0);
            double x_normal_before;
            double y_normal_before;
            if (r >= 1)
            {
                x_normal_before = x_normal;
                y_normal_before = y_normal;
            }
            else {
                x_normal_before = cos(angle) * x_normal - sin(angle) * y_normal;
                y_normal_before = sin(angle) * x_normal + cos(angle) * y_normal;
            }
            //将中心归一化的坐标还原
            double x_before = (x_normal_before + 1) * 0.5 * row;
            double y_before = (y_normal_before + 1) * 0.5 * col;
            if (x_before < 0) x_before = 0;
            if (x_before > row - 1) x_before = row - 1;
            if (y_before < 0) y_before = 0;
            if (y_before > col - 1) y_before = col - 1;
            //双线性插值
            int i1 = cvFloor(x_before);
            int i2 = cvCeil(x_before);
            int j1 = cvFloor(y_before);
            int j2 = cvCeil(y_before);
            double v = x_before - i1;
            double u = y_before - j1;
            dst.at<Vec3b>(i, j)[0] = cvFloor((1 - u) * (1 - v) * src.at<Vec3b>(i1, j1)[0] + (1 - u) * v * src.at<Vec3b>(i2, j1)[0] + u * (1 - v) * src.at<Vec3b>(i1, j2)[0] + u * v * src.at<Vec3b>(i2, j2)[0]);
            dst.at<Vec3b>(i, j)[1] = cvFloor((1 - u) * (1 - v) * src.at<Vec3b>(i1, j1)[1] + (1 - u) * v * src.at<Vec3b>(i2, j1)[1] + u * (1 - v) * src.at<Vec3b>(i1, j2)[1] + u * v * src.at<Vec3b>(i2, j2)[1]);
            dst.at<Vec3b>(i, j)[2] = cvFloor((1 - u) * (1 - v) * src.at<Vec3b>(i1, j1)[2] + (1 - u) * v * src.at<Vec3b>(i2, j1)[2] + u * (1 - v) * src.at<Vec3b>(i1, j2)[2] + u * v * src.at<Vec3b>(i2, j2)[2]);
        }
    }
    return dst;
```

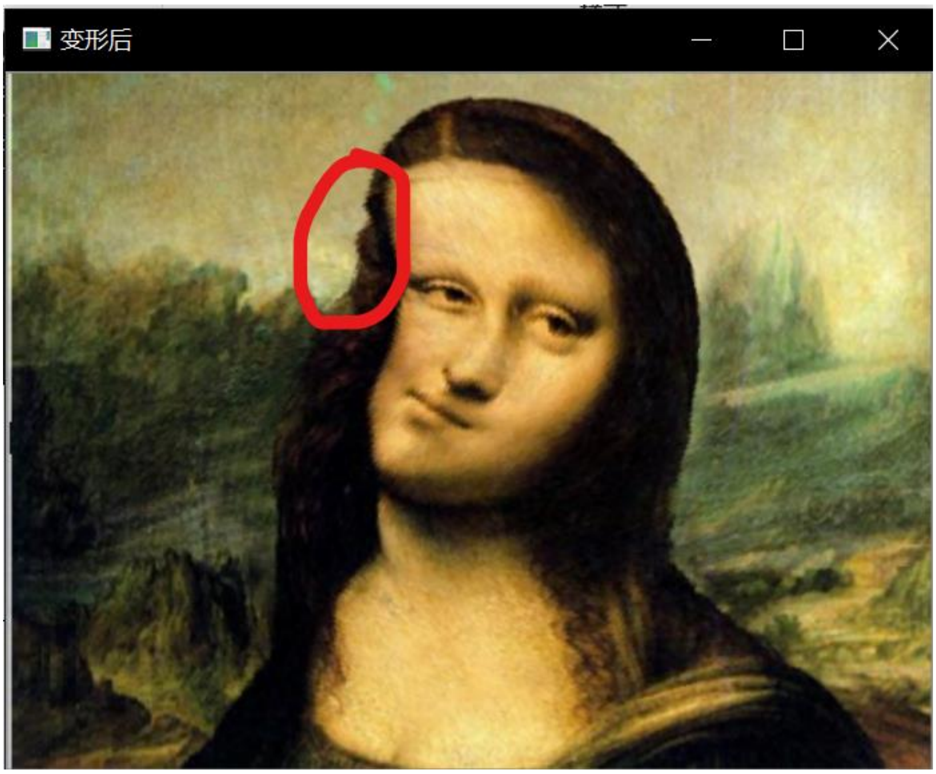
双线性插值的关键步骤：

```cpp
//双线性插值
int i1 = cvFloor(x_before);
int i2 = cvCeil(x_before);
int j1 = cvFloor(y_before);
int j2 = cvCeil(y_before);
double v = x_before - i1;
double u = y_before - j1;
dst.at<Vec3b>(i, j)[0] = cvFloor((1 - u) * (1 - v) * sr
dst.at<Vec3b>(i, j)[1] = cvFloor((1 - u) * (1 - v) * sr
dst.at<Vec3b>(i, j)[2] = cvFloor((1 - u) * (1 - v) * sr
```

对比：

非插值：

通过对比很清晰的发现，没有经过插值处理的图像在蒙娜丽莎的轮廓上有明显的锯齿。而经过双线性插值的蒙娜丽莎在轮廓处有一种模糊马赛克效果，锯齿不明显。



双线性插值：

通过对比很清晰的发现，没有经过插值处理的图像在蒙娜丽莎的轮廓上有明显的锯齿。而经过双线性插值的蒙娜丽莎在轮廓处有一种模糊马赛克效果，锯齿不明显。

实验 2.2 电子哈哈镜

思路就是通过 opencv 提供的 videocapture 从摄像头获取图像,然后一帧一帧 frame 地处理。同时创建 videowriter 变量用于视频的保存。

```
int main()
{
    VideoCapture cap;
    VideoWriter writer;
    int codec = writer.fourcc('M', 'J', 'P', 'G');
    writer = VideoWriter("E:\\计算机视觉\\exp2\\哈哈镜效果.avi", codec, 10, cv::Size(640, 480));
    cap.open(0);
```

```
while (1)
{
    Mat frame;
    cap >> frame;
    if (frame.empty()) {
        cout << "finish" << endl;
        break;
    }
    Mat framet = bilinearpolation(frame);
    writer << framet;
    //imshow("input video", solve1(frame));
    imshow("input video1", framet);
    waitKey(1);
}
writer.release();
cap.release();
```

我先用前面那个小题的图形变换方法试了一下，效果如下

然后自己做了一个放大哈哈镜

非线性插值实现方法:

```
Mat solve1(Mat frame) {
    Mat frame_after(frame.rows, frame.cols, CV_8UC3);
    int row = frame_after.rows;
    int col = frame_after.cols;
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
```

```
            //中心归一化坐标
            double x_normal = i / (0.5 * row) - 1;
            double y_normal = j / (0.5 * col) - 1;
            double r = 0.8;
            double x_normal_after;
            double y_normal_after;
            //变换后的新坐标
            x_normal_after = (x_normal / 2) * (sqrt(x_normal * x_normal + y_normal * y_normal) / r);
            y_normal_after = (y_normal / 2) * (sqrt(y_normal * y_normal + x_normal * x_normal) / r);

            //将中心归一化的坐标还原
            int x_after = (x_normal_after + 1) * 0.5 * row;
            int y_after = (y_normal_after + 1) * 0.5 * col;
            frame_after.ptr(i, j)[0] = frame.ptr(x_after, y_after)[0];
            frame_after.ptr(i, j)[1] = frame.ptr(x_after, y_after)[1];
            frame_after.ptr(i, j)[2] = frame.ptr(x_after, y_after)[2];
        }
    }
    return frame_after;
}
```

线性插值实现方法：

```
Mat bilinearpolation(Mat& src) {
    int row = src.rows;
    int col = src.cols;
    Mat dst(row, col, CV_8UC3);
    for (int i = 0; i < row; i++) {
        double x_normal = i / (0.5 * row) - 1;
        double x_normal_after;
        for (int j = 0; j < col; j++) {
            double y_normal = j / (0.5 * col) - 1;
            double y_normal_after;
            x_normal_after = (x_normal / 2) * (sqrt(x_normal * x_normal + y_normal * y_normal));
            y_normal_after = (y_normal / 2) * (sqrt(y_normal * y_normal + x_normal * x_normal));
            //将中心归一化的坐标还原
            double x_after = (x_normal_after + 1) * 0.5 * row;
            double y_after = (y_normal_after + 1) * 0.5 * col;
            if (x_after < 0) x_after = 0;
            if (x_after > row - 1) x_after = row - 1;
            if (y_after < 0) y_after = 0;
            if (y_after > col - 1) y_after = col - 1;
            //双线性插值
            int i1 = cvFloor(x_after);
            int i2 = cvCeil(x_after);
            int j1 = cvFloor(y_after);
```

```
            int j2 = cvCeil(y_after);
            double v = x_after - i1;
            double u = y_after - j1;
            //dst.at<Vec3b>(i, j)[0] = cvFloor((1 - u) * (1 - v) * src.at<Vec3b>(i1, j1)[0] + (1 - u) * v * src.at<Vec3b>(i2, j1)[0] + u * (1 - v) * src.at<Vec3b>(i1, j2)[0] + u * v * src.at<Vec3b>(i2, j2)[0]);
            //dst.at<Vec3b>(i, j)[1] = cvFloor((1 - u) * (1 - v) * src.at<Vec3b>(i1, j1)[1] + (1 - u) * v * src.at<Vec3b>(i2, j1)[1] + u * (1 - v) * src.at<Vec3b>(i1, j2)[1] + u * v * src.at<Vec3b>(i2, j2)[1]);
            //dst.at<Vec3b>(i, j)[2] = cvFloor((1 - u) * (1 - v) * src.at<Vec3b>(i1, j1)[2] + (1 - u) * v * src.at<Vec3b>(i2, j1)[2] + u * (1 - v) * src.at<Vec3b>(i1, j2)[2] + u * v * src.at<Vec3b>(i2, j2)[2]);

            //优化算法减少乘法次数
            for (int x = 0; x < 3; x++) {
                double h1 = src.at<Vec3b>(i1, j1)[x]*1.0 + 1.0*v * (src.at<Vec3b>(i2, j1)[x]*1.0 - 1.0*src.at<Vec3b>(i1, j1)[x]);
                double h2 = src.at<Vec3b>(i1, j2)[x]*1.0 + 1.0*v * (src.at<Vec3b>(i2, j2)[x]*1.0 - 1.0*src.at<Vec3b>(i2, j1)[x]);
                dst.at<Vec3b>(i, j)[x] = cvFloor(h1 + u * (h2 - h1));
            }
        }
    }
    return dst;
}
```

优化代码执行效率：

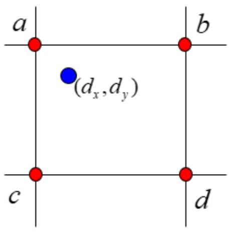
我想到的提高效率的方法是减少乘法次数。

在计算差值的时候本来计算一次有 6 个乘法：

```
dst.at<Vec3b>(i, j)[x] = cvFloor((1 - u) * (1 - v) * src.at<Vec3b>(i1, j1)[x] + (1 - u) * v * src.at<Vec3b>(i2, j1)[x] + u * (1 - v) * src.at<Vec3b>(i1, j2)[x] + u * v * src.at<Vec3b>(i2, j2)[x]);
```

根据老师上课讲的方法可以优化成 3 个乘法

```
float bilinear(float a, float b, float c, float d, float dx, float dy)
{
    float  h1=a+dx*(b-a);        // = (1-dx)*a + dx*b
    float  h2=c+dx*(d-c);
    return h1+dy*(h2-h1);
}
```



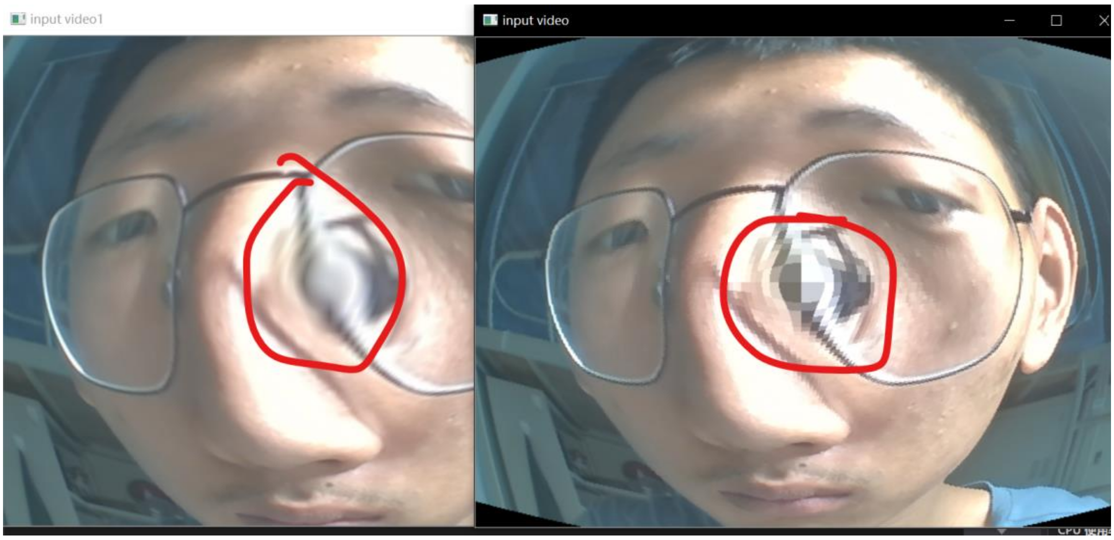
代码实现如下

```
            for (int x = 0; x < 3; x++) {
                double h1 = src.at<Vec3b>(i1, j1)[x]*1.0 + 1.0*v * (src.at<Vec3b>(i2, j1)[x]*1.0 -
```

```
1.0*src.at<Vec3b>(i1, j1)[x]);
                    double h2 = src.at<Vec3b>(i1, j2)[x]*1.0 + 1.0*v * (src.at<Vec3b>(i2, j2)[x]*1.0 - 
1.0*src.at<Vec3b>(i2, j1)[x]);
                    dst.at<Vec3b>(i, j)[x] = cvFloor(h1 + u * (h2 - h1));
```

这样大大减少了乘法次数，提高了计算速度，改进了实时性。

最终效果：左边是进行了双线性插值优化的，右边是没有进行插值优化的，明显右边的有很多小方块（锯齿）。左边就比较模糊，过渡自然。



随实验报告提交了一个几秒钟的视频。

结果分析与体会：

在本次实验中，我学习了利用 opencv 进行对图像的几何变换与变形。在第二个小实验中我将第一个实验中的方法函数进行了封装，实现了对视频进行处理，只做了一个简易的电子哈哈镜。

同时，通过对比非插值方法和双线性插值生成的图像之间的差异，更加深刻地认识到了插值计算的重要性。

在改善实时性方面，我通过减少双线性插值中的乘法次数提高运算速度，进而提高实时性。

总之，在本次试验中收获很多，而且感觉趣味性很高，可以通过自己编写函数实现不同样式的"哈哈镜"。