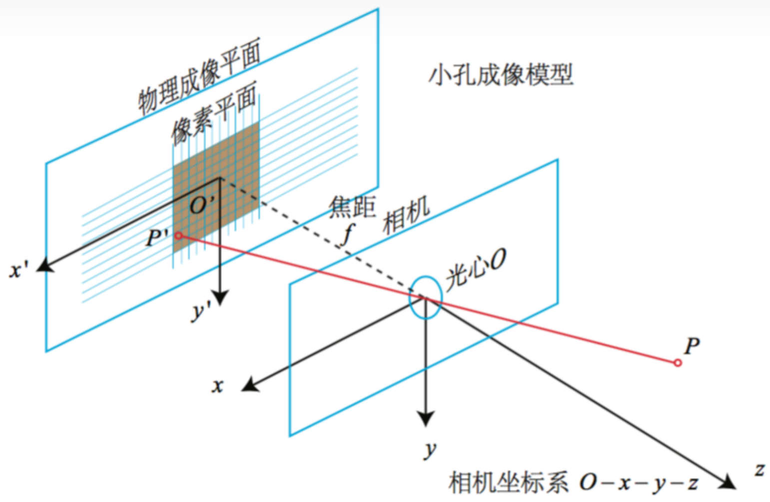


EPnP原理与源码详解

流星雨
SLAM小学生

一、 EPnP解决的问题

输入：世界坐标系下的n 个3D点坐标以及与之对应的 2D 像点坐标、相机内参 $\mathbf{K}$ 。

输出：恢复影像的位姿 $\mathbf{R}$和 $\mathbf{t}$ 。

二、 为什么选择EPnP恢复相机的姿态

- 与其他方法相比，EPnP方法的复杂度为$O(n)$，对于点对数量较多的PnP问题，非常高效。
- 核心思想是将3D点表示为4个控制点的组合，优化也只针对4个控制点，所以速度很快，在求解 $\mathbf{Mx}=0$ 时，最多考虑了4个奇异向量，因此精度也很高。

三、 步骤

Step 1 : 在世界坐标系下确定4个控制点 $x_1^w \ x_2^w x_3^w x_4^w$。（点云的质心为 x_1^w 作为原点，通过主成分分析PCA得到另外的三个点$x_2^w x_3^w x_4^w$，建立坐标系）。

Step 2 : 计算每个3D物方点对应的4个HB坐标 $\alpha_{i1} \ \alpha_{i2}\alpha_{i3}\alpha_{i4}$，每个3D物方可以通过控制点和其对应的HB坐标表示。根据欧式变换的不变特性，HB坐标在世界坐标系和相机坐标系下都适用。因此，通过相机内参K、HB坐标$\alpha_{i1}\alpha_{i2}\alpha_{i3}\alpha_{i4}$、3D物方点坐标、像点坐标，建立投影模型方程，求解4个控制点在相机坐标系下的坐标 $c_1^c \ c_2^c c_3^c c_4^c$。

Step 3 : 已知相机坐标系下的控制点$c_1^c c_2^c c_3^c \ c_4^c$ 和4个HB坐标$\alpha_{i1}\alpha_{i2}\alpha_{i3}\alpha_{i4}$。求解3D物方点在相机坐标系下的坐标。

Step 4 : 已知两组具有对应关系的点云，通过SVD分解，即可求出$R \ t$。(该步骤可以参考ICP的原理)。

Least-Squares Rigid Motion Using SVD
igl.ethz.ch/projects/ARAP/svd_rot.pdf

四、 理论推导

1. Homo

- 3D物方点在世界坐标系中的坐标为：$\mathbf{p}_i^w$，$i=1,2,\cdots n$
- 3D物方点在相机坐标系下的坐标为：$\mathbf{p}_i^c$，$i=1,2,\cdots n$
- 4个控制点在世界坐标系中的坐标为：$\mathbf{c}_j^w$，$j=1,2,\cdots 4$
- 4个控制点在相机坐标系下的坐标为：$\mathbf{c}_j^c$，$j=1,2,\cdots 4$

EPnP算法中引入了控制点，在三维坐标系中，任意一个3D点都可以由4个不共面的控制点线性表示。

\[$\mathbf{p}_i^w=\sum_{j=1}^{4}\alpha_{ij}\mathbf{c}_j^w$\] (1a)

$\sum_{j=1}^{4}\alpha_{ij}=1$ (1b)

（1）中的式子可以写成如下形式：

$$\begin{bmatrix}\mathbf{p}_i^w\\1\end{bmatrix}=\mathbf{C}\begin{bmatrix}\alpha_{i1}\\\alpha_{i2}\\\alpha_{i3}\\\alpha_{i4}\end{bmatrix}=\begin{bmatrix}\mathbf{c}_1^w&\mathbf{c}_2^w&\mathbf{c}_3^w&\mathbf{c}_4^w\\1&1&1&1\end{bmatrix}\begin{bmatrix}\alpha_{i1}\\\alpha_{i2}\\\alpha_{i3}\\\alpha_{i4}\end{bmatrix} \quad (2)$$

式中 α_{ij} 是齐次重心坐标（homogeneous barycentric coordinates）。*本质上就是：3D参考点的齐次坐标是控制点齐次坐标的线性组合。*

假设相机的外参为 $[\mathbf{R}\quad\mathbf{t}]$，那么控制点在相机坐标系和世界坐标系下存在以下转换关系：

$$\mathbf{c}_j^c=[\mathbf{R}\quad\mathbf{t}]\begin{bmatrix}\mathbf{c}_j^w\\1\end{bmatrix} \quad (3)$$

对于3D物方点：

$$\mathbf{p}_i^c=[\mathbf{R}\quad\mathbf{t}]\begin{bmatrix}\mathbf{p}_i^w\\1\end{bmatrix}=[\mathbf{R}\quad\mathbf{t}]\begin{bmatrix}\sum_{j=1}^{4}\alpha_{ij}\mathbf{c}_j^w\\1\end{bmatrix} \quad (4a)$$

$$\mathbf{p}_i^c=[\mathbf{R}\quad\mathbf{t}]\begin{bmatrix}\sum_{j=1}^{4}\alpha_{ij}\mathbf{c}_j^w\\\sum_{j=1}^{4}\alpha_{ij}\end{bmatrix}=\sum_{j=1}^{4}\alpha_{ij}[\mathbf{R}\quad\mathbf{t}]\begin{bmatrix}\mathbf{c}_j^w\\1\end{bmatrix}=\sum_{j=1}^{4}\alpha_{ij}\mathbf{c}_j^c \quad (4b)$$

通过上面的推导可以发现，α_{ij}在世界坐标系和相机坐标系下相同，这就意味着，我们可以在世界坐标系下求出α_{ij}，然后应用在相机坐标系下。根据公式（2），我们也可以得到 α_{ij} 的计算方法：

$$\begin{bmatrix}\alpha_{i1}\\\alpha_{i2}\\\alpha_{i3}\\\alpha_{i4}\end{bmatrix}_{4\times1}=\begin{bmatrix}\mathbf{c}_1^w&\mathbf{c}_2^w&\mathbf{c}_3^w&\mathbf{c}_4^w\\1&1&1&1\end{bmatrix}_{4\times4}^{-1}\begin{bmatrix}\mathbf{p}_i^w\\1\end{bmatrix}_{4\times1}=\mathbf{C}^{-1}\begin{bmatrix}\mathbf{p}_i^w\\1\end{bmatrix} \quad (5)$$

2. 控制点的选择

理论上，只要满足(2)式中的 C 可逆即可，论文中给出了一个具体的控制点确定方法。3D物方点集为 $\{\mathbf{p}_i^w,\ i=1,2,\cdots n\}$，选择3D参考点的重心为第一个控制点：

$$\mathbf{c}_1^w=\frac{1}{n}\sum_{i=1}^{n}\mathbf{p}_i^w \quad (6)$$

对3D物方

$$\left[(\mathbf{p}_n^w)^T - (\mathbf{c}_1^w)^T \right]$$

计算 $\mathbf{A}^T\mathbf{A}$ 的三个特征值 $\lambda_1,\lambda_2,\lambda_3$ 对应的特征向量 $\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3$,那么剩余的三个控制点可以按照下面的公式来确定：

$$\begin{aligned}\mathbf{c}_2^w&=\mathbf{c}_1^w+\sqrt{\lambda_1}\mathbf{v}_1\\ \mathbf{c}_3^w&=\mathbf{c}_1^w+\sqrt{\lambda_2}\mathbf{v}_2 \quad (8)\\ \mathbf{c}_4^w&=\mathbf{c}_1^w+\sqrt{\lambda_3}\mathbf{v}_3\end{aligned}$$

世界坐标系下控制点的计算：第一步找到点云的重心作为坐标系的原点，然后通过主成分分析(PCA)确定坐标轴的三个方向。

```
/**
 * @brief 从给定的匹配点中计算出四个控制点
 *
 */
void PnPsolver::choose_control_points(void)
{
  // Take C0 as the reference points centroid:
  // Step 1：第一个控制点：参与PnP计算的参考3D点的质心（均值）
  // cws[4][3] 存储控制点在世界坐标系下的坐标，第一维表示是哪个控制点，第二维表示是哪个坐标(x,y
  // 计算前先把第1个控制点坐标清零
  cws[0][0] = cws[0][1] = cws[0][2] = 0;

  // 遍历每个匹配点中世界坐标系3D点，然后对每个坐标轴加和
  // number_of_correspondences 默认是 4
  for(int i = 0; i < number_of_correspondences; i++)
    for(int j = 0; j < 3; j++)
      cws[0][j] += pws[3 * i + j];

  // 再对每个轴上取均值
  for(int j = 0; j < 3; j++)
    cws[0][j] /= number_of_correspondences;


  // Take C1, C2, and C3 from PCA on the reference points:
  // Step 2：计算其它三个控制点，C1, C2, C3通过特征值分解得到
  // ref: https://www.zhihu.com/question/38417101
  // ref: https://yjk94.wordpress.com/2016/11/11/pca-to-layman/

  // 将所有的3D参考点写成矩阵，(number_of_correspondences * 3)的矩阵
  CvMat * PW0 = cvCreateMat(number_of_correspondences, 3, CV_64F);

  double pw0tpw0[3 * 3], dc[3], uct[3 * 3];         // 下面变量的数据区
  CvMat PW0tPW0 = cvMat(3, 3, CV_64F, pw0tpw0);    // PW0^T * PW0，为了进行特征值分解
  CvMat DC      = cvMat(3, 1, CV_64F, dc);         // 特征值
  CvMat UCt     = cvMat(3, 3, CV_64F, uct);        // 特征向量

  // Step 2.1：将存在pws中的参考3D点减去第一个控制点(均值中心)的坐标（相当于把第一个控制点作为
  for(int i = 0; i < number_of_correspondences; i++)
    for(int j = 0; j < 3; j++)
      PW0->data.db[3 * i + j] = pws[3 * i + j] - cws[0][j];

  // Step 2.2：利用特征值分解得到三个主方向
  // PW0^T * PW0
  // cvMulTransposed(A_src,Res_dst,order, delta=null,scale=1):
  // Calculates Res=(A-delta)*(A-delta)^T (order=0) or (A-delta)^T*(A-delta) (order=1)
  cvMulTransposed(PW0, &PW0tPW0, 1);

  // 这里实际是特征值分解
  cvSVD(&PW0tPW0,                         // A
```

```
    cvReleaseMat(&PW0);

    // Step 2.3: 得到C1, C2, C3三个3D控制点，最后加上之前减掉的第一个控制点这个偏移量
    for(int i = 1; i < 4; i++) {
      // 这里只需要遍历后面3个控制点
      double k = sqrt(dc[i - 1] / number_of_correspondences);
      for(int j = 0; j < 3; j++)
        cws[i][j] = cws[0][j] + k * uct[3 * (i - 1) + j];
    }
}
```


到目前为止我们知道了世界坐标系下的4个控制点 $\mathbf{c}_j^c$ ， $j=1,2,\cdots 4$ ，通过控制点和3D物方点云 $\mathbf{p}_i^w$ ， $i=1,2,\cdots n$ ，我们可以根据公式(5)计算每个点对应的4个齐次重心坐标 α_{ij} 。

$$\begin{bmatrix}\alpha_{i1}\\ \alpha_{i2}\\ \alpha_{i3}\\ \alpha_{i4}\end{bmatrix}_{4\times 1}=\begin{bmatrix}\mathbf{c}_1^w & \mathbf{c}_2^w & \mathbf{c}_3^w & \mathbf{c}_4^w\\ 1 & 1 & 1 & 1\end{bmatrix}_{4\times 4}^{-1}\begin{bmatrix}\mathbf{p}_i^w\\ 1\end{bmatrix}_{4\times 1}=\mathbf{C}^{-1}\begin{bmatrix}\mathbf{p}_i^w\\ 1\end{bmatrix}$$

```
/**
 * @brief 求解世界坐标系下四个控制点的系数alphas，在相机坐标系下系数不变
 *
 */
void PnPsolver::compute_barycentric_coordinates(void)
{
  // pws为世界坐标系下3D参考点的坐标
  // cws1 cws2 cws3 cws4为世界坐标系下四个控制点的坐标
  // alphas 四个控制点的系数，每一个pws，都有一组alphas与之对应
  double cc[3 * 3], cc_inv[3 * 3];
  CvMat CC     = cvMat(3, 3, CV_64F, cc);       // 除第1个控制点外，另外3个控制点在控制点坐
  CvMat CC_inv = cvMat(3, 3, CV_64F, cc_inv);   // 上面这个矩阵的逆矩阵

  // Step 1: 第一个控制点在质心的位置，后面三个控制点减去第一个控制点的坐标（以第一个控制点为原
  // 减去质心后得到x y z轴
  //
  // cws的排列 |cws1_x cws1_y cws1_z|  ---> |cws1|
  //           |cws2_x cws2_y cws2_z|       |cws2|
  //           |cws3_x cws3_y cws3_z|       |cws3|
  //           |cws4_x cws4_y cws4_z|       |cws4|
  //
  // cc的排列  |cc2_x cc3_x cc4_x|  --->|cc2 cc3 cc4|
  //           |cc2_y cc3_y cc4_y|
  //           |cc2_z cc3_z cc4_z|

  // 将后面3个控制点cws 去重心后 转化为 cc
  for(int i = 0; i < 3; i++)                      // x y z 轴
    for(int j = 1; j < 4; j++)                    // 哪个控制点
      cc[3 * i + j - 1] = cws[j][i] - cws[0][i];  // 坐标索引中的-1是考虑到跳过了第1个控制

  cvInvert(&CC, &CC_inv, CV_SVD);
  double * ci = cc_inv;
  for(int i = 0; i < number_of_correspondences; i++)
  {
    double * pi = pws + 3 * i;                    // pi指向第i个在世界坐标系下的3D点的首地
    double * a = alphas + 4 * i;                  // a指向第i个控制点系数alphas的首地址（

    // pi[]-cws[0][]表示去质心
    // a0,a1,a2,a3 对应的是四个控制点的齐次重心坐标
    for(int j = 0; j < 3; j++)
      // +1 是因为跳过了a0
      /*
       *
```

```
     *    [cc2 cc3 cc4] * [a2 a3 a4]^T = cp
     *  => [a2 a3 a4]^T = [cc2 cc3 cc4]^(-1) * cp
     */
    a[1 + j] = ci[3 * j    ] * (pi[0] - cws[0][0]) +
               ci[3 * j + 1] * (pi[1] - cws[0][1]) +
               ci[3 * j + 2] * (pi[2] - cws[0][2]);
  // 最后计算用于进行归一化的a0
  a[0] = 1.0f - a[1] - a[2] - a[3];//因为限制条件a[0] + a[1] + a[2] + a[3] = 1.0f所以c
 } // 遍历每一个匹配点
}
```


3. 计算控制点在相机坐标系下的坐标

3.1 公式推导

假 $\mathbf{K}$ 为相机的内参矩阵，$\{\mathbf{u}_i\}$ $i=1,\cdots n$ 是3D物方点 $\mathbf{p}_i^w$ ，$i=1,2,\cdots n$ 对应的2D像点坐标，那么根据相机投影模型可得到如下公式：

$$w_i\begin{bmatrix}\mathbf{u}_i\\1\end{bmatrix}=\mathbf{K}\mathbf{p}_i^c=\mathbf{K}\sum_{j=1}^{4}\alpha_{ij}\mathbf{c}_j^c \quad (9)$$

$$w_i\begin{bmatrix}u_i\\v_i\\1\end{bmatrix}=\begin{bmatrix}f_x&0&c_x\\0&f_y&c_y\\0&0&1\end{bmatrix}\sum_{j=1}^{4}\alpha_{ij}\begin{bmatrix}x_j^c\\y_j^c\\z_j^c\end{bmatrix} \quad (10)$$

将公式（10）展开，消掉最后一行，可得：

$$\begin{cases}\sum_{j=1}^{4}\left(\alpha_{ij}f_x x_j^c+\alpha_{ij}\left(c_x-u_i\right)z_j^c\right)=0\\ \sum_{j=1}^{4}\left(\alpha_{ij}f_y y_j^c+\alpha_{ij}\left(c_y-v_i\right)z_j^c\right)=0\end{cases} \quad (11)$$

$$\alpha_{i1}f_x x_1^c+\alpha_{i1}\left(c_x-u_i\right)z_1^c+\alpha_{i2}f_x x_2^c+\alpha_{i2}\left(c_x-u_i\right)z_2^c+\alpha_{i3}f_x x_3^c+\alpha_{i3}\left(c_x-u_i\right)\cdots$$
$$\alpha_{i1}f_y y_1^c+\alpha_{i1}\left(c_y-v_i\right)z_1^c+\alpha_{i2}f_y y_2^c+\alpha_{i2}\left(c_y-v_i\right)z_2^c+\alpha_{i3}f_y y_3^c+\alpha_{i3}\left(c_y-v_i\right)z_3^c+\cdots$$

(12)

$$\begin{bmatrix}\alpha_{11}f_x&0&\alpha_{11}\left(c_x-u_1\right)&\alpha_{12}f_x&0&\alpha_{12}\left(c_x-u_1\right)&\alpha_{13}f_x&0&\cdots\\0&\alpha_{11}f_y&\alpha_{11}\left(c_y-v_1\right)&0&\alpha_{12}f_y&\alpha_{12}\left(c_y-v_1\right)&0&\alpha_{13}f_y&\cdots\\&&&&&\vdots&&&\\&&&&&\vdots&&&\\\alpha_{n1}f_x&0&\alpha_{n1}\left(c_x-u_n\right)&\alpha_{n2}f_x&0&\alpha_{n2}\left(c_x-u_n\right)&\alpha_{n3}f_x&0&\cdots\\0&\alpha_{n1}f_y&\alpha_{n1}\left(c_y-v_n\right)&0&\alpha_{n2}f_y&\alpha_{n2}\left(c_y-v_n\right)&0&\alpha_{n3}f_y&\cdots\end{bmatrix}$$

(13)

上式中，

坐标系下的

```
/**
 * @brief 根据提供的每一对点的数据来填充矩阵 M. 每对匹配点的数据可以填充两行
 * @param[in] M                cvMat对应,存储矩阵M
 * @param[in] row              开始填充数据的行
 * @param[in] as           世界坐标系下3D点用4个虚拟控制点表达时的4个系数
 * @param[in] u                2D点坐标u
 * @param[in] v                2D点坐标v
 */
void PnPsolver::fill_M(CvMat * M,
                  const int row, const double * as, const double u, const double v)
{
  // 第一行起点
  double * M1 = M->data.db + row * 12;
  // 第二行起点
  double * M2 = M1 + 12;

  // 对每一个参考点对:
  // |ai1*fu, 0,      ai1(uc-ui),|  ai2*fu, 0,      ai2(uc-ui),|  ai3*fu, 0,      ai3(
  // |0,      ai1*fv, ai1(vc-vi),|  0,      ai2*fv, ai2(vc-vi),|  0,      ai3*fv, ai3(
  // 每一个特征点i有两行,每一行根据j=1,2,3,4可以分成四个部分,这也就是下面的for循环中所进行的工
  for(int i = 0; i < 4; i++) {
    M1[3 * i    ] = as[i] * fu;
    M1[3 * i + 1] = 0.0;
    M1[3 * i + 2] = as[i] * (uc - u);

    M2[3 * i    ] = 0.0;
    M2[3 * i + 1] = as[i] * fv;
    M2[3 * i + 2] = as[i] * (vc - v);
  }
}
```

公式(14) 中 即为4个待求的3D控制点坐标,共有12个未知数维度是12×1。 $\mathbf{M}$ 的大小为2n×12。公式（14）的解$\mathbf{x}$在 $\mathbf{M}$ 的右零空间中，所以:

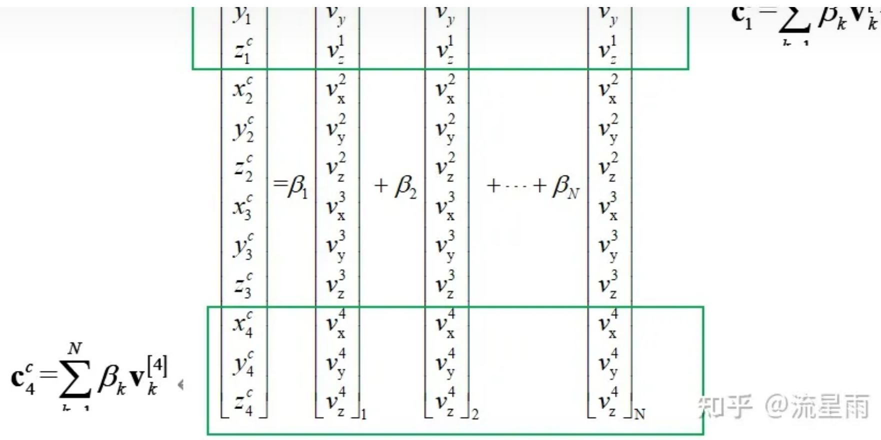
$$\mathbf{x}=\sum_{k=1}^{N}\beta_k\mathbf{v}_k \text{ (15a)}$$

$$\begin{bmatrix} x_1^c \\ y_1^c \\ z_1^c \\ x_2^c \\ y_2^c \\ z_2^c \\ x_3^c \\ y_3^c \\ z_3^c \\ x_4^c \\ y_4^c \\ z_4^c \end{bmatrix} = \beta_1 \begin{bmatrix} v_x^1 \\ v_y^1 \\ v_z^1 \\ v_x^2 \\ v_y^2 \\ v_z^2 \\ v_x^3 \\ v_y^3 \\ v_z^3 \\ v_x^4 \\ v_y^4 \\ v_z^4 \end{bmatrix}_1 + \beta_2 \begin{bmatrix} v_x^1 \\ v_y^1 \\ v_z^1 \\ v_x^2 \\ v_y^2 \\ v_z^2 \\ v_x^3 \\ v_y^3 \\ v_z^3 \\ v_x^4 \\ v_y^4 \\ v_z^4 \end{bmatrix}_2 + \cdots + \beta_N \begin{bmatrix} v_x^1 \\ v_y^1 \\ v_z^1 \\ v_x^2 \\ v_y^2 \\ v_z^2 \\ v_x^3 \\ v_y^3 \\ v_z^3 \\ v_x^4 \\ v_y^4 \\ v_z^4 \end{bmatrix}_N \text{ (15b)}$$

其中 $\mathbf{v}_k$ 为$\mathbf{M}$的第k个零特征值对应的第k个右奇异向量，维度为12×1。具体的解法为，首先求解 $\mathbf{M}^T\mathbf{M}$ 的特征值和特征向量，特征值为0的特征向量即为 $\mathbf{v}_i$ 。值得注意的是，$\mathbf{M}^T\mathbf{M}$ 的维度是12×12。计算 的复杂度为$O(n)$。因此算法的整体复杂度为$O(n)$。

上式中对于第i个控制点，我们可以采取如下的表示方式：

$$\mathbf{c}_i^c=\sum_{k=1}^{N}\beta_k\mathbf{v}_k^{[i]} \text{ (16)}$$



(16b)

上式中， $\mathbf{v}_k^{[i]}$ 是特征向量 $\mathbf{v}_k$ 的第i个3×1 sub-vector共4个。通过对 $\mathbf{M}^T\mathbf{M}$ 进行特征分解我们能够求出N个 $\mathbf{v}_k$ 。但还需要求出 $\{\beta_k\}\ k=1,2,\cdots N$ 。才能最终求出在相机坐标系下的控制点坐标。

```
// Step 4.1 先计算其中的特征向量vi
// 求M'M
cvMulTransposed(M, &MtM, 1);
// 该函数实际是特征值分解，得到特征值D，特征向量ut，对应EPnP论文式(8)中的vi
cvSVD(&MtM, &D, &Ut, 0, CV_SVD_MODIFY_A | CV_SVD_U_T);
cvReleaseMat(&M);
```

在原始论文中指出， $\mathbf{M}^T\mathbf{M}$ 特征值的个数与点对的数量以及焦距有关，EPnP算法建议只考虑N=1, 2, 3, 4的情况。

控制点在相机坐标系和世界坐标系的相对位置关系是不会发生改变的，引入相对位置约束条件：

$$\left\|\mathbf{c}_i^c-\mathbf{c}_j^c\right\|^2=\left\|\mathbf{c}_i^w-\mathbf{c}_j^w\right\|^2 \quad (17)$$

$$\left\|\sum_{k=1}^{N}\beta_k\mathbf{v}_k^{[i]}-\sum_{k=1}^{N}\beta_k\mathbf{v}_k^{[j]}\right\|^2=\left\|\mathbf{c}_i^w-\mathbf{c}_j^w\right\|^2 \quad (18)$$

$\mathbf{v}_k^{[i]}$ 的维度是3×1， $\mathbf{v}_k^{[j]}$ 的维度是3×1，分别为特征向量 $\mathbf{v}_k$ （维度为12×1）的第i和第j个分量。对于4个控制点，根据排列组合可以得到 $C_4^2=6$ 个这样的方程。1-2，1-3，1-4，2-3， 2-4，3-4。

- 第1个控制点和第2个控制点: $\left\|\sum_{k=1}^{N}\beta_k\mathbf{v}_k^{[1]}-\sum_{k=1}^{N}\beta_k\mathbf{v}_k^{[2]}\right\|^2=\left\|\mathbf{c}_1^w-\mathbf{c}_2^w\right\|^2$
- 第1个控制点和第3个控制点: $\left\|\sum_{k=1}^{N}\beta_k\mathbf{v}_k^{[1]}-\sum_{k=1}^{N}\beta_k\mathbf{v}_k^{[3]}\right\|^2=\left\|\mathbf{c}_1^w-\mathbf{c}_3^w\right\|^2$
- 第1个控制点和第4个控制点: $\left\|\sum_{k=1}^{N}\beta_k\mathbf{v}_k^{[1]}-\sum_{k=1}^{N}\beta_k\mathbf{v}_k^{[4]}\right\|^2=\left\|\mathbf{c}_1^w-\mathbf{c}_4^w\right\|^2$
- 第2个控制点和第3个控制点: $\left\|\sum_{k=1}^{N}\beta_k\mathbf{v}_k^{[2]}-\sum_{k=1}^{N}\beta_k\mathbf{v}_k^{[3]}\right\|^2=\left\|\mathbf{c}_2^w-\mathbf{c}_3^w\right\|^2$
- 第2个控制点和第4个控制点: $\left\|\sum_{k=1}^{N}\beta_k\mathbf{v}_k^{[2]}-\sum_{k=1}^{N}\beta_k\mathbf{v}_k^{[4]}\right\|^2=\left\|\mathbf{c}_2^w-\mathbf{c}_4^w\right\|^2$
- 第3个控制点和第4个控制点: $\left\|\sum_{k=1}^{N}\beta_k\mathbf{v}_k^{[3]}-\sum_{k=1}^{N}\beta_k\mathbf{v}_k^{[4]}\right\|^2=\left\|\mathbf{c}_3^w-\mathbf{c}_4^w\right\|^2$

Case N=1

$\left\|\beta \mathbf{v}^{[i]}-\beta \mathbf{v}^{[j]}\right\|^2=\left\|\mathbf{c}_i^w-\mathbf{c}_j^w\right\|^2$ (19)

$$\beta=\frac{\sum\limits_{\substack{i=1,1,1,2,2,3\\ j=2,3,4,3,4,4}}\left\|\mathbf{v}^{[i]}-\mathbf{v}^{[j]}\right\|\cdot\left\|\mathbf{c}_i^w-\mathbf{c}_j^w\right\|}{\sum\limits_{\substack{i=1,1,1,2,2,3\\ j=2,3,4,3,4,4}}\left\|\mathbf{v}^{[i]}-\mathbf{v}^{[j]}\right\|^2}$$

(20)

Case N=2

当N=2时，公式 (15a) 就变成了最简单的形式 $\mathbf{x}=\sum\limits_{k=1}^{N}\beta_k\mathbf{v}_k \rightarrow \mathbf{x}=\beta_1\mathbf{v}_1+\beta_2\mathbf{v}_2$ ，带入公式(18)可得：

$\left\|\left(\beta_1\mathbf{v}_1^{[i]}+\beta_2\mathbf{v}_2^{[i]}\right)-\left(\beta_1\mathbf{v}_1^{[j]}+\beta_2\mathbf{v}_2^{[j]}\right)\right\|^2=\left\|\mathbf{c}_i^w-\mathbf{c}_j^w\right\|^2$ (21a)

将公式(21)展开

$\left\|\beta_1\underbrace{\left(\mathbf{v}_1^{[i]}-\mathbf{v}_1^{[j]}\right)}_{\mathbf{S}_1}+\beta_2\underbrace{\left(\mathbf{v}_2^{[i]}-\mathbf{v}_2^{[j]}\right)}_{\mathbf{S}_2}\right\|^2=\underbrace{\left\|\mathbf{c}_i^w-\mathbf{c}_j^w\right\|^2}_{c}$ (21b)

\[\begin{array}{l} {\kern 1pt} {\left\| {{\beta _1}{{\bf{S}}_1}{\rm{ + }}{\beta _2}{{\bf{S}}_2}} \right\|^2} = c\\ {\beta _1}^2{{\bf{S}}_1}^T{{\bf{S}}_1} + 2{\beta _1}{\beta _2}{{\bf{S}}_1}^T{{\bf{S}}_2}{\rm{ + }}{\beta _2}^2{{\bf{S}}_2}^T{{\bf{S}}_2} = c \end{array}\] (22)

其中 \[{\bf{S}}\] 的维度为3×1，将\[{\left\| {{\bf{c}}_i^w{\rm{ - }}{\bf{c}}_j^w} \right\|^2}\]展开后用 表示维度为1×1，为标量。引入3个中间变量 \[{\beta _1}_1{\rm{ = }}{\beta _1}^2{\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\beta _1}_2{\rm{ = }}{\beta _1}{\beta _2}{\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\beta _2}_2{\rm{ = }}{\beta _2}^2\] 带入公式 (22)。

\[{\beta _{11}}{{\bf{S}}_1}^T{{\bf{S}}_1} + 2{\beta _1}_2{{\bf{S}}_1}^T{{\bf{S}}_2}{\rm{ + }}{\beta _{22}}{{\bf{S}}_2}^T{{\bf{S}}_2} = c\] (23)

\[\left[{\begin{array}{*{20}{c}} {{{\bf{S}}_1}^T{{\bf{S}}_1}}&{2{{\bf{S}}_1}^T{{\bf{S}}_2}}&{{{\bf{S}}_2}^T{{\bf{S}}_2}} \end{array}} \right]\left[{\begin{array}{*{20}{c}} {{\beta _{11}}}\\ {{\beta _1}_2}\\ {{\beta _{22}}} \end{array}} \right] = c\] (24)

\[{{\bf{S}}_1}^T{{\bf{S}}_1}\] 的维度1×1为标量。将上式用一下公式表示

因为有6个方程式所以，\[{\bf{L}}\] 的维度6×3， \[{\bf{\beta }}\] 的维度3×1, \[{\bf{\rho }}\] 的维度为6×1。上式子为线性非齐次方程，通过SVD分解，即可求得\[{\bf{\beta }}\]。考虑到控制点在相机的前方，所以坐标的分量 应该要 \[z\] 大于0，从而 \[{\beta _1}\] ，\[{\beta _2}\]可以求得唯一解。

Case N=3

当N=3时， \[{\bf{x}}{\rm{ = }}\sum\limits_{k = 1}^N {{\beta _k}{{\bf{v}}_k}} \] → \[{\bf{x}}{\rm{ = }}{\beta _1}{{\bf{v}}_1}{\rm{ + }}{\beta _2}{{\bf{v}}_2}{\rm{ + }}{\beta _3}{{\bf{v}}_3}\] 。类似公式(22)，容易写出一下表达式：

\[\begin{array}{l} {\kern 1pt} {\left\| {{\beta _1}{{\bf{S}}_1}{\rm{ + }}{\beta _2}{{\bf{S}}_2}{\rm{ + }}{\beta _3}{{\bf{S}}_3}} \right\|^2} = c\\ {\beta _1}^2{{\bf{S}}_1}^T{{\bf{S}}_1} + 2{\beta _1}{\beta _2}{{\bf{S}}_1}^T{{\bf{S}}_2}{\rm{ + }}2{\beta _1}{\beta _3}{{\bf{S}}_1}^T{{\bf{S}}_3}{\rm{ + }}{\beta _2}^2{{\bf{S}}_2}^T{{\bf{S}}_2}{\rm{ + }}2{\beta _2}{\beta _3}{{\bf{S}}_2}^T{{\bf{S}}_3}{\rm{ + }}{\beta _3}^2{{\bf{S}}_3}^T{{\bf{S}}_3} = c \end{array}\]

(26)

引入6个中间变量 \[{\beta _1}_1{\rm{ = }}{\beta _1}^2{\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\beta _1}_2{\rm{ = }}{\beta _1}{\beta _2} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\beta _1}_3{\rm{ = }}{\beta _1}{\beta _3}{\kern 1pt} \] \[{\beta _2}_2{\rm{ = }}{\beta _2}^2{\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\beta _2}_3{\rm{ = }}{\beta _2}{\beta _3}{\kern 1pt} {\kern 1pt} \] \[{\beta _{33}}{\rm{ = }}{\beta _3}{\beta _3}\] 带入公式 (22)。

\[\begin{array}{l} \left[{\begin{array}{*{20}{c}} {{{\bf{S}}_1}^T{{\bf{S}}_1}}& {2{{\bf{S}}_1}^T{{\bf{S}}_2}}&{2{{\bf{S}}_1}^T{{\bf{S}}_3}}&{{{\bf{S}}_2}^T{{\bf{S}}_2}}& {2{{\bf{S}}_2}^T{{\bf{S}}_3}}&{{{\bf{S}}_3}^T{{\bf{S}}_3}} \end{array}} \right]\left[{\begin{array}{*{20}{c}} {{\beta _1}_1}\\ {{\beta _1}_2}\\ {{\beta _1}_3}\\ {{\beta _2}_2}\\ {{\beta _2}_3}\\ {{\beta _{33}}} \end{array}} \right] = c\\ {\kern 1pt} {\bf{L\beta }} = {\bf{\rho }} \end{array}\] (27)

因为有6个方程式,所以 \[{\bf{L}}\] 的维度6×6， \[{\bf{\beta }}\] 的维度6×1, \[{\bf{\rho }}\] 的维度为6×1。

当N=3时，\[{\bf{x}}{\rm{ = }}\sum\limits_{k = 1}^N {{\beta _k}{{\bf{v}}_k}} \]→\[{\bf{x}}{\rm{ = }}{\beta _1}{{\bf{v}}_1}{\rm{ + }}{\beta _2}{{\bf{v}}_2}{\rm{ + }}{\beta _3}{{\bf{v}}_3}{\rm{ + }}{\beta _4}{{\bf{v}}_4}\]。

\[{\left\| {{{\bf{x}}^{[i]}} - {{\bf{x}}^{[j]}}} \right\|^2} = {\left\| {{\bf{c}}_i^w - {\bf{c}}_j^w} \right\|^2}\] (28)

\[{\left\| {\left({{\beta _1}{\bf{v}}_1^{\left[i \right]}{\rm{ + }}{\beta _2}{\bf{v}}_2^{\left[i \right]}{\rm{ + }}{\beta _3}{\bf{v}}_3^{\left[i \right]}{\rm{ + }}{\beta _4}{\bf{v}}_4^{\left[i \right]}} \right) - \left({{\beta _1}{\bf{v}}_1^{\left[j \right]}{\rm{ + }}{\beta _2}{\bf{v}}_2^{\left[j \right]}{\rm{ + }}{\beta _3}{\bf{v}}_3^{\left[j \right]}{\rm{ + }}{\beta _4}{\bf{v}}_4^{\left[j \right]}} \right)} \right\|^2} = {\left\| {{\bf{c}}_i^w - {\bf{c}}_j^w} \right\|^2}\]

(29)

\[{\left\| {{\beta _1}\left({{\bf{v}}_1^{\left[i \right]} - {\bf{v}}_1^{\left[j \right]}} \right){\rm{ + }}{\beta _2}\left({{\bf{v}}_2^{\left[i \right]} - {\bf{v}}_2^{\left[j \right]}} \right){\rm{ + }}{\beta _3}\left({{\bf{v}}_3^{\left[i \right]} - {\bf{v}}_3^{\left[j \right]}} \right){\rm{ + }}{\beta _4}\left({{\bf{v}}_4^{\left[i \right]} - {\bf{v}}_4^{\left[j \right]}} \right)} \right\|^2} = {\left\| {{\bf{c}}_i^w - {\bf{c}}_j^w} \right\|^2}\]

(30)

$$\left\|\beta_1\mathbf{S}_1+\beta_2\mathbf{S}_2+\beta_3\mathbf{S}_3+\beta_4\mathbf{S}_4\right\|^2=c \quad (31)$$

$$\begin{array}{l}
{\beta_1}^2{\mathbf{S}_1}^T\mathbf{S}_1+2\beta_1\beta_2{\mathbf{S}_1}^T\mathbf{S}_2+2\beta_1\beta_3{\mathbf{S}_1}^T\mathbf{S}_3+2\beta_1\beta_4{\mathbf{S}_1}^T\mathbf{S}_4\\
+{\beta_2}^2{\mathbf{S}_2}^T\mathbf{S}_2+2\beta_2\beta_3{\mathbf{S}_2}^T\mathbf{S}_3+2\beta_2\beta_4{\mathbf{S}_2}^T\mathbf{S}_4\\
+{\beta_3}^2{\mathbf{S}_3}^T\mathbf{S}_3+2\beta_3\beta_4{\mathbf{S}_3}^T\mathbf{S}_4\\
+{\beta_4}^2{\mathbf{S}_4}^T\mathbf{S}_4=c
\end{array} \quad (32)$$

$$[{\mathbf{S}_1}^T\mathbf{S}_1 \quad 2{\mathbf{S}_1}^T\mathbf{S}_2 \quad 2{\mathbf{S}_1}^T\mathbf{S}_3 \quad 2{\mathbf{S}_1}^T\mathbf{S}_4 \quad {\mathbf{S}_2}^T\mathbf{S}_2 \quad 2{\mathbf{S}_2}^T\mathbf{S}_3 \quad 2{\mathbf{S}_2}^T\mathbf{S}_4 \quad {\mathbf{S}_3}^T\mathbf{S}_3 \quad 2\mathbf{S}$$

$$\mathbf{L}\beta=\rho$$

(33)

上式是根据第i个控制点和第j个控制点写出的方程。4个控制点选出2个控制点，$C_4^2=6$可以列出6个上述方程。因此，$\mathbf{L}$的维度6×10，β的维度10×1，ρ的维度为10×1。

```
/**
 * @brief 计算矩阵L,论文式13中的L矩阵,不过这里的是按照N=4的时候计算的
 *
 * @param[in]  ut               特征值分解之后得到的12x12特征矩阵
 * @param[out] l_6x10           计算的L矩阵结果，维度6x10
 */
void PnPsolver::compute_L_6x10(const double * ut, double * l_6x10)
{
  // Ste
  const
```

```
  // 以这里的v[0]为例，它是12x1的向量，会拆成4个3x1的向量v[0]^[0]，v[0]^[1]，v[0]^[1]，v[0]
  v[0] = ut + 12 * 11;    // v[0] : v[0][0]~v[0][2]  => v[0]^[0]  , * \beta_0 = c0  (理
                          //        v[0][3]~v[0][5]  => v[0]^[1]  , * \beta_0 = c1
                          //        v[0][6]~v[0][8]  => v[0]^[2]  , * \beta_0 = c2
                          //        v[0][9]~v[0][11] => v[0]^[3]  , * \beta_0 = c3
  v[1] = ut + 12 * 10;
  v[2] = ut + 12 *  9;
  v[3] = ut + 12 *  8;

  // Step 2 提前计算中间变量dv
  // dv表示中间变量，是difference-vector的缩写
  // 4 表示N=4时对应的4个12x1的向量v, 6 表示4对点一共有6种两两组合的方式，3 表示v^[i]是一个3维
  double dv[4][6][3];


  // N=4时候的情况. 控制第一个下标的就是a,第二个下标的就是b,不过下面的循环中下标都是从0开始的
  for(int i = 0; i < 4; i++)
  {
    // 每一个向量v[i]可以提供四个控制点的"雏形"v[i]^[0]~v[i]^[3]
    // 这四个"雏形"两两组合一共有六种组合方式:
    // 下面的a变量就是前面的那个id,b就是后面的那个id
    int a = 0, b = 1;
    for(int j = 0; j < 6; j++)
        {
      // dv[i][j]=v[i]^[a]-v[i]^[b]
      // a,b的取值有6种组合 0-1 0-2 0-3 1-2 1-3 2-3
      dv[i][j][0] = v[i][3 * a    ] - v[i][3 * b    ];
      dv[i][j][1] = v[i][3 * a + 1] - v[i][3 * b + 1];
      dv[i][j][2] = v[i][3 * a + 2] - v[i][3 * b + 2];

      b++;
      if (b > 3)
          {
        a++;
        b = a + 1;
      }
    }
  }

  // Step 3 用前面计算的dv生成L矩阵
  // 这里的6代表前面每个12x1维向量v的4个3x1子向量v^[i]对应的6种组合
  for(int i = 0; i < 6; i++)
  {
    double * row = l_6x10 + 10 * i;
    // 计算每一行中的每一个元素,总共是10个元素      // 对应的\beta列向量
    row[0] =        dot(dv[0][i], dv[0][i]);  //*b11
    row[1] = 2.0f * dot(dv[0][i], dv[1][i]);  //*b12
    row[2] =        dot(dv[1][i], dv[1][i]);  //*b22
    row[3] = 2.0f * dot(dv[0][i], dv[2][i]);  //*b13
    row[4] = 2.0f * dot(dv[1][i], dv[2][i]);  //*b23
    row[5] =        dot(dv[2][i], dv[2][i]);  //*b33
    row[6] = 2.0f * dot(dv[0][i], dv[3][i]);  //*b14
    row[7] = 2.0f * dot(dv[1][i], dv[3][i]);  //*b24
    row[8] = 2.0f * dot(dv[2][i], dv[3][i]);  //*b34
    row[9] =        dot(dv[3][i], dv[3][i]);  //*b44
  }
}



/**
 * @brief 计算四个控制点任意两点间的距离，总共6个距离，对应论文式13中的向量\rho
 * @param[in] rho  计算结果
 */
void PnP
```

```
    rho[0] = dist2(cws[0], cws[1]);
    rho[1] = dist2(cws[0], cws[2]);
    rho[2] = dist2(cws[0], cws[3]);
    rho[3] = dist2(cws[1], cws[2]);
    rho[4] = dist2(cws[1], cws[3]);
    rho[5] = dist2(cws[2], cws[3]);
  }
```


上式中有10个未知数，但是只有6个方程，在OpenCV中开源代码并没按照上述4个Case中的方法去求解，而是采用了近似的解法，具体的可以去看一下源码。

值得说明的是，在代码中$\mathbf{L}$ 和β 的排序有点不同，但不影响求解只要 $\mathbf{L}$ 和β 的顺序对应即可，以β 为例说明。

本文中 $\mathbf{L}$ 元素的排序是：
$[\mathbf{S_1}^T\mathbf{S_1} \quad 2\mathbf{S_1}^T\mathbf{S_2} \quad 2\mathbf{S_1}^T\mathbf{S_3} \quad 2\mathbf{S_1}^T\mathbf{S_4} \quad \mathbf{S_2}^T\mathbf{S_2} \quad 2\mathbf{S_2}^T\mathbf{S_3} \quad 2\mathbf{S_2}^T\mathbf{S_4} \quad \mathbf{S_3}^T\mathbf{S_3} \quad 2\mathbf{S_3}^T\mathbf{S_4} \quad \mathbf{S_4}^T\mathbf{S_4}]$
而OpenCV源码中$\mathbf{L}$ 排序：
$[\mathbf{S_1}^T\mathbf{S_1} \quad 2\mathbf{S_1}^T\mathbf{S_2} \quad 2\mathbf{S_2}^T\mathbf{S_2} \quad 2\mathbf{S_1}^T\mathbf{S_3} \quad \mathbf{S_2}^T\mathbf{S_3} \quad 2\mathbf{S_3}^T\mathbf{S_3} \quad 2\mathbf{S_1}^T\mathbf{S_4} \quad \mathbf{S_2}^T\mathbf{S_4} \quad 2\mathbf{S_3}^T\mathbf{S_4} \quad \mathbf{S_4}^T\mathbf{S_4}]$

OpenCV的解法：

因为 $\beta_{11}\ \beta_{12}\ \beta_{13}\ \beta_{14}\ \beta_{22}\ \beta_{23}\ \beta_{24}\ \beta_{33}\ \beta_{34}\ \beta_{44}$ 这10个未知数是相关的，所以我们只需求出$\beta_{11}\beta_{12}\beta_{13}\beta_{14}$，就能从中解出$\beta_1\ \beta_2\beta_3\ \beta_4$的值。

在OpenCV的源码中取 的0,1,3,6列组成了新的矩阵L_6x4，然后进行SVD分解。由于，我们写出的公式跟代码列出的公式顺序不一样。因此我们对应的选择 的0,1,2,3列组成了新的矩阵L_6x4。

$$[\mathbf{S_1}^T\mathbf{S_1} \quad 2\mathbf{S_1}^T\mathbf{S_2} \quad 2\mathbf{S_1}^T\mathbf{S_3} \quad 2\mathbf{S_1}^T\mathbf{S_4}]\begin{bmatrix}\beta_{11}\\ \beta_{12}\\ \beta_{13}\\ \beta_{14}\end{bmatrix} = c \ (34)$$

上述方程可以列出6个，所以表示如下：

$$\mathbf{L}_{6\times4}\beta_{4\times1} = \rho_{6\times1} \ (35)$$

公式(35)中6个方程4个未知数，通过SVD最小二乘分解即可求得 $\beta_{11}\beta_{12}\beta_{13}\beta_{14}$，然后可以计算得出 $\beta_1\ \beta_2\beta_3\beta_4$。

```
/**
  * @brief 计算N=4时候的粗糙近似解，暴力将其他量置为0
  *
  * @param[in]  L_6x10  矩阵L
  * @param[in]  Rho     非齐次项 \rho, 列向量
  * @param[out] betas   计算得到的beta
  */
void PnPsolver::find_betas_approx_1(const CvMat * L_6x10, const CvMat * Rho,
                                    double * betas)
{
  // 计算N=4时候的粗糙近似解，暴力将其他量置为0
  // betas10        = [B11 B12 B22 B13 B23 B33 B14 B24 B34 B44]  -- L_6x10中每一行的内容
  // betas_approx_1 = [B11 B12     B13         B14            ]  -- L_6x4 中一行提取出来

  double l_6x4[6 * 4], b4[4];
  CvMat L_6x4 = cvMat(6, 4, CV_64F, l_6x4);
  CvMat B4    = cvMat(4, 1, CV_64F, b4);

  // 提取L_6x10矩阵中每行的第0,1,3,6个元素，得到L_6x4
  for(int i = 0; i < 6; i++)
  {
    cvmS
```

```
    cvmSet(&L_6x4, i, 3, cvmGet(L_6x10, i, 6));
  }

  // SVD方式求解方程组 L_6x4 * B4 = Rho
  cvSolve(&L_6x4, Rho, &B4, CV_SVD);
  // 得到的解是 b00 b01 b02 b03 因此解出来b00即可
  if (b4[0] < 0)
  {
    betas[0] = sqrt(-b4[0]);
    betas[1] = -b4[1] / betas[0];
    betas[2] = -b4[2] / betas[0];
    betas[3] = -b4[3] / betas[0];
  }
  else
  {
    betas[0] = sqrt(b4[0]);
    betas[1] = b4[1] / betas[0];
    betas[2] = b4[2] / betas[0];
    betas[3] = b4[3] / betas[0];
  }
}
```


3.2 Gauss-Newton优化参数β

优化的目标：缩小两个坐标系下控制点间距差。

优化的目标函数：

$$Error(\beta)_{ij} = \left\|c_i^c - c_j^c\right\|^2 - \left\|c_i^w - c_j^w\right\|^2 \quad (36)$$

$$\beta^* = \arg\min_{\beta} \sum_{(i,j)s.t.i<j} \left\|Error_{ij}(\beta)\right\|^2 \quad (37)$$

这是一个无约束的非线性最优化问题，根据高翔老师视觉SLAM十四讲（第二版）P129所讲的Gauss-Newton求解式，首先求解\[Error\left(\beta \right)\]相对于\[\beta \] 的雅克比矩阵。

\[\begin{array}{l} Erro{r_{ij}}\left({\bf{\beta }} \right){\rm{ = }}{\kern 1pt} {\beta _1}^2{{\bf{S}}_1}^T{{\bf{S}}_1} + 2{\beta _1}{\beta _2}{{\bf{S}}_1}^T{{\bf{S}}_2}{\rm{ + }}2{\beta _1}{\beta _3}{{\bf{S}}_1}^T{{\bf{S}}_3}{\rm{ + }}2{\beta _1}{\beta _4}{{\bf{S}}_1}^T{{\bf{S}}_4}\\ {\kern 1pt} {\rm{ + }}{\beta _2}^2{{\bf{S}}_2}^T{{\bf{S}}_2}{\rm{ + }}2{\beta _2}{\beta _3}{{\bf{S}}_2}^T{{\bf{S}}_3}{\rm{ + }}2{\beta _2}{\beta _4}{{\bf{S}}_2}^T{{\bf{S}}_4}\\ {\kern 1pt} {\rm{ + }}{\beta _3}^2{{\bf{S}}_3}^T{{\bf{S}}_3}{\rm{ + }}2{\beta _3}{\beta _4}{{\bf{S}}_3}^T{{\bf{S}}_4}\\ {\kern 1pt} {\rm{ + }}{\beta _4}^2{{\bf{S}}_4}^T{{\bf{S}}_4}\\ {\kern 1pt} {\ker

1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} {\kern 1pt} - {c^w} \end{array}\] (38)

$$\mathbf{J}_{ij} = \frac{\partial[Error_{ij}(\beta)]}{\partial\beta} = \begin{bmatrix} 2\beta_1 \mathbf{S_1}^T\mathbf{S}_1 + 2\beta_2 \mathbf{S_1}^T\mathbf{S}_2 + 2\beta_3 \mathbf{S_1}^T\mathbf{S}_3 + 2\beta_4 \mathbf{S_1}^T\mathbf{S}_4 \\ 2\beta_1 \mathbf{S_1}^T\mathbf{S}_2 + 2\beta_2 \mathbf{S_2}^T\mathbf{S}_2 + 2\beta_3 \mathbf{S_2}^T\mathbf{S}_3 + 2\beta_4 \mathbf{S_2}^T\mathbf{S}_4 \\ 2\beta_1 \mathbf{S_1}^T\mathbf{S}_3 + 2\beta_2 \mathbf{S_2}^T\mathbf{S}_3 + 2\beta_3 \mathbf{S_3}^T\mathbf{S}_3 + 2\beta_4 \mathbf{S_3}^T\mathbf{S}_4 \\ 2\beta_1 \mathbf{S_1}^T\mathbf{S}_4 + 2\beta_2 \mathbf{S_2}^T\mathbf{S}_4 + 2\beta_3 \mathbf{S_3}^T\mathbf{S}_4 + 2\beta_4 \mathbf{S_4}^T\mathbf{S}_4 \end{bmatrix}^T \quad (39)$$

$\mathbf{J}_{ij}$ 的维度为1×4，将6个小雅克比矩阵 $\mathbf{J}_{ij}$ 合成为6×4的大雅克比矩阵

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_{12} \\ \mathbf{J}_{13} \\ \mathbf{J}_{14} \\ \mathbf{J}_{23} \\ \mathbf{J}_{24} \\ \mathbf{J}_{34} \end{bmatrix} \quad (40)$$

记残差为

$$\mathbf{e} = \mathbf{Error} = \begin{bmatrix} Error_{12}(\beta) \\ Error_{13}(\beta) \\ Error_{14}(\beta) \\ Error_{23}(\beta) \\ Error_{24}(\beta) \\ Error_{34}(\beta) \end{bmatrix} \quad (41)$$

增量方程为

$$\mathbf{J}^T\mathbf{J}\delta\beta = -\mathbf{J}^T\mathbf{e} \quad (42)$$

$\delta\beta$的求解在OpenCV中没有采用$\delta\beta = -\left(\mathbf{J}^T\mathbf{J}\right)^{-1}\mathbf{J}^T\mathbf{e}$的方式求解，而是对$\mathbf{J}\delta\beta = -\mathbf{e}$QR分解，从而求得$\delta\beta$。

因为$\mathbf{J}$是一个超定矩阵(over-determined)，对于over-determined的线性最小二乘问题，正规方程组$\mathbf{J}\delta\beta = -\mathbf{e}$是不稳定的，通常需要用QR分解来处理：

更新 β

$$\beta := \beta + \delta\beta \quad (43)$$

```
/**
 * @brief 对计算出来的Beta结果进行高斯牛顿法优化,求精. 过程参考EPnP论文中式(15)
 *
 * @param[in] L_6x10
 * @param[in] Rho
 * @param[in] betas
 */
void PnPsolver::gauss_newton(const CvMat * L_6x10, const CvMat * Rho,
                             double betas[4])
{
  // 只进行5次迭代
  const int iterations_number = 5;

  /** 这
```

```
   * 而根据高斯牛顿法,增量方程为:
   * \f$ \mathbf{H}\mathbf{\Delta x}=\mathbf{g} \f$
   * 也就是:(参考视觉SLAM十四讲第一版P112式6.21 6.22)
   * \f$ \mathbf{J}^T\mathbf{J}\mathbf{\Delta x}=-\mathbf{J}^T f(x) \f$
   * 不过这里在计算的时候将等式左右两边的雅克比 \f$ \mathbf{J}^T \f$ 都给约去了,得到精简后的增
   * \f$  \mathbf{J}\mathbf{\Delta x}=-f(x) \f$
   * 然后分别对应为程序代码中的系数矩阵A和非齐次项B.
   */
  double a[6*4], b[6], x[4];
  CvMat A = cvMat(6, 4, CV_64F, a);   // 系数矩阵
  CvMat B = cvMat(6, 1, CV_64F, b);   // 非齐次项
  CvMat X = cvMat(4, 1, CV_64F, x);   // 增量，待求量

  // 对于每次迭代过程
  for(int k = 0; k < iterations_number; k++)
  {
    // 计算增量方程的系数矩阵和非齐次项
    compute_A_and_b_gauss_newton(L_6x10->data.db, Rho->data.db,
                                 betas, &A, &B);
    // 使用QR分解来求解增量方程,解得当前次迭代的增量X
    qr_solve(&A, &B, &X);

    // 应用增量,对估计值进行更新;估计值是beta1~beta4组成的向量
    for(int i = 0; i < 4; i++)
      betas[i] += x[i];
  }
}
```


至此我们通过Case N=1, Case N=2, Case N=3, Case N=4可以确定四组 β 和 $\mathbf{v}$ 。

- Case N=1：求得β和 $\mathbf{v}$
- Case N=2：求得$\beta_1\mathbf{v}_1$ 和 $\beta_2\mathbf{v}_2$
- Case N=3：求得$\beta_1\mathbf{v}_1$ ， $\beta_2\mathbf{v}_2$和 $\beta_3\mathbf{v}_3$
- Case N=4：求得$\beta_1\mathbf{v}_1$ ， $\beta_2\mathbf{v}_2$ ， $\beta_3\mathbf{v}_3$ 和 $\beta_4\mathbf{v}_4$

最后通过公式(15a) $\mathbf{x}=\sum_{k=1}^{N}\beta_k\mathbf{v}_k$ 计算控制点在相机坐标系下的坐标。具体选择哪种情况需要根据恢复影像的外方位元素后，计算的反投影误差决定。

五、求解 $\mathbf{R}$ 和 $\mathbf{t}$

在相机坐标系下的控制点 $\mathbf{x}$ 已知，每个物方点的对应4个HB坐标$\alpha_{i1}\,\alpha_{i2}\,\alpha_{i3}\,\alpha_{i4}$已知，那么我们可以根据公式(4b)$\mathbf{p}_i^c=\sum_{j=1}^{4}\alpha_{ij}\mathbf{c}_j^c$求出3D物方点在相机坐标系下的坐标。

至此我们需要的条件都已得到满足，因此可以通过ICP求解出两组点云的$\mathbf{R}$和$\mathbf{t}$，即影像的姿态。

Step1: 计算点云质心坐标

$$\mathbf{p}_c^c=\frac{1}{n}\sum_{i=1}^{n}\mathbf{p}_i^c \quad (44)$$

$$\mathbf{p}_c^w=\frac{1}{n}\sum_{i=1}^{n}\mathbf{p}_i^w \quad (45)$$

Step2: 点云去质心

$$\bar{\mathbf{P}}^c=\begin{bmatrix}(\mathbf{p}_1^c)^T-(\mathbf{p}_c^c)^T\\ \cdots\\ (\end{bmatrix} \quad (46)$$

$$\left[(\mathbf{p}_n^w)^T - (\mathbf{p}_c^w)^T\right]$$

Step3: 计算旋转矩阵$\mathbf{R}$

$$\mathbf{W} = \left(\bar{\mathbf{P}}^c\right)^T \bar{\mathbf{P}}^w \quad (48)$$

$$[\mathbf{U}\Sigma\mathbf{V}] = SVD(\mathbf{W}) \quad (49)$$

$$\mathbf{R} = \mathbf{U}\mathbf{V}^T \quad (50)$$

如果 $det(\mathbf{R}) < 0, \mathbf{R}(2,:) = -\mathbf{R}(2,:)$

Step4: 计算平移向量t

$$\mathbf{t} = \mathbf{p}_c^c - \mathbf{R}\mathbf{p}_c^w \quad (51)$$

注意上面提到 有四种情况的解，所以根据上面四种情况我们求得了四种情况的$\mathbf{R}$和$\mathbf{t}$。根据$\mathbf{R}$和$\mathbf{t}$计算反投影误差。采用反投影误差最小的求解结果。

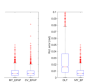
```cpp
/**
 * @brief 根据已经得到的控制点在当前相机坐标系下的坐标来恢复出相机的位姿
 * @param[in]  ut         vi
 * @param[in]  betas      betas
 * @param[out] R          计算得到的相机旋转R
 * @param[out] t          计算得到的相机位置t
 * @return double         使用这个位姿,所得到的重投影误差
 */
double PnPsolver::compute_R_and_t(const double * ut, const double * betas,
                                  double R[3][3], double t[3])
{
  // Step 1 根据前面的计算结果来"组装"得到控制点在当前相机坐标系下的坐标
  compute_ccs(betas, ut);
  // Step 2 将世界坐标系下的3D点的坐标转换到控制点的坐标系下
  compute_pcs();
  // Step 3 调整点坐标的符号,来保证在相机坐标系下点的深度为正
  solve_for_sign();

  // Step 4 ICP计算R和t
  estimate_R_and_t(R, t);

  // Step 5 计算使用这个位姿,所得到的每对点平均的重投影误差,作为返回值
  return reprojection_error(R, t);
}
```

六、参考文献

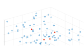
EPnP_ An Accurate O(n) Solution to the PnP Problem
www.researchgate.net/profile/Pascal-Fua/publication/...

Least-Squares Rigid Motion Using SVD
igl.ethz.ch/projects/ARAP/svd_rot.pdf

小葡萄：[PnP]PnP问题之EPnP解法
242 赞同 · 19 评论 文章



深入EPnP算法_我就是Jesse-CSDN博客



13 条评论 默认 最新

刘强
想请教一个问题，现在很多位姿估计的方法都是找到2d关键点，那么是如何确定哪个3d点与之对应的呢?
01-07 回复 1

鹗萧萧
det(R<0)的处理方法好像不对吧？应该是将矩阵V或者U的最后一列取反，参考论文：[1] Arun K S, Huang T S, Blostein S D. Least-Squares Fitting of Two 3-D Point Sets[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1987, PAMI-9(5): 698-700.
2022-12-31 回复 喜欢

鹗萧萧
(34)式，也就是OpenCV的EPnP在N=4下的求解方法是不是一个近似解而非精确解?
2022-11-23 回复 喜欢

鹗萧萧
式33的ρ的维度应该是6×1吧
2022-11-23 回复 喜欢

Jayden
N=1是β的求解公式怎么出来的
2022-10-21 回复 喜欢

> **叨叨叨**
> 请问您明白了吗😘
> 08-21 回复 喜欢

zhchyang2004
博主,好!源码求beta那里find_betas_approx_1, 为什么使用svd求解会出现b4[0]<0的情况? beta11等于beta1*beta1, 并且Rho是等于世界系控制点差的模方. 不应该出现b4[0]<0的情况吧?
2022-10-20 回复 喜欢

Cogman
精辟，到位!
2021-04-02 回复 喜欢

张欣
公式34中的c是什么？和$\beta{22}$、...、$\beta{44}$有关吗?
2022-06-20 回复 喜欢

chun丶
1b的等式怎么来的?有点不明白
2022-06-09 回复 喜欢

cp=p_i-c1
cp=a1(c1-c1)+a2(c2-c1)+a3(c3-c1)+a4(c4-c1),
所以:
p_i=c1+a2(c2-c1)+a3(c3-c1)+a4(c4-c1) = (1-a2-a3-a4)c1+a2*c2+a3*c3+a4*c4
c1前面的系数就是a1=1-a2-a3-a4. 即(1b)式.
从另一个角度, cp就是由三个正交基底v1,v2,v3进行线性组合的结果, 线性组合系数对应(8)式, 分别是a2*lamda1**.5, a3*lamda2**.5, a4*lamda3**.5. 由此也能推出博主的注释.

2022-10-20 回复 喜欢

点击查看全部评论 >

已赞同 49

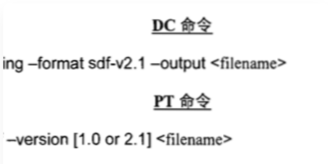
发布一条带图评论吧

牛啊

2022-05-08 回复 喜欢

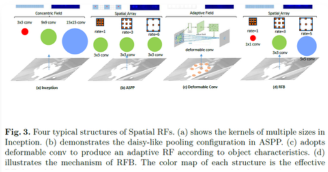
SLAM

推荐阅读



SDF生成—为动态时序仿真

Osris 发表于veril...

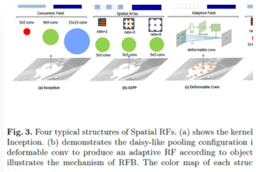


RFBNet(4)_源码(2)_ECCV2018

胡孟 发表于目标检测

Classic DH参数与Modified DH参数

薛定谔的猫



RFBNet(4)_源码(4)_ECCV2018

胡孟