Tracking People and Recognizing their Activities

by

Deva Kannan Ramanan

B.C.E. (University of Delaware) 2000

A dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy
in

Engineering—Electrical Engineering and
Computer Science

and the Designated Emphasis
in

Communication, Computation, and Statistics

in the

GRADUATE DIVISION
of the
UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor David Forsyth, Chair
Professor Jitendra Malik
Professor Michael Jordan
Professor Stephen Palmer

Fall 2005

The dissertation of Deva Kannan Ramanan is approved:

Professor David Forsyth, Chair — Date

Professor Jitendra Malik — Date

Professor Michael Jordan — Date

Professor Stephen Palmer — Date

University of California, Berkeley

Fall 2005

Tracking People and Recognizing their Acitivies

by

Deva Kannan Ramanan

Abstract

Tracking People and Recognizing their Acitivies

by

Deva Kannan Ramanan

Doctor of Philosophy in Engineering—Electrical Engineering and Computer Science

University of California, Berkeley

Professor David Forsyth, Chair

An important, open vision problem is to automatically describe what people are doing in a sequence of video. This problem is difficult for several reasons. Firstly, one needs to determine how many people (if any) are in each frame and estimate where they are and what their arms and legs are doing. But finding people and localizing their limbs is hard because people (a) move fast and unpredictably, (b) wear a variety of different clothes, and (c) appear in a variety of poses. Secondly, one must describe what each person is doing; this problem is poorly understood, not least because there is no known natural or canonical set of categories into which to classify activities.

This thesis addresses a number of key issues that are needed to build a working system. Firstly, we develop a completely automatic person tracker that accurately tracks torsos, arms, legs, and heads. Our system works in two stages; it first (a) builds a model of appearance of each person in a video and then (b) tracks by detecting those models in each

frame ("tracking by model-building and detection"). By looking for coherence across a video, our system can also build models of unknown objects. We use it to build articulated models of various animals; these models can be used to detect the animals in new images. This way, we can think of our tracking algorithm as a system that builds models for object detection.

We then marry our tracker with a motion synthesis engine that works by re-assembling pre-recorded motion clips. The synthesis engine generates new motions that are human-like and close to the image measurements reported by the tracker. By using labeled motion clips, our synthesizer also generates activity labels for each image frame ("analysis by synthesis"). We have extensively tested our system, running it on hundreds of thousands of frames of unscripted indoor and outdoor activity, a feature-length film ('Run Lola Run'), and legacy sports footage (from the 2002 World Series and 1998 Winter Olympics).

Professor David Forsyth, Chair Date

Acknowledgements

I owe a great thanks to my advisor, David Forsyth. Anyone who has met him is struck by his intelligence, wit, and generosity. Its a pleasure to have known him and I consider myself lucky for having been his student.

Thanks to Jitendra Malik as well; his classic one-liners about vision ("real men use one camera" is my favorite) are always clever and insightful. Thanks to Michael Jordan; his statistical learning theory class shaped a large part of this thesis. Thanks to Steve Palmer for serving on my committee and providing quite useful suggestions.

Thanks to my collaborators; Andrew Zisserman and Kobus Barnard. Spending the summer at Oxford with Andrew was a wonderful experience, not least because of the intelligent and friendly folks there; Josef Sivic, Fred Schaffalitzky, Mark Everingham, Rob Fergus, Manik Varma, and Maud Poissonier to name a few.

Thanks to the Berkeley Vision group for many enlightening chats: Andrea Frome, Stella Yu, Andras Ferencz, Charless Fowlkes, David Martin, Erik Miller, Greg Mori, Alyosha Efros, Xiaofeng Ren, Michael Maire, Hao Zhang, Ryan White, Roger Bock, John Flynn, and Alex & Tamara Berg. Thanks to the Graphics folks as well: Okan Arikan, Leslie Ikemoto, Ashley Eden, Tony Lobay, Adam Kirk, Jordan Smith, Pushkar Joshi, Adam Bargteil, Bryan Feldman, Hayley Iben, Chen Shen, Tolga Goktekin, Ravi Kolluri, and Lillian Chu. Thanks to Alyosha Efros for his 'unofficial' mentoring and generosity.

Thanks to my undergraduate advisor, Ken Barner, for introducing me to research (and

matlab). Thanks to Ruth Gjerde for making the front office feel a bit like home. Thanks to my roommates, Dave, Bryan, and Josh (and many others) for making Ward Street tons of fun, making the past five years memorable, and making graduate school (to my surprise) just as fun as college. Thanks to Keerthi for heart-ing (and putting up with) a nerd. Thanks to Durga and Phillip for always lending a hand. And thanks to my parents for everything.

To my parents

Contents

List of Figures

List of Tables

Chapter 1

Introduction

One of the grand goals of artificial intelligence is to build a true "autonomous agent"; something that can go out in the real world and interact with its environment, interact with other objects, and perhaps most importantly, *interact with people*.

This dissertation focuses on building systems that understand people through videos and images. Given that one wants to build systems that understand people, an immediate question is: what level of understanding is needed? Many approaches treat people as blobs or rectangular image regions (Figure 1.1); "understanding" in this context simply means tracking the rectangle as it moves across an image. Such information by itself can be useful, for example, to architects interested in how people move around in public spaces.

But people are more than a blob of pixels; they are animate beings that gesture with their entire bodies. People articulate their limbs to communicate and express themselves. This dissertation focuses on *kinematic tracking*, where one explicitly models how limbs

Figure 1.1: What level of people-understanding do we need? On the **left**, we show a frame from a webcam pointed at Sproul Plaza in Berkeley. Since people are small, most algorithms analyzing this form of data treat people as *blobs*. On the **right**, we show a news photo from the 2004 World Series. From this scale, a proper analysis should capture how the bodies are *articulated*. This dissertation focuses on building systems that understand the articulations of people from a video.

deform and articulate over time.

A working articulated tracker has numerous applications. It can be used for video summary/mining – one could summarize daily activities in public areas. It can be used for Human-Computer Interaction (HCI) – gesturing interfaces could enable smart offices and smart homes. Tracking would also enable video motion capture – the recovered motion could be used to later animate a virtual character. Finally, one could track people for surveillance purposes – from medical analysis of patients to automatic flagging of suspicious behavior in high security areas. Indeed, because of the many potential applications, people-tracking has been a core challenge in the vision community for over 25 years. Although those years showcased major improvements in algorithmic technique (notably, the use of particle filters), most resulting systems require some limiting assumption (such as no fast limb movement, controlled backgrounds, or manual initialization; see [34, 72]).

1.1 The approach

We cast tracking in a probabilistic reasoning framework, and apply techniques from machine learning to develop algorithms. By using formal probability models, we can take advantage of principled learning and inference algorithms. Often, simply writing down the model is a useful exercise because it makes explicit the hidden assumptions in standard approaches. While developing our people tracker, we encounter a new model for uncertainty in video data, with implications not just for tracking but also for object detection and video encoding.

An important question when building a practical system is: how does one know that it works? We produce a novel evaluation methodology and apply it on a massive scale, evaluating our articulated tracker on *hundreds of thousands* of frames (several orders of magnitude more than previous systems). We demonstrate that we *can* fairly accurately track people that are moving fast and that are surrounded by cluttered backgrounds without manual initialization.

1.2 Thesis overview

In Chapter 2, we give an overview of past work on person tracking. Most approaches for tracking people (and other objects) track some model over time. The basic observation behind this thesis is that *model-based tracking is easier with a better model*. Thus the process of tracking becomes intertwined with the process of building a good model for the object being tracked. After learning a good model from a video, we then track by detecting it in each frame. In Chapter 3, we introduce a tracking algorithm that automatically builds

models by looking for *coherence* across a video. Once the models are built, they can be used to find the object in the original video as well as in novel images. We demonstrate our algorithm on videos and images of animals. In Chapter 4, we apply the model-building algorithm to the task of tracking people. We develop an algorithm that automatically builds people-models *opportunistically* from select frames which are convenient. We also show that building *discriminative* models is quite helpful. We demonstrate the resulting people-tracker on hours of commercial and unscripted video. Finally, in Chapter 5, we show an application of our people tracker; activity recognition. By tracking the configuration of arms and legs over time, we can recognize whether someone is waving, jumping, catching a ball, etc.

Chapter 2

Background

A practical person tracker should meet a veritable laundry-list of demands. It should self-start, recover configurations of arms and legs, track accurately for long sequences; track independent of activity, be robust to drift, track multiple people, track through brief occlusions, and be computationally efficient. It should also avoid using background subtraction; we want to track people who happen to stand still against backgrounds that happen to move. Most current trackers fail to meet most of these demands. In this chapter, we look at the current state of the art and argue that the underlying model most trackers follow might be at fault.

2.1 Hidden Markov Models

By far the most common approach is to use a hidden Markov model (HMM), where the hidden states are the poses (X_t) to be estimated, and the observations are images (Im_t) of

a video sequence (see [18, 26, 40, 42, 45, 83, 93, 95, 100, 101, 114]). For now, let us think of pose X_t as a vector of 2D joint positions, though we will later look at 3D poses. Standard Markov assumptions allow us to decompose the joint into

$$\Pr(X_{1:T}, Im_{1:T}) = \prod_t \Pr(X_t|X_{t-1}) \Pr(Im_t|X_t),$$

where we use the shorthand $X_{1:T} = \{X_1, \ldots, X_T\}$. Tracking corresponds to inference on this probability model; typically one searches for the maximum a posteriori (MAP) sequence of poses given an image sequence

$$\begin{aligned} \hat{X}_{1:T} &= \operatorname*{argmax}_{X_{1:T}} \Pr(X_{1:T}|Im_{1:T}) = \operatorname*{argmax}_{X_{1:T}} \Pr(X_{1:T}, Im_{1:T}) && (2.1) \\ &= \operatorname*{argmax}_{X_{1:T}} \prod_t \Pr(X_t|X_{t-1}) \Pr(Im_t|X_t). && (2.2) \end{aligned}$$

Four issues need to be addressed: (a) What is the inference algorithm? (b) What is the dynamic model, $\Pr(X_t|X_{t-1})$? (c) What is the image likelihood model $\Pr(Im_t|X_t)$? (d) How do we initialize the track (obtain $\hat{X}_1$)?

2.1.1 Inference Algorithm

The space of poses (the domain of X_t) is too large to discretize and explicitly search by dynamic programming, though approximate strategies exist [51]. Early approaches originating from O'Rourke and Badler in 1980 and Hogg in 1983 used classic AI search techniques [45, 82]. Most approaches perform inference by variants of kalman filtering [18, 42, 45, 95]

or particle filtering [9, 11, 26, 58, 67, 68, 83, 100, 101, 106, 108, 114]. One uses the pose in the current frame and a dynamic model to predict the next pose; these predictions are then refined using image data. Particle filtering uses multiple predictions — obtained by running samples of the prior through a model of the dynamics — which are reweighted by comparing them with the local image data (the likelihood).

2.1.2 Motion Model

Humans move in very specific, but hard-to-model ways. Consider the moving light displays of Johannson [55]. One can perceive human movement simply by observing lights positioned at limb joints, but such precepts disappear once joints stop moving. This suggests that motion might play a role in identifying/tracking people. One method of representing such dynamics is to learn a model from motion capture data [100, 101], but this often requires choosing the specific motion (walking,running, etc.) *a priori*. Alternatively, one can build a dynamic model that selects a motion online from a set of models [2, 83]. Such finely-tuned models are valid in some settings (e.g., HCI, when a user is performing a gesture from a limited vocabulary), but are difficult to apply in general (e.g., video motion-capturing a novel performance). Obtaining a generic parametric model of human motion is an active area of research in the graphics community [5, 59, 61], but until one is found, we believe that general purpose trackers should be limited to simpler models (such as constant velocity).

2.1.3 Image Likelihood

One needs a person model to compute an image likelihood $\Pr(X_t|Im_t)$. Typically one uses a template encoding appearance [41, 51, 74, 112], local motion [109] or both [117]). The template can be updated on-line or learned off-line. Constructing a good likelihood model is hard because it must be general enough to represent bodies in various poses and clothes but specific enough not to be confused by background clutter. A common approach is to evaluate the model likelihood only at certain image locations (e.g., particle filters restrict themselves to predictions returned by a motion model). Such a scheme is susceptible to drift because of clutter; if the predicted arm position lies on an "arm-like" pole, the track will drift. One partial remedy is to perform local searches of the likelihood [26, 107]. Another is to globally search the entire likelihood; this is computationally taxing unless one makes simplifying assumptions. Algorithms that track by detection search the likelihood at each frame and keep only likely poses [41, 51, 70, 74, 79, 109, 110, 112, 117]. Such an approach to tracking as detection is attractive because the resulting trackers re-initialize themselves every frame. This makes them robust to drift and able to self-start. However, these approaches require templates that reliably detect people, which is a non-trivial task.

2.1.4 Initialization/Detecting People

Most previous algorithms require hand-initialization, which is clearly unacceptable for many applications. The task of initializing a tracker and constructing a good likelihood model are similar; both require a good template that can detect people. Good results have been

Figure 2.1: To build a usable people tracker, one needs a good likelihood model and a method to automatically initialize tracks – both require a template that can detect people. Detecting people is difficult because of the large variation in appearance due to body shape and clothing (**left**) and articulations of the body (**middle**). Detection is further complicated by cluttered backgrounds (**right**).

demonstrated when skin regions are reliably detected [62], for clutter-less backgrounds [53, 74], and for background subtracted images (see the large body of literature on silhouette-based methods [1, 31, 43, 48, 99, 105, 126]). A solution for reliably finding people in general images is yet to be found. This detection problem seems very hard (more so than, say, face detection) for two reasons: people vary greatly in appearance (due to individual body shape, clothing, and articulations), and there exists lots of limb-like clutter in the background (Figure 2.1). The former suggests enormous intra-class variation and the latter implies little inter-class variation. A common strategy to handle the pose variation is to use a bottom-up model of body parts [69, 73, 75, 96, 102]. These methods still require likelihood models for each part. Constructing part models for unclad people is straightforward [36], but clothing considerably complicates matters.

2.2 Data Association

Before we go further, it is useful to examine why HMMs are so ubiquitous in tracking literature. This phenomenon can be traced back to the well-established success of HMMs

in the radar community [10]. In that continuous-domain context, the traditional algorithm for inference on a HMM is a kalman filter [71]. When tracking an airplane's position where the observations are global positioning system (GPS) reports, a kalman filter might work quite well. However, this situation is fundamentally different when observations are images from a video; the bulk of computation goes toward *data association* [10]: which part of the image comes from the object, and which does not? A better comparison for radar data is when we observe a million GPS measurements ($\approx$ the number of pixels in an image) at each time instant, and we must simultaneously track our object *and* determine which is the true measurement. Here, we want a tracking algorithm that, instead of simply smoothing a position measurement (as a Kalman filter does), identifies *what* measurement to smooth. Particle filters use the dynamic model $\Pr(X_t|X_{t-1})$ to identify the correct measurement; a motion prediction tells us where to look in an image. This approach, though computationally efficient, is susceptible to drift when tracking in a cluttered background. The alternative is to do data association using the likelihood model $\Pr(Im_t|X_t)$, but this requires an accurate model that produces a low likelihood for background regions. Indirect methods of doing this include background subtraction and skin detection, but again, these are only valid in restricted situations.

Alternatively, one could build a likelihood model that directly identifies what image regions are people. Assume we have a video of a single person (Fred); then ideally, we want a "Fred" detector rather then a generic person detector; i.e. a likelihood model $\Pr(Im_t|X_t)$ that captures the fact that Fred is wearing a red shirt and has blond hair. This model *can*

perform data association since it can quickly ignore the pixels which are not red. In certain situations, one could obtain such models off-line (consider a soccer match where the team uniforms are known). In general, we will not know who will be in a video and so will need to build models on-the-fly. One mechanism for doing this is to augment the state variable X_t with an appearance term A_t that captures the color of a shirt [100, 101] (algorithms based on optical flow often implicitly do this [18]). Such Markovian appearance models are attractive because standard inference algorithms still apply. But in practice, they suffer from two severe limitations; (a) one cannot automatically initialize them because one does not know appearance *a priori*, and (b) the resulting trackers tend to drift. Note that in some sense these are similar problems, since one method of ensuring a robust tracker is to continually re-initialize it.

2.3 Why do Markovian models fail?

Consider the toy case of building a template-matching blob tracker for a torso (Figure 2.2). The underlying model is a HMM where the hidden state at each frame is

$$X^i = \begin{bmatrix} P^i \\ A^i \end{bmatrix},$$

where P^i is (x^i, y^i) position of the torso blob and A^i is the appearance. Throughout this thesis, we consider alternate encodings of appearance. For the time being, we can think of A^i as a vectorized patch of pixels. The observations are images from a video sequence. Let

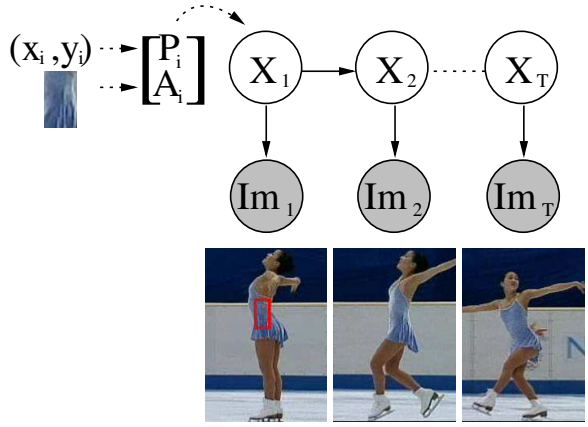


Figure 2.2: Hidden Markov Model for a blob tracker. Typically, the hidden state X_t is comprised of the blob position P_t and appearance A_t, while the observation is an entire image frame Im_t.

us assume the following probability models:

$$\Pr(X_t|X_{t-1}) \propto e^{-||X_t - X_{t-1}||^2} \tag{2.3}$$

$$\Pr(Im_t|X_t) \equiv \Pr(Im_t(P_t)|A_t) \propto e^{-||Im_t(P_t) - A_t||^2}, \tag{2.4}$$

where we have ignored constants for simplicity. Our motion model $\Pr(X_t|X_{t-1})$ is Brownian motion. Recall that our object state X_t encodes both blob position $P_t = (x_t, y_t)$ and blob appearance A_t. Our likelihood model favors pixel positions P_t at which the encompassing image patch $Im_t(P_t)$ looks like the current blob template A_t. Note this likelihood only models the image at the blob position, ignoring the background (a common assumption in the tracking literature). We show in Chapter 4 that building discriminative likelihood models exploiting background information do better.

If we perform inference on the model from Figure 2.2, the resulting track tends to drift; see Figure 2.3. This is because there is no constraint in the model that requires the

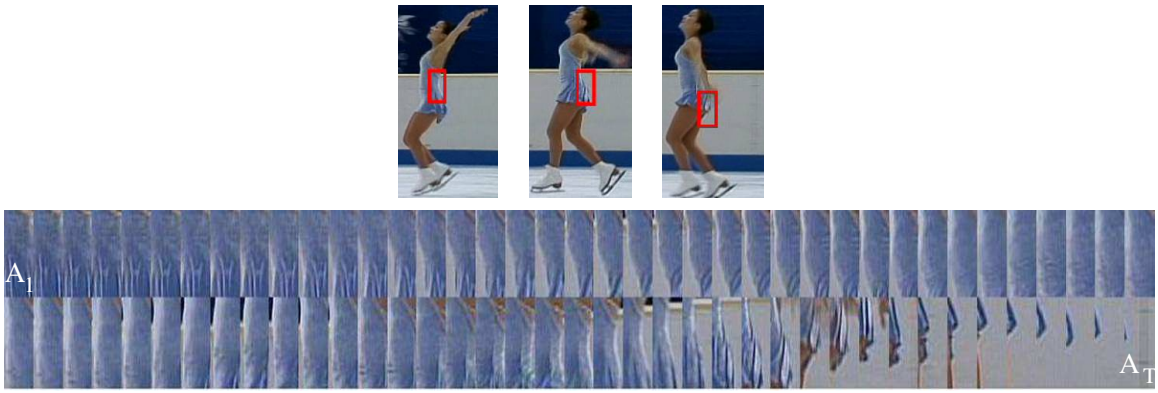


Figure 2.3: We perform inference on the model from Figure 2.2 with a standard template blob-tracker. Even with manual initialization, the model tends to drift because there is no constraint that the appearance at the end of the track A_T should be similar to the initial appearance A_1.

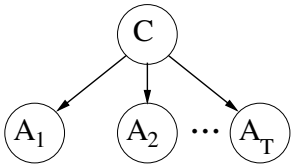


Figure 2.4: An alternative to the Markov model of appearance. Here, we model the views A_t as i.i.d. samples from a canonical appearance C. This model forces the first A_1 and last views A_T to be similar.

appearance at the last frame A_T to be similar to the appearance from the first frame A_1. Small errors in the estimated appearance A_t accumulate; for example, the torso becomes temporarily occluded by the arm, and so the track drifts to compensate. We can avoid the drifting behavior with a constant model of torso appearance A_t:

$$\Pr(A_t|C) \propto e^{-||A_t - C||^2}.$$

Here, the torso image patch at each frame A_t is modeled as a i.i.d sample from a Gaussian centered at an underlying "true" torso patch C. With a constant appearance

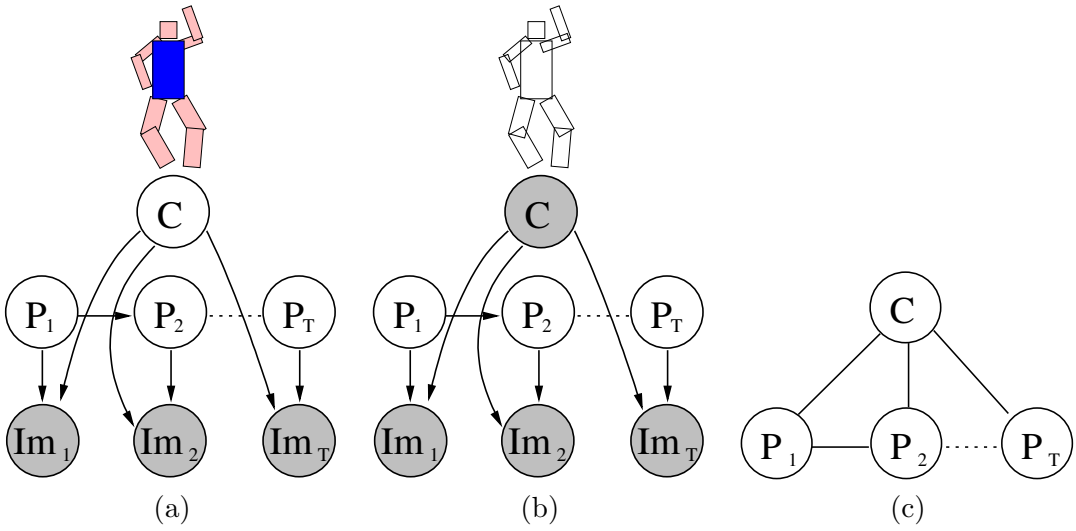


Figure 2.5: Above (**a**), we show resulting model from incorporating the constant appearance model from Figure 2.4 with the HMM from Figure 2.2. Blob position P_t still follows a Markovian model, but the image likelihoods are tied together by the common appearance C. If the appearance C is known, then our model reduces to a standard HMM (**b**); model-based tracking algorithms take this approach. Since C is given *a priori*, the appearance must be invariant to clothing; a common approach is to use an edge template [41, 51, 74, 112, 114]. By treating C as a random variable, we *build* a template specific to the particular person in a video as we track his/her position P_t. We show an undirected model in (**c**) that is equivalent to (**a**).

model, we force our initial appearance A_1 and final appearance A_T to be similar – a blue shirt stays blue, even when occluded by an arm.

Note that A_T and A_1 can still differ – indeed, two samples from the same gaussian can be arbitrarily far away. The *variance* of A_T is bounded in a constant model $var_c(A_T|A_1) = 2$, while it grows linearly with time in a Markov model $var_m(A_T|A_1) = T - 1$ (assuming a noise model with unit variance). This is why Markov-based trackers typically work only on short sequences; if we wait long enough, the tracked object can change arbitrarily.

2.4 Constant Appearance Models

When we insert our constant appearance model into the HMM from Figure 2.2, the image likelihoods become linked by the canonical appearance C:

$$\Pr(P_t|P_{t-1}) \propto e^{-||P_t - P_{t-1}||^2} \tag{2.5}$$

$$\Pr(Im_t|P_t, C) \propto e^{-||Im_t(P_t) - C||^2}. \tag{2.6}$$

Equation 2.6 (shown graphically in Figure 2.5-(a)) expresses the fact that we want to select a blob position P_t whose encompassing image patch is close to the canonical appearance C.

If we condition on C and assume the canonical appearance is given, our model reduces to a standard HMM. Algorithms that track by template matching follow this approach; the constant appearance model is represented by templates built *a priori* [41, 51, 74, 112, 114]. These templates are detuned because they must generalize across all people and be invariant to clothing. Our model in Figure 2.5 allows us to build a template tuned to the specific person in a video; we build a torso model that captures the specific color of a person's shirt.

2.4.1 Tracking by EM

If we treat C as a model parameter, then Figure 2.5-(a) looks like a standard HMM where the *emission model* is unknown. One could then apply the well-known Expectation Maximization algorithm for HMMs (the Baum-Welch algorithm) to infer simultaneously the

hidden variables P_t and the model parameter C. Such an algorithm would take the following iterative approach:

E-step: Assume we have some estimate of the torso appearance C. Then our model reduces to a standard HMM, and we perform dynamic-programming to estimate the sequence of torso positions P_t. Formally, one would perform the *forward-backward* algorithm to compute "soft" positions $\Pr(P_t|Im_{1:T}, C), \forall t$.

M-step: Given a track of torso positions from the E-step, we can re-estimate the torso model C by calculating the average image patch at the tracked positions. Formally speaking,

$$C_{new} = \frac{1}{T}\sum_t \mathsf{E}_{P_t}[Im_t(P_t)] = \frac{1}{T}\sum_t \sum_{P_t} \Pr(P_t|Im_{1:T}, C) Im_t(P_t).$$

Given the new estimate or torso appearance C_{new}, we repeat the E-step.

Such a method has two practical limitations: (1) once we augment P_t to incorporate articulated pose, maintaining probability distributions over the space of all poses becomes difficult and (2) we do not know the number of people in a given video. If there are multiple people present, then we will have to learn multiple appearance models C. In this case there is no single true value of C, and it is convenient to treat it as a random variable (so we can represent uncertainty).

2.4.2 Tracking by model-building

We can write our HMM as an undirected graphical model (Figure 2.5-(c)) where C is an explicit variable:

$$\Pr(P_{1:T}, C|Im_{1:T}) \propto \prod_t \Psi(P_t, P_{t-1})\Psi_t(P_t, C), \tag{2.7}$$

where the potentials are:

$$\Psi(P_t, P_{t-1}) \equiv \Pr(P_t|P_{t-1}) \propto e^{-||P_t - P_{t-1}||^2} \tag{2.8}$$

$$\Psi_t(P_t, C) \equiv \Pr(Im_t|P_t, C) \propto e^{-||Im_t(P_t) - C||^2}. \tag{2.9}$$

This characterization emphasizes the fact that when we track from video, we want both to recover object position P_t *and* to build a model of object appearance C. Direct inference is difficult because we cannot search over all possible pixel positions P_t and appearance models C; both variables are inherently continuous. Furthermore, the potential $\Psi_t(P_t, C)$ is typically multi-modal and non-Gaussian. Even if we are told that the blob appearance is a white patch, there might be multiple places in an image that locally look similar to it.

However, the iterative EM approach hints at a useful framework to follow. Given a detuned or rough appearance model C, we can use it to obtain a rough track. Given the rough track, we can tune the model to (possibly multiple) people in a video. Given the tuned models, we can re-track, and iterate as necessary. The difficulty with this approach is the initial *model-building* stage of using a detuned appearance model to build multiple tuned models.

This view of tracking as model-building is rare in the video analysis community; one important exception is the the layered sprite model of Jojic and Frey [56]. The authors develop an EM algorithm to learn blob appearances (or, in their case, layers). One important difference is that their model ignores motion constraints $\Pr(P_t|P_{t-1})$. The authors also assume they know the number of layers *a priori* and do not deal with articulated models. We develop an algorithm in the next chapter that addresses these shortcomings.

Chapter 3

Building Models of Animals

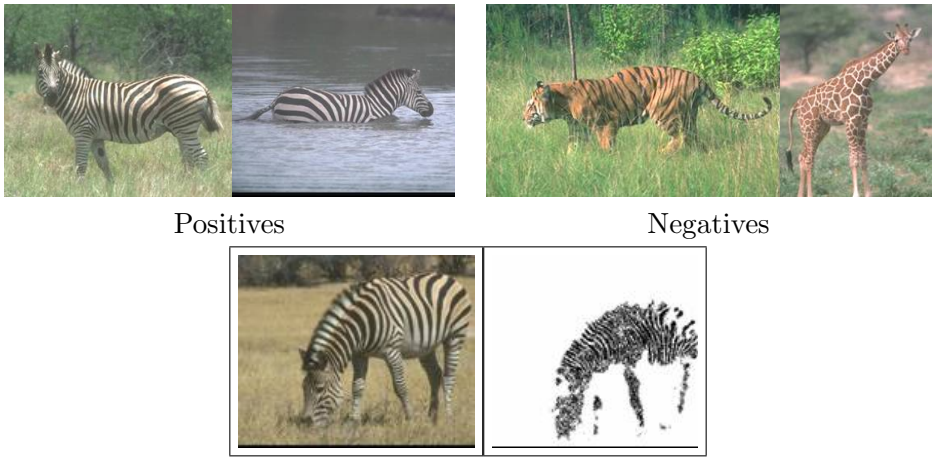


Figure 3.1: To build models from videos, we look at techniques that build models from images. Assume one wants to learn a zebra model from an image collection of zebra images (**left**) and non-zebra images (**right**). Given this input, learning algorithms look for image features that *consistently* appear in the positive set, but that never appear in the negative set. Algorithms do this using variants of clustering or EM. In this case, Schmid [98] learns a zebra model that looks for black-and-white striped patches. Given the learned model, it can be used to find zebra regions in a novel image (**bottom**). We develop a similar algorithm that builds models from videos by looking for *consistency* across a set of frames.

To construct an algorithm that builds models from videos, we look to the object de-

tection community for inspiration. Many authors have developed algorithms that build object models from image collections [29, 33, 60, 98, 119]. Say we want to use one of these algorithms to learn a model of a zebra (Figure 3.1). We assemble a set of positive example images containing zebras, and a set of negative example images not containing zebras. This form of input is often called *semi-supervised* data because we are labeling which images contain a zebra, but for a given zebra image, we do not label which image regions are zebra and which are background. The task of the learning algorithm is to "finish" the partial labeling; learn a zebra model that labels zebra image regions. Most algorithms do this by variants of clustering or EM; basically one looks for image regions that consistently appear in the positive set, but not in the negative set. In the case of Figure 3.1, the algorithm consistently finds black-and-white striped image patches in the positive set, and so learns a corresponding zebra texture model.

We can apply the same kinds of learning algorithms to frames from a video sequence of a zebra. We treat the frames as images from the positive set. Unfortunately, we do not have a negative set with which to compare, but we do have an alternate source of information: smoothness of motion. We know that a zebra region is likely to appear consistently in most frames of a zebra video, *and* that those zebra regions are likely to move smoothly from one frame to the next. In essence, we can use temporal coherence in a video sequence to provide supervisory signals.

Assume we are given a video sequence of single unknown animal. This chapter presents an algorithm that automatically builds a visual model of the animal. Section 3.1 describes

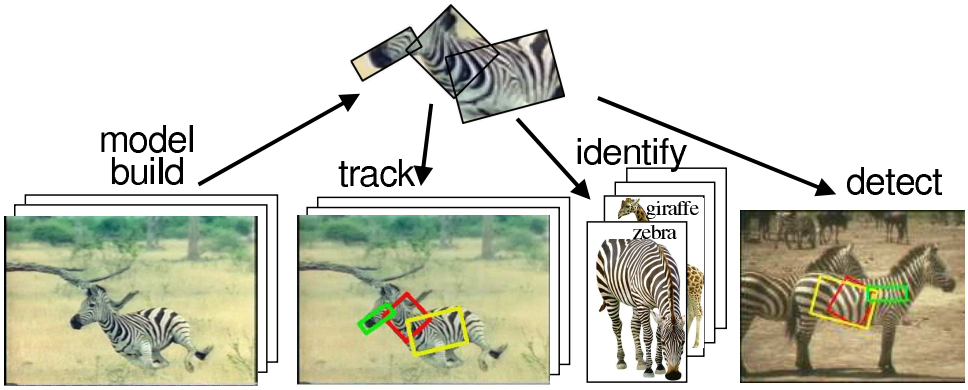


Figure 3.2: An overview of this chapter. Given a video sequence with a single animal, we cluster candidate segments (Section 3.1) to build a visual model of the animal (Section 3.2). We then use the model to track the animal in the original video (Section 3.3), identify the animal (Section 3.5), and detect the animal in new images (Section 3.6).

a clustering method that constructs rough spatio-temporal tracks of body segments over a sequence. In Section 3.2, we use the tracks to learn a visual model known as a *pictorial structure* [31, 35].

Once we learn the model, there are several neat applications. We use it to find the animal in the original video (so that we can **track** it better; Section 3.3). By looking up the visual model in a library, we can also **identify** the animal (Section 3.5). Finally, we can use the model to **detect** the animal in other images or videos (Section 3.6). This last application is interesting because it suggests an alternative motivation for this chapter (besides tracking): an algorithm for building models for object detection [91].

We significantly improve the quality of the visual model by augmenting it with a animal texture model learned from a library of textures. Examining various texture descriptors, we find they do not characterize animal textures well. We develop a novel texture representation in Section 3.4 that outperforms the state-of-the-art.

3.1 Building a spatio-temporal track

Suppose we are given a video with a single animal, and we want to build a spatio-temporal track of how its body parts deform over time. If we assume the animal is made up of body segments, we can:

1. *Detect* candidate segments with a detuned segment detector.
2. *Cluster* the resulting segments to identify body segments that look similar across time.
3. *Prune* segments that move too fast in some frames.

3.1.1 Detecting Segments

We model segments as cylinders that project to rectangles in an image. One might construct a rectangle detector using a Haar-like template of a light bar flanked by a dark background (Figure 3.3). To ensure a zero DC response, one would weight values in white by 2 and values in black by -1. To use the template as a detector, one convolves it with an image and defines locally maximal responses above a threshold as detections. This convolution can be performed efficiently using integral images [118]. We observe that a bar template can be decomposed into a left and right edge template $f_{bar} = f_{left} + f_{right}$. By the linearity of convolution (denoted *), we can write the response as

$$im * f_{bar} = im * f_{left} + im * f_{right}.$$

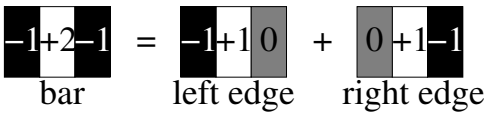


Figure 3.3: One can create a rectangle detector by convolving an image with a bar template and keeping locally maximal responses. A standard bar template can be written as the summation of a left and right edge template. The resulting detector suffers from many false positives, since *either* a strong left or right edge will trigger a detection. A better strategy is to require *both* edges to be strong; such a response can be created by computing the minimum of the edge responses as opposed to the summation.

In practice, using this template results in many false positives since either a single left or right edge triggers the detector. We found taking a *minimum* of a left and right edge detector resulted in response function that (when non-maximum suppressed) produced more reliable detections

$$\min(im * f_{left}, im * f_{right})$$

With judicious bookkeeping, we can use the same edge templates to find dark bars on light backgrounds. We explicitly searched over 15 template orientations (at 12^o intervals) and 25 scales (5 lengths crossed with 5 widths).

It turns out to be hard to build accurate low-level segment detectors. Figure 3.4-(a) shows three frames from a video of a zebra in which the detectors often fire on the animal body, but also fire on clutter in the background. We would like to pick out the true animal body parts from the set of candidate detections. Unfortunately, we do not know what the animal segments should look like (since we are not told a zebra is present). But we know that animal segments should be *consistent* in appearance over time; if the head is striped in the first frame, it should be striped in the final frame. We find collections of segments

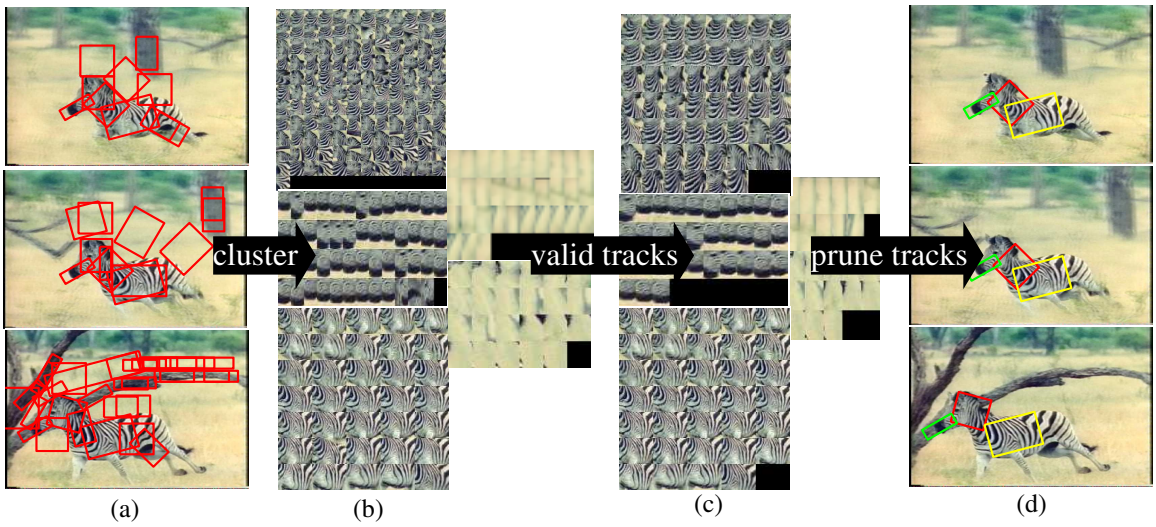


Figure 3.4: Obtaining spatio-temporal tracks by clustering. We first search for candidate segments using local detectors (we show 3 sample frames in (a)). We cluster the image patches together in (b). From each cluster we extract a valid sequence obeying our motion model in (c). We prune away the short sequences to retain the final spatio-temporal tracks in (d).

that look similar to each other by *clustering* the entire set of detected segments.

3.1.2 Clustering Segments

Since we do not know the number of segments in our model (or for that matter, the number of segment-like things in the background), we do not know the number of clusters *a priori*. Hence, clustering segments with parametric methods like Gaussian mixture models or k-means would be difficult. We opted for the mean shift procedure [22], a non-parametric density estimation technique.

We create a feature vector for each candidate segment, consisting of a normalized color histogram in the Lab color space, appended with shape information (in our case, simply the length and width of the candidate patch). Note that this feature vector is to be used

for *clustering*, for which it is sufficient. The representation of appearance is not limited to this feature vector.

The color histogram is represented with projections onto the L, a, and b axis, using 10 bins for each projection. Hence our feature vector is $10 + 10 + 10 + 2 = 32$ dimensional. We scale the histogram and scale dimensions so as to obtain a meaningful $\mathcal{L}_2$ distance for this space. Further cues — for example, image texture — might be added by extending the feature vector, but appear unnecessary for clustering in the cases we have considered thus far.

Identifying segments with a coherent appearance across time involves finding points in this feature space that are (a) close and (b) from different frames. Because this is difficult; we drop requirement (b), which can be imposed on clusters *post hoc*, and concentrate on (a). The mean shift procedure is an iterative scheme in which we find the mean position of all feature points within a hypersphere of radius h, recenter the hypersphere around the new mean, and repeat until convergence. We initialize this procedure at each original feature point and regard the resulting points of convergence as cluster centers. For example, for the zebra sequence in Figure 3.4, starting from each original segment patch yields five points of convergence (denoted by the centers of the five clusters in (b)).

Sometimes, illumination changes will cause a single animal part to appear in two or more clusters. As a post-processing step we greedily merge clusters which contain members within h of each other (starting with the two closest clusters). We account for over-merging of clusters by extracting multiple valid sequences from each cluster during step (c). That

is, for each cluster during the third step in Figure 3.4 (explained further in the following section), we keep extracting sequences of sufficient length until none are left. Hence for a single arm appearance cluster, we might discover two valid tracks, one of a left arm and one of a right arm.

3.1.3 Enforcing a motion model

As Figure 3.4 indicates, not every coherent patch is associated with a moving figure. The second column of clusters in 3.4-(b) are background regions. However, at this point cluster elements are neither constrained to move with bounded velocity nor required to form a sequence — there might be several elements from the same frame.

We now find the most likely sequence of candidates for each cluster that obeys the velocity constraints. By fitting an appearance model to each cluster (typically a Gaussian, with mean at the cluster mean and standard deviation computed from the cluster), we can formulate this optimization as a straightforward dynamic programming problem. Let P_t be the position of a segment in the t^{th} frame. We assume that these have a Markovian behavior: i.e., $\Pr(P_t|P_{1:t-1}) = \Pr(P_t|P_{t-1})$. The reward for a given candidate is its likelihood under the Gaussian appearance model, and the temporal rewards are '0' for links violating our velocity bounds and '1' otherwise. We add a dummy candidate to each frame to represent a "no match" state with a fixed charge. By applying dynamic programming, we obtain a sequence of segments, at most one per frame, where the segments are within a fixed velocity bound of one another and where *all* lie close to the cluster center in appearance. As Figure 3.4-(c) demonstrates, this results in a somewhat smaller set of segments associated

with each cluster. This is particularly true for the second column of background clusters; background segments that happen to cluster together often do not move like true segments.

We now discard those tracks which are not long enough. In Figure 3.4-(c), this results in pruning away the second two clusters. Note that we could impose other tests of validity beyond the length of a track. For example, we might require that a segment move at some point, and so we would prune away a track which is completely still. Alternatively, if we are given two different videos of the same animal, we might prune away those clusters that do not appear in both.

The segments belonging to the remaining three clusters are shown in Figure 3.4-(d). We can now learn a visual model from the spatio-temporal tracks in Section 3.2. But first, it is useful to cast our clustering procedure in light of our constant appearance model developed in Section 2.4.

3.1.4 Approximate Inference

The segment-finding procedure discussed above is, in fact, an approximate inference procedure for the graphical model shown in Figure 3.5. Recall our original blob model from Figure 2.5 (shown again in Figure 3.5-(a)); this model captured the fact that we want to track a segment while building a model of its appearance. The algorithm described in this section is a loopy inference procedure for our blob model (see also [23, 77, 89]). We pass max-product messages asynchronously and visualize our message schedule with the embedded trees shown in Figure 3.5.

In the first subtree, we want to infer a posterior over C given the image patches from a

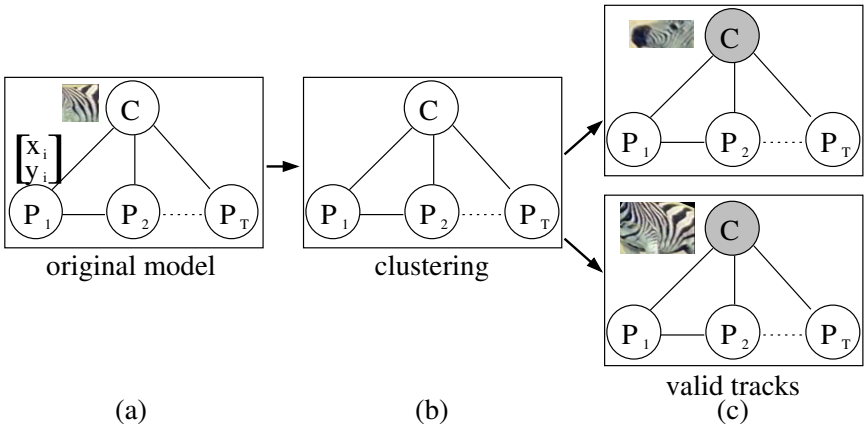


Figure 3.5: Approximate inference on our blob tracking model from Figure 2.5. The original model (**a**) encodes the fact that we want to track a segment while simultaneously building a model of its appearance. An alternative interpretation is that we want to cluster, or learn a coherent appearance, while simultaneously enforcing that all the patches from a cluster obey a motion model. We can do the latter (approximately) by dropping the motion constraint. We naively cluster, looking for collections of coherent segments (**b**). In this case, we find multiple coherent appearances (corresponding to the zebra head and body). We instantiate the model multiple times, for each cluster. Given the learned appearance, we do dynamic programming to extract a sequence of valid tracks where all the segments look similar to the learned model (**c**).

sequence. We show in Appendix 3.A that the mean shift clustering procedure finds modes in the posterior of C. We interpret each mode, or cluster, as a *unique* segment. We instantiate multiple copies of the model Figure 3.5-(b), one for each cluster. We can partly justify this procedure by our aggressive post-clustering merging of clusters; any left-over clusters that remain separate are likely to be different segments rather than multiple appearance modes of a single segment.

We now can treat C as an observed quantity, for each instantiation of Figure 3.5-(c). Inferring P_t from such a model is straightforward; this is just our dynamic programming solution to find the most likely sequence of candidates given a known appearance. Note our

initial claim of segment configurations (P_t) being Markovian is only true when we condition on C. Finally, we disregard those instantiations we deem to be invalid (i.e., not existing for enough frames).

3.2 Learning a pictorial structure

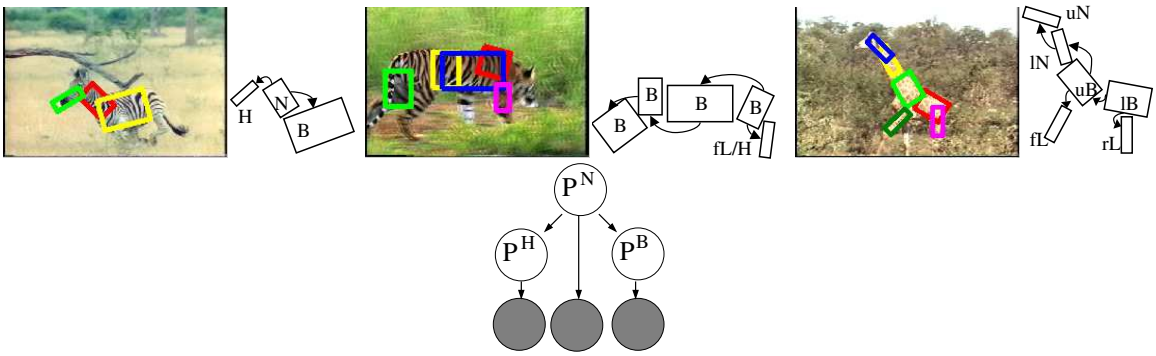


Figure 3.6: We show the pictorial structures learned from videos of a zebra **left**, tiger **center**, and giraffe **right**. On the **bottom**, we visualize the zebra pictorial structure as a graphical model. This model is parameterized by probability distributions capturing geometric arrangement of parts $\Pr(P^i|P^j)$ and local part appearances $\Pr(Im(P^i)|P^i)$ (the vertical arrows into the shaded nodes). These distributions and the tree structure of the graph are automatically learned from the video. We manually attach a semantic description to each limb as Head, upper/lower Neck, upper/lower Body, or front/rear Leg. Labeling the tiger model is tricky; many limbs swim around the animal Body, and one flips between the Head and front Leg. We use these labels to help evaluate localization performance in our results; they are *not* part of the shape model. Obtaining a set of canonical labels appears difficult.

We use the spatio-temporal tracks (obtained by clustering) to build a visual model called a *pictorial structure*. A pictorial structure model is a parts-based model of an object consisting of two terms; a geometric term that relates the spatial arrangement of parts, and

an appearance term that describes the local appearance of each part [31, 35, 51].

$$\Pr(P^1 \dots P^N | Im) \propto \prod_{(i,j) \in E} \Pr(P^i | P^j) \prod_{i=1}^{n} \Pr(Im(P^i) | P^i). \tag{3.1}$$

$\Pr(P^i|P^j)$ are geometric terms that capture the spatial arrangement of part i with respect to part j, and $\Pr(Im(P^i)|P^i)$ captures the local appearance of the image at part i. Here, we extend part configuration P^i to include both position (x, y) and orientation θ. The position of each non-root segment P^i is represented with respect to the coordinate system of its parent P^j. E is a set of edges that capture the dependency structure of the model. A model is fully specified when the edge structure and the probability distributions along each edge are known. If E is a tree, one can efficiently match these models to an image using Dynamic Programming (DP). Felzenszwalb and Huttenlocher [31] describe efficient DP-based techniques for computing the MAP estimate and for sampling from the posterior in Equation 3.1. One can also use the (unnormalized) posterior as an animal detector by only accepting those maximal configurations above a threshold.

Learning by maximum likelihood: Standard methods learn pictorial structures by maximum likelihood estimation (MLE) given images where parts are labeled [31, 52]. Theoretically, one could learn pictorial structures from unlabeled data using EM, where labels are the hidden variables marginalized out. This approach is taken in [33], where the learned models are called *constellation models*. To avoid local maxima issues with EM, those models must be learned from uncluttered images and with finely tuned part detectors. In our case, we learn pictorial structures by direct MLE without any manual intervention;

we use the coherence in a video to provide an implicit labeling. Note that we do not need precise labels, but rather correspondence between parts over time – this is provided by the cluster membership.

Learning E: Typical methods for learning the spatial structure E will not work in our case; we describe the approach from [31, 51, 90] here. Consider a fully connected bi-directional graph where each vertex represents a segment P^h, P^n, and P^b (the precise segment labels are not needed so long as their correspondence between frames is known). Directed edges in this graph are weighted by the entropy of $\Pr(P^i|P^j)$. To learn the tree structure that maximizes the likelihood of the observations, [31, 51, 90] finds the minimum-entropy spanning tree. This tends to result in poor models. Often two far away limbs will be directly linked in the learned spatial model. This is because the position P^i of the detected limbs are quite noisy (due to the detuned limb detector), in turn producing noisy entropy estimates (see Figure 3.7). To enforce the natural prior that two limbs that tend to appear near each other should be spatially linked together, we replace the entropy term with the mean distance between those two limbs, and then compute the minimum spanning tree. We root this tree at the most "stable" limb (the limb detected most often in the original video). This produces the tree spatial models in Figure 3.7.

$\mathbf{Pr(P^i|P^j)}$: We fit the geometric terms $\Pr(P^i|P^j)$ by standard Gaussian MLE assuming diagonal covariance matrices. For example, assume one has a collection of zebra images where *H*ead, *N*eck, and *B*ody segments are labeled. The sample mean and standard deviation of head positions relative to the neck would define the MLE estimate of $\Pr(P^H|P^N)$.

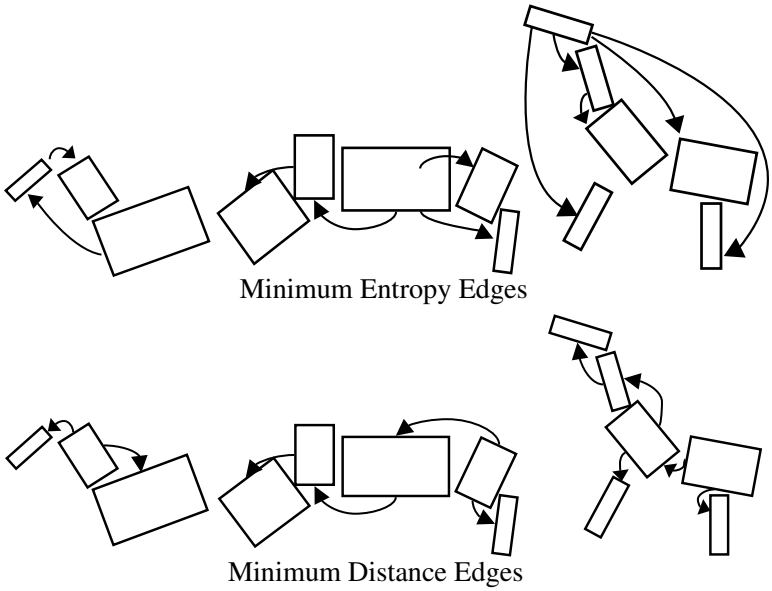


Figure 3.7: On the **top**, we show the edge structure E learned by a minimum entropy spanning tree (this are the edges that maximize the likelihood of the observed spatio-temporal tracks). This can create links between two far away segments (the giraffe's head and leg) because of noisy entropy estimates. A better strategy is to link the parts that tend to lie near each other (**bottom**).

In our case, we use the cluster membership to provide the labels.

$\Pr(Im(P^i))$: One could also fit the appearance terms $\Pr(Im(P^i))$ by MLE; however, we found this yields a poor detector since we have ignored the background. We learn a **discriminative** part appearance model by learning a animal texture classifier from the video. We use the spatio-temporal tracks to segment the video into animal (foreground) and background pixels. We fit a 5-component Gaussian mixture model in RGB space for the foreground/background, as in [97].[1] We use our pictorial structure to find an animal in a image using the procedure in Figure 3.8. Given an image, we first use our texture model to label the animal pixels. We evaluate the part likelihood $\Pr(Im(P^i))$ by convolving

[1]Technically speaking, we learn generative models for the foreground and background. They are "discriminative" in the sense they are used to classify pixels.

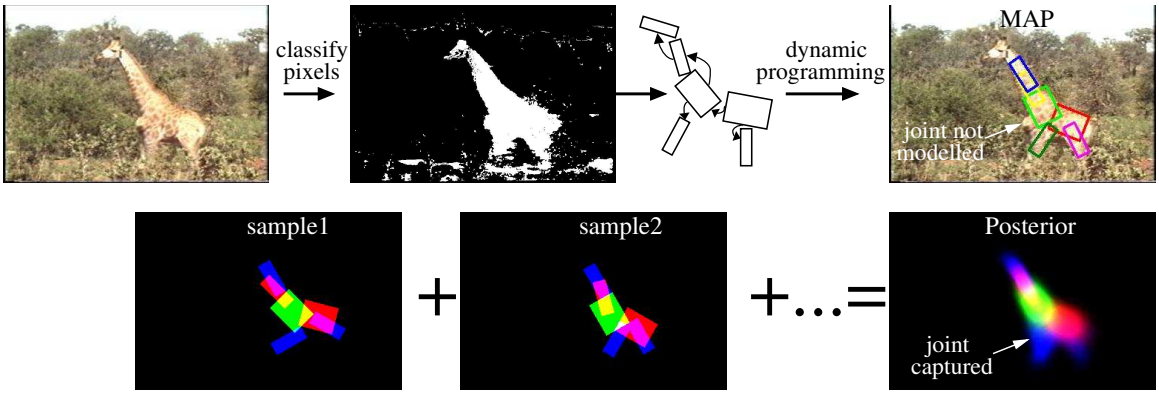


Figure 3.8: Detecting a pictorial structure $\Pr(P^{head}, P^{neck}, \ldots | Im)$. Given the image on the **left**, we first classify foreground pixels using a color model learned from the spatio-temporal tracks. We use dynamic programming to find an arrangement of limbs that lie on foreground pixels and that look like the shape prior; this yields the MAP estimate on the **right**. We also can use a foreground mask and shape prior to generate sample body poses from the posterior $\Pr(P^{head}, P^{neck}, \ldots | Im)$ (using the method of [31]). We superimpose the samples to yield the final posterior map on the **bottom**. The posterior models the front leg joint better than the MAP estimate; this suggests we can use uncertainty in how we match a model to compensate for its inadequacies.

the label mask with rectangle templates looking for light bars on dark backgrounds (as in Section 3.1.1); we want parts to lie on animal pixels and not the background. An alternative would have been to learn a separate texture model for each animal limb (we do this for people in Section 4.3.2); we found this was not necessary for animals with homogeneous texture.

Note that our RGB-based texture classifier is specific to the video it is trained on. Hence they are limited to being used to find animals in the original videos (for tracking, as in Section 3.3). However, if we want to find animals in new images, we need to build more sophisticated texture models (Section 3.4).

3.3 Tracking by finding

Given the learned pictorial structure, we can use it to track the animal in the original video. For each frame, we find the best-matching body pose (by dynamic programming), or obtain a distribution over body poses (by sampling); see Figure 3.8. We show results for three sequences depicting different moving animals in Figures 3.9, 3.10, and 3.11. The tracks were not hand initialized, and the same program was used in each case. The program automatically learns a pictorial structure and then identifies instances in each frame. In each sequence, the animal's body deforms considerably, the zebra because it is moving very fast, the giraffe and the tiger because giraffes and tigers deform a lot when they move. Nonetheless, the program is able to build an appearance model that is clearly sufficient to capture the essence of the moving animal, but lacks some details. In particular, legs are narrow, fast, and hard to detect; consequently, the learned pictorial structures fails to model them. Furthermore, the temporal correspondences for the segments — which are indicated by colored outlines in the figures — are largely correct. Finally, the tracker has been able to identify the main pool of pixels corresponding to the animal in each frame.

Following [109], we evaluate our tracker using detection rates (Figure 3.12) obtained from the original video. Our algorithm builds a representation of each animal as a collection of parts. We define a correct localization to occur when the majority of pixels covered by an estimated part have the correct semantic label from Figure 3.6. The quality of the track using the pictorial structure is quite good, and is much better than the original spatio-temporal track. This suggests that tracking is easier with a better model. One can envision

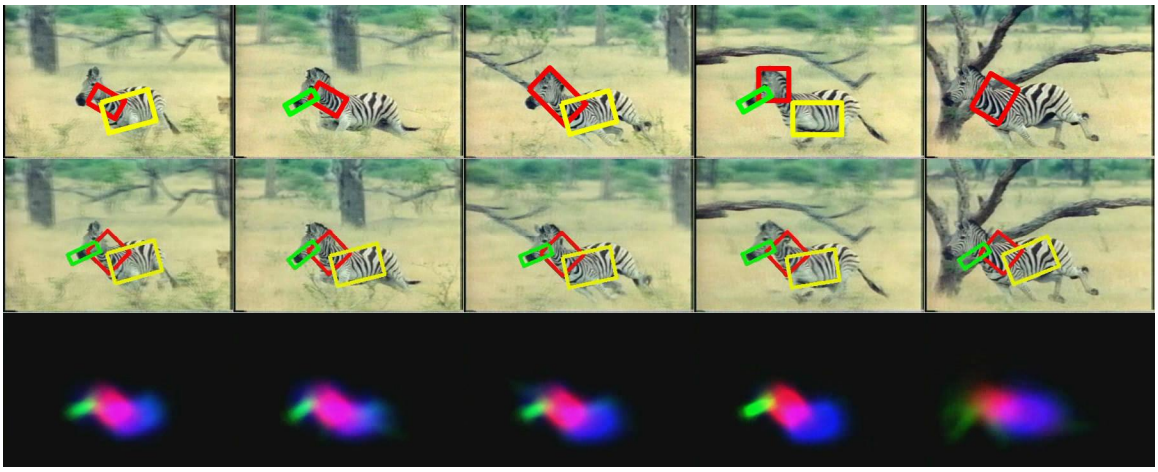


Figure 3.9: Tracking results for the zebra video. In the **top** row, we show the spatio-temporal tracks obtained by clustering together segments that obeyed our motion model. Correspondence over time (denoted by the colors) are given by cluster membership. Given those segments, we learn the zebra pictorial structure model shown in Figure 3.6. Given the learned model, we can re-track by computing MAP estimates for each frame in the video (**middle**). We can also visualize the entire posterior using the sampling method from Figure 3.8 (**bottom**). Note that the tracks tend to get significantly better as we build an improved visual model of the zebra; we quantify this in Figure 3.12.

Figure 3.10: Tracking results for the tiger video, using the same conventions as Figure 3.9. Note the tracks tend to get significantly better as we build an improved visual model (we quantify this in Figure 3.12).

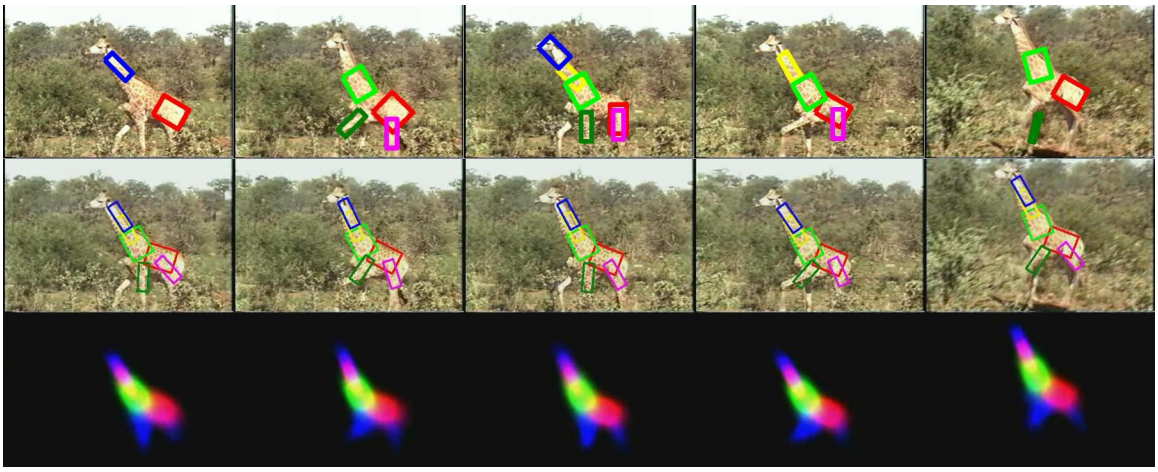


Figure 3.11: Tracking results for the giraffe video, using the same conventions as Figure 3.9. Note the tracks tend to get significantly better as we build an improved visual model (we quantify this in Figure 3.12).

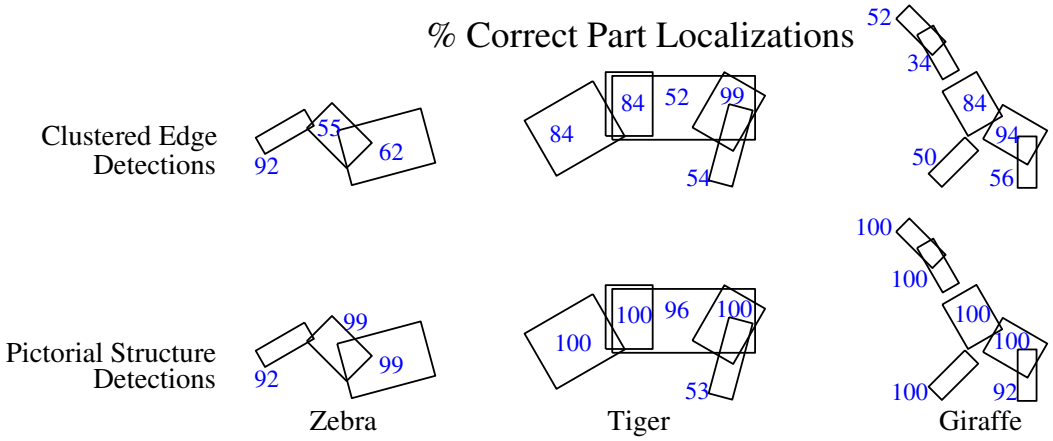


Figure 3.12: We evaluate our trackers using detection rates for the original videos. Our algorithm builds representations of animals as a collection of parts. We overlay the percentage of frames where parts are correctly localized. We define a part to be correctly localized when it overlaps a pixel region with the correct semantic label from Figure 3.6. On the **top**, we show results from the original spatio temporal tracks obtained by clustering edge detections. After learning a pictorial structure from the tracks, we use the model to re-detect the animal. This results in significantly better performance, as shown on the **bottom**.

iterating this procedure (in a manner quite similar to EM) by refitting a pictorial structure model to the newly tracked segments, and then tracking given the pictorial structure model.

Figure 3.13: Our library of animal textures built from Hemera. We show a subset of 100 17X17 patches for each of our 38 animals; we mark the giraffe, tiger, and zebra rows in yellow. Our recognition task requires texture classification *and segmentation* (we need to separate the animal from its background). This means we need to evaluate textures on a local image patch. We use this library to evaluate patch descriptors in Table 3.1.

We performed only one iteration.

Our tracker is successful largely because of the quality of the foreground masks produced by the learned animal classifiers (Figure 3.8). These classifiers do not generalize well to novel images (with say, different illumination conditions); we build robust animal texture classifiers in Section 3.4.

3.4 Building a texture model

Both to identify a pictorial structure (Section 3.5) and to detect it in new images (Section 3.6), we need a good animal texture descriptor. We want a descriptor capable of producing foreground masks like those of Figure 3.8, but for novel images. Specifically, it must be able to **segment** out an animal from cluttered backgrounds (typically foliage). Descriptors developed for standard vision datasets (such as CUReT [24]) may not be appropriate

Comparing texture descriptors for detecting animal patches

Descriptor	All	Zebras	Tigers	Giraffe
Patches	8.2	8.93	5.56	5.97
Textons	11.1	31.3	12.7	12.5
SIFT	13.6	40.0	19.1	21.9

Table 3.1: We count how often we can correctly identify an animal based on texture from a single patch from Figure 3.13. We report percentage of correct detections in cross validation experiments for a 1-NN classifier using 1500 prototypes per class. For the full (38 class) multi-class problem 'All', we perform quite poorly. Many animal classes (such as elephants and rhinoceroses) are hard to discriminate using texture alone. When scoring correct detections solely on zebra, tiger, and giraffe test patches, we do much better, indicating those animals have distinctive texture. Looking at various patch representations (normalized patch pixel values, histograms of textons, and a SIFT descriptor), we find SIFT performs the best. We adopt it as our texture descriptor, and examine its behavior further in Figure 3.14.

for segmentation since they classify entire images of homogeneous texture. Giraffe textures in particular are notorious for being difficult to capture (e.g. [60, 98]; see Figure 3.14).

3.4.1 Texture library

To evaluate descriptors on small image patches, we create a texture library. We use the Hemera Photo-Object[44] database of image clip art; these annotated images have associated foreground masks. We use all the images in the "animals" category, throwing away those animals with less than 3 example images. This leaves us with about 500 images spanning 38 animals. We assemble a texture library by randomly sampling 1500 17X17 patches from each animal class (Figure 3.13).

3.4.2 Descriptor Evaluation

We use the library to compare 3 different patch descriptors; histograms of textons [63], intensity-normalized patch pixel values [116], and the SIFT descriptor [66]. **Textons** are

quantized filter bank outputs that capture small scale phenomena (such as t-junctions, corners, bars, etc.). They are typically binned into a histogram over some spatial neighborhood. The SIFT patch descriptor is a 128 dimensional descriptor of gradients binned together according to their orientation and location; it is designed to be robust to small changes in pixel intensity and position. Note that we use the raw descriptor on a 17X17 patch, without normalizing for scale or dominant orientation (as in[66]).

We evaluate our texture model by 3-fold cross-validation. We tried a variety of classifiers, such as K-way logistic regression, SVMs, and K-Nearest Neighbors (NN). This classification problem is difficult because of the large number of classes (almost 40); training an all-pairs SVM classifier took exorbitantly long. When training a SVM on 2 animal classes, we did not observe any sparsity. This suggests that the decision boundary is curvy (and so we need all sample points as support vectors). K-NN performed the best, with K = 1. Hence we evaluate patch descriptors using 1-NN classification (again using 3-fold cross-validation) in Table 3.1.

There are three conclusions we can draw: (1) Classifying small patches seems much harder than classifying entire images of homogeneous texture. Our results are worse than those reported for texture databases like CUReT [60, 63, 116]. (2) There is a large variance in performance depending on the animal class. Discriminating between elephants and rhinoceros is hard because of their similar hides, but highly textured animals such as zebras, tigers, and giraffes stand out. Finally, (3) SIFT seems much better suited for detecting animal textures from small patches. We look at the ability of SIFT to segment out animals

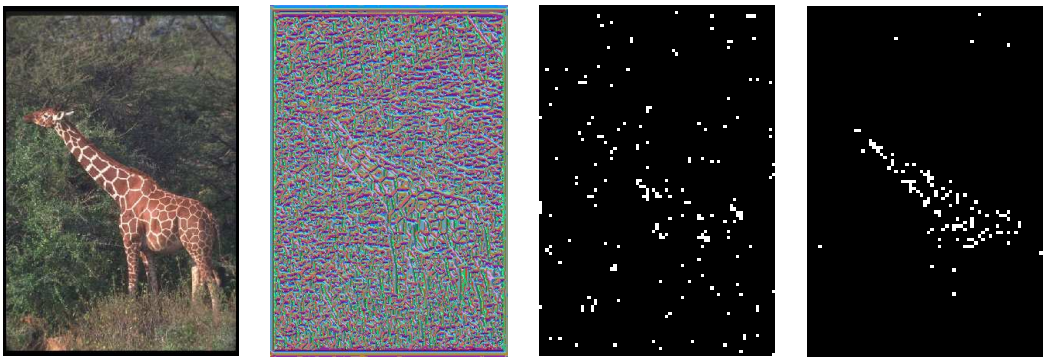


Figure 3.14: Given a query image **left**, we replace each 17X17 patch with its closest match from a patch in our texture library. This means we need a good animal texture descriptor; one that captures the long thin stripes that lie within big blobs typical in a giraffe. Standard approaches use histograms of textons (quantized filter bank outputs) [60, 63, 98, 116]. We show a texton map on the **middle left**, where each color maps to an individual texton. The big blobs that distinguish the giraffe from the background are only apparent from the *long-scale spatial arrangement* of textons. Looking at histograms of textons over small neighborhoods looses this spatial arrangement. Hence classifying giraffe patches based on texton histograms is a poor approach, as seen in the **middle right** (and as acknowledged by [60, 98]). Rather, if we classify patches using a descriptor capturing spatial arrangement of pixels (e.g. SIFT), we are better at detecting giraffe patches (**right**).

from real images in Section 3.4.3.

3.4.3 Why are giraffes hard?

Segmenting giraffes present particular difficulties for texton based descriptions (e.g. [60, 98]; Figure 3.14). The texture is characterized by phenomena at two scales (long thin stripes that lie in between big blobs). If we calculate textons over a large scale, we miss the thin stripes. If we calculate textons on a small scale, the long scale spatial structure of the textons defines the big blobs (Figure 3.14). This spatial structure is lost when we construct a histogram of local neighborhoods from the texton map. This suggests that we should not think of a giraffe texture as an unordered collection of textons, but rather simply a patch, or

a collection of spatially ordered pixels. A robust patch descriptor such as SIFT is a natural choice, although other descriptors such as [8] may also prove useful.

Our results are surprising because SIFT was not designed to represent texture (as noted in [66]). However we find it can represent texture given we store enough examples. The drawback to our nearest neighbor approach is the time required to classify a new patch; we must compare it against 1500 prototypes from 38 classes. Obtaining a simpler parametric representation of animal texture remains future work.

We now can use our texture models to identify the animal in a video (Section 3.5) and detect the animal in new images (Section 3.6).

3.5 Identifying the animal

We use our patch-based texture model from Section 3.4 to identify the animal in a video. We assume that the animal in a given video is one of the 38 animals in Hemera. We scale the Hemera images and video clips to be similar sizes, and assume the animals are present at similar scales. We use a two-part matching procedure, shown in Figure 3.15.

Texture cue: We match the texture models built from *Hemera to the video*. We use the animal tracks (Section 3.3) to segment the video into animal/non-animal pixels. We extract the set of all 17X17 animal patches from the video, and classify each as one of the 38 animals. We do 1-NN classification on each patch, finding the closest match from our library of animal textures (by matching SIFT descriptors). We obtain a texture posterior for animal labels given a video by counting the number of times the classifier votes for the

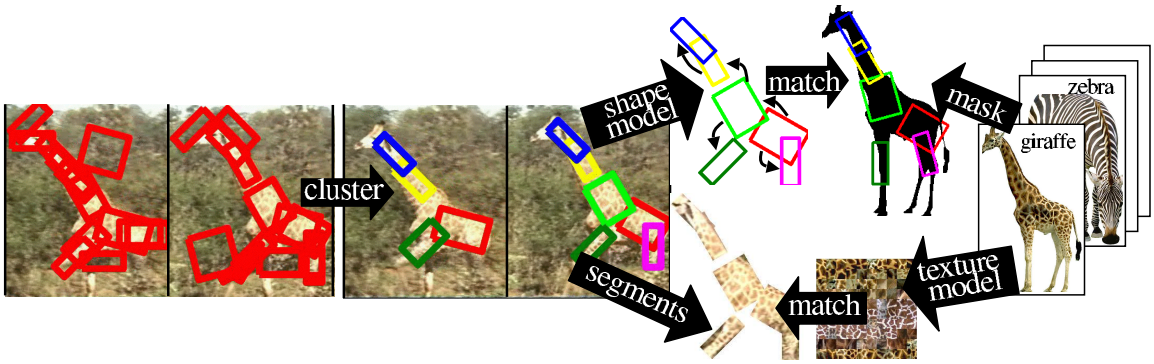


Figure 3.15: We identify animals in videos by matching to labeled image collections. Given an animal video (**left**), we obtain spatio-temporal tracks of limbs by clustering (Section 3.1). We use the tracks to learn a spatial model (Section 3.2) and segment the video into animal/non-animal pixels (Section 3.3). On the **right**, we build a texture model for various animals from the Hemera collection of labeled and segmented images. We link our models by matching the shape model built from video to the foreground mask of the Hemera images and matching the texture model built from Hemera to the segmented video (Section 3.5). This automatic matching *identifies* the animal in the video. We use the combined shape and texture model for recognition in Figure 3.18.

i^{th} animal class. Looking at Figure 3.16), we see that matching solely based on texture does not procude the correct animal label; the giraffe video matches best with a 'leopard' texture.

Shape cue: We add a shape cue by matching the shape model built from the *video to Hemera*. For each image in the Hemera collection, we use dynamic programming to find a configuration of limbs that occupies the foreground mask *and* that is arranged according to the shape prior learned from the video. We show 4 matches for our giraffe shape model in Figure 3.17. Note that the model matches quite well to giraffe images in Hemera. For each animal class, we take the best shape match score obtained over all images in that class. We normalize the scores to obtain a shape posterior over animal labels in Figure 3.16. Using shape, we label the giraffe video as 'giraffe', but mislabel both the zebra and tiger videos.

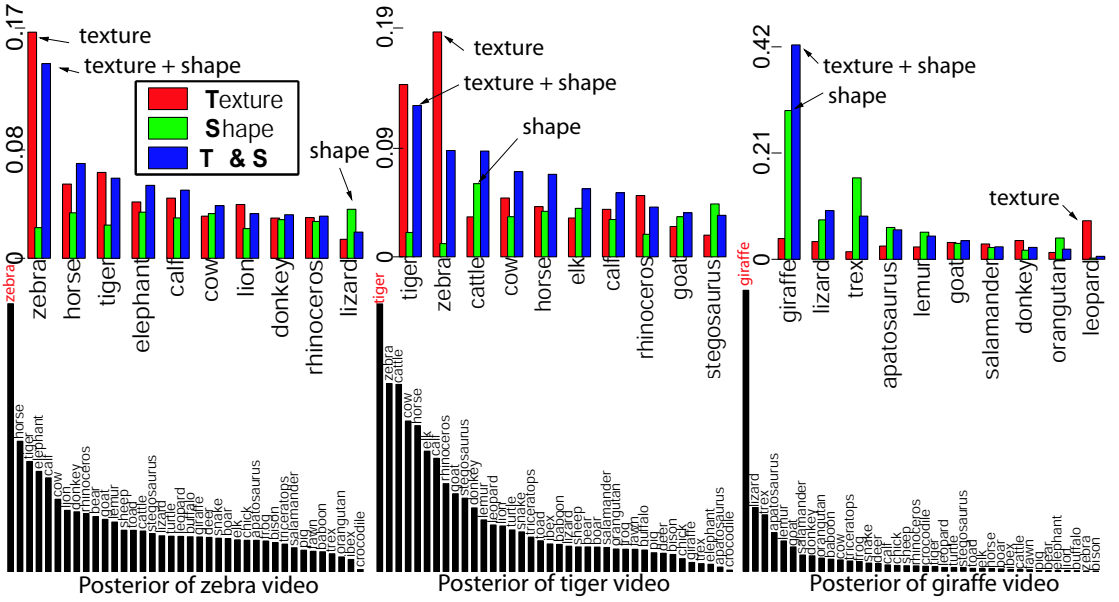


Figure 3.16: *We identify the animals in our videos by linking the shape models built from video to the texture models built from the labeled Hemera image collection. We show posteriors of animal class labels given the zebra (***left***), tiger (***middle***), and giraffe (***right***) videos. In the* **top row**, *we show posteriors of the ten best labels based on a texture cue, shape cue, and the combination of the two. We mark the MAP class estimate for each cue. Matching texture models built from Hemera to the segmented videos, we mislabel the giraffe video as 'leopard'. By matching shape models built from videos to Hemera images, we match the giraffe correctly, but incorrectly label the zebra and tiger videos. Combining the two cues, we match all the videos to the correct animal label. We show posteriors for the final combined cue over the entire set of labels in the* **bottom row**. *Note that the graphs are not scaled equally.*

We compute a final posterior by adding the (log) texture and shape posteriors (weighting shape by $\frac{1}{2}$) in the bottom row of Figure 3.16. Selecting the best class, we identify the correct animal label for each of our videos.

3.6 Finding animals in new images

We use our patch-based texture model from Section 3.4 and the correspondence obtained from Section 3.5 to build a system for finding animals in new images. We follow the approach

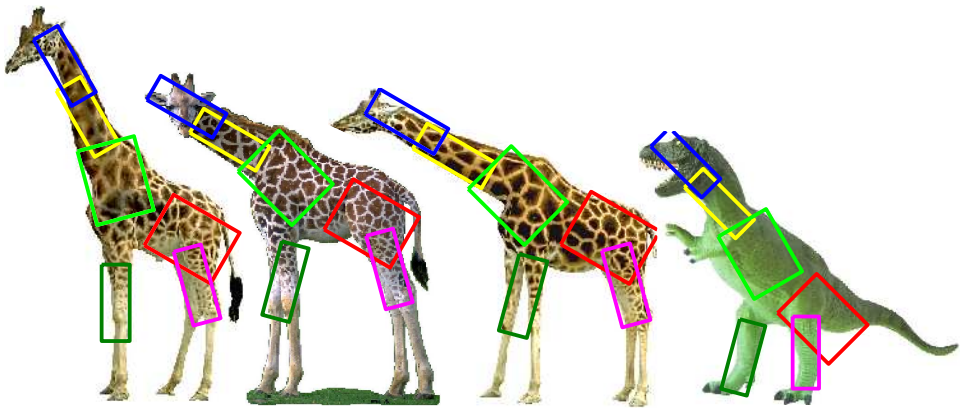


Figure 3.17: The top 4 matches (the top match on the **left**) in the Hemera collection for the shape model learned from the giraffe video. Note that our shape model captures the articulated variation in pose, resulting in accurate detections and reasonable false positives.

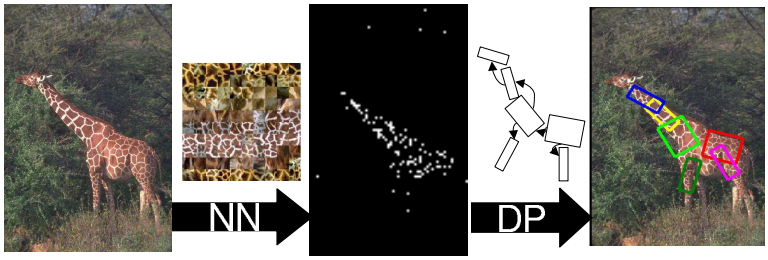


Figure 3.18: Our model recognition algorithm, as described in Section 3.6. Assume we wish to detect/localize a giraffe in a query image (**left**). We replace each image patch with its closest match from our library of Hemera animal and background textures (NN or nearest neighbor classification). We construct a binary label image with '1's for those patches replaced with a giraffe patch (**center**). We use dynamic programming (DP) to find a configuration of limbs that are likely under the shape model (learned from the video) *and* that lie on top of giraffe pixels in the label image (constructed from the image texture library). We show MAP limb configurations on the **right**.

outlined in Figure 3.18.

Given a query image, we first obtain a "foreground" mask using the texture library built in Section 3.4. We replace each 17X17 image patch with the closest match from our library, using a SIFT descriptor. We append the Hemera animal texture library with a 'background' texture class of 20000 patches extracted from random Corel images (not in our test pool and

not containing animals). We then construct a binary label image with a '1' if a patch was replaced with a given animal patch. We interpret this binary image as a foreground mask for that animal label, and use DP to find rectangles in the foreground arranged according to the shape model learned from video (Section 3.2). For the 'zebra','tiger', and 'giraffe' animal labels, we know the correct shape model to use because we have *automatically* linked them (Section 3.5). Hence our final animal detection system is completely automatic.

In practice, it is too expensive to classify every patch in a query image. Fortunately, the SIFT descriptor is designed to be somewhat translation invariant; off-by-one pixel errors should not affect it. This suggests we sample patches from the image and match them to our texture library. We match 5000 patches per image, which takes about 2 minutes in our implementation. Speeding up the matching using approximate nearest neighbor techniques [50] or building a parametric texture model may allow us to classify more patches from an image.

3.6.1 Evaluation

We tested our models on two datasets: images from the Corel collection and various animal images returned from Google. We scaled images to be roughly the same dimension as our video clips. Our Corel set contained 304 images; 50 zebras, 120 tigers, 34 giraffes, and 100 random images from Corel. Note that these random images are *different* from the set used to learn a background patch library. The second collection of 1418 images was constructed by assembling a random subset of animal images found by Google. It contains 315 zebras, 70 tigers, 472 giraffes, and 561 images of other animals ('leopard', 'koala', 'beaver', 'cow',

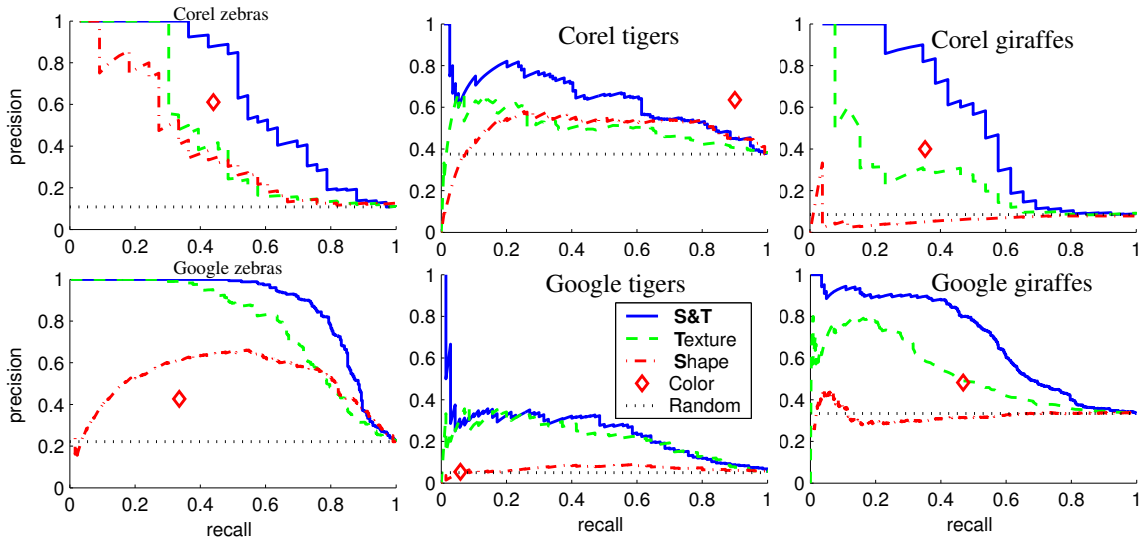


Figure 3.19: Precision recall curves for zebra, tiger, and giraffe detectors run on a set of 304 Corel images (**top**) and 1418 images returned by Google (**bottom**). The '**S**hape' detectors are built using shape models and crude texture models learned from the video. The '**T**exture' detectors are built using texture models trained on the image collection. The **S & T** detectors use texture models from the image collection and shape models from the video (where the linking was automatic, as described in Section 3.5). We compare with 2 baselines; a 1-NN classifier trained on color histograms and random guessing. For the tiger detector run on Corel, the color histogram does quite well, suggesting we should look at the Corel dataset with suspicion. We show that, in general, shape improves detection performance. Comparing our zebra and giraffe detection results to [60, 98], we show *better performance on a demonstrably harder dataset*.

'deer', 'elephant', 'monkey','antelope', 'parrot', and 'polar bear').

Detection. We show precision-recall (PR) curves in Figure 3.19. For the **S**hape detector, we build an animal detector using *only* the video and not Hemera. We build a crude texture library using positive and negative patches inside and outside the spatio-temporal tracks. Given a new image, we construct a binary label image by replacing patches with their closest match from this limited texture library. We then use DP to find the MAP configuration of limbs from the binary label image. For the **T**exture detector, we build a detector using *only* Hemera and not the video. We compute a binary label image using the

entire patch library (Hemera animal patches plus background patches). Our final detector is a threshold on the sum of animal pixels (as in [60, 98]). For the **S & T** detector, we construct a binary label image using the entire patch library, and then use DP to find the MAP limb configuration. We compare our detectors with 2 baselines; a 1-NN classifier trained on color histograms and random guessing. We tried a variety of other classifiers as baselines (such as logistic regression and SVMs) but 1-NN performed the best.

Difficulty of datasets. Recognition is still relatively poorly understood, meaning that reports of absurdly high recognition rates can usually be ascribed to simplicity of the test set. Careful experimentation requires determining how difficult a dataset is; to do so, one should assess how simple baselines perform on that dataset [20, 28, 78, 84]. This is often informative: for example, it is known that variations in reported performance between different face recognition algorithms are almost entirely explained by variations in the performance of the baseline on the dataset [84]. In almost all cases, our shape and texture animal models outperform the baselines of random guessing and color histogram classification. The one notable exception is our tiger detector on the Corel data, for which a color histogram outperforms all our methods. This can be ascribed to the insufficiently well known fact that Corel backgrounds are strongly correlated with Corel foregrounds (so that a Corel CD number can be predicted using simple color histogram features [20]). In the Google set, our baselines do worse, but our detectors do better. This *negative* correlation seems to stem from the fact that Google images have varied backgrounds, unlike Corel images (see Figure 3.21 versus Figure 3.22). Such backgrounds hurt our global histogram

baseline but may help our animal detector (since the animal might be easier to segment). Comparing to detection results reported in [60, 98], we obtain *better performance on a demonstrably harder dataset*.

Importance of shape. In almost all cases, adding shape greatly improves detection accuracy. An exception is detecting tigers in the Google set (Figure 3.19). We believe this is the case because of severe changes in scale; many tiger pictures are head shots, for which our shape model is not a good match (this also confuses our texture model, resulting in the lower overall performance). However, for low recall rates, shape is still useful in yielding high precision. The top few matches for the tiger detector will be tigers only if we use shape as a cue. Our results for shape are particularly impressive given the quality of our texture detector baseline. It has been shown that feature matching with SIFT features [28] produces quite good performance on established object recognition datasets [33]. Such a scheme is equivalent to our texture baseline, which we demonstrate is outperformed by our shape and texture detector.

Location and kinematic recovery. Looking at the best matches to our detectors (Figure 3.21 and Figure 3.22), we see that we reliably localize the detected animal and quite often we recover the correct configuration of limbs. We quantify this by manually evaluating the recovered configurations in Table 3.2. We define a correct localization to occur when a majority of the pixels covered by the estimated limbs are animal pixels (if we shoot at the estimated limbs, we'll most likely hit the animal). We define a kinematic recovery as correct when a majority of the limbs overlap a pixel region with the correct

Percentage of correct localizations

Dataset	Zebra	Tiger	Giraffe
Corel	84.9	92.0	76.9
Google	94.0	94.0	68.0

(a)

Percentage of correctly estimated kinematics

Dataset	Zebra	Tiger	Giraffe
Corel	24.2	28.0	38.4
Google	30.0	34.0	46.0

(b)

Table 3.2: Results for localization (**a**) and kinematic recovery (**b**). We define a correct localization to occur when a majority of the pixels within the estimated limbs are true animal pixels (we have a greater than 50% chance of hitting the animal if we shoot at the estimated limbs). We also show the percentage of animal images where the correct kinematics are recovered. By hand, we mark a configuration to be correct if a majority of the estimated limbs overlap a pixel region matching the semantic labeling from Figure 3.6. The kinematic results for the giraffe are impressive given the large number of different semantic labels; correct configurations tend to align the upper neck, the lower neck, the upper body, the lower body, the front leg, and the rear leg. Our animal detector localizes the animal quite well and often recovers reasonable configurations.

semantic label from Figure 3.6. The pose results for the giraffe are impressive given the large number of different semantic labels; correct configurations tend to align the upper neck, the lower neck, the upper body, the lower body, the front leg, and the rear leg. In general, we correctly localize the animal, and often we recover a reasonable estimate of its configuration, although we suffer from scale issues (see Figure 3.22).

Counting. We detect multiple instances of the same animal in a single image by finding the MAP animal configuration, masking away those pixels covered by the estimated limbs, and repeating. We are able to successfully classify 20% of the tiger pictures from our Corel set that contain one tiger as having one tiger. In general, we do quite poorly at counting because animals often occur together in herds; this confuses our greedy counting proce-

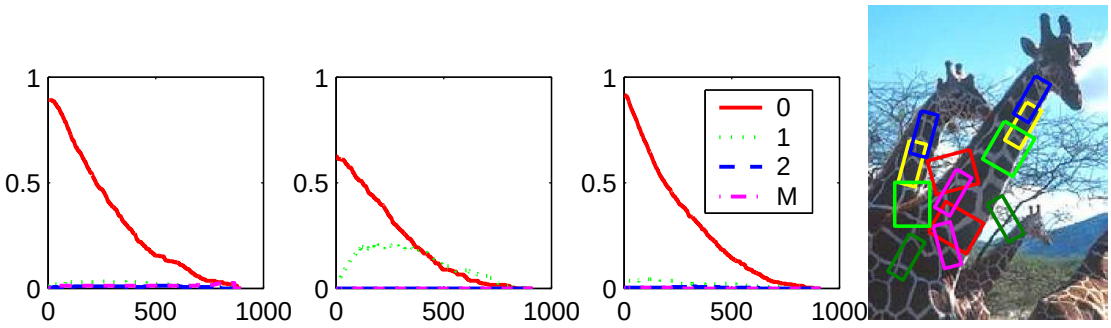


Figure 3.20: Counting results for the zebra (**left**), tiger (**middle left**), and giraffe (**middle right**) models. We plot results for Corel. We show fraction of images with 'i' animals that were correctly classified as a function of our detector threshold (where $i \in \{0, 1, 2, many\}$ and many is 3 or more). We see that 20% percent of tiger images can be correctly classified as having 1 tiger. However, since animals often appear in herds and overlap, counting them in general is a difficult problem. We show an example of a difficult image on the **right**. Depending upon how one scores partial occlusions and multiple scales, there could be two to four giraffes present. Counting appears to be a quite difficult object recognition task [53].

dure, which would work better on well-separated animals in an image. Counting remains a challenging problem for object recognition; relatively few systems have demonstrated results [53].

Another important application of accurate localization is the ability to apply *mutual exclusion*. Since our tiger detector often becomes confused by zebras, we would expect much better performance if, upon finding a zebra with our zebra detector, we masked away those pixels before applying the tiger detector. This strategy will only work with reasonably accurate localization.

3.7 Discussion

One contribution of this chapter is a novel (but simple) representation of texture; rather than using a histogram of textons, we represent texture with a patch of pixels. We demonstrate

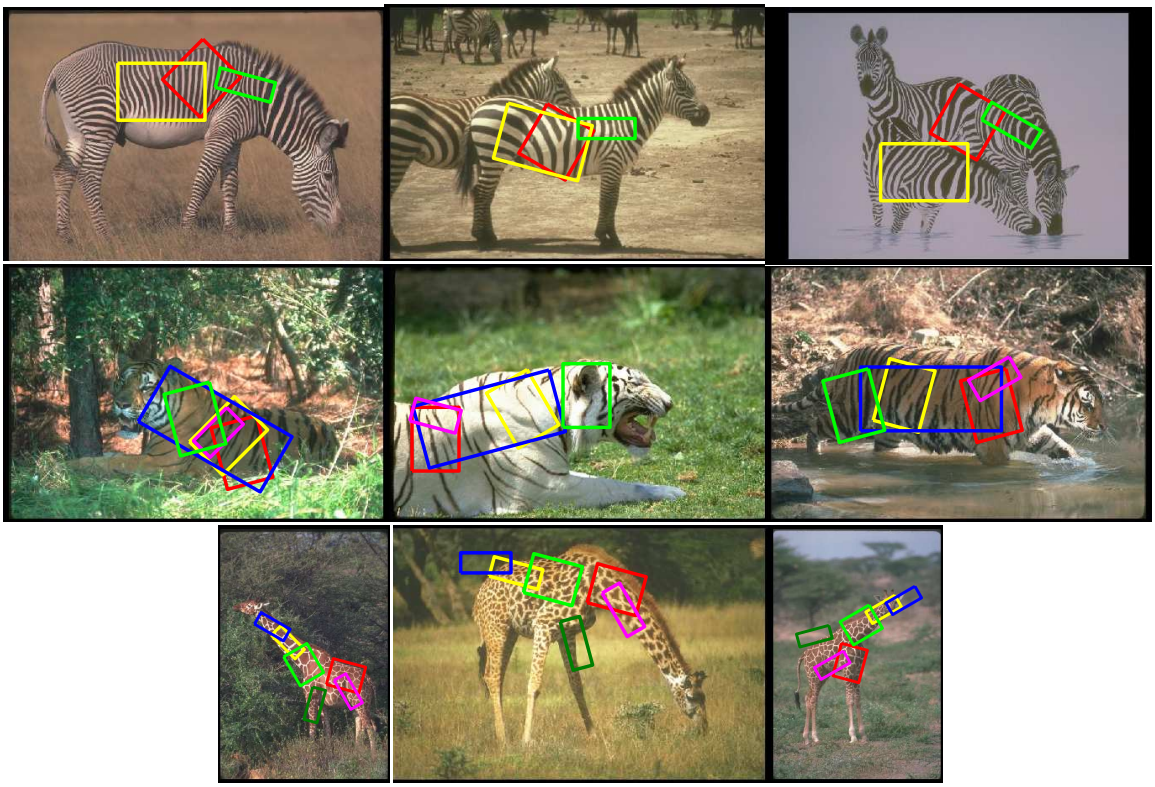


Figure 3.21: Results for our zebra (**top row**), tiger (**middle row**), and giraffe (**bottom row**) models using shape and texture on a test pool of 304 Corel animal images. We show the top scoring detections for each detector. Even though this dataset is relatively easy for detection (by evidence of good baseline performance), we can still evaluate localization and kinematic recovery results. We localize the animal quite well, and often recover reasonable kinematic estimates (though sometimes we have trouble determining which direction an animal is facing).

that this representation outperforms the state-of-the-art for our task of detecting animals.

Broadly speaking, we introduce (and rigorously evaluate) an unsupervised system for learning articulated models using video. Video is useful because both motion and appearance consistency are strong cues for learning. Such cues allow us to learn fairly complex pictorial structures with internal kinematics. These models allow us to track a deforming animal in a video *automatically* and to identify the animal from an image library. One

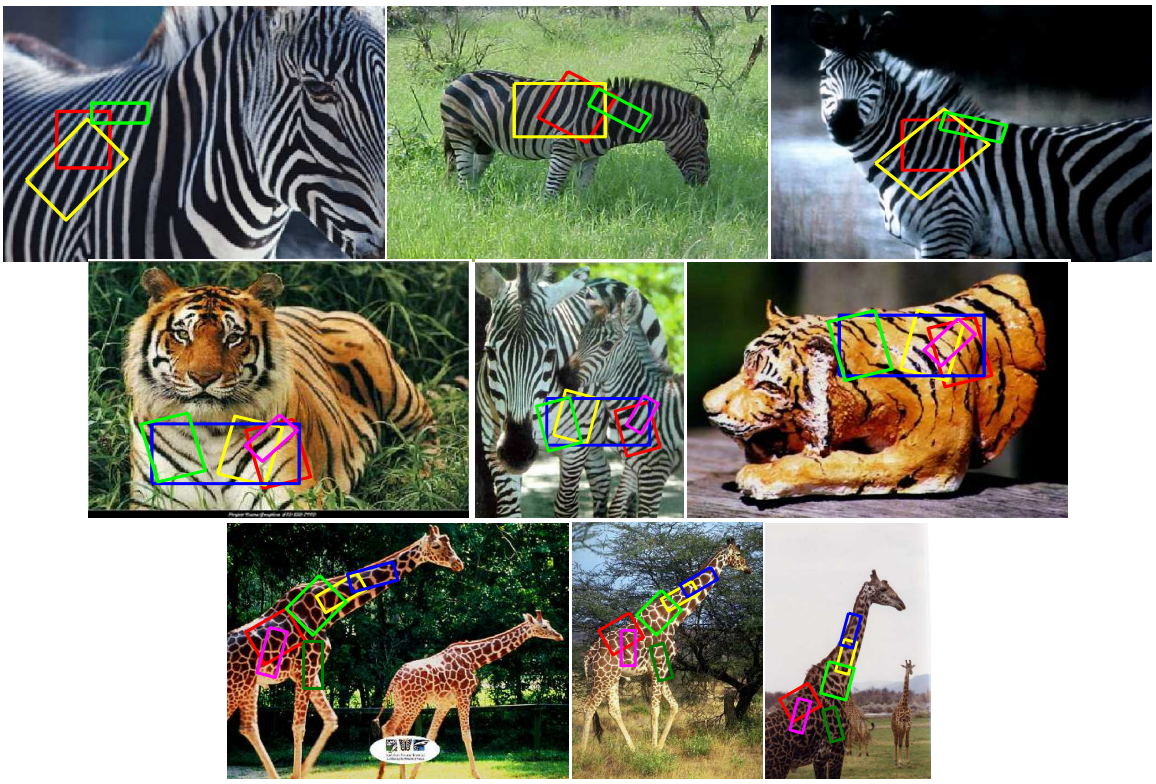


Figure 3.22: Results for our zebra (**top row**), tiger (**middle row**), and giraffe (**bottom row**) models using shape and texture on a test pool of 1418 animal images obtained from Google. We show the top scoring detections for each detector. Our tiger model mistakenly fires on a Google zebra due to the similar texture. The quasi-correct zebra configurations suggest our shape model might perform better if we searched over scale. The giraffe configurations tend to be quite good. The Google results are impressive given the poor performance of our baselines; we are detecting, localizing, and often recovering reasonable pose estimates for objects in a dataset *demonstrably hard for object recognition*.

would also hope to use the models to find animals in new images. This turns out to be hard because of a fundamental limitation of video; only a single object instance is observed, and so the learned appearance is too specific. We show a useful strategy of combining models learned from video and image collections (where multiple instances are observed). These learned models appear promising for recognition tasks beyond detection, such as localization, kinematic recovery, and (possibly) counting.

Appendix 3.A Clustering as inference

In this section, we show the equivalence of clustering and inferring appearance of C. The mean shift algorithm from Section 3.1.2 finds modes in the log posterior on C given image patches from a sequence $Im_t(P_t)$. For tractability, we only consider a finite set of image positions $\{\hat{P}_t\}$ detected by a segment finder. Throughout this section, we ignore additive and multiplicative constants since scaling and offsetting a function does change the location of its extrema. We follow the same naming conventions of [22].

Using standard max-product equations, the max posterior can be written as

$$
\begin{aligned}
m(C) &= \prod_t \max_{\hat{P}_t} \exp^{-||Im_t(\hat{P}_t)-C||^2} && (3.2)\\
\log(m(C)) &= \sum_t \max_{\hat{P}_t} \{-||Im_t(\hat{P}_t)-C||^2\} && (3.3)\\
&= \sum_t \max_{\hat{P}_t} k_h(||Im_t(\hat{P}_t)-C||^2). && (3.4)
\end{aligned}
$$

In Equation 3.4, we generalize the negative squared error term to be a robust m estimator

$$
\begin{aligned}
k_h(||x-y||^2) &= -\min(||x-y||^2, h) && (3.5)\\
&= \max(1-\frac{||x-y||^2}{h}, 0). && (3.6)
\end{aligned}
$$

Since we expect our detector to miss the true limb in some frames, we wish to only pay a truncated cost h for those missed detections. Massaging $k_h(x)$ into Equation 3.6 (by shifting and scaling by h) emphasizes the relationship to an epanechnikov kernel; this connection between m-estimators and kernel functions has been drawn before [22].

Equation 3.4 looks like a Parzen's window estimate of C, except for the max term. The mean shift algorithm is a gradient ascent algorithm for finding local modes in Parzen's window estimates; we show a modified mean shift procedure that finds modes of Equation 3.4:

$$C_{k+1} = \frac{\sum_t Im_t^*(C_k) g_h(||Im_t^*(C_k) - C_k||^2)}{\sum_t g_h(||Im_t^*(C_k) - C_k)||^2)}.$$

where

$$\begin{aligned} g_h(x) &= -\frac{\delta k_h(x)}{\delta x} && (3.7)\\ &= \mathcal{I}(x < h) && (3.8)\\ Im_t^*(C_k) &= \underset{Im_t(\hat{P}_t)}{\text{argmax}}\ k_h(||Im_t(\hat{P}_t) - C_k||^2) && (3.9)\\ &= \underset{Im_t(\hat{P}_t)}{\text{argmin}}\ ||Im_t(\hat{P}_t) - C_k||^2, && (3.10) \end{aligned}$$

where $\mathcal{I}$ is the standard identity function, and $Im_t^*(C_k)$ is the image patch from each frame closest to C_k. The resulting mode-finding algorithm is quite simple;

1. Initialize a guess $C_k = C_1$.

2. Find the set of detected patches within a radius of h of C_k.

3. If there are multiple patches from the same frame, only keep the closest patch to C_k.

4. Set C_{k+1} to be the average of the set, and if $||C_{k+1} - C_k|| > \epsilon$, goto Step 2.

We initialize the search C_1 to each detected patch, and denote the various convergence points of modes on $m(C)$. We show a proof of convergence in the next section.

3.A.1 Proof of Convergence

At iteration k of the algorithm, we are at a current estimate C_k. Let us collect the closest patches from each frame $Im_t^*(C_k)$ (Equation 3.10). We can use them to construct an approximate Parzen's window estimate of the log posterior

$$f_k(C) = \sum_t k_h(||Im_t^*(C_k) - C||^2).$$

Computing C_{k+1} from Step 4 is exactly equivalent to performing a single *ordinary* mean shift step on the function $f_k(C)$ from the position C_k. This step maintains all the standard properties of the mean shift algorithm. For $C_{k+1} \neq C_k$, we can write

$$f_k(C_k) < f_k(C_{k+1}) \leq f_{k+1}(C_{k+1}).$$

The first inequality is a standard mean shift property, applied to the step on $f_k(C)$ [22]. The second is true because $f_{k+1}(C_{k+1})$ was constructed using the patches closest to C_{k+1}, and so any other choice of patches is sub-optimal. Since $\{f_k(C_k), f_{k+1}(C_{k+1}) \ldots\}$ is strictly

increasing and is upper-bounded by the number of frames, the series converges. We can also write

$$||f_{k+1}(C_{k+1}) - f_k(C_k)||^2 \geq ||f_k(C_{k+1}) - f_k(C_k)||^2 \tag{3.11}$$

$$\geq M||C_{k+1} - C_k||^2, \tag{3.12}$$

for some strictly positive constant M. The first inequality is true because $f_{k+1}(C_{k+1}) \geq f_k(C_{k+1})$ and the second is a standard mean shift property, applied to the step on $f_k(C)$ [22]. The above result, combined with the convergence of $\{f_k(C_k), f_{k+1}(C_{k+1}) \ldots\}$, implies $\{C_k, C_{k+1} \ldots\}$ is convergent.

Chapter 4

Building Models of People

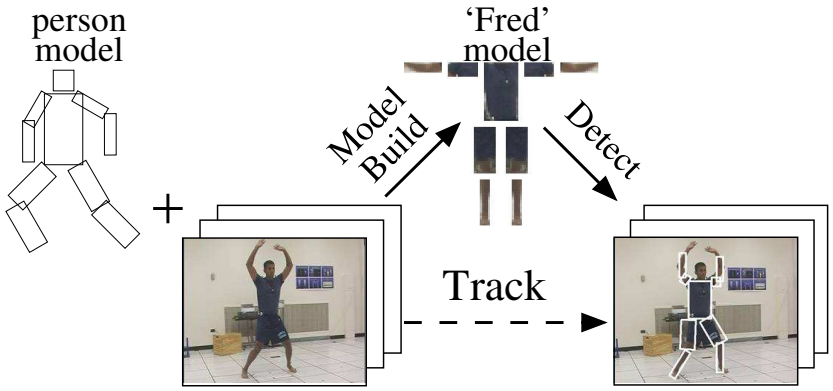


Figure 4.1: We use a model-based people tracker. Initially, we use a detuned edge-template as a generic person model. From the video data, we **build** an instance-specific model capturing a person's appearance. We then track by **detecting** that model in each frame.

This chapter describes a system that, given a video sequence of possibly one or more people, automatically tracks each person. Our fundamental assumption is that coherence in appearance is a stronger cue to body configuration than dynamics because body segments may move very fast but it takes time to change clothes. This suggests the following two

part strategy. We first **build** a model of what the person looks like, and then we track by **detecting** that model in each frame.

We use a pictorial structure representation that models the human body as a puppet of rectangles (Figure 4.1). We describe the full model in Section 4.1. Since methods for detecting pictorial structures are well understood [31], we focus on algorithms for learning them (from a given video sequence). The previous chapter details an algorithm for learning arbitrary pictorial structures from simple videos (where we know a single object was present). To track multiple people, we exploit the fact that *a priori* we know the geometric model $\Pr(P^i|P^j)$ for a human body: a torso is connected to two arms and legs, etc. Hence all we need to learn is the appearance of each part $\Pr(Im(P^i))$.

We describe two methods of building appearance models. One is a **bottom-up** approach that looks for candidate body parts in each frame; this is a straightforward extension of the algorithm from Chapter 3. We cluster the candidates to find assemblies of parts that might be people (Section 4.2). Another is a **top-down** approach that looks for an entire person in a single frame. This method directly exploits the known geometric model $\Pr(P^i|P^j)$; we know people tend to occupy certain key poses, and so we build models from those poses that are easy to detect (Section 4.3). Once we have learned an appearance model by either method, we detect it in each frame to track a person (Section 4.4). We finally conclude with results in Section 4.5.

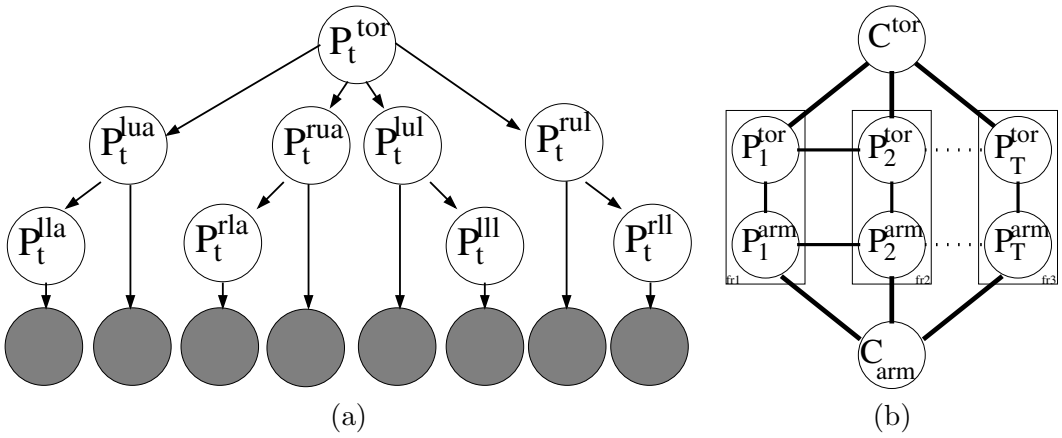


Figure 4.2: Our person model (**a**) is a tree pictorial structure. This model is parameterized by probability distributions capturing geometric arrangement of parts $\Pr(P_t^i|P_t^j)$ and local part appearances $\Pr(Im_t(P_t^i)|P_t^i, C^i)$ (the vertical arrows into the shaded nodes). Our full temporal model replicates (a) for each frame t, adds in a part motion model $\Pr(P_{t+1}^i|P_t^i)$, and explicitly models part appearance C^i. We show the full model for a torso-arm assembly in (**b**) (marginalizing out the image observations for convenience).

4.1 Temporal Pictorial Structures

Our basic representation is a pictorial structure [31] (see Figure 4.2-(a)). We replicate the standard model T times, once for each frame:

$$\Pr(P_{1:T}^{1:N}, Im_{1:T}|C^{1:N}) = \prod_t^T \prod_i^N \Pr(P_t^i|P_{t-1}^i) \Pr(P_t^i|P_t^{\pi(i)}) \Pr(Im_t(P_t^i)|P_t^i, C^i). \quad (4.1)$$

As a notation convention, we use superscripts to denote body parts $i \in \{tor, arm, \ldots\}$ and subscripts to denote frames $t \in \{1 \ldots T\}$. The term $\pi(i)$ denotes the parent of part i, following the tree in Figure 4.2-(a). The variable P_t^i is a 3-vector capturing the position (x, y) and orientation θ of part i at time t. The first term in the right hand side (RHS) of

Equation 4.1 is a motion model for an individual part; the last two terms are the standard geometric and local image likelihood terms in a pictorial structure [31].

For a fixed sequence of images $Im_{1:T}$, we can interpret the RHS as an energy function of P_t^i and C^i. We visualize the function (for a torso-arm assembly) as an undirected graphical model in Figure 4.2-(b). Effectively, we want to find an arm position P_t^{arm} such that the arm lies nearby a torso $\Pr(P_t^{arm}|P_t^{tor})$, the arm lies near its position in the previous frame $\Pr(P_t^{arm}|P_{t-1}^{arm})$, and the local image patch looks like our arm model $\Pr(Im_t(P_t^{arm})|P_t^{arm}, C^{arm})$.

Our image likelihood models the local image patch with a Gaussian centered at the template C^i

$$\Psi_t(P_t^i, C^i) = \Pr(Im_t(P_t^i)|P_t^i, C^i) \tag{4.2}$$

$$\propto \exp^{-||Im_t(P_t^i)-C^i||^2}. \tag{4.3}$$

We ignore constants for simplicity. We model the spatial kinematics of the human body with a puppet of rectangles with freely rotating revolute joints, using potentials of the form

$$\Psi(P_t^{arm}, P_t^{tor}) = \Pr(P_t^{arm}|P_t^{tor}) \tag{4.4}$$

$$\propto \mathcal{I}(\mathcal{D}(P_t^{tor}, P_t^{arm}) < d_{max}), \tag{4.5}$$

where $\mathcal{I}$ is the standard identity function and $\mathcal{D}(P1, P2)$ is the distance between the hinge points for the two segments (as in [31]). For the upper leg segments, we add angular bounds

preventing them from pointing up into the torso.

Our motion model is bounded velocity

$$\Psi(P_t^i, P_{t-1}^i) = \Pr(P_t^i | P_{t-1}^i) \tag{4.6}$$

$$\propto \mathcal{I}(||P_t^i - P_{t-1}^i|| < v_{max}) \tag{4.7}$$

Finding an optimal track given a video sequence now corresponds to finding the maximum *a posteriori* (MAP) estimate of C_t^i and P_t^i from Figure 4.2-(b). Exact inference on this model is difficult for two reasons; (1) the graph contains large induced cliques [57] and (2) the state spaces of the variables are quite large. Surprisingly, *even if we know the appearance of a person*, exact inference is intractable. That is, even by shading in the C^i nodes from Figure 4.2-(b), the size of the induced cliques are exponential in the number of body parts; a nine-segment model proves far too complex. In addition, both P_t^i and C^i are discretized versions of underlying continuous quantities. We could directly work with continuous variables if we restricted our potentials to some parametric family (e.g., Gaussian), but we expect $\psi_t(C^i, P_t^i)$ to be highly multi-modal (we commonly encounter many torso and arm-like objects in the background).

Our model has both loopy structure and variables with large state spaces; an attractive strategy is to ignore the loops and pass local messages [124], and to represent those messages with a set of samples [54, 111]. Since we want MAP estimates, we will pass max-product messages [37]. We present two algorithms that, although seemingly quite different, are essentially the result of different message-passing *schedules*.

4.2 Building models by clustering

We present in this section an algorithm that passes messages for a single part i across a set of T frames. We pass messages across C^i until we learn the appearance of part i; we then pass messages for C^j until we learn j's appearance, and so on (until we learn the appearance of all body parts). We first procedurally describe the algorithm and then relate it to our model.

An important observation is that we have some *a priori* notion of part appearance C^i as having rectangular edges. We would like to refine this model to capture the full appearance of a part. This suggests the following approach:

1. *Detect* candidate parts in each frame with an edge-based part detector.
2. *Cluster* the resulting image patches to identify body parts that look similar across time.
3. *Prune* clusters that move too fast in some frames.

4.2.1 Detecting parts with edges

We use the same part detector as described in Section 3.1.1. We repeat the information here for clarity. We model body parts as cylinders which project to rectangles in an image. One might construct a rectangle detector using a Haar-like template of a light bar flanked by a dark background (Figure 4.3). To ensure a zero DC response, one would weight values in white by 2 and values in black by -1. To use the template as a detector, one convolves it

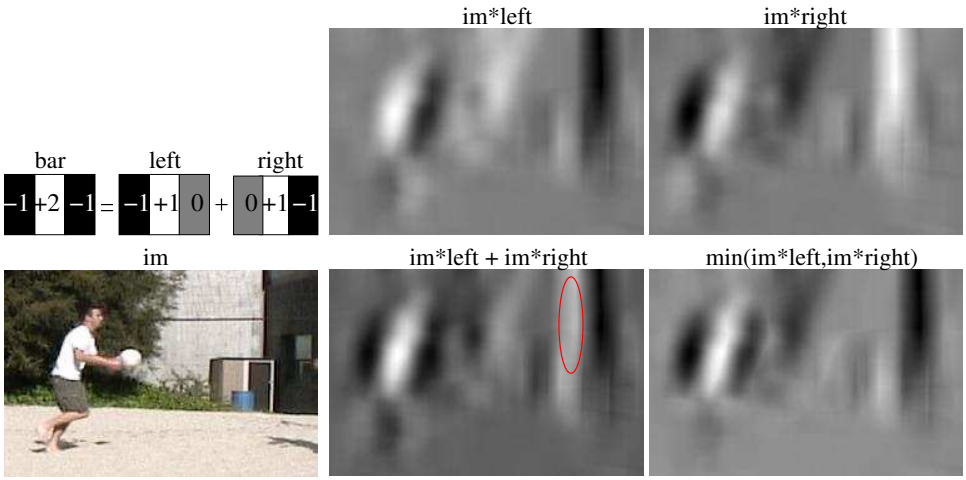


Figure 4.3: One can create a rectangle detector by convolving an image with a bar template and keeping locally maximal responses. A standard bar template can be written as the summation of a left and right edge template. Given the image on the **bottom left**, we convolve it with left and right edge templates (all images are scaled from [0,1]). The resulting bar detector suffers from many false positives, since *either* a strong left or right edge will trigger a detection (see the red ellipse in the **bottom middle**). A better strategy is to require *both* edges to be strong; such a response can be created by computing the minimum of the edge responses as opposed to the summation.

with an image and defines locally maximal responses above a threshold as detections. This convolution can be performed efficiently using integral images [118]. We observe that a bar template can be decomposed into a left and right edge template $f_{bar} = f_{left} + f_{right}$. By the linearity of convolution (denoted *), we can write the response as

$$im * f_{bar} = im * f_{left} + im * f_{right}$$

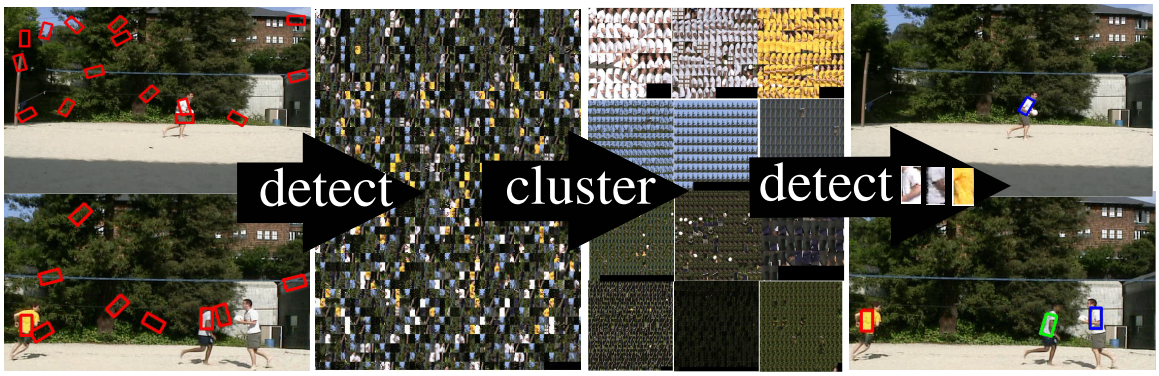
In practice, using this template results in many false positives since either a single left or right edge triggers the detector. We found taking a *minimum* of a left and right edge detector resulted in response function that (when non-maximum suppressed) produced more reliable detections

$$min(im * f_{left}, im * f_{right})$$

With judicious bookkeeping, we can use the same edge templates to find dark bars on light backgrounds. We assume we know the scale of people in a given video, and so search over a single scale for each body part. We expect our local detectors to suffer from false positives and missed detections, such as those shown in Figure 4.4-(a).

4.2.2 Clustering image patches

Since we do not know the number of people in a video (or, for that matter, the number of segment-like things in the background), we do not know the number of clusters *a priori*. Hence, clustering segments with parametric methods like Gaussian mixture models or k-means is difficult. We opted for the mean-shift procedure [22], a non-parametric density



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

Figure 4.4: Building a model of torso appearance. We detect candidate torsos using an edge template on the **left**. We cluster the candidate image patches to enforce a constant appearance in the **center**, and then prune away those clusters that do not move (all but the top 3). We now use the medoid image patch from each cluster as templates to find the (dynamically valid) torso tracks on the **right**. Note that since we are using appearance and not edges to find segments, we can track against weak contrast backgrounds (the blue segment on the **lower right**).

estimation technique.

We create a feature vector for each candidate segment, consisting of a 512 dimensional RGB color histogram (8 bins for each color axis). Further cues — for example, image texture — might be added by extending the feature vector, but appear unnecessary for clustering.

Identifying segments with a coherent appearance across time involves finding points in this feature space that are (a) close and (b) from different frames. The mean-shift procedure is an iterative scheme in which we find the mean position of all feature points within a hypersphere of radius h, recenter the hypersphere around the new mean, and repeat until convergence. We initialize this procedure at each original feature point, and regard the resulting points of convergence as cluster centers. For example, for the sequence in Fig

4.4, starting from each original segment patch yields 12 points of convergence (denoted by the centers of the 12 clusters in (c)).

As a post-processing step we greedily merge clusters which contain members within h of each other (starting with the two closest clusters). We account for over-merging of clusters by extracting multiple valid sequences from each cluster during step (c). For each cluster we keep extracting sequences of sufficient length until none are left; this is explained further in Section 4.2.3. Hence, for a single arm appearance cluster, we might discover two valid tracks of a left and right arm.

4.2.3 Enforcing a motion model

For each cluster, we want to find a sequence of candidates that obeys our bounded velocity motion model. By fitting an appearance model to each cluster (typically a Gaussian, with mean at the cluster mean and standard deviation computed from the cluster), we can formulate this optimization as a straightforward dynamic programming problem. Let P_t be the position of a segment in the t^{th} frame. We assume that these have a Markovian behavior; i.e. $Pr(P_t|P_{1:t-1}) = Pr(P_t|P_{t-1})$. The reward for a given candidate is its likelihood under the Gaussian appearance model, and the temporal rewards are '0' for links violating our velocity bounds and '1' otherwise. We add a dummy candidate to each frame to represent a "no match" state with a fixed charge. By applying dynamic programming, we obtain a sequence of segments, at most one per frame, where the segments are within a fixed velocity bound of one another and where *all* lie close to the cluster center in appearance.

We finally prune the sequences that are too small or that *never move*. The claim that we

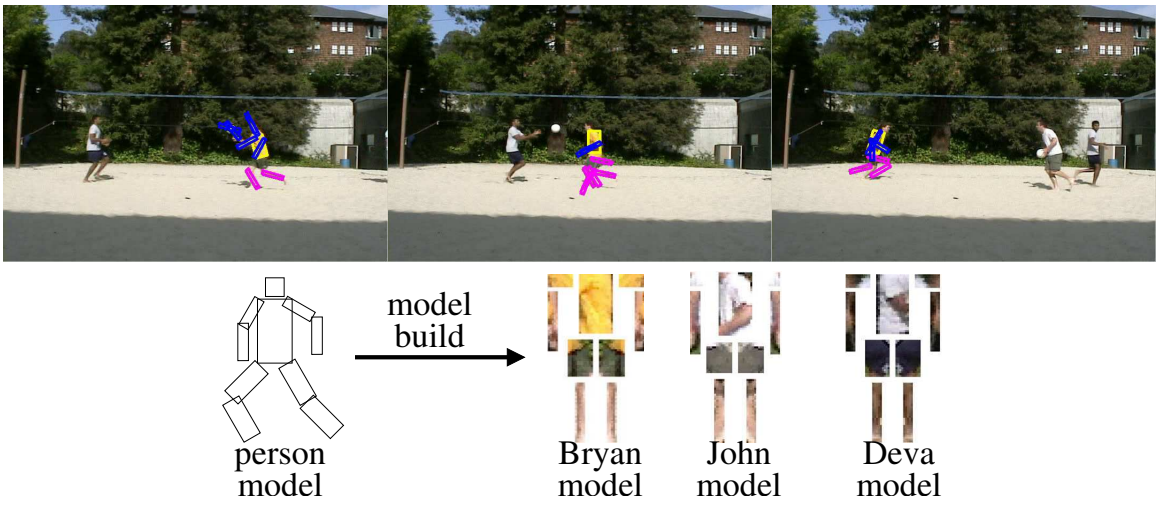


Figure 4.5: Given the clusters from Figure 4.4, we search near each of the valid torso clusters for candidate arms and legs (using our segment detector set at a smaller scale). We use the spatial model from our person detector to limit the search space; we only look for arms near the top of torsos. We show 3 frames of candidate detections from the yellow-shirt cluster. We cluster the new candidates to learn arm and leg appearances. Repeating for each torso cluster, we build a "John", "Bryan", and "Deva" detector.

should only concern ourselves with segments that are coherent over time and that move (a notion we call *foreground enhancement*) is markedly different from traditional background subtraction since it is used to learn appearance and not to find people. Once the appearance is known, we can track people who stand still (so long as they move at some point).

4.2.4 Learning multiple appearance models

We use the learned appearance to build better segment detectors; e.g., we now know that the torso is a yellow rectangle, rather than just two parallel edges. We search for *new* candidates using the medoid image patch of the valid clusters from Figure 4.4-(c) as a template. We link up those candidates that obey our velocity constraints into the final

torso track in Figure 4.4-(c). Since we found 3 valid torso clusters, we have 3 final torso tracks. We then search near *each* of the torso tracks for candidate arm and leg segments. This step exploits our geometric model $\Psi(P_t^{arm}, P_t^{tor})$ from Equation 4.5; we know arms lie near the top of torsos and legs lie near the bottom. We repeat the clustering procedure to learn appearance models for those body parts. Using the medoid image patch from the largest arm and leg clusters as templates, we find new arm and leg tracks near each of the existing torso tracks (Figure 4.5). We now have constructed multiple appearance templates for each person in a video. Effectively, we have taken our initial person detector, which consisted of a deformable template of parallel edges, and built from the the video a collection of *person-specific* people detectors (a "John", a "Bryan", and a "Deva" detector).

We now can run these detectors independently in each frame to obtain the final tracks shown in our results (Section 4.5.1). Note that by building instance-specific object models, we can perform **multiple-object tracking as detection**. For long sequences, we build appearance models by clustering only an initial subset of the frames. We found that using 50-100 frames suffices, though we explore this point further in Section 4.5.1. Given the learned models, we then use them in an online fashion, detecting them in new frames as they arrive.

4.2.5 Approximate Inference

We now cast our algorithm in light of our model from Figure 4.2-(b). We visualize our message-passing schedule with a set of trees in Figure 4.6-(a). We show in Appendix 3.A that the mean shift clustering algorithm finds modes in the posterior of C^i, as computed

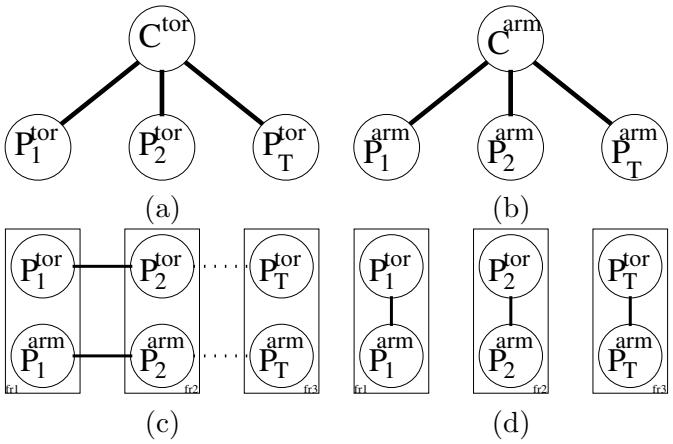


Figure 4.6: A set of trees for loopy inference on Figure 4.2-(b). Trees (a) and (b) learn and apply torso and arm appearance templates. Tree (c) enforces our motion model. Tree (d) restricts arm and torso patches to those which obey our kinematic constraints.

by the tree. We interpret each torso cluster as a *unique* person, instantiating one copy of the model from Figure 4.2-(b) for each cluster. For each instantiation, we use the mode estimate (the medoid of the cluster) to construct a new torso likelihood template. We use this template to perform inference on the remaining trees from Figure 4.6. We perform inference on tree (c) to find a sequence of torso detections that all look like the template and that move smoothly; this is our previous dynamic programming step from Section 4.2.3. We then infer on tree (c) to find a set of arm segments near the found torsos. We then infer on tree (d) to learn an arm appearance. We finally end with the arm chain in tree (b) to extract a dynamically valid arm sequence.

For the entire 9 segment human body, we learn appearance in an order which reflects the quality of our segment detectors. Our torso detector performs the best, followed by the lower arm & leg detectors (since they are more likely to lie away from the body and have a cleaner edge profile), and finally the upper arm & legs. The upper limbs are hard to detect,

so we constrain their position by first detecting and learning the appearance of the torso and lower limbs.

4.3 Building models with stylized detectors

The approach of clustering part detectors works well when parts are reliable detected. However, building a reliable part detector is hard, a well-known difficulty of **bottom-up** approaches. An alternative strategy is to look for an entire person in single frame. This is difficult because people are hard to detect because of variability in shape, pose, and clothing, a well-known difficulty of **top-down** approaches.

We could detect people by restricting our temporal pictorial structure from Equation 4.1 to a single time slice. Such a pictorial-structure detector can cope with pose variation. But since we do not know part appearances C^i *a priori*, we must use generic edge templates as part models. Such a detector will be confused by background clutter (see Figure 4.21).

However, our detector is not trying to "detect" a person, but rather build a model of appearance. This is an important distinction because typically one wants detectors with high precision and recall performance. In our case, we want a person detector with rather unique properties: (a) it must accurately localize limbs (since we will use the estimated limbs to build appearance models) and (b) it should have high precision (we want most detections to be of people). Given both, we can tolerate a low recall rate since we can use the learned appearance models to find the figure in those frames where the detector failed [92].

Figure 4.7: What poses are easy to detect and build appearance from? We cannot learn appearance when body parts are occluded. People are hard to detect when they occupy an awkward pose, suffer from motion blur, look like the background, or are too small. As such, we build a stylized detector for large people in lateral walking poses (**bottom right**).

We build a person detector that only detects people in typical poses. Even though the detector will not fire on atypical poses, we can use the appearance learned from the standard poses to track in those atypical frames. This notion of **opportunistic detection** states that we can choose those poses we want to detect. This way we concentrate our efforts on easy poses rather than expending considerable effort on difficult ones. Convenient poses are ones that are (a) easy to detect and (b) easy to learn appearance from (see Figure 4.7). For example, consider a person walking in a lateral direction; their legs form a distinctive scissor pattern that one tends not to find in backgrounds. The same pose is also fairly easy to learn appearance from since there is little self-occlusion; both the legs and arms are swinging away from the body. Following our observations, we build a single-frame people detector that finds people in the mid-stance of a lateral-walk.

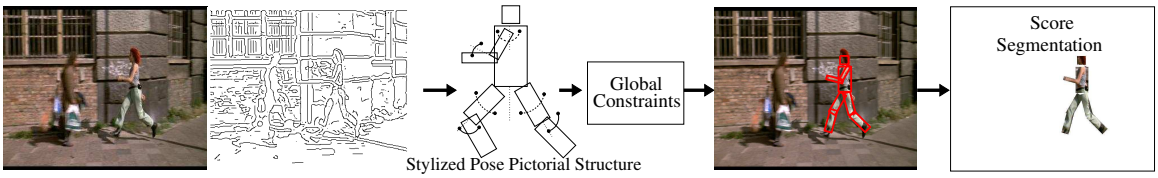


Figure 4.8: Our lateral-walking pose finder. Given an edge image on the **left**, we search for a tree pictorial structure [31] using rectangle chamfer template costs to construct limb likelihoods. We restrict limbs to be positioned and oriented within bounded intervals consistent with walking left. We set these bounds (designated by the arcs overlaid on the model) by hand. We also search a mirror-flipped version of the image to find people walking right. To enforce global constraints (left and right legs should look similar), we sample from the pictorial structure posterior (using the efficient method of [31]), and re-compute a global score for the sampled configurations. The best configuration is shown on the **right**. In general, this procedure also finds walking poses in textured backgrounds; to prune away such false detections, we re-evaluate the score by computing the goodness of a segmentation into person/non-person pixels. We do this by building an appearance model for each limb (as in Figure 4.9) and then use the model to classify pixels from this image. We define the final cost of a walking-pose detection to be the number of mis-classified pixels.

Our system initially detects a lateral-walking pose with a stylized detector (Section 4.3.1). That detection segments an image into person/background pixels. This allows us to build a **discriminative** appearance model (Section 4.3.2) – we learn the features that discriminate the figure from its background (and assume those features will also discriminate the figure in other frames).

4.3.1 Detecting lateral walking poses

An overview of our approach to people detection is found in Figure 4.8. We will use a sequence from the film "Run Lola Run" as our running example (pun intended). We construct a (stylized) person detector by restricting our full temporal model from Figure

4.2-(b) to a single frame. We write a single-frame pictorial structure model as:

$$\Pr(P^{1:N}, Im|C^{1:N}) = \prod_{i}^{N} \Pr(P^i|P^{\pi(i)}) \Pr(Im(P^i)|P^i, C^i). \tag{4.8}$$

We use an 8-part model, searching for only one arm since we assume the other arm will be occluded in our lateral walking pose. We modify our geometric and image likelihood terms (originally defined in Section 4.1) to look for stylized poses.

$\mathbf{Pr(P^i|P^{\pi(i)})}$: We manually set our kinematic shape potentials to be uniform within a bounded range consistent with walking laterally (Figure 4.8). For example, we force θ for our upper legs to be between 45 and 15 degrees with respect to the torso axis. We do not allow them to be 0 degrees because we want to detect people in a distinctive scissor-leg pattern. Learning these potentials automatically from data is interesting future work.

$\mathbf{Pr(Im(P^i)|P^i, C^i)}$: We evaluate the local image likelihood with a chamfer template edge mask [113]. Our part template C^i must be invariant to clothing variations and so we use a rectangular edge template (Figure 4.8). Given an image, the chamfer cost of an edge template is the average distance between each edge in the template and the closest edge in the image. We compute this efficiently by convolving the distance-transformed edge image with the edge template. To exploit edge orientation cues, we quantize edge pixels into one of 12 orientations and compute the chamfer cost separately for each orientation (and add the costs together). To capture the deformations from Figure 4.8, we convolve using rotated versions of our templates.

Since we use our lateral-walking detector in a high-precision/low-recall regime, we need

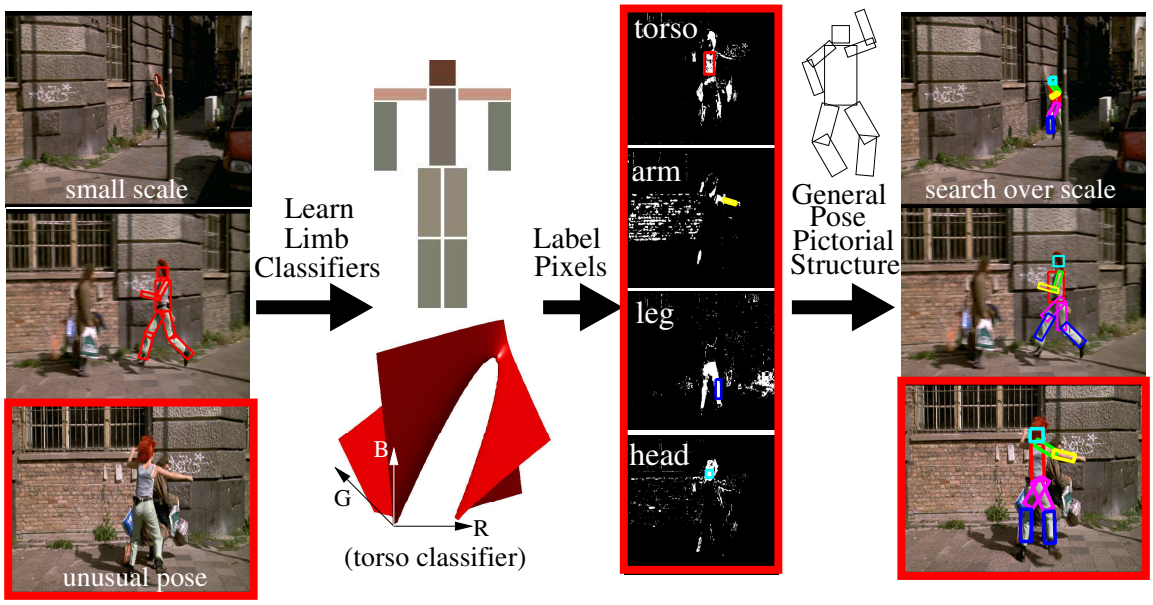


Figure 4.9: An overview of our approach; given a video sequence, we run a single-scale walking pose detector on each frame. Our detector fails on the small scale figure and the on a-typical pose, but correctly detects the walking pose (**left**). Given the estimated limb positions from that detection, we learn a quadratic logistic regression classifier for each limb in RGB space, using the masked limb pixels as positives and all non-person pixels as negatives. In the **middle left**, we show the learned decision boundary for the torso and crudely visualize the remaining limb classifiers with a Gaussian fit to the positive pixels. Note that the visual models appear to be poor; many models look like the background because some of the limb pixels happen to be in shadow. The classifiers are successful precisely because they learn to ignore these pixels (since they do not help discriminate between positive and negative examples). We then run the classifiers on *all* frames from a sequence to obtain limb masks on the **middle right** (we show pixels from the third frame classified as torso, lower arm, lower leg, and head). We then search these masks for candidate limbs arranged in a pictorial structure [31], searching over general pose deformations at multiple scales. This yields the recovered configurations on the **right**. We show additional frames in Figure 4.19

to look only at those configurations where all the limbs have high likelihoods. Before evaluating the kinematic potentials, we perform non-maximum suppression on the chamfer likelihood response functions (and only keep candidate limbs above a likelihood threshold). We also throw away arm candidates that are vertical or horizontal (since there tends to be

many vertical and horizontal rectangles in images of man-made structures). This is again justified for high-precision/low-recall detection; even though the arm of a person may in fact be horizontal or vertical, we *choose* not to learn their appearance in this pose, since we would encounter many false positive detections (and build incorrect appearance models).

Global constraints: We found it useful to enforce global constraints in our person model. For example, left and right legs tend to be similar in appearance [75]. Also, our kinematic leg potentials still allow for overlap if the left leg happens to be translated over onto the right leg. These dependencies cannot be captured by a tree pictorial structure. Instead of finding the MAP estimate of Eq.4.8, we generate samples from the posterior (using the efficient method of [31]), and throw away those samples that violate our global constraints. We generate 2000 samples per image. To find configurations where the left and right legs look similar, we add the disparity in leg appearance (as measure by the $\mathcal{L}_2$ distance between color histograms) to the negative log probability of the sampled configuration. To force left and right legs to be far apart, we discard samples where leg endpoints are within a distance d of each other, where d is the width of the torso. We finally keep the sample with the lowest cost.

Segmentation score: Given an image with a laterally walking person, the procedure above tends to correctly localize the limbs of the figure. But it does not perform well as a people detector; it fires happily on textured regions. We add a region-based cue to the detection score. We can interpret the recovered figure as a proposed segmentation of the image (into person/non-person pixels), and directly evaluate the segmentation [75] as the

Figure 4.10: We show tracking results for a sequence with large changes in illumination. On the **left**, we show the frame on which our walking pose detector fired. Our system automatically learns discriminative limb appearance models from that *single* frame, and uses those models to track the figure when the background changes (**right**). This suggests that our logistic regression appearance model is quite generalizable. Note that since our system tracks by detection, it can track through partial and full occlusion.

final detection cost. Rather than use a standard segmentation measure, we adopt a simpler approach.

We build classifiers (in RGB space) for each limb, as described in Section 4.3.2. For each limb classifier, we create a test pool of limb pixels (from inside the corresponding limb mask) and background pixels (from identically-sized rectangles flanking both sides of the true limb). We then classify the all test pixels and define the cost of the segmentation to be the total number of misclassified pixels. This strategy would not work if we used classifiers with high Vapnik-Chervonenkis (VC) dimension (a nearest neighbor classifier always returns 0 errors when training and testing on the same data [115]). Restricting ourselves to a near-linear classifier (such as quadratic logistic regression) seems to address this issue. We threshold this final segmentation score to obtain good stylized-pose detections.

4.3.2 Discriminative appearance models

Since our person detector localizes a complete person in a single frame, we know both the person pixels *and* the non-person pixels. This suggests that we can build a discriminative

model of appearance. We assume each limb is (more or less) constant in color, and train a quadratic logistic regression classifier. We use all pixels inside the estimated limb rectangle as positives, and use all non-person pixels (i.e., those pixels not inside any limb mask) as negatives. Our appearance model for each limb is a quadratic surface that splits RGB space into limb/non-limb pixels (Figure 4.9). Recall that our set of limbs are the head, torso, upper/lower arm, and left/right upper/lower leg. We fit one model for the upper leg using examples from both left and right limbs (and similarly for the lower leg). We find our appearance models to be quite generalizable (see Figure 4.10).

4.4 Tracking by model detection

Given either model-building method (from Section 4.2 or 4.3), we now have a representation of the appearance of each part C^i. Since people tend to be symmetric in appearance, we maintain a single appearance for left and right limbs. The representation may be *generative* (a template patch) or *discriminative* (a classifier). We evaluate the local image (log) likelihood of a template patch using $\mathcal{L}_2$ distance of RGB histograms (Section 4.2.2). We evaluate the (log) likelihood of a classifier by summing up the number of misclassified pixels in a local image region (by running the rectangle templates of Section 4.2.1 on limb masks).

Since our learned part models C^i provide a good description of a person's appearance, we can localize a person quite well just by looking at a single frame. As such, we use a single frame pictorial structure (using the known part appearances) to detect each person in each frame.

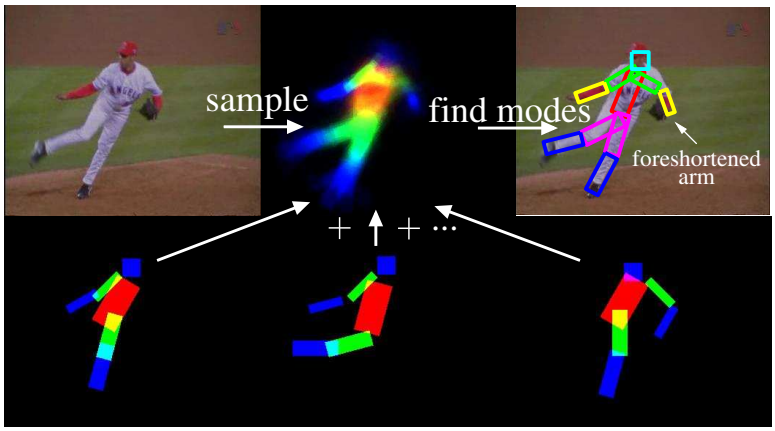


Figure 4.11: We detect people by sampling from a one-leg, one-arm pictorial structure $\Pr(P^{1:N}|C^{1:N}, Im)$. For the image on the **top left**, we can visualize the posterior (**top center**) by superimposing the samples (**bottom**). We find individual legs and arms by finding modes in the posterior; in images where only one arm is visible, we will find only one arm mode. Our mode-finding algorithm smooths the posterior. When we represent the samples in an appropriate pose space, smoothing allowing us to recover a foreshortened arm even when all sampled arms are too long (**top right**).

Occlusion: One practical difficulty of tracking people is dealing with self-occlusion; poses where we see both arms and legs are quite rare. Rather than use a formal occlusion model, we found the following procedure to work well. We draw 1000 samples from the posterior of $\Pr(P^{1:N}|Im, C^{1:N})$ for a *single arm, single leg* pictorial structure model. We use the efficient method of [31]. The samples tend to lie on actual arms and legs because of the quality of our limbs masks. We can visualize this posterior by rendering the samples on top of one another (Figure 4.11). Interestingly, we see two distinct legs in this map; they correspond to different *modes* in the posterior. The *uncertainty* in the matches captures the presence of two arms and two legs. We can explicitly find the modes with the meanshift algorithm [22]. Recall that this algorithm performs gradient ascent from an initial starting point. We represent each sample pose as a vector of 2D limb endpoints. We start from the

sample with the highest posterior value, and we typically need a few mean shift iterations to reach a mode. We remove those samples whose legs and arms overlap, and repeat to find a second mode (only keeping the new mode if its above some threshold).

Spatial Smoothing: The above procedure has a neat side affect; it spatially smooths the posterior function $\Pr(P^{1:N}|Im, C^{1:N})$. The amount of smoothing is proportional to the bandwidth of the meanshift procedure. We found a "smoothed mode" pose to be better than a direct MAP estimate in two regards: 1) the smoothed pose tends to be stable since nearby poses also have high posterior values and 2) the smoothed pose contains "sub-pixel" accuracy since it is a local average. Note that this is sub-pixel accuracy in the pose space. Recall that our poses are now represented as a vector of 2D joint positions. All poses contain arms of equal length, but, by averaging joint positions, we might obtain arms of different lengths. Interestingly, this averaging often captures the foreshortening of a limb (see Figure 4.11).

Temporal Smoothing: Independently estimating the pose at each frame has its obvious drawbacks; the final track will look jumpy. One can produce a smooth track by feeding the pose posterior at each frame (as represented by the 1000 samples) into a formal motion model. We perform local smoothing at a given frame by adding all samples from the previous and next frame when performing our mode-finding procedure. Our final pose estimate will then be a weighted average of nearby poses from nearby frames.

Multiple People: In general, we must account for multiple people in a video. The clustering procedure from Section 4.2 naturally handles multiple people with multiple clusters.

Given a set of walking-pose detections from Section 4.3, we need to establish automatically the number of different people that are actually present. For each detection, we learn a *generative* appearance model (by fitting a Gaussian in RGB space for each limb mask). This returns a vector of RGB values. We cluster these vectors to obtain sets of people models with similar appearance, again using the meanshift procedure. After obtaining clusters of similar looking people, we use positive and negative examples from across the cluster when training the logistic regression for each limb appearance. We then use these people models as described in the next two paragraphs.

Multiple instances: If a video has multiple people that look similar, our algorithms might only learn a single set of part models C^i (consider a video of a soccer team). In this case, when visualizing the posterior of the pictorial structure, we will see many modes corresponding to different instances. We use the same mode finding procedure to find a set of unique detections.

In general, we will have multiple appearance models, each possibly instanced multiple times. For each model, we independently find all instances of it in a frame. Many models will compete to explain the same or overlapping image regions. We use a simple greedy assignment; we first assign the best-scoring instance to the image pixels it covers. For all the remaining instances that do not overlap, we find the best-scoring one, assign it, and repeat. This greedy strategy works best when people are well separated in an image.

4.5 Experimental results

Typically, a primary criterion for evaluating the performance of a tracker is its expected time-to-failure. This pre-supposes that all trackers eventually fail — we argue this is not true. If one tracks a face by running a face detector on each frame, one can track for essentially *infinitely* long [118]. In this case, the quality of the track can measured by the quality of the face detector (in terms of correct detections and false positives). We extend this analysis to the detection of individual body parts. In probabilistic terms, we argue that when tracking with a weak motion model, performance is determined by the quality of the likelihood model (our person template).

4.5.1 Building models by clustering

We have tested our clustering algorithm on four different sequences (Table 4.1). "Jumping Jacks" and "Walk" were both taken indoors (at 15 *frames per second*, or fps) while the subjects were simultaneously motion captured. Ground truth for those sequences was obtained by registering the motion capture data with the video. We clustered an initial subset of the total frames; 75/100 and 150/288, respectively. "Street Pass" and "Weave Run" were both taken outdoors (at 30 fps), and ground truth was hand-labeled for a random subset of frames. For these sequences, we clustered the first 200/380 and 150/300 frames respectively. We show localization rates in Table 4.1. We define a part to be correctly localized when the majority of pixels covered by the estimated part have the correct labeling (we manually label a random set of 100 frames from a sequence). We only score the first arm and leg

% of frames correctly localized (clustering)

Sequence	Torso	Arm	Leg
J. Jacks	94.6	87.4	91.8
Walk	99.5	84.3	68.2
Street Pass	91.2	57.4	38.7
Weave Run	90.3	21.2	61.8

Table 4.1: Tracker performance on various sequences. We evaluate performance by looking at the percentage of frames where body parts have been correctly localized. Our torso rates are quite good, while rates for limbs suffer in the challenging outdoor sequences.

mode found for a given person detection. We localize torsos quite well, but limbs are hard, particularly in the challenging outdoor sequences.

Self-starting: None of these tracks were hand initialized. However, we do optimize thresholds for the segment detectors and the bandwidth for the mean-shift procedure for the sequences shown. More sophisticated segment detection and clustering algorithms may eliminate the need for tweaking.

Multiple activities: In Figure 4.12, we see frames from the two indoor sequences; "Jumping Jacks" and "Walk." In both left rows, we show the original edge-based candidates that clustered together. When limbs are close to the body or surrounded by a weak-contrast background, few candidates are found. By building an appearance model (using the surrounding frames where candidates *are* detected), we now can track using the learned appearance. Because we track by detection without a strong motion model, we can track both activities with the same system.

Lack of background subtraction: In Figure 4.13, we show frames from a sequence that contains a moving background. We still learn accurate appearance models, although varying lighting conditions often result in poor matches (as the poor localization results

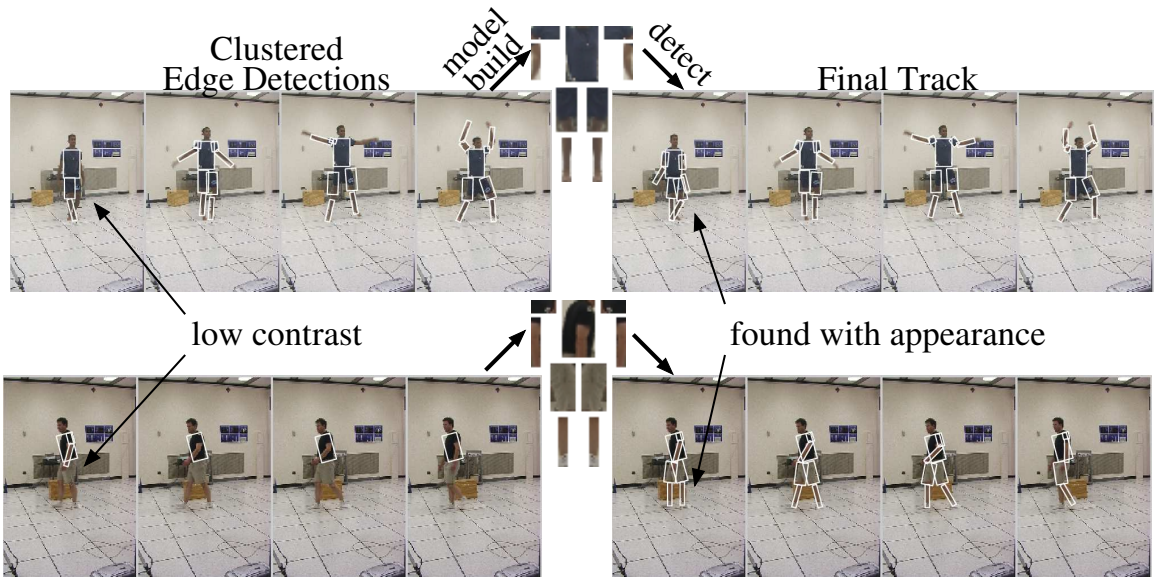


Figure 4.12: Self-starting tracker on "Jumping Jacks" (**top**) and "Walk" (**bottom**) sequences. In both **left** rows, we show the original edge-based candidates that clustered together. When limbs are close to the body or surrounded by a weak-contrast background, few candidates are found. By building an appearance model (using the surrounding frames where candidates *are* detected), we now can track using the learned appearance (**right**).

for arms and legs in Table 4.1). By using a metric more robust to lighting changes, the appearance models learned in the clustering and the segments found may be more accurate. Alternatively, one might add explicit illumination variables to C^i to deal with temporal changes in brightness.

Multiple people, recovery from occlusion and error: In Figure 4.14, we show frames from the "Weave Run" sequence, in which three figures are running in a weave fashion. In the top row, we see two tracks crossing. When the two figures lie on top of each other, we correctly disambiguate who is in front, and furthermore, recover the interrupted track of the occluded figure. In the bottom row, a track finds a false arm in the background but later recovers. We also see a new track being born, increasing the count of tracked

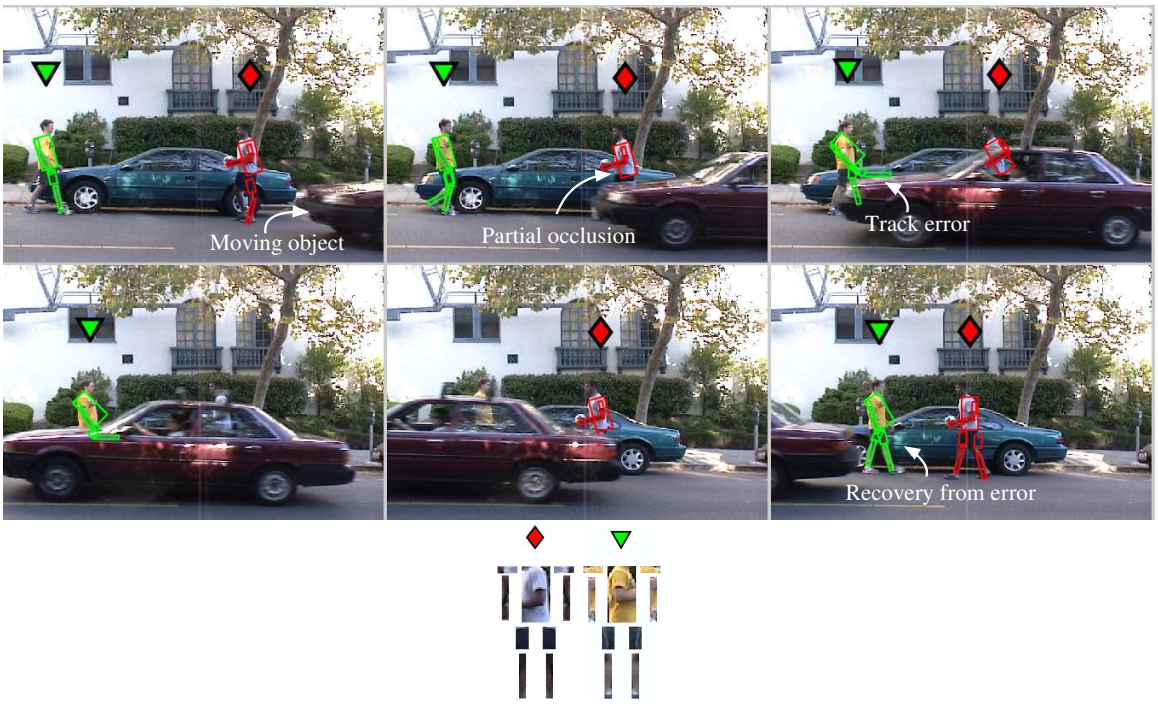


Figure 4.13: Self-starting tracker on "Street Pass" sequence containing multiple moving objects. The learned appearance templates are shown **below** select frames from the track. We denote individual tracks by a token displayed above the figure. The tracker successfully learns the correct number of appearance models, and does not mistake the moving car for a new person. We are also able to recover from partial and complete occlusion, as well as from errors in configuration (which drifting appearance models would typically fail on).

people to three.

Number of frames to cluster: For long sequences, we only cluster the first K frames. A natural question is how does K affect the final performance? We evaluate localization performance versus K in Figure 4.15 for the "Walk" sequence. We also show results for models learned with part detectors augmented with a skin classifier. Both detectors in Figure 4.15 happen to perform well for small K since our subject is initially against a uncluttered background. As we increase K and our subject walks into the room, our edge detectors pick up extraneous background candidates which cluster into poor appearance

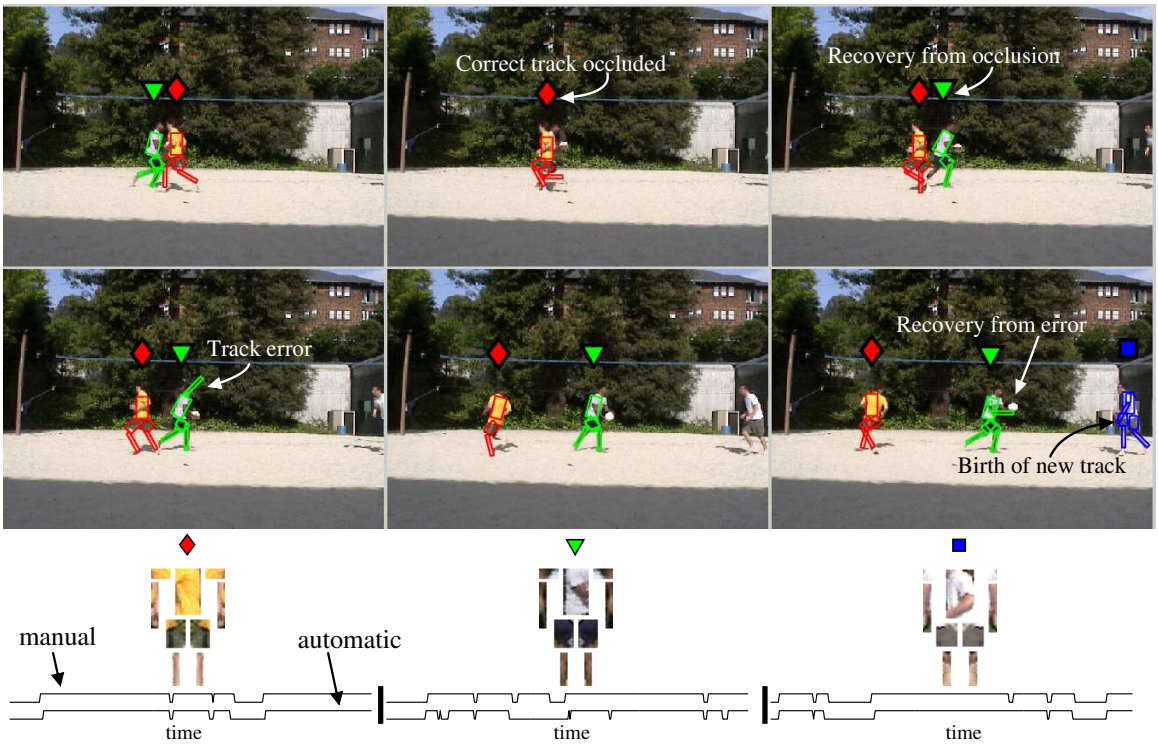


Figure 4.14: Self-starting tracker on "Weave Run". We show a subset of frames illustrating one figure passing another **above** the set of learned appearance templates. Note that the correct figure is occluded and that the correct track is recovered once it reappears. An earlier incorrect arm estimate is also fixed (this would prove difficult assuming a drifting appearance model). The final frame new track being born, increasing the count of found people to three. **Below** each figure we show detection signals, as a function of time, specifying when it is in view. The manual and automatic signal extracted from the tracker agree – the tracker tends to detect figures when they are in view and misses them when they are not.

models. However, both perform well as K is increased sufficiently. This result suggests that high-level clustering can compensate for poor low-level detection, given that we cluster over enough frames. Also note that performance does not change if we measure it on the initial set of K training frames or the entire sequence. This suggests that our learned appearance models generalize well.

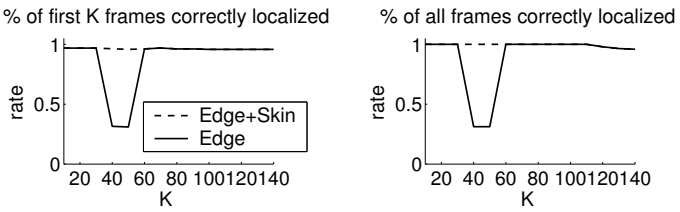


Figure 4.15: How do our generic part detectors affect performance? We plot performance for edge versus edge+skin detectors for the "Walk" sequence. We vary the initial size of K frames used to cluster and learn an appearance model. For small K, both detectors fortuitously learn a good model due to a lack of background clutter. As K increases, background clutter leads our edge detectors to construct poor appearance models. For large K, clustering yields working models irrespective of the detectors. Note that performance is equivalent when measured on the initial training frames (**left**) or the entire sequence (**right**); this suggests that the learned appearance models generalize well.

4.5.2 Building models with a stylized detector

Our stylized pose system is faster than our clustering system, because it operates on single frames rather than groups of frames. This allowed us to test this system on hundreds of thousands of frames. Our dataset includes the feature length film "Run Lola Run", an hour of footage of a local park, and long sequences of legacy sports footage, including Michelle Kwan's entire 1998 Olympic performance.

Our automatic tracker consists of two stages. The system first runs a stylized pose detector in each frame of a sequence to find select frames from which to build discriminative appearance models (Section 4.3). It then tracks by detecting the learned appearance models in all frames (Section 4.4). We evaluate each component separately.

Automatic Initialization: We stress that all these tracks were obtained completely automatically with no manual intervention; the same program with identical parameters was used in each case. We did scale the video clips so that the figure was of a size expected

by the stylized detector.

4.5.2.1 Lateral-Walking Pose Detection

Evaluating our walking pose detector is a difficult task by itself. Labeling false positives is straightforward; if the detector finds a person in the background, that is incorrect. But labeling missed detections is difficult because our detector is not trying to detect all people, but only people in certain configurations.

In the case of "Run Lola Run", we can exploit *shots* (sequences where the camera is filming continuously) in our evaluation. We label each shot (as determined by a histogram-based shot detector) as containing a full-body figure or not. We define the score of a shot to be the best score from our walking detector on its set of frames. The implicit assumption is that if a shot contains a person, our walking pose detector will fire at some point. In Figure 4.16, we show precision-recall curves for the task of detecting shots with full-body figures using our walking detector. Note that we would expect to perform much better if we use a learned appearance model to track throughout a video, and not just in the shot it was found (particularly for "Run Lola Run" since Lola never changes clothes!). Even without exploiting that fact, we still do quite reasonably at high-precision/low-recall regions of the graph, and significantly better than chance.

For the park sequence, there are no natural shots, making recall measurements awkward to define. Since this video contains multiple people (many of whom look similar), we cluster the appearance models to obtain a set of different-looking people models, and then use them to search the entire video. In this case, we would like to select a detector threshold for our

Figure 4.16: Evaluating our stylized detector on "Run Lola Run". We score each shot with the best detection score from our stylized detector. On the **top** show the top 12 shots from the first 30000 frames of the movie. On the **bottom**, we show precision-recall curves for detecting shots with a full-body person. At low-recall, high-precision regions, our detector performs quite well. Many running shots of Lola show her running toward or away from the camera, for which our detector does not fire. If we use the model learned from one shot across the whole video, we would expect to do significantly better (since Lola never changes clothes!).

walking detector where most of the accepted detections are correct, and we still accept enough different-looking models to capture most people in the video. As such, we plot precision versus number of appearance model clusters spanned by the accepted detections

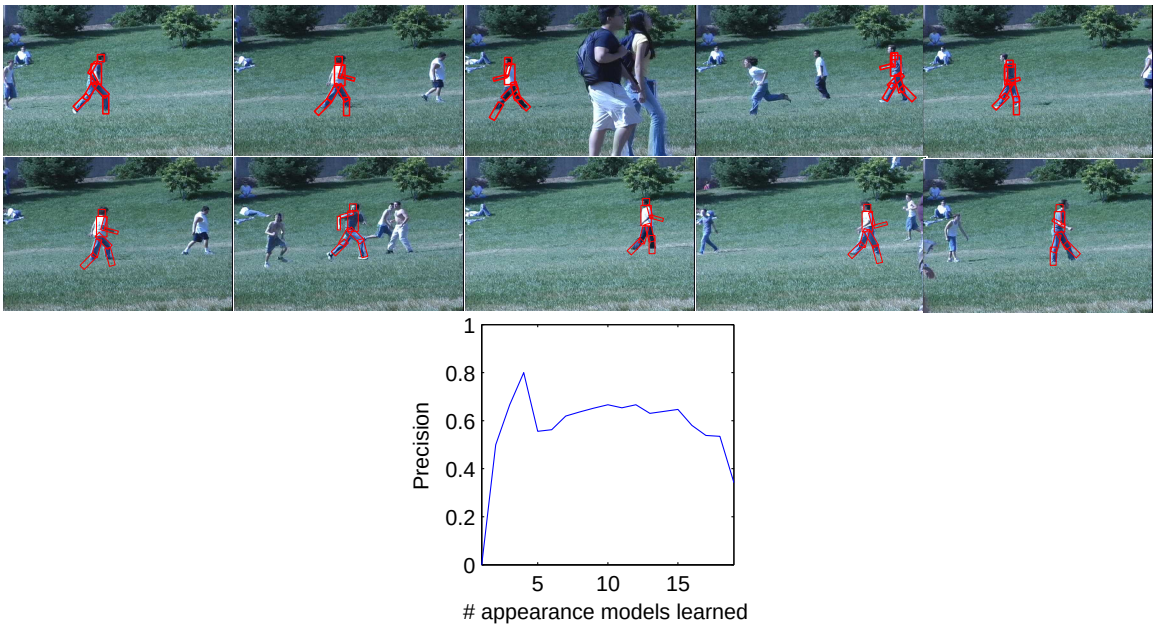


Figure 4.17: Evaluating our stylized detector on unscripted outdoor footage. On the **top**, we show the top 10 detections for the first 30000 frames of the video. Even though multiple people are frequently interacting (see Figure 4.20), our walking pose detector tends to fire on frames where the figures are well separated, since they have a better detection score. Note that we do not use any form of background subtraction. On the **bottom**, we show precision curves. Recall is awkward to define since we are not trying to detect people in every frame; rather we want to fire at least once on different looking people in the sequence. We obtain a set of models of different looking people by clustering the correct detections of our walking detector (which we validate by hand). For a given detector threshold, we can explicitly calculate precision (how many of the reported detections are correct) and the number of different models spanned by correct detections. As we lower the threshold, we span more models.

in Figure 4.17. We do quite reasonably; we can find most of the different looking people in the video while still maintaining about 50% precision. In other words, if our appearance model detection was perfect (Section 4.5.2.2) and we were willing to deal with 50% of the tracks being junk, we could track all the people in the video.

We look at the ability of our detector to find people performing unusual activities in Figure 4.18. Perhaps surprisingly, we are still able to find frames where our detector fires.

This means our algorithm is capable of tracking long and challenging sports footage, where people are moving fast and taking extreme poses. We show results on a baseball pitch from the 2002 World Series and Michelle Kwan's entire medal winning performance from the 1998 Winter Olympics.

4.5.2.2 Appearance model detection

In the second phase of our algorithm, we track by detecting the learned appearance models in each frame. Following the convention from Section 4.5.1, we evaluate performance by considering localization results for limbs (Table 4.2). The results for our commercial sequences are impressive considering the fast movement and the complexity of the background. Our success is due to the simple fact that people often dress to be visually distinctive. If baseball players wore green uniforms, it would be hard to spot one's teammate on a playing field. Likewise, filmmakers often want characters to stand out so they can be seen by an audience; a wonderful example is Lola's red hair. Once our person model learns to look for red hair (Figure 4.19) or a white uniform (Figure 4.18), it is essentially impossible for it to loose track because *by design* there is little white or red in the background.

In our park sequence (Figure 4.20), we learn multiple appearance models for multiple people. Here, we consider a localization to be correct if it fires on any person in the image because we do not look for consistency of detections from one frame to the next. Since many people in the sequence look like each other, we need additional constraints (such as motion) to pull out individual tracks. Our results here are also good, though not near the performance we achieve on our commercial sequences. We do quite well at detecting torsos,

Figure 4.18: Our automatic tracker on commercial sports footage with fast and extreme motions. On the **top**, we show results from a 300 frame sequence of a baseball pitch from the 2002 World Series. On the **bottom**, we show results from the *complete* medal-winning performance of Michelle Kwan from the 1998 Winter Olympics. We label frame numbers from the 7600-frame sequence. For each sequence, our system first runs a walking pose finder on each frame, and uses the single frame with the best score (shown in the **left insets**) to train the discriminative appearance models. In the baseball sequence, our system is able to track through frames with excessive motion blur and interlacing effects (the **center inset**). In the skating sequqnce, our system is able to track through extreme poses for thousands of frames. This means that our system can track long sequences with extreme poses, complex backgrounds, and fast movement *without manual intervention*.

with about 90% accuracy, while arms are still difficult because they are small and move fast.

This task is hard for many reasons; the video is washed out, there are significant shadow effects, and there are many small people interacting with each other (Figure 4.20).

% of frames correctly localized (stylized pose)

Sequence	Torso	Arm	Leg
Baseball	98.4	93.75	95.3
Skating	99.2	77.5	97.6
Lola	95.6	89.8	92.6
Park	79.1	29.2	58.3

Table 4.2: We evaluate our appearance model detection by calculating how often individual limbs are correctly localized. For "Run Lola Run", we average performance over three shots on which the stylized detector correctly fired (a separate appearance model was learned for each shot; see Figure 4.10,4.21,4.19). Our torso and leg detection tends to be quite good, with arms being harder to track because they are small. The outdoor "Park" sequence proves difficult because of crowd scenes and lighting changes. Our performance is impressive given the difficulty and length of sequences we test on; our commercial sequences contain extremely fast movement and non-stationary complex backgrounds, and are tracked *completely automatically*.

Figure 4.19: A sequence from "Run Lola Run" of Lola running around corner and bumping into a character while undergoing extreme scale changes. Note that the other character is wearing bulky clothing and so our person detector has no hope of finding him. Our initial walking detector is run at a single scale, but once we learn an appearance model (as shown in Figure 4.9), we track over multiple scales by searching an image pyramid at each frame.

4.6 Discussion

This chapter presents an approach to people-tracking that bootstraps a generic person model into an instance-specific one. The basic framework is to (1) run a generic model on

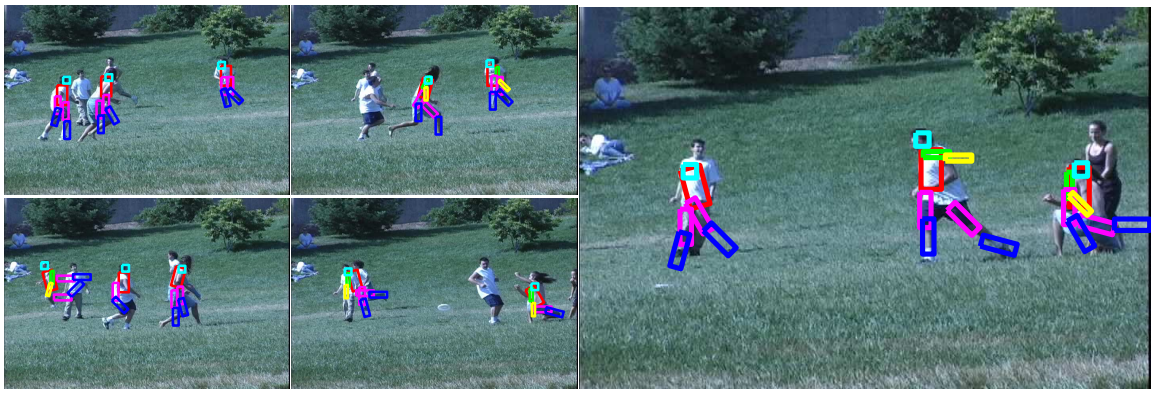
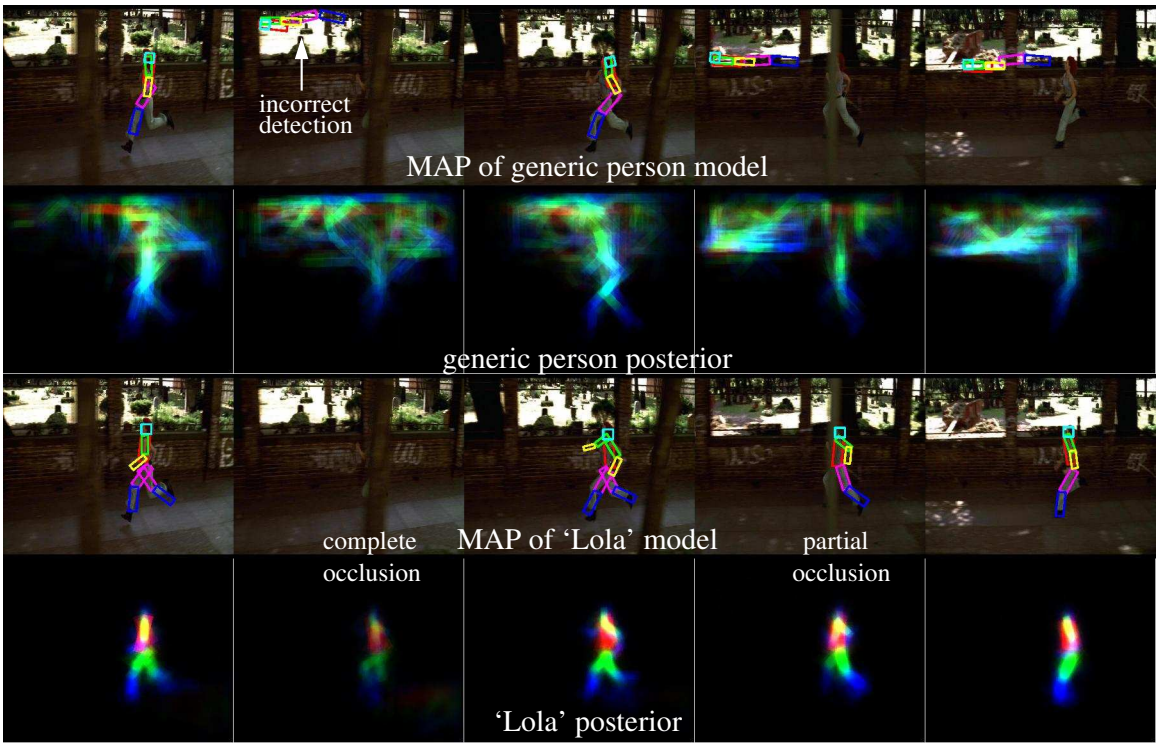


Figure 4.20: Automatic tracking of a sequence from the 30000 frame park sequence. This sequence is harder than our commercial footage because we have poor color resolution, many small figures are in shadow, and many are occluding each other while performing fast motions. We still obtain good detections and reasonable localization of arms and legs. Since many of the learned models look similar, we do not try to disambiguate instances from frame to frame. One might do that using motion constraints.

a video, (2) build a specific model from the detections, and then (3) use the specific model to track.

Do better models help? The fundamental premise of this chapter is that tracking is easier with an instance-specific person model, rather than a generic one. A natural question to ask is how well one could do simply with a generic model. We examine this in Figure 4.21. We construct two pictorial structures using *identical* code except for the part appearance model $\Pr(Im(P^i)|P^i, C^i)$. We use edge templates for the generic model, and we use a color classifier for the 'Lola' model (trained on a stylized detection). Looking at the posterior map, the 'Lola' model performs much better at *data association*; it readily ignores most of the background clutter. Background clutter confuses the generic model, causing spurious modes in its posterior. These modes also suggest why propagating a dynamic model is hard; it must maintain uncertainty across all of these modes. The MAP estimates of the 'Lola'



% of frames correctly localized

Model	Torso	Arm	Leg
Generic	31.4	13.0	22.2
'Lola'	98.1	94.3	100

Figure 4.21: Tracking people is easier with an instance-specific model as opposed to a generic model. In the **top** 2 rows, we show detections of a pictorial structure where parts are modeled with edge templates. We show both the MAP pose and visualize the entire posterior using the method of Figure 4.11. Note that the generic edge model is confused by the texture in the background, as evident by the bumpy posterior map. In the **bottom** 2 rows, we show results using a 'Lola' model where part appearances are learned from a stylized detection (Section 4.3). This model does a much better job of *data association*; it eliminates most of the background pixels. We can quantify this by looking at the percentage of frames where limbs are accurately localized (the **table**). Note that because we track by detection, our system can recover from partial and complete occlusions.

model also produce better tracks. We quantify this by looking at localization rates.

Fundamentally, we build models by looking for *coherence* in the detections. We develop an algorithm in Section 4.2 that operationalizes this notion by clustering candidate body

parts. A quite useful observation is that this initial detection can be done *opportunistically*; we describe an algorithm in Section 4.3 that looks for stylized poses that are reliably detected and easy to build appearance from. One could also look for stylized motions over short frames; such a detector might perform better because it pools information from across frames. Another practical observation is that *discriminative* appearance models learned from a few frames can discriminate an object in other frames. Discriminative features for tracking are not new [21], but by learning them from select frames in which we trust our detections, they become quite powerful.

Comparison of model-building algorithms: We find the two model-building algorithms complementary. The stylized-pose system appears more robust, since a single set of parameters worked for all the sequences shown. If we can observe people for a long time, or if we expect them to behave predictably, detecting stylized poses is likely the better approach. These scenarios might be realistic for public areas monitored by security cameras or athletic performances. However, if we observe a person moving for a short time in a contrived manner, clustering a pool of candidate parts may work better than a stylized detector. A simple example of this is our jumping jack sequence (Figure 4.12); here, the clustering algorithm easily learns the body appearance, but stylized pose algorithm fails because the figure never walks laterally. This suggests another improvement for our stylized system; increase the set of stylized poses to cover different aspects.

Chapter 5

Activity Recognition

Let us assume that we have a pictorial structure model for each actor in a video. The model can be obtained off-line or online using the methods described in Chapter 4. In this chapter, we show how to use such a model to obtain a description of the activities of each actor automatically. Our basic approach is to match 2D poses (obtained by matching the pictorial structure to each frame) to 3D poses from a motion capture library. Off-line, we label the 3D poses with activity labels, and so we retrieve the labels during the matching.

5.1 Previous Work

There has been a substantial amount of work on recognizing activities from video (for review papers, see [3, 12, 13, 40, 49]).

There seems to be a general consensus that different taxonomies are needed for interpreting events at different time scales. Bobick [13] refers to an atomic event as a *movement*,

a sequence of movements as an *activity*, and finally terms large-scale events as *actions*.

Methods for recognizing small-scale movements typically match some video signature to a **template**. If the background is stationary or uncluttered, than one can match low-level spatio-temporal descriptors without explicitly tracking arms and legs [14, 25, 30, 87, 125]. Alternatively, one could explicitly extract body pose, and directly match based on pose estimates [7, 9, 112]. These approaches are more similar to our work.

Recognizing large scale movements probably requires more sophisticated techniques; one approach is that of **temporal logics**. Here actions and events are represented with a set of logic primitives; valid temporal relations are represented by interval bounds (to pick up a cup we must first grab it) [4, 85, 86]. Given a video with low-level event detections, one obtains high level descriptions by forms of constraint propagation [103, 104]. Logic formalisms can be difficult to both author and perform inference with because they require clean input; it is not clear if they will work given missing or noisy video summaries.

5.1.1 Methods based on Hidden Markov Models

By far the most popular approach is to use a HMM, where the hidden state is an activity to be inferred, and observations are image measurements. Models used have tended to be small (for example, one sees three to eight state models in [15, 17, 32]). Yamato *et al.* describe recognizing tennis strokes with HMM's [122]. Wilson and Bobick describe the use of HMM's for recognizing gestures such as pushes [121]. Yang *et al* use HMM's to recognize handwriting gestures [123].

There has been a great deal of interest in models obtained by modifying a basic activity-

state HMM. The intention is to improve the expressive power of the model without complicating the processes of learning or inference. Variations include a coupled HMM (CHMM) [15, 17], a layered HMM (LHMM) [76, 80, 81], a parametric HMM (PHMM) [120], an entropic HMM (EHMM) [16], and variable length Markov models (VLMM) [38, 39]. All these HMM variations could be seen as ways of simplifying the process of *model authoring*: i.e., learning the state transition matrix. Fitting a large state space model from data is hard because one needs lots of data.

5.2 Our approach

We focus on recognizing small-scale activities (*movements*, in Bobick's terms). An immediate concern is what activities to recognize? For limited domains, such as tennis footage, there maybe natural categories – a forehand stroke, a backhand, etc. However, when placing a camera in a public park, it is not so obvious (see Figure 5.1). In our opinion, in order to capture everyday movements, one must use a vocabulary that is large and expressive. To achieve this, we allow activity labels to be **composed**; one can both `walk` and `wave` simultaneously. With such a representation, we must take care to not allow invalid combinations (e.g., one cannot simultaneously `run walk`).

A large vocabulary complicates matters considerably because learning and inference are now difficult. In particular, estimating the parameters of a large state-space HMM is hard because one needs an exorbitant amount of training data. We *decouple* both learning and inference from the choice of vocabulary by taking an **analysis as synthesis** approach; we

Figure 5.1: What is the correct annotation vocabulary? Most likely, an annotation system will not contain a `'Trying to catch a frisbee between the legs, but dropping it'` term. This makes it difficult to construct a canonical vocabulary for everyday motion. We construct a class structure that allows for different labels to be composed; such a system might label the above motion as `crouching`, `reaching`, and `kicking`. This creates a large effective vocabulary size which makes direct analysis difficult. We build a system that decouples analysis from the choice of vocabulary by matching poses rather than annotations.

recognize activities by synthesizing a motion that looks like a video. We synthesize motion by re-arranging pre-recorded clips from a motion capture database. By pre-labeling clips with activity labels, we synthesize an activity labeling "for free".

Essentially, we replace an activity state-space HMM with a pose state-space HMM. Learning a pose model is easier because poses live in a metric space; this means we can construct transition matrices by looking for poses that are nearby in this space. Such matrices are typically sparse and known as *motion graphs* in the graphics literature [5, 59, 61]. From our HMM perspective, synthesis is equivalent to inferring a sequence of hidden pose states. In practice, one performs synthesis by using graph search algorithms on an underlying motion graph.

Figure 5.2 shows an overview of our approach to activity recognition. We use 3 core components: annotation, detection, and motion synthesis. Initially, a user labels a collection

of 3D motion capture frames with annotations; this can be done with minimal supervision (Section 5.3). Given a new video sequence to annotate, we use a pictorial structure to detect 2D poses of each figure in a sequence [31]. We then synthesize 3D motion sequences which look like the detections by matching them to our annotated motion capture library (Section 5.4). We finally accept the annotations associated with the synthesized 3D motion sequence.

Our approach of linking video tracks with a motion capture database dates back to at least the work of Sidenbladh *et al* [100]. Similar approaches of synthesizing motions given constraints from video are taken in [61, 94]. Those works use a motion graph to enforce a human smoothness prior; we use a motion graph as a mechanism for explicit *analysis* – obtaining an activity description [88].

5.3 Obtaining Annotated Data

We have annotated a body of motion data using the system described in [6]. We repeat the process here for convenience.

There is no reason to believe that a canonical annotation vocabulary is available for everyday motion, meaning that the system of annotation should be flexible. In practice, this means that it should be relatively simple for a user to revise the annotations attached to the data set. Annotations should allow for composition as one can `wave` while `walk`ing, for example. We achieve this by representing each separate term in the vocabulary as a bit in a bit string. Our annotation system attaches a bit string to each frame of motion. Each

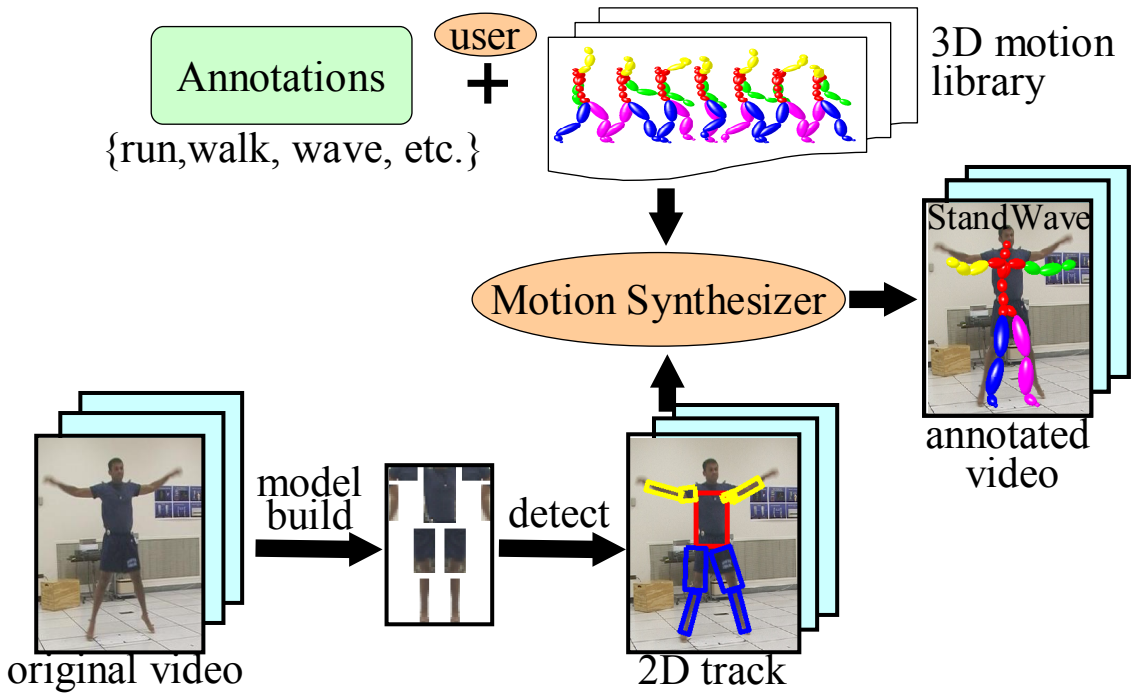


Figure 5.2: Our annotation system consists of 3 main components; annotation, detection, and motion synthesis. A **user** initially labels a collection of 3D motion capture frames with annotations. Given a new video sequence to annotate, we track by detecting a pictorial structure model in each frame. We then **synthesize** 3D motion sequences which look like the 2D tracks by lifting tracks to 3D and matching them to our annotated motion capture library. We accept the annotations associated with the synthesized 3D motion sequence as annotations for the underlying video sequence.

bit in the string represents annotation with a particular element of the vocabulary, meaning that elements of the vocabulary can be composed arbitrarily.

That said, we still need an initial vocabulary that can be composed/revised as needed. We use a set that is convenient for **synthesizing** motions from a given database; this suggests the vocabulary will be tied to that specific database. We use 7 minutes of football motions, collected for a commercial video game. An independent user decided a set of 13 labels would be useful for synthesizing motions from this collection; `run, walk, wave, jump,`

`turn left`, `turn right`, `catch`, `reach`, `carry`, `backwards`, `crouch`, `stand`, and `pick up`. These labels are allowed to occur in any combination: `turn left` while `walk`ing, or `catch` while `jump`ing and `run`ning. This produces an enormous vocabulary of $2^{13} = 8192$ different annotations. Learning a HMM with such a large state space may be difficult; we avoid such difficulties by dealing with poses rather than labels. In practice, many label combinations will never be used; we can't conceive of a motion that should be annotated with both `stand` and `run`. Examining our database, we see that only 46 combinations are ever observed. We must take care to ensure that the final analysis/synthesis algorithm respects these valid combinations (see Section 5.4.3).

Actual annotation is simplified by using an online learning approach where a user bootstraps a classifier. Initially, a user hand-labels a random subset of frames from the database. One SVM classifier is learned for each element of our vocabulary *independently*; this means one learns a `run/not-run` classifier, a `stand/not-stand` classifier, etc. The classifiers use Gaussian kernels, and use the joint positions for one second of motion centered at the frame being classified as a feature vector. Since the motion is sampled in time, each joint has a discrete 3D trajectory in space for the second of motion centered at the frame. We used a public domain SVM library (`libsvm [19]`). The out of margin cost for the SVM is kept high to force a good fit.

After the initial training, we use the classifiers to label the remaining frames in the database. The user is incrementally presented with newly classified frames, and makes corrections as necessary. The user verified data is then added to the SVM training set and

the classifier's decision boundary is re-optimized. It is our experience that after annotating 3-4 example variants of an annotation, the user rarely needs to correct the auto-annotation results. It is remarkable that such a procedure tends to respect constraints implicit in the user-labelings; the classifiers tend not to label a new motion as both a `run` and a `stand`. This means it is possible to annotate large databases of motions quite rapidly.

5.4 3D Motion Synthesis

Given a pictorial structure for each actor, we match it to each frame to obtain 2D pose estimates. We assume that our sequences are filmed with stationary cameras (or a moving camera with known motion, so that it can be factored out). We expect our 2D estimates to be noisy, and to suffer from left/right ambiguities.

We can think of the pictorial structure as returning a sequence of 2D stick figures. Intuitively, it might seem that identifying the rough 3D pose is possible by looking at a single 2D pose; we define a matching criteria that does this in Section 5.4.1. However, a 2D stick figure is still ambiguous in many ways; it is difficult to recover the orientation of the torso with respect to the camera, and it is difficult to label the left/right limbs. We resolve these ambiguities in Section 5.4.2 by propagating information throughout a video.

5.4.1 Matching by minimizing reprojection error

In this section, we describe an approach for matching 3D poses to 2D poses, assuming a scaled orthographic camera model. We represent each 2D pose as a point cloud of joint

positions. We define a joint position at the upper & lower torso, shoulders, elbows, wrists, knees, and ankles (for a total of 12 points). We can similarly represent each pose from our motion capture library as a cloud of 12 corresponding 3D points. Note that this is a subset of the 30 joints in our original motion capture skeleton (see Figure 5.2). Hence our matching process also recovers *extra* degrees of freedoms that are difficult to directly estimate from video. Computing the matching 3D pose reduces to the well-studied problem of matching a 3D point cloud to a 2D point cloud, where correspondences are known [46, 47].

To incorporate local dynamics (such as velocities and accelerations), we represent all poses by a point cloud of joint positions collected from 15 frame ($\frac{1}{2}$ second) windows. We write the 2D point cloud extracted from the t^{th} frame as m_t, a $2X(12 * 15)$ matrix. We compute m_t at every frame, and so the windows overlap. Similarly, we represent our motion capture library as a collection of 3D point clouds M_i (where i varies over the 11000 poses in our database). For convenience, we subtract out the centroids and subsample the clouds to be equal to the video frame rate.

Alignment: An important issue when comparing point clouds is that of alignment [46, 47]. We find the transformation that best aligns the 3D pose M_i with the 2D pose m_t, in terms of minimizing the reprojection error. We search over scaled-Euclidean transformations; a rotation, translation, and an isotropic scale. We explicitly search over rotations, rendering M_i from different orthographic cameras. We assume that the camera is at 0° elevation, and explicitly search over 20 azimuth (ϕ) values. Since the camera is fixed over a sequence, we can interpret ϕ as the orientation of the root body (i.e. the center torso) of

M_i with respect to the camera. Given a pose M_i and orthographic camera matrix R_ϕ, we compute a 2D rendered point cloud $R_\phi M_i$. Aligning two 2D point clouds is straightforward [46]; we translate and scale $R_\phi M_i$ so that its centroid and variance match that of m_t.

Reprojection error: We write the squared reprojection error after alignment as $||R_\phi M_i - m_t||^2$. We cannot directly compute this error because of left/right ambiguities. The pictorial structure model cannot distinguish between left and right arms because they look similar. This means that correspondence is not fully known. Interestingly, the alignment procedure described thus far still applies; the centroid and variance of a point cloud do not change when points are permuted. We compute a final reprojection error by finding the left/right assignment that minimizes the reprojection error:

$$||R_\phi M_i - m_t||^2_{lr} = \sum_{j=1}^{15} \min_k ||R_\phi M_i(j) - m_t(j) F_k||^2, \tag{5.1}$$

where we write $M_i(j)$ for the joint positions from the j^{th} frame in the 15-frame window. We write F_k for a $12X12$ binary matrix that permutes the left/right leg/arm joints of $m_t(j)$; hence $k \in \{1, 2, 3, 4\}$.

Missed Detections: In most frames, our pictorial structure will not recover all the limbs. We would like to omit any missing joint positions from the 2D and 3D clouds when performing alignment and computing the reprojection error. However, missing limbs complicate alignment; if we detect only one leg in a frame of m_t, we do not know whether to align it to a left or right reference leg in M_i (if we detect two legs, correspondence doesn't matter as explained above). We explicitly search over the 4 left/right labelings of

the center frame for alignment. To avoid searching over labelings for the remaining frames in our window, we only use unambiguous points for computing alignment (points from the center frame, points on the torso, and points from limbs for those frames when both limbs are detected). After alignment, we use all detected joint positions to compute the reprojection error, as in Equation 5.1.

Alternatives: One could build an aspect model exploiting missed detections. If the left side of a person is not detected, this suggests something about his/her orientation with respect to the camera. We can formalize this notion by computing the visibility of joint positions of the rendered 3D point clouds $R_\phi M_i$. When matching to a specific m_t with a given set of detected (i.e., visible) limbs, we add a penalty for mis-matched visibility flags (because they suggest a false positive or missed limb detection). In our experience, this aspect model did not significantly improve results.

Rather than computing strict $\mathcal{L}_2$ reprojection error, one might want to compute a weighted error where joint positions from the center frame are weighted more heavily. All our steps (our alignment procedure and reprojection error computation) still follow; in our experience this did not significantly change the final results.

5.4.2 Synthesis by Dynamic Programming

Given the matching procedure from Section 5.4.1, one might find the pose M_i that best matches m_t for each frame independently. For some poses, such as lateral views of the stance phase of a walk, it is hard to estimate both the torso orientation and left/right assignment from the 2D stick figure m_t. If someone supplied the correct camera azimuth

(or equivalently the torso orientation) and left/right assignment, the recovered pose M_i would match m_t rather well. We propagate information temporally to achieve this effect; we find a sequence of poses M_i that match m_t *and* that change smoothly from frame to frame.

For each frame t, let us compute the best match M_i given a particular camera orientation $\{1, 2, \ldots, 20\}$ and given a particular left/right assignment of the center frame. This yields a pool of $20 \times 4 = 80$ 3D point clouds rotated, translated, and scaled to align with m_t. Let us write these *aligned* point clouds as $M_t(l_t)$ and their reprojection error as $\epsilon_t(l_t)$ where $l_t \in \{1, 2, \ldots, 80\}$.

We now want to find a sequence of 3D point clouds that all match the image data well and that match future and past 3D estimates well. We can efficiently compute this by dynamic programming. Let us define

$$f(l_{1:T}) = \sum_t \epsilon_t(l_t) + w||M_t(l_t) - M_{t-1}(l_{t-1})||_o^2, \tag{5.2}$$

where $||.||_o^2$ is the squared error of the points that overlap temporally (the first 14 frame window in M_t and the last 14 frame window in M_{t-1}). The user-defined weight w controls how closely the synthesized motion should follow the image data (the first term) versus how continuous it should be (the second term). We use $w = .1$.

Estimating Depth: Aligning our point clouds $M_t(l_t)$ in the out-of-image plane direction is difficult; this is a well-known difficulty of estimating depth from a single camera. In theory, one could use the recovered scale (a person becomes bigger when he/she moves

closer to the camera). In many practical sequences, people's changes in depth are small relative to the distance to the camera, resulting in little scale change. As such, we do not try to explicitly recover the dZ movement, but rather make the continuity cost $||.||_o^2$ depth-invariant. We do this by z-translating each cloud $M_t(l_t)$ so that its centroid lies on $z = 0$ [46].

We compute the sequence of $\hat{l}_t$ that minimizes Equation 5.2 by dynamic programming. Doing so recovers a sequence of 3D clouds $M_t(\hat{l}_t)$ that all match the image data, and that temporally match each other well.

5.4.3 Smoothing

For each frame t, the recovered point cloud $M_t(\hat{l}_t)$ spans a 15 frame window. This means, for each frame t, we have 15 possible poses, obtained from neighboring frames with overlapping windows. Since our 3D point clouds $M_t(\hat{l}_t)$ are subsampled representations of larger point clouds (constructed from skeletons with additional joints), we align the full point clouds to the images using the recovered scaled-Euclidean transformations. To compute a final 3D pose, we average the joint positions from the 15 skeleton poses that overlap frame t.

Recall that each 3D pose was labeled off-line with a binary vector of 13 flags representing an annotation. To compute an activity vector for the t^{th} frame, one might be tempted to average the binary vectors of the overlapping poses. However, this might produce a final annotation where both `run` and `stand` are 'on'. To respect the constraints implicit in the off-line labeling, we use a weighted voting scheme. Each overlapping pose votes for 1 of the 2^{13} possible annotation vectors, and we finally choose the annotation vector with the most

votes. In practice, we precompute all valid annotation vectors from the database (of which there are 46), and only record votes for one of them.

5.5 Experimental Results

We use a motion database of 118 motions of football players. Each frame was annotated using the procedure and vocabulary of section 5.3 by a user who had not seen the videos to be annotated. We test our system on three sequences; a video of a person walking back and forth, a video a person performing jumping jacks (or star jumps), and finally a video of three people passing a ball back and forth. For each video, our system automatically built a pictorial structure for each figure using the clustering method of Section 4.2. Our system then detects the pictorial structure(s) in each frame following the procedure of Section 4.4. Next, our system synthesizes a 3D motion by matching annotated motion clips to the 2D detections (Section 5.4).

Evaluation of general-purpose activity recognition systems is difficult precisely because there is no canonical vocabulary. Given a test video, we manually annotate it with labels appropriate for that video. For example, given a video of a figure performing jumping jacks, we use the labels `closed` and `extended` to describe phases of the jump. These labels need not correspond with those used to describe a collection of football motions. This makes applying standard evaluation criteria such as ROC curves or confusion matrices awkward, since there is no clear correct detection. Furthermore, there is no meaningful standard with which to compare.

Scoring by plotting signals: Qualitatively, we lay out a visual comparison between the human and automatic annotations signals. We show an example in Figure 5.3, which displays annotation results for a 91-frame jumping jack sequence. The top 4 lower case annotations are hand-labeled over the entire 91 frame sequence. Generally, automatic annotation is successful: the figure is detected correctly, oriented correctly (this is recovered from the torso orientation estimates ϕ), and the description of the figure's activities is largely correct.

Scoring by conditional entropy: We quantify the degree to which the user and automatic signals agree (without explicit correspondence) by computing the *mutual information*, or reduction in entropy. We interpret the user annotation as a (bit) vector valued random variable U observed at every frame t. It takes on one of the finite set of values observed for that sequence. We likewise define A to be a random variable capturing the automatic annotation signal. Mutual information captures the reduction in uncertainty of U given A, or $M(U, A) = H(U) - H(U|A)$, where

$$\begin{aligned} H(U) &= \sum_i \Pr(U = i) \log \Pr(U = i) \\ H(U|A) &= \sum_{i,j} \Pr(U = i, A = j) \log \Pr(U = i|A = j), \end{aligned}$$

where the probabilities are computed by counting co-occurrences throughout a given video sequence. Given a test video with user annotation U, the entropy $H(U)$ is equivalent to the number of bits required to encode U. Given knowledge of some other signal A, the entropy (or uncertainty) in U can only decrease. We evaluate different automatic

Entropy $H(U)$ and Mutual Information $(H(U) - H(U|A))$

Sequence	$H(U)$	$H(U\|A)$	$H(U) - H(U\|A)$
Jumping Jacks	.5862	.4800	.1062
Walk	2.7358	1.6336	1.1022
Weave (Fig. 1)	2.2450	.9618	1.2832
Weave (Fig. 2)	2.1940	1.3188	0.8752
Weave (Fig. 3)	2.4011	1.1380	1.2631

Table 5.1: We score our performance using the reduction in entropy of (manually-labeled) user annotations U given the output of our automatic system A. For the 'Jumping Jack' sequence, we did not allow `null` annotations. For the 'Weave' sequence, we evaluate results for the three tracked figures. In almost all cases, the automatic system reduces the entropy by a factor of 2. The user annotations for the Jumping Jack sequence prove too simple; there is less than a bit of uncertainty in them (since they are nearly constant), and so our system provides little improvement.

annotations A for a given U by computing which reduces the entropy $H(U|A)$ the most. We tabulate results in Table 5.1.

Is temporal consistency required? Figure 5.4 compares two versions of our system on a 288 frame sequence of a figure walking back and forth. The annotations on the left were obtained by matching poses M_i independently for each frame t. Note that the automatic `LFace` and `RFace` signals flip back and forth – it is difficult to estimate the orientation of the figure when he is in the stance phase of a lateral walk. By performing explicit motion synthesis (ensuring the poses M_i are smooth over time, as in Section 5.4.2), orientation is estimated much more reliably. We quantify this by computing the conditional entropy of each set of automatic annotation signals. The smoothed annotation signal correlates better with the ground-truth user signal.

Annotating multiple people interacting: In Figure 5.6, we show annotations for three figures from one sequence passing a ball back and forth. Each actor is correctly

detected, and the system produces largely correct descriptions of the actor's orientation and actions. The inference procedure interprets a run as a combination of `run` and `walk`. Quite often, the `walk` annotation will fire as the figure slows down to turn from `face right` to `face left` or vice versa. When the figures use their arms to catch or throw, we see increased activity for the similar annotations of `catch`, `wave`, and `reach`.

Annotating novel motions: When a novel motion is encountered, we want the system to either respond by (1) recognizing that it cannot annotate this sequence, or (2) annotating it with the best possible label. We can implement (2) by throwing out those poses M_i that are annotated with a 'null' bit vector (all flags are off) when matching (Section 5.4.1). Interestingly, almost $\frac{1}{4}$ of the 3D poses in our library have a null label. This implies our original annotation vocabulary is still too small for our motion library. In Figure 5.3, we use our football library to annotate a jumping jack sequence. Our library does not contain any jumping jack motions, and lacks a `jumping jack` annotation label. System (1) responds with a `stand` label when the figure is near a `closed` stance. By forcing the annotations to come from non-null poses, we see an additional `walkwave` label toward the `extend` phase, with the occasional `turn`. Quantitatively, (2) results in better annotations as seen by a lower conditional entropy score. Note the entropy estimates for this sequence are quite small; this is because the sequence is short and the user signal U is almost constant.

5.6 Discussion

In this chapter, we describe a method for automatically annotating a video of everyday movements. We do this by taking an "analysis by synthesis" approach; we first estimate the configurations of a figure over time, and then we *re-create* that estimate by matching to an existing set of examples. If the examples are labeled, we generate labels for free.

This approach requires a method that can reasonably estimate the configurations of the body over time; we demonstrate that a pictorial structure can generate such configuration reports. By using real data as opposed to hand-labeled body estimates [27, 64, 65], we find that our synthesis engine must compensate for ambiguities such as torso orientation, left/right labelings, and missed detections.

We introduce mutual information as a scoring criteria. Evaluation is hard because there is no canonical vocabulary for describing everyday behavior (see Figure 5.1). Our measure provides a reasonable scheme for ranking different systems given labeled test footage.

Our analysis-by-synthesis approach appears to label small-scale activities reasonable well. We hope the analogy can apply to large-scale actions as well. It suggests one approach for authoring complex models; we find rules that synthesize complex actions realistically, and apply them to analyze such actions from video.

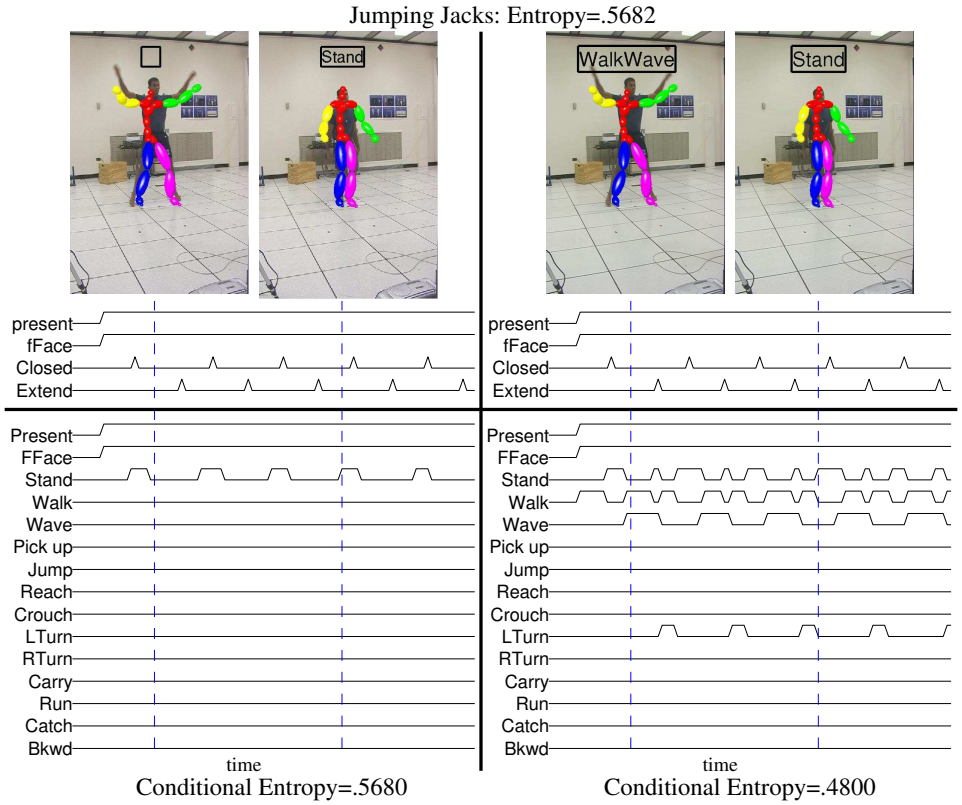


Figure 5.3: Unfamiliar configurations can either be annotated with 'null' or with the closest match. We show annotation results for a sequence of jumping jacks (sometimes known as star jumps) from two such annotation systems. In the top row, we show the same two frames run through each system. The recovered 3D pose from Section 5.4.3 has been reprojected back to the image. In the bottom, we show signals representing annotation bits over time. The manual annotator records whether or not the figure is `present`, `front faceing`, in a `closed` stance, and/or in an `extended` stance. The automatic annotation consists of a total of 16 bits; `present`, `front faceing`, plus the 13 bits from the annotation vocabulary of Section 5.3. In first dotted line, corresponding to the image above it, the manual annotator asserts the figure is `present`, frontally `faceing`, and about to reach the `extended` stance. The automatic annotator asserts the figure is `present`, frontally `faceing`, and is **not** `standing`, **not** `jumping`, etc. The annotations for both systems are reasonable *given there are no corresponding categories available* (this is like describing a movement that is totally unfamiliar). On the **left**, we freely allow 'null' annotations (where no annotation bit is set). On the **right**, we discourage 'null' annotations as described in Section 5.5. Configurations near the `extend` stance are now labeled as `walkwave`, a reasonable approximation. We quantitatively show this approach results in better annotations, by computing the reduction in entropy of the user signal given the automatic reports (as described in Section 5.5.).

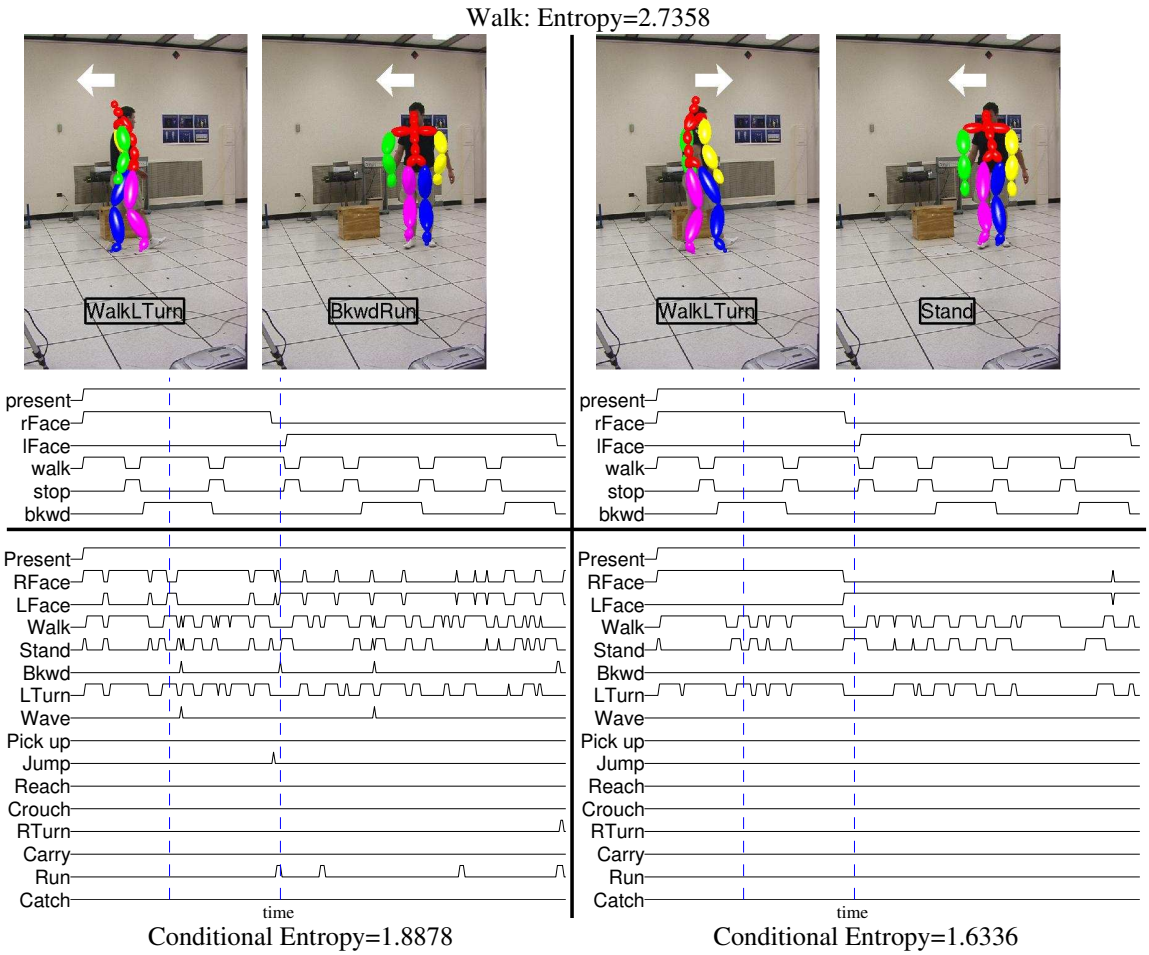


Figure 5.4: We show annotation results for a walking sequence from two versions of our system using the convention of Figure 5.3. Null matches are allowed. On the **left**, we infer the 3D pose M_i (and associated annotation) independently for each frame. On the **right**, we synthesize a smooth set of poses (and associated annotations) by dynamic programming (Section 5.4.2). Each image is labeled with an arrow pointing in the direction the inferred figure is facing, not moving. By enforcing smoothness, we are able to fix spurious `run`'s and incorrect torso orientations present on the **left** (i.e., the first image frame and the automatic `left faceing` and `right faceing` annotation bits). The system on the **right** correlates better with the user annotations, as shown by a lower conditional entropy score.

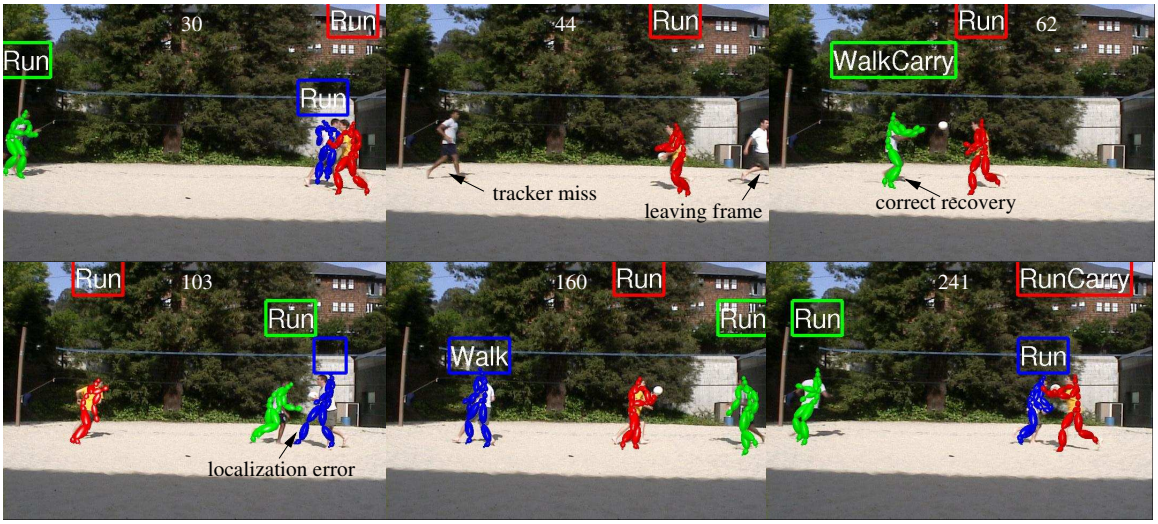


Figure 5.5: Frames sampled from a 21 second sequence of three actors playing with a ball. The numbers on each frame give the order in which the frame appears in the sequence; the spacing is roughly even. The white annotations are automatically generated by our system (we manually add the black words for clarity). Overlaid on each frame is the best configuration chosen for the body of each of the three actors detected — both number and appearance are obtained automatically — using camera consistency as in Section 5.4.2. Individuals are associated with a color and a fixed-height annotation label to show the tracker has consistently identified them. We see two tracks interrupted, one because of a missed detection and the other because the figure leaves the view. Both tracks are recovered, but we see an incorrect pose estimation because of a missed leg detection.

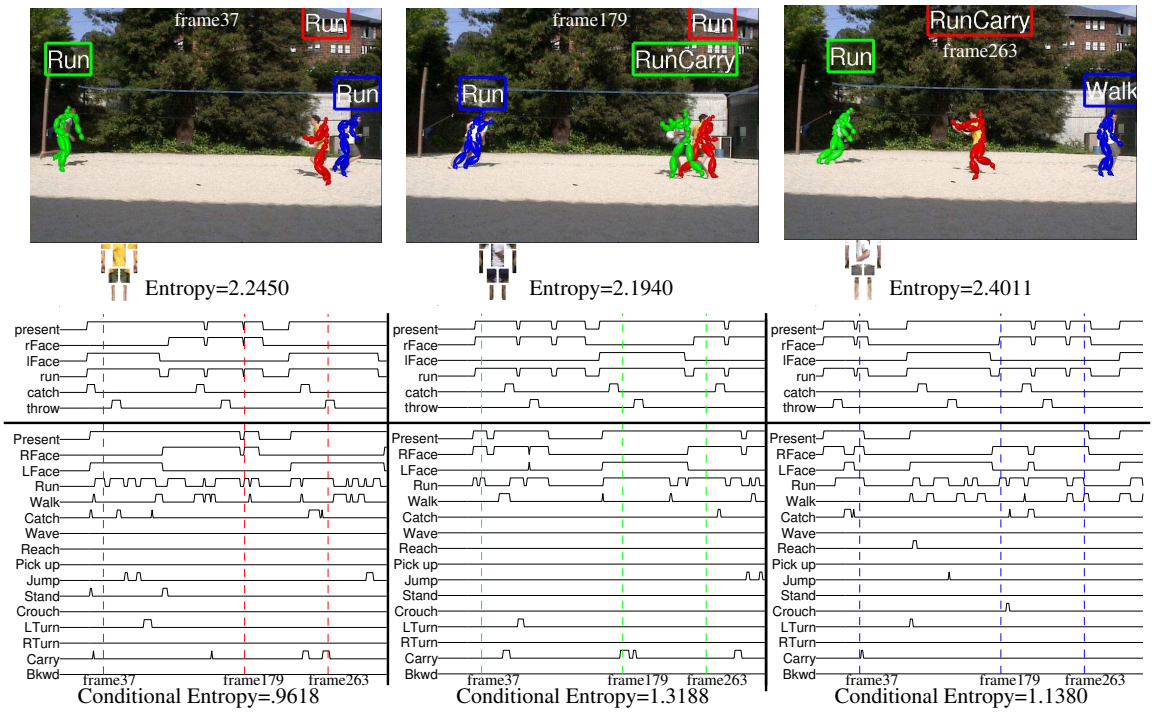


Figure 5.6: Annotations of 3 figures from a video sequence of the three passing a ball back and forth using the conventions of figure 5.3. Null matches are allowed. The dashed vertical lines indicate annotations corresponding to the frames shown. The automatic annotations are largely accurate: the figures are correctly identified, and the direction in which the figures are facing are largely correct. Most of the time, people are `running`, but slow down to `walk` when turning or passing the ball. Throws appear to be mislabeled as `catches`. Generally, when the figure has the ball (after `catching` and before `throwing`, as denoted in the manual annotations), he is annotated as `carrying`, though there are some missed detections. There are no spurious `crouches`, `waves`, etc.

Bibliography

[1] A. Agarwal and B. Triggs. 3d human pose from silhouettes by relevance vector regression. In *CVPR*, 2004.

[2] A. Agarwal and B. Triggs. Tracking articulated motion with piecewise learned dynamic models. In *ECCV*, 2004.

[3] J. K. Aggarwal and Q. Cai. Human motion analysis: A review. *Computer Vision and Image Understanding: CVIU*, 73(3):428–440, 1999.

[4] James F. Allen. Towards a general theory of action and time. *Artifical Intelligence*, 23(2):123–154, 1984.

[5] O. Arikan and D. Forsyth. Interactive motion generation from examples. In *Proc. ACM SIGGRAPH*, 2002.

[6] O. Arikan, D.A. Forsyth, and J. O'Brien. Motion synthesis from annotations. In *Proc. ACM SIGGRAPH*, 2003.

[7] J. Ben-Arie, Z. Wang, P. Pandit, and S. Rajaram. Human activity recognition using multidimensional indexing. *PAMI*, 24(8):1091–1104, August 2002.

[8] Alexander Berg and Jitendra Malik. Geometric blur for template matching. In *CVPR*, 2001.

[9] M.J. Black and Y. Yacoob. Parametrised modelling and tracking of human activities. *Computer Vision and Image Understanding*, 73(2):232–247, 1999.

[10] S. Blackman and R. Popoli. *Design and Analysis of Modern Tracking Systems*. Artech House, 1999.

[11] A. Blake and M. Isard. Condensation - conditional density propagation for visual tracking. *Int. J. Computer Vision*, 29(1):5–28, 1998.

[12] A. F. Bobick and J.W. Davis. The recognition of human movement using temporal templates. *IEEE T. Pattern Analysis and Machine Intelligence*, 23(3):257–267, 2001.

[13] A.F. Bobick. Movement, activity, and action: The role of knowledge in the perception of motion. *Philosophical Transactions of Royal Society of London*, B-352:1257–1265, 1997.

[14] A.F. Bobick and J.W. Davis. Real-time recognition of activity using temporal templates. In *Workshop on Applications of Computer Vision*, 1996.

[15] M. Brand. Coupled hidden markov models for complex action recognition. Media lab vision and modelling tr-407, MIT, 1997.

[16] M. Brand and V. Kettnaker. Discovery and segmentation of activities in video. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 22(8):844–851, August 2000.

[17] M. Brand, N. Oliver, and A.P. Pentland. Coupled hidden markov models for complex action recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 994–999. IEEE Press, 1997.

[18] C. Bregler and J. Malik. Tracking people with twists and exponential maps. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 8–15, 1998.

[19] C. C. Chang and C. J. Lin. Libsvm: Introduction and benchmarks. Technical report, Department of Computer Science and Information Engineering, National Taiwan University, 2000.

[20] O. Chapelle, P. Haffner, and V. Vapnik. Svm's for histogram-based image classification. In *IEEE Trans. on Neural Networks*, 1999.

[21] R. Collins and Y. Liu. On-line selection of discriminitive tracking features. In *ICCV*, 2003.

[22] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE PAMI*, 24(5):603–619, 2002.

[23] J. Coughlan and S.J. Ferreira. Finding deformable shapes using loopy belief propogation. In *Proc ECCV*, 2002.

[24] K.J. Dana, S.K. Nayar, B. van Ginneken, and J.J. Koenderink. Reflectance and texture of real-world surfaces. In *CVPR*, pages 151–157, 1997.

[25] J.W. Davis and A.F. Bobick. The representation and recognition of action using temporal templates. In *CVPR*, 1997.

[26] Jonathan Deutscher, Andrew Blake, and Ian Reid. Articulated body motion capture by annealed particle filtering. In *CVPR*, 2000.

[27] D. DiFranco, T.J. Cham, and J.M. Rehg. Reconstruction of 3-d figure motion from 2-d correspondences. In *CVPR*, December 2001.

[28] G. Dorko and C. Schmid. Object class recognition using discriminative local features. IEEE PAMI, under preparation.

[29] P. Duygulu, K. Barnard, N. de Freitas, and D.A. Forsyth. Object recognition as machine translation. In *Proc. European Conference on Computer Vision*, pages IV: 97–112, 2002.

[30] A. Efros, A. Berg, G. Mori, and J. Malik. Recognizing action at a distance. In *ICCV*, 2003.

[31] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Pictorial structures for object recognition. *Int. J. Computer Vision*, 61(1), January 2005.

[32] Xiaolin Feng and P. Perona. Human action recognition by sequence of movelet codewords. In *First International Symposium on 3D Data Processing Visualization and Transmission*, pages 717–721, 2002.

[33] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *CVPR*, 2003.

[34] Michael Gleicher Nicola Ferrier. Evaluating video-based motion capture. In *Proceedings of Computer Animation*, June 2002.

[35] M. A. Fischler and R. A. Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on Computer*, 1(22):67–92, January 1973.

[36] D.A. Forsyth, M.M. Fleck, and C. Bregler. Finding naked people. In *ECCV*, 1996.

[37] W. T. Freeman and Y. Weiss. On the fixed points of the max-product algorithm. *IEEE T. Information Threory*, 2000.

[38] A. Galata, N. Johnson, and D. Hogg. Learning behaviour models of human activities. In *British Machine Vision Conference*, page Modelling Human Behaviour, 1999.

[39] A. Galata, N. Johnson, and D. Hogg. Learning structured behavior models using variable length markov models. In *IEEE International Workshop on Modelling People*, pages xx–yy, 1999.

[40] D. M. Gavrila. The visual analysis of human movement: A survey. *Computer Vision and Image Understanding: CVIU*, 73(1):82–98, 1999.

[41] D. M. Gavrila. Pedestrian detection from a moving vehicle. In *European Conference on Computer Vision*, pages 37–49, 2000.

[42] D.M. Gavrila and L.S. Davis. 3d model-based tracking of humans in action: a multi-view approach. In *CVPR*, pages 73–80, 1996.

[43] K. Grauman, G. Shakhnarovich, and T. Darrell. Inferring 3d structure with a statistical image-based shape model. In *ICCV*, pages 641–648, 2003.

[44] Hemera Technologies, Inc. *Hemera Photo Objects*. http://www.hemera.com.

[45] D. Hogg. Model based vision: a program to see a walking person. *Image and Vision Computing*, 1(1):5–20, 1983.

[46] B.K.P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society A.*, 4(4):629–642, April 1987.

[47] B.K.P. Horn. Relative orientation. *IJCV*, 4(1):59–78, January 1990.

[48] N.R. Howe. Silouette lookup for automatic pose tracking. In *Non-Rigid04*, page 15, 2004.

[49] Weiming Hu, Tieniu Tan, Liang Wang, and Steve Maybank. A survey on visual surveillance of object motion and behaviors. *IEEE Transactions on Systems, Man and Cybernetics, Part C*, 34(3):334–352, Aug 2004.

[50] P. Indyk and R. Motwani. Approximate nearest neighbor - towards removing the curse of dimensionality. In *30th Symposium on Theory of Computing*, 1998.

[51] S. Ioffe and D. A. Forsyth. Human tracking with mixtures of trees. In *ICCV*, 2001.

[52] S. Ioffe and D.A. Forsyth. Finding people by sampling. In *ICCV*, pages 1092–1097, 1999.

[53] S. Ioffe and D.A. Forsyth. Probabilistic methods for finding people. *Int. J. Computer Vision*, 2001.

[54] M. Isard. Pampas: Real-valued graphical models for computer vision. In *CVPR*, 2003.

[55] G. Johansson. Visual perception of biological motion and a model for its analysis. *Perception and Psychophysics*, 1973.

[56] N. Jojic and B. Frey. Learning flexible sprites in video layers. In *CVPR*, 2001.

[57] M. I. Jordan, editor. *Learning in Graphical Models*. MIT Press, Cambridge, MA, 1999.

[58] G. Kitagawa. Non-gaussian state space modelling of non-stationary time series with discussion. *J. Am. Stat. Assoc.*, 82:1032–1063, 1987.

[59] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. In *SIGGRAPH*, 2002.

[60] S. Lazebnik, C. Schmid, and J. Ponce. Affine-invariant local descriptors and neighborhood statistics for texture recognition. In *ICCV*, 2003.

[61] J. Lee, J. Chai, P. Reitsma, J. Hodgins, and N. Pollard. Interactive control of avatars animated with human motion data. In *SIGGRAPH*, 2002.

[62] M.W. Lee and I. Cohen. Human upper body pose estimation in static images. In *ECCV*, 2004.

[63] T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *Int. J. Computer Vision*, 43(1):29–44, 2001.

[64] M.E. Leventon and W.T. Freeman. Bayesian estimation of 3D human motion from an image sequence. Technical Report TR-98-06, MERL, 1998.

[65] D. Liebowitz and S. Carlsson. Un-calibrated motion capture exploiting articulated structure constraints. In *ICCV*, July 2001.

[66] David Lowe. Object recognition from local scale-invariant features. In *ICCV*, 1999.

[67] J.P. MacCormick and A. Blake. A probabilistic exclusion principle for tracking multiple objects. In *Proceedings, Seventh International Conference on Computer Vision*, pages 572–578, 1999.

[68] J.P. MacCormick and M. Isard. Partitioned sampling, articulated objects, and interface-quality hand tracking. In *European Conference on Computer Vision*, pages xx–yy, 2000.

[69] K. Mikolajcyk, C. Schmid, and A. Zisserman. Human detection based on a probabilistic assembly of robust part detectors. In *ECCV*, 2004.

[70] K. Mikolajczyk, R. Choudhury, and C. Schmid. Face detection in a video sequence - a temporal approach. In *CVPR*, 2001.

[71] Thomas P. Minka. From hidden markov models to linear dynamical systems. Technical Report 531, MIT, 1998.

[72] T.B. Moeslund. Summaries of 107 computer vision-based human motion capture papers. Technical Report LLA 99-01, University of Aalborg, 1999.

[73] A. Mohan, C. Papageorgiou, and T. Poggio. Example-based object detection in images by components. In *PAMI*, 2001.

[74] G. Mori and J. Malik. Estimating human body configurations using shape context matching. In *ECCV*, 2002.

[75] G. Mori, X. Ren, A. Efros, and J. Malik. Recovering human body configurations: Combining segmentation and recognition. In *CVPR*, 2004.

[76] T. Mori, Y. Segawa, M. Shimosaka, and T. Sato. Hierarchical recognition of daily human actions based on continuous hidden markov models. In *Automatic Face and Gesture Recognition*, pages 779–784, 2004.

[77] M.Wainwright, T. Jaakola, and A.Willsky. Tree-based reparameterization for approximate inference on loopy graphs. In *NIPS*, 2001.

[78] M. E. Nilsback and B. Caputo. Cue integration through discriminative accumulation. In *CVPR*, 2004.

[79] K. Okumna, A. Taleghani, N. de Freitas, J. Little, and D. Lowe. A boosted particle filter: Multitarget detection and tracking. In *ECCV*, 2004.

[80] N. Oliver, A. Garg, and E. Horvitz. Layered representations for learning and inferring office activity from multiple sensory channels. *CVIU*, 96(2):163–180, November 2004.

[81] N. Oliver, E. Horvitz, and A. Garg. Layered representations for human activity recognition. In *Fourth IEEE International Conference on Multimodal Interfaces*, pages 3–8, 2002.

[82] J. O'Rourke and N. Badler. Model-based image analysis of human motion using constraint propagation. *IEEE PAMI*, 2:522–546, 1980.

[83] V. Pavlovic, J.M. Rehg, Tat-Jen Cham, and K.P. Murphy. A dynamic bayesian network approach to figure tracking using learned dynamic models. In *ICCV*, pages 94–101, 1999.

[84] P. Phillips and E. Newton. Meta-analysis of face recognition algorithms. In *Proceeedings of the Int. Conf. on Automatic Face and Gesture Recognition*, 2002.

[85] C.S. Pinhanez and A.F. Bobick. Pnf propagation and the detection of actions described by temporal intervals. In *Proc. DARPA IU Workshop*, pages 227–234, 1997.

[86] C.S. Pinhanez and A.F. Bobick. Human action detection using pnf propagation of temporal constraints. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 898–904. IEEE Press, 1998.

[87] R. Polana and R.C. Nelson. Detecting activities. In *Proceedings, International Conference on Pattern Recognition*, pages A:815–818, 1994.

[88] D. Ramanan and D. A. Forsyth. Automatic annotation of everyday movements. In *NIPS*, 2003.

[89] D. Ramanan and D. A. Forsyth. Finding and tracking people from the bottom up. In *CVPR*, 2003.

[90] D. Ramanan and D. A. Forsyth. Using temporal coherence to build models of animals. In *ICCV*, 2003.

[91] Deva Ramanan, D.A. Forsyth, and Kobus Barnard. Detecting, localizing, and recovering kinematics of textured animals. In *CVPR*, June 2005.

[92] Deva Ramanan, D.A. Forsyth, and Andrew Zisserman. Strike a pose: Tracking people by finding stylized poses. In *CVPR*, June 2005.

[93] J. Rehg and T. Kanade. Model-based tracking of self-occluding articulated objects. In *ICCV-95*, pages 612–617, 1995.

[94] L. Ren, G. Shakhnarovich, J. Hodgins, H. Pfister, and P. Viola. Learning silhouette features for control of human motion. *ACM Transactions on Graphics*, October 2004.

[95] K. Rohr. Incremental recognition of pedestrians from image sequences. In *CVPR*, pages 9–13, 1993.

[96] R. Ronfard, C. Schmid, and B. Triggs. Learning to parse picture of people. In *ECCV*, 2002.

[97] C. Rother, V. Kolmogorov, and A. Blake. Grabcut - interactive foreground extraction using iterated graph cuts. *Proc. ACM Siggraph*, 2004.

[98] C. Schmid. Constructing models for content-based image retreival. In *CVPR*, 2001.

[99] G. Shakhnarovich, P. Viola, and T. Darrell. Fast pose estimation with parameter-sensitive hashing. In *ICCV*, pages 750–757, 2003.

[100] H. Sidenbladh, M. J. Black, and L. Sigal. Implicit probabilistic models of human motion for synthesis and tracking. In *ECCV*, 2000.

[101] Hedvig Sidenbladh, Michael J. Black, and David J. Fleet. Stochastic tracking of 3d human figures using 2d image motion. In *ECCV*, 2000.

[102] L. Sigal, M. Isard, B. Sigelman, and M. Black. Attractive people: Assembling loose-limbed models using non-parametric belief propagation. In *NIPS*, 2003.

[103] Jeffrey Mark Siskind. Grounding language in perception. *Artifical Intelligence Review*, 8:371–391, 1995.

[104] Jeffrey Mark Siskind. Reconstructing force-dynamic models from video sequences. *Artifical Intelligence*, 151:91–154, 1995.

[105] C. Sminchisescu, A. Kanaujia, Z. Li, and D. Metaxas. Discriminative density propagation for 3d human motion estimation. In *CVPR*, 2005.

[106] C. Sminchisescu and B. Triggs. Covariance scaled sampling for monocular 3d body tracking. In *CVPR*, 2001.

[107] C. Sminchisescu and B. Triggs. Building roadmaps of local minima of visual models. In *ECCV*, 2002.

[108] C. Sminchisescu and B. Triggs. Kinematic jump processes for monocular 3d human tracking. In *CVPR*, 2003.

[109] Yang Song, Xiaolin Feng, and P. Perona. Towards detection of human motion. In *CVPR*, pages 810–17, 2000.

[110] B. Stenger, A. Thayananthan, P. H. S. Torr, and R. Cipolla. Filtering using a tree-based estimator. In *Proc. 9th International Conference on Computer Vision*, volume II, pages 1063–1070, Nice, France, October 2003.

[111] E. Sudderth, A. Ihler, W. Freeman, and A. Willsky. Nonparametric belief propagation. In *CVPR*, 2003.

[112] J. Sullivan and S. Carlsson. Recognizing and tracking human action. In *European Conference on Computer Vision*, 2002.

[113] A. Thayananthan, B. Stenger, P. Torr, and R. Cipolla. Shape context and chamfer matching in cluttered scenes. In *CVPR*, 2003.

[114] K. Toyama and A. Blake. Probabilistic tracking with exemplars in a metric space. *Int. J. Computer Vision*, 48(1):9–19, 2002.

[115] V.N. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, 1996.

[116] M. Varma and A. Zisserman. Texture classification: Are filter banks necessary? In *CVPR*, 2003.

[117] P. Viola, M. Jones, and D. Snow. Detecting pedestrians using patterns of motion and appearance. In *ICCV*, 2003.

[118] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, 2001.

[119] Markus Weber, M. Welling, and Pietro Perona. Unsupervised learning of models for recognition. In *ECCV (1)*, pages 18–32, 2000.

[120] A.D. Wilson and A.F. Bobick. Parametric hidden markov models for gesture recognition. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 21(9):884–900, September 1999.

[121] A.D. Wilson, A.F. Bobick, and J. Cassell. Recovering the temporal structure of natural gesture. In *Proceedings of the Second Int. Workshop on Automatic Face- and Gesture-Recognition*, 1996.

[122] J. Yamato, J. Ohya, and K. Ishii. Recognizing human action in time-sequential images using hidden markov model. In *CVPR*, pages 379–385, 1992.

[123] J. Yangan, Y. Xu, and C. S. Chen. Human action learning via hidden markov model. *IEEE Transactions on Systems Man and Cybernetics*, 27:34–44, 1997.

[124] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. In *International Joint Conference on Artificial Intelligence*, August 2001.

[125] L. Zelnik-Manor and M. Irani. Event-based analysis of video. In *CVPR*, 2001.

[126] J. Zhang, R. Collins, and Y. Liu. Representation and matching of articulated shapes. In *CVPR*, 2004.