第四次作业

2图像去畸变

```
int main(int argc, char **argv)
{
    // 本程序需要你自己实现去畸变部分的代码。尽管我们可以调用OpenCV的去畸变，但自己实现一遍有助于理解。
    // 畸变参数
    double k1 = -0.28340811, k2 = 0.07395907, p1 = 0.00019359, p2 = 1.761
87114e-05;
    // 内参
    double fx = 458.654, fy = 457.296, cx = 367.215, cy = 248.375;
    cv::Mat image = cv::imread(image_file,0);   // 图像是灰度图，CV_8UC1
    int rows = image.rows; //x
    int cols = image.cols; //y
    cout<<"rows: "<<rows<<" clos: "<<cols<<endl;
    cv::imshow("image undistorted", image);
    cv::waitKey();
    cv::Mat image_undistort = cv::Mat(rows, cols, CV_8UC1);   // 去畸变以后的图
    // 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_d
istorted) (~6 lines)
            // start your code here
            double x_distorted = 0 , y_distorted = 0;
            double x = 0, y = 0, r = 0;
            x = (v - cx)/fx;
            y = (u - cy)/fy;
            r = sqrt(x*x + y*y);

            x_distorted = x * ( 1 + k1*r*r + k2*pow(r,4) ) + 2*p1*x*y + p
2*(r*r + 2*x*x); //row
            y_distorted = y * ( 1 + k1*r*r + k2*pow(r,4) ) + p1*(r*r + 2*
y*y) + 2*p2*x*y; //col

            v_distorted = fx * x_distorted + cx; //x
            u_distorted = fy * y_distorted + cy; //y
            // end your code here

            // 赋值 (最近邻插值)
            if (u_distorted >= 0 && v_distorted >= 0 && u_distorted < col
s && 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();
        return 0;
    }
```

3双目视差的使用

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

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

    // 间距
    double d = 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

    // 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++) // y
    {
        for (int u = 0; u < left.cols; u++) //x
        {
            //mat.at q
            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

           // unsigned char di = disparity.ptr<unsigned short> (v)[u]; // 视差图

           int di = disparity.ptr<uchar> (v)[u]; // 视差图
           // if ( d==0 ) continue; // 为0表示没有测量到
            // z = f *b / d
            point[2] =fx*d/di; //z ;d为间距
            point[0] = (u-cx)*point[2]/fx; //x
            point[1] = (v-cy)*point[2]/fy; //y
            pointcloud.push_back(point);
            // end your code here
        }
    }
    // 画出点云
    showPointCloud(pointcloud);
    return 0;
}
```

4矩阵微分

1. d(Ax)/dx是实值向量函数的行向量偏导数，称之为向量函数f(x)在x处的Jacobian矩阵
2. d(x^TAx)/dx是实值标量函数的行向量偏导数，称之为实值标量函数f(x)在x处的梯度向量

5高斯牛顿

```
int main(int argc, char **argv)
{
    double ar = 1.0, br = 2.0, cr = 1.0;         // 真实参数值
    double ae = 2.0, be = -1.0, ce = 5.0;        // 估计参数值
    int N = 100;                                 // 数据点
    double w_sigma = 1.0;                        // 噪声Sigma值
    cv::RNG rng;                                 // OpenCV随机数产生器

    vector<double> x_data, y_data;      // 数据
    for (int i = 0; i < N; i++)
    {
        double x = i / 100.0;
        x_data.push_back(x);
        y_data.push_back(exp(ar * x * x + br * x + cr) + rng.gaussian(w_s
igma));
    }

    // 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++) //迭代 100 times
    {

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

        for (int i = 0; i < N; i++)//N个数据点
        {
            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,f(x)
             // 填写计算error的表达式
            error = yi - (ae * xi * xi + be * xi + ce); //@liyubo
            Vector3d J; // 雅可比矩阵    f()的导数
            //分别求导
            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,transpose求转置
            b += -error * J;

            // end your code here
            cost += error * error; //本次迭代
```

```
        }
        // 求解线性方程 Hx=b，建议用ldlt
        // start your code here
        Vector3d dx;
        dx = H.ldlt().solve(b);    //ldlt Cholesky来解方程

        // 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 << en
dl;
            break;
        }
        // 更新abc估计值
        ae += dx[0];
        be += dx[1];
        ce += dx[2];

        lastCost = cost;
        cout<<"ae: "<<ae<<" be:"<<be<<" ce:"<<ce<<endl;
        cout << "total cost: " << cost << endl;
    }

    cout << "estimated abc = " << ae << ", " << be << ", " << ce << endl;
    return 0;
```