Jindong_Shi
2021-10-12

第 4 节课习题

2. 图像去畸变

源文件 undistort_image.cpp 和可执行文件位于文件夹 image_correction 内，部分代码如下:

```
double u_centre = rows/2 - 1;              //先计算图像中心点，因为图像中心点(u_centre, v_centre)对应成像平面坐标系原点
double v_centre = cols/2 - 1;
// 计算去畸变后图像的内容
for (int v = 0; v < rows; v++)
    for (int u = 0; u < cols; u++) {

        double u_distorted = 0, v_distorted = 0;
        // TODO 按照公式，计算点(u,v)对应到畸变图像中的坐标(u_distorted, v_distorted) (~6 lines)
        // start your code here
        double x = (u - cx)/fx;                                                    // x,y表示去畸变图像(u,v)点对应的归一化成像平面上的坐标(深度Z=1)
        double y = (v - cy)/fy;
        double r = sqrt(pow((u - u_centre)/fx,2) + pow((v - v_centre)/fy,2));       // r表示该点离归一化成像平面坐标系原点的距离
        double x_distorted = x*(1 + k1*r*r + k2*pow(r,4)) + 2*p1*x*y + p2*(r*r + 2*x*x);     //(x,y)对应到畸变图像中的成像平面上的坐标(x_distorted, y_distorted)
        double y_distorted = y*(1 + k1*r*r + k2*pow(r,4)) + p1*(r*r + 2*y*y) + 2*p2*x*y;
        u_distorted = fx*x_distorted + cx;       // 计算点(x_distorted, y_distorted)对应到畸变图像中的坐标(u_distorted, v_distorted)
        v_distorted = fy*y_distorted + cy;
        // end your code here

        // 赋值（最近邻插值）
        if (u_distorted >= 0 && v_distorted >= 0 && u_distorted < cols && v_distorted < rows) {
            image_undistort.at<uchar>(v, u) = image.at<uchar>((int) v_distorted, (int) u_distorted);
        } else {
            image_undistort.at<uchar>(v, u) = 0;
        }
    }

// 画图去畸变后图像
cv::imshow("image undistorted", image_undistort);
cv::waitKey();
```

去畸变图像如下图所示:

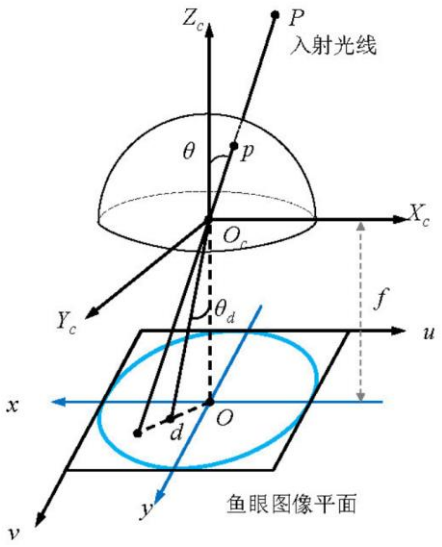
3. 鱼眼模型与去畸变

3.1 鱼眼相机相比于普通针孔相机在 SLAM 方面的优势

近年来,在复杂、大规模的室内外环境中,人们越来越关注相机位姿估计方法的实时性、通用性和可操作性。其中相机的视场角起着重要的作用, 特别是在具有挑战性的室内场景中, 往往是通过鱼眼镜头或相机镜头组合来增加视场角。鱼眼相机在 SLAM 应用当中相比于普通针孔相机可以捕获更丰富的环境特征信息, 从而获得更鲁棒的位姿跟踪效果[1]。

3.2 OpenCV 中使用的鱼眼畸变模型(等距投影)的定义

参考 OpenCV 文档[2]我们有以下鱼眼畸变模型[3]:



对于三维空间中任意点 P 在世界坐标系下的坐标值为$[X_w, Y_w, Z_w]$, 我们可以通过平移向量 T 和旋转矩阵 R 转换到相机坐标系下:

$$[X_c, Y_c, Z_c]^T = R \cdot [X_w, Y_w, Z_w]^T + T$$

再将其转换到相机归一化三维成像平面上(深度为 1):

$$[x, y, 1]^T = [X_c/Z_c, Y_c/Z_c, 1]^T$$

则鱼眼半球体截面半径 r 为:

$$r = sqrt(x \cdot x + y \cdot y)$$

点 P 在鱼眼半球体上的投影与光心的连线和光轴的夹角θ为:

$$\theta = atan(r)$$

利用畸变模型求出发生畸变后θ对应的夹角θ_d:

$$\theta_d = \theta \cdot (1 + k1 \cdot \theta^2 + k2 \cdot \theta^4 + k3 \cdot \theta^6 + k4 \cdot \theta^8)$$

利用等距投影模型 ($r = f \cdot \theta$) 可知成像点的半径 r 与入射角 θ 成正比，故可利用带畸变的夹角θ_d求出点 (x, y, 1) 对应的带畸变的相机归一化三维成像平面上的坐标 (x_d, y_d, 1):

$$x_d = (\theta_d/r) \cdot x$$

$$y_d = (\theta_d/r) \cdot y$$

利用带畸变的夹角θ_d，将带畸变的相机归一化三维成像平面上的坐标(x_d, y_d, 1)重新投影到二维图像平面上得到(u_d, v_d):

$$u_d = fx \cdot (x_d + \alpha \cdot y_d) + cx$$ // 参数α表示偏度系数 (skew coefficient)，默认为 0

$$v_d = fy \cdot y_d + cy$$

3.3 源文件 fisheye.cpp 和可执行文件位于文件夹 fisheye 内, 部分代码如下:

```
// 计算去畸变后图像的内容
for (int v = 0; v < rows; v++)
  for (int u = 0; u < cols; u++) {

    double u_distorted = 0, v_distorted = 0;
    // TODO 按照公式，计算点(u,v)对应到畸变图像中的坐标(u_distorted,
    // v_distorted) (~6 lines)
    // start your code here
    double x = (u - cx)/fx;                         // x,y表示去畸变图像(u,v)点对应的相机归一化三维成像平面上的坐标(深度Z=1)
    double y = (v - cy)/fy;
    double r = sqrt(x * x + y * y);                 // r为鱼眼半球体截面半径
    double theta = atan(r);                         // theta为鱼眼半球体上一点与光心的连线和光轴的夹角
    // 利用畸变模型求出发生畸变后theta对应的夹角theta_distorted
    double theta_distorted = theta * (1 + k1 * theta * theta + k2 * pow(theta,4) + k3 * pow(theta,6) + k4 * pow(theta,8));
    // 利用带畸变的夹角theta_distorted求出点(x, y, 1)对应的带畸变的相机归一化三维成像平面上的坐标(x_distorted, y_distorted, 1)
    double x_distorted = (theta_distorted/r) * x;
    double y_distorted = (theta_distorted/r) * y;
    // 利用带畸变的夹角theta_distorted, 将带畸变的相机归一化三维成像平面上的坐标(x_distorted, y_distorted, 1)重新投影到二维图像平面上得到(u_distorted, v_distorted)
    // u_distorted = fx * (x_distorted + alpha * y_distorted) + cx;
    u_distorted = fx * x_distorted + cx;        // 令输入参数alpha = 0
    v_distorted = fy * y_distorted + cy;
    // end your code here
```

去畸变后的图像如下所示:

3.4 令畸变参数 k1, ..., k4 = 0 依然可以产生畸变效果，因为该鱼眼畸变模型中只存在径向畸变，不存在切向畸变，也就是说点 P 在鱼眼半球体上的投影与光心的连线和光轴的夹角θ经过光心后依然保持不变或变化很小，光线入射该鱼眼半球几乎不发生折射。

3.5 鱼眼模型中去畸变会带来图像内容的损失，因为鱼眼模型的畸变为桶形畸变，远离图像中心的地方成像放大率小，因此越远离图像中心的位置畸变程度越明显，像点越向内移动。畸变校正后，原本挤在一起的像素点们被校正到原来的位置，导致四周的像素被拉伸，会造成四周出现模糊的情况。如何避免: 尽量让 fx = fy?

4. 双目视差的使用

理论部分:

1) 已知 XZY 推导像素点 (u, v) 对应的视差 d (d 的单位为像素):

根据内参数矩阵构成的等式我们有: (其中双目的基线为 b, 另外α, β为 u 和 v 方向上的缩放)

$$u = f\alpha\frac{X}{Z} + c_x = f_x\frac{X}{Z} + c_x$$
$$v = f\beta\frac{Y}{Z} + c_y = f_y\frac{Y}{Z} + c_y$$

视差为

$$d = \frac{f\alpha b}{Z} = \frac{f_x b}{Z}$$

已知 u,v,d 推导 XZY:

像素点 (u, v) 对应的视差为 d (d 的单位为像素), 则该点对应的深度为

$$Z = \frac{fb}{d/\alpha} = \frac{f\alpha b}{d} = \frac{f_x b}{d}$$

根据内参数矩阵构成的等式我们有:

$$X = \frac{(u - c_x)Z}{f_x}$$
$$Y = \frac{(v - c_y)Z}{f_y}$$

2) 右目相机下该模型相对于左目相机下的模型整体水平向左平移了基线长度 b。

编程部分:

源文件 disparity.cpp 和可执行文件位于文件夹 binocular_stereo 内, 部分代码如下:

```
// 生成点云
vector<Vector4d, Eigen::aligned_allocator<Vector4d>> pointcloud;

// TODO 根据双目模型计算点云
// 如果你的机器慢，请把后面的v++和u++改成v+=2, u+=2
for (int v = 0; v < left.rows; v++)
    for (int u = 0; u < left.cols; u++) {

        Vector4d point(0, 0, 0, left.at<uchar>(v, u) / 255.0); // 前三维为xyz,第四维为颜色

        // start your code here (~6 lines)
        // 根据双目模型计算 point 的位置
        double disp = disparity.at<uchar>(v,u);            //disp为从视差图得到的该点的视差(单位为像素)
        double Z = fx*d/disp;                              //Z为该点的深度
        double X = (u - cx)*Z/fx;
        double Y = (v - cy)*Z/fy;
        point(0) = X;
        point(1) = Y;
        point(2) = Z;
        pointcloud.push_back(point);
        // end your code here
    }

// 画出点云
showPointCloud(pointcloud);
return 0;
```

所有点云均基于左图和视差图计算得出，生成点云如下图所示:

5. 矩阵运算微分

设变量为$x \in \mathbb{R}^N$， 那么

5.1 矩阵$A \in \mathbb{R}^{N\times N}$, 那么$d(Ax)/dx$是什么？(注意: 严格的写法必须对行向量求导，即$d(Ax)/dx^T$, 但是我们一般省略这个T)

$$\frac{d(Ax)}{dx} = \frac{Adx}{dx} = A^T$$

5.2 矩阵$A \in \mathbb{R}^{N\times N}$, 那么$d(x^TAx)/dx$是什么?

$$\frac{d(x^TAx)}{dx} = \frac{d(x^T)Ax}{dx} + \frac{x^TAdx}{dx} = Ax + (x^TA)^T = (A + A^T)x$$

5.3 证明: $x^TAx = tr(Axx^T)$

根据矩阵的迹的公式，我们有 $tr(ABC) = tr(BCA) = tr(CAB)$

所以 $tr(Axx^T) = tr(x^TAx)$, 由于 x^TAx 是一个标量，所以 $tr(x^TAx) = x^TAx$

所以 $tr(Axx^T) = x^TAx$ 成立

6. 高斯牛顿的曲线拟合实验

源文件 gaussnewton.cpp 和可执行文件位于文件夹 gauss_newton 内, 部分代码如下:

```
for (int iter = 0; iter < iterations; iter++) {

    Matrix3d H = Matrix3d::Zero();             // Hessian = J^T J in Gauss-Newton
    Vector3d b = Vector3d::Zero();             // bias
    cost = 0;

    // 第 2 步: 对于第iter次迭代，求出雅可比矩阵和误差
    for (int i = 0; i < N; i++) {
        double xi = x_data[i], yi = y_data[i];  // 第i个数据点
        double error = yi - exp(ae * xi * xi + be * xi + ce);      // 第i个数据点的计算误差
        Vector3d J = Vector3d::Zero(); // 雅可比矩阵
        J[0] = -xi * xi * exp(ae * xi * xi + be * xi + ce);  // de/da
        J[1] = -xi * exp(ae * xi * xi + be * xi + ce);  // de/db
        J[2] = -exp(ae * xi * xi + be * xi + ce);  // de/dc
        H += J * J.transpose(); // GN近似的H
        b += -error * J;
        cost += error * error;
    }

    // 第 3 步: 对于第iter次迭代，求解增量方程 H*dx=b，由于这里H为半正定矩阵，故用ldlt()求解方程组
    Vector3d dx = H.ldlt().solve(b);
```

(小技巧: 为了计算每一个迭代步骤下最终的 Hessian 矩阵 H 和 bias 向量 b，将每一组数据 (i = 0 ~99) 对应计算得到的的 H_i 和 b_i 分别进行累加。)

程序计算得到的最终估计参数如下:

这和书中的结果是拟合的。

7. 批量最大似然估计

7.1:

$$\underbrace{\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}}_{z} = \underbrace{\begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{H} \underbrace{\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x} \quad => \quad e = z - Hx$$

7.2:

由最大似然估计(书中式 6.10)可知, 若噪声之间互不相关, 则W为对角矩阵，对角元素包含每次输入噪声和测量噪声的方差, 本例中为:

$$W = \begin{bmatrix} Q & 0 & 0 & 0 & 0 & 0 \\ 0 & Q & 0 & 0 & 0 & 0 \\ 0 & 0 & Q & 0 & 0 & 0 \\ 0 & 0 & 0 & R & 0 & 0 \\ 0 & 0 & 0 & 0 & R & 0 \\ 0 & 0 & 0 & 0 & 0 & R \end{bmatrix}$$

7.3:

x 的最优估计值 $\hat{x}$ 可由最小二乘解 (通过对$z = Hx$ 等式两边左乘伪逆) 得出:

$$H^T W^{-1} H x = H^T W^{-1} z$$

显然当$H^T W^{-1} H$满秩时(也就是H列满秩时)，x存在唯一解: $\hat{x} = (H^T W^{-1} H)^{-1} H^T W^{-1} z$

判断H是否列满秩比较麻烦，另一种方法是通过系统矩阵A和观测矩阵C, 计算该系统的能观性矩阵O:

$$O = \begin{bmatrix} C \\ CA \\ \dots \\ CA^{N-1} \end{bmatrix}$$

若$rank(O) = N$, 且$Q > 0, R > 0$, 则估计值 $\hat{x}$ 存在唯一解[4]。

本例中系统矩阵$A = 1$，观测矩阵$C = 1$，状态量个数$N = 1$, 则 $O = C = 1$, 满足$rank(O) = N$，故该系统估计值$\hat{x}$ 存在唯一解。

由此可在系统能观性和$H^T W^{-1} H$的可逆性之间建立联系。

参考文献:

[1] 鱼眼摄像头 SLAM
https://zhuanlan.zhihu.com/p/137114715
[2] 鱼眼相机 OpenCV 文档
https://docs.opencv.org/3.4/db/d58/group__calib3d__fisheye.html
[3] 鱼眼相机的成像及畸变校正模型研究, 知识研究学报 Vol. 1 No. 1, Jan. 2020
[4] State Estimation For Robotics, Timothy D. Barfoot.