参赛密码 _____

（由组委会填写）

“华为杯”第十四届全国研究生数学建模竞赛

学　　校　上海交通大学

参赛队号　10248299

队员姓名
1. 陈若冰
2. 江 杉
3. 陈 晓

参赛密码 _____

（由组委会填写）

“华为杯”第十四届全国研究生数学建模竞赛

题　　目　基于监控视频的前景目标提取

摘　　要：

在计算机视觉领域中，前景目标提取是一个重要的分支。本文对视频监控中的前景提取问题进行了研究，分析了固定摄像头静态背景、固定摄像头动态背景、摄像头抖动以及多摄像头场合下的前景提取所面临的一系列问题，通过合理的假设，结合一些已有的算法和技术，分别建立了可以应对这些情况的前景提取模型，并通过对题目所给样例进行测试，验证了模型的有效性。对于使用前景提取结果进行群体行为的分析问题，本文提出了相应的检测模型，并选取一些样例来测试模型的合理性。

针对问题一、二和四中的固定摄像头前景提取问题，本文结合前景提取任务的一般流程：背景建模，像素值比对与后续处理，分别建立了像素点的相似性模型——LBSP 算子模型；背景比对与更新模型——ViBe 动态样本集合模型；离线背景生成模型——一维 Gabor 滤波与直方图分析模型。其中一维 Gabor 滤波的方法采用离线分析的手段生成背景的初始化模型，同时保留了在线处理视频的能力，在图像处理领域较为少见，为本文的一个创新之处。以上模型共同构成了固定摄像头前景提取的整体模型。经过测试，该模型很好地解决了固定摄像头下前景提取问题，在静态背景和动态背景下均表现良好，与预期的结果一致。

针对问题三中摄像头抖动情况下的前景提取问题，本文分析了该情景与固定摄像头情景下的一些区别，并从常规思路出发，通过建立前后帧的单应矩阵

映射关系，寻找抖动前后摄像头画面的公共区域进行比对，将固定摄像头下的前景提取方法应用到摄像头抖动的场合中，同时还建立了动态参数反馈模型进一步消除映射带来的误差，提升模型的性能，它们与 LBSP 算子和 ViBe 动态样本集合共同构成了摄像头抖动情况下的前景提取模型。经过测试，该模型很好地解决了摄像头抖动前景提取问题，取得了优异的结果。

针对问题五中的多摄像头前景提取问题，本文在固定单个摄像头前景提取已经取得良好结果的基础上，建立了多摄像头三维模型来寻找摄像头之间的画面的映射关系，并在问题一的基础上利用融合信息改进了 ViBe 算法的背景比对与更新模型，进一步优化了每个摄像头的提取结果，为本文的又一个创新点。经过测试，这一改进后的模型提取结果良好，完成了多摄像头前景提取任务。

针对问题六，本文分析了从视频当中检测群体行为的一般流程，针对特征提取环节构建群体运动特征模型。本文在进行了大胆而合理的假设之后，创造性的提出用角点的统计特性来估计前景中个体的统计特性的方法，建立了角点检测模型和群体运动特征模型，该模型给出了从前景图片当中计算与检测群体行为有关的特征的方法，此类特征可用于结合大量数据进一步训练分类器，构成完整的群体行为检测系统。为测试模型的合理性，本文选取了不同场景下一些小的视频片段，计算了有关的特征，经过验证，模型能表达前景中的群体行为。

本文立足于数学建模的研究方法，在没有使用复杂的机器学习模型以及大量数据进行训练的情况下，仅使用题目所给的视频数据，完成了全部问题的分析、建模与求解，体现了数学建模这一思想的简洁、优雅与高效。

关键字：前景提取　LBSP 算子　Gabor 滤波　多摄像头建模　单应变换　角点检测

目录

1. 引言

在如今这个信息膨胀的时代，我们可以随时从各种网络，手机，通讯设备里获得大量的视频信息。由于数据过于庞大，因此对其进行智能处理的需求就变得日渐显著。在视频处理中，前景检测是其首要的任务，这不仅可以大大减少计算机在之后视觉处理时的计算负载，同时也对视频监控等实际存在的问题有着至关重要的帮助。

前景检测指的是在视频或者图像当中，以一定的技术或者手段将人们感兴趣的区域提取出来，或者是将运动的目标定位。前景检测之所以一直受到人们的广泛关注，究其根本，原因主要是其在民用、军事等各领域都有着非常重要的应用，并且从技术层面来说，它还是计算机视觉处理中跟踪，定位，运动分析识别等技术的基础。

在交通系统中，前景检测通常被用在车辆检测，行人行为分析等方面。在未来的自动驾驶技术中，前景检测也会大放异彩。在民事安全中，前景检测则被广泛运用在了各种公众场合，与家庭住所的监控中，以此来保证社会稳定与人们的人生安全。在军事方面，前景检测则主要展现在定位，追踪等方面。通过对地方军事目标的定位，可以对其进行有效的打击与制定作战策略。同样在制导中，前景检测也有着很重要的辅助作用。

由于视频在拍摄时候的条件与环境不同，可能导致两种情况，一种是固定摄像头下拍摄的视频，而另外一种则是移动摄像头下拍摄的视频。在处理这两类视频时，往往采用的是完全不同的前景提取技术，而由于移动拍摄设备的日渐普及，移动场景下的前景提取在视频处理中的地位日渐显著。此外，越来越多的场合应用多摄像头协作的方式进行监控，此类环境下的前景提取技术也逐渐引起研究者的关注。

随着近几年人工智能技术的飞速发展以及机器学习领域算法和应用的不断进步，采用海量数据训练庞大的计算机视觉模型逐渐成为解决诸如前景提取这类问题的主流方法，并且可以得到非常优秀的处理效果。然而本文认为此类方法立足于复杂的神经网络结构，以及大量用于训练的带标签的样本数据，庞大的计算资源，较长的训练时间，并不符合数学建模问题中简洁、高效、额外数据少的特点。因此本文并没有选择此类方法，而是决定仅仅立足于题目中所给的参考视频，没有使用任何带标签的视频数据进行训练，在此基础上建立有效的模型解决题目要求中的各种任务，取得了不错的结果。

2. 问题重述

对于一段由摄像头拍摄到的或长或短的监控视频，本文要解决的问题为提取视频前景目标。其中前景目标定义为人们感兴趣物体，以及在一个固定场景中的运动物体。一般来说，人们概念中的背景是指视频中与外界静止的大环境融为一体的物体或区域，例如天空、大地、植被、街道、建筑物等，一些虽然具有移动属性但是也可算入此类的物体依然会被认为是背景，如风吹过的树叶、水流、喷泉、窗帘等等。与之相比，前景则是指视频中与背景具有不同运动特性的物体，它们在结构上与背景是分离开的，一般来说其纹理特征也与背景不同，如车辆、行人、动物、船只、飞机等。

前景提取问题为在一个固定的视频中，采用在线或离线的方式，判断视频中每一帧的每个像素点属于前景还是背景。一般来说，属于前景的像素点的像素值会被设为 255（白），而背景像素点的像素值会被设为 0（黑）。将视频中所有的像素点分类过后，便得到了一个只包含像素点属性信息的 mask 视频，该视频即可作为前景提取问题的输出。

由于各问题的目标相同，而难度和所考虑的条件是递进式的，所以本文将一些情况类似的问题合并分析并建模求解。问题一、二、四均固定摄像头下的前景提取；问题三为移动摄像头下的前景提取；问题五为多摄像头下的前景提取；问题六为基于前景提取结果的人群行为分析。以下分别对这些问题进行分析、建模及求解。

3. 问题一、二、四

3.1 问题分析

问题一、二、四均为固定摄像头的前景提取问题，所不同的是问题二与四还要考虑背景本身存在的一些动态特性（如树叶抖动、水流、窗帘等），由于其使用方法相同故合并讨论。

在过去，人们已经提出了许多相关技术和模型来解决固定摄像头的前景提取问题。然而，在一些复杂场景中，这些方法一般都拥有一些局限性，如前景目标物体（交通灯下的车辆）可能断断续续的运动；他们可能不是摄像机视野的焦点；不感兴趣的背景物体可能有前景的行为（如：摇摆的树枝，喷泉等等）；背景中的阴影，光照变化；新物体的出现和消失都会对前景的获取带来很大的困难。简单的背景相减和传统的图像分割方法可能不适合实际视频监控任务，因此本文试图结合现有方法的一些尝试进行改进，提出一些适合于本问题及所给示例的新模型。

一般来说较实用的前景提取模型需要应用在实时系统上，如监控摄像头每获得一帧图像数据，就要实时的处理并得出前景提取结果，也就是说既没有视频内容的先验知识，也没有在第一帧开始前对背景建立任何的模型，只能基于历史数据进行建模。本文采用的模型也是从此类实时处理任务出发，然而在实际应用中此类模型不可避免地在处理视频的前几帧时会将待检测的前景误认为背景，一些不太鲁棒的模型还会导致对背景进行了错误的建模，使得处理结果存在一个伪前景。为解决这个问题，由于题目中并未规定要实时处理的要求，本文认为可以在正式处理之前，先对视频做一个整体的分析和对背景的初步建模，将其作为模型的初始化依据，这样将在线处理模型变为离线处理模型，获得更好的运行效果，同时保留模型处理在线视频的能力。

前景提取方法大致分为 3 个部分：

1. 首先，对场景的背景进行建模，以及分析视频帧进行模型的更新；
2. 其次，根据当前帧与背景模型的相似性，对所有的像素点都进行前景/背景分割标签的赋予；
3. 最后，对整个图像像素点的信息进行形态学变换，确保均匀的区域被赋予相同的标签并消除噪点。

本文在综合比对了若干现有方法的基础上，从 SuBSENSE[4]（Self-Balanced SENsitivity SEgmenter）算法当中吸收和学习了建模思路，采用 LBSP 算子作为像素点之间相似度的匹配模型，使用与 ViBe 算法类似的背景比对与更新模型，最后利用形态学变换的方法进行综合处理。如前文所述，本文还建立了一个离线分析全部视频帧来获得一个初始化背景的粗提取模型。

3.2 模型假设

1. 待处理的视频为连续、分辨率固定的灰度图像；
2. 摄像头面对背景的视角不会发生变化；
3. 视频中背景占到画面比例的大部分，前景只占小部分
4. 视频中不存在像素级别的前景；
5. 视频中不存在自始至终未移动过的前景；
6. 视频的前景与背景当中，相邻的像素点具有一定的相似特性。

3.3 符号说明

符号	意义
i_x	LBSP 中心像素的像素值
i_p	LBSP 周围像素 p 的像素值
$B(x)$	像素点 x 的背景样本模板集合
$I_t(x)$	第 t 帧图像上像素点 x 的 LBSP 模型序列
$S_t(x)$	第 t 帧图像上像素点 x 是否属于前景，值为 1 表示属于前景，值为 0 表示属于背景
R	相似度的距离阈值
T	时间采样因子
I_g^t	灰度视频的第 t 帧图像
F	视频的运动度量

3.4 模型建立与求解

模型一：相似性模型——LBSP 算子

像素之间的相似性模型是为了在不同帧之间比对像素之间的差异，为了更好的抵抗外部干扰，同时保持对前景提取的准确性，相似性模型最好包含像素点在时空邻域内的信息。经过分析与测试，本文选择了 LBSP[2]（Local Binary Similarity Pattern，局部二值相似模式）算子作为不同像素之间比较的相似性模型。

LBSP 算子是在 LBP（局部二值模式）算子的基础上发展而来的。LBP 算子是使用中心像素与邻近像素比较大小并进行编码，以此反应像素点周围的纹理特征。LBSP 算子将 LBP 算子的思路引入了前景提取中，它将 LBP 算子的计算模板扩展为 16 位，计算中心像素与周围像素的相似性, 若其差距超过一定的阈值则将此位置编码为 1，否则编码为 0。将所有位的编码组合成 16 位的二进制串，便得到了该像素点的 LBSP 编码，如图1所示。

每个模板位的编码规则为

$$LBSP(x) = \sum_{p=0}^{P-1} d(i_p, i_x) \cdot 2^p \tag{1}$$

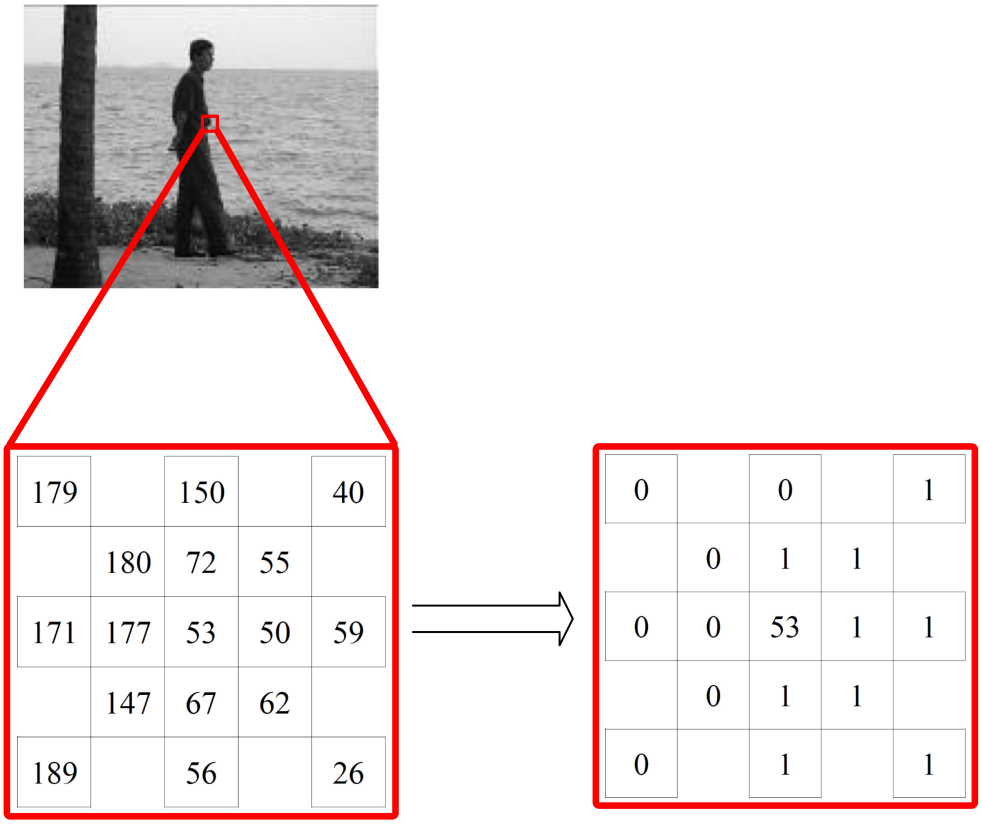


图 **1**　**LBSP** 算子示意图

$$d(i_p, i_x) = \begin{cases} 1 & \text{if } |i_p - i_x| \leq T_d \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

其中，T_d 为相似度阈值。

对于两个像素点之间的 16 位 LBSP 编码，它们的汉明距离（不同位的个数）归一化到 $[0, 1]$ 区间后，可作为彼此之间相似性的模型输出。相似度阈值 T_d 的大小，将显著影响到 LBSP 模型对像素变化的敏感程度，T_d 越大，像素值的变化越不容易体现在模板位的变化中，相似匹配的结果越保守，但也会对一些细微的变化失去反应。结合前景提取任务的具体特点，我们将原始的 LBSP 算子的固定阈值变成阈值与当前像素点的在前一帧图片中的像素值相关，如公式 (3)，这样一来将时间上的信息也传达给了相似度模型之中。同时，增强了该模型对阴影的抵抗能力。由于阴影较多的发生在像素点由亮到暗的变化中，因此阴影像素所保留的阈值相对较高，不容易使相似性模型的结果发生变化，也就不容易被判断为前景。相比之下由暗到亮的变化更有可能是因为前景的出现，而此时

相似性模板的阈值较低，相似性匹配的结果容易发生改变。

$$d(i_p, i_x) = \begin{cases} 1 & \text{if } |i_p - i_x| \leq T_r \cdot i_x \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

模型二：背景比对与更新模型——动态样本集合

有了相似性模型，下一步需要做的是依据视频流对背景进行建模。由于 LBSP 模型不易通过求取平均或是计算混合高斯分布等方式来进行建模，本文对背景建模采用类似 ViBe[3]（Visual Background Extractor）算法的形式，建立动态样本集合模型对背景进行建模。

该模型针对每一个像素点，均建立了一个大小固定的背景样本模板集合

$$B(x) = \{B_1(x), B_2(x), \ldots, B_N(x)\} \tag{4}$$

对于每一个新加入的像素点，利用前述的相似性模型，计算在背景集合中与新像素点的距离小于某一固定阈值的样本的个数，将其作为考量像素点是否属于前景的指标

$$S_t(x) = \begin{cases} 1 & \text{if } \#\{dist(I_t(x), B_n(x)) < R, \forall n\} < \#_{min} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

其中，R 是相似度的距离阈值，$\#_{min}$ 是背景判断所需要的最小匹配个数。

经过实验，将 $\#_{min}$ 固定在 2 [4] 可以同时保障对噪声的干扰能力和匹配的灵活性，而样本集合 N 一般来说至少要达到 20 个以上，这是由于匹配的编码多达 16 位所致。因此最终模型的性能由参数 R 直接决定。R 较小时，新像素点需要与样本模型十分相似才能被判断为前景，而 R 较大时，模型对噪声的抵抗能力便得到加强。

样本集合的构建采用的是随机更新策略。每当一个像素点被分类为背景后，该点对应的背景样本集合中某一个样本有 $\frac{1}{T}$ 的概率被该点的特征模板所替代，同时，该点的邻域内的某一随机选中的点同样有 $\frac{1}{T}$ 的概率取代样本集合中的一个随机样本。因此，这一动态变化的样本集合模型就能反映该像素点处的背景在时空上的特征。T 被称为时间采样因子，它反映了背景样本模型的更新速率，T 较大时样本的更新策略比较保守，T 较小时样本的更新则比较频繁，对新环境的适应性较好。

模型三：离线生成背景模型——一维 Gabor 滤波与直方图分析

在本问题中，由于摄像头的背景内容不变，在离线分析的情况下，可以先对视频流的全部帧做整体分析，初始化一个较为理想的背景模型，随后再开始前

景的分割提取流程。本文采用像素时间序列的一维 Gabor 滤波 [6] 结果的分布直方图做背景粗提取，作为每个像素的动态样本集的初始化结果。

Gabor 滤波是一种加 Gauss 窗的 Fourier 变换，广泛应用于信号分析领域。一维 Gabor 滤波的公式如下：

$$G(\omega,\alpha,t_0)=\int_{-\infty}^{\infty}e^{-j\omega t}\frac{1}{\sqrt{2\pi}\sigma}\exp\Big(-\frac{1}{2}\frac{(t'-t_0)^2}{\sigma^2}f(t')\mathrm{d}t'\Big) \tag{6}$$

一维 Gabor 滤波的大部分能量集中在以 t_0 为中心，间隔为 $[-\sigma,\sigma]$ 的区域内。一维 Gabor 滤波能很好地反应信号的时域局部特征。

本文将单个像素值的灰度值变化的序列视为离散时间信号，将一维 Gabor 滤波器应用于各像素灰度值的变化过程，获取视频在不同时间尺度下的运动特征，输入视频

$$V_g=\{I_g^t,I_g^{t+1},\ldots,I_g^T\} \tag{7}$$

其中，I_g^t 是视频的第 t 帧图像，T 是视频的帧数。

I_g^t 中坐标为 (x,y) 的像素的灰度值表示为 $I_g^t(x,y)$。令式 (6) 中 $f(t')=I_g^{t'}(x,y)$，且当对第 t 帧图像中的像素进行滤波时 $t_0=t$，这样可得

$$G_{xy}(\omega,\alpha,t)=\sum_{t'=1}^{T}e^{-j\omega t}\frac{1}{\sqrt{2\pi}\sigma}\exp\Big(-\frac{1}{2}\frac{(t'-t)^2}{\sigma^2}\Big)I_g^{t'}(x,y) \tag{8}$$

按上式运用 Gauss 时间窗口长度为 σ 的一维 Gabor 滤波器对灰度图像序列 Vg 进行滤波处理，得到滤波结果

$$F=\{F^1,F^2,\ldots,F^T\} \tag{9}$$

F 为视频的运动度量。其中，F^t 是一个 $W\times H$ 的矩阵，其元素 $F_{xy}^t=\|G_{xy}(\omega,\alpha,t)\|$，表示经 Gabor 滤波后，坐标为 (x,y) 的像素在 I_g^t 中灰度值的滤波值。其中滤波值较大的区域是相邻帧变化明显的区域。对每个像素的滤波结果统计分布直方图，直方图中的低值部分对应便是背景和静止的前景区域。设置适当的阈值，便可区分出像素点的哪些帧可作为背景样本。

以上分割只能作为粗提取的结果，并且包含噪声，然而用该结果初始化模型二中的动态样本集合，可以大大提高本文所述模型对短视频的处理效果，因此在静止摄像头情况下均使用本模型进行背景建模的初始化。

3.5 模型测试与评价

固定摄像头前景提取的整体模型计算流程如下：

步骤 1：对于一段固定长度的视频，分别获取每个像素点的时间序列 I_g^t，按照一维 Gabor 滤波器的公式进行滤波，得到每个像素时间序列的滤波结果 F^t；

步骤 2：计算每个像素时间序列滤波结果的直方图，将其中的低值部分对应的像素设为背景，并将其对应的 LBSP 模型序列加入该像素的初始化样本集 $B(x)$；

步骤 3：对于每一帧图片，计算每个像素点 x 的 LBSP 模型序列与其样本集中所有样本的汉明距离，将其映射到区间 $[0,1]$，并按式 (5) 输出分类结果；

步骤 4：对于每个被判定为背景的像素点，将其本身以及邻近像素点的 LBSP 序列以概率 $\frac{1}{T}$ 加入样本集中，并剔除样本集中等量的元素；

步骤 5：对该帧提取的二值图像做形态学变换：高斯模糊、膨胀、腐蚀，输出该帧的提取结果，若当前帧为最后一帧，流程结束，否则读入下一帧数据并转到步骤 3。

问题一结果

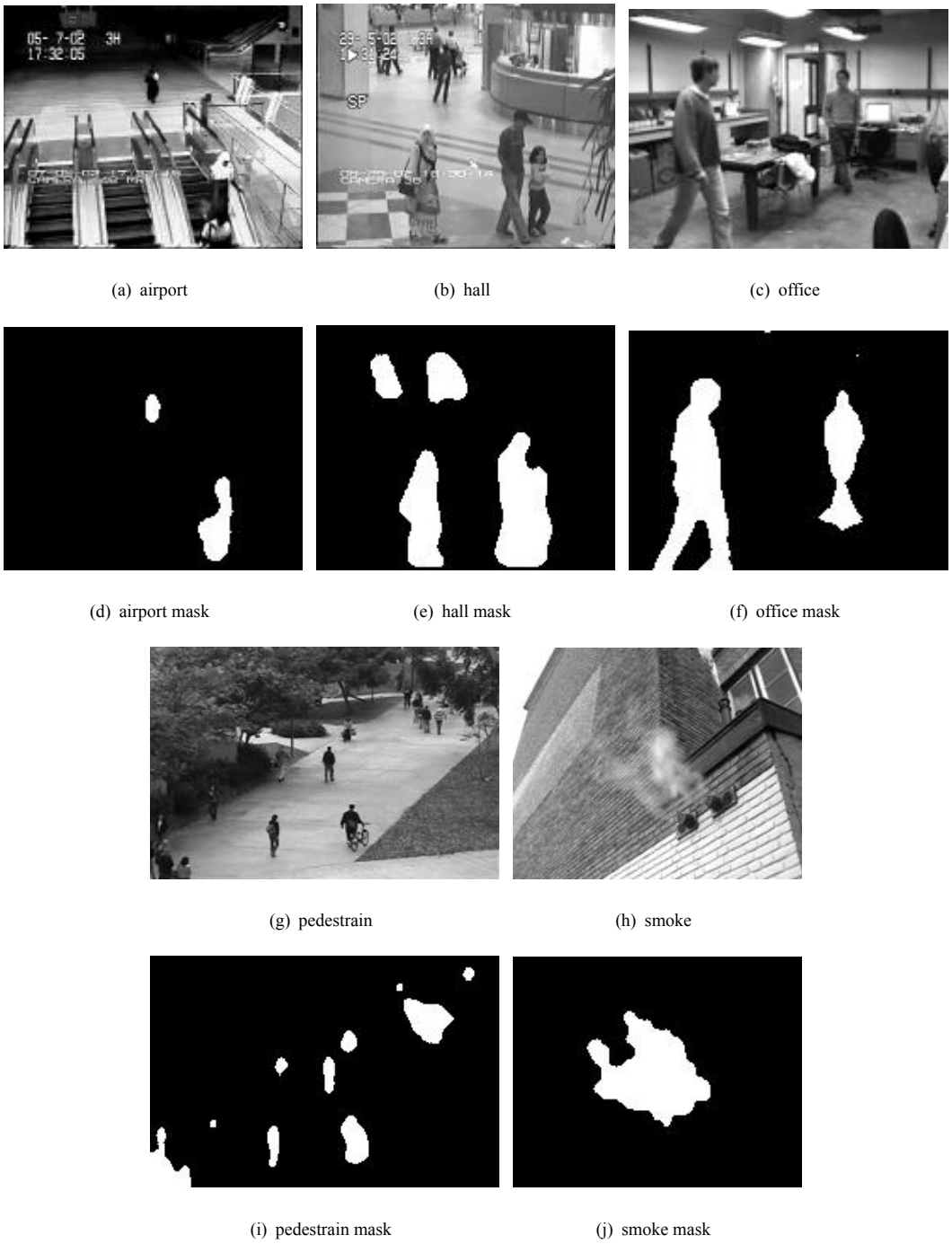
为检验 Gabor 滤波器与直方图模型得到的初始化效果，从每个像素的初始样本集中选取像素构成背景图片如图2所示。

(a) airport background　　(b) office background

图 **2**　**Gabor** 滤波得到的背景

由图中可得，Gabor 滤波能够很好地分离出视频中的背景画面，说明样本集的初始化成功。

由上述模型得到的 mask 视频附在附件中，在此列出部分帧的效果，如图3所示。



(a) airport (b) hall (c) office

(d) airport mask (e) hall mask (f) office mask

(g) pedestrain (h) smoke

(i) pedestrain mask (j) smoke mask

图 **3** 固定摄像头静止背景的前景提取结果

观察图像可得，模型对各种尺度、角度和环境下的前景均有很好的提取效果。对于烟囱的烟雾这种通常意义下不认为是前景的物体，在适当调整模型参数后也能够准确检测出。

问题二结果

问题二的结果 mask 视频附在附件中，在此给出部分帧的效果，如图4所示。

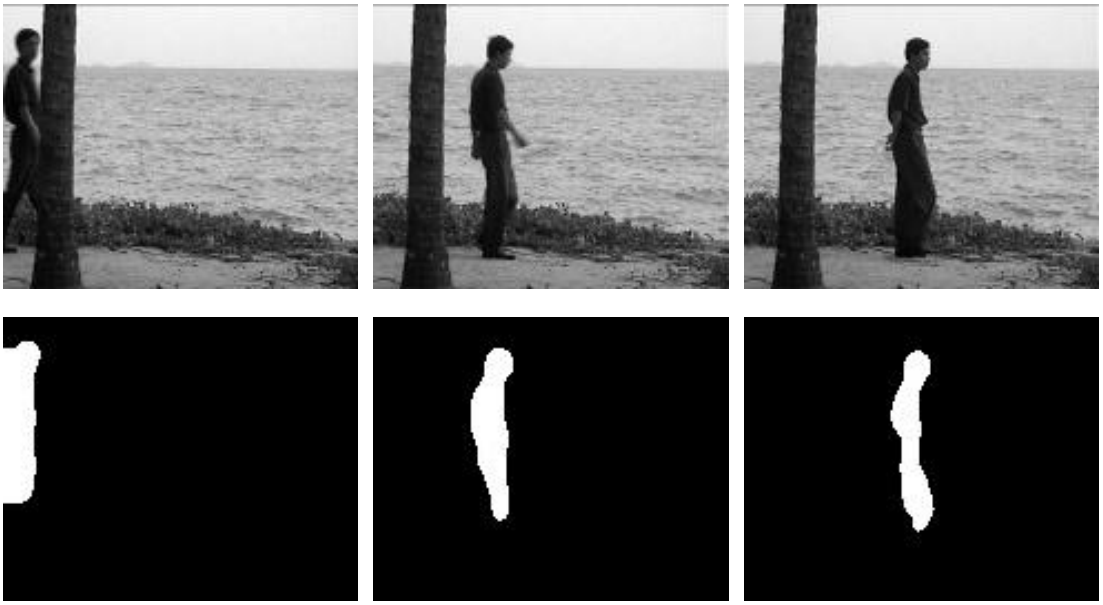


图 **4**　固定摄像头动态背景的前景提取结果

由图中可知，所建立的固定摄像头前景提取模型对于动态背景的过滤效果很好，得到的 mask 图像中没有出现伪前景。由于视频画面中人的小腿部分与背景像素点的灰度值比较接近，故没有被完全提取出来。

表 **1**　显著前景目标的视频帧结果

视频名称	包含显著前景目标的视频帧标号
Campus	85, 200-224，310-321，352-443，479-483，485-488，506，510-515，600，644-682，692-711，741-806，811-859，861-864，866-873，891，1006-1035，1193，1264，1330-1374，1377-1404，1408-1409
Curtain	411，967，1561，1762-1889，1895，1897，2126，2176-2311，2642，2770-2913，2916-2926
Escalator	1-90，92-100，127-128，254-1929，1931，1956-1958，1960-2081，2084-2110，2112-2388，2415，2539，2754，2774-3313，3323-3417
Fountain	141，157-209，259，335，408-523
Hall	1-70，79-141，153-458，469-476，478-481，483-484，493-494，578，616-621，625，635-723，725-733，795，819-847，856-876，878-973，977-1052，1138，1154-1208，1246，1280-3344，3406-3534
Lobby	79，155-195，259，346-392，426，450-451，521，623-668，870，963-1038，1093-1095，1161，1240-1282，1335-1538
Office	197，372，501，585-2037，2080
Overpass	374，968，1551，1881，2098，2336-2617

问题四结果

将所建立的固定摄像头前景提取模型应用于问题四的视频中，并计算前景在整幅图片中所占的比例，若比例超过一定的阈值，则认为视频帧中包含显著的前景目标。得到包含显著前景目标的视频帧标号，列表如表1所示。

4. 问题三

4.1 问题分析

在本问中，图像出现变化的原因不仅仅是前景的移动，还加入了摄像头本身的抖动，这一类视频往往具有以下特点：1）摄像机的抖动具有非预测性，2）背景与前景的绝对位置变化较大，3）背景与前景的相对位置无规律性。

这些特性导致了在固定摄像头下的模型往往变得不可行，或者效果很差，需

要重新挖掘新的模型思路，或者在已有模型的基础上做出调整与改进。同时，由于背景非固定，前述模型三所述的离线背景生成模型作为初始化的方法也不再可用，需要建立新的摄像头去抖动方法。

经过观察，虽然摄像头本身的移动规律难以通过分析视频进行估计，然而相邻几帧之间的图像变化不是特别剧烈，可以近似为透视变换，同时，相邻几帧之间的公共区域依然占画面的大部分，可以将公共区域作为静止摄像头捕捉到的图像进行前景提取。因此，解决本问题的思路是建立相近帧之间的投影映射关系，将原本发生运动的相邻帧透视变换到相同的平面内，再在叠合的公共区域内按照固定摄像头提取前景的模型进行前景提取。

在前述模型二当中，有两个重要的参数：最大样本距离阈值 R 和时间采样因子 T。在固定摄像头场合中，由于背景的区域不会发生移动，对像素点进行建模时的参数要求不是很严格。然而在抖动摄像头任务中，由于相邻帧之间的比对是通过透视变换之后做的，这种固定参数的模型对于干扰的抵抗和前景提取的精准性已经不再能满足任务要求，因此有必要设计模型的改进策略。

4.2 模型假设

1. 待处理的视频为连续、分辨率固定的灰度图像；
2. 视频中背景占到画面比例的大部分，前景只占小部分
3. 视频中不存在像素级别的前景；
4. 视频中不存在自始至终未移动过的前景；
5. 视频的前景与背景当中，相邻的像素点具有一定的相似特性。
6. 摄像头抖动情况下，临近几帧之间的图像变化可近似为是透视变换；
7. 摄像头抖动情况下，临近几帧之间的公共区域依然占到整个画面的大部分，太靠近边界部分的前景可忽略不计。

4.3 符号说明

符号	意义
H	单应矩阵
D_min、$v(x)$	反馈模型中间变量

4.4 模型建立与求解

模型四：单应矩阵映射模型

为了在两个不同帧之间找到对应关系，本文建立了两帧图像画面间的单应矩阵模型。

在计算机视觉中，平面的单应性被定义为一个平面到另一个平面的投影映

射。因此若两帧图像之间满足透视变换关系 [7]，便可以通过矩阵相乘的方式将公共区域内一帧图像上的点对应到另一帧的相应位置。

单应矩阵是大小为 3×3 的矩阵 $H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h32 & 1 \end{bmatrix}$，满足给定一个点 $p_1 = \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$（齐次坐标），H 把点 p_1 映射到另一个平面上的点 $p_2 = \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = Hp_1$。假设已知两个图像上的一对对应点的坐标 $[x_1, y_1]^T$，$[x_2, y_2]^T$，它们的齐次坐标为 $[x_1, y_1, 1]^T$，$[x_2, y_2, 1]^T$，代入上面的公式中，有

$$x_2 = \frac{x_1 h_{11} + y_1 h_{12} + h_{13}}{x_1 h_{31} + y_1 h_{32} + 1} \tag{10}$$

$$y_2 = \frac{x_1 h_{21} + y_1 h_{22} + h_{23}}{x_1 h_{31} + y_1 h_{32} + 1} \tag{11}$$

将以上两式重新组织，可得等价的矩阵形式：

$$Au = v \tag{12}$$

其中，

$$A = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1 x_2 & -x_2 y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1 y_2 & -y_1 y_2 \end{bmatrix}$$

$$u = \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{21} & h_{22} & h_{23} & h_{31} & h_{32} \end{bmatrix}^T$$

$$v = \begin{bmatrix} x_2 & y_2 \end{bmatrix}^T$$

由于 H 中有八个未知数，如果有四对不共线的匹配点，一对点可以提供两个方程，这个方程组就可以有八个方程，可以求出 H 的解。因此在求取两个画面之间的单应矩阵映射关系时需要四对已知的对应点坐标。

在寻找两帧画面中的对应点时，首先分别对两个画面进行特征提取得到一组关键点。要求提取的特征具有尺度、旋转、仿射不变性，经典的特征算子有 SIFT(Scale Invatiant Feature Transform)、SURF(Speeded Up Robust Feature) 等，综合考虑特征的检测效果和计算速度，本文选用 SURF 算子来进行特征提取。

在考虑两帧画面的变换关系时，由于前景是运动的，我们并不希望将前面帧的前景映射到当前帧的相同位置，期望获取的仅为背景的映射。因此，我们在获取两帧画面中的匹配点时，应忽略前景中的关键点，仅采用背景中的点来进行匹配并求解单应矩阵。具体来说，我们将前面帧得到的 mask 图像取反，作为关键点选择的 mask，过滤掉前景中的关键点。

然后利用 k-近邻算法找到在特征空间上最相近的 k 个点（本文中取 k=2）。因此，对于每个特征点，在另一幅画面中都有两个候选的匹配点。如果最优匹配点与待匹配点在特征空间上的距离很小，而次优匹配点的距离要大许多，那么我们接受该最优点，因为它是候选点中无歧义的最佳选择。相对的，如果两个候选匹配点非常相近，那么选择其中之一可能出错，因此这两个匹配点都会被拒绝。在模型中，计算最优距离和次优距离的比值，如果该值小于给定的阈值，则接受最优匹配点，否则，拒绝两个匹配点。这一步得到了两帧画面 A，B 中点的若干匹配。但是，即使画面 B 中的点 b 是画面 A 中的点 a 在 B 中所有关键点的最佳匹配点，它仍有可能是一个错误的匹配，即待匹配点 a 在 B 平面中的实际对应点 a' 不包含在 B 的关键点组中，可能的原因有：1）a' 不在画面 B 内；2）a' 在画面 B 内但未被特征提取算法检出。由于错误匹配的存在，采用所有匹配点进行最小二乘拟合得到单应矩阵可能具有较大误差。因此，我们采用如下方法计算单应矩阵：

1. 在筛选得到的若干匹配点中，随机选取 4 对不共线的点计算单应矩阵.
2. 将其他的匹配点代入得到的单应矩阵 H，计算估计的匹配点位置和实际匹配点位置间的距离，若距离满足 $\|p_2 - Hp_1\| \leq \sigma$，$\sigma$ 为设定的阈值，则认为该匹配为单应矩阵为 H_0 时的正确匹配，称为该估计的内点，反之为外点.
3. 不断迭代，一个好的单应矩阵估计应使得内点尽可能多，当估计达到设定的置信度时，停止迭代.

利用所得的单应矩阵建立具有重叠部分的两帧图像之间的透视变换关系，将前面帧映射到当前帧的图像平面。

模型五：反馈动态参数模型

在模型二的 ViBe 算法中，有两个重要的参数，最大样本距离阈值 R 和时间采样因子 T，它们是影响模型准确性与适应性的关键参数。在移动摄像头场景下，需要对这两个参数进行调整以达到最佳效果。本文建立了反馈动态参数模型 [4]，对于新到来的每一帧图片，首先求取像素点与各个样本值之间的最小归一化距离 $d_t(x)$，并对该值进行一个滑动平均

$$D_{min}(x) := D_{min}(x) \cdot (1 - \alpha) + d_t(x) \cdot \alpha \tag{13}$$

其中，α 是学习率。

由此计算得来的 $D_{min}(x)$ 是一个固定在 $[0, 1]$ 区间的值，当背景近似于静止图像时，$D_{min}(x) \approx 0$；而当像素值变化比较剧烈时，$D_{min}(x) \approx 1$。将该值作为更新模型二参数的第一个反馈值。

为了处理前景提取结果不断闪烁的情况，本模型使用一种类似异或的机制

反馈一个会随着结果的闪烁不断增大的量，如下所示

$$v(x) := \begin{cases} v(x) + v_{incr} & \text{if } X_t(x) = 1 \\ v(x) - v_{decr} & \text{otherwise} \end{cases} \tag{14}$$

其中，v_{incr} 和 v_{decr} 分别为 1 和 0.1，$v(x) \geq 0$。$X_t(x)$ 的值为当前像素在此帧中与前一帧中识别结果的异或，即识别结果不同时为 1，相同时为 0。

当前景提取结果较为稳定时，$v(x) \approx 0$；反之，$v(x)$ 会不断增大。以此作为模型的第二个反馈值。

最后，便可以依据下述公式实时更新模型二中的参数 R 和 T

$$R(x) := \begin{cases} R(x) + v(x) & \text{if } R(x) < (1 + D_{min}(x) \cdot 2)^2 \\ R(x) - \dfrac{1}{v(x)} & \text{otherwise} \end{cases} \tag{15}$$

$$R_{lbsp}(x) = 2^{R(x)} + R^0_{lbsp} \tag{16}$$

$$T(x) := \begin{cases} T(x) + \dfrac{1}{v(x) \cdot D_{min}(x)} & \text{if } S_t(x) = 1 \\ T(x) - \dfrac{v(x)}{D_{min}(x)} & \text{if } S_t(x) = 0 \end{cases} \tag{17}$$

其中，$R(x) \geq 1$。

当 $D_{min}(x)$ 较小时，即使 $v(x)$ 由于分类结果的抖动而变大，也并不会改变 $R(x)$ 的值，而当 $D_{min}(x)$ 接近 1 时，$R(x)$ 会随着分类结果的波动而变大，增大像素点被识别为背景的概率。R^0_{lbsp} 为设定好的初始距离阈值，之后每一帧的每个像素都将依据公式 (16) 计算实际的距离阈值 $R_{lbsp}(x)$。

$T(x)$ 的值在更新过程中存在上下界，本文中选定为 $[2, 256]$，其更新策略为当目前像素点被识别为前景时，$T(x)$ 增大，且 $v(x)$ 和 $D_{min}(x)$ 越小，说明模型越稳定，其增大的幅度就越大，动态样本集合也越不容易被改变。反之，当目前像素被识别为背景时，$T(x)$ 将适当减小，以增大样本集合被更新的可能性。

至此，通过式 (13)-(17)，我们建立了模型二中参数的动态反馈模型。

4.5 模型测试与评价

摄像头抖动情况下前景提取的整体模型计算流程如下：

步骤 1：对于一段固定长度的视频，将其第一帧图片的每个像素点及其邻域做随机采样，将采样结果对应的 LBSP 模型序列加入该像素的初始化样本集；

步骤 2：后来的每一帧图片，按照模型四所列方法求取前若干帧图片与当前帧的单应矩阵，前若干帧图片映射到当前帧所在的图像平面，并获得它们叠合后的公共区域；

步骤 3：对于公共区域的每个像素点，如果它在前面的处理过程中已经建立过样本集合，则直接按照静态摄像头的比对方法求取 LBSP 模型序列并按照式 (5) 输出分类结果；如果在前面的处理过程中并未建立样本集合（一般来说是与之前的所有帧都没有过重叠区域的边缘像素点），则按照步骤 1 所述的初始化方法生成样本集合；

步骤 4：对于每个公共区域内被判定为背景的像素点，将其本身以及邻近像素点的 LBSP 序列以概率 $\frac{1}{T}$ 加入样本集中，并剔除样本集中等量的元素；

步骤 5：更新反馈参数 D_{min} 和 v 并按照式 (15)-(17) 更新参数 R 和 T；

步骤 6：对该帧提取的二值图像做形态学变换：高斯模糊、膨胀、腐蚀，输出该帧的提取结果，其中非公共区域均视为背景；若当前帧为最后一帧，流程结束，否则读入下一帧数据并转到步骤 2。

两帧间关键点匹配及透视变换结果如图5所示。

图 **5**　关键点匹配及透视变换示意图

由上图可得，由于采用了前面帧的 mask 取反过滤前景上的关键点，最后生成的匹配点都在背景上，且 k-近邻算法得到的匹配效果非常好。观察图中前二帧变换后的图像和当前帧图像，背景中对应点的位置基本相同，而由图中红色圈出的部分可以明显观察到车的运动，表明了所求得的单应矩阵的有效性。

由所建立的摄像头抖动的前景提取模型得到的 mask 视频附在附件中，在此给出部分帧的效果，如图6所示。

由图中可得，前景提取的效果良好，采用我们建立的透视映射和反馈模型，由摄像头抖动产生的伪前景基本被消除。

(a) cars6 (b) cars7 (c) people1 (d) people2

(e) cars6 mask (f) cars7 mask (g) people1 mask (h) people2 mask

图 **6** 摄像头抖动的前景提取结果

5. 问题五

5.1 问题分析

在本问中，任务变成了从多个固定的近似拍摄同一地点监控视频中建立前景提取模型。固定摄像头的前景提取模型在问题一、二、四的分析和求解中已经取得了不错的效果，本问决定沿用当时的建模思路，结合多摄像头的特点和信息融合策略对已有模型作出改进。

经过观察，多个摄像头之间的位置和角度差异较大，各个摄像头画面间的映射关系不能用投影变换来近似，需要考虑摄像头在三维空间中的位置关系，因此无法直接沿用模型四的方法，必须设计在此类情况下求取摄像头之间画面映射关系的模型。

同样，由于计算映射关系存在误差，在本问中依然采用了依据提取结果反馈更新模型二中参数值的方法。

5.2 模型假设

假设 1、2、3、4、5、6 与问题一、二、四中相同，不再赘述。

7. 拍摄视频所用的多个摄像头具有相同的内参矩阵；
8. 在多摄像头拍摄情况下，不同摄像头对空间中同一位置拍摄到的像素具有一定的相似性。

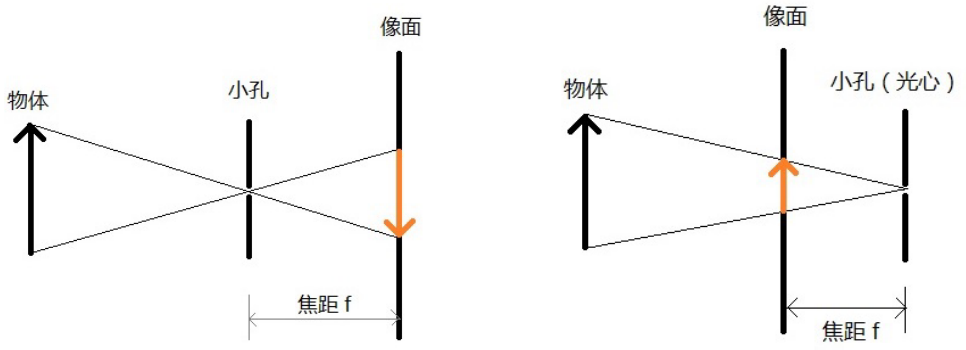
5.3 符号说明

符号	意义
P^W	P 在世界坐标系中的三维坐标
P^C	P 在相机坐标系中的三维坐标
P_i^C	P 在相机 i 坐标系中的三维坐标
P^I	P 在相机像平面的像点 p 在图像坐标系中的齐次坐标
P_i^I	P 在相机 i 像平面的像点 p_i 在其图像坐标系中的齐次坐标
K	相机内参矩阵
E	两个相机间的本征矩阵
$\mathcal{R}$	两个三维坐标系间变换的旋转矩阵
$\mathcal{T}$	两个三维坐标系间变换的平移向量

5.4 模型建立与求解

模型六：多摄像头三维模型

在本文中，我们将相机成像近似为小孔成像模型。如图7(a)所示，小孔成像在像平面上成倒立实像。为了便于表述与分析，将像平面置于小孔之前，且到小孔的距离仍为相机焦距 f，成正像，如图7(b)所示，得到的模型与原模型等价。



(a) 小孔成像模型　　(b) 等价模型

图 7　相机成像示意图

首先考虑单个摄像头的情况，设三维空间中有一点 P，取世界坐标系与相机（光心）坐标系重合，点 P 在像平面上的像为 p，如图8所示。

设点 P 在世界坐标系中的三维坐标为 $P^W = [X, Y, Z]^T$，p 以像平面中心为

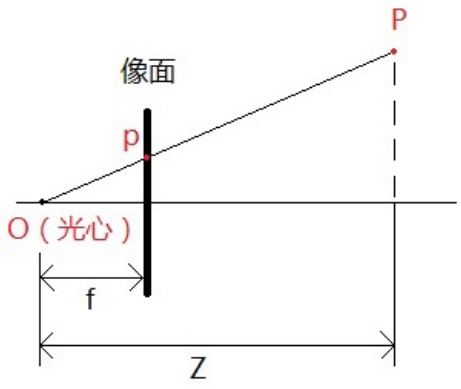


图 **8**　计算内参矩阵

原点的二维坐标为 $p^C = [x_0, y_0]^T$。由于世界坐标系与相机坐标系重合，则 Z 即为点 P 到相机光心的垂直距离。由上图中的相似三角形关系可得

$$\begin{cases} x_0 = \dfrac{fX}{Z} \\ y_0 = \dfrac{fY}{Z} \end{cases} \tag{18}$$

我们描述二维图像上的点时，通常采用图像的左上角作为坐标原点，称坐标系为图像坐标系。设点 p 在图像坐标系中的坐标为 $p^I = [x, y]^T$，图像中心点坐标为 $[c_x, c_y]^T$，则

$$\begin{cases} x = \dfrac{fX}{Z} + c_x \\ y = \dfrac{fY}{Z} + c_y \end{cases} \tag{19}$$

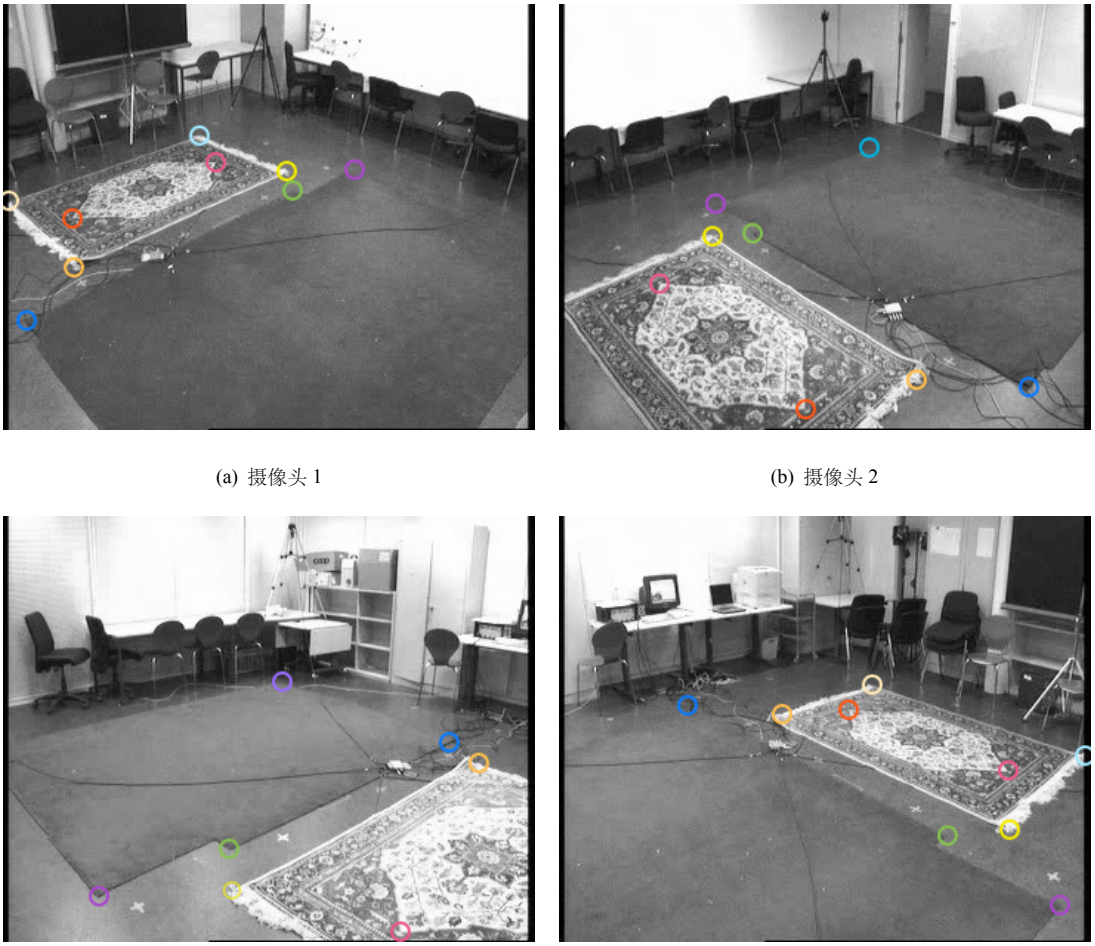
将上式表示成矩阵形式

$$Z \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{20}$$

其中，矩阵 $K = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}$ 称为相机的内参矩阵，因为它只与相机的内部参数有关。

内参矩阵 K 通常采用对摄像头进行棋盘格标定的方法 [7] 进行计算，即用相机拍摄多个角度的棋盘格图像，并对其中的角点进行标定，以相机的计算内参矩阵。在本问题中，对于每一个摄像头，由于我们仅有该摄像头在同一角度拍摄的视频画面，不足以用于计算。因此我们假设摄像头组中的每个摄像头具有

相同的内参矩阵，以得到多个角度的图像。另外，画面中还需要有用于标定的平面，这里我们采用视频画面中地面和墙壁等平面上的显著点进行标定。如图9所示。



(a) 摄像头 1　　(b) 摄像头 2

(c) 摄像头 3　　(d) 摄像头 4

图 **9**　多摄像头标定

而在一般情况下，世界坐标系和相机坐标系不重合，在将世界坐标系中的某一点 P 投影到像平面上时，要先将该点的坐标转换到相机坐标系下。设 P 在世界坐标系中的坐标为 P^W，P 到光心的垂直距离为 s，P 在像平面上的像点 p 在图像坐标系中坐标对应的齐次坐标为 P^I，世界坐标系与相机坐标系之间的相对旋转矩阵和位移向量分别为 $\mathcal{R}, \mathcal{T}$，则 $P^C = \mathcal{R}P^W + \mathcal{T}$ 即为 P 在相机坐标系下的坐标，故有

$$sP^I = K[\mathcal{R}P^W + \mathcal{T}] = K\begin{bmatrix}\mathcal{R} & \mathcal{T}\end{bmatrix}\begin{bmatrix}P^W \\ 1\end{bmatrix} \tag{21}$$

其中，$\begin{bmatrix}\mathcal{R} & \mathcal{T}\end{bmatrix}$ 是一个 3×4 的矩阵，称为外参矩阵，因为它只与相机在世

界坐标系中的位置有关，与相机参数无关。

接着考虑三维空间中的点 P 在两个相机中的像的关系。如图10所示，P 在相机 a 的像平面所成的像为 p_a, 在相机 b 的像平面所成的像为 p_b。

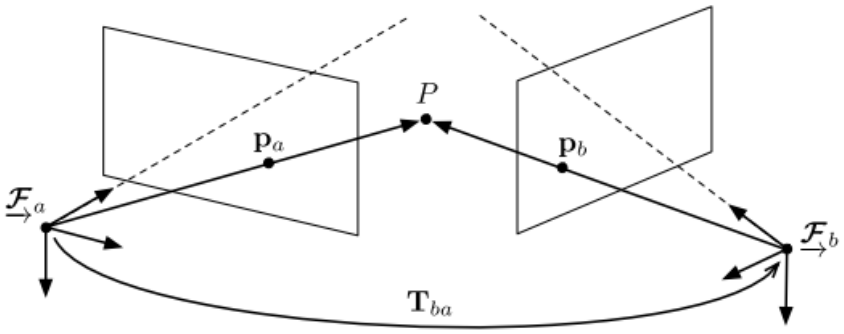


图 **10**　两个摄像头示意图

设 P 到两个相机像面的垂直距离分别为 s_a 和 s_b，且这两个相机具有相同的内参矩阵 K，与世界坐标系之间的变换关系分别为 $\begin{bmatrix}\mathcal{R}_a & \mathcal{T}_a\end{bmatrix}$ 和 $\begin{bmatrix}\mathcal{R}_b & \mathcal{T}_b\end{bmatrix}$，可得

$$s_a P_a^I = K(\mathcal{R}_a P^W + \mathcal{T}_a) \tag{22}$$

$$s_b P_b^I = K(\mathcal{R}_b P^W + \mathcal{T}_b) \tag{23}$$

内参矩阵 K 可逆，将上式左乘 K 的逆，有

$$s_a K^{-1} P_a^I = \mathcal{R}_a P^W + \mathcal{T}_a \tag{24}$$

$$s_b K^{-1} P_b^I = \mathcal{R}_b P^W + \mathcal{T}_b \tag{25}$$

设 $K^{-1}P_a^I = (P_a^I)'$，$K^{-1}P_b^I = (P_b^I)'$，则

$$s_a (P_a^I)' = \mathcal{R}_a P^W + \mathcal{T}_a \tag{26}$$

$$s_b (P_b^I)' = \mathcal{R}_b P^W + \mathcal{T}_b \tag{27}$$

由于世界坐标系可以任意选择，我们将世界坐标系选为第一个相机的相机坐标系，这时 $\mathcal{R}_a = I, \mathcal{T}_1 = 0$。上式变为

$$s_a (P_a^I)' = P^W \tag{28}$$

$$s_b (P_b^I)' = \mathcal{R}_b P^W + \mathcal{T}_b \tag{29}$$

把式 (28) 代入 (29)，可得

$$s_b(P_b^I)' = s_a\mathcal{R}_b(P_a^I)' + \mathcal{T}_b \tag{30}$$

将式 (30) 两边对 $\mathcal{T}_b$ 和 $(P_b^I)'$ 的叉积 $\mathcal{T}_b \times (P_b^I)'$ 做内积，有

$$0 = s_a(\mathcal{T}_b \times (P_b^I)')^T\mathcal{R}_b(P_a^I)' \tag{31}$$

即

$$(P_b^I)'^T\widehat{\mathcal{R}_b}^T\mathcal{R}_b(P_a^I)' = 0 \tag{32}$$

令 $E = \widehat{\mathcal{R}_b}^T\mathcal{R}_b$，则有

$$(P_b^I)'^T E(P_a^I)' = 0 \tag{33}$$

其中，E 称为关于相机 a，b 的本征矩阵。

为了求解本征矩阵，需要获取若干组两个摄像头画面中的相应点的匹配，且不能仅采用三维空间中同一平面的点。同时考虑到两个摄像头的视角可能存在较大的差距，在背景上利用 SURF 特征进行匹配的结果不太理想，并且在画面背景中难以找到足够的匹配点。为了获取足够的正确匹配点，在同步的画面中出现前景时，利用前景提取算法得到的 mask 提取出前景，然后采用 SURF 特征得到前景上的显著点（如，前景为人时，人的头顶、脚等）并进行匹配，如图11所示。由于前景在不断地运动，在不同的时刻，我们可以取得不同的匹配点，故可以得到足够多的匹配点。

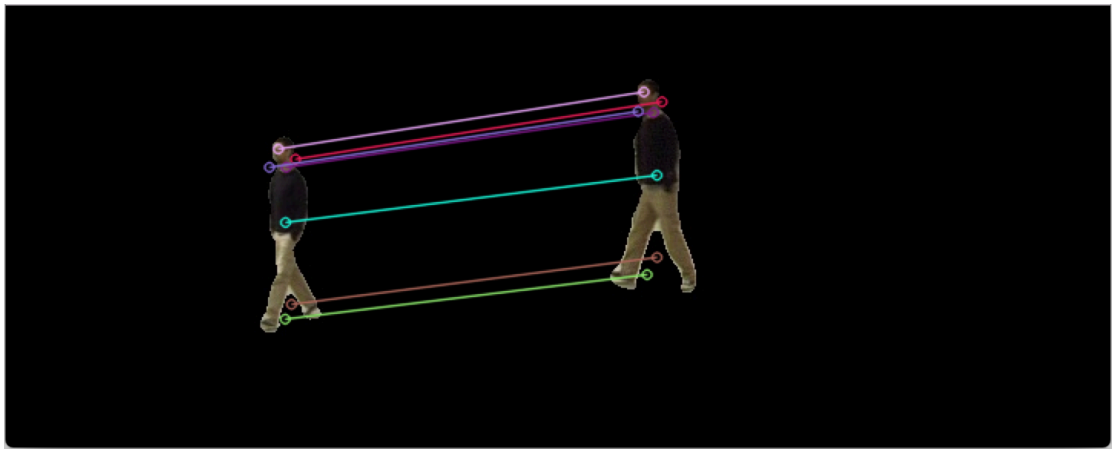


图 **11**　前景关键点匹配

综合上述分析可得，得到了摄像头的内参矩阵 K 后，可以根据两个图像上对应点的像素坐标求得若干组 $(P_b^I)', (P_a^I)'$，进而求解出本征矩阵 E。得到本征矩阵 E 即得到了空间中同一点在两个相机的像的映射关系。多摄像头的情况可以分解成若干组两个摄像头类似地进行处理。

模型七：多摄像头下的背景比对与更新模型

在多摄像头下，背景比对与更新模型与单摄像头下的模型大致相同，唯一的区别是利用多摄像头之间的信息融合丰富动态样本集合的表达能力。

具体来说，在建立了摄像头之间像素点的映射关系后，对于每一个被判定为背景点的像素点，在前述的随机更新策略中，除了该像素点本身与周围邻域有一定的概率被更新至动态样本集合之外，在其他摄像头中对应位置的像素点及其邻域也有一定的概率被更新至该动态样本集合当中。假设共有 N 个摄像头，在对其中一个摄像头的某个背景像素点 x 做样本库更新的时候，除了其自身以及邻域像素的 LBSP 模板序列有 $\frac{1}{T}$ 的概率被更新至样本库之外，假设该像素点出现在其他 n 个摄像头的范围内（n<N），那么该像素点对应在这些摄像头中对应位置的像素以及周围邻域内的像素点的 LBSP 模板序列也各自有 $\frac{1}{nT}$ 的概率被更新至原像素点的样本集合中，这样一来，就可以利用多摄像头环境下对公共区域的信息融合，来增加算法的准确性和鲁棒性。

5.5 模型测试与评价

在多摄像头前景提取中，整体模型计算流程如下：

1. 通过在多个摄像头画面中标定空间中处在一个平面上的点在摄像头图像平面中的对应位置，得到摄像头的内参矩阵 K；
2. 同一时刻在不同摄像头中匹配前景中的关键点，得到几组匹配点，选取不同时刻的匹配，得到若干组不同摄像头画面中点的匹配；
3. 根据内参矩阵 K 和匹配点坐标求解具有公共拍摄区域的两个摄像头的本征矩阵 E；
4. 利用内参矩阵 K 和本征矩阵 E，求解一个摄像头拍摄图像中的点在另一个摄像头拍摄图像中的对应位置，在原模型的基础上利用多个摄像头的信息来更新的背景样本库，得到融合多摄像头信息的前景提取结果。

由上述模型得到的 mask 视频附在附件中，在此列出部分帧的效果，如图12所示。

(a) 摄像头 1 画面　(b) 摄像头 2 画面　(c) 摄像头 3 画面

(d) 摄像头 1 mask　(e) 摄像头 2 mask　(f) 摄像头 3 mask

图 **12**　多摄像头前景提取

由图中可得，融合了多摄像头信息后提取前景的效果很好。

6. 问题六

6.1 问题分析

在本问中，问题变成了利用所获取的前景信息，判断监控视频中的一些群体的集体变化和异常事件等。这已经不属于前景提取问题的范畴，而是一个行为检测的问题，需要构建新的模型在前述模型结果的基础上做进一步求解。

群体行为检测技术一般来说是可以看做是将模式识别方法应用于计算机视觉领域的一个典型案例。群体行为检测技术的基本流程如下所示：

1. 预处理：对摄像头拍到的每一帧图像进行相关算法的预处理，本文中预处理即为前述的前景提取过程；
2. 特征提取：图像的特征可以将图像中某一类属性经过计算体现出来，一般将一幅图像提取出的特征保存为一个向量；
3. 分类器设计：采用离线学习或在线生成等方式训练用于将特征分类的判别函数；
4. 分类决策：对于样本的特征向量按判别函数的计算结果进行分类。

由于本文立足于数学建模的思想和方法，同时限于准备时间和篇幅以及无法完

成大量样本数据的训练，因此在本问中将重点放在了针对群体行为识别的特点建立运动特征模型，并通过一些简单的样例证实该模型对于群体行为具有一定的区分特性。对于判别函数的设计和测试不进行模型的求解。

群体行为检测的常用思路是对视频前景中物体的静态及动态特性进行估计，其中静态特性包括群体中的个体数量、整体密度、局部密度等。动态特性包括群体的平均运动方向、群体平均动能、群体距离势能等。

本题中群体行为识别是在前景提取后的二值图上进行的，一个最简单的思路便是直接按照群体的每一个个体统计静态和动态特性，但由于群体数量众多，并且前景二值图中会经常出现粘连、重叠等情况，难以从图像中分辨出每个个体的位置，除非为每一个捕捉到的个体建立目标跟踪模型，但这样的模型计算量过于庞大，实用性并不好。另一种思路是统计前景中所有像素点的静态和动态特性，这在一定程度上可以反应群体的一些集体行为，但由于画面中近处的前景物体像素点较多，远处则较少，采用此种方法分析的结果极易被一些近处个体的行为所影响。

综合以上考虑，本文采用在前景图中进行角点检测 [11] 的方式。角点是指像素的局部窗口延各方向移动时，均产生明显变化的点，在二值图中可简单认为是边缘的局部曲率突变的点。假设每个个体以及个体之间的边缘纹理特性不会发生较大的变化，那么角点检测的结果的统计特性就可以用来表示该个体的运动特性。当个体的数量足够多的时候，统计图中全部角点并分析其静态与动态特性就可以作为分析群体行为的依据。同时，由于个体的数量足够多，在局部即使出现角点的误判也不会对分析结果产生影响。

6.2 模型假设

1. 在进行监控视频异常事件判断时，不考虑前景提取的错误（即认为所提取的前景大致形状上是准确的，并且不存在对背景干扰的误判）；
2. 摄像头画面涵盖一个较广阔的区域，前景中包含一定数量的活动物体；
3. 画面中的前景具有群体特性，即画面中运动物体的大小和质量相近。

6.3 符号说明

符号	意义
I_x, I_y	图像 I 在 X、Y 方向的梯度
M	harris 椭圆函数矩阵
N	一帧图片中的角点总数
N_i	角点 i 邻域内其它角点的个数

6.4 模型建立与求解

模型八：角点检测模型

在一个二值图中检测角点的算法很多，经过前面的分析，本问题对于角点检测精度的要求并不高，因此本文简单选择了 Harris 角点检测算法做角点检测。

图像中的角点定义为，在像素点附近选取一个小窗口，如果在各个方向上移动这个窗口，窗口内区域点的像素值都会发生较大的变化，那么就认为在窗口内遇到了角点。如图13所示。

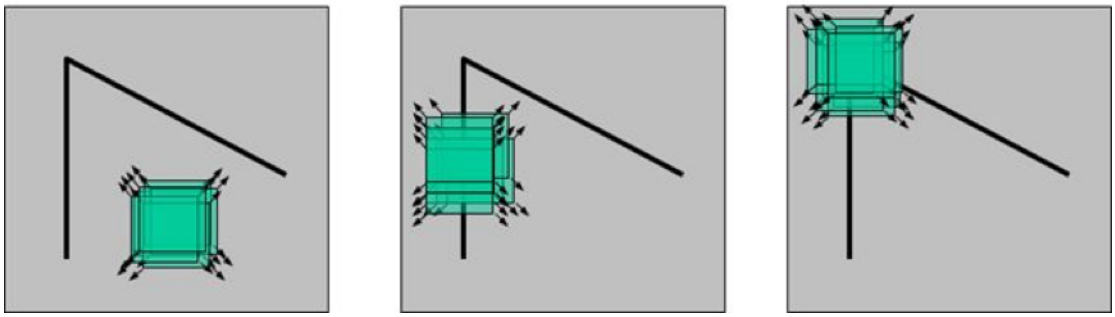


图 **13** 角点示意图

Harris 角点检测 [8] 的流程为

1. 计算图像 $I(x,y)$ 在 X 和 Y 两个方向的梯度 I_x、I_y.

$$I_x = \frac{\partial I}{\partial x} = I \otimes \begin{pmatrix} -1 & 0 & 1 \end{pmatrix}, \quad I_y = \frac{\partial I}{\partial y} = I \otimes \begin{pmatrix} -1 & 0 & 1 \end{pmatrix}^T$$

2. 计算图像两个方向梯度的乘积.

$$I_x^2 = I_x \cdot I_x, \quad I_y^2 = I_y \cdot I_y, \quad I_{xy} = I_x \cdot I_y$$

3. 使用高斯函数对 I_x^2、I_y^2 和 I_{xy} 进行高斯加权（取 $\sigma = 1$），生成矩阵 $M = \begin{bmatrix} A & C \\ C & B \end{bmatrix}$ 的元素 A、B 和 C.

$$A = g(I_x^2) = I_x^2 \otimes \omega, \quad C = g(I_y^2) = I_y^2 \otimes \omega, \quad B = g(I_{xy}) = I_{xy} \otimes \omega$$

4. 计算每个像素的 Harris 响应值 R，并对小于某一阈值 t 的 R 置为零.

$$R = \{R \mid \det M - \alpha(\mathrm{tr}(M))^2 < t\}$$

5. 在 3×3 或 5×5 的邻域内进行非最大值抑制，得到的局部最大值点即为图像中的角点.

在计算出图像中全部角点之后，由于 Harris 算法对图像中的像素噪声较为敏感，为了后续计算的准确性，要通过计算每个角点邻域内点的个数来消除孤立的角点。

至此，我们建立了前景图像中的角点检测模型。

模型九：群体运动特征模型

检测到角点之后，便可以统计群体的运动特征了。本文从群体运动的静态和动态两个角度出发，建立了群体运动特征模型。

1. 静态特性

静态特性主要包括群体数量、分布以及密度等等。依据前面的假设，可认为角点的数量与群体的数量成正比，角点的分布可以代表群体的分布。由此，群体的静态特性可以通过统计角点的静态特性来得到，也就是分别用角点的数量、其二维坐标的均值和协方差分别代表群体的数量、分布和密度。当群体的静态特性发生较大变化时，如数量急剧变化，中心位置快速向某一固定方向移动，或者以固定的周期做往复移动时，可作为估计群体运动特征的依据。

2. 动态特性

动态特性是通过分析和统计每个个体的运动特性得到的。传统的方法通过计算光流，或是进行目标跟踪的方式获得个体的运动方向和速度，本模型中可用每个角点与之前若干帧图像中角点的最短距离来代替，虽然一般来说该距离并不能真实的表征个体的运动特性，但其统计特性依然可作为群体的动态特性。

(1) 群体平均动能：动能表示物体因运动而产生的能量，群体平均动能即表示群体运动时的平均能量，群体平均动能表征群体运动的剧烈程度。依据假设，可以忽略群体中的质量差异，本模型中群体平均动能可用下式计算

$$E_{avg} = \frac{\sum_{i=1}^{N} |v_i|^2}{N} \tag{34}$$

其中，v 为前文所述相邻帧之间角点的最短距离，N 为一帧图片中的角点总数。

当群体的行为发生变化时，群体平均动能也会产生突变，可作为判断异常状况的依据。

(2) 群体距离势能：群体中各个个体之间的位置分布是判断是否有异常行为的重要判据，距离势能是指个体之间的相对距离，可以来描述个体之间的相对位置，实现对群体中个体之间总体远近程度的概括描述。如果群体个体之间的相对平均距离较小，说明人与人较为聚集；若群体个体之间的相对距离较大，则说明群体中个体的分布情况较为分散。本文利用角点与角点之间距离的平均值来表征个体之间的相对距离，通过计算每个角点与邻域内角点间的欧式距离平均值，再对图像中的所有角点求取该平均距离的均值，来表达群体的距离势能，

如下式所示：

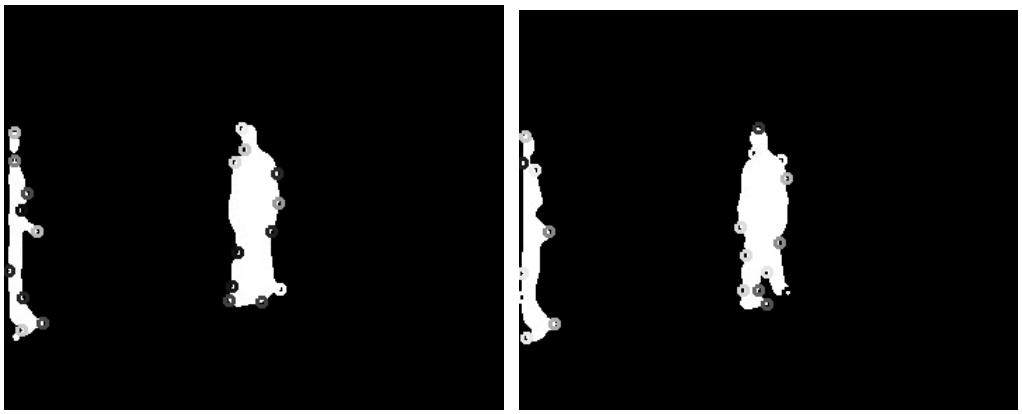
$$D = \varphi \cdot \frac{1}{N} \sum_{i=1}^{N} (\frac{1}{N_i} \sum_{j=1}^{N_i} C_{ij}) \tag{35}$$

其中，C_{ij} 表示两角点之间的欧氏距离，φ 是修正因子，取常量，本文设定 $\varphi = 1$，N_i 为邻域内的角点总数。

至此，我们通过计算总共得到了五个表征群体运动特性的特征值，建立了群体运动特征模型。

6.5 模型验证

为了验证群体动态特征模型的准确性，本文分别选择了一段消防演习中的短视频和一段人群正常行走的视频，按照前述方法提取每一帧的角点信息，如图14、15所示。并通过相邻帧之间的比对计算静态和动态特性。

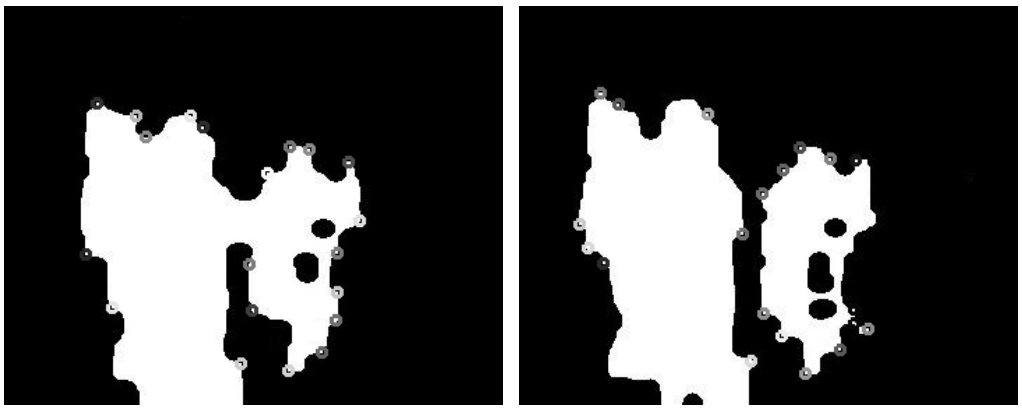


(a) 前一帧图像　　(b) 当前帧图像

图 **14**　普通场景下 **mask** 角点检测图

由上图可知，角点提取结果基本正确。根据模型九所述计算方法，可得该情景下的运动特征为

- 角点个数：20
- x 均值：110.6，y 均值：162.5
- x 方差：6743.17，y 方差：2100.85
- 群体平均动能：67.95
- 群体距离势能：14.1018



(a) 前一帧图像　　(b) 当前帧图像

图 **15**　消防演习场景的 **mask** 角点检测图

该情景下的运动特征为

- 角点个数：20
- x 均值：165.9，y 均值：168.5
- x 方差：4037.91，y 方差：4235.75
- 群体平均动能：152.6
- 群体距离势能：9.76684

经过比对发现，消防演习场景中的静态方差与动态特性中的动能和势能与普通场景相比有明显的差异。在数据量足够庞大的情况下，静态特性的统计分布也能体现以上两种场景的差异。由此证明了群体运动特征模型的确对群体的活动和异常行为具有区分能力。

7. 模型分析与总结

7.1 模型优点

本文充分考虑了问题在各个场合下的现实条件，针对不同的情况进行了讨论，对解决问题过程中可能遇到的困难做了综合分析，并利用题目所给的数据进行了合理的假设，结合一些现有的技术和算法，完成了题目要求中全部的建模和求解任务。

在问题一、二和四中，采用专门用于空间相似度匹配 LBSP 算子作为像素之间的比对模型，并且在原有的固定阈值基础上加入了与像素值相关的动态阈值，既引入了时间上的特性，又有对阴影的消除效果；采用 ViBe 算法对背景进行建模和比对，该模型被认为是比传统的混合高斯模型效果更优秀的背景比对模型，尤其是适合处理问题二中的动态背景前景提取任务，其测试结果也符合我们的

预期。此外，本文还采用了在图像处理领域较为少见的一维 Gabor 变换，对每个像素点在整个视频中的时间序列进行滤波，将其结果分布的直方图作为粗提取结果，该结果虽然并不十分准确，但本文将其与 ViBe 算法内部的动态样本集合进行结合，使得模型的初始化流程十分顺利，解决了许多前景提取模型在前几帧会出现错误甚至错误一直持续无法修正的情况。这样的处理思路借助离线信息优化了原本在线的提取模型，同时如果去掉该流程那么原模型就还是一个可以在线处理的模型，可供后续的问题使用。

在问题三中，本文从常规思路出发，寻找摄像头抖动的监控视频中临近几帧的变换关系，并在公共区域中搜索前景点。但本文同时也在原有的提取算法上做了优化，结合摄像头抖动条件下误差和干扰大的特点，对 ViBe 算法中的两个参数进行了动态的修正，使得原有算法的鲁棒性进一步提升，离线处理的能力增强。检测结果证明，本文的这套处理摄像头抖动前景提取问题的模型效果十分出色，符合我们对模型的预期。

在问题五中，本文在建立多摄像机之间的 3D 模型的同时，又一次将其与 ViBe 算法内部的样本集合结合，采用多摄像头之间的信息融合策略更新背景模型的样本集合，丰富了背景建模的信息来源，同时也利用上了多摄像头下处理前景提取任务的优势，所得的结果良好，符合模型的预期。

在问题六中，本文重点解决在前景图的基础上分析群体行为的问题。在从常规思路出发，分析了几种较为简单粗糙的方案之后，本文最终建立了采用角点之间的统计特性来估计前景群体的静态和动态特性的模型。该模型相比于光流法和群体目标跟踪法，具有简洁直观，计算量小等优点。同时在画面内个体足够多的假设前提下依然可以保证检测的正确率。

7.2 模型的不足与改进方向

首先，本文模型的识别结果与目前计算机视觉领域顶尖的卷积神经网络模型依然存在一定差距，这是因为我们所用数据来源少，图像的分辨率低且没有色彩信息，处理的视频较短，模型建立时间有限，计算资源不足等等因素所致。倘若在这些方面有所改观的话，本模型的处理效果还会得到进一步的提升。

其次，本文所述的模型是从现有的一些前景提取的算法上发展过来的，虽然本文尽力试图解决前景提取中的问题，但目前学术上采用传统方法进行前景提取依然面临着初始化出错，短视频处理效果不够好，在复杂背景下容易出现误判，在抖动摄像头环境下的提取不够理想等一系列问题，这些问题在本文所述模型提取结果的某些帧中有不同程度的体现。

最后，对于问题六中的群体行为判断和异常检测，限于篇幅和建模的时间限制，本文没能寻找到足够的样本数据，训练出完整的分类器，而是停留在了特

征提取这一步，仅对特征描述群体行为变化的能力进行了测试。

8. 参考文献

[1] 姜启源，谢金星，叶俊. 数学建模 [M]. 北京：高等教育出版社, 2011.1.

[2] St-Charles P L, Bilodeau G A. Improving background subtraction using Local Binary Similarity Patterns[C]// Applications of Computer Vision. IEEE, 2014:509-515.

[3] Barnich O, Van D M. ViBe: a universal background subtraction algorithm for video sequences.[J]. IEEE Transactions on Image Processing A Publication of the IEEE Signal Processing Society, 2011, 20(6):1709.

[4] St-Charles P L, Bilodeau G A, Bergevin R. SuBSENSE: a universal change detection method with local adaptive sensitivity[J]. IEEE Transactions on Image Processing A Publication of the IEEE Signal Processing Society, 2015, 24(1):359.

[5] 朱力旻. 视频中前景提取技术的研究 [D]. 上海交通大学, 2014.

[6] 闵华清, 陈聪, 罗荣华, 等. 基于时空分析的视频前景提取 [J]. 模式识别与人工智能, 2011, 24(04):582-590.

[7] RobertLaganiere, 拉加尼耶, 张静. OpenCV 2 计算机视觉编程手册 [M]. 科学出版社, 2013.

[8] Derpanis K G. The Harris Corner Detector[J]. Symposium Svenska Sllskapet Fr Bildanalys, 2004.

[9] 王晶, 许志杰. 基于时空纹理的实时群体行为检测 [J]. 西安邮电大学学报, 2015, 20(02):64-76.

[10] 陈磊, 吴悦, 岳晓冬. 基于小波变换的群体异常行为检测 [J]. 小型微型计算机系统, 2016, 37(8):1837-1842.

[11] 基于视频监控的群体异常行为检测研究 [D]. 西华大学, 2016.

附录 A　　OpenCV 代码

1.1　Gabor 变换代码

```
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <cmath>
#include <iostream>

using namespace cv;
using namespace std;

const double PI = 3.14159265;

Mat getMyGabor(int width, int height, int U, int V, double Kmax, double f,
   double sigma, int ktype, const string& kernel_name)
{
   int half_width = width / 2;
   int half_height = height / 2;
   double Qu = PI*U/8;
   double sqsigma = sigma*sigma;
   double Kv = Kmax/pow(f,V);
   double postmean = exp(-sqsigma/2);

   Mat kernel_re(width, height, ktype);
   Mat kernel_im(width, height, ktype);
   Mat kernel_mag(width, height, ktype);

   double tmp1, tmp2, tmp3;
   for(int j = -half_height; j <= half_height; j++){
      for(int i = -half_width; i <= half_width; i++){
         tmp1 = exp(-(Kv*Kv*(j*j+i*i))/(2*sqsigma));
         tmp2 = cos(Kv*cos(Qu)*i + Kv*sin(Qu)*j) - postmean;
         tmp3 = sin(Kv*cos(Qu)*i + Kv*sin(Qu)*j);

         if(ktype == CV_32F)
            kernel_re.at<float>(j+half_height, i+half_width) =
               (float)(Kv*Kv*tmp1*tmp2/sqsigma);
         else
            kernel_re.at<double>(j+half_height, i+half_width) =
               (double)(Kv*Kv*tmp1*tmp2/sqsigma);

         if(ktype == CV_32F)
            kernel_im.at<float>(j+half_height, i+half_width) =
               (float)(Kv*Kv*tmp1*tmp3/sqsigma);
```

```
            else
               kernel_im.at<double>(j+half_height, i+half_width) =
                  (double)(Kv*Kv*tmp1*tmp3/sqsigma);
         }
      }

      magnitude(kernel_re, kernel_im, kernel_mag);

      if(kernel_name.compare("real") == 0)
         return kernel_re;
      else if(kernel_name.compare("imag") == 0)
         return kernel_im;
      else if(kernel_name.compare("mag") == 0)
         return kernel_mag;
      else
         printf("Invalid kernel name!\n");
   }

   void construct_gabor_bank()
   {
      double Kmax = PI/2;
      double f = sqrt(2.0);
      double sigma = 2*PI;
      int U = 7;
      int V = 4;
      int GaborH = 129;
      int GaborW = 129;

      Mat kernel;
      Mat totalMat;
      for(U = 0; U < 8; U++){
         Mat colMat;
         for(V = 0; V < 5; V++){
            kernel = getMyGabor(GaborW, GaborH, U, V,
               Kmax, f, sigma, CV_64F, "real");

            //show gabor kernel
            normalize(kernel, kernel, 0, 1, CV_MINMAX);
            printf("U%dV%d\n", U, V);

            if(V == 0)
               colMat = kernel;
            else
               vconcat(colMat, kernel, colMat);
         }
         if(U == 0)
            totalMat = colMat;
```

```
      else
         hconcat(totalMat, colMat, totalMat);
   }

   imshow("gabor bank", totalMat);
   waitKey(0);
}

Mat gabor_filter(Mat& img)
{
   // variables for gabor filter
   double Kmax = PI/2;
   double f = sqrt(2.0);
   double sigma = 2*PI;
   int U = 7;
   int V = 4;
   int GaborH = 129;
   int GaborW = 129;

   Mat kernel_re, kernel_im;
   Mat dst_re, dst_im, dst_mag;

   // variables for filter2D
   Point archor(-1,-1);
   int ddepth = -1;
   double delta = 0;

   // filter image with gabor bank
   Mat totalMat;
   for(U = 0; U < 8; U++){
      Mat colMat;
      for(V = 0; V < 5; V++){
         kernel_re = getMyGabor(GaborW, GaborH, U, V,
            Kmax, f, sigma, CV_64F, "real");
         kernel_im = getMyGabor(GaborW, GaborH, U, V,
            Kmax, f, sigma, CV_64F, "imag");

         filter2D(img, dst_re, ddepth, kernel_re);
         filter2D(img, dst_im, ddepth, kernel_im);

         dst_mag.create(img.rows, img.cols, CV_32FC1);
         magnitude(Mat_<float>(dst_re),Mat_<float>(dst_im),
            dst_mag);

         //show gabor kernel
         normalize(dst_mag, dst_mag, 0, 1, CV_MINMAX);
         printf("U%dV%d\n", U, V);
```

```
            if(V == 0)
                colMat = dst_mag;
            else
                vconcat(colMat, dst_mag, colMat);
        }
        if(U == 0)
            totalMat = colMat;
        else
            hconcat(totalMat, colMat, totalMat);
    }

    return totalMat;
}
```

1.2 固定摄像头前景提取代码

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <fstream>
#include"opencv2/xfeatures2d.hpp"
#include "package_bgs/bgslibrary.h"

using namespace cv::xfeatures2d;

//形态学处理功能
//膨胀函数
int dilation_elem = 0;
/** @function Dilation */
void Dilation(const cv::Mat &in, cv::Mat &out, int dilation_size = 13)
{
    int dilation_type;
    if (dilation_elem == 0) { dilation_type = MORPH_RECT; }
    else if (dilation_elem == 1) { dilation_type = MORPH_CROSS; }
    else if (dilation_elem == 2) { dilation_type = MORPH_ELLIPSE; }

    Mat element = getStructuringElement(dilation_type,
        Size(2 * dilation_size + 1, 2 * dilation_size + 1),
        Point(dilation_size, dilation_size));
    /// 膨胀操作
    dilate(in, out, element);
}

//腐蚀函数
int erosion_elem = 0;
/** @function Erosion */
```

```
void Erosion(const cv::Mat &in, cv::Mat &out, int erosion_size = 15)
{
   int erosion_type;
   if (erosion_elem == 0) { erosion_type = MORPH_RECT; }
   else if (erosion_elem == 1) { erosion_type = MORPH_CROSS; }
   else if (erosion_elem == 2) { erosion_type = MORPH_ELLIPSE; }

   Mat element = getStructuringElement(erosion_type,
      Size(2 * erosion_size + 1,2 * erosion_size + 1),
      Point(erosion_size, erosion_size));
   /// 腐蚀操作
   erode(in, out, element);
}

int main1(int argc, char **argv)
{
   std::cout << "Using OpenCV " << CV_MAJOR_VERSION << "." <<
       CV_MINOR_VERSION << "." << CV_SUBMINOR_VERSION << std::endl;
   string filefold("vedio/");
   string inputname("office");
   VideoCapture capture(filefold + inputname + ".avi");
   VideoWriter writer;
   writer.open(filefold + inputname + "-out.avi", CV_FOURCC('D', 'I', 'V',
       'X'), 30, cv::Size(capture.get(CV_CAP_PROP_FRAME_WIDTH),
       capture.get(CV_CAP_PROP_FRAME_HEIGHT)), false);
   IBGS *bgs;

   //SuBSENSE类包含LBSP算子与ViBe
   bgs = new SuBSENSE;

   bgs->setShowOutput(false);
   ofstream mycout(filefold + inputname+ "-frame.txt");
   int timeForSetBack = 0;
   int filthre = 0.9;
   cv::Mat orib;
   if (timeForSetBack > 0) {
      orib = cv::imread(filefold + inputname + ".jpg",
          CV_LOAD_IMAGE_GRAYSCALE);
   }
   int numframestore = 1;
   int framecount = 0;
   bool hastargetbefore1 = false;
   bool ins = false;
   bool hastargetnow;
   int key = 0;
   cv::Mat img_mask, img_mask1;
   cv::Mat img_bkgmodel;
```

```
cv::Mat img_input1, img_input, img_mask_old,
    inputold,changed,changed1,draw, inputold1;
Mat zero(img_input1.rows, img_input1.cols, CV_8UC1);
memset(zero.data, 0, zero.cols*zero.rows);
//图像预处理
for (int i = 0;i < timeForSetBack;i++) {
   //cv::cvtColor(orib, img_input, CV_BGR2GRAY);
   bgs->process(orib, img_mask, img_bkgmodel);
}
while (key != 'q')
{

   capture >> img_input1;
   if (img_input1.empty()) break;
   //图像预处理
   cv::cvtColor(img_input1, img_input, CV_BGR2GRAY);
   if (framecount == 0) {
      for (int k = 0;k < 0;k++) {
         bgs->process(img_input, img_mask, img_bkgmodel);
      }
   }

   Mat fin;
   //对每一帧图像进行处理
   bgs->process(img_input, img_mask, img_bkgmodel);
   //形态学变换，按需采用，可调节
// cv::medianBlur(img_mask, fin, 7);
// bgs3->process(fin, img_mask, img_bkgmodel);
// Dilation(img_mask, fin, 5);
// Erosion(fin, img_mask,5 );
   hastargetnow = false;

   int nf = 0;
   //遍历一副图像的像素点
   int nr = img_mask.rows;
   int nl = img_mask.cols*img_mask.channels();
   for (int k = 0;k < nr;k++) {
      const uchar* inData = img_mask.ptr<uchar>(k);
      uchar* outData = img_mask.ptr<uchar>(k);
      for (int i = 0;i < nl;i++) {
         if (outData[i] > UCHAR_MAX / 2) nf++;
      }
   }

   if (framecount++ <0) {
      nf = 0;
```

```
            memset(img_mask.data, 0, img_mask.cols*img_mask.rows);
        }

        //只有符合条件（足够显著）的前景目标才会被算法识别
        if (nf >0.005*(nr*nl)) {
            cout << 1.0*nf / (nr*nl) << endl;
            if (!hastargetbefore1) {
                mycout << framecount << "-";//输出有前景目标的帧数
            }
            hastargetbefore1 = true;

        }
        else {//不够显著则认为前景目标不存在，此举可过滤噪声
            memset(img_mask.data, 0, img_mask.cols*img_mask.rows);
            if (hastargetbefore1) {
                mycout << framecount - 1 << endl;
            }
            hastargetbefore1 = false;
        }
        writer << img_mask;
    }
    if(hastargetbefore1)mycout << framecount << endl;
    mycout.close();
    delete bgs;
    writer.release();
    capture.release();
    cvDestroyAllWindows();

    return 0;
}
```

1.3 摄像头抖动的前景提取代码

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <fstream>
#include "opencv2/xfeatures2d.hpp"
#include "package_bgs/bgslibrary.h"

using namespace cv::xfeatures2d;

//改进前的仿射变换（SUFT）变换，本函数实现：给两张相邻图，求出从前一图到后一图的变换H，以及按此变换得到的前一图
//有前景目标过滤的功能，特征点只会选在背景的图像上
void superChange(const cv::Mat &img_input1, const cv::Mat &img_input2,
    cv::Mat &input1_out, cv::Mat &imgMatches, const cv::Mat &mask) {
    Mat frame_prev, frame;
```

```
int minHessian = 400;

//SURF算法中的hessian阈值
double threshold = 0.3; //过滤匹配点时的阈值
Ptr<SURF> detector;
    //创建方式和OpenCV2中的不一样,并且要加上命名空间xfreatures2d</span>

detector = SURF::create(minHessian);

vector<KeyPoint> keypoints_prev,
    keypoints;//vector模板类，存放任意类型的动态数组
Ptr<Feature2D> extractor = SURF::create();
Mat descriptors_prev, descriptors;
FlannBasedMatcher matcher; // 实例化一个匹配器
vector<Point2f> queryCorners(4);

queryCorners[0] = Point2f(0, 0);
queryCorners[1] = Point2f(frame.cols - 1, 0);
queryCorners[2] = Point2f(frame.cols - 1, frame.rows - 1);
queryCorners[3] = Point2f(0, frame.rows - 1);

img_input1.copyTo(frame_prev);
img_input2.copyTo(frame);
//cvtColor(frame_prev, frame_prev, CV_BGR2GRAY);
//cvtColor(frame, frame, CV_BGR2GRAY);

//-- Step 1: 使用SURF算子检测关键点
// 调用detect函数检测出SURF特征关键点，保存在vector容器中
Mat mask_inv(mask.rows, mask.cols, CV_8UC1);
bitwise_not(mask, mask_inv);

//keypoints_prev = keypoints;
detector->detect(frame_prev, keypoints_prev, mask_inv);
detector->detect(frame, keypoints, mask_inv);

//-- Step 2: 使用SURF算子提取特征（计算特征向量）
extractor->compute(frame_prev, keypoints_prev, descriptors_prev);
extractor->compute(frame, keypoints, descriptors);

//-- Step 3: 使用FLANN法进行匹配
//匹配两幅图中的描述子（descriptors）
vector< vector< DMatch > > knnMatches;
matcher.knnMatch(descriptors_prev, descriptors, knnMatches, 2);
//cout << "number of matches before filtering: " << knnMatches.size() <<
    endl;

//-- 过滤匹配点，保留好的匹配点
```

```
vector< DMatch > goodMatches;
//vector < DMatch >().swap(goodMatches); //清空容器并最小化它的容量
for (int i = 0; i < knnMatches.size(); i++)
{
   if (knnMatches[i][0].distance < threshold * knnMatches[i][1].distance)
      goodMatches.push_back(knnMatches[i][0]);
}
//cout << "number of matches after filtering: " << goodMatches.size() <<
    endl;

//-- 绘制从两个图像中匹配出的关键点
//Mat imgMatches;
drawMatches(frame_prev, keypoints_prev, frame, keypoints, goodMatches,
    imgMatches,
   Scalar::all(-1), Scalar::all(0), vector<char>(),
   DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);//不显示未匹配的点

                                    //-- Step 4:
                                        使用findHomography找出相应的透视变换
vector<Point2f> scene_prev, scene;
//vector < Point2f >().swap(scene_prev); //清空容器并最小化它的容量
//vector < Point2f >().swap(scene);
for (int i = 0; i < goodMatches.size(); i++)
{
   //-- 从好的匹配中获取关键点:
       匹配关系是关键点间具有的一一对应关系，可以从匹配关系获得关键点的索引
   //-- e.g.
       这里的goodMatches[i].queryIdx和goodMatches[i].trainIdx是匹配中一对关键点的索引
   scene_prev.push_back(keypoints_prev[goodMatches[i].queryIdx].pt);
   scene.push_back(keypoints[goodMatches[i].trainIdx].pt);
}
Mat H = findHomography(scene_prev, scene, CV_RANSAC);

//-- Step 5: 使用perspectiveTransform映射点群，在场景中获得匹配位置
vector<Point2f> transCorners(4);
perspectiveTransform(queryCorners, transCorners, H);
//-- 变换后的四个顶点之间划线
int w = frame.cols;
line(imgMatches, transCorners[0] + Point2f(w, 0), transCorners[1] +
    Point2f(w, 0), Scalar(0, 255, 0), 4);
line(imgMatches, transCorners[1] + Point2f(w, 0), transCorners[2] +
    Point2f(w, 0), Scalar(0, 255, 0), 4);
line(imgMatches, transCorners[2] + Point2f(w, 0), transCorners[3] +
    Point2f(w, 0), Scalar(0, 255, 0), 4);
line(imgMatches, transCorners[3] + Point2f(w, 0), transCorners[0] +
    Point2f(w, 0), Scalar(0, 255, 0), 4);
```

```
    //-- 对前一帧进行变换
    warpPerspective(frame_prev, input1_out, H, frame.size());
}


//改进后的SUFT变换，比原先多去掉了一些较差的特征点，
void superChange1(const cv::Mat &img_input1, const cv::Mat &img_input2,
     cv::Mat &input1_out, cv::Mat &imgMatches, const cv::Mat &mask) {
    Mat frame_prev, frame;
    int minHessian = 400;

    //SURF算法中的hessian阈值
    double threshold = 0.4; //过滤匹配点时的阈值
    Ptr<SURF> detector; //创建方式和OpenCV2中的不一样,并且要加上命名空间xfreatures2d

    detector = SURF::create(minHessian);

    vector<KeyPoint> keypoints_prev,
        keypoints;//vector模板类，存放任意类型的动态数组
    Ptr<Feature2D> extractor = SURF::create();
    Mat descriptors_prev, descriptors;
    FlannBasedMatcher matcher; // 实例化一个匹配器
    vector<Point2f> queryCorners(4);

    //img_input1.copyTo(frame);
    queryCorners[0] = Point2f(0, 0);
    queryCorners[1] = Point2f(frame.cols - 1, 0);
    queryCorners[2] = Point2f(frame.cols - 1, frame.rows - 1);
    queryCorners[3] = Point2f(0, frame.rows - 1);

    img_input1.copyTo(frame_prev);
    img_input2.copyTo(frame);
    //cvtColor(frame_prev, frame_prev, CV_BGR2GRAY);
    //cvtColor(frame, frame, CV_BGR2GRAY);

    //-- Step 1: 使用SURF算子检测关键点
    // 调用detect函数检测出SURF特征关键点，保存在vector容器中
    Mat mask_inv(mask.rows, mask.cols, CV_8UC1);
    bitwise_not(mask, mask_inv);

    detector->detect(frame_prev, keypoints_prev, mask_inv);
    detector->detect(frame, keypoints, mask_inv);

    //-- Step 2: 使用SURF算子提取特征（计算特征向量）
    extractor->compute(frame_prev, keypoints_prev, descriptors_prev);
    extractor->compute(frame, keypoints, descriptors);
```

```
//-- Step 3: 使用FLANN法进行匹配
//匹配两幅图中的描述子（descriptors）
vector< vector< DMatch > > knnMatches1, knnMatches2;
vector< DMatch > goodMatches, outMatches;
matcher.knnMatch(descriptors_prev, descriptors, knnMatches1, 2);
matcher.knnMatch(descriptors, descriptors_prev, knnMatches2, 2);
cout << "number of matches before filtering: " << knnMatches1.size() <<
    endl;
cout << "number of matches before filtering: " << knnMatches2.size() <<
    endl;

//-- 根据knn过滤匹配点
for (vector<vector<DMatch> >::iterator matchIterator =
    knnMatches1.begin();
   matchIterator < knnMatches1.end();)
{
   if ((*matchIterator)[0].distance > threshold *
       (*matchIterator)[1].distance)
      knnMatches1.erase(matchIterator);
   else
      matchIterator++;
}
for (vector<vector<DMatch> >::iterator matchIterator =
    knnMatches2.begin();
   matchIterator < knnMatches2.end();)
{
   if ((*matchIterator)[0].distance > threshold *
       (*matchIterator)[1].distance)
      knnMatches2.erase(matchIterator);
   else
      matchIterator++;
}
cout << "number of matches after filtering: " << knnMatches1.size() <<
    endl;
cout << "number of matches after filtering: " << knnMatches2.size() <<
    endl;
//-- 根据对称性过滤匹配点
for (vector<vector<DMatch> >::iterator matchIterator1 =
    knnMatches1.begin();
   matchIterator1 < knnMatches1.end(); matchIterator1++)
{
   for (vector<vector<DMatch> >::iterator matchIterator2 =
       knnMatches2.begin();
      matchIterator2 < knnMatches2.end(); matchIterator2++)
   {
      if (((*matchIterator1)[0].queryIdx ==
          (*matchIterator2)[0].trainIdx) &&
```

```
            ((*matchIterator2)[0].queryIdx ==
                (*matchIterator1)[0].trainIdx))
        {
            goodMatches.push_back((*matchIterator1)[0]);
            break;
        }
    }
}
cout << "number of matches after symatic: " << goodMatches.size() <<
    endl;
//-- 根据F矩阵过滤匹配点
vector<Point2f> scene_prev, scene;
for (int i = 0; i < goodMatches.size(); i++)
{
    //-- 从好的匹配中获取关键点:
        匹配关系是关键点间具有的一一对应关系，可以从匹配关系获得关键点的索引
    //-- e.g.
        这里的goodMatches[i].queryIdx和goodMatches[i].trainIdx是匹配中一对关键点的索引
    scene_prev.push_back(keypoints_prev[goodMatches[i].queryIdx].pt);
    scene.push_back(keypoints[goodMatches[i].trainIdx].pt);
}
vector<uchar> status(scene_prev.size());
Mat F = findFundamentalMat(Mat(scene_prev), Mat(scene),
    CV_FM_RANSAC, 1, 0.99, status);
for (int i = 0; i < status.size(); i++) {
    if (status[i]) //根据模板，重新优化匹配组及特征点，找到最优的匹配组
        outMatches.push_back(goodMatches[i]);
}
cout << "number of matches after RANSAC: " << outMatches.size() << endl;

scene_prev.clear();
scene.clear();
for (int i = 0; i < outMatches.size(); i++)
{
    scene_prev.push_back(keypoints_prev[outMatches[i].queryIdx].pt);
    scene.push_back(keypoints[outMatches[i].trainIdx].pt);
}
F = findFundamentalMat(Mat(scene_prev), Mat(scene), CV_FM_8POINT);


//-- 绘制从两个图像中匹配出的关键点
drawMatches(frame_prev, keypoints_prev, frame, keypoints, goodMatches,
    imgMatches,
    Scalar::all(-1), Scalar::all(0), vector<char>(),
    DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);//不显示未匹配的点

Mat H = findHomography(scene_prev, scene, CV_RANSAC);
```

```
    //-- Step 5: 使用perspectiveTransform映射点群，在场景中获得匹配位置
    vector<Point2f> transCorners(4);
    perspectiveTransform(queryCorners, transCorners, H);
    //-- 变换后的四个顶点之间划线
    int w = frame.cols;
    line(imgMatches, transCorners[0] + Point2f(w, 0), transCorners[1] +
        Point2f(w, 0), Scalar(0, 255, 0), 4);
    line(imgMatches, transCorners[1] + Point2f(w, 0), transCorners[2] +
        Point2f(w, 0), Scalar(0, 255, 0), 4);
    line(imgMatches, transCorners[2] + Point2f(w, 0), transCorners[3] +
        Point2f(w, 0), Scalar(0, 255, 0), 4);
    line(imgMatches, transCorners[3] + Point2f(w, 0), transCorners[0] +
        Point2f(w, 0), Scalar(0, 255, 0), 4);

    //-- 对前一帧进行变换
    warpPerspective(frame_prev, input1_out, H, frame.size());
}


int dilation_elem = 2;
/** @function Dilation */
void Dilation(const cv::Mat &in, cv::Mat &out, int dilation_size = 13)
{
    int dilation_type;
    if (dilation_elem == 0) { dilation_type = MORPH_RECT; }
    else if (dilation_elem == 1) { dilation_type = MORPH_CROSS; }
    else if (dilation_elem == 2) { dilation_type = MORPH_ELLIPSE; }

    Mat element = getStructuringElement(dilation_type,
       Size(4 * dilation_size + 1, 4 * dilation_size + 1),
       Point(dilation_size, dilation_size));
    /// 膨胀操作
    dilate(in, out, element);
}
int erosion_elem =2;
/** @function Erosion */
void Erosion(const cv::Mat &in, cv::Mat &out, int erosion_size = 15)
{
    int erosion_type;
    if (erosion_elem == 0) { erosion_type = MORPH_RECT; }
    else if (erosion_elem == 1) { erosion_type = MORPH_CROSS; }
    else if (erosion_elem == 2) { erosion_type = MORPH_ELLIPSE; }

    Mat element = getStructuringElement(erosion_type,
       Size(2 * erosion_size + 1, 2 * erosion_size + 1),
       Point(erosion_size, erosion_size));
```

```
    /// 腐蚀操作
    erode(in, out, element);
}

//case处理样例
int main354(int argc, char **argv)
{
    //一些程序上的准备工作
    std::cout << "Using OpenCV " << CV_MAJOR_VERSION << "." <<
        CV_MINOR_VERSION << "." << CV_SUBMINOR_VERSION << std::endl;
    string filefold("vedio/");
    string inputname("people1");
    VideoCapture capture(filefold + inputname + ".avi");
    VideoWriter writer;
    writer.open(filefold + inputname + "-out.avi", CV_FOURCC('D', 'I', 'V',
        'X'), 30, cv::Size(capture.get(CV_CAP_PROP_FRAME_WIDTH),
        capture.get(CV_CAP_PROP_FRAME_HEIGHT)), false);
    IBGS *bgs;
    IBGS *bgs2;
    bgs = new FrameDifference;
    bgs2 = new SuBSENSE;
    bgs2->setShowOutput(false);
    ofstream mycout(filefold + "temp.txt");
    int timeForSetBack = 0;
    int filthre = 0.9;
    cv::Mat orib;
    if (timeForSetBack > 0) {
        orib = cv::imread(filefold + inputname + ".jpg",
            CV_LOAD_IMAGE_GRAYSCALE);
    }
    int numframestore = 1;
    int framecount = 0;
    bool *hastargetbefore = new bool[numframestore];
    bool hastargetbefore1 = false;
    for (int i = 0;i < numframestore;i++) {
        hastargetbefore[i] = false;
    }
    bool ins = false;
    bool hastargetnow;
    int key = 0;
    cv::Mat img_mask, img_mask1;
    cv::Mat img_bkgmodel;
    cv::Mat img_input1, img_input, img_mask_old, inputold, changed,
        changed1, draw, inputold1;//k
    capture >> img_input;
    if (img_input.empty()) {
        mycout.close();
```

```
    delete bgs;
    writer.release();
    capture.release();
    return 0;
}
cv::cvtColor(img_input, img_input1, CV_BGR2GRAY);
Mat zero(img_input1.rows, img_input1.cols, CV_8UC1);
memset(zero.data, 0, zero.cols*zero.rows);
zero.copyTo(img_mask);
bgs2->process(zero, img_mask, img_bkgmodel);
bgs2->process(zero, img_mask, img_bkgmodel);
bgs2->process(zero, img_mask, img_bkgmodel);
img_input1.copyTo(inputold);

while(key != 'q')
{//本循环即对每一帧图像进行处理
    bool ifshowSUFT = true;//是否实时显示处理过程
    //bgs = new AdaptiveSelectiveBackgroundLearning;
    bgs->setShowOutput(false);
    //保存旧图像，因为之后的仿射变换为每帧对前两帧进行处理
    inputold.copyTo(inputold1);
    img_input1.copyTo(inputold);
    //读取新图像
    capture >> img_input;//capture >> img_input;capture >> img_input;
    if (img_input.empty()) {
        mycout.close();
        delete bgs;
        writer.release();
        capture.release();
        return 0;
    }
    cv::cvtColor(img_input, img_input1, CV_BGR2GRAY);
    //在这里使用仿射变换
    superChange1(inputold1, img_input1, changed, draw, img_mask);
    //变换完后需要去掉图像边缘的无效区域
    Mat img_input_ROI = changed(Rect(changed.cols / 20, changed.rows /
        20, changed.cols / 20 * 18, changed.rows / 20 * 18));
    memset(zero.data, 0, zero.cols*zero.rows);
    Mat zero_ROI = zero(Rect(zero.cols / 20, zero.rows / 20, zero.cols /
        20 * 18, zero.rows / 20 * 18));
    img_input_ROI.copyTo(zero_ROI);
    bgs->process(zero, img_mask, img_bkgmodel);

    if(ifshowSUFT)imshow("zero1", zero);
    //去掉图像边缘的无效区域
    img_input_ROI = img_input1(Rect(img_input1.cols / 20, img_input1.rows
        / 20, img_input1.cols / 20 * 18, img_input1.rows / 20 * 18));
```

```
        memset(zero.data, 0, zero.cols*zero.rows);
        zero_ROI = zero(Rect(zero.cols / 20, zero.rows / 20, zero.cols / 20 *
            18, zero.rows / 20 * 18));
        img_input_ROI.copyTo(zero_ROI);
        if (ifshowSUFT)imshow("zero2", zero);
        bgs->process(zero, img_mask, img_bkgmodel);
        if (ifshowSUFT)imshow("draw", draw);
        if (ifshowSUFT)imshow("changed", changed);

        Mat fin(img_mask.rows, img_mask.cols, CV_8UC1);

        //中值滤波
        cv::medianBlur(img_mask, fin,13);//cv::medianBlur(fin, img_mask,1);
        //先膨胀
        Dilation(fin, img_mask,2);
        //后腐蚀
        Erosion(img_mask, fin,2);
        if (ifshowSUFT)imshow("img_mask", fin);
        writer << fin;
        cvWaitKey(200);
    }

    mycout.close();
    delete bgs;
    writer.release();
    capture.release();
    cvDestroyAllWindows();

    return 0;
}
```

1.4 群体运动特征提取代码

```
#include<iostream>
#include<opencv2/opencv.hpp>
#include<vector>
#include <limits>
#include<math.h>

using namespace cv;
using namespace std;

//各个分函数计算特征点列（样本）的各项统计学属性数值

//计算一个点与一个点列中的点的最短距离的评方
```

```
float calDistSq(Point2f point, vector<Point2f> points) {
   float minDis = numeric_limits<float>::max();
   for (int i = 0;i < (int)points.size();i++) {
      float dis = (point.x - points[i].x)*(point.x - points[i].x) +
          (point.y - points[i].y)*(point.y - points[i].y);
      if (dis < minDis) {
         minDis = dis;
      }
   }
   return minDis;
}

//计算一个点列相对于另一个点列的动能
float calKE(vector<Point2f> points, vector<Point2f> points2) {
   float ke = 0;
   for (int i = 0;i < (int)points.size();i++) {
      ke += calDistSq(points[i], points2);
   }
   return ke / points.size();
}

//计算一个点列的中心位置
Point2f* calAvg(vector<Point2f> points) {
   float ax = 0, ay = 0;
   for (int i = 0;i < (int)points.size();i++) {
      ax += points[i].x;
      ay += points[i].y;
   }
   ax = ax / points.size();
   ay = ay / points.size();
   return new Point2f(ax, ay);
}

//计算一个点列的分散程度（方差）
Point2f* calVar(vector<Point2f> points, Point2f avg) {
   float axx = 0, ayy = 0;
   for (int i = 0;i < (int)points.size();i++) {
      axx += (points[i].x - avg.x)*(points[i].x - avg.x);
      ayy += (points[i].y - avg.y)*(points[i].y - avg.y);
   }
   axx = axx / points.size();
   ayy = ayy / points.size();
   return new Point2f(axx, ayy);
}

int r = 30;
float calPE(vector<Point2f> points) {
```

```
    float PE = 0;
    for (int i = 0;i < points.size();i++) {
        int Ni = 0;
        float PEi = 0;
        for (int j = 0;j<points.size();j++) {
            float dis = sqrt((points[i].x - points[j].x)*(points[i].x -
                points[j].x) + (points[i].y - points[j].y)*(points[i].y -
                points[j].y));
            if (dis < r) {
                PEi += dis;
                Ni++;
            }
        }
        PE += PEi / Ni;
    }
    PE = PE / points.size();
    return PE;
}

int main()
{
    int lastframe = 238,beginframe=234;
    VideoCapture capture("vedio/campus4-c0-out.avi");
    Mat input,lastinput;
    int fc = 0;
    capture >> lastinput;
    capture >> input;
    cout << input.size() << endl;
    while (fc <= lastframe) {
        input.copyTo(lastinput);
        capture >> input;
        if (input.empty()) break;
        fc++;
        if (fc >= beginframe) {
            cout << "当前第" << fc << "帧,优质角点一共有";
            Mat srcImage;
            cvtColor(lastinput, srcImage, CV_BGR2GRAY);

            //求取上一帧的角点
            //强角点检测函数的输入图像是一个单通道的图像
            //开始进行强角点检测
            //所用角点为Harris角点
            vector<Point2f> points;
            goodFeaturesToTrack(srcImage, points, 20, 0.01, 10, Mat(), 3,
                true);
            /*cout << "pic1:" << points.size() << endl;
            Point2f* avg = calAvg(points);
```

```
        cout << "x均值:" << (*avg).x << ",y均值:" << (*avg).y << endl;
        Point2f* var = calVar(points, (*avg));
        cout << "x方差:" << (*var).x << ",y方差:" << (*var).y << endl;
        delete avg;
        delete var;*/
        //遍历每个点，进行绘制，便于显示
        Mat dstImage;
        srcImage.copyTo(dstImage);
        for (int i = 0; i < (int)points.size(); i++)
        {
           circle(dstImage, points[i], 3, Scalar(theRNG().uniform(0, 255),
               theRNG().uniform(0, 255), theRNG().uniform(0, 255))
              , 2, 8);
        }
        imshow("【上一帧检测到的角点图】", dstImage);

        Mat srcImage2;// = imread("34.jpg", 0);
        cvtColor(input, srcImage2, CV_BGR2GRAY);

        //求取当前帧的角点
        //强角点检测函数的输入图像是一个单通道的图像
        //开始进行强角点检测
        //所用角点为Harris角点
        vector<Point2f> points2;
        goodFeaturesToTrack(srcImage2, points2, 20, 0.01, 10, Mat(), 3,
            true);
           cout << int(points2.size()) << "个。"<<endl;
        Point2f* avg = calAvg(points2);
        cout << "x均值:" << (*avg).x << ",y均值:" << (*avg).y << endl;
        Point2f* var = calVar(points, (*avg));
        cout << "x方差:" << (*var).x << ",y方差:" << (*var).y << endl;
        delete avg;
        delete var;

        //遍历每个点，进行绘制，便于显示
        Mat dstImage2;
        srcImage2.copyTo(dstImage2);
        for (int i = 0; i < (int)points2.size(); i++)
        {
           //cout << points2[i].x << "," << points2[i].y << endl;
           circle(dstImage2, points2[i], 3, Scalar(theRNG().uniform(0,
               255), theRNG().uniform(0, 255), theRNG().uniform(0, 255))
              , 2, 8);
        }
        imshow("【当前帧检测到的角点图】", dstImage2);
        // cout << "<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<" << endl;
        cout << "当前帧相对于上一帧的动能:" << calKE(points2, points) << endl;
```

```
            cout << "当前帧势能:" << calPE(points2) << endl;
            cout << "<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<" << endl;
            waitKey(100);
        }
    }
    waitKey(0);
    return 0;
}
```