

# Stereo Project Report

赵懿-武汉大学网络安全学院

邮箱:*yizhaome@gmail.com*

项目github地址:<https://github.com/zhaone/ProjectStereo>

2018 年 9 月 6 日

目录

1 Camera Basics 3

1.1 第一问: intrinsics, extrinsics, camera matrix 3

1.1.1 模型 3

1.1.2 Extrinsics matrix的含义 4

1.1.3 Intrinsics matrix的含义 4

1.2 第二问: 3D到2D (自己推的) 6

1.3 第三问: 2D到3D (自己推的) 6

1.4 第四问: Distortion, 相机畸变 6

1.5 第五问: Calibration相机标定 7

1.6 第六问: openCV相机标定 7

1.7 第七问: openCV去畸变undistort 8

1.8 第八问: 张友正标定的实现 (自己写的) 8

1.9 单相机深度估计可行性 (自己推的) 9

2 Binocular Basics 9

2.1 第九问: Projection 9

2.2 第十问: Epipolar Line 10

2.3 第十一问: Fundamental Matrix 11

2.4 第十二问: Stereo Calibration 12

2.5 第十三问: 3D坐标点 (自己推的) 12

2.6 第十四问: Rectification 12

2.7 第十五问: Baseline (自己推的) 13

2.8 第十六问: Depth-Disparity (自己推的) 14

3 Stereo Matching 15

A 附录: 相机标定openCV与自己实现结果比较 16

B 附录: 立体标定openCV结果与自己实现结果 18

# 1 Camera Basics

## 1.1 第一问: intrinsics, extrinsics, camera matrix

解释这个问题首先需要解释相机拍摄真实世界物体进行的模型。

### 1.1.1 模型

对于拍摄问题存在三个坐标系 [4]:

- 世界坐标系: 用于表示世界中的物体 (点, 线, 表面等) 的固定坐标系, 3D
- 相机坐标系: 以摄像机中心为原点 (光轴为Z轴) 的坐标系, 3D
- 底片坐标系: 测量图像平面中像素位置的坐标系, 位于相机底片上, 2D

拍摄过程即是对真实世界的一个点进行投影变换到底片上的过程。即世界坐标系的一个点经过投影变换到底片坐标系中的一个点。我这对模型画了一个示意图 (#图是自己画的有点难看):

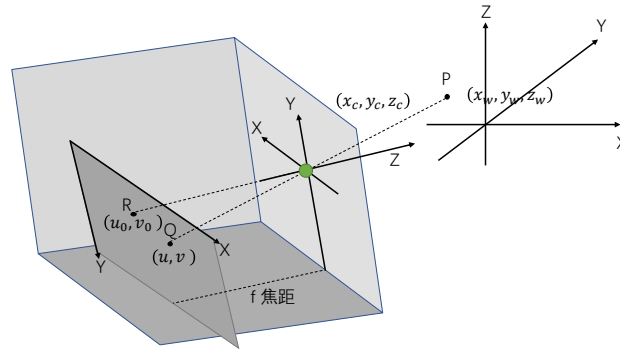


图 1: 投影模型

在我画的模型其中: 点P值真实世界中一点;  $(x_w, y_w, z_w)$ : 点P在世界坐标系中的坐标;  $(x_c, y_c, z_c)$ : 点P在相机坐标系中的坐标; 点R: 底片中心坐标, 也是相机坐标系Z轴 (光轴) 和底片相交的点; 点Q: 点P在底片的投影;  $(u, v)$ : Q 在底片坐标系中的坐标;  $f$ 为相机焦距。

图(2)显示了从P到Q包含两个变换:

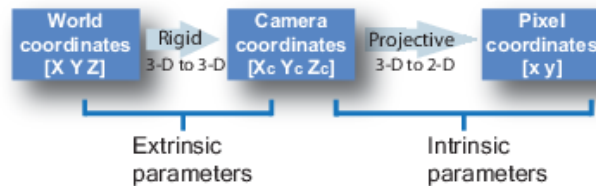


图 2: 投影变换的坐标系变换 [5]

写成矩阵形式如下：

$$z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \left[ R \mid T \right] \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \quad (1)$$

其中 $K$ 为intrinsic matrix（也即是camera matrix）， $[R|T]$ 为extrinsic matrix [8]。

### 1.1.2 Extrinsic matrix的含义

由模型可以看出，extrinsic matrix( $[R|T]$ )是世界坐标点 $P(3D)$ 到相机坐标点 $P(3D)$ 的投影转换矩阵。真实世界点 $P$ 坐标左乘 $[R|T]$ 后得到点 $P$ 在相机坐标系中的坐标值：

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \left[ R \mid T \right] \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \quad (2)$$

$$\left[ R \mid T \right] = \left[ \begin{array}{ccc|c} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \end{array} \right] \quad (3)$$

这个变换可以理解为一个坐标系的旋转和平移，其中 $R$ 是旋转矩阵， $T$ 是位移向量。由线性代数的知识可以指导，两个坐标系都是3D的，所以 $R$ 的shape是3\*3， $T$ 的shape是3\*1。关于 $R$ 的解释我是参考的wiki [10]，解释得比较清楚； $T$ 是表示平移变换矩阵，为真实世界坐标系原点在相机坐标系中的坐标值，可以理解成真实世界坐标系原点平移到相机坐标系原点的向量。

### 1.1.3 Intrinsic matrix的含义

Intrinsic matrix  $K$  是相机坐标点(3D)到底片坐标点(2D)的映射转换矩阵。即：

$$z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \quad (4)$$

$$K = \begin{bmatrix} f_x & \gamma & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5)$$

我参考wiki [9]看了针孔摄像机原理，参考了这篇CSDN博客 [3]（开始一直不理解 $f_x, f_y$ 的意义）了解了参数的意义：

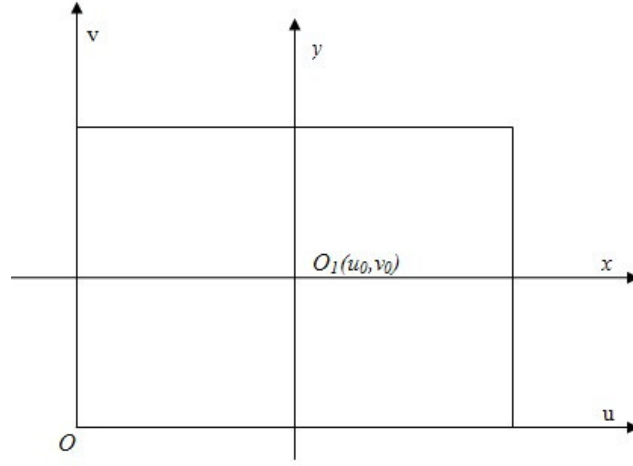


图 3: 底片上的两个坐标系

图(3)是建立在相机底片平面上的两个坐标系。其中原点在左下角的 $(u, v)$ 坐标系是像素坐标系， $u, v$ 单位为像素的宽 $d_x$ 和高 $d_y$ ；原点在中间的 $(x, y)$ 坐标系是成像平面坐标系， $x, y$ 单位为物理意义的长度（毫米之类的），记 $(x, y)$ 坐标系原点在 $(u, v)$ 坐标系中的坐标为 $(u_0, v_0)$ 。

通过 [9]针孔成像原理（相似三角形）可以推出相机坐标系中点 $P(x_c, y_c, z_c)$ 在底片成像平面坐标系中的坐标 $(x_m, y_m)$ ：

$$z_c \begin{bmatrix} x_m \\ y_m \\ 1 \end{bmatrix} = M \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}, \text{ where } M = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (6)$$

同时，可以推出成像平面坐标系中的坐标点 $(x_m, y_m)$ 到像素坐标系点 $(u, v)$ 的公式：

$$z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = Q \begin{bmatrix} x_m \\ y_m \\ 1 \end{bmatrix}, \text{ where } Q = \begin{bmatrix} \frac{1}{d_x} & 0 & u_0 \\ 0 & \frac{1}{d_y} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \quad (7)$$

所以相机内矩阵 $K$ 即为：

$$K = QM = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{d_x} & 0 & u_0 \\ 0 & \frac{1}{d_y} & v_0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{f}{d_x} & 0 & u_0 \\ 0 & \frac{f}{d_y} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \quad (8)$$

## 1.2 第二问：3D到2D（自己推的）

通过section 1.1的原理，显然：

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, [R|T] = (R|t), so : z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ 1 \end{bmatrix}$$

$(u, v)$ 即为 $(X_1, X_2, X_3)$ 在image plane上的投影点。

## 1.3 第三问：2D到3D（自己推的）

给定2D的点 $(u, v)$ ，该点在3D相机矩阵中对应的shape是一条直线。推理如下：设 $(u, v)$ 在3D相机矩阵中对应的点（集）坐标为 $(x, y, z)$ ，相机内矩阵为 $K$ 。则有：

$$Zc(u, v, 1)^T = K(x, y, z)^T$$

因为 $K$ 为3D方阵，且其行列式 $|K| = f_x * f_y * 1 > 0$ , because :  $f_x > 0, f_y > 0$ ，所以 $K$ 的逆矩阵 $K^{-1}$ 存在，则：

$$(x, y, z)^T = zK^{-1}(u, v, 1)^T = z(a, b, c)^T \quad (9)$$

现在1.给定 $u, v$ ，2. $K$ 不随 $u, v$ 变化，为确定值，则上式中 $a, b, c$ 为确定值， $z$ 未知。所以在相机矩阵中，在

$$\frac{x}{a} = \frac{y}{b} = \frac{z}{c} \quad (10)$$

直线上的点都满足(9)式。所以，2D点 $(u, v)$ 对应3D相机中形状 $(x, y, z)$ 的公式为：

$$\frac{x}{a} = \frac{y}{b} = \frac{z}{c}, where : (a, b, c)^T = K^{-1}(u, v, 1) \quad (11)$$

#我没推出来为什么 $a, b, c \neq 0$ ，为0的话有可能对应一个平面

## 1.4 第四问：Distortion，相机畸变

#理解相机的畸变主要参考了这篇CSDN的博客 [3]。畸变（distortion）是对直线投影（rectilinear projection）的一种偏移。相机的畸变简单来说就是真实世界中一条直线投影到图片上不能保持为一条直线了。相机畸变主要包括两种类型：

- 径向畸变

这种畸变来自于透镜形状。对某些透镜，该现象是由于光线在远离透镜中心的地方比靠近中心的地方更

加弯曲，产生“筒形”或“鱼眼”形成的。一般来讲，成像仪中心的径向畸变为0，越向边缘移动，畸变越严重。对于这种畸变，可以通过泰勒展开式矫正：

$$x'' = x'(1 + k_1r^2 + k_2r^4), y'' = y'(1 + k_1r^2 + k_2r^4), \text{where } r = x'^2 + y'^2$$

其中 $(x'', y'')$ 是矫正后的理想（无畸变）坐标， $(x', y')$ 是存在畸变的真实相机坐标系找那个坐标。我对上式的理解是，由于随着 $r$ 增大，该类型畸变越严重，所以上面的矫正方法也是 $r$ 越大，矫正力度越大。

切向畸变 当成像仪被粘贴在摄像机的时候，会存在一定的误差，使得图像平面和透镜不完全平行，从而产生切向畸变。也就是说，如果一个矩形被投影到成像仪上时，可能会变成一个梯形。对于这种畸变，可以以下公式来矫正：

$$x'' = x' + (2p_1y', p_2(r^2 + 2x'^2)), y'' = y' + (2p_1x', p_2(r^2 + 2y'^2)), \text{where } r = x'^2 + y'^2$$

以上公式中的符号定义与径向畸变相同。

同时考虑两种畸变，则纠正畸变的公式为：

$$x'' = x'(1 + k_1r^2 + k_2r^4) + (2p_1y', p_2(r^2 + 2x'^2)) \quad (12)$$

$$y'' = y'(1 + k_1r^2 + k_2r^4) + (2p_1x', p_2(r^2 + 2y'^2)) \quad (13)$$

以上公式中的符号定义与径向畸变相同。

## 1.5 第五问：Calibration相机标定

像机标定主要是求出相机的内、外参数，以及畸变参数。其主要步骤是 [11]：

1. 打印一张棋盘格贴到一个平面上
2. 移动平面或摄像机，从不同方向拍摄多张模板平面的图像
3. 从图像中检测特征点
4. 在不考虑畸变的情况下，估计相机内矩阵的5个参数和每幅图片的外参
5. 用相机内矩阵和外参估计畸变参数
6. 通过极大似然估计优化以上求出的参数

## 1.6 第六问：openCV相机标定

openCV相机标定实现的代码见<https://github.com/zhaone/ProjectStereo/blob/master/cameraBasic/cameraBasicMain.py>。主要的步骤：

1. `getFilelist()`：获得进行图片文件和boardSize

2. `findPoints()`: 用API `findChessboardCorners()`和`cornerSubPix()`获得图片中棋盘格角点坐标, 同时设置角点在真实世界中的坐标 (该坐标系 $z = 0$ 平面在棋盘格平面上)。
3. `cv_calibrate()`: 用API `calibrateCamera()`进行标定, 获得相机内参、畸变系数、旋转向量、位移向量

#实现这个程序主要参考了 [6]的官方API

## 1.7 第七问: openCV去畸变undistort

openCV去畸变实现的代码见<https://github.com/zhaone/ProjectStereo/blob/master/cameraBasic/cameraBasicMain.py> 中的`undistort`函数, 主要采用了openCV的`cv.undistort()`函数。去畸变效果如图(4)所示 (效果还不错)。

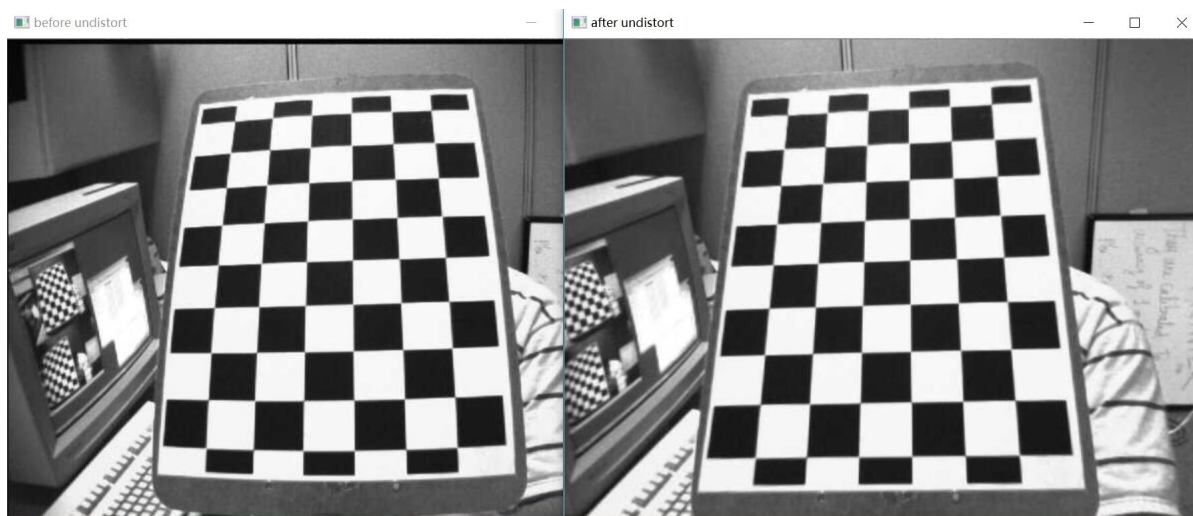


图 4: 去畸变前后效果

## 1.8 第八问: 张友正标定的实现 (自己写的)

我自己实现的张友正标定程序见<https://github.com/zhaone/ProjectStereo/blob/master/cameraBasic/callImpZhang.py>。实现的数学原理参考了 [11], 特别是附录里面的算法。程序主要包括:

1. `homographys()` 计算单应矩阵
2. `getV()` 计算 [11]中Section 3.1中的 $V$ 矩阵
3. `getCameraMatrix()` 计算相机内参矩阵
4. `getRtvecs()` 计算每幅图片的旋转矩阵和位移向量
5. `getDisCoeffs()` 计算畸变系数, 只考虑了 $k_1, k_2$
6. `ramt2rvecs()` 将旋转矩阵转化为旋转向量 (为了和openCV输出一致便于比较)



### 7. zhangCalibrateCamera() 标定总流程

具体的实现方法都是按照paper [11]中的公式实现的，不过最后没有用最小二乘法迭代优化，代码实现的细节可以在github代码注释中看到。我自己实现的张友正标定效果和openCV的API 效果比较见附录A。可以看到，我自己实现的张友正标定得到的相机内参camera Matrix，位移向量tvecs和旋转向量rvecs与openCV的出的结果差别不大，但是畸变系数和openCV 的结果有些差距。我仔细检查了代码的getDisCoeffs()函数和调试的中间结果，计算 $k_1, k_2$ 的代码应该没有错误。导致 $k_1, k_2$  与openCV 计算出的结果差别较大的原因我想可能三个：

1. paper [11]中只考虑了 $k_1, k_2$ ，忽略了 $k_3, p_1, p_2$  等参数
2. 我得出的camera matrix中比openCV的camera matrix多了一个 $\gamma$ 参数，可能会对结果造成影响
3. 没有最后的最小二乘法的优化步骤

其中3的可能性应该比较小，因为我的结果和openCV的结果有数量级的差别。下一步我会排查下原因进行改进，另外最后用最小二乘法对参数进行优化。

#这个写了好长时间

## 1.9 单相机深度估计可行性（自己推的）

给定单个标定的相机和其拍摄的图片无法获得像素点对应的深度。

**推理：**给定标定相机和图片，即已知点图片像素点 $Q(u, v)$ ，相机内参 $K$ ，对应该图片外参 $[R|T]$ 。由式(11) 可确定点 $Q$ 在相机3D坐标系中对应的直线 $l_c$ 。又 $[R|T]$ 已知，若将 $[R|T]$ 写成其次坐标，其维度为 $4 \times 4$ ，取直线 $l_c$ 上任一点 $A(x_l, y_l, z_l, 1)^T$ ，则其对应世界坐标系中一点 $B(x_w, y_w, z_w, 1)^T = ([R|T]^{-1})(x_l, y_l, z_l, 1)^T$ 。由于 $[R|T]$ 是一个旋转位移矩阵，其逆矩阵 $([R|T])^{-1}$  也有相同的性质，所以直线 $l$ 对应与世界坐标系中一条直线 $l_w$ ，所以无法知道 $Q(u, v)$ 在世界坐标系中对应点的位置，其深度也就无法得出。

**解决方案：**两条直线可以相交于一点。可以使用两台标定好的相机对同一场景图片中反映的真实世界同一点 $Q$ 做上述推理变换，得到真实世界 $l_1, l_2$  得到两条直线，解出两直线交点 $P$ 即可。不过我觉得要通过一些条件限制保证两条 $l_1, l_2$ 在同一平面有交点。

## 2 Binocular Basics

### 2.1 第九问：Projection

首先对左侧摄像机，由于世界坐标系与左侧相机坐标系对齐，则点 $X$ 在左侧摄像机中坐标是就是 $(X_1, X_2, X_3)$ 。所以点 $X$  在左侧相机底片坐标 $x_l(u_l, v_l, 1)$ ：

$$X_3(u_l, v_l, 1)^T = M_l(X_1, X_2, X_3)^T$$

也即是：

$$u_l = u_0 + \frac{f_x * X_1}{X_3}, v_l = v_0 + \frac{f_y * X_2}{X_3}$$

对于右侧摄像机，点 $\mathbf{X}$ 在左侧摄像机中坐标是 $(R|t)(X_1, X_2, X_3)^T$ ，所以点 $X$ 在左侧相机底片坐标 $x_r(u_r, v_r, 1)$ ：

$$z(u_r, v_r, 1)^T = M_r(R|t)(X_1, X_2, X_3, 1)^T$$

## 2.2 第十问：Epipolar Line

两台相机模型如下图(5)所示 [2] [7]：

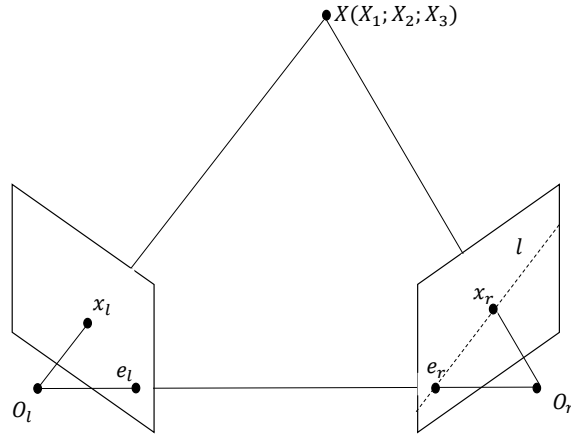


图 5: 对极坐标

上图中点 $X$ 是点 $x_l$ 对应世界坐标系直线上一点， $x_r$ 是点 $X$ 在右侧相机像素坐标系中的投影点位置，点 $e_l, e_r$ 是两相机光心 $O_l, O_r$ 连线与两相机像素坐标系平面的交点，记做左右极点。设极线 $l$ ：

$$Ax + By + C = 0, \quad \text{equal to : } (x, y, 1) \begin{bmatrix} A \\ B \\ C \end{bmatrix} = 0 \quad (14)$$

可以看到 $l$ 是平面 $(X, O_1, O_2)$ 与右侧相机底板平面交线，由于 $e_r$ 同时处与两个平面上，所以 $e_r$ 在极线 $l$ 上。

由上图可以看出 $e_r$ 可以看做左侧光心 $O_1(0, 0, 0, 1)$ （世界坐标系齐次坐标）在右侧像素坐标系的投影点，所以 $e_r : (u_e, v_e, 1)$ 可以由以下公式得到：

$$z(u_e, v_e, 1)^T = M_r(R|t)(0, 0, 0, 1)^T = M_r t \quad (15)$$

对于点 $X(X_1, X_2, X_3)$ ,  $x_l(u_l, v_l, 1)$ ,  $x_r(u_r, v_r, 1)$ 有:

$$\begin{aligned} z_1(u_l, v_l, 1)^T &= M_l(X_1, X_2, X_3) \\ z_2(X_1, X_2, X_3) &= M_l^{-1}(u_l, v_l, 1) \\ z_3(u_r, v_r, 1)^T &= M_r(R|t)(X_1, X_2, X_3, 1)^T \\ z_4(u_r, v_r, 1)^T &= M_r(RM_l^{-1}(u_l, v_l, 1)^T + t) \end{aligned}$$

由于点 $e_r, x_r$ 都在交线 $l$ 上, 在右侧相机坐标系中计算向量 $V = e_r \times x_r$  ( $\times$ 是外积的意思):

$$\begin{aligned} V &= e_r \times x_r \\ &= M_r t \times M_r(RM_l^{-1}(u_l, v_l, 1)^T + t) \\ &= M_r t \times (M_r RM_l^{-1}(u_l, v_l, 1)^T + M_r t) \\ &= M_r t \times M_r RM_l^{-1}(u_l, v_l, 1)^T \end{aligned}$$

则由外积的性质可知交线 $l$ 上点 $(x, y, z)V = 0$ 。所以式子(14)中 $(A, B, C) = V$ 。综上所述, 极线的方式如下:

$$\begin{aligned} \text{equation : } \quad Ax + By + C &= 0 \\ \text{where : } (A, B, C)^T &= M_r t \times M_r RM_l^{-1}(u_l, v_l, 1)^T \end{aligned} \tag{16}$$

### 2.3 第十一问: Fundamental Matrix

由式(16)可知:

$$(u_r, v_r, 1)(M_r t \times M_r RM_l^{-1}(u_l, v_l, 1)^T) = 0 \tag{17}$$

记 $[t]_\times$  = 为外积的矩阵形式, 即 [7]:

$$[t]_\times = \begin{bmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{bmatrix},$$

则有 (此处详细推导过程和外积的性质见 [1]):

$$(u_r, v_r, 1)(M_r^{-T}[t]_\times RM_l^{-1})(u_l, v_l, 1)^T = 0$$

所以

$$F = M_r^{-T}[t]_{\times} R M_l^{-1} \quad (18)$$

$$\text{makes } x_r^T F x_l = 0 \quad (19)$$

该式的 $F$ 也对应于式子(16)中的 $M_r t \times M_r R M_l^{-1}$ , 即直线参数 $(A, B, C) = M_r^{-T}[t]_{\times} R M_l^{-1} x_l$ 。

#第十、十一问主要参考了 [1], [7] 和 [2], 其中 [1]对基础矩阵和标定的理解帮助特别大。

## 2.4 第十二问: Stereo Calibration

主要参考了opencv `stereoCalibrate()` 函数 [6]。实现代码见<https://github.com/zhaone/ProjectStereo/blob/master/binocularBasics/stereoCalibration.py>。通过立体标定可以获得左侧相机坐标到右侧相机坐标的旋转矩阵 $R$ , 位移矩阵 $T$ , 以及本质矩阵 $E$ 和基础矩阵 $F$ , 这份代码里有opencv API实现和自己实现的计算 $R, T, F$ 的方法, 结果见附录B, 可以看到两者相差不大。

## 2.5 第十三问: 3D坐标点 (自己推的)

由式(19)可知, 极线方程为 $Ax + By + C = 0$ , where :  $(A, B, C)^T = F p_l$ 。

假设选定的 $p_r$ 坐标为 $(u_r, v_r, 1)$ ,  $p_l$ 和 $p_r$ 在世界坐标系中对应的点为 $P_l(l_1, l_2, l_3), P_r(r_1, r_2, r_3)$ 。则有以下公式成立:

$$P_l(l_1, l_2, l_3)^T = M_l^{-1} p_l \quad (20)$$

$$p_r = M_r(RP_r + t) \quad \text{equal to : } P_r(r_1, r_2, r_3)^T = R^{-1}(M_r^{-1} p_r - t) \quad (21)$$

由左右光心 $O_l, O_r$ 在世界坐标系中坐标分别为 $(0, 0, 0)^T, t$ 。则3D坐标为直线 $O_l, P_l$ 和 $O_r, P_r$ 的角点。即:

$$\frac{x}{l_1} = \frac{y}{l_2} = \frac{z}{l_3} \quad (22)$$

$$\frac{x - t_1}{r_1 - t_1} = \frac{x - t_2}{r_2 - t_2} = \frac{x - t_3}{r_3 - t_3} \quad (23)$$

直线(22)(23)的交点即为3D点的坐标。

## 2.6 第十四问: Rectification

主要参考了opencv的`stereoRectify(), initUndistortRectifyMap(), remap()` 函数 [6]。实现代码见<https://github.com/zhaone/ProjectStereo/blob/master/binocularBasics/rectification.py>。主要流程如下:

1. `stereoRectify()`函数通过Stereo Calibration获得的 $R, T$ 和相机内矩阵和畸变参数计算左右相机对齐需要的旋转矩阵 $R_l, R_r$ 和新的投影矩阵 $P_l, P_r$ 。
2. `initUndistortRectifyMap()`函数对每个相机通过相机内矩阵，畸变参数和该相机的旋转矩阵，投影矩阵计算像素转换的map
3. `remap()`函数通过上面计算的map对图片进行变换，得到对齐的左右图片

一个校正的结果如下图(6)所示：



图 6: 双相机图片校正结果

## 2.7 第十五问：Baseline（自己推的）

根据opencv [6]document的介绍，通过`stereoRectify()`可以获得左右相机旋转矩阵 $R_1, R_2$ 。设有世界坐标系点 $P$ 。则其在旋转后左相机坐标系中坐标为 $R_1 P$ ，在旋转后右相机坐标系中坐标为 $R_2(RP + T)$ ，又有校正后左相机坐标系到右相机坐标系的转换矩阵为 $(I|t)$ ，则：

$$R_2(RP + T) = (I|t)(R_1 P) \quad (24)$$

$$\text{equals to: } R_2 R P + R_2 T = R_1 P + t \quad (25)$$

$$\text{so: } R_2 T = t = (b, 0, 0)^T \quad (26)$$

记

$$R_2 = \begin{pmatrix} r_1^T \\ r_2^T \\ r_3^T \end{pmatrix}$$

则

$$b = r_1^T T \quad (27)$$

## 2.8 第十六问: Depth-Disparity (自己推的)

首先要理解问题里面的长度, 其中 $b$ 是一个物理长度,  $x_l, x_r, d$ 都是像素长度, 要通过像素单位长度进行转化, 求出的点坐标也是物理长度的坐标。对齐后投影模型如图(7) 所示:

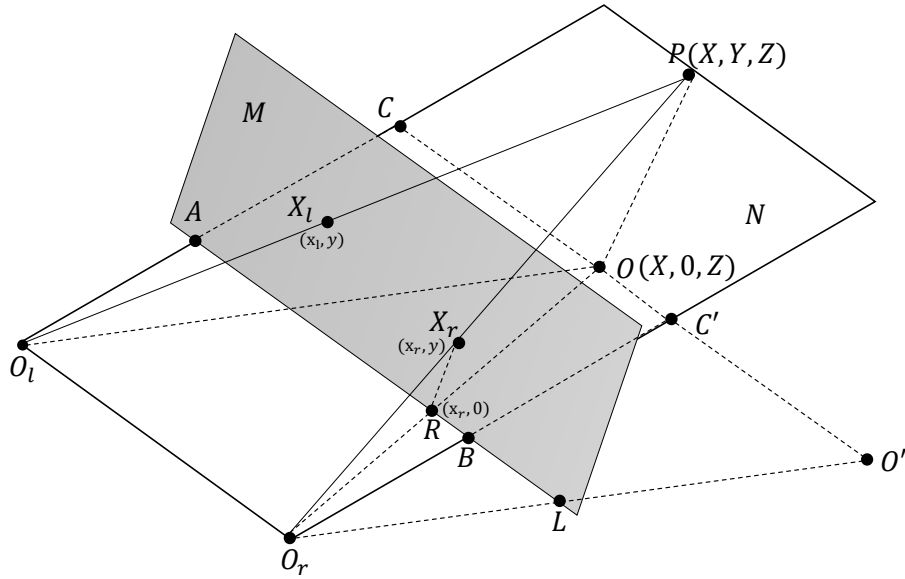


图 7: Depth Disparity

其中 $O_l, O_r$ 为左右相机光心,  $A, B$ 为两光轴与相机底板平面 $M$ 交点。平面 $O_l O_r AB$  记为平面 $N$ 。点 $P(X, Y, Z)$ 为现实世界成像点, 其在平面 $N$ 上的点投影点为 $O(X, 0, Z)$ , 在左右相机底片上投影坐标记为 $X_l(x_l, y), X_r(x_r, y)$ 。点 $B$  为点 $X_r$  在平面 $N$  上的投影点。直线 $CC'$ 为平面 $N$ 上过点 $O$ 与光轴垂直的直线,  $CC'$  在左右相机光轴上。三角形 $O_r C' O'$ 由三角形 $O_l C O$ 延 $CC'$ 方向平移得到,  $L$ 为 $AB$ 与 $O_r O'$ 的交点。相机内矩阵已知, 记为 $K$ :

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

#这里没有给两个相机像素宽高是否相同，因为只有相同才能往下计算，所以我默认相同。

记相机焦距为 $f$ ，则相机单像素宽度高度分别为 $d_x = \frac{f}{f_x}, d_y = \frac{f}{f_y}$  显然有（下面的线段长度都是物理长度）：

$$BR = dd_r$$

$$BL = x_l d_x$$

$$BL = (x_l - x_r) d_x = dd_x$$

$$OO' = O_l O_r = b$$

$$BX_r = y d_y$$

$$BO_r = f$$

$$C' O' = X$$

$$OP = Y$$

$$O_r C' = Z$$

由三角形相似，可以得到：

$$\begin{aligned} \frac{Z}{f} &= \frac{O_r C'}{BO_r} = \frac{OO'}{LR} = \frac{b}{dd_x} \\ \frac{X}{x_l d_x} &= \frac{O' C'}{BL} = \frac{b}{dd_x} \\ \frac{Y}{y d_y} &= \frac{OP}{RX_r} = \frac{OO_r}{RO_r} = \frac{b}{dd_x} \end{aligned}$$

所以：

$$X = \frac{bx_l d_x}{dd_x} = \frac{bx_l}{d} \quad (28)$$

$$Y = \frac{by d_y}{dd_x} = \frac{y b f_x}{d f_y} \quad (29)$$

$$Z = \frac{b f}{dd_x} = \frac{b f_x}{d} \quad (30)$$

3D点坐标为 $P(\frac{bx_l}{d}, \frac{y b f_x}{d f_y}, \frac{b f_x}{d})^T$ ，即 $\frac{b}{d}(x_l, \frac{y f_x}{d f_y}, f_x)$ 。

关于 $d_x = d_y$ 时 $z$ 能否写成 $\frac{b f}{d}$ ，从我推出的结果看应该不能 $z = \frac{b f_x}{d}$ ，当 $f_x$ 不和 $f$ 相等时就不能写成 $\frac{b f}{d}$ ，不过 $Y$ 可以化简为 $\frac{y b}{d}$ 。

### 3 Stereo Matching

待补充

## 参考文献

- [1] Johnny Chien. Beginner's guide to fundamental matrix, essential matrix and camera motion recovery. [https://www.researchgate.net/publication/303522230\\_qiantanjichujuzhenbenzhijuzhenyuxiangjiyidong\\_Beginner's\\_Guide\\_to\\_Fundamental\\_Matrix\\_Essential\\_Matrix\\_and\\_Camera\\_Motion\\_Recovery](https://www.researchgate.net/publication/303522230_qiantanjichujuzhenbenzhijuzhenyuxiangjiyidong_Beginner's_Guide_to_Fundamental_Matrix_Essential_Matrix_and_Camera_Motion_Recovery).
- [2] 小旭同学A0000 CSDN. 立体视觉. <https://blog.csdn.net/xiao4399/article/details/48037287>.
- [3] Jessica&jie CSDN. 相机标定原理介绍（一）. <https://www.cnblogs.com/Jessica-jie/p/6596450.html>.
- [4] cs.toronto.edu. Camera models and parameters. <http://ftp.cs.toronto.edu/pub/psala/VM/camera-parameters.pdf>.
- [5] MathWorks. What is camera calibration. <https://www.mathworks.com/help/vision/ug/camera-calibration.html>.
- [6] openCV. Camera calibration and 3d reconstruction. [https://docs.opencv.org/2.4/modules/calib3d/doc/camera\\_calibration\\_and\\_3d\\_reconstruction.html](https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html).
- [7] Wikipedia. 基础矩阵. <https://zh.wikipedia.org/wiki/%E5%9F%BA%E7%A1%80%E7%9F%A9%E9%98%B5>.
- [8] Wikipedia. Camera resectioning. [https://en.wikipedia.org/wiki/Camera\\_resectioning](https://en.wikipedia.org/wiki/Camera_resectioning).
- [9] Wikipedia. Pinhole camera model. [https://en.wikipedia.org/wiki/Pinhole\\_camera\\_model](https://en.wikipedia.org/wiki/Pinhole_camera_model).
- [10] Wikipedia. Rotation matrix. [https://en.wikipedia.org/wiki/Rotation\\_matrix](https://en.wikipedia.org/wiki/Rotation_matrix).
- [11] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22, 2000.

## A 附录：相机标定openCV与自己实现结果比较

表 1: openCV和自己实现的方法畸变系数比较

|                | $k_1$      | $k_2$      | $k_3$     | $p_1$      | $p_2$     |
|----------------|------------|------------|-----------|------------|-----------|
| openCV         | -0.2808809 | 0.02517166 | 0.0012166 | -0.0001355 | 0.1634489 |
| Slef Implement | 0.0020839  | -2.667E-05 | 0         | 0          | 0         |

#表2中采用是left文件夹中的图片，*tvecs*表示位移向量，*rvecs*表示旋转向量，每行表示一张图片的向量值，左侧3个为openCV获得，右侧三个为自己实现的方法获得。



表 2: openCV 和自己实现的方法结果比较

|               | OpenCV    |           |           | Self Implement |           |            |
|---------------|-----------|-----------|-----------|----------------|-----------|------------|
| camera matrix | 532.827   | 0         | 342.486   | 543.50163      | -4.237409 | 357.191875 |
|               | 0         | 532.945   | 233.855   | 0              | 545.78158 | 235.08919  |
|               | 0         | 0         | 1         | 0              | 0         | 1          |
| tvecs         | -3.91978  | 1.5759185 | 17.15951  | -4.409501      | 1.5567137 | 17.4500936 |
|               | 3.4192304 | 5.2865887 | 14.22006  | 3.0608411      | 5.222831  | 14.8762691 |
|               | -4.718518 | 1.108129  | 11.47173  | -4.960351      | 1.0878467 | 11.4622953 |
|               | -4.98153  | 3.3264306 | 12.71391  | -5.287045      | 3.2865817 | 13.1606797 |
|               | -3.686437 | -4.020144 | 13.28904  | -4.08676       | -4.020437 | 13.6411581 |
|               | 1.3962968 | -4.271462 | 15.85035  | 0.9631481      | -4.495755 | 16.7796074 |
|               | -4.315564 | -5.489637 | 17.48175  | -4.640167      | -5.240114 | 17.3780545 |
|               | -2.301478 | -5.350052 | 14.52772  | -2.732646      | -5.294209 | 14.8935799 |
|               | -4.575739 | 2.5409072 | 11.62056  | -4.904191      | 2.5232139 | 12.0795987 |
|               | -3.138365 | -4.262156 | 10.01088  | -3.481912      | -4.275332 | 10.559435  |
|               | -3.966455 | -4.802239 | 13.57548  | -4.372979      | -4.768601 | 14.0972409 |
|               | -4.666377 | -2.953178 | 12.22287  | -5.014534      | -2.949935 | 12.6544459 |
|               | -3.721262 | -3.893277 | 9.904011  | -4.082615      | -3.926067 | 10.541913  |
| rvecs         | -0.084627 | 0.3453051 | -1.543025 | -0.067938      | 0.2583436 | -1.5570806 |
|               | -0.22738  | 1.0244723 | -2.792092 | -2.978611      | -0.709744 | -0.7026674 |
|               | -0.349252 | -0.069653 | -1.200653 | 0.6233055      | -0.384735 | -2.0363102 |
|               | -0.276421 | 0.0970042 | -1.563284 | -0.173886      | -0.503776 | -2.987277  |
|               | -0.476413 | 0.0885465 | -0.225949 | 0.1318448      | -0.482891 | -1.5052072 |
|               | 0.0621603 | 0.4465449 | 0.107134  | 0.2589092      | -0.170135 | 1.54422695 |
|               | -0.104693 | 0.3177094 | 0.314246  | 0.4640564      | -0.162215 | 1.52179571 |
|               | -0.356238 | 0.2402072 | 0.209317  | -0.475752      | 0.044344  | 0.2511271  |
|               | 0.4812064 | -0.173949 | -1.406844 | 0.5352236      | 0.7207306 | -2.3915298 |
|               | 0.0494733 | -0.60134  | -0.183568 | 0.5986967      | -0.088322 | -1.4785658 |
|               | -0.377069 | 0.0683159 | -0.019444 | -0.339023      | 0.1098992 | -0.0330621 |
|               | 0.4974412 | 0.1197422 | -0.296026 | 0.6618053      | 0.4031866 | -0.2198149 |
|               | 0.1933082 | -0.420165 | -0.19427  | 0.3631638      | -0.250878 | -0.1428785 |

B 附录：立体标定 *openCV* 结果与自己实现结果

表 3: 旋转矩阵, 位移矩阵和基础矩阵

|   | Opencv     |          |              | Slef Implement |            |           |
|---|------------|----------|--------------|----------------|------------|-----------|
|   | 0.99998115 | 0.003936 | 0.003885037  | 0.9999852      | 0.00412905 | 0.0035307 |
| R | -0.0039357 | 0.999989 | 0.000201332  | -0.004128      | 0.99999144 | -0.000278 |
|   | -0.0038852 | -0.00022 | 0.99998652   | -0.003532      | 0.00026353 | 0.9999937 |
| T | 3.34674023 | 0.032175 | 0.05392305   | 3.3442471      | 0.04172117 | 0.0529604 |
|   | -4.547E-09 | 2.81E-06 | -0.001559624 | -3.84E-09      | 2.8554E-06 | -0.001878 |
| F | -2.136E-06 | 2.69E-08 | -0.092939862 | -2.22E-06      | -5.94E-08  | -0.096032 |
|   | 0.00106631 | 0.093711 | 1            | 0.0013663      | 0.09689419 | 1         |