计算机视觉作业4: 目标跟踪

唐麒 21120299
qitang@bjtu.edu.cn

摘要—目标跟踪是利用一个视频或图像序列的上下文信息，对目标的外观和运动信息进行建模，从而对目标运动状态进行预测并标定目标位置的一种技术。本文分别在不同的数据集上实现了基于 Mean Shift (均值漂移) 目标跟踪和基于分类思想的目标跟踪，两种不同目标跟踪方法的优劣将在实验部分进行分析。

关键词—计算机视觉、目标跟踪、Mean Shift、AdaBoost、SVM

1 简介

目标跟踪是计算机视觉领域的重要研究方向之一，其目标是在连续的图像中对感兴趣物体进行检测、提取、识别和跟踪，从而获得目标物体的相关参数，如位置、速度、尺度、轨迹等，并对其进一步处理和分析，实现对目标物体的行为理解，或完成更高一级的任务。

目标跟踪技术有一些典型的应用场景，包括 a) 安全监控：车站、机场、银行及超市等公共场所的实时监控；b) 交通检测：对行人及车辆行为进行判定，完善智能交通系统；c) 军事领域：导弹制导、武器观测瞄准、敌方目标定位及跟踪；d) 医学应用：标记、增强及跟踪生物特征来帮助医生诊断疾病；等。

虽然目标跟踪技术在很多领域都有着广泛的应用前景，但由于实际环境复杂多变，目标跟踪技术也面临着诸多挑战，如光照、尺寸的变化，目标的快速运动，形变，相机抖动，遮挡等复杂的因素。而一个好的目标跟踪算法，应该是要又快又准。“快”主要表现在计算量小和所需的存储空间小，“准”就是在复杂因素的干扰下，预测出的目标位置（bonding box）要尽可能地接近真实值。按照模式划分，目标跟踪的方法可以分为 2 类，即生成式模型和鉴别式模型。

早期的工作主要集中于生成式模型跟踪算法的研究, 如光流法、粒子滤波、Meanshift 算法、Camshift 算法等。这类方法首先建立目标模型或者提取目标特征, 在后续帧中进行相似特征搜索，逐步迭代实现目标定位。但是这类方法也存在明显的缺点, 就是图像的背景信息没有得到全面的利用，且目标本身的外观变化有随机性和多样性的特点, 因而通过单一的数学模型描述待跟踪目标具有很大的局限性。例如在光照变化, 运动模糊, 分辨率低, 目标旋转形变等情况下, 模型的建立会受到巨大的影响, 从而影响跟踪的准确性。

鉴别式模型是指将目标模型和背景信息同时考虑在内, 通过对比目标模型和背景信息的差异, 将目标模型提取出来, 从而得到当前帧中的目标位置。一些研究对跟踪算法的评估中发现, 通过将背景信息引入跟踪模型, 可以很好地实现目标跟踪。因此研究人员逐渐尝试使用经典的机器学习方法训练分类器, 如 MIL、TLD、支持向量机等。近年来, 随着深度学习技术的广泛应用, 人们开始将深度学习技术用于目标跟踪，并取得了更加显著的性能。

(a) (b)

图 1: 目标跟踪示例：(a) 初始化第 1 帧目标状态, (b) 预测第 N 帧目标状态.

本文分别在两个视频序列上实现了一种生成式模型跟踪算法和一种鉴别式模型跟踪算法，即基于 Mean Shift 的目标跟踪基于分类思想的目标跟踪。本文剩余部分将对两种目标跟踪方法的原理、实现及实验进行介绍，所有实验涉及的方法、函数实现均基于 Python 语言。

2 方法

这一章节将对基于 Mean Shift 和基于分类思想的两种目标跟踪方法进行介绍，其结果将在实验部分给出。

2.1 目标跟踪的基本框架

跟踪算法通常同时依赖外观建模和运动信息建模。如图 1 所示，通过利用初始帧的目标外观信息进行建模后，跟踪器具有了目标和背景的辨别能力。在后续帧中，跟踪算法首先通过运动模型，如粒子滤波，粗略地估计目标位置并得到一系列的候选样本，进一步结合外观模型进行目标的精准定位。

准确的定位用于更新运动模型和外观模型。因此，两种目标跟踪方法的初始目标位置标定都依赖于手工标定，后续帧中的目标位置则分别通过均值漂移方法和在线分类器得到。

2.2 基于均值漂移的目标跟踪

均值漂移 (Mean Shift) 的基本思想是利用概率密度的梯度爬升来寻找局部最优，即 Mean Shift 向量逐步漂移到局部密度最大点并停止，从而达到跟踪目的。

2.2.1 Mean Shift

给定 d 维空间中的 n 个样本点 $x_i(i=1,\ldots,n)$，在 x 点的 Mean Shift 向量的基本形式定义为:

$$M_h(x)=\left(\frac{1}{k}\sum_{x_i\in S_h}x_i\right)-x=\frac{1}{k}\sum_{x_i\in S_h}(x_i-x) \tag{1}$$

其中, S_h 是一个半径为 h 的高维球区域, k 表示 n 个样本点中有 k 个点落入区域 S_h 中。直观地, Mean Shift 向量表示区域中 k 个样本点相对于点 x 求偏移向量再平均。该向量指向概率密度梯度的方向。

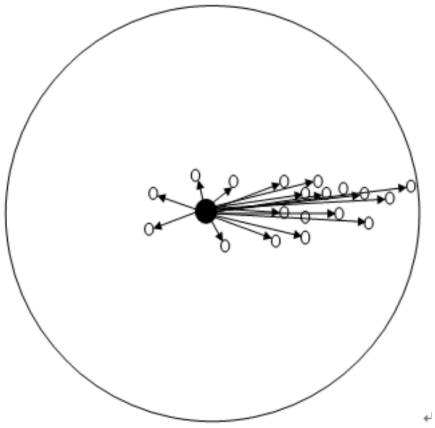


图 2: Mean Shift 示意图

如图 2 所示，大圆所圈定的范围即为 S_h，小圆代表落入 S_h 区域内的样本点 $x_i\in S_h$，中心的黑色实心圆即为 Mean Shift 的基准点 x，箭头表示样本点相对于基准点 x 的偏移向量。可以看出，平均的偏移向量 $M_h(x)$ 会指向样本分量最多的区域，也就是概率密度函数的梯度方向。

$$M_h(x)=\frac{1}{k}\sum_{x_i\in S_h}(x_i-x) \tag{2}$$

2.2.2 核函数

为了解决所有样本点对均值平移向量的贡献相同的问题，在 Mean Shift 的基础上引入两个参数，即核函数和权重。核函数的定义为 $x\in R^D,\|x\|^2=x^Tx$，若函数 $K()$ 存在一个剖面函数: $k:[0,\infty)\to R$, 即 $K(x)=k\left(\|x\|^2\right)$，并且 $k()$ 满足:1) 非负的;2) 非增的;3) 分段连续的，且 $\int_0^\infty k(r)dr<\infty$，那么函数 $K()$ 就被称为核函数。常用的核函数包括 Epanechikov 核函数、均匀核函数和高斯核函数。本文采用的是 Epanechikov 核函数，即计算空间任意一点到中心位置的欧氏距离。其表达式如下:

$$K_E(\mathrm{x})=\begin{cases}c\left(1-\|\mathrm{x}\|^2\right) & \|\mathrm{x}\|\le 1\\ 0 & \text{otherwise}\end{cases} \tag{3}$$

在跟踪问题中，核函数起到了位置加权的作用，即更加相信中心位置像素点的信息。

2.2.3 均值漂移目标跟踪算法

基于均值漂移的目标跟踪算法通过分别计算目标区域和候选区域内像素的特征值概率得到关于目标模型和候选模型的描述，然后利用相似函数度量初始帧目标模型和当前帧的候选模版的相似性，选择使相似函数最大的候选模型并得到关于目标模型的 Mean Shift 向量，这个向量正是目标由初始位置向正确位置移动的向量。由于均值漂移算法的快速收敛性，通过不断迭代计算 Mean Shift 向量，算法最终将收敛到目标的真实位置，达到跟踪的目的。流程如下:

（1）目标核函数直方图:

核函数, 中心像素权重更大

$$\hat{q}_u=C\sum_{i=1}^{n}k\left(\|x_i\|^2\right)\delta\left[b\left(x_i\right)-u\right]. \tag{4}$$

统计直方图

其中，$C=1/\sum_{i=1}^{n}k\left(\|x_i\|^2\right)$ 为归一化常数，k 表示核函数 (在核估计中通常是平滑作用), 目标区域共 n 个像素点 $\{x_i\}_{i=1,\ldots,r}$，该区域颜色分布离散成 m 级,$b\left(x_i\right)$ 表示像素点 x_i 的量化值。

（2）在当前帧，计算候选 (待跟踪目标) 核函数直方图:

$$\hat{p}_u(y)=C_h\sum_{i=1}^{n_h}k\left(\left\|\frac{y-x_i}{h}\right\|^2\right)\delta\left[b\left(x_i\right)-u\right] \tag{5}$$

其中，$C_h=1/\sum_{i=1}^{n}k\left(\left\|\frac{y-x_i}{h}\right\|^2\right)$，候选目标区域 $\{x_i\}_{i=1,\ldots,n_h}$，该区域的中心位置为 y，h 表示核函数 k 的窗宽。其余变量物理意义同上。

（3）计算候选目标与初始目标的相似度:

$$\hat{\rho}(y)=\sum_{u=1}^{m}\sqrt{\hat{p}_u(y)\hat{q}_u} \tag{6}$$

进行 Taylor 展开，将 $\hat{p}_u(y)$ 代入并化简得到:

$$\hat{\rho}(y) \approx \frac{1}{2}\sum_{u=1}^{m}\sqrt{\hat{p}_u\left(y_0\right)\hat{q}_u} + \frac{C_h}{2}\sum_{i=1}^{n_h}\mathrm{w}_i k\left(\left\|\frac{y-x_i}{h}\right\|^2\right) \quad (7)$$

其中，$\mathrm{w}_i = \sum_{u=1}^{m}\sqrt{\frac{\hat{q}_u}{\hat{p}_u(\hat{y}_0)}}\delta\left[b\left(x_i\right)-u\right]$，$y_0$ 是 Mean Shift 迭代的起点位置，在跟踪中通常是目标在上一帧的位置。由于第一项为常量，因此我们最大化 $\hat{\rho}(y)$ ，本质上寻找新的质心 y 使得候选区域和模板的相似程度最大化。

（4）计算权值 $\{W\}_{i=1,2,\ldots,m}$

（5）利用 Mean Shift 方法求解目标的新位置 y_1:

$$y_1 = \frac{\sum_{i=1}^{n_h} x_i w_i g\left(\left\|\frac{y_0-x_i}{h}\right\|^2\right)}{\sum_{i=1}^{n_h} w_i g\left(\left\|\frac{y_0-x_i}{h}\right\|^2\right)} \quad (8)$$

代码实现如下:

```
def main():
    global image
    image = cv2.imread('data/Car_Data/car001.bmp')
    # 框选目标并返回相应信息
    location, patch = calibrate_location(image)
    (height, width, c) = patch.shape
    center = [height / 2, width / 2]
    # 计算目标图像的权值矩阵
    weight = np.zeros((height, width))
    for i in range(height):
        for j in range(width):
            distance = (i - center[0]) ** 2 + (j -
                center[1]) ** 2
            weight[i, j] = 1 - distance / (center
                [0] ** 2 + center[1] ** 2)
    # 计算目标权值直方图
    C = 1 / sum(sum(weight))
    hist1 = np.zeros(16 ** 3)
    for i in range(height):
        for j in range(width):
            q_b = math.floor(float(patch[i, j, 0])
                / 16)
            q_g = math.floor(float(patch[i, j, 1])
                / 16)
            q_r = math.floor(float(patch[i, j, 2])
                / 16)
            q_temp1 = q_r * 256 + q_g * 16 + q_b
            hist1[int(q_temp1)] = hist1[int(
                q_temp1)] + weight[i, j]
    hist1 = hist1 * C
    # 读取视频并进行目标跟踪
    for pic_i in range(2, 101):
        if pic_i < 10:
            pic_name = "car00%d.bmp" % pic_i
        elif 100 > pic_i >= 10:
            pic_name = "car0%d.bmp" % pic_i
        else:
            pic_name = "car%d.bmp" % pic_i
        path = os.path.join(r"data/Car_Data/",
            pic_name)
        frame = cv2.imread(path)
        num = 0
        offset = [1, 1]
        # mean shift 迭代
        while (np.sqrt(offset[0] ** 2 + offset[1]
            ** 2) > 0.5) & (num < 20):
            num = num + 1
            # 计算候选区域直方图
            patch2 = frame[int(location[1]):int(
                location[1] + location[3]),
                    int(location[0]):int(location
                        [0] + location[2])]
            hist2 = np.zeros(16 ** 3)
            q_temp2 = np.zeros((height, width))
            for i in range(height):
                for j in range(width):
                    q_b = math.floor(float(patch2[
                        i, j, 0]) / 16)
                    q_g = math.floor(float(patch2[
                        i, j, 1]) / 16)
                    q_r = math.floor(float(patch2[
                        i, j, 2]) / 16)
                    q_temp2[i, j] = q_r * 256 +
                        q_g * 16 + q_b
                    hist2[int(q_temp2[i, j])] =
                        hist2[int(q_temp2[i, j])]
                        + weight[i, j]
            hist2 = hist2 * C
            w = np.zeros(16 ** 3)
            for i in range(16 ** 3):
                if hist2[i] != 0:
                    w[i] = math.sqrt(hist1[i] /
                        hist2[i])
                else:
                    w[i] = 0
            sum_w = 0
            sum_xw = [0, 0]
            for i in range(height):
                for j in range(width):
                    sum_w = sum_w + w[int(q_temp2[
                        i, j])]
                    sum_xw = sum_xw + w[int(
                        q_temp2[i, j])] * np.array
                        ([i - center[0], j -
                        center[1]])
            offset = sum_xw / sum_w
            # 位置更新
            location[0] = location[0] + offset[1]
            location[1] = location[1] + offset[0]
        x = int(location[0])
        y = int(location[1])
        width = int(location[2])
        height = int(location[3])
        pt1 = (x, y)
        pt2 = (x + width, y + height)
```

main.py

2.3 基于分类思想的目标跟踪

如前文所述，基于分类思想的目标跟踪将目标和背景信息同时考虑在内，其基本原理是把跟踪看作分类问题，通过训练分类器来区分背景和目标。具体来讲就是是把目标跟踪看作二分类问题，在线训练作为整体的多个弱分类器用来区分目标和背景。使用 AdaBoost 把作为整体的多个弱分类器合并为一个强的分类器，该分类器用于下一帧的分类，区分像素属于目标还是背景，并得出置信图，并在置信图上利用 Mean Shift 算法找出峰值点也就是目标的位置。在跟踪过程中通过在线训练新的弱分类器并加入到分类器集合里从而在连贯时间上保持更新弱分类器这一整体。

首先需要在第一帧将目标的位置进行手工标注，对于弱分类器而言，所有像素都是潜在的训练数据。为了减少训练时间，本文根据视频训练中目标的大小，将标注目标及其周围的一部分像素（背景）作为训练数据。每个像素都使用一个 11 维的特征向量来描述（由 8 维的 5×5 邻域局部方向直方图和 3 维的像素颜色组成），来自目标和背景的像素对应的标签分别为 1 和 -1，即

$$y_i = \begin{cases} +1 & \text{inside } (r_j) \\ -1 & \text{otherwise} \end{cases} \tag{9}$$

其中，r_j 为当前帧的目标位置。

本文使用支持向量机作为弱分类器。对于每一个弱分类器，可以计算其误差率，计算方法为:

$$err = \sum_{i=1}^{N} w_i \left| h_t\left(x_i\right) - y_i \right| \tag{10}$$

其中，w_i 是像素的权重，为像素被选择训练的概率。$h_t\left(x_i\right)$ 是像素 x_i 的类别，而 y_i 为其“正确”的类别标签。在初始状态下，所有的像素权重都是相等的，并随着训练过程而不断更新，如果像素被错误分类，则像素权重会随之增加。也就是说，在随后的训练集中，难以分类的像素更有可能被选择。权重的更新公式为:

$$w_i = w_i \exp\left\{\alpha_t \left| h_t\left(x_i\right) - y_i \right|\right\} \tag{11}$$

集成学习技术把若干弱分类器组合成一个强的分类器。如本文使用的 AdaBoost 每次在更难的样本上训练一个弱分类器加入到强分类器使得最终的分类器比任何一个弱分类器都好。在跟踪过程中一直更新弱分类器集合，将目标从背景中分离。因此，该方法并不准确的描述目标，而是使用分类器集合来决定一个像素属于目标还是背景，从而可以更好地应对光照、尺寸的变化，目标的变形和遮挡等问题。在集成为强分类器时，需要为每个若分类器赋予一个权重，性能较好的分类器则具有较高的权重，该权重是由弱分类器的误差率决定的:

$$\alpha_t = \frac{1}{2} \log \frac{1 - err}{err} \tag{12}$$

其算法流程如算法 1 所示。

Algorithm 1: Ensemble Tracking

Input: n video frames $I_1, \cdots, I_n$, Rectangle r_1 of object in first frame
Output: Rectangles $r_2, \cdots, r_n$
Initialization (for frame I_1):
Extract $\{x_i\}_i^N = 1$ examples with labels $\{y_i\}_i^N = 1$
Initialize weights $\{w_i\}_i^N = 1$ to be $\frac{1}{N}$
for $t = 1 \rightarrow T$ do
  (a) Make $\{w_i\}_i^N = 1$ a distrubition
  (b) Train weak classifier h_t
  (c) Set err $= \sum_{i=1}^{N} w_i \left| h_t\left(\mathrm{x_i}\right) - y_i \right|$
  (d) Set weak classifier weight $\alpha_t = \frac{1}{2} \log \frac{1-err}{err}$
  (e) Update example weights $w_i = w_i e^{(\alpha_t |h_t(\mathrm{x_i}) - y_i|)}$
The strong classifier is given by sign$(H(\mathrm{x}))$ where $H(x) = \sum_{t=1}^{T} \alpha_t h_t(\mathrm{x})$
for each new frame I_j do
  Extract $\{\mathrm{x_i}\}_{i=1}^{N}$ examples
  Test the examples using the strong classifier $H(x)$ and create confidence image L_j
  Run mean-shift on L_j with r_{j-1} as the initial guess. Let r_j be the result of the mean shift algorithm
  Define labels $\{y_i\}_{i=1}^{N}$ with respect to the new rectangle r_j
  Remove K oldest weak classifiers
  Initialize weights $\{w_i\}_{i=1}^{N}$ to be $\frac{1}{N}$
  for $l = K + 1 \rightarrow T$ do // Update weights
    (a) Make $\{w_i\}_{i=1}^{N}$ a distribution
    (b) Choose $h_t(\mathrm{x})$, with minimal error err, from $\{h_{K+1}(\mathrm{x}), \ldots, h_T(\mathrm{x})\}$
    (c) update α_t and $\{w_i\}_{i=1}^{N}$
    (d) Remove $h_t(\mathrm{x})$ from $\{h_{K+1}(\mathrm{x}), \ldots, h_T(\mathrm{x})\}$
  for $l = 1 \rightarrow K$ do // Add new weak classifiers
    (a) Make $\{w_i\}_{i=1}^{N}$ a distribution
    (b) Train weak classifier h_t
    (c) Compute err and α_t
    (d) Update example weights $\{w_i\}_{i=1}^{N}$
  The updated strong classifier is given by sign$(H(\mathrm{x}))$ where $H(x) = \sum_{t=1}^{T} \alpha_t h_t(\mathrm{x})$

代码实现如下[1]：

```
def svm(features, labels):
    clf = SVC()
    clf.fit(features, labels)
    return clf

def weak_classifier(features, labels, w):
    features_flatten = np.reshape(features, (-1, 11))
    labels_flatten = labels.ravel()
    clf = svm(features_flatten, labels_flatten)
    err = calculate_err(w, clf.predict(features_flatten), labels_flatten)
    alpha = 0.5 * math.log(10, abs(1 - err) / err)
    w = w * np.exp(alpha * np.abs(clf.predict(features_flatten) - labels_flatten))
    return clf, alpha, w

def adaboost(features, labels, top_y, top_x, n_rows, n_cols):
    global location
    num_of_classifiers = 5
    classifiers = []
    alpha_array = []
    h_array = []
    err_array = []
    w0 = np.full(labels.ravel().shape, 1 / (np.reshape(features, (-1, 11)))).shape[0])
    w = w0.copy()
    for index in range(0, num_of_classifiers):
        clf, alpha, w = weak_classifier(features, labels, w)
        classifiers.append(clf)
        alpha_array.append(alpha)
    for pic_i in range(2, 127):
        if pic_i < 10:
            pic_name = "00%d.bmp" % pic_i
        elif 100 > pic_i >= 10:
            pic_name = "0%d.bmp" % pic_i
        else:
            pic_name = "%d.bmp" % pic_i
        path = os.path.join(r"./data/football/", pic_name)
        image = cv2.imread(path)
        if pic_i == 2:
            padding = 12
            top_y -= padding
            top_x -= padding
            n_rows += 2 * padding
            n_cols += 2 * padding
        image_cut = image[top_y:top_y + n_rows, top_x:top_x + n_cols]
        features = calculate_features(image_cut, 5)
        labels = generate_labels(features, *location)
        features_flatten = np.reshape(features, (-1, 11))
        labels_flatten = labels.ravel()
        prediction = np.zeros(labels_flatten.shape)
        h_array.clear()
        for _, clf in enumerate(classifiers):
            h = clf.predict(features_flatten)
            h_array.append(h)
            prediction += alpha_array[_] * h
        prediction[prediction < 0] = 0
        prediction = (prediction - np.min(prediction)) / np.ptp(prediction)
        height, width, channel = features.shape
        c_map = np.reshape(prediction * 255, (height, width))
        (top_y, top_x, n_rows, n_cols) = mean_shift(image, c_map, *location)
        location = (top_y, top_x, n_rows, n_cols)
        padding = 12
        top_y -= padding
        top_x -= padding
        n_rows += 2 * padding
        n_cols += 2 * padding
        image_cut = image[top_y:top_y + n_rows, top_x:top_x + n_cols]
        features = calculate_features(image_cut, 5)
        labels = generate_labels(features, *location)
        labels_flatten = labels.ravel()
        del classifiers[0]
        del alpha_array[0]
        del h_array[0]
        err_array.clear()
        for h in h_array:
            err_array.append(calculate_err(w0, h, labels_flatten))
        w = w0.copy()
        while len(err_array) > 0:
            idx = np.argmin(np.array(err_array))
            alpha = 0.5 * math.log(10, abs(1 - err_array[idx]) / err_array[idx])
            w = w * np.exp(alpha * np.abs(h_array[idx] - labels_flatten))
            alpha_array[idx] = alpha
            del err_array[idx]
        clf, alpha, w = weak_classifier(features, labels, w)
        classifiers.append(clf)
        alpha_array.append(alpha)
```

main2.py

3 实验

这一章节将会利用上一章介绍的两种目标跟踪方法在两段视频序列上进行实验，并对其实现效果、实时性进行展示和分析。

1. 特征提取及标签生成等相关代码由于篇幅原因在此不再展示

(a) (b) (c) (d)

图 3: 在 Car_Data 视频序列上实现基于 Mean Shift (均值漂移) 目标跟踪: (a) 第一帧（手工标定目标初始位置）, (b) 第十三帧, (c) 第六十八帧, and (d) 第一百帧.

3.1 基于均值漂移的目标跟踪

本文在 Car_Data 视频序列上实现基于 Mean Shift (均值漂移) 目标跟踪，待跟踪的目标为场景中的车辆，初始目标位置标定需手工标定[2]，后续帧中的目标位置需通过均值漂移方法得到。实验结果如图 3 所示，可以看到，在整个视频序列上，基于均值漂移的目标跟踪表现较好，可以比较准确地对场景中的车辆进行跟踪，仅仅在部分帧中，目标框的绘制与实际目标位置有所偏移。但其并不一定有如此良好的性能表现，与初始目标位置的标定（如目标框大小）有关，如图 4 所示，在初始目标位置的标定发生改变后，在直线道路上跟踪效果依旧比较理想，但在弯道上目标框的绘制发生大幅度偏移，更甚至跟踪到另一辆车辆。这也印证了 Mean Shift 依赖对目标的描述而又缺乏必要的模板更新等原因，致使其对边缘遮挡、目标旋转、变形和背景运动不敏感。

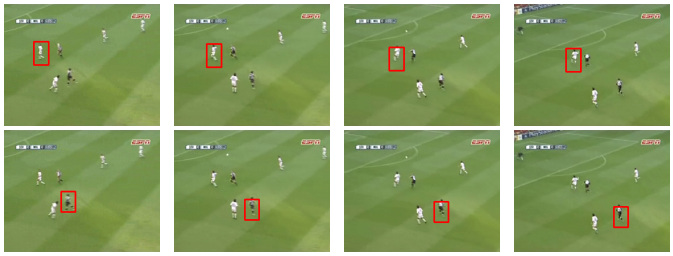
(a) (b) (c)

图 4: 在 Car_Data 视频序列上实现基于 Mean Shift (均值漂移) 目标跟踪: (a) 第一帧（手工标定目标初始位置）, (b) 第二帧, and (c) 第九十七帧.

3.2 基于分类思想的目标跟踪

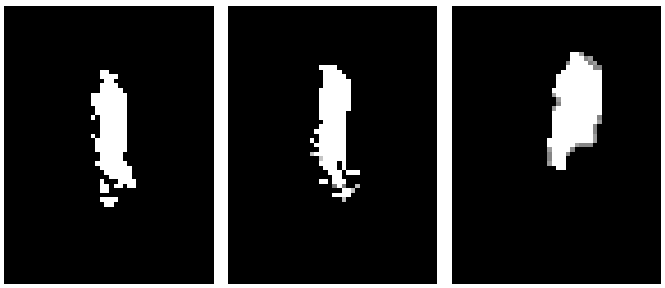
本文在 Football 视频序列上实现基于分类思想的目标跟踪，分类器采用二分类的支持向量机，并设置分类器的个数为 5，待跟踪的目标为场景中的某个运动员，初始目标位置标定需手工标定，后续帧中的目标位置需通过在线分类器得到。实验结果如图 5 所示，可以看到，在部分视频序列上，基于分类思想的的目标跟踪表现较为理想，基本可以在复杂的环境中准确地目标进行跟踪，即使目标的形状发生了改变。但

2. 本文所进行的实验假设视频序列中目标尺度没有很大变化，故在实现算法中只考虑单一尺度，即首帧中的目标大小。



(a) (b) (c) (d)

图 5: 在 Football 视频序列上实现基于分类思想的目标跟踪: 上：对左上角足球运动员进行跟踪；下：对右下角足球运动员进行跟踪. (a) 第一帧（手工标定目标初始位置）, (b) 第十帧, (c) 第二十帧, and (d) 第三十帧.



(a) (b) (c)

图 6: 在 Football 视频序列上实现基于分类思想的目标跟踪获得的置信图: (a) 第二帧, (b) 第十帧, and (c) 第二十七帧.

由于训练数据的标注较为粗糙，将目标框内的所有像素都标注为目标，从而导致训练得到一个好的分类器更加困难，反映到目标跟踪上为随着视频序列的演进，目标的跟踪逐渐偏移。除此以外，也与目标的尺度变化有一定的关系，后续改进可以在不同尺度的图片上对模型进行训练并将不同尺度的预测结果进行融合。基于分类思想的目标跟踪在分类得到的置信图上利用 Mean Shift 算法对目标位置进行预测，如图 6 所示，置信图与真实情况基本相符，整体跟踪效果比较好，在出现遮挡的情况下仍能够实现较好的跟踪，没有丢失目标。

表 1: 不同跟踪方法的实时性比较

	均值漂移	Adaboost
单张耗时（秒）	0.019	3.26
处理帧数（fps）	51.36	0.3067

对于算法的实时性，如表 1 所示，基于均值漂移的目标跟踪方法计算量不大，在目标区域已知的情况下完全可以做到实时跟踪；而对于基于分类思想的目标跟踪方法，虽然采用了在线更新的策略，且算法简单，但由于算法的实现采样点过密，计算量相对较大，实时性较差。尤其是对于第一帧要对多个若分类器进行初始化训练，所需时间更长。

4 总结

本文主要对目标跟踪的两种方法进行了探讨和实验，即基于均值的目标跟踪方法和基于分类思想的目标跟踪方法。前者计算量不大，在目标区域已知的情况下完全可以做到实时跟踪，但相对而言对边缘遮挡、目标旋转、变形和背景运动不敏感。而后者将背景信息考虑在内，将目标跟踪看作分类问题，把耗费时间的训练阶段分解成一系列简单的可以在线的学习计算任务，同时自动调整了不同分类器的权值，在不同的特征值空间训练，可以保持稳定的变化来适应光照和遮挡等。但由于粗糙的特征标注和缺乏尺度的变化，在实际实验中稳定性和实时性还有待进一步提高。