相机标定(一)——内参标定与程序实现

white_Learner

分类专栏：机器视觉
发布时间 2020.05.29　　阅读数 4500　　评论数 0

相机标定(一)——内参标定与程序实现

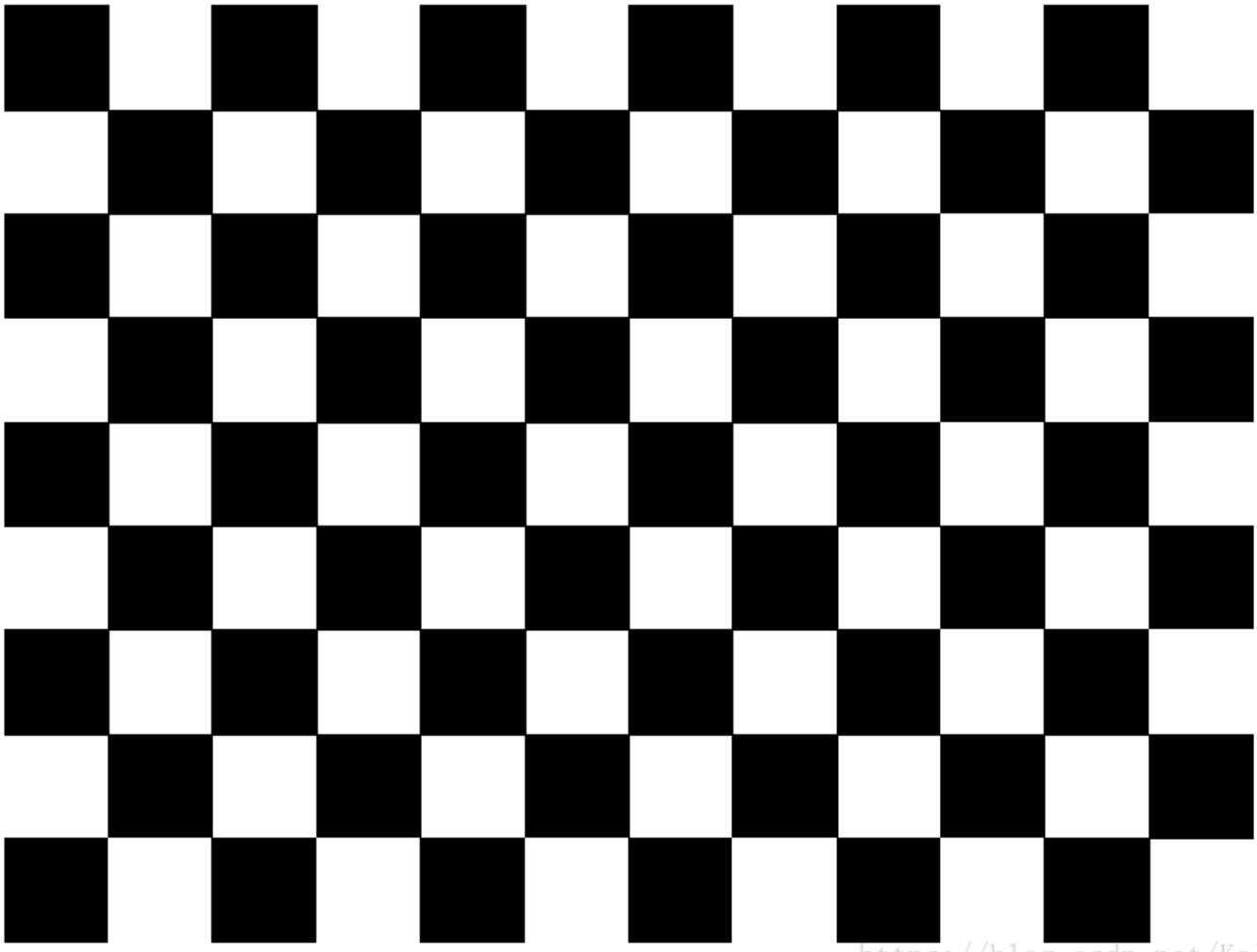
相机标定(二)——图像坐标与世界坐标转换

相机标定(三)——手眼标定

一、张正友标定算法实现流程

1.1 准备棋盘格

备注：棋盘格黑白间距已知，可采用打印纸或者购买黑白棋盘标定板（精度要求高）



1.2 针对棋盘格拍摄若干张图片

此处分两种情况

标定畸变系数和相机内参，拍摄照片需要包含完整棋盘，同时需要不同距离，不同方位，同时需要有棋盘不同倾斜角度。
标定畸变系数，相机内参和相机外参，图片包含上述要求，同时标定程序生成结果中每张照片会计算一个相机外参数因此根据实际需求，增加几张棋盘在工作位置的照片。（相机外参建议采用solvePnP函数获取）

1.3 检测图片中的内角点

角点检测

```
1
2bool findChessboardCorners(InputArray image,
3
4                           Size patternSize,
5
6                           OutputArray corners,
7
8                           int flags=CALIB_CB_ADAPTIVE_THRESH+CALIB_CB_NORMALIZE_IMAGE )
9
```

image，传入拍摄的棋盘图Mat图像，必须是8位的灰度或者彩色图像；
patternSize，每个棋盘图上内角点的行列数，一般情况下，行列数不要相同，便于后续标定程序识别标定板的方向；
corners，用于存储检测到的内角点图像坐标位置，一般用元素是Point2f的向量来表示：vector<Point2f> image_points_buf;
flage：用于定义棋盘图上内角点查找的不同处理方式，有默认值。

提取亚像素角点信息

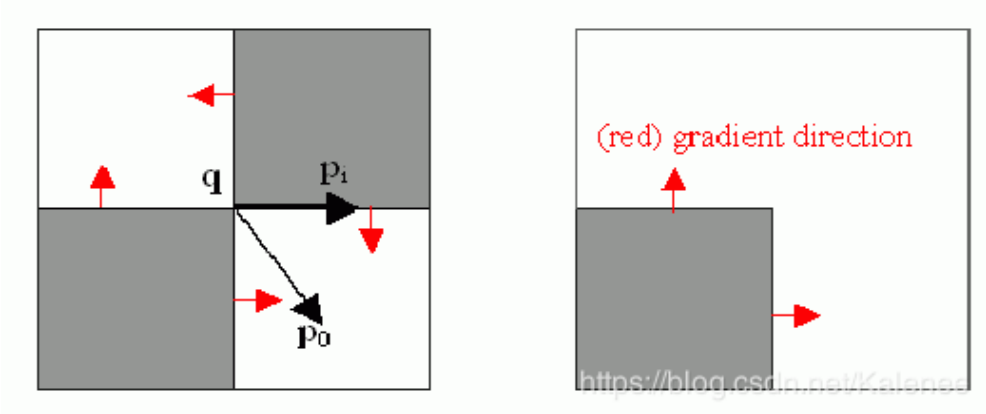
(1)cornerSubPix

```
1void cornerSubPix(InputArray image, InputOutputArray corners, Size winSize, Size zeroZone, TermCriteria criteria)
```

img，输入的Mat矩阵，最好是8位灰度图像，检测效率更高；

corners，初始的角点坐标向量，同时作为亚像素坐标位置的输出，所以需要是浮点型数据，一般用元素是Pointf2f/Point2d的向量来表示：vector<Point2f/Point2d> iamgePointsBuf；
winSize，搜索窗口边长的一半，例如如果winSize=Size(5,5)，则一个大小为的搜索窗口将被使用。
zeroZone，搜索区域中间的dead region边长的一半，有时用于避免自相关矩阵的奇异性。如果值设为(-1,-1)则表示没有这个区域。
criteria，角点精准化迭代过程的终止条件。也就是当迭代次数超过criteria.maxCount，或者角点位置变化小于criteria.epsilon时，停止迭代过程。

该函数通过迭代法查找角点亚像素精度下的精确位置，函数实现流程如下：



(2) find4QuadCornerSubpix

```
bool find4QuadCornerSubpix(InputArray img, InputOutputArray corners, Size region_size);
```

img，输入的Mat矩阵，最好是8位灰度图像，检测效率更高；
corners，初始的角点坐标向量，同时作为亚像素坐标位置的输出，所以需要是浮点型数据，一般用元素是Pointf2f/Point2d的向量来表示：vector<Point2f/Point2d> iamgePointsBuf；
region_size，角点搜索窗口的尺寸；

1.4 相机标定

```
double calibrateCamera(InputArrayOfArrays objectPoints,
                       InputArrayOfArrays imagePoints,
                       Size imageSize,
                       InputOutputArray cameraMatrix,
                       InputOutputArray distCoeffs,
                       OutputArrayOfArrays rvecs,
                       OutputArrayOfArrays tvecs,
                       int flags=0,
                       TermCriteria criteria=TermCriteria( TermCriteria::COUNT+TermCriteria::EPS, 30, DBL_EPSILON) )
```

objectPoints，为世界坐标系中的三维点。在使用时，应该输入一个三维坐标点的向量的向量，即vector<vector<Point3f>> object_points。需要依据棋盘上单个黑白矩阵的大小，计算出（初始化）每一个内角点的世界坐标。
imagePoints，为每一个内角点对应的图像坐标点。和objectPoints一样，应该输入vector<vector<Point2f>> image_points_seq形式的变量；
imageSize，为图像的像素尺寸大小，在计算相机的内参和畸变矩阵时需要使用到该参数；
cameraMatrix为相机的内参矩阵。输入一个Mat cameraMatrix即可，如Mat cameraMatrix=Mat(3,3,CV_32FC1,Scalar::all(0));
distCoeffs为畸变矩阵。输入一个Mat distCoeffs=Mat(1,5,CV_32FC1,Scalar::all(0))即可；
rvecs为旋转向量；应该输入一个Mat类型的vector，即vector<Mat>rvecs;
tvecs为位移向量，和rvecs一样，应该为vector<Mat> tvecs；
flags为标定时所采用的算法。有如下几个参数：
CV_CALIB_USE_INTRINSIC_GUESS：使用该参数时，在cameraMatrix矩阵中应该有fx,fy,u0,v0的估计值。否则的话，将初始化(u0,v0）图像的中心点，使用最小二乘估算出fx，fy。
CV_CALIB_FIX_PRINCIPAL_POINT：在进行优化时会固定光轴点。当CV_CALIB_USE_INTRINSIC_GUESS参数被设置，光轴点将保持在中心或者某个输入的值。
CV_CALIB_FIX_ASPECT_RATIO：固定fx/fy的比值，只将fy作为可变量，进行优化计算。当CV_CALIB_USE_INTRINSIC_GUESS没有被设置，fx和fy将会被忽略。只有fx/fy的比值在计算中会被用到。
CV_CALIB_ZERO_TANGENT_DIST：设定切向畸变参数（p1,p2）为零。
CV_CALIB_FIX_K1,...,CV_CALIB_FIX_K6：对应的径向畸变在优化中保持不变。
CV_CALIB_RATIONAL_MODEL：计算k4，k5，k6三个畸变参数。如果没有设置，则只计算其它5个畸变参数。
criteria是最优迭代终止条件设定。

在使用该函数进行标定运算之前，需要对棋盘上每一个内角点的空间坐标系的位置坐标进行初始化，默认参数下生成的标定的结果为相机内参矩阵cameraMatrix、相机的5个畸变系数distCoeffs，另外每张图像都会生成属于自己的平移向量和旋转向量。

1.5 结果评价

对标定结果进行评价的方法是通过得到的摄像机内外参数，对空间的三维点进行重新投影计算，得到空间三维点在图像上新的投影点的坐标，计算投影坐标和亚像素角点坐标之间的偏差，偏差越小，标定结果越好。

空间的三维点进行重新投影计算

```
1
2 void projectPoints( InputArray objectPoints,
3
4                             InputArray rvec,
5
6                             InputArray tvec,
7
8                             InputArray cameraMatrix,
9
10                            InputArray distCoeffs,
11
12                            OutputArray imagePoints,
13
14                            OutputArray jacobian=noArray(),
15
16                            double aspectRatio=0 );
17
```

objectPoints，为相机坐标系中的三维点坐标；
rvec为旋转向量，每一张图像都有自己的选择向量；
tvec为位移向量，每一张图像都有自己的平移向量；
cameraMatrix为求得的相机的内参数矩阵；
distCoeffs为相机的畸变矩阵；
iamgePoints为每一个内角点对应的图像上的坐标点；
acobian是雅可比行列式；
aspectRatio是跟相机传感器的感光单元有关的可选参数，如果设置为非0，则函数默认感光单元的dx/dy是固定的，会依此对雅可比矩阵进行调整；

1.6 保存结果

保存结果有两种格式。

txt用于查看结果。

```
//保存定标结果

cout << "开始保存定标结果……" << endl;

Mat rotation_matrix = Mat(3, 3, CV_32FC1, Scalar::all(0)); /* 保存每幅图像的旋转矩阵 */

fout << "相机内参数矩阵：" << endl;

fout << cameraMatrix << endl << endl;

fout << "畸变系数：\n";

fout << distCoeffs << endl << endl << endl;

for (int i = 0; i<image_count; i++)

{

        fout << "第" << i + 1 << "幅图像的旋转向量：" << endl;

        fout << rvecsMat[i] << endl;

        /* 将旋转向量转换为相对应的旋转矩阵 */

        Rodrigues(rvecsMat[i], rotation_matrix);

        fout << "第" << i + 1 << "幅图像的旋转矩阵：" << endl;

        fout << rotation_matrix << endl;

        fout << "第" << i + 1 << "幅图像的平移向量：" << endl;

        fout << tvecsMat[i] << endl << endl;

}

fout << endl;
```

xml用于将标定结果应用于视觉处理。

```
//保存相机内参数矩阵和畸变系数和相机距离到xml

cout << "开始保存相机内参数矩阵和畸变系数……" << endl;

string xmlResult = filePath + "\\calibrateImage\\caliberation_camera.xml";

FileStorage fs(xmlResult, FileStorage::WRITE); //创建XML文件

fs << "zConst" << 100.0;

fs << "cameraMatrix" << cameraMatrix << "distCoeffs" << distCoeffs;

fs.release();
```

二、程序实现

运行程序后输入保存图片目录的绝对路径，标定结果默认在该目录的calibrateImage文件夹内。

```
1
2 #include "stdafx.h"
3
4 #include <iostream>
5
6 #include <time.h>
7
8 #include <fstream>
9
10 #include <io.h>
11
12 #include <string>
13
14 #include <direct.h>
15
16 #include <vector>
17
18 using namespace std;
19
20
21
22 #include <opencv2/opencv.hpp>
23
24 using namespace cv;
25
26
27
28 //标定板方格边长，行角点，列角点
29
30 #define BOARD_SCALE 30
31
32 #define BOARD_HEIGHT 11
33
34 #define BOARD_WIDTH 8
35
36
37
38 //获取特定格式的文件名
39
40 void GetAllFormatFiles(string path, vector<string>& files, string format)
41
42 {
43
44         //文件句柄
45
46         //long   hFile = 0;//win7使用
47
48         intptr_t hFile = 0;//win10使用
49
50                                         //文件信息
51
52         struct _finddata_t fileinfo;
```

```
53
54          string p;
55
56          if ((hFile = _findfirst(p.assign(path).append("\\*" + format).c_str(), &fileinfo)) != -1)
57
58          {
59
60                  do
61
62                  {
63
64                          if ((fileinfo.attrib &  _A_SUBDIR))
65
66                          {
67
68                                  if (strcmp(fileinfo.name, ".") != 0 && strcmp(fileinfo.name, "..") != 0)
69
70                                  {
71
72                                          //files.push_back(p.assign(path).append("\\").append(fileinfo.name) );
73
74                                          GetAllFormatFiles(p.assign(path).append("\\").append(fileinfo.name), files, format);
75
76                                  }
77
78                          }
79
80                          else
81
82                          {
83
84                                  //files.push_back(p.assign(path).append("\\").append(fileinfo.name));//将文件路径保存
85
86                                  files.push_back(p.assign(fileinfo.name));  //只保存文件名:
87
88                          }
89
90                  } while (_findnext(hFile, &fileinfo) == 0);
91
92                  _findclose(hFile);
93
94          }
95
96  }
97
98
99
100 void main()
101
102 {
103
104         vector<string> imageFilesName;
105
106         vector<string> files;
107
108         imageFilesName.clear(); files.clear();
109
110         string filePath;
111
112         cout << "请输入标定照片文件绝对目录路径" << endl;
113
114         cin >> filePath;
```

```
115
116     string format = ".jpg";
117
118     GetAllFormatFiles(filePath, imageFilesName, format);
119
120     cout << "找到的文件有" << endl;
121
122     for (int i = 0; i < imageFilesName.size(); i++)
123
124     {
125
126             files.push_back(filePath + "\\" + imageFilesName[i]);
127
128             cout << files[i] << endl;
129
130     }
131
132     string calibrateDir = filePath + "\\calibrateImage";
133
134     _mkdir(calibrateDir.c_str());
135
136
137
138     //读取每一幅图像，从中提取出角点，然后对角点进行亚像素精确化
139
140     cout << "开始提取角点……" << endl;
141
142     int image_count = 0;  /* 图像数量 */
143
144     Size image_size;  /* 图像的尺寸 */
145
146     Size board_size = Size(BOARD_HEIGHT, BOARD_WIDTH);    /* 标定板上每行、列的角点数 */
147
148     vector<Point2f> image_points_buf;  /* 缓存每幅图像上检测到的角点 */
149
150     vector<vector<Point2f>> image_points_seq; /* 保存检测到的所有角点 */
151
152
153
154     for (int i = 0; i<files.size(); i++)
155
156     {
157
158             cout << files[i] << endl;
159
160
161
162             Mat imageInput = imread(files[i]);
163
164             /* 提取角点 */
165
166             if (0 == findChessboardCorners(imageInput, board_size, image_points_buf))
167
168             {
169
170                     cout << "can not find chessboard corners!\n"; //找不到角点
171
172                     continue;
173
174             }
175
176             else
```

```
177
178             {
179
180                     //找到一幅有效的图片
181
182                     image_count++;
183
184                     if (image_count == 1)  //读入第一张图片时获取图像宽高信息
185
186                     {
187
188                             image_size.width = imageInput.cols;
189
190                             image_size.height = imageInput.rows;
191
192                             cout << "image_size.width = " << image_size.width << endl;
193
194                             cout << "image_size.height = " << image_size.height << endl;
195
196                     }
197
198
199
200                     Mat view_gray;
201
202                     cvtColor(imageInput, view_gray, CV_RGB2GRAY);
203
204
205
206                     /* 亚像素精确化 */
207
208                     find4QuadCornerSubpix(view_gray, image_points_buf, Size(5, 5)); //对粗提取的角点进行精确化，更适合用于棋盘标定
209
210                     /*cornerSubPix(view_gray, image_points_buf,//另一种对粗提取的角点进行精确化
211
212                                             Size(5, 5),
213
214                                             Size(-1, -1),
215
216                                             TermCriteria(TermCriteria::MAX_ITER + TermCriteria::EPS,
217
218                                             30,        // max number of iterations
219
220                                             0.1));     // min accuracy*/
221
222                     image_points_seq.push_back(image_points_buf);  //保存亚像素角点
223
224
225
226                                                                                                     /* 在图像上显示角点位置 */
227
228                     drawChessboardCorners(view_gray, board_size, image_points_buf, true); //用于在图片中标记角点
229
230                     //string filePath = files[i];//写入文件
231
232                     string filePath = calibrateDir + "\\"+ imageFilesName[i] + ".jpg";
233
234                     imwrite(filePath, view_gray);
235
236             }
237
```

```
    }


    int total = image_points_seq.size();
    cout << "共使用了" << total << "幅图片" << endl;
    cout << "角点提取完成！\n";
    cout << "开始标定……\n";
    /*棋盘三维信息*/
    Size square_size = Size(BOARD_SCALE, BOARD_SCALE);  /* 实际测量得到的标定板上每个棋盘格的大小 */
    vector<vector<Point3f>> object_points; /* 保存标定板上角点的三维坐标 */


    /*内外参数*/
    Mat cameraMatrix = Mat(3, 3, CV_32FC1, Scalar::all(0)); /* 摄像机内参数矩阵 */
    vector<int> point_counts;  // 每幅图像中角点的数量
    Mat distCoeffs = Mat(1, 5, CV_32FC1, Scalar::all(0)); /* 摄像机的5个畸变系数：k1,k2,p1,p2,k3 */
    vector<Mat> tvecsMat;  /* 每幅图像的旋转向量 */
    vector<Mat> rvecsMat; /* 每幅图像的平移向量 */
                                          /* 初始化标定板上角点的三维坐标 */
    int i, j, t;
    for (t = 0; t<image_count; t++)
    {
            vector<Point3f> tempPointSet;
            for (i = 0; i<board_size.height; i++)
            {
                    for (j = 0; j<board_size.width; j++)
                    {
                            Point3f realPoint;
                            /* 假设标定板放在世界坐标系中z=0的平面上 */
                            realPoint.x = i * square_size.width;
                            realPoint.y = j * square_size.height;
                            realPoint.z = 0;
                            tempPointSet.push_back(realPoint);
```

```
                }

        }

        object_points.push_back(tempPointSet);

    }


    /* 初始化每幅图像中的角点数量，假定每幅图像中都可以看到完整的标定板 */

    for (i = 0; i<image_count; i++)

    {

        point_counts.push_back(board_size.width*board_size.height);

    }


    /* 开始标定 */

    calibrateCamera(object_points, image_points_seq, image_size, cameraMatrix, distCoeffs, rvecsMat, tvecsMat, 0);

    cout << "标定完成！\n";


    //对标定结果进行评价

    string txtResult = filePath + "\\calibrateImage\\caliberation_result.txt";

    ofstream fout(txtResult);  /* 保存标定结果的文件 */


    double total_err = 0.0; /* 所有图像的平均误差的总和 */

    double err = 0.0; /* 每幅图像的平均误差 */

    vector<Point2f> image_points2; /* 保存重新计算得到的投影点 */

    cout << "\t每幅图像的标定误差：\n";

    fout << "每幅图像的标定误差：\n";

    for (i = 0; i<image_count; i++)

    {

        vector<Point3f> tempPointSet = object_points[i];

        /* 通过得到的摄像机内外参数，对空间的三维点进行重新投影计算，得到新的投影点 */

        projectPoints(tempPointSet, rvecsMat[i], tvecsMat[i], cameraMatrix, distCoeffs, image_points2);

        /* 计算新的投影点和旧的投影点之间的误差*/

        vector<Point2f> tempImagePoint = image_points_seq[i];
```

```
362             Mat tempImagePointMat = Mat(1, tempImagePoint.size(), CV_32FC2);
363
364             Mat image_points2Mat = Mat(1, image_points2.size(), CV_32FC2);
365
366             for (int j = 0; j < tempImagePoint.size(); j++)
367
368             {
369
370                     image_points2Mat.at<Vec2f>(0, j) = Vec2f(image_points2[j].x, image_points2[j].y);
371
372                     tempImagePointMat.at<Vec2f>(0, j) = Vec2f(tempImagePoint[j].x, tempImagePoint[j].y);
373
374             }
375
376             err = norm(image_points2Mat, tempImagePointMat, NORM_L2);
377
378             total_err += err /= point_counts[i];
379
380             cout << "第" << i + 1 << "幅图像的平均误差：" << err << "像素" << endl;
381
382             fout << "第" << i + 1 << "幅图像的平均误差：" << err << "像素" << endl;
383
384     }
385
386
387
388     cout << "总体平均误差：" << total_err / image_count << "像素" << endl;
389
390     fout << "总体平均误差：" << total_err / image_count << "像素" << endl << endl;
391
392
393
394     //保存定标结果
395
396     cout << "开始保存定标结果……" << endl;
397
398     Mat rotation_matrix = Mat(3, 3, CV_32FC1, Scalar::all(0)); /* 保存每幅图像的旋转矩阵 */
399
400     fout << "相机内参数矩阵：" << endl;
401
402     fout << cameraMatrix << endl << endl;
403
404     fout << "畸变系数：\n";
405
406     fout << distCoeffs << endl << endl << endl;
407
408     for (int i = 0; i<image_count; i++)
409
410     {
411
412             fout << "第" << i + 1 << "幅图像的旋转向量：" << endl;
413
414             fout << rvecsMat[i] << endl;
415
416             /* 将旋转向量转换为相对应的旋转矩阵 */
417
418             Rodrigues(rvecsMat[i], rotation_matrix);
419
420             fout << "第" << i + 1 << "幅图像的旋转矩阵：" << endl;
421
422             fout << rotation_matrix << endl;
423
```

```
424             fout << "第" << i + 1 << "幅图像的平移向量：" << endl;
425
426             fout << tvecsMat[i] << endl << endl;
427
428         }
429
430         fout << endl;
431
432
433
434         //保存相机内参数矩阵和畸变系数和相机距离到xml
435
436         cout << "开始保存相机内参数矩阵和畸变系数……" << endl;
437
438         string xmlResult = filePath + "\\calibrateImage\\caliberation_camera.xml";
439
440         FileStorage fs(xmlResult, FileStorage::WRITE); //创建XML文件
441
442         fs << "zConst" << 100.0;
443
444         fs << "cameraMatrix" << cameraMatrix << "distCoeffs" << distCoeffs;
445
446         fs.release();
447
448         //保存平移矩阵和旋转矩阵和s到xml
449
450         string xml2Result = filePath + "\\calibrateImage\\solvePnP_camera.xml";
451
452         FileStorage fs2(xml2Result, FileStorage::WRITE); //创建XML文件
453
454         double s = 100.0;
455
456         fs2 << "s" << s;
457
458         fs2 << "rotation_matrix" << rotation_matrix << "tvecsMat" << tvecsMat[0];
459
460         fs2.release();
461
462         cout << "保存完成" << endl;
463
464
465
466         //保存矫正图像
467
468         Mat mapx = Mat(image_size, CV_32FC1);
469
470         Mat mapy = Mat(image_size, CV_32FC1);
471
472         Mat R = Mat::eye(3, 3, CV_32F);
473
474         cout << "保存矫正图像" << endl;
475
476         initUndistortRectifyMap(cameraMatrix, distCoeffs, R, cameraMatrix, image_size, CV_32FC1, mapx, mapy);
477
478         for (int i = 0; i != image_count; i++)
479
480         {
481
482             cout << "Frame #" << i + 1 << "..." << endl;
483
484             Mat imageSource = imread(files[i]);
485
```

```
            Mat newimage = imageSource.clone();

            //另一种不需要转换矩阵的方式

            //undistort(imageSource,newimage,cameraMatrix,distCoeffs);

            remap(imageSource, newimage, mapx, mapy, INTER_LINEAR);//效率更高

            string imageFilePath = calibrateDir + "\\" + imageFilesName[i] +"_d"+ ".jpg";

            imwrite(imageFilePath, newimage);

        }

        cout << "保存结束" << endl;


        cout << "全部工作结束" << endl;

        int key = cvWaitKey(0);

        return;

}
```

参考

https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

https://blog.csdn.net/u010128736/article/details/52860364

https://blog.csdn.net/dcrmg/article/details/52939318