

(由组委会填写)



“华为杯”第十四届中国研究生 数学建模竞赛

学 校	西安电子科技大学
-----	----------

参赛队号	10701013
------	----------

队员姓名	1.	李宜烜
	2.	张楷伟
	3.	柳林

参赛密码 _____

(由组委会填写)



“华为杯”第十四届中国研究生 数学建模竞赛

题 目 基于监控视频的前景目标提取

摘 要

智能视频监控，作为计算机视觉的重要应用方面，近年得到广泛重视。前景检测是其关键步骤之一，其性能对于目标分类、跟踪和行为理解等后期处理是非常重要的。目前几种常见的前景检测方法有：帧差法、背景差分法、光流法等，但均难以适应存在复杂背景和运动情况复杂的场景。此外应用最广泛的背景提取模型是高斯混合模型 GMM (Gaussian Mixture Model)，但该模型计算复杂，且对慢速前景检测效果不好，会将纹理和对比度低的前景当作背景，另外对全局亮度的突然变化敏感，会造成误检。

本文采用一种新的前景目标提取模型，以 ViBe+模型作为核心，进行优化建模。该模型是对 ViBe 模型的一种改良，继承了 ViBe 模型有效抑制阴影的优点，具有较好的前景检测性能。但在 ViBe 存在的鬼影问题上，依然解决不够彻底，因此本文采取多帧取平均的方式，产生一张虚拟背景图像，用于 ViBe 的初始化，有效的抑制了鬼影的产生。此外针对晃动视频容易存在背景干扰及噪声影响的问题，进一步提出用形态学方法结合采用提取 SIFT 特征的方式，对视频隔帧进行匹配，有效的抑制了噪声以及错误信息的识别。因此，本文以 ViBe+为核心，提

出了新的前景目标提取的优化模型，并将其用 C++ 和 opencv 以及 Matlab 进行实现。

对于问题一，考虑到视频过短以及光照对前景目标提取的影响，本文建立了**改进的帧差法模型**，对静态背景视频进行前景目标提取，通过滤波和图像形态学的方法进行优化，并与高斯混合模型所提取前景目标视频进行了对比，所提取的结果（见表 3-1）显示效果明显优于高斯混合模型。

对于问题二，利用**改进的 ViBe+模型**和**虚拟背景图像生成模型**，对动态场景视频中的前景目标进行提取，并通过高斯滤波以及去除小连通区域的方式，优化了提取结果。所得结果与高斯混合模型提取的前景目标视频进行了对比（见表 4-1），可以看出该模型有效的降低了前景错误识别率。

对于问题三，首先采用**基于 SIFT 特征点的隔帧匹配模型**，对图像进行矫正，然后采用问题二的解决方法，对背景晃动的视频进行前景识别，最后通过形态学的方式，对生成的前景检测视频进行处理。结果与直接使用问题二算法提取的前景目标视频进行对比（见表 5-1），可以看出该模型有效的抑制了因抖动而产生的噪声。

对于问题四，首先利用问题二的模型，对附件三中所给视频进行前景目标检测，对输出的二值化视频的每一帧，计算前景目标所占全帧的比例。将每帧的比例统计后绘制前景比例折线图，按一定的比例阈值，找出前 N 个局部最大值，接着向局部最大值两侧进行搜索，按一定阈值，高精度的找出每个视频中包含显著前景目标的视频帧标号，提取帧号的具体操作以及结果见第六章，通过误差分析，得出问题四模型误差为显著目标出现前后 20 帧左右。

对于问题五，首先建立基于**对极几何模型**约束下的**SIFT 特征点匹配模型**，实现不同角度摄像机视图中的特征点检测结果的匹配，进而提取出不同角度的相同目标，在第七章中对提取相同目标的方法进行了描述与实现，通过结果（图 7-4，图 7-5）可看出，该方法可以较准确的提取出不同场景中的相同目标。

对于问题六，首先提出用时空特征点来描述人类群体的运动，再利用高斯混合模型对群体正常异常事件进行概率描述，最后使用 SVM 对行为属性和视频建立的向量进行训练学习，得到的系统模型可以准确的对视频的群体事件进行分类。

关键字：帧差法，ViBe+，SIFT 特征点，虚拟背景图像，隔帧匹配，对极几何，SVM，高斯混合模型

1 问题重述

视频监控是中国安防产业中最为重要的信息获取手段。随着“平安城市”建设的顺利开展，各地普遍安装监控摄像头，利用大范围监控视频的信息，应对安防等领域存在的问题。近年来，中国各省市县乡的摄像头数目呈现井喷式增长，大量企业、部门甚至实现了监控视频的全方位覆盖。如北京、上海、杭州监控摄像头分布密度约分别为 71、158、130 个/平方公里，摄像头数量分别达到 115 万、100 万、40 万，为我们提供了丰富、海量的监控视频信息。

目前，监控视频信息的自动处理与预测在信息科学、计算机视觉、机器学习、模式识别等多个领域中受到极大的关注。而如何有效、快速抽取出监控视频中的前景目标信息，是其中非常重要而基础的问题[1-6]。这一问题的难度在于，需要有效分离出移动前景目标的视频往往具有复杂、多变、动态的背景[7, 8]。这一技术往往能够对一般的视频处理任务提供有效的辅助。以筛选与跟踪夜晚时罪犯这一应用为例：若能够预先提取视频前景目标，判断出哪些视频并未包含移动前景目标，并事先从公安人员的辨识范围中排除；而对于剩下包含了移动目标的视频，只需辨识排除了背景干扰的纯粹前景，对比度显著，肉眼更易辨识。因此，这一技术已被广泛应用于视频目标追踪，城市交通检测，长时场景监测，视频动作捕捉，视频压缩等应用中。

在本题中，需要解决如下四个方面的问题：

问题 1： 对一个不包含动态背景、摄像头稳定拍摄时间大约 5 秒的监控视频，构造提取前景目标（如人、车、动物等）的数学模型，并对该模型设计有效的求解方法，从而实现类似图 1 的应用效果。



图 1 左图：原视频帧；右图：分离出的前景目标

问题 2： 对附件二中包含的动态背景信息的监控视频，设计有效的前景目标提取方案。

问题 3： 在监控视频中，当监控摄像头发生晃动或偏移时，视频也会发生短暂的抖动现象（该类视频变换在短时间内可近似视为一种线性仿射变换，如旋转、平移、尺度变化等）。对这种类型的视频，进行有效地提取前景目标。

问题 4： 在附件 3 中提供了 8 组视频（avi 文件与 mat 文件内容相同）。利用所构造的建模方法，从每组视频中选出包含显著前景目标的视频帧标号，并将其在建模论文正文中独立成段表示。务须注明前景目标再视频中的出现（如 Campus 视频）的帧数（如 241-250，421-432 帧）。

问题 5： 通过从不同角度同时拍摄的近似同一地点的多个监控视频中（如图 2 所示）有效检测和提取视频前景目标。请充分考虑并利用多个角度视频的前景之间（或背景之间）相关性信息。



图 2 在室内同一时间从不同角度拍摄同一地点获得的视频帧

问题 6: 利用所获取前景目标信息, 自动判断监控视频中有人群短时聚集、人群惊慌逃散、群体规律性变化 (如跳舞、列队排练等)、物体爆炸、建筑物倒塌等异常事件。可考虑的特征信息包括前景目标奔跑的线性变化形态特征、前景规律性变化的周期性特征等。尝试对更多的异常事件类型, 设计相应的事件检测方案。

2 模型假设和符号说明

2.1 模型假设

为了使本文提出的前景目标提取优化模型再贴近实际的前提下, 能够更合理的检测出前景目标, 再建模使提出如下假设:

- 1) 假设监控视频信号稳定, 不受其他电子信号干扰。
- 2) 假设对前景目标的提取无时间上的要求。
- 3) 假设视频噪声主要由视频抖动和动态背景产生。
- 4) 假设每秒帧率大于 30。
- 5) 假设视频不拍摄于恶劣环境下。

2.2 符号说明

表 2.1 常用符号说明

符号	符号说明
$I_t(x, y)$	表示 t 时刻 (x, y) 位置像素值
$Mask_t(x, y)$	当前时刻目标前景蒙版
$Grad_k(x, y)$	第 k 帧图像中坐标为 (x, y) 的像素点在 5×5 窗口中的梯度值
$Fd_k(x, y)$	梯度系数的帧差值
$N_G(x)$	x 的 8 邻域
$M(x)$	背景模型
$S_R(v(x))$	表示以 $v(x)$ 为中心, 半径为 R 的 2-D 欧氏空间
$v(x)$	在给定的欧氏颜色空间中图像在 x 处的像素值
$P(x) = \{f_1, f_2, \dots, f_n\}$	监控视频模型
$G(x, y, \sigma)$	尺度可变高斯函数

3 问题一模型的建立和求解

3.1 问题分析

针对问题一，对于静态背景中的前景目标进行提取，通过对视频的分析，可以发现，每个视频时间较短，帧数较少，针对这种短的视频，采用需要训练的模型，如下文要用到的 ViBe 算法[1]，得出的结果会产生鬼影等不好的结果，且视频中多有灯光的微弱闪烁，因为帧差法[2]对光照不是很敏感，所以在效果上会比高斯混合模型出色很多。因此本文，在问题一中，应用一种改进的帧差法来进行实现，并获得了较好的效果。

3.2 模型建立与求解

3.2.1 帧差法模型：

利用视频序列中相邻两帧或多帧进行差分，从而提取出图像中短时间内变换剧烈的部分， $I_t(x,y)$ 和 $I_{t-1}(x,y)$ 表示 t 和 $t-1$ 时刻图像中 (x,y) 位置像素值，则帧差法利用下式进行判决：

$$Mask_t(x,y) = \begin{cases} 1 & |I_t(x,y) - I_{t-1}(x,y)| > T_d \\ 0 & |otherwise| \end{cases} \quad (3-1)$$

其中 T_d 为阈值， $Mask_t(x,y)$ 为当前时刻目标前景蒙版， $Mask_t(x,y)$ 为1表示 (x,y) 位置像素为目标前景，为0表示该像素为背景。在目标检测图像处理过程中，目标前景的检测结果通常用一个二值图像进行表示，一般称其为前景的蒙版(Mask)，蒙版图像中非零的像素对应图像中的前景位置，为零的像素对应图像的背景位置。上式中将连续视频帧中对应位置像素点亮度值相减得到的差值与预先设置的阈值 T_d 进行比较，差值大于阈值就判定为目标前景，即运动区域。

本文采用相邻帧差[3]并采用全局阈值获得变化检测模板，然后采用局部阈值的迭代松弛技术来完成边沿的光滑滤波，为了分割不连续的运动目标，使用了一个深度为 L 的可调节运动目标，则该像素属于运动目标。问题一利用帧差法建立模型，该模型的优点是，每次侦查都是每一次的帧差和当前图像进行比较，所以对环境动态变化的运动目标具有较强的鲁棒性。

3.2.2 帧差法梯度比较模型

在原始帧差法的基础上，令 $Grad_k(x,y)$ 表示第 k 帧图像中坐标为 (x,y) 的像素点在 5×5 窗口中的梯度值，则：

$$Grad_k(x,y) = \sum_{i=-2}^2 \sum_{j=-2}^2 (|f_k(x+i, y+j+1) - f_k(x+i, y+j)|) + \sum_{i=-2}^2 \sum_{j=-2}^2 (|f_k(x+i+1, y+j) - f_k(x+i, y+j)|) \quad (3-2)$$

比较相邻两帧相同坐标位置像素的梯度值，梯度相差越大在计算帧差时的权重系数就越大。令 r 表示梯度系数， $Fd_k(x,y)$ 表示引入梯度系数的帧差值，则：

$$Fd_k(x,y) = d_k(x,y) \times r \quad (3-3)$$

$$r = \begin{cases} 1 & \text{Grad}_{\max} \leq \frac{3}{2} \text{Grad}_{\min} \\ \frac{3}{2} & \frac{3}{2} \text{Grad}_{\min} < \text{Grad}_{\max} \leq \frac{5}{2} \text{Grad}_{\min} \\ 2 & \text{Grad}_{\max} > \frac{5}{2} \text{Grad}_{\min} \end{cases} \quad (3-4)$$

其中 $\text{Grad}_{\min} = \min\{\text{Grad}_k, \text{Grad}_{k-1}\}$, $\text{Grad}_{\max} = \max\{\text{Grad}_k, \text{Grad}_{k-1}\}$ 。

梯度差一般应用在图像边缘检测、分割和合并中，因为在物体边缘像素点变化比较大。引入梯度差能够很好地突出运动物体边界以及内部边缘，突出运动像素点，为后面的连通分量处理提供方便。式(4)中的系数可以根据不同的场景进行设置。但是因为采集到的图像中具有随机性的噪声，梯度系数也会将噪声的影响放大。为了消除噪声的影响，还需要将帧差通过中值滤波。

将 $Fd_k(x, y)$ 与前 4 帧的帧差数据进行排序，将中间的值作为输出，再将中值滤波的值与阈值进行比较，得到最终阈值化结果。噪声一般都是随机性出现的，而且持续时间很短，所以采用中值滤波就可以消去那些突然变化且持续时间较短的噪声。设中值滤波后的值为 $Fd_{\text{mid}_k}(x, y)$ ：

$$Fd_{\text{mid}_k}(x, y) = \text{middle}\{Fd_{k-4}(x, y), Fd_{k-3}(x, y), Fd_{k-2}(x, y), Fd_{k-1}(x, y), \\ Fd_{k-1}(x, y), Fd_k(x, y)\} \quad (3-5)$$

经过阈值处理后的结果还是比较粗糙的，为了得到比较精确的运动物体的轮廓和大小，还需要对阈值化后的图像数据进行连通区域的标记。大部分的连通域标记算法 [10,11] 都需要对图像进行多次的扫描或采用标记对查找表，增加了时间和存储空间的花销。本方法中采用的是 8 邻域求连通区域。如果当前点满足外部阈值设置的要求，则修改该点的值使其不再满足阈值要求，并对其周围 8 个像素点进行搜索，如果周围 8 个点中有满足阈值的点，则修改这些点的值使其不再满足阈值要求并记录这些点，然后再分别以这些点为中心进行搜索。重复上述过程，直至没有满足阈值的点。

经过连通分量处理，把图像分为暂时运动区域和背景区域。其中暂时运动区域包括相邻帧差法判定的运动物体和其内部空洞或内部背景。

3.2.3 模型求解步骤

(1) 高斯滤波

首先对输入的监控视频进行微小的滤波，进行平滑处理，可以更好的抑制了内部的空洞产生。

(2) 改进帧差法提取前景目标

接着利用本节建立的帧差法，提取视频中的前景目标，接着采用帧差法梯度比较模型，引入梯度差能够很好地突出运动物体边界以及内部边缘，突出运动像素点。并按帧存储。

(3) 形态学处理

利用图像处理中形态学的办法，将得到的帧先进行膨胀操作，使提取出的前景轮廓和内部内容更充实。接着采用腐蚀的方式，使轮廓更加准确。

(4) 去除小连通域

为了进一步的降低噪声，最后采用去除小连通域的方式，完善每一帧，最后

生成视频。

3.3 实验结果与分析

3.3.1 实验结果：

针对附件一所给视频的前景目标识别，并与同样高斯滤波后的高斯混合模型进行对比，效果如下表(3-1)所示：








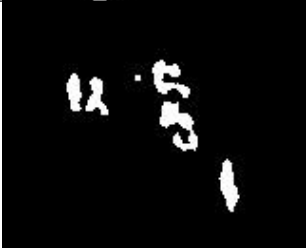
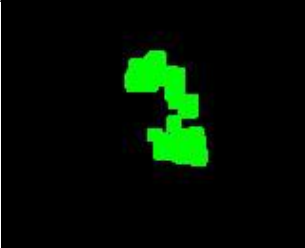


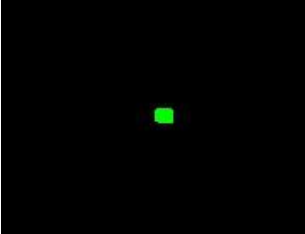



原图像	ViBe	高斯混合模型
		
		
		
		
		

表 3-1

3.3.2 结果分析

通过实验结果，我们可以看出，再最后两帧显示结果，高斯模型对烟雾的识别十分不好，对人群等细节的提取也不如利用本节所描述的改进帧差法，在前三个视频上的前景目标提取也漏掉了细节，在除前景目标以外的地方，改进的帧差法基本没有噪声存在，并基本识别出全部前景目标，除颜色相近区域检测前景有残缺外，对前景的检测轮廓基本清晰，较 GMM 混合模型效果优秀。

4 问题二模型的建立和求解

4.1 问题分析

上述问题一中，采用了目前应用最广泛的背景建模方法是高斯混合模型 GMM (Gaussian Mixture Model)。GMM 算法在背景建模过程中允许运动目标存在，因此适合室外有轻微光线和天气变化的小而速度快和静态背景的前景检测，但是对慢速前景检测以及动态背景的前景检测效果不好，会将纹理和对比度低的前景当作背景，另外对全局亮度的突然变化敏感，会造成误检。因此，在问题 2 对于包含动态背景信息的监控视频的前景目标进行检测中，不适于应用 GMM 算法来进行前景目标提取。因此，本文问题二应用 ViBe 算法得改进算法 ViBe+算法为建模的核心算法，并通过多帧取平均的方式，产生一张虚拟背景图像，用于 ViBe 的初始化，来弥补 ViBe+算法在鬼影问题上没有完全解决的缺点。最有通过形态学的方法来完善本模型算法。

4.2 模型建立与求解

4.2.1 ViBe+模型基本原理

ViBe(visual background extractor)算法，即视觉背景提取子算法，用来实现快速的背景提取和运动目标检测。它是一种基于样本随机聚类的背景建模算法，具有实时性和鲁棒性较高的优点，ViBe+在 ViBe 在基础上，可以更好的提取前景目标，并可以保留前景的精致目标，算法的核心思想是：

(1) 背景模型定义：背景模型中的每一个像素由 n 个背景样本组成（此处， n 取值为 20），记 $v(x)$ 表示在给定的欧氏颜色空间中图像在 x 处的像素值， v_i 表示索引为 i 的背景样本值。背景模型 M 定义如式(4-1)所示：

$$M(x) = \{v_1, v_2, \dots, v_{n-1}, v_n\} \quad (4-1)$$

(2) 背景初始化：初始化背景的过程也是选取 $v(x)$ 的过程，ViBe 算法从 x 的 8 邻域 $N_G(x)$ 中随机选取 20 个样本值用于初始化背景模型，如式(4-2)所示：

$$M^0(x) = \{v^0(y) | y \in N_G(x)\} \quad (4-2)$$

(3) 随机选择策略：ViBe 算法采用随机选择策略从 x 的邻域中选取用于初始化背景模型的样本，这种随机选择技术保证了邻域中的每一个样本都有同等的概率被选中，从而使得构造出的背景模型不存在主观偏见因素，保证了背景模型的真实可靠。

(4) 像素分类方法：ViBe 算法采用 2-D 空间中的欧氏距离对像素进行分类，记 $S_R(v(x))$ 表示以 $v(x)$ 为中心， $M(x)$ 表示半径为 R 的 2-D 欧氏空间(见图 4-1)，若 $S_R(v(x))$ 与 $M(x)$ 的交集满足一定的基数($H\{\}$ 表示交集不小于 2)，则认为 $v(x)$ 是背景像素，如式(4-3)所示：

$$H\{S_R(v(x)) \cap M(x)\} \quad (4-3)$$

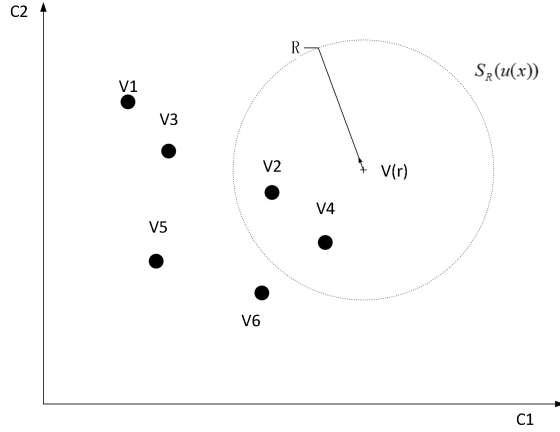


图 4-1 2-D 欧氏空间中像素分类

$$v_label(v(x)) = \begin{cases} 1 & S_R(v(x)) \cap M(x) \geq 2 \\ 0 & S_R(v(x)) \cap M(x) < 2 \end{cases}$$

$$M(x_r) = \begin{cases} v(x) & v_label(v(x)) = 1 \\ M(x_r) & v_label(v(x)) = 0 \end{cases} \quad (4-4)$$

(5) 背景模型更新策略: 式中 x_r 代表 M 中随机选取的像素位置, 选取帧中任一像素 $p(x)$, 当 $v_label(p(x)) = 1$, 则判断为背景像素, 反之则为前景像素, 如果像素 $p(x)$ 是背景像素, 则随机地从 $M(x)$ 中选取一个值采用 $p(x)$ 代替, 这种统一分布的随机选取法保证了样本集合中每一个样本的生命周期成指数递减, 在更新一个像素的样本集中的 20 个样本值的时候, 是随机的从 20 个中选一个的, 这样的样本在 T 时刻不被更新的概率为 $(N-1) \div N$ 。假设时间是连续的, 那么在 dt 时间之后, 样本集中的样本值仍然保留不被更新的概率为:

$$P(t, t+dt) = \left(\frac{N-1}{N} \right)^{(t+dt)-t} \quad (4-5)$$

避免了像素长期保留在背景模型中影响模型的精确性;

(6) 信息传播机制: 为了保持像素邻域空间的一致性, ViBe 算法对 $v(x)$ 的样本进行更新的同时, 采用同样的更新方法来更新邻域 $N_G(x)$ 中像素的样本; 例如, 采用 $v(x)$ 替换样本模型 $M(x)$ 中的一个样本, 同时采用 $v(x)$ 更新 $N_G(v(x))$ 中某一个像素的样本。由 ViBe 算法的核心描述可以看出, 背景模型的定义、初始化、更新以及像素的分类都比较简单; 整个算法中没有复杂的计算, 从而保证了算法的实时性。

4.2.2 ViBe+算法的优点表现在:

(1) ViBe+更好的提取前景区域。

ViBe+算法在背景模型更新策略中, 强行决定前景像素不被用来更新模型。因此 x 位置的选取不是完全随机寻找的。移除面积小于等于 10 (像素) 的前景斑点区域, 填充面积小于等于 20 的前景斑点区域; 贴近边缘的斑点, 无论尺寸全部保留, 使其保持原状, 填充面积小于等于 50 (像素) 的前景空洞区域, 该操作用来限制散布在前景物体中的错误背景。

(2) 适用于动态视频场景中的静止物体的保留。

在背景模型更新策略中，通过计算像素点的梯度，根据梯度大小，确定是否需要更新邻域。梯度值越大，说明像素值变化越大，说明该像素值可能为前景，不应该更新。

(3) 检测闪烁像素点，并对其进行抑制。

引入闪烁程度的概念，在 ViBe+算法中，建立一个闪烁等级图表[4]，在背景模型更新策略中，当一个像素点在前一帧认为背景像素，在当前帧被认为前景像素时，改像素值得闪烁等级增加 15，否则减少 1，然后根据闪烁等级的大小判断该像素点是否为闪烁点。如果一个像素的闪烁等级大于等于 30，则该像素被认为正在闪烁，则不更新这个点。换言之，这种技术增强了我们算法对于多峰背景分布的处理。

(4) 增加更新因子

ViBe 算法中，默认的更新因子是 16，当背景变化很快时，背景模型无法快速的更新，将会导致前景检测的较多的错误。因而，需要根据背景变化快慢程度，调整更新因子的大小，可将更新因子分多个等级，如 $rate = 16, rate = 5, rate = 1$ 。

4.2.3 构建虚拟背景图像模型

虚拟背景[5]的创建，是为了更好的抑制鬼影的存在，因 ViBe+算法，当第一帧出现前景目标的时候，接下来的十多帧中会出现连续的鬼影，因此，本模型采用对监控视频 $P(x) = \{f_1, f_2, \dots, f_n\}$ 前 N 帧去平均的方式，构建出一张虚拟的没有前景目标的背景图像作为第 0 帧 f_0 ，其中 x 表示帧的索引数， f_x 表示第 x 帧，虚拟背景 f_0 的具体描述如下式：

$$f_0 = \frac{f_1 + f_2 + \dots + f_N}{N} \quad (4-6)$$

将 f_0 加入视频中，可以很好的解决了鬼影问题的存在。

4.2.4 模型求解步骤

(1) 高斯滤波。

针对问题二，高斯滤波起到了十分重要的作用，如树叶、水波等动态背景图，通过高斯滤波的方式，对每一帧进行模糊，减弱光亮对模型的影响程度。

(2) 构建虚拟背景图像。

通过添加虚拟背景的方式，有效的抑制鬼影的出现，保证了前景目标检测的效果。提取后的效果图如下图(4-2)所示：



图 4-2 左为视频第一帧，图右为构建出的虚拟背景图，作为第 0 帧

(3) ViBe+算法提取前景目标。

添加虚拟背景后，通过 ViBe+算法，提取监控视频中的前景目标，按帧顺序存储。






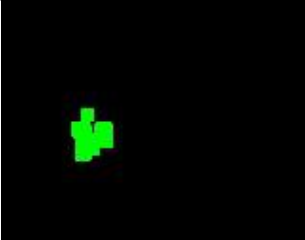






(4) 去除小连通域。

接着通过去除小连通域的方法降低动态背景噪声，并生成提取后视频。

4.3 实验结果与分析

4.3.1 实验结果

本节针对附件二中的动态视频，以及附件三中的部分视频进行实现，每个监控视频分别进行了三帧的展示，分别为（（视频 campus 的 422、1026、1390 帧）、（视频 curtain 的 1775、1829、1842 帧）、（视频 fountain 的 414、442、523 帧）、（视频 watersurface 的 10、19、28 帧）），并与高斯混合模型进行了对比，结果如下表(4-1)所示：

原图	ViBe+	高斯混合模型
		
		
		
		

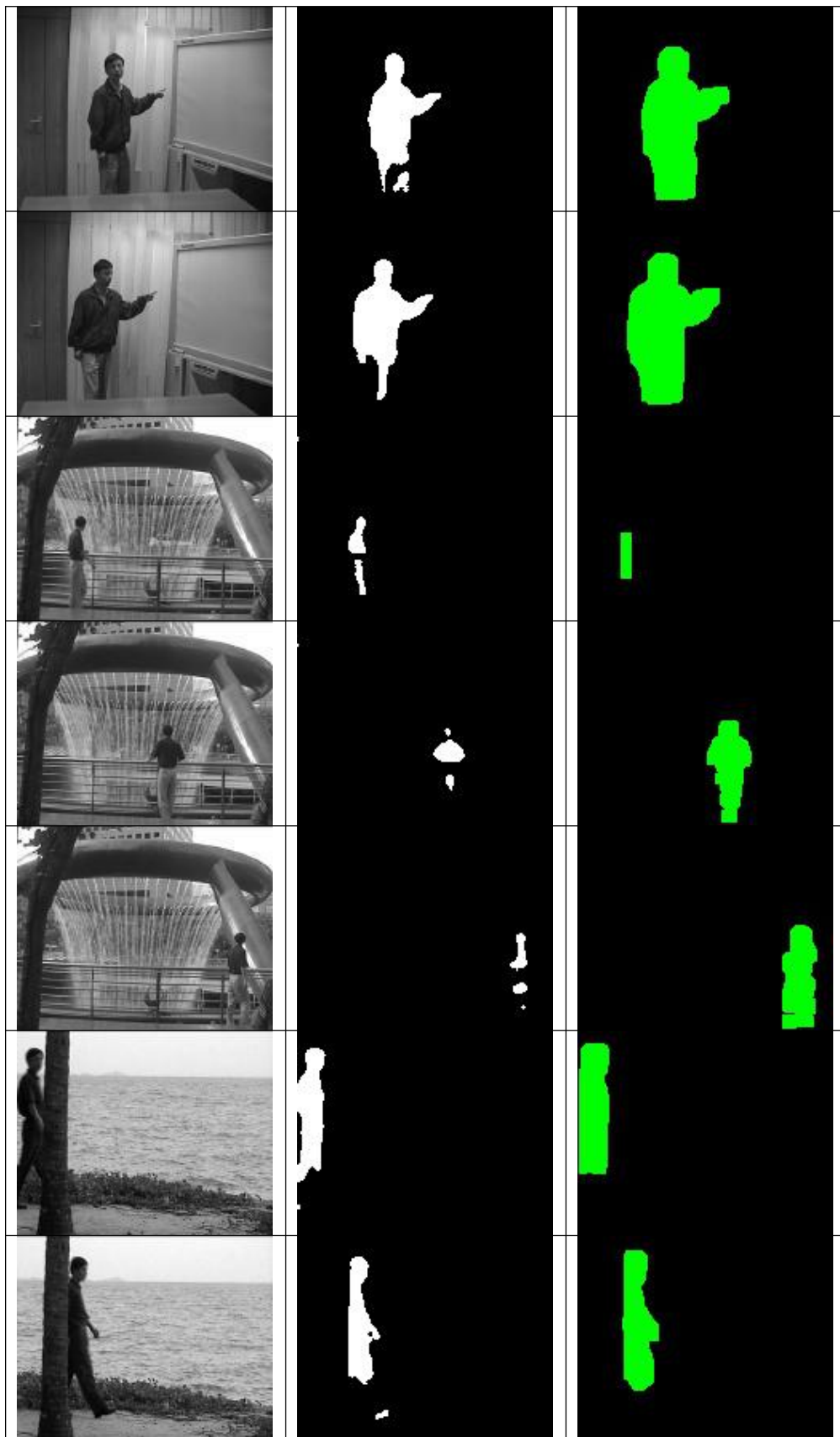




表 4-1

4.3.2 结果分析

通过上表，可以看出应用 ViBe+算法，相比于高斯混合算法，可以准确提取监控视频的前景目标，并抑制噪声的产生，并且所提取的前景目标具有部分轮廓信息，有效的抑制了前景目标的错误识别率，但对于像素值相似的地方，没有高斯模型精确，这也是高斯模型对光较为敏感的有点所在（图第三个视频所示），其他地方均可以准确的检测。

5 问题三的建立和求解

5.1 问题分析

针对问题三，对于背景晃动视频的前景目标提取，视频的晃动在短时间内可近似视为一种线性仿射变换，如旋转、平移、尺度变化等。对于这种类型的视频，首先要进行视频矫正，及通过仿射变换加配准的方式，对每一帧的图像进行提取 SIFT 关键点[6]，接着将 SIFT 关键点进行配准，为了起到很好的防抖效果，本文运用隔帧配准的方式，将整个视频进行配准。接着运用 ViBe+模型对配准后的视频进行前景目标提取。

5.2 模型建立及求解

5.2.1 基于 SIFT 特征点的隔帧匹配模型建立。

SIFT (Scale-invariant feature transform) 是一种检测局部特征的算法，该算法通过求一幅图中的特征点 (interest points, or corner points) 及其有关 scale 和 orientation 的描述子得到特征并进行图像特征点匹配，获得了良好效果，详细解析如下：

SIFT 特征不只具有尺度不变性，即使改变旋转角度，图像亮度或拍摄视角，仍然能够得到好的检测效果。整个算法分为以下几个部分：

(1) 构建尺度空间

这是一个初始化操作，尺度空间理论目的是模拟图像数据的多尺度特征。高斯卷积核是实现尺度变换的唯一线性核，于是一副二维图像的尺度空间定义为：

$$L(x, y, \sigma) = G(x, y, \sigma) \times I(x, y)$$

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \quad (5-1)$$

其中 $G(x, y, \sigma)$ 是尺度可变高斯函数。

(x, y) 是空间坐标，是尺度坐标。 σ 大小决定图像的平滑程度，大尺度对应图像的概貌特征，小尺度对应图像的细节特征。大的 σ 值对应粗糙尺度(低分辨率)，反之，对应精细尺度(高分辨率)。为了有效的在尺度空间检测到稳定的关键点，提出了高斯差分尺度空间 (DOG scale-space)。利用不同尺度的高斯差分核与图像卷积生成。

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) \times I(x, y)$$

$$= L(x, y, k\sigma) - L(x, y, \sigma) \quad (5-2)$$

(2) 在不同尺度空间中寻找特征点进行描述，最后匹配是 SIFT 算法的核心。因此可以将 SIFT 算法概括成以下几个步骤：

- a. 搜索特征点；
- b. 对提取的特征点进行详细的描述；
- c. 对关键点进行匹配。

(2.1) 搜索特征点。

根据尺度空间理论，建立高斯金字塔与高斯差分金字塔，将高斯差分金字塔中的每个像素与它同尺度的 8 个相邻点和上下相邻尺度对应的 9×2 个点共 26 个点比较，得到待修正的特征点集。通过特征点精确定位和去除边缘响应 2 个环节，得到修正的特征点集。

(2.2) 特征点描述。

对特征点的邻域进行梯度计算，使用直方图将 0° 到 360° 分为 36 个柱，每 10° 一个柱，柱高为对应梯度方向的累加值。直方图的峰值即为该特征点的主方向。以特征点为中心取 16×16 像素为临域，再将该邻域分为 4×4 个子区域，在每个子区域计算 8 个梯度方向的直方图。

(2.3) 隔帧特征点匹配。

对特征点集采用 KD 树[7]进行隔帧匹配搜索，分别得到最邻近匹配对与次临近匹配对，将最邻近欧式距离与次临近欧式距离进行相比，若比值小于某一设定阈值则认为匹配正确。最后采用 RANSC(random sample consensus)进行进一步提纯，得到最终的特征点匹配对。这样做可以有效的抑制视频抖动现象的发生。

5.2.2 模型求解步骤

(1) 高斯滤波。

首先通过高斯滤波的方法，对图像进行降噪处理，这样可以降低下一步中 SIFT 特征点检测的难度，使得一些视频中的闪烁点不会被检测为特征点进行匹配，并输出每一帧。

(2) 提取 SIFT 特征点

接着，对滤波后的图片进行特征点检测，通过定义阈值，检测出每一帧上的前 N 个关键点。

(3) 隔帧特征点匹配。

这一步是问题三防抖的关键，通过隔一帧进行特征点匹配，进行放射变换和

配准的过程，优化视频的连续性，并对每一帧仿射变换后的图片进行边框处理，去掉因仿射变换而产生的扭曲边界。

(4) 提取前景目标。

运用问题二所述的改进的 ViBe+模型，提取前景目标。


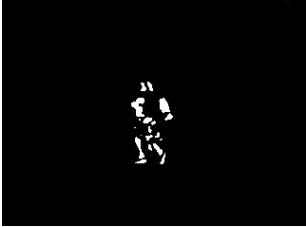


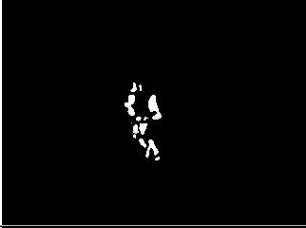





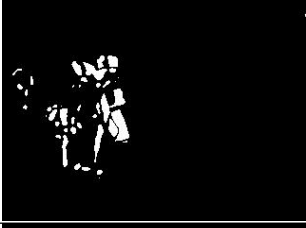




(5) 去除小连通域。

最后通过去除小连通区域的方式，对视频进行降噪处理，生成提取前景目标后的模型。

5.3 实验结果与分析

5.3.1 实验结果

本小节针对附加二中抖动视频的前景目标识别进行了实现和展示，展示帧数为（视频 1 的（11、27、39 帧）、视频 2 的（2、18、39 帧）、视频 car6 的（4、12、27 帧）视频 car7 的（7、12、20 帧）），并与不进行防抖的 ViBe+模型进行了对比，效果结果如下表（5-1）所示

原图	Sift 特征匹配加 ViBe+模型	ViBe+模型
		
		
		
		
		

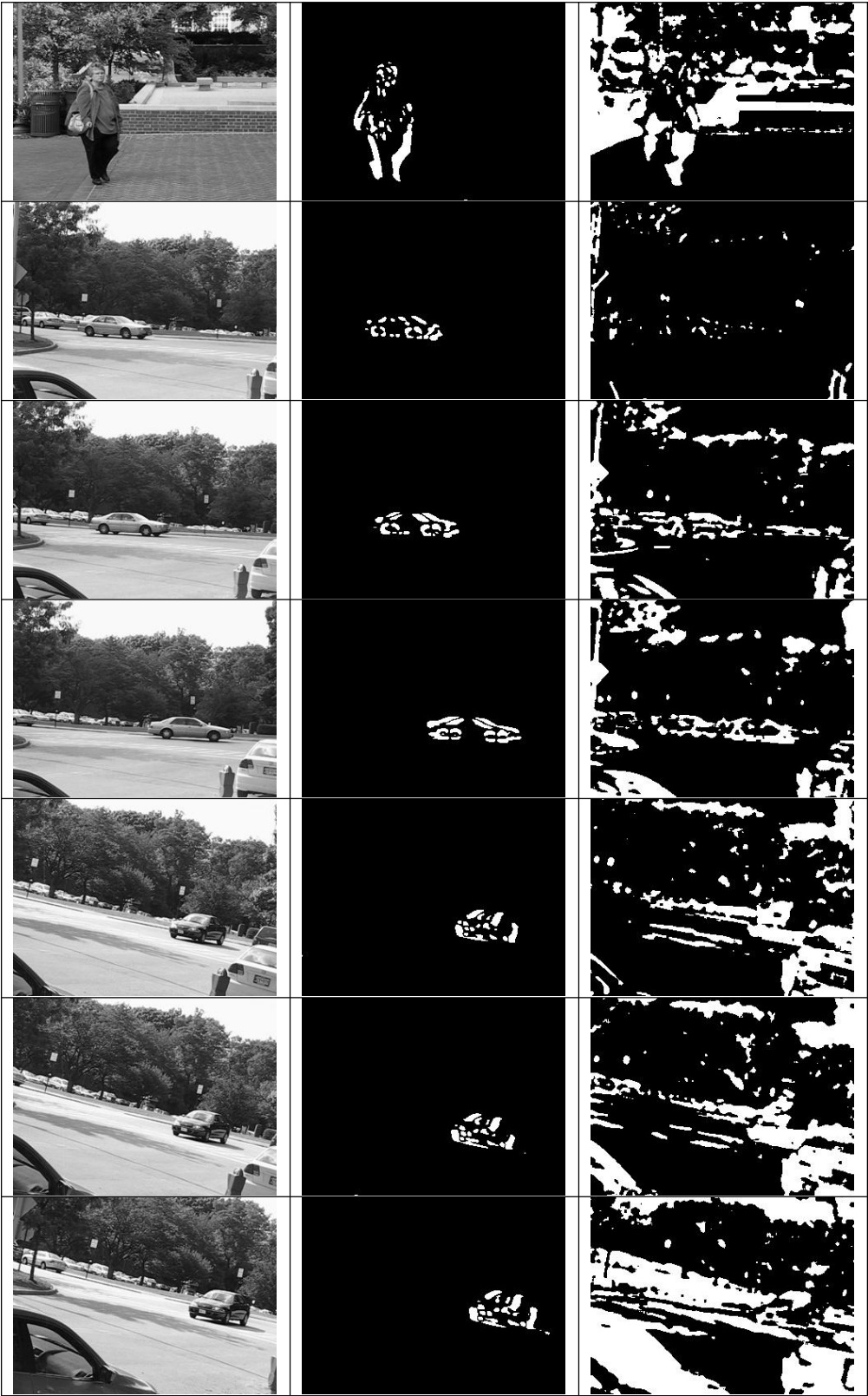


表 5-1

5.3.2 结果分析

从对比图中可以明确看出，对于不是通过 SIFT 进行特征点匹配的 ViBe+模型，对抖动视频的前景目标提取，效果十分不好，很多背景噪声因视频的抖动而

被检测出来。使得前景目标被淹没于噪声中无法正确提取。经过 SIFT 特征比配后的 ViBe+模型，有效的抑制了噪声，可以清晰的找出前景目标的位置，并且具有较为详细的目标轮廓。

6 问题四的分析及模型建立及求解

6.1 问题分析

针对问题四，从一组时长较大的视频中选出包含显著前景目标的视频帧标号。我们对通过问题二模型进行前景提取后的二值化图像进行分析，对每帧图像的前景目标所占像素比例进行量化，并按帧序构建变化折线图。再按照一定的量化标准将跳变物体所在帧号进行标记，并在折线图上将其所在帧的前景目标像素比例数据置零，再利用高斯平滑滤波，将折线平滑滤波，对平滑后的前景比例曲线图进行波形分析。检测曲线图的波峰，将曲线图中符合标准的凸起，标记为显著前景目标所在的帧区间。

6.2 模型建立及求解

6.2.1 模型建立

(1) 使用 ViBe+方法对视频场景进行前景目标提取，得到所有视频帧提取前景目标后的的二值化图像

$$\{V_f(x)|x \in \text{Frame}_i\} \quad (6-1)$$

(2) 对每一帧的二值化图像进行像素统计，将每一帧的前景目标像素表示为 $pixel \in \text{ForeGround}$ ，对其进行数量统计，因此属于第 i 帧前景目标的像素数量为

$$\sum G(i, pixel), \quad pixel \in \text{ForeGround} \quad (6-2)$$

(3) 将对显著前景目标的检测问题模型化为对前景目标像素比例的变化进行量化分析，前景目标像素比例表示为：

$$P(i) = \frac{\sum G(i, pixel)}{\text{width} \times \text{height}}, \quad pixel \in \text{ForeGround} \quad (6-3)$$

其中， $G(i, pixel)$ 表示该像素属于第 i 帧的前景目标区域。

由于，跳变物体的出现，是出现在某一帧的图像里，在前景目标像素比例中，表现成一个在某一帧上的一个数据阶跃；而显著前景目标的出现，伴随一个从无到有的过程，从而在前景目标像素比例变化中，表现成一个有明显波峰的或有重叠波峰的数据变化凸起。

因此，对显著前景目标的检测，可以归纳成是对前景目标像素比例变化曲线的波形分析检测问题。

6.2.2 模型求解

(1) 用 ViBe+方法对视频场景进行前景目标提取，得到前景目标分离的二值化图像，按帧播放时序，统计出每帧图像的前景目标像素比例，并以帧标号为横轴，前景目标像素比例为纵轴构建出变化折线图。

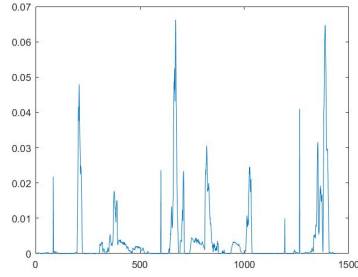


图 6-1

(2) 对变化折线图进行波峰检测，再根据公式 (6-4) 量化标准，找到跳变点，即为检测到的跳变物体出现帧数，将跳变物体出现的帧数进行记录。

$$\begin{aligned} |P(\text{left}) - P(\text{peak})| &> \alpha \times P(\text{peak}) \quad \text{s.t.} \quad \text{left} = \text{peak} - 1 \\ |P(\text{right}) - P(\text{peak})| &> \alpha \times P(\text{peak}) \quad \text{s.t.} \quad \text{right} = \text{peak} + 1 \end{aligned} \quad (6-4)$$

其中 peak 为检测到的波峰点， left 、 right 分别是 peak 的左右邻域。再把跳变帧的前景目标像素比例置零，以更大程度的消除干扰。

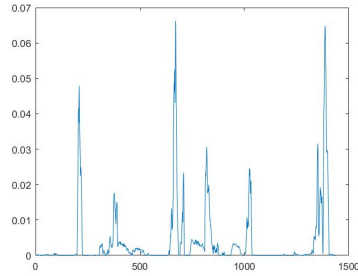


图 6-2

(3) 对变化折线图进行高斯平滑滤波。高斯滤波器是一类根据高斯函数的形状来选择权值的线性平滑滤波器。高斯平滑滤波器对于抑制服从正态分布的噪声非常有效。一维零均值高斯函数为：

$$G(X) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \quad (6-5)$$

其中，高斯分布参数 Sigma 决定了高斯函数的宽度。通过对折线图进行高斯平滑滤波，抑制了一定程度噪声的干扰，平滑了曲线，减轻了波形的突触，更有利于波峰的检测。在波形处理中，我们选取的高斯滤波器方差为 10，卷积核长度为 30。

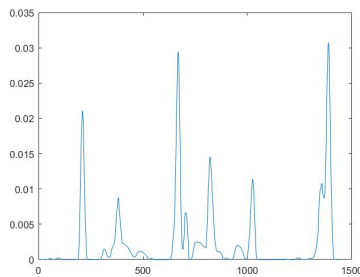


图 6-3

(4) 得到的平滑后的曲线图进行波峰检测，按照公式 (6-6)，标记符合变化幅度的波凸起区间，即为显著前景目标出现的帧区间。

$$\begin{aligned} |P(\text{left})-P(\text{peak})| &> \alpha \times P(\text{peak}) \quad \text{s.t.} \quad \text{left} > \text{peak}-1 \\ |P(\text{right})-P(\text{peak})| &> \alpha \times P(\text{peak}) \quad \text{s.t.} \quad \text{right} < \text{peak}+1 \end{aligned} \quad (6-6)$$

其中，Left 和 right 为显著目标出现的区间左右边界。

(5) 对出现的重合区间进行区间合并，因为视频的关联性，可将连续出现的前景目标归为一个帧数范围区间内。

6.3 实验结果与分析

(1) Campus:

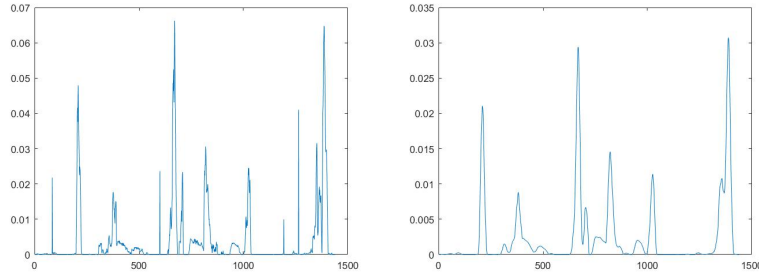


图 6-4

结果展示:

跳变物体目标出现在 85 帧、600 帧、1193 帧和 1264 帧。

显著前景目标出现在 197-224 帧、302-520 帧、635-682 帧、695-716 帧、737-890 帧、933-985 帧、1007-1039 帧、1233-1253 帧、1324-1342 帧。

实际显著前景目标出现在 200-224 帧、305-522 帧、641-683 帧、689-712 帧、736-907 帧、1003-1037 帧、1324-1417 帧。

(2) Curtain:

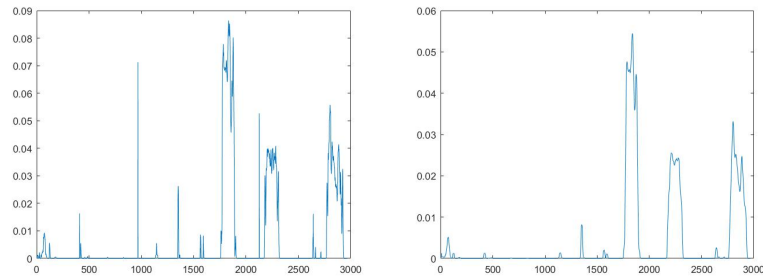


图 6-5

结果展示:

跳变物体目标出现在 411 帧、967 帧、2126 帧。

显著前景目标出现在 1342-1362 帧、1765-1893 帧、2171-2318 帧、2764-2932 帧。

实际显著前景目标出现在 1758-1903 帧、2172-2316 帧、2766-2933 帧。

(3) Escalator

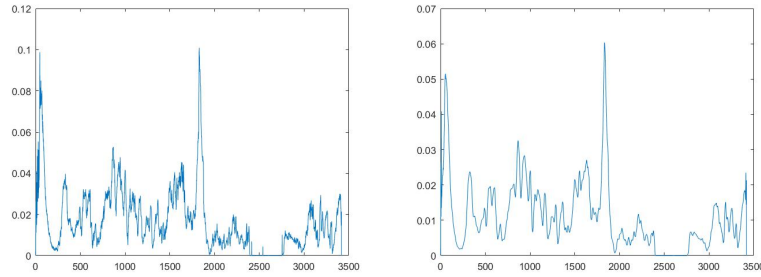


图 6-6

结果展示：

跳变物体目标出现在 2415、2539、2754 帧。

显著前景目标出现在 3-1928 帧、1955-2394 帧、2769-3417 帧。

实际显著前景目标出现在 3-2393 帧、2764-3417 帧。

误差分析：

模型将视频的前半部分归为两个较大的前景目标出现区间，由于前半部分人群是不间断出现的。

(4) Fountain

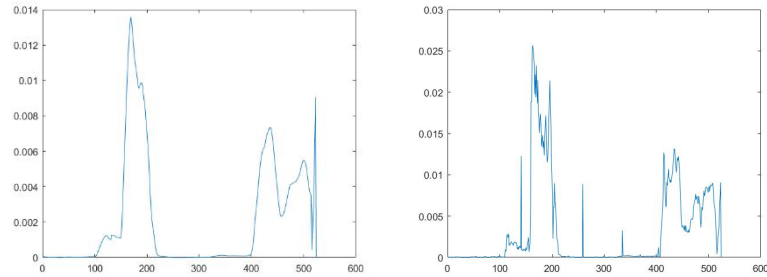


图 6-7

结果展示：

跳变物体目标出现在 259 帧、335 帧。

显著前景目标出现在 136-209 帧、402-523 帧。

实际显著前景目标出现在 154-215 帧、402-523 帧。

(5) hall

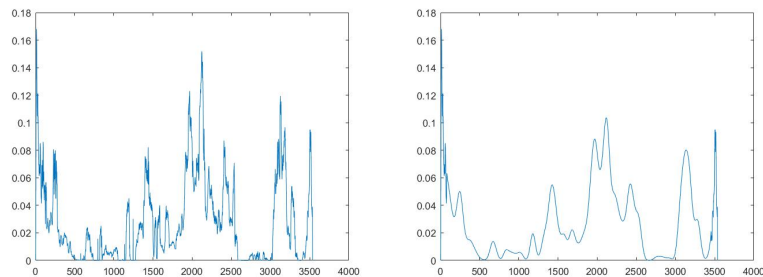


图 6-8

结果展示：

跳变物体目标出现在 578 帧、795 帧、1246 帧。

显著前景目标出现在 5-504 帧、601-746 帧、789-1063 帧、1114-2604、2732-3343 帧、3367-3534 帧。

实际显著前景目标出现在 1-513 帧、598-750 帧、816-1057 帧、1152-1212 帧、1279-3355 帧、3401-3534 帧。

(6) lobby

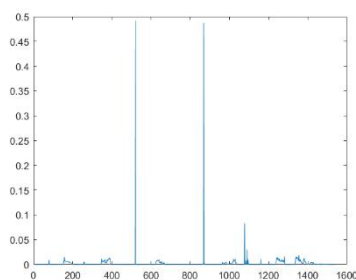


图 6-9

结果展示:

跳变物体目标出现在 79 帧、259 帧、521 帧、870 帧、1079 帧、1161 帧。

显著前景目标出现在 144-197 帧、336-405 帧、613-670 帧、956-1050 帧、1229-1291 帧、1327-1444 帧。

实际显著前景目标出现在 153-197 帧、344-395 帧、621-672 帧、959-1047 帧、1237-1286 帧、1332-1538 帧。

误差分析:

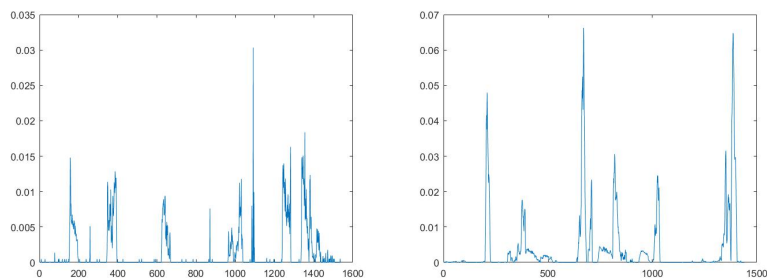


图 6-10

从原始变化折线图可以看出，521 和 870 帧出的两次开灯对波形有很大的影响，在对 6 处跳变点进行置零后，得到了置零后的折线图，相比原始图，变化率有了很大程度的改进，因此可以反映置零操作的必要性。再对折线图进行高斯平滑滤波，得到上图。

由于，该 lobby 视频在 521、870、1079 时分别进行了几次开关灯操作，对图像的灰度有较大的影响，因此模型判定上述帧出现了跳变物体目标。由于在检测波形波峰时模型所取的判定步长较大，因此在 1092 时的开灯操作和 449 时的关灯操作被判定模型漏检了。

(7) office

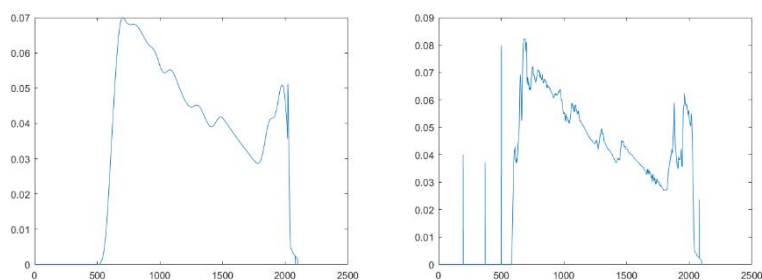


图 6-11

结果展示:

跳变物体目标出现在 197 帧、372 帧、501 帧、2080 帧。

显著前景目标出现在 556-2041 帧。
实际显著前景目标出现在 579-2041 帧。

(8) overpass

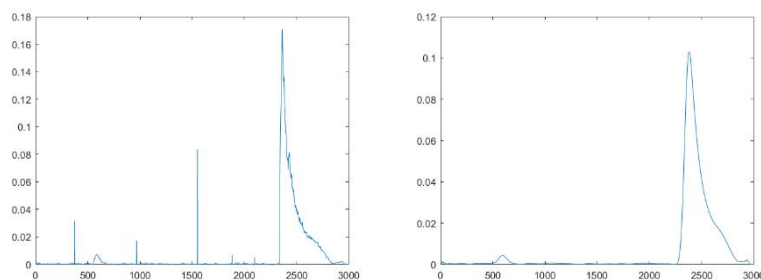


图 6-12

结果展示：

跳变物体目标出现在 374 帧、968 帧、1551 帧。
显著前景目标出现在 2295-2842 帧。
实际显著前景目标出现在 2335-2967 帧。

6.4 误差分析

下图为各视频显著目标出现帧的边界对比。左栏为原视频的目标出现帧，中间栏为模型判定的目标出现边界帧。







		视频边界帧为 2766 标记边界帧为 2764
		视频边界帧为 2336 标记边界帧为 2295
		视频边界帧为 579 标记边界帧为 556

图 6-13

通过前面真实显著目标帧和检测目标帧的对比发现，在前景目标提取较好的视频中，判定最大误差在 20 帧左右。判定的目标出现帧范围比真实情况区间大的原因是，在高斯平滑滤波后，前景像素比例曲线的波形变缓变宽，在模型量化时，对变化率的判定也出现了一定偏差。因此对于参数的设置，以及阈值的设置，应该进一步通过优化目标函数进行优化调节，已产生更为精准的结果。

另外，在对 lobby、Escalator 的判定中，由于图像的光照条件变化很明显，模型对于光照变化的反应很尖锐，对于变化波形图也产生了很频繁的突触，因此

对实验结果影响较大。

7 问题 5 的分析及模型建立及求解

7.1 问题分析

针对问题 5 的应用环境，可看出存在如下问题和困难：

(1) 无法获得相机的标定数据，也就意味着无法对该问题添加关于相机的强约束，从而无法准确的定位目标的空间位置。

(2) 目标图像质量较差，含有噪声，且不能保证目标点在同一个平面内运动。针对上述问题，我们建立了基于特征点检测的多视角匹配模型，我们以视频中的特征点作为观测，利用两摄像机师徒之间的对极关系建立匹配关系。下文当中将对对极几何，特征点检测以及基于此的目标匹配分别进行介绍。

7.2 模型建立及求解

7.2.1 基于对极几何模型约束下的 SIFT 特征点匹配模型建立。

(1) 对极几何模型

对极几何[8]是两幅师徒之间内在的射影几何。它独立于景物结构，只依赖于摄像机的相对姿态。基础矩阵 F 则集中体现了对极几何的关键作用。 F 矩阵描述的是从不同拍摄点拍摄同一场景的图像时两图中点的对应关系，它还提供有关位置和摄像机参数的信息。

对极几何描述的是场景中的三维点和一对图像点之间的关系。给定一个场景中三维点 P 被映射到 I 和 I_0 两图像中，这三个点就构成一对极几何（图 1）。点 C 和 C_0 表示分别图像 I 和 I_0 相机中心。三点 P 、 C 和 C_0 组成平面，这就是所谓的极平面。每个图像平面与极平面相交，称为极线；一点投影在一个图像会有一个对应点位于其他图像平面的极线上（极线约束）。这个约束是指当我们有两个图像 I 和 I_0 ，和一点 P 在 I 上，如果我们要寻找在 I_0 上对应点 P_0 ，仅沿极线搜索就可以了。假设 I 平面上的点 $P = (u, v)^T$ 对图像和 I_0 平面上的点 $P' = (u', v')^T$ 都是 P 的投影点，增加了齐次坐标的二维图像坐标，这样笛卡尔坐标下 P 和 P' 旋转为射影坐标 $m = (u, v, 1)^T$ 和 $m' = (u', v', 1)^T$ 。采用这种画法极线约束表现为以下形式：

$$m'^T F m = 0 \quad (7-1)$$

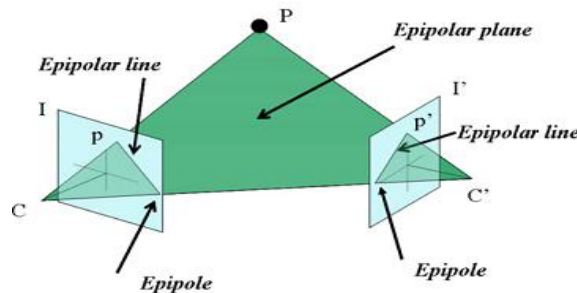


图 7-1 对极几何

在两幅图像之间， F 矩阵将点 m 映射为对应的对极线[9]，将对极点映射为

0。不能提供对应点间的一一对应。F 是一个 3x3 的矩阵。F 矩阵的秩等于 2 的约束，这是由于所有极线必须通过图像的极点。

性质：

a.对极线： $p' = Fm$ 是对应点 x 的对极线， $p = F^T m'$ 是对应极点 m' 的对极线。

b.对极点： $Fe = 0, F^T e' = 0$

(2) 基础矩阵[10]的计算

对于给定图像时，常用八点法来线性的估计 F 矩阵。

对应的点 (u_i, v_i) 和 (u'_i, v'_i) ，由线性系统得到方程：

$$Af = (a_1 a_2 \dots a_n)^T f = 0 \quad (7-2)$$

其中 $a_i = (u_i u'_i, v_i u'_i, u'_i, u_i v'_i, v_i v'_i, v'_i, u_i, v_i, 1)^T$ ， $f = (f_{11}, f_{12}, f_{13}, f_{21}, f_{22}, f_{23}, f_{31}, f_{32}, f_{33})^T$

其中 f 由 F 矩阵获得：

$$F = \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{pmatrix} \quad (7-3)$$

同时有约束条件 $|f| = (f_{11})^2 + (f_{12})^2 + \dots + (f_{33})^2 = 1$ 需要获得至少八个对，我们才能求解出 (2) 的估计矩阵。在刚才的过程中我们没有考虑“矩阵秩等于 2”的约束。为了满足这一约束，选取满足约束和 $|F - F'|$ 的 Frobenius 范数最小的 F' 作为 F 矩阵。具体的方法如下：

首先，对没有秩等于 2 约束条件下的矩阵进行奇异值分解 (SVD) 如下：

$$F = U \Sigma V^T = U \begin{pmatrix} \sigma_1 & & \\ & \sigma_2 & \\ & & \sigma_3 \end{pmatrix} V^T$$

$$\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq 0 \quad (7-4)$$

如果有在上述过程中无噪音，第三个奇异值， σ_3 应该是 0。因此，以下矩阵 F' 可以取代作为约束矩阵的秩等于 2：

$$F' = U \begin{pmatrix} \sigma_1 & & \\ & \sigma_2 & \\ & & 0 \end{pmatrix} V^T \quad (7-5)$$

这个重构矩阵 F' 可以作为 F 矩阵。这个过程实现了估计矩阵的秩为 2 的约束，但它引入附加噪声。

为了解决这个问题，大多数工程在下一步采用非线性优化方法。由于对应点必须位于极线的约束，我们可以设计下面的几何成本函数[11]：

$$\min_{F'} \sum (d^2(m', F'm) + d^2(m, F'^T m')) \quad (7-6)$$

$d(m', F'm)$ 表示极线上点 P_0 和对应点 P 在图像的坐标系下的距离的平方。

(3) 提取 SIFT 特征点并根据对极约束关系[12]进行特征点的匹配，特征点提取及匹配在问题三的解决中进行了详细描述。

7.2.2 模型求解

(1) 检测 SIFT 特征点并进行对极匹配

本节采用基于 SIFT 特征点检测的对极匹配方法，对不同角度的视频进行匹配求解，本节讨论如何根据对极约束关系实现不同摄像机视图中的特征点检测结果的匹配，SIFT 特征点的提取在对问题三的解决中已经进行了介绍。

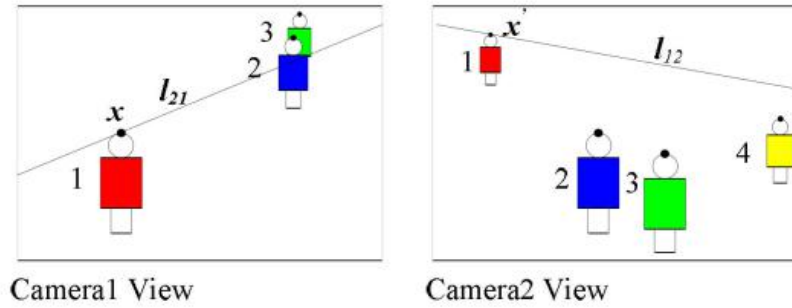


图 7-2 对极关系示意图

根据对极几何，当两个摄像机观测的目标为同一目标时，目标在图像中所处的位置应该落在相应的对极线上。如上图所示，目标 1 在摄像机 1 中的观测值 x 和视图 2 中的观测值 x' ，它们分别落在对应的极线 l_{21} 和 l_{12} 上。由于观测误差，对应的观测点一般不会准确的落在极线上，但是一般都会在对极线的附近。在本文中，我们假设观测点与对极线的距离满足正态分布 $N(0, s)$ ，其中 σ^2 与目标面积成正比。定义观测值间的匹配权值 w 为观测 x 和 x' 源自同一个空间目标 X 的概率，即 X 的投影出现在 x 和 x' 时的概率，假设 X 出现在 x 和 x' 时的概率相互独立，有：

$$w = P(x, x' | X) = P(x | X)P(x' | X) \quad (7-7)$$

$$w \approx P(x | l_{21})P(x' | l_{12}) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{1}{2}(\frac{d}{\sigma})^2) \frac{1}{\sqrt{2\pi}\sigma'} \exp(-\frac{1}{2}(\frac{d'}{\sigma'})^2) \quad (7-8)$$

简化得到：

$$w \propto \frac{1}{\sigma\sigma'} \exp(-\frac{1}{2}((\frac{d}{\sigma})^2 + (\frac{d'}{\sigma'})^2)) \quad (7-9)$$

其中： $d = |X^T l_{21}| / \|l_{21}\|$ ， $d' = |X'^T l_{12}| / \|l_{12}\|$ 为观测点到对极线的距离， x

和 x' 和 l_{12} 、 l_{21} 分别为观测点和相应的对极线的列矢量， $l_{12} = F_{12}x$ ， $l_{21} = F_{12}^T x'$ 。

σ^2 和 σ'^2 分别设置为前景运动目标的面积。当视图中存在多个观测点时，我们利用权值匹配 w 建立匹配矩阵 W 。这样，匹配的问题就转化为在 W 中寻找一组权值和最大的最优匹配组合。

(2) 选取最优匹配组合[13]的选取

在实际应用中，权值和最大的最优匹配组合常常极不稳定。如当 c 个目标距离较近是，会有 $c!$ 个匹配组合的权值比较接近。由于噪声的影响，最优匹配组合往往会出现错误。对于这种情况，我们对于最优组合，逐一的隔离掉组合中的一项，重新选择最优组合。如果去掉该项后权值发生较大变化，说明没有其他合理的匹配存在，就认为该项是稳定的，则可以接受该项。

7.3 实验结果与分析

7.3.1 实验结果

针对问题五，实验结果如下：


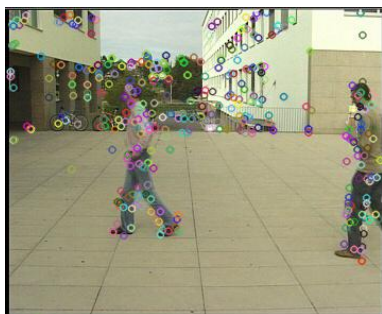

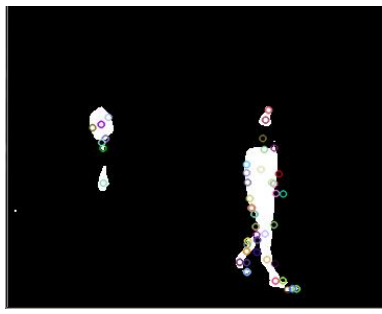
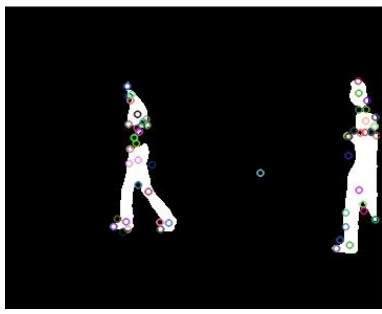

场景 1 特征点	场景 2 特征点	场景 3 特征点
		
场景 1 前景提取特征点	场景 2 前景提取特征点	场景 3 前景提取特征点
		

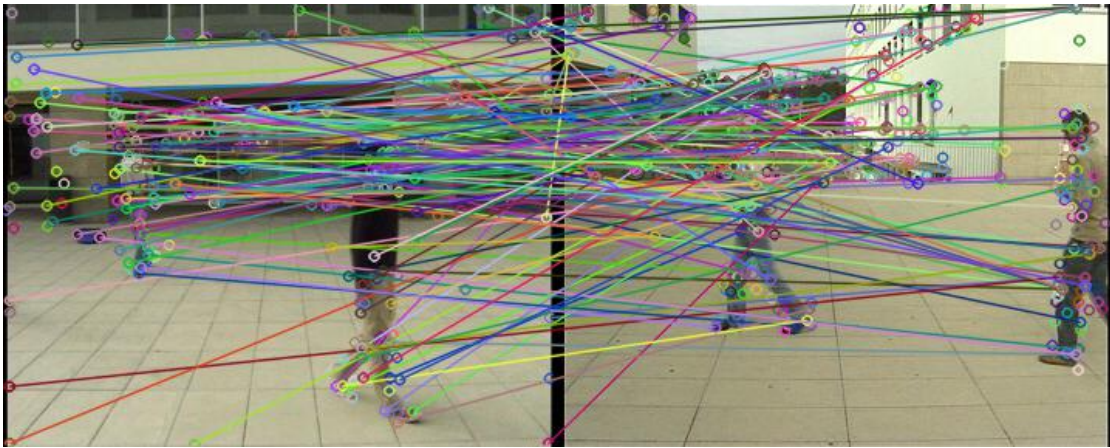
图 7-3

以下实验表明了多视角下的匹配结果，并用上文中提到的策略剔除坏点进行重新匹配。

场景 1 和场景 2 的所有特征点匹配结果



场景 1 和场景 2 剔除坏点之后的匹配结果



场景 1 和场景 2 的前景提取所有特征点匹配结果

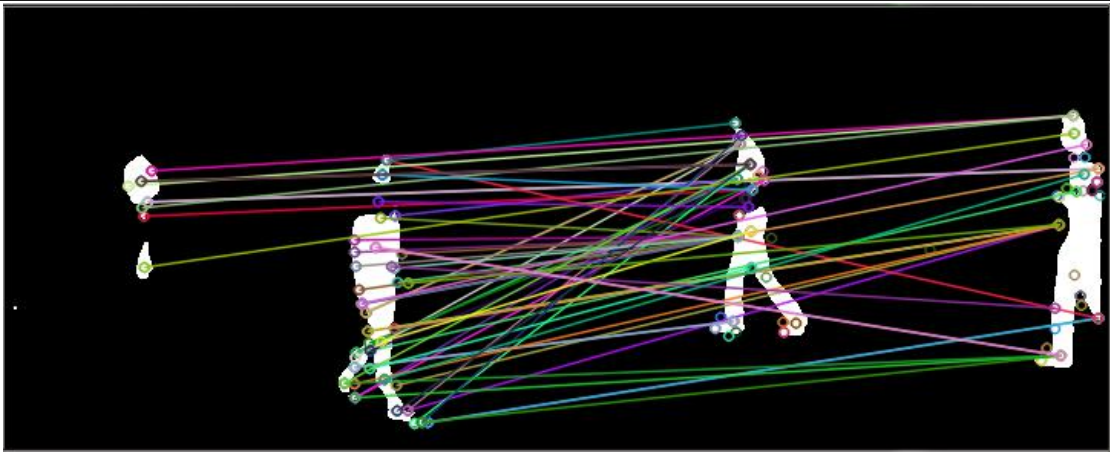


图 7-4

场景 1 和场景 3 的所有特征点匹配结果



场景 1 和场景 3 剔除坏点之后的匹配结果



场景 1 和场景 3 的前景提取所有特征点匹配结果

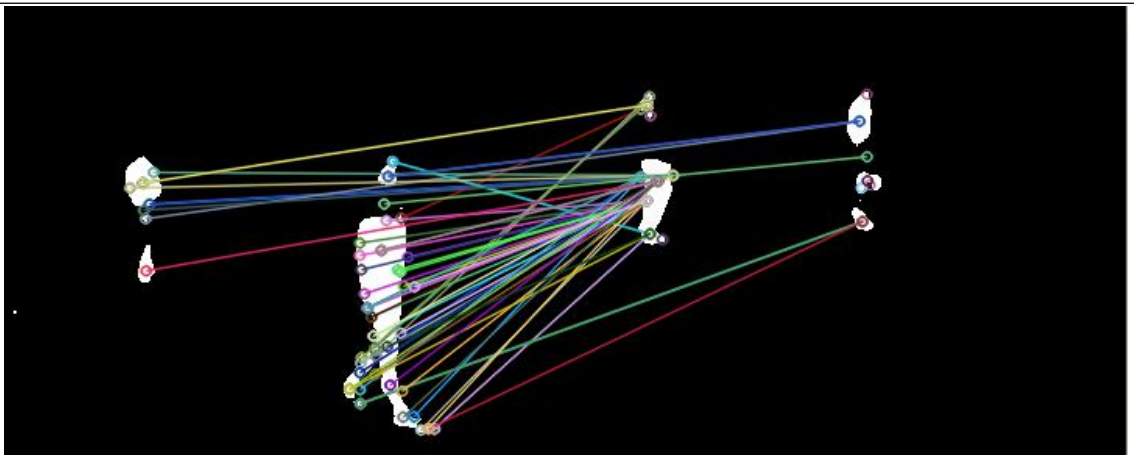


图 7-5

7.3.2 结果分析

通过上述结果，可以看出，在对极几何约束下，提取的大多数 SIFT 特征点进行了正确的匹配，剔除坏点的操作，提出了绝大多数没前景目标的关键点，本题模型可以有效的完成多视角匹配问题，可以从视频中识别多角度中的同一个前景目标。

8 问题六的分析及模型建立及求解

8.1 问题分析

在对公共区域的监控中，由于危险类型的不可预知性和人群移动的复杂性，各种各样的人群异常事件都有可能发生。人群的异常事件又可以分为个体异常事件和群体异常事件，其中，个体异常事件是指人群中某些个体或目标的行为迥异于其他个体的行为，如步行街上正在移动的汽车。而群体异常事件是指监控区域中的多个个体的行为不同于之前群体的行为，如人群恐慌、道路上的群体聚集。目前已经公开多个用于人群异常事件检测的视频数据库，比较常用的数据库包括 UCSD 数据库[1],UMN 数据库[2], subway surveillance 数据库[3]等。本文对群体异常行为检测采用了基于群体特征的 SVM 分类算法。主要分为以下几个步骤：

(1) 根据人类群体运动的视频中存在时间和空间方向上剧烈变化的位置的特点及时空特征点方法在简单人体行为识别中取得的成功，提出了用时空特征点来描述人类群体的运动，选择了鲁棒方法 Gabor 小波函数方法用来提取时空特征点，通过实验验证了这种方法可以有效的解决人类群体异常行为检测问题。

(2) 利用高斯混合模型对正常行为的特征点集建立模型，为每个关键词分配不同的概率权重，并且准确的描述了不同类别出现的概率和输入的时空特征属于各个类别的概率。

(3) 为正常行为以及异常行为中的每个视频片段建立视频向量，将生成的视频向量对 SVM 进行训练以及学习，输入已知类别的测试视频，利用已训练好的 SVM 对视频进行群体异常行为的测试，根据已知数据和得出的结果分析研究，得出 SVM 的识别率，并根据实际的结果对 SVM 进行参数调整和进一步的完善。对已经完善的 SVM 分类器进行未知视频的预测，用于检测群体行为是正常或是异常。

8.2 模型原理

8.2.1 基于时空特征的运动特征提取和聚类建模

对人体运动进行特征提取是行为识别的关键。人体运动特征存在于视频图像中时间和空间方向像素值发生剧烈变化的位置，因为它们往往包含了丰富的信息，同时也具有较强的稳定性。通过离散的特征点捕获表征这些位置就可以有效的描述人体行为，通常这些点我们称作为特征点。

8.2.2 基于高斯和 Gabor 小波函数提取时空特征点

Dollár 等人提出任何进行非匀速运动的人体区域都可以看作是时空特征点 [14]，使用简单的检测器也可以获得稠密的时空特征点。在这种方法中，通过计算视频流时空中每一个像素点处的二维空间高斯和一维时间 Gabor 小波函数响应值来确定时空特征点，若视频三维空间中某点响应值为局部极值且大于某一阈值，则认为该极值点为时空特征点。

响应函数定义为：

$$R = (I * g * h_{ev})^2 + (I * g * h_{od})^2 \quad (8-1)$$

其中, I 为视频流图像, $g(x, y; \sigma)$ 为二维空域高斯平滑滤波器, h_{ev} 和 h_{od} 为一维 Gabor 时域滤波器, 定义分别为:

$$\begin{aligned} h_{ev}(t; \tau, \omega) &= -\cos(2\pi t\omega) e^{-t^2/\tau^2} \\ h_{od}(t; \tau, \omega) &= -\sin(2\pi t\omega) e^{-t^2/\tau^2} \end{aligned} \quad (8-2)$$

σ 和 τ 分别为空域和时域尺度, $w = 4/\tau$ 。



图8-1 根据Dollar方法对不同行人走路行为提取的时空特征点分布图

8.2.3 基于 GMM 建立特征点集模型

在高斯混合模型 (GMM) 中, 设 $y_i = (x_i, z_i)$ 为完整数据, x_i 为可观察到的变量, $z_i = (z_{i1}, \dots, z_{iG})$ 为隐含变量, 则:

$$Z_{ik} = \begin{cases} 1 & \text{if } x_i \text{ belongs to group } k \\ 0 & \text{otherwise} \end{cases} \quad (8-3)$$

设 z_i 的概率分别为 π_1, \dots, π_G 且与 G 类独立同分布, 通过观察变量 x_i 给出的 z_i 的概率密度为: $\prod_{k=1}^G f_k(x_i | \theta_k)^{z_{ik}}$ 。完整数据的对数似然函数可以表示为:

$$L(\theta_k, \pi_k, z_{ik} | x) = \sum_{i=1}^n \sum_{k=1}^G z_{ik} [\log \pi_k f_k(x_i | \theta_k)] \quad (8-4)$$

在混合模型参数求解中应用聚类的 EM 算法需要在 E-step 和 M-step 之间进行迭代。在 E-step 中, 通过观察变量 x 和当前的参数估计, 可以计算出完整数据对数似然函数的条件期望 z_{ik} ; 在 M-step 中, 根据 E-step 获得的值, 计算参数估计, 即权重, 均值, 协方差矩阵, 使得似然函数值达到最大。

8.3 对群体异常行为事件检测的模型设计

当一幅视频图像进入系统后, 我们首先按上一节中介绍的方法得到该视频的时空特征点, 建立描述符, 聚类建模, 根据描述符, 将描述符聚类, 看属于高斯模型的概率, 如此对所有的训练视频进行特征点提取, 储存训练基, 最后将所存

取的每幅图像的每段视频的视频向量连同该幅图像所代表的分类（在二分类中，即为 1 或-1）一同输入进行 SVM 分类器训练，在训练过程中，正常行为作为正样本，异常行为作为负样本。对于正样本，系统输出为+1，对于负样本则输出为-1。测试时，将测试样本出入到经过训练得到的分类器中，如果输出为+1，则该样本该行为正常行为；否则，该行为是异常行为。

为了评价分类结果，引入正确分类率 PCR 作为评价标准，正确识别率的定义如下：正确识别率=识别正确的样本数 T/总测试样本数 N。

（1）视频库：可以采用常用的数据库包括 UCSD 数据库,UMN 数据库，subway surveillance 数据库作为训练集。

（2）程序实现及结果分析：对视频图像进行检测，其程序实现框图可以归纳如下：

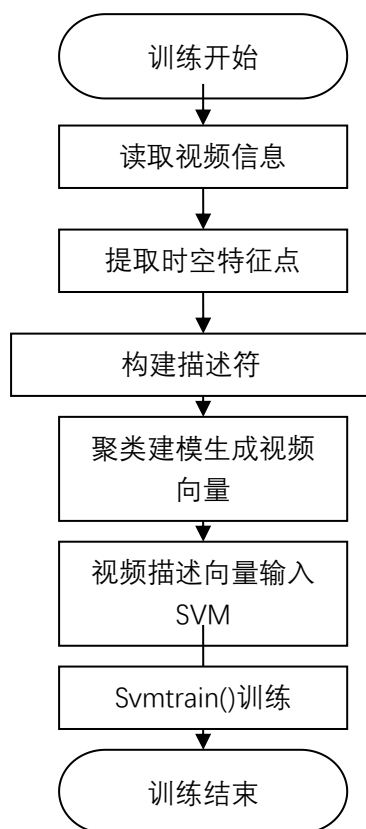


图 8-2 训练的流程图

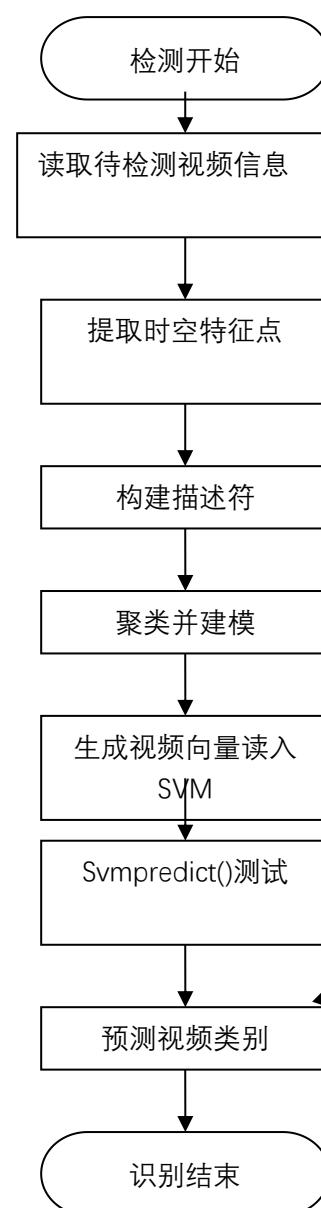


图 8-3 检测的流程图

(3) 前期训练机过程处理：输入训练样本图像：对样本视频进行底层运动特征提取处理后,进行聚类 and 建模,生成视频向量，依次对所有的视频样本进行上述操作，每一幅都会得到一组视频矢量，所有样本图像进行 SVM 高斯径核函数训练，训练机器完成。

(4) SVM 的训练和测试：将上述训练好的 SVM 进行已知的视频图像的异常行为测试，计算准确率，如果准确率大于 80%，则 SVM 就可用于未知视频的异常行为的检测中去。其中，正确率=正确样本视频数 T/参与测试的总的样本视频数 N

(5) 利用 SVM 去进行群体异常行为的检测

利用已经完善的 SVM 进行群体异常行为的检测，判断未知视频是否是异常行为。

8.4 结果展示和分析



图 8-4

针对训练好的 SVM 模型，选取 Abnormal/Normal Crowds dataset 数据集中的两段视频进行验证，对视频一模型判定其为人群逃散异常行为，对视频二模型判定其为正常人群行为。可以看出对视频特征信息提取并训练的 SVM 模型，对人群群体事件检测描述，具有较好的表现。

9 模型评价和推广

9.1 模型的优点

(1) 改进的帧差法模型，对帧数较少、光照变化较强的监控视频，进行目标提取具有较好的效果。

(2) 改进的 ViBe+算法，可以很好的抑制阴影和虚影，所提取前景目标轮廓清晰，并对抑制噪声有较好的表现。

(3) 虚拟背景图像生成模型有效的抑制了 ViBe+算法的鬼影的产生。

(4) 基于 SIFT 特征点的隔帧匹配模型，可以有效的防止抖动对前景目标提取的影响。

(5) 基于对极几何模型约束下的 SIFT 特征点匹配模型有效的检查出不同角度的同一前景目标。

9.2 模型的缺点

(1) 改进的 ViBe+算法对于视频中像素值相似部分的前景目标提取不完全，不适于对开灯关灯这种亮度快速的变化。

(2) 基于 SIFT 特征的隔帧匹配模型，计算复杂度高，对于分辨率高的视频前景提取速度过慢，不能做到实时提取。

(3) 模型参数设置较粗糙，没有经过优化模型进行优化，对于出现帧数的统计无法进行十分精细。

9.3 模型的推广

(1) 模型可用于人群流量检测，对人群聚集处可及时采取措施，防止人群拥堵，踩踏等时间的发生。

(2) 进一步更新算法后，防抖可用于车载摄像机中，用于检测前方突然出现的前景目标，及时作出反应，避免危险发生。

参考文献

- [1]O Barnich,MV Droogenbroeck,ViBE:A powerful random technique to estimate the background in video sequences ,IEEE International Conference on Acoustics,2009:945-948
- [2]赵光明,韩光,李晓飞,车少帅,刘浏,基于融合帧间差的改进 Vibe 方法,《计算机技术与发展》,2015(3):76-80
- [3]於正强,潘赞,宦若虹,一种结合帧差法和混合高斯的运动检测算法.《计算机应用与软件》,2015(04):129-132
- [4]孙水发,覃音诗,马先兵,雷帮军,室外视频前景检测中的形态学改进 ViBe 算法《计算机工程与应用》,2013,49 (10) :159-162
- [5]甘玲,赵华翔,一种改进的 ViBe 算法结合多特征融合的阴影移除方法,《微电子学与计算机》,2015 (11):152-157
- [6]傅卫平,秦川,刘佳,杨世强,王雯,基于 SIFT 算法的图像目标匹配与定位,《仪器仪表学报》,2011,32(1):163-169
- [7]刘佳,傅卫平,王雯,李娜,基于改进 SIFT 算法的图像匹配,《仪器仪表学报》,2013,34(5):1107-1112
- [8] Black J, Ellis T, Rosin P,Multi View Image Surveillance and Tracking[M],Multi view image surveillance and tracking,2002:169-174.
- [9] Mittal A, Davis L S,M2Tracker: A Multi-View Approach to Segmenting and Tracking People in a Cluttered Scene Using Region-Based Stereo[M],Kluwer Academic Publishers, 2003.
- [10]Kim K, Davis L S,Multi-camera Tracking and Segmentation of Occluded People on Ground Plane Using Search-Guided Particle Filtering[M],Computer Vision – ECCV 2006. Springer Berlin Heidelberg, 2006:98-109.
- [11] Val A D,Li M R,Shoham Y. In Proceedings of the 15th International Joint Conference on Arti[C],M. E. Pollack. 2000:508--513.
- [12] Kettner V, Zabih R. Bayesian Multi-Camera Surveillance[C],Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on. IEEE, 1999:259 Vol. 2.
- [13] Pasula H, Russell S J, Ostland M, et al. Tracking Many Objects with Many Sensors[C],Sixteenth International Joint Conference on Artificial Intelligence,Morgan Kaufmann Publishers Inc,1999:1160-1171.
- [14]王莉娜,尹艳鹏,周世付,张之江,面向密集人群的运动分类与跟踪算法,《科学与技术》,2014,14(18) :243-246

附录

问题一程序：

Main.m

```
clear; clc; close all;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%
```

```
% Author: Brilliantdo
```

```
% Last modified time : 2016/12/1
```

```
% Blog: http://blog.csdn.net/brilliantdo
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%
```

```
%% Parameters
```

```
param.numberOfSamples = 10;
```

```
param.matchingThreshold = 10;
```

```
param.matchingNumber = 2;
```

```
param.updateFactor = 5;
```

```
param.numberOfHistoryImages = 5;
```

```
param.lastHistoryImageSwapped = 0;
```

```
%% Video Information
```

```
filename = 'Campus.avi';
```

```
vidObj = VideoReader(filename);
```

```
firstFrame = true;
```

```
height = vidObj.Height;
```

```
width = vidObj.Width;
```

```
param.height = height;
```

```
param.width = width;
```

```
%% ViBe Moving Object Detection
```

```
while hasFrame(vidObj)
```

```
    vidFrame = readFrame(vidObj);
```

```
%    figure(1), imshow(vidFrame), title('Original Image');
```

```
    vidFrame = rgb2gray(vidFrame);
```

```
    vidFrame = double(vidFrame);
```

```
    tic;
```

```
    if firstFrame
```

```
        firstFrame = false;
```

```
        initViBe;
```

```
    end
```



```

        segmentationMap = vibeSegmentation(vidFrame, historyImages, historyBuffer,
param);
        [historyImages, historyBuffer] = vibeUpdate(vidFrame, segmentationMap,
historyImages, historyBuffer, param, ...
            jump, neighborX, neighborY, position);
        segmentationMap = medfilt2(segmentationMap);
        toc;

        figure(2), imshow(segmentationMap), title('Segmentation');
End
=====

=====

Img2avi.m
%将所有单帧图片转换为视频
DIR='E:\0917 数学建模\ViBe_Matlab\airport_frame\binary\';           %图片所在文
件夹
file=dir(strcat(DIR,'*.bmp'));                                     %读取所有 jpg 文件
filenum=size(file,1);                                           %图片总数

obj_gray = VideoWriter('E:\0917 数学建模
\ViBe_Matlab\airport_frame\binary\highway_gray.avi'); %所转换成的视频名称
writerFrames = filenum;                                         %视频帧数

%将单张图片存在 avi 文件
open(obj_gray);
for k = 1: writerFrames
    fname = strcat(DIR, num2str(k), '.bmp');
    frame = imread(fname);
    writeVideo(obj_gray, frame);
end
close(obj_gray);
=====

multiObjectTracking.m

function multiObjectTracking()
% 创建用于读取视频，检测运动物体的系统对象，
%显示结果。
obj = setupSystemObjects();

tracks = initializeTracks(); % Create an empty array of tracks.

nextId = 1; % ID of the next track

% Detect moving objects, and track them across video frames.

```

```

while ~isDone(obj.reader)
    frame = readFrame();
    [centroids, bboxes, mask] = detectObjects(frame);
    predictNewLocationsOfTracks();
    [assignments, unassignedTracks, unassignedDetections] = ...
        detectionToTrackAssignment();

    updateAssignedTracks();
    updateUnassignedTracks();
    deleteLostTracks();
    createNewTracks();

    displayTrackingResults();
end
function obj = setupSystemObjects()
    % 初始化视频 I/O
    %创建对象用于读取视频 , 绘制每一帧的跟踪目标 播放视频

    % 创建视频读取器.
    obj.reader = vision.VideoFileReader('input3.avi');

    % 创建两个 视频播放窗口 一个用于播放视频
    % 一个用于播放前景掩模.
    obj.videoPlayer = vision.VideoPlayer('Position', [20, 400, 700, 400]);
    obj.maskPlayer = vision.VideoPlayer('Position', [740, 400, 700, 400]);

    %创建用于前景检测和斑点分析的系统对象

    % The foreground detector is used to segment moving objects from
    % the background. 它输出二值掩模, 前景为 1 背景为 0.
    obj.detector = vision.ForegroundDetector('NumGaussians', 3, ...
        'NumTrainingFrames', 5, 'MinimumBackgroundRatio', 0.7);

    % Connected groups of foreground pixels are likely to correspond to
moving
    % objects. The blob analysis System object is used to find such groups
    % (called 'blobs' or 'connected components'), and compute their
    % characteristics, such as area, centroid, and the bounding box.

    obj.blobAnalyser = vision.BlobAnalysis('BoundingBoxOutputPort', true, ...
        'AreaOutputPort', true, 'CentroidOutputPort', true, ...
        'MinimumBlobArea', 400);
end

```

```

function tracks = initializeTracks()
    % create an empty array of tracks
    tracks = struct(...
        'id', {}, ...
        'bbox', {}, ...
        'kalmanFilter', {}, ...
        'age', {}, ...
        'totalVisibleCount', {}, ...
        'consecutiveInvisibleCount', {});
end
function frame = readFrame()
    frame = obj.reader.step();
end
function [centroids, bboxes, mask] = detectObjects(frame)

    % Detect foreground.
    mask = obj.detector.step(frame);

    % Apply morphological operations to 清除噪声 填补空洞.
    mask = imopen(mask, strel('rectangle', [3,3]));
    mask = imclose(mask, strel('rectangle', [15, 15]));
    mask = imfill(mask, 'holes');

    % 进行斑点分析 找到连接部件.
    [~, centroids, bboxes] = obj.blobAnalyser.step(mask);
end
function predictNewLocationsOfTracks()
    for i = 1:length(tracks)
        bbox = tracks(i).bbox;

        % 预测轨道当前位置.
        predictedCentroid = predict(tracks(i).kalmanFilter);

        % 矩形框随质心移动而移动
        predictedCentroid = int32(predictedCentroid) - bbox(3:4) / 2;
        tracks(i).bbox = [predictedCentroid, bbox(3:4)];
    end
end
function [assignments, unassignedTracks, unassignedDetections] = ...
    detectionToTrackAssignment()

    nTracks = length(tracks);
    nDetections = size(centroids, 1);

```

```

    %计算轨道分配给检测到目标的成本.
    cost = zeros(nTracks, nDetections);
    for i = 1:nTracks
        cost(i, :) = distance(tracks(i).kalmanFilter, centroids);
    end

    % 解决分配问题.
    costOfNonAssignment = 20;
    [assignments, unassignedTracks, unassignedDetections] = ...
        assignDetectionsToTracks(cost, costOfNonAssignment);
end

function updateAssignedTracks()
    numAssignedTracks = size(assignments, 1);
    for i = 1:numAssignedTracks
        trackIdx = assignments(i, 1);
        detectionIdx = assignments(i, 2);
        centroid = centroids(detectionIdx, :);
        bbox = bboxes(detectionIdx, :);

        % Correct the estimate of the object's location
        % using the new detection.
        correct(tracks(trackIdx).kalmanFilter, centroid);

        % Replace predicted bounding box with detected
        % bounding box.
        tracks(trackIdx).bbox = bbox;

        % Update track's age.
        tracks(trackIdx).age = tracks(trackIdx).age + 1;

        % Update visibility.
        tracks(trackIdx).totalVisibleCount = ...
            tracks(trackIdx).totalVisibleCount + 1;
        tracks(trackIdx).consecutiveInvisibleCount = 0;
    end
end

function updateUnassignedTracks()
    for i = 1:length(unassignedTracks)
        ind = unassignedTracks(i);
        tracks(ind).age = tracks(ind).age + 1;
        tracks(ind).consecutiveInvisibleCount = ...
            tracks(ind).consecutiveInvisibleCount + 1;
    end
end
end

```

```

function deleteLostTracks()
    if isempty(tracks)
        return;
    end

    invisibleForTooLong = 20;
    ageThreshold = 8;

    % Compute the fraction of the track's age for which it was visible.
    ages = [tracks(:).age];
    totalVisibleCounts = [tracks(:).totalVisibleCount];
    visibility = totalVisibleCounts ./ ages;

    % Find the indices of 'lost' tracks.
    lostInds = (ages < ageThreshold & visibility < 0.6) | ...
        [tracks(:).consecutiveInvisibleCount] >= invisibleForTooLong;

    % Delete lost tracks.
    tracks = tracks(~lostInds);
end

function createNewTracks()
    centroids = centroids(unassignedDetections, :);
    bboxes = bboxes(unassignedDetections, :);

    for i = 1:size(centroids, 1)

        centroid = centroids(i,:);
        bbox = bboxes(i, :);

        % Create a Kalman filter object.
        kalmanFilter = configureKalmanFilter('ConstantVelocity', ...
            centroid, [200, 50], [100, 25], 100);

        % Create a new track.
        newTrack = struct(...
            'id', nextId, ...
            'bbox', bbox, ...
            'kalmanFilter', kalmanFilter, ...
            'age', 1, ...
            'totalVisibleCount', 1, ...
            'consecutiveInvisibleCount', 0);

        % Add it to the array of tracks.
        tracks(end + 1) = newTrack;
    end
end

```

```

        % Increment the next id.
        nextId = nextId + 1;
    end
end

function displayTrackingResults()
    % Convert the frame and the mask to uint8 RGB.
    frame = im2uint8(frame);

    mask = uint8(repmat(mask, [1, 1, 3])) .* 255;

    %gmm
    fore = rgb2gray(mask);
    figure();
    imshow(fore);

    minVisibleCount = 8;
    if ~isempty(tracks)

        % Noisy detections tend to result in short-lived tracks.
        % Only display tracks that have been visible for more than
        % a minimum number of frames.
        reliableTrackInds = ...
            [tracks(:).totalVisibleCount] > minVisibleCount;
        reliableTracks = tracks(reliableTrackInds);

        % Display the objects. If an object has not been detected
        % in this frame, display its predicted bounding box.
        if ~isempty(reliableTracks)
            % Get bounding boxes.
            bboxes = cat(1, reliableTracks.bbox);

            % Get ids.
            ids = int32([reliableTracks(:).id])

            % Create labels for objects indicating the ones for
            % which we display the predicted rather than the actual
            % location.
            labels = cellstr(int2str(ids));
            predictedTrackInds = ...
                [reliableTracks(:).consecutiveInvisibleCount] > 0;
            isPredicted = cell(size(labels));
            isPredicted(predictedTrackInds) = {' predicted'};
        end
    end
end

```



```

        labels = strcat(labels, isPredicted);

        % Draw the objects on the frame.
        frame = insertObjectAnnotation(frame, 'rectangle', ...
            bboxes, labels);

        % Draw the objects on the mask.
        mask = insertObjectAnnotation(mask, 'rectangle', ...
            bboxes, labels);
    end
end

% Display the mask and the frame.
obj.maskPlayer.step(mask);
obj.videoPlayer.step(frame);
end
End
=====
neighborMain.m

function multiObjectTracking()
% 创建用于读取视频，检测运动物体的系统对象，
%显示结果。
obj = setupSystemObjects();

tracks = initializeTracks(); % Create an empty array of tracks.

nextId = 1; % ID of the next track

% Detect moving objects, and track them across video frames.
while ~isDone(obj.reader)
    frame = readFrame();
    [centroids, bboxes, mask] = detectObjects(frame);
    predictNewLocationsOfTracks();
    [assignments, unassignedTracks, unassignedDetections] = ...
        detectionToTrackAssignment();

    updateAssignedTracks();
    updateUnassignedTracks();
    deleteLostTracks();
    createNewTracks();

    displayTrackingResults();
end

```

```

function obj = setupSystemObjects()
    % 初始化视频 I/O
    %创建对象用于读取视频 , 绘制每一帧的跟踪目标 播放视频

    % 创建视频读取器.
    obj.reader = vision.VideoFileReader('input3.avi');

    % 创建两个 视频播放窗口 一个用于播放视频
    % 一个用于播放前景掩模.
    obj.videoPlayer = vision.VideoPlayer('Position', [20, 400, 700, 400]);
    obj.maskPlayer = vision.VideoPlayer('Position', [740, 400, 700, 400]);

    %创建用于前景检测和斑点分析的系统对象

    % The foreground detector is used to segment moving objects from
    % the background. 它输出二值掩模, 前景为 1 背景为 0.
    obj.detector = vision.ForegroundDetector('NumGaussians', 3, ...
        'NumTrainingFrames', 5, 'MinimumBackgroundRatio', 0.7);

    % Connected groups of foreground pixels are likely to correspond to
moving
    % objects.  The blob analysis System object is used to find such groups
    % (called 'blobs' or 'connected components'), and compute their
    % characteristics, such as area, centroid, and the bounding box.

    obj.blobAnalyser = vision.BlobAnalysis('BoundingBoxOutputPort', true, ...
        'AreaOutputPort', true, 'CentroidOutputPort', true, ...
        'MinimumBlobArea', 400);
end
function tracks = initializeTracks()
    % create an empty array of tracks
    tracks = struct(...
        'id', {}, ...
        'bbox', {}, ...
        'kalmanFilter', {}, ...
        'age', {}, ...
        'totalVisibleCount', {}, ...
        'consecutiveInvisibleCount', {});
end
function frame = readFrame()
    frame = obj.reader.step();
end
function [centroids, bboxes, mask] = detectObjects(frame)

```

```

% Detect foreground.
mask = obj.detector.step(frame);

% Apply morphological operations to 清除噪声 填补空洞.
mask = imopen(mask, strel('rectangle', [3,3]));
mask = imclose(mask, strel('rectangle', [15, 15]));
mask = imfill(mask, 'holes');

% 进行斑点分析 找到连接部件.
[~, centroids, bboxes] = obj.blobAnalyser.step(mask);
end
function predictNewLocationsOfTracks()
    for i = 1:length(tracks)
        bbox = tracks(i).bbox;

        % 预测轨道当前位置.
        predictedCentroid = predict(tracks(i).kalmanFilter);

        % 矩形框随质心移动而移动
        predictedCentroid = int32(predictedCentroid) - bbox(3:4) / 2;
        tracks(i).bbox = [predictedCentroid, bbox(3:4)];
    end
end
function [assignments, unassignedTracks, unassignedDetections] = ...
    detectionToTrackAssignment()

    nTracks = length(tracks);
    nDetections = size(centroids, 1);

    %计算轨道分配给检测到目标的成本.
    cost = zeros(nTracks, nDetections);
    for i = 1:nTracks
        cost(i, :) = distance(tracks(i).kalmanFilter, centroids);
    end

    % 解决分配问题.
    costOfNonAssignment = 20;
    [assignments, unassignedTracks, unassignedDetections] = ...
        assignDetectionsToTracks(cost, costOfNonAssignment);
end
function updateAssignedTracks()
    numAssignedTracks = size(assignments, 1);
    for i = 1:numAssignedTracks

```

```

        trackIdx = assignments(i, 1);
        detectionIdx = assignments(i, 2);
        centroid = centroids(detectionIdx, :);
        bbox = bboxes(detectionIdx, :);

        % Correct the estimate of the object's location
        % using the new detection.
        correct(tracks(trackIdx).kalmanFilter, centroid);

        % Replace predicted bounding box with detected
        % bounding box.
        tracks(trackIdx).bbox = bbox;

        % Update track's age.
        tracks(trackIdx).age = tracks(trackIdx).age + 1;

        % Update visibility.
        tracks(trackIdx).totalVisibleCount = ...
            tracks(trackIdx).totalVisibleCount + 1;
        tracks(trackIdx).consecutiveInvisibleCount = 0;
    end
end

function updateUnassignedTracks()
    for i = 1:length(unassignedTracks)
        ind = unassignedTracks(i);
        tracks(ind).age = tracks(ind).age + 1;
        tracks(ind).consecutiveInvisibleCount = ...
            tracks(ind).consecutiveInvisibleCount + 1;
    end
end

function deleteLostTracks()
    if isempty(tracks)
        return;
    end

    invisibleForTooLong = 20;
    ageThreshold = 8;

    % Compute the fraction of the track's age for which it was visible.
    ages = [tracks(:).age];
    totalVisibleCounts = [tracks(:).totalVisibleCount];
    visibility = totalVisibleCounts ./ ages;

    % Find the indices of 'lost' tracks.

```

```

lostInds = (ages < ageThreshold & visibility < 0.6) | ...
    [tracks(:).consecutiveInvisibleCount] >= invisibleForTooLong;

% Delete lost tracks.
tracks = tracks(~lostInds);
end
function createNewTracks()
    centroids = centroids(unassignedDetections, :);
    bboxes = bboxes(unassignedDetections, :);

    for i = 1:size(centroids, 1)

        centroid = centroids(i,:);
        bbox = bboxes(i, :);

        % Create a Kalman filter object.
        kalmanFilter = configureKalmanFilter('ConstantVelocity', ...
            centroid, [200, 50], [100, 25], 100);

        % Create a new track.
        newTrack = struct(...
            'id', nextId, ...
            'bbox', bbox, ...
            'kalmanFilter', kalmanFilter, ...
            'age', 1, ...
            'totalVisibleCount', 1, ...
            'consecutiveInvisibleCount', 0);

        % Add it to the array of tracks.
        tracks(end + 1) = newTrack;

        % Increment the next id.
        nextId = nextId + 1;
    end
end
function displayTrackingResults()
    % Convert the frame and the mask to uint8 RGB.
    frame = im2uint8(frame);

    mask = uint8(repmat(mask, [1, 1, 3])) .* 255;

    %gmm
    fore = rgb2gray(mask);
    figure();

```

```

imshow(fore);

minVisibleCount = 8;
if ~isempty(tracks)

    % Noisy detections tend to result in short-lived tracks.
    % Only display tracks that have been visible for more than
    % a minimum number of frames.
    reliableTrackInds = ...
        [tracks(:).totalVisibleCount] > minVisibleCount;
    reliableTracks = tracks(reliableTrackInds);

    % Display the objects. If an object has not been detected
    % in this frame, display its predicted bounding box.
    if ~isempty(reliableTracks)
        % Get bounding boxes.
        bboxes = cat(1, reliableTracks.bbox);

        % Get ids.
        ids = int32([reliableTracks(:).id])

        % Create labels for objects indicating the ones for
        % which we display the predicted rather than the actual
        % location.
        labels = cellstr(int2str(ids));
        predictedTrackInds = ...
            [reliableTracks(:).consecutiveInvisibleCount] > 0;
        isPredicted = cell(size(labels));
        isPredicted(predictedTrackInds) = {' predicted'};
        labels = strcat(labels, isPredicted);

        % Draw the objects on the frame.
        frame = insertObjectAnnotation(frame, 'rectangle', ...
            bboxes, labels);

        % Draw the objects on the mask.
        mask = insertObjectAnnotation(mask, 'rectangle', ...
            bboxes, labels);
    end
end

% Display the mask and the frame.
obj.maskPlayer.step(mask);

```

```

        obj.videoPlayer.step(frame);
    end
end

```

问题二程序（使用 c++和 opencv）

originalVibe.h

```

#ifndef ORIGINALVIBE_H
#define ORIGINALVIBE_H
#include<opencv2\core\core.hpp>
#include<opencv2\imgproc\imgproc.hpp>
#include<opencv2\highgui\highgui.hpp>
#include<vector>
using namespace cv;

class OriginalVibe{
public:
    //构造函数
    OriginalVibe(){};
    OriginalVibe(int _numberSamples, int _minMatch, int _distanceThreshold, int
_updateFactor, int _neighborWidth, int _neighborHeight);
    ~OriginalVibe(){};

    int distanceL1(const Vec3b &src1, const Vec3b &src2);

    float distanceL2(const Vec3b &src1, const Vec3b &src2);
    //操作成员变量
    void setUpdateFactor(int _updateFactor);
    //灰度图像
    void originalVibe_Init_GRAY(const Mat &firstFrame);
    void originalVibe_ClassifyAndUpdate_GRAY(const Mat &frame,OutputArray
&_segmentation);
    //RGB 三通道
    void originalVibe_Init_BGR(const Mat & firstFrame);
    void originalVibe_ClassifyAndUpdate_BGR(const Mat &frame,OutputArray
&_segmentation);

    int minMatch;
    //BGR 的距离计算

    //背景模型
    int numberSamples;
    std::vector<Mat> backgroundModel;
    //像素点的分类判断的参数

```



```

    int distanceThreshold;
    //背景模型更新概率
    int updateFactor;
    //8-邻域(3 x 3)
    int neighborWidth;
    int neighborHeight;

};
#endif
=====

originalVibe.cpp

#include"originalVibe.h"
#include<iostream>
//const unsigned char OriginalVibe::BACK_GROUND = 0;
//const unsigned char OriginalVibe::FORE_GROUND = 255;

//前景和背景分割
//const unsigned char BACK_GROUND = 0;
//const unsigned char FORE_GROUND = 255;

OriginalVibe::OriginalVibe(int _numberSamples, int _minMatch, int
_distanceThreshold, int _updateFactor, int _neighborWidth, int _neighborHeight)
{
    numberSamples = _numberSamples;
    minMatch = _minMatch;
    distanceThreshold = _distanceThreshold;
    updateFactor = _updateFactor;
    neighborWidth = _neighborWidth;
    neighborHeight = _neighborHeight;
}

//操作成员变量
void OriginalVibe::setUpdateFactor(int _updateFactor)
{
    this->updateFactor = _updateFactor;
}
//第一种方法：最原始的 vibe 灰度通道

```

```

void OriginalVibe::originalVibe_Init_GRAY(const Mat &firstFrame)
{
    int height = firstFrame.rows;
    int width = firstFrame.cols;
    //背景模型分配内存
    backgroundModel.clear();
    for(int index = 0;index < this->numberSamples;index++)
    {
        backgroundModel.push_back(Mat::zeros(height,width,CV_8UC1));
    }
    //随机数
    RNG rng;
    int cshift;
    int rshift;
    for(int r = 0;r < height ;r++)
    {
        for(int c = 0;c < width ; c++)
        {
            if( c < neighborWidth/2 || c > width - neighborWidth/2 -1 || r <
neighborHeight/2 || r > height - neighborHeight/2 -1)
            {
                /*随机数的生成方式有很多种*/
                /*
                cshift = randu<int>()%neighborWidth - neighborWidth/2;
                rshift = randu<int>()%neighborHeight - neighborHeight/2;
                */
                cshift = rand()%neighborWidth - neighborWidth/2;
                rshift = rand()%neighborHeight - neighborHeight/2;

                for(std::vector<Mat>::iterator it = backgroundModel.begin();it !=
backgroundModel.end();it++)
                {
                    for(;;)
                    {
                        /*
                        cshift = rng.uniform(-neighborWidth/2,neighborWidth/2 +
1);
                        rshift = rng.uniform(-neighborHeight/2,neighborHeight/2
+1 );
                        */
                        cshift = abs(randu<int>()%neighborWidth) -
neighborWidth/2;
                        rshift = abs(randu<int>()%neighborHeight) -
neighborHeight/2;

```



```

 segmentation.create(frame.size(),CV_8UC1);
Mat segmentation = segmentation.getMat();

RNG rng;
for(int r = 0; r < height;r++)
{
    for(int c = 0;c < width ;c++)
    {
        int count = 0;
        unsigned char pixel = frame.at<uchar>(r,c);
        //让 pixel 和背景模板中 backgroundModel 进行比较
        for(std::vector<Mat>::iterator it = backgroundModel.begin();it !=
backgroundModel.end();it++)
        {
            if( abs( int(pixel) - int( (*it).at<uchar>(r,c)) ) <
(this->distanceThreshold) )
            {
                count++;
                //循环到一定阶段, 判断 count 的值是否大于 minMatch,更新
背景模型

                if( count >= this->minMatch)
                {
                    int random = rng.uniform(0,this->updateFactor);
                    if(random == 0)
                    {
                        int updateIndex =
rng.uniform(0,this->numberSamples);
                        backgroundModel[updateIndex].at<uchar>(r,c) = pixel;
                    }
                    random = rng.uniform(0,this->updateFactor);
                    if(random == 0)
                    {
                        if(c < neighborWidth/2 || c > width -
neighborWidth/2-1 || r < neighborHeight/2 || r > height - neighborHeight/2-1)
                        {
                            for(;;)
                            {
                                /*
                                cshift =
rng.uniform(-neighborWidth/2,neighborWidth/2 + 1);
                                rshift =
rng.uniform(-neighborHeight/2,neighborHeight/2 + 1 );
                                */
                                cshift = abs(randu<int>()%neighborWidth) -

```

```

neighborWidth/2;
neighborHeight/2;

rshift = abs(randu<int>()%neighborHeight) -
neighborHeight/2;

if(!(cshift == 0 && rshift ==0))
    break;
}
if(c + cshift < 0 || c + cshift >=width)
    cshift *= -1;
if(r + rshift < 0 || r + rshift >= height)
    rshift *= -1;
int updateIndex =
rng.uniform(0,this->numberSamples);

backgroundModel[updateIndex].at<uchar>(r+rshift,c+cshift) = pixel;
}
else
{
    for(;;)
    {
        /*
        rng.uniform(-neighborWidth/2,neighborWidth/2 + 1);
        rshift =
        rng.uniform(-neighborHeight/2,neighborHeight/2 + 1 );
        */
        cshift = abs(randu<int>()%neighborWidth) -
neighborWidth/2;
        rshift = abs(randu<int>()%neighborHeight) -
neighborHeight/2;

        if(!(cshift == 0 && rshift==0))
            break;
    }
    int updateIndex =
rng.uniform(0,this->numberSamples);

    backgroundModel[updateIndex].at<uchar>(r+rshift,c+cshift) = pixel;
    }
    }
    segmentation.at<uchar>(r,c) = 0;
    break;
}
}
}
if( count < this->minMatch)

```

```

        segmentation.at<uchar>(r,c) = 255;
    }
}
}

```

//第三种方法： BGR 通道

```

void OriginalVibe::originalVibe_Init_BGR(const Mat & fristFrame)
{
    int height = fristFrame.rows;
    int width = fristFrame.cols;
    //背景模型分配内存
    backgroundModel.clear();
    for(int index = 0; index < this->numberSamples; index++)
    {
        backgroundModel.push_back( Mat::zeros(height,width,CV_8UC3) );
    }
    //随机数
    RNG rng;
    int cshift;
    int rshift;
    for(int r = 0 ; r < height; r++)
    {
        for(int c = 0; c < width ; c++)
        {
            if( c < neighborWidth/2 || c > width - neighborWidth/2 - 1 || r <
neighborHeight/2 || r > height - neighborHeight/2 - 1 )
            {
                /*
                初始化背景模型： 开始
                */
                for(vector<Mat>::iterator iter = backgroundModel.begin(); iter !=
backgroundModel.end(); iter++)
                {
                    for(;;)
                    {
                        cshift = abs(randu<int>()%neighborWidth) -
neighborWidth/2;
                        rshift = abs(randu<int>()%neighborHeight) -
neighborHeight/2;
                        if(!(cshift == 0 && rshift==0))
                            break;
                    }
                    if(c + cshift < 0 || c + cshift >=width)

```

```

        cshift *= -1;
        if(r + rshift < 0 || r + rshift >= height)
            rshift *= -1;
        (*iter).at<Vec3b>(r,c) =
fristFrame.at<Vec3b>(r+rshift,c+cshift);
    }
}
/*初始化背景模型： 结束*/
else
{
    /*****初始化背景模型： 开始*****/
    for(vector<Mat>::iterator iter = backgroundModel.begin(); iter !=
backgroundModel.end();iter++)
    {
        for(;;)
        {
            cshift = abs(randu<int>()%neighborWidth) -
neighborWidth/2;
            rshift = abs(randu<int>()%neighborHeight) -
neighborHeight/2;
            if( !(cshift == 0 && rshift==0) )
                break;
        }
        (*iter).at<Vec3b>(r,c) =
fristFrame.at<Vec3b>(r+rshift,c+cshift);
    }
    /*****初始化背景模型： 结束 *****/
}
}
}

int OriginalVibe::distanceL1(const Vec3b & src1, const Vec3b& src2)
{
    return abs(src1[0] - src2[0]) + abs(src1[1] - src2[1]) + abs(src1[2] - src2[2]);
}

float OriginalVibe::distanceL2(const Vec3b & src1,const Vec3b& src2)
{
    return pow( pow(src1[0]-src2[0],2.0) +pow(src1[1]-src2[1],2.0) + pow(src1[2] -
src2[2],2.0),0.5);
}

```



```

void OriginalVibe::originalVibe_ClassifyAndUpdate_BGR(const Mat
&frame,OutputArray &_segmentation)
{/**编号 1
    int height = frame.rows;
    int width = frame.cols;
    int cshift;
    int rshift;
    _segmentation.create(frame.size(),CV_8UC1);
    Mat segmentation = _segmentation.getMat();

    RNG rng;

    for(int r=0 ;r < height; r++)
    {/**编号 1-1
        for(int c = 0;c < width ;c++)
        {/**编号 1-1-1
            int count = 0;
            Vec3b pixel = frame.at<Vec3b>(r,c);
            for( vector<Mat>::iterator iter = backgroundModel.begin() ;iter !=
backgroundModel.end(); iter++)
                {/**编号 1-1-1-1
                    //
                    //
                    if( distanceL1(pixel,(*iter).at<Vec3b>(r,c)) <
4.5*this->distanceThreshold )
                        {
                            count++;
                            if(count >= this->minMatch)
                                {
                                    //第一步:更新模型 update
                                    /*****开始更新模型*****/
                                    int random = rng.uniform(0,this->updateFactor);
                                    if(random == 0)
                                        {
                                            int updateIndex =
rng.uniform(0,this->numberSamples);
                                            backgroundModel[updateIndex].at<Vec3b>(r,c) =
pixel;
                                        }

                                    random = rng.uniform(0,this->updateFactor);
                                    if(random == 0)
                                        {

```

```

        /*****
        if( c < neighborWidth/2 || c > width -
neighborWidth/2-1 || r < neighborHeight/2 || r > height - neighborHeight/2-1 )
        {
            for(;;)
            {
                cshift = abs(randu<int>()%neighborWidth) -
neighborWidth/2;
                rshift = abs(randu<int>()%neighborHeight) -
neighborHeight/2;

                if(!(cshift == 0 && rshift==0))
                    break;
            }
            if(c + cshift < 0 || c + cshift >=width)
                cshift*=-1;
            if(r + rshift < 0 || r + rshift >= height)
                rshift*=-1;
            int updateIndex =
rng.uniform(0,this->numberSamples);

            backgroundModel[updateIndex].at<Vec3b>(r+rshift,c+cshift) = pixel;
        }
        else
        {
            for(;;)
            {
                cshift = abs(rand()%neighborWidth) -
neighborWidth/2;
                rshift = abs(rand()%neighborHeight) -
neighborHeight/2;

                if(!(cshift == 0 && rshift==0))
                    break;
            }
            int updateIndex =
rng.uniform(0,this->numberSamples);

            backgroundModel[updateIndex].at<Vec3b>(r+rshift,c+cshift) = pixel;
        }
        /*****
    }
    /*
    *****结束更新模型*****
    */
    //第二步： 分类 classify

```

```

                segmentation.at<uchar>(r,c) = 0;
                break;
            }
        }
    }//编号 1-1-1-1
    if(count < this->minMatch)//classify
        segmentation.at<uchar>(r,c) = 255;
    }//编号 1-1-1
} //编号 1-1

} // *编号 1
=====

```

Main.cpp

```

#include "cv.h"
#include "highgui.h"
#include "originalVibe.h"
#include<iostream>
using namespace std;
using namespace cv;
int main()
{
    /*视频流的输入*/
    VideoCapture cap("car7.avi");
    if(!cap.isOpened())
        return -1;

    /*视频帧图像*/
    Mat frame;
    /*前景-背景检测及显示窗口*/
    Mat seg;
    /*创建 vibe 背景建模的对象*/
    OriginalVibe vibe(20,2,20,16,3,3);
    Mat frameGray;
    int frame_width = cap.get(CV_CAP_PROP_FRAME_WIDTH);
    int frame_height = cap.get(CV_CAP_PROP_FRAME_HEIGHT);
    int frame_fps = cap.get(CV_CAP_PROP_FPS);
    //cv::VideoWriter writer;
    //writer = VideoWriter("output.avi", CV_FOURCC('X', 'V', 'T', 'D'), frame_fps,
    Size(frame_width, frame_height), 1);

    int number =0;
    for(;;)

```

```

{
    cap >> frame;
    if(! frame.data)
        break;
    number++;
    char pathsource[20] ="source/sou";
    sprintf(pathsource, "source/sou%d.jpg", number);
    imwrite(pathsource, frame);

    if(number == 1)
    {
        vibe.originalVibe_Init_BGR( frame );
        imwrite("result/pic1.jpg", frame);
        continue;
    }
    else
    {
        vibe.originalVibe_ClassifyAndUpdate_BGR(frame,seg);
        medianBlur(seg,seg,5);
        imshow("segmentation", seg);
    }
    char pathresult[20] = "result/pic";
    sprintf(pathresult, "result/pic%d.jpg", number);
    imwrite(pathresult,seg);
    //writer.write(seg);
    imshow("frame",frame);
    if(waitKey(10) >= 0)
        break;
}

return 0;
}

```

问题 3 实现程序

```

#include <iostream>
#include <string>
#include "removesmall.h"
// #include "opencv2/highgui/highgui.hpp"
#include "opencv2/core/core.hpp"
#include "opencv2/video/background_segm.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <opencv2/features2d/features2d.hpp>
#include <opencv2/nonfree/nonfree.hpp>
#include "sift_warp.h"

```

```

#include "surf.h"

using namespace std;
using namespace cv;
void Video_To_Image(string filename, int &framenum, int &wide, int &len);
void Image_To_Video(int framenumb, int &wid, int &lenth);
int main()
{
    string video_name = "waterSurface.avi"; //注意，使用 string 时，若不用 using
    namespace std, 需要使用 std::string
    int frame_num=0;
    int frame_wid = 0;
    int frame_len = 0;
    Video_To_Image(video_name, frame_num, frame_wid, frame_len);
    Image_To_Video(frame_num, frame_wid, frame_len);
    return 0;
}
void Video_To_Image(string filename,int &framenum,int &wide,int &len)
{
    cout << "-----Video_To_Image-----" << endl;
    cv::VideoCapture capture(filename);
    //CvCapture* capture = cvCreateFileCapture(filename);
    if (!capture.isOpened())
    {
        cout << "open video error";
    }
    /*CV_CAP_PROP_POS_MSEC - 视频的当前位置（毫秒）
    CV_CAP_PROP_POS_FRAMES - 视频的当前位置（帧）
    CV_CAP_PROP_FRAME_WIDTH - 视频流的宽度
    CV_CAP_PROP_FRAME_HEIGHT - 视频流的高度
    CV_CAP_PROP_FPS - 帧速率（帧 / 秒）*/
    int frame_width = (int)capture.get(CV_CAP_PROP_FRAME_WIDTH);
    int frame_height = (int)capture.get(CV_CAP_PROP_FRAME_HEIGHT);
    float frame_fps = capture.get(CV_CAP_PROP_FPS);
    int frame_number = capture.get(CV_CAP_PROP_FRAME_COUNT); //总帧数
    framenum = frame_number;
    wide = frame_width;
    len = frame_height;
    cout << "frame_width is " << frame_width << endl;
    cout << "frame_height is " << frame_height << endl;
    cout << "frame_fps is " << frame_fps << endl;

    int num = 0; //统计帧数
    cv::Mat img;

```

```

string img_name;
char image_name[20];
cv::namedWindow("MyVideo", CV_WINDOW_AUTOSIZE);
vector<Mat>single_frame;
while (true)
//while (num<frame_number)
{
    cv::Mat frame;
    //从视频中读取一个帧
    //cvSetCaptureProperty(capture,CV_CAP_PROP_POS_FRAMES, num);
    bool bSuccess = capture.read(frame);
    if (!bSuccess)
    {
        cout << "不能从视频文件读取帧" << endl;
        break;
    }
    cv::Mat out_temp;
    Mat_WarpImage;
    //medianBlur(frame, out_temp, 7);
    GaussianBlur(frame, out_temp, Size(7, 7), 0, 0);
    //blur(frame, out_temp, Size(5, 5), Point(-1, -1));
    //RemoveSmallRegion(out_temp, out_temp, 20, 1, 1);
    //medianBlur(frame, out_temp, 5);
    if (num==0)
    {
        //Mat *temp=new Mat();
        single_frame.push_back(out_temp);
        single_frame.push_back(out_temp);
        single_frame.push_back(out_temp);
        //single_frame.push_back(out_temp);

    }
    else
    {
        single_frame.push_back(out_temp);
    }
    //
    //std::vector<KeyPoint>keypoints;
    // 构造 SIFT 特征检测器
    //cv::SiftFeatureDetector sift(
        //0.03, // 特征的阈值
        //10.); // 用于降低

    // 检测 SIFT 特征值

```

```

//sift.detect(out_temp, keypoints);

//cv::drawKeypoints(out_temp, // 原始图像
// keypoints, // 特征点的向量
// featureImage, // 生成图像
// cv::Scalar(255, 255, 255), // 特征点的颜色
// cv::DrawMatchesFlags::DRAW_RICH_KEYPOINTS); // 标志位

//cv::namedWindow("SIFT Features");
//cv::imshow("SIFT Features", featureImage);

//在 MyVideo 窗口上显示当前帧
//imshow("MyVideo", out_temp);
//保存图片名
//sprintf(const_cast<char*>(img_name.data()), "%s%d%s", "image", ++num,
".jpg");//保存图片名
//WarpImage = siftwarp(2, single_frame[num+1], single_frame[num]);
//WarpImage = surf_capture(2, single_frame[num+2], single_frame[num]);
WarpImage = sift_capture(2, single_frame[num + 2], single_frame[num]);
//single_frame.push_back(WarpImage);
sprintf(image_name, "%s%d%s", "image", num, ".jpg");//保存图片名
num++;
img_name = image_name;
imwrite(img_name, WarpImage);//保存保存一帧图片
if (cv::waitKey(30) == 27 || num == frame_number)
{
    cout << "按下 ESC 键" << endl;
    break;
}
}
capture.release();//这句话貌似不需要
}

void Image_To_Video(int framenumb , int &wid, int &lenth)
{
    cout<< "-----Video_To_Image-----"<<endl;
    char image_name[20];
    string s_image_name;
    cv::VideoWriter writer;
    int isColor = 1;//不知道是干啥用的
    int frame_fps = 30;
    int frame_width = wid;
    int frame_height = lenth;
    string video_name = "car7out.avi";
    writer= VideoWriter(video_name, CV_FOURCC('X', 'V', 'T', 'D'), frame_fps,

```

```

Size(frame_width, frame_height), isColor);
cout << "frame_width is " << frame_width << endl;
cout << "frame_height is " << frame_height << endl;
cout << "frame_fps is " << frame_fps << endl;
cv::namedWindow("image to video", CV_WINDOW_AUTOSIZE);
int num = 2000;//输入的图片总张数
int i = 0;
Mat img;
while (i <= num)
{
    sprintf(image_name, "%s%d%s", "image", i, ".jpg");
    i++;
    s_image_name = image_name;
    img = imread(s_image_name);//读入图片
    if (!img.data)//判断图片调入是否成功
    {
        cout << "Could not load image file...\n" << endl;
    }
    imshow("image to video", img);
    //写入
    writer.write(img);
    if (cv::waitKey(30) == 27 || i == framenumb)
    {
        cout << "按下 ESC 键" << endl;
        break;
    }
}
}

```

Sift_warp.cpp

```
#include"surf.h"
```

```
using namespace cv;
using namespace std;
```

```
Mat sift_capture(int argc, Mat argv1, Mat argv2)
```

```

{
    //VideoCapture capture(0);
    //VideoCapture capture(filename);

    Mat image01, image02, imgdiff;
    image01 = argv1;
    image02 = argv2;

```



```

//灰度图转换
Mat image1, image2;
cvtColor(image01, image1, CV_RGB2GRAY);
cvtColor(image02, image2, CV_RGB2GRAY);
//while (true)
//{
//隔两帧配准

//if (image01.empty())
//{
//;
// }

//capture >> image02;

// if (image02.empty())
// {
//     break;
// }
//GaussianBlur(image02, image02, Size(3,3), 0);

double time0 = static_cast<double>(getTickCount());//开始计时

//灰度图转换
/*Mat image1, image2;
cvtColor(image01, image1, CV_RGB2GRAY);
cvtColor(image02, image2, CV_RGB2GRAY);*/

//提取特征点
SiftFeatureDetector siftDetector(2500); // 海塞矩阵阈值，高一点速度会快
些
vector<KeyPoint> keyPoint1, keyPoint2;
siftDetector.detect(image1, keyPoint1);
siftDetector.detect(image2, keyPoint2);

//特征点描述，为下边的特征点匹配做准备
SiftDescriptorExtractor siftDescriptor;;
Mat imageDesc1, imageDesc2;
siftDescriptor.compute(image1, keyPoint1, imageDesc1);
siftDescriptor.compute(image2, keyPoint2, imageDesc2);

//获得匹配特征点，并提取最优配对

```

```

FlannBasedMatcher matcher;
vector<DMatch> matchePoints;
matcher.match(imageDesc1, imageDesc2, matchePoints, Mat());
sort(matchePoints.begin(), matchePoints.end()); //特征点排序

//获取排在前 N 个的最优匹配特征点
vector<Point2f> imagePoints1, imagePoints2;
if ((int)(matchePoints.size()*0.25) < 4)
{
    for (int i = 0; i < (int)(matchePoints.size()); i++)
    {
        imagePoints1.push_back(keyPoint1[matchePoints[i].queryIdx].pt);
        imagePoints2.push_back(keyPoint2[matchePoints[i].trainIdx].pt);
    }
}
else
{
    for (int i = 0; i < (int)(matchePoints.size()*0.25); i++)
    {

        imagePoints1.push_back(keyPoint1[matchePoints[i].queryIdx].pt);
        imagePoints2.push_back(keyPoint2[matchePoints[i].trainIdx].pt);
    }
}

//获取图像 1 到图像 2 的投影映射矩阵 尺寸为 3*3
Mat homo = findHomography(imagePoints1, imagePoints2, CV_RANSAC);
//cout<<"变换矩阵为: \n"<<homo<<endl<<endl; //输出映射矩阵

//图像配准
Mat imageTransform1, imgpeizhun, imgerzhi;
warpPerspective(image01, imageTransform1, homo, Size(image02.cols,
image02.rows));
imshow("经过透视矩阵变换后", imageTransform1);

absdiff(image02, imageTransform1, imgpeizhun);
imshow("配准 diff", imgpeizhun);

threshold(imgpeizhun, imgerzhi, 40, 255.0, CV_THRESH_BINARY);
imshow("配准二值化", imgerzhi);

//输出所需时间
time0 = ((double)getTickCount() - time0) / getTickFrequency();
cout << 1 / time0 << endl;

```

```

Mat temp, image02temp;
float m_BiLi = 0.9;

image02temp = image02.clone();
cvtColor(imgerzhi, temp, CV_RGB2GRAY);

//检索连通域
Mat se = getStructuringElement(MORPH_RECT, Size(10, 10));
Mat result_temp;
//dilate(temp, result_temp, Mat(), Point(-1, -1), 3, BORDER_DEFAULT);
morphologyEx(temp, result_temp, MORPH_DILATE, se);

//vector<vector<Point>> contours;
//findContours(result_temp, contours, RETR_EXTERNAL,
CHAIN_APPROX_NONE);

//for (int k = 0; k < contours.size(); k++)
//{
//    Rect bomen = boundingRect(contours[k]);

//    //省略由于配准带来的边缘无效信息
//    if (bomen.x > image02temp.cols * (1 - m_BiLi) && bomen.y >
image02temp.rows * (1 - m_BiLi)
//        && bomen.x + bomen.width < image02temp.cols * m_BiLi &&
bomen.y + bomen.height < image02temp.rows * m_BiLi)
//    {
//        //rectangle(image02temp, bomen, Scalar(255, 0, 255), 2, 8, 0);
//        rectangle(image02temp, bomen, Scalar(255, 255, 255), 2, 8, 0);
//        //threshold(image02temp, image02temp, 0, 255,
CV_THRESH_BINARY);
//    }

//}

/*
for (int i = 50; i < image02.rows - 100; i++)
{
for (int j = 50; j < image02.cols - 100; j++)
{
uchar pixel = temp.at<uchar>(i,j);
if (pixel == 255)
{
Rect bomen(j-7, i-7, 14, 14);

```

```

        rectangle(image02, bomen, Scalar(255,255,255),1,8,0);
    }
}
*/
imshow("检测与跟踪", image02temp);
//return  image02temp;
return temp;
//waitKey(20);
}

```

Removesmall.cpp

```

#include"removesmall.h"

```

```

void RemoveSmallRegion(Mat &Src, Mat &Dst, int AreaLimit, int CheckMode, int
NeihborMode)

```

```

{
    int RemoveCount = 0;
    //新建一幅标签图像初始化为 0 像素点，为了记录每个像素点检验状态的标
    签，0 代表未检查，1 代表正在检查,2 代表检查不合格（需要反转颜色），3 代
    表检查合格或不需检查
    //初始化的图像全部为 0，未检查
    Mat PointLabel = Mat::zeros(Src.size(), CV_8UC1);
    if (CheckMode == 1)//去除小连通区域的白色点
    {
        cout << "去除小连通域.";
        for (int i = 0; i < Src.rows; i++)
        {
            for (int j = 0; j < Src.cols; j++)
            {
                if (Src.at<uchar>(i, j) < 10)
                {
                    PointLabel.at<uchar>(i, j) = 3;//将背景黑色点标记为合格，像
                    素为 3
                }
            }
        }
    }
    else//去除孔洞，黑色点像素
    {
        cout << "去除孔洞";
        for (int i = 0; i < Src.rows; i++)
        {

```

```

        for (int j = 0; j < Src.cols; j++)
        {
            if (Src.at<uchar>(i, j) > 10)
            {
                PointLabel.at<uchar>(i, j) = 3; //如果原图是白色区域，标记为
合格，像素为 3
            }
        }
    }
}

```

```

vector<Point2i>NeighborPos; //将邻域压进容器
NeighborPos.push_back(Point2i(-1, 0));
NeighborPos.push_back(Point2i(1, 0));
NeighborPos.push_back(Point2i(0, -1));
NeighborPos.push_back(Point2i(0, 1));
if (NeighborMode == 1)
{
    cout << "Neighbor mode: 8 邻域." << endl;
    NeighborPos.push_back(Point2i(-1, -1));
    NeighborPos.push_back(Point2i(-1, 1));
    NeighborPos.push_back(Point2i(1, -1));
    NeighborPos.push_back(Point2i(1, 1));
}
else cout << "Neighbor mode: 4 邻域." << endl;
int NeighborCount = 4 + 4 * NeighborMode;
int CurrX = 0, CurrY = 0;
//开始检测
for (int i = 0; i < Src.rows; i++)
{
    for (int j = 0; j < Src.cols; j++)
    {
        if (PointLabel.at<uchar>(i, j) == 0) //标签图像像素点为 0，表示还未检
查的不合格点
        {
            //开始检查
            vector<Point2i>GrowBuffer; //记录检查像素点的个数
            GrowBuffer.push_back(Point2i(j, i));
            PointLabel.at<uchar>(i, j) = 1; //标记为正在检查
            int CheckResult = 0;

            for (int z = 0; z < GrowBuffer.size(); z++)
            {

```

```

        for (int q = 0; q < NeihborCount; q++)
        {
            CurrX = GrowBuffer.at(z).x + NeihborPos.at(q).x;
            CurrY = GrowBuffer.at(z).y + NeihborPos.at(q).y;
            if (CurrX >= 0 && CurrX<Src.cols&&CurrY >= 0 &&
CurrY<Src.rows) //防止越界
            {
                if (PointLabel.at<uchar>(CurrY, CurrX) == 0)
                {
                    GrowBuffer.push_back(Point2i(CurrX, CurrY));
//邻域点加入 buffer
                    PointLabel.at<uchar>(CurrY, CurrX) = 1;
//更新邻域点的检查标签，避免重复检查
                }
            }
        }
        if (GrowBuffer.size()>AreaLimit) //判断结果（是否超出限定的大小），1 为未超出，2 为超出
            CheckResult = 2;
        else
        {
            CheckResult = 1;
            RemoveCount++;//记录有多少区域被去除
        }

        for (int z = 0; z < GrowBuffer.size(); z++)
        {
            CurrX = GrowBuffer.at(z).x;
            CurrY = GrowBuffer.at(z).y;
            PointLabel.at<uchar>(CurrY, CurrX) += CheckResult;//标记不
合格的像素点，像素值为 2
        }
        //*****结束该点处的检查*****
    }
}

}

```

```

    CheckMode = 255 * (1 - CheckMode);
    //开始反转面积过小的区域
    for (int i = 0; i < Src.rows; ++i)
    {
        for (int j = 0; j < Src.cols; ++j)
        {
            if (PointLabel.at<uchar>(i, j) == 2)
            {
                Dst.at<uchar>(i, j) = CheckMode;
            }
            else if (PointLabel.at<uchar>(i, j) == 3)
            {
                Dst.at<uchar>(i, j) = Src.at<uchar>(i, j);
            }
        }
    }
    cout << RemoveCount << " objects removed." << endl;
}

```

问题四实现程序

Main.m

```

clear all
close all
clc
pics = dir('F:\md5\代码\vibe-sources\vibe-sources\vibeplus\vibeplus\result\*.jpg');
nums=length(pics)
path='F:\md5\代码\vibe-sources\vibe-sources\vibeplus\vibeplus\result';
cd(path);    %切换目录
for i=1:nums
    STR=sprintf('%s%d','image',i);
    filename=[STR,'.jpg'];
    img_in=imread(filename);
    if i==1
        img=rgb2gray(img_in);
        a(i)=0;
    else
        img=rgb2gray(img_in);
        %img=img_in;
        [m,n]=size(img);
        aver(i)=sum(sum(img))/255/(m*n);
    end
end
end

```

```

aver(nums+1)=0;

max(aver);
plot(aver);
%axis([0,200,0,1])
%set(handles,'xtick',0:1:200)          % handles 可以指定具体坐标轴的句柄
%set(gca,'YTick',[0:0.1:1]) %改变 y 轴坐标间隔显示 这里间隔为 10

%size(peaksign)
%对 aver 数组进行计算

%检测跳变物体
tiao_num=0;
[peakvalue,peaksign]=findpeaks(aver,'minpeakheight',max(aver)*0.01,'minpeakdistance',30)
for i=1:length(peaksign)
    left=0;
    right=0;
    for j=1:2
        if abs(peakvalue(i)-aver(peaksign(i)-j))>0.8*peakvalue(i)
            left=peaksign(i)-j+1;
            %disp('find left');
            break;
        end
    end
    for j=1:2
        if abs(peakvalue(i)-aver(peaksign(i)+j))>0.8*peakvalue(i)
            right=peaksign(i)+j-1;
            %disp('find right');
            break;
        end
    end
    if (left==0 | right==0)

        %disp('非跳变物体点=====');
        continue;
    end

    tiao_num=tiao_num+1;
    tiao_frame(tiao_num)=peaksign(i);
    str='跳变物体目标出现在';
    putout=[str,num2str(peaksign(i)),'帧'];
    disp(putout);
    num_frame_fore=[left,peaksign(i),right];

```



```

        %disp('跳变物体=====');
end

%将跳变帧归零，再进行高斯去噪平滑。
for i=1:tiao_num
    aver(tiao_frame(i))=0;
end
figure();
plot(aver);

aver_gauss = Gaussianfilter(30, 10, aver);
I=aver_gauss;
%I=imnoise(aver,'salt & pepper',0.02);
%I=imnoise(aver,'gaussian',1);
%I = medfilt1(aver,5)
figure();
plot(aver_gauss);

[peakvalue,peaksign]=findpeaks(aver,'minpeakheight',max(aver)*0.1,'minpeakdistance',50)
%检测显著前景
abc=0;
for i=1:length(peaksign)
    for j=1:1000
        if abs(peakvalue(i)-I(peaksign(i)-j))>0.9*peakvalue(i)
            left=peaksign(i)-j+1;
            %disp('find left');
            break;
        end
    end
    for j=1:1000
        if abs(peakvalue(i)-I(peaksign(i)+j))>0.9*peakvalue(i)
            right=peaksign(i)+j-1;
            %disp('find right');
            break;
        end
    end
    str='显著前景目标出现在';

```

```

        putout=[str,num2str(left),'-',num2str(right),'帧'];
        disp(putout);
        num_frame_fore=[left,peaksign(i),right];
        abc=abc+1;
        %out_frame_num[abc]=left;
        out_frame_num(abc,1)=left;
        out_frame_num(abc,2)=right;
        ann=[left,right];
    end
    %整合重合的帧范围
    for i=1:abc
        for j=1:abc-i
            if out_frame_num(j,1)>out_frame_num(j+1,1)
                t1=out_frame_num(j,1);
                out_frame_num(j,1)=out_frame_num(j+1,1);
                out_frame_num(j+1,1)=t1;
                t2=out_frame_num(j,2);
                out_frame_num(j,2)=out_frame_num(j+1,2);
                out_frame_num(j+1,2)=t2;
            end
        end
    end

    flag=1;
    out(flag,1)=out_frame_num(1,1);
    out(flag,2)=out_frame_num(1,2);
    %out[flag][2]=out_frame_num[1][2];
    for i=2:abc
        if out_frame_num(i,1)<out(flag,2)
            if out_frame_num(i,2)>out(flag,2)
                out(flag,2)=out_frame_num(i,2);
            else
                continue;
            end
        else
            flag=flag+1;
            out(flag,1)=out_frame_num(i,1);
            out(flag,2)=out_frame_num(i,2);
        end
    end
    end
    disp('-----');
    for i=1:flag
        str='显著前景目标出现在';
        putout=[str,num2str(out(i,1)),'-',num2str(out(i,2)),'帧'];
        disp(putout);
    end

```

End

=====

Gaussianfilter.m

% 功能：对一维信号的高斯滤波，头尾 $r/2$ 的信号不进行滤波

% r :高斯模板的大小推荐奇数

% sigma :标准差

% y :需要进行高斯滤波的序列

function y_filted = Gaussianfilter(r, sigma, y)

% 生成一维高斯滤波模板

GaussTemp = ones(1,r*2-1);

for i=1 : r*2-1

 GaussTemp(i) = exp(-(i-r)^2/(2*sigma^2))/(sigma*sqrt(2*pi));

end

% 高斯滤波

y_filted = y;

for i = r : length(y)-r

 y_filted(i) = y(i-r+1 : i+r-1)*GaussTemp';

end

=====

问题五实现程序

Homo_mat.cpp

#define _CRT_SECURE_NO_WARNINGS

#include <iostream>

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv/cv.h>

#include <opencv2/nonfree/features2d.hpp>

#include <opencv2/legacy/legacy.hpp>

#include <vector>

#include <time.h>

using namespace cv;

using namespace std;

void getAffineMat(const Mat* img_ref, const Mat* img_shake, double** affine_mat)
{

 vector<KeyPoint> keyPoint1;

 vector<KeyPoint> keyPoint2;

```

SURF surf(500);
surf.detect(*img_shake, keyPoint1);
surf.detect(*img_ref, keyPoint2);
Mat descriImage1, descriImage2;
surf.compute(*img_shake, keyPoint1, descriImage1);
surf.compute(*img_ref, keyPoint2, descriImage2);

BruteForceMatcher<L2<float>>matcher;
vector<DMatch>matches;
matcher.match(descriImage1, descriImage2, matches);

vector<Point2f> srcPoints(matches.size());
vector<Point2f> dstPoints(matches.size());

for (size_t i = 0; i < 35; i++) {
    srcPoints[i] = keyPoint1[matches[i].trainIdx].pt;
    dstPoints[i] = keyPoint2[matches[i].queryIdx].pt;
}

//find homography matrix and get inliers mask
vector<uchar> inliersMask(srcPoints.size());
Mat homography = findHomography(srcPoints, dstPoints, CV_FM_RANSAC, 3,
inliersMask);
cout << homography << endl;
vector<DMatch> inliers;
for (size_t i = 0; i < inliersMask.size(); i++){
    if (inliersMask[i])
        inliers.push_back(matches[i]);
}
matches.swap(inliers);

//int index = 0;
//for (unsigned i = 0; i < matches.size(); i++) {
//    if (status[i] != 0){
//        leftInlier.push_back(alignedKps1[i]);
//        rightInlier.push_back(alignedKps2[i]);
//        matches[i].trainIdx = index;
//        matches[i].queryIdx = index;
//        inlierMatch.push_back(matches[i]);
//        index++;
//    }
//}
//leftPattern->keypoints = leftInlier;

```

```

//rightPattern->keypoints = rightInlier;
//matches = inlierMatch;

//暴力匹配 orflann 匹配
/*FlannBasedMatcher FLMatcher;
vector<DMatch> g_vMatches;
FLMatcher.match(descriImage1, descriImage2, g_vMatches);*/

//double minDistance = g_vMatches[0].distance, maxDistance =
g_vMatches[0].distance;
//for (size_t i = 0; i < keyPoint1.size(); i++)
//{
//    double currDistance = g_vMatches[i].distance;
//    if (currDistance < minDistance)
//        minDistance = currDistance;
//    if (currDistance > maxDistance)
//        maxDistance = currDistance;
//}

////定义一个新的变量，用来存储 通过距离检测后 通过阈值的点
//vector<DMatch> newMatches;
//int count = 0;
//for (size_t i = 0; i < keyPoint1.size(); i++)
//{

//    if (g_vMatches[i].distance <= 10 * minDistance)
//    {
//        newMatches.push_back(g_vMatches[i]);
//        count++;

//        if (count > 10)
//            break;
//    }
//}

//nth_element(newMatches.begin(), newMatches.begin() + 4, newMatches.end());
//newMatches.erase(newMatches.begin() + 5, newMatches.end());

//Point2f srcTri[3];
//Point2f dstTri[3];
//Point2f A1 = keyPoint1[newMatches[0].queryIdx].pt;
//Point2f B1 = keyPoint2[newMatches[0].trainIdx].pt;

```

```

//Point2f A2 = keyPoint1[newMatches[1].queryIdx].pt;
//Point2f B2 = keyPoint2[newMatches[1].trainIdx].pt;
//Point2f A3 = keyPoint1[newMatches[2].queryIdx].pt;
//Point2f B3 = keyPoint2[newMatches[2].trainIdx].pt;

//srcTri[0] = A1;
//dstTri[0] = B1;
//srcTri[1] = A2;
//dstTri[1] = B2;
//srcTri[2] = A3;
//dstTri[2] = B3;

//Mat affine(2, 3, CV_64FC1, Scalar(0.0));
//affine = getAffineTransform(srcTri, dstTri);
//cout << affine << endl;
//Mat affine_T(3, 2, CV_64FC1, Scalar(0.0));
//affine_T = affine.t();
//affine_mat[2][2] = 1;

//for (int i = 0; i < 3; i++)
//{
//    for (int j = 0; j < 2; j++)
//    {
//        affine_mat[i][j] = affine_T.at<double>(i, j);
//    }
//}
}

```

```

int main()
{
    double **aff = new double*[3]();
    for (int i = 0; i < 3; i++)
    {
        aff[i] = new double[3]();
    }
    //Mat* img_shake = new Mat();
    //Mat* img_ref = new Mat();
    //img_shake->create(512, 512, CV_64FC1);
    //img_ref->create(512, 512, CV_64FC1);

    Mat img_ref = imread("GF4_sub8_2.tiff", CV_LOAD_IMAGE_ANYDEPTH);

```

```

    Mat img_shake = imread("GF4_sub8_3.tiff",
CV_LOAD_IMAGE_ANYDEPTH);

```

```

img_ref.convertTo(img_ref, CV_64FC1);
img_shake.convertTo(img_shake, CV_64FC1);

```

```

img_ref = img_ref / 4;
img_shake = img_shake / 4;

```

```

img_ref.convertTo(img_ref, CV_8UC1);
img_shake.convertTo(img_shake, CV_8UC1);

```

```

clock_t start, end;
start = clock();

```

```

getAffineMat(&img_ref, &img_shake, aff);
end = clock();
cout << "time = " << double(end - start) / CLK_TCK << " sec" << endl;

```

```

for (int i = 0; i < 3; ++i)
{
    delete[] aff[i];
}
delete[] aff;

```

```

return 0;
}

```

```

=====
=====

```

Main.cpp

```

#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv/cv.h>
#include <opencv2/nonfree/features2d.hpp>
#include <opencv2/legacy/legacy.hpp>

```

```

using namespace std;
using namespace cv;

```

```

int main()
{
    Mat A = imread("A.bmp");
    //cvNamedWindow("A");
    //imshow("A",A);

    Mat B = imread("B.bmp");
    //cvNamedWindow("B");
    //imshow("B", B);

    // ===== HARRIS
    ===== //
    /// 检测 Harris 角点
    //Mat cornerStrength;
    //cornerHarris(A, cornerStrength, 3, 3, 0.01);

    /// 角点强度的阈值
    //Mat harrisCorners;
    //double threshold = 0.0001;
    //cv::threshold(cornerStrength, harrisCorners, threshold, 255,
cv::THRESH_BINARY_INV);
    //imshow("corner", harrisCorners);

    // ===== FAST =====
    //
    //Mat fast_img;
    //A.copyTo(fast_img);
    //vector<cv::KeyPoint> keypoints_fast;
    //FastFeatureDetector fast(10);
    //fast.detect(fast_img, keypoints_fast);

    //drawKeypoints(fast_img, keypoints_fast, fast_img, Scalar(0, 120, 220),
DrawMatchesFlags::DRAW_OVER_OUTIMG);
    //imshow("A_fast", fast_img);

    // ===== SURF ===== //
    Mat surf_img_A;
    A.copyTo(surf_img_A);
    vector<cv::KeyPoint> keypoints_surf_A;
    SurfFeatureDetector surf_A(2500);
    surf_A.detect(surf_img_A, keypoints_surf_A);

```



```

    drawKeypoints(surf_img_A, keypoints_surf_A, surf_img_A, Scalar(0, 128, 228),
DrawMatchesFlags::DRAW_RICH_KEYPOINTS);
    imshow("A_surf", surf_img_A);

    Mat surf_img_B;
    B.copyTo(surf_img_B);
    vector<cv::KeyPoint> keypoints_surf_B;
    SurfFeatureDetector surf_B(2500);
    surf_B.detect(surf_img_B, keypoints_surf_B);
    drawKeypoints(surf_img_B, keypoints_surf_B, surf_img_B, Scalar(0, 128, 228),
DrawMatchesFlags::DRAW_RICH_KEYPOINTS);
    imshow("B_surf", surf_img_B);

    // ===== SURF Descriptor ===== //
    SurfDescriptorExtractor surfDesc;
    Mat descriptors_surf_A, descriptors_surf_B;
    surfDesc.compute(surf_img_A, keypoints_surf_A, descriptors_surf_A);
    surfDesc.compute(surf_img_B, keypoints_surf_B, descriptors_surf_B);

    // ===== SURF matcher ===== //
    BruteForceMatcher<cv::L2<float>> matcher;
    // 匹配两幅图像的描述子
    vector<cv::DMatch> matches;
    matcher.match(descriptors_surf_A, descriptors_surf_B, matches);
    int match_num = 10;
    std::nth_element(matches.begin(), matches.begin() + match_num - 1,
matches.end());
    matches.erase(matches.begin() + match_num, matches.end());

    Mat imageMatches;
    drawMatches(A, keypoints_surf_A, B, keypoints_surf_B, matches,
imageMatches, Scalar(0, 128, 228));
    imshow("surf Matches", imageMatches);

    // ===== 单应矩阵 ===== //
    //vector<uchar> inliers(keypoints_surf_A.size(), 0);
    //Mat homography = findHomography(keypoints_surf_A, keypoints_surf_B,
CV_RANSAC, 1.0, inliers);
    //Mat affine_mat = getAffineTransform(keypoints_surf_A, keypoints_surf_B);

```

```

// 创建匹配点

Point2f srcTri[3];
Point2f dstTri[3];

Mat warp_mat(2, 3, CV_32FC1);

Point2f A1 = keypoints_surf_A[matches[0].trainIdx].pt;
Point2f B1 = keypoints_surf_B[matches[0].queryIdx].pt;

Point2f A2 = keypoints_surf_A[matches[1].trainIdx].pt;
Point2f B2 = keypoints_surf_B[matches[1].queryIdx].pt;

Point2f A3 = keypoints_surf_A[matches[2].trainIdx].pt;
Point2f B3 = keypoints_surf_B[matches[2].queryIdx].pt;


srcTri[0] = A1;
dstTri[0] = B1;

srcTri[1] = A2;
dstTri[1] = B2;

srcTri[2] = A3;
dstTri[2] = B3;
Mat affine_mat = getAffineTransform(srcTri, dstTri);

cout << affine_mat << endl;


waitKey(0);


return 0;

}
=====

```

```

=====
Main_1l.cpp
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv/cv.h>
#include <opencv2/nonfree/features2d.hpp>
#include <opencv2/legacy/legacy.hpp>
#include <vector>

using namespace cv;
using namespace std;

int main()
{
    Mat srcImage1 = imread("lena.jpg", 0);
    Mat srcImage2 = imread("lena1.jpg", 0);
    imshow("【原图 1】", srcImage1);
    imshow("【原图 2】", srcImage2);

    //首先对两幅图像进行特征点的检测
    //先准备参数
    vector<KeyPoint> keyPoint1;
    vector<KeyPoint> keyPoint2;
    SURF surf;
    surf.detect(srcImage1, keyPoint1);
    surf.detect(srcImage2, keyPoint2);

    //利用得到的特征点计算特征描述子
    //目的：对得到的每个特征点进行特征描述，整合到 Mat 类型的矩阵中（计算结果是 Mat 类型的）
    //该得到的结果矩阵的行数就是特征点的个数，因为是对每个点进行描述，所以每行都会有一个描述的字子向量，共同构成 Mat 矩阵
    Mat descriImage1, descriImage2;
    surf.compute(srcImage1, keyPoint1, descriImage1);
    surf.compute(srcImage2, keyPoint2, descriImage2);

    //正式开始在两幅图像中进行匹配
    //先得到一个匹配向量
    //FlannBasedMatcher FLMatcher;
    //vector<DMatch> g_vMatches;
    ////g_vMatches 就是得到的匹配向量

```

```

        //FLMatcher.match(descriImage1, descriImage2, g_vMatches);
        //Mat midImage;
        //drawMatches(srcImage1, keyPoint1, srcImage2, keyPoint2, g_vMatches,
midImage
        // , Scalar(theRNG().uniform(0, 255), theRNG().uniform(0, 255),
theRNG().uniform(0, 255))
        // , Scalar(theRNG().uniform(0, 255), theRNG().uniform(0, 255),
theRNG().uniform(0, 255)), Mat(), 2);
        //imshow("【直接匹配的图像】", midImage);
        BruteForceMatcher<L2<float>>matcher;
        vector<vector<DMatch>>matches1;
        matcher.knnMatch(descriImage1, descriImage2, matches1, 2);//返回从图一到图
二两个最近邻
        vector<vector<DMatch>>matches2;
        matcher.knnMatch(descriImage2, descriImage1, matches2, 2);//返回从图二到图
一两个最近邻

        //移除 NN 比率大于阈值的匹配
        int removed = ratioTest(matches1);
        removed = ratioTest(matches2);
        vector<vector<DMatch>>symMatches;
        symmetryTest(matches1, matches2, symMatches);
        //使用 ransac 进行最终验证
        Mat fundamental = ransacTest(symMatches, keyPoint1, keyPoint2, matches);
    }
}

```

```

=====
=====
My_main.cpp
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv/cv.h>
#include <opencv2/nonfree/features2d.hpp>
#include <opencv2/legacy/legacy.hpp>
#include <vector>

```

```

using namespace cv;
using namespace std;

```

```

int main()
{
    Mat srcImage2 = imread("A.bmp", 0);

```

```

Mat srcImage1 = imread("B.bmp",0);
imshow("【原图 1】", srcImage1);
imshow("【原图 2】", srcImage2);

//首先对两幅图像进行特征点的检测
//先准备参数
vector<KeyPoint> keyPoint1;
vector<KeyPoint> keyPoint2;
SURF surf;
surf.detect(srcImage1, keyPoint1);
surf.detect(srcImage2, keyPoint2);

//利用得到的特征点计算特征描述子
//目的：对得到的每个特征点进行特征描述，整合到 Mat 类型的矩阵中（计算结果是 Mat 类型的）
//该得到的结果矩阵的行数就是特征点的个数，因为是对每个点进行描述，所以每行都会有一个描述的字子向量，共同构成 Mat 矩阵
Mat descriImage1, descriImage2;
surf.compute(srcImage1, keyPoint1, descriImage1);
surf.compute(srcImage2, keyPoint2, descriImage2);

//正式开始在两幅图像中进行匹配
//先得到一个匹配向量
FlannBasedMatcher FLMatcher;
vector<DMatch> g_vMatches;
//g_vMatches 就是得到的匹配向量
FLMatcher.match(descriImage1, descriImage2, g_vMatches);
Mat midImage;
drawMatches(srcImage1, keyPoint1, srcImage2, keyPoint2, g_vMatches,
midImage
, Scalar(theRNG().uniform(0, 255), theRNG().uniform(0, 255),
theRNG().uniform(0, 255))
, Scalar(theRNG().uniform(0, 255), theRNG().uniform(0, 255),
theRNG().uniform(0, 255)), Mat(), 2);
imshow("【直接匹配的图像】", midImage);

//用找最大最小值的方式找到 两幅图像中匹配的点的距离的最大值和最小值
//这里的 keyPoint1.size() 和 descriImage1.rows 是一样的值，因为 descriImage1 的行数就是检测到的特征点的个数
double minDistance = g_vMatches[0].distance, maxDistance =
g_vMatches[0].distance;
for (size_t i = 0; i < keyPoint1.size(); i++)
{

```

```

        double currDistance = g_vMatches[i].distance;
        if (currDistance < minDistance)
            minDistance = currDistance;
        if (currDistance > maxDistance)
            maxDistance = currDistance;
    }

    //定义一个新的变量，用来存储 通过距离检测后 通过阈值的点
    vector<DMatch> newMatches;
    int count = 0;
    for (size_t i = 0; i < keyPoint1.size(); i++)
    {

        if (g_vMatches[i].distance <= 10 * minDistance)
        {
            newMatches.push_back(g_vMatches[i]);
            count++;
            if (count > 10)
                break;
        }
    }

    //用绘制函数对匹配向量进行绘制
    Mat dstImage;
    drawMatches(srcImage1, keyPoint1, srcImage2, keyPoint2, newMatches,
dstImage
        , Scalar(theRNG().uniform(0, 255), theRNG().uniform(0, 255),
theRNG().uniform(0, 255))
        , Scalar(theRNG().uniform(0, 255), theRNG().uniform(0, 255),
theRNG().uniform(0, 255)), Mat(), 2);

    imshow("【特征提取后的图像】", dstImage);

    nth_element(newMatches.begin(), newMatches.begin() + 4, newMatches.end());
    newMatches.erase(newMatches.begin() + 5, newMatches.end());
    //输入为二维点集和
    Point2f srcTri[3];
    Point2f dstTri[3];

```

```

Point2f A1 = keyPoint1[newMatches[0].queryIdx].pt;
Point2f B1 = keyPoint2[newMatches[0].trainIdx].pt;

Point2f A2 = keyPoint1[newMatches[1].queryIdx].pt;
Point2f B2 = keyPoint2[newMatches[1].trainIdx].pt;

Point2f A3 = keyPoint1[newMatches[2].queryIdx].pt;
Point2f B3 = keyPoint2[newMatches[2].trainIdx].pt;

srcTri[0] = A1;
dstTri[0] = B1;

srcTri[1] = A2;
dstTri[1] = B2;

srcTri[2] = A3;
dstTri[2] = B3;

Mat affine_mat = getAffineTransform(srcTri, dstTri);

//cout << affine_mat << endl;
//warpAffine
//C++: void warpAffine(InputArray src, OutputArray dst, InputArray M, Size
dsizesize, int flags = INTER_LINEAR, intborderMode = BORDER_CONSTANT, const
Scalar& borderValue = Scalar())
// 第一个参数，InputArray 类型的 src，输入图像，即源图像，填 Mat 类的
对象即可。
// 第二个参数，OutputArray 类型的 dst，函数调用后的运算结果存在这里，
需和源图片有一样的尺寸和类型。
// 第三个参数，InputArray 类型的 M，2×3 的变换矩阵。
// 第四个参数，Size 类型的 dsizesize，表示输出图像的尺寸。
// 第五个参数，int 类型的 flags，插值方法的标识符。此参数有默认值
INTER_LINEAR(线性插值)，可选的插值方式如下：
/* INTER_NEAREST - 最近邻插值
INTER_LINEAR - 线性插值（默认值）
INTER_AREA - 区域插值
INTER_CUBIC - 三次样条插值
INTER_LANCZOS4 - Lanczos 插值
CV_WARP_FILL_OUTLIERS - 填充所有输出图像的像素。如果部分象
素落在输入图像的边界外，那么它们的值设定为 fillval.
CV_WARP_INVERSE_MAP - 表示 M 为输出图像到输入图像的反变
换，即 。因此可以直接用来做像素插值。否则，warpAffine 函数从 M 矩阵得到
反变换。
第六个参数，int 类型的 borderMode，边界像素模式，默认值为

```

BORDER_CONSTANT。

第七个参数，const Scalar&类型的 borderValue，在恒定的边界情况下取的值，默认值为 Scalar()，即 0。*/

```
Size dsize = srcImage2.size();
Mat dst_img(srcImage2.size().height, srcImage2.size().width, CV_64FC1,
Scalar(0));
warpAffine(srcImage2, dst_img, affine_mat, dsize, INTER_CUBIC,
BORDER_CONSTANT, 0);
imshow("输出图像", dst_img);
```

```
Mat affine_mat_33(3, 3, CV_64FC1, Scalar(0));
for (int i = 0; i < 2; i++)
{
    for (int j = 0; j < 3; j++)
    {
        affine_mat_33.at<double>(i, j) = affine_mat.at<double>(i, j);
    }
}

affine_mat_33.at<double>(2, 2) = 1;
cout << affine_mat_33 << endl;

Mat affine_mat33_t(3, 3, CV_64FC1, Scalar(0));

affine_mat33_t = affine_mat_33.t();
cout << affine_mat33_t << endl;

waitKey(0);
return 0;
}
```

```
=====
=====
```

Refine_affine.cpp

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <vector>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/features2d/features2d.hpp>
#include <opencv2/calib3d/calib3d.hpp>
#include <opencv2/nonfree/nonfree.hpp>
#include <opencv2/legacy/legacy.hpp>
```



```

#include "opencv2/nonfree/features2d.hpp"
#include <opencv2/opencv.hpp>
#include <time.h>

using namespace cv;
using namespace std;
int ratioTest(vector<vector<DMatch>>& matches)
{
    int removed = 0;
    double ratio = 0.65;
    //遍历所有匹配
    vector<vector<DMatch>>::iterator matchIt;
    for (matchIt = matches.begin(); matchIt != matches.end(); matchIt++)
    {
        if (matchIt->size() > 1)
        {
            if (((*matchIt)[0].distance / (*matchIt)[1].distance) > ratio)
            {
                matchIt->clear();
                removed++;
            }
        }
        else
        {
            matchIt->clear();
            removed++;
        }
    }
    return removed;
}

void symmetryTest(vector<vector<DMatch>>& matches1,
vector<vector<DMatch>>& matches2, vector<DMatch>& symMatches)
{
    vector<vector<DMatch>>::iterator matchIt1;
    vector<vector<DMatch>>::iterator matchIt2;
    for (matchIt1 = matches1.begin(); matchIt1 != matches1.end(); matchIt1++)
    {
        if (matchIt1->size() < 2)
        {
            continue;
        }
        for (matchIt2 = matches2.begin(); matchIt2 != matches2.end(); matchIt2++)
        {

```

```

        if (matchIt2->size() < 2)
        {
            continue;
        }
        //对成型测试
        if ((*matchIt1)[0].queryIdx == (*matchIt2)[0].trainIdx &&
(*matchIt2)[0].queryIdx == (*matchIt1)[0].trainIdx)
        {
            //添加对称的匹配
            symMatches.push_back(
                DMatch((*matchIt1)[0].queryIdx,
                    (*matchIt1)[0].trainIdx,
                    (*matchIt1)[0].distance ));
            break;
        }
    }
}

Mat ransacTest(vector<DMatch>& matches, vector<KeyPoint> keypoints_1,
vector<KeyPoint> keypoints_2, vector<DMatch>& out_match)
{
    vector<Point2f> points1, points2;
    vector<DMatch>::iterator matchIt;
    for (matchIt = matches.begin(); matchIt != matches.end(); matchIt++)
    {
        float x = keypoints_1[matchIt->queryIdx].pt.x;
        float y = keypoints_1[matchIt->queryIdx].pt.y;
        points1.push_back(Point2f(x, y));

        x = keypoints_2[matchIt->trainIdx].pt.x;
        y = keypoints_2[matchIt->trainIdx].pt.y;
        points2.push_back(Point2f(x, y));
    }
    vector<uchar> inliners(points1.size(), 0);
    /*Mat fundamental = findFundamentalMat(Mat(points1), Mat(points2), inliners,
CV_FM_RANSAC, 3, 0.99);
    cout << "基础矩阵:"<< endl << fundamental << endl;*/

    Mat affine_mat = estimateRigidTransform(points1, points2, true);
    cout << "刚体变换矩阵:" << endl << affine_mat << endl;

    //Mat homo_mat = findHomography(Mat(points1), Mat(points2), inliners,
CV_FM_RANSAC, 3.0);
    ///cout << "单应矩阵:" << endl << homo_mat << endl;

```

```

    return affine_mat;
}
void getAffineMat(const Mat* img_ref, const Mat* img_shake, double* affine_mat)
{
    double minHessian = 0.1;
    SurfFeatureDetector detector(minHessian);
    std::vector<KeyPoint> keypoints_1, keypoints_2;
    detector.detect(*img_shake, keypoints_1);
    detector.detect(*img_ref, keypoints_2);

    SurfDescriptorExtractor extractor;
    Mat descriptors_1, descriptors_2;
    extractor.compute(*img_shake, keypoints_1, descriptors_1);
    extractor.compute(*img_ref, keypoints_2, descriptors_2);

    BFMatcher matcher;
    vector< DMatch > out_match;
    vector< vector<DMatch> > matches1;
    matcher.knnMatch(descriptors_1, descriptors_2, matches1,2);
    vector< vector<DMatch> > matches2;
    matcher.knnMatch(descriptors_2, descriptors_1, matches2, 2);
    int removed = ratioTest(matches1);
    removed = ratioTest(matches2);

    vector<DMatch> symMatches;
    symmetryTest(matches1, matches2, symMatches);
    Mat fundamental = ransacTest(symMatches, keypoints_1, keypoints_2,
out_match);
    //for (int i = 0; i < 2; i++)
    //{
    //    for (int j = 0; j < 3; j++)
    //    {
    //        affine_mat[i * 3 + j] = fundamental.at<double>(i,j);
    //    }
    //}

}

int main()
{
    double aff[9] = {0};
    Mat img_ref = imread("new_sub3_1.tiff", CV_LOAD_IMAGE_ANYDEPTH);
    Mat img_shake = imread("new_sub3_2.tiff",
CV_LOAD_IMAGE_ANYDEPTH);

```

```

img_ref.convertTo(img_ref, CV_64FC1);
img_shake.convertTo(img_shake, CV_64FC1);

img_ref = img_ref / 4;
img_shake = img_shake / 4;

img_ref.convertTo(img_ref, CV_8UC1);
img_shake.convertTo(img_shake, CV_8UC1);

clock_t start, end;
start = clock();

getAffineMat(&img_ref, &img_shake, aff);
end = clock();
//for (int i = 0; i < 9; i++)
//{
//    cout << aff[i] << endl;
//}
cout << "time = " << double(end - start) / CLK_TCK << " sec" << endl;
return 0;
}

```

```

=====
=====

```

Test.cpp

```

#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv/cv.h>
#include <opencv2/nonfree/features2d.hpp>
#include <opencv2/legacy/legacy.hpp>
#include <vector>
#include <time.h>
using namespace cv;
using namespace std;

void getAffineMat(const Mat* img_ref, const Mat* img_shake, double* affine_mat)

```

```

{

    vector<KeyPoint> keyPoint1;
    vector<KeyPoint> keyPoint2;
    SURF surf;
    surf.detect(*img_shake, keyPoint1);
    surf.detect(*img_ref, keyPoint2);
    Mat descriImage1, descriImage2;
    surf.compute(*img_shake, keyPoint1, descriImage1);
    surf.compute(*img_ref, keyPoint2, descriImage2);
    //暴力匹配 orflann 匹配
    FlannBasedMatcher FLMatcher;
    vector<DMatch> g_vMatches;
    FLMatcher.match(descriImage1, descriImage2, g_vMatches);

    double    minDistance    =    g_vMatches[0].distance,    maxDistance    =
g_vMatches[0].distance;
    for (size_t i = 0; i < keyPoint1.size(); i++)
    {
        double currDistance = g_vMatches[i].distance;
        if (currDistance < minDistance)
            minDistance = currDistance;
        if (currDistance > maxDistance)
            maxDistance = currDistance;
    }

    //定义一个新的变量，用来存储 通过距离检测后 通过阈值的点
    vector<DMatch> newMatches;
    int count = 0;
    for (size_t i = 0; i < keyPoint1.size(); i++)
    {

        if (g_vMatches[i].distance <= 10 * minDistance)
        {
            newMatches.push_back(g_vMatches[i]);
            count++;

            if (count > 10)
                break;
        }
    }
}

```

```

nth_element(newMatches.begin(), newMatches.begin() + 4, newMatches.end());
newMatches.erase(newMatches.begin() + 5, newMatches.end());

```

```

Point2f srcTri[3];
Point2f dstTri[3];
Point2f A1 = keyPoint1[newMatches[0].queryIdx].pt;
Point2f B1 = keyPoint2[newMatches[0].trainIdx].pt;
Point2f A2 = keyPoint1[newMatches[1].queryIdx].pt;
Point2f B2 = keyPoint2[newMatches[1].trainIdx].pt;
Point2f A3 = keyPoint1[newMatches[2].queryIdx].pt;
Point2f B3 = keyPoint2[newMatches[2].trainIdx].pt;

```

```

srcTri[0] = A1;
dstTri[0] = B1;
srcTri[1] = A2;
dstTri[1] = B2;
srcTri[2] = A3;
dstTri[2] = B3;

```

```

Mat affine(2, 3, CV_64FC1, Scalar(0.0));
affine = getAffineTransform(srcTri, dstTri);
cout << affine << endl;
Mat affine_T(3, 2, CV_64FC1, Scalar(0.0));
affine_T = affine.t();
affine_mat[8] = 1;

```

```

for (int i = 0; i < 3; i++)
{
    for (int j = 0; j < 2; j++)
    {
        affine_mat[i*3+j] = affine_T.at<double>(i,j);
    }
}

```

```

int main()
{
    double **aff = new double*[3]();
    for (int i = 0; i < 3; i++)
    {
        aff[i] = new double[3]();
    }
}

```

```

}
//Mat* img_shake = new Mat();
//Mat* img_ref = new Mat();
//img_shake->create(512, 512, CV_64FC1);
//img_ref->create(512, 512, CV_64FC1);

Mat img_ref = imread("GF4_sub8_2.tiff", CV_LOAD_IMAGE_ANYDEPTH);
Mat img_shake = imread("GF4_sub8_3.tiff",
CV_LOAD_IMAGE_ANYDEPTH);

img_ref.convertTo(img_ref, CV_64FC1);
img_shake.convertTo(img_shake, CV_64FC1);

img_ref = img_ref / 4;
img_shake = img_shake / 4;

img_ref.convertTo(img_ref, CV_8UC1);
img_shake.convertTo(img_shake, CV_8UC1);

clock_t start, end;
start = clock();

getAffineMat(&img_ref, &img_shake, aff);
end = clock();
cout << "time = " << double(end - start) / CLK_TCK << " sec" << endl;

for (int i = 0; i < 3; ++i)
{
    delete[] aff[i];
}
delete[] aff;

return 0;
}
=====
=====

```