STEREO

关于深度估计的学习报告

上海大学 王聪豪

深度估计的学习报告

深度估计是指估计场景中对象到观察者的距离。有多种方法可以完成任务。一些流行的方法，包括飞行时间（time-of-flight）设备，结构光相机（structured light cameras）和多视图几何（multiple-view geometry），其中多视图几何结构的方法成本最低。在多视图几何方法中，我们关心如何使用普通相机来估计场景深度。我将逐步解决 Project_Stereo 中的各个问题，设计程序来使用相机估算深度。

有一些问题我可能会合并到一节的内容中去，一些重要的点或是没理解的部分会这样强调，以便日后查看。本项目相关代码和实验结果开源在 https://github.com/BindyAtobe/stereo，本人水平有限，如有任何错误或您对我有任何建议，非常希望您能联系我。

上海大学 王聪豪

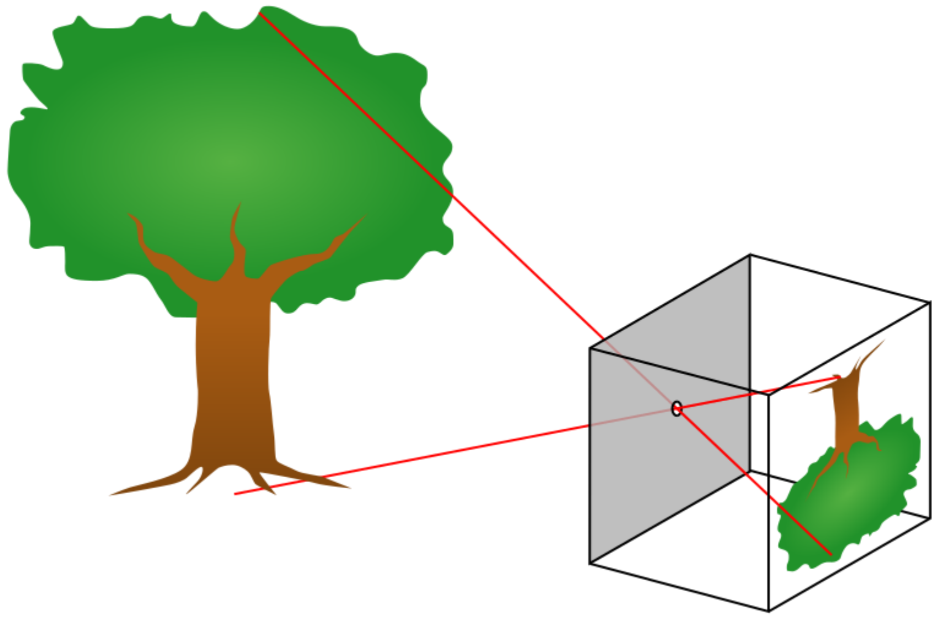
邮箱：919818192@qq.com

目录

1. 相机基础

1.1. 相机矩阵，内参、外参（第 1，2 题）

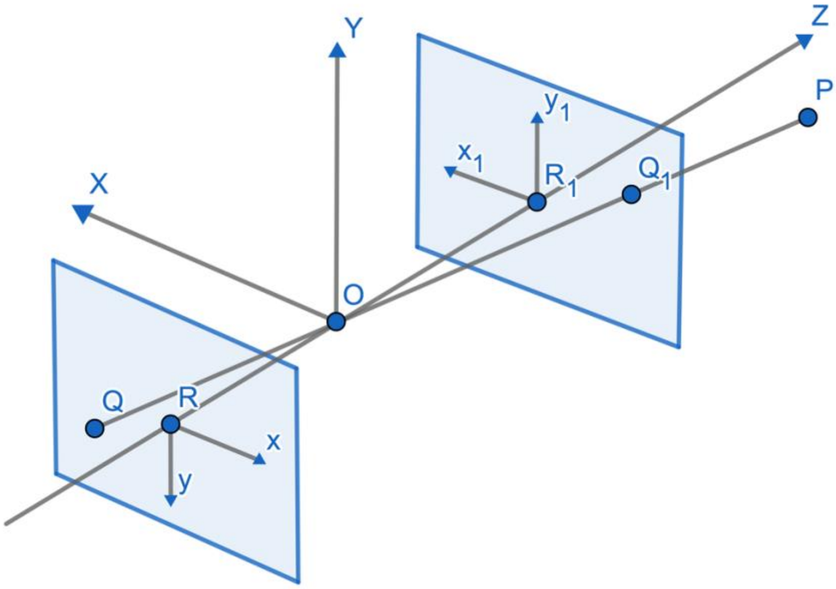
假定我们的相机符合针孔相机模型[1]，那么相机成像大致如下图所示，



我们可以发现物体和投影有倒立相似的关系。

[1] 维基百科-Pinhole camera model，https://en.wikipedia.org/wiki/Pinhole_camera_model

加上 3D 和 2D 坐标系来描述相机成像，其中 $OR=OR_1=f$（焦距），维基百科中的坐标系是左手的，且和第 2 题描述不同，所以我直接按第 2 题描述的方向来画坐标系，



由于倒像，我们往往会 180°旋转位于-f 位置的图像 2D 坐标系，或者将图像移至 f 位置处。

成像的过程其实就是 3D 坐标到 2D 坐标的投影（projection），设图中 $P(X_c,Y_c,Z_c)$，$Q(x,y)$，由相似易得，

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{f}{Z_c} \begin{pmatrix} X_c \\ Y_c \end{pmatrix}$$

写成齐次形式（能用统一形式表示旋转和平移等好处，参考[2]），

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \frac{f}{Z_c} \begin{pmatrix} X_c \\ Y_c \\ \frac{Z_c}{f} \end{pmatrix}$$

[2] 简书-齐次坐标系入门级思考，https://www.jianshu.com/p/80d0018ed24c

用矩阵形式来写，

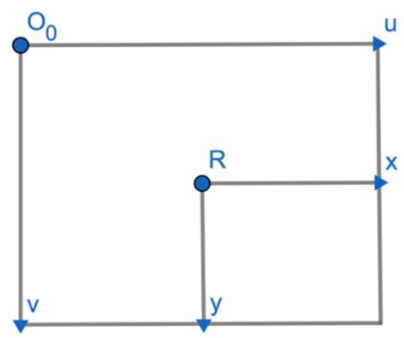
$$Z_c\begin{pmatrix}x\\y\\1\end{pmatrix}=\begin{pmatrix}f&0&0&0\\0&f&0&0\\0&0&1&0\end{pmatrix}\begin{pmatrix}X_c\\Y_c\\Z_c\\1\end{pmatrix}\qquad(1)$$

上述过程只是完成了点 P 从相机坐标到图像坐标的映射。实际上我们在观察点 P 的时候基于一个世界坐标系，它不一定会将原点设在针孔 O，方向也不一定与相机坐标系相同。观察图像时，往往基于像素坐标系，而不是图中的图像坐标系。建立从世界坐标到像素坐标映射关系的矩阵叫相机矩阵（camera matrix）。几个坐标系可以参考[3]，完整的坐标转换如下所示，

世界坐标→相机坐标→图像坐标→像素坐标

所以我们还需要知道如何从世界坐标映射到相机坐标，以及如何从图像坐标映射到像素坐标。

先讲图像坐标映射到像素坐标，下图中 $R(u_0,v_0)$，坐标系 uO_0v 单位是像素，坐标系 xRy 单位是 mm，



有 $$\begin{pmatrix}u\\v\\1\end{pmatrix}=\begin{pmatrix}\frac{1}{dx}&\gamma&u_0\\0&\frac{1}{dy}&v_0\\0&0&1\end{pmatrix}\begin{pmatrix}x\\y\\1\end{pmatrix}\qquad(2)$$

[3] 百度百科-像素坐标，https://baike.baidu.com/item/像素坐标/5372225

其中，dx 和 dy 分别表示 x 和 y 方向上一个像素的宽度，其倒数表示单位长度有多少像素（mm/像素），γ是 x 和 y 不垂直时的倾斜系数，通常为 0。

由(1)(2)可得，

$$Z_c\begin{pmatrix}u\\v\\1\end{pmatrix}=\begin{pmatrix}\frac{f}{dx} & 0 & u_0 & 0\\ 0 & \frac{f}{dy} & v_0 & 0\\ 0 & 0 & 1 & 0\end{pmatrix}\begin{pmatrix}X_c\\Y_c\\Z_c\\1\end{pmatrix}$$

将其写作，

$$Z_c\begin{pmatrix}u\\v\\1\end{pmatrix}=\begin{pmatrix}f_x & 0 & c_x & 0\\ 0 & f_y & c_y & 0\\ 0 & 0 & 1 & 0\end{pmatrix}\begin{pmatrix}X_c\\Y_c\\Z_c\\1\end{pmatrix} \quad (3)$$

其中 f_x，f_y 分别为 x，y 方向上焦距长度（像素），$\begin{pmatrix}f_x & 0 & c_x & 0\\ 0 & f_y & c_y & 0\\ 0 & 0 & 1 & 0\end{pmatrix}$被称为内参矩阵（intrinsics）。

世界坐标系可以通过刚体变换转换到相机坐标系，所以世界坐标转换到相机坐标可经由旋转和平移变换，将这两个变换复合，

$$\begin{pmatrix}\boldsymbol{I} & \boldsymbol{t}\\ \boldsymbol{0} & 1\end{pmatrix}\begin{pmatrix}\boldsymbol{R} & \boldsymbol{0}\\ \boldsymbol{0} & 1\end{pmatrix}=\begin{pmatrix}\boldsymbol{R} & \boldsymbol{t}\\ \boldsymbol{0} & 1\end{pmatrix}$$

$\begin{pmatrix}\boldsymbol{R} & \boldsymbol{t}\\ \boldsymbol{0} & 1\end{pmatrix}$被称为外参矩阵（extrinsics），那么有，

$$\begin{pmatrix}X_c\\Y_c\\Z_c\\1\end{pmatrix}=\begin{pmatrix}\boldsymbol{R} & \boldsymbol{t}\\ \boldsymbol{0} & 1\end{pmatrix}\begin{pmatrix}X_w\\Y_w\\Z_w\\1\end{pmatrix} \quad (4)$$

由(3)(4)可得，

$$Z_c\begin{pmatrix}u\\v\\1\end{pmatrix}=\begin{pmatrix}f_x & 0 & c_x & 0\\ 0 & f_y & c_y & 0\\ 0 & 0 & 1 & 0\end{pmatrix}\begin{pmatrix}\boldsymbol{R} & \boldsymbol{t}\\ \boldsymbol{0} & 1\end{pmatrix}\begin{pmatrix}X_w\\Y_w\\Z_w\\1\end{pmatrix}$$

可以整理一下，

$$Z_c\begin{pmatrix}u\\v\\1\end{pmatrix}=\begin{pmatrix}f_x&0&c_x\\0&f_y&c_y\\0&0&1\end{pmatrix}\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\end{pmatrix}\begin{pmatrix}\boldsymbol{R}&\boldsymbol{t}\\\boldsymbol{0}&1\end{pmatrix}\begin{pmatrix}X_w\\Y_w\\Z_w\\1\end{pmatrix}$$

$$Z_c\begin{pmatrix}u\\v\\1\end{pmatrix}=\begin{pmatrix}f_x&0&c_x\\0&f_y&c_y\\0&0&1\end{pmatrix}\begin{pmatrix}\boldsymbol{R}&\boldsymbol{t}\end{pmatrix}\begin{pmatrix}X_w\\Y_w\\Z_w\\1\end{pmatrix}\qquad(5)$$

第 2 题答案应该就是(5)式。

这一节的推导过程主要参考了博客[4]和维基百科[5]。因为网上很多资料坐标系方向、坐标符号不同，所以自己画了坐标系，重打了公式，表述更加清晰一点。对于齐次坐标，和内参矩阵中的γ系数还没理解透彻，在这记录下。

1.2. 图像上一点对应的 3D 形状（第 3 题）

给定一个 2D 点 Q (u,v)，求它在相机坐标系中对应的形状（第 3 题）。

从 1.1 的图就能看出是一条由 OQ 决定的直线。1.1 中的式子前面总含有 Z_c，我们可以把它看做是一个任意的比例因子。因为空间到图像的映射是多对 1 的，联系实际，远处一个大的物体和近处一个小的物体在图像上投影确实有可能一样，这个比例因子就反映这种缩放关系。

不妨使(3)式的各分量相等，

$$\begin{cases}Z_cu=f_xX_c+c_xZ_c\\Z_cv=f_yY_c+c_yZ_c\end{cases},Z_c>0$$

[4] CSDN-深入解读相机矩阵，https://blog.csdn.net/lingchen2348/article/details/83052214

[5] 维基百科-Camera_matrix，https://en.wikipedia.org/wiki/Camera_matrix

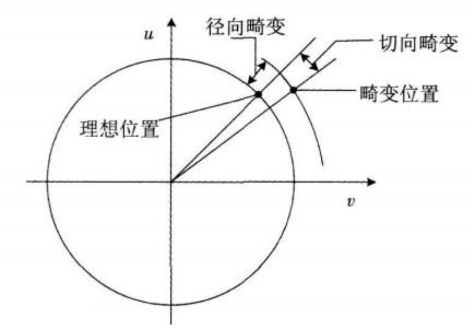
$$
\Rightarrow \begin{cases} X_c = \dfrac{Z_c}{f_x}(u - c_x) \\ Y_c = \dfrac{Z_c}{f_y}(v - c_y) \\ Z_c = Z_c \end{cases}, Z_c > 0
$$

显然这是一条端点为点 O 的射线。引入齐次坐标，并写成矩阵形式，

$$
\begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} = \begin{pmatrix} \dfrac{Z_c}{f_x} & 0 & \dfrac{-Z_c c_x}{f_x} \\ 0 & \dfrac{Z_c}{f_y} & \dfrac{-Z_c c_y}{f_y} \\ 0 & 0 & Z_c \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}, Z_c > 0
$$

1.3. 图像畸变（第 4 题）

受镜头制造精度的影响，图像会出现不同程度的畸变（distortion），这种畸变可以分为径向畸变和切向畸变两种。下图来自百度百科[6]，



径向畸变的影响远大于切向畸变，OpenCV 文档显示了其两种常见类型，

6 百度百科-切向畸变，https://baike.baidu.com/item/切向畸变/4947159?fr=aladdin

看了这篇博客[7]才更好地理解了畸变，其实就是正确的像素值产生了偏移，本属于点(u,v)的像素值，由于畸变，跑到了点(u_d,v_d)上，所以为了校正畸变，需要找到点的映射关系，将点(u_d,v_d)的像素值赋给点(u,v)。

以下参考 OpenCV 文档[8]，1.1 中的转换等效于以下公式，

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = \boldsymbol{R} \begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix} + \boldsymbol{t}$$

$$x' = X_c / Z_c$$

$$y' = Y_c / Z_c$$

$$u = f_x x' + c_x$$

$$v = f_y y' + c_y$$

存在畸变时（只考虑 k_1,k_2,p_1,p_2，k_1,k_2 是径向畸变系数，p_1,p_2 是切向畸变系数），

$$x'' = (1 + k_1 r^2 + k_2 r^4) x' + 2 p_1 x' y' + p_2 (r^2 + 2 x'^2)$$

$$y'' = (1 + k_1 r^2 + k_2 r^4) y' + p_1 (r^2 + 2 y'^2) + 2 p_2 x' y'$$

其中，$r^2 = x'^2 + y'^2$

$$u_d = f_x x'' + c_x$$

$$v_d = f_y y'' + c_y$$

给定一个点(u,v)，可以算出 x′和 y′，代入上式可以得到点(u_d,v_d)。该点像素值是畸变前本应属于点(u,v)的，重新赋给它，就达到了校正的目的。这篇

7 CSDN-OpenCV 学习(5): 图像畸变校正，
https://blog.csdn.net/xholes/article/details/80599802

8 OpenCV-Camera Calibration and 3D Reconstruction，
https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

博客[9]提到畸变后的坐标 u_d 和 v_d 往往不是整数，所以需要利用相邻的四个像素点，通过双线性插值来得到点(u_d,v_d)的像素值，这种方法比较简单，网上资料讲的也很清楚，这里不再具体展开。

1.4. 相机标定，畸变校正（第 5，6，7 题）

需要安装 OpenCV，这篇博客[10]讲了 macOS 下的安装过程以及会碰到的问题，cmake 不一定要像他一样用 GUI，用命令也很方便。再记录一个知乎上看到的 tip：从 github 上 git clone 速度很慢，可以先 fork 到自己的 github 仓库，然后导入到国内的码云仓库，从码云上 git clone 的速度超级快，亲测有效。

在网上找资料的过程中，发现官方提供了标定和畸变校正的 sample，所以直接拿来学习，一开始看不懂，放到 CLion 中 debug 单步跟一遍，一边查一下函数，就差不多了。所有函数都能在官方文档[11]检索到，这篇报告就不再对函数的使用展开说明。

相机标定就是根据几组 3D 世界坐标-2D 像素坐标的点对，来求出相机矩阵。我们经常利用黑白相间的棋盘来标定，因为标准棋盘的制作成本低，将世界坐标系的原点设在棋盘的左上角，容易得到所有角点的世界坐标；至于各角点的像素坐标可以调用 OpenCV 的函数 findChessboardCorners 来得到。

再调用 calibrateCameraRO 就可以算出内参，外参和畸变参数，可能与旧版本的 OpencCV 提供的接口不同，但功能一样。

9 CSDN-[图像]畸变校正详解，https://blog.csdn.net/humanking7/article/details/45037239

10 简书-OpenCV macOS：编译安装 OpenCV4+Opencv Contrib，https://www.jianshu.com/p/162f2cdf4f88

11 OpenCV 官方文档，https://docs.opencv.org/master/

原本的 sample 的使用[12]还是有些麻烦，需要事先生成图像列表，查到有个 glob 函数可以读取一个文件夹下的所有图片路径，利用它可以使程序的使用更简单。原本的 sample 还考虑了影像，我也对此做了简化。

关于畸变校正，相机标定后我们已经得到了内参矩阵（OpenCV 的示例和源码中都将其命名为 cameraMatrix，但我认为相机矩阵应该是内参矩阵和外参矩阵的乘积）和畸变参数。可以利用 undistort 函数，一步到位得到校正好的图像；也可以先利用 getOptimalNewCameraMatrix 函数得到考虑了畸变的参数矩阵，再用 initUndistortRectifyMap 函数得到畸变前后点的映射关系，最后用 remap 得到校正好的图像。

1.5. 张正友标定（第 8 题）

张正友标定的原文[13]写的很详细，有了之前的基础并不难看懂，这里不把公式一一列举了，大致理一下实现思路：

1. *一张图（至少 4 个角点）可计算出一个单应性矩阵 H（8 自由度）*
2. *每个 H 可得到相应的 v，并组成 V*
3. *用 Vb=0 解出 b（没有其他条件的话，至少 3 张图才有非 0 解）*
4. *由 b 算出各内参，即算出了论文中的矩阵 A*
5. *由 A 和之前每张图的 H 算出每张图对应的外参*
6. *用 Levenberg-Marquardt 算法（以下简称 LM 算法）优化参数*

12 百度经验-OpenCV：相机标定示例程序的使用，https://jingyan.baidu.com/article/7f41ecec5877eb593d095ce9.html

13 Z. Zhang, “A flexible new technique for camera calibration,” IEEE Trans- actions on pattern analysis and machine intelligence, vol. 22, no. 11, pp. 1330–1334, 2000.

畸变参数不会很大，不妨就把初值设为 0，和其他参数一起用 LM 算法优化，切向畸变影响比较小，通常不需要考虑。论文中几何解释的部分没怎么看懂，虽不影响实现，但我认为是重要的。

做之前还是先看看 OpenCV 相关函数的源码，用 CLion，发现 debug 不进去，查了资料知道自己之前安装了 release 版本，又重装了一遍 OpenCV，cmake 命令还不是很熟悉，在这记录下：

- *cmake -D CMAKE_BUILD_TYPE=DEBUG -D CMAKE_INSTALL_PREFIX=/usr/local -D OPENCV_EXTRA_MODULES_PATH=/Users/wangconghao/opencv_contrib/modules ..*

源码中经常出现 InputArray 这种参数类型，这样的话用户给出 Mat，Matx 或 vector 类型都能接受，然后再用 getMat 转换成 Mat。

跟到初始化参数时，感觉源码和论文的方法有些出入，这部分我自己按论文实现了，一边查一下 Mat 的基本操作不很困难。有一些细节的实现论文没有给出具体方法，但 OpenCV 提供了许多强大的函数：

- *转换为齐次坐标，convertPointsToHomogeneous*
- *求单应性矩阵，findHomography*
- *求解齐次线性方程组的非 0 解，SVD::solveZ*
- *将旋转矩阵转换为旋转向量，Rodrigues*

优化阶段，我对于 LM 算法不是很熟悉，看了知乎[14]，再结合比较熟悉的梯度下降，大致了解了它是利用雅克比矩阵来计算当前优化的方向。

[14] 知乎-如何用 LM 算法求解目标函数最小值？，https://www.zhihu.com/question/269579938/answer/349205519

Algorithms	Update Rules	Convergence	Computation Complexity
EBP algorithm	$w_{k+1} = w_k - \alpha g_k$	Stable, slow	Gradient
Newton algorithm	$\mathbf{w}_{k+1} = \mathbf{w}_k - H_k^{-1} g_k$	Unstable, fast	Gradient and Hessian
Gauss–Newton algorithm	$\mathbf{w}_{k+1} = \mathbf{w}_k - \left(J_k^T J_k\right)^{-1} J_k \mathbf{e}_k$	Unstable, fast	Jacobian
Levenberg–Marquardt algorithm	$w_{k+1} = w_k - \left(J_k^T J_k + \mu I\right)^{-1} J_k e_k$	Stable, fast	Jacobian
NBN algorithm [08WC][a]	$\mathbf{w}_{k+1} = \mathbf{w}_k - Q_k^{-1} g_k$	Stable, fast	Quasi Hessian[a]

本想使用 cminpack，但参考资料比较少。找资料的过程中发现 OpenCV 还提供了一个 LMSolve 抽象类，除了官方文档也没什么其他资料，只能在 OpenCV 官方 github 上搜搜看源码对其的使用，但对于用在论文复现上也没什么头绪。

OpenCV 源码中实现相机标定的函数用了 CvLevMarq，所以修改了一下源码拿过来用了，过程中还用到了一些在新版本中已经被封装的内部接口，如 cvProjectPoints2Internal，为了在新版本下能运行，我找到它的声明和定义，抓出来放在自己的项目下，work！

最终结果，跑出来的误差是 1.4 中结果的两倍。内参相差不大，有些图的外参相差很大，我发现源码中在初始化外参的过程中也用到了优化技术，应该和这个有点关系。这里不考虑切向畸变，径向畸变也只考虑了 k_1,k_2，如果考虑和 1.4 中相同的畸变参数，得到结果会稍许相近一些，但影响不大。按论文的方法初始化得到的内参γ不是 0，但相对于其他内参很小，所以将它设为 0 进行后面的工作。

在畸变校正的时候有一些问题，用 1.4 中第二种方法校正后会出现很奇怪的图像，就如同 OpenCV 问答论坛[15]有人提到过的这样，检查了一下发现由

[15] OpenCV 问答论坛-Undistortion at far edges of image，https://answers.opencv.org/question/28438/undistortion-at-far-edges-of-image/

getOptimalNewCameraMatrix（参数α=1）得到的新矩阵很奇怪，不知何原因。用 undistort 表现正常。

OpenCV 源码实现相机标定还是考虑了比较多的，主要体现在 flags 参数上，比如，如果用户设置了 CV_CALIB_FIX_ASPECT_RATIO 模型，就是固定了 f_x/f_y 的比值，那么只用其中一个量进行优化计算，将更准确。相关内容网上有不少的参考资料，这里不再将每个模型拿来讨论。

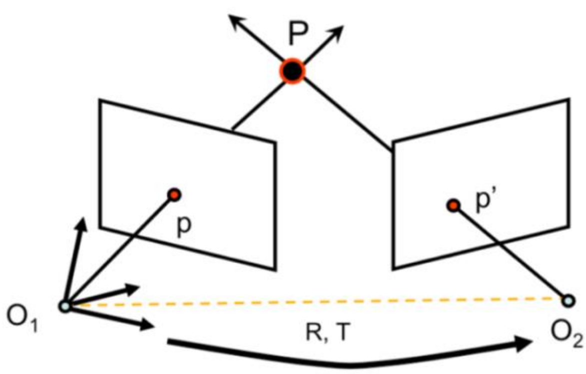
2. 双目基础

双目立体视觉（binocular stereo）是指由两个相机组成的立体视觉系统。您可以找到一个点的 3D 坐标，在两个不同视角的相机中提供其投影。从现在开始，我们将学习如何使用两个相机来估计一个点的深度。

简单起见，考虑具有内参矩阵 M_l 和 M_r 的左右两个相机，并且暂时忽略畸变。让 3D 世界坐标系与左相机的相机坐标系对齐。将从左相机到右相机的变换表示为（R | T）。已知 M_l =（f_x,0,c_x; 0,f_y,c_y; 0,0,1）,M_r =（f_x',0,c'_x; 0,f_y',c'_y; 0,0,1）。

2.1. 双目中的投影（第 9 题）

给定一个 3D 点 P，分别求其在左右相机的投影 p，p’。



注意从左相机到右相机位置的变换表示为（R | T），p'是相对于右相机的物理坐标，则相对于左相机的坐标应该是 Rp'+T；p 是相对于左相机的物理坐标，则相对于右相机的坐标应该是 $R^T(p-T)$。参考了 CS321a 的课件[16]，因为它和题目条件相同，有些资料的条件可能会有所不同。

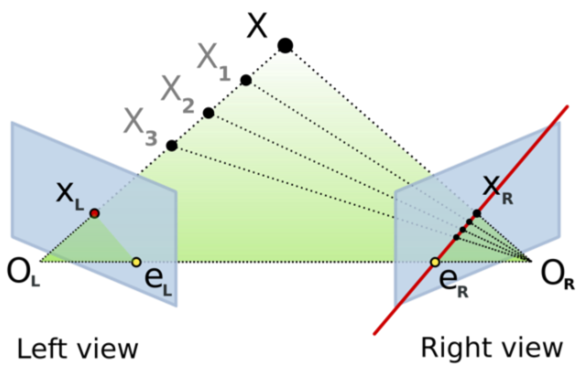
设点 p，p'的像素坐标分别为x_l，x_r（齐次坐标），因为 3D 世界坐标系与左相机的 3D 相机坐标系对齐，所以有

$$\begin{cases} Z_c x_l = M_l(I|0)P \\ Z'_c x_r = M_r(R^T|-R^T T)P \end{cases}$$

Z_c 和 Z'_c 表示两点相机坐标的第三维，也是为了使齐次坐标第三维为 1 而提取出来的一个系数。

2.2. 对极几何（第 10 题）

给定一个左相机的 2D 点像素坐标为 x_l，其对应的 3D 点也将投影到右相机图像平面上。在不知道其精确 3D 坐标的情况下，右相机图像上的可能投影点将位于一条线上，这称为对极约束（epipolar constraint），如维基百科[17]的图。这条线称为 x_l 的对极线（epipolar line），试求出对极线。



[16] Github-CS231a 课件，https://github.com/chizhang529/cs231a

[17] 维基百科-Epipolar geometry，https://en.wikipedia.org/wiki/Epipolar_geometry#Epipolar_line

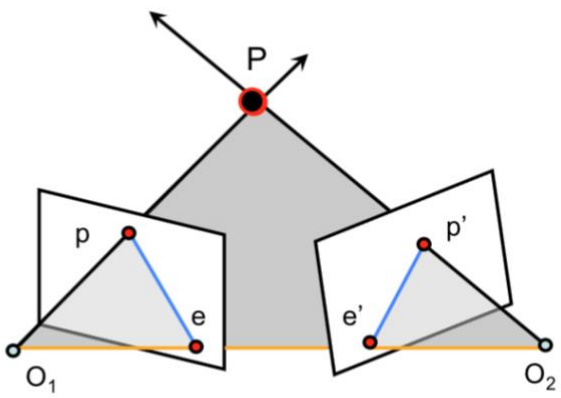
易得 x_l 对应的可能 3D 点在左相机坐标系的坐标为$Z_cM_l^{-1}x_l$，那么其在右相机坐标系的坐标为$R^T(Z_cM_l^{-1}x_l - T)$，投影到右相机平面上就得到了对极线，

$$Z'_c x_r = M_r R^T(Z_c M_l^{-1}x_l - T)$$

这里的 Z_c 可以看作是自变量，表示 x_l 对应的可能 3D 点在射线 X_LX 上移动，所以 x_r 表示的就是 x_l 的对极线。

2.3. 本征矩阵，基础矩阵（第 11 题）

还是来看这张图，



PO_lO_r 称为极面，O_lO_r 称为基线，基线和两个相机像平面的交点 e，e'称为极点，极面和两个相机像平面的交线 pe，p'e'称为极线。

点 p'相对左相机的物理坐标是$Rp' + T$，向量$Rp' + T$和向量T都在极面上，我们取这两个向量的叉积$T \times (Rp' + T) = T \times Rp'$，就得到了一个垂直于极面的向量，向量 p 也在极面上，那么有以下约束，

$$p^T(T \times Rp') = 0$$

向量叉乘可以转换为反对称矩阵和向量的乘，

$$p^T T_\times R p' = 0$$

$$\text{其中} T_\times = \begin{pmatrix} 0 & -T_3 & T_2 \\ T_3 & 0 & -T_1 \\ -T_2 & T_1 & 0 \end{pmatrix}$$

$$令E = T_\times R，则p^T E p' = 0$$

我们把 E 称作本征矩阵（essential matrix），反映了左右相机坐标点（物理）的对应关系。

设点 p，p'的像素坐标分别为x_l，x_r（齐次坐标），因为$Z_c x_l = M_l p$，$Z'_c x_r = M_r p'$，所以有

$$x_l^T M_l^{-T} E M_r^{-1} x_r = 0$$

$$令F = M_l^{-T} E M_r^{-1} = M_l^{-T} T_\times R M_r^{-1}，则x_l^T F x_r = 0$$

我们把 F 称作基础矩阵（fundamental matrix），反映了左右像素坐标点的对应关系。

2.4. 立体标定（第 12 题）

立体标定可以使用 OpenCV 的 stereoCalibrate 函数完成，双目相机需要标定的参数有两个相机的内参矩阵、畸变系数矩阵，描述左右两个相机的相对位置关系的旋转矩阵 R、平移矩阵 T，以及反映左右相机坐标点（物理）对应关系的本征矩阵 E、反映左右像素坐标点对应关系的基础矩阵 F。

由 2.3 可知 E、F 可以由 R、T 计算得到，而 R，T 可以由两个相机的外参计算得到。设（R_l | t_l）和（R_r | t_r）分别是左右相机的外参，那么任意一点 P 相对于左相机坐标为$R_l P + T_l$，相对于右相机坐标为$R_r P + T_r$，有

$$R_l P + T_l = R(R_r P + T_r) + T$$

$$\Rightarrow \begin{cases} R_l = R R_r \\ T_l = R T_r + T \end{cases}$$

$$\Rightarrow \begin{cases} R = R_l R_r^T \\ T = T_l - R T_r \end{cases}$$

双目标定可以在单目标定的基础上计算得到的，也有根据点对估计的方法。本节代码来源于 OpenCV 的立体标定 sample。

2.5. 对极线方程，3D 坐标点的计算（第 13 题）

给定一个左像素坐标点 p_l，写出它的对极线方程。

我们知道一条直线方程可以写作 ax+by+c=0，把 p_l 在右相机平面的对极线参数写成向量形式 $l'=(a,b,c)^T$，那么对极线的方程可以写作 $x_r^T l'=0$。

我们的 $x_l^T F x_r=0$ 和这种形式很相似，取转置会得到 $x_r^T F^T x_l=0$，既然给定了左像素坐标点 p_l，所以可知 $l'=F^T p_l$，也就是其对极线的参数。所以 p_l 的对极线方程为 $x_r^T(F^T p_l)=0$。

再在对极线上给出任意一点 p_r 作为 3D 点 P 在右相机平面上的投影，如何求出点 P 的 3D 坐标。

法 1：既然已经给定了两点的像素坐标，就不难计算这两点的物理坐标，那么由相机中心 O_1，O_2 和 p_l，p_r 分别确定的两条直线 $l=Ep_r$，$l'=E^T p_l$，那么两直线交点 $P=l\times l'$。

法 2：设 p_l，p_r 的像素坐标分别为 $(u,v,1)^T$，$(u',v',1)^T$，3D 点 P 世界坐标为 $(X,Y,Z,1)^T$，因为是标定好的两个相机，相机矩阵已知，所以有，

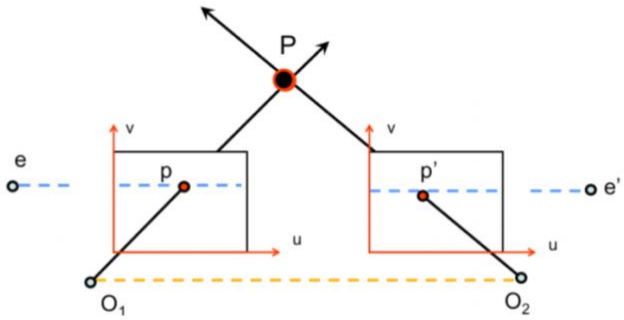
$$\begin{cases} Z_c\begin{pmatrix}u\\v\\1\end{pmatrix}=\begin{pmatrix}k_{11}&k_{12}&k_{13}&k_{14}\\k_{21}&k_{22}&k_{23}&k_{24}\\k_{31}&k_{32}&k_{33}&1\end{pmatrix}\begin{pmatrix}X\\Y\\Z\\1\end{pmatrix}\\ Z'_c\begin{pmatrix}u'\\v'\\1\end{pmatrix}=\begin{pmatrix}k'_{11}&k'_{12}&k'_{13}&k'_{14}\\k'_{21}&k'_{22}&k'_{23}&k'_{24}\\k'_{31}&k'_{32}&k'_{33}&1\end{pmatrix}\begin{pmatrix}X\\Y\\Z\\1\end{pmatrix}\end{cases}$$

$$\Rightarrow\begin{cases} u=(k_{11}-k_{31}u)X+(k_{12}-k_{32}u)Y+(k_{13}-k_{33}u)Z+k_{14}\\ v=(k_{21}-k_{31}v)X+(k_{22}-k_{32}v)Y+(k_{23}-k_{33}v)Z+k_{24}\\ u'=(k'_{11}-k'_{31}u')X+(k'_{12}-k'_{32}u')Y+(k'_{13}-k'_{33}u')Z+k'_{14}\\ v'=(k'_{21}-k'_{31}v')X+(k'_{22}-k'_{32}v')Y+(k'_{23}-k'_{33}v')Z+k'_{24}\end{cases}$$

解上述线性方程组即可。

2.6. 立体校正（第 14，15 题）

我们现实摆放的两个相机的姿势往往是任意的，而如果能使两个相机平行，如下图所示，



极点 e，e’在无穷远处

那么对极线就会处于同一水平线上（也可以是竖直线），意味着这两条线上的像素点坐标的 v 相同，这使得我们在对极线上的搜索变得更加容易。想要达到这样的效果，需要对两个相机做旋转。

我们在标定过程中已经得到了两个相机矩阵，畸变参数以及两个相机的相对位置关系 R,T，可以通过 OpenCV 的 stereoRectify 函数得到两个相机的校正变换（旋转）矩阵和新的投影相机矩阵。然后我们就可以对得到的图像进行一个校正，如同之前的畸变校正一样，我们需要建立点的映射关系，可以通过 initUndistortRectifyMap 函数得到，再用 remap 函数得到校正好的图像。可以用 rectangle 绘制感兴趣矩形区域，用 line 绘制对极线。本节代码来自 OpenCV 的 sample。

一旦图像被校正，对极线将变得平行于图像轴。对于这样的双目相机系统，它们之间的变换将简化为（I | t）。由于这两个相机的坐标轴是平行的，

因此旋转矩阵们将成为单位矩阵。根据 OpenCV 文档中定义的坐标，平移矩阵 t 将变为（b，0，0）。可以从第 14 题的结果中得出基线 b 吗？

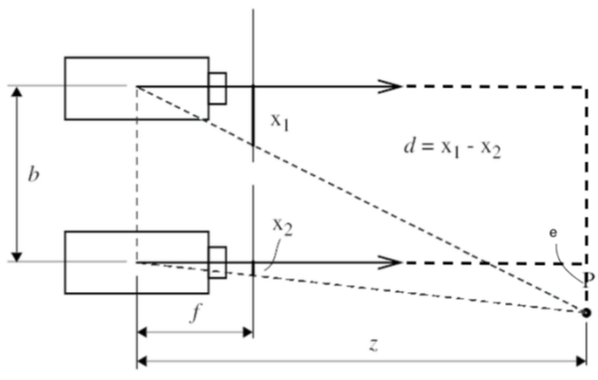
我们只需要计算右相机中心 O_2 在校正变换后在左相机坐标系中的坐标，就得到了 O_1O_2 这条基线。OpenCV 文档里是将左相机坐标转换到右相机坐标的变换表示为(R | T)，也就是说任意一点 P，若相对于左相机坐标为 P_l，则相对于右相机坐标 $P_r=RP_l+T$，和题目条件有所不同，这也是 OpenCV 文档基础矩阵公式$F = cameraMatrix2^{-T} E cameraMatrix1^{-1}$和我们不同的原因。

所以校正前右相机中心坐标为$-R^{-1}T$（按题目条件的话应该是 T）。在第 14 题中，已经得到了左相机的校正（旋转）变换矩阵 R_1（也有一定歧义，描述坐标变换还是坐标轴变换？），两个相机没有发生平移，右相机旋转不影响其中心位置，所以校正后右相机中心坐标为$-R_1R^{-1}T$。

2.7. 深度视差（第 16 题）

对于校正的双目相机系统，对极线就会处于同一水平线上。对于左侧图像上具有坐标（x_1，y）的像素，其匹配点（右侧图像上 3D 点的投影）具有形式（x_1-d，y）的坐标。$d=x_1-x_2$ 称为像素的视差。给定基线 b 和相机矩阵，能否得出 3D 点的坐标？如果相机的垂直和水平焦距相同，是否可以将像素的深度 z（Z 坐标值）写为 z = bf / d？写下您的推导。

谷歌搜索“Depth Disparity”，看到下面这张图，抓下来加了辅助线，



由相似可得，

$$\begin{cases} \dfrac{x_2}{e} = \dfrac{f}{z} = \dfrac{x_1}{b+e} \\ \mathrm{d} = x_1 - x_2 \end{cases}$$

$$\Rightarrow z = \frac{bf}{d}$$

已知 2D 坐标，相机各参数和 z，代入 1.2 求得的公式即可。

3. 立体匹配

从第 2 章最后一个问题，我们可以找到一种使用双目相机系统估计像素深度的方法。校准可以为我们提供基线 b 以及所有相机的内外参数。我们可以校正图像畸变，然后立体校正它们以使其具有视差和深度之间的简单关系，剩下的就是计算每个像素的视差。这是沿着水平对极线针对每个像素进行的一维搜索。该任务称为立体匹配。

3.1. 更新中...