

学士学位论文

基于邻域直方图分布的Vibe算法

学 院 计算机科学与技术

专 业 计算机科技与技术

研 究 方 向 数字图像复原

学 生 姓 名 杨心如

学 号 20131202081

指导教师姓名 洪留荣

指导教师职称 教授

2017年4月22日

基于邻域直方图分布的Vibe算法

**摘要** 运动目标检测已经称为计算机视觉领域中举足轻重的一个分支，其中有很多方法用以分割运动目标，而最为经典的即为前景分割，即使用背景模板来分割出前景运动物体，而Vibe即为这一思想下的典型算法产物。然而因为光线、硬件甚至温度等原因，背景中的鬼影或阴影会为分割带来很大的干扰。为解决原Vibe算法中存在的鬼影或阴影消除滞后的问题，利用鬼影与背景帧相似而运动目标与背景帧差异更大的特点，提出一种基于邻域背景像素直方图相似度匹配的方法来快速检测鬼影并更新背景模型，本文使用相关系数来表征相似度。在汽车快速通过道路的视频上进行仿真和实验，最终结果表明，改进后的算法将鬼影消除效率提高了至少50%，事实表明，这种方法在原理上完全可行，但亦需要更多的优化来提高其工作效率。

**关键词** Vibe，帧差法，直方图，目标检测**Ghost Removal and Detection of Moving Target based on Vibe Algorithm**

**Abstract** Moving target detection has been called a pivotal branch in the field of computer vision. There are many ways to segment moving targets, and the most classic is foreground segmentation, which uses background templates to segment foreground moving objects, and Vibe is this. A typical algorithm product under the idea. However, due to light, hardware, or even temperature, ghosts or shadows in the background can cause a lot of interference in the segmentation. In order to solve the problem of ghost or shadow elimination lag in the original Vibe algorithm, using the similarity of ghost image and background frame and the difference between moving target and background frame, a similarity based on neighboring background pixel histogram similarity matching is proposed. The method is to quickly detect ghosts and update the background model. This paper uses correlation coefficients to characterize similarity. Simulations and experiments were carried out on the video of the car passing the road quickly. The final result shows that the improved algorithm improves the ghost elimination efficiency by at least 50%. The facts show that this method is completely feasible in principle, but it also needs more Optimize to increase its productivity.

**Key words** Vibe, Frame difference method, Histogram, Target Detection

**目 录**

[绪论 1](#_Toc6822223)

[1.1论文研究的目的及意 1](#_Toc6822224)

[1.3论文主要研究内容 2](#_Toc6822225)

[2. Vibe算法基础 2](#_Toc6822226)

[2.1算法概述 2](#_Toc6822227)

[2.2算法模型 3](#_Toc6822228)

[2.3实验结果 4](#_Toc6822229)

[3. 基于直方图改进的Vibe算法 5](#_Toc6822230)

[3.1算法概述 5](#_Toc6822231)

[3.2算法模型 5](#_Toc6822232)

[3.3代码实现 6](#_Toc6822233)

[3.4实验结果 7](#_Toc6822234)

[总结 8](#_Toc6822235)

[参考文献 9](#_Toc6822236)

[致 谢 10](#_Toc6822237)

[附录A 11](#_Toc6822238)

绪论

## 论文研究的目的及意

人类第一次真正意义上的数字图像处理是在1960年左右，喷气推进实验室 （Jet Propulsion Laboratory，JPL）使用数字图像处理技术处理了大量的月球表明图像。在之后的几十年发展中，数字图像处理已经成为计算机视觉（CV）中不可忽略的一部分。现在数字图像处理的应用已经深入生活，大到太空探索，例如人类获取到的第一张黑洞的图片就是由成吨的数据最终处理而得，小到我们日常的支付和医疗，如支付宝的线下人脸识别付款及X-ray辅助诊断软件等。

而对运动中的目标进行检测（Vibe）是当下继目标识别及目标追踪后，在计算机视觉中举足轻重的一项关键技术。最常用的方法是利用当前图像与背景图像进行差分运算，进而得到前景图像，但由于摄影镜片及光线等原因，在阈值差分操作后，图中会有一些“鬼影”，那些鬼影可以通过一些颜色通道的方式来去除，但其中的参数很难普适。其实鬼影和运动目标相比，它和背景模型的相似度更高，我们可以根据这一特性来去除“鬼影”。鬼影或阴影的存在，其实也是像素随机变化的一种反应，而Vibe算法最大亮点就是在两个维度上的随机性，在某种程度上也模拟了像素的随机变化，这样可以更好的优化背景模型。

鬼影或阴影有的时候会和移动物体具有相似的几何、颜色等特征，但其颜色直方图的分布和移动物体具有很大的差异性，也就是说从统计学这一新的角度来解决分割问题，这给后续问题的解决引导了一个新的方向。

* 1. **国内外研究现状**

计算机视觉，顾名思义，就是使计算机具有人类的视觉功能，即通过一幅或多幅图获得周边的信息。作为CV中一个重要的分支，运动目标检测常常用于对视野内的目标，特别是运动目标进行观察、识别及追踪等。目前，全世界范围内几乎所有高校以及很多公司都投入极大人力和财力来研究智能监控系统的相关项目，国外如IBM，Microsoft以及Google等，国内也有百度，腾讯，阿里巴巴，京东等巨头公司，他们甚至也推出了很多相关产品来服务自己或服务客户。如支付宝的线下人脸识别支付，京东的无人生产线等等。

针对鬼影检测问题的文献[1]结合了帧间历史信息，使用二次判断的策略，提出与帧差法相结合来抑制或消除鬼影的方法；文献[2]利用更大的更新参数快速消除鬼影，但带来相应的问题，会将运动较慢的物体加入到背景模型中。针对阴影检测主要有基于几何模型[3]以及基于阴影特征[4-5]等两类方法。基于几何模型的方法一般只能在特定环境条件下使用，而且用它进行建模和计算都是极其复杂的；基于阴影特征的检测方法通常假设投射阴影亮度低于某个阈值而且没有明显的色度变化，这一般用于容易进行色彩比较的色彩空间，比如RGB或者HSV等。但其要求相对比较严苛，而且遇到彩色阴影时，其行能又会下降。

## 1.3 论文主要研究内容

鬼影区域和它邻近背景区域的像素特征非常相似，而运动目标与背景像素特征差异较大，本文利用这一特性，在Vibe算法提取前景的基础上计算前景区域与邻近背景区域的像素分布直方图，设立匹配阈值从而有效的区分出运动目标和鬼影。至于色彩空间的选取问题，本文选取RGB色彩空间，并且没有直接比较，而是直接转为灰度图进行处理，最后进行阈值分割。

本论文主体分为章，文章首先探讨了Vibe算法的原型。绪论部分介绍了计算机视觉的发展及意义，运动目标检测在计算机视觉中占据的地位，国内外发展前景，本文算法的改进以及论文的架构思路。第二章介绍了Vibe算法的原型，并给出测试影像在原Vibe算法下的实验结果，在该章的最后，会根据实验结果具体分析出现相应结果的原因及具体改进方法。第三章主要针对原Vibe算法中出现的鬼影现象提出基于邻近背景像素直方图的方法来抑制甚至消除鬼影的影响。第四章最后作为总结，将前后两种方法以及单纯使用差分法的结果进行对比探究，提出本文中所显现的局限性及对运动目标检测技术的未来展望。

2. Vibe算法基础

## 2.1 算法概述

运动物体检测的主要目标是找出ROI（region of interest）,确定其位置、大小以及颜色分布等信息。由于各类物体在不同角度下可能呈现不同的形状，大小甚至颜色，再加上光线的影响，这些都使得目标检测称为计算机视觉领域最具挑战性的问题之一。计算机视觉对于图像处理的四大任务是：分类（Classification），定位（Location），检测（Detection），分割（Segmentation），其中分割是本文的重点。

Vibe算法被第一次提出是在2011年， Olivier Barnich以及Marc Van Droogen- broeck联合研究并公布了这一算法，在最基本的帧差法，也就是固定背景模型的基础上，Vibe算法是一种通过实时更新背景模型的前景检测算法。ViBe依靠通过对背景建模从而分割出前景物体的方法，而且其操作都是像素等级的，不需要认为构造一些结构。和其同类型算法相比，Vibe算法使用了不同的背景模型更新策略，主要体现在随机性上，即随机选择样本来替换像素，随机选择邻域范围内的像素进行更新，通常使用8邻域等。对于阴影或者鬼影来说，他们在某种程度上是不确定的，而Vibe算法的随即更新策略又可以在某种程度上来恰好模拟他们的这种不确定性，降低了人工建模的复杂度，又可以提高建模效率。

## 2.2算法模型

我们先假设，每个像素及其邻域像素值有着相似的分布。基于这种假设，每一个像素模型都可以用其邻域中的像素来表示。。当输入第一帧图

。

符合条件N次之后，为背景像素，否则就视其为前景像素，紧接着将他们更新到背景模型中。

。

当我们得到新的一帧图像时，则需要被更新。

用的来替换掉，

可以去测目标更加准确。

## 2.3实验结果

如图，左边一列为未经过Vibe算法进行背景模型更新对原视频第101帧到104帧的处理结果，右边一列为经过背景模型更新的Vibe算法对原视频第101帧到104帧的处理结果，图中用红色圆圈标注的即为鬼影，可以看出，由于测试视频在前面的部分背景变化不大，所以没有使用Vibe算法更新背景模型的分割效果在前半段和更新后的效果相差不大。

但从303帧之后，分割的结果中的背景噪声猛增，而且持续增长，无法控制。下图分别为第303帧即第367帧的分割结果，可以看出由于无法更新背景模型，最后分割结果只会被湮灭在嘈杂的背景之中。

由于背景相对比较复杂多变，可以看到如果对背景模型没有一定的更新策略，分割结果由于自身无法及时改正很容易受到干扰。而由于Vibe采用随机的更新策略，所以当出现干扰时，算法会自适应改变背景模型来消除干扰，可以看到Vibe算法下的101帧到104帧不单鬼影大小及形态会收敛很多，而且整体有减小的趋势，可以明显看到算法对鬼影和阴影的抑制效果。



3. 基于直方图改进的Vibe算法

## 3.1算法概述

鬼影和阴影是图像中不可避免的一部分，但Vibe原算法很难快速消除这些影响，容易带来一些意外的结果。本文利用鬼影和邻域背景像素的直方图分布更相近，而与前景运动目标像素的直方图分布差异更大这一特性来检测鬼影及阴影。

## 3.2算法模型

，目标区域，，应的程，RF

如果其中相关系数计算得到融入背景模型。

## 3.3 代码实现

本文代码均由matlab编写，软件版本为16.0 a，具体代码详见[附录A](#_附录A)。

在分割完成后，将分割结果转换为8位的binary图像，结果中带有一些孔洞结构，可以进一步通过形态学操作（膨胀，腐蚀等）来对其进行修正，但为了更直观的查看改进后的Vibe算法的处理效果，没有对分割结果进行进一步的修正。

为统计各连通域的颜色直方图分布，使用了matlab中自带的函数bwboundaries来求出分割结果中存在的所有连通域（不包括孔洞），然后分别对每个连通域及其背景邻域求其颜色直方图分布，然后求出。

在具体实现过程中，由两个矩阵相关系数的倒数决定，即相关系数越大，两个颜色直方图的距离越小，那么我们设定一个阈值，如果某一块区域（连通域）的前景与随机背景区域的相关系数大于这个阈值，即可将这块区域更新到背景模型之中。

## 3.4实验结果

在Vibe算法参数一致的情况下，可以明显观察到，利用邻域直方图的结果中，鬼影或阴影消退的更快，其影响更小。如图，左边一列为原Vibe算法对原视频第101帧到104帧的处理结果，右边一列为基于直方图分布对比的Vibe算法对原视频第101帧到104帧的处理结果，图中用红色圆圈标注的即为鬼影，可以明显看到，第101帧为出现鬼影的第一帧，两种方法的处理结果只更新到了背景模型上，没有什么大的差别，而第102帧里，改进后的算法和原算法相比无论从大小还是形态上都明显抑制了鬼影，第二次更新之后，改进后的算法完全消除了鬼影，而原算法仍然需要两帧才可以达到同样效果，即效率上提高了50%以上。

其实对于运动目标的检测效果来说，也是有一定的优化效果，可以看到物体周边的毛刺少了很多，其实那些也都是汽车运动时的残影或阴影，去除那些连通域之后，整个分割效果也平滑了很多。

总结

随着计算机科学技术的日益发展，数字图像处理以及机器视觉也随之蓬勃发展，其需求也日益扩张，而对运动目标的检测更是变得尤为重要。由于日光、灯光、拍摄角度等因素的影响，对运动物体的检测变得困难，而提出Vibe等基于背景模型更新的算法，降低了调参的时间成本，可以使模型自适应。

但由于鬼影和阴影的影响，原Vibe算法对其的消除速度非常缓慢，可以调整更新速率来改善这一现象，但这样又会带来新的问题，所以本文提出的基于邻域直方图的鬼影去除算法在某种程度上很好的解决了这一问题，可以看出他极大的削弱了鬼影和阴影对分割的影响。

本算法亦有很多不足之处。其时间复杂度增加，在不对其数学模型进一步优化的前提下，时间基本上为原来的1.5倍。本文没有继续讨论4邻域，8邻域及更大参数下的结果，使得最总的实验结果有些许的局限，但不影响算法本身的展示。本文中使用相关系数来计算，但亦可以使用欧式距离等其他方法来计算这个参考值。希望本文之方法的提出可以为后人抛砖引玉，为计算机视觉的事业增砖添瓦！

参考文献

[1] Mao Z C,Shen X S. Improved Vibe algorithm integrated with multiscale transformation[J].Laser & Optoelectronics Progress, 2018, 55(11): 111501.茅正冲，沈雪松.融合多尺度变换的改进Vibe算法[J]. 激光与光电子学进展, 2018, 55(11):111501.

[2] Xie S R, Ye S B, Yang B H, et al. Moving targets detection based on an improved YUV\_Vibe fusion algorithm[J]. Laser & Optoelectronics Progress, 2018, 55(11):111002.

谢申汝,叶生波,杨宝华,王学梅,何红霞.基于改进的 YUV\_Vibe 融合算法的运动目标检测[J].激光与光电子学进展,2018, 55(11):111002.

[3] Martel-Brisson N ,Zaccarin A. Moving cast shadow detection from a Gaussian mixture shadow model[C].IEEE Computer Society Conference on Computer Vision & Pattern Recognition,2005,1(2): 643-648.

[4] Yuan J, Wu J, Cheng Y. Shadow detecting algorithms research for moving objects base on self-adaptive background[C]. IEEE International Conference on Modeling, Identification &Control, 2012: 197-200.

[5] Salvador E, Cavallaro A, Ebrahimi T. Cast shadow segmentation using invariant color features[J]. Computer Vision and Image Understanding, 2004, 95(2): 238-259.

# 致 谢

首先，特别感谢我的本科导师洪留荣教授在我写本篇论文期间给予我宝贵的论文指导意见。在大学四年的学习中，洪留荣老师严谨的教学风格以及科学的工作方法给予了我很大的帮助以及影响，是我在计算机科学与技术专业的领路人，再次真诚的表达我对导师的感激之情。

另外，特别感谢我的辅导员张娟老师给予我大学四年在学习生活上的帮助，因为老师的用心，才使我们拥有这样一个积极向上的班集体，温暖有爱的大家庭。感谢给我授课的老师们，你们的教学扩展了我的视野，解除了我学业上的迷惑。衷心的向老师们道一声感谢，老师辛苦了！也特别感谢一同与我在学习生活上共同奋斗的同学，是他们的陪伴，鼓励，帮助使得我在大学四年的生活过得如此充实，对未来充满着美好的憧憬。

最后要感谢的是我的父母，感谢他们抚养我长大，感谢他们在我成长路上的一路支持，因为有他们给与我最坚实的臂膀，才可以更加勇敢的前行。

# 附录A

|  |
| --- |
| **run.m**  % this is a main "function" file  % so you just need to ensure that you are at the correct folder and just  % type run    % clear to clean all the variables in Workspace in case of interruption  % clc to clean the output show, it's not necessary  % close all to close all figures, it's not necessary either.  clear; clc; close all;  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  % Initiate Parameters  % Initial sample numbers is 10  param.numberOfSamples = 10;  % Initial matching theshold is 10  param.matchingThreshold = 10;  % Initial matching number is 2  param.matchingNumber = 2;  % Initial update factor is 5, can't be too great  param.updateFactor = 5;  % Initial history images number is 2  param.numberOfHistoryImages = 2;  % Initial swapped histroy image is 0  param.lastHistoryImageSwapped = 0;    % read a video as input  % it must be placed at the same directory with this file  filename = 'video.avi';    % read one frame first.  vidObj = VideoReader(filename);    % a flag variable to flag the first time to initiate  firstFrame = true;    % get frame's size  height = vidObj.Height;  width = vidObj.Width;    % store frame's size info into parameter  param.height = height;  param.width = width;    % frame counter, just to show debug infomation  frame = 0;    % show figure 1 window  % you'd better put it out of the loop to speed scope up  figure(1)    % Moving object detection till the video ends up  while hasFrame(vidObj)  % count for frame  frame = frame + 1;  % read an another new frame  vidFrame = readFrame(vidObj);  % it's just for debug  % because 1 to 64 frames are nothing  % just for time reducing  if frame < 55  continue;  end  % create a 2 by 2 suplots  % show original image at first subplot  subplot(2,2,1), imshow(vidFrame),title('original');  % shou frame number at coordinate (10, 10)  text(10,10,num2str(frame));  % trans RGB to gray for more conenient operation  vidFrame = rgb2gray(vidFrame);  % for more precision, make it double  vidFrame = double(vidFrame);    % start timing  tic;  % if it is the first frame  % then initiate the Vibe model first  if firstFrame  firstFrame = false;  initViBe;  end  % use threshold to get segmentation  % be careful, this function input a double matrix and output an uint8  % one instead  segmentationMap = vibeSegmentation(vidFrame, historyImages, historyBuffer, param);  % update background model  [historyImages, historyBuffer] = vibeUpdate(vidFrame, segmentationMap, historyImages, historyBuffer, param, ...  jump, neighborX, neighborY, position);  % to get a better vision, make segmentation binary  segmentationMap = medfilt2(segmentationMap);    % get all Connected domains except for holes  [B,L] = bwboundaries(segmentationMap,'noholes');  % get max number of kind for all the connnected domains  max\_ = size(B,1);  % declear a variable for storing a random index  index = 1;  % show segmentation image at second subplot  subplot(2,2,2),imshow(segmentationMap),title('segmentation');  hold on;  % check if there is any detection  if max\_ ~= 0  % handle every single situation independently  for iii=1:max\_  % get a random index in it's range  % you must make it ceilling  % because it must greater than 0  index = uint8(ceil(rand()\*size(historyBuffer,2)));  % get boundary ceil  boundary = B{iii};  % filter the iii th connected domain  tempItem = (L == iii);  % get the background of iii th connected domain  tempBackground = historyBuffer{index}.\*tempItem;  % get Histogram distribution of item image  itemRhist = hist(tempItem(:),1:1:256);  % get Histogram distribution of background image of item  bgRhist = hist(tempBackground(:),1:1:256);  % get the Relationship coefficient between the histogram  % distribution of item image and Histogram distribution  % of background image of item  g = corrcoef(itemRhist ,bgRhist);  % in expriment, I found if it's ghost, it's Relationship  % coefficient are almost all greater than 0.999  % which is really big but really works  if g(1,2) > 0.9999  % to get its row and col range  row = boundary(:,2);  col = boundary(:,1);  % try not to make it out of normal range  row(row<=0) = 1;  col(col<=0) = 1;  row(row > param.height) = param.height;  col(col > param.width) = param.width;  % update its pixels to the background  historyBuffer{index}(row,col) = vidFrame(row,col);    end  % display the kind number and Relationship coefficient  disp([iii,g(1,2)]);  % display the kind number in the connected domain in the plot  % to display better, random it's loation around the range  rndRow = ceil(length(boundary)/(mod(rand\*iii,7)+1));  col = boundary(rndRow,2); row = boundary(rndRow,1);  h = text(col+1, row-1, num2str(L(row,col)));  set(h,'Color','m','FontSize',14,'FontWeight','bold');  end  end  % end of timing  toc;  % format file name with it's index  name = sprintf('%d.jpg',frame);  % save the segmentation for this frame  imwrite(segmentationMap,name,'jpg');  end  initVibe.m  % copy Parameters as alias  numberOfSamples = param.numberOfSamples;  matchingThreshold = param.matchingThreshold;  matchingNumber = param.matchingNumber;  updateFactor = param.updateFactor;  numberOfHistoryImages = param.numberOfHistoryImages;    % Initialize ViBe virables  % history images are stored at 1 by 2(number of history images) cell  historyImages = cell(1, numberOfHistoryImages);  % initiate the first 2 images with current original images  % or there must be some errors after  for ii = 1:length(historyImages)  historyImages{ii} = vidFrame;  end    % history images are stored at 1 by 8(number of samples minus  % number of history images) cell  historyBuffer = cell(1, numberOfSamples - numberOfHistoryImages);  for ii = 1:length(historyBuffer)  % it's not exactly equal to current orignal frame  % it has a random bias  historyBuffer{ii} = vidFrame + double(floor(rand(height, width))\*20 - 10);  end    %% Random Part  size\_ = 2\*max(height, width) + 1;  % jump[] from 1 to 2\*updateFactor  jump = floor(rand(1, size\_)\*2\*updateFactor) + 1;  % neighborX, Y represent the neighbor index  neighborX = floor(rand(1, size\_)\*3) - 1;  neighborY = floor(rand(1, size\_)\*3) - 1;  % position[] from 1 to numberOfSamples  position = floor(rand(1, size\_)\*numberOfSamples) + 1;    % show initiation finished signal  disp('Initialize ViBe')  **segmentationMap.m**  function segmentationMap = vibeSegmentation(buffer, historyImages, historyBuffer, param)  %% Parameters  height = param.height;  width = param.width;  numberOfSamples = param.numberOfSamples;  matchingThreshold = param.matchingThreshold;  matchingNumber = param.matchingNumber;  numberOfHistoryImages = param.numberOfHistoryImages;    %% Segmentation  segmentationMap = uint8(ones(height, width)\*(matchingNumber - 1));  % First and Second history Image structure  distance1 = abs(buffer - historyImages{1}) <= matchingThreshold;  distance2 = abs(buffer - historyImages{2}) <= matchingThreshold;    for ii = 1:height  for jj = 1:width  % check if distance 1 is a zero matrix  % make it into matching number  if ~distance1(ii, jj)  segmentationMap(ii, jj) = matchingNumber;  end  % check if distance 2 is an one matrix  % make it minus 1 to pull it off the updating  if distance2(ii, jj)  segmentationMap(ii, jj) = segmentationMap(ii, jj) - 1;  end  end  end  % match the image and samples  numberOfTests = numberOfSamples - numberOfHistoryImages;  % update the mask in time  for kk = 1:numberOfTests  distance3 = uint8(abs(buffer - historyBuffer{kk}) <= matchingThreshold);  segmentationMap = segmentationMap - distance3;  end  % make the segmentation image from double to unsigned int 8  segmentationMap = uint8(segmentationMap\*255);  end  **vibeUpdate.m**  function [historyImages, historyBuffer] = vibeUpdate(buffer, updatingMask, historyImages, historyBuffer, param, ...  jump, neighborX, neighborY, position)  % copy Parameters as alias  height = param.height;  width = param.width;  numberOfHistoryImages = param.numberOfHistoryImages;    % Update Model  for indY = 2:height - 1  shift = floor(rand()\*width) + 1;  indX = jump(shift) + 1;  while indX < width  if updatingMask(indY, indX) == 0  value = buffer(indY, indX);  if position(shift) <= numberOfHistoryImages  historyImages{position(shift)}(indY, indX) = value;  historyImages{position(shift)}...  (indY + neighborY(shift), indX + neighborX(shift)) = value;  else  pos = position(shift) - numberOfHistoryImages;  historyBuffer{pos}(indY, indX) = value;  historyBuffer{pos}...  (indY + neighborY(shift), indX + neighborX(shift)) = value;  end  end  shift = shift + 1;  indX = indX + jump(shift);  end  end    end |