第9章 视频分析

这一章将对目标跟踪进行介绍。我们首先介绍光流的定义和使用光流跟踪像素点，接着介绍光流的一个应用，视频稳像。OpenCV 中实现了 8 种不同的目标跟踪方法，我们将对其中部分方法进行介绍并比较它们的性能。之后我们还将对使用 OpenCV 跟踪多目标进行介绍。最后两个知识点 Kalman Filter、MeanShift 和 CAMShift 为选学内容。

9.1 使用光流进行运动估计

什么是光流？如所示假设我们要跟踪某段视频第t帧图像中坐标为(x, y)的点，在下一帧$t + dt$帧图像中，这一点运动到了位置$(x + dx, y + dy)$，dx和dy分别是这个点在x和y方向上的小位移。如图 9-1 所示。

第t帧图像

第$t + dt$帧图像

图 9-1

某一点的光流是这一点的速度矢量，它由式 9-1 定义：

$$(u, v) = (\frac{dx}{dt}, \frac{dy}{dt})$$ 式 9-1

这里，x方向的速度矢量u为x方向上的位移除以时间dt。同理，y方向的速度矢量v为y方向上的位移除以时间dt。如图 9-2 所示。

第$t + dt$帧图像

图 9-2

当对整幅图像的所有像素点计算光流时称为稠密光流，若只对图像中的某些特征点计算光流则称为稀疏光流。如图 9-3 所示。

稠密光流

稀疏光流

图 9-3

光流可用于如下应用：

- 运动估计（Motion Estimation）
  视频压缩中经常使用光流。计算光流后可以进行运动补偿，只使用若干位（bits）即可表示图像帧差。
- 目标检测和跟踪（Object Detection and Tracking）
- 视频稳像（Video Stabilization）
- 图像优势平面提取（Dominant Plane Extraction）
- 机器人导航（Robot Navigation）
- 视觉里程计（Visual Odometry）

光流有时是使用光流传感器直接进行测量的。例如，图 9-4 所示的光学鼠标便使用了光流传感器。

图 9-4

下面我们来推导光流公式式 9-2

$$\frac{\partial I}{\partial x}u+\frac{\partial I}{\partial y}v+\frac{\partial I}{\partial t}=0 \qquad \text{式 9-2}$$

式 9-2 是基于亮度恒定（brightness constancy）假设的，即同一点的亮度不随时间变化而改变。也就是说，将图 9-1 的视频转换为灰度图像，在第t帧图像中的(x,y)和第$t+dt$帧图像中的$(x+dx,y+dy)$的亮度是相等的。

如果将视频用三维坐标系统来表示，除x和y维度，还有时间维度t，如图 9-5 所示。我们用I来表示视频，则图 9-5 中的(x,y)和$(x+dx,y+dy)$可表示为$I(x,y,t)$和$I(x+dx,y+dy,t+dt)$。

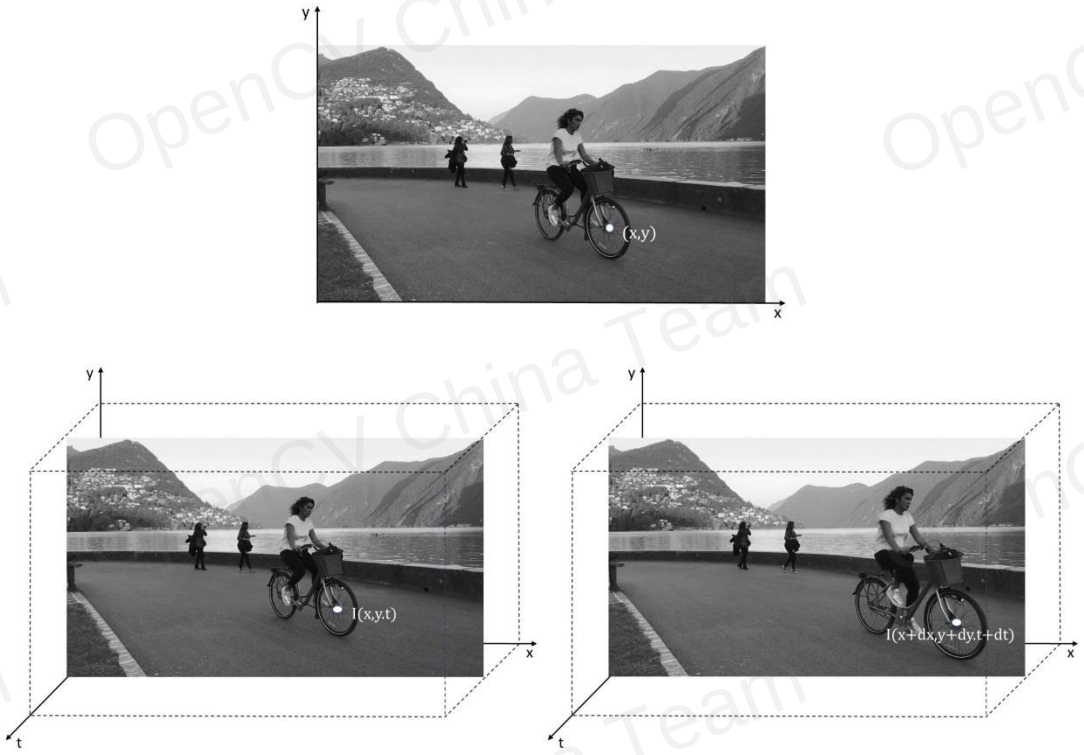


图 9-5

亮度恒定假设可以用式 9-3 来表示：

$$I(x,y,t) = I(x+dx, y+dy, t+dt) \quad \text{式 9-3}$$

将等式右边用一阶泰勒展开为式 9-4：

$$I(x,y,t) = I(x,y,t) + \frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt \quad \text{式 9-4}$$

得到即式 9-5

$$\frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt = 0 \quad \text{式 9-5}$$

将式 9-5 两边除以dt并将式 9-1 代入便得到了光流公式式 9-2。

使用前面章节介绍了 Sobel 算子如图 9-6，可以计算$\frac{\partial I}{\partial x}$和$\frac{\partial I}{\partial y}$，而$\frac{\partial I}{\partial t}$实际上是连续两帧间的帧差。

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

Sobel 算子

图 9-6

式 9-2 是一个有两个未知变量u和v的线性等式，u和v分别是x和y方向的速度矢量。一个等式，两个位置变量，我们必须要做一些假设才能求解u和v。不同的光流算法进行了不同的假设。早期的 Horn and Schunck 光流方法假设整幅图像的光流是平滑的，这个假设提供了足够的约束，使得式 9-2 可以求解。本章我们介绍 Lucas-Kanade 光流。

9.2 Lucas-Kanade 光流算法

Lucas-Kanade 光流算法用于计算离散点的光流，即它属于稀疏光流。如图 9-7 所示，Lucas-Kanade 方法假设，在以需要计算光流的点为中心的邻域内，所有像素点的光流相等。若此邻域为 3 邻域，则有 9 个等式，2 个未知变量，那么式 9-2 就可解。

图 9-7

我们不对计算过程进行深入，式 9-6 给出了 Lucas-Kanade 的求解结果：

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \sum I_x^2 & \sum I_y I_x \\ \sum I_x I_y & \sum I_y^2 \end{pmatrix}^{-1} \begin{pmatrix} -\sum I_x I_t \\ -\sum I_y I_t \end{pmatrix} \quad \text{式 9-6}$$

其中，$I_x = \frac{\partial I}{\partial x}, I_y = \frac{\partial I}{\partial y}, I_t = \frac{\partial I}{\partial t}$。

注意，式 9-4 中泰勒展开只适用于小的dx，dy和dt，即当运动小于 1 个像素时才有效。但通常相邻帧间的运动都大于 1 个像素，我们可以使用图像金字塔（image pyramid）来解决这个问题。图像金字塔是图像的多分辨率表示，最常见的图像金字塔是高斯金字塔，如图 9-8 所示。它是通过高斯核对图像做高斯平滑然后以 2 为比例系数逐级降采样产生的。高斯平滑可以消除高频噪声信号。

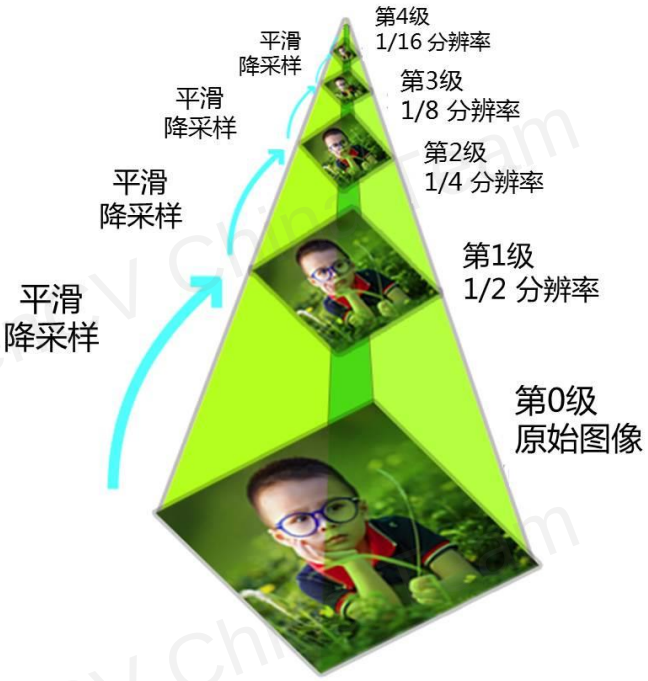


图 9-8

计算光流时，运动矢量是从最顶端图像开始计算，此时运动是亚像素级的。光流通过金字塔逐级向下传播，新一级的准确光流计算则以前一级的光流做为初始值，重复这个过程直到到达金字塔的底端。下面用一个实例来说明 Lucas-Kanade 光流算法。（详细内容见视频讲解和源代码 Lucas_Kanade_tracker.cpp）

9.3 应用：视频稳像

视频稳像是指消除因摄像机的运动而产生的视频图像的抖动，最终产生一个稳定的视频。视频稳像的应用很广泛，在消费级或专业级的摄影中尤其重要。即便只是拍摄图像，对于长时间曝光的情况，稳像可以维持图像静止。在医学诊断的应用中，如内窥镜和结肠镜，需要获得稳定的视频图像以对病灶进行精准定位。类似的，在军事应用中，侦查飞行器获取的视频也需要进行稳定以进行定位、导航、目标跟踪等。机器人应用也如此。

视频稳像的方法按机制可分为机械方法、光学方法和电子方法。

- 机械稳像是通过陀螺仪、加速度传感器等检测摄像机的运动，然后调整图像传感器来补偿摄像机运动。
- 光学稳像是通过光学部件调整光路来补偿摄像机的运动
- 电子稳像不需要物理补偿摄像机的运动，它是基于对连续帧图像进行运动估计，然后对图像做数字处理来达到稳像。其主要有 3 步：
  1) 运动估计：估计连续两帧图像见的变换矩阵
  2) 运动补偿：对运动进行修正
  3) 图像生成：根据补偿后的运动生成新的图像

这一节我们介绍如何使用 OpenCV 来实现一个简单的视频稳像器。

摄像机的运动一般包括平移、旋转和缩放。这一节介绍一种快速鲁棒的数字视频稳像算法，它是基于二维运动模型的，即包含平移、旋转和缩放的欧氏变换（相似变换）。在图 9-9 可以看到，在欧氏运动模型中，图像中的正方形可以变换为不同位置、不同大小、旋转不同角度的其它正方形。它比仿射变换和单应性变换更严格，但对于稳像应用而言已经足够了，这是由于连续视频帧间摄像机的运动通常比较小。

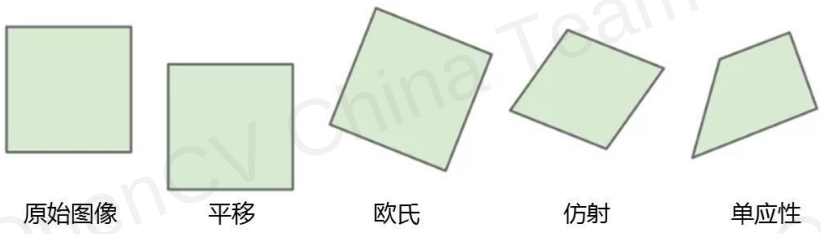


图 9-9

这个算法通过跟踪连续帧间的特征点来估计帧间的运动进而进行补偿。图 9-10 所示的流程图显示了此方法的基本过程。

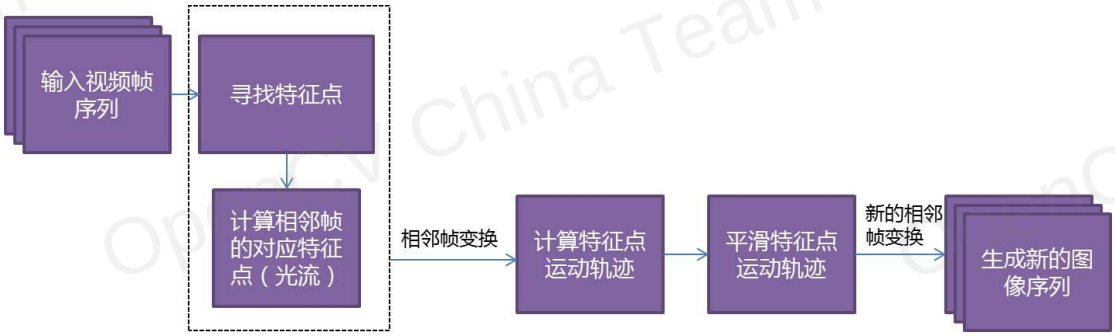


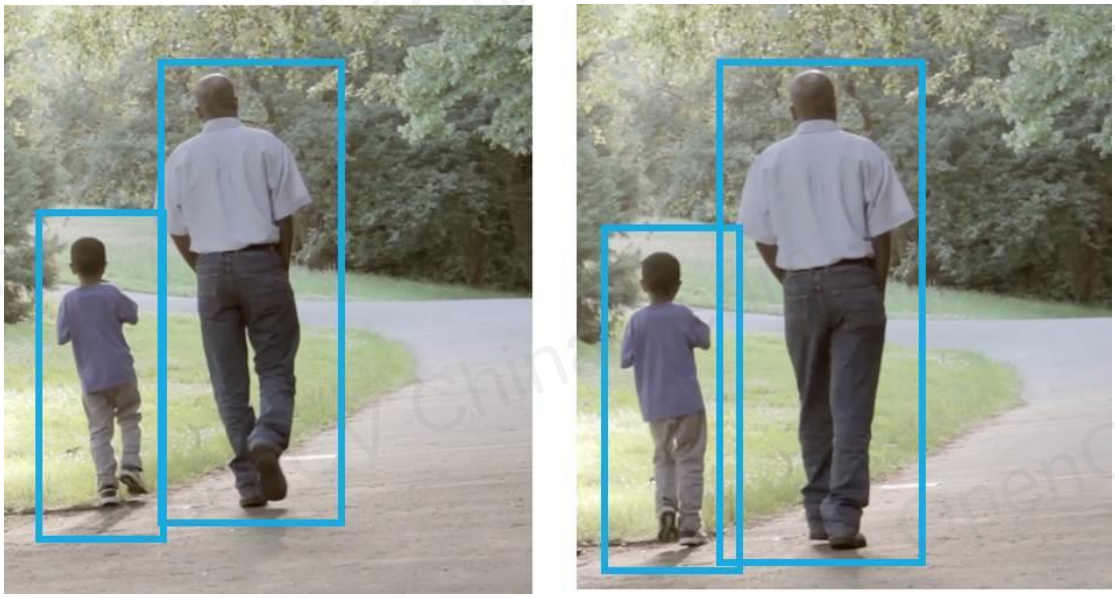
图 9-10

下面我们具体来看是如何实现的。（详细内容见视频讲解）

9.4 目标跟踪算法

跟踪有不同的类型

- 稀疏光流：跟踪某些特征点的运动
- 稠密光流：跟踪图像中每一个像素点的运动
- Kalman 滤波：应用于登月系统、跟踪导弹、飞机导航系统等
- MeanShift 和 CamShift
- 直接匹目标检测的矩形框，如图 9-11 示



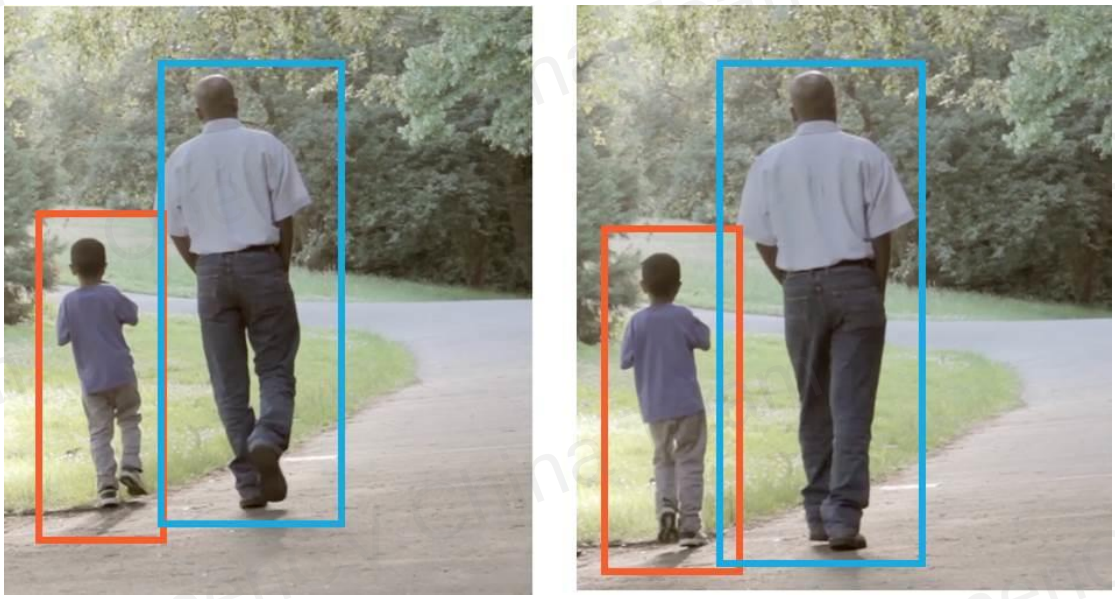


图 9-11

回想一下上一章中我们介绍的目标检测，但如果直接对每一帧图像进行目标检测，然后用检测的结果进行跟踪是不可行的。这涉及到一下几点：

- 跟踪保留了目标的 ID。对每帧图像进行目标检测，结果可能会有若干矩形框，前后两帧的目标矩形框的排列顺序可能不相同，这就需要匹配连续两帧图像上的矩形框。
- 跟踪通常速度比检测快。目标检测需要在整幅图像上进行搜索，而跟踪只需要在当前帧中对上一帧目标位置的小区域进行搜索。
- 当目标的外观变化很大，或者目标被遮挡时，目标检测可能失败，但跟踪可以恢复目标位置。
- 在实际应用中，检测和跟踪通常是联合起来使用。如先进行目标检测，然后使用跟踪在后续若干帧中进行目标跟踪；当目标运动速度太快，跟踪可能失败，这时可以使用检测再找出目标物体。也就是说每隔若干帧可进行目标检测确保没有丢失目标，而连续帧之间使用跟踪提升处理速度。

下面我们介绍几个 OpenCV 中实现目标跟踪算法。所有跟踪算法均有三个输入，即当前帧图像、上一帧图像和上一帧图像中目标的矩形框（bounding box），而算法的输出为当前帧图像中目标的矩形框。

我们首先看一下模板匹配（template matching）方法。模板匹配的方法做了几个假设：

- 目标的外观在连续两帧图像中的变换很小
- 目标在连续两帧的运动很小

基于上面两个假设，模板匹配的方法从上一帧图像中剪裁出目标区矩形框，然后在当前帧中搜索上一帧中目标矩形框的最佳匹配区域。判断两个区域是否匹配的方法可以用像素差的平方和，或者用速度更快的归一化互相关。由于假设目标在连续两帧间的运动很小，搜索区域可以限制在上一帧的矩形框区域小邻域内。

在实际问题中，目标的外观在连续帧间可能变化较大，仅用像素差来判断两个区域是否匹配是不够的。同时，目标矩形框也可能会因为跟踪过程因模板匹配产生的误差累积而漂移

目标。那么，能否在跟踪过程中持续的学习目标外观特征来克服这个问题？多示例学习跟踪算法（MIL Tracker）便是这种方法。MIL 是 Multiple Instance Learning 的缩写，即多示例学习。

在介绍多示例学习之前，先来了解一下监督学习（supervised learning）。在上一章介绍图像分类（image classification）时，我们介绍了有监督的图像分类方法。例如，要得到一个狗的分类器，需要给算法输入各种狗的图像样本和非狗的图像样本，如图 9-12。

图 9-12

很多应用中较难获得非常干净的训练样本，多示例学习可以缓解这个问题。对于多示例学习，无需给训练集中的每一个样本附加一个标签，只需对每个图像包（a bag of images）赋予一个标签即可，如图 9-13 所示。

图 9-13

图像包中图像样本可以包含不正确的样本。如果一个图像包被标记为正样本标签，必须保证包里至少有一张图像为正样本图像，而其他图像可以为负样本图像；但是，如果一个图像包被标记为负样本标签，那么包里所有图像必须为负样本图像。对于狗分类器这个例子，正样本图像包至少要包含一张狗的图像，如图 9-13 中绿色圆圈所示；负样本图像包则不能包含狗的图像，如图 9-13 橙色圆圈所示。

再来比较一下监督学习和多示例学习的区别，如图 9-14 所示。在监督学习中，$x_1, x_2, \dots, x_n$代表训练样本，$y_1, y_2, \dots, y_n$为训练样本的标签，1 表示样本为狗，0 表示样本为非狗。在多示例学习中，$\{X_1, X_2, \dots, X_n\}$代表图像包，每个图像包X_i包含若干张训练样本图像$\{x_{i1}, x_{i2}, \dots, x_{in}\}$，仅每个图像包有一个标签$y_i$，而非每个训练样本。

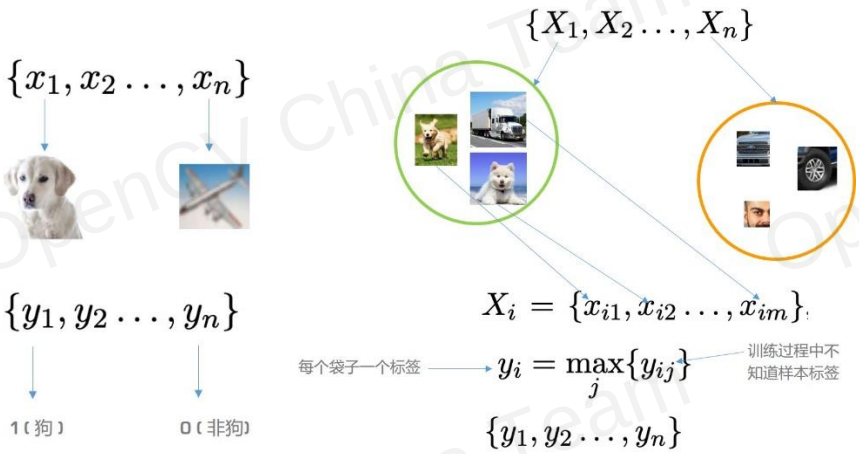


图 9-14

监督学习有多种方法，如逻辑回归（logistic regression）、支持向量机（support vector machine）、boosting、随机森林（random forest）等。而对于多示例学习的方法，这些方法需要进行修改才能运用，这里我们不进行深入介绍。

下面再介绍两个概念，离线训练（offline training）和在线训练（online training）。离线训练是指在训练时已经获取所有训练样本，训练生成的模型在使用时保持不变。而有些情况，不能一次性获取所有的训练数据，这时需要使用在线训练算法，即模型会随新的训练数据输入而进行更新，这可以使得模型自适应新的应用场景，但由于同步在进行训练，这也造成算法运行速度比较慢。

OpenCV 中实现的 MIL 跟踪算法使用了在线 MIL Boost 方法，即跟踪的同时进行学习。前面我们介绍过 Boosting，即将若干弱分类器组合在一起构成一个强分类器，例如 Haar 级联中的使用的 AdaBoost。

$h(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$　式 9-7

MILBoost 是 Boosting 的一种，具体的实现可以参考相关文章。

下面来简单了解一下 MIL 跟踪器是如何工作的。

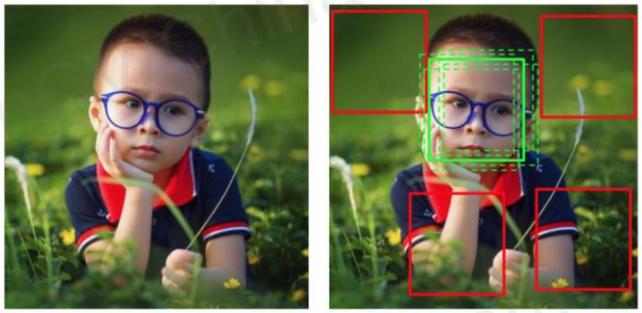


图 9-15

在图 9-15 中，上一帧图像中含有已知的目标矩形框，为了学习目标的特征，可以将矩形框区域剪裁下来，另外再在目标矩形框区域进行随机采样获得更多的正样本。然后将所有的样本放进 MIL 的正样本包中。这样，即便在上一帧图像中矩形框可能并没有准确定位目标，但是经过了随机采样，总有一个剪裁的样本在目标的准确位置，那么就有可能从上一帧不准确的目标位置中恢复准确的目标位置。负样本包中的图像是在目标矩形框以外的区域采样剪裁得到的。

现在来介绍另一种跟踪算法，GOTURN。GOTURN 是一种基于深度学习的跟踪方法。与 OpenCV 中实现的其他跟踪算法不同，GOTURN 的模型是离线训练得到的，所以在使用时需要预先载入模型。

图 9-16 显示了 GOTURN 如何工作。

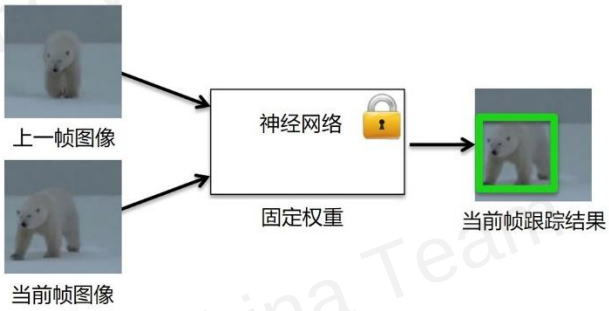


图 9-16

GOTURN 的网络输入为上一帧图像和当前帧图像的目标剪裁区域，上一帧图像的目标矩形框是已知的，当前帧的目标矩形框使用上一帧矩形框的位置进行剪裁。显然，由于目标的运动，从当前帧中剪裁的图片，目标物体已经不在图像中心，所以需要得到当前帧中准确的目标矩形框大小和位置，如图 9-16 绿色矩形框所示。

下面来看如图 9-17 所示神经网络的内部结构，如图 9-17 所示。两幅剪裁区域的图像作为输入送进一系列的卷积层和全连接层，最后输入四个数值。卷积层进行特征提取，全连接层对输入一对图像特征，即当前图像剪裁区域的特征和前一帧图像剪裁区域的特征，输出目标矩形框的左上角和右下角坐标。训练时，使用了大量的有已知目标矩形框的视频图片对，也会使用一些通过几何变换人为生成的图片对，这样减轻了收集数据的工作量。

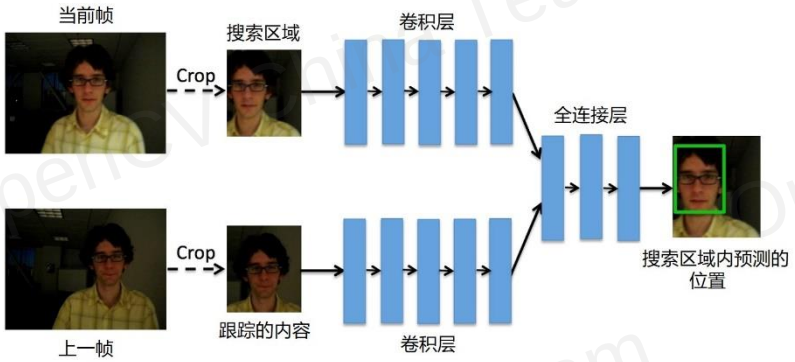


图 9-17

载入训练好的模型后，就可以根据当前帧、上一帧和上一帧的矩形框预测目标矩形框的位置。模型在使用时是不进行更新的，这是与 MIL 跟踪器的不同之处。

总结一下 GOTURN 的特点:

1. 基于深度学习的方法
2. 离线训练
3. 预测当前帧的目标位置，只需要输入上一帧的目标矩形框、当前帧和上一帧图像
4. 较其他方法速度快
5. 对于跟踪在训练中使用的目标效果很好

9.5 OpenCV 中的目标跟踪 API

OpenCV 4 提供了一个新的跟踪 API，它实现了多种单目标跟踪算法。OpenCV 4.1.0 中总共有 8 种不同的跟踪方法（如何使用 API 见视频讲解和源代码 trackSingleObject.cpp）。

1. BOOSTING

   此方法是基于在线的 AdaBoost（AdaBoost 在 HAAR 级联人脸分类器中用到）。这个分类器需要用目标的正样本和负样本在运行时进行训练。初始矩形框由用户给定，或者由另外的检测算法给出。这个矩形框被作为目标的正样本，矩形框以外的剪裁图像作为负样本。对于一幅新的视频帧，分类器会对前一个位置的邻域内的每个像素做判断并记录分类器返回的值，值最大的位置即为新视频帧中目标的位置，这样又得到一个新的正样本。随着新的视频帧的出现，分类器根据新的样本进行更新。

   优势：虽然这个方法已经很老，但其效果还可以。

   劣势：跟踪效果一般。当跟踪失败时，算法也不能得知。

2. MIL

   MIL 的思路跟 BOOSTING 相似。区别在于 MIL 除了将当前目标位置作为正样本，也会在这个位置的小邻域内采样产生其他正样本，显然，产生的这些正样本的目标并不在中心位置。如前面介绍到的，MIL 将若干样本放进一个图像包，这个图像包将有一个标签而非每一个样本。如果图像包中的样本只要有一个正样本，这个图像包就被标记为正样本，如果图像包全是负样本，则图像包被标记为负样本。

   优势：MIL 的跟踪性能很好，它不像 BOOSTING 那样漂移，它也可以处理目标部分遮挡的情况。

劣势：并不能有效知道跟踪失败的情况，也不能处理完全遮挡的情况。

3. KCF（Kernelized Correlation Filters）

   KCF 跟踪算法利用了 MIL 跟踪算法中多个正样本间有很大的重叠区域的特点，进行快速和更准确的跟踪。

   优势：速度比 MIL 快，准确度比 MIL 高，比 BOOSTING 和 MIL 更好的发现跟踪失败。

   劣势：不能处理完全遮挡的情况。

4. TLD （Tracking, learning and detection）

   TLD 将长时间的跟踪任务分解为三个部分：（短时间的）跟踪、学习和检测。根据作者的文章，TLD 对目标进行帧间的跟踪，检测定位目前所有的目标，如果必要修正跟踪，学习估计检测的误差并更新检测器以减少将来的误差。

   优势：在多帧都存在遮挡的情况在 OpenCV 实现的这几个方法中结果最好。同时，在目标有大尺度变化的情况下也是效果最好的。

   劣势：过多的假阳（false positive）。

5. MEDIANFLOW

   MEDIANFLOW 对目标同时进行前向（forward）和后向（backward）跟踪，然后测量这两条轨迹间的差异。最小化“前向后向”误差使得可以可靠的检测到跟踪失败的情况，并给视频序列选取一条可靠的轨迹。在我们的测试中，当目标运动是可预测且运动幅度不大时，MEDIANFLOW 的跟踪效果是最好的。这个算法也能得知跟踪是否失败。

   优势：能够判断跟踪是否失败。当运动是可预测且目标没有遮挡时，跟踪效果非常好。

   劣势：目标运动比较大是会跟踪失败。

6. GOTURN

   在 OpenCV 的跟踪 API 中，GOTURN 是唯一一个基于深度学习的跟踪算法。它也是唯一一个使用离线训练模型的，因此它比其他几个跟踪算法的速度快。OpenCV 的文档对 GOTURN 的描述是不受视角、光线变化和形变的影响。但是实际上，GOTURN 并不能很好的处理目标受遮挡的情况。

7. MOSSE

   很早就已经有使用相关滤波器（correlation filters）来进行跟踪的思想。但由于目标的特征在不同帧间可能差异比较大，仅用上一帧中的目标图像，然后在当前帧图像使用相关性来匹配这个目标的效果是不好的。

   MOSSE（minimum output sum of squared error）使用判别相关滤波（discriminative correlation filter, DCF）进性目标跟踪。MOSSE 的文章发表在 2010 年，简洁是它的一大亮点。它只是对旧的方法做了小的修改，但效果却比其他复杂的模型、随机搜索技术都要好。同时，它的运行速度也非常快。

   MOSSE 跟踪不受光照变化、尺度、姿态和非刚体形变的影响。它也可以根据峰值旁瓣比（peak-to-sidelobe ratio）检测遮挡，这使得跟踪器可以暂停，然后当目标再出现的时候继续跟踪。MOSSE 跟踪算法的速度也更快（450fps 或者更高）。

8. CSRT

   CSRT 跟踪方法扩展了 MOSSE 方法里的 DCF，或者称为 Channel and Spatial Reliability（DCF-CSR）。他们在执行搜索时同时扩展搜索区域，这就保证了选定区域的扩大和定位，改进了对非矩形区域或目标的跟踪。CSRT 只利用了两个特征（HOG 和 Colomames）。它的运行速度相对较慢（25fps），但是跟踪的更准确。

有点意外的是，OpenCV 的跟踪类没有用目标检测来更新跟踪结果的功能。如果想实现这个目的，只能将跟踪结果与检测算法结合并新创建一个新的跟踪类对象。

以上跟踪算法的性能比较，详细内容见视频讲解。

前面是如何使用 OpenCV 跟踪视频中的单个目标，实际应用中可能需要同时跟踪多个目标物体。OpenCV 提供了 Multiobject Tracker 类，实现了简单的多目标跟踪，它仅是独立的处理每一个被跟踪的目标，没有对被跟踪的多目标进行目标间优化。下面我们以一个实例来说明如何使用 OpenCV 的多目标跟踪 API。（详细内容见视频讲解和源代码 trackMultiObjects.cpp）

进一步阅读：

http://www.vision.ee.ethz.ch/boostingTrackers/Grabner2006RealTimeTracking.pdf
http://vision.ucsd.edu/~bbabenko/new/project_miltrack.shtml
https://arxiv.org/abs/1404.7584
http://kahlan.eps.surrey.ac.uk/featurespace/tld/Publications/2011_tpami
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.231.4285&rep=rep1&type=pdf
http://davheld.github.io/GOTURN/GOTURN.pdf
http://docs.opencv.org/trunk/d0/d0a/classcv_1_1Tracker.html

9.6 使用 Kalman 滤波器进行跟踪（选学）

1969 年的阿波罗计划实现了将人送上月球，而 Kalman 滤波是导航系统中不可或缺的一部分。维基百科里介绍到：“Kalman 滤波器对美国海军核弹道导弹潜艇导航系统、巡航导弹如美国海军的 Tomahawk 导弹、美国空军的空中发射巡航导弹的导航系统的实现很重要。可重复使用发射系统的导航系统和飞机器对接国际空间站的姿态控制和导航系统中也用到了 Kalman 滤波。”

Kalman 滤波器（也叫 Kalman-Bucy 滤波器），是由 Rudolf E. Kalman 和 Richard S. Bucy 设计并在 1960 年和 1961 年发表在机械工程期刊上。实际上，物理学家 Peter Swerling 也设计了 Kalman 滤波，并早于 Kalman 一年在天文学期刊上发表。

Kalman 滤波器是每个惯性导航系统的核心。所有商用飞机的导航系统、机器人导航系统和自动驾驶车辆也使用了 Kalman 滤波器。

下面我们来看一下如何使用 Kalman 滤波来跟踪目标。

首先，我们用一个特殊的例子来解释什么是 Kalman 滤波器。假设要构建一个无人机，它可以沿着指定路径飞行到指定位置。虽然无人机装载了 GPS，但不能完全依赖 GPS 来进行导航，原因有两点：1. GPS 的精度为几米，它对高度的精度更差；2. 在指定的路径上 GPS 信号可能不稳定、不可靠。

所以，Kalman 滤波器使用了以下策略：

1. **预测**：根据前一个状态和控制输入（例如无人机的推进力），对系统内部状态进行预测（比如无人机的位置和速度）
2. **更新**：当获得对系统内部状态的新的测量时（比如 GPS 信息），用此更新第 1 步中的预测

先来理解预测阶段。在无人机飞行过程中，我们想准确的知道其位置。假设用一个简单的运动模型来表示无人机的飞行，第k步的位置可以用第$k-1$步时的位置、第$k-1$步时的速度以及第k步与第$k-1$步间的时间间隔进行预测，没有加速度。运动模型由式 9-8 和式 9-9 表示。

$$\boldsymbol{p_k} = \boldsymbol{p_{k-1}} + \boldsymbol{v_{k-1}}\Delta t \quad \text{式 9-8}$$

$$\boldsymbol{v_k} = \boldsymbol{v_{k-1}} \quad \text{式 9-9}$$

其中，p代表三维位置，v代表三维速度矢量，Δt表示时间间隔。

$\boldsymbol{p_k}$和$\boldsymbol{v_k}$称为系统状态，式 9-8 和式 9-9 可以合起来写为式 9-10 所示的一个**状态转移等式**

$$\begin{bmatrix}\boldsymbol{p_k}\\ \boldsymbol{v_k}\end{bmatrix} = \boldsymbol{F_k}\begin{bmatrix}\boldsymbol{p_{k-1}}\\ \boldsymbol{v_{k-1}}\end{bmatrix} \quad \text{式 9-10}$$

矩阵$\boldsymbol{F_k}$称为状态转移矩阵。将上面的等式展开如下：

$$\begin{bmatrix}x_k\\ y_k\\ z_k\\ u_k\\ v_k\\ w_k\end{bmatrix} = \begin{bmatrix}1 & 0 & 0 & \Delta t & 0 & 0\\ 0 & 1 & 0 & 0 & \Delta t & 0\\ 0 & 0 & 1 & 0 & 0 & \Delta t\\ 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x_{k-1}\\ y_{k-1}\\ z_{k-1}\\ u_{k-1}\\ v_{k-1}\\ w_{k-1}\end{bmatrix} \quad \text{式 9-11}$$

将状态向量定义为：

$$\boldsymbol{x_k} = \begin{bmatrix}\boldsymbol{p_k}\\ \boldsymbol{v_k}\end{bmatrix} \quad \text{式 9-12}$$

则式 9-10 可以写为式 9-13

$$\boldsymbol{x_k} = \boldsymbol{F_k}\boldsymbol{x_{k-1}} \quad \text{式 9-13}$$

虽然可以预测位置和速度，但预测是概率的，有**不确定性**。这种不确定性通常可以用均值为μ、方差为δ^2的高斯矩阵分布来表示。对于无人机的情况，状态向量为 6 维向量，即有 3 个位置分量和 3 个速度分量。这种情况可以用一个多变量的高斯分布来表示，即均值向量为 6x1 和协方差矩阵为 6x6，协方差矩阵的对角元素代表了不同维度间的相关性。

在任意一步，我们都记录状态向量$\boldsymbol{x_k}$和协方差矩阵$\boldsymbol{P_k}$，所以我们不仅知道当前的预测，也知道我们对估计的有多确定。$\boldsymbol{x_k}$可以使用式 9-13 来更新。协方差矩阵$\boldsymbol{P_k}$如何更新？

数学恒等式

如果一个随机变量$\boldsymbol{y}$的分布的协方差为$\boldsymbol{\Sigma}$，那么随机变量$\boldsymbol{Ay}$的协方差矩阵则为$\boldsymbol{A\Sigma A^T}$，$\boldsymbol{A}$是任意矩阵。

根据上面的恒等式原理，$\boldsymbol{P_k}$的更新公式如式 9-14：

$$\boldsymbol{P_k} = \boldsymbol{F_k}\boldsymbol{P_{k-1}}\boldsymbol{F_k^T} \quad \text{式 9-14}$$

无人机的状态还可能受某些**控制输入**的影响，例如根据内部的计算，无人机可能朝某个方向推进自己以矫正路径。外部控制输入可以用向量$\boldsymbol{u}$来表示，如式 9-15 所示。

$$\boldsymbol{x_k} = \boldsymbol{F_k}\boldsymbol{x_{k-1}} + \boldsymbol{B_k}\boldsymbol{u_k} \quad \text{式 9-15}$$

矩阵$\boldsymbol{B_k}$是控制矩阵，它表示了$\boldsymbol{u_k}$如何影响状态向量$\boldsymbol{x_k}$。例如，如果$\boldsymbol{u_k}$是一个三维加速度向量，$\boldsymbol{B_k}$则为

$$\boldsymbol{B_k} = \begin{bmatrix} \frac{(\Delta t)^2}{2} & 0 & 0 \\ 0 & \frac{(\Delta t)^2}{2} & 0 \\ 0 & 0 & \frac{(\Delta t)^2}{2} \\ \Delta t & 0 & 0 \\ 0 & \Delta t & 0 \\ 0 & 0 & \Delta t \end{bmatrix} \quad \text{式 9-16}$$

以上是根据前一状态和已知的控制输入对系统状态建立的模型。除此以外还有一些**未知因素**会影响无人机的状态，比如阵风可以影响无人机的导航。类似这样的未知因素可以用均值为0、协方差为$\boldsymbol{Q_k}$的高斯噪声来表示。式 9-14 和式 9-15 现在就可以写成式 9-18 和式 9-17

$$\boldsymbol{x_k} = \boldsymbol{F_k x_{k-1}} + \boldsymbol{B_k u_k} \quad \text{式 9-17}$$

$$\boldsymbol{P_k} = \boldsymbol{F_k P_{k-1} F_k^T} + \boldsymbol{Q_k} \quad \text{式 9-18}$$

这就是 Kalman 滤波器的预测阶段。

下面我们来看 Kalman 滤波的更新阶段。仍然以无人机为例，无人机装载了 GPS 可以对其位置进行估计，以$\boldsymbol{z_k}$表示，它是一个三维的位置向量，这个位置估计也包含了以协方差为$\boldsymbol{R_k}$的不确定性。当获得这个信息时，将其跟预测阶段得到的当前状态结合就可以计算出更准确的位置估计。但是，GPS 只有位置信息，而状态变量包含位置和速度。观测矩阵$\boldsymbol{H_k}$可将当前状态映射到观测空间（也就是 GPS 的空间）。在我们无人机的例子中，

$$\boldsymbol{H_k} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad \text{式 9-19}$$

这样，$\boldsymbol{H_k x_k}$就只有位置信息，可在观测空间跟$\boldsymbol{z_k}$进行比较。根据之前介绍的恒等式，$\boldsymbol{H_k x_k}$的协方差矩阵为$\boldsymbol{H_k P_k H_k^T}$。

得到了预测值$\boldsymbol{H_k x_k}$和测量值$\boldsymbol{z_k}$，需要融合这两个值以计算出更准确的值。那么要怎样处理？我们先来看一下什么是信息融合。假设有两个不同温度计对温度的估计。第一个温度计显示为 5°C，此温度计的误差在±2°，如图 9-18 中的蓝色曲线所示；第二个温度计显示的温度为 10°C，其误差在±1.5°C，如图 9-18 中的红色曲线所示。

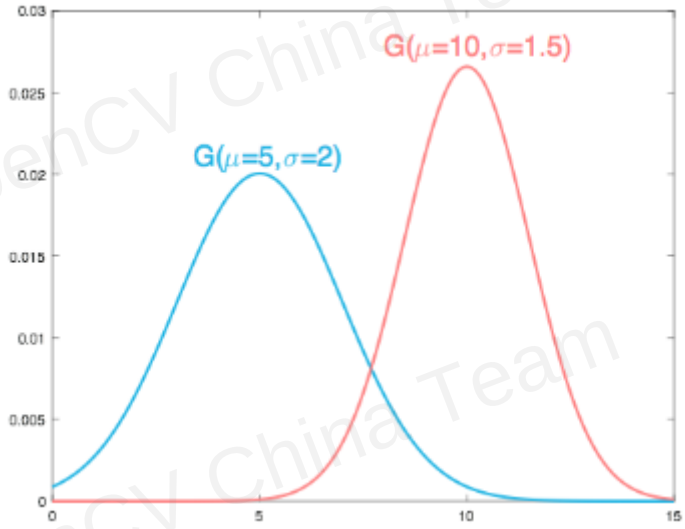


图 9-18

那么哪一个温度计显示的温度是实际温度？我们可能更相信第二个温度计，因为它的误差更小。这时，可以认为实际的温度是一个多高斯模型。将图 9-18 中的两个高斯分布相乘，就得到图 9-19 中的绿色曲线显示的对温度的新的估计。

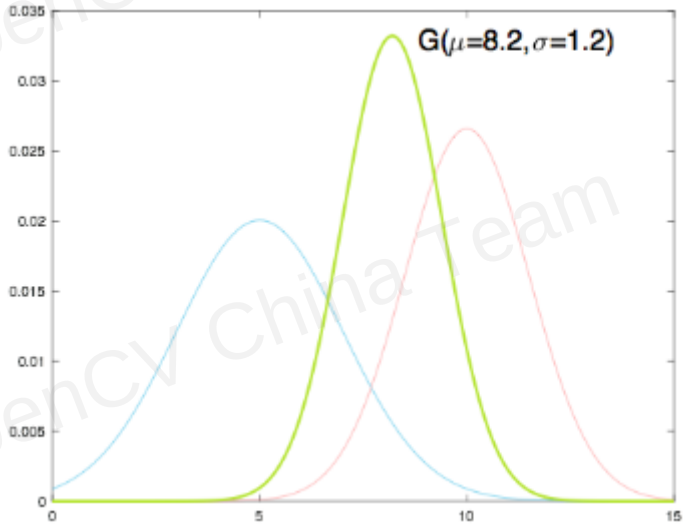


图 9-19

可以看到这个新的估计更靠近红色曲线，这是由于红色曲线代表的温度计比蓝色曲线代笔的温度计更精确。绿色曲线的标准差σ比蓝色和红色都小，说明通过融合后对实际的温度估计更准确。

回到无人机的问题上。我们有对无人机位置的不同值：

1. 来自预测的$\boldsymbol{H_k x_k}$，协方差为$\boldsymbol{H_k P_k H_k^T}$
2. 来自测量的$\boldsymbol{z_k}$，协方差为$\boldsymbol{R_k}$

我们可以将这两个值像上述对温度信息的处理一样进行融合，不同的只是对于无人机的情况是一个多变量的高斯分布。

通过推导可以得出对状态的最优估计，如式 9-20 和式 9-21 所示。

$$\hat{\mathbf{x}}_k = \boldsymbol{x_k} + \boldsymbol{K}(\boldsymbol{z_k} - \boldsymbol{H_k x_k}) \quad \text{式 9-20}$$

$$\widehat{\boldsymbol{P}}_k = \boldsymbol{P_k} - \boldsymbol{K H_k P_k} \quad \text{式 9-21}$$

其中，$\boldsymbol{K} = \boldsymbol{P_k H_k^T}(\boldsymbol{H_k P_k H_k^T} + \boldsymbol{R_k})^{-1}$

综上，Kalman 滤波器是一种用于跟踪系统内部状态的方法。它有两步：

1. **预测**：根据内部动力学和控制输入进行预测，如式 9-17 和式 9-18
2. **更新**：根据独立的测量值对预测值进行更新，如式 9-20 和式 9-21

下面通过一个实例来说明如何使用 Kalman 滤波进行跟踪。OpenCV 中的 KalmanFilter 类实现了 Kalman 滤波器，下面这个例子将使用 Kalman 滤波跟踪一个行人。首先用 HOG 算子检测行人，然后初始化一个 Kalman 滤波器对行人矩形框的左上角(x, y)和矩形框的宽w进行跟踪，这里假设x, y, w仅有速度而没有加速度。这样，状态向量有六个分量即

(x, y, w, v_x, v_y, v_w)，测量值有三个分量即(x, y, w)。对于录制的视频，显然没有控制输入。实例的源代码和详细注释见 kalmanTracker.cpp。

运行程序，可以看到跟踪的结果比检测的结果平滑。这也就是说，跟踪过程的噪声比重复检测的噪声小，Kalman 滤波的滤波就反应在此，平滑了噪声。其次，跟踪的质量依赖于运动中的不确定性。如果目标的运动与我们的运动模型近似，那么跟踪就能产生很好的结果。如果目标突然改变运动方向，预测值就会失败，知道下一次观测更新才能恢复。最后，这个例子中的用 Kalman 滤波进行跟踪，只是用了运动信息，完全没有使用像素的信息。如果加入像素的信息，可以改进跟踪结果，我们下一节将进行介绍。

进一步阅读：

https://medium.com/self-driving-cars/all-about-kalman-filters-8924abe3aa88

https://www.researchgate.net/publication/224138621_Applications_of_Kalman_Filtering_in_Aerospace_1960_to_the_Present_Historical_Perspectives

http://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/

Video Credit (Licensed under "Videvo Attribution License" from videvo.net. Author: Kiril Dobrev).

9.7 MeanShift 和 CamShift（选学）

上一节中介绍了使用 Kalman 滤波器进行目标跟踪。这一节，我们将目标跟踪看成是一个寻找模式的问题，来介绍 MeanShift 方法（均值漂移）跟踪目标。MeanShift 非常直接，在实时跟踪可形变目标时效果很好，它受遮挡、场景混乱和小的尺度变化的影响比较小。

MeanShift 是一种非参数化的方法寻找一组点的模式，也就是说它是寻找密度函数的最大值。它最先由 Fukunaga 和 Hostetler 在 1975 年的文章中发表。在类似图像分割、保留边缘的滤波中也用到了 MeanShift 算法。

假设有如图 9-20 所示的一组点，MeanShift 使用迭代的方法找出密度中心/质心。图中红星代表以它为中心的区域需要计算均值，红色圆圈代表需要计算的区域。绿星代表了红色圆圈区域的质心。可以看到，它与红星代表的圆心是不同的。

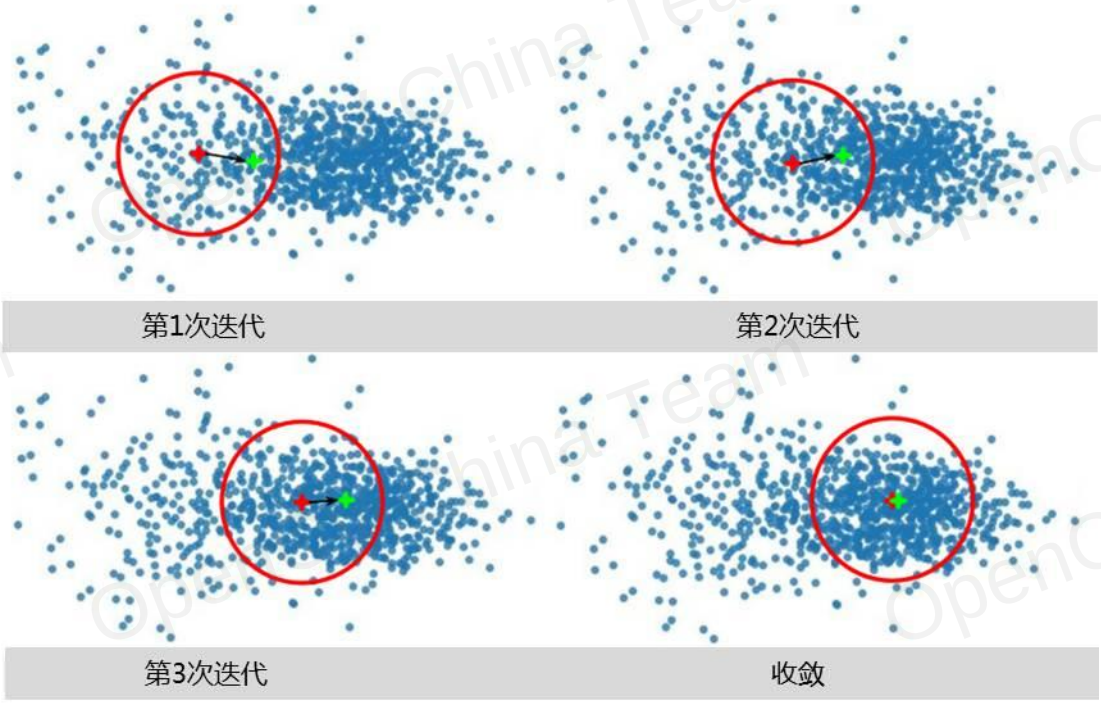


图 9-20

在初始状态，我们可以以任意位置作为质心，然后计算圆圈区域的质心，之后将圆心移到计算出的质心，再重复之前的过程，如图 9-20 所示。黑色箭头即为均值漂移向量，代表了每次迭代均值移动的方向和幅度。红色圆圈的半径，也叫窗口大小，是最关键的参数。如果窗口太小很容易陷入到局部最大值，如果窗口太大也不会找到真实的最大值，如果有两种模式，容易被合并。所以，窗口大小应该是可以自适应的。CAMShift（continuously adaptive meanshift）就是这样一种方法，我们等一下再来介绍。

式 9-22 是上面介绍的过程的数学表达。

$$m_x = \frac{\sum_i K(x-x_i)x_i}{\sum_i K(x-x_i)} \quad \text{式 9-22}$$

其中，x为窗口中心，K为决定窗口大小和点权重的核，x_i为窗口内的其他点，m_x为计算得到的质心。

K有不同的形式，如均匀核：

$$K_U(x) = \begin{cases} 1, if||x|| \le d \\ 0, if||x|| > d \end{cases} \quad \text{式 9-23}$$

在大小为d的窗口内，所有的点对质心的贡献相同。

高斯核:

$$K_G(x) = \begin{cases} e^{-\frac{||x||^2}{d^2}}, if\ ||x|| \le d \\ 0, \quad\quad if\ ||x|| > d \end{cases} \quad \text{式 9-24}$$

或者 epanechnikov 核:

$$K_E(x) = \begin{cases} 1 - \frac{||x||^2}{d^2}, if\ ||x|| \le d \\ 0, \qquad\quad if\ ||x|| > d \end{cases}$$ 式 9-25

这两个核对于靠近x的点赋予了更大的权重，权重随着点与x的距离逐渐变小。

上面介绍了如何用 MeanShift 找到密度函数的最大值，现在来看怎么将其应用到目标跟踪上，步骤如下：

1. 计算目标物体的彩色直方图
2. 对于每一帧图像，计算 likelihood 图像（这与密度函数类似），图像的像素值表示这一点与目标物体颜色分布的相似度。likelihood 图像可以通过直方图反向投影计算得到。
3. 用 MeanShift 计算出 likelihood 图像的累计密度函数的最大值，这个值就给出了目标物体在下一帧图像中的位置。

现在来看什么是直方图反向投影。直方图反向投影是找出两幅图相似度的一种方法，它记录给定图像中的像素点如何适应直方图模型像素分布。设H代表目标物体的直方图，对于新一帧图像中的每个像素，直方图反向投影会找出该像素属于的H的 bin，然后在创建一幅新的图像，此图像上的对应这个像素的像素值为前面找到的 bin 的值。

假如要在图 9-21 的中找到橙色的球。我们已有橙色球的彩色直方图，这个直方图中属于橙色的 bins 的值很大而其它颜色的 bins 的值非常小。那么，在反向投影图中，橙色的像素对应的值将会很大，而其它非橙色像素对应的值很小。如图 9-21 所示。

新的一帧图像　　反向投影图

图 9-21

OpenCV 中使用 MeanShift 进行目标跟踪的流程如下（源代码见 meanShift.cpp）：

1. 计算人脸区域的直方图
   首先使用 Dlib 的人脸检测器检测人脸，然后用 OpenCV 的 calcHist()函数计算人脸区域在 HSV 空间 H 分量的直方图。calcHist()还将直方图的值归一化到[0, 255]。需要注意的是，颜色信息对光照变化是十分敏感的。
2. 计算反向投影图
   对于每一帧新的图像，将其转换到 HSV 空间，然后用 OpenCV 的 calcBackProject()计算反向投影图BP。
3. 应用 MeanShift
   调用 OpenCV 的 meanShift()函数，在反向投影图中的上一个目标位置区域找出最大值，这个最大值的位置即目标在这一帧新图像中的位置。这个迭代过程的前几次迭

代显示如图 9-22。

图 9-22

下面的图 9-23 显示了其中的一些结果。

图 9-23

第一行图像，目标对于摄像机只有横向的运动，在图像中的大小并没有发生变化，跟踪结果是正确的。第二行图像目标的尺寸发生了变化，我们看到通过 MeanShift 跟踪的矩形框并没有随之发生相应的变化。CAMShift 就很好的解决了这个问题。

CAMShift 是 Continuously Adaptive Meanshift 的缩写，它在跟踪过程中会自动改变窗口大小。它是由 Gary Bradski 设计并于 1998 年发表。CAMShift 的工作流程如下：

1. 设定初始搜索窗口的宽w、高h以及中心(x_c, y_c)
2. 用 MeanShift 计算出新的中心位置(x_c, y_c)并保存式 9-26 计算的 0 阶矩：
   $M_{00} = \sum_{x,y} BP(x,y)$ 式 9-26
   它是反向投影图窗口内所有像素的和。
3. 将窗口中心移至 2 中计算的新中心位置，按式 9-27 改变窗口的宽高
   $w = 2\sqrt{\frac{M_{00}}{256}},\ h = 1.2w$ 式 9-27
4. 重复 2 和 3 直至收敛。

目标窗口的方向可由二阶矩计算得到，二阶矩可用于计算头的转动、脸的长宽等。如何从反向投影图的二阶矩计算方向可以在这篇文章中找到。

OpenCV 中用 CAMShift 进行目标跟踪的第 1 步和第 2 步与 MeanShift 相同。第 3 步，在计算得到反向投影图后，调用 OpenCV 中 CamShift()函数来跟踪目标。CamShift()函数使用 meanshift 来找到目标中心，并调整窗口大小。另外，它还找出了目标的最优旋转方向。函数返回目标的位置、大小和方向。图 9-24 的蓝色矩形框是目标，绿色矩形框就是 CAMShift 跟踪目标的结果。

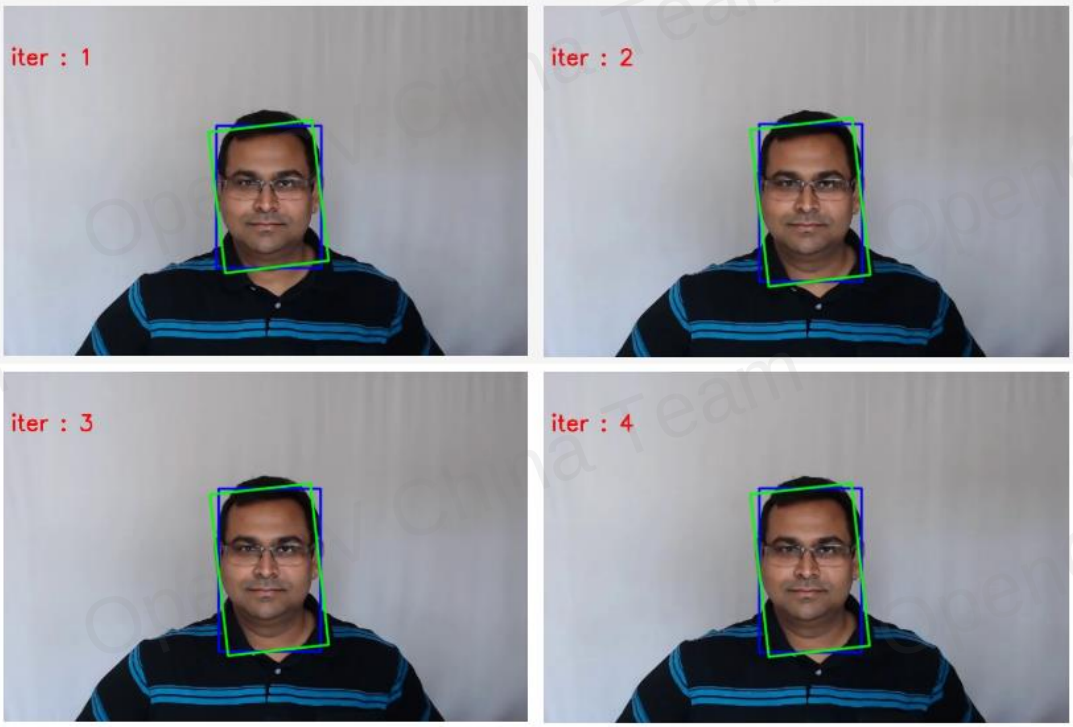


图 9-24

图 9-25 显示了之前 MeanShift 失败的场景，CAMShift 很好的跟踪到了。

图 9-25

CAMShift 进行目标跟踪的演示源代码见 camShift.cpp。