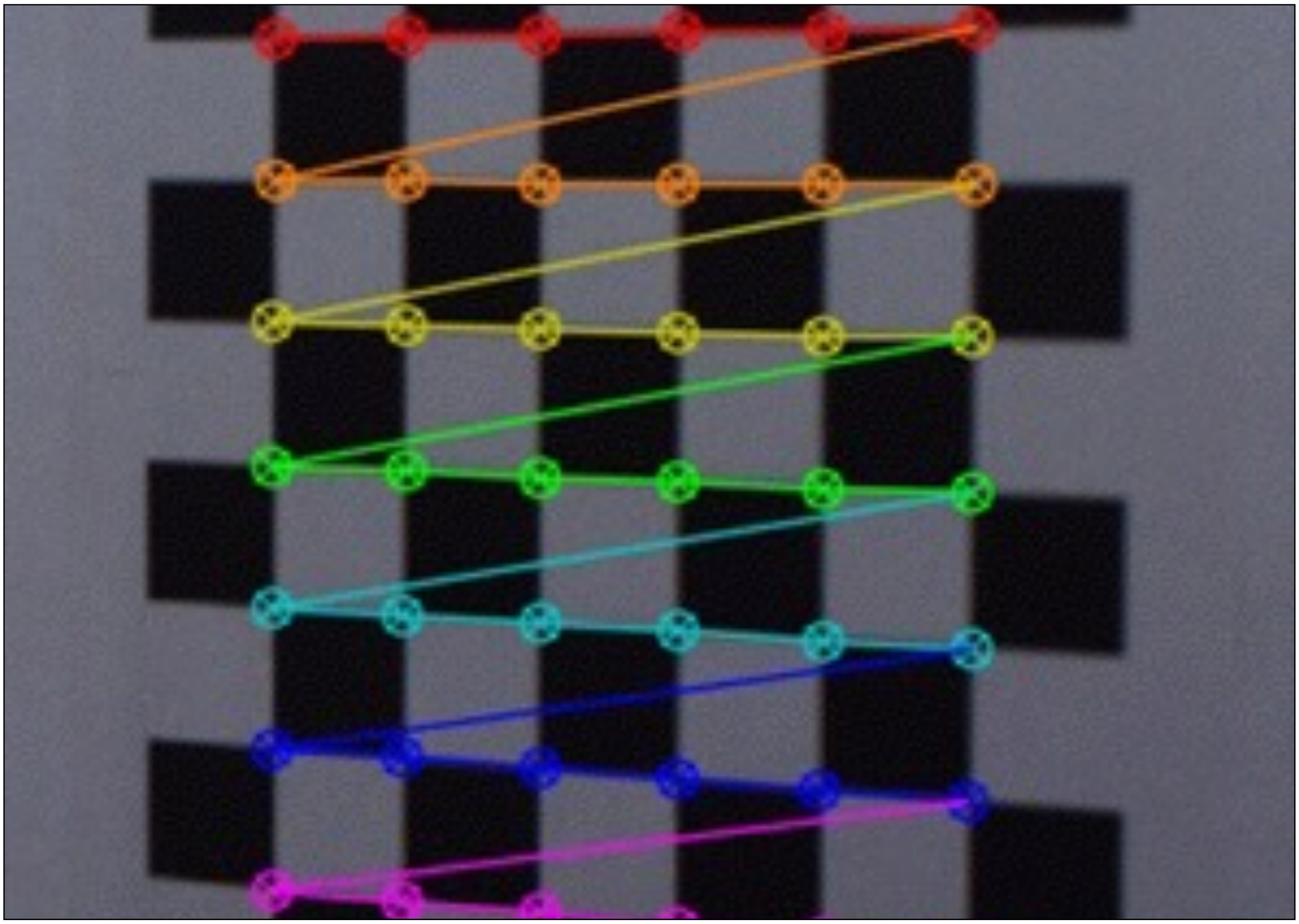
计算机视觉

实验四-摄像机标定及鸟瞰视图

3130000696 黄昭阳 - January 4, 2016



个人信息

3130000696
黄昭阳

开发软件说明

提供Mac下的可执行文件，或者使用make生成可执行文件。可执行文件格式如下：

```
./Lab4 棋盘宽度 棋盘长度 扫描棋盘个数 鸟瞰图转变图片
```

eg.

```
./Lab4 7 7 10 sample.jpg
```

解释：软件将使用摄像头扫描得到10张7*7棋盘的图像用来做摄像机标定，用户需要将摄像头对一张格点不少于7*7的棋盘进行多次拍摄用来对摄像机镜头的畸变进行矫正，并且恢复出摄像机内部参数，然后软件会根据获取的数据对图片sample.jpg进行鸟瞰图的转换。

算法具体步骤与实现要点

一、算法具体步骤：

```
    board_w = atoi(argv[1]);
    board_h = atoi(argv[2]);
    n_boards = atoi(argv[3]);
//根据命令行输入得到棋盘的长宽以及个数
    int board_n = board_w*board_h;
    CvSize board_sz = cvSize(board_w,board_h);
    CvCapture* capture = cvCreateCameraCapture(0);
    assert(capture);
//接通摄像头，使用摄像头获取数据
    cvNamedWindow("Calibration");
    CvMat* image_points = cvCreateMat(n_boards*board_n, 2, CV_32FC1);
    CvMat* object_points = cvCreateMat(n_boards*board_n, 3, CV_32FC1);
    CvMat* point_counts = cvCreateMat(n_boards, 1, CV_32SC1);
//以上矩阵分别用来存储：图片上棋盘格点坐标、空间三维坐标、点数目
    CvMat* intrinsic_matrix = cvCreateMat(3, 3, CV_32FC1);
    CvMat* distortion_coeffs = cvCreateMat(5, 1, CV_32FC1);
//以上矩阵分别用来是摄像头的内参矩阵、摄像头畸变数据
   CvPoint2D32f* corners = new CvPoint2D32f[board_n];
//存储棋盘格点
    int corner_count;
//统计当前图像帧棋盘的格点数目
    int successes = 0;
//用来记录当前已经成功获取多少个有效地棋盘图像
    int step,frame = 0;

    IplImage *image = cvQueryFrame(capture);
    IplImage *gray_image = cvCreateImage(cvGetSize(image), 8, 1);
//从摄像头获取一帧图像
    while(successes < n_boards) //当获取的有效棋盘图像未达到输入要求则继续
    {
        if (frame++ % board_dt == 0) {
            int found = cvFindChessboardCorners(image, board_sz,
corners,&corner_count,CV_CALIB_CB_ADAPTIVE_THRESH |
CV_CALIB_CB_FILTER_QUADS);
//调用OpenCV函数寻找传入图片中的棋盘格点，豁得出初步地址
            cvCvtColor(image, gray_image, CV_BGR2GRAY);
//将原图像转为灰度图
            cvFindCornerSubPix(gray_image, corners, corner_count,
cvSize(11, 11), cvSize(-1, -1), cvTermCriteria(CV_TERMCRIT_EPS
+CV_TERMCRIT_ITER,30,0.1));
//使用该函数精确查找角点位置，提高去畸变的精确度
             cvDrawChessboardCorners(image, board_sz, corners,
corner_count, found);  //将找到的棋盘角点全部画出来
            cvShowImage("Calibration", image);   //将获取到的图片显示到屏幕
            if( corner_count == board_n )
  //如果查找到的角点数目与预计的相同，说明没有找错
            {
                step = successes*board_n;
                for(int i=step,j=0;j<board_n;i++,j++)
                {
                    CV_MAT_ELEM(*image_points, float, i, 0) =
corners[j].x;
                    CV_MAT_ELEM(*image_points, float, i, 1) =
corners[j].y;
//将找到的棋盘格点的坐标放入image_points中用来作为之后标定摄像机的数据
                    CV_MAT_ELEM(*object_points, float, i, 0) = j/board_w;
                    CV_MAT_ELEM(*object_points, float, i, 1) = j%board_w;
                    CV_MAT_ELEM(*object_points, float, i, 2) = 0.0f;
//以棋盘(0,0)位置为原点建立世界坐标系
                }
                CV_MAT_ELEM(*point_counts, int, successes,0) = board_n;
//记录或得到的角点数量
                successes++;
            }
        }
        int c = cvWaitKey(15);
        if( c== 'p' )
        {
            c = 0;
            while( c!='p' && c!=27 ) c = cvWaitKey(250);
        }
       if( c == 27 ) return 0;
//用来实现按下p键暂停收录,按下esc键退出的功能
        image = cvQueryFrame(capture);
//获取下一帧图片
    }

    CvMat* object_points2 = cvCreateMat(successes*board_n, 3, CV_32FC1);
    CvMat* image_points2 = cvCreateMat(successes*board_n, 2, CV_32FC1);
    CvMat* point_counts2 = cvCreateMat(successes,1, CV_32SC1);
    for(int  i=0;i<successes*board_n;i++)
    {
        CV_MAT_ELEM(*image_points2, float, i, 0) =
CV_MAT_ELEM(*image_points, float, i, 0);
        CV_MAT_ELEM(*image_points2, float, i, 1) =
CV_MAT_ELEM(*image_points, float, i, 1);
        CV_MAT_ELEM(*object_points2, float, i, 0) =
CV_MAT_ELEM(*object_points, float, i, 0);
        CV_MAT_ELEM(*object_points2, float, i, 1) =
CV_MAT_ELEM(*object_points, float, i, 1);
        CV_MAT_ELEM(*object_points2, float, i, 2) =
CV_MAT_ELEM(*object_points, float, i, 2);
    }
//将所有获得的数据拷贝一份放到对应的矩阵里面
```

```
    for(int i=0;i<successes;i++)
        CV_MAT_ELEM(*point_counts2, int, i, 0) = 
CV_MAT_ELEM(*point_counts, int, i, 0);
    cvReleaseMat(&object_points);
    cvReleaseMat(&image_points);
    cvReleaseMat(&point_counts);
//释放原来的矩阵
    CV_MAT_ELEM(*intrinsic_matrix, float, 0, 0) = 1.0f;
    CV_MAT_ELEM(*intrinsic_matrix, float, 1, 1) = 1.0f;
//内参矩阵的fx和fy都置为1
    cvCalibrateCamera2(object_points2, image_points2, point_counts2, 
cvGetSize(image), intrinsic_matrix, distortion_coeffs);
//调用OpenCV接口进行摄像机标定，获得两个矩阵
    cvSave("Intrinsics.xml", intrinsic_matrix);
    cvSave("Distortion.xml", distortion_coeffs);
//将两个矩阵的参数以XML的格式存储下来
    CvMat *intrinsic = (CvMat*)cvLoad("Intrinsics.xml");
    CvMat *distortion = (CvMat*)cvLoad("Distortion.xml");
//导入存储的矩阵参数数据
    IplImage *mapx = cvCreateImage(cvGetSize(image), IPL_DEPTH_32F, 1);
    IplImage *mapy = cvCreateImage(cvGetSize(image), IPL_DEPTH_32F, 1);

    CvPoint2D32f objPoints[4],imgPoints[4];
//homography矩阵共8个未知数，因此需要四个对应点进行恢复
    cvInitUndistortMap(intrinsic, distortion, mapx, mapy);
//设定去畸变的参数
    CvMat* H;
    float Z = 25;
    cvNamedWindow(“birdview”);

    IplImage* img = cvLoadImage(argv[4]);
//载入传入的需要转化的图片
    cvResize(img, image);
    IplImage* t = cvCloneImage(image);
    cvRemap(t, image, mapx,mapy);
//之前已经使用cvinitundistort设置了变换方法，去掉畸变
    int found = cvFindChessboardCorners(image, board_sz, 
corners,&corner_count,CV_CALIB_CB_ADAPTIVE_THRESH | 
CV_CALIB_CB_FILTER_QUADS);
//找到传入的图片中的棋盘格点
    if(!found)
    {
        cout << "Can't find!" << endl;
        return 0;
    }
    cvCvtColor(image, gray_image, CV_BGR2GRAY);
    cvFindCornerSubPix(gray_image, corners, corner_count, cvSize(11, 11),
cvSize(-1, -1), cvTermCriteria(CV_TERMCRIT_EPS+CV_TERMCRIT_ITER,30,0.1));
//与上方一样，转灰度图后精确查找角点所在位置

    objPoints[0].x = 0; objPoints[0].y = 0;
    objPoints[1].x = board_w-1; objPoints[1].y = 0;
    objPoints[2].x = 0; objPoints[2].y = board_h-1;
    objPoints[3].x = board_w-1; objPoints[3].y = board_h-1;

    imgPoints[0] = corners[0];
    imgPoints[1] = corners[board_w-1];
    imgPoints[2] = corners[(board_h-1)*board_w];
    imgPoints[3] = corners[board_h*board_w-1];
//设置4对点对来求解homograph矩阵

    cvCircle(image, cvPointFrom32f(imgPoints[0]), 9, CV_RGB(255, 0, 0),
3);
    cvCircle(image, cvPointFrom32f(imgPoints[1]), 9, CV_RGB(0, 255, 0),
3);
    cvCircle(image, cvPointFrom32f(imgPoints[2]), 9, CV_RGB(0, 0, 255),
3);
    cvCircle(image, cvPointFrom32f(imgPoints[3]), 9, CV_RGB(255, 255, 0),
3);
//使用不同的颜色将选定的四个点在原图上圈出来
    H = cvCreateMat(3, 3, CV_32F);
   cvGetPerspectiveTransform(objPoints, imgPoints, H);
//调用OpenCV的函数使用获取得到的点对生成透视变换矩阵
    int key = 0;
    IplImage *bird_image = cvCloneImage(image);

    while(key!=27)
    {
        CV_MAT_ELEM(*H, float, 2, 2) = Z;
//H矩阵的第三行第三列的参数是起到齐次坐标的作用，即用来控制镜头与目标的距离
        //  cvShowImage("Calibration", image);
        cvWarpPerspective(image, bird_image, H,CV_INTER_LINEAR |
CV_WARP_INVERSE_MAP | CV_WARP_FILL_OUTLIERS);
//使用H矩阵进行homography矩阵变成鸟瞰图
        cvShowImage("birdview", bird_image);
//显示图片
        key= cvWaitKey();
        if( key == 'u' ) Z+=0.5;
        if( key == 'd' ) Z-=0.5;
//获取键盘数据来调整摄像头的位置
        if( key == 27 ) break;
//当按下ESC键退出软件
    }
```

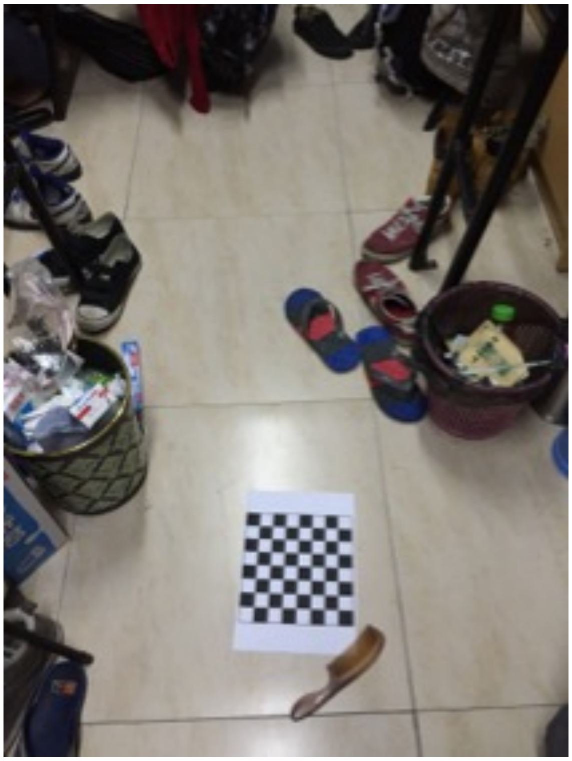
二、实现要点：

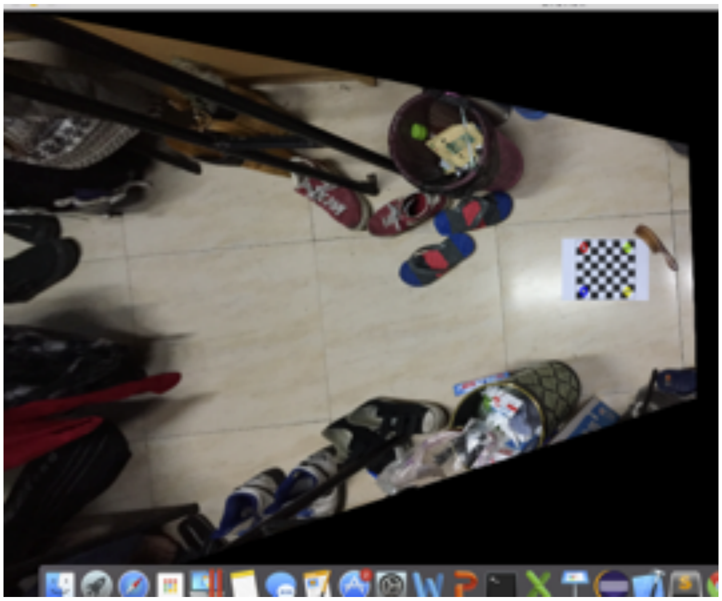
1. 摄像机标定与鸟瞰视图的转换这两段代码在《Learning OpenCV》上都有样例，这里将两段代码整合在了一起，详细步骤已在上方说明。

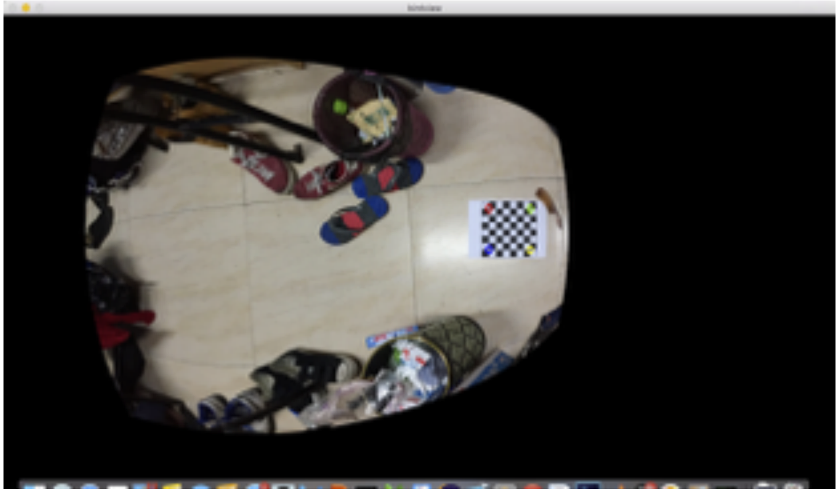
实验结果及分析

使用命令：./Lab4 7 7 10 sample.JPG。

输入图片：　　　　　　　　　　　　　　鸟瞰结果图片：







结果分析：

左边那个变圆了的图应该是我摄像机定标时棋盘的纸有些翘起来引起的，之前还出现过更加奇怪的形状，不过忘记截图了，后来将棋盘用力碾平了，得到的结果就比较正常了，如正上方的图片所示，可以使用u,p进行上下的调整。

编程体会

1. 了解到了使用棋盘进行摄像机定标的方法并实现了出来，并且学会了使用透视投影矩阵进行homograph的变换。

2. 深刻了解了摄像机单目视觉的原理。