柱面全景拼接 Panorama 实现
计算摄影学

杨晗
3150104251

June 11, 2018

1 背景

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 实验目标

实现一个 Panorama 类，实现给定一组序列图片和焦距，输出拼接的全景图像的功能。

3 实验环境

Windows 10, i5-4210H

OpenCV 3.3

3.1 补充

OpenCV 3 与 OpenCV 2 相比，提取特征的接口发生了较大变化，SIFT 等特征提取的代码被移到 contrib 的 xfeatures2d 中。此为编译运行需要注意的点。

4 算法原理

4.1 柱面投影

4.1.1 目标

把平面图像投影到柱面上。

4.1.2 原理

$$x^{'} = f * atan\left(\frac{x - 0.5 * width}{f}\right) + f * atan\left(\frac{0.5 * width}{f}\right) \tag{1}$$

$$y^{'} = \frac{f * (y - 0.5 * height)}{\sqrt{(x - 0.5 * width)^2 + f^2}} + 0.5 * height \tag{2}$$

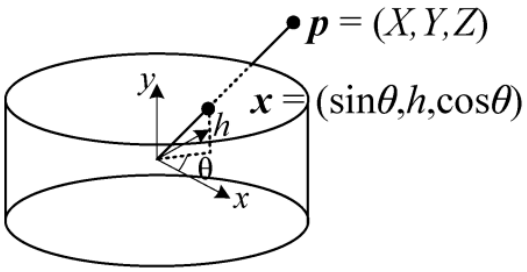

Figure 1: 柱面投影变换

4.2 特征抽取与匹配

4.2.1 目标

对每两幅相邻的柱面图像进行特征提取和匹配，寻找两幅相邻图像的对应关系。

4.2.2 原理

SIFT 特征是基于物体上的一些局部外观的兴趣点而与影像的大小和旋转无关。对于光线、噪声、些微视角改变的容忍度也相当高。
通过 SIFT 特征的提取, 然后用 BruteForceMatch 或者 KnnMatch 可以对 SIFT 计算出匹配。用匹配的特征点可以训练出 homography。

4.3 计算变换，进行拼接

4.3.1 目标

使用得到的匹配关系，求出每两幅柱面图像的平移变换，利用平移变换将所有图像拼接到一起。得到全景图。

4.3.2 原理

通过 RANSAC 之后的匹配特征点，可以从中计算得出 homography。利用这个 homography，可以算出图片的变换，利用此变换可以将两幅图像拼接在一起。

5 代码实现

5.1 接口

核心代码是实现如下的接口：

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

我实现的类如下：

```
class Panorama4251 : public CylindricalPanorama {
    public:
        Panorama4251();
        ~Panorama4251();
        bool makePanorama(vector<Mat>& img_vec, Mat& img_out, double f);
    private:
        Mat cylinder(Mat& img, double f);
        Mat stitch(Mat& img, Mat& img1);
}
```

5.2 代码流程

我实现的流程如下：

a. 对列表中所有图片进行柱面投影，并存下来

b. 对于上一次的拼接结果和下一张图片求 SIFT 特征点

c. 匹配 SIFT 特征点

d. 计算 homography

e. 利用 homography 进行变换，拼接

f. 重复上述步骤直到用完所有图片，完成全景拼接

这里有一个细节是如果从左往右拼接的话，最好是把左边的图片变换到右边图片的坐标系中，这样可以方便之后的特征点匹配和 homography 的计算。

5.3 柱面投影

这里通过最近邻插值算法来求柱面图上的点到原图的对应位置，并用此位置的像素值作为此点的像素值。

```
Mat cylinder(Mat& img, double f) {
    Mat output;
    int cols = (int)2 * f * atan(0.5*img.cols / f);
    int rows = (int)img.rows;
    output.create(rows, cols, CV_8UC3);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            int x = (int)(f * tan((float)(j - cols * 0.5) / f) +
                img.cols*0.5);
            int y = (int)((i - 0.5*rows)*sqrt(pow(x - img.cols*0.5, 2) +
                f*f) / f + 0.5*img.rows);
            if (0 <= x && x < img.cols && 0 <= y && y < img.rows) {
                output.at<Vec3b>(i, j) = img.at<Vec3b>(y, x);
            }
            else {
                output.at<Vec3b>(i, j) = Vec3b(0, 0, 0);
            }
        }
    }
    return output;
}
```

5.3.1 特征点提取

这里的 SIFT 特征点是用 OpenCV 3 的写法。

```
    Ptr<Feature2D> f2d = xfeatures2d::SIFT::create();
    vector<KeyPoint> kps_0, kps_1;
    f2d->detect(img_1, kps_0);
    f2d->detect(img_2, kps_1);

    Mat descriptors_0, descriptors_1;
    f2d->compute(img_1, kps_0, descriptors_0);
    f2d->compute(img_2, kps_1, descriptors_1);
```

5.3.2 特征点的匹配和筛选

其中对于 distance 过大的点进行了筛选处理，保留比较好的点。

```
FlannBasedMatcher matcher;
//BFMatcher matcher;
vector<DMatch> matches;
matcher.match(descriptors_0, descriptors_1, matches);
sort(matches.begin(), matches.end());
float min_v = numeric_limits<float>::max();
float max_v = 0;
for (int i = 0; i < matches.size(); ++i) {
    min_v = min(min_v, matches[i].distance);
    max_v = max(max_v, matches[i].distance);
}
vector<Point2f> ps_0, ps_1;
//assert(matches.size() > 500);
cout << "min_v " << min_v << endl;
cout << "max_v " << max_v << endl;
for (int i = 0; i<matches.size(); ++i) {
    DMatch m = matches[i];
    if (m.distance > max_v / 2 )continue;
    ps_0.push_back(kps_0[m.queryIdx].pt);
    ps_1.push_back(kps_1[m.trainIdx].pt);
}
```

5.3.3 计算 homography 并计算图像扩大行列

利用匹配点来计算出 Homography。
并且利用边界点计算出拼接后的图像的大小。

```
Mat rev_H = findHomography(ps_1, ps_0, RANSAC);
Mat H = findHomography(ps_0, ps_1, RANSAC);

cout << "begin stitcher.... " << i << endl;
vector<Point2f> corners_1(4);
vector<Point2f> corners_2(4);
corners_1[0] = Point2f(0, 0);
corners_1[1] = Point2f((float)img_1.cols, 0);
corners_1[2] = Point2f((float)img_1.cols, (float)img_1.rows);
corners_1[3] = Point2f(0, (float)img_1.rows);

perspectiveTransform(corners_1, corners_2, H);
int down_rows = (int)min(corners_2[0].y, corners_2[1].y);
down_rows = min(0, down_rows) * -1;
int right_cols = (int)min(corners_2[0].x, corners_2[3].x);
right_cols = min(0, right_cols) * -1;
```

5.3.4 计算变换后的坐标并进行变换，拼接

这里计算了每个点在新坐标系下的位置，并进行了插值。

```
Mat stitch_img = Mat::zeros(img_2.rows+down_rows,
    img_2.cols+right_cols, CV_8UC3);
img_2.copyTo(Mat(stitch_img, Rect(right_cols, down_rows, img_2.cols,
    img_2.rows)));
for (int i = 0; i < stitch_img.rows; ++i) {
   for (int j = 0; j < stitch_img.cols; ++j) {
      if (stitch_img.at<Vec3b>(i, j) != Vec3b(0, 0, 0)) {
         continue;
      }
      int x0 = j - right_cols;
      int y0 = i - down_rows;
      vector<Point2f> pix, dst;
      pix.emplace_back(x0, y0);
      perspectiveTransform(pix, dst, rev_H);
      Point2f pos = dst[0];
      //cout << pos << endl;
      int x = (int)floor(pos.x);
      int y = (int)floor(pos.y);
      if (0 < y && y < img_1.rows && 0 < x && x < img_1.cols &&
          img_1.at<Vec3b>(y,x) != Vec3b(0,0,0) ) {
         Vec3b c = img_1.at<Vec3b>(y, x);
         //if (stitch_img.at<Vec3b>(i,j) != Vec3b(0, 0, 0)) { c +=
             (stitch_img.at<Vec3b>(i,j)-c)/2; }
         stitch_img.at<Vec3b>(i, j) = c;
      }
   }
}
last_result = stitch_img;
```

6 实验结果

对于两组图像，拼接得到的结果如下所示

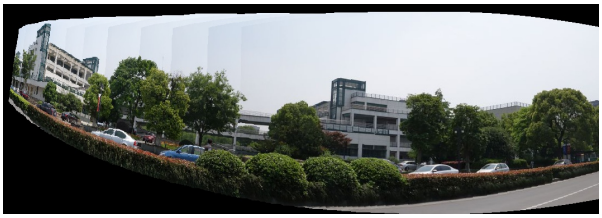

Figure 2: Panorama 1

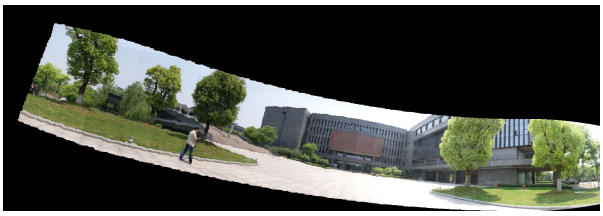

Figure 3: Panorama 2

7 实验心得

本次实验实现了柱面全景拼接。整个流程中熟悉了 OpenCV 对于图像的处理，SIFT 特征点的提取和匹配，基于匹配点的变换的计算。