Lab6 - 全景图拼接 - 3190102721

一、实验要求

本次实验我们实现全景图拼接算法。

二、理论基础

2.1全景图拼接回顾

全景图拼接是利用同一场景的多张图像通过重叠部分寻找匹配关系，从而生成整个场景图像的技术。 全景图的拼接方法有很多，如按场景和运动的种类可以分为单视点全景拼接和多视点全景拼接。

对于平面场景和只通过相机旋转拍摄的场景来说，可以使用求每两幅图像之间的一个Homography变换来映射到一张图像的方法，还可以使用恢复相机的旋转的方式得到最终的全景图。当相机固定只有水平方向旋转时，也可以使用柱面或球面坐标映射的方式求得全景图。

2.2柱面坐标变换

对于每一幅图像来说，我们都可以把它们投影到一个柱面上，得到柱面上的图像 。柱面图像的坐标变换为：

$$x' = r\tan^{-1}(\frac{x}{f})$$
$$y' = \frac{ry}{\sqrt{x^2+f^2}}$$

其中(x', y')为柱面上的坐标，(x, y)为平面图像坐标，其坐标原点都已移至图像中心，r为柱面半径，f为焦距。

然而为了得到柱面投影图像，我们往往需要将柱面图像上的点逆变换到平面图像上的对应像素点，进行插值，得到完整的柱面图像，逆变换的变换公式为：

$$x = f\tan\left(\frac{x'}{r}\right)$$
$$y = \frac{y'}{r}\sqrt{x^2+f^2}$$

2.3特征匹配

对每两幅相邻的柱面图像进行特征提取和匹配（特征可以选用SIFT、ORB等，可以使用OpenCV的函数实现），寻找两幅相邻图像的对应关系。

2.4全景图拼接

使用上一步得到的匹配关系，求出每两幅柱面图像的一个平移变换，利用平移变换将所有图像拼接到一起。得到一幅全景图。

三、运行环境

编译环境：Visual Studio 2017

运行环境：Windows10

四、具体实现

4.1 环境配置

首先安装了`opencv_cntrib`，利用CMake重新编译了`opencv`和`opencv_contrib`，编译了`visual studio`并重新配置了`opencv`环境。（这一步花费了我比较多的精力，虽然之后发现似乎并不需要进行这一步操作。）

4.2 接口设计

本次实验中实现名为`Panorama2721`的类，这个类继承自接口类`CylindricalPanorama`。

```
class CylindricalPanorama {
public:
    virtual bool makePanorama(vector<Mat>& img_vec, Mat& img_out, double f) = 0;
};

class Panorama2721 : public CylindricalPanorama {
    double r = 500;
public:
    vector<Mat> image;
    void readImage(vector<string> name, double f);
    void OperateImage(vector<Mat>& image, double f);
    bool makePanorama(vector<Mat>& img_vec, Mat& img_out, double f);
    Mat CylinderTransform(Mat image, double r, double f);
    Mat ImageStitch(Mat imageA, Mat imageB);
    void ShowImage();
};
```

下面是本次实验中利用的数据集，在此未作展开，利用全局变量读取存储图片。

```
vector<string> data1 = {...};

vector<string> data2 = {...};
```

4.3 函数定义

首先是读入图片的函数，从图片集中读取图片并且将柱面坐标变换后得到的图像存入`image`中。

```
void Panorama2721::readImage(vector<string> name, double f) {
    for (int i = 0; i < name.size(); i++) {
        Mat s = imread(name[i]);
        s = CylinderTransform(s, r, f);
        image.push_back(s);
    }
}
```

也可以在读取所有图片之后再利用下面的函数对所有图片进行柱面坐标变换得到新的图像，不过本实验中利用上面的`readImage`函数一步到位，本函数是在实验调试过程中的产物。

```
void Panorama2721::OperateImage(vector<Mat>& image, double f) {
    int num = image.size();
    for (int i = 0; i < num; i++) {
        image[i] = CylinderTransform(image[i], r, f);
    }
}
```

下面是柱面坐标变换的函数，根据传入的参数对图像做变换，变换公式为$x' = r\tan^{-1}(\frac{x}{f})$和$y' = \frac{ry}{\sqrt{x^2+f^2}}$，逆变换公式为$x = f\tan\left(\frac{x'}{r}\right)$和$y = \frac{y'}{r}\sqrt{x^2+f^2}$，根据两组公式对图象做变换，变换后的图像背景部分由黑色填充。

```
Mat Panorama2721::CylinderTransform(Mat image,double r,double f){
    int rows = image.rows;
    int cols = image.cols;
    int xmax = (cols - 1) / 2;
    int ymax = (rows - 1) / 2;
    int x1 = r * atan(xmax / f);
    int y1 = r * ymax / f;
    int col1 = x1 * 2 + 1;
    int row1 = y1 * 2 + 1;
    Mat M(row1, col1, CV_8UC3, cv::Scalar::all(1));
    for (int i = 0; i < row1; i++) {
        for (int j = 0; j < col1; j++) {
            double x = f * tan((j - x1) / r);
            double y = (i - y1) / r * sqrt(x*x + f * f);
            x = x + xmax;
            y = y + ymax;
            if (x < 0 || y < 0 || x >= cols || y >= rows) {
                M.at<cv::Vec3b>(i, j) = cv::Vec3b(0, 0, 0);
            }
            else {
                M.at<cv::Vec3b>(i, j) = image.at<cv::Vec3b>(y, x);
            }
        }
    }
    return M;
}
```

下面是本实验中最主要的部分，将两幅图拼接在一起，并且返回拼接后得到的图像。

首先我利用`opencv`库中的`SIFT`算法提取特征点，并存储得到的特征点，之后我对得到的特征点做近似度匹配。在实验的过程中发现，在后面的部分我并不需要这么多点，因此我选取最佳匹配的一部分特征点。在完成匹配部分之后，我利用`findHomography()`函数计算得到变换矩阵。然后我利用`warpPerspective()`函数进行操作，对两张图进行合并，得到拼接后的图像。我发现在完成图像的拼接后，最终得到的图片效果中会有比较明显的黑色边缘轮廓线，因此我利用像素替换的方式消除轮廓线，通过比较RGB范数的形式来判断是否替换像素，但是这也带来了一些问题，后面会进行分析。

```
Mat Panorama2721::ImageStitch(Mat imageA, Mat imageB){
    //imshow("imageA", imageA);
    //imshow("imageB", imageB);
    copyMakeBorder(imageA, imageA, 10, 10, 50, 50, BORDER_CONSTANT, Scalar(0, 0,
0));
    copyMakeBorder(imageB, imageB, 10, 10, 50, 50, BORDER_CONSTANT, Scalar(0, 0,
0));
```

```
    //imshow("imageA_1", imageA);
    //imshow("imageB_1", imageB);

    int Hessian = 1000;
    Ptr<SIFT>detector = SIFT::create(Hessian);

    vector<KeyPoint>keypointA, keypointB;
    Mat descriptorA, descriptorB;
    detector->detectAndCompute(imageA, Mat(), keypointA, descriptorA);
    detector->detectAndCompute(imageB, Mat(), keypointB, descriptorB);

    //Mat drawsrc, drawsrc2;
    //drawKeypoints(imageA, keypointA, drawsrc);
    //imshow("drawsrc", drawsrc);
    //drawKeypoints(imageB, keypointB, drawsrc2);
    //imshow("drawsrc2", drawsrc2);

    //FlannBasedMatcher matcher;
    BFMatcher matcher(NORM_L2);
    vector<DMatch>matches;
    matcher.match(descriptorA, descriptorB, matches);

    vector<DMatch>goodmatches;
    vector<Point2f>goodkeypointA, goodkeypointB;
    sort(matches.begin(), matches.end());

    for (int i = 0; i < 300; i++){
        goodkeypointA.push_back(keypointA[matches[i].queryIdx].pt);
        goodkeypointB.push_back(keypointB[matches[i].trainIdx].pt);
        goodmatches.push_back(matches[i]);
    }

    //Mat temp_A = imageA.clone();
    //for (int i = 0; i < goodkeypointA.size(); i++){
        //circle(temp_A, goodkeypointA[i], 3, Scalar(0, 255, 0), -1);
    //}
    //imshow("goodkeypoints1", temp_A);

    //Mat temp_B = imageB.clone();
    //for (int i = 0; i < goodkeypointB.size(); i++){
        //circle(temp_B, goodkeypointB[i], 3, Scalar(0, 255, 0), -1);
    //}
    //imshow("goodkeypoints2", temp_B);

    Mat img_matches;
    drawMatches(imageA, keypointA, imageB, keypointB, goodmatches, img_matches);
    //imshow("Match", img_matches);

    Mat H = findHomography(goodkeypointB, goodkeypointA, RANSAC);

    Mat DstImg;
    warpPerspective(imageB, DstImg, H, Size(imageA.cols + imageB.cols,
imageA.rows));
    //copyMakeBorder(imageB, imageB, 0, 0, 0, 200, BORDER_CONSTANT, Scalar(0, 0,
0));
    //imshow("DstImg", DstImg);
```

```
    //copyMakeBorder(imageA, imageA, 0, 0, 0, 100, BORDER_CONSTANT, Scalar(0, 0,
0));
    for (int i = 0; i < imageA.rows; i++) {
        for (int j = 0; j < imageA.cols; j++) {
            if (norm(imageA.at<Vec3b>(i, j)) < norm(DstImg.at<Vec3b>(i, j)))
imageA.at<Vec3b>(i, j) = DstImg.at<Vec3b>(i, j);
        }
    }

    //imshow("imgA", imageA);
    //imwrite("imgA.jpg", imageA);
    return imageA;
}
```

下面的函数实现全景图的拼接功能，将数据集中的所有图像拼接成一个完整的全景图，本次实验的数据集其相机固定只有水平方向的旋转，因此最后我们会得到一张长宽比例较大的全景图。

在全景图的拼接过程中，我们采取的是从中间开始拼的策略，选取中间的图片并在每次操作时左右各拼接一张，最后拼完所有图像得到全景图。

```
bool Panorama2721::makePanorama(vector<Mat>& image, Mat& img_out, double f) {
    int num = image.size();
    //cout << num << endl;
    if (num <= 0) {
        return false;
    }
    int mid = num / 2;
    Mat img = image[mid];
    if (num % 2 == 1) {
        for (int i = 1; i <= mid; i++) {
            img = ImageStitch(img, image[mid - i]);
            img = ImageStitch(img, image[mid + i]);
            imshow("img_now", img);
            waitKey(0);
        }
    }
    else {
        for (int i = 1; i < mid; i++) {
            img = ImageStitch(img, image[mid - i]);
            img = ImageStitch(img, image[mid + i]);
            imshow("img_now", img);
            waitKey(0);
        }
        img = ImageStitch(img, image[0]);
        imshow("img_now", img);

    }
    imwrite("img_result.jpg", img);
    img.copyTo(img_out);
    return true;
}
```

4.4 主程序

本次实验中进行柱面坐标变换时参考了一些建议，最后用到的参数选取为$f = 512.89$，$r = 500$。

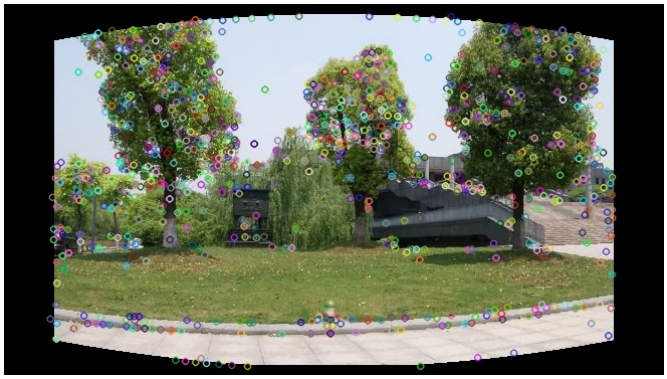
```
int main() {
    double f = 512.89;
    Panorama2721 myparanoma;
    myparanoma.readImage(data1, f);
    //myparanoma.readImage(data2, f);
    //myparanoma.ShowImage();

    Mat img;
    myparanoma.makePanorama(myparanoma.image, img, f);
    //imwrite("img.jpg", img);
    //imshow("img", img);
    waitKey(0);
}
```

五、实验结果及分析

5.1 实验过程效果图

首先我们看柱面坐标变换得到的效果图，可以看到效果图符合预期。

现在我们选取两张图做特征点的提取和匹配，下面是效果图，可以发现效果还是比较好的。

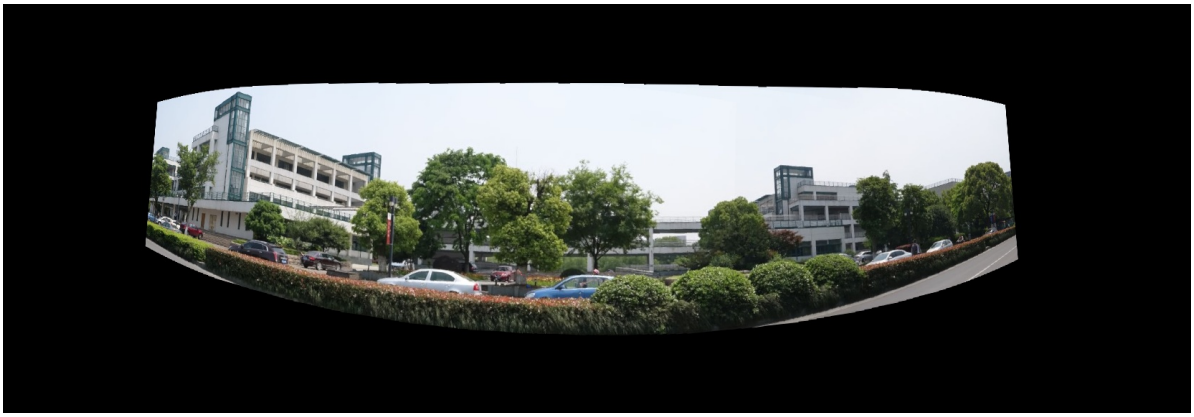


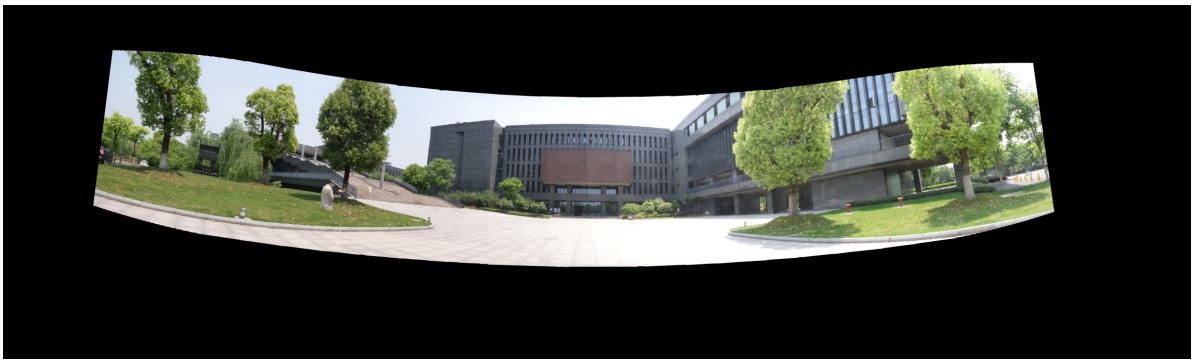
下面的图是特征点匹配连线的图，只显示了最佳匹配部分特征点的匹配效果。

下面是两幅图拼接后得到的结果，可以发现效果还是比较好的，没有明显的拼接痕迹。

5.2 实验结果

下面是两个数据集最终的拼接结果，可以发现两张全景图乍一看效果还是非常好的，较好地实现了多图的全景拼接，除了个别处的小瑕疵，其余大部分地方均达到了实验预期。





5.3 问题分析与改进

本次实验中遇到了很多问题。在计算变换矩阵时我遇到了图像缺失的问题，最终通过在变换前对图像进行扩充的方法解决了。在图像拼接的时候也出现了很多离奇的效果图，例如图像在拼接的过程中发生了非常诡异的形变，也有过大规模区域是类似色彩的问题，其中有很多原因诸如特征点选取数量不合适等，最终也在多次尝试后选取了合适的参数和其他的一些操作成功解决。

本次实验最后也有很多可以改进的地方，例如第二张全景拼接的成果图中，可以发现两个人像变得模糊不清了，在分析原因后，我认为可能是由于原数据集含人像的图片较少导致特征点匹配时效果不好，但更重要的一点应该是我在拼接函数最后一部分消除拼接图的轮廓线的时候，采取了利用较亮处像素代替较暗处像素的方法，因此导致人像区域的像素点最终被周围较亮的像素点代替了，这也解释了我最终的效果图看起来亮度较高的原因，可以看到地面处有类似过曝的现象，应该也是这部分的操作导致的。除了这一点之外，还有很多改进之处，例如成果图的显示对于黑色边界可以进行适当的去除，图像的拼接处还是有一定的痕迹可以采取适当操作使之变得更加平滑。

六、心得体会

这次实验是我花费精力和时间最多的一次实验。首先在环境配置的探究上就花费了半天的时间，之后在实验具体的过程中，一方面因为实验指导比较简略，同时又由于我本人对本次实验中很多全新的操作不熟悉，导致实验进度缓慢，不过最终效果还是比较令我满意的，自己内心也是比较有成就感的。

实验中我因为对 `opencv` 的不熟悉，浪费了很多不必要的时间在一些无伤大雅的问题上，例如在拼接全景图的过程中我发现无论我如何扩充边界总是无法完整地显示图像导致我认为我在拼接的过程中失去了一部分图像，但我最后无意中发现保存下来的图像效果是非常好的，我在发现这个的时候是非常崩溃的因为我花费了几乎半天的时间在调整参数但最终发现这个问题根本没有太大必要去解决，虽然这个结果令我非常哭笑不得但我认为这个过程也是有益的，让我更加熟悉一些操作同时也对我之后的学习过程到了一定的提醒作用。

这次实验我觉得我的代码和结果还有较大的进步空间，对于实验的理解也有待加深，实验中的很多数学原理和操作我知其然但不知其所以然。总体而言，这次实验收获很大，在实验的过程中巩固了课堂学到的新鲜知识，也自学了很多有用的知识和操作。

七、参考文献

[1] https://blog.csdn.net/weijifen000/article/details/93377143

[2] https://www.jb51.net/article/233661.htm#_label3

[3] https://blog.csdn.net/weixin_61023120/article/details/122592033

[4] https://blog.csdn.net/Zero___Chen/article/details/122274445

[5] https://www.microsoft.com/en-us/research/publication/image-alignment-and-stitching-a-tutorial/?from=http%3A%2F%2Fresearch.microsoft.com%2Fpubs%2F75695%2Fszeliski-fnt06.pdf