0. 说明

本PDF文档为自动生成，如有遗漏的格式错误请及时告知！

---

1. 图像去畸变

- undistort_image.cpp

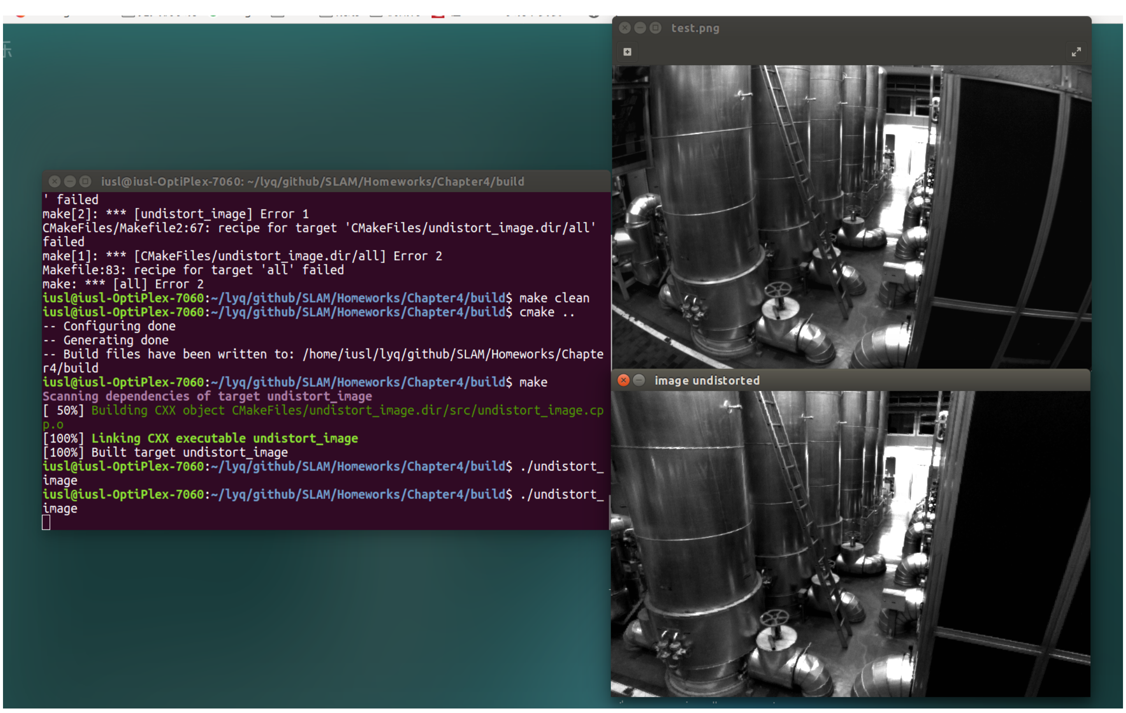
```
#include <opencv2/opencv.hpp>
#include <string>

using namespace std;

string image_file = "./test.png";   // 请确保路径正确

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 = 1.76187114e-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, cols = image.cols;
    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_distorted) (~6 lines)
            // start your code here
            double x = (u-cx)/fx;
            double y = (v-cy)/fy;

            double r_square =  x*x + y*y;

            double x_distorted = x*(1+k1*r_square+k2*r_square*r_square)+2*p1*x*y+p2*(r_square+2*x*x);
            double y_distorted = y*(1+k1*r_square+k2*r_square*r_square)+p1*(r_square+2*y*y)+2*p2*x*y;

            u_distorted = fx*x_distorted+cx;
            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();

    return 0;
}
```

- CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
SET(CMAKE_BUILD_TYPE "Release")
PROJECT (Chapter4)

add_compile_options(-std=c++11)

INCLUDE_DIRECTORIES(${PROJECT_SOURCE_DIR}/include)
INCLUDE_DIRECTORIES("/usr/include/opencv2")

find_package( OpenCV REQUIRED )


SET(SRC_LIST ${PROJECT_SOURCE_DIR}/src/undistort_image.cpp)
ADD_EXECUTABLE(undistort_image ${SRC_LIST})
target_link_libraries(undistort_image ${OpenCV_LIBRARIES})
```

- 运行结果



2. 双目视差的使用

- disparity.cpp

```
#include <opencv2/opencv.hpp>
#include <string>
#include <Eigen/Core>
#include <pangolin/pangolin.h>
#include <unistd.h>

using namespace std;
using namespace Eigen;

// 文件路径，如果不对，请调整
string left_file = "./left.png";
string right_file = "./right.png";
string disparity_file = "./disparity.png";

// 在panglin中画图，已写好，无需调整
void showPointCloud(const vector<Vector4d,
Eigen::aligned_allocator<Vector4d>> &pointcloud);

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++)
        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
            unsigned int dis = disparity.ptr<unsigned short>(v)[u];

            if(dis == 0)
                continue;

            double x = (u-cx)/fx;
```

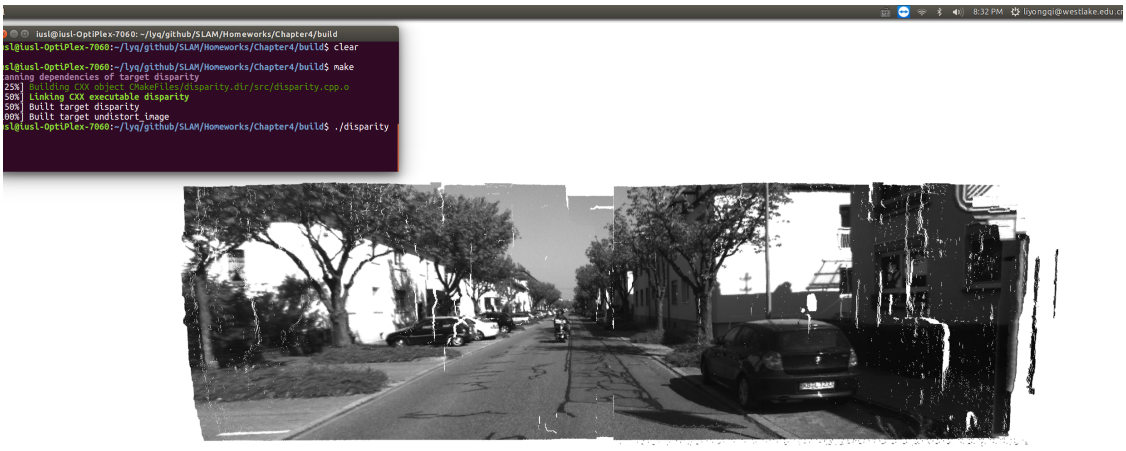
```
            double y = (v-cy)/fy;
            double z = (fx*d*1000)/dis;

            point[2] = z;
            point[0] = x*z;
            point[1] = y*z;

            pointcloud.push_back(point);
            // end your code here
        }

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

void showPointCloud(const vector<Vector4d,
Eigen::aligned_allocator<Vector4d>> &pointcloud) {

    if (pointcloud.empty()) {
        cerr << "Point cloud is empty!" << endl;
        return;
    }

    pangolin::CreateWindowAndBind("Point Cloud Viewer", 1024, 768);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    pangolin::OpenGlRenderState s_cam(
            pangolin::ProjectionMatrix(1024, 768, 500, 500, 512, 389, 0.1,
1000),
            pangolin::ModelViewLookAt(0, -0.1, -1.8, 0, 0, 0, 0.0, -1.0,
0.0)
    );

    pangolin::View &d_cam = pangolin::CreateDisplay()
            .SetBounds(0.0, 1.0, pangolin::Attach::Pix(175), 1.0, -1024.0f /
768.0f)
            .SetHandler(new pangolin::Handler3D(s_cam));

    while (pangolin::ShouldQuit() == false) {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        d_cam.Activate(s_cam);
        glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

        glPointSize(2);
        glBegin(GL_POINTS);
        for (auto &p: pointcloud) {
            glColor3f(p[3], p[3], p[3]);
            glVertex3d(p[0], p[1], p[2]);
        }
        glEnd();
        pangolin::FinishFrame();
        usleep(5000);   // sleep 5 ms
    }
    return;
```

```
}
```

- CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
SET(CMAKE_BUILD_TYPE "Release")
PROJECT (Chapter4)

add_compile_options(-std=c++11)

INCLUDE_DIRECTORIES(${PROJECT_SOURCE_DIR}/include)
INCLUDE_DIRECTORIES("/usr/include/opencv2")

find_package( OpenCV REQUIRED )

find_package(Pangolin REQUIRED)
INCLUDE_DIRECTORIES(${Pangolin_INCLUDE_DIRS})

SET(SRC_LIST ${PROJECT_SOURCE_DIR}/src/undistort_image.cpp)
ADD_EXECUTABLE(undistort_image ${SRC_LIST})
target_link_libraries(undistort_image ${OpenCV_LIBRARIES})


ADD_EXECUTABLE(disparity ${PROJECT_SOURCE_DIR}/src/disparity.cpp)
target_link_libraries(disparity ${Pangolin_LIBRARIES} ${OpenCV_LIBRARIES})
```

- 运行结果



---

3. 矩阵运算微分

设变量为 $x \in \mathbb{R}^N$ ，那么：

问3-1：矩阵$A \in \mathbb{R}^{N \times N}$，那么$d(Ax)/dx$是什么？

答：$d(Ax)/dx = A^T$ ，即矩阵A的转置。

问3-2：矩阵$A \in \mathbb{R}^{N \times N}$，那么$d(x^T Ax)/dx$是什么？

答： $(A+A^T)x$

简单写一下证明思路：

$x^TAx = \sum_{k=1}^N \sum_{l=1}^N A_{kl}x_kx_l$

$[d(x^TAx)/dx]_i = \sum_{k=1}^N \sum_{l=1}^N A_{kl}x_kx_l/dx_i = \sum_{k=1}^N A_{ki}x_k + \sum_{l=1}^N A_{il}x_l$

$\therefore d(x^TAx)/dx = Ax + A^Tx = (A+A^T)x$

证3-3： $x^TAx = tr(Axx^T)$

证：

等 式 两 边 同 时 取 微 分 可 得 ： $d(x^TAx)/dx = d(tr(Axx^T))/dx$

由(问3-2)可知左边 = $(A+A^T)x$

$d(tr(Axx^T)) = tr(d(Axx^T)) = tr(A(dx)x^T + Ax(dx)^T) = tr(A(dx)x^T) + tr(Ax(dx)^T)$

$= tr(x^TAdx) + tr((Ax)^Tdx) = tr(x^T(A+A^T)dx)$

得到梯度矩阵：

$d(tr(Axx^T))/dx = (x^T(A+A^T))^T = (A+A^T)x$

综上可证。

---

4. 高斯牛顿法的曲线拟合实验

- gaussnewton.cpp

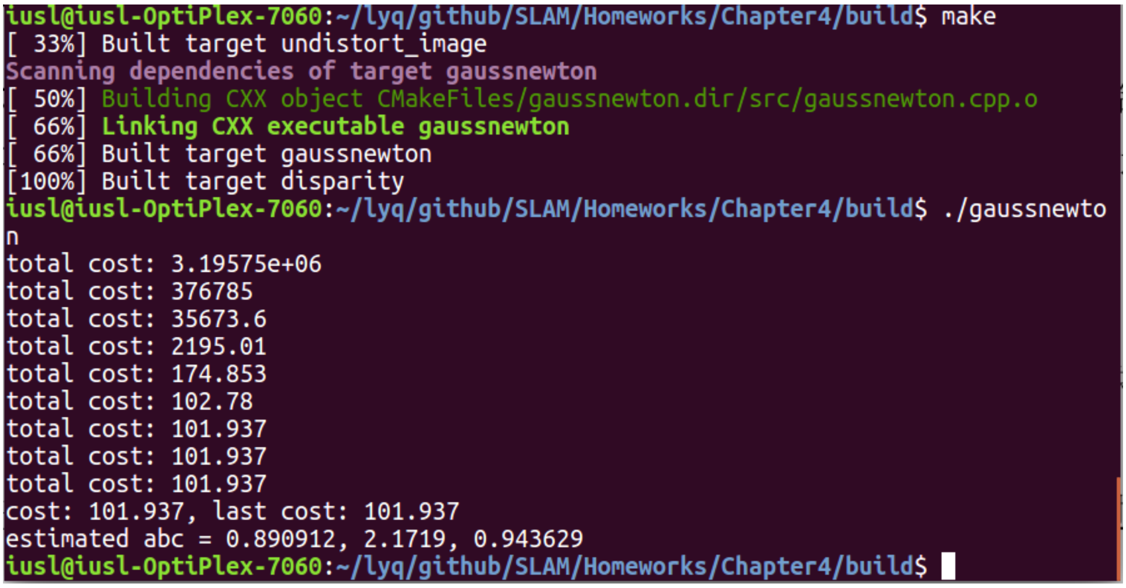
```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <Eigen/Core>
#include <Eigen/Dense>

using namespace std;
using namespace Eigen;

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_sigma));
    }

    // 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 b = 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] = -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;
            // end your code here

            cost += error * error;
        }

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

        // 更新abc估计值
        ae += dx[0];
        be += dx[1];
        ce += dx[2];

        lastCost = cost;

        cout << "total cost: " << cost << endl;
    }

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

- CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
SET(CMAKE_BUILD_TYPE "Release")
PROJECT (Chapter4)

add_compile_options(-std=c++11)

INCLUDE_DIRECTORIES(${PROJECT_SOURCE_DIR}/include)
INCLUDE_DIRECTORIES("/usr/include/opencv2")

find_package( OpenCV REQUIRED )

find_package(Pangolin REQUIRED)
INCLUDE_DIRECTORIES(${Pangolin_INCLUDE_DIRS})

SET(SRC_LIST ${PROJECT_SOURCE_DIR}/src/undistort_image.cpp)
ADD_EXECUTABLE(undistort_image ${SRC_LIST})
target_link_libraries(undistort_image ${OpenCV_LIBRARIES})


ADD_EXECUTABLE(disparity ${PROJECT_SOURCE_DIR}/src/disparity.cpp)
target_link_libraries(disparity ${Pangolin_LIBRARIES} ${OpenCV_LIBRARIES})

ADD_EXECUTABLE(gaussnewton ${PROJECT_SOURCE_DIR}/src/gaussnewton.cpp)
target_link_libraries(gaussnewton ${OpenCV_LIBRARIES})
```

- 运行结果



```
iusl@iusl-OptiPlex-7060:~/lyq/github/SLAM/Homeworks/Chapter4/build$ make
[ 33%] Built target undistort_image
Scanning dependencies of target gaussnewton
[ 50%] Building CXX object CMakeFiles/gaussnewton.dir/src/gaussnewton.cpp.o
[ 66%] Linking CXX executable gaussnewton
[ 66%] Built target gaussnewton
[100%] Built target disparity
iusl@iusl-OptiPlex-7060:~/lyq/github/SLAM/Homeworks/Chapter4/build$ ./gaussnewto
n
total cost: 3.19575e+06
total cost: 376785
total cost: 35673.6
total cost: 2195.01
total cost: 174.853
total cost: 102.78
total cost: 101.937
total cost: 101.937
total cost: 101.937
cost: 101.937, last cost: 101.937
estimated abc = 0.890912, 2.1719, 0.943629
iusl@iusl-OptiPlex-7060:~/lyq/github/SLAM/Homeworks/Chapter4/build$
```

---

5. 批量最大似然估计

问5-1：可以定义矩阵 H,使得批量误差为 e = z − Hx。请给出此处 H 的具体形式。

设 $e_{v,k} = v_k - (x_k - x_{k-1})$，$e_{y,k} = y_k - x_k$ 则：

$$
\begin{aligned}
e_{v,1} &= v_1 - (x_1 - x_0) \\
e_{v,2} &= v_2 - (x_2 - x_1) \\
e_{v,3} &= v_3 - (x_3 - x_2) \\
e_{y,1} &= y_1 - x_1 \\
e_{y,2} &= y_2 - x_2 \\
e_{y,3} &= y_3 - x_3
\end{aligned}
$$

化为矩阵形式可得：

$$
e = \begin{bmatrix} e_{v,1} \\ e_{v,2} \\ e_{v,3} \\ e_{v,4} \\ e_{v,5} \\ e_{v,6} \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} - \begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = z - Hx
$$

问5-2：最大似然估计问题转换为最小二乘问题：$x^* = argmin\frac{1}{2}(z-Hx)^T W^{-1}(z-Hx)$,请给出W的具体取值。

根据题意，W应该是协方差矩阵，即

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

问5-3：假设所有噪声相互无关,该问题存在唯一的解吗?若有,唯一解是什么?若没有,说明理由。

当且仅当$H^T W^{-1} H$ 可逆时，存在唯一解。

又因为W^{-1} 是实对称且正定的，所以这里我们只需要验证：

$rank(H^T H) = rank(H^T) = 4$

更具第一问中得到的矩阵H可以知道其满足要求，所以存在唯一解。