《计算机视觉-openCV应用技术》

第八章 目标跟踪

李策

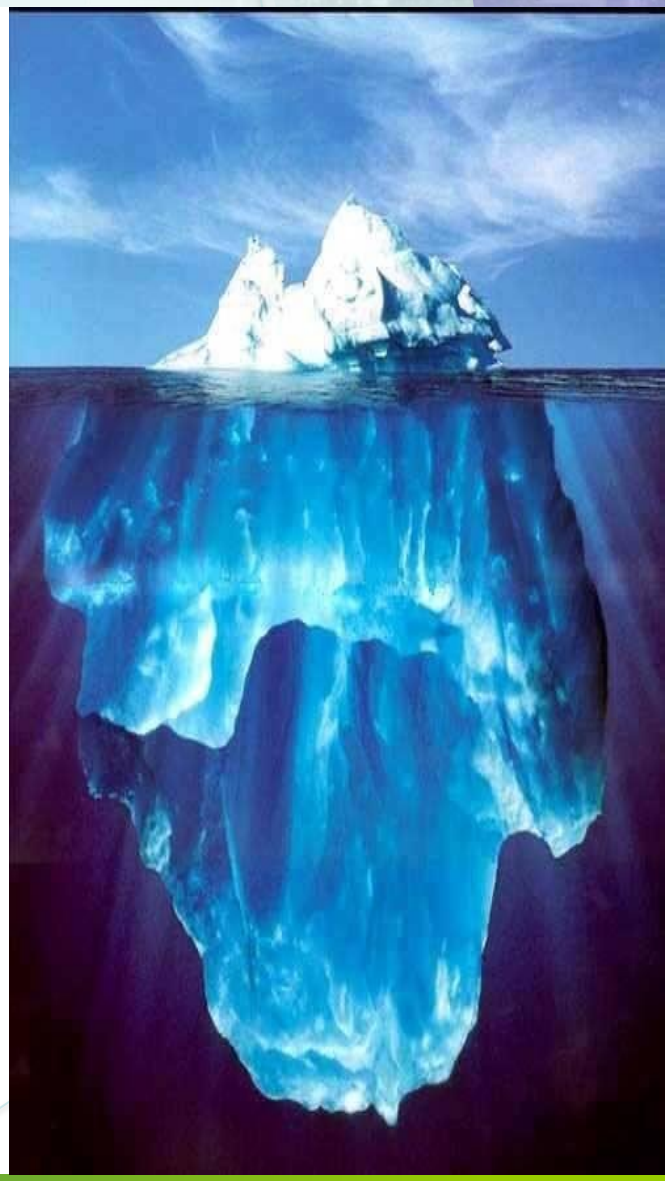
中国矿业大学（北京）计算机科学与技术系

E-mail: celi@cumtb.edu.cn

提纲

第八章 目标跟踪

- 8.1 检测移动的目标
- 8.2 背景分割器：KNN、MOG2、GMG
  - 8.2.1 均值漂移和CAMShift
  - 8.2.2 彩色直方图
  - 8.2.3 返回代码
- 8.3 CAMShift
- 8.4卡尔曼滤波器
  - 8.4.1 预测和更新
  - 8.4.2 范例
  - 8.4.3 一个基于行人跟踪的例子
  - 8.4.4 Pedestrian类
  - 8.4.5 主程序
- 8.5 总结



8.1 检测移动的目标

为了跟踪视频中的所有目标，首先要完成的任务是识别视频帧中那些可能包含移动目标的区域

有很多实现视频跟踪的方法：

- 跟踪所有移动目标时，帧之间的差异
- 跟踪视频中移动的手，基于皮肤颜色的均值漂移
- 跟踪对象的一方面时，模板匹配

8.1 检测移动的目标

基本的运动检测

计算帧之间的差异，或考虑“背景”帧之间与其他帧之间的差异

下面来介绍一个例子（P136）：

A.在导入模块之后，打开通过系统默认的摄像头获得视频图像，将第一帧设置为整个输入的背景

以后读取的帧都会计算其与背景之间的差异

```
diff = cv2.threshold(diff, 25, 255, cv2.THRESH_BINARY)[1]
```

B.对帧进行预处理，将帧转换为灰阶，模糊处理

```
gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
gray_frame = cv2.GaussianBlur(gray_frame, (21, 21), 0)
```

8.1 检测移动的目标

C.计算与背景帧的差异，得到差分图，应用阈值得到一幅黑白图像，膨胀处理图像，对孔和缺陷进行归一化处理

```
diff = cv2.absdiff(background, gray_frame)
diff = cv2.threshold(diff, 25, 255, cv2.THRESH_BINARY)[1]
diff = cv2.dilate(diff, es, iterations = 2)
```

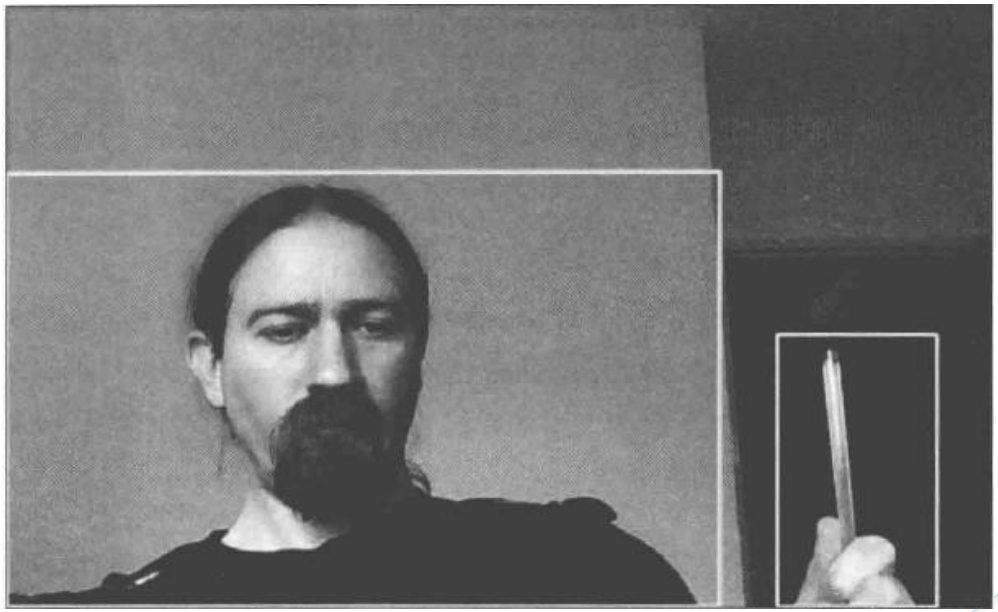
8.1 检测移动的目标

OpenCV中提供了两个非常有用的函数：

•cv2.findContours: 该函数计算一幅图像中目标的轮廓

•cv2.boundinRect : 该函数计算矩形的边界框

最后，我们得到如下结果：



8.2 背景分割器：KNN、MOG2和GMG

OpenCV中提供了一个称为BackgroundSubtractor 的类：

•便于分割前景与背景

•通过机器学习提高背景检测效果

•将分类结果保存到文件

下面通过例子介绍BackgroundSubtractor 类：

背景分割器：

•K-Nearest(KNN)

•Mixture of Gaussians(MOG2)

•Geometric Multigid(GMG)

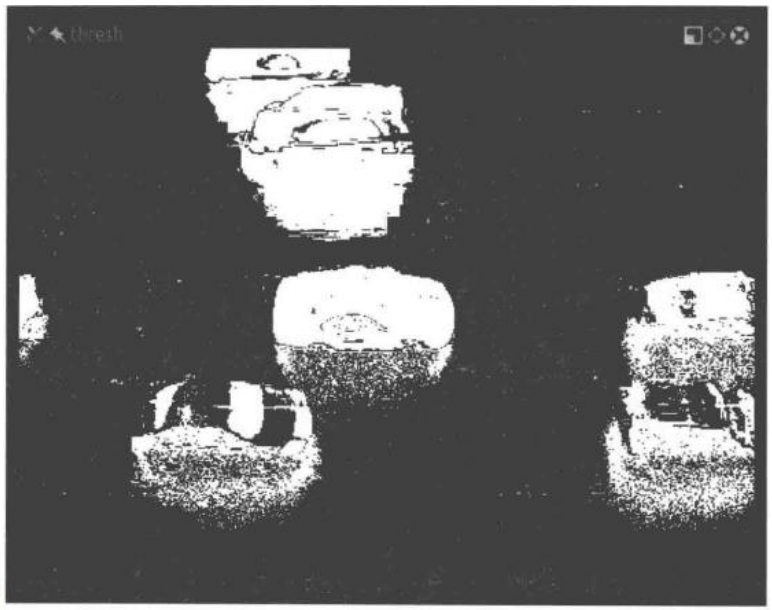
8.2 背景分割器：KNN、MOG2和GMG

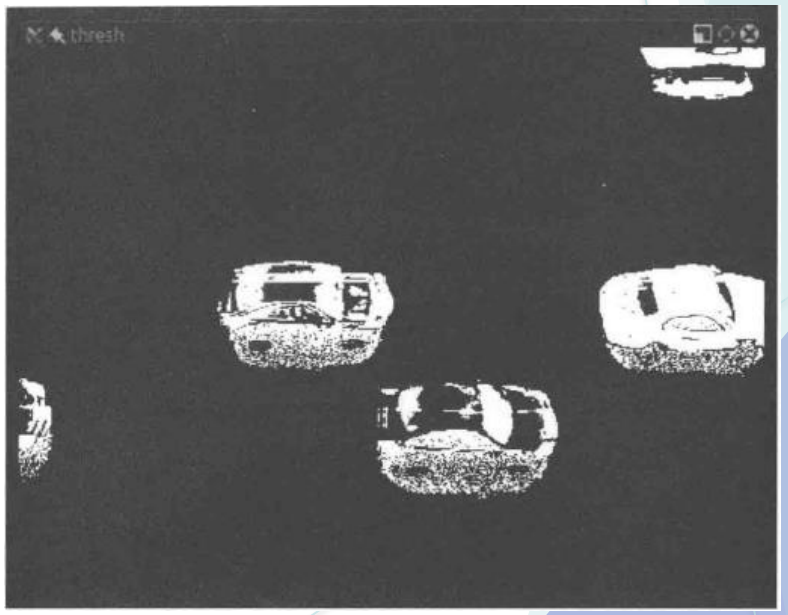
BackgroundSubtractor 类的特征：

•对每帧的环境进行“学习”

•计算阴影，排除检测图像的阴影区域

左图是没有经过阴影检测的背景分割，右图是阴影检测的例子



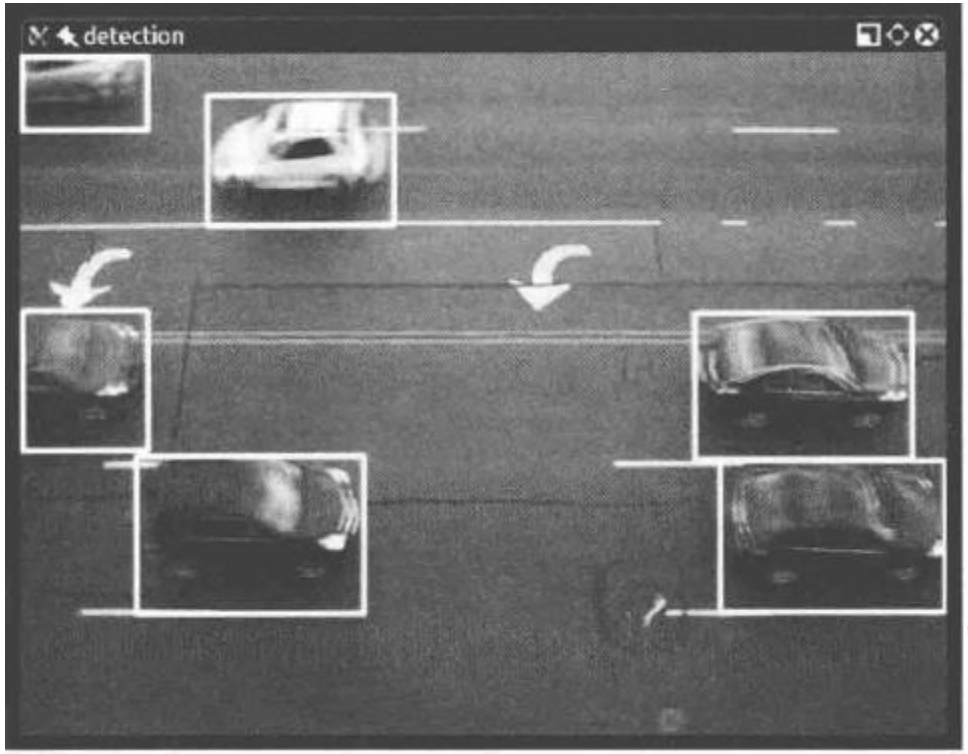


8.2 背景分割器：KNN、MOG2和GMG

阴影检测并非完美，但有助于将轮廓按原始形状进行还原。

下面是个用BackgroundSubtractorKNN来实现运动检测的例子（p140）：

运动检测的结果如下图:



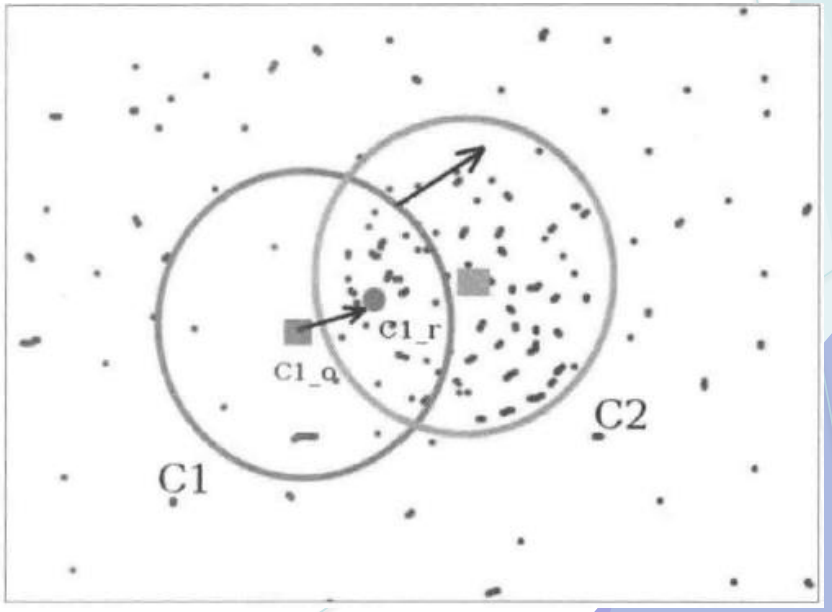
8.2.1 均值漂移和CANShift

背景分割是一种非常有效的技术，但并不是唯一可用的技术

均值漂移（Meanshift）是一种目标跟踪算法：

- 寻找概率函数离散样本的最大密度
- 计算在下一帧中的最大密度
- 给出目标的移动方向

下图是这个过程的可视化表示：



8.2.1 均值漂移和CANShift

在上面的代码中，作者通过HSV值来跟踪阴影，结果如下：

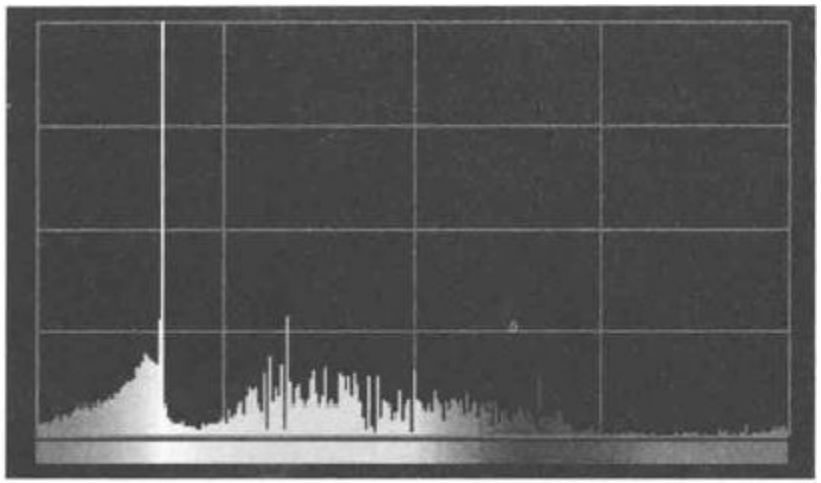
8.2.2 彩色直方图

OpenCV内建函数：

•calcHist

•calcBackProject

X轴是色彩值，Y轴是相应色彩值的像素数量



8.2.2 彩色直方图

1.calcHist函数

OpenCV的calcHist()函数具有以下的python签名：

calcHist(…)

calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]]) -> hist

参数描述：

参数	参数说明
images	该参数是源数组。它们应该具有相同的深度，如 CV_8U 或 CV_32F，以及相同的尺寸。每个图像都可以有任意数目的通道
channels	该参数是 dims 个通道列表，它们用来计算直方图
mask	该参数是可选的掩码。如果矩阵不是空矩阵，则必须是与 images[i] 大小相同的 8 位数组。非零掩码元素标记直方图中计数过的数组元素
histSize	该参数表示每个维度下直方图数组的大小
ranges	该参数是每一维度下直方图 bin 的上下界的 dims 数组的数组
hist	该参数表示输出直方图，是一个 dims(维度) 维稠密 (或稀疏) 数组
accumulate	该参数是累计标志。如果设置它，那么当分配时，直方图在开始时不清零。这个特征使用户可以计算多个数组集合的单个直方图，或者及时更新直方图

8.2.2 彩色直方图

2. calcBackProject函数

•calcBackProject函数在均值漂移算法中发挥着至关重要的作用

•可得到直方图并将其投影到一幅图像上，其结果是概率

•calcBackProject函数给出的是一种概率估计：一幅图像等于或类似于模型图像的概率

8.2.2 彩色直方图

3.总结

calcHist函数从图像提取色彩直方图，对图像的颜色进行展示

calcBackProject函数可用来计算图像的每个像素属于原始图像的概率

8.2.3 返回代码

A.导入模块，标记初始感兴趣区域

```
cap = cv2.VideoCapture(0)
ret,frame = cap.read()
r,h,c,w = 300,200,400,300
track_window = (c,r,w,h)
```

B.提取ROI并将其转换为HSV色彩空间

```
roi = frame[r:r+h, c:c+w]
hsv_roi =  cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

8.2.3 返回代码

C.创建一个包含具有HSV值的ROI所有像素的掩码，HSV值在上界与下界之间：

```
mask = cv2.inRange(hsv_roi, np.array((100., 30.,32.)),
np.array((180.,120.,255.)))
```

D.下面是计算ROI的直方图

```
roi_hist = cv2.calcHist([hsv_roi],[0],mask,[180],[0,180])
cv2.normalize(roi_hist,roi_hist,0,255,cv2.NORM_MINMAX)
```

8.2.3 返回代码

E.均值漂移在达到收敛之前会迭代多次，但并不能保证一定收敛。

OpenCV允许指定停止条件，这是一种指定均值漂移终止一系列计算行为的方式：

```
term_crit = ( cv2.TERM_CRITERIA_EPS |
cv2.TERM_CRITERIA_COUNT, 10, 1 )
```

F.设置无限循环从摄像头中获取帧，切换到HSV色彩空间：

```
if ret == True:
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

G.现有一个HSV数组，执行直方图反向投影：

```
dst = cv2.calcBackProject([hsv],[0],roi_hist,[0,180],1)
```

8.2.3 返回代码

H.该矩阵最终会传递给meanShift，它与跟踪窗口和终止条件一起作为cv2.meanShift函数的Python签名：

```
ret, track_window = cv2.meanShift(dst, track_window, term_crit)
```

I.计算窗口新坐标，在帧上绘制矩形并显示：

```
x,y,w,h = track_window
img2 = cv2.rectangle(frame, (x,y), (x+w,y+h), 255, 2)
cv2.imshow('img2',img2)
```

8.3 CAMShift

虽然CAMShift增加了均值漂流的复杂性，但用CAMShift实现前面程序的功能与用均值漂流实现在复杂性上差不多

注意区别：

在调用CAMShift后，会根据具体的旋转来绘制矩阵，这种旋转会与被跟踪对象一起旋转

阅读CAMShift重新实现前面例子的代码，可以看出，不同之处在于：

```
ret, track_window = cv2.CamShift(dst, track_window, term_crit)
pts = cv2.boxPoints(ret)
pts = np.int0(pts)
img2 = cv2.polylines(frame,[pts],True, 255,2)
```

8.3 CAMShift

CAMShift方法的签名与均值漂流的相同

boxPoints函数会找到被旋转矩阵的顶点，而折线函数会在帧上绘制矩形的线段

上面介绍了三种目标跟踪方法：

- 基本的运动检测
- 均值漂移
- CAMShift

下面介绍另一种目标跟踪方法：Kalman(卡尔曼)滤波器

8.4 卡尔曼滤波器

卡尔曼滤波器会对含有噪声的输入数据流进行递归操作，并产生底层系统状态在统计意义上的最优估计

8.4.1 预测和更新

卡尔曼滤波器可分为两个部分：

- 预测：使用当前点计算的协方差来估计目标的新位置
- 更新：记录目标的位置，为下一次循环计算修正协方差

从书中的代码可推测：

Predict()函数是用来估计目标位置的

Correct()函数是用来修正卡尔曼滤波器的预测结果

8.4.2 范例

将卡尔曼滤波器和 CAMShift 结合起来，获得更高的精确度和性能

先分析一个“鼠标跟踪”的例子，将绘制一个空帧和两条线：

一条线对应于鼠标的实际运动，另一条线对应于卡尔曼滤波器预测的轨迹

思路：

- 在导入包后，创建大小为800×800的空帧，初始化测量坐标和鼠标运动预测的数组
- 定义鼠标移动的回调函数（Callback），用来绘制跟踪结果
- 窗口初始化并设置回调函数

8.4.2 范例

卡尔曼滤波器类的构造函数有如下可选参数：

- dynamParams：状态的维度
- MeasureParams：测量的维度
- ControlParams：控制的维度
- Vector.type：所创建的矩阵类型

8.4.3 一个基于行人跟踪的例子

跟踪监控视频中的行人

1.应用程序工作流程

遵循以下的逻辑：

a)检查第一帧

b)检查后面输入的帧，从场景的开始通过背景分割器来识别场景中的行人

c)为每个行人建立ROI，并利用Kalman/CAMShift来跟踪行人ID

d)检查下一帧是否有进入场景的新行人

在实际应用中，需要识别进入场景的新行人，但现在的重点是利用CAMShift和Kalman滤波器算法来跟踪一开始出现在视频场景中的目标

8.4.3 一个基于行人跟踪的例子

2.函数式编程与面向对象编程

函数式编程：是一种编程范式，将程序当成估算数学函数，允许函数返回函数，允许函数作为另一个函数的参数

优势：

- 可以做什么
- 可以避免什么

8.4.4 Pedestrian类

创建Pedestrian类的主要原因是卡尔曼滤波器的性质

可以通过历史观测来预测对象的位置，根据实际数据来校正预测，但只能对一个对象执行这些操作。因此，每个被跟踪的对象都需要一个卡尔曼滤波器。

由此，Pedestrian类会包含卡尔曼滤波器、彩色直方图以及感兴趣区域的信息，这些信息会被CAMShift算法使用。

教材P155是Pedestrian类

8.4.4 Pedestrian类

Pedestrian类：

核心：背景分割器对象，能识别感兴趣的区域和与之相对应的运动目标

- 当程序启动时，提取每个区域，实例化Pedestrian类，传递ID、帧以及跟踪窗口的坐标
- 构造函数：计算给定ROI的直方图，设置卡尔曼滤波器，并将其与对象的属性关联
- Update方法：传递当前帧并将其转换为HSV，计算行人HSV直方图的反向投影
- CAMShift或均值漂移：跟踪行人的运动，根据行人实际位置校正卡尔曼滤波器
- 以点的形式来绘制CAMShift/均值漂移和卡尔曼滤波器
- 打印行人信息

8.4.5 主程序

现在获得了Pedestrian类中每个对象的所有信息，分析程序的主函数：

- 加载视频，初始化背景分割器，设置20帧作为影响背景模型的帧
- 创建主显示窗口，设置行人字典和firstFrame标志，该标志使得背景分割器利用这些帧来构建历史
- 设置循环，一行一行地读取摄像头的帧
- 用BackgroundSubtractorKNN来构建背景模型的历史
- 通过对前景掩模采用膨胀和腐蚀的方法来识别斑点及周围边框
- 识别图像轮廓，对第一帧中行人的每个轮廓进行实例化
- 对于每个检测到的行人，执行update()来传递当前帧
- 将firstFrame标志设置为False,表示不会跟踪更多的行人，只是跟踪已有的行人
- 窗口显示结果

8.4.5 主程序

进一步改进的方案：

如果卡尔曼预测的行人位置在帧之外，就可以删除该行人对象

检验是不是每个检测到的运动目标都与现有的行人实例相对应，如果不是，则为其创建一个实例

训练SVM，并对每个运动的目标执行分类操作，以此来确定该运动目标的特性与要跟踪目标的特性是否一致

8.5 总结

视频分析和目标跟踪：

- 具有基本运动检测技术的视频背景分割器
- 视频分析算法

a) 均值漂移

b) CAMShift

- 卡尔曼滤波器

通过本章学习，巩固了OpenCV和机器学习的基础知识，这为下一章学习人工神经网络打下了坚实的基础。