第四讲作业

崔华坤

达闼科技（北京）有限公司

2018.3.21

二、图像去畸变

```
// 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_ud = (u-cx)/fx;
        double y_ud = (v-cy)/fy;
        double r2 = x_ud*x_ud + y_ud*y_ud;
        double r4 = r2 * r2;
        double x_d = x_ud*(1+k1*r2+k2*r4)+2*p1*x_ud*y_ud+p2*(r2+2*x_ud*x_ud);
        double y_d = y_ud*(1+k1*r2+k2*r4)+p1*(r2+2*y_ud*y_ud)+2*p2*x_ud*y_ud;

        u_distorted = fx*x_d + cx;
        v_distorted = fy*y_d + 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;
        }
    }
```

三 双目视差的使用

答：投影公式为$ZP_{uv} = KP_c$，如下所示：

$$Z\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

则有：

$$x = \frac{(u - c_x)Z}{f_x}$$

$$y = \frac{(v - c_y)Z}{f_y}$$

```
int main(int argc, char **argv) {

    // 内参
    double fx = 718.856, fy = 718.856, cx = 607.1928, cy = 185.2157;
    // 间距
    double b = 0.573;

    // 读取图像
    cv::Mat left = cv::imread(left_file, 0);
    cv::Mat right = cv::imread(right_file, 0);
    cv::Mat disparity = cv::imread(disparity_file, 0); // disparty 为CV_8U,单位为像素
    cout<<"depth w= "<<disparity.cols<<" h= "<<disparity.rows<<" t= "<<disparity.type()<<endl;
    cv::imshow("left",left);
    cv::imshow("disparity",disparity);
    cv::waitKey();
    // 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
            uchar d = disparity.at<uchar>(v,u);
            point[2] = fx * b / d;
            point[0] = (u-cx)*point[2]/fx;
            point[1] = (v-cy)*point[2]/fy;
            //cout<<"point: "<<point<<endl;
            pointcloud.push_back(point);
            // end your code here
        }
    cout<<"pointcloud size: "<<pointcloud.size()<<endl;
    // 画出点云
    showPointCloud(pointcloud);
    return 0;
}
```

四、矩阵微分

1. 答：

$$d(Ax) = Adx = \left(\frac{\partial(Ax)}{\partial x}\right)^T dx$$

$$\frac{\partial(Ax)}{\partial x} = A^T$$

2. 令$f = x^T Ax$，则有：

$$df = (dx)^T Ax + x^T Adx = x^T A^T dx + x^T Adx = x^T(A^T + A)dx = \left(\frac{\partial f}{\partial x}\right)^T dx$$

$$\frac{\partial f}{\partial x} = [x^T(A^T + A)]^T = (A^T + A)x$$

3. 证明：

$$x^T \mathrm{A}^T x = \mathrm{tr}(Axx^T)$$

因为$Ax \in \mathbb{R}^{N\times 1}$，$x^T \in \mathbb{R}^{1\times N}$，相乘展开易得：对角线之和，即$\mathrm{tr}(Axx^T)$，等于两个向量转置后相乘，即：

$$\mathrm{tr}(Axx^T) = (Ax)^T x = x^T \mathrm{A}^T x$$

五、高斯牛顿法的曲线拟合实验

```
// 开始Gauss-Newton迭代
int iterations = 100;    // 迭代次数
double cost = 0, lastCost = 0;  // 本次迭代的cost和上一次迭代的cost

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

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

    for (int i = 0; i < N; i++) {
        double xi = x_data[i], yi = y_data[i];  // 第i个数据点
        // start your code here
        double error = 0;   // 第i个数据点的计算误差
        error = yi - exp(ae * xi * xi + be * xi + ce); // 填写计算error的表达式
        Vector3d J; // 雅可比矩阵
        J[0] = -exp(ae * xi * xi + be * xi + ce) * xi * xi;  // de/da
        J[1] = -exp(ae * xi * xi + be * xi + ce) * xi;  // de/db
        J[2] = -exp(ae * xi * xi + be * xi + ce);  // de/dc
        H += J * J.transpose(); // GN近似的H
        g += -error * J;
        // end your code here
        cost += 0.5 * error * error;
    }
    // 求解线性方程 Hx=b，建议用ldlt
    // start your code here
    Vector3d dx;
    dx = H.ldlt().solve(g);
    //cout<<"iter "<<iter<<" H: \n"<<H.matrix()<<" \n g: "<<g.transpose()<<"\n dx: "<
    // end your code here

    if (isnan(dx[0])) {
        cout << "result is nan!" << endl;
        break;
    }

    if (iter > 0 && cost > lastCost) {
        // 误差增长了，说明近似的不够好
        cout << "cost: " << cost << ", last cost: " << lastCost << endl;
        break;
    }
```

```
[100%] Built target gauss-newton
stevencui@ubuntu:~/Project/SLAMCourse/l4-5-Gauss-Newton/build$ ./gauss-newton
iter 0 total cost: 1.59787e+06
iter 1 total cost: 188393
iter 2 total cost: 17836.8
iter 3 total cost: 1097.51
iter 4 total cost: 87.4266
iter 5 total cost: 51.3898
iter 6 total cost: 50.9686
dx[-0.00117081  0.00196749 -0.00081055] is small! stop iter
estimated abc = 0.890908, 2.1719, 0.943628
```

六、批量最大似然估计

批量状态变量为：$x = [x_0, x_1, x_2, x_3]^T$

批量观测为：$z = [x_0, v_1, v_2, v_3, y_1, y_2, y_3]^T$

1. $\boldsymbol{v_k = x_k - x_{k-1}}$

$$H^{7\times4}=\begin{bmatrix}1&0&0&0\\-1&1&0&0\\0&-1&1&0\\0&0&-1&1\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$$

2.

$$W^{7\times7}=\begin{bmatrix}0&&&&&&\\&Q&&&&&\\&&Q&&&&\\&&&Q&&&\\&&&&R&&\\&&&&&R&\\&&&&&&R\end{bmatrix}$$

3. 是否有唯一解？

答：令误差项为：

$$\mathrm{J}(x)=\frac{1}{2}(z-Hx)^TW^{-1}(z-Hx)$$

$$\mathrm{dJ}(x)=-(z-Hx)^TW^{-1}Hdx$$

$$\frac{\partial\mathrm{J}(x)}{\partial x}=[-(z-Hx)^TW^{-1}H]^T=-H^TW^{-1}(z-H\hat{x})=0$$

$$\Longrightarrow H^TW^{-1}H\hat{x}=H^TW^{-1}z$$

待求的 x 为四个自由度，其中初始状态x_0已知，所以需要保证$\mathrm{rank}(H^TW^{-1}H)=N(4)$，由于 Q、R 是对称正定的，所以 W^{-1} 也是对称正定的。

只需$\mathrm{rank}(H^TH)=\mathrm{rank}(H^T)=\mathrm{N}(4)$，将$H^T$展开之后易知是行满秩的，可知有唯一解$\hat{x}=(H^TW^{-1}H)^{-1}H^TW^{-1}z$