基于图像匹配的运动背景补偿方法

李王伟，雷蕴奇
（厦门大学计算机科学系 福建 厦门 361005）

【摘 要】 本文提出了在运动背景下车辆检测的一种有效的算法。首先对前后两帧进行稀疏块匹配，计算出摄像机全局运动速度和方向，经过运动补偿后再通过两帧差分检测出运动区域，最后利用形态学膨胀消除边缘断裂，后经过种子填充，计算连通域面积，去除噪声区域。实验表明，在运动背景的车辆检测应用上，能够较准确地提取出目标车辆。

【关键词】 运动背景 车辆检测 匹配算法

1 引言

随着社会经济的不断发展，城市规模和私人拥有车辆的增加，交通监管的工作量大大增加，利用电子摄像机对各种交通违法违规行为进行监控越来越受到重视。应用先进的计算机视频处理技术，利用摄像机采集车辆信息，并自动进行检测，跟踪和识别，能及时发现违法违规车辆，并进行抓拍，有效地抑制交通事故的发生，保护人民的生命财产安全。

现阶段，利用摄像机和计算机进行视频自动监测和识别，主要应用于摄像机静止的情况下。在这种情况下，由于背景固定，相对来说，利用背景减除，帧差分等方法能够得到较为理想的结果。而基于运动背景的车辆检测和识别由于技术上的难点，至今并没有比较通用的方法出现。

Wouter Beck [1] 提出利用频域相关匹配法计算全局运动参数，这种方法需要对整幅图进行运算，计算量大，而且在目标车辆比较大的情况下误差较大。Weng[2]采用图像边缘和角点作为特征的光流方法，Cumani[3]在每个边缘点处提取三个属性来进行光流特征匹配。以上光流方法缺点在于每个点的计算量大，虽然可以较准确地估算全局运动，但是对于噪声比较敏感，特征的匹配问题还没有很好的得到解决。

本文在单摄像机的条件下，对运动背景下车辆检测做了初步的研究，通过对两帧间进行稀疏块匹配，利用运动补偿来消除动态背景的影响，通过帧差分来定位运动区域，通过数学形态学的腐蚀膨胀来去除杂点和噪声，再计算连通区域，提取出运动车辆，取得了较为满意的结果。

2 运动目标的检测过程

2.1 前后两帧图像的运动补偿

由于摄像机运动的影响，前后两帧上的背景虽然相差无几，但是却发生了一定程度的近似平移，我们不认为它是绝对平移，是因为实际背景是三维的，而反映在镜头上是二维的图像，因此远处的景物和近处的景物发生的平移量是不同的。但是无论如何，整个背景的运动是由于摄像机的运动产生的，我们可以估计摄像机的运动参数，进而估算每一点的平移量。

由于背景运动的存在，直接进行帧差分难以取得良好的效果（图 1a，图 1b），可以看到，图片上很难辨别出车辆。因此我们先对图像帧做运动补偿来尽量消除它对后续工作的影响。

为了提高背景运动估计的鲁棒性，我们把 640*480 的后一帧图像分为 20*15 的块，对前一帧进行匹配。为了提高实时性，我们基于以下几个方面进行了优化：

1. 由于假设背景平移，不必对所有的块进行匹配，水平每四个块随机选取其中的一块，垂直每三个块随机选取一块。

2. 一般监控视频中车辆位于中心位置，为尽量避免选取到前景车辆，我们对中间的 8*6 块不做处理。

3. 考虑到实际车辆的速度，利用摄像机标定的先验知识，行驶的车辆，相邻两帧间的背景运动一般不超过 10 个像素。对于每个 32*32 的图像块，我们只要在 56*56 的框内进行计算，而不必全屏匹配，缩小了计算量。

匹配结束后，采用八参数的双线性模型来估计摄像机的运动参数。

$$\begin{cases} v_x = axy + bx + cy + d \\ v_y = exy + fx + gy + h \end{cases} \quad (1)$$

其中，(x, y) 是点坐标，v_x 和 v_y 是该点在 x 轴和 y 轴的分量，a,b,c,d,e,f,g,h 是要估算的参数。

我们采用最小二乘估计[5]来估算八个参数，由于最小二乘估计对于噪声非常敏感，采用了如下的迭代算法来尽量消除误差的影响：

(1) 取所有的匹配块估计全局运动参数 v_x' 和 v_y'。

(2) 计算平均误差值

$$E=\frac{1}{n}\sum^{N}\sqrt{(v_x-v_x')^2+(v_y-v_y')^2} \quad (2)$$

其中 N 是所有的匹配块数，E 为平均误差。

(3) 以 E 为阈值，把所有和平均误差的差值大于 E 的匹配块删除，剩下的匹配块转到第 (1) 步重新迭代。如果删除的块为 0，则迭代结束。

迭代结束后，可以认为留下来的匹配块都是有效的，此时可以利用二乘估计法估算出八个参数。

在得到摄像机的八个运动参数后，可以利用匹配算法来估计背景运动的大小和方向：

$$\begin{cases} v_x' = x + v_x \\ v_y' = y + v_y \end{cases} \quad (3)$$

其中，x'，y' 和 x，y 分别是修正过的和原来的像素点，v_x 和 v_y 是背景运动在 x 轴和 y 轴的分量，可以利用公式 (1) 计算。

把后一帧图像经过运动补偿后，可以近似认为，它和前一帧图像背景不变。直接进行差分，同时把图像二值化以便后续处理：

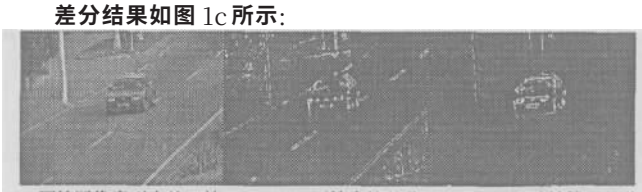
令 {In} 为输入的图像帧，则连续两帧之间像素的差值 Dn 定义为：

$$D_n(i,j) = | I_n(i,j) - I_{n-1}(i,j) |$$

设定阈值 T1，则：

$$f_n(i,j)=\begin{cases} 255, D_n(i,j) >= T1 \\ 0 \quad , OTHERS \end{cases} \quad (4)$$

差分结果如图 1c 所示：



(a)原始图像序列中的一帧 (b) 两帧直接差分 (c) 运动补偿后差分

图 1 视频序列图像直接处理

2.1 噪声的处理

从图 1(c) 中我们可以看到车辆边缘有一些断裂，这是由于运动补偿过程中的误差，以及背景本身运动等因素造成的。我们采用数学形态学中的条件膨胀方法[4]进行处理：

(1) 如果当前像素 A(x0,y0) 是黑色，则遍历其 3×3 邻域。

(2) 若有一点 B(x1,y1) 是白色，转 (3)；否则转 (4)。

(3)遍历A点7×7邻域,统计邻域内黑点数目,若大于阈值T2,则把A进行膨胀,即赋值为白色。

(4)处理下一像素。

膨胀算法的结果如图2(c)所示。

由于帧差算法的局限性,车辆内部会产生空洞,这些空洞有时会导致下面空穴检出的失败。我们先利用种子填充算法[4]对车辆内部进行填充(图2d)。

图像由于摄像机本身抖动等因素会存在一些噪声和杂点,但是这些相对于车辆来说,区域较小,在这里我们对图像进行空穴检出[4],检查每个空穴即连通的白色区域内白点数目,如果小于阈值T3,就将当前空穴所有像素赋以背景值0,进行消去。最后图像中留下的大块白色区域,就是我们所要提取的运动目标(图2e)。

3 实验结果及分析

本文选取充足光照下出租车的情况进行处理,采用帧大小为640×480,帧速25fps的24位RGB视频序列图像,阈值分别取为T1=50,T2=15,T3=100。

从图2中我们可以看到,运动车辆被比较完整的提取出来了,这个结果基本可以让人满意,这对以后的车型识别以及跟踪等后续工作都是一个不错的前提,但是也可以看到,由于噪声以及算法本身的影响,提取结果并不十分精确,提高精确度及鲁棒性,这是我们今后需要改善的地方。

经过上述处理的结果如下图所示:

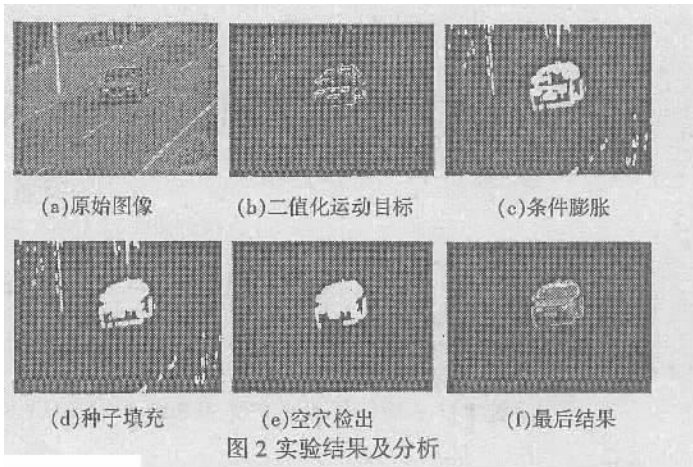


图2 实验结果及分析

4 结论

本文针对当前交通监控系统的实际状况,对摄像机运动条件下的车辆检测做了一些尝试,采用了稀疏块匹配的方法,结合八参数双线性模型估算摄像机的运动,进而预测背景运动的位置,经过补偿后利用两帧差分检测出运动目标,取得了比较满意的结果。本文的算法稳定性和鲁棒性都有待进一步提高,考虑到交通监控系统的实时性要求,算法也有待优化以提高运算速度。

参考文献

[1] Wouter Beck,Image enhancement and moving target detection in IR image sequence, SPIE Vol.2020, 1993, 187- 197.
[2] Weng J, A huja N, Huang T S, Matching two perspective views, IEEE trans Pattern Analysis and Machine Intelligence 1992, PAM I- 14(8):814- 820
[3] Cumani A, Guiducci A, Grattoni P, Image description of dynamic scenes, Pattern Recognition 1991, 24(7):661- 673
[4] Milan Sonka, Vaclav Hlavac, Roger Boyle 著, 艾海舟 武勃等译, 图像处理、分析与机器视觉(第二版), 人民邮电出版社, .2003.9
[5] 陈昌明, 数值分析, 厦门大学出版社, 1998.7

(上接第82页)

4 避免java内存泄漏

Java中的内存泄漏是很难解决的,但庆幸的是大多数java程序中的内存泄漏都十分相似。因此,典型的内存泄漏的种类并不多,而且丰富的经验和合适的工具,将有助于更容易地找出它们。下面,我将描述一些常见的内存泄漏类型。

4.1 忘记删除失效的对象

(Ⅰ)一个十分常见的错误是,把对象放入混编表或混编图中,但在不再需要它们时却忘了删除它们。

(Ⅱ)另一种常见的内存泄漏是,把对象插入到一些其他类的集合中,但在使用之后没有完全删除它们。

4.2 类装载器

Java类装载器创建就存在很多导致内存泄漏的漏洞。由于类装载器的复杂结构,使得很难得到内存泄漏的透视图。这些困难不仅仅是由于类装载器只与"普通的"对象引用有关,同时也和对象内部的引用有关,比如数据变量,方法和各种类。这意味着只要存在对数据变量,方法,各种类和对象的类装载器,那么类装载器将驻留在JVM中。既然类装载器可以同很多的类关联,同时也可以和静态数据变量关联,那么相当多的内存就可能发生泄漏。

4.3 全局集合

在大型应用程序中存在各种各样的全局数据仓库是很普遍的,比如一个JNDI- tree或者一个session table。在这些情况下,注意力就被放在了管理数据仓库的大小上。当然是有一些适当的机制可以将仓库中的无用数据移除。可以有很多不同的解决形式,其中最常用的是一种周期运行的清除作业。这个作业会验证仓库中的数据然后清除一切不需要的数据。另一个办法是计算引用的数量。集合负责跟踪集合中每个元素的引用者数量。这要求引用者通知集合什么时候已经对元素处理完毕。当引用者的数目为零时,就可以移除集合中的相关元素。

5 总结

Java也存在内存泄露问题,其原因主要是一些对象虽然不再被使用,但它们仍然被引用。为了解决这些问题,我们可以通过软件工具来检查内存泄露,检查的主要原理就是暴露出所有堆中的对象,让程序员寻找那些无用但仍被引用的对象。内存泄漏查找起来非常困难,为了在程序开发中减少内存泄漏,文章中列举了一些避免泄漏的例子,以便于开发出安全,健壮的应用系统。

参考文献

[1] BILL VENNERS. Inside the Java Virtual Machine,Secend Edition.
[2] BRUCE ECKEL. Thinking in Java, Third Edition.American:Prentice Hall,2003
[3] JIM PATRICK. Handling memory leaks in Java programs.
http://www- 106.ibm.com/developerWorks/library/j- leaks/index.html, Feb 2001.
[4] HELENA ABERG OSTLUND. Memory Leak Detection with the JRockit JVM, Feb 2005.
[5] Jrockit Userguide. http://e- docs.bea.com/wljrockit/docs142/userguide/memleak.html.