

从对极几何恢复相机运动

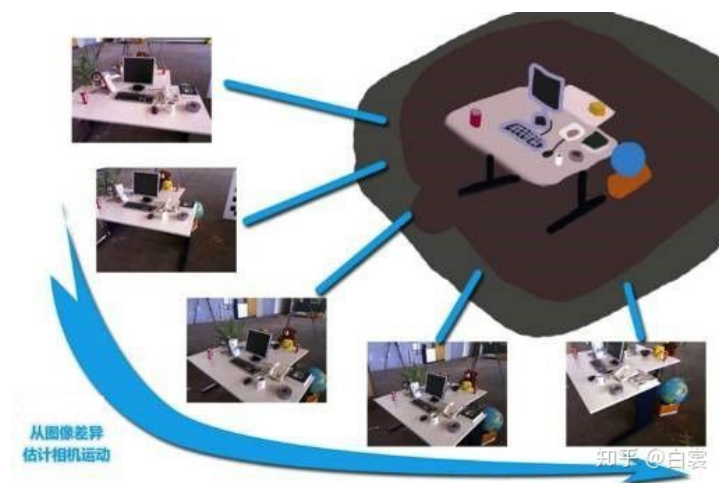


白兰

关注他

60 人赞同了该文章

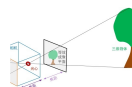
在机器人SLAM、自动驾驶中经常会遇到一个问题：如何通过相机拍摄的一组画面反推出相机在真实世界中的运动轨迹。这就是典型的视觉里程计问题。



一般来说，对于通过单相机拍摄的画面估计运动轨迹，需要用到对极几何知识。所以本文主要分析如何通过两张图对极几何估计相机运动，也是对《视觉SLAM十四讲》中视觉里程计章节中没有说的地方进行补充。

理解对极几何依赖于相机模型，所以建议先阅读相机模型文章。记得点赞哦！

相机模型与视觉测距不完全指南
zhuatlan.zhihu.com



一、基础知识

在讨论对极几何之前，先来看一些向量和矩阵的基础知识。

• 内积（点乘）

▲ 赞同 60 ▼

● 6 条评论

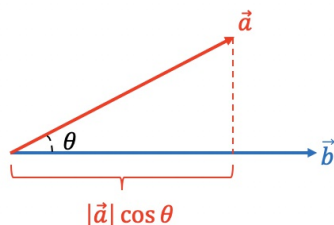
➤ 分享

♥ 喜欢

★ 收藏

📄 申请转载

...



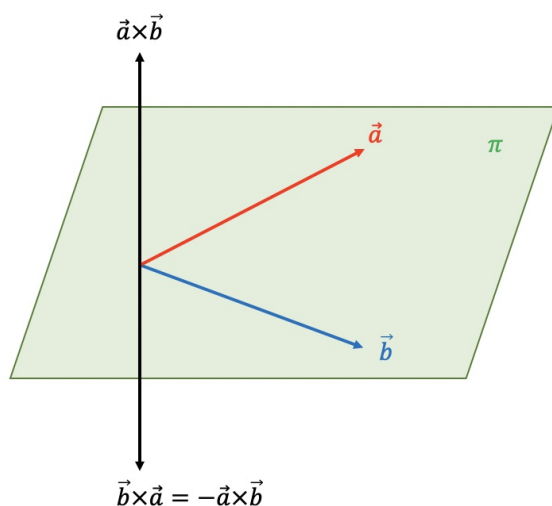
知乎 @白裳

有向量 $\vec{a} = \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}$ 和 $\vec{b} = \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix}$ ，他们的内积为：

$$\vec{a} \cdot \vec{b} = |\vec{a}| |\vec{b}| \cos \theta = (x_1, y_1, z_1) \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} \quad (1.1)$$

显然当 \vec{a} 和 \vec{b} 互相垂直时 $\vec{a} \cdot \vec{b} = 0$ 。

• 外积 (叉乘)



知乎 @白裳

与内积 $\vec{a} \cdot \vec{b}$ 结果是一个数值不同，外积 $\vec{a} \times \vec{b}$ 的结果是一个向量。

假设 \vec{a} 和 \vec{b} 向量所在平面 π ，那么从几何上看 $\vec{a} \times \vec{b}$ 是一个垂直于 π 平面的向量。

$$\begin{aligned} \vec{a} \times \vec{b} &= \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{vmatrix} \\ &= (y_1 z_2 - y_2 z_1) \vec{i} - (x_1 z_2 - x_2 z_1) \vec{j} + (x_1 y_2 - x_2 y_1) \vec{k} \end{aligned} \quad (1.2)$$

写成坐标形式：

$$\vec{a} \times \vec{b} = \left((y_1 z_2 - y_2 z_1), -(x_1 z_2 - x_2 z_1), (x_1 y_2 - x_2 y_1) \right)^T \quad (1.3)$$

由于 $\vec{a} \times \vec{b}$ 垂直于 \vec{a} 和 \vec{b} ，那么必然有：

$$(\vec{a} \times \vec{b}) \cdot \vec{a} = (\vec{a} \times \vec{b}) \cdot \vec{b} = 0 \quad (1.4)$$

将向量叉乘写成矩阵乘以坐标点形式：

$$\vec{a} \times \vec{b} = \begin{bmatrix} 0 & -z_1 & y_1 \\ z_1 & 0 & -x_1 \\ -y_1 & x_1 & 0 \end{bmatrix} \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = [\vec{a}]_{\times} \vec{b} \quad (1.5)$$



已知 $\vec{a} = \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}$ ，定义叉乘矩阵 $[\vec{a}]_{\times}$ 为：

$$[\vec{a}]_{\times} = \begin{bmatrix} 0 & -z_1 & y_1 \\ z_1 & 0 & -x_1 \\ -y_1 & x_1 & 0 \end{bmatrix} \quad (1.6)$$

这个矩阵之后会用到。

• 反对称矩阵

若 A 满足 $-A = A^T$ 关系，则 A 为反对称矩阵（其中 A^T 代表矩阵转置）。

显然上面的叉乘矩阵 $[\vec{a}]_{\times}$ 是反对称矩阵。

此处不经证明给出，对于任意3x3实反对称矩阵 A 都可以表示为：

$$A = UZU^T = U \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} U^T \quad (1.7)$$

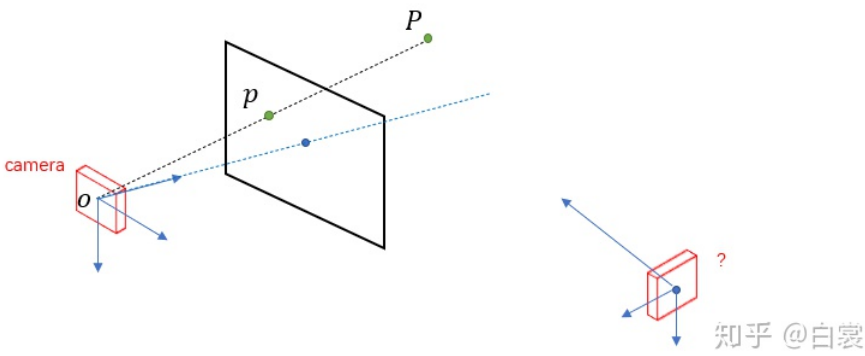
其中 U 是3x3正交矩阵（ $UU^T = U^T U = I$ ）。这个结论很重要，一会要用到。

二、对极几何（本质矩阵和基础矩阵）

在之前相机模型文章中有分析到，从相机坐标系 xyz 变换为像素坐标系 uv 的公式为：

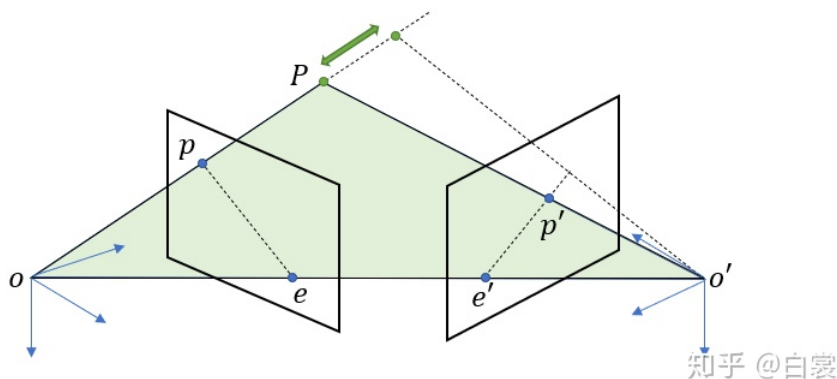
$$z \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = K \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad (2.1)$$

其中相机内参矩阵 K 是已知的（预先标定好）。



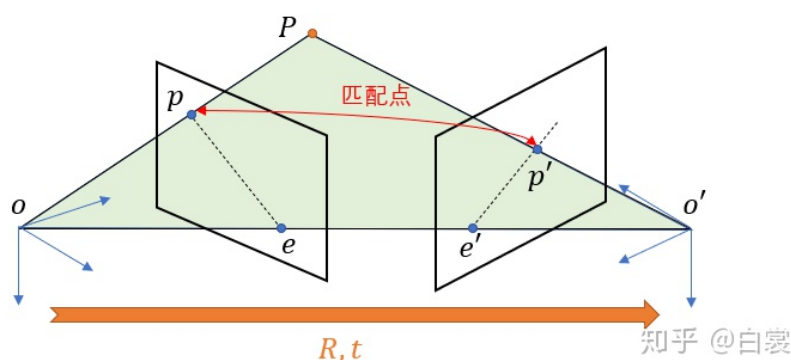
世界中有一点 $P(x, y, z)$ ，相机 o 拍摄 P 并在成像平面行成像素点 $p(u, v)$ 。由于相机在成像过程中丢失了深度信息，所以我们无法知道 P 的深度（即 z ）。

那么问题来了，当相机 o 移动到另一个外置 o' 再拍一张图时如何？



假设相机 o 拍摄 P 在成像平面形成对应的像素点 p ，此时我们只能知道 P 在 \vec{op} 射线上（但是无法确定具体位置）；当相机移动到 o' 后再去拍摄 P 生成新的像素点 p' ，如果图像通过特征点匹配确定 p 和 p' 是同一点，那么这时 P 的位置也就确定了。

• 本质矩阵



需要特别说明，这里的 $\vec{oP} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ 是坐标系 o 下的向量；而 $\vec{o'P} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$ 是坐标系 o' 下的向量。

现在我们要将所有的向量和坐标统一在同一个坐标系内，那么在 o' 坐标系下 \vec{oP} 为：

$$\vec{oP} = R \begin{pmatrix} x \\ y \\ z \end{pmatrix} + t \quad (2.2)$$

其中 R 是3x3旋转矩阵， t 是3x1平移向量。

由于 $\vec{oP} = R \begin{pmatrix} x \\ y \\ z \end{pmatrix} + t$ 、 $\vec{o'P} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$ 和 $\vec{o'o} = -t$ 共面，所以他们的混合积为0：

$$\vec{o'P} \cdot (\vec{o'o} \times \vec{oP}) = 0 \quad (2.3)$$

那么在 o' 坐标系下，可以表示为：

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}^T \left(-[t]_{\times} \left(R \begin{pmatrix} x \\ y \\ z \end{pmatrix} + t \right) \right) = 0 \quad (2.4)$$

由于 $[t]_{\times} \cdot t = \vec{t} \times \vec{t} = 0$

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}^T \left(-[t]_{\times} R \begin{pmatrix} x \\ y \\ z \end{pmatrix} \right) = 0$$



这里的 $E = [t]_{\times} R$ 就是对极几何中的本质矩阵(essential matrix)。特别注意 E 是由反对称矩阵 $[t]_{\times}$ 和正交矩阵 R 相乘获得，这是一个非常重要的性质！

• 基础矩阵

进一步，又由相机模型可以得到像素坐标的转换关系：

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = zK^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \quad (2.6)$$

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = z'K'^{-1} \begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix} \quad (2.7)$$

带入后：

$$\left[z'K'^{-1} \begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix} \right]^T E \left[zK^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \right] = 0 \quad (2.8)$$

等式右边为0，直接消去常数 z 和 z' ：

$$\left[K'^{-1} \begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix} \right]^T E \left[K^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \right] = 0 \quad (2.9)$$

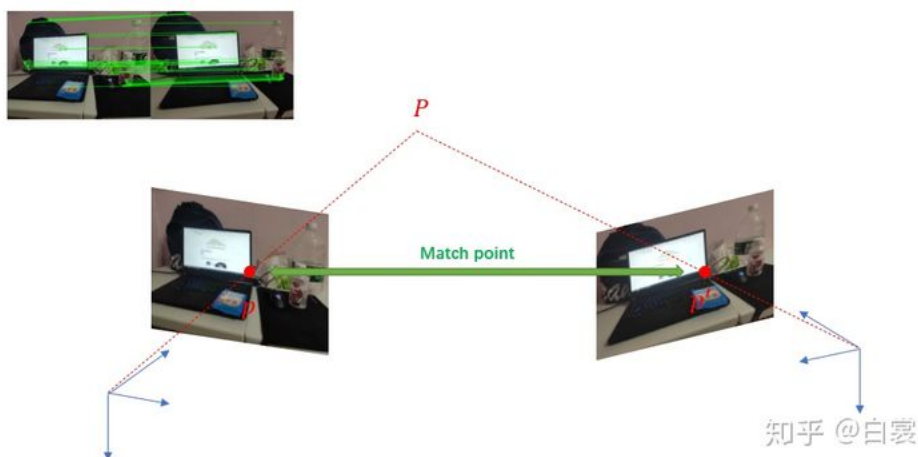
整理得：

$$\begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix}^T \left[(K')^{-1} E K^{-1} \right] \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = 0 \quad (2.10)$$

这里的 $F = (K')^{-1} E K^{-1}$ 就是基础矩阵(fundamental matrix)，基础矩阵实际与本质矩阵只是相差了相机内参 K 。

三、本质矩阵求解相机运动

在实际中我们可以通过特征点匹配（如ORB-SLAM使用ORB特征）计算出相机运动前后两幅视图中较多组匹配特征点坐标 $p(u, v)$ 与 $p'(u', v')$ 。另外相机内参矩阵 K 已知。



通过对极几何恢复相机运动的过程为：特征点匹配 → 计算本质矩阵 → 恢复相机运动。

• 计算本质矩阵 E

以OpenCV `cv::findEssentialMat`函数计算 E 为例，具体代码如下：



```
// _points1对应p, _points2对应p', _cameraMatrix是相机内参矩阵K
cv::Mat cv::findEssentialMat( InputArray _points1, InputArray _points2, InputArray _cameraMatrix,
                              int method, double prob, double threshold, OutputArray _E )
{
    CV_INSTRUMENT_REGION();

    // 将匹配的特征点和相机矩阵等数据转化为浮点型
    Mat points1, points2, cameraMatrix;
    _points1.getMat().convertTo(points1, CV_64F);
    _points2.getMat().convertTo(points2, CV_64F);
    _cameraMatrix.getMat().convertTo(cameraMatrix, CV_64F);

    int npoints = points1.checkVector(2);
    CV_Assert( npoints >= 0 && points2.checkVector(2) == npoints &&
               points1.type() == points2.type());

    CV_Assert(cameraMatrix.rows == 3 && cameraMatrix.cols == 3 && cameraMatrix.channels() == 1);

    if (points1.channels() > 1)
    {
        points1 = points1.reshape(1, npoints);
        points2 = points2.reshape(1, npoints);
    }

    double fx = cameraMatrix.at<double>(0,0);
    double fy = cameraMatrix.at<double>(1,1);
    double cx = cameraMatrix.at<double>(0,2);
    double cy = cameraMatrix.at<double>(1,2);

    // 按照公式 (19) 和公式 (20) 计算 E
    points1.col(0) = (points1.col(0) - cx) / fx;
    points2.col(0) = (points2.col(0) - cx) / fx;
    points1.col(1) = (points1.col(1) - cy) / fy;
    points2.col(1) = (points2.col(1) - cy) / fy;

    // Reshape data to fit opencv ransac function
    points1 = points1.reshape(2, npoints);
    points2 = points2.reshape(2, npoints);

    threshold /= (fx+fy)/2;

    Mat E;
    if( method == RANSAC )
        createRANSACPointSetRegistrator(makePtr<EMEstimatorCallback>(), 5, threshold,
        else
            createLMeDSPointSetRegistrator(makePtr<EMEstimatorCallback>(), 5, prob)->run(p

    return E;
}
```

以本质矩阵 E 的定义公式 (2.5) 来看，两边乘以 $\frac{1}{zz'}$ ，得：

$$\begin{bmatrix} \frac{1}{z'} \begin{pmatrix} x' \\ y' \end{pmatrix}^T \end{bmatrix} E \begin{bmatrix} \frac{1}{z} \begin{pmatrix} x \\ y \end{pmatrix} \end{bmatrix} = 0 \quad (3.1)$$

化简得

$$\begin{pmatrix} x'/z' \\ y'/z' \\ 1 \end{pmatrix}^T E \begin{pmatrix} x/z \\ y/z \\ 1 \end{pmatrix} = 0 \quad (3.2)$$



$$\begin{cases} z \cdot u = f_x x + z \cdot v_0 \\ z \cdot v = f_x x + z \cdot v_0 \end{cases} \Rightarrow \begin{cases} \frac{x}{z} = \frac{u - u_0}{f_x} \\ \frac{y}{z} = \frac{v - v_0}{f_y} \end{cases} \quad (3.3)$$

其中 $c_x = u_0$ 且 $c_y = v_0$ ，带相机主点位置。这时明显可以看到findEssentialMat代码通过上述过程计算本质矩阵 E 。

• 奇异值分解 E 计算相机运动

通过特征点+findEssentialMat求解得到 E 后，接下来需要计算旋转矩阵 R 和平移向量 t 来估计相机运动。已知：

$$E = [t]_{\times} R = U Z U^T R \quad (3.4)$$

对 Z 进行初等行变换：

$$[Z|I] = \left[\begin{array}{ccc|ccc} 0 & 1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right] \Rightarrow \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right] = [\text{diag}(1, 1, 0)|W] \quad (3.5)$$

这里的 W 是一个正交矩阵（ $WW^T = W^T W = I$ ）。通过上述初等行变换可以得到：

$$ZW = \text{diag}(1, 1, 0) \Rightarrow Z = \text{diag}(1, 1, 0)W^T \quad (3.6)$$

另外同时有（不信你自己算）：

$$ZW^T = -\text{diag}(1, 1, 0) \Rightarrow Z = -\text{diag}(1, 1, 0)W \quad (3.7)$$

那么通过公式 (3.6) 改写 E 为：

$$\begin{aligned} E &= U Z U^T R = U [\text{diag}(1, 1, 0)W^{-1}] U^T R \\ &= U \text{diag}(1, 1, 0)W^T U^T R \\ &= U \text{diag}(1, 1, 0)V_1^T \end{aligned} \quad (3.8)$$

同理通过公式 (3.7) 改写 E 为：

$$\begin{aligned} E &= U Z U^T R = U \text{diag}(1, 1, 0)(-W U^T R) \\ &= U \text{diag}(1, 1, 0)V_2^T \end{aligned} \quad (3.9)$$

观察公式 (3.8) 和 (3.9)，这就是典型的奇异值分解（SVD）：

$$E = U \Sigma V^T = U \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix} V^T \quad (3.10)$$

其中 U 和 V 都是正交矩阵， $\sigma_1 = \sigma_2 \neq 0$ 且 $\sigma_3 = 0$ 。同时可以得到结论：

一个矩阵是本质矩阵的充要条件是其奇异值中有两个相等且第三个是0。

那么看到这儿应该比较清晰了，通过两幅图对极几何计算相机运动的方法是：

1. 特征点匹配计算 p_i 和 p'_i
2. 通过对极约束 $(p_i)^T E p'_i = 0$ 计算本质矩阵 E
3. 用奇异值分解 $E = U \Sigma V^T$ 计算 R 和 $[t]_{\times}$ ，从而得到旋转矩阵 R 和平移向量 t

• 相机运动 R 和 t 的四个解

已知 E 并进行奇异值分解求得 U 和 V 之后，那么根据公式 (3.8) 得到旋转矩阵第一个解：

$$V = W^T U^T R \Rightarrow R_1 = U W V^T \quad (3.11)$$

$$V = -WU^T R \Rightarrow R_2 = UW^T V^T \quad (3.12)$$



注意，对极约束 $(p')^T E p = 0$ ，可以知道求解 E 时并不关心正负号，所以求解 R_2 也可以直接去掉负号。

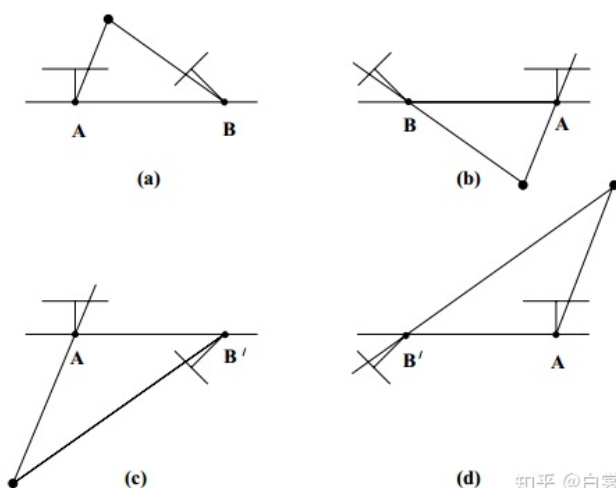
求解出 R 接着可以得到：

$$[t]_{\times} = ER^T \quad (3.13)$$

同时：

$$[-t]_{\times} = -ER^T \quad (3.14)$$

当然对本质矩阵的数值解来说， E 和 $-E$ 只是相差一个负号，但是对于平移向量 t 和 $(-t)$ 代表方向相反。综合考虑，求解得到的 R 和 t 组合起来共有4种如下可能位置：



由于真实物理条件限定，两个相机和被拍摄点在同一侧，且被拍摄点在相机前方，所以只有上图中(a)是符合真实的解。

在OpenCV中cv::recoverPose函数帮我们做这一堆事情：

```
int cv::recoverPose( InputArray E, InputArray _points1, InputArray _points2,
                    InputArray _cameraMatrix, OutputArray _R, OutputArray _t,
                    InputOutputArray _mask, OutputArray triangulatedPoints)
```

输入本质矩阵、匹配特征点 $_points1 + _points2$ 、相机矩阵 $_cameraMatrix$ ，输出旋转矩阵 $_R$ 和平移向量 $_t$ 。代码太长就不列出来了。

四、若干问题

• 对极几何不能求解纯旋转问题

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}^T ([t]_{\times} R) \begin{pmatrix} x \\ y \\ z \end{pmatrix} = 0 \quad (4.1)$$

当相机无平移且只有旋转时 $[t]_{\times}$ 为全 0 矩阵，等式两边都等于 0 显然会导致无法求解 R 。反过来当相机只有平移且无旋转时 $R = I$ ，并不影响 $[t]_{\times}$ 的求解，这时问题退化成双目测距。

• 对极几何“结构性”恢复运动

对于对极约束 (4.1)，在等式两边乘以任意不为 0 的常数 k 都成立：

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}^T K([t]_x R) \begin{pmatrix} x \\ y \\ z \end{pmatrix} = 0 \quad (4.2)$$



换句话说，如果 $\vec{t}_1 = (t_x, t_y, t_z)^T$ 是 (4.1) 的解，那么 $\vec{t}_2 = (kt_x, kt_y, kt_z)^T$ 也是 (4.1) 的解。

这就是说，即使求解出平移向量 \vec{t} ，我们也无法知道 \vec{t} 的单位具体是米、厘米还是毫米。所以计出的 \vec{t} 只是恢复移动的“结构性”，并不是真实值。

最近研究了对极几何，记录在此。谢谢观看，OVER!

编辑于 2020-12-12

「真诚赞赏，手留余香」

赞赏

还没有人赞赏，快来当第一个赞赏的人吧！

[同时定位和地图构建 \(SLAM\)](#)

[计算机视觉](#)

[图像处理](#)

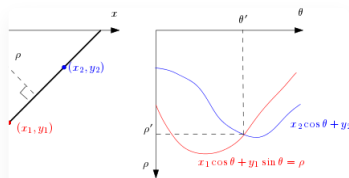
文章被以下专栏收录



机器学习随笔

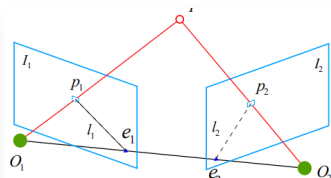
想了解机器学习就点这里

推荐阅读



【计算机视觉】8. 边缘检测与Hough变换

Encoder



VSLAM 笔记——我们如何通过图像来计算位姿的变化：对...

菠萝包包包



图像理解(第3版) 第1章学习记录

贤鱼卓君

6 条评论

⇌ 切换为时间排序

只有关注了作者的人才可以评论

补也 路漫漫其修远兮

2020-11-20

写的很好，赞一个。那如果要计算真实 \mathbf{t} 的话，可不可以先通过三角测量计算出点P的物理位置，然后通过第二个相机的成像过程得出那个比例呢？还是有更好的计算方法吗？博主有研究过吗？

👍 1

▲ 赞同 60 ▼

● 6 条评论

➦ 分享

♥ 喜欢

★ 收藏

📄 申请转载

...



我也在想这个问题。基于对级约束得到的位移它没有准确的尺度，是不是可以通过三角测量测出两个深度值来反算真实的位移量呢？

👍 赞

非也 路漫漫其修远兮 回复 晋王兼河东节度使 2020-12-30

对极约束计算的 t 是归一化的。
三角测量也需要真实位姿 才能求解空间点的物理位置。
所以我之前的问题是不行的。
如果知道像素深度的话，就用PnP 和 ICP去计算位姿，这个是真实的。

👍 赞

展开其他 1 条回复

 coeus 2020-12-03

写的很透彻

👍 赞

 大脸君 2020-10-20

多谢楼主总结！

👍 赞