第 35 卷　第 18 期　　计　算　机　工　程　　2009 年 9 月
Vol.35　No.18　　Computer Engineering　　September 2009
·多媒体技术及应用·　文章编号：1000—3428(2009)18—0217—02　文献标识码：A　中图分类号：TP391

视频序列中二值图像的快速目标检测方法

李国辉[1]，焦　波[1]，涂　丹[1]，李　燃[2]，汪彦明[1]
(1. 国防科技大学信息系统与管理学院，长沙 410073；2. 总参工程兵科研三所，洛阳 471023)

摘　要：针对视频序列中二值图像目标检测时间效率不稳定的问题，提出一种快速目标检测方法，其主要步骤包括逐层膨胀、填补空洞、逐层腐蚀和检测目标。理论分析与实验结果证明，该算法的时间效率、时间稳定性和检测质量较高，其最差时间效率与采用 3×3 结构元素的形态学闭算子相当。

关键词：快速目标检测；二值图像；视频序列

Fast Object Detection Method for Binary Image in Video Sequence

LI Guo-hui[1], JIAO Bo[1], TU Dan[1], LI Ran[2], WANG Yan-ming[1]
(1. School of Information System and Management, National University of Defense Technology, Changsha 410073;
2. The Third Engineer Scientific Research Institute, Headquarters of the General Staff, Luoyang 471023)

【Abstract】 Aiming at the instability problem of the object detection time efficiency of binary image in video sequence, this paper proposes a fast object detection method. The main steps of this method include dilating layer by layer, filling holes, eroding layer by layer and detecting objects. Theory analysis and experimental results confirm high time efficiency, time stability and detection quality of this method, and its worst time efficiency is approximately equal to 3×3 structuring element's morphological closing operator's time efficiency.

【Key words】 fast object detection; binary image; video sequence

图像目标检测是计算机视觉的一项主要工作，利用目标特征信息[1-2]或减背景技术[3]可以检测出每帧视频图像中的多数目标像素点，检测与未检测出的像素点构成一幅二值图像。通过图像目标检测将检测出的像素点分类到不同目标或噪声中，并对目标中未被检测出的像素点进行填充。现有二值图像目标检测方法主要包括 *k*-Means[4]和形态学闭算子[1]，它们在数据量大或数据分布不好的情况下，必须在时间效率与检测质量间进行折中。鉴于此，本文提出一种快速目标检测方法。

1　视频序列中二值图像的快速目标检测方法

对于二值图像 *BI* 中的像素点 u，设 $BI(u)=1$ 表示检测出的像素点，$BI(u)=0$ 表示未检测出的像素点，$N_8(u)$表示与像素点 u 相邻的 8 个像素点组成的集合。本文方法的主要步骤包括逐层膨胀、填补空洞、逐层腐蚀和检测目标。

1.1　逐层膨胀

逐层膨胀函数 *IterateDilate*(*BI*, *Iterate*)描述如下：

输入 二值图像 *BI*，膨胀次数 *Iterate*

输出 更新后的 *BI*，连通分支集$\{C_1, C_2, \cdots, C_N\}$

Step1 设 *BI* 包含的全部像素点集为 *AS*，边界像素点集为 *BS*，*AS*/*BS* 的边界像素点集为 *CS*。对 $\forall u \in BS$，令 $BI(u)=2$，令 $LS := \Phi$。遍历 *AS*/*BS* 中的每个像素点 u，若 $BI(u)=1$，则令 $LS := LS+\{u\}$。令 $I := 0$，$HS := \Phi$，转 Step2。

Step2 若 $I<Iterate$，则转 Step3，否则转 Step4。

Step3 遍历 *LS* 中的每个像素点 u，对 $\forall v \in N_8(u)$，若 $BI(v)=0$，则令 $BI(v)=1$, $HS := HS+\{v\}$。遍历完 *LS* 中每个像素点，令 $LS := HS$, $HS := \Phi$, $I := I+1$，转 Step2。

Step4 遍历 *LS* 中的每个像素点 u，对 $\forall v \in N_8(u)$，若 $BI(v)=0$，则令 $BI(v)=3$, $HS := HS+\{v\}$。遍历完 *LS* 中每个像素点。遍历 *CS* 中的每个像素点 u，若 $BI(u)=1$，则令 $BI(u)=3$, $HS := HS+\{u\}$。令 $N := 0$，转 Step5。

Step5 遍历 *HS* 中的每个像素点 u，若 $BI(u)=3$，则令 $N := N+1$，调用子函数 *Search*(*u*, *BI*, *N*)，产生新连通分支 C_N，并更新 *BI*。遍历完 *HS* 中每个像素点，算法终止。

子函数 *Search*(*u*, *BI*, *N*)描述如下：

输出 连通分支 C_N，更新后的 *BI*

Step1 令 $CurSet := \{u\}$, $BI(u)= -N$, $NeiSet := \Phi$, $C_N := \{u\}$，转 Step2。

Step2 若 $CurSet = \Phi$，则算法终止，否则转 Step3。

Step3 遍历 *CurSet* 中的每个像素点 v，对 $\forall s \in N_8(v)$，若 $BI(s)=3$，则令 $BI(s)= -N$, $C_N := C_N+\{s\}$, $NeiSet := NeiSet+\{s\}$。遍历完 *CurSet* 中每个像素点，令 $CurSet := NeiSet$, $NeiSet := \Phi$，转 Step2。

将函数 *IterateDilate* 应用于图 1(a)所示的二值图像，得到图 1(b)所示的结果，其中，灰色区域为膨胀区域，边界线条为连通分支 $C_i(1 \le i \le 6)$。

(a)初始二值图像 (b)膨胀结果　(c)填补结果　(d)腐蚀结果　(e)检测结果

图 1　本文方法处理过程

1.2　填补空洞

填补空洞函数 *FillHoles*(*BI*, C_1, C_2,⋯, C_N)需要将连通分支 $C_i(1 \le i \le N)$分为 2 类，即处于膨胀区域内部和处于膨胀区

基金项目：国家自然科学基金资助项目(60273066)
作者简介：李国辉(1964－)，男，教授、博士生导师，主研方向：多媒体信息系统，多媒体数据挖掘，计算机视觉；焦　波，博士研究生；涂　丹，副教授；李　燃，硕士；汪彦明，博士研究生
收稿日期：2009-04-19　**E-mail：**jiaobonudt116@163.com

域外部，并对膨胀区域内部的连通分支包裹的空洞区域进行填补。填补空洞函数 *FillHoles* 描述如下：

输入 图像 *BI*，连通分支集$\{C_1, C_2, \cdots, C_N\}$

输出 更新后的 *BI*，膨胀区域外边缘的像素点集 *OS*

Step1 用位置坐标(x, y)表示像素点，其中，$0 \le x \le width-1$，$0 \le y \le height-1$。用初始化为 0 的 $L[i]$标记 $C_i(1 \le i \le N)$的状态，令 $L[0]=1$，令 $x:=0$，转 Step2。

Step2 若 $x \le width-1$，则令 $y:=0$, $UP:=0$, $DOWN:=0$，转 Step3，否则令 $OS:=\Phi$，转 Step5。

Step3 若 $y \le height-1$，则转 Step4，否则令 $x:=x+1$，转 Step2。

Step4 若 $BI(x, y)=1$ 且 $UP=0$，则

{令 $UP:=-BI(x, y-1)$，若 $L[UP]=0$，则{若 $L[DOWN]=1$，则令 $L[UP]=1$，否则令 $L[UP]=2$；} }

若 $BI(x, y)=0$ 且 $L[DOWN]=2$，则{令 $BI(x, y)=1$；}

若 $BI(x, y)<0$ 且 $UP \ne 0$，则{

令 $DOWN:=-BI(x, y)$；

若 $L[DOWN]=0$，则令 $L[DOWN]=2$；

令 $UP:=0$； }

令 $y:=y+1$，转 Step3。

Step5 对 $\forall\ 1 \le i \le N$，{若 $L[i]=1$，则对 $\forall\ u \in C_i$，令 $BI(u)=0$，令 $OS:=OS+\{u\}$。若 $L[i]=2$，则对 $\forall\ u \in C_i$，令 $BI(u)=1$；}，算法终止。

将函数 *FillHoles* 应用于图 1(b)所示图像，得到图 1(c)所示的结果。

1.3 逐层腐蚀

逐层腐蚀函数 *IterateErode*(*BI*, *OS*, *Iterate*)描述如下：

输入 图像 *BI*，膨胀区域外边缘的像素点集 *OS*，腐蚀次数 *Iterate*(与膨胀次数相同)

输出 更新后的 *BI*，连通分支集$\{S_1, S_2, \cdots, S_T\}$

Step1 令 $HS:=OS$, $LS:=\Phi$, $I:=0$，转 Step2。

Step2 若 $I < Iterate$，则转 Step3，否则转 Step4。

Step3 遍历 *HS* 中的每个像素点 u，对 $\forall\ v \in N_8(u)$，若 $BI(v)=1$，则令 $BI(v)=0$, $LS:=LS+\{v\}$。遍历完 *HS* 中每个像素点，令 $HS:=LS$, $LS:=\Phi$, $I:=I+1$，转 Step2。

Step4 遍历 *HS* 中的每个像素点 u，对 $\forall\ v \in N_8(u)$，若 $BI(v)=1$，则令 $BI(v)=3$, $LS:=LS+\{v\}$。遍历完 *HS* 中每个像素点，令 $T:=0$，转 Step5。

Step5 遍历 *LS* 中的每个像素点 u，若 $BI(u)=3$，则令 $T:=T+1$，调用子函数 *Search*(*u*, *BI*, *T*)，产生新连通分支 S_T，并更新 *BI*。遍历完 *LS* 中每个像素点，算法终止。

将函数 *IterateErode* 应用于图 1(c)所示图像，得到图 1(d)所示的结果，其中，灰色区域为腐蚀后区域，边界线条为连通分支 S_1, S_2, S_3，它们处于腐蚀后区域的边界处。

1.4 目标检测

目标检测函数 *ObjectDetection*(*BI*, *Size*, $S_1, S_2, \cdots, S_T$)描述如下：

输入 图像 *BI*，噪声阈值 *Size*，连通分支集$\{S_1, S_2, \cdots, S_T\}$

输出 更新后的 *BI*

Step1 用位置坐标(x, y)表示像素点，用初始化为 0 的 $Num[i]$记录 $S_i(1 \le i \le T)$对应连通域内像素点的个数，用 *State* 标记最近刚扫描过的 $S_{(-State)}$。令 $x:=0$，转 Step2。

Step2 若 $x \le width-1$，则令 $y:=0$，转 Step3，否则转 Step5。

Step3 若 $y \le height-1$，则转 Step4，否则令 $x:=x+1$，转 Step2。

Step4 若 $BI(x, y)<0$，则令 $State:=BI(x, y)$，并令 $Num[-State]:=Num[-State]+1$。若 $BI(x, y)=1$，则令 $BI(x, y)=State$，并令 $Num[-State]:=Num[-State]+1$。令 $y:=y+1$，转 Step3。

Step5 对 $\forall\ 1 \le i \le T$，若 $Num[i]<Size$，则 S_i 对应连通域为噪声，否则 S_i 对应连通域为检测到的目标，其中，目标区域内像素点在 *BI* 中的值都为 $-i$，算法终止。

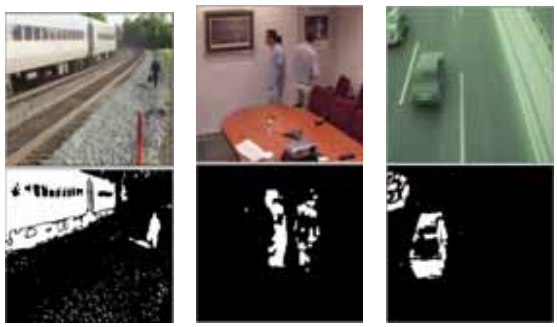
函数 *ObjectDetection* 应用于图 1(d)所示图像，得到图 1(e)所示的结果，其中，相同颜色的像素点在 *BI* 中被赋为相同的值，且较小的深色连通域将被视为噪声并去除。

2 本文方法输入参数的设定

本文方法的输入参数包括膨胀(腐蚀)次数 *Iterate* 和噪声阈值 *Size*。目标特征与背景特征的差异性越小，则目标内检测出的像素点越少，参数 *Iterate* 应适当增大，而参数 *Iterate* 应适当减小。参数 *Size* 用于区分噪声和目标，因为视频中噪声尺寸与目标尺寸的差异很大，所以容易根据目标尺寸的先验信息进行设定。

3 实验结果与对比

采用 3 个 320×240 视频序列中的 3 帧图片对应的二值图像作为测试图像，如图 2 所示，其中，*C* 表示每帧图片中检测出的像素点个数。对本文方法、*k*-Means 和形态学闭算子进行测试。将形态学闭算子之后的连通域标记作为形态学闭算子的一部分。本文实验的硬件环境是 Pentium(R) 4 CPU 2.40 GHz，内存 512 MB。软件环境是 Microsoft Windows XP 操作系统，算法采用 VC6.0 编写。

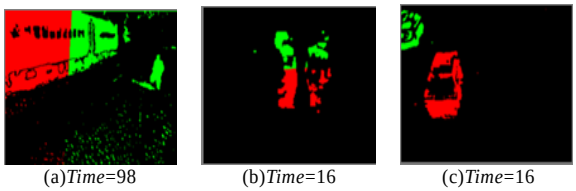


(a)*C*=19 929 (b)*C*=4 059 (c)*C*=4 498

图 2 测试图像

k-Means 的输入参数为 *k*(聚类个数)和 *t*(迭代次数)，形态学闭算子的输入参数为 *m*(结构元素半径)和 *T*(噪声阈值)，本文方法的输入参数为 *I*(膨胀或腐蚀次数 *Iterate*)和 *S*(噪声阈值 *Size*)。本文采用 clock()函数测试算法的 CPU 执行时间，以 100 次运行结果的均值代表最终的 CPU 执行时间。*k*-Means、形态学闭算子和本文方法的 CPU 执行时间都用 *Time*(单位为 ms)表示。

图 3~图 5 分别给出了 *k*-Means、形态学闭算子和本文方法对上述测试图像的实验结果，图 4 和图 5 中的白色像素点代表噪声。



(a)*Time*=98 (b)*Time*=16 (c)*Time*=16

图 3 *k*=2, *t*=9 时 *k*-Means 的实验结果

(a)*m*=2,*T*=300,*Time*=15 (b)*m*=7,*T*=300,*Time*=24 (c)*m*=6,*T*=300,*Time*=22

图 4 形态学闭算子的实验结果

(下转第 226 页)