华东师范大学计算机科学技术系上机实践报告

课程名称：计算机视觉	**年级：**2015	**上机实践成绩：**
指导教师：文颖	**姓名：**朱勇赤	**上机实践日期：**2018/6/19
实践编号：实验 5：稠密运动视觉估计	**学号：**10152130131	**上机实践时间：**10:00~11:40

一、 实验名称

稠密运动视觉估计

二、 实验目的

输入：给定视频中的前后两帧图像
输出：给出图像中的运动光流图像

三、 实验内容

1 采用差分图像运动估计；

2 采用Brox光流算法实现；

3 采用Horn-Schunck实现；

四、 实验原理

1 差分图像运动估计：

摄像机采集的视频序列具有连续性的特点。如果场景内没有运动目标，则连续帧的变化很微弱，如果存在运动目标，则连续的帧和帧之间会有明显地变化。

帧间差分法就是借鉴了上述思想。由于场景中的目标在运动，目标的影像在不同图像帧中的位置不同。该类算法对时间上连续的两帧或三帧图像进行差分运算，不同帧对应的像素点相减，判断灰度差的绝对值，当绝对值超过一定阈值时，即可判断为运动目标，从而实现目标的检测功能

帧间差分：在运动目标检测中，简单来说，就是背景与当前帧之间的差异。数字图像可以表示成一个矩阵，矩阵中每一个元素叫一个像素点。帧间差分=绝对值（背景-当前帧图像）。

我们取帧间差分足够大的像素点，将这些像素点作为前景像素。

针对背景建模问题，采用动态建模。每一帧通过加权和更新背景，公式如下：

背景=（1-a）背景+a*第t帧图像（去除前景）

参数a时学习率，如a=0.001。当第t帧时，通过将当前的背景和第t帧图像（除掉前景像素点）加权和，获得新的背景，用来检测第t+1帧前景

2 Horn-Schunck算法

Horn-Schunck光流法求得的是稠密光流，需要对每一个像素计算光流值，计算量比较大。而Lucas-Kanade光流法只需计算若干点的光流，是一种稀疏光流。

用uij与vij分别表示图像像素点(i，j)处的水平方向光流值与垂直方向光流值，每次迭代后的更新方程为

$$u^{(k+1)}=\bar{u}^{(k)}-I_x\frac{I_x\bar{u}^{(k)}+I_y\bar{v}^{(k)}+I_t}{\lambda^2+I_x^2+I_y^2}$$

$$v^{(k+1)}=\bar{v}^{(k)}-I_y\frac{I_x\bar{u}^{(k)}+I_y\bar{v}^{(k)}+I_t}{\lambda^2+I_x^2+I_y^2}$$

n为迭代次数，lamda反映了对图像数据及平滑约束的可信度，当图像数据本身含有较大噪声时，此时需要加大lamda的值，相反，当输入图像含有较少的噪声时，此时可减小lamda的值。

代表u邻域与v邻域的平均值，一般采用相应4邻域内的均值

$$\begin{cases}\overline{u_{i,j}}=\frac{1}{4}(u_{i-1,j}+u_{i+1,j}+u_{i,j-1}+u_{i,j+1})\\ \overline{v_{i,j}}=\frac{1}{4}(v_{i-1,j}+v_{i+1,j}+v_{i,j-1}+v_{i,j+1})\end{cases}$$

也可以采用3*3、5*5的窗口用模板平滑，窗口不宜过大，过大会破坏光流假设。

Ix、Iy分别是图像对x、y的偏导数。It是两帧图像间对时间的导数。

$$I_x=I(x,y,t)-I(x-1,y,t)$$

$$I_y=I(x,y,t)-I(x,y-1,t)$$

$$I_t=I(x,y,t)-I(x,y,t-1)$$

考虑相邻像素及相邻两帧图像的影响，Horn-Schunck 提出通过 4 次有限差分来得到

$$I_x(i,j,t)=\frac{1}{4}[I(i,j+1,t)-I(i,j,t)+I(i+1,j+1,t)-I(i+1,j,t)$$
$$+I(i,j+1,t+1)-I(i,j,t+1)+I(i+1,j+1,t+1)-I(i+1,j,t+1)]$$

$$I_y(i,j,t)=\frac{1}{4}[I(i+1,j,t)-I(i,j,t)+I(i+1,j+1,t)-I(i,j+1,t)$$
$$+I(i+1,j,t+1)-I(i,j,t+1)+I(i+1,j+1,t+1)-I(i,j+1,t+1)]$$

$$I_t(i,j,t)=\frac{1}{4}[I(i,j,t+1)-I(i,j,t)+I(i+1,j+1,t+1)-I(i+1,j+1,t)$$
$$+I(i,j+1,t+1)-I(i,j+1,t)+I(i+1,j,t+1)-I(i+1,j,t)]$$

这里只考虑了前后两帧图像。考虑连续三帧图像的话有如下方法：

一种性能更优的 3D-Sobel 算子 如下图所示。该算子在x 、y 、t方向上分别使用不同的模板对连续3帧图像进行卷积计算 得出中间帧的位于模板中心的像素在三个方向上的梯度 。

	前帧	中帧	后帧
x 方向梯度	$\begin{matrix}-1&0&1\\-2&0&2\\-1&0&1\end{matrix}$	$\begin{matrix}-2&0&2\\-4&0&4\\-2&0&2\end{matrix}$	$\begin{matrix}-1&0&1\\-2&0&2\\-1&0&1\end{matrix}$
y 方向梯度	$\begin{matrix}1&2&1\\0&0&0\\-1&-2&-1\end{matrix}$	$\begin{matrix}2&4&2\\0&0&0\\-2&-4&-2\end{matrix}$	$\begin{matrix}1&2&1\\0&0&0\\-1&-2&-1\end{matrix}$
t 方向梯度	$\begin{matrix}-1&-2&-1\\-2&-4&-2\\-1&-2&-1\end{matrix}$	$\begin{matrix}0&0&0\\0&0&0\\0&0&0\end{matrix}$	$\begin{matrix}1&2&1\\2&4&2\\1&2&1\end{matrix}$

图 3-2　3D-Sobel 算子示意图

迭代一定次数后u、v收敛，光流计算停止。在实际的计算中迭代初值可取U(0) =0、V(0)=0。

算法改进

对于一般场景基本等式只有在图像中灰度梯度值较大的点处才成立。因此为了增强算法的稳定性和准确性 我们仅在梯度较大的点处才使用亮度恒常性约束，而在梯度较小的点处只使用流场一致性约束。定义如下权函数

$$\varepsilon(x,y)=\begin{cases}0 & I_x^2+I_y^2>threshold\\ 1 & \text{其他}\end{cases}$$

Horn-Schunck 的光流场计算公式则为：

$$\iint\left\{\varepsilon(x,y)\cdot(I_xu+I_yv+I_t)^2+\alpha^2\left[\left(\frac{\partial u}{\partial x}\right)^2+\left(\frac{\partial u}{\partial y}\right)^2+\left(\frac{\partial v}{\partial x}\right)^2+\left(\frac{\partial v}{\partial y}\right)^2\right]\right\}dx\,dy=\min$$

对应的欧拉方程为：

$$\varepsilon I_x(I_xu^{(n+1)}+I_yv^{(n+1)}+I_t)=-\alpha^2\nabla^2u$$

$$\varepsilon I_y(I_xu^{(n+1)}+I_yv^{(n+1)}+I_t)=-\alpha^2\nabla^2v$$

解该欧拉方程可得如下瞬时速度的估计公式：

$$u^{(n+1)}=\overline{u}^{(n)}-I_x\cdot\frac{(I_x\overline{u}^{(n)}+I_y\overline{v}^{(n)}+I_t)\varepsilon}{\alpha^2+(I_x^2+I_y^2)\varepsilon}$$

$$v^{(n+1)}=\overline{v}^{(n)}-I_y\cdot\frac{(I_x\overline{u}^{(n)}+I_y\overline{v}^{(n)}+I_t)\varepsilon}{\alpha^2+(I_x^2+I_y^2)\varepsilon}$$

五、 实验算法

OPENCV下使用帧间差分法提取光流：

1.将 background 和 frame 转为灰度图

```
Mat gray1, gray2;
cvtColor(temp, gray1, CV_BGR2GRAY);
cvtColor(frame, gray2, CV_BGR2GRAY);
```

2.将 background 和 frame 做差

```
Mat diff;
absdiff(gray1, gray2, diff);
imshow("diff", diff);
```

3.对差值图 diff_thresh 进行阈值化处理

```
Mat diff_thresh;
threshold(diff, diff_thresh, 10, 255, CV_THRESH_BINARY);
imshow("diff_thresh", diff_thresh);

GaussianBlur(diff_thresh, diff_thresh, Size(3, 3), 0, 0);
```

4.腐蚀（以下步骤可选，对本次试验效果提升不够）

```
Mat kernel_erode = getStructuringElement(MORPH_RECT, Size(3, 3));
Mat kernel_dilate = getStructuringElement(MORPH_RECT, Size(5, 5));
erode(diff_thresh, diff_thresh, kernel_erode);
//imshow("erode", diff_thresh);
```

5.膨胀

```
dilate(diff_thresh, diff_thresh, kernel_dilate);
imshow("dilate", diff_thresh);
```

6.查找轮廓并绘制轮廓

```
vector<vector<Point>> contours;
findContours(diff_thresh, contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_NONE);
//drawContours(result, contours, -1, Scalar(0, 0, 255), 2);//在 result 上绘制轮廓
```

7.查找正外接矩形

```
vector<Rect> boundRect(contours.size());
for (int i = 0; i < contours.size(); i++)
{
    boundRect[i] = boundingRect(contours[i]);
    rectangle(result, boundRect[i], Scalar(0, 255, 0), 2);//在 result 上绘制正外接矩形
    }
```

OPENCV下使用Horn-Schunck算法提取稠密光流：

1、使用opencv内置的库读取两幅图片

2、初始化结果U,V分别用来存储横向与纵向的光流

3、确定lambda 值为 0.05

4、分别将图像对于x,y,t求导

实现如下：

```
//自定义 x 方向求导函数
Mat get_fx(Mat &src1, Mat &src2) {
Mat fx;
Mat kernel = Mat::ones(2, 2, CV_64FC1);
kernel.ATD(0, 0) = -1.0;
kernel.ATD(1, 0) = -1.0;

Mat dst1, dst2;
filter2D(src1, dst1, -1, kernel);
filter2D(src2, dst2, -1, kernel);

fx = dst1 + dst2;
return fx;
}
```

```
//自定义 y 方向求导函数
Mat get_fy(Mat &src1, Mat &src2) {
    Mat fy;
    Mat kernel = Mat::ones(2, 2, CV_64FC1);
    kernel.ATD(0, 0) = -1.0;
    kernel.ATD(0, 1) = -1.0;

    Mat dst1, dst2;
    filter2D(src1, dst1, -1, kernel);
    filter2D(src2, dst2, -1, kernel);

    fy = dst1 + dst2;
    return fy;
}

//自定义对 t 求导函数
Mat get_ft(Mat &src1, Mat &src2) {
    Mat ft;
    Mat kernel = Mat::ones(2, 2, CV_64FC1);
    kernel = kernel.mul(-1);

    Mat dst1, dst2;
    filter2D(src1, dst1, -1, kernel);
    kernel = kernel.mul(-1);
    filter2D(src2, dst2, -1, kernel);

    ft = dst1 + dst2;
    return ft;
}
```

5、获取四邻域均值：

```
//四领域均值
double get_Average4(Mat &m, int y, int x) {
    if (x < 0 || x >= m.cols) return 0;
    if (y < 0 || y >= m.rows) return 0;

    double val = 0.0;
    int tmp = 0;
    if (isInsideImage(y - 1, x, m)) {
        ++tmp;
        val += m.ATD(y - 1, x);
    }
    if (isInsideImage(y + 1, x, m)) {
        ++tmp;
        val += m.ATD(y + 1, x);
    }
    if (isInsideImage(y, x - 1, m)) {
        ++tmp;
        val += m.ATD(y, x - 1);
    }
    if (isInsideImage(y, x + 1, m)) {
        ++tmp;
        val += m.ATD(y, x + 1);
    }
    return val / tmp;
```

```
}

Mat get_Average4_Mat(Mat &m) {
    Mat res = Mat::zeros(m.rows, m.cols, CV_64FC1);
    for (int i = 0; i < m.rows; i++) {
        for (int j = 0; j < m.cols; j++) {
            res.ATD(i, j) = get_Average4(m, i, j);
        }
    }
    return res;
}
```

5、循环，即依照公式进行超松弛迭代。当迭代至上一次的均值的绝对值小于等于本次均值的绝对值时停止

```
while (1) {
        Mat Uav = get_Average4_Mat(u);
        Mat Vav = get_Average4_Mat(v);
        Mat P = fx.mul(Uav) + fy.mul(Vav) + ft;
        Mat D = fx.mul(fx) + fy.mul(fy) + lambda;
        Mat tmp;
        divide(P, D, tmp);
        Mat utmp, vtmp;
        utmp = Uav - fx.mul(tmp);
        vtmp = Vav - fy.mul(tmp);
        Mat eq = fx.mul(utmp) + fy.mul(vtmp) + ft;
        double thistime = mean(eq)[0];
        cout << "i = " << i << ", mean = " << thistime << endl;
        if (i != 0 && fabs(last) <= fabs(thistime)) break;
        i++;
        last = thistime;
        //垂直水平方向光流
        u = utmp;
        v = vtmp;
            }
```

6、保存横向与纵向光流图像矩阵，显示结果：

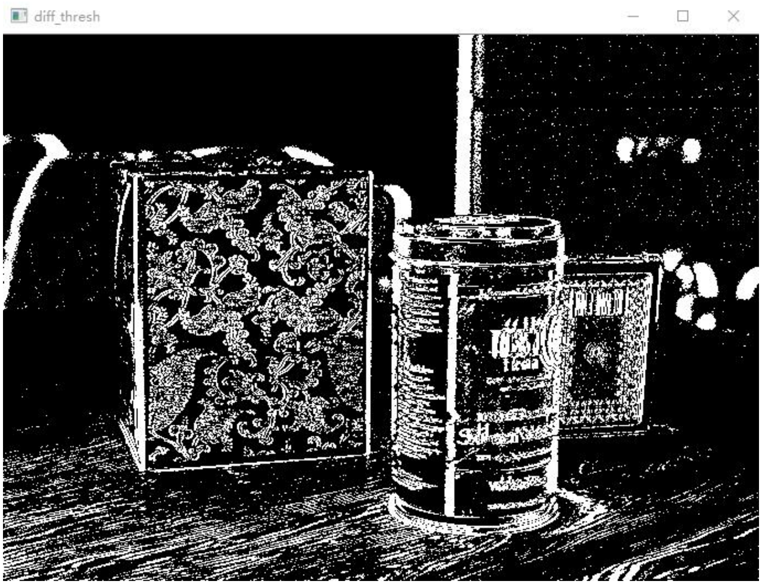
六、 实验结果及分析

实验1：

运动中两帧图像如下所示：

帧间差分法获得的光流图像

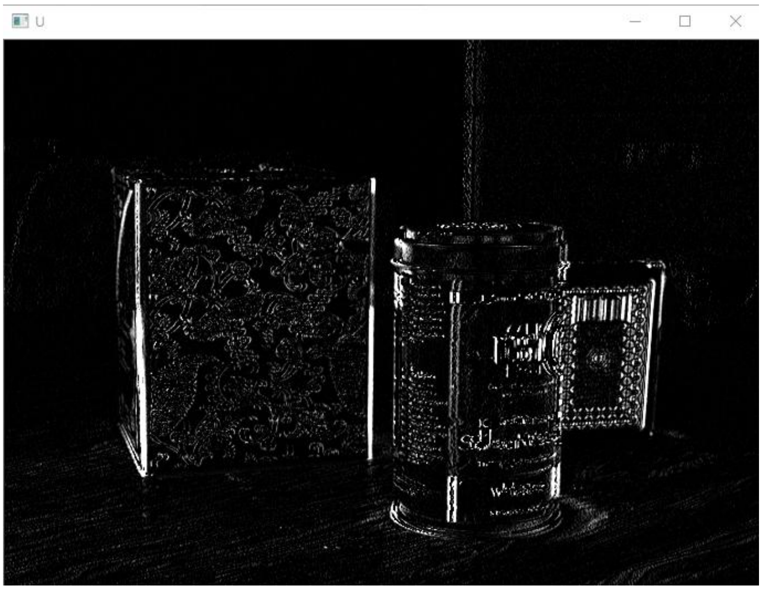
进行阈值化后得到的结果



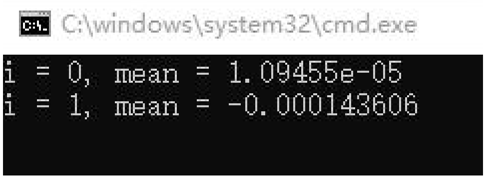
Horn-Schunck算法得到光流结果如下

横向的光流变化：

纵向的光流变化：



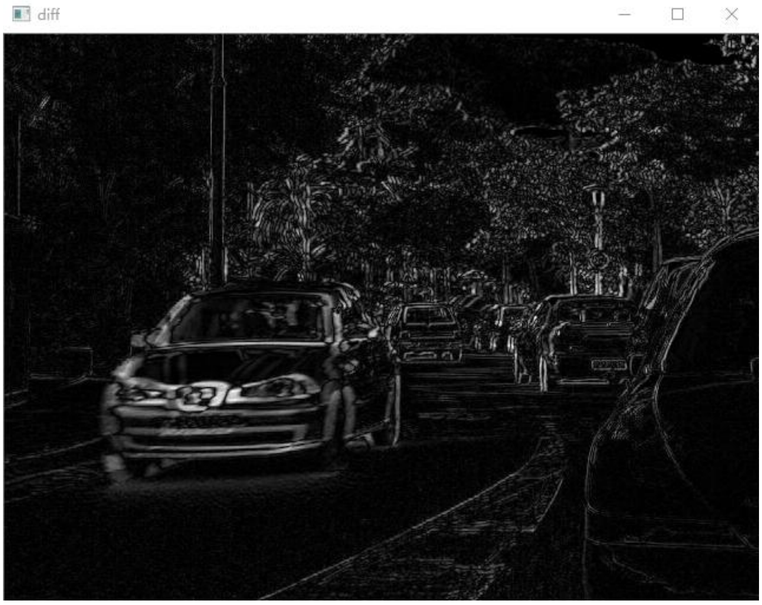
迭代次数以及每次所获得均值



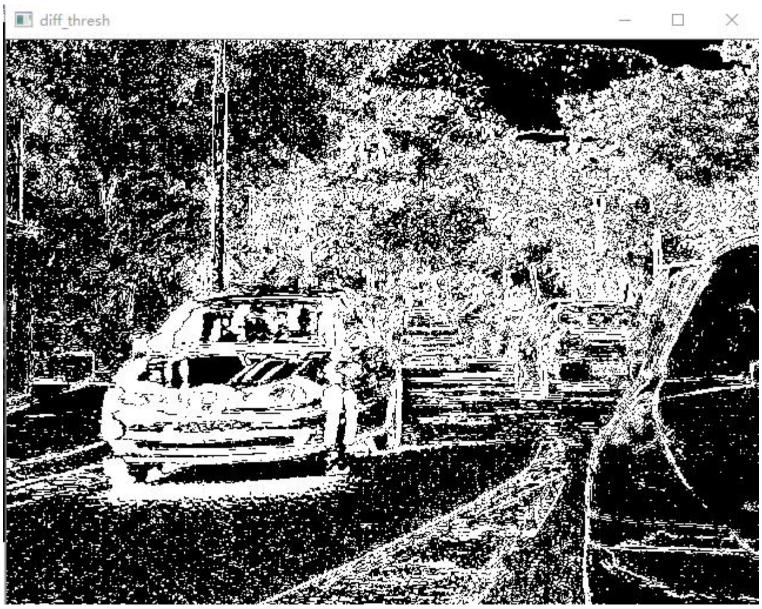
实验2：

运动中两帧图像如下所示：

帧间差分法获得的光流图像

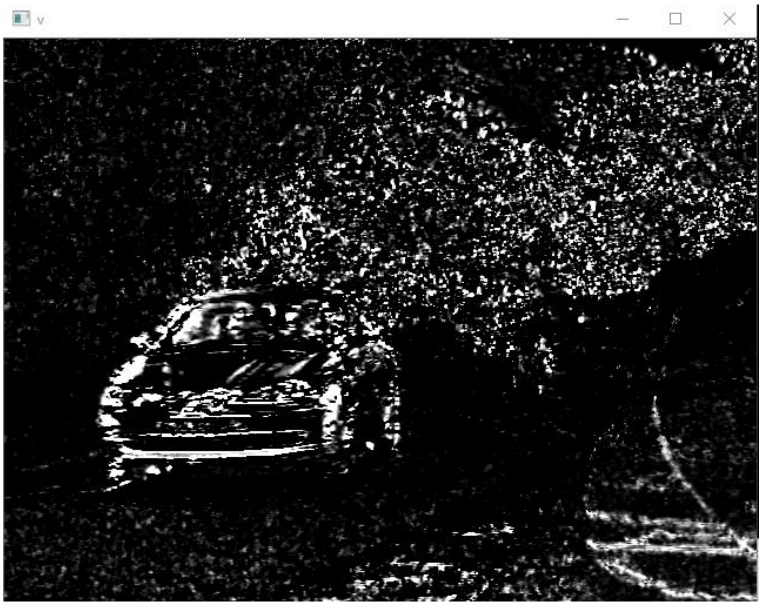


阈值化后得到的结果

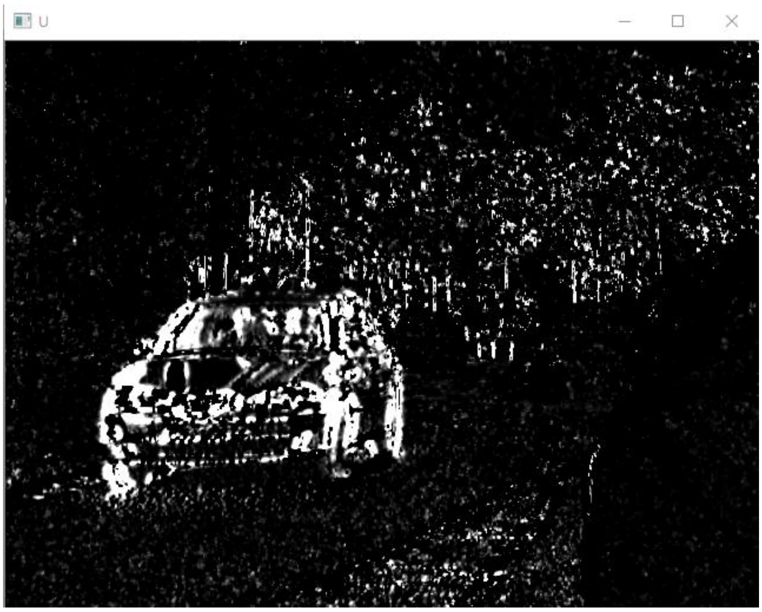


Horn-Schunck算法得到光流结果如下

横向的光流变化：



纵向的光流变化：



迭代次数以及每次所获得均值：

C:\windows\system32\cmd.exe

```
i = 0, mean = -0.00460179
i = 1, mean = -0.00328107
i = 2, mean = -0.00273525
i = 3, mean = -0.00250112
i = 4, mean = -0.00239359
i = 5, mean = -0.00235428
i = 6, mean = -0.00234777
i = 7, mean = -0.00236217
```

七、 问题讨论

Horn-Schunck 方法得到图像的光流需要多次迭代，但可以准确看出物体在横纵两个方向上的运动情况，相当于将物体的运动分解了，更容易应用于对物体运动状态的判断与速度计算；

帧间差分法较为简单，直接通过两帧做差，得出运动光流，处理极快，但仅可以判断物体在运动以及得知物体的简单的运动状态。