专业：求是科学班（计算机）
姓名：蒋仕彪
学号：3170102587
日期：2019/12/10

浙江大学实验报告

课程名称：计算机视觉　　指导老师：宋明黎　　成绩：_____
实验名称：HW#4: calibration and transformation

一、实验目的和要求

- Camera calibration
  Ex 11-1, 11-2
- Bird's-eye view transformation
  Ex 12-1
- Combine them together

二、实验内容和原理

2.1 相机标定之内参

通过相机，我们把三维世界的信息展示在两维的像素平面上，如图是简单的相似三角形原理：

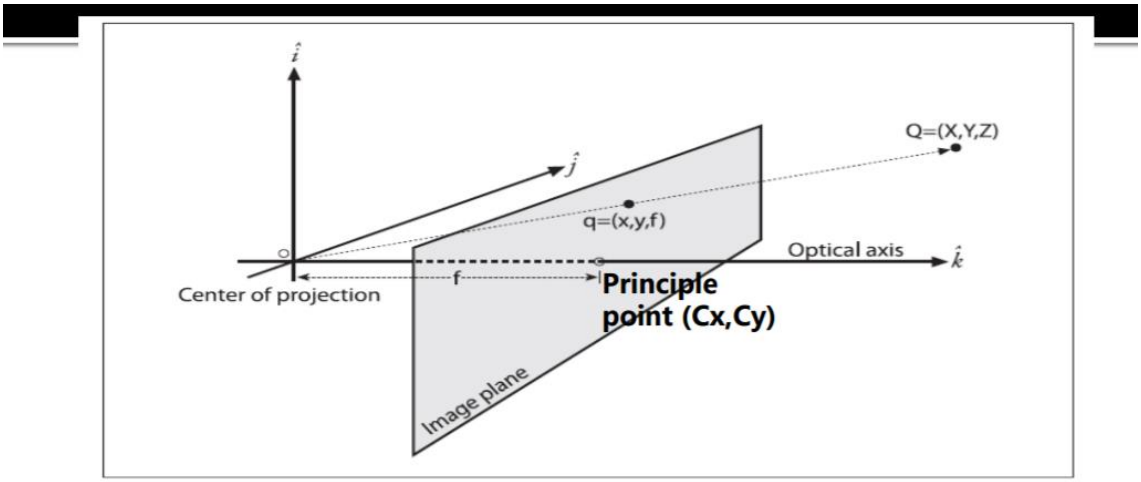


Figure 11-2. A point Q = (X, Y, Z) is projected onto the image plane by the ray passing through the center of projection, and the resulting point on the image is q = (z, y, f); the image plane is really just the projection screen "pushed" in front of the pinhole (the math is equivalent but simpler this way)

However, principle point is not at the center.

$$x = f\frac{X}{Z} \longrightarrow x = f\left(\frac{X}{Z}\right) + c_x$$

注意到，每一维坐标进行变换的时候，还要额外加一个 Cx（因为投影的时候不是正好经过中心点，像素坐标要加上相对于像素平面的原点的位移）。这样，x 坐标对应比例因子 Fx 和位移因子 Cy，y 坐标也有相应的 Fy 和 Cy，这四个参数结合起来，被称作相机的内参。

还有一点是，Fx 和 Fy 其实是由原来的长度 X 和 Y 除上了放缩因子 Z。有了除法后，我们很难把这个投影的行为写成线性的矩阵运算。这里要用到一个 Trick：多加一维，强行改成线性运算：

$$(x,y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w)$$

我们在运算时加入常量那一维。在矩阵运算完后，要除掉最后那一维才是真正的坐标值。这样，我们可以把这个投影运算正式地写成一个线性矩阵乘法，如下：

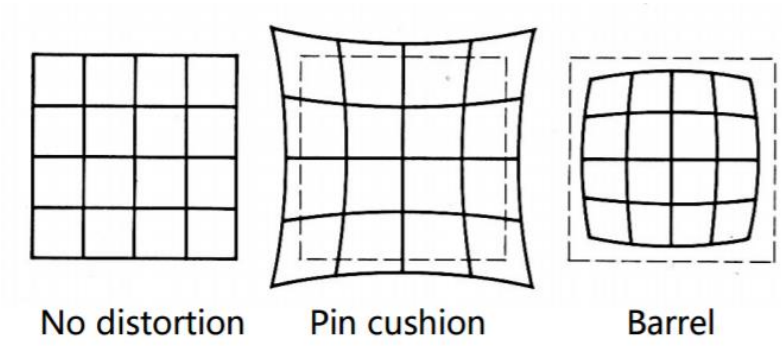
$$q = MQ, \quad \text{where} \quad q = \begin{bmatrix} x \\ y \\ w \end{bmatrix}, \quad M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \quad Q = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

M 矩阵被称为**内参矩阵，**对应的五个参数被称为**内参**（intrinsic parameters）。

2.2 相机标定之畸变

在现实生活中，投影不是规则的数学形式，可能会产生畸变。所以我们要用相应的模型去刻画畸变。一共有两种畸变的模型，被称为**径向畸变**（Radial distortion）和**切向畸变**（Tangential distortion）。

径向畸变原理很简单，所有点按照离中心的距离远近进行对应的畸变，如下图：

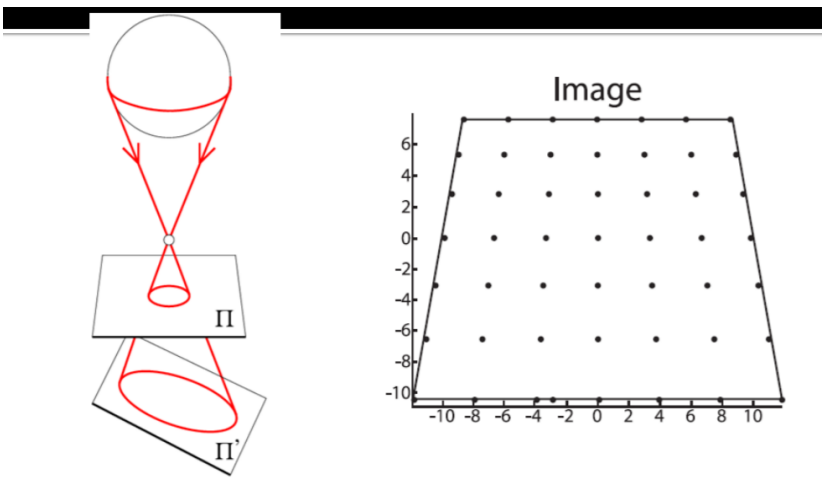


因为是和距离有关的，我们用与 r^2 有关的高次项去拟合：

$$x_{\text{corrected}} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$y_{\text{corrected}} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

切向畸变的形成是因为透镜平面和像素平面不在同一个平行面上，如下图：



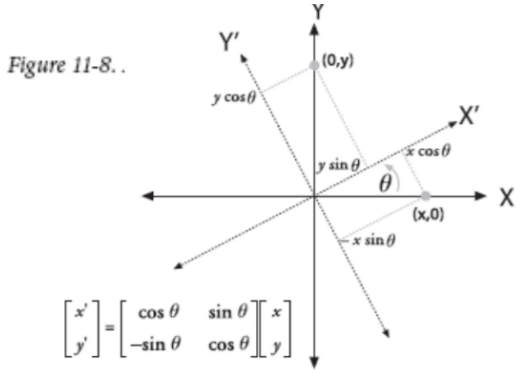
我们用这样的公式去拟合切向畸变：

$$x_{\text{corrected}} = x + [2p_1 y + p_2(r^2 + 2x^2)]$$
$$y_{\text{corrected}} = y + [p_1(r^2 + 2y^2) + 2p_2 x]$$

以上两种畸变一共带来了 k1,k2,k3,p1,p2 五个参数。他们和相机属性无关但和投影息息相关，所以被称为**畸变参数**（distortion parameters）。

2.3 相机标定之外参

一般情况下，世界坐标系和相机坐标系不重合，这时，世界坐标系中的某一点 P 要投影到像面上时，先要将该点的坐标转换到相机坐标系下。刚体从世界坐标系转换到相机坐标系的过程，可以通过旋转和平移来得到。旋转的操作如下：



$$R = R_z(\theta),\ R_y(\varphi),\ R_x(\psi)$$
$$\mathbf{t}_{3\times1} = (t_x, t_y, t_z)'$$
$$P' = \begin{bmatrix} \mathbf{R}_{3\times3} & \mathbf{t}_{3\times1} \\ 0\ \ 0\ \ 0 & 1 \end{bmatrix} P$$

$$R_x(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & \sin\psi \\ 0 & -\sin\psi & \cos\psi \end{bmatrix} \quad R_y(\varphi) = \begin{bmatrix} \cos\varphi & 0 & -\sin\varphi \\ 0 & 1 & 0 \\ \sin\varphi & 0 & \cos\varphi \end{bmatrix} \quad R_z(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

注意要先旋转再平移，因为基于原点的旋转会很简单。三个旋转角和三维对应的平移量，加起来一共是六个参数，被称为 **外参**（extrinsic parameters）

2.4 相机标定（Camera calibration）

经过上面三节的分析，我们可以把投影过程看成如下的一系列矩阵运算：

$$\Pi = \begin{bmatrix} fs_x & 0 & x_c \\ 0 & fs_y & y_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R}_{3x3} & \mathbf{0}_{3x1} \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I}_{3x3} & \mathbf{T}_{3x1} \\ \mathbf{0}_{1x3} & 1 \end{bmatrix}$$

intrinsics　projection　rotation　translation

相机标定的目的是，通过已经配对的三维坐标和二维坐标，去还原这些参数。

本次实验提供的第一个程序就是在相机标定[1]，我对其进行了简要的研究和分析。

该程序的任务是：从一个文件夹里读取所有图片（假设都是和 chessboard 相关），运用 opencv 的一些函数，先求出 chessboard 的每一个格点；然后调用 opencv 的另一些 API，求出畸变参数和内参参数并保存在一个 xml 里，表明拍摄相机对应的参数。

[1] 因教材改版，新版中 Camera calibration 对应的例程是 18-1。18-1 又分为两个代码，一个是直接通过视频抓拍图片，还有一个是从文件里读入。为了调用附带的图片，显然是要选取 example_18-01_from_disk.cpp 这份代码。

当程序处理了文件读入的信息，获得了每张图片后，先要通过内置函数求出对应的格子点坐标：

```
// Find the board
vector<cv::Point2f> corners;
bool found = cv::findChessboardCorners(image, board_sz, corners);

// Draw it
drawChessboardCorners(image, board_sz, corners, found);  // will draw only if found
```

FindChessboardCorners 即为从 image 里找 Size 为 board_sz 的格点，把结果存在 corners 里。其中 corners 的定义为：vector<cv::Point2f> corners;

找完后用 drawChessBoardCorners 可以把格子用彩色标出来方便调试。

预处理完信息后就可以开始标定了，标定也是调用 API：

```
cv::Mat intrinsic_matrix, distortion_coeffs;
double err = cv::calibrateCamera(
    object_points, image_points, image_size, intrinsic_matrix,
    distortion_coeffs, cv::noArray(), cv::noArray(),
    cv::CALIB_ZERO_TANGENT_DIST | cv::CALIB_FIX_PRINCIPAL_POINT);

// SAVE THE INTRINSICS AND DISTORTIONS
cout << " *** DONE!\n\nReprojection error is " << err
    << "\nStoring Intrinsics.xml and Distortions.xml files\n\n";
cv::FileStorage fs("intrinsics.xml", cv::FileStorage::WRITE);
fs << "image_width" << image_size.width << "image_height" << image_size.height
    << "camera_matrix" << intrinsic_matrix << "distortion_coefficients"
    << distortion_coeffs;
fs.release();
```

代码原理很简单，但是我在 Windows 下的 VS2017 里直接跑这份代码的时候还是遇到了很多问题。

首先，可能是因为跨平台的缘故，文件目录结构会出现问题：

```
for (size_t i = 0; (i < filenames.size()) && (board_count < n_boards); ++i) {
    cv::Mat image, image0 = cv::imread(filenames[i]);
    board_count += 1;
    if (!image0.data) {  // protect against no file
        cerr << filenames[i] << ", file #" << i << ", is not an image" << endl;
        continue;
    }
    image_size = image0.size();
    cv::resize(image0, image, cv::Size(), image_sf, image_sf, cv::INTER_LINEAR);
```

标红处原来的代码是 folder + filename[i]，但是我这里调用了文件夹下的文件后，filename[i] 已经包含了 folder 的目录，所以要把 folder 这一句给去掉。

其次，代码里做了一个奇怪的颜色变换操作：

```
if (found) {
    image ^= cv::Scalar::all(255);
    cv::Mat mcorners(corners);
```

刚开始跑这份代码时，我输出的图片的色彩一直有问题。直到我找到这句话并删掉它后才输出了正确的颜色。**对此我的推理是：不同平台下读入的 RGB 通道顺序可能不同，这个操作可能是在调通道的顺序。**

进行一系列的小修改后，这份代码就可以正常的运行了。

这份代码的调用规则如下：

```
ERROR: Wrong number (0) of arguments, should be (6) input parameters


Example 18-1 (from disk):Enter a chessboard鈥樷 width and height,
             reading in a directory of chessboard images,
             and calibrating the camera

Call:
calibrate <1board_width> <2board_height> <3number_of_boards> <4ms_delay_framee_capture> <5image_scaling_factor> <6path_t
o_calibration_images>


Example:
./example_18-01_from_disk 9 6 14 100 1.0 ../stereoData/
or:
./example_18-01_from_disk 12 12 28 100 0.5 ../calibration/

 * First it reads in checker boards and calibrates itself
 * Then it saves and reloads the calibration matricies
 * Then it creates an undistortion map and finaly
 * It displays an undistorted image
```

六个参数分别表示 board 的长宽交点数（**注意是点数，不是格子数，这里坑了我好久**），棋盘图片的数量，幻灯片展示时的间隔，图片的缩放因子，图片存储的文件目录。具体的测试结果见第三章。

2.6 变换程序分析

还有一个程序是用来生成 Bird's Eye Review 的[2]。

该程序的任务是：从一个文件夹里读取所有图片（假设都是和 chessboard 相关）以及一个参数文件。同样先利用 API 求出 chessboard 的格点坐标。然后运用这个参数文件里的相机参数，生成这些图片对应的 Bird's Eye Review 的结果。

找到格子坐标后，该程序会首先输出原图像上画格子的结果，作为一个对照（**我运行该程序的时候输出的图片过大难以查看情况，底下的 md 变量是对图片进行放缩，并用 namedWindows 控制图片位置**）

```
cv::drawChessboardCorners(image, board_sz, corners, found);
cv::Mat md; resize(image, md, image.size());
cv::namedWindow("Checkers", 0);
cv::moveWindow("Checkers", 300, 500);
cv::imshow("Checkers", md);
```

随后程序进行变换，变换的核心代码其实也就是调用一个 API：

```
cv::warpPerspective(image,           // Source image
                    birds_image,     // Output image
                    H,               // Transformation matrix
                    image.size(),    // Size for output image
                    cv::WARP_INVERSE_MAP | cv::INTER_LINEAR,
                    cv::BORDER_CONSTANT, cv::Scalar::all(0) // Fill border with black
                    );

cv::Mat md; resize(birds_image, md, birds_image.size() / 4);
cv::imshow("Birds_Eye", md);
```

同时还用一个变量 Z 来控制 view，使得键盘可以进行交互。

```
int key = cv::waitKey() & 255;
if (key == 'u') Z += 0.5;
if (key == 'd') Z -= 0.5;
```

显然，上述 API 里的 H 是由 Z 来决定的：`H.at<double>(2, 2) = Z;`

[2] 因教材改版，新版中 Camera calibration 对应的例程是 19-1。

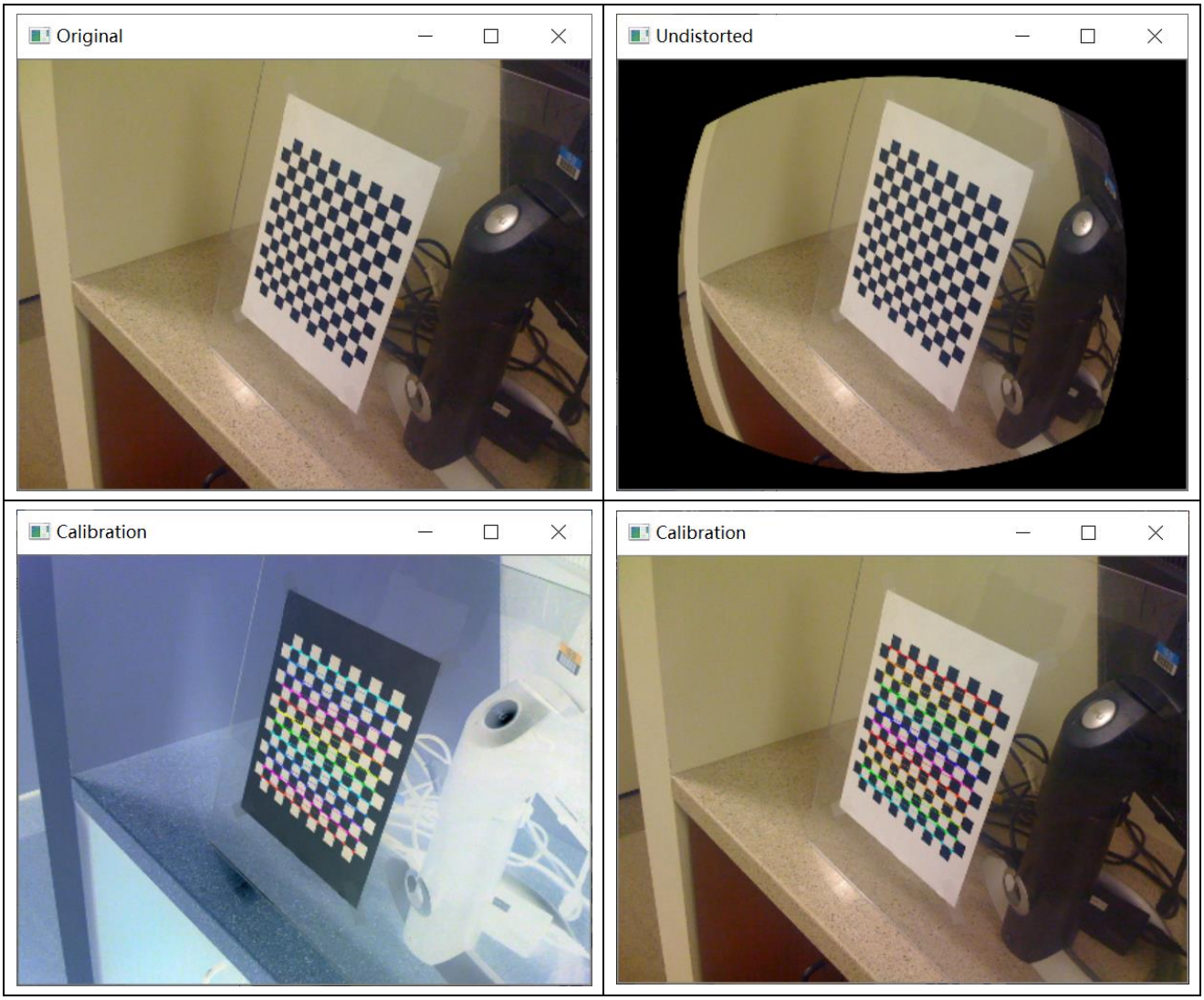
程序所需的参数如下：

```
ERROR: too few parameters

Example 19-01, using homography to get a bird's eye view.
This file relies on you having created an intrinsic file via example_18-01_from_disk
   but here, that file is already stored in ../birdseye/intrinsics.xml
Call:
./example_19-01 <chessboard_width> <chessboard_height> <path/camera_calib_filename> <path/chessboard_image>

Example:
./example_19-01 12 12 ../birdseye/intrinsics.xml ../birdseye/IMG_0215L.jpg

Press 'd' for lower birdseye view, and 'u' for higher (it adjusts the apparent 'Z' height), Esc to exit
```

前两个和之前的程序一样，是网格的格子数。第三个参数是相机参数文件存放位置，第四个参数是棋盘图片存放位置。这个参数还是很好理解的。具体的测试结果见下。

三、成果展示

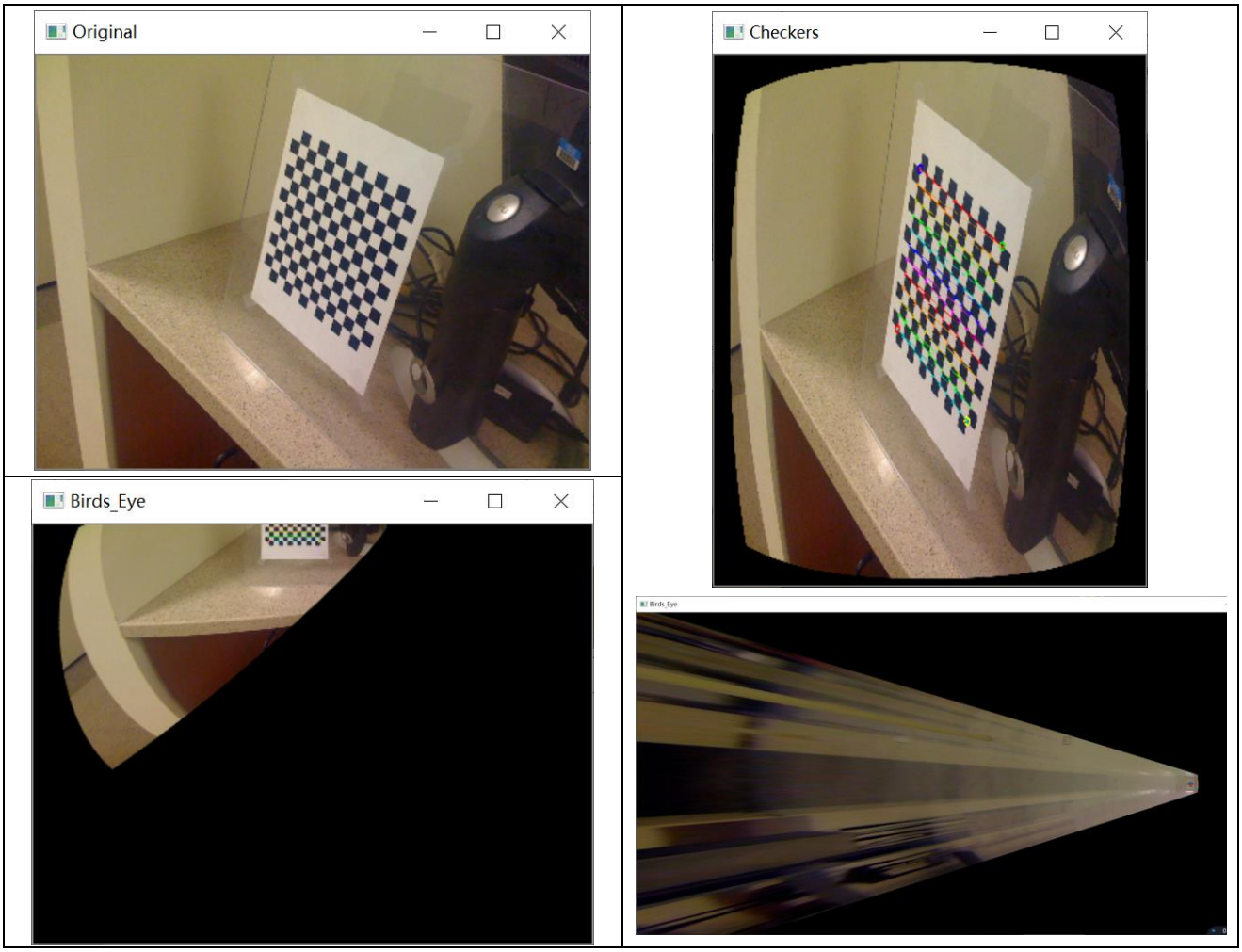
2.1 标定程序测试



左上角是 194.jpg 的原图，右上角是“将扭曲的棋盘去畸变”后的图像。可以看到，把棋盘强行扳正后，周围一圈的物体产生相应的畸变；可惜棋盘的变化观察起来比较困难，说明相机质量好，不太有畸变。

下方两张都是棋盘检测后的 check 图像，其中左侧是未修改代码跑的结果，右侧是正确代码跑的结果。

2.2 变换程序测试



左上角是 194.jpg 的原图，右上角是变换程序检测 chessboard 后的图像。

左下角是对应 intrinsics.xml 生成出来的 view。**可以发现，虽然周围的东西都扭曲了，棋盘变成了一个正四边形**。右下角是错误 intrinsics.xml 生成出来的 view（完全不能看的结果）

四、感想

本次实验看似简单（实现两个程序的对接），其实能学到很多东西。

Opencv 的基本操作还是获得了一些长进。直接复制下来的程序是无法正确运行的，要通过一些调试才能跑出令人满意的答案。我遇到的问题主要有：①文件路径里已经包含了父目录，无需叠加；②RGB 通道无需做变换，要删掉变换的那一行；③生成的图片过大无法查看，要在输出前将其 Resize；④imwrite 生成的图片位置不固定，要用 namedWindows 控制其方位……

刚开始调好程序后，Bird's Eye Review 跑出来的结果特别奇怪：完全失去了图片本来的性质。我只能硬着头皮一步步 debug。在判断 intrinsics.xml 里的数据是否生成错误的时候，可以把 calibration 文件夹里提供的 intrinsics.xml 拿过来，直接用变换程序跑一跑；如果结果正确说明求参数程序有问题，反之变换程序有问题。经过不懈的努力，变换程序终于能还原出正四面体的网格啦~成就感满满。

以本实验为契机，我好好复习了一下相机的内参矩阵、外参矩阵和畸变系数相关的知识，并做了相关的整理。这也为理解本实验的内涵打下了理论基础。