目录

1 实验目标

利用课上学习的Lucas Kanade方法计算稀疏光流
(1)加载一段视频。
(2)检测其中的特征点，使用Lucas-Kanade光流迭代跟踪这些点，并绘制线段。提交内容:
a.源码。
使用Opencv库函数完成(2)(75分)
自行实现LS光流计算（90分)
使用c++，提交cpp文件。
b.报告。
包含实验结果分析及运行结果截图。原视频截图+光流示意图。Pdf文件，10分

2 程序

2.1 程序简介

1. 主函数main()：初始化视频读取器capture，读取视频帧，调用tracking()函数进行光流跟踪，按ESC键退出循环。
2. tracking()函数：转换帧为灰度图，将当前帧复制到输出结果中，调用addNewPoints()函数检测是否需要添加新的特征点。若需要添加新的特征点，则使用goodFeaturesToTrack()函数在当前帧中检测新的特征点，将检测到的特征点添加到points[0]和initial向量中。如果之前的帧不为空，则使用calcOpticalFlowPyrLK()函数计算前一帧中特征点的光流，并将计算结果保存到points[1]中。使用acceptTrackedPoint()函数过滤掉不好的特征点，将剩余的特征点保存到points[1]和initial向量中。绘制特征点和运动轨迹到输出结果中，交换points[0]和points[1]，交换gray_prev和gray向量，最后在窗口中显示输出结果。
3. addNewPoints()函数：返回特征点数量是否小于等于10的结果，用于判断是否需要添加新的特征点。
4. acceptTrackedPoint()函数：返回特征点状态是否为1以及它们之间距离是否大于2的结果，用于过滤掉不好的特征点。

2.2 算法

Lucas-Kanade方法是计算稀疏光流的一种经典算法。该算法基于假设，在一个局部窗口内，相邻的像素具有相似的运动。因此，该算法利用最小二乘法来计算每个像素的运动。

设$I(x,y,t)$表示图像在时刻t处的灰度值，u和v表示x和y方向的运动量，则$I(x+u,y+v,t+1)$表示在$t+1$时刻的像素值。假设在一个大小为$W\times W$的窗口内，相邻的像素具有相同的运动，因此可以用以下方程组表示:

$$\begin{bmatrix} I_x\left(p_1\right) & I_y\left(p_1\right) \\ I_x\left(p_2\right) & I_y\left(p_2\right) \\ \vdots & \vdots \\ I_x\left(p_n\right) & I_y\left(p_n\right) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t\left(p_1\right) \\ I_t\left(p_2\right) \\ \vdots \\ I_t\left(p_n\right) \end{bmatrix}$$

其中，p_1到p_n表示窗口内的像素点，I_x、I_y和I_t分别表示图像在x方向、y方向和时间上的梯度。使用最小二乘法可以求出解向量$[u,v]^T$:

$$[u,v]^T=(A^TA)^{-1}A^Tb$$

其中，A是梯度矩阵，b是残差向量。计算过程中还要对矩阵A^TA进行奇异值分解（SVD）来防止矩阵不可逆的情况。

2.3 实现细节

2.3.1 加载一段视频

```
int main()
{
    Mat frame;
    Mat result;

    //VideoCapture capture(0);摄像头
    VideoCapture capture("hw3-1.mp4");

    if (capture.isOpened())
    {
        while (true)
        {
            capture >> frame;
            if (!frame.empty())             //不为空
            {
                tracking(frame, result);     //跟踪
            }
            else
            {
                printf("No Capture frame , Break");
                break;
            }
            int c = waitKey(50);
            if (27 == (char)c)
            {
                break;
            }
        }
    }

    return 0;
}
```

该代码是主函数部分，主要实现了读取视频流（或者摄像头捕获的图像），并调用tracking函数进行运动目标跟踪。

首先创建了两个Mat类型的变量：frame和result，用于存储读取到的视频帧和处理后的结果帧。然后通过VideoCapture类读取视频文件"hw3-1.mp4"，并判断文件是否成功打开。接下来进入一个while循环，不断读取视频帧，直到读取结束或按下ESC键退出。

在每次循环中，通过capture >> frame将视频流读取到frame中，然后判断frame是否为空，若为空则说明读取到了视频流的末尾，需要退出程序。否则，调用tracking函数进行运动目标跟踪，并将结果帧存储到result中。在处理完一帧后，通过waitKey函数等待50ms，以便观察结果。如果按下ESC键，也退出程序。

2.3.2 特征点的检测与LK光流跟踪

```
void tracking(Mat& frame, Mat& output)
{
    cvtColor(frame, gray, COLOR_BGR2GRAY);
    frame.copyTo(output);

    //添加特征点
    if (addNewPoints())
    {
        goodFeaturesToTrack(gray, features, maxCount, qLevel, minDest);
        points[0].insert(points[0].end(), features.begin(), features.end());
        initial.insert(initial.end(), features.begin(), features.end());
    }
    if (gray_prev.empty())
    {
        gray.copyTo(gray_prev);
    }
    //l-k流光法运动估计
    calcOpticalFlowPyrLK(gray_prev, gray, points[0], points[1], status, err);

    //去掉一些不好的特征点
    int k = 0;
    for (size_t i = 0; i < points[1].size(); i++)
    {
        if (acceptTrackedPoint(i))
        {
            initial[k] = initial[i];
            points[1][k++] = points[1][i];
        }
    }
    points[1].resize(k);
    initial.resize(k);
    //显示特征点和运动轨迹
    for (size_t i = 0; i < points[1].size(); i++)
    {
        line(output, initial[i], points[1][i], Scalar(0, 0, 255));
        circle(output, points[1][i], 3, Scalar(0, 255, 0), -1);
    }

    //把当前跟踪结果作为下一次的参考
    swap(points[1], points[0]);
    swap(gray_prev, gray);
    imshow(window_name, output);
}
```

1. tracking()函数中首先将当前视频帧转为灰度图，并将当前帧保存到输出图像output中。
2. 然后调用addNewPoints()函数判断是否需要添加新的跟踪点，如果需要则调用goodFeaturesToTrack()函数进行特征点检测，并将新检测到的特征点加入到跟踪点points[0]和初始点initial中。
3. 接着使用calcOpticalFlowPyrLK()函数计算跟踪点在当前帧中的位置，并将跟踪点更新到points[1]中。

4. 根据跟踪点的运动情况，通过acceptTrackedPoint()函数判断哪些跟踪点需要保留，并更新跟踪点和初始点。
5. 最后，将跟踪点和运动轨迹绘制到输出图像output中，并将当前跟踪结果作为下一次的参考。

```
bool addNewPoints()
{
    return points[0].size() <= 10;       //points.size()求行数     points.size()求列数
}
```

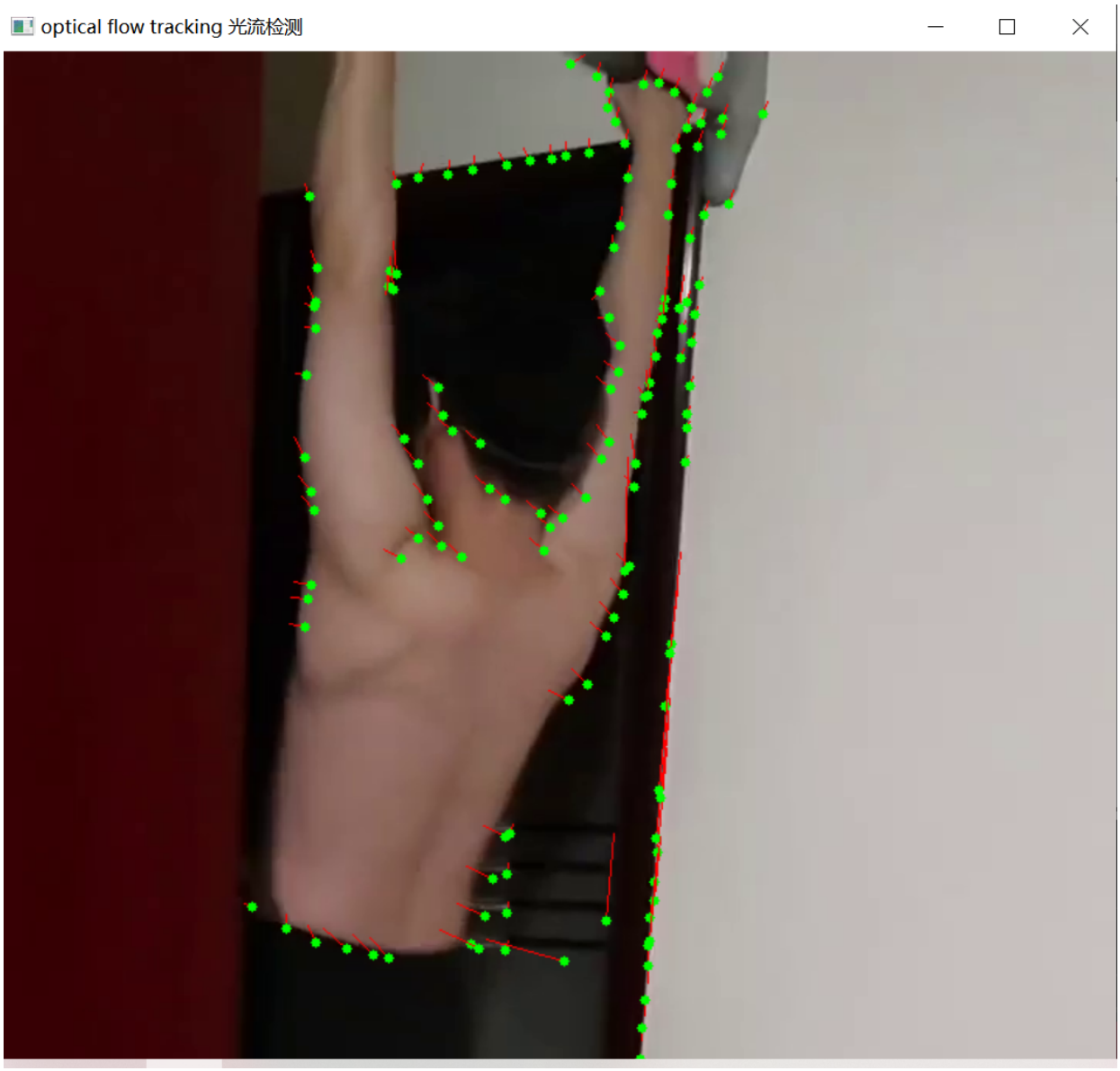
addNewPoints()函数用于检测是否需要添加新的跟踪点，其判断条件是跟踪点的数量小于等于10个；

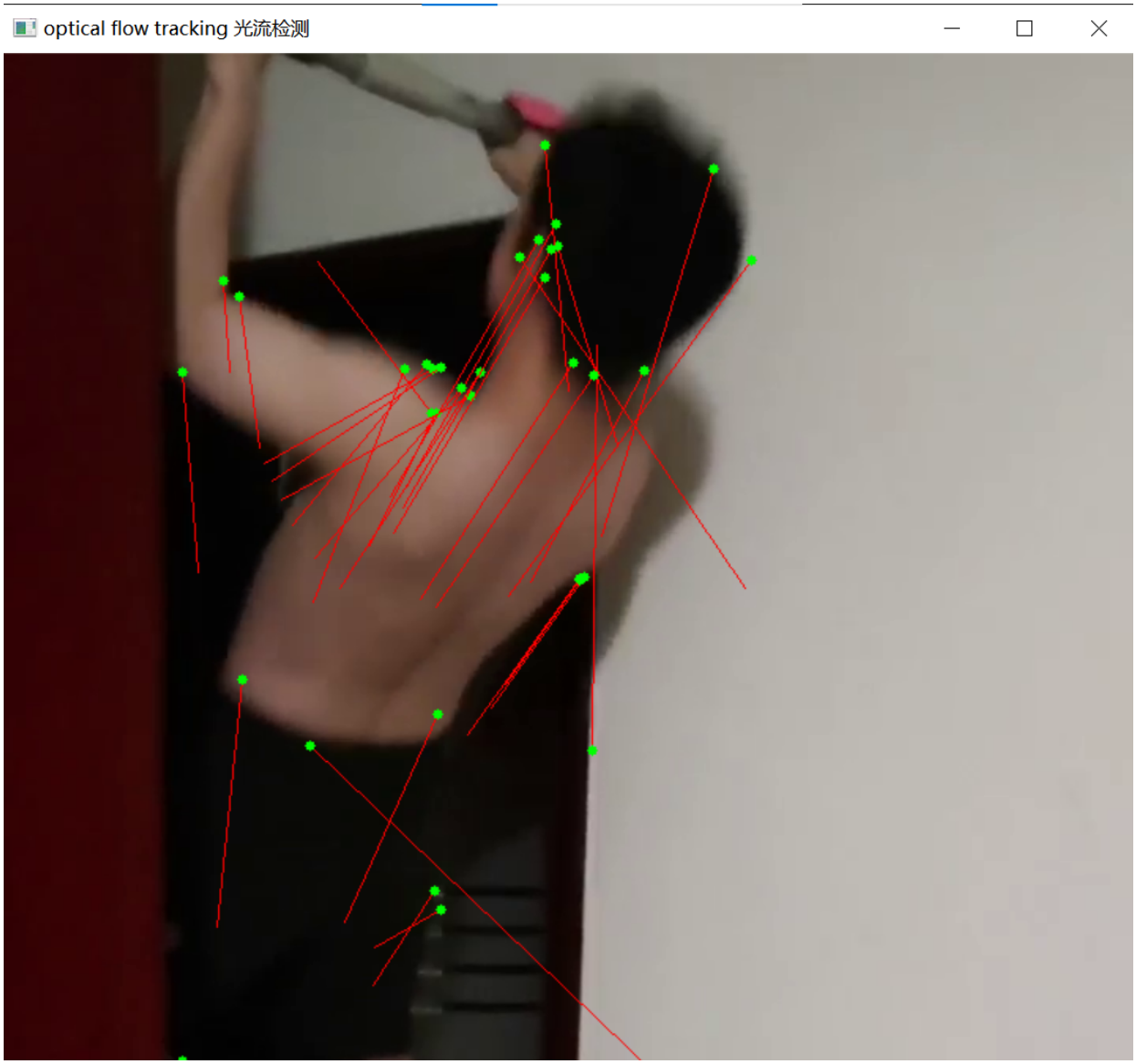
```
bool acceptTrackedPoint(int i)
{
    return status[i] && ((abs(points[0][i].x - points[1][i].x) + abs(points[0][i].y
- points[1][i].y)) > 2);
}
```

acceptTrackedPoint()函数用于决定哪些跟踪点需要保留，其判断条件是跟踪点在当前帧中有运动，并且运动距离大于2个像素。

3 结果与讨论

3.1 实验结果





3.2 实验分析

光流表达了图像的变化，由于它包含了目标运动的信息，因此可被观察者用来确定目标的运动情况。对于实验结果中的第一张图片，此时人物还未开始运动，该图中的显示出了较多的特征点。在第二张图片中，此时图片中的特征点较第一张图片有所减少，但特征点后面的红线表明了运动趋势。对比观察两张图片中，我们不难发现视频中人物在做引体向上，人物整体在向上移动。

3.3 讨论

通过这个实验我对LK光流跟踪这一算法更加了解，也让我对图像处理方法和OpenCV函数有了更深入的了解。我相信这些知识和经验对我的未来学习和工作会产生一定的帮助。