目标追踪实验报告

沈书杨

东南大学 吴健雄学院,江苏 南京 61518218

摘 要: 目标追踪就是利用连续的运动图像去识别，观察并记录感兴趣的运动目标，近年来已成为计算机视觉领域研究的热点。本文将光流法应用在运动目标追踪中，对目标追踪算法的实现过程进行了具体研究，并针对光流法在目标追踪中的一些缺陷进行了改进。首先,研究了经典的 LK 光流法。其中 LK 光流计算方法因为灵活性高，计算量相对较小，适合应用在目标追踪中。对于帧间运动位移较大的光流计算,将图像进行金字塔分解来提高光流矢量求解的精确度。整个过程主要包括图像预处理、角点检测、光流计算、光流聚类、区域分析以及标定出目标位置，得出每帧的追踪结果。最后，本文引入定位精度较高的 SIFT 特征点匹配追踪算法，将该算法得到的目标位置与光流法融合进行了实验性探究。

关键词: 目标追踪;光流法;角点检测

目标追踪是使用摄像头随时间定位正在移动的物体的过程，是计算机视觉领域的一个重要问题。目前广泛应用在人机交互、安全和监控、视频通信和压缩、增强现实、交通控制、医学成像、视频编辑等领域。由于视频中通常包含大量数据，目标追踪一般是一个耗时的过程。目标识别和目标分割在目标追踪中的进一步应用也会在优化结果的同时增加任务的复杂性和计算量。

目标追踪的目标是将连续视频帧中的目标对象关联起来。当目标对象相对于帧速率快速移动时，这种关联可能会特别困难。有时，目标对象还会随着时间的推移改变运动方向。对于这些情况，目标追踪系统通常会采用运动模型，该模型描述了目标的图像如何随着对象的不同可能运动而变化。光流法既是其中的一种运动模型。

光流或光流是由观察者和场景之间的相对运动引起的视觉场景中物体、表面和边缘的明显运动模式。光流也可以定义为图像中亮度模式的表观运动速度的分布。光流的概念是由美国心理学家 James J. Gibson 在十九世纪四十年代引入的，用于描述为在世界中移动的动物提供的视觉刺激。吉布森强调了光流对于可供性感知的重要性，即辨别环境中行动可能性的能力。Gibson 的后继者和他的心理学方法进一步证明了光流刺激对世界观察者运动感知的作用，感知世界上物体的形状、距离、运动和运动控制。

光流是图像亮度的运动信息描述。光流法计算最初是由 Horn 和 Schunck 于 1981 年提出的，这种方法创造性地将二维速度场与灰度相联系，引入光流约束方程，得到光流计算的基本算法。光流计算基于物体移动的光学特性提出了两个假设：1. 运动物体的亮度在很短的间隔时间内保持不变 2. 给定邻域内的速度向量场变化是缓慢的。

在二维图像中，考虑点(x,y)在第 t 时间的像素亮度为 I(x,y,t)，假设这个点将要在 Δt 时间后运动到(x+Δx,y+Δy)处。根据上述的光亮度一致性假设，我们可以使用下列等式将两个图像间的点联系起来：

$$I(x,y,t) = I(x+\Delta x, y+\Delta y, t+\ \Delta t)$$

可以根据泰勒公式得出：

$$I(x+\Delta x, y+\Delta y, t+\Delta t) = I(x,y,t) + I_x\Delta x + I_y\Delta y + I_t$$

因此可以推出：

$$I_x\Delta x + I_y\Delta y + I_t\Delta t = 0$$

对等式两边同时除以 Δt 进行求导，得到（p 指目标点的位置）：

$$I_x(p)v_x + I_y(p)v_y + I_t(p) = 0$$

在这一个方程中有两个未知数，无法求解。为了找到光流，需要由附加的约束给出另一组方程进行结果的求解。Lucas-Kanade 方法通过添加的假设是空间相干性，即两帧之间的图像内容在以目标点 p 为中心的一个较小邻域内近似恒定。再附加条件的加持之下，根据上述得到的公式得出附加等式 $Ax = b$，其中：

$$A = \begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \cdots & \cdots \\ I_y(p_n) & I_y(p_n) \end{bmatrix} \qquad v = \begin{bmatrix} v_x \\ v_y \end{bmatrix} \qquad b = \begin{bmatrix} -I_t(p_1) \\ -I_t(p_2) \\ \cdots \\ -I_t(p_n) \end{bmatrix}$$

现在，我们可以通过解方程 $A^TAv = A^Tb$ 来求得光流向量 v。

本文重点对基于 LK 光流法的目标追踪进行分析与实现。分模块对基于 LK 光流法的目标追踪进行了代码的编写，同时也使用了图像金字塔优化了 LK 光流法；引入非极大值抑制方法对实际情况下的目标物体进行了目标追踪的实验，并在最后使用了 SIFT 的特征检测方法进行尝试。

1 实现步骤

本节介绍了 Lucas-Kanade 光流法的实现步骤。

1.1 特征点检测

特征点检测广泛应用到目标匹配、目标追踪、三维重建等应用中，在进行目标建模时会对图像进行目标特征的提取，常用的有颜色、角点、特征点、轮廓、纹理等特征。现在开始讲解常用的特征点检测，其中 Harris 角点检测是特征点检测的基础，提出了应用邻近像素点灰度差值概念，从而进行判断是否为角点、边缘、平滑区域。Harris 角点检测原理是利用移动的窗口在图像中计算灰度变化值，其中关键流程包括转化为灰度图像、计算差分图像、高斯平滑、计算局部极值、确认角点。在特征点检测中经常提出尺度不变、旋转不变、抗噪声影响等，这些是判断特征点是否稳定的指标。

Harris 角点检测首先对二维图像在 x 方向和 y 方向上求导，得到 I_x，I_y，然后使用I_x，I_y 计算得到二维图像每一个像素点处的 I_x^2，I_y^2，I_{xy}。然后计算每一个像素点处的矩阵 M

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_{xy} \\ I_{xy} & I_y^2 \end{bmatrix}$$

并由此获得每一个像素点处的角点相应 $R = Det(M) - k(Trace(M)^2)$，输出最后的角点响应矩阵 $R(x,y)$。

在本次实验中，我们直接使用了 python 库 skimage 中的封装函数进行角点提取，结果如下：

1.2 经典光流法

经典光流法的原理已在引言中进行简要介绍，以下为本次实验中需要实现的代码：

```
y_i, x_i = max(y - w, 0), max(x - w, 0)
y_j, x_j = y + w + 1, x + w + 1
A = np.hstack((Ix[y_i:y_j, x_i:x_j].reshape(-1, 1), Iy[y_i:y_j, x_i:x_j].reshape(-1, 1)))
b = It[y_i:y_j, x_i:x_j].reshape(-1, 1)
flow_vectors.append((np.linalg.inv(A.T@A)@A.T@-b).flatten())
```

其中，x_i，y_i，x_j 和 y_j 分别标定应用空间相干性假设的 3*3 区域。从已经获得的图像 x，y 方向的导数中获得这个区域的对应数据并将其转化为如下形式。

$$A = \begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \cdots & \cdots \\ I_y(p_n) & I_y(p_n) \end{bmatrix} \qquad b = \begin{bmatrix} -I_t(p_1) \\ -I_t(p_2) \\ \cdots \\ -I_t(p_n) \end{bmatrix}$$

对等式 $Ax = b$ 应用最小二乘法，获得光流向量。我们可以通过将光流向量与特征点相加来预测下一帧中对应特征点的位置。

1.3 特征追踪

使用 Lucas-Kanade 方法来追踪跨多个帧的关键点：计算第 i 帧中关键点处的光流向量，并将光流向量添加到点集以追踪第 i+1 帧中的点。但是在实际实验过程中，可以注意到，有些点只是逐渐消失，而没有很

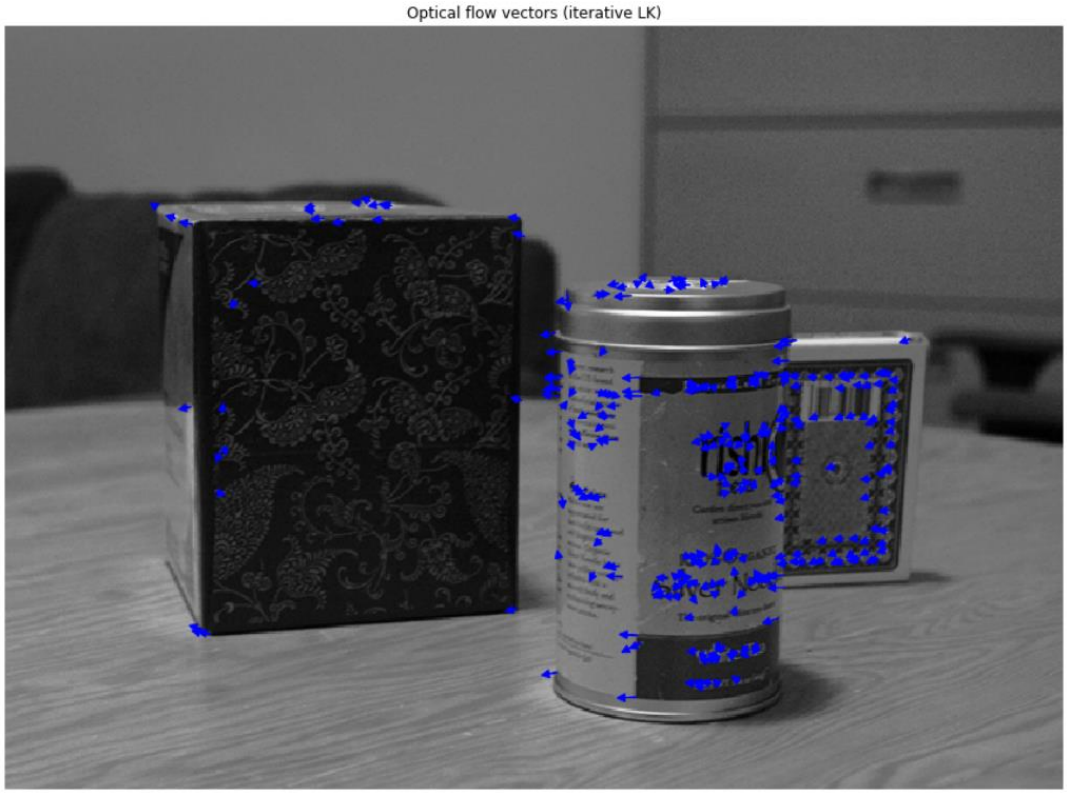
好地被追踪。我们不想保留这些不能很好描述物体特征的特征点，而是想以某种方式声明这些点丢失追踪。方法是比较两个后续帧中目标点周围的区域。如果一个点周围的区域与下一帧中对应点周围的区域不相似，那么我们认为该特征点丢失。在本次实验中，我们将使用两个归一化区域之间的均方误差作为丢失特征点的标准。

在实验中，我们使用的均方误差阈值为 1.5，对于 0-255 的灰度来说，这是一个相当小的阈值，相对来说十分严格。在筛选后，追踪的特征点如下图所示：

1.4 金字塔光流法

从经典光流法的计算中可以看出，当两帧图像之间的物体移动速度过快时，光流特性的假设会失效，算法会出现较大的误差。此时，我们希望减少两帧图像之间的物体的运动。由此，提出金字塔光流法，将图像进行逐层分解，逐层降采样，使得两帧图像之间的光流的变化变小。利用金字塔的结构，自上而下修正光流向量，优化结果（最上层的图像分辨率最小）。

对于应用了金字塔光流法的实验，可以得到如下结果：



与下面的经典光流法求得的光流向量比较，金字塔光流法得到的光流向量对于目标特征的运动方向的描述更加准确和清晰，达到了更好的效果。

在金字塔光流法中，如果我们缩小图像的分辨率，目标特征在大尺度上的的运动将变得更容易追踪。 另一方面，随着我们丢失图像中的细节，较小尺度上的运动将变得更难追踪。运用类似的思想为了解决这个问题，我们可以用多尺度表示图像，并从大尺度到小尺度计算光流向量。

如此，在精确的光流向量引导下，我们可以在相同的阈值内追踪更多的特征点：

1.5 人脸追踪

使用已实现的 Lucas-Kanade 光流法构建一个简单的目标追踪器。 以人脸图像数据集进行实验，人脸图像序列中的每一帧都用标有人脸的正确标签框。目标追踪器在第一帧得到标注的人脸框，通过预测后续帧中的边界框来追踪目标。

为了追踪人脸，首先在第一帧的人脸框内找到要追踪的特征点。在接下来的每一帧中追踪这些点，并在追踪点周围输出一个紧密的边界框。 为了防止所有特征点丢失，无法进行追踪，每 20 帧检测一次边界框内的特征点并加入特征点集。

由于数据集中的每一帧图像都有标注的人脸边框，所以可以使用 intersection over union (IoU)方法对实验结果进行评估：设目标追踪器预测的人脸边框为 A，图像标注的人脸边框为 B，则评估标准为，

$$\frac{A \cap B}{A \cup B}$$

在本次实验中，根据之前的实现步骤，我的平均 IoU 预测准确率为 0.3657。

1.6 调参

使用相同的方法对超参数进行调节一达到更高的准确率，调节的超参数有：1. 检测新特征点的间隔帧数

n_frame 2. 特征点追踪的阈值 error_threshold。

本次的数据集共有 134 帧，n_frame 过大可能会丢失所有的特征点，导致追踪中止；而过小可能会加入一些无效的特征点，干扰目标追踪且会大大增加计算量。error_threshold 的特点和 n_frame 类似。由此进行调参。

在实际调参中，发现 n_frame 大小对结果的影响不明显，error_threshold 对结果影响较大。

当 n_frame＝20，error_threshold＝2 时，平均 IoU 准确率为 0.1369。保持 n_frame 不变，当 error_threshold＝3 时，结果为 0.3657；而当 error_threshold＝5 时，结果为 0.3998；而当 error_threshold＝10 时，结果依然为 0.3998。可以看出，当 error_threshold 大于 5 时，阈值已经失去了筛选的作用。

在上述基础上进行调参，当检测新特征点的间隔帧数为 19，特征点追踪的阈值在 3.2 上下时，平均 IoU 准确率达到 0.5030。

2 优化探索

SIFT 特征是一种对旋转、尺度缩放、亮度变化等保持不变性的非常稳定的局部特征。一般认为，描述子采用 4×4×8＝128 维向量表征，综合效果最优。这种邻域方向性信息联合增强了算法的抗噪声能力，同时对于含有定位误差的特征匹配也提供了比较理性的容错性。Opencv 库提供了完备的 SIFT 算子调用接口，本次实验中使用 opencv 来进行目标追踪器的探索。

2.1 基于SIFT的特征提取

建立 SIFT 提取图片特征的对象以简化代码。具体如下：

```
class SiftImg:
    def __init__(self,img):
        self.img= cv2.normalize(img, None, 0, 255, cv2.NORM_MINMAX).astype('uint8')

    def getFeatures(self,sift):
        self.kp, self.des = sift.detectAndCompute(self.img, None)
```

其中，函数 getFeature 接受自定义的 SIFT 描述子，以保证程序的自由性，并对图像进行特征提取，将关键点以及特征表示分别保存在 self.kp 以及 self.des 中。

在特征匹配步骤中，先分析代码。(以 BFMatcher 为例，实际实验中为了运算速度使用了 Flann)

```
def getGoodMatch(des1, des2, ratio=0.6):
    bf = cv2.BFMatcher()
    bf_matches = bf.knnMatch(des1, des2, k=2)

    matches = sorted(bf_matches, key=lambda x: x[0].distance / x[1].distance)
    good = []
    for m, n in matches:
        if m.distance < ratio * n.distance:
            good.append(m)
        else:
            break
    if not good:
        raise Exception('No good match under ratio %.2f!' % ratio)
    return good
```

首先定义一个 BFMatcher 的对象。为什么这里采用 knnMatch 而非 match 呢。是因为在 opencv 中 match 函数只能返回在对应算法下匹配度最高的点对。而 knnMatch 可以返回对应算法下匹配度 topk 的点对，这样的好处是可以引入一个人为选择的阈值对关键点对进行二次筛选。对于一个待查找物体图上的一个关键点，若在目标图中有两个关键点与他的距离都是相近的，那么不论这个距离又多近，这个关键点对都不能称之为有效的关键点对。因为这个关键点对并不能说是“独一无二”的。

接着使用 sort 算法以“阈值”为标准进行排序。在排序后，列表靠前的点对都是小于阈值的可靠点对。

这里为了提高算法效率，引入了 break 语句。由于 sort 函数已经从小到大排列，所以只要遇到第一个大于阈值的关键点列表，就可以直接跳出。如果没有一组数据是满足阈值条件的，就转入异常处理并提示。

2.2 假设

SIFT 算子一般用于目标识别，但是本实验要求的是目标追踪，所以在本实验中，对 SIFT 特征提取做了一些改进。由于数据集中仅给到第一帧的标注框，所以做一个假设：两帧之间的同一目标移动的距离很短，以至于目标的特征所在的范围重合度很高。在这个假设下，可以在第二帧图像中以第一帧图像的标注框为界限进行 SIFT 图像提取，并通过特征匹配得到优秀的特征点计入点集。具体代码如下：

```
sift = cv2.xfeatures2d.SIFT_create()
first=SiftImg(roi)
first.getFeatures(sift)
second=SiftImg(frames[1][y:y + h, x:x + w])
second.getFeatures(sift)
good=getGoodMatch(first.des,second.des)
kp=[]
for match in good:
    kp.append(first.kp[match.queryIdx].pt)
keypoints=np.array(kp)
```

添加新特征点同理，会在代码中给出。

2.3 结果

在原始条件：阈值为 3，添加特征点间隔为 20 的情况下准确率为 0.3471。虽然比 harris 角点特征方法略差，但是也较好，在调参和优化 SIFT 算子后可能会取得更好的效果

```
FLANN time comsuption: 0.0010, # of pairs: 20.
FLANN time comsuption: 0.0000, # of pairs: 4.
FLANN time comsuption: 0.0000, # of pairs: 9.
FLANN time comsuption: 0.0000, # of pairs: 12.
FLANN time comsuption: 0.0000, # of pairs: 9.
FLANN time comsuption: 0.0010, # of pairs: 18.
FLANN time comsuption: 0.0010, # of pairs: 37.
0.3471363279405585
```

References:

[1] Wikipedia. https://en.wikipedia.org/wiki/Harris_Corner_Detector

[2] Wikipedia. https://en.wikipedia.org/wiki/Optical_flow.

附中文参考文献:

[1] mini 猿要成长 QAQ. 总结：光流--LK 光流--基于金字塔分层的 LK 光流--中值流.

https://blog.csdn.net/sgfmby1994/article/details/68489944