

z504727099

码龄6年 暂无认证

20

10万+

9万+

2万+

原创

周排名

总排名

访问

等级

421

14

19

37

180

积分

粉丝

获赞

评论

收藏

私信

关注

创作者榜单

创作稿酬 200元/篇

供稿得现金奖励，多劳多得

点此查看详情

博客之星--博主的年度最高成就表彰活动

成为博客之星不仅可获博客之星专属荣誉还可获博客之星年度大奖，一年仅有一次。

去创作

搜博文文章

热门文章

3D视觉工坊-手眼标定（附opencv实现代码） 6006

UNet语义分割多分类学习 5355

opencv调用实例分割yolact 3038

orbslam2+azure kinect DK稠密重建 1887

速腾32线激光雷达和小觅相机外参标定 1825

最新评论

orbslam2+azure kinect DK稠密重建
bingo_haha: 给tracking里面加语句太厉害了，解决了困扰我好几天的问题，感谢...

orbslam2+azure kinect DK稠密重建
acanab: 我的没有viewer

orbslam2+azure kinect DK稠密重建
男神早上好: 跑通了.(但viewer没有稠密点云画面,一直在绿红黑界面.请问博主是什么...

orbslam2+azure kinect DK稠密重建
weixin_45669776: 博主知道原因么

orbslam2+azure kinect DK稠密重建
weixin_45669776: rosrund报错realoc(): inv alid pointer Aborted (core dumped)

您愿意向朋友推荐“博客详情页”吗？

😞

😐

😌

😄

😁

强烈不推荐

不推荐

一般般

推荐

强烈推荐

最新文章

泊车场景中的slam

opencv调用实例分割yolact

实例分割yolact C++ nncnn调用失败经验

2023年 1篇

2022年 2篇

2021年 12篇

2020年 4篇

2019年 1篇

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

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

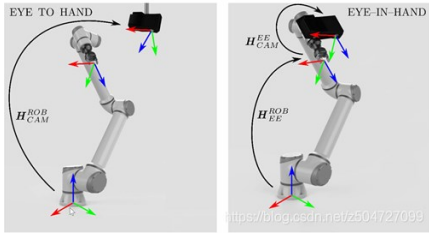
6 订阅 1 篇文章 订阅专栏

1.基本介绍

手眼标定两种形式

眼在手外 eye to hand

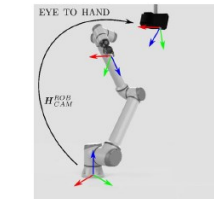
眼在手上 eye in hand



2.公式推导

2 手眼标定公式推导 — 眼在手外

公众号：3D视觉工坊



手眼标定坐标系表示：
机械臂基底坐标系 -- base
机械臂末端坐标系 -- end
相机坐标系 -- camera
标定板坐标系 -- board

图片来源：https://blog.zivid.com/importance-of-3d-hand-eye-calibration

求解目标： 基底坐标系 到 相机的变换矩阵 ${}^{camera}_{base}M$

实现方法： 1 把标定板固定在机械臂末端
2 使用相机拍摄不同机械臂姿态下的标定板图片n张 n>3

已知对每张图片：

$$\begin{aligned} {}^{board}_{end}M &= {}^{camera}_{board}M^{-1} * {}^{camera}_{base}M * {}^{end}_{base}M^{-1} \\ \text{则可以得到如下等式:} \\ {}^{camera}_{board}M^{-1}_1 * {}^{camera}_{base}M * {}^{end}_{base}M^{-1}_1 &= {}^{camera}_{board}M^{-1}_2 * {}^{camera}_{base}M * {}^{end}_{base}M^{-1}_2 \\ \left. \begin{aligned} &{}^{camera}_{board}M^{-1}_2 * {}^{camera}_{base}M * {}^{end}_{base}M^{-1}_2 \\ &\dots\dots\dots \\ &{}^{camera}_{board}M^{-1}_n * {}^{camera}_{base}M * {}^{end}_{base}M^{-1}_n \end{aligned} \right\} \begin{matrix} A \\ \\ B \end{matrix} \end{aligned}$$

眼在手上类似

3.方程AX=XB求解

3 解方程 AX=XB - Tais 方法

公众号：3D视觉工坊

步骤一：把旋转矩阵变为旋转向量

$$\begin{cases} r_a = Roderiques(R_a) \\ r_b = Roderiques(R_b) \end{cases}$$

步骤二：旋转向量归一化

$$\begin{cases} \theta_a = \|r_a\|_2 \quad N_a = \frac{r_a}{\theta_a} \\ \theta_b = \|r_b\|_2 \quad N_b = \frac{r_b}{\theta_b} \end{cases}$$

步骤三：计算修正的罗德里格斯向量

$$\begin{cases} p_a = 2\sin\frac{\theta_a}{2} N_a \\ p_b = 2\sin\frac{\theta_b}{2} N_b \end{cases}$$

步骤四：计算初始旋转向量

$$skew(p_a + p_b)p'_x = p_b - p_a$$

步骤五：计算旋转向量

$$p_x = \frac{2p'_x}{\sqrt{1+|p'_x|^2}}$$

步骤六：计算旋转矩阵

$$R_x = \left(1 - \frac{|p_x|^2}{2}\right)I + \frac{1}{2}(P_x P_x^T + \sqrt{4 - |p_x|^2} * skew(p_x))$$

步骤七：计算平移矩阵

$$(R_A - I)T_X = R_X T_B - T_A$$

3 解方程 AX=XB - Tais 方法

公众号：3D视觉工坊

步骤一：把旋转矩阵变为旋转向量

$$\begin{cases} r_a = Roderiques(R_a) \\ r_b = Roderiques(R_b) \end{cases}$$

步骤二：旋转向量归一化

$$\begin{cases} \theta_a = \|r_a\|_2 \quad N_a = \frac{r_a}{\theta_a} \\ \theta_b = \|r_b\|_2 \quad N_b = \frac{r_b}{\theta_b} \end{cases}$$

步骤三：计算修正的罗德里格斯向量

$$\begin{cases} p_a = 2\sin\frac{\theta_a}{2} N_a \\ p_b = 2\sin\frac{\theta_b}{2} N_b \end{cases}$$

步骤四：计算初始旋转向量

$$skew(p_a + p_b)p'_x = p_b - p_a$$

步骤五：计算旋转向量

$$p_x = \frac{2p'_x}{\sqrt{1+|p'_x|^2}}$$

步骤六：计算旋转矩阵

$$R_x = \left(1 - \frac{|p_x|^2}{2}\right)I + \frac{1}{2}(P_x P_x^T + \sqrt{4 - |p_x|^2} * skew(p_x))$$

步骤七：计算平移矩阵

$$(R_A - I)T_X = R_X T_B - T_A$$

$$\begin{cases} p_b = 2\sin\frac{\theta_a}{2}N_b \end{cases}$$

假设一个向量a为:

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

则 \mathbf{a} 的反对称矩阵 \mathbf{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^Q 完成手眼标定

4 使Opencv 完成手眼标定

公众号：3D视觉工坊



```
void cv::calibrateHandEye( InputArrayOfArrays R_gripper2base,
                          InputArrayOfArrays R_gripper2base,
                          InputArrayOfArrays R_target2cam,
                          InputArrayOfArrays t_target2cam,
                          OutputArray R_cam2gripper,
                          OutputArray t_cam2gripper,
                          HandEyeCalibrationMethod method = CALIB_HAND_EYE_TSAI
                        )
```

眼在手上：

$$\begin{aligned} \text{camera}_{board} M_i * \text{camera}_{board} M_{i-1}^{-1} * \text{camera}_{end} M &= \text{camera}_{end} M * \text{base}_{end} M_i^{-1} * \text{base}_{end} M_{i-1} \\ \text{base}_{end} M_{i-1}^{-1} * \text{base}_{end} M_i * \text{camera}_{end} M &= \text{camera}_{end} M * \text{camera}_{board} M_{i-1} * \text{camera}_{board} M_i^{-1} \end{aligned}$$

$R_grripper2base:$	$base_{end}R$	$t_grripper2base:$	$base_{end}T$
$R_target2cam:$	$cam_{board}R$	$t_target2cam:$	$cam_{board}T$
$R_cam2grripper:$	$end_{cam}R$	$t_target2cam:$	$end_{cam}T$

```
gripper: 机械臂坐标系 --- end
base: 基底坐标系
target: 标定板坐标系 --- board
cam: 相机坐标系
```

眼在手外：

$$\begin{aligned} & \text{camera}_{\text{board}} M_i * \text{camera}_{\text{board}} M_{i-1}^{-1} * \text{camera}_{\text{base}} M = \text{camera}_{\text{base}} M * \text{end}_{\text{base}} M_i^{-1} * \text{end}_{\text{base}} M_{i-1} \\ & \quad \downarrow \\ & \text{end}_{\text{base}} M_{i-1}^{-1} * \text{end}_{\text{base}} M_i * \text{camera}_{\text{base}} M = \text{camera}_{\text{base}} M * \text{camera}_{\text{board}} M_{i-1} * \text{camera}_{\text{board}} M_i^{-1} \end{aligned}$$

R_gripper2base:	$\frac{end}{base}R$	t_gripper2base:	$\frac{end}{base}T$
R_target2cam:	$\frac{cam}{board}R$	t_target2cam:	$\frac{cam}{board}T$
R_cam2gripper:	$\frac{base}{camera}R$	t_target2cam:	$\frac{base}{camera}T$

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直接订阅相关话题
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。

$$\text{④: } {}^{base}_{end}R = R_z R_y R_x \quad {}^{base}_{end}T = (t_x \ t_y \ t_z)^T$$

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

```
• 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**。

```

1 #include <opencv2/opencv.hpp>
2 #include <iostream>
3 #include <math.h>
4 #include "B2C.h"
5 using namespace std;
6 using namespace cv;
7
8 Mat R_T2HomogeneousMatrix(const Mat& R, const Mat& T);
9 void HomogeneousMtr2RT(Mat& HomoMtr, Mat& R, Mat& T);
10 bool isRotatedMatrix(Mat& R);
11 Mat eulerAngleToRotateMatrix(const Mat& eulerAngle, const std::string& seq);
12 Mat quaternionToRotatedMatrix(const Vec4d& q);
13 Mat attitudeVectorToMatrix(const Mat& m, bool useQuaternion, const string& seq);
14 void getRT_fromTxt(double data[], Mat& R, Mat& T);
15 void inputParameter(cv::Mat _infrared_image, cv::Mat _deep_image, cv::Size _board_size, double _step_length, cv::Mat _data);
16 //数据使用的已有的值
17 //相机中13组标定板的位姿, x,y,z, rx,ry,rz,
18 Vec3f rotationMatrixToEulerAngles(Mat& R);
19 const double PI = atan(1)*4.;
20 //机械臂末端13组位姿, x,y,z,rx,ry,rz
21 Mat<double> ToolPose = (cv::Mat<double>(20, 6) <<
22 -664.634,-403.438,78.555,138.355,-78.188,173.584,
23 -660.134,-398.965,-12.025,-178.356,-81.341,129.322,
24 -782.915,-251.656,-17.289,170.170,-79.537,143.385,
25 -828.512,-116.886,98.719,163.091,-67.357,163.263,
26 -715.725,-104.34,282.172,160.366,-57.154,174.381,
27 -621.014,-74.66,362.723,148.567,-56.028,-170.945,
28 -574.93,27.628,377.899,158.905,-61.177,-178.347,
29 -484.241,35.059,333.183,153.922,-55.22,-172.486,
30 -502.543,0.931,347.799,166.151,-63.492,-174.493,
31 -380.766,28.325,454.274,158.463,-57.679,-171.976,
32 -384.061,41.977,370.506,158.33,-70.196,-164.039,

```

```

33  -501.707,243.658,317.67,150.897,-67.592,-157.391,
34  -467.975,313.209,199.931,-174.969,-67.423,170.218,
35  -446.036,251.093,204.299,-179.317,-74.887,-174.295,
36  -376.644,332.261,120.076,170.465,-68.618,-173.132,
37  -378.888,288.551,88.038,-176.476,-81.741,-176.555,
38  -486.6,463.166,-0.983,-174.493,-77.633,-178.958,
39  -526.581,501.952,-7.103,-154.059,-79.925,170.357,
40  -465.285,502.733,-29.938,-71.563,-88.506,91.941,
41  -426.73,453.231,-81.055,-28.951,-85.232,35.079
42
43
44  );
45
46
47  int main(int argc, char** argv)
48  {
49      / 数据声明
50      vector<Mat> R_gripper2base;
51      vector<Mat> T_gripper2base;
52      vector<Mat> R_target2cam;
53      vector<Mat> T_target2cam;
54      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
55      Mat T_cam2gripper = Mat(3, 1, CV_64FC1);
56      Mat Homo_cam2gripper = Mat(4, 4, CV_64FC1);
57
58      vector<Mat> Homo_target2cam;
59      vector<Mat> Homo_gripper2base;
60      Mat tempR, tempT, temp;
61
62
63      std::vector<cv::String> filenames; // notice here that we are using the Opencv's embedded "String" class
64      cv::String folder = "/home/cai/hans_robot/PIC/hand_calibrate/2021.11.22old3"; // again we are using the Open
65
66      cv::glob(folder, filenames); // new function that does the job ;-)
67      string fin_name = folder + "/Tcw.txt";
68      cout << fin_name << endl;
69      ifstream fin(fin_name);
70
71
72
73      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）
74      {
75
76
77          double data[12] = { 0 };
78          for (auto &d : data)
79          {
80              fin >> d;
81              //cout << " " << d << endl;
82          }
83          getRT_fromTxt(data, tempR, tempT);
84          temp = R_T2HomogeneousMatrix(tempR, tempR);
85          Homo_gripper2base.push_back(temp);
86          //tempT = tempT / 1000;
87          /*cout << i << " : " << temp << endl;*/
88          //cout << i << " : " << tempR << endl;
89          cout << i << "end to base : " << tempT.t() << endl;
90          R_gripper2base.push_back(tempR);
91          T_gripper2base.push_back(tempT);
92      }
93
94      cout << "board to camera:" << endl;
95      //for (size_t i = 0; i < ToolPose.rows; ++i)
96      for (size_t i = 0; i < filenames.size() / 2; ++i)
97      {
98
99
100      string rgb_name = folder + "/" + to_string(i) + "_color.jpg";
101      string depth_name = folder + "/" + to_string(i) + "_depth.png";
102      //std::cout << rgb_name << std::endl;
103      cv::Mat rgb = cv::imread(rgb_name);
104      cv::Mat depth = cv::imread(depth_name, -1);
105      /*cv::imshow("depth", depth);
106      cv::waitKey(0);*/
107      if (!rgb.data)
108          std::cerr << "Problem loading image!!!" << std::endl;
109
110
111
112
113      cv::Mat pnp_R, pnp_t;
114      InputParameter(rgb, depth, cv::Size(11, 8), 30, pnp_R, pnp_t);
115
116      //cout << "标定板到相机mat_camera" << mat_camera << endl;
117      Homo_target2cam.push_back(mat_camera);
118
119      HomogeneousMtr2RT(mat_camera, tempR, tempT);
120      //tempT = tempT / 1000;
121      //cout << i << "标定板到相机 : " << mat_camera << endl;
122
123      //cout << i << " : " << tempT << endl;
124
125      Vec3f oula = rotationMatrixToEulerAngles(tempR);
126      cout << i << " translation:" << tempT.t() << " angle:" << oula << endl;
127      //cout << "欧拉角:" << oula << endl;
128      // R_target2cam.push_back(tempR);
129      // T_target2cam.push_back(tempT);
130      R_target2cam.push_back(pnp_R);
131      T_target2cam.push_back(pnp_t);
132
133
134
135      /* do whatever you want with your images here */
136      }
137
138      //TSA I计算速度最快
139      calibrateHandEye(R_gripper2base, T_gripper2base, R_target2cam, T_target2cam, R_cam2gripper, T_cam2gripper, C
140
141      Homo_cam2gripper = R_T2HomogeneousMatrix(R_cam2gripper, T_cam2gripper);
142      cout << Homo_cam2gripper << endl;
143      cout << "Homo_cam2gripper: " << isRotatedMatrix(Homo_cam2gripper) << endl;
144
145      / / /
146
147      /*****
148      * @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，
149      * 可以根据这一点进行推算
150      *****/
151      / 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
152      cout << "1 : " << Homo_gripper2base[0] * Homo_cam2gripper * Homo_target2cam[0] << endl;

```

```

153 cout << "2 : " << Homo_gripper2base[1] * Homo_cam2gripper * Homo_target2cam[1] << endl;
154 // 标定板在相机中的位姿
155 cout << "3 : " << Homo_target2cam[1] << endl;
156 cout << "4 : " << Homo_cam2gripper.inv() * Homo_gripper2base[1].inv() * Homo_gripper2base[0] * Homo_cam2grip
157
158 cout << "----haneye test-----" << endl;
159 cout << "robot base :" << endl;
160 Mat ave_R, ave_T;
161 for (int i = 0; i < Homo_target2cam.size(); i++)
162 {
163     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
164     Mat worldPos = Homo_gripper2base[i] * Homo_cam2gripper * Homo_target2cam[i];/* chessPos
165     HomogeneousMtr2RT(worldPos, tempR, tempT);
166     //tempT = tempT / 1000;
167     //cout << i << "标定板到相机 : " << mat_camera << endl;
168
169     //cout << i << " : " << tempT << endl;
170
171     Vec3f oula = rotationMatrixToEulerAngles(tempR);
172     cout << i << " translation: " << tempT.t() << "    angle: " << oula << endl;
173 }
174 /*if (i != 0)
175 {
176     ave_R = ave_R + oula;
177     ave_T = ave_T + tempT.t();
178 }
179 else
180 {
181     ave_R = oula;
182     ave_T = tempT.t();
183 } * /
184
185 //cout << "欧拉角: " << oula << endl;
186 //cout << i << " : " << worldPos.t() << endl;
187 //cout << i << " : " << worldPos << endl;
188 // }
189 //Mat error_R, error_T;
190 //for (int i = 0; i < Homo_target2cam.size(); i++)
191 // {
192 // 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
193 // Mat worldPos = Homo_gripper2base[i] * Homo_cam2gripper * Homo_target2cam[i];/* chessPos
194 // HomogeneousMtr2RT(worldPos, tempR, tempT);
195
196 // Vec3f oula = rotationMatrixToEulerAngles(tempR);
197 // if (i != 0)
198 // {
199 //     error_T += abs(ave_T - tempT.t());
200 //     error_R += abs(ave_R - oula);
201 // }
202
203 // else
204 // {
205 //     error_T = abs(ave_T - tempT.t());
206 //     error_R = abs(ave_R - oula);
207 // }
208 // /
209 // /
210 // / }
211 //cout << " 平移误差: " << error_T / Homo_target2cam.size() << "    欧拉角: " << error_R / Homo_target2cam.s
212 waitKey(0);
213
214 return 0;
215 }
216
217 Vec3f rotationMatrixToEulerAngles(Mat &R)
218 {
219
220 //assert(isRotationMatrix(R));
221
222 float sy = sqrt(R.at<double>(0, 0) * R.at<double>(0, 0) + R.at<double>(1, 0) * R.at<double>(1, 0));
223
224 bool singular = sy < 1e-6; // If
225
226 float x, y, z;
227 if (!singular)
228 {
229     x = atan2(R.at<double>(2, 1), R.at<double>(2, 2));
230     y = atan2(-R.at<double>(2, 0), sy);
231     z = atan2(R.at<double>(1, 0), R.at<double>(0, 0));
232 }
233 else
234 {
235     x = atan2(-R.at<double>(1, 2), R.at<double>(1, 1));
236     y = atan2(-R.at<double>(2, 0), sy);
237     z = 0;
238 }
239 return Vec3f(x, y, z) / PI * 180.0;
240 }
241
242
243
244 void getRT_fromTxt(double data[], Mat& R, Mat& T)
245 {
246     //Mat R_HomoMtr = HomoMtr(Rect(0, 0, 3, 3)); //注意Rect取值
247     //Mat T_HomoMtr = HomoMtr(Rect(3, 0, 1, 3));
248     //R_HomoMtr.copyTo(R);
249     //T_HomoMtr.copyTo(T);
250     /*HomoMtr(Rect(0, 0, 3, 3)).copyTo(R);
251     HomoMtr(Rect(3, 0, 1, 3)).copyTo(T);*/
252
253     R = (Mat_<double>(3, 3) <<
254         data[0], data[1], data[2],
255         data[4], data[5], data[6],
256         data[8], data[9], data[10]
257     );
258     T = (Mat_<double>(3, 1) <<
259         data[3], data[7], data[11]
260     );
261
262
263 }
264
265 /*****
266 * @brief 将旋转矩阵与平移向量合成为齐次矩阵
267 * @note
268 * @param Mat& R 3*3旋转矩阵
269 * @param Mat& T 3*1平移矩阵
270 * @return Mat 4*4齐次矩阵
271 *****/
272 Mat R_T2HomogeneousMatrix(const Mat& R, const Mat& T)

```

```

274 {
275     Mat HomoMtr;
276     Mat<double> R1 = (Mat<double>(4, 3) <<
277         R.at<double>(0, 0), R.at<double>(0, 1), R.at<double>(0, 2),
278         R.at<double>(1, 0), R.at<double>(1, 1), R.at<double>(1, 2),
279         R.at<double>(2, 0), R.at<double>(2, 1), R.at<double>(2, 2),
280         0, 0, 0);
281     Mat<double> T1 = (Mat<double>(4, 1) <<
282         T.at<double>(0, 0),
283         T.at<double>(1, 0),
284         T.at<double>(2, 0),
285         1);
286     cv::hconcat(R1, T1, HomoMtr); //矩阵拼接
287     return HomoMtr;
288 }
289
290 /*****
291  * @brief 齐次矩阵分解为旋转矩阵与平移矩阵
292  * @note
293  * @param const Mat& HomoMtr 4*4齐次矩阵
294  * @param Mat& R 输出旋转矩阵
295  * @param Mat& T 输出平移矩阵
296  * @return
297  *****/
298 void HomogeneousMtr2RT(Mat& HomoMtr, Mat& R, Mat& T)
299 {
300     //Mat R_HomoMtr = HomoMtr(Rect(0, 0, 3, 3)); //注意Rect取值
301     //Mat T_HomoMtr = HomoMtr(Rect(3, 0, 1, 3));
302     //R_HomoMtr.copyTo(R);
303     //T_HomoMtr.copyTo(T);
304     //HomoMtr(Rect(0, 0, 3, 3)).copyTo(R);
305     HomoMtr(Rect(3, 0, 1, 3)).copyTo(T);
306     Rect R_rect(0, 0, 3, 3);
307     Rect T_rect(3, 0, 1, 3);
308     R = HomoMtr(R_rect);
309     T = HomoMtr(T_rect);
310 }
311 }
312
313 /*****
314  * @brief 检查是否是旋转矩阵
315  * @note
316  * @param
317  * @param
318  * @param
319  * @return true : 是旋转矩阵, false : 不是旋转矩阵
320  *****/
321 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
322 {
323     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
324     Mat Rt;
325     transpose(temp33, Rt); //转置矩阵
326     Mat shouldBeIdentity = Rt * temp33; //是旋转矩阵则乘积为单位矩阵
327     Mat I = Mat::eye(3, 3, shouldBeIdentity.type());
328
329     return cv::norm(I, shouldBeIdentity) < 1e-6;
330 }
331
332 /*****
333  * @brief 欧拉角转换为旋转矩阵
334  * @note
335  * @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)
336  * @param const Mat& eulerAngle 欧拉角 (1*3矩阵), 角度值
337  * @param
338  * @return 返回3*3旋转矩阵
339  *****/
340 Mat eulerAngleToRotateMatrix(const Mat& eulerAngle, const std::string& seq)
341 {
342     CV_Assert(eulerAngle.rows == 1 && eulerAngle.cols == 3); //检查参数是否正确
343
344     eulerAngle /= (180 / CV_PI); //度转弧度
345
346     Matx13d m(eulerAngle); //<double, 1, 3>
347
348     auto rx = m(0, 0), ry = m(0, 1), rz = m(0, 2);
349     auto rxs = sin(rx), rxc = cos(rx);
350     auto rys = sin(ry), ryc = cos(ry);
351     auto rzs = sin(rz), rzc = cos(rz);
352
353     //XYZ方向的旋转矩阵
354     Mat RotX = (Mat<double>(3, 3) << 1, 0, 0,
355         0, rxc, -rxs,
356         0, rxs, rxc);
357     Mat RotY = (Mat<double>(3, 3) << ryc, 0, rys,
358         0, 1, 0,
359         -rys, 0, ryc);
360     Mat RotZ = (Mat<double>(3, 3) << rzc, -rzs, 0,
361         rzs, rzc, 0,
362         0, 0, 1);
363     //按顺序合成后的旋转矩阵
364     cv::Mat rotMat;
365
366     if (seq == "zyx") rotMat = RotX * RotY * RotZ;
367     else if (seq == "yzx") rotMat = RotX * RotZ * RotY;
368     else if (seq == "zxy") rotMat = RotY * RotX * RotZ;
369     else if (seq == "yxz") rotMat = RotZ * RotX * RotY;
370     else if (seq == "xyz") rotMat = RotZ * RotY * RotX;
371     else if (seq == "xzy") rotMat = RotY * RotZ * RotX;
372     else
373     {
374         cout << "Euler Angle Sequence string is wrong...";
375     }
376     if (!isRotatedMatrix(rotMat)) //欧拉角特殊情况下会出现死锁
377     {
378         cout << "Euler Angle convert to RotatedMatrix failed..." << endl;
379         exit(-1);
380     }
381     return rotMat;
382 }
383
384 /*****
385  * @brief 将四元数转换为旋转矩阵
386  * @note
387  * @param const Vec4d& q 归一化的四元数: q = q0 + q1 * i + q2 * j + q3 * k;
388  * @return 返回3*3旋转矩阵R
389  *****/
390 Mat quaternionToRotatedMatrix(const Vec4d& q)
391 {
392     double q0 = q[0], q1 = q[1], q2 = q[2], q3 = q[3];

```

```

393
394 double q0q0 = q0 * q0, q1q1 = q1 * q1, q2q2 = q2 * q2, q3q3 = q3 * q3;
395 double q0q1 = q0 * q1, q0q2 = q0 * q2, q0q3 = q0 * q3;
396 double q1q2 = q1 * q2, q1q3 = q1 * q3;
397 double q2q3 = q2 * q3;
398 // 根据公式得来
399 Mat RotMtr = (Mat_<double>(3, 3) << (q0q0 + q1q1 - q2q2 - q3q3), 2 * (q1q2 + q0q3), 2 * (q1q3 - q0q2),
400 2 * (q1q2 - q0q3), (q0q0 - q1q1 + q2q2 - q3q3), 2 * (q2q3 + q0q1),
401 2 * (q1q3 + q0q2), 2 * (q2q3 - q0q1), (q0q0 - q1q1 - q2q2 + q3q3));
402 // 这种形式等价
403 /*Mat RotMtr = (Mat_<double>(3, 3) << (1 - 2 * (q2q2 + q3q3)), 2 * (q1q2 - q0q3), 2 * (q1q3 + q0q2),
404 2 * (q1q2 + q0q3), 1 - 2 * (q1q1 + q3q3), 2 * (q2q3 - q0q1),
405 2 * (q1q3 - q0q2), 2 * (q2q3 + q0q1), (1 - 2 * (q1q1 + q2q2)));*/
406
407 return RotMtr;
408 }
409
410 /*****
411 * @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)
412 * @note
413 * @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
414 * @param bool useQuaternion 原始数据是否使用四元数表示
415 * @param string& seq 原始数据使用欧拉角表示时, 坐标系的旋转顺序
416 * @return 返回转换完的齐次矩阵
417 *****/
418 Mat attitudeVectorToMatrix(const Mat& m, bool useQuaternion, const string& seq)
419 {
420 CV_Assert(m.total() == 6 || m.total() == 10);
421 //if (m.cols == 1) //转置矩阵为行矩阵
422 // m = m.t();
423
424 Mat temp = Mat::eye(4, 4, CV_64FC1);
425
426 if (useQuaternion)
427 {
428 Vec4d quaternionVec = m({ 3,0,4,1 }); //读取存储的四元数
429 quaternionToRotatedMatrix(quaternionVec).copyTo(temp({ 0,0,3,3 }));
430 }
431 else
432 {
433 Mat rotVec;
434 if (m.total() == 6)
435 {
436 rotVec = m({ 3,0,3,1 }); //读取存储的欧拉角
437 }
438 if (m.total() == 10)
439 {
440 rotVec = m({ 7,0,3,1 });
441 }
442 // 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
443 if (0 == seq.compare(""))
444 {
445 Rodrigues(rotVec, temp({ 0,0,3,3 })); //罗德里斯转换
446 }
447 else
448 {
449 eulerAngleToRotateMatrix(rotVec, seq).copyTo(temp({ 0,0,3,3 }));
450 }
451 }
452 // 存入平移矩阵
453 temp({ 3,0,1,3 }) = m({ 0,0,3,1 }).t();
454 return temp; //返回转换结束的齐次矩阵
455 }
456
457 void inputParameter(cv::Mat _infrared_image, cv::Mat _deep_image, cv::Size _board_size, double _step_length, cv:
458 {
459 deep_image.create(_infrared_image.size(), CV_16UC1);
460 infrared_image = _infrared_image.clone();
461 deep_image = _deep_image.clone();
462 board_size = _board_size;
463 step_length = _step_length;
464
465 num_of_corner = board_size.height*board_size.width;
466
467 // 读取相机内参畸变参数, 矫正图片
468 cv::FileStorage fs("../register/calib_ir_10K.yaml", cv::FileStorage::READ);
469 cv::Mat intrinsic_matrix(3, 3, CV_64FC1), dist_coeffs(1, 5, CV_64FC1);
470 fs["cameraMatrix"] >> intrinsic_matrix;
471 fs["distortionCoefficients"] >> dist_coeffs;
472 fs.release();
473 cv::Mat map1, map2;
474
475
476 // 构造世界坐标点, 右手坐标系, Y轴是短边, X轴是长边
477 for (int y = 0; y < board_size.height; y++) //8
478 for (int x = 0; x < board_size.width; x++) //11
479 {
480 board_points.push_back(cv::Point3f(step_length*x, step_length*y, 0.0f));
481 }
482
483 // 检测角点
484 cv::Mat infrared_gray;
485 cv::cvtColor(infrared_image, infrared_gray, cv::COLOR_BGR2GRAY);
486 cv::imshow("infrared_gray", infrared_gray);
487 //cv::waitKey(0);
488
489
490 is_found = cv::findChessboardCorners( infrared_gray, board_size, image_corners, cv::CALIB_CB_ADAPTIVE_TH
491 b r e a k ;
492
493
494 is_found = cv::findCirclesGrid(255 - infrared_gray, board_size, image_corners, cv::CALIB_CB_SYMMETRIC_GR
495
496
497 if (image_corners[0].x > image_corners.back().x)
498 {
499 //交换列
500 for (int i = 0; i < (int)board_size.height; i++) //行
501 for (int j = 0; j < (int)board_size.width / 2; j++) //列
502 std::swap(image_corners[i*board_size.width + j], image_corners[(i + 1)*board_size.width - j - 1
503 }
504 if (image_corners[0].y > image_corners.back().y)
505 {
506 //交换行
507 for (int i = 0; i < (int)board_size.width; i++) //列
508 for (int j = 0; j < (int)board_size.height / 2; j++) //行
509 std::swap(image_corners[j*board_size.width + i], image_corners[(board_size.height - j - 1)*board
510 }
511

```

```
512 cv::Mat rvec,tvec;
513
514 cv::solvePnP(Ransac(board_points,image_corners,intrinsic_matrix,dist_coeffs,rvec,pnp_t);
515 cv::Mat R_;
516 cv::Rodrigues(rvec,pnp_R);
517 std::cout<<" PNP R_ "<<R_<<" t" <<tvec<<std::endl;
518
519
520 }
```

这篇文章知识点与官方知识档案匹配，可进一步学习相关知识

OpenCV技能树 > 首页 > 概览 20740 人正在系统学习中

机器人视觉伺服中的**手眼标定**和单目定位实现代码 04-13
机器人视觉伺服中的**手眼标定**和单目定位实现代码

虽迟但到，**手眼标定代码实现篇** 生活不止眼前的苟且 6205
你好，我是小鱼。今天周末，在小仙女带领下，剪了个帅气发型。今天说说**手眼标定**的代码实现。之前介绍过**手眼标定**算法Tsai的原理，今天介绍算法的...

5 条评论 weixin_47566038 热评 B2c.h怎么弄 写评论

手眼标定eye-in-hand(一)AX=XB方程推导_白水煮蝎子的博客 9-11
手眼标定eye-in-hand(一)AX=XB方程推导 1. 前言 本文主要介绍**eye-in-hand**的**手眼标定**.引用OpenCV的文档原话和原图1:"The following picture describes...

机器人**手眼标定**_慢下去、静下来的博客 9-8
Opencv在**Opencv4**以后有一个专门用于**手眼标定**的函数calibrateHandeye() .直接调用即可.里头集成了多中求解AX=XB的方法,Tsai只是其中的一种.对...

手眼标定实战(二)-基于**opencv**的**Eye to Hand**相机标定 向阳而生 6237
在手眼系统的坐标系统理论变换中，从像素坐标到机器人坐标系统转换过程中需要经过内参矫正、外参矩阵平移旋转的变换。本章将进一步讨论手眼系统在...

今晚开课！深度剖析面向机器人领域的**3D**激光SLAM技术原理、**代码**与**实战** 最新发布 3D视觉工坊 77
激光slam中，LOAM(Lidar Odometry and Mapping in Real-time)系列具有举足轻重的地位。下面的表格是近年来LOAM系列算法的汇总。算法名称发表名...

...**eye in hand**_**tsai****手眼标定代码 opencv_十年一梦**实验室的博客-CSDN... 9-3
【机器人**手眼标定**】**eye in hand** /"hand-eye calibration using TSAl method"/ #include<stdlib.h> #include<iostream> #include<fstream> #include<**opencv**...

...**手眼标定**_**机械臂运动**_ **opencv** **手眼标定** 做人嘛最重要的是开心啦的... 9-6
几种情况都试了几下,发现需要求逆的是**eye to hand**中的bTg,也就是机器人末端位姿取个逆,或许这里能解释广大的csdn友**eye in hand**中机器人末端矩阵不...

手眼标定详述(坐标系介绍, 二维、三维的**手眼标定**方法@九点法, AX=XB) m0_52785249的博客 4890
写在前面 **手眼标定**基本分类 **手眼标定**坐标系 眼在手外 (EYE TO HEAD) 眼在手上 (EYE IN HEAD) 九点法 (二维) - 算法**实现**流程 AX=XB方法 (二维...

机器人抓取 (五) —— **手眼标定** hand eye calibration zxxRobot的博客 7976
1. 相机标定 1.1 相机内参 2. 机器人**手眼标定**

opencv 图像上画出目标运动的轨迹_基于**opencv**的单目和双目标定平台手眼... 9-10
Eye-in-Hand手眼系统中,视觉传感器固定在末端执行器上,随机械臂一起运动。视觉系统的视野大,且**视觉**传感器识别和定位的误差会随着机械臂靠近目标物...

使用**opencv-python**进行**手眼标定**_qq_43607792的博客 8-28
opencv-python参考文档里面关于这一块资料较少,函数的输入条件也试了好半天才试明白,归根结底还是要懂这个**eye-in-hand**的原理。

手眼标定 向暖阳的博客 181
根据相机固定位置不同可分为两种情况,一种是相机固定在机械臂上,称之为“眼在手” (eye in hand), 另外一种一种是相机固定在独立支架上,称为“眼在外”...

经典**手眼标定**算法C++**代码** 06-09
经典**手眼标定**算法C++**代码**,程序是基于OpenCV 2.0以上版本,下载程序后需要配置OpenCV。工程主要包括三个文件, handeye.h为各种**手眼标定**的实...

手眼标定(eye in hand),求解 AX=XB 小记_opencv **手眼标定**_Quelquefois... 9-12
用于EYE IN HAND 理论结束,代码如下: #include<vector> #include<opencv2/opencv.hpp> using namespace std; vector<cv::Mat> RT_Topij/ /movement of ...

3D手眼标定1 (原理) qq_58220938的博客 2538
说明: 3D视觉机器人是配备有3D视觉相机的机械臂,能够观测场景的3D信息,以3D点云的形式交给机械臂,可以用于物体抓取、无序分拣、装配、打磨...

OpenCV**手眼标定** (calibrateHandeye()) 热门推荐 hello 3万+
文章目录说明Code实验效果参考 说明 Code #include <opencv2/opencv.hpp> #include <iostream> #include <math.h> using namespace std; using nam...

python **手眼标定**OpenCV**手眼标定** (calibrateHandeye()) 一 weixin_43134049的博客 1万+
以下**代码**来源 本篇博客通过该**代码**,附上记录的公式与查找连接,方便以后调用能弄懂各个参数的意思 1.参数说明 calibrateHandeye() 参数描述如下: R...

03第三课 **手眼标定**之**3D**位姿 zip 10-24
halcon软件开发包基础上做机器人的**手眼标定**,涉及许多专业知识,在这些讲义里详细讲解了标定的数学基础,原理等,以及**实现**手段,3D标定

新的单目视觉系统两步**手眼标定**方法 05-06
为了实现单目视觉系统的快速、精确的**手眼标定**,本文提出了一种新的两步式**手眼标定**方法,将**手眼标定**分为求解旋转关系和平移关系两步.首先机器人携...

三维机器视觉_多目相机标定_机器人**手眼标定**.pdf 07-04
三维视觉位姿转换原理,多目立体视觉原理,多目相机标定,机器人**手眼标定**。部分例题基于HALCON讲解。

利用SolvePnP求解相机位姿,并且验证9点**手眼标定**法 Alonewaiting 1690
9点标定法 用来将相机坐标系转到一个已知坐标系的方法,也就是求相机坐标系到已知世界坐标系下的坐标变换。如图所示,寻找R,T即为所求目标。Sol...

手眼标定 (眼在手外,眼在手上**代码**) m0_47433284的博客 676
分享一种可以快速求解眼在手上跟眼在手外的C++**代码**,直接就可以计算出**手眼**矩阵的。

机械臂**手眼标定**原理及**代码** weixin_42156097的博客 1万+
描述 本文将简要介绍机械臂**手眼标定**原理及相关知识 基础知识 了解**手眼标定**原理,就必须先了解一句话,叫做“右乘连体左乘基”这句话实际上是谈硕士...

C++**手眼标定**opencv**代码** 06-01
下面是基于OpenCV**实现手眼标定**的C++**代码**示例: ``C++ #include <iostream> #include <opencv2/opencv.hpp> using namespace std; using namespa...

“相关推荐”对你有帮助？
😞 非常没帮助 😞 没帮助 😞 一般 😊 有帮助 😄 非常有帮助

关于我们 招贤纳士 商务合作 寻求报道 400-660-0108 kefu@csdn.net 在线客服 工作时间 8:30-22:00

z504727099 关注 10 95 5 专栏目录