《 数 字 媒 体 》 实 验 报 告

实 验 名 称 运动图像检测（背景差分法实现部分）

姓　　名 蔡宇

学　　号 Z12030727

专　　业 计算机技术

学　　院 计算机科学与技术

工 作 单 位 浙江宇视科技有限公司

指 导 老 师 袁 昕

2014 年 4 月 17 日

一、 实验目的和要求

通过背景差分的试验方法实现运动图像检测。

二、 实验内容和原理

编程环境选用 Opencv2.1 和 Visual C++ 2010。Opencv2.1 是一种开源的用于图像处理和计算机视觉的函数库，由 Intel 公司使用 C++高级语言开发。Visual C++ 2010 是微软公司开发的编程工具，支持最新的 C++0x 标准。

减背景方法是常用的运动目标检测方法。其基本思想是将视频流中当前一帧所有像素点与事先通过某种方法计算得到的背景图像中对应像素点相减并取绝对值，如果绝对值超过某个预先设定好的阀值，则认为当前帧中对应的像素点是运动目标的像素点；否则，就认为对应的像素点是背景像素点。相减运算的结果还提供了视频流中运动目标的位置、大小及形状等信息。但是该方法在应用过程中常会遇到如下的问题：

（1）背景获取：最简单的背景获取方法就是在视频场景没有运动目标的情况下直接将某一帧存储为背景图像，但在大多数的视频应用中，如交通监控和行人检测，这一要求很难得到满足，故需要一种能够在运动目标存在的视频流中实时获取背景图像的方法。

（2）背景扰动：背景中经常存在一些对象轻微的扰动，如树枝的摇动，扰动部分不应该看作是前景运动目标。

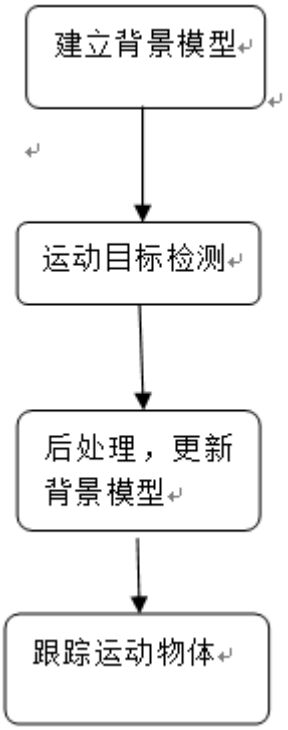
（3）光照变化：天气、光线等因素随着时间的变化也会影响运动目标的检测结果，这是必须要考虑的问题。

（4）背景更新：为了适应外界各种条件的不断变化，有必要对建立起来的背景模型进行实时更新。

以往的运动目标检测方法或者不能解决以上所有问题，或者是通过构造复杂的模型来解决以上问题，其计算复杂性和对系统的要求都比较高，有时可能无法满足实时处理的要求。 本文在减背景方法的基础上，提出了一种更为有效的运动 目标检测方法。在背景的提取阶段，允许视频流中有运动目标的存在，在这种情况下，首先采用基于统计的方法建立背景模型，然后进行减背景操作来检测视频中的运动目标，并对背景模型进行实时更新，以适应光线的变化和场景本身的变化，最后对检测结果使用形态学运算和连通区域面积限制目标大小的方法进行后处理，消除噪声和背景扰动带来的影响。当运动目标确定后，采用区域跟踪技术对目标进行实时跟踪，跟踪技术中使用了两个参数以避免运动目标之间的遮挡问题。实验结果证明，提出的方法取得了比较理想的结果。

三、 程序分析和说明

1. 程序的流程图如下：

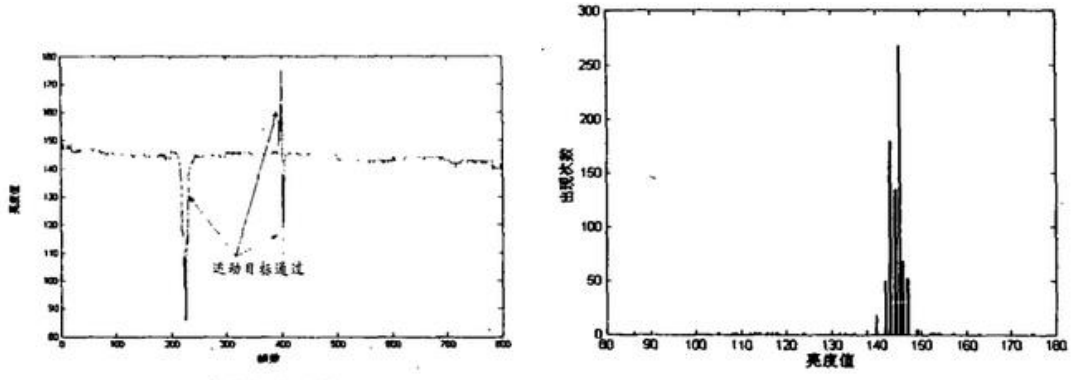


2. 原理分析

1） 背景模型建立

背景模型建立的准确与否，直接关系到运动目标检测结果的准确性。国内外已经提出了许多背景模型的建立方案，如 W4 方法，该方法是在视频中没有目标出现的情况下，对场景进行一段时间的测量，记录每个像素点的最大和最小亮度值，以及相邻两帧间亮度最大差异值，然后使用这 3 个值表示背景模型。还有使用前几帧图像像素亮度值的平均值作为背景模型的。以上这些方法都要求视频中至少一段时间内没有运动目标的出现，但是这种要求在多数场合下并不能得到满足。

本文提出一种简单、有效的背景模型建立方法，它可以在视频场景中存在运动目标的情况下提取出背景图像。此方法是基于这样的假设，在背景模型建立阶段，运动目标虽然在场景区域中运动 ，但是它并不会长时间地停留在某一位置上。



(a)亮度变化曲线图　　(b)亮度统计直方图

图 1

对视频流中某一像素点进行一段时间的观测，可以发现，它的亮度值只是在前景运动目标通过该点时，才会发生较大的变化，如图 1(a)所示。对该像素点的亮度值进行统计，可以看出，在一段时间内，它的亮度值主要集中在一个很小的区域中，如图 1(b)所示。因此，可以用这个区域内的平均值作为该点的背景值。

具体实现过程如下：在灰度模式下，像素亮度值的变化范围为 0 ~ 255，将该范围划分成若干区间[0，T]，[T，2T]，[nT，255]，n = 255/T。对于每个像素点，统计一段视频内每个区间亮度值出现的次数，找出出现次数最多的那个区间，接着计算该区间内所有亮度值的平均值，用该平均值作为背景图像在该点的亮度值。实验表明，该方法提取背景过程的中，不受前景运动目标的影响。

2）运动目标检测

运动目标检测是计算视频流中当前帧的像素点和背景图像中对应像素点的差异并取其绝对值，如果绝对值大于给定的阈值，则判定该像素点为前景运动目标的像素点。一对像素点在做相减运算时，可以使用它们的亮度值、色度值或其它参数，本文采用亮度值相减，检测规则如下：

$$M(x,y) = \begin{cases} 0 & if\ I(x,y) - B(x,y) < TH \\ 1 & if\ I(x,y) - B(x,y) \geq TH \end{cases} \quad (1)$$

其中$M(x,y)$是运动目标像素点的集合，$I(x,y)$表示当前帧中的像素点亮度值，$B(x,y)$表示背景图像中对应像素点的亮度值，TH为阀值，目前阀值的选择主要依靠先验知识，比如根据目标与背景的亮度差别等。

3）后处理及背景模型的更新

由于视频本身不可避免的噪声影响，会使上述检测结果出现一些问题，比如本是背景图像的区域像素点经过检测后被当成运动区域的像素点，运动目标内的部分区域因为某种原因被漏检，以及背景图像的分部区域由于树枝、树叶的轻微晃动而被误判为运动目标的区域等等。为了消除这些影响，首先对上一步的检测结果用形态学的腐蚀、膨胀方法进行处理，再找出其经过形态学处理后的连通区域，接着计算每个连通区域的面积，对于面积小于一定阀值的区域，将其抛弃，不看作是前景运动目标区域。形态学运算使用 5X5 的矩阵算子，连通区域内像素点个数的阈值选择要视具体情况而定。

背景模型的实时更新是要使背景能够对外界环境的变化具有一定的自适应性。对于光线的变化，我们的方法与文献[6]中采用的方法类似。对于检测到的存在运动目标的区域不进行更新，而只对没有检测到运动目标的区域进行实时更新，对于这部分区域：

$$B_{n+1}(x,y) = \alpha B_n(x,y) + (1-\alpha) I_n(x,y) \quad (2)$$

在具体的检测过程中，有时前景与背景对应像素点的亮度值相差很小，导致前景目标的部分区域被漏检掉，如果对背景中对应部分也进行更新，就会生成错误的背景模型，对后面的检测过程造成一定的影响，因此，选择最近 N 帧没有前景目标通过的像素点进行更新 。具体更新过程如下 ：

$$B_{n+1}(x,y) = \alpha B_n(x,y) + (1-\alpha)\frac{I_{n-N}(x,y) + I_{n-N+1}(x,y) + \cdots + I_n(x,y)}{N} \quad (3)$$

其中$B_n(x,y)$，$B_{n+1}(x,y)$分别代表当前和下一帧背景，$I_{n-N}(x,y)$，$I_n(x,y)$分别代表该点最近的 N 个亮度值，$\alpha \in (0,1)$为更新系数，控制背景更新的速度。

图 2 是一个像素点亮度值的观测曲线，可以看出，该方法对于光线变化有很强的自适应性。

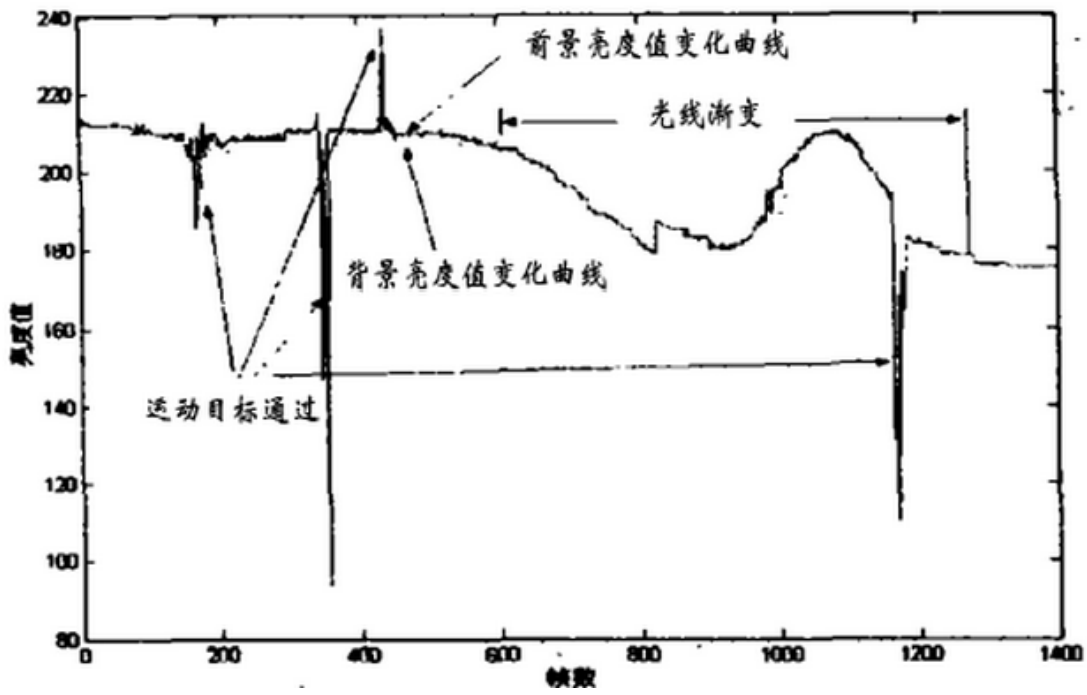


图 2

4）运动物体的跟踪

当目标检测出来以后，就要对运动的目标进行跟踪，本文采用区域跟踪的方法实现对运动目标的跟踪， 并在区域跟踪算法中选用了二个参数实现匹配。该方法能够在物体重叠和遮挡的情况下准确地定位出运动物体在图像中的位置。很好地实现了物体的跟踪， 也满足了实时性处理的要求。

图像中标号为 I 的目标的大小用目标面积 SizeI 来描述,每一目标都对应一个外截矩形框，用 BoxI 来进行描述。Box 可用一条对角线的两端点 S ， T 的坐标来表示:

$$Box\ S,T\ ,\quad S=\ s1,s2\ ,T=\ t1,t2\ ,\quad si\le ti\ \ i=1,2 \qquad (4)$$

定义 1:相邻两帧中标号为 i 的目标与标号为 j 的目标面积大小差:

$$Dif\ SizeI\ \ K\ ,sizeJ\ K+1\ \ =\ SizeI\ K\ -SizeJ\ K+1\ \ ,\ i,j\ \ =1,2,\dots,N \qquad (5)$$

其中 N 为图像中目标的标号。

定义 2:标号为 L 的目标的质心定义如下

$$CtrL=\ X,Y\ ,\ X=\sum_{i=1}^{N} x_i,\ Y=\sum_{i=1}^{N} y_i \qquad (6)$$

这里 N 是图像中标号为 L 的目标含有象素点个数， xi 是象素点 i 的横坐标， yi 是象素点 i 的纵坐标。

相邻两帧中标号为 I 的目标与标号为 J 的目标质心之间的距离

$$Dis\ CtrI\ \ K\ ,CtrJ\ K+1\ \ =\ \sqrt{X_I-X_J{}^2+\ Y_I-Y_j{}^2} \qquad (7)$$

具体跟踪步骤如下：

(1)将第一帧中的目标检测出来，并计算出被跟踪目标(假设为目标 M)在本帧中面积的大小、质心的位置等数据。

(2)检测出下一帧中的目标 ，计算出各自的面积以及质心的位置。

(3)分别计算出目标 M 与下一帧中各目标的质心间的距离 Dis 和面积差

$$If\ \alpha * Dis\ CtrM\ \ K\ ,Ctri\ K+1\ \ +\beta * Dif\ SizeM\ \ K\ ,Sizei\ K+1\ \ \leq \delta \quad (8)$$

Then i 就是 M

这里δ是阀值，α，β是加权系统，i 为下一帧中各目标的标号。

(4)更新被跟踪车辆 M 的面积和质心位置的数据。转到第(2)步继续实现对后续帧的跟踪。

运用上述的跟踪算法就可以实现对某一目标的跟踪， 也能够实现对多目标的同时跟踪。以上算法是建立在这样的假设条件下的：由于两相邻帧的时间间隔很短，因而在相邻两帧之间目标的移动距离并不大， 并且目标在图像中的面积变化率不大。实验表明这种假设是完全满足的， 能很好地实现物体的跟踪。

当两个物体在场景中相重叠时需要首先将它们分开后才能够用上述的算法进行跟踪。

3. 核心代码：

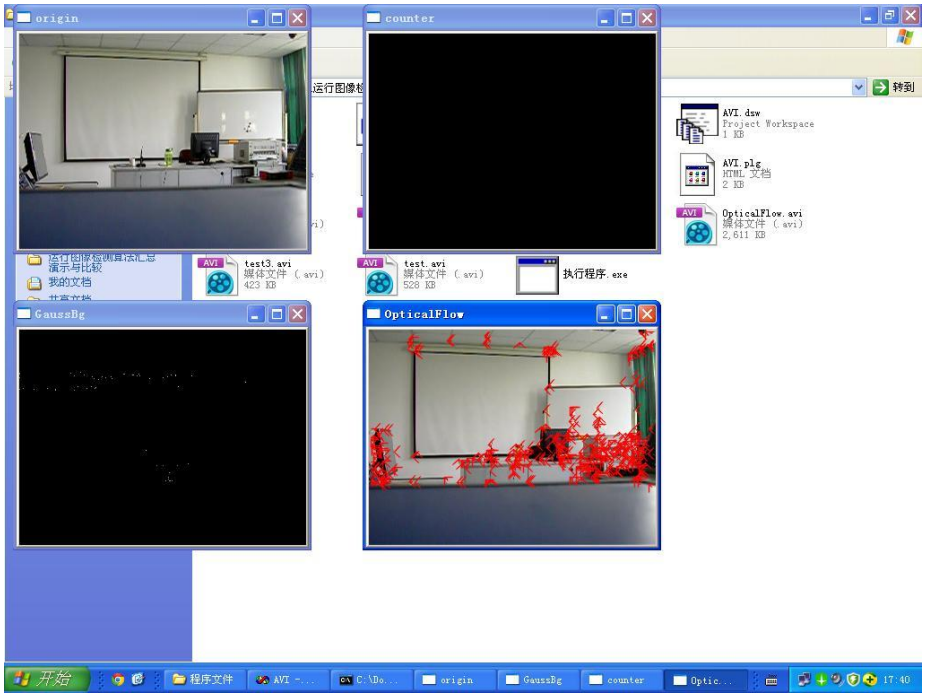
```
IplImage* Avi::ShowGaussBgImage (int nFrmNum,IplImage* pFrImg_cur)
{
    if(nFrmNum == 1)
    {
        cvNamedWindow("GaussBg",1);
        cvMoveWindow("GaussBg",10,330);

        pBkImg = cvCreateImage(frame_size,IPL_DEPTH_8U,3);
        pFrImg = cvCreateImage(frame_size,IPL_DEPTH_8U,3);
        dstB = cvCreateImage(frame_size,IPL_DEPTH_8U,1);
        dstG = cvCreateImage(frame_size,IPL_DEPTH_8U,1);
        dstR = cvCreateImage(frame_size,IPL_DEPTH_8U,1);

        pMatB = cvCreateMat(frame_size.height,frame_size.width,CV_8UC1);
        pMatG = cvCreateMat(frame_size.height,frame_size.width,CV_8UC1);
        pMatR = cvCreateMat(frame_size.height,frame_size.width,CV_8UC1);

        bg_model                                                    =
(CvGaussBGModel*)cvCreateGaussianBGModel(pFrImg_cur,0);
    }
    else
    {
        cvUpdateBGStatModel(pFrImg_cur,(CvBGStatModel *)bg_model);
        cvCopy(bg_model->background,pBkImg,0);

        for(int i=0; i<pFrImg_cur->height; i++)
            for(int j=0; j<pFrImg_cur->width; j++)
            {
                s_fr = cvGet2D(pFrImg_cur,i,j);
```

```
            s_bg = cvGet2D(pBkImg,i,j);

            if((fabs(s_fr.val[0] - s_bg.val[0]) >= threshold) &&
                (fabs(s_fr.val[1] - s_bg.val[1]) >= threshold) &&
                (fabs(s_fr.val[2] - s_bg.val[2]) >= threshold))
            {
                cvSet2D(pFrImg,i,j,s_fr);
            }
            else
            {
                s_fr.val[0] = 0;
                s_fr.val[1] = 0;
                s_fr.val[2] = 0;
                cvSet2D(pFrImg,i,j,s_fr);
            }
        }
    cvErode(pBkImg,pBkImg,0,1);
    cvDilate(pBkImg,pBkImg,0,1);

    //通道分离的 BGR，进行高斯滤波
    /*cvSplit(pFrImg,dstB,dstG,dstR,0);
    cvConvert(dstB,pMatB);
    cvConvert(dstG,pMatG);
    cvConvert(dstR,pMatR);

    cvSmooth(pMatB,pMatB,CV_GAUSSIAN,3,0,0);
    cvSmooth(pMatG,pMatG,CV_GAUSSIAN,3,0,0);
    cvSmooth(pMatR,pMatR,CV_GAUSSIAN,3,0,0);

    cvGetImage(pMatB,dstB);
    cvGetImage(pMatG,dstG);
    cvGetImage(pMatR,dstR);

    cvMerge(dstB,dstG,dstR,0,pFrImg);*/

    //形态学滤波
    /*cvErode(pFrImg,pFrImg,0,1);
    cvDilate(pFrImg,pFrImg,0,1);*/

    pBkImg->origin = 1;
    pFrImg->origin = 1;

    cvShowImage("GaussBg",pFrImg);
    cvWaitKey(10);
```

```
        }
        return pFrImg;
    }

    inline static double square(int a)
    {
        return a*a;
}
```

四、 实验结果

1. 主程序界面：



2. 演示效果一：

3. 演示效果二

五、 感想

本文提出的背景模型在提取过程中允许运动目标存在，因此需要 20~30 秒的时间来完成背景模型的建立。通过对背景模型的实时更新，可以适应光线、天气等外界条件变化带来的影响。这种方法能够在运动物体频繁出现的场合下获得满意的效果，然后运用

质心间的距离与物体面积之差两个参数对检测出来的目标进行实时跟踪。从上述实验结果中可以看到，该方法快速、准确，有着广泛的适用性。

还可以在上述基础上，进行目标识别、流量统计等其它操作，这需要进一步的研究工作。方法还存在一些问题，第 1 个问题是当前景运动目标与背景亮度很接近时，运动目标很难被检测出来，或者是运动目标出现较大的空洞，对于这种 情况，是因为在判断运动目标时，仅仅使用了亮度信息，一个解决的途径是使用颜色与轮廓及深度等相结合的方法。第 2 个问题是如何实时更新背景来适应场景光线的突变，如室内开灯、关灯的情况，目前还没有比较有效的解决方法，有人采用事先存储有代表性的背景的方法，但这种方法对于场景本身发生变化的情况，明显无法适应。因此，这个问题需要在后续工作中进一步深入研究。