AUTOMATIC PEDESTRIAN DETECTION AND TRACKING WITH A MULTIPLE-CUE MAX-MARGIN FRAMEWORK

by

Ferdinand Stefanus
B.Eng., Nanyang Technological University, 2003

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

© Ferdinand Stefanus 2011
SIMON FRASER UNIVERSITY
Spring 2011

All rights reserved. However, in accordance with the Copyright Act of Canada, this work may be reproduced without authorization under the conditions for Fair Dealing. Therefore, limited reproduction of this work for the purposes of private study, research, criticism, review and news reporting is likely to be in accordance with the law, particularly if cited appropriately.
APPROVAL

Name: Ferdinand Stefanus
Degree: Master of Science
Title of Thesis: Automatic Pedestrian Detection and Tracking with a Multiple-Cue Max-Margin Framework
Examinining Committee: Dr. Brian Funt
Chair
Dr. Greg Mori, Co-Senior Supervisor
Dr. Ze-Nian Li, Co-Senior Supervisor
Dr. Anoop Sarkar, SFU Examiner

Date Approved: 13 December 2010
Declaration of Partial Copyright Licence

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection (currently available to the public at the “Institutional Repository” link of the SFU Library website <www.lib.sfu.ca> at: <http://ir.lib.sfu.ca/handle/1892/112>) and, without changing the content, to translate the thesis/project or extended essays, if technically possible, to any medium or format for the purpose of preservation of the digital work.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author’s written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

While licensing SFU to permit the above uses, the author retains copyright in the thesis, project or extended essays, including the right to change the work for subsequent purposes, including editing and publishing the work in whole or in part, and licensing other parties, as the author may desire.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

Simon Fraser University Library
Burnaby, BC, Canada
Abstract

Object tracking is a computer vision task of predicting objects’ locations in a video sequence. Estimating an object’s trajectory is usually accomplished by a combination of cues, including an appearance model that describes the appearance of the target object and a motion model that describes object dynamics. In this thesis, we present MMTrack, a principled framework for integrating multiple cues in object tracking. The framework formularizes object tracking as a structured prediction problem whose resulting optimization is solved with Structural Support Vector Machine. The formulation features joint learning of appearance and motion model parameters, as well as incorporation of descriptive and discriminative appearance models. We also show a fully automatic pedestrian detection and tracking system based on MMTrack, and present its performance on real-world data sets.
“Let’s go exploring!”

— Calvin and Hobbes, Bill Watterson, 1995
Acknowledgments

I would like to thank my supervisors, Dr. Greg Mori and Dr. Ze-Nian Li, without whose guidance this thesis would not exist. I am also thankful to my fellow Masters student Bahman Yari Saaed Khanloo, with whom I collaborated on this tracking project. Extremely valuable discussions with Mani Ranjbar also helped shape this work. Finally, I thank all Vision and Media Lab students for their help and companionship.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approval</td>
<td>ii</td>
</tr>
<tr>
<td>Abstract</td>
<td>iii</td>
</tr>
<tr>
<td>Quotation</td>
<td>iv</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>v</td>
</tr>
<tr>
<td>Contents</td>
<td>vi</td>
</tr>
<tr>
<td>List of Tables</td>
<td>viii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>ix</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2 Related Work</td>
<td>5</td>
</tr>
<tr>
<td>3 MMTrack</td>
<td>10</td>
</tr>
<tr>
<td>3.1 Features for appearance model</td>
<td>11</td>
</tr>
<tr>
<td>3.1.1 Histogram of Oriented Gradient score</td>
<td>12</td>
</tr>
<tr>
<td>3.1.2 Color histogram distance</td>
<td>12</td>
</tr>
<tr>
<td>3.1.3 Appearance templates</td>
<td>15</td>
</tr>
<tr>
<td>3.2 Trajectory scoring</td>
<td>15</td>
</tr>
<tr>
<td>3.2.1 Emission model</td>
<td>16</td>
</tr>
<tr>
<td>3.2.2 Transition model</td>
<td>17</td>
</tr>
<tr>
<td>3.3 Parameter learning</td>
<td>19</td>
</tr>
</tbody>
</table>
3.4 Inference .................................................. 21
3.5 Implementation details .................................. 22

4 Automatic Detection and Tracking Using MMTrack 24

5 Experimental Results 30
  5.1 UBC Fireworks Dataset .................................... 30
    5.1.1 Quantitative evaluation ............................... 31
    5.1.2 Qualitative evaluation ................................. 33
  5.2 PETS 2009 Dataset ....................................... 36

6 Discussions and Conclusions 41
  6.1 Future Work ............................................... 42

Bibliography 43
List of Tables

5.1 Tracking results on 22 UBC dataset test sequences. . . . . . . . . . . . . . . . . 32
## List of Figures

3.1 A frame and its corresponding HOG feature map. . . . . . . . . . . . . . . . . . 12  
3.2 Color histogram distance features generation. . . . . . . . . . . . . . . . . . 14  
3.3 Emission model as weighted combination of appearance features. . . . . . . . 17  
3.4 Transition model binning and motion prior learned on the bins. . . . . . . . . 18  
3.5 Euclidean distance loss function used in MMTrack formulation. . . . . . . . . 20  
4.1 Automatic tracking result for one pedestrian. . . . . . . . . . . . . . . . . . . 28  
5.1 Trajectories obtained with different feature combinations. . . . . . . . . . . . 33  
5.2 Block division method used to make inference feasible on long sequences. . 34  
5.3 Sample tracking results on crowded conditions. . . . . . . . . . . . . . . . . . . 35  
5.4 HOG and histogram features on nighttime sequence. . . . . . . . . . . . . . . 35  
5.5 Camera calibration result on PETS dataset. . . . . . . . . . . . . . . . . . . . 37  
5.6 Tracking results under occlusion on PETS dataset. . . . . . . . . . . . . . . . 38  
5.7 Sample tracking results on PETS dataset. . . . . . . . . . . . . . . . . . . . 40
Chapter 1

Introduction

Object tracking is a computer vision task that deals with estimating trajectories of target objects in a sequence of frames. Depending on the application, the target objects can be pedestrians, people’s faces, vehicles, or other objects. There are many potential applications where knowing locations of certain target objects can be useful. Some example applications based on object tracking are a perceptual user interface that tracks user’s pupils and categorizes the person’s head gesture based on the pupils’ trajectory [9], a vehicle and crowd tracking system that automatically generates flow data used in city planning and crowd monitoring [18], and a surveillance application that tracks foreground objects and raises real-time alerts according to certain preconditions set by a user [7].

Due to human’s excellent cognitive ability to detect and follow objects visually, the tracking task may seem deceptively simple at first. However, developing computer algorithms to track objects in realistic scenarios turns out to be a non-trivial task. To illustrate the challenges faced by an object tracking algorithm, consider the following example of pedestrian tracking in a video recorded using a stationary camera. A robust pedestrian tracking algorithm should be able to handle changes in pedestrians’ appearance caused by human articulation or change in illumination. This suggests that the object model should be continually updated to reflect changes in pedestrian’s pose. On the other hand, by continually adapting the object model, there is a possibility that a small error in the pedestrian’s hypothesized location will cause incorrect information to be absorbed by the object model. Over time, the errors may accumulate, and the object model may not accurately reflect the target pedestrian’s appearance anymore, causing the tracker to drift to another object. Further, because there can also be multiple pedestrians in the view at any one time, the
CHAPTER 1. INTRODUCTION

An object tracking algorithm should also be able to differentiate between different instances of the pedestrians. This task is especially difficult if the pedestrians are similar in appearance. The pedestrians may also interact, introducing issues such as partial or full occlusion. Occlusions can also occur due to interaction between the target and background object, for example when a tracked pedestrian walks behind a traffic sign. Additionally, the tracker should also be able to handle change in scale caused by the pedestrian’s relative distance to the camera. A fully automatic pedestrian tracking algorithm should also be able to automatically initialize a new trajectory when a pedestrian enters the scene, and to terminate an existing trajectory when a pedestrian exits the scene.

All these challenges have resulted in a good number of solutions being proposed in the subject of object tracking. Yilmaz et al. present a survey on object tracking methods in [32]. In general, tracking algorithms predict a target’s location by modeling two of its characteristics: its appearance and its motion. The target’s appearance is represented by an appearance model, which usually describes the target’s shape or its distinctive features such as color or texture. Some common representation of a target’s shape are its silhouettes [31] or simple shapes such as ellipses [6, 4] or rectangles [5, 2]. Appearance features that can be used to describe a target include color histogram [6, 4], textures [23, 2], and edges [21].

Whereas the appearance model describes how a target looks like, the motion model, on the other hand, encodes prior knowledge or assumption about the target’s movement patterns. A motion model serves to restrict the range of possible target’s movement, and is useful because a target’s positional state usually does not change abruptly between consecutive frames. An example of a motion model is the Brownian motion model [22], which models the target’s dynamics as Gaussians centered on its previous state. Babenko et al. [2] use a simpler motion model that assumes the target to be equally likely to appear at any location within a certain radius from its previous location. Another example of a motion model is the constant heading model [1] that assumes the target does not change its direction between a pair of consecutive frames.

Most object tracking algorithms treat appearance and motion models independently. The two models are usually integrated by using the motion model to guide the search for the location that provides the best match to the appearance model [2, 5, 6, 4]. In this thesis, we present a principled method to integrate both models in a uniform manner. This integration is achieved by learning weight parameters indicating relative importance of
CHAPTER 1. INTRODUCTION

each model, where the weights basically indicate which model is trusted more, and to what degree. The weighting of the models implies a balancing act between appearance and motion cues. In fact, the appearance model itself can also be a combination of different cues, each representing different aspect of the target’s appearance. For example, the appearance of a target can be described by a combination of its color distribution and its silhouettes. There may also be different types of cues used in the appearance model. Referring back to the pedestrian tracking example, the appearance model can be composed of both continually-updated and static cues, in the hope to strike a balance between appearance adaptation and resistance to tracking drift. In both cases, different appearance cues can also be integrated by weighting each cue according to their relative importance. Unfortunately, deciding on good parameters to weigh the different cues, either by arbitrary fixing of weights or empirical study of available data, is often a difficult or time-consuming task. This is especially the case for the integration of appearance and motion models, as there is often no clear relationship between them.

In this thesis, we present a machine learning-based method to automatically determine the relative importance of different cues. Although the framework is general, in this work we present its application in the context of pedestrian tracking by choosing appearance features that describe pedestrians. The framework uses Structural Support Vector Machine [28] to find weight parameters that measure the relative importance of appearance and motion models, where the appearance model is composed of various appearance cues describing the appearance of a pedestrian. Structural SVM is a recent development in machine learning that generalizes SVM formulation to deal with interdependent and structured variables. The choice of Structural SVM as our learning mechanism is supported by the observation that object tracking can be formulated as structured prediction problem, as it tries to find the sequence of coordinates that best explain input features. The sequence of coordinates represents a structured output space, because each pair of coordinates are strongly interdependent as the valid range of possible movements of the object is restricted by the motion model. In other words, an object is more likely to move to nearby locations in the next frame than to locations that are far away. In addition, by formulating the problem as a linear chain structure with emission and transition models, we will show that Structural SVM also provides an intuitive and principled way to treat the feature representation and motion model in a unified way while jointly learning the relative contributions of multiple cues.
This work is a collaboration with Bahman Khanloo, Mani Ranjbar, Ze-Nian Li, and Greg Mori, parts of which are presented in our CRV paper [13]. My contributions to the work are focused on the appearance feature extraction, implementation of the approximate inference algorithm, and extensions of the framework to a fully automatic tracker and to handle scale variations. Bahman Khanloo, my fellow Masters student, handled the application of Structural Support Vector Machine in the context of single target tracking.

The organization of this thesis is as follows. In the next chapter, we present related work in object tracking. We then propose our tracking framework, called MMTrack, in Chapter 3. This is followed by an explanation of how MMTrack can be extended to be a fully automatic object tracking system, which is presented in Chapter 4. Experimental results on real-world datasets are shown in Chapter 5. Finally, we conclude in Chapter 6.
Chapter 2

Related Work

Most tracking algorithms predict a target’s trajectory by utilizing cues obtained from two characteristics of the target, namely its appearance and its dynamics. As mentioned in the previous chapter, an *appearance model* is used to describe the how the target looks like, whereas a *motion model* is used to describe how the target moves in the scene. The appearance model is built from the target’s visual features such as shape, color, or textures [32], and is used to distinguish the target from other objects, which include objects of different class from the target (i.e. background objects) and other objects of interest. The motion model, on the other hand, is built from prior knowledge or assumption about the target’s dynamics, and is used to complement the appearance model by providing information about the target’s possible movement patterns.

One feature that is frequently used for building an appearance model is the target’s color histogram. An object’s color histogram is an attractive choice for an appearance feature because it provides a nonparametric, discrete density estimate of the target’s color distribution, and it is also relatively inexpensive to compute. Comaniciu et al. [6] use the target’s color histogram as the appearance feature by computing the target’s 3D RGB histogram at a certain manually-specified frame and iteratively applying mean-shift procedure in each subsequent frame to search for the location whose color distribution has the minimum Bhattacharyya distance to the target’s histogram. Bradski et al. [4] extends this method to handle varying target size by running multiple mean-shift iterations with various window sizes, where the window size at each iteration is initialized according to the area under the converged search window in the previous iteration.

Another example of a histogram-based appearance model is the model proposed by
Ramanan et al. [20]. The work is mainly concerned with tracking of animals, and its main idea is that animal tracking can be automatically done by automatic detection and tracking of segments belonging to animal parts. The method first detects potential animal segments by running simple detectors such as Haar-like edge detectors over the whole image sequence. One-dimensional RGB histograms are then built for locations with high detector responses, and these histograms are clustered with mean-shift clustering to group similar segments. A Gaussian distribution model is then computed over each cluster, and used to represent each segment’s appearance. Combining this appearance model with a velocity constraint that specifies the maximum velocity at which the segments can move, tracking of an animal then becomes a problem of finding good matches for the segments, such that the hypothesized locations form a chain in the video sequence that obeys the bounded velocity constraint and whose appearances conform to the segments’ appearances obtained in the clustering step. Inference in this setting is efficiently solved with dynamic programming in the form of Viterbi algorithm [30].

It should be noted that the appearance models used in the above two approaches are static models - the models are pre-computed once, and they are never updated throughout the inference process. This is in contrast to an adaptive appearance model wherein the target’s appearance is continually updated to reflect changes in the target’s appearance. For example, in the work by Grove et al. [11], a histogram-based appearance model is updated by adding observations from the current frame to the histogram. The decision on which updating mechanism to use is one that should be made carefully. A naive updating mechanism that completely replaces the appearance model at each frame by current observation will easily cause track drift when there is an error in the hypothesized target location. At the other extreme, an appearance model with no updating mechanism has no means to adapt to changes in the target’s appearance. This template update problem [17] suggests that a good tracking algorithm should strike a balance between a fixed appearance model and an adaptive one.

One approach for building an adaptive appearance model that has gained popularity recently is to build an appearance model by automatically selecting the best appearance features out of a fixed pool of appearance features. This approach was first proposed by Collins et al. [5]. In this work, a good appearance feature is defined as one that can best discriminate the target from its surrounding. The method described in the paper uses a fixed-surround configuration to obtain feature samples from the target and its surrounding,
where the features consist of one-dimensional histograms computed from linear combinations of RGB values. A Fisher-like criterion is then used to give a quantitative measure of each feature’s discriminative power. The final appearance model is then given by the collection of features with the highest discriminative power. This feature selection procedure is executed for each incoming frame, thus enabling the appearance model to handle change in the target’s appearance. At every frame, each histogram feature is back-projected to build a probability image where each pixel location indicates how compatible the input features are to the histogram at that location. The target’s location is then obtained by running mean-shift procedure on each probability image. Finally, the target’s location in that frame is determined as the median of its locations in the probability images.

Unfortunately, updating of the most discriminative features done by Collins et al. raises the possibility of tracker drift. This is because the appearance model is updated at each frame according to the observed features at current tracker position, and this updating mechanism has an inherent assumption that the current tracking result is correct. In practice, the tracking result may not be precisely aligned to the target, and thus updating the appearance model will result in incorrect observations being learned by the appearance model. This ambiguity of the tracking result is the main cause of tracking drift - current tracker position is the strongest indication to the actual target’s location, and yet it cannot be trusted to be free of errors. The MILTrack algorithm proposed by Babenko et al. [2] attempts to address this issue by employing multiple instance learning boosting mechanism to handle the ambiguities inherent in the tracking results.

In Multiple Instance Learning (MIL), training data are not labelled as instance-label pairs but as bag-label pairs, where a bag consists of a collection of training instances, and it is labelled as positive if it contains at least one positive sample, or as negative otherwise. The MIL training objective is to find a set of weak instance classifiers that can best classify the bags. MIL training mechanism is specifically handled to handle the ambiguities present in the training data which arise from labels being assigned to bags and not instances.

The argument for using MIL in the context of object tracking is that MIL can make the resulting tracker more resistant to drift, as its learning mechanism handles ambiguity of the training data. As mentioned earlier, the target’s hypothesized position obtained from a tracker may contain ambiguities as the position may contain misalignment with respect to the target’s true position. If the hypothesis made in the next frame is dependent solely on the current tracking result, this misalignment may be accumulated over time, which will
eventually cause tracking drift error. MILTrack minimizes this dependence on the current tracking result by collecting multiple samples near to current tracking result, putting them into a positive bag, and letting the boosting mechanism handle the ambiguities of the training instances. After the boosting process, the target’s new location is found by simply running the resulting strong classifier on patches around the target’s current location, and selecting the location with the highest response. In other words, the system uses a motion model where each location within a certain radius from the target’s current location is equally likely to be the next location.

One thing to note in the trackers mentioned above is that in all those examples, the motion model is treated as a separate component from the appearance model. The mean-shift procedure used by Comaniciu et al., Bradski et al., and Collins et al., the bounded velocity model used in Ramanan’s animal tracking algorithm, and the search for local maximum used in MILTrack are essentially additional constraints placed on top of the appearance model’s results of searching for locations with good appearance matches. The motion models in these examples do not encode prior knowledge about object movement pattern, nor do they allow the system to express how much it trust the motion model over the appearance model or vice versa.

A number of authors have explored the approach of combining different cues and assigning relative trust, or importance, to the cues. For example, Berclaz et al. [3] propose an appearance model built from a combination of ground plane occupancy and color model cues, where both cues are assigned equal importance. Song et al. [25] combine color and texture cues by assuming that they are independent and thus the likelihood arising from the cue combination is defined simply as multiplication of the likelihoods arising from each of the cues. Spengler et al. [26] adopt democratic integration process to combine five different cues. In democratic integration process, different cues are combined by a weighted combination of the cues, and the weight for each cue is updated according to its quality measurement for every input frame. It should be noted that in these examples, the relative importance of each cue is determined independently of other cues.

Another approach of combining different sources of information is proposed by Stenger et al. [27]. Instead of combining different tracking cues, the proposed method combines full tracker systems, or observers, with each system having its own appearance and motion models. The method estimates a confidence value for each observer from training data, by independently evaluating how well a particular observer performs on the training data.
These confidence values are used to weigh the different observers, and performance is evaluated on different configurations of observers (e.g. parallel, cascade configurations).

All the above examples report better performance when multiple sources of information are used. Intuitively, using a combination of cues may help in achieving a good balance between the advantages and disadvantages of each cue, if the parameters weighting the different cues are chosen carefully. However, current methods to determine the weights of each cue either rely on some ad-hoc method such as setting a fixed weight, or do not consider the sources jointly. Joint estimation of the weights is of interest here because each weight actually represents the importance of a cue relative to other cues, and thus it should be determined in consideration with other cues. With our MMTrack system, we focus on formulating a principled framework to determine the parameters used to combine different cues, such that the parameters are determined jointly within a structured prediction learning mechanism.
Chapter 3

MMTrack

In this chapter we describe our proposed offline single-target tracker mechanism called MMTrack and show how it can be applied in the context of pedestrian tracking. MMTrack is a framework based on Structural SVM [28] in which different tracking cues (which may include a combination of appearance and motion models) can be combined in a principled manner. MMTrack combines the cues by learning parameters, or weights, for each of the cues.

There are several differences between MMTrack and the single-object trackers described in the previous chapter. First, the trackers described previously use only single type of features for their appearance model: the offline tracker proposed by Ramanan et al [20] uses Lab-space color histogram, the tracker proposed by Collins et al [5] uses linear combinations of RGB channels, and MILTrack [2] uses Haar-like appearance features. In contrast, MMTrack allows the use of different types of features. In subsequent sections we will describe how we use a combination of offline and online appearance models in MMTrack framework.

Additionally, the trackers mentioned in the previous chapter are mainly focused with building appearance models and their motion models are treated independently from the appearance models. For example, bounded velocity model is used in Ramanan et al’s animal tracker [20] to handle object dynamics, whereas gradient ascent with mean-shift and finding local maximum are used in the works by Collins et al [5] and Babenko et al [2], respectively. MMTrack formulation, on the other hand, provides a unified model to handle both motion and appearance model in the same fashion.

Although trackers that use multiple cues exist, they usually set parameters associated with each cue independently. The multiple-cue trackers proposed by Berclaz et al [3],
Spengler et al. [26], and Song et al. [25] assign empirically-set constant weights to each of the cues. Other multi-cue trackers such as the ones proposed by Stenger et al [27], Kolsch et al. [14], and Liu et al. [16] learn weights for each cue independently from each other. Determining the weights for different cues jointly is not trivial as there is often no clear relationship between them. With Structural SVM adopted by MMTrack, weights for the cues can be learned jointly in a principled manner.

The rest of this chapter is organized as follows. In Section 3.1, we describe the appearance features we use in MMTrack. Section 3.2 describes how the appearance features, together with the motion model, are used to score trajectories. In Section 3.3, we show how Structural SVM is used to learn the appearance and motion model parameters. Finally, Section 3.4 shows how the learned parameters are used in the inference procedure.

### 3.1 Features for appearance model

We use a combination of descriptive and discriminative features in our tracking framework. While descriptive features are used to describe appearance of the object being tracked, discriminative features are intended to distinguish the tracked object from other objects. We use color histogram distance and appearance templates as the descriptive features and Histogram of Oriented Gradient (HOG) [8] score map as our discriminative feature.

The intuition for using the different features is that each of the features will provide the tracking system with different information. We expect the use of HOG feature to help the system discriminate between pedestrians and non-pedestrian objects (e.g. car, trees, etc.). Color histogram distance features are used to provide information about static appearance of the tracked pedestrian. Appearance template features, on the other hand, provides information about a particular pedestrian at a finer level than the histogram distance features. Further, unlike the static histogram distance features, one type of the appearance template features is also used to provide a continually-updated appearance model of a pedestrian. With the combination of both static and continually-updated appearance model, we hope to achieve a good balance between resistance to track drift and adaptation to a pedestrian’s changing appearance throughout its trajectory.
3.1.1 Histogram of Oriented Gradient score

The first appearance feature that we use is the SVM classifier output of Histogram of Oriented Gradient (HOG). HOG is a descriptor-based system designed for object detection, with pedestrian detection being one of its main applications. HOG uses a sliding window approach to detect objects, where a fixed-size window is scanned along an input image, and at each window position, HOG descriptors are computed, and fed to a classifier (the implementation used in our experiments uses linear SVM as its classifier) to determine the presence of object of interest at that location. To detect objects at various scales, the input image is resized to different resolutions, and the process is repeated for each resolution.

The purpose of using HOG as an appearance feature is to help the system distinguish between pedestrians and non-pedestrians (e.g. trees, pavement). We retrained the SVM classifier to detect pedestrians in top down view to make it suitable for our particular experimental setup. The detection window is set to 48x112 and the input image is rescaled up to 130% of its original resolution. The feature is built by taking at each pixel position the maximum response of all detector windows centered at that pixel. We then linearly normalize the map so its values fall within the range $[0, 1]$, and use this map as our feature. The resulting feature map ideally has peaks at the pedestrian locations. Fig. 3.1 shows a sample frame along with its normalized HOG score map.

![Figure 3.1: A frame and its corresponding HOG feature map.](image)

3.1.2 Color histogram distance

Although HOG scores can help the system differentiate between pedestrians and other objects, they are not informative in distinguishing among different pedestrians, as HOG
detections usually have peaks on all pedestrians. Thus, features that can uniquely represent
the appearance of a pedestrian is needed. Color histogram is an appearance feature that we
use to represent the average appearance of a particular pedestrian.

The color histogram distance feature is inspired by the method used by Ramanan et al
[20], in which segments with strong edge responses are clustered with mean-shift clustering
and the resulting clusters are used to represent appearance of animal segments. Adapting the
method to our pedestrian tracking system, we detect pedestrians by running HOG detector
on every frame in the video, compute histograms on each HOG detection, and cluster all
the HOG detections. Each resulting cluster is then used to represent the appearance of a
particular pedestrian. The intuition here is that by clustering the histograms obtained from
bounding boxes around the pedestrians throughout the video, we can gain a good insight
into how the average appearance statistics of each person looks like.

The detail of the color histogram distance feature generation is as follows. First, each
HOG detection window is divided into nine sections as depicted in the second row in Fig. 3.2,
and a normalized RGB histogram is computed for each section. Each section’s histogram
consists of 30 bins, with 10 bins for each of the R, G, and B channels. The purpose of
dividing the detection windows into nine sections it to provide information to the system
about the spatial color distribution of a pedestrian. The spatial color distribution may
help to distinguish different parts of a pedestrian’s body, such as his pants or shirt. The
histograms from all the nine sections are then concatenated to give one big histogram, and
histograms from all the frames are then clustered with mean-shift clustering algorithm.

At inference time, the cluster mean corresponding to the person being tracked is used to
represent his appearance. This is done by using the cluster mean to compute nine feature
maps, one for each of the nine body sections, in which each pixel indicates the similarity of
a patch centered at the pixel to the target’s appearance. The feature maps are generated by
sliding a pedestrian-sized window across the input frame, computing nine normalized RGB
histogram at each location, and calculating the $\chi^2$ distance of each section to the cluster
mean. This results in a feature map where ideally the tracked person has low response. The
whole process is depicted in Fig. 3.2.

It should be noted that the appearance features built with this method are static fea-
tures, as the cluster means are computed prior to inference and are unchanged throughout
the inference process. Thus, these features are more resistant to drift; however, they cannot
handle cases when the tracked pedestrian’s appearance changes over the course of his
Figure 3.2: Color histogram distance features generation.
trajectory.

3.1.3 Appearance templates

Two other appearance features that we use are appearance template features. These features use two RGB templates of the target pedestrian, obtained from initial frame and previous frame. For each input image, two distance maps are generated by scanning the templates across the input image and computing the sum of absolute difference between the templates and the input image and normalizing the response to between 0 and 1. This results in feature maps where ideally the tracked pedestrian has low response. The template corresponding to the initial frame template stays unchanged throughout inference. On the other hand, the previous frame template is continually updated at every frame according to the hypothesis made at that frame.

The main purpose of using the frame templates is to provide an appearance model that can describe appearance of a person at a finer level of detail than histogram feature. Like histogram feature, the initial frame template is a static template, and thus it acts as a memory template which ensures that the tracker does not completely forget about the appearance of the target when it first showed up. Previous frame template, on the other hand, is continually updated at every frame, and it helps give the system capability to deal with appearance change.

3.2 Trajectory scoring

Central to our tracking system is the trajectory scoring model, which describes how compatible a trajectory is to the video sequence (i.e. how 'good' a trajectory is). The trajectory scoring model provides a method to quantitatively compare different trajectories. In our framework, the best trajectory hypothesis given a starting position (obtained from e.g. a detector) and a trajectory scoring model is defined as the trajectory with the highest score among all other trajectories that start from that position.

Because we are using many features, the scoring function should take into account the relative contribution of each feature in describing the trajectories. These relative contributions are represented by weights learned with Structural SVM (more details about the learning mechanism are described in Section 3.3). Thus, the scoring function is defined as
a mapping in the form of

$$F(x, y; w) : X^T \times Y^T \rightarrow \mathbb{R}$$ (3.1)

where $x = \{x^{(1)}, ..., x^{(T)}\}$ is a sequence of frames, $y = \{y^{(1)}...y^{(T)}\}$ is a trajectory, $T$ is the total number of frames, and $w$ indicates the weight parameters associated with the features.

The features include both appearance model (described in Section 3.1) and motion model, which are represented in the scoring model as emission model and transition model, respectively. For the transition model, we assume that if the location of the object at a particular frame is known, the locations of the object in previous and following frames will be independent from each other. Thus, our transition model treats a trajectory as a linear chain structure. Note that this results in a first-order model that does not consider velocity or acceleration. The decomposition of the scoring function to emission model and transition model is expressed as

$$F(x, y; w) = \sum_{t=2}^{T} F^E_t(x^{(1)}, x^{(t-1)}, x^{(t)}, y^{(1)}, y^{(t-1)}, y^{(t)}; w_E) + \sum_{t=2}^{T} F^T_t(y^{(t-1)}, y^{(t)}; w_T)$$ (3.2)

where $F^E_t(\cdot)$ and $F^T_t(\cdot)$ are linear models describing emission and transition contributions at time $t$, respectively. These functions are parameterized by $w_E$ and $w_T$ whose concatenation we denote by $w$.

### 3.2.1 Emission model

The emission model includes the appearance features, each weighted according to the weights obtained from Structural SVM. As described in Section 3.1, the features consist of HOG feature, histogram distance feature, and first and previous frame template features. Formally, the emission model is defined as:

$$F^E_t(\cdot; w_E) = w_H^T \psi_H(x^{(t)}, y^{(t)}) + w_C^T \psi_C(x^{(t)}, y^{(t)}) + w_F^T \psi_F(x^{(1)}, x^{(t)}, y^{(1)}, y^{(t)}) + w_P^T \psi_P(x^{(t-1)}, x^{(t)}, y^{(t-1)}, y^{(t)})$$ (3.3)

where $w_H$ is a weight associated with HOG feature, $w_C$ is a nine-element weight vector associated with the color histogram distance features, $w_F$ is a weight associated with first frame template feature, and $w_P$ is a weight associated with previous frame template feature. $\psi_N(x^{(t)}, y^{(t)})$ indicates a combined feature representation between input frame and
hypothesized location at frame $t$ for a particular feature $\mathcal{X}$ (i.e. $\mathcal{X}$ is one of $\mathcal{H}$, $\mathcal{C}$, $\mathcal{F}$, and $\mathcal{P}$). In the case of appearance features, $\psi_{\mathcal{X}}(x^{(t)}, y^{(t)})$ is simply the feature map value at location $y^{(t)}$. Additional terms in the $\psi$ function formulation for first and previous frame templates indicate dependence on initial and previous frame, respectively.

Intuitively, the emission model represents weighted combination of the appearance features, and the weights for the features indicate the importance of each of the features. This is depicted in Fig. 3.3.

### 3.2.2 Transition model

We use a symmetric, first-order model to model pedestrian dynamics. This means that a pedestrian’s previous movement patterns have no effect on its current movement. We don’t use higher-order models that consider velocity or acceleration as these models can potentially make the inference more complex or computationally infeasible. Although simple, the model we adopt proves to be sufficient for our dataset.

In our transition model, the area that a pedestrian can travel is discretized into four
bins that represent four concentric circles centered at the target’s current position. Weights are learned on each circle to represent motion priors on each area. Thus, the weights can be thought of as an indicator of how likely a pedestrian will move to each of the circle bins. This is depicted in Fig. 3.4.

The formal definition of the transition model is given as

\[
F_T^{(t)}(y^{(t-1)}, y^{(t)}; w^T) = w_T^T \psi_T(y^{(t-1)}, y^{(t)}) \tag{3.4}
\]

where \( \psi_T(y^{(t-1)}, y^{(t)}) \) is a binning operation on the possible target movement, and \( w^T \) represents the weights associated with each bin. Intuitively, each element in \( w^T \) can be thought of as a value that indicates how likely the target is to move to each circle in Fig. 3.4. The binning operation on the possible target movement can be represented as follows

\[
\psi_T^{(t)}(y^{(t-1)}, y^{(t)}) = \text{bin}(d(y^{(t-1)}, y^{(t)})), \tag{3.5}
\]

\[
\text{bin}_k(d') = 1_{[d' = k]}, \quad k = 0, ..., d_{\text{max}}. \tag{3.6}
\]

Here \( d(y, y') \) is the Euclidean distance between the 2d images of \( y \) and \( y' \), \( 1_{[\cdot]} \) is the indicator function and \( \text{bin}(\cdot) \) acts as a selection operator that generates a vector of length \( d_{\text{max}} + 1 \) with all the elements set to 0 except one being 1. The upper bound \( d_{\text{max}} \) on the travelled distance from one frame to the next one is empirically determined from the dataset.
CHAPTER 3. MMTRACK

3.3 Parameter learning

As mentioned earlier, we use a scoring function parameterized by a set of weights \( w \) which puts together a variety of features. The learning task amounts to jointly learning the parameters that best explain the dependencies between the features and the trajectories using video-trajectory training pairs. We use a discriminative approach, namely we try to discriminate between a compatible video-trajectory pair and all the runner-ups. Hence, we find a predictor that estimates the best trajectory given an input video by learning a set of parameters that maximize the score of training set examples.

Learning the model parameters in this problem setting is challenging since we do not have negative examples. In other words, we do not know how a bad trajectory looks like and more importantly, how it differs from a good one because this information is not included in the dataset. Notice that the scoring function in Eq. 3.2 can be viewed as a \( w \)-parameterized discriminant function. Further, the locations in a trajectory are highly interdependent and so we are dealing with a structured output problem. Hence, it is natural to adopt Structural SVM to jointly estimate the parameters.

The underlying philosophy of Structural SVM is very similar to traditional binary SVM classifier. Binary SVM finds the hyperplane that separates positive and negative training examples such that the distance between the support vectors (data points closest to the hyperplane) and the hyperplane is maximized. Structural SVM generalizes this formulation for structured output spaces with no negative examples and possibly infinite number of output hypothesis by finding a separating hyperplane, in some feature space, such that its distance to the positive example and the closest runner up is maximized. This is expressed as the following program

\[
\min_{w, \xi} \frac{1}{2}||w||^2 + \frac{C}{N} \sum_{i=1}^{N} \xi_i, \quad \text{s.t.} \quad \forall i, \xi_i \geq 0, \quad \forall i = 1, ..., N, \forall y \in \mathcal{Y} \setminus y_{(i)}. \tag{3.7}
\]

\[
F(., y_{(i)}) - F(., y) \geq \Delta(y_{(i)}, y) - \xi_i, \forall i = 1, ..., N, \forall y \in \mathcal{Y} \setminus y_{(i)}. \tag{3.8}
\]

where Eq. 3.7 guarantees unique solution for \( w \), and Eq. 3.8 specifies the constraints for correct classification. \( N \) indicates the number of training examples, and \( y_{(i)} \) indicates the label for the \( i \)-th training example. Similar to soft margin binary SVM formulation, Eq. 3.7 and Eq. 3.8 incorporate slack variables \( \xi_i \) to allow for misclassified examples in case the
training data are not linearly separable. The constant $C$ controls the relative importance of margin maximization and error minimization, and is usually determined by cross validation.

Another component of Eq. 3.8 is the loss function $\Delta(y_{(i)}, y)$ which measures how different a trajectory hypothesis $y$ is from the groundtruth $y_{(i)}$, or in other words, it measures the 'good'-ness of hypothesis $y$. Loss function is needed for structured output, as the space of $Y$ can be large and using standard 0−1 loss will not be appropriate. Note that the loss function when incorporated into the constraints as in Eq. 3.8 directly impacts the margin, and hence this method of incorporating loss function is called *margin rescaling*. It is one of a couple of methods to incorporate loss function into the constraints; another option is to rescale the slack variable. The loss function that we use in our framework measures the Euclidean distance between corresponding points in the trajectories:

$$\Delta(y_{(i)}, y) = \sum_{t \in T} d^2(y_{(i)}^{(t)}, y^{(t)}). \quad (3.9)$$

This loss function is depicted in Fig. 3.5. In Fig. 3.5, a target pedestrian is shown in red rectangle on the left, with a corresponding ground truth shown as the green trajectory and a trajectory hypothesis shown as the red trajectory. The loss is computed as the sum of the distances between corresponding points in the ground truth and the hypothesis, i.e. the sum of the blue lines’ lengths.

![Figure 3.5: Euclidean distance loss function used in MMTrack formulation.](image)

As can be seen from Eq. 3.8, $N|Y|$ margin constraints need to be considered in the Structural SVM formulation. For structured spaces, $Y$ can be very large or even infinite - in the case of object tracking, the output space is all possible trajectories, which means there is an exponentially large number of constraints (i.e. in the order of $R^T$, where $R$ is the image resolution, and $T$ is the total number of frames). However, by using the cutting plane method, an approximate solution for the margin constraints can be found by considering
only a polynomial-sized subset of the constraints [28]. With the cutting plane method, all the constraints are guaranteed to be satisfied within a precision $\epsilon$. The method works by maintaining a working set of constraints, and at each iteration through the training examples, constraints are added to the set if the constraint violates the margin by more than $\epsilon$. Note that Eq. 3.8 can be reordered as follows

$$F(., y) + \Delta(y_{(i)}, y) - F(., y_{(i)}) \leq \xi_i, \forall i = 1, ..., N, \forall y \in \mathcal{Y}^T \setminus y_{(i)} \tag{3.10}$$

At each iteration, the cutting plane algorithm finds the most violated constraint, i.e. the constraint whose output variable gives the highest value at the left-hand side of Eq. 3.10. The output variable $\hat{y}$ for this constraint is given as

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} F(., y) + \Delta(y_{(i)}, y) \tag{3.11}$$

Note that $\hat{y}$ is found when both the scoring function $F(., y)$ and the loss function $\Delta(y_{(i)}, y)$ are high. Intuitively, the output variable $\hat{y}$ can be thought of as the 'most confusing output variable', because the scoring function indicates that $\hat{y}$ is very compatible with the features, and yet the loss function indicates that $\hat{y}$ is very different from the ground truth. It should also be noted that because our loss function is decomposable, finding the right-hand side of Eq. 3.11 can be done with the same complexity as the inference (described in Section 3.4). This is also the reason for us choosing margin rescaling instead of slack rescaling - margin rescaling results in simple addition as can be seen in Eq. 3.11, whereas with slack rescaling, there will be a multiplication term which will make solving Eq. 3.10 harder.

### 3.4 Inference

After the model parameters are learned, the inference task becomes one of finding the highest scoring trajectory given the model parameters. Naive search for the highest scoring trajectory in the space of possible trajectories is computationally intractable. Fortunately, we can efficiently find the highest scoring trajectory for this problem using the Viterbi algorithm which is given by the following dynamic program

$$M^{(t)}(l_C) = \max_{l_N} \left( M^{(t-1)}(l_N) + F_T^{(t)}(\hat{y}^{(t-1)} = l_C, \hat{y}^{(t)} = l_N) + F_E^{(t)}(., \hat{y}^{(t)} = l_N), \right)$$

\[ t = 1, ..., T, \quad l_N \in \mathcal{N}(l_C). \tag{3.12} \]
CHAPTER 3. MMTRACK

Each element of the message vector $M^{(t)}$ corresponds to a pixel and indicates the marginal score of the highest scoring track that originates at the initial location and terminates at that pixel at time $t$. A traceback from the final highest scoring location is done to recover the track. In our notation, $l_C$ and $l_N$ refer to the current and next location respectively and we just look in the neighbourhood $N(l_C)$ when searching for the next possible location instead of full search. Note that this local search is valid since it complies with the nature of the movements of a pedestrian, because the pedestrian is not expected to jump to a pixel which is far away from the current location. Namely, we are finding an exact solution in the space of valid trajectories. Viterbi algorithm allows us to find the highest-scoring trajectory in the exponentially-large trajectory space (i.e. in the order of $R^T$, where $R$ is the image resolution) with a $O(TR^2)$ complexity.

3.5 Implementation details

The main bulk of computation time for the inference comes from the feature generation. Because HOG features are not dependent on the tracked pedestrian, we eliminate duplication of HOG feature generation by precomputing HOG features for all frames once and loading them from disk during inference. For the histogram distance feature, we use integral histogram optimization technique [19] to efficiently compute the histogram of any rectangular window in the image. With the integral histogram technique, all nine histograms representing the different body sections can be computed with a few arithmetic operations once the integral histogram is built. We also optimize computation of the appearance template features by computing the sum-of-absolute difference with a modified integral image technique [29].

However, due to the high resolution of the datasets that we use (1400 × 1080 and 768 × 576), full inference at the original resolution turns out to be computationally prohibitive, even with optimized feature generation process. To overcome this, we resort to pruning strategies for inference at test time by performing beam search. With beam search, we just evaluate the $K$ highest-scoring trajectories at each time step and discard the rest. Thus, beam search reduces the complexity of our inference to $O(TK^2)$, where $K$ is much smaller than $R$. This allows us to perform inference at the original resolution, at the expense of suboptimality because beam search only considers a small subset of the whole hypothesis space. We set $K$ to a small value (3) in our experiments, because our tests suggest that
larger values that still maintain computational feasibility do not produce significant change to the results, as the best trajectories are usually located near to each other. However, approximate inference with beam search cannot be applied at training phase as Structural SVM training requires exact inference. To make exact inference feasible at training phase, we subsample the input to one-sixth its original resolution. Even though both beam search and input subsampling explore only a subset of the whole hypothesis space, our experimental results show that both approximations work well in practice.
Chapter 4

Automatic Detection and Tracking Using MMTrack

The previous chapter describes MMTrack formulation and its inference procedure, where the target’s initial position is assumed to be known during inference. In this chapter, we aim to fill in the remaining pieces on how to build a completely automatic detection and tracking system with MMTrack inference as its core.

In order to build a fully automatic detection and tracking system, we have to add capability to detect entering and exiting pedestrians to MMTrack. Detection of entering objects is usually accomplished with some form of detector. Examples of detectors used to initialize object tracking algorithms include a hand detector [14], a foreground object detector in the form of background subtraction [3], and a pupil detector [9]. We use HOG pedestrian detector to automatically initialize our tracker. Because HOG is also used as one of our appearance features, using it to initialize MMTrack inference has an additional benefit of not incurring additional computational cost.

On the other hand, our aim is to extract pedestrians’ full trajectories, starting from when they enter the scene until they exit the scene. A trajectory obtained by running MMTrack inference starting from a HOG detection is not guaranteed to be a full trajectory, as a HOG detection can occur at an arbitrary point in the pedestrian’s trajectory. As an example, if the detection used to initialize MMTrack captures the pedestrian when he is already in the middle of the image, the inference will only capture his trajectory starting from that point and his earlier trajectory will be lost. One way to overcome this is to initialize inference
only on HOG detections located at the image borders, assuming pedestrian entry points are located at the image borders. However, this method is dependent on the pedestrian being detected at the image borders - if HOG never detects the pedestrian at the image borders, his trajectory will be lost.

Instead, we propose to run MMTrack inference twice for each HOG detection, consisting of one forward pass and one backward pass. In the backward pass, we run MMTrack inference starting from the HOG detection in reverse temporal direction. Thus, the forward inference will extract the pedestrian’s trajectory from the HOG detection until he exits the scene, while the backward inference will extract the pedestrian’s trajectory from the HOG detection until he enters the scene. Combining the two resulting trajectories gives the pedestrian’s full trajectory. With this forward-backward mechanism, it now does not matter where the pedestrian is detected in the frame.

However, performing forward-backward inference on all HOG detections in every frame is definitely not computationally efficient. We instead sample the input every $M$ frames, and run forward-backward on all HOG detections on the sampled frames. Ideally, the value of $M$ should be small enough so that all pedestrians can be tracked, while still maintaining computational feasibility. Depending on the field of view and frame rate of the dataset, $M$ is set to be between 10 and 25 in our experiments. Another advantage of sampling the HOG detection is that it makes the system more robust to HOG misdetections. This is because a pedestrian’s trajectory can still be extracted even if he is not detected by HOG in one frame, as long as he is detected in some of the other sampled frames.

On the other hand, using sampled HOG detection to initialize the tracking may result in multiple trajectories being extracted for one pedestrian. To deal with these duplicate trajectories, a method is needed to associate trajectories belonging to the same pedestrian. Because the duplicate trajectories often have very similar temporal and spatial properties, we use an agglomerative clustering-based approach to group similar trajectories into a cluster. All the trajectories belonging to a cluster are then represented as one trajectory by the cluster mean. The details of our trajectory clustering is shown in Algorithm 1. It should be noted that unlike multi-target trackers that associate short trajectory segments to build complete trajectories [24, 15], our clustering method group complete trajectories in order to deal with duplicate trajectories.

Each iteration in the agglomerative clustering algorithm creates a new trajectory cluster and initializes it with a random trajectory not yet assigned to any clusters. The algorithm
Algorithm 1 Trajectory clustering algorithm

Require: Set of trajectories \( Tr \), StopThreshold \( th \), Bandwidth \( bw \)

\[
\text{Unassigned} \leftarrow Tr
\]

\[
\text{Clusters} \leftarrow \emptyset
\]

loop

\[
\text{Pop a random trajectory } u \text{ from } \text{Unassigned}
\]

\[
\text{Create integer array } v \text{ with size of } |Tr|, \text{ initialize as all zero}
\]

\[
\text{Create cluster } c \text{ with } \text{member} \leftarrow \emptyset, \text{ vote} \leftarrow v, \text{ and } \text{mean} \leftarrow u
\]

\[
\text{prevmean} \leftarrow \vec{0}
\]

\[
\text{loop}
\]

\[
\text{member} \leftarrow \emptyset
\]

\[
\text{loop}
\]

\[
\text{Get a trajectory } t \text{ from } Tr
\]

\[
\text{if } \text{dist}(t, \text{mean}) \leq bw \text{ then}
\]

\[
\text{if } t \in \text{Unassigned} \text{ then}
\]

\[
\begin{align*}
\text{Remove } t \text{ from } \text{Unassigned} \\
\text{member} \leftarrow \text{member} \cup \{t\}
\end{align*}
\]

\[
\text{vote}[t] \leftarrow \text{vote}[t] + 1
\]

\[
\text{end if}
\]

\[
\text{if All trajectories are processed then}
\]

\[
\text{Exit Loop}
\]

\[
\text{end if}
\]

\[
\text{end loop}
\]

\[
\text{prevmean} \leftarrow \text{mean}
\]

\[
\text{Recompute } \text{mean} \text{ from trajectories in } \text{member}
\]

\[
\text{if } \|\text{mean} - \text{prevmean}\|^2 \leq th \text{ then}
\]

\[
\text{Exit Loop}
\]

\[
\text{end if}
\]

\[
\text{end loop}
\]

\[
\text{Clusters} \leftarrow \text{Clusters} \cup \{c\}
\]

\[
\text{if } \text{Unassigned} = \emptyset \text{ then}
\]

\[
\text{Exit Loop}
\]

\[
\text{end if}
\]

\[
\text{end loop}
\]

loop

\[
\text{Get a trajectory } t \text{ from } Tr
\]

\[
\text{Assign } t \text{ to the cluster whose } \text{vote}[t] \text{ is the highest}
\]

\[
\text{if All trajectories are processed then}
\]

\[
\text{Exit Loop}
\]

\[
\text{end if}
\]

\[
\text{end loop}
\]
then iterates over all trajectories and sets all trajectories close to the mean as the cluster’s members. Based on the trajectories assigned to be the cluster’s member, the cluster’s mean is recomputed, and the algorithm iterates over all trajectories again. This process is repeated until no significant changes occur in the cluster’s mean. The algorithm also records an integer array \( \text{vote} \) to keep track of the number of times a trajectory is assigned to a particular cluster. This array is used as a tie-breaker in case one trajectory is assigned to multiple clusters.

The \( \text{dist}(\cdot) \) function is used in Algorithm 1 to determine how close two trajectories are. Consider two trajectories \( U = \{ y(u)^{(t_1)}, \ldots, y(u)^{(t_M)} \} \) and \( V = \{ y(v)^{(t'_1)}, \ldots, y(v)^{(t'_N)} \} \) which may have different starting or ending frames, or different lengths. Suppose the two trajectories are temporally overlapping during the time duration of \( t_m \) to \( t_n \), where \( t_m = \max(t_1, t'_1) \) and \( t_n = \min(t_M, t'_N) \), then the distance between the two trajectories is defined as:

\[
\sum_{t=m}^{n} \mathbb{1}[\|y(u)^{(t)} - y(v)^{(t)}\| \leq \text{Th}_{\text{dist}}] \quad (4.1)
\]

where \( \mathbb{1}[\cdot] \) is the indicator function and \( \text{Th}_{\text{dist}} \) is set to \( \sqrt{50} \) in our experiments.

We have shown how we extend MMTrack to handle track initialization and duplicate trajectories. An automatic detection and tracking system should also be able to terminate trajectories once the tracked object leaves the scene. We adopt a simple method of terminating a trajectory once the tracked object enters some predefined image border area. We define a rectangular area with width of either 100 or 150 pixels, depending on the dataset, from the image sides as the border area. In our beam search approximate inference procedure, the inference is stopped once all \( K \) hypothesis lie inside this area, thus terminating the trajectory. This method of terminating a trajectory works because our dataset all have exit points on the image border.

One thing to note is that our tracker system implicitly assumes that HOG detections are correct. It has no mechanism to verify whether an object detected by HOG is actually a pedestrian, and thus HOG false alarm will result in non-pedestrian object being tracked by the system.

Figure 4.1 shows the result of our automatic tracker system for one particular pedestrian. The image on the left shows MMTrack forward-backward inference results as red trajectories and the HOG detections as shadows with green borders. Sampling the input every 25 frames
results in ten HOG detections. We use these HOG detections to initialize our forward-
backward inference, resulting in ten trajectories obtained by MMTrack. This example is
relatively challenging as the target pedestrian is walking very near to another pedestrian,
and in fact some of the ten trajectories incorrectly track the neighboring pedestrian, with one
error occurring in the forward inference pass, and a couple others occurring in the backward
pass.

Three trajectory clusters are detected after clustering, and the membership of the clus-
ters are shown in the middle image of Figure 4.1, where the trajectories are colored according
to the cluster to which they belong. The blue cluster has the most members with eight tra-
jectories, and each of the green and purple clusters has one of the remaining two trajectories.
The means for the clusters are shown in the rightmost image, with the width of a cluster
trajectory being proportional to the number of trajectories belonging to that cluster.
One trend that we notice in our experiments is that clusters with many members tend to give better trajectories compared to clusters with few members. The example shown in Figure 4.1 also exhibits this trend. Insets in the rightmost image show the tracked objects at various points of the cluster means’ trajectories, with an inset’s border color indicating the cluster shown by the inset. It can be seen from the insets that the blue cluster (which contains eight trajectories) manages to track the pedestrian correctly, whereas the purple and green singleton clusters drift to a neighboring pedestrian at some point in their trajectories. Based on this observation, we remove clusters with only one member trajectory in the postprocessing step, as these clusters often correspond to outliers caused by mistracks. Removing the singleton clusters in our example leaves us with only the blue trajectory, which represents the correct trajectory.
Chapter 5

Experimental Results

We use the UBC Fireworks dataset [12] and S2.L1 dataset from the 2009 PETS competition [10] in our experiments, each providing our tracker system with different sets of challenges. The UBC Fireworks dataset is taken with top-down viewpoint and wide camera coverage, and it contains both daytime and nighttime sequences. With the UBC Fireworks dataset, we show the performance of our system in handling large pedestrian crowds and challenging illumination condition. The PETS dataset, on the other hand, is relatively short and contains sparser crowd compared to UBC Fireworks dataset. The challenges with PETS dataset are pedestrian scale change caused by the dataset’s oblique viewpoint, and full occlusion of pedestrians. We present quantitative and qualitative results on UBC Fireworks dataset, and qualitative results on PETS dataset.

5.1 UBC Fireworks Dataset

The UBC Fireworks dataset consists of clips recorded at $1440 \times 1080$ resolution using a stationary camera installed on top of a building in downtown Vancouver (an example frame is shown in the left side of Fig 3.1). Hence, a top-down view of a moderately crowded scene is captured with variety of moving objects typical to an urban setting present in the image. This includes cars, bikers and pedestrians. The amount of change in scale and pose is not significant but one needs to deal with background clutter and partial occlusions. The main challenges in the dataset are the presence of occasional crowded blobs of moving pedestrians that introduces many potential distractors and significant background change that occurs when people move from sidewalk to street area and vice versa.
Because the dataset does not contain significant scale variation, we assume the pedestrians to be of a certain fixed size, which is manually determined based on typical pedestrian size in the dataset. Histogram distance and appearance template features are then computed over a bounding box of that size.

5.1.1 Quantitative evaluation

We use 10 manually-labelled trajectory sequences for training and 22 other manually-labelled sequences for quantitative evaluation with the labels being used as ground truth. Both the training and test sequences contain easy, moderate and hard sequences ranging from a solitary person going through the scene to a pedestrian walking within a crowd.

We compare the results of our tracking system with Collins-Liu tracker [5] and MILTrack [2]. To gain insight into the importance of having a combination of all the features, we also provide the results of our algorithms when some of the features are turned off. Note that we learn different sets of parameters for each feature combination.

The procedure for extracting trajectories for quantitative evaluation is similar to one described in Chapter 4. We first initialize a tracker from a manually-specified HOG detection. We then run the tracker forward and backward in time in order get a complete trajectory regardless of the initial position provided by the HOG detection. The tracker is terminated once it is within a certain number of pixels from the image borders. Note that since each HOG detection is manually picked to correspond to different pedestrians, the trajectory clustering step is not performed.

One measure we use to evaluate our tracker is the average pixel error measure, which simply measures the average pixel distance between corresponding points in the ground truth and the resulting trajectory. Besides the pixel error measure, we also use two other performance measures: Correct Detected Track and Closeness of Track [33]. Correct Detected Track (CDT) indicates the number of correct trajectories. A track is defined as a CDT if the amount of spatial and temporal overlap with the ground truth exceed thresholds $T_{ov}$ and $TR_{ov}$ respectively, where $T_{ov}$ and $TR_{ov}$ are both set to 0.5 in our experiments. This roughly means that at least half of a CDT must temporally coincide with its ground truth, its length cannot be less than half of its ground truth, and the average spatial overlap must be at least 0.5. Closeness of Track (CT) is defined as the average spatial overlap between a ground truth bounding box and a system track bounding box in the temporally coincident portion of the track. Its value ranges from 0 to 1, with 1 indicating that the track is exactly
the same as the ground truth in the temporally coincident section of the track.

Table 5.1: Tracking results on 22 UBC dataset test sequences.

<table>
<thead>
<tr>
<th>Tracker</th>
<th># CDT</th>
<th>Avg CT</th>
<th>Avg Pixel Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMTrack: All</td>
<td>21</td>
<td>0.66</td>
<td>7.01</td>
</tr>
<tr>
<td>MMTrack: HOG+Hist</td>
<td>10</td>
<td>0.47</td>
<td>14.40</td>
</tr>
<tr>
<td>MMTrack: HOG+Template</td>
<td>14</td>
<td>0.52</td>
<td>22.24</td>
</tr>
<tr>
<td>MILTrack [2]</td>
<td>19</td>
<td>0.61</td>
<td>19.87</td>
</tr>
</tbody>
</table>

As can be seen from Table 5.1, our proposed tracker achieves comparable performance to MILTracker [2], and is better than Collins-Liu tracker [5] in this dataset. One can explain this promising performance by reasoning about our system having different cues that describe the desired pedestrian as well as distinguishing it from background and our principled way of cue combination. More specifically, HOG feature helps the tracker eliminate areas belonging to non-pedestrian objects, static histogram distance feature provides rough description of the pedestrian and helps alleviate drift whereas appearance templates provide finer levels of a pedestrian model, with the previous frame appearance template allowing some degree of adaptability to appearance change over time.

Removing some of the features significantly reduces the performance of our tracker, indicating that the combination of HOG, histogram distance, and template appearance features is essential in achieving good performance. An example illustrating the importance of our cue combination strategy is shown in Fig 5.1. The image shows the results of MMTrack with different feature combinations. The leftmost image shows the trajectory obtained with HOG + histogram distance features, the middle image shows the trajectory obtained with HOG and appearance template features, and the rightmost image shows the trajectory obtained with all three features. A red inset in each of the images shows the tracked objects throughout the trajectory, which is also superimposed in the image as shadows.

As can be seen in the red inset, our tracker with only HOG and histogram distance feature drifts to a nearby pedestrian at some parts of the track, because there is a pedestrian with similar color to the tracked person nearby. On the other hand, HOG and appearance template drifts to a background area at the boundary between the pavement and the street, probably because the sum of absolute difference measure used in the appearance template is sensitive to the significant change in background pixels that occurs at this boundary. The combination of HOG, color histogram distance and appearance templates manages to track...
5.1.2 Qualitative evaluation

We use three UBC Fireworks dataset clips for our qualitative evaluation. The three video clips are recorded at 25 frames per second, with durations of 6m:18s, 6m:46s, and 3m:20s. The clips with lengths of 6m:18s and 6m:46s are taken in daytime, whereas the clip with length of 3m:20s is taken in nighttime. We use the same weight parameters obtained in the quantitative evaluation section in all our experiments. To generate the trajectories, we run our automatic detection and tracking system as described in Chapter 4.

However, due to the clips’ long durations (a six-minute clip corresponds to roughly 9000...
frames), complete clustering of pedestrians’ histograms needed by the histogram distance feature (described in Fig. 3.2) is not feasible. To overcome this issue, we divide the frames into 500-frames non-overlapping blocks, depicted as blue rectangles in Fig. 5.2. Because most pedestrians in UBC Fireworks dataset travel through the scene in less than 500 frames, choosing block size of 500 frames allows us to capture the whole trajectory of most pedestrians. However, if a pedestrian enters the scene very late in his 500-frame block, it is possible that his trajectory will cross the block boundary. To fully capture the appearance of such pedestrian, each block’s histogram clustering process also uses histograms from the next and previous blocks. Thus, the histograms are clustered over 1500-frame period, shown by red rectangles in Fig. 5.2. After the appearance feature is generated, forward-backward inference is started from HOG detections in each block. Note that the inference can cross the block boundary. Finally, trajectories from all blocks are combined and clustered to give the final trajectories, shown as green rectangles in Fig. 5.2.

![Figure 5.2: Block division method used to make inference feasible on long sequences.](image)

All results are available at www.cs.sfu.ca/research/groups/VML/mmresults.html. The system shows good performance on the daytime clips, even on crowded conditions, as can be seen in Fig. 5.3. Two main sources of errors in our trackers are HOG false alarms causing non-pedestrian objects such as cars to be tracked, and pedestrian full occlusion that occurs when a pedestrian walks under a tree.

The main challenges with the nighttime sequence are the low contrast and the lack of color information due to insufficient lighting. Because our histogram distance and appearance template features are based on color information, these issues reduce the effectiveness of
Chapter 5. Experimental Results

Figure 5.3: Sample tracking results on crowded conditions.

The nighttime sequence result shows how these issues affect the tracking system. Although the tracker still manages to track many pedestrians correctly, there are more cases of...
tracker drift and mistrack. At times, noisy HOG feature causes tracker failure in distinguishing between background area and pedestrians, resulting in the tracker drifting to pavement or road area with relatively strong HOG response. Additionally, HOG misdetections also cause the tracker to miss a number of pedestrians, especially in the darker sections of the image where some pedestrians are never detected by HOG.

5.2 PETS 2009 Dataset

The second dataset that we use is the S2.L1 dataset taken from PETS 2009 competition. The dataset consists of a 794-frame video recorded at about 7 frames per second, taken from a pedestrian path at a university campus. Unlike the UBC Fireworks dataset, this dataset is taken at a more oblique angle, and thus there are significant scale variations due to perspective effect. The viewpoint also introduces occlusion issue, with occasions where pedestrians are completely occluded either by a background object or other pedestrians. Another challenge of the dataset is the appearance similarity of many pedestrians in the frames.

Scale variation affects extraction of both the appearance model and the motion model. As mentioned in Section 5.1, we extract color histogram distance features for UBC Fireworks dataset by fitting a fixed-size window to each pixel, and extracting histograms for the nine subsections of the window. Obviously, the same procedure cannot be applied to a dataset with scale variations - instead of fitting a fixed sized window to each pixel, a variable-sized window should be assigned to each pixel, where the size of the window is computed as a function of the pixel’s location. Similar process should also be used for extraction of the template difference appearance features. Further, the motion model should also modified to take into account perspective effect - for example, a pedestrian traveling three pixels at the far end of camera field of view is traveling considerably farther than another pedestrian traveling the same number of pixels in front of the camera.

To handle scale variation and perspective effect, we utilize the camera calibration information provided with PETS 2009 dataset. The camera calibration can be used to map 2D pixel coordinates (with height information) to 3D world coordinates and vice versa. The calibration is set up such that the \(Z = 0\) world coordinate plane indicates the ground plane. Thus, if we assume a fixed pedestrian height in meters, we can compute the pixel height of a pedestrian’s bounding box centered at any particular pixel. We get the pixel width of the
CHAPTER 5. EXPERIMENTAL RESULTS

bounding box by using a fixed height:width ratio. The bounding box measurements for all pixels can then be used to extract appropriate appearance features. Thus, for any hypothesized pedestrian location, instead of using a fixed-size bounding box to extract appearance features as we do with the UBC Fireworks dataset, we use the measured bounding box size for that pixel location. For color histogram distance feature, this means computing the histograms over the nine sections of the measured bounding box. For appearance template features, both the template and the hypothesis are scaled to a fixed resolution, at which the sum of absolute difference is computed.

The camera calibration information is also used to handle perspective effect on the motion model. With the calibration information and an assumed fixed pedestrian height, we can determine how far two pedestrian center pixels are in world coordinate. We use this information to estimate the maximum distance a pedestrian can travel in two frames, and discretize this distance to the same number of bins as the motion model we use in the UBC Fireworks dataset.

Fig. 5.5 shows the camera calibration results. The green bounding boxes show pedestrian’s sizes measured with camera calibration information, assuming pedestrian height of 1.6m and height:width ratio of 4:1. As can be seen from the figure, the computed bounding box sizes align well with actual pedestrian sizes. The ellipses surrounding the bounding
boxes show the calibrated motion model assuming that the farthest distance a pedestrian can travel in two frames is 1.8m. Similar to the motion model used for UBC Fireworks dataset, the motion model is discretized into four bins, indicated by four concentric ellipses surrounding each bounding box.

Figure 5.6: Tracking results under occlusion on PETS dataset.

We use the same weight parameters obtained from our UBC experiment on PETS dataset. Generally, the tracker can track most pedestrians when they are not occluded. Two main causes of occlusions in the dataset are the signpost located in the middle of the image and other pedestrians. The rows of Fig. 5.6 show three sample tracking results under occlusion. On the first row, a pedestrian is fully occluded when he walks behind the signpost, and even though the tracker stops following the pedestrian when he is fully occluded (middle column), the tracker manages to find the target once he reappears. Similar situation occurs in the example shown in the second row. However, in this example the tracker drifts to another pedestrian when the target is fully occluded, as the distracting pedestrian has higher trajectory score compared to the occluded target. Because the distracting pedestrian walks in opposite direction from the target, the tracker becomes more distant to the target, and the motion model makes it unable to reattach itself to the target when he reappears.
Tracking drift may also occur when the target is fully occluded by another pedestrian with a similar appearance, as shown in the third row of Fig. 5.6.

The video containing complete tracking results on the PETS 2009 dataset is available at www.cs.sfu.ca/research/groups/VML/mmtrack/petsresult.avi. In the video, 44 trajectory clusters are displayed sequentially in descending order according to the ratio of number of members to length of cluster mean. The results show promising performance, with only one mistrack in the first ten trajectories, and five mistracks in the first twenty trajectories. Two full-trajectory tracking results are shown in Fig. 5.7. In the figure, target pedestrians are indicated by red bounding boxes, the tracker's resulting trajectories are shown in red trajectories, and snapshots from various points during the trajectories are shown in the blue insets. In the top image, the tracker managed to track the target when he was partially occluded by a signpost, but drifted when the target was occluded by a similar-looking pedestrian. The bottom image shows a successful long-duration tracking, even on occasions of full occlusion. In this example, the target pedestrian's appearance is fortunately very different from other pedestrians he interacts with, and the tracker can still find him after occlusion as he never moves farther than the limit of the motion model.

It should be noted that our MMTrack system is a single-target tracker, and thus it has no capability to reason about full occlusions that occur in this dataset. Successful tracking in the presence of full occlusions as shown in some of the above examples is largely conditional upon the target having a distinct appearance and him not moving outside the boundary imposed by the motion model, not because of an inherent mechanism in the algorithm to handle occlusion. We believe that rather than adding occlusion handling capability to a single-target tracker, better result can be achieved by using multi-target tracking framework. In contrast to single-target trackers that assign a trajectory to each target without considering other objects, multi-target trackers jointly consider the state of all targets in determining their trajectories. This joint reasoning allows multi-target trackers to handle full occlusions. A single object tracker can be incorporated into a multi-target tracker by using the single-target tracker to generate short trajectories or 'tracklets', and using data association to sequence the tracklets to longer trajectories [15, 1, 24].
Figure 5.7: Sample tracking results on PETS dataset.
Chapter 6

Discussions and Conclusions

In this thesis, we introduce MMTrack, an offline single-target tracking system that employs a max margin learning criterion to combine different sources of information effectively. MMTrack views the tracking task of predicting sequence of interdependent locations as a structured prediction problem, and solves the resulting optimization problem with Structured SVM learning. This formulation allows MMTrack to use different feature classes, including descriptive features such as color histogram information and color templates, and discriminative features such as Histogram or Oriented Gradient (HOG) feature. By modeling the target object’s dynamics with a chain-structured CRF, MMTrack also allows learning of the target’s movement priors jointly with its appearance features, essentially combining both appearance and motion models within a unified framework.

This thesis presents an application of MMTrack as a pedestrian tracker by explicitly selecting features that are specific to pedestrians such as the HOG score map. This thesis then shows how this tracker can be extended to an automatic pedestrian detection and tracking system by addressing the issue of entry/exit detection and clustering of duplicate trajectories. An extension to handle varying pedestrian sizes is also proposed. This extension adapts the window size used for feature extraction and the coverage of the motion parameters according to camera calibration information. Quantitative and qualitative results are shown on UBC Fireworks and PETS 2009 datasets.

Although MMTrack is used for pedestrian tracking in this work, we believe that our framework is general and can be used to track other object categories as long as the features describe the object of interest well. We also believe that MMTrack can also be used to combine other types of cues other than the ones we use.
6.1 Future Work

Although MMTrack achieves good performance on UBC Fireworks dataset, there are still instances of tracking failure caused by drastic change in background. A framework that emphasizes different features for different part of the image, for example a method that emphasizes motion model on darker areas and emphasizes appearance models on areas where pedestrians are clearly visible, may be able to alleviate this issue. This framework can be built by designing a more sophisticated learning framework that learns sets of weights for different areas in the scene.

There are also a number of tracking failures caused by track hijack that happen especially when a pedestrian is walking close to another pedestrian with similar appearance. Additionally, the single-target tracker nature of MMTrack also means that it has no inherent mechanism to handle occlusion, as can be observed from a number of tracking errors occurring in the PETS dataset. A possible extension to address these issues is to embed MMTrack to a multi-target tracking framework, similar to what is done by Yuan Li et al. [15]. In a multi-target tracking framework, MMTrack can be used to generate short tracklets that will be used as input to a data association-based tracker to generate complete trajectories.
Bibliography


