目录

视频中的异常状况检测

孙淼

合肥工业大学计算机与信息学院

摘 要：本文综合利用图像、视频分析技术实现现实生活中异常状况的检测/监测。即实时发现并报告异常状况，以及时对异常进行处理。具体内容有：通过提取特征点进行异常的发现和定义（不限定应用场景）；没有利用网上现成的数据集，而是自行收集数据集；输入的是视频，当视频中发生（我们设定的）异常时，进行提示。最后利用上课所介绍的实验评价方法，以及从其他资料中获取的评价方法，对所做的实验进行评价。

关键词：图像处理；异常检测；视频；评价。

1 算法原理

1.1 实际生活中异常状况的发现与定义

对于第一个问题“进行异常的发现和定义（可以限定应用场景，如：高速公路、校园等）”；我们首先在生活中发现异常，我选择的是摄像头场景，我们都知道，在实际项目中（如经典的检测道路车辆并分析车辆行为时，需要按照事先规定的方法绘制检测区包含道路方向、车道区域等），如下图 1-1 所示。由于各种原因(人为、天气)，获取视频数据的摄像头的角度容易偏移原来设定的位置，造成检测区域和实际画面不匹配，系统容易产生误检误报等错误数据。

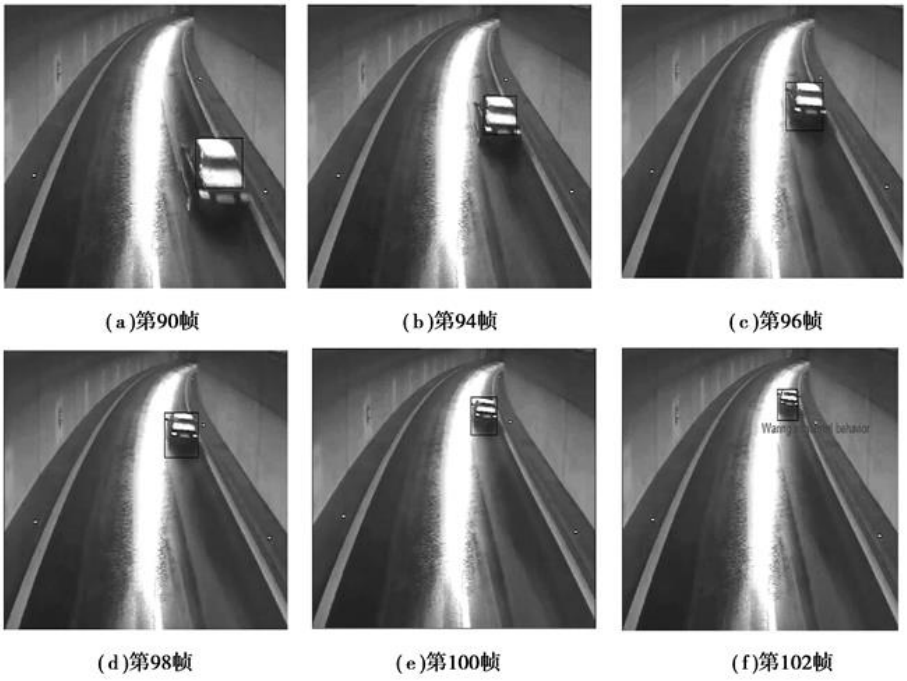


图 1-1 常见的检测道路车辆并分析车辆行为的画面

因此实际生活中应该是需要在摄像机位置偏移第一时间告诉系统检测模块停止工作，直到摄像机归位后再进行检测。摄像机角度偏移告警属于‘视频诊断’中的一类，本文利用提取图片特征点实现摄像机偏移告警。在我设计的摄像头偏移告警功能中，异常定义有以下四类：1）未偏移；2）轻微偏移；3）严重偏移；4）完全偏移。其中 1）与 2）3）4）是通过特征点个数区分的，2）3）4）彼此之间是通过特征点偏移距离区分的。

1.2 数据集的选择

对于第二个问题“可以利用网上现成的数据集，或者自行收集数据集”，由于我没有摄像头数据，就在寝室模拟了一个摄像头的拍摄；刚开始我将手机摄像头稳住，保持一段时间，这段时间即为未偏移；然后非常轻微的偏移手机，并持续一段时间，这段时间即为轻微偏移；接着大幅度晃动手机，但是摄像头内的画面仍旧保持和之前的画面差别不大，这段时间即为严重偏移；最后直接跳转摄像头，拍摄寝室内的其他景物，也就是使得画面与之前完全不同，这段时间即为完全偏移。逐帧画面大致如下图 1-2 所示：

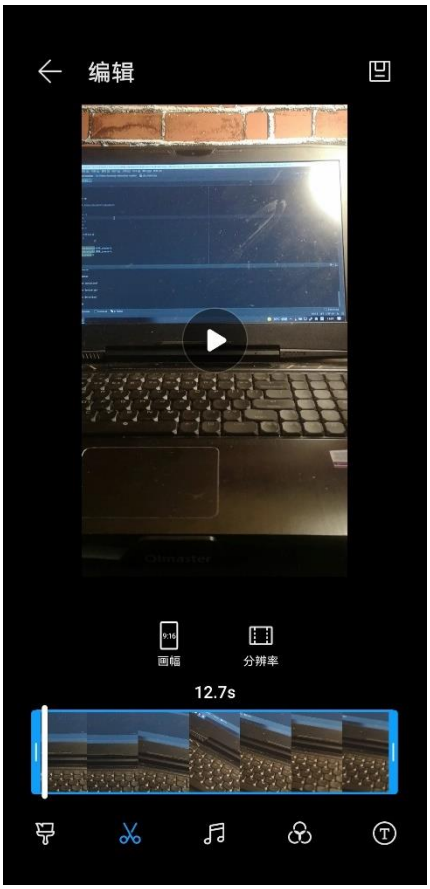


图 1-2 手机中录制的视频的逐帧展示

1.3 输入视频，发现异常

对于第三个问题“输入需要是视频，当视频中发生异常时，进行提示”，对于读取视频这

一问题，我的解决方案是逐帧读取视频中的画面，通过 index % i 中的 i 和 index += 1 来完成选择哪些帧，然后将选择的这些帧的每个前后帧进行特征点分析和比对，得到结果，在第二部分我将详细分析帧的选择问题。

对于特征点检测，我选择的是 SIFT 特征点检测。要学习 SIFT 特征点检测算法，首先要了解什么是尺度空间和图像金字塔。如果一张照片的像素数目被不断压缩，或者观察者距离照片越来越远，它将逐渐变得模糊，而这个导致照片的呈现内容发生变化的连续的自变量就可以称为尺度。对物体观察的尺度不同，物体呈现的方式也不同。对计算机视觉而言，无法预知某种尺度的物体结构是否有意义，因此有必要将所有尺度的结构表示出来。

利用图像金字塔的方法，我们可以得到一系列大小不一的图像，由大到小，从下到上构成的塔状模型。假设高斯金字塔的第 1 层图像为 G1，则有：

$$G_l(i,j)=\sum_{m=-2}^{2}\sum_{n=-2}^{2}\omega(m,n)G_{l-1}(2i+m,2j+n)$$

$$(1\le l\le N,0\le i\le R_l,0\le j\le C_l)$$

式中，N 为高斯金字塔层数；R1 和 G1 分别为高斯金字塔第 1 层的行数和列数；ω(m,n)是一个二维可分离的 5× 5 窗口函数，表达式为：

$$\omega=\frac{1}{256}\begin{pmatrix}1&4&6&4&1\\4&16&24&16&4\\6&24&36&24&6\\4&16&24&16&4\\1&4&6&4&1\end{pmatrix}=\frac{1}{16}\begin{pmatrix}1&4&6&4&1\end{pmatrix}\times\frac{1}{16}\begin{pmatrix}1\\4\\6\\4\\1\end{pmatrix}$$

写成上面的形式是为了说明，二维窗口的卷积算子，可以写成两个方向上的 1 维卷积核（二项核）的乘积。上面卷积形式的公式实际上完成了 2 个步骤：1）高斯模糊；2）降维。按上述步骤生成的 G0,G1,...,GN 就构成了图像的高斯金字塔，其中$G0$为金字塔的底层(与原图像相同)，GN为金字塔的顶层。可见高斯金字塔的当前层图像是对其前一层图像先进行高斯低通滤波，然后做隔行和隔列的降采样(去除偶数行与偶数列)而生成的。将 G1 进行内插（这里内插用的不是双线性而是用的与降维时相同的滤波核）得到放大图像，使他们的尺寸相同，表示为：

$$G_l^*(i,j)=4\sum_{m=-2}^{2}\sum_{n=-2}^{2}\omega(m,n)G_l(\frac{i+m}{2},\frac{j+n}{2})$$

其中上面的系数 4，是因为每次能参与加权的项，权值和为 4/256，这个与我们用的 ω 的值有关。式中，

$$G_l(\frac{i+m}{2},\frac{j+n}{2})=\begin{cases}G_l(\frac{i+m}{2},\frac{j+n}{2}), & \frac{i+m}{2},\frac{j+n}{2}\in\mathbb{Z}\\ 0, & \text{others}\end{cases}$$

令

$$\begin{cases}LP_l=G_l-G_{l+1}^*, & 0\le l\le N\\ LP_N=G_N, & l=N\end{cases}$$

式中，N 为拉普拉斯金字塔顶层号，LP1 是拉普拉斯金字塔分解的第1层图像。由 LP0,LP1,\cdots,LP1,...,LPN 构成的金字塔即为拉普拉斯金字塔。它的每一层图像是高斯金字塔本层图像与其高一级的图像经内插放大后图像的差，此过程相当于带通滤波，因此拉普拉斯金字塔又称为带通金字塔分解。

图像的金字塔化能高效地（计算效率也较高）对图像进行多尺度的表达，但它缺乏坚实的理论基础，不能分析图像中物体的各种尺度。

我们知道，信号的尺度空间刚提出是就是通过一系列单参数、宽度递增的高斯滤波器将原始信号滤波得到到组低频信号。那么一个很明显的疑问是：除了高斯滤波之外，其他带有参数 t 的低通滤波器是否也可以用来生成一个尺度空间。后来 Koenerink、Lindeberg、Florack 等人用精确的数学形式通过不同的途径都证明了高斯核是实现尺度变换的唯一变换核。

虽然很多研究者从可分性、旋转不变性、因果性等特性推出高斯滤波器是建立线性尺度空间的最优滤波器。然后在数字图像处理中，需要对核函数进行采样，离散的高斯函数并不满足连续高斯函数的的一些优良的性质。所以后来出现了一些非线性的滤波器组来建立尺度空间，如 B 样条核函数。使用高斯滤波器对图像进行尺度空间金塔塔图的构建，让这个尺度空间具有加权平均和有限孔径效应和局部极值递性的性质。

下面开始介绍 SIFT 算法：令原图像为金字塔的第一层，每次降采样所得到的新图像为金字塔的一层(每层一张图像)，每个金字塔共 n 层。金字塔的层数根据图像的原始大小和塔顶图像的大小共同决定，其计算公式如下：

$$n=\log_2[\min(M,N)]-t$$

其中 M，N 为原图像的长和宽，t 为塔顶图像的最小维数的对数值。接下来，我们通过尺度空间理论模拟图像数据的多尺度特征,以高斯函数为实现尺度变换的唯一线性核

，则二维图像 I(x, y)的尺度空间。

$$L(x,y,\sigma) = I(x,y) * g(x,y,\sigma)$$

$$\text{where } g(x,y,\sigma) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

根据高斯函数的性质，我们知道高斯窗的宽度约为 6σ，即在[-3σ，3σ]外的区间外，信号通过高斯滤波器的响应为 0，因此，高斯函数不同的标准差就可以作为不同尺度的量度，事实上对高斯函数进行傅里叶变换后有

$$\mathscr{F}[g(x,k\sigma)] = k\mathscr{F}[g(x,\sigma)] \quad \mathscr{F}[g(x,y,k\sigma)] = k^2\mathscr{F}[g(x,y,\sigma)]$$

即一维高斯核是线性核，二维高斯核使之仍是线性的。这样就可以定义高斯差算子为：

$$D(x,y,\sigma,k) = L(x,y,k\sigma) - L(x,y,\sigma) \approx (k-1)\nabla^2 h$$

图像与某一个二维函数进行卷积运算实际就是求取图像与这一函数的相似性。同理，图像与高斯拉普拉斯函数的卷积实际就是求取图像与高斯拉普拉斯函数的相似性。当图像中的斑点尺寸与高斯拉普拉斯函数的形状趋近一致时，图像的拉普拉斯响应达到最大。从概率的角度解释为：假设原图像是一个与位置有关的随机变量 X 的密度函数，而 LOG 为随机变量 Y 的密度函数，则随机变量 X+Y 的密度分布函数即为两个函数的卷积形式。如果想让 X+Y 能取到最大值，则 X 与 Y 能保持步调一致最好，即 X 上升时，Y 也上升，X 最大时，Y 也最大。实际上每一次采样返回的结果，就像音乐上的一个八度，为了让尺度体现其连续性，高斯金字塔在简单降采样的基础上加上了高斯滤波。将图像金字塔每层的一张图像使用不同参数做高斯模糊，使得金字塔的每层含有多张高斯模糊图像，将金字塔每层多张图像合称为八度，金字塔每层只有一组图像，组数和金字塔层数相等，每组含有多张图像。另外，降采样时，高斯金字塔上一组图像的初始图像(底层图像)是由前一组图像的倒数第三张图像隔点采样得到的。在下一个八度，高斯算子的标准差扩大一倍。在高斯差分尺度空间检测局部极大或极小值，检测点与其同尺度的 8 个相邻点、上下相邻尺度对应的 9 × 2 个点进行比较，以确保在尺度空间和二维图像空间都检测到极值点，极值点的位置可以通过对高斯差分算子求一阶导数得到。离散空间的极值点并不是真正的极值点，利用已知的离散空间点插值得到的连续空间极值点的方法叫做亚像素插值。为了提高关键点的稳定性，需要对尺度空间 DoG 函数进行曲线拟合。利用 DoG 函数在尺度空间的 Taylor 展开式(拟合函数)为：

$$D(X) = D + \frac{\partial D^T}{\partial X}X + \frac{1}{2}X^T\frac{\partial^2 D}{\partial X^2}X$$

$$\text{where } X = [x, y, \sigma, k]^T$$

求导并让方程等于零，可以得到极值点的偏移量，对于这样得到的极值点，首先要舍去高斯差分算子绝对值很小（一般指|D(\hat X)|<0.03）的点，通过主曲率分析去除边缘响应过大的极值点，即计算差分图像 D 的 Hessian 矩阵：进而确定特征的方向参数：梯度方向直方图的范围：[0, 360]；特征点的主方向：直方图的峰值 ；特征点的辅方向：直方图的次峰值，能量超过主峰值能量的 80\%；当图像发生旋转时，特征点的主方向相对于像素的梯度方向不变；将多幅待匹配的图像都旋转到令特征点方向为 0 的位置再匹配，使特征具有旋转不变性。现在我们将坐标轴方向旋转为特征点的方向，以特征点为中心取窗口，通过高斯加权增强特征点附近像素梯度方向信息的贡献，即在 4 × 4 的小块上计算梯度方向直方图（ 取 8 个方向），计算梯度方向累加值，形成种子点，构成 4× 4 × 8= 128 维特征向量。SIFT 特征对旋转、尺度缩放、亮度变化保持不变性，对视角变化、仿射变换、噪声也保持一定程度的稳定性。

SIFT 特征检测的步骤：检测尺度空间的极值点；精确定位特征点；设定特征点的方向参数；生成特征点的描述子（128 维向量）。

SIFT 特征检测算法的特点：SIFT 特征是图像的局部特征，对旋转、尺度缩放、亮度变化保持不变性，对视角变化、仿射变换、噪声也保持一定程度的稳定性；信息量丰富，适用于在海量特征数据库中进行匹配；多量性，少数物体也可以产生大量 SIFT 特征；高速性，经优化的 SIFT 匹配算法甚至可以达到实时性。

1.4 实验评价

对于第三个问题“利用上课所介绍的实验评价方法，或你从其他资料中获取的评价方法，对你所做的实验进行评价。”，目前常用的对匹配结果进行量化评价一般包括两个部分：检测的评价和匹配的评价。

1） 特征点检测评价

评价特征点检测方法的优劣常常用到 repeatability（重复率）作为特征点检测器和匹配结果的评价准则，我的理解是：图 A、B 是两幅待匹配图像，图 A 映射到图 B 有一个单应性矩阵 H1，图 B 映射到图 A 有单应性矩阵 H2，图 A 检测出 N1 个特征点，图 B 检测出 N2 个特征点，因为图像 A 和 B 有部分图像不重叠，故将 A 图检测的特征点坐标由 H1

算出在 B 图的坐标，去掉不合格（计算结果超出在 B 图像坐标）的特征点，剩下的特征点数记为 n1；同样，B 图的特征点经过处理剩下 n2 个；分母便是 min(n1,n2)。将图 A 剩下的特征点由 H1 计算出在图 B 中的坐标，与图 B 检测出的特征点的坐标求距离，即 dist(h1*a1,b1)，若距离小于阈值 ε，则认为是重复的，这么做是因为得到的单应性矩阵不一定完全精确以及一些别的误差原因。

repeatability 计算式的分母可以和原来定义相同，分子的定义 correspondence 满足两个条件：1）同之前定义，即设定 ε=1.5；2）regions 映射到另一幅图像中重叠误差小于 0.2。这里第二点是因为特征点检测器得到的特征点实际上是根据特征点附近区域计算梯度来选取特征点，实际上检测的特征区域，比如 Harris-Affine 检测的是一个椭圆区域，SIFT 检测的是一个圆形区域；当左图一特征点区域在经过乘上单应性矩阵 H 后实际，该区域会发生变化，所以需要判断变化之后的区域与右图某一特征点区域的重叠误差，误差<0.2，则判断自是一对特征点。

2）特征点匹配评价

特征点匹配的评价一般涉及到两个概念，即 Recall（召回率）和 Precision（精确率），我们先来了解一下这两个概念。精确率和召回率是广泛用于信息检索和统计学分类领域的两个度量值，用来评价结果的质量。精确率是检索出相关文档数与检索出的文档总数的比率，衡量的是检索系统的查准率；召回率是指检索出的相关文档数和文档库中所有的相关文档数的比率，衡量的是检索系统的查全率。在信息检索中：精确率 = 提取出的正确信息条数 / 提取出的信息条数；召回率 = 提取出的正确信息条数 / 样本中的信息条数。Precision 和 Recall 指标有时候会出现的矛盾的情况，可以绘制 Precision-Recall 曲线，曲线越靠上部，结果越好。

2 算法设计

2.1 实际生活中异常状况的发现与定义的算法

对于生活在异常状况的发现与定义，我将借助 Python 实现，根据第一部分的介绍，我们设计的异常状况有四种：1）未偏移；2）轻微偏移；3）严重偏移；4）完全偏移；对于基于特征点检测算法的代码，我们首先肯定要设计对应的阈值，前面我们也提到，1）与 2）3）4）是通过特征点个数区分的，2）3）4）彼此之间是通过特征点偏移距离区分的。所以我们首先要确定阈值，在库 do_match 中代码如下：

```
1.  # 至少 10 个点匹配
```

```
MIN_MATCH_COUNT = 10
# 完全匹配偏移 d<1
BEST_DISTANCE = 1
# 微量偏移  1<d<10
GOOD_DISTANCE = 10
```

在主函数 test 中相关代码如下：

```
texts = ["Occlusion", "Severe deviation!", "deviation", "No offset"]
#texts = ["视频遮挡或严重偏移"], ["严重偏移"], ["轻微偏移"], ["无偏移"]
```

2.2 数据集的选择和输入视频，发现异常的算法

输入视频才能截取有效帧进行处理，我们使用 python 中 cv2 库自带的读取视频帧的方法即可，如果没有读取到，就返回“error opening video stream or file”，代码如下：

```
cap = cv2.VideoCapture('D:\\video.mp4')

if (cap.isOpened() == False):
    print("Error opening video stream or file")
```

读取到视频之后，就可以开始处理帧，通过一个 while 循环来一直读取到帧的 RGB 形式矩阵，对于特殊的第一帧，就通过一个标志位 first_frame 单独多一次传递，然后置 first_frame 为 false，开始下一轮循环，代码如下：

```
first_frame = True
pre_frame = 0

index = 0

while (cap.isOpened()):
    '''''
    ret, frame = cap.read()返回值含义：
    参数 ret 为 True 或者 False,代表有没有读取到图片
    第二个参数 frame 表示截取到一帧的图片
    是该帧图像的三维矩阵 BGR 形式。
    '''
    ret, frame = cap.read()

    #cv2.imshow("Initial",frame)
    if ret == True:#读取到图片
        if first_frame:#如果是第一帧
            pre_frame = frame
```

```
            first_frame = False
            continue
```

接下来就是设计最精妙的部分，通过 index 递增，如果 index % i == 0 就进入代码块，也就是说 i 是选择帧的间隔，比如 i＝2，那就是选择每个偶数帧，进入代码块之后，通过 datetime.utcnow().microsecond 来得到国际时区的时间 time1，然后进行核心方法 do_match 的调用，调用完之后将其返回值赋给 result，然后再次运行 datetime.utcnow().microsecond 得到 time2，通过 time2－time1 可以得知当前这次 do_match 的运行时间。并通过 result 得到一个对应的标签，通过 putText 方法写到视频上去：

```
index += 1
if index % 2 == 0:#间隔
    time1 = datetime.utcnow().microsecond
    result = do_match.match2frames(pre_frame, frame)#将得到 result
    time2 = datetime.utcnow().microsecond
    print(str(time2 - time1))#显示这两帧处理需要的时间
    #result = 1;
    print("检测结果===>", texts[result])

    if result == 0:#前后两张图片不满足匹配条件
        pre_frame = frame
    size = frame.shape

    if size[1] > 720:  # 缩小显示
        frame = cv2.resize(frame, (int(size[1] * 0.5), int(size[0] * 0.5)), cv2.INTER_LINEAR)

    text_frame = cv2.putText(frame, texts[result], (123,456),cv2.FONT_HERSHEY_PLAIN, 2, (0,255,0), 3)#

    cv2.imshow('result', text_frame)
if cv2.waitKey(1) & 0xFF == ord('q'):#1 为参数，单位毫秒 ms，表示读帧间隔时间 按 q 键退出
    break
:
break
cap.release()
cv2.destroyAllWindows()
```

接下来我们介绍 do_match 库，首先把 cv2 库内置的几种特征点提取方法都得到：

```
# 特征点提取方法，内置很多种
```

```
algorithms_all = {

    "SIFT": cv2.xfeatures2d.SIFT_create(),
    "SURF": cv2.xfeatures2d.SURF_create(),
    "ORB": cv2.ORB_create()

}
```

然后就是这个库最主要的方法 match2frame，具体思路就先把 test 中得到的前后帧对应的两张图像变成灰度图，然后对其大小略作调整，将灰度图输入 detectAndCompute，使用 Hessian 算法检测关键点，并且对每个关键点周围的区域计算特征向量。返回关键点的信息和描述符。然后声明一个 BFMatcher 蛮力匹配器 bf，调用其方法 knnMatch 返回参数 k 个最佳匹配，对 k 个最佳匹配进行变换，得到在限定阈值内的若干个 good，通过 RANSAC 算法得到变换矩阵，revel 方法进行图像变换，最后进行排序，如此就是 de_match 的前半部分，得到一个排序好的 good[]，代码如下：

```
def match2frames(image1, image2):
    img1 = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
    img2 = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)

    size1 = img1.shape
    size2 = img2.shape

    img1 = cv2.resize(img1, (int(size1[1] * 0.3), int(size1[0] * 0.3)), cv2.INTER_LIN
EAR)#
    img2 = cv2.resize(img2, (int(size2[1] * 0.3), int(size2[0] * 0.3)), cv2.INTER_LIN
EAR)#

    surf = algorithms_all["SIFT"]

    #使用 Hessian 算法检测关键点，并且对每个关键点周围的区域计算特征向量。
    #返回关键点的 信息 和 描述符
    kp1, des1 = surf.detectAndCompute(img1, None)#输入灰度图
    kp2, des2 = surf.detectAndCompute(img2, None)#输入灰度图

    '''''
    蛮力匹配器
    可以采用第一组的一个特征的描述符并且使用一些距离计算与第二组中的所有其它特征匹配。
    将最近的一个返回。
    '''
    bf = cv2.BFMatcher()
```

```
    '''''
    knnMatch 为每个关键点返回 k 个最佳匹配，其中 k 是由用户设定的。
    我们使用函数 cv2.drawMatchsKnn 为每个关键点和它的 k 个最佳匹配点绘制匹配线。
    如果要选择性绘制就要给函数传入一个掩模。
    '''
    matches = bf.knnMatch(des1, des2, k=2)
    # 过滤
    good = []
    for m, n in matches:
        if m.distance < 0.69 * n.distance:
            good.append(m)
    src_pts = np.float32([kp1[m.queryIdx].pt for m in good]).reshape(-1, 1, 2)#特征点
    dst_pts = np.float32([kp2[m.trainIdx].pt for m in good]).reshape(-1, 1, 2)#特征点

    # 第四个参数取值范围在 1 到 10 , 绝一个点对的阈值。
    # 原图像的点经过变换后点与目标图像上对应点的误差
    # 超过误差就认为是异常值
    # 返回值中 M 为变换矩阵。mask 是掩模
    #单应性矩阵是一个 3 * 3 的矩阵，可以从 cv2.findHomography 获得
    # H, status = cv2.findHomography(ptsA, ptsB, cv2.RANSAC, 5)
    # src_pts, dst_pts 对应两幅图获得的 sift 特征点
    # 获取变换矩阵，RANSAC 算法
    M, mask = cv2.findHomography(src_pts, dst_pts,cv2.RANSAC,5.0)
    # 图像变换，将原图像变换为检测图像中匹配到的形状
    matchesMask = mask.ravel().tolist()
    good = [good[matchesMask.index(i)] for i in matchesMask if i == 1]
    #img3 = cv2.drawMatches(img1, kp1, img2, kp2, good, None)
    good = sorted(good,key=lambda x:x.distance)#根据 good 字段中的 distance 进行排序
```

最后通过 good[]中的值，判断当前前后帧属于哪个范围（三个阈值划定了四个范围）。

```
    if len(good) < MIN_MATCH_COUNT:
        return 0  # 完全不匹配
    # 如果 good > 10
    # 则开始求 distance
    else:
        good = good[0:MIN_MATCH_COUNT]
        distance_sum = 0  # 特征点 2d 物理坐标偏移总和
        for m in good:
            distance_sum += get_distance(kp1[m.queryIdx].pt, kp2[m.trainIdx].pt)
        distance = distance_sum / len(good)  # 单个特征点 2D 物理位置平均偏移量

```

```
        if distance < BEST_DISTANCE:# distance < 1
            return 3  # 完全匹配
        elif distance < GOOD_DISTANCE and distance >= BEST_DISTANCE: # distance >= 1 && distance < 10
            return 2  # 部分偏移
        else:
            return 1  # 仅仅场景匹配 # distance >= 10

'''''
计算 2D 物理距离
'''


def get_distance(p1, p2):
    x1, y1 = p1
    x2, y2 = p2
    return np.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)
```

2.3 实验评价的算法

相对于一般的分类问题如 sklearn.datasets 的 load_iris 数据集，是针对蝴蝶类型进行分类，对每一种类别同等关心，这里的“异常检测”主要指在多分类中，更关心或只关心其中的“某一类”（异常类）。异常检测中最常见的是二分类异常检测，即只分“正常”和“异常”两类（一般在标签中“正常”标记为“0”，“异常”标记为“1”）。例如：在一系列信号中找异常信号、在一系列行为特征中找异常行为。

对于一般分类问题，因为对每一种类别都很关心，所以在评价时一般直接计算其正确率：

正确率 = 所有匹配的样本数目 / 所有的样本数目

对于异常检测问题，评价指标更为复杂。主要原因一是一般异常检测的数据集都是不平衡数据集，即正常数据多异常数据少，如果只看正确率的话，可以想象一种场景：如果测试集中 70%都是“正常”，当预测结果全是“正常”时，其正确率也能达到 70%，但事实上这对于更关心异常的我们来说，是并没有什么用。可能大家会说，那你把数据弄平衡不就好了？但是本身异常数据就很少（例如内部威胁检测），处理不平衡数据集，无非就是抽样啊减少正常样本啊之类的，那总数据还是太少（例如异常样本 70+正常的 1000+），或者干脆比如用生成对抗网络 GAN，自己生成新的异常样本、或者利用特征相似性自己造新的异常数据，但是这造出来的数据对不对又是个问题了。

所以异常检测只看正确率是万万不行的，那主要有哪些指标呢？这里有四个关键字母：T F P N

True 真　　False 假　　Positive 正　　Negative 负

在异常检测中，P 和 N 一般定义里是针对预测来说的而 Positive 正类指的是你更关心的那一类，即“异常”（也就是标签中的“1”），顾名思义 Negative 负类就是指正常了。总结来说就是：P 指预测为正类，即预测为异常。

T 和 F 是针对预测与实际的比较结果，True 真是指正确匹配！具体来说就是：你预测它是正它实际也是正，你预测它是负他实际也是负，False 假就是不匹配的。总结就是下表：

T/F 代表预测与实际之间是否“匹配”，P/N　代表预测的为“正/负”

	实际正例	实际负例	
预测正例 P	预正实正 TP	预正实负 FP	所有预测为正个数 TP+FP
预测负例 N	预负实正 FN	预负实负 TN	所有预测为负个数 FN+TN
	所有实际正例个数 TP+FN	所有实际负例个数 FP+TN	

知道 TP、FP、FN、TN 的概念了，就可以计算各种指标了，常用指标如下：

TPR：真正类率，又叫真阳率，代表预测是异常实际也是异常的样本数，占实际总异常数的比例——值越大、性能越好；

FPR：假正类率，又叫假阳率，代表预测是异常但实际是正常的样本数，占实际正常总数的比例——值越小、性能越好；

R：召回率，意义同 TPR——值越大、性能越好；

P：精确率，代表预测是异常实际也是异常的样本数，占预测是异常的总数的比例——值越大、性能越好；

F：P 和 R 的加权调和平均，常用的是 F1 值——值越大、性能越好；

A：正确率，与精确率的区别是，不仅考虑异常类也考虑正常类，即所有匹配样本数，占所有样本的比例——值越大、性能越好；

距离说明的话，假设现有的测试集样本的标签为 3 个异常 7 个正常，以标签“1”代表异

常，全部标签可以表示为：[1, 1, 1, 0, 0, 0, 0, 0, 0, 0]；

假设预测的标签为[1, 0, 1, 0, 0, 0, 1, 0, 0, 1]

实：[1, 1, 1, 0, 0, 0, 0, 0, 0, 0]

预：[1, 0, 1, 0, 0, 0, 1, 0, 0, 1]，则画表如下：

	实际正例	实际负例
预测正例 P	TP = 2	FP = 2
预测负例 N	FN = 1	TN = 5

则，

TPR = TP/(TP+FN) = 2/(2+1) = 2/3

FPR = FP/(FP+TN) = 2/(2+5) = 2/7

R = TPR = 2/(2+1) = 2/3

P = TP/(TP+FP) = 2/(2+2) = 1/2

A = (TP+TN)/总样本数 = 7/10

PS：

1.在异常检测中，有的更注重召回率甚至需要保证是 100%，有的更注重假阳率或者说 FP 值，即要求误判为异常实际正常的数目不能太多。具体看什么问题。

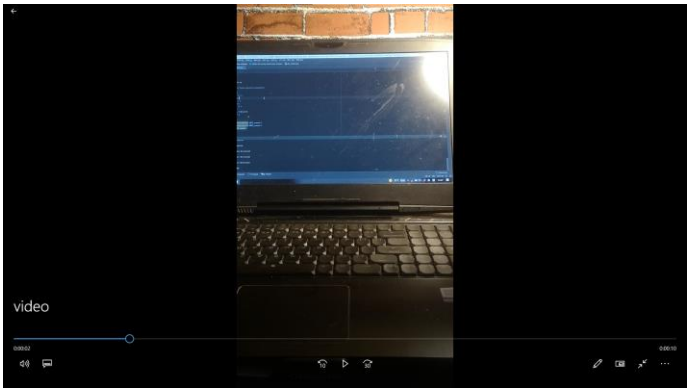
2. 在一次预测中，如果预测结果是异常分数值，那么一般会选定某分数作为阈值，然后高于此阈值的分数为异常，低于此阈值的分数判定为正常。此时，根据阈值的取值不同，我们可以得到不同的 FPR 和 TPR 值，将所有的值以 FPR 为横轴，TPR 为纵轴画图，即为 ROC 曲线图；曲线的向下覆盖的面积即为 AUC 值。

3 实验结果

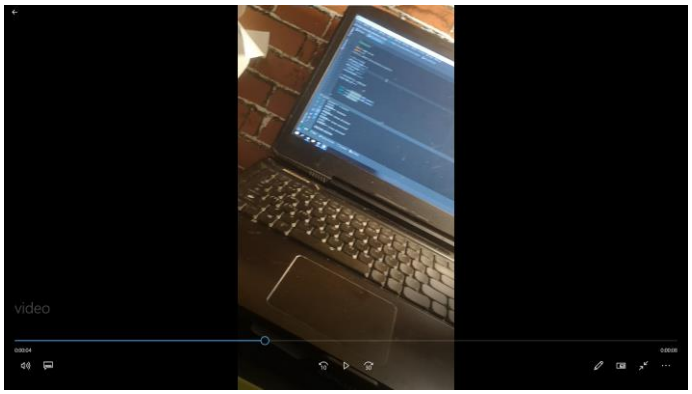
3.1 实验数据

实验数据是指在实验中控制实验对象而搜集到的变量的数据。搜集数据的另一类方法是通过实验，在实验中控制一个或多个变量，在有控制的条件下得到观测结果。

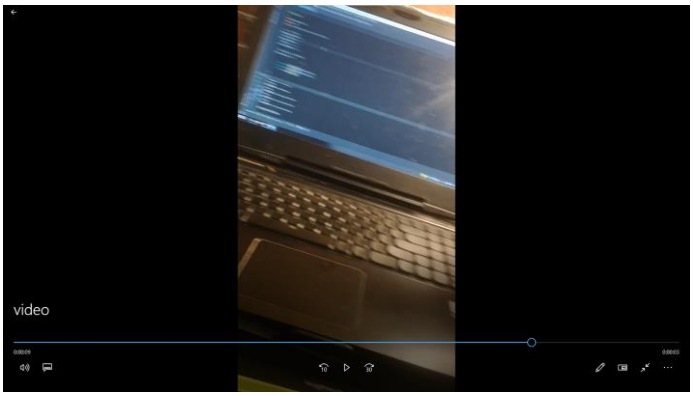
实验数据取自在寝室用手机拍摄的一段 12 秒的短视频，拍摄的对象是我的电脑，在 0~3 秒内，摄像头没有晃动，目的是得到“未偏移”的检测结果；在 3~8 秒内，摄像头轻微晃动，目的是得到“轻微偏移”的检测结果；在 8~10 秒摄像头强烈晃动，目的是得到“严重偏移”的检测结果；在 10~12 秒，镜头转向寝室阳台一侧，也就是为了得到“完全偏移”的检测结果，如下四图所示。



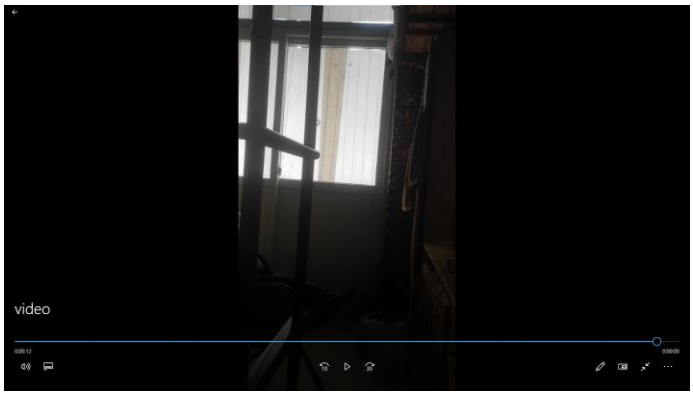
（a）摄像头没有晃动（0~3 秒）



（b）摄像头轻微晃动（3~8 秒内）



（c）摄像头剧烈晃动（8~10 秒）



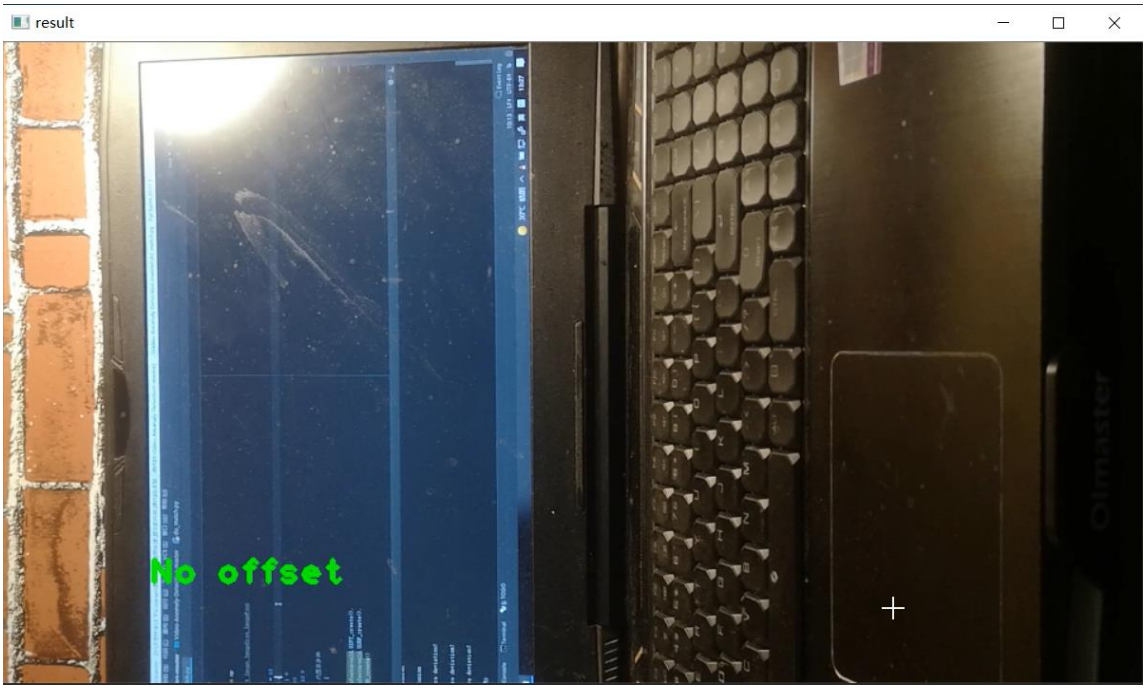
（d）摄像头转向阳台（10~12 秒）

图 3-1 实验数据——12 秒的短视频

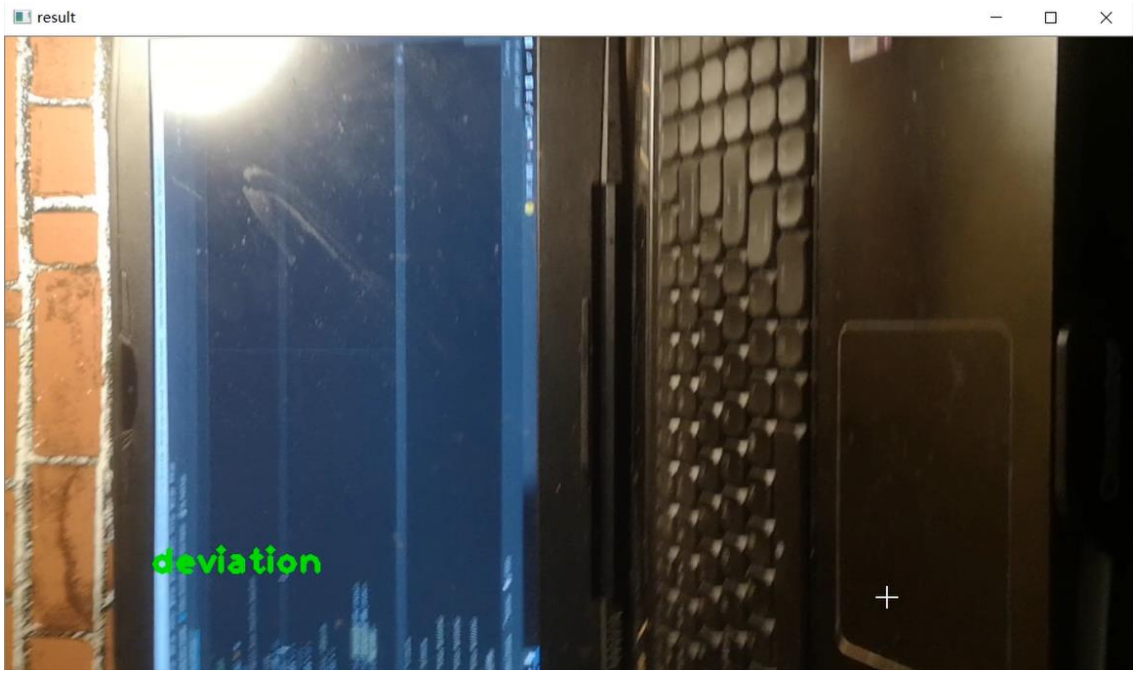
相关视频也在附录内，命名为 video.mp4。

3.2 实验结果

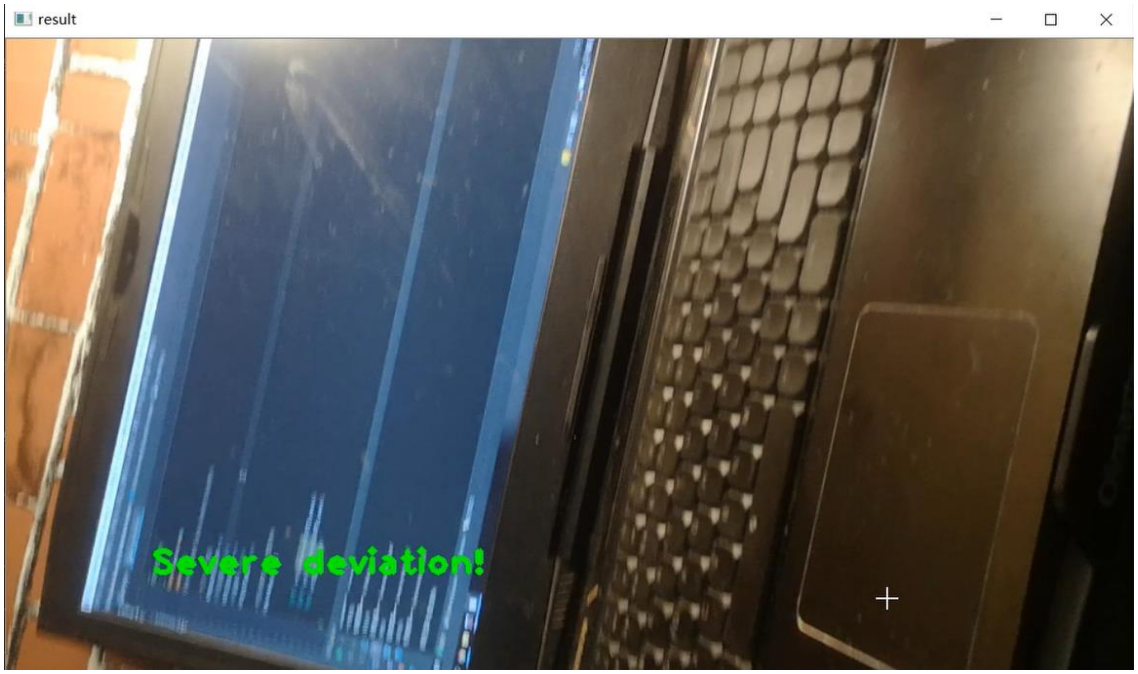
在 0~3 秒内，摄像头没有晃动，得到了“No offset（未偏移）”的检测结果并保持；在 3~8 秒内，摄像头轻微晃动，得到了“deviation（轻微偏移）”的检测结果；在 8~10 秒摄像头强烈晃动，得到了“Severe deviation（严重偏移）”的检测结果；在 10~12 秒，镜头转向寝室阳台一侧，得到了“Occlusion（完全偏移）”的检测结果，如下四图所示



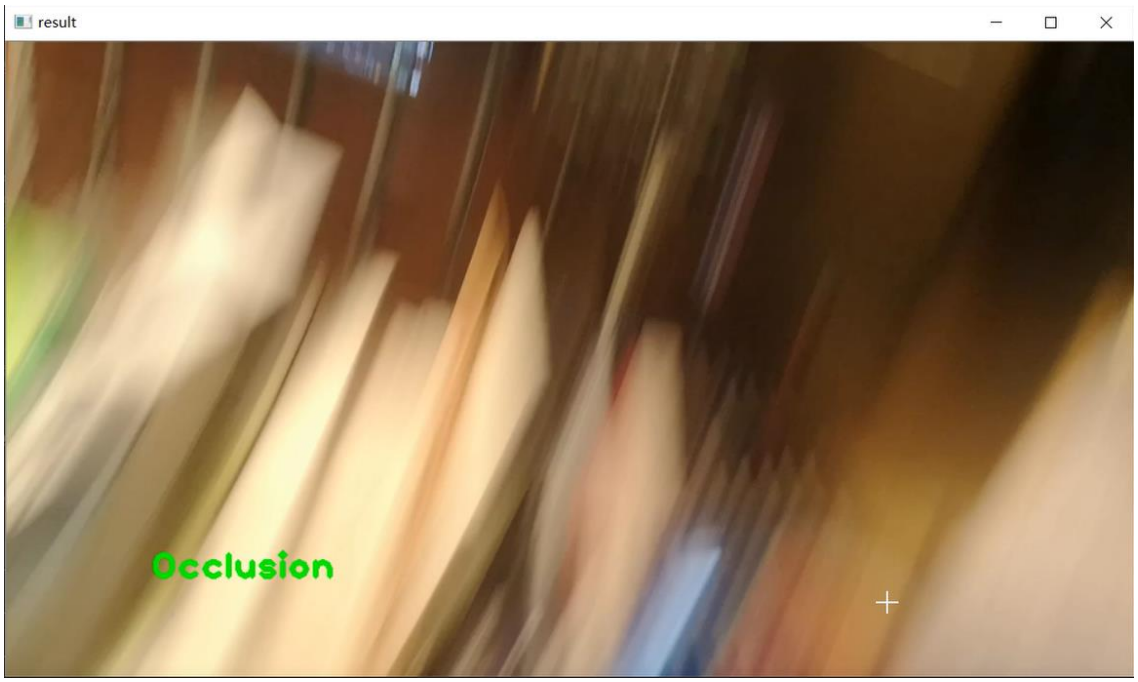
（a）No offset（无偏移）



（b）deviation（轻微偏移）



（c）Severe deviation（严重偏移）



（d）Occlusion（完全偏移）

图 3-2 摄像头视频实时检测结果

相关视频在附录内，命名为 result.mp4.

4 结论与思考

4.1 实验评价的结论

由于我们在代码中设计了前后帧的检测结果输出，所以可以通过输出的结果进行统计，结合录制的视频result.mp4慢回放,然后根据第二部分介绍的评价算法进行分析,python

输出的结果如下：

```
E:\Anaconda32020.02\python.exe "D:/大学作业/3 下/Courses/图像与视频信息处理/提交的内容/源代码/实验三源代码/Video-Anomaly-Detection/test.py"
104710
检测结果===> deviation
105849
检测结果===> No offset
109894
检测结果===> No offset
102764
检测结果===> No offset
113308
检测结果===> No offset
108727
检测结果===> No offset
146557
检测结果===> deviation
106628
检测结果===> deviation
119307
检测结果===> deviation
105666
检测结果===> deviation
-894865
检测结果===> deviation
97751
检测结果===> deviation
107665
检测结果===> deviation
100452
检测结果===> deviation
102961
检测结果===> deviation
139823
检测结果===> deviation
124410
检测结果===> deviation
110897
检测结果===> deviation
110682
检测结果===> deviation
98604
检测结果===> deviation
```

```
102954
检测结果===> deviation
106542
检测结果===> deviation
100279
检测结果===> deviation
107379
检测结果===> deviation
104089
检测结果===> deviation
100713
检测结果===> deviation
107972
检测结果===> deviation
95522
检测结果===> deviation
110445
检测结果===> deviation
100733
检测结果===> deviation
102930
检测结果===> deviation
106411
检测结果===> deviation
107127
检测结果===> deviation
101067
检测结果===> deviation
106775
检测结果===> deviation
96519
检测结果===> deviation
111333
检测结果===> deviation
96459
检测结果===> deviation
105078
检测结果===> deviation
108708
检测结果===> deviation
105362
检测结果===> deviation
100607
检测结果===> deviation
```

```
102264
检测结果===> deviation
102462
检测结果===> deviation
107009
检测结果===> deviation
102835
检测结果===> deviation
105618
检测结果===> deviation
121318
检测结果===> deviation
107544
检测结果===> deviation
107886
检测结果===> deviation
107428
检测结果===> Severe deviation!
103777
检测结果===> Severe deviation!
109380
检测结果===> Severe deviation!
101438
检测结果===> Severe deviation!
115931
检测结果===> Severe deviation!
96420
检测结果===> Severe deviation!
-894282
检测结果===> Severe deviation!
103791
检测结果===> Severe deviation!
106713
检测结果===> Severe deviation!
100956
检测结果===> Severe deviation!
111762
检测结果===> Severe deviation!
107670
检测结果===> Severe deviation!
114532
检测结果===> Severe deviation!
112200
检测结果===> Severe deviation!
```

```
-888682
检测结果===> Severe deviation!
109983
检测结果===> Severe deviation!
112439
检测结果===> Severe deviation!
116891
检测结果===> Severe deviation!
115892
检测结果===> Occlusion
113389
检测结果===> Severe deviation!
107169
检测结果===> Severe deviation!
107990
检测结果===> Severe deviation!
109495
检测结果===> Severe deviation!
103835
检测结果===> Severe deviation!
111468
检测结果===> Severe deviation!
96130
检测结果===> Severe deviation!
122500
检测结果===> Severe deviation!
127304
检测结果===> Severe deviation!
121208
检测结果===> Severe deviation!
-888741
检测结果===> Severe deviation!
104459
检测结果===> Severe deviation!
105453
检测结果===> Severe deviation!
114800
检测结果===> Severe deviation!
107656
检测结果===> Severe deviation!
115623
检测结果===> Severe deviation!
106724
检测结果===> Severe deviation!
```

```
107060
检测结果===> Severe deviation!
111567
检测结果===> Severe deviation!
107351
检测结果===> Severe deviation!
102316
检测结果===> Severe deviation!
119724
检测结果===> Severe deviation!
102674
检测结果===> Severe deviation!
107926
检测结果===> Severe deviation!
109890
检测结果===> Severe deviation!
-890991
检测结果===> Severe deviation!
104900
检测结果===> Severe deviation!
170816
检测结果===> Severe deviation!
110056
检测结果===> Severe deviation!
102456
检测结果===> Severe deviation!
105293
检测结果===> deviation
108995
检测结果===> deviation
110291
检测结果===> deviation
114307
检测结果===> deviation
98050
检测结果===> Severe deviation!
106060
检测结果===> Severe deviation!
93124
检测结果===> Severe deviation!
103809
检测结果===> Severe deviation!
106403
检测结果===> Severe deviation!
```

```
114987
检测结果===> Severe deviation!
-896753
检测结果===> Severe deviation!
103933
检测结果===> Severe deviation!
102278
检测结果===> Severe deviation!
108374
检测结果===> Severe deviation!
105400
检测结果===> Severe deviation!
104025
检测结果===> Severe deviation!
103879
检测结果===> Severe deviation!
112782
检测结果===> Severe deviation!
-898205
检测结果===> Severe deviation!
106374
检测结果===> Severe deviation!
102528
检测结果===> Severe deviation!
101615
检测结果===> Severe deviation!
98953
检测结果===> Severe deviation!
98286
检测结果===> Severe deviation!
104624
检测结果===> Severe deviation!
106787
检测结果===> Severe deviation!
102198
检测结果===> Severe deviation!
113725
检测结果===> Severe deviation!
100397
检测结果===> Severe deviation!
111863
检测结果===> Severe deviation!
98450
检测结果===> Severe deviation!
```

```
103810
检测结果===> Severe deviation!
103054
检测结果===> Severe deviation!
104647
检测结果===> Severe deviation!
-894885
检测结果===> Severe deviation!
128346
检测结果===> Severe deviation!
104771
检测结果===> Severe deviation!
109256
检测结果===> Severe deviation!
99928
检测结果===> Severe deviation!
106415
检测结果===> Severe deviation!
109936
检测结果===> Severe deviation!
107458
检测结果===> Severe deviation!
-902272
检测结果===> Severe deviation!
101729
检测结果===> Severe deviation!
108773
检测结果===> Severe deviation!
108964
检测结果===> Occlusion
98724
检测结果===> Severe deviation!
100350
检测结果===> Severe deviation!
110966
检测结果===> Severe deviation!
107186
检测结果===> Severe deviation!
-896744
检测结果===> Occlusion
108999
检测结果===> Severe deviation!
94964
检测结果===> Severe deviation!
```

```
99385
检测结果===> Severe deviation!
106959
检测结果===> Severe deviation!
96273
检测结果===> Severe deviation!
98858
检测结果===> Severe deviation!
110619
检测结果===> Severe deviation!
-896956
检测结果===> Severe deviation!
102204
检测结果===> Severe deviation!
120918
检测结果===> Severe deviation!
115085
检测结果===> Severe deviation!
95546
检测结果===> Severe deviation!
111024
检测结果===> Severe deviation!
99334
检测结果===> Severe deviation!
107032
检测结果===> Severe deviation!
-906833
检测结果===> Severe deviation!
109814
检测结果===> Severe deviation!
106402
检测结果===> Severe deviation!
155826
检测结果===> Occlusion
102593
检测结果===> Occlusion
96532
检测结果===> Occlusion
Traceback (most recent call last):
  File "D:/大学作业/3 下/Courses/图像与视频信息处理/提交的内容/源代码/实验三源代码/Video-Anomaly-Detection/test.py", line 46, in <module>
    result = do_match.match2frames(pre_frame, frame)#将得到 result
  File "D:\大学作业\3 下\Courses\图像与视频信息处理\提交的内容\源代码\实验三源代码\Video-Anomaly-Detection\do_match.py", line 73, in match2frames
```

```
348.     M, mask = cv2.findHomography(src_pts, dst_pts,cv2.RANSAC,5.0)
349. cv2.error: OpenCV(3.4.2) C:\projects\opencv-
    python\opencv\modules\calib3d\src\ptsetreg.cpp:169: error: (-
    215:Assertion failed) count >= 0 && count2 == count in function 'cv::RANSACPointSetRe
    gistrator::run'
350.
351.
352. 进程已结束,退出代码 1
```

我们将上述结果复制到 word 中，借助查找功能，统计结果如下图所示：

图 4-1 171 个总结果数

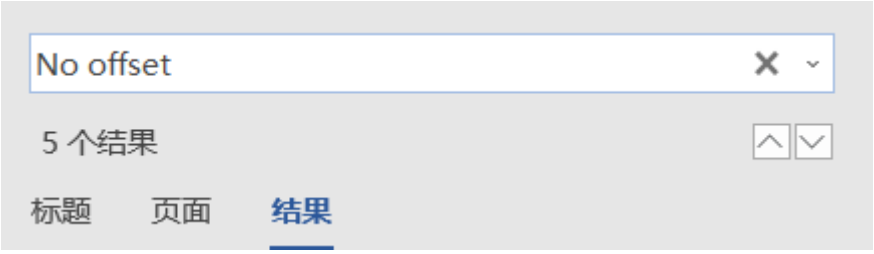


图 4-2 5 个未偏移数

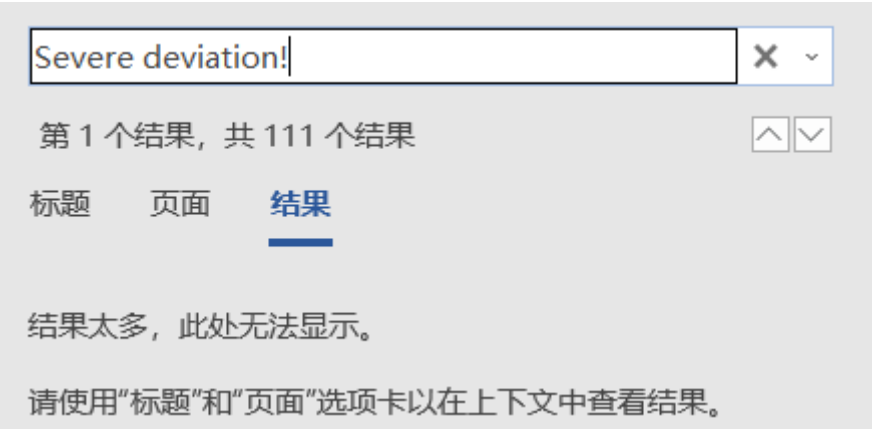


图 4-3 111 个严重偏移数

图 4-4 160-111=49 个轻微偏移数

图 4-5 6 个完全偏移数

按照比例，我们认为在 171 个“帧”中，2.5/12 为未偏移；5.5/12 为轻微偏移；2/12 为严重偏移；2/12 为完全偏移。（实际上由于没有经过训练，我这个评价是没有什么大意义的，因为各个状态之间的阈值只是经过我肉眼的微调）。

那么就以控制变量法为例，如果设定无偏移是正常（0），其余都是异常（1），表如下：

	实际正例	实际负例
预测正例 P	TP = 135	FP = 31
预测负例 N	FN = 0	TN = 5

则：

TPR = TP/(TP+FN) = 135/135

FPR = FP/(FP+TN) = 31/(31+5) = 31/36

R = TPR = 135/135

P = TP/(TP+FP) = 135/(135+31) = 135/166

A = (TP+TN)/总样本数 = 140/171

其余三种（以轻微偏移为正常，其余为异常；以严重偏移为正常，其余为异常；以完全偏移为正常，其余为异常）就不分析了。

4.2 思考题

- 如何有效地对异常进行描述和建模；

异常检测经典模型思想有：

1）统计检验方法

使用这类方法基于的基本假设是，正常的数据是遵循特定分布形式的，并且占了很大比例，而异常点的位置和正常点相比存在比较大的偏移。

比如高斯分布，在平均值加减 3 倍标准差以外的部分仅占了 0.2%左右的比例，一般我们把这部分数据就标记为异常数据。

使用这种方法存在的问题是，均值和方差本身都对异常值很敏感，因此如果数据本身不具备正态性，就不适合使用这种检测方法。

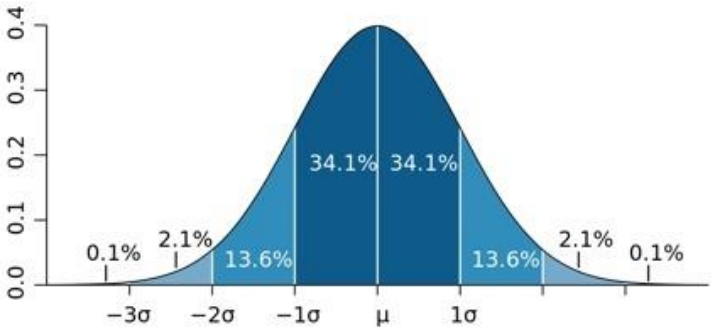


图 4-6 统计检验方法

2）基于深度的方法

基于深度的方法，即从点空间的边缘定位异常点，按照不同程度的需求，决定层数及异常点的个数。如下图所示，圆中密密麻麻的黑点代表一个个数据点，基于的假设是点空间中心这些分布比较集中、密度较高的点都是正常点，而异常点都位于外层，即分布比较稀疏的地方。

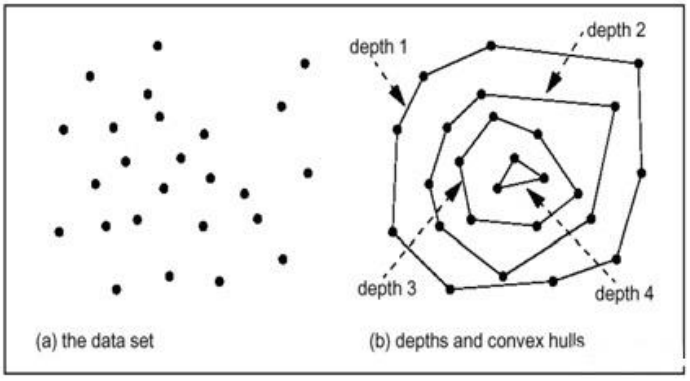


图 4-7 基于深度的方法

如下图，最外层点的深度为 1，再往内几层深度一次为 2、3、4…… 若我们设置阈值 k=2，那么深度小于等于 2 的点就全部为异常点。这一方法最早由 Tukey 在 1997 年首次提出。但这个基础模型仅适用于二维、三维空间。现在有很多流行的算法都借鉴了这种模型的思想，但通过改变计算深度的方式，已经可以实现高维空间的异常检测，如孤立森林算法。

3）基于偏差的方法

这是一种比较简单的统计方法，最初是为单维异常检测设计的。给定一个数据集后，对每个点进行检测，如果一个点自身的值与整个集合的指标存在过大的偏差，则该点为异常点。

具体的实现方法是，定义一个指标 SF（Smooth Factor），这个指标的含义就是当把某个点从集合剔除后方差所降低的差值，我们通过设定一个阈值，与这些偏差值进行比较来确定哪些点存在异常。这个方法是由 Arning 在 1996 年首次提出的。

4）基于距离的方法

基于距离的方法，即计算每个点与周围点的距离，来判断一个点是不是存在异常。基于的假设是正常点的周围存在很多个近邻点，而异常点距离周围点的距离都比较远。

有一个比较古老的 DB 基础模型，是 1997 年被首次提出的，基本思想是：我们给定一个半径 ε 和比例 π，假设对点 p 进行异常检测，若与 p 点的距离小于半径 ε 的点在所有点中的占比低于 π，则点 p 为异常点。比如下图中的 p1 和 p2 两个点，它们方圆 ε 的范围内没有点，就会被模型标记异常。

之后基于最初这个模型，又出现了基于嵌套循环、基于网格的距离模型，再之后就是我们熟知的 kNN、KMeans，都可以通过计算距离来做异常检测。

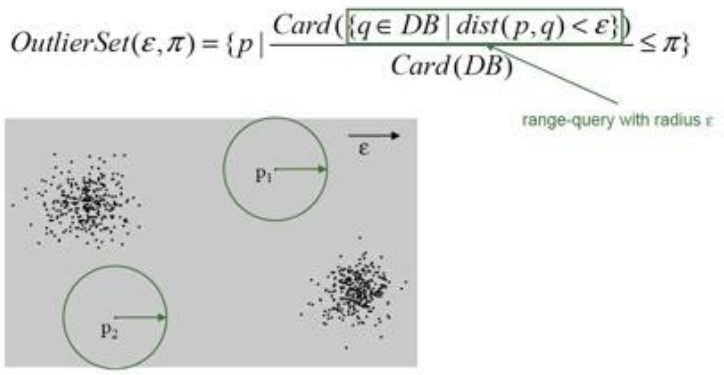


图 4-8 基于距离的方法

5）基于密度的方法

与基于距离的方法类似，该类方法是针对所研究的点，计算它的周围密度和其临近点的周围密度，基于这两个密度值计算出相对密度，作为异常分数。即相对密度越大，异常程度越高。基于的假设是，正常点与其近邻点的密度是相近的，而异常点的密度和周围的点存在较大差异。

设计这种方法的动机是，基于距离的异常检测方法不能很好地处理一些密度存在差异的数据集。如下图中的数据点分布，如果使用基于距离的模型，半径和比例已经设定好了，点 o2 很容易被识别为异常点，因为右上的 C1 子集中很多点与周围点的距离要比 o2 还小，其中很多点就会被标为正常点。但如果从密度的角度来看，o2 更像是一个正常点。所以无论单从哪个角度看，我们都可能会忽略另一个维度上的特征，因此还是要根据具体场景和目标任务，以及数据集本身的特点来进行算法的选择，或是进行算法的结合。

- 不同场景下的异常是否可以用统一的模型来进行建模。

根据思考题的第一题的详细分析，我们不难知道，在不同的场景下，有着不一样的异常检测模型，所以我觉得不同场景下的异常应该是需要多种模型来建模的。

参考文献

[1]. SIFT: design and analysis of a fault-tolerant computer for aircraft control : John H. Wenslfy, Leslje Lamport, Jack Goldberg, Milton W. Green, Kari N. Levttl, P. M. Melliar-Smith, Robert E. Shosiak and Charles B. Weinstock. Proc. IEEE 66, (10) 1240 (October 1978)[J]. Pergamon,1979,19(3).

[2]郝海江. 基于监控视频的人群异常行为检测方法研究与实现[D].桂林理工大学,2020.

[3]周涵,马增强,许丹丹,钱荣威.基于改进 ORB 算法的特征点匹配研究[J].石家庄铁道大学学报(自然科学版),2021,34(02):52-58.

[4]李俊学,张国良,陈钰婕.移动机器人VSLAM和VISLAM技术综述[J].四川轻化工大学学报(自然科学版),2021,34(03):77-86.

备注

关于源代码的使用环境等问题的说明：

JetBrains PyCharm 2017.1 x64，Anaconda2020.02,python3.7.6，直接运行 test.py 即可弹出窗口，显示视频的实时处理结果。