

## 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. LK 光流

### 问1-1：光流文献综述<sup>1</sup>

- 按照该文的分类方法，光流法可以分为哪几类？  
additive or compositional  
forwards or inverse
- 在compositional中，为什么有时候需要做原始图像的warp？该warp有何物理意义？
  - 需要在当前位姿估计之前引入增量式warp以建立半群约束要求；
  - warp的物理意义：对图像做微小的平移或者仿射变换。
- forward 和 inverse 有何差别？
  - forward对输入图像参数化，inverse同时对输入图像和模板图像参数化；
  - forward中Hessian是运动参数的函数，inverse中Hessian是固定的；
  - 目标函数不一样。

### 问1-2：forward-addtive Gauss-Newton 光流的实现

- optical\_flow.cpp的OpticalFlowSingleLevel函数

```
void OpticalFlowSingleLevel(  
    const Mat &img1,  
    const Mat &img2,  
    const vector<KeyPoint> &kp1,  
    vector<KeyPoint> &kp2,  
    vector<bool> &success,  
    bool inverse  
) {  
  
    // parameters  
    int half_patch_size = 4;  
    int iterations = 10;  
    bool have_initial = !kp2.empty();  
  
    for (size_t i = 0; i < kp1.size(); i++) {  
        auto kp = kp1[i];  
        double dx = 0, dy = 0; // dx,dy need to be estimated  
        if (have_initial) {  
            dx = kp2[i].pt.x - kp.pt.x;  
            dy = kp2[i].pt.y - kp.pt.y;  
        }  
  
        double cost = 0, lastCost = 0;  
        bool succ = true; // indicate if this point succeeded  
  
        // Gauss-Newton iterations
```

```

        for (int iter = 0; iter < iterations; iter++) {
            Eigen::Matrix2d H = Eigen::Matrix2d::Zero();
            Eigen::Vector2d b = Eigen::Vector2d::Zero();
            cost = 0;

            if (kp.pt.x + dx <= half_patch_size || kp.pt.x + dx >= img1.cols
- half_patch_size ||
                kp.pt.y + dy <= half_patch_size || kp.pt.y + dy >= img1.rows
- half_patch_size) { // go outside
                succ = false;
                break;
            }

            // compute cost and jacobian
            for (int x = -half_patch_size; x < half_patch_size; x++)
                for (int y = -half_patch_size; y < half_patch_size; y++) {

                    // TODO START YOUR CODE HERE (~8 lines)
                    double error = 0;
                    Eigen::Vector2d J; // Jacobian
                    if (inverse == false) {
                        // Forward Jacobian
                        J[0]=
(GetPixelValue(img2, kp.pt.x+dx+x+1, kp.pt.y+dy+y)-
GetPixelValue(img2, kp.pt.x+dx+x-1, kp.pt.y+dy+y))/2;
                        J[1]=
(GetPixelValue(img2, kp.pt.x+dx+x, kp.pt.y+dy+y+1)-
GetPixelValue(img2, kp.pt.x+dx+x, kp.pt.y+dy+y-1))/2;
                    } else {
                        // Inverse Jacobian
                        // NOTE this J does not change when dx, dy is
updated, so we can store it and only compute error
                    }

                    // compute H, b and set cost;
                    error = GetPixelValue(img2, kp.pt.x+dx+x, kp.pt.y+dy+y)-
GetPixelValue(img1, kp.pt.x+dx+x, kp.pt.y+dy+y);
                    H += J*J.transpose();
                    b += -J*error;
                    cost += error*error/2;
                    // TODO END YOUR CODE HERE
                }

            // compute update
            // TODO START YOUR CODE HERE (~1 lines)
            Eigen::Vector2d update;
            update = H.ldlt().solve(b);
            // TODO END YOUR CODE HERE

            if (isnan(update[0])) {
                // sometimes occurred when we have a black or white patch
and H is irreversible
                cout << "update is nan" << endl;
                succ = false;
                break;
            }
            if (iter > 0 && cost > lastCost) {

```

```

        cout << "cost increased: " << cost << ", " << lastCost <<
endl;

        break;
    }

    // update dx, dy
    dx += update[0];
    dy += update[1];
    lastCost = cost;
    succ = true;
}

success.push_back(succ);

// set kp2
if (have_initial) {
    kp2[i].pt = kp.pt + Point2f(dx, dy);
} else {
    KeyPoint tracked = kp;
    tracked.pt += cv::Point2f(dx, dy);
    kp2.push_back(tracked);
}
}
}

```

**注意：**因为OpenCV版本过低，没有GFTTDetector构造器，可以用goodFeaturesToTrack () 提取GFTT角点，然后转为KeyPoint.

- CMakeLists

```

cmake_minimum_required(VERSION 2.8.3)
SET(CMAKE_BUILD_TYPE "Release")
PROJECT (Chapter6)

add_compile_options(-std=c++11)

INCLUDE_DIRECTORIES(${PROJECT_SOURCE_DIR}/include)
INCLUDE_DIRECTORIES("/usr/include/opencv2")
INCLUDE_DIRECTORIES("/usr/include/eigen3")

find_package( OpenCV REQUIRED )

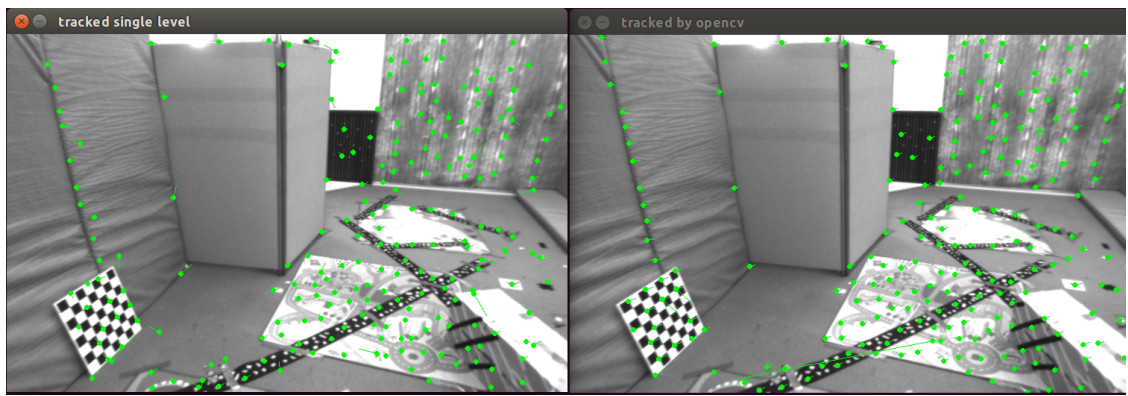
find_package(Sophus REQUIRED)
include_directories(${Sophus_INCLUDE_DIRS})

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

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

```

- 运行结果



- 问答
  - 从最小二乘角度来看,每个像素的误差怎么定义?

$$e(p) = I(W(x; p + \Delta p)) - T(x)$$

- 误差相对于自变量的导数如何定义?

$$\frac{\partial e}{\partial p} = \left[ \frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right]$$

### 问1-3：反向法

- 代码

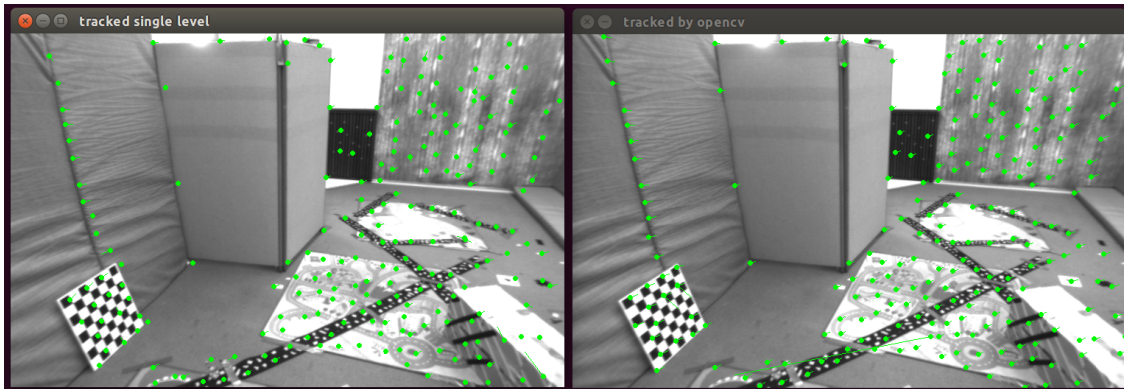
```
for (int x = -half_patch_size; x < half_patch_size; x++)
    for (int y = -half_patch_size; y < half_patch_size; y++) {

        // TODO START YOUR CODE HERE (~8 lines)
        double error = 0;
        Eigen::Vector2d J; // Jacobian
        if (inverse == false) {
            // Forward Jacobian
            J[0] =
                (GetPixelValue(img2, kp.pt.x+dx+x+1, kp.pt.y+dy+y) -
                 GetPixelValue(img2, kp.pt.x+dx+x-1, kp.pt.y+dy+y))/2;
            J[1] =
                (GetPixelValue(img2, kp.pt.x+dx+x, kp.pt.y+dy+y+1) -
                 GetPixelValue(img2, kp.pt.x+dx+x, kp.pt.y+dy+y-1))/2;
        } else {
            // Inverse Jacobian
            // NOTE this J does not change when dx, dy is
            // updated, so we can store it and only compute error
            J[0] = (GetPixelValue(img1, kp.pt.x+x+1, kp.pt.y+y) -
                   GetPixelValue(img1, kp.pt.x+x-1, kp.pt.y+y))/2;
            J[1] = (GetPixelValue(img1, kp.pt.x+x, kp.pt.y+y+1) -
                   GetPixelValue(img1, kp.pt.x+x, kp.pt.y+y-1))/2;
        }

        // compute H, b and set cost;
        error = GetPixelValue(img2, kp.pt.x+dx+x, kp.pt.y+dy+y) -
                GetPixelValue(img1, kp.pt.x+x, kp.pt.y+y);
        H += J*J.transpose();
        b += -J*error;
        cost += error*error/2;
        // TODO END YOUR CODE HERE
    }
```

```
bool inverse = true;
OpticalFlowSingleLevel(img1, img2, kp1, kp2_single, success_single,
inverse);
```

- 运行结果



## 问1-4：推广至金字塔

- optical\_flow.cpp的OpticalFlowMultiLevel函数

```
void OpticalFlowMultiLevel(
    const Mat &img1,
    const Mat &img2,
    const vector<KeyPoint> &kp1,
    vector<KeyPoint> &kp2,
    vector<bool> &success,
    bool inverse) {

    // parameters
    int pyramids = 4;
    double pyramid_scale = 0.5;
    double scales[] = {1.0, 0.5, 0.25, 0.125};

    // create pyramids
    vector<Mat> pyr1, pyr2; // image pyramids
    // TODO START YOUR CODE HERE (~8 lines)
    for (int i = 0; i < pyramids; i++) {
        Mat img1_temp, img2_temp;

        resize(img1, img1_temp, Size(img1.cols*scales[i], img1.rows*scales[i]));

        resize(img2, img2_temp, Size(img2.cols*scales[i], img2.rows*scales[i]));
        pyr1.push_back(img1_temp);
        pyr2.push_back(img2_temp);
    }
    // TODO END YOUR CODE HERE

    // coarse-to-fine LK tracking in pyramids
    // TODO START YOUR CODE HERE
    vector<KeyPoint> kp1_pyr, kp2_pyr;
    for(auto &kp:kp1){
        auto kp_temp = kp;
        kp_temp.pt *= scales[pyramids-1];
        kp1_pyr.push_back(kp_temp);
    }
}
```

```

        kp2_pyr.push_back(kp_temp);
    }

    for(int i = pyramids-1; i>=0;i--){
        success.clear();

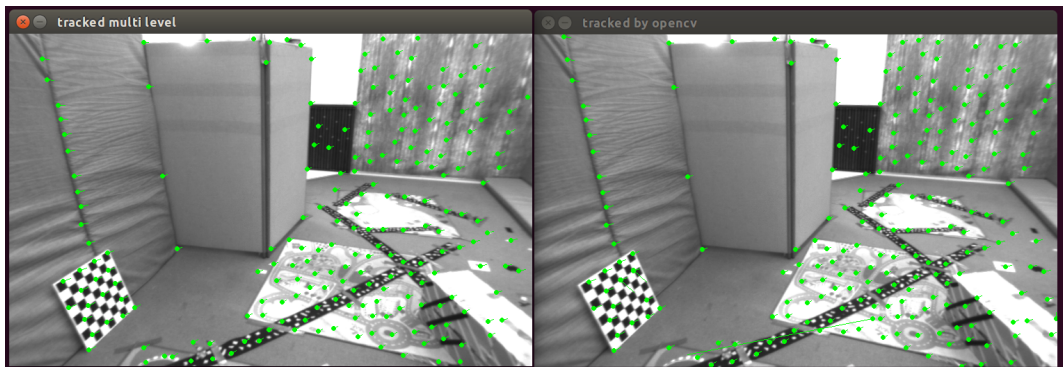
        OpticalFlowSingleLevel(pyr1[i],pyr2[i], kp1_pyr, kp2_pyr, success, true);
        if(i>0){
            for(auto &kp:kp1_pyr){kp.pt =
Point2f((Vec2f)kp.pt/pyramid_scale);}
            for(auto &kp:kp2_pyr){kp.pt =
Point2f((Vec2f)kp.pt/pyramid_scale);}
        }
    }

    for(auto &kp:kp2_pyr){kp2.push_back(kp);}
    // TODO END YOUR CODE HERE
    // don't forget to set the results into kp2
}

```

- 运行结果

- 反向多层



- 正向多层



- 问答

- 所谓 coarse-to-fine 是指怎样的过程?

以原始图像作为图像金字塔的底层，每往上一层，对下层图像进行缩放，计算光流时，先从最顶层开始（低分辨率，对运动不敏感），然后将结果作为下层的初始值进行计算，直到最底层（原始图像），从而得到一个更为准确的结果。

- 光流法中的金字塔用途和特征点法中的金字塔有何差别?

- 光流法中的图像金字塔通过多层迭代的方式来解决光流在快速运动中难以检测的问题；
    - 特征点法中的图像金字塔用于解决特征点的尺度不变性。

## 问1-5：讨论

- 我们优化两个图像块的灰度之差真的合理吗?哪些时候不够合理?你有解决办法吗?  
光流法基于灰度不变假设这一前提，当运动过快时，无法满足。可以通过建立光流金字塔来解决。
- 图像块大小是否有明显差异?取 16x16 和 8x8 的图像块会让结果发生变化吗?
  - 建立图像金字塔时，窗口固定，在每一层金字塔上都用同样大小的窗口进行计算所以图像块大小不会有明显差异；
  - 当不使用图像金字塔时，设置窗口时，窗口大光流鲁棒性好，窗口小光流精确性好
- 金字塔层数对结果有怎样的影响?缩放倍率呢?
  - 层数一般越多越好，但层数过多可能导致特征点像素太紧密引起误追踪；
  - 缩放倍率小，层数增加。

---

## 2. 直接法

### 2.1 单层直接法

- direct\_method.cpp的DirectPoseEstimationSingleLayer函数

```
void DirectPoseEstimationSingleLayer(  
    const cv::Mat &img1,  
    const cv::Mat &img2,  
    const VecVector2d &px_ref,  
    const vector<double> &depth_ref,  
    Sophus::SE3 &T21  
) {  
  
    // parameters  
    int half_patch_size = 4;  
    int iterations = 100;  
  
    double cost = 0, lastCost = 0;  
    int nGood = 0; // good projections  
    VecVector2d goodProjection;  
  
    for (int iter = 0; iter < iterations; iter++) {  
        nGood = 0;  
        goodProjection.clear();  
  
        // Define Hessian and bias  
        Matrix6d H = Matrix6d::Zero(); // 6x6 Hessian  
        Vector6d b = Vector6d::Zero(); // 6x1 bias  
  
        for (size_t i = 0; i < px_ref.size(); i++) {  
  
            // compute the projection in the second image  
            // TODO START YOUR CODE HERE  
            double X_ref = depth_ref[i]*(px_ref[i][0]-cx)/fx;  
            double Y_ref = depth_ref[i]*(px_ref[i][1]-cy)/fy;  
            double Z_ref = depth_ref[i];
```



```

        Matrix4d T_21 = T21.matrix();
        double
X_cur=T_21(0,0)*X_ref+T_21(0,1)*Y_ref+T_21(0,2)*Z_ref+T_21(0,3);
        double
Y_cur=T_21(1,0)*X_ref+T_21(1,1)*Y_ref+T_21(1,2)*Z_ref+T_21(1,3);
        double
Z_cur=T_21(2,0)*X_ref+T_21(2,1)*Y_ref+T_21(2,2)*Z_ref+T_21(2,3);

        float u =0, v = 0;
        u = (float)((X_cur*fx)/Z_cur+cx);
        v = (float)((Y_cur*fy)/Z_cur+cy);

        if(u-half_patch_size<0 || u+half_patch_size>=img2.cols || v-
half_patch_size <0 || v+half_patch_size>=img2.rows)
        {
            continue;
        }
        nGood++;
        goodProjection.push_back(Eigen::Vector2d(u, v));

        // and compute error and jacobian
        for (int x = -half_patch_size; x < half_patch_size; x++)
            for (int y = -half_patch_size; y < half_patch_size; y++) {

                double error =0;
                error = GetPixelValue(img1,px_ref[i][0]+x,px_ref[i]
[1]+y)-GetPixelValue(img2,u+x,v+y);

                Matrix26d J_pixel_xi;    // pixel to \xi in Lie algebra
                Eigen::Vector2d J_img_pixel;    // image gradients

                J_img_pixel[0]=(GetPixelValue(img2,u+x+1,v+y)-
GetPixelValue(img2,u+x-1,v+y))/2;
                J_img_pixel[1]=(GetPixelValue(img2,u+x,v+y+1)-
GetPixelValue(img2,u+x,v+y-1))/2;

                J_pixel_xi(0,0)=fx/Z_cur;
                J_pixel_xi(0,1)=0;
                J_pixel_xi(0,2)=-(fx*X_cur)/(Z_cur*Z_cur);
                J_pixel_xi(0,3)=-(fx*X_cur*Y_cur)/(Z_cur*Z_cur);
                J_pixel_xi(0,4)=fx+(fx*X_cur*X_cur)/(Z_cur*Z_cur);
                J_pixel_xi(0,5)=-(fx*Y_cur)/(Z_cur);
                J_pixel_xi(1,0)=0;
                J_pixel_xi(1,1)=fy/Z_cur;
                J_pixel_xi(1,2)=-(fy*Y_cur)/(Z_cur*Z_cur);
                J_pixel_xi(1,3)=-fy-(fy*Y_cur*Y_cur)/(Z_cur*Z_cur);
                J_pixel_xi(1,4)=(fy*X_cur*Y_cur)/(Z_cur*Z_cur);
                J_pixel_xi(1,5)=(fy*X_cur)/(Z_cur);

                // total jacobian
                Vector6d J=-1*(J_pixel_xi.transpose()*J_img_pixel);

                H += J * J.transpose();
                b += -error * J;
                cost += error * error;
            }
        // END YOUR CODE HERE

```



```

    }

    // solve update and put it into estimation
    // TODO START YOUR CODE HERE
    Vector6d update;
    update=H.ldlt().solve(b);
    T21 = Sophus::SE3::exp(update) * T21;
    // END YOUR CODE HERE

    cost /= nGood;

    if (isnan(update[0])) {
        // sometimes occurred when we have a black or white patch and H
is irreversible
        cout << "update is nan" << endl;
        break;
    }
    if (iter > 0 && cost > lastCost) {
        cout << "cost increased: " << cost << ", " << lastCost << endl;
        break;
    }
    lastCost = cost;
    //cout << "cost = " << cost << ", good = " << nGood << endl;
}
cout << "good projection: " << nGood << endl;
cout << "T21 = \n" << T21.matrix() << endl;

// in order to help you debug, we plot the projected pixels here
/*    cv::Mat img1_show, img2_show;
    cv::cvtColor(img1, img1_show, CV_GRAY2BGR);
    cv::cvtColor(img2, img2_show, CV_GRAY2BGR);
    for (auto &px: px_ref) {
        cv::rectangle(img1_show, cv::Point2f(px[0] - 2, px[1] - 2),
cv::Point2f(px[0] + 2, px[1] + 2),
                cv::Scalar(0, 250, 0));
    }
    for (auto &px: goodProjection) {
        cv::rectangle(img2_show, cv::Point2f(px[0] - 2, px[1] - 2),
cv::Point2f(px[0] + 2, px[1] + 2),
                cv::Scalar(0, 250, 0));
    }
    cv::imshow("reference", img1_show);
    cv::imshow("current", img2_show);
    cv::waitKey();*/
}

```

- CMakeLists

```

cmake_minimum_required(VERSION 2.8.3)
SET(CMAKE_BUILD_TYPE "Release")
PROJECT (Chapter6)

add_compile_options(-std=c++11)

```

```

INCLUDE_DIRECTORIES(${PROJECT_SOURCE_DIR}/include)
INCLUDE_DIRECTORIES("/usr/include/opencv2")
INCLUDE_DIRECTORIES("/usr/include/eigen3")

find_package( OpenCV REQUIRED )

find_package(Sophus REQUIRED)
include_directories(${Sophus_INCLUDE_DIRS})

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

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

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

```

- 运行结果

```

iusl@iusl-OptiPlex-7060:~/lyq/github/SLAM/Homeworks/Chapter6/build$ make
Scanning dependencies of target direct_method
[ 25%] Building CXX object CMakeFiles/direct_method.dir/src/direct_method.cpp.o
[ 50%] Linking CXX executable direct_method
[ 50%] Built target direct_method
[100%] Built target optical_flow
iusl@iusl-OptiPlex-7060:~/lyq/github/SLAM/Homeworks/Chapter6/build$ ./direct_method
cost increased: 13036.6, 13036.6
good projection: 989
T21 =
  0.999991  0.00242132  0.00337215 -0.00184407
-0.00242871  0.999995  0.00218895  0.0026733
-0.00336684 -0.00219713  0.999992 -0.725126
      0      0      0      1
cost increased: 29782.6, 29782.5
good projection: 928
T21 =
  0.999972  0.00137274  0.00728934  0.0074022
-0.00140109  0.999991  0.00388429 -0.00131324
-0.00728395 -0.0038944  0.999966 -1.4707
      0      0      0      1
cost increased: 86185.9, 86184.1
good projection: 878
T21 =
  0.999911  0.000481451  0.0133037 -0.226637
-0.000554052  0.999985  0.00545403 -0.00142486
-0.0133009 -0.00546092  0.999897 -1.87775
      0      0      0      1
cost increased: 157684, 157610
good projection: 865
T21 =
  0.999858  0.00265471  0.0166511 -0.289632
-0.00274098  0.999983  0.00516068  0.0222435
-0.0166371 -0.00520559  0.999848 -2.02282
      0      0      0      1
cost increased: 167627, 167445
good projection: 759
T21 =
  0.999734  0.00162393  0.0230258 -0.409766
-0.0017288  0.999988  0.00453526  0.063028
-0.0230182 -0.00457385  0.999725 -2.96648
      0      0      0      1
iusl@iusl-OptiPlex-7060:~/lyq/github/SLAM/Homeworks/Chapter6/build$

```

- 问答

- 误差项是什么?

$$I_{ref}(\pi(\mathbf{p}_i)) - I_{cur}(\pi(T_{cur,ref}\mathbf{p}_i))$$

- 误差相对于自变量的雅可比维度是多少?如何求解?

2 x 6

$$\frac{\partial u}{\partial \delta \xi} = \begin{bmatrix} \frac{f_x}{Z} & 0 & -\frac{f_x X}{Z^2} & -\frac{f_x XY}{Z^2} & f_x + \frac{f_x X^2}{Z^2} & -\frac{f_x Y}{Z} \\ 0 & \frac{f_y}{Z} & -\frac{f_y Y}{Z^2} & -f_y - \frac{f_y Y^2}{Z^2} & \frac{f_y XY}{Z^2} & \frac{f_y X}{Z} \end{bmatrix}$$

$$J = -\frac{\partial I}{\partial u} \frac{\partial u}{\partial \delta \xi}$$

- 窗口可以取多大?是否可以取单个点?
  - 窗口过大会导致计算量过大;
  - 可以取单个点,但是会导致在计算像素梯度时误差增大。

## 2.2 多层直接法

- direct\_method.cpp的DirectPoseEstimationMultiLayer函数

```
void DirectPoseEstimationMultiLayer(
    const cv::Mat &img1,
    const cv::Mat &img2,
    const VecVector2d &px_ref,
    const vector<double> depth_ref,
    Sophus::SE3 &T21
) {

    // parameters
    int pyramids = 4;
    double pyramid_scale = 0.5;
    double scales[] = {1.0, 0.5, 0.25, 0.125};

    // create pyramids
    vector<cv::Mat> pyr1, pyr2; // image pyramids
    // TODO START YOUR CODE HERE
    for(int i = 0; i < pyramids; i++){
        cv::Mat img1_temp, img2_temp;

        cv::resize(img1, img1_temp, cv::Size(img1.cols*scales[i], img1.rows*scales[i])
        );

        cv::resize(img2, img2_temp, cv::Size(img2.cols*scales[i], img2.rows*scales[i])
        );

        pyr1.push_back(img1_temp);
        pyr2.push_back(img2_temp);
    }
    // END YOUR CODE HERE

    double fxG = fx, fyG = fy, cxG = cx, cyG = cy; // backup the old values
    for (int level = pyramids - 1; level >= 0; level--) {
        VecVector2d px_ref_pyr; // set the keypoints in this pyramid level
        for (auto &px: px_ref) {
```

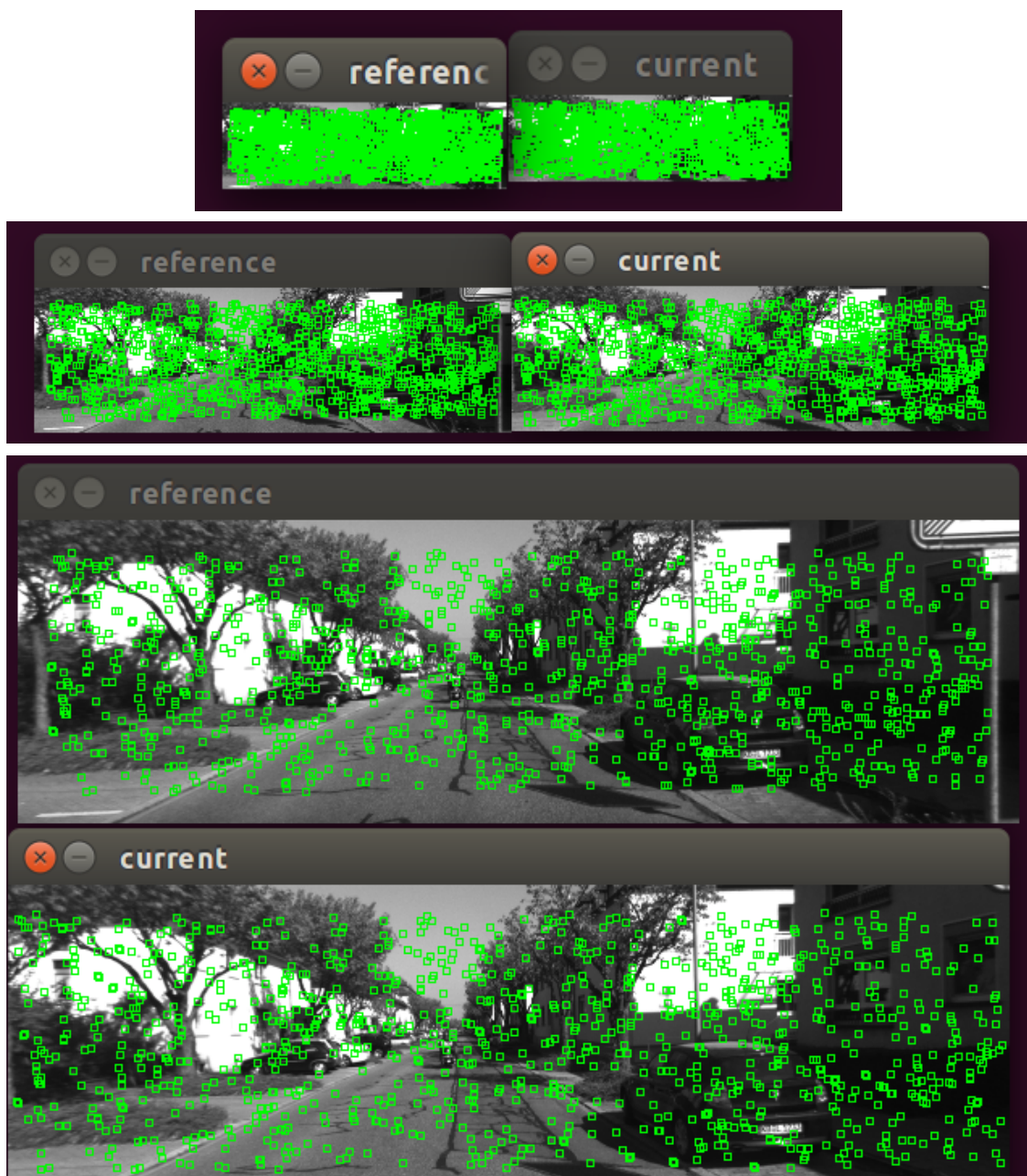
```

        px_ref_pyr.push_back(scales[level] * px);
    }

    // TODO START YOUR CODE HERE
    // scale fx, fy, cx, cy in different pyramid levels
    fx = fxG*scales[level];
    fy = fyG*scales[level];
    cx = cxG*scales[level];
    cy = cyG*scales[level];
    // END YOUR CODE HERE
    DirectPoseEstimationSingleLayer(pyr1[level], pyr2[level],
    px_ref_pyr, depth_ref, T21);
}
}

```

- 运行结果





```
tust@tust-OptiPlex-7060:~/Lyq/github/SLAM/Homeworks/Chapter6/build$ ./dire
T_cur_ref =
  0.999991  0.00242145  0.00337204 -0.00183495
 -0.00242884  0.999995  0.00218915  0.00266685
 -0.00336672 -0.00219732  0.999992  -0.725144
      0      0      0      1
T_cur_ref =
  0.999972  0.00137284  0.00728917  0.00741017
 -0.00140118  0.999991  0.00388447 -0.00131846
 -0.00728378 -0.00389458  0.999966  -1.4707
      0      0      0      1
T_cur_ref =
  0.999937  0.00161501  0.01111191  0.00706419
 -0.00167162  0.999986  0.00508391  0.00378663
 -0.0111107 -0.00510218  0.999925  -2.20885
      0      0      0      1
T_cur_ref =
  0.999874  0.00035404  0.0158972  0.00786997
 -0.000445342  0.999983  0.00574013  0.00282849
 -0.0158949 -0.00574648  0.999857  -2.99537
      0      0      0      1
T_cur_ref =
  0.999803  0.00120288  0.0198227  0.0189922
 -0.00133249  0.999978  0.00652679 -0.010237
 -0.0198144 -0.00655191  0.999782  -3.79313
      0      0      0      1
```

## 2.3 延伸讨论

- 直接法是否可以类似光流,提出 inverse, compositional 的概念?它们有意义吗?  
理论上可以提出这些概念,但是意义不大。
- 请思考上面算法哪些地方可以缓存或加速?  
窗口设计部分
- 在上述过程中,我们实际假设了哪两个 patch 不变?
  - 灰度不变假设;



- 同窗口内深度不变假设。
- 为何可以随机取点?而不用取角点或线上的点?那些不是角点的地方,投影算对了吗?  
直接法是对灰度优化,是不是角点无所谓。
- 请总结直接法相对于特征点法的异同与优缺点。
  - 优点:
    - 省去计算特征点和描述子的时间;
    - 可以构建半稠密甚至稠密的地图。
  - 缺点:
    - 灰度不变假设实际中很容易不满足;
    - 容易陷入局部最优。

---

### 3. 使用光流计算视差

- 代码

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

using namespace std;
using namespace cv;

// this program shows how to use optical flow

string file_left = "./left.png"; // left image
string file_right = "./right.png"; // right image
string file_disparity = "./disparity.png"; //disparity image

void OpticalFlowSingleLevel(
    const Mat &img1,
    const Mat &img2,
    const vector<KeyPoint> &kp1,
    vector<KeyPoint> &kp2,
    vector<bool> &success,
    bool inverse = false
);

void OpticalFlowMultiLevel(
    const Mat &img1,
    const Mat &img2,
    const vector<KeyPoint> &kp1,
    vector<KeyPoint> &kp2,
    vector<bool> &success,
    bool inverse = false
);
```

```

inline float GetPixelValue(const cv::Mat &img, float x, float y) {
    uchar *data = &img.data[int(y) * img.step + int(x)];
    float xx = x - floor(x);
    float yy = y - floor(y);
    return float(
        (1 - xx) * (1 - yy) * data[0] +
        xx * (1 - yy) * data[1] +
        (1 - xx) * yy * data[img.step] +
        xx * yy * data[img.step + 1]
    );
}

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

    // images, note they are CV_8UC1, not CV_8UC3
    Mat img1 = imread(file_left, 0);
    Mat img2 = imread(file_right, 0);
    Mat img_disparity = imread(file_disparity, 0);

    // key points, using GFTT here.
    vector<KeyPoint> kp1;
    vector<Point2f> kp1_temp;
    //Ptr<GFTTDetector> detector = GFTTDetector::create(500, 0.01, 20); //
maximum 500 keypoints
    //detector->detect(img1, kp1);
    goodFeaturesToTrack(img1, kp1_temp, 500, 0.01, 20);
    KeyPoint::convert(kp1_temp, kp1, 1, 1, 0, -1);

    // now lets track these key points in the second image
    // first use single level LK in the validation picture
    vector<KeyPoint> kp2_single;
    vector<bool> success_single;
    bool inverse = true;
    OpticalFlowSingleLevel(img1, img2, kp1, kp2_single, success_single,
inverse);

    // then test multi-level LK
    vector<KeyPoint> kp2_multi;
    vector<bool> success_multi;
    OpticalFlowMultiLevel(img1, img2, kp1, kp2_multi, success_multi);

    // use opencv's flow for validation
    vector<Point2f> pt1, pt2;
    for (auto &kp: kp1) pt1.push_back(kp.pt);
    vector<uchar> status;
    vector<float> error;
    cv::calcOpticalFlowPyrLK(img1, img2, pt1, pt2, status, error,
cv::Size(8, 8));

    int count_single = 0;
    int abs_error_sum_single = 0;
    for(int i=0; i<kp2_single.size();i++){
        if(success_single[i]){
            count_single ++;
            int disparity1 = img_disparity.at<uchar>
(kp1[i].pt.y, kp1[i].pt.x);

```



```

        int disparity2 = kp1[i].pt.x-kp2_single[i].pt.x;
        int error = disparity1-disparity2;
        abs_error_sum_single += abs(error);
        cout<<count_single<<": "<<"    示例视差："<<disparity1<<"    单层光
流视差："<<disparity2<<"    误差："<<error<<endl;
    }
}

int count_multi =0;
int abs_error_sum_multi =0;
for(int i=0; i<kp2_multi.size();i++){
    if(success_multi[i]){
        count_multi ++;
        int disparity1 = img_disparity.at<uchar>
(kp1[i].pt.y,kp1[i].pt.x);
        int disparity2 = kp1[i].pt.x-kp2_multi[i].pt.x;
        int error = disparity1-disparity2;
        abs_error_sum_multi += abs(error);
        cout<<count_multi<<": "<<"    示例视差："<<disparity1<<"    多层光
流视差："<<disparity2<<"    误差："<<error<<endl;
    }
}

int count_opencv =0;
int abs_error_sum_opencv =0;
for(int i=0; i<pt2.size();i++){
    if(status[i]){
        count_opencv ++;
        int disparity1 = img_disparity.at<uchar>
(kp1[i].pt.y,kp1[i].pt.x);
        int disparity2 = pt1[i].x-pt2[i].x;
        int error = disparity1-disparity2;
        abs_error_sum_opencv += abs(error);
        cout<<count_opencv<<": "<<"    示例视差："<<disparity1<<"
opencv光流视差："<<disparity2<<"    误差："<<error<<endl;
    }
}

int abs_error_mean_single =abs_error_sum_single/count_single;
int abs_error_mean_multi =abs_error_sum_multi/count_multi;
int abs_error_mean_opencv =abs_error_sum_opencv/count_opencv;
cout<<"单层光流视差误差平均值："<<abs_error_mean_single<<endl;
cout<<"多层光流视差误差平均值："<<abs_error_mean_multi<<endl;
cout<<"opencv光流视差误差平均值："<<abs_error_mean_opencv<<endl;

return 0;

}

void OpticalFlowSingleLevel(
    const Mat &img1,
    const Mat &img2,
    const vector<KeyPoint> &kp1,
    vector<KeyPoint> &kp2,
    vector<bool> &success,
    bool inverse
) {

```

```

// parameters
int half_patch_size = 4;
int iterations = 10;
bool have_initial = !kp2.empty();

for (size_t i = 0; i < kp1.size(); i++) {
    auto kp = kp1[i];
    double dx = 0, dy = 0; // dx,dy need to be estimated
    if (have_initial) {
        dx = kp2[i].pt.x - kp.pt.x;
        dy = kp2[i].pt.y - kp.pt.y;
    }

    double cost = 0, lastCost = 0;
    bool succ = true; // indicate if this point succeeded

    // Gauss-Newton iterations
    for (int iter = 0; iter < iterations; iter++) {
        Eigen::Matrix2d H = Eigen::Matrix2d::Zero();
        Eigen::Vector2d b = Eigen::Vector2d::Zero();
        cost = 0;

        if (kp.pt.x + dx <= half_patch_size || kp.pt.x + dx >= img1.cols
- half_patch_size ||
            kp.pt.y + dy <= half_patch_size || kp.pt.y + dy >= img1.rows
- half_patch_size) { // go outside
            succ = false;
            break;
        }

        // compute cost and jacobian
        for (int x = -half_patch_size; x < half_patch_size; x++)
            for (int y = -half_patch_size; y < half_patch_size; y++) {

                // TODO START YOUR CODE HERE (~8 lines)
                double error = 0;
                Eigen::Vector2d J; // Jacobian
                if (inverse == false) {
                    // Forward Jacobian
                    J[0]=
(GetPixelValue(img2, kp.pt.x+dx+x+1, kp.pt.y+dy+y)-
GetPixelValue(img2, kp.pt.x+dx+x-1, kp.pt.y+dy+y))/2;
                    J[1]=
(GetPixelValue(img2, kp.pt.x+dx+x, kp.pt.y+dy+y+1)-
GetPixelValue(img2, kp.pt.x+dx+x, kp.pt.y+dy+y-1))/2;
                } else {
                    // Inverse Jacobian
                    // NOTE this J does not change when dx, dy is
updated, so we can store it and only compute error
                    J[0]=(GetPixelValue(img1, kp.pt.x+x+1, kp.pt.y+y)-
GetPixelValue(img1, kp.pt.x+x-1, kp.pt.y+y))/2;
                    J[1]=(GetPixelValue(img1, kp.pt.x+x, kp.pt.y+y+1)-
GetPixelValue(img1, kp.pt.x+x, kp.pt.y+y-1))/2;
                }

                // compute H, b and set cost;
                error = GetPixelValue(img2, kp.pt.x+dx+x, kp.pt.y+dy+y)-
GetPixelValue(img1, kp.pt.x+x, kp.pt.y+y);
            }
        }
    }
}

```

```

        H += J*J.transpose();
        b += -J*error;
        cost += error*error/2;
        // TODO END YOUR CODE HERE
    }

    // compute update
    // TODO START YOUR CODE HERE (~1 lines)
    Eigen::Vector2d update;
    update = H.ldlt().solve(b);
    // TODO END YOUR CODE HERE

    if (isnan(update[0])) {
        // sometimes occurred when we have a black or white patch
        and H is irreversible
        //cout << "update is nan" << endl;
        succ = false;
        break;
    }
    if (iter > 0 && cost > lastCost) {
        //cout << "cost increased: " << cost << ", " << lastCost <<
endl;

        break;
    }

    // update dx, dy
    dx += update[0];
    dy += update[1];
    lastCost = cost;
    succ = true;
}

success.push_back(succ);

// set kp2
if (have_initial) {
    kp2[i].pt = kp.pt + Point2f(dx, dy);
} else {
    KeyPoint tracked = kp;
    tracked.pt += cv::Point2f(dx, dy);
    kp2.push_back(tracked);
}
}

}

void OpticalFlowMultiLevel(
    const Mat &img1,
    const Mat &img2,
    const vector<KeyPoint> &kp1,
    vector<KeyPoint> &kp2,
    vector<bool> &success,
    bool inverse) {

    // parameters
    int pyramids = 4;
    double pyramid_scale = 0.5;
    double scales[] = {1.0, 0.5, 0.25, 0.125};

```

```

// create pyramids
vector<Mat> pyr1, pyr2; // image pyramids
// TODO START YOUR CODE HERE (~8 lines)
for (int i = 0; i < pyramids; i++) {
    Mat img1_temp, img2_temp;

    resize(img1, img1_temp, Size(img1.cols*scales[i], img1.rows*scales[i]));

    resize(img2, img2_temp, Size(img2.cols*scales[i], img2.rows*scales[i]));
    pyr1.push_back(img1_temp);
    pyr2.push_back(img2_temp);
}
// TODO END YOUR CODE HERE

// coarse-to-fine LK tracking in pyramids
// TODO START YOUR CODE HERE
vector<KeyPoint> kp1_pyr, kp2_pyr;
for(auto &kp:kp1){
    auto kp_temp = kp;
    kp_temp.pt *= scales[pyramids-1];
    kp1_pyr.push_back(kp_temp);
    kp2_pyr.push_back(kp_temp);
}

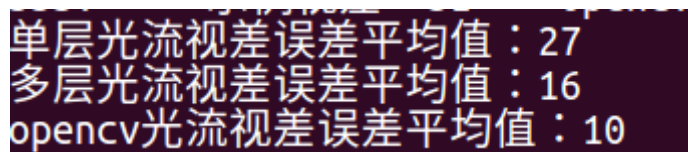
for(int i = pyramids-1; i>=0; i--){
    success.clear();

    OpticalFlowSingleLevel(pyr1[i], pyr2[i], kp1_pyr, kp2_pyr, success, false);
    if(i>0){
        for(auto &kp:kp1_pyr){kp.pt =
Point2f((Vec2f)kp.pt/pyramid_scale);}
        for(auto &kp:kp2_pyr){kp.pt =
Point2f((Vec2f)kp.pt/pyramid_scale);}
    }
}

for(auto &kp:kp2_pyr){kp2.push_back(kp);}
// TODO END YOUR CODE HERE
// don't forget to set the results into kp2
}

```

- 运行结果



```

单层光流视差误差平均值：27
多层光流视差误差平均值：16
opencv光流视差误差平均值：10

```

opencv效果好于多层光流，多层光流效果好于单层光流。

## 参考资料

[https://blog.csdn.net/orange\\_littlegirl/article/details/103852133#t3](https://blog.csdn.net/orange_littlegirl/article/details/103852133#t3)

<https://www.xlmaverick.me/post/视觉里程计2-lk/>

1. <https://blog.csdn.net/wendox/article/details/52505971>↵