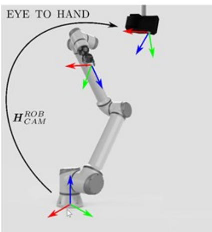
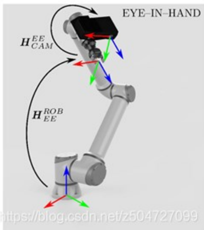
3D视觉工坊-手眼标定（附opencv实现代码）

原创 z504727099 于 2021-04-07 20:25:39 发布 6017 收藏 95 版权
分类专栏： 机械臂 文章标签： opencv 3d 计算机视觉
机械臂 专栏收录该内容 6 订阅 1 篇文章 订阅专栏

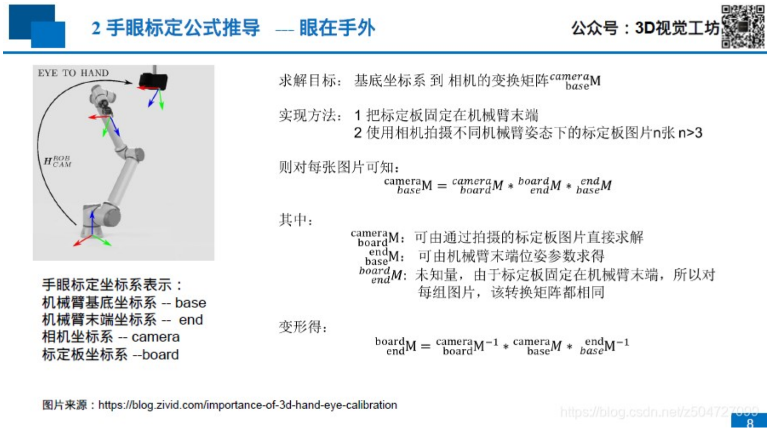
1.基本介绍

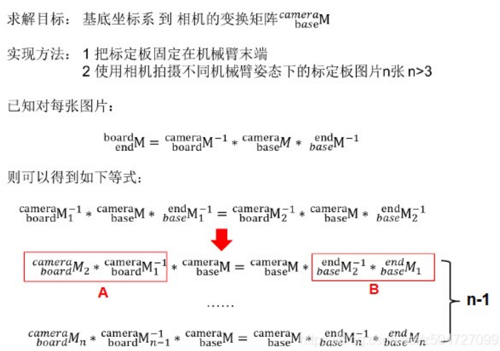
手眼标定两种形式
眼在手外 eye to hand
眼在手上 eye in hand





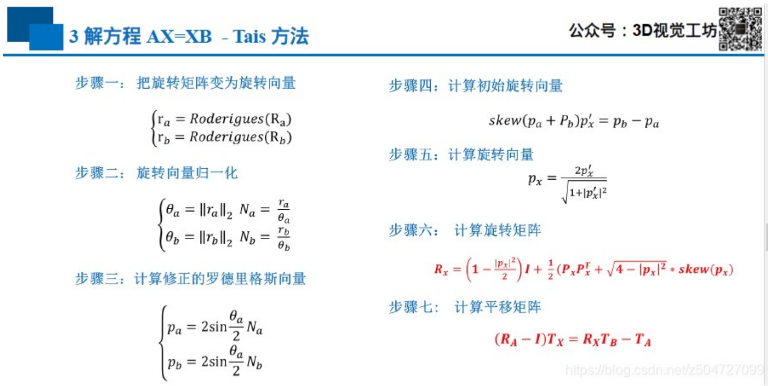
2.公式推导

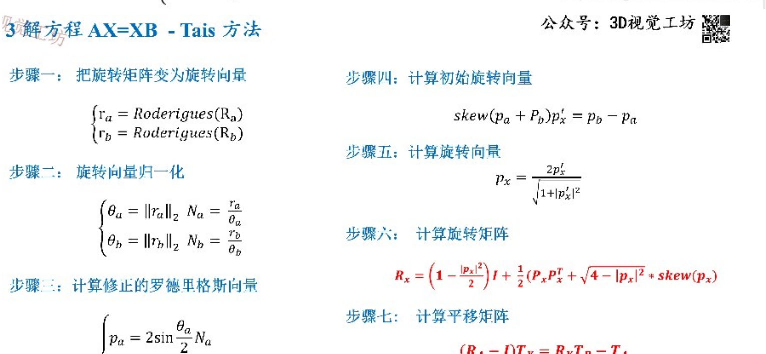




眼在手上类似

3.方程AX=XB求解





目录

$$p_b = 2\sin\frac{\theta_a}{2}N_b$$

假设一个向量a为:

$$a = (a_1, a_2, a_3)$$

则a的反对称矩阵A:

$$A = skew(a) = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}$$

4.opencv完成手眼标定

眼在手上
1.Rend2base机械臂末端到基点的变换矩阵，可从示教器或者在ROS直接订阅相关tf
2.Rboard2cam 标定板到相机，pnp求出

眼在手外
1.Rbase2end，跟眼在手上相反
2.跟眼在手上相同。

对一般机械臂，对于每个位姿，通常会返回六个参数：$\theta_x\ \theta_y\ \theta_z\ t_x\ t_y\ t_z$，这六个参数是机械臂末端在基底坐标系下的位姿的表示

即：$^{base}_{end}R = R_z R_y R_x$ $\qquad ^{base}_{end}T = (t_x \quad t_y \quad t_z)^T$

对于相机到标定板坐标系的求解，可以使用Opencv中 solvePnP 函数， solvePnP 是用来求解2D-3D的位姿对应关系，在这里，图片（2D），而标定板坐标系是（3D），利用solvePnP函数，就可以得到图片（相机坐标系）与标定板坐标系的变换关系。 具体输入输出如下：

ObjectPoints: 棋盘格坐标系，由棋盘格真实坐标生成，一般以棋盘格左上角顶点建立。 Z为0。

imagePoints: 图片识别到的角点坐标，与objectPoints 中的值一一对应。

cameraMatrix: 相机内参

distCoeffs: 相机畸变

rvec: 标定板坐标系到相机坐标系的旋转向量， 可用cv:: Rodrigues() 变为旋转矩阵

tvec: 标定板坐标系到相机坐标系的旋转矩阵

即得：$^{camera}_{board}R \quad ^{camera}_{board}T$

```
solvePnP()
bool cv::solvePnP ( InputArray objectPoints,
                    InputArray imagePoints,
                    InputArray cameraMatrix,
                    InputArray distCoeffs,
                    OutputArray rvec,
                    OutputArray tvec,
                    bool useExtrinsicGuess = false ,
                    int flags = SOLVEPNP_ITERATIVE
                  )
```

注： 这里使用的输入是vectorofvector， 输出是vector， 意味着可以一次性输入所有位姿图片，然后计算得到每张图的变换矩阵

5.初学者易错点

一：用calibrateCamera求标定板到相机的R,t，跟抓取用的内参不同，造成误差
二：标定板角点方向反了。默认从左到右，有时会出现从右到左，导致不在统一坐标系。
三：标定板面积过小，而且只在中心移动。会导致边缘不准。
四：标定时要旋转
五：图片太少。要10张以上
六。某些相机有问题，rgbd的变换矩阵要注意

6.实际实现

链接: 手眼标定（一）：Opencv4实现手眼标定及手眼系统测试_jian_1996的博客-CSDN博客 .
链接: OpenCV手眼标定（calibrateHandeye()）_hello-CSDN博客 .
下面展示一些 内联代码片。

```
#include <opencv2/opencv.hpp>
#include <iostream>
#include <math.h>
#include "B2C.h"
using namespace std;
using namespace cv;

Mat R_T2HomogeneousMatrix(const Mat& R, const Mat& T);
void HomogeneousMtr2RT(Mat& HomoMtr, Mat& R, Mat& T);
bool isRotatedMatrix(Mat& R);
Mat eulerAngleToRotateMatrix(const Mat& eulerAngle, const std::string& seq);
Mat quaternionToRotatedMatrix(const Vec4d& q);
Mat attitudeVectorToMatrix(const Mat& m, bool useQuaternion, const string& seq);
void getRT_fromTxt(double data[], Mat& R, Mat& T);
void inputParameter(cv::Mat _infrared_image, cv::Mat _deep_image, cv::Size _board_size, double _step_length,cv:
//数据使用的已有的值
//相机中13组标定板的位姿, x,y,z, rx,ry,rz,
Vec3f rotationMatrixToEulerAngles(Mat &R);
const double PI = atan(1.)*4.;
//机械臂末端13组位姿,x,y,z,rx,ry,rz
Mat_<double> ToolPose = (cv::Mat_<double>(20, 6) <<
 -664.634,-403.438,78.555,138.355,-78.188,173.584,
 -660.134,-398.965,-12.025,-178.356,-81.341,129.322,
 -782.915,-251.656,-17.289,170.170,-79.537,143.385,
 -828.512,-116.886,98.719,163.091,-67.357,163.263,
 -715.725,-104.34,282.172,160.366,-57.154,174.381,
 -621.014,-74.66,362.723,148.567,-56.028,-170.945,
 -574.93,27.628,377.899,158.905,-61.177,-178.347,
 -484.241,35.059,333.183,153.922,-55.22,-172.486,
 -502.543,0.931,347.799,166.151,-63.492,-174.493,
 -380.766,28.325,454.274,158.463,-57.679,-171.976,
 -384.061,41.977,370.506,158.33,-70.196,-164.039,
```

```
 -501.707,243.658,317.67,150.897,-67.592,-157.391,
 -467.975,313.209,199.931,-174.969,-67.423,170.218,
 -446.036,251.093,204.299,-179.317,-74.887,-174.295,
 -376.644,332.261,120.076,170.465,-68.618,-173.132,
 -378.888,288.551,88.038,-176.476,-81.741,-176.555,
 -486.6,463.166,-0.983,-174.493,-77.633,-178.958,
 -526.581,501.952,-7.103,-154.059,-79.925,170.357,
 -465.285,502.733,-29.938,-71.563,-88.506,91.941,
 -426.73,453.231,-81.055,-28.951,-85.232,35.079


  ) ;


int main(int argc, char** argv)
{
  //数据声明
 vector<Mat> R_gripper2base;
 vector<Mat> T_gripper2base;
 vector<Mat> R_target2cam;
 vector<Mat> T_target2cam;
 Mat R_cam2gripper = Mat(3, 3, CV_64FC1);     //相机与机械臂末端坐标系的旋转矩阵与平移矩阵
 Mat T_cam2gripper = Mat(3, 1, CV_64FC1);
 Mat Homo_cam2gripper = Mat(4, 4, CV_64FC1);

 vector<Mat> Homo_target2cam;
 vector<Mat> Homo_gripper2base;
 Mat tempR, tempT, temp;


 std::vector<cv::String> filenames; // notice here that we are using the Opencv's embedded "String" class
 cv::String folder = "/home/cai/hans_robot/PIC/hand_calibrate/2021.11.22old3"; // again we are using the Open

 cv::glob(folder, filenames); // new function that does the job ;-)
 string fin_name = folder + "/Tcw.txt";
 cout << fin_name << endl;
 ifstream fin(fin_name);



 for (int i = 0; i <filenames.size() / 2; i++)     //计算标定板与相机间的齐次矩阵（旋转矩阵与平移向量）
   {


  double data[12] = { 0 };
  for (auto &d : data)
      {
    fin >> d;
   //cout << " " << d << endl;
      }
  getRT_fromTxt(data, tempR, tempT);
  temp = R_T2HomogeneousMatrix(tempR, tempR);
  Homo_gripper2base.push_back(temp);
  //tempT = tempT / 1000;
  /*cout << i << "::" << temp << endl;*/
  //cout << i << "::" << tempR << endl;
  cout << i << "end to base ::" << tempT.t() << endl;
  R_gripper2base.push_back(tempR);
  T_gripper2base.push_back(tempT);
   }

 cout <<  "board to camera::" << endl;
 //for (size_t i = 0; i < ToolPose.rows; ++i)
   for (size_t i = 0; i < filenames.size() / 2;++i)
   {


  string rgb_name = folder + "/" + to_string(i) + "_color.jpg";
  string depth_name = folder + "/" + to_string(i) + "_depth.png";
  //std::cout << rgb_name << std::endl;
  cv::Mat rgb = cv::imread(rgb_name);
  cv::Mat depth = cv::imread(depth_name, -1);
  /*cv::imshow("depth", depth);
  cv::waitKey(0);*/
  if (!rgb.data)
   std::cerr << "Problem loading image!!!" << std::endl;




  cv::Mat pnp_R,pnp_t;
  inputParameter(rgb, depth, cv::Size(11, 8), 30,pnp_R,pnp_t);

  //cout << "标定板到相机mat_camera" << mat_camera << endl;
  Homo_target2cam.push_back(mat_camera);

  HomogeneousMtr2RT(mat_camera, tempR, tempT);
  //tempT = tempT / 1000;
  //cout << i << "标定板到相机 ::" << mat_camera << endl;

  //cout << i << "::" << tempT << endl;

  Vec3f oula = rotationMatrixToEulerAngles(tempR);
  cout << i << " translation::" << tempT.t() << "  angle::" << oula << endl;
  //cout <<  "欧拉角::" << oula << endl;
//  R_target2cam.push_back(tempR);
//  T_target2cam.push_back(tempT);
      R_target2cam.push_back(pnp_R);
      T_target2cam.push_back(pnp_t);



  /* do whatever you want with your images here */
   }

 //TSAI计算速度最快
 calibrateHandEye(R_gripper2base, T_gripper2base, R_target2cam, T_target2cam, R_cam2gripper, T_cam2gripper, C

 Homo_cam2gripper = R_T2HomogeneousMatrix(R_cam2gripper, T_cam2gripper);
 cout << Homo_cam2gripper << endl;
 cout << "Homo_cam2gripper: " << isRotatedMatrix(Homo_cam2gripper) << endl;

 ///

 /**************************************************
 * @note   手眼系统精度测试，原理是标定板在机器人基坐标系中位姿固定不变，
  *        可以根据这一点进行演算
 **************************************************/
  //使用1,2组数据验证  标定板在机器人基坐标系中位姿固定不变
 cout << "1 : " << Homo_gripper2base[0] * Homo_cam2gripper * Homo_target2cam[0] << endl;
```

```
 cout << "2 : " << Homo_gripper2base[1] * Homo_cam2gripper * Homo_target2cam[1] << endl;
  / /标定板在相机中的位姿
 cout << "3 : " << Homo_target2cam[1] << endl;
 cout << "4 : " << Homo_cam2gripper.inv() * Homo_gripper2base[1].inv() * Homo_gripper2base[0] * Homo_cam2grip

 cout << "----haneye test-----" << endl;
 cout << "robot base :" << endl;
 Mat ave_R, ave_T;
 for (int i = 0; i < Homo_target2cam.size(); i++)
    {
  Mat chessPos{ 0.0,0.0,0.0,1.0 };  //4*1矩阵，单独求机械臂坐标系下，标定板XYZ
  Mat worldPos = Homo_gripper2base[i] * Homo_cam2gripper * Homo_target2cam[i];//* chessPos
  HomogeneousMtr2RT(worldPos, tempR, tempT);
  //tempT = tempT / 1000;
  //cout << i << "标定板到相机 ::" << mat_camera << endl;

  //cout << i << "::" << tempT << endl;

  Vec3f oula = rotationMatrixToEulerAngles(tempR);
  cout << i << " translation::" << tempT.t() << "     angle::" << oula << endl;
    }
  /*if (i != 0)
      {
    ave_R = ave_R + oula;
    ave_T = ave_T + tempT.t();
      }
    else
      {
    ave_R = oula;
    ave_T = tempT.t();
     } */

  //cout <<  "欧拉角::" << oula << endl;
  //cout << i << ": " << worldPos.t() << endl;
  //cout << i << ": " << worldPos << endl;
 / / }
 //Mat error_R, error_T;
 //for (int i = 0; i < Homo_target2cam.size(); i++)
 / / {
 // Mat chessPos{ 0.0,0.0,0.0,1.0 };  //4*1矩阵，单独求机械臂坐标系下，标定板XYZ
 // Mat worldPos = Homo_gripper2base[i] * Homo_cam2gripper * Homo_target2cam[i];//* chessPos
 // HomogeneousMtr2RT(worldPos, tempR, tempT);

 // Vec3f oula = rotationMatrixToEulerAngles(tempR);
 // if (i != 0)
 / /  {
 //  error_T += abs(ave_T - tempT.t());
 //  error_R += abs(ave_R - oula);
 / /  }

 //  else
 / /  {
 //  error_T = abs(ave_T - tempT.t());
 //  error_R= abs(ave_R - oula);
 / /  }
 / /
 / /
 / / }
 //cout << " 平移误差::" << error_T / Homo_target2cam.size() << "     欧拉角::" << error_R / Homo_target2cam.s
 waitKey(0);

 return 0;
}

Vec3f rotationMatrixToEulerAngles(Mat &R)
{

 //assert(isRotationMatrix(R));

 float sy = sqrt(R.at<double>(0, 0) * R.at<double>(0, 0) + R.at<double>(1, 0) * R.at<double>(1, 0));

 bool singular = sy < 1e-6; // If

 float x, y, z;
 if (!singular)
    {
  x = atan2(R.at<double>(2, 1), R.at<double>(2, 2));
  y = atan2(-R.at<double>(2, 0), sy);
  z = atan2(R.at<double>(1, 0), R.at<double>(0, 0));
    }
 else
    {
  x = atan2(-R.at<double>(1, 2), R.at<double>(1, 1));
  y = atan2(-R.at<double>(2, 0), sy);
   z = 0;
    }
 return Vec3f(x, y, z) / PI * 180.0;

}


void getRT_fromTxt(double data[], Mat& R, Mat& T)
{
 //Mat R_HomoMtr = HomoMtr(Rect(0, 0, 3, 3)); //注意Rect取值
 //Mat T_HomoMtr = HomoMtr(Rect(3, 0, 1, 3));
 //R_HomoMtr.copyTo(R);
 //T_HomoMtr.copyTo(T);
 /*HomoMtr(Rect(0, 0, 3, 3)).copyTo(R);
 HomoMtr(Rect(3, 0, 1, 3)).copyTo(T);*/

 R = (Mat_<double>(3, 3) <<
  data[0], data[1], data[2],
  data[4], data[5], data[6],
  data[8], data[9], data[10]
     ) ;
 T = (Mat_<double>(3, 1) <<
  data[3], data[7], data[11]
     ) ;


}

/**************************************************
* @brief   将旋转矩阵与平移向量合成为齐次矩阵
* @note
* @param   Mat& R   3*3旋转矩阵
* @param   Mat& T   3*1平移矩阵
* @return  Mat      4*4齐次矩阵
**************************************************/
Mat R_T2HomogeneousMatrix(const Mat& R, const Mat& T)
```

```
{
 Mat HomoMtr;
 Mat_<double> R1 = (Mat_<double>(4, 3) <<
  R.at<double>(0, 0), R.at<double>(0, 1), R.at<double>(0, 2),
  R.at<double>(1, 0), R.at<double>(1, 1), R.at<double>(1, 2),
  R.at<double>(2, 0), R.at<double>(2, 1), R.at<double>(2, 2),
   0, 0, 0);
 Mat_<double> T1 = (Mat_<double>(4, 1) <<
  T.at<double>(0, 0),
  T.at<double>(1, 0),
  T.at<double>(2, 0),
   1 ) ;
 cv::hconcat(R1, T1, HomoMtr);  //矩阵拼接
 return HomoMtr;
}


/**************************************************
* @brief   齐次矩阵分解为旋转矩阵与平移矩阵
* @note
* @param const Mat& HomoMtr  4*4齐次矩阵
* @param Mat& R              输出旋转矩阵
* @param Mat& T              输出平移矩阵
* @return
**************************************************/
void HomogeneousMtr2RT(Mat& HomoMtr, Mat& R, Mat& T)
{
 //Mat R_HomoMtr = HomoMtr(Rect(0, 0, 3, 3)); //注意Rect取值
 //Mat T_HomoMtr = HomoMtr(Rect(3, 0, 1, 3));
 //R_HomoMtr.copyTo(R);
 //T_HomoMtr.copyTo(T);
 /*HomoMtr(Rect(0, 0, 3, 3)).copyTo(R);
 HomoMtr(Rect(3, 0, 1, 3)).copyTo(T);*/
 Rect R_rect(0, 0, 3, 3);
 Rect T_rect(3, 0, 1, 3);
 R = HomoMtr(R_rect);
 T = HomoMtr(T_rect);

}

/**************************************************
* @brief   检查是否是旋转矩阵
* @note
* @param
* @param
* @param
* @return  true : 是旋转矩阵， false : 不是旋转矩阵
**************************************************/
bool isRotatedMatrix(Mat& R)  //旋转矩阵的转置矩阵是它的逆矩阵，逆矩阵 * 矩阵 = 单位矩阵
{
 Mat temp33 = R({ 0,0,3,3 }); //无论输入是几阶矩阵，均提取它的三阶矩阵
 Mat Rt;
 transpose(temp33, Rt);  //转置矩阵
 Mat shouldBeIdentity = Rt * temp33;//是旋转矩阵则乘积为单位矩阵
 Mat I = Mat::eye(3, 3, shouldBeIdentity.type());

 return cv::norm(I, shouldBeIdentity) < 1e-6;
}

/**************************************************
* @brief   欧拉角转换为旋转矩阵
* @note
* @param   const std::string& seq  指定欧拉角的排列顺序；（机械臂的位姿类型有xyz,zyx,zyz几种，需要区分）
* @param   const Mat& eulerAngle   欧拉角（1*3矩阵），角度值
* @param
* @return  返回3*3旋转矩阵
**************************************************/
Mat eulerAngleToRotateMatrix(const Mat& eulerAngle, const std::string& seq)
{
 CV_Assert(eulerAngle.rows == 1 && eulerAngle.cols == 3);//检查参数是否正确

 eulerAngle /= (180 / CV_PI);  //度转弧度

 Matx13d m(eulerAngle);    //<double, 1, 3>

 auto rx = m(0, 0), ry = m(0, 1), rz = m(0, 2);
 auto rxs = sin(rx), rxc = cos(rx);
 auto rys = sin(ry), ryc = cos(ry);
 auto rzs = sin(rz), rzc = cos(rz);

 //XYZ方向的旋转矩阵
 Mat RotX = (Mat_<double>(3, 3) << 1, 0, 0,
  0, rxc, -rxs,
  0, rxs, rxc);
 Mat RotY = (Mat_<double>(3, 3) << ryc, 0, rys,
  0, 1, 0,
  -rys, 0, ryc);
 Mat RotZ = (Mat_<double>(3, 3) << rzc, -rzs, 0,
  rzs, rzc, 0,
  0, 0, 1);
 //按顺序合成后的旋转矩阵
 cv::Mat rotMat;

 if (seq == "zyx") rotMat = RotX * RotY * RotZ;
 else if (seq == "yzx") rotMat = RotX * RotZ * RotY;
 else if (seq == "zxy") rotMat = RotY * RotX * RotZ;
 else if (seq == "yxz") rotMat = RotZ * RotX * RotY;
 else if (seq == "xyz") rotMat = RotZ * RotY * RotX;
 else if (seq == "xzy") rotMat = RotY * RotZ * RotX;
 else
  {
  cout << "Euler Angle Sequence string is wrong...";
  }
 if (!isRotatedMatrix(rotMat))  //欧拉角特殊情况下会出现死锁
  {
  cout << "Euler Angle convert to RotatedMatrix failed..." << endl;
  exit(-1);
  }
 return rotMat;
}

/**************************************************
* @brief   将四元数转换为旋转矩阵
* @note
* @param   const Vec4d& q   归一化的四元数: q = q0 + q1 * i + q2 * j + q3 * k;
* @return  返回3*3旋转矩阵R
**************************************************/
Mat quaternionToRotatedMatrix(const Vec4d& q)
{
 double q0 = q[0], q1 = q[1], q2 = q[2], q3 = q[3];
```

```
 double q0q0 = q0 * q0, q1q1 = q1 * q1, q2q2 = q2 * q2, q3q3 = q3 * q3;
 double q0q1 = q0 * q1, q0q2 = q0 * q2, q0q3 = q0 * q3;
 double q1q2 = q1 * q2, q1q3 = q1 * q3;
 double q2q3 = q2 * q3;
  //根据公式得来
 Mat RotMtr = (Mat_<double>(3, 3) << (q0q0 + q1q1 - q2q2 - q3q3), 2 * (q1q2 + q0q3), 2 * (q1q3 - q0q2),
  2 * (q1q2 - q0q3), (q0q0 - q1q1 + q2q2 - q3q3), 2 * (q2q3 + q0q1),
  2 * (q1q3 + q0q2), 2 * (q2q3 - q0q1), (q0q0 - q1q1 - q2q2 + q3q3));
  //这种形式等价
 /*Mat RotMtr = (Mat_<double>(3, 3) << (1 - 2 * (q2q2 + q3q3)), 2 * (q1q2 - q0q3), 2 * (q1q3 + q0q2),
 2 * (q1q2 + q0q3), 1 - 2 * (q1q1 + q3q3), 2 * (q2q3 - q0q1),
 2 * (q1q3 - q0q2), 2 * (q2q3 + q0q1), (1 - 2 * (q1q1 + q2q2)));*/

 return RotMtr;
}

/**************************************************
* @brief      将采集的原始数据转换为齐次矩阵（从机器人控制器中获得的）
* @note
* @param   Mat& m    1*6//1*10矩阵 ，  元素为： x,y,z,rx,ry,rz  or x,y,z, q0,q1,q2,q3,rx,ry,rz
* @param   bool useQuaternion      原始数据是否使用四元数表示
* @param   string& seq          原始数据使用欧拉角表示时，坐标系的旋转顺序
* @return   返回转换完的齐次矩阵
**************************************************/
Mat attitudeVectorToMatrix(const Mat& m, bool useQuaternion, const string& seq)
{
 CV_Assert(m.total() == 6 || m.total() == 10);
 //if (m.cols == 1) //转置矩阵为行矩阵
 // m = m.t();

 Mat temp = Mat::eye(4, 4, CV_64FC1);

 if (useQuaternion)
    {
  Vec4d quaternionVec = m({ 3,0,4,1 });   //读取存储的四元数
  quaternionToRotatedMatrix(quaternionVec).copyTo(temp({ 0,0,3,3 }));
    }
  else
    {
   Mat rotVec;
   if (m.total() == 6)
       {
    rotVec = m({ 3,0,3,1 });    //读取存储的欧拉角
       }
   if (m.total() == 10)
       {
    rotVec = m({ 7,0,3,1 });
       }
      //如果seq为空，表示传入的是3*1旋转向量，否则，传入的是欧拉角
   if (0 == seq.compare(""))
       {
   Rodrigues(rotVec, temp({ 0,0,3,3 }));   //罗德利斯转换
       }
    else
       {
   eulerAngleToRotateMatrix(rotVec, seq).copyTo(temp({ 0,0,3,3 }));
       }
    }
  //存入平移矩阵
 temp({ 3,0,1,3 }) = m({ 0,0,3,1 }).t();
 return temp;   //返回转换结束的齐次矩阵
}

void inputParameter(cv::Mat _infrared_image, cv::Mat _deep_image, cv::Size _board_size, double _step_length,cv:
{
 deep_image.create(_infrared_image.size(), CV_16UC1);
 infrared_image = _infrared_image.clone();
 deep_image = _deep_image.clone();
 board_size = _board_size;
 step_length = _step_length;

 num_of_corner = board_size.height*board_size.width;

  //读取相机内参和畸变参数，矫正图片
  cv::FileStorage fs("../register/calib_ir_1DK.yaml", cv::FileStorage::READ);
  cv::Mat intrinsic_matrix(3, 3, CV_64FC1), dist_coeffs(1, 5, CV_64FC1);
  fs["cameraMatrix"] >> intrinsic_matrix;
  fs["distortionCoefficients"] >> dist_coeffs;
  fs.release();
  cv::Mat map1, map2;


  //构造世界坐标点,右手坐标系，Y轴是短边，X轴是长边
 for (int y = 0; y < board_size.height; y++)  //8
 for (int x = 0; x < board_size.width; x++)  //11
    {
  board_points.push_back(cv::Point3f(step_length*x, step_length*y, 0.0f));
    }

  //检测角点
 cv::Mat infrared_gray;
 cv::cvtColor(infrared_image, infrared_gray, cv::COLOR_BGR2GRAY);
 cv::imshow("infrared_gray", infrared_gray);
    //cv::waitKey(0);


  is_found = cv::findChessboardCorners( infrared_gray, board_size, image_corners, cv::CALIB_CB_ADAPTIVE_TH
    break;


  is_found = cv::findCirclesGrid(255 - infrared_gray, board_size, image_corners, cv::CALIB_CB_SYMMETRIC_GR


    if (image_corners[0].x > image_corners.back().x)
    {
        //交换列
        for (int i = 0; i < (int)board_size.height; i++)  //行
            for (int j = 0; j < (int)board_size.width / 2; j++) //列
                std::swap(image_corners[i*board_size.width + j], image_corners[(i + 1)*board_size.width - j - 1
    }
    if (image_corners[0].y > image_corners.back().y)
    {
        //交换行
        for (int i = 0; i < (int)board_size.width; i++) //列
            for (int j = 0; j < (int)board_size.height / 2; j++) //行
                std::swap(image_corners[j*board_size.width + i], image_corners[(board_size.height - j - 1)*boar
    }

```

```
    cv::Mat rvec,tvec;

    cv::solvePnPRansac(board_points,image_corners,intrinsic_matrix,dist_coeffs,rvec,pnp_t);
    cv::Mat R_;
    cv::Rodrigues(rvec,pnp_R);
    std::cout<<" PNP R_ "<<R_<<" t" <<tvec<<std::endl;


}
```