Component-Based Tracking

by

Xiangtian Dai

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

August, 2005

Abstract

This thesis presents a visual tracking framework, Component-based Tracking. This framework combines low-level visual clues and constraints in a systematic way. It is based on a probabilistic graphical model to infer the marginal probability density function of the state of a tracked target. In this framework, all potential functions are approximated as weighted sums of multivariate Gaussians so probabilistic inference by message passing is equivalent to hypotheses combination and evaluation. The possibility of low level detectors not providing useful information is taken into account by the introduction of a null hypothesis. Simulation is conducted to verify the framework.

In addition, a new kind of image-level region tracking method is studied. The estimation error of transformation parameters from using kernels is analyzed, and algorithms are proposed to select and construct optimal kernels to minimize the estimation errors of various types of geometric transformations. The algorithms are verified and compared against the method of sum-of-square differences (SSD) by both simulation and using real images. This optimal kernel tracker can be either used by itself or as a part of the component-based tracking framework.

Finally, an experimental application on face tracking by the component-based tracking framework with optimal kernels is developed. Its performance is analyzed and compared against that of particle filters with various numbers of particles.

Advisor: Dr. Gregory D. Hager

Readers: Dr. Gregory D. Hager, Dr. Darius Burschka, and Dr. Rene Vidal

Contents

Abstract ii

List of Figures v

List of Tables vii

Acknowledgements viii

1 Introduction 1
- 1.1 Challenges in General Tracking . . . 1
- 1.2 Image Level Detection and Tracking . . . 2
- 1.3 Kalman Filtering and Data Association . . . 3
- 1.4 Particle Filtering . . . 4
- 1.5 Bottom-up Approaches . . . 4
- 1.6 Probabilistic Graphical Models . . . 5
- 1.7 Thesis Contribution . . . 5

2 Heterogeneous Regions 7
- 2.1 Motivation . . . 7
- 2.2 General Problem Formulation . . . 8
- 2.3 Sum of Square Difference . . . 8
- 2.4 Optimal Kernels . . . 10
  - 2.4.1 Kernels . . . 11
  - 2.4.2 Connection to SSD . . . 12
- 2.5 One-Dimensional Translation Kernels . . . 13
  - 2.5.1 General Error Analysis . . . 13
  - 2.5.2 Roof Kernels . . . 15
  - 2.5.3 "Zigzag" Kernels . . . 17
  - 2.5.4 Multiple Kernels . . . 19
  - 2.5.5 Effectiveness of Kernel Selection . . . 20
- 2.6 More General Kernels . . . 22
  - 2.6.1 Two-dimensional Translation . . . 22
  - 2.6.2 One-Dimensional Scaling . . . 24
  - 2.6.3 Two-Dimensional Scaling and Rotation . . . 28

3 The Tracking Framework 32
- 3.1 Rationale . . . 32
- 3.2 The Graphical Model . . . 33
- 3.3 Message-passing for Inference . . . 34

3.4 Component-based Tracking . . . 35
3.4.1 Components . . . 35
3.4.2 Constraints . . . 37
3.5 Gaussian Mixtures . . . 37
3.6 Linearization of Potentials . . . 39
3.7 Null Hypotheses . . . 42
3.7.1 Purpose . . . 42
3.7.2 Weight Adjustment . . . 46
3.7.3 Operation Rules . . . 47
3.8 Pruning and Merging Hypotheses . . . 48
3.9 Observations . . . 51
3.9.1 Corners . . . 51
3.9.2 Blobs . . . 52
3.9.3 Region Tracker and Sensor Hints . . . 52
3.10 Relationship with Established Works . . . 53

4 Algorithm and Simulation **55**
4.1 Algorithm Description . . . 55
4.2 The Implementation . . . 60
4.3 Simulation . . . 62
4.3.1 Setup . . . 62
4.3.2 Results . . . 63

5 Experiments **68**
5.1 Setup . . . 68
5.2 Particle Filtering . . . 73
5.3 Results . . . 75

6 Conclusion **81**
6.1 Contributions . . . 81
6.2 Future Work . . . 82

Bibliography **84**

A Proofs **89**
A.1 Equation (2.4.13) by Equation (2.4.12) . . . 89
A.2 Equation (3.5.2) and (3.5.3) . . . 90
A.3 Equation (3.5.4) and (3.5.5) . . . 91
A.4 Equivalence of Figure 3.8 and the Kalman Filter . . . 91
A.5 Equation (2.5.19) . . . 93
A.6 Equation (3.6.9) and (3.6.10) . . . 94

B Two Forms of the Multivariate Gaussian **95**
B.1 Natural Form . . . 95
B.2 Homogeneous Form . . . 96

C A Sample of the Source Code **97**

Vita **99**

List of Figures

2.1 The Algorithm to Select Peak Position of 1-D Roof Kernel 17
2.2 Error of Roof Kernel over Peak Position 21
2.3 The Algorithm to Construct 2-D Roof Kernel for X-translation 24
2.4 Image for 2-D Translation Tracking by Composed Roof Kernel 25
2.5 Error of Two-parameter Estimation of Composed Roof Kernel and SSD 26
2.6 Convergence Ranges of Composed Roof Kernel and SSD 27
2.7 The Algorithm to Construct 1-D Scaling Kernel 28
2.8 Estimation Error of 1-D Scaling Kernel 29
2.9 The Algorithm to Construct 2-D Scling and Rotation Kernels 31
2.10 Estimation of Rotation Kernel 31

3.1 A Simple Graph Example 33
3.2 History Components: the Simplification of Temporal Structure 36
3.3 A Forbidden Graph 41
3.4 A Simple Graph (again) 42
3.5 A Possible Situation with Node X Missing. 43
3.6 The Ensemble for All Four Possible Situations 44
3.7 Message from a Potentially Missing Sub-graph 47
3.8 Kalman Filter Equivalence Graph 54

4.1 The Standard Algorithm of Running the Graph for Each Time-step 56
4.2 The Algorithm of Running the Graph for Each Time-step 57
4.3 The Algorithm of Merging Gaussian Terms 58
4.4 The Algorithm of Pruning the Number of Gaussian Terms 58
4.5 Handling of Delayed Linearization of Potentials 59
4.6 Adjusting the Weights of the Null Hypotheses 59
4.7 Structure of a Typical Application 60
4.8 Simplified Class Diagram of the System 61
4.9 The Graph for Simulated Experiment 62
4.10 Result from a Typical Frame in Simulation 64
4.11 Result from Missing Object Part in Simulation 65
4.12 Tracking Error for Part A in Simulation 66
4.13 Weight of the Best Hypothesis for Part A in Simulation 67

5.1 An Image from the Video Sequence 69
5.2 Templates of Tracked Regions 69
5.3 Difficult Scenes in the Video Sequence 70
5.4 The Graph for the Face-tracking Experiment 71
5.5 Estimation Error with and without Constraints 76

5.6 Comparison between Tracking with and without Constraints 76
5.7 Estimation Errors of Component-based Tracking and Particle Filtering 77
5.8 Estimation Errors of Component-based Tracking and Particle Filtering (more) . . . 77
5.9 Comparison between Component-based Tracking and Particle Filtering 78
5.10 An Image from Another Video Sequence 79
5.11 Estimation Errors of Component-based Tracking and Particle Filtering (2) 80

A.1 Labelled Kalman Filter Equivalence Graph 91

List of Tables

2.1 The Effect of Peak Selection for Roof Kernels 22

3.1 Examples of Types of Observation Components 35
3.2 Examples of Types of Observation Components 37

5.1 Result Summary of Tracking Methods 80

Acknowledgements

I would like to express my deep gratitude to Professor Gregory D. Hager, without whom the entire work would be impossible.

I would like to thank Le Lu for our collaboration that inspired this very idea of component-based tracking, to thank Maneesh Dewan for discussions on optimal kernels, and to thank the rest of the Computational Interaction and Robotics Laboratory for all kinds of suggestions and help.

Dedicate to my dearest wife Qian

Chapter 1

Introduction

1.1 Challenges in General Tracking

A general visual tracking problem can be briefly summarized as estimating the state sequence of a target from an observed image sequence or sequences, given a description of the target in some predefined form. A usual assumption is the Markovian property of the state:[1] in this case the problem takes the simpler form of estimating the target state from the most recently observed image or images and the existing estimate of the previous state.

If we consider the relationship between the state of the target and the observed image or images, we can view tracking as a visual detection and recognition problem. A great part of the image processing and machine vision literature is about this kind of problem, and a large number of techniques and algorithms have been developed, some of which are already applied in industry. However, if unfavorable conditions and a less well-controlled environment are taken into account, for example, ambiguity, occlusion, and unstable illumination, this problem is still far from solved in general.

A second relationship is between the previous state and the current state. This is about predicting the target's temporal behavior, either by having some knowledge about the mechanism that changes the target's state, or by statistical or intuitive means. The temporal behavior of the target state is hardly deterministic, sometimes not even well modeled. This means that even if the previous state estimate is accurate, it is not a reliable source of current state estimate by itself.

Further, if the target object consists of several parts, either physically or visually, or both, and these parts can be detected or recognized individually, even in a less reliable and accurate manner, in addition to composing the overall state of the target, the relationship between the parts may contain constraints that can be exploited to refine the result.

In summary, there are many sources of information that can be used for visual tracking,

[1] Even the state in its nature form is not Markovian, often we can find an expanded state that is.

but none of them are fully dependable. Only a combination of them can produce the best result.

Component-based tracking is a probabilistic visual tracking framework that aims to organize these sources of information according to the description of the target. Each component is a set of probabilistic variables, represented as a node in the graph. Each constraint is a potential function, represented as an edge in the graph. Probabilistic inference is done by passing messages, which are functions. The intention is to organize all sources of information in a formal way so that visual tracking can be performed accurately, robustly and efficiently.

Obviously, from the point of view of artificial intelligence and machine learning, inference in a graphical model is the easier part of the problem; learning the parameters of the model is hard, and learning the structure of the model is even harder. However, this thesis is focused on the application of this probabilistic graphical model in this particular visual tracking problem, so these learning problems are subjects of further research. Instead, the structure of the model is formed in natural and intuitive ways, and the parameters are derived by empirical methods.

In the rest of the chapter, we review some of the related work in the field.

1.2 Image Level Detection and Tracking

There are all kinds of visual cues available on images for tracking. For example, the Canny edge detector[4] locates the edge points by following the direction of local gradient while performing non-maximal suppression; the Harris corner detector[21] computes the "strength" of corner points by comparing the eigenvalues of the local moment matrices computed from the image gradients. A review[48] compares several types of features on how well they can be tracked and provides a feature selection criterion. However, in this thesis we mainly focus on region tracking.

A widely applied region tracking technique, blob tracking, relies on the statistics of the pixels of the target region. For example, Pfinder[56] describes the target by a multi-class statistical model of color and shape in different viewing conditions. BraMBLe[25] computes image likelihood by comparison of unified foreground and background statistical models. On the other hand, Sum-of-Square-Difference (SSD) tracking[17] relies on the spatial structure of the target region. It computes a linear operator for estimating the transformation parameters by minimizing the linearized sum of square difference between the transformed image and the template. Hyperplane Tracking[29] extends this idea by using artificial training instead of internal deduction of SSD to construct the operator. More recently, kernels are used to exploit both the pixel statistics and structure of the region. For example, Mean-Shift Tracking[6] detects the modes of an underlying density function in a multi-model feature space by recursively applying the mean-shift procedure. Kernel Tracking with SSD[18] estimates the transformation parameters from multiple kernels by solving the kernel-based objective function in the SSD-like way.

In many cases, a target is composed of multiple visual cues. XVision[20] uses a simple constraint hierarchy to organize the trackers for individual visual cues. Incremental Focus of Attention

(IFA)[53] suggests a layered hierarchy for selecting visual cues and accordingly adjusts estimated ranges of configurations. Data Fusion with Particles[44] uses generic important sampling for weighing and selecting a number of sources of information[2] under the particle filter framework.

1.3 Kalman Filtering and Data Association

In a dynamical linear system that indicates how a state vector changes over time, and a linear observation model that relates the state vector to an observation vector, Kalman filtering[30] recursively estimates the state vector from observed measurements under the assumption that the noises in both equations are white Gaussians. The Kalman filter gives optimal results in the sense of MMSE (Minimum Mean Squared Error) or Bayesian equivalently MAP (Maximum A Posteriori).

There are various Kalman filters extensions. For non-linear systems, the Extended Kalman Filter (EFK)[50] is the most natural extension: it simply linearizes all non-linear models. The Unscented Kalman Filter (UKF)[28] employs an unscented transformation to capture the higher order statistics of the uncertainty via sigma points in an effort to make prediction and estimation more accurate.

Furthermore, the assumption of white Gaussian noise may not be a good approximation. If the noise can be modeled, then the equations of Kalman filtering can be adjusted accordingly, such as in [5]. Bias-aware Kalman Filters (BAKF) in [14] and [8] automatically correct the prediction bias to some extent without an explicit model, under the assumption that a set of unbiased observations exists.

A general method to deal with non-linear and non-white-Gaussian system is the Ensemble Kalman Filter (EnKF)[11]. Statistics are being collected from the ensemble of Monta Carlo samples that go through the system. In some way, EnKF is similar to the particle filter which will be discussed in next sub-section.

Another way to approximate non-parameteric probability density is to use a sum (or a mixture) of Gaussians[51][1]. The Gaussian-sum Filter (GSF)[15], in particular, uses a mixture of Gaussians to approximate the probability distribution. As a result, the "core" of the distribution is described by the "main" Gaussian component and the "tails" of the distribution are described by either one or several additional Gaussian components.

A related problem in applying Kalman filters is data association[3][19]. The mission is to associate multiple targets with their corresponding measurements (or observations), under the presence of false measurements or absence of part of the measurements. Some approaches are target-oriented: the Probabilistic Data Association Filter (PDAF) puts the target through all plausible measurement, including the possibility that no measurement is associated with the target at all, and weighs them in probabilistic terms. The Joint Probabilistic Data Association Filter (JPDAF) further takes into

[2] One of their sources of information is stereo audio

account the constraints between targets or target parts. Some other approaches to the data association problem are measurement-oriented: Multiple-hypothesis Tracking (MHT)[46] is an algorithm that manages the creation, evaluation and removal of hypotheses from measurements. The efficiency of the algorithm can be improved by taking the cost ranking[42] of hypotheses into consideration[7]. Similarly, the Multiple Multistage Hypothesis Test Tracking (MMHTT) algorithm[47] uses a truncated sequential probability ratio test to prune the hypothesis combination tree for computation efficiency.

1.4 Particle Filtering

Particle Filtering[33], or Condensation[23], sometimes referred as Sequential Importance Sampling (SIS)[10], is another family of tracking algorithms based on a totally different representation of the probability density function: a large set of samples instead of a few parameterized functions. In particle filters, a set of weighted particles is used to represent the probabilistic distribution of the state as its samples. Temporal dynamics are described as resampling functions. Thus, Particle Filtering has no limit on system linearity or Gaussianity. However, like any other sampling-based method, Particle Filtering only provides accurate results in an asymptotic sense[32].

As the dimension of the problem increases and the information carried from observations becomes complex in structure and unevenly distributed in the state space, the sampling efficiency of particle filters becomes an issue. One way to avoid using more samples than necessary is to adjust the number of samples on the fly according to a difference measurement between prior and posterior probability distributions[13]. If the state space can be somehow divided into subspaces, then either partitioned sampling methods can be used[40], or Kalman filters can be used in some subspaces[39]. On the other hand, more promising particles can have a better chance to be resampled by taking hints from low-level visual clues directly[24], or by introducing an auxiliary variable[45].

It is worth noting that if we think of every sample in the particle filter as a hypothesis or "proposal", Kalman filtering can be applied on each of them to greatly increase the efficiency of individual samples and thus that of the whole sample set, as in the Gaussian Sum Particle Filter (GSPF)[34]. Existing improvement of Kalman filters, such as the unscented transform, can be further applied as well, as in the Unscented Particle Filter (UPF)[54].

1.5 Bottom-up Approaches

Particle filtering is mostly a top-down, model-driven approach: the only requirement of the data likelihood function is that it can be evaluated at any arbitrary point. In other words, observed data is used to validate generated hypotheses. In contrast, there are some bottom-up, data-driven algorithms: constraints are used to validate hypotheses generated by observed data. RANSAC[12] randomly picks data points to generate hypotheses, each of them is verified by a

robust estimator and the best one is used. The Hough transform[22] basically lets data points "vote" for the best hypothesis. Randomized Hough transform[57], like RANSAC, randomly samples from sets of data points, then accumulates votes in the parameters state space with improved data structure. Random Window RHT (RWRHT)[31] further increases the efficiency of vote by limiting the voter in a randomly sized window.

These bottom-up approaches can be utilized in the Particle Filtering framework to greatly increase the efficiency of the particles, as in[37, 38]. The tracking framework proposed here is also inspired by these approaches.

1.6 Probabilistic Graphical Models

Probabilistic graphical models[43, 35] use graph structures to describe probabilistic relationships between random variables, each represented by a node or vertex of the graph. It is a general mathematical form for which both Markov Random Field (MRF)[16] and Bayesian network[26] are example as undirected and directed graphs. More specific models, such as Hidden Markov Models (HMM), can be thought of as a special case of probabilistic graphical models[49].

In probabilistic graphical models, probabilistic inference is done by a belief propagation algorithm. Functions of partial sums, or "messages", are delivered between vertexes (or nodes) of the graph. If the graph is loop-less, such algorithms can be completed in finite number of iterations with the exact answer[3][27]. However, for multiple connected graphs, the same algorithms are not guaranteed to converge. Nevertheless, they are still used for approximate inference with good experimental results[41]. Furthermore, if the vertexes are divided by regions (or clusters), more efficient methods can be used, such as General Belief Propagation (GBP)[58]. An interesting point is that Gaussian graphical models have better properties in theory[55]. Nonparametric Belief Propagation (NBP)[52] uses sums of Gaussians representation and uses sampling for their operations.

1.7 Thesis Contribution

The thesis has two closely related topics, each of which addresses the limitation of existing methods and proposes the new approach.

The first topic, image level tracking of heterogeneous regions, is discussed in chapter 2. SSD tracking is good in accuracy and efficiency, but since it directly depends on the local image gradient structure, its convergence range is rather unpredictable. Hyperplane tracking extends the convergence range, but it is computational costly on initialization. In this thesis, optimal kernel tracking is proposed as a novel method to track heterogeneous regions. The introduction of the kernel not only makes the algorithm focus on the large-scale structure of the image region, but

[3] put aside the practical issue of representing and summing or integrating functions for the moment

also provides a new angle to optimize on. This results in a tracker that is both accurate and computationally efficient, and in addition has greater convergence range.

The second topic is a new general tracking framework called component-based tracking, which is described in chapter 3 and 4. All sources of information for tracking mentioned in the beginning of the chapter are systematically combined in this framework. While the proposed framework is mathematically based on probabilistic graphical models, it does not use sampling to make inference on the graph as most of other graphical-model-based approaches and the particle filter do. This is to avoid the cost associated with the large number of samples or particles which is necessary to accurately describe the probability density. Weighted sums of Gaussians are used to described the probability density instead. The inference algorithm is driven by both target model and measurement data, to avoid the overly reliance of either. In addition, robustness is achieved by the introduction of "null hypothesis" as a general notation of lack of information from a branch of the graph.

Chapter 2

Heterogeneous Regions

2.1 Motivation

Regions are one of the basic elements used for tracking. There are roughly two classes of regions to be tracked: homogeneous regions are better described by their pixel (or feature) statistics, and heterogeneous regions are better described by their internal structure. More specifically, such a description for the latter can be in the terms of one or a set of model images, or templates, and a family of feature-level and geometric deformations. This chapter shall focus on one class of approach to these kind of problems.

Assuming the target in the observed image is a distorted version of the known original image under a class of transformation, the task now is to find the best transformation parameters. The Sum-of-Square-Difference (SSD) method defines "best" by the minimal sum of square difference between the observed image and transformed image. First order approximation of this error term leads to a linear operator on the image to generate the new transformation parameter estimate. However, its convergence range is subject to local fluctuation of the image gradient, which is the derivative of the image.

If the derivative is not taken directly on the image itself, but on some other function that we have more control on, one would expect the tracker to have better convergence range. This way of thinking naturally leads to the introduction of kernels: the product of the transformed image and the kernel can be expressed as the product of the original image and the inversely transformed kernel, so variation on transformation parameters only directly changes the kernel but not the image. Since we construct the kernel ourselves in the first place, it can have desirable properties for the purpose of taking derivatives. Figuratively, the kernel "smoothes" the image.

The introduction of a kernel provides an additional benefit: if the class of kernels has simple analytical properties, we can estimate the cost associated with parameter estimation error. Thus, we can select the "best" kernel for the given image in the sense that the estimated cost of estimation

error is minimal.

In this chapter, we estabilsh the general problem formulation in section 2.2, briefly review SSD method in section 2.3, and derive optimal kernel tracking in the rest of the sections.

2.2 General Problem Formulation

Assume we have

1. a known foreground signal $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^r$, where n is the number of dimensions of the coordinate space and r is the number of signal bands;
2. a family of transformations $\mathbf{T} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$, where m is the number of dimensions of transformation parameters τ;
3. a background signal $\mathbf{b} : \mathbb{R}^n \to \mathbb{R}^r$, in the sense of "background" that $\forall_{\mathbf{x}\in\mathbf{P}}\mathbf{b}(\mathbf{x}) = 0$, where $\mathbf{P}$ is a given region in $\mathbb{R}^n$; and
4. a random additive noise process $n : \mathbb{R}^n \to \mathbb{R}^r$.

We are asking for a function[1] $\mathbf{k} : \mathbb{R}^n \to \mathbb{R}^{m\times r}$, such that for observation $\mathbf{g}$ generated by

$$\mathbf{g}(\mathbf{x};\tau) = (\mathbf{f} + \mathbf{b})(\mathbf{T}(\mathbf{x};\tau)) + \mathbf{n}(\mathbf{x}) \tag{2.2.1}$$

$\mathbf{k}$ can be used to estimate the transformation parameter as

$$\hat{\tau} = \int_{\mathbf{x}\in\mathbf{P}} \mathbf{k}(\mathbf{x})(\mathbf{g}(\mathbf{x};\tau) - \mathbf{f}(\mathbf{x}))\ \mathrm{d}\mathbf{x} + \tau_0 \tag{2.2.2}$$

This formulation is for tracking so that we expect τ be in a small neighborhood of $\tau_\mathbf{0}$ and we assume

$$\mathbf{T}(\mathbf{x};\tau_\mathbf{0}) = \mathbf{x} \tag{2.2.3}$$

2.3 Sum of Square Difference

As noted previouly, SSD method directly deals with 2-D grey-level images. so $I(\mathbf{x})$ is used in this section as the notation instead of $f(\mathbf{x})$. The objective of SSD can be stated as given a template image $I_0(\mathbf{x})$, to find the parameters τ for input image $I_0(\mathbf{x})$ so that

$$e = \sum_{i=1}^{N} \left(I(\mathbf{T}(\mathbf{x}^{(i)};\tau)) - I_0(\mathbf{x}^{(i)})\right)^2 \tag{2.3.1}$$

is minimized. Here $\mathbf{x}^{(i)}$, $i = 1,\ldots,N$ is the pixel grid of the image. I may need to be interpolated as $\mathbf{T}(\mathbf{x}^{(i)};\tau)$ is not necessarily on the grid.

[1] The value is an m by r matrix.

Note that the above minimization is equivalent[2] to the equation below under the assumption of additive white Gaussian noise n:

$$I(\mathbf{T}(\mathbf{x}^{(i)};\tau)) = I_0(\mathbf{x}^{(i)}) + n_i \tag{2.3.2}$$

If we linearize (approximate to first order) the right hand side of equation (2.3.1) at point $\tau = \tau_0$, we have

$$e \approx \sum_{i=1}^{N} \left(I(\mathbf{T}(\mathbf{x}^{(i)};\tau_0)) + \left.\frac{\partial \mathbf{I}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{T}(\mathbf{x}^{(i)};\tau_0)} \left.\frac{\partial \mathbf{T}}{\partial \tau}\right|_{\tau=\tau_0} (\tau - \tau_0) - I_0(\mathbf{x}^{(i)}) \right)^2 \tag{2.3.3}$$

Let n and m be the numbers of dimensions of $\mathbf{x}$ and τ, respectively. Using subscript to denote elements of a vector or a matrix, we can rewrite right hand side of equation (2.3.3) in the matrix form:

$$e \approx |GD(\tau - \tau_0) - E|^2 \tag{2.3.4}$$

where G is the N by m image gradient matrix defined as

$$G_{ij} = \left.\frac{\partial I}{\partial x_j}\right|_{\mathbf{x}=\mathbf{x}^{(i)}} \tag{2.3.5}$$

D is the m by n transformation derivative (Jacobian) matrix defined as

$$D_{ij} = \left.\frac{\partial T_j}{\partial \tau_i}\right|_{\tau=\tau_0} \tag{2.3.6}$$

E is the N by 1 "error" (column) vector

$$E_i = I(\mathbf{T}(\mathbf{x}^{(i)};\tau_0)) - I_0(\mathbf{x}^{(i)}) \tag{2.3.7}$$

and $|V|$ stands for the L2-norm (magnitude) of vector V.

Equation (2.3.4) is in the form of standard over-constrained (normally $N \gg n$) least-squares problem and can be solved by existing routines using SVD (or QR or LQ) decomposition.

Furthermore, if τ_0 is already a close guess[3] in the sense that

$$I(\mathbf{T}(\mathbf{x};\tau_0)) \approx I(\mathbf{x}) \tag{2.3.8}$$

G can be approximated by

$$G_{ij} \approx \left.\frac{\partial I_0}{\partial x_j}\right|_{\mathbf{x}=\mathbf{x}^{(i)}} \tag{2.3.9}$$

so that G no longer depends on the input image I. Now we can precompute the pseudo-inverse of GD denoted by $(GD)^-$ and use it to estimate τ for all input image I by

$$\hat{\tau} = \tau_0 - (GD)^- E \tag{2.3.10}$$

[2] In the sense that the τ that minimizes equation (2.3.1) is also a maximum-likelihood estimator to the parameters τ under equation (2.3.2).

[3] To allow equation (2.3.3), τ_0 has to be in the neighborhood of τ anyway.

which has time complexity of only $O(mN)$.

In addition to geometric transformations, photometric transformations can also be allowed easily into the setting of SSD. If equation (2.3.1) becomes

$$e = \sum_{i=1}^{N} \left(F(I(\mathbf{T}(\mathbf{x}^{(i)}; \tau)); \sigma) - I_0(\mathbf{x}^{(i)}) \right)^2 \tag{2.3.11}$$

where $F(I; \sigma)$ is a class of photometric transformation with parameters σ on the image, then by similar deductions we have

$$\begin{pmatrix} \hat{\tau} \\ \hat{\sigma} \end{pmatrix} = \begin{pmatrix} \hat{\tau_0} \\ \hat{\sigma_0} \end{pmatrix} - \begin{pmatrix} GD & P \end{pmatrix}^{-} E \tag{2.3.12}$$

where P is the derivative (Jacobian) matrix of F:

$$P_{ij} = \left. \frac{\partial F}{\partial \sigma_j} \right|_{\mathbf{x}=\mathbf{x}^{(i)}, \sigma=\sigma_0} \tag{2.3.13}$$

2.4 Optimal Kernels

SSD tracking is an established technique. Its accuracy is up to sub-pixel level for a close initial guess, or after a few iterations provided it converges. However, because of the significant spatial fluctuation of image gradient which is used for first-order approximation, the convergence range of SSD for transformation parameters is highly dependent on the smoothness of the image gradients. In practice, both the template and the input image need to be smoothed beforehand to reduce the gradient fluctuation.

If we look at the equation (2.3.10) or equation (2.3.12), the estimate of each parameter in addition to its initial value is actually the inner product between an "operator" vector and the error vector. Specifically, in equation (2.3.10), if we define $A = (GD)^{-}$ and A_i be the i-th row of A, then A_i is that "operator" vector:

$$\tau_i = A_i E \tag{2.4.1}$$

Of course, A_i is derived from the templates image I_o and class of transformation T. This form is desirable for its minimal computational cost. SSD provides one way to derive A. In this and the following sections we seek another way to derive A to minimize the estimation error while avoiding the relatively unpredictable convergence range of SSD caused by relying on the fine structure of image gradients.

Instead of the SSD objective function on the image in equation (2.3.1), our criteria in optimal kernel tracking is directly based on the accuracy of the estimator in equation (2.2.2). This is because our task here is not to restore image or put images into a mosaic, but to estimate the transformation parameters which are to be used by applications or high-level tracking frameworks. Thus, the "cost" of estimation error is a function of the actual parameters and the estimated parameters.

Here we use weighted mean square error, a common cost function:

$$e = E\left[\sum_{i=1}^{m} w_i(\hat{\tau}_i - \tau_i)^2\right] \tag{2.4.2}$$

The weights w_i, $i = 1, \ldots, m$ are used to balance parameter dimensions. They are omitted for one dimensional problems. The expectation is taken over both the randomness of $\mathbf{n}$ and τ, and possibly over $\mathbf{b}$ as well, namely

$$e = E_{\mathbf{b}}E_{\mathbf{n}}E_{\tau}\left[\sum_{i=1}^{m} w_i(\hat{\tau}_i - \tau_i)^2\right] \tag{2.4.3}$$

We can further assume that the noise $\mathbf{n}$ is zero-mean white Gaussian, i.e.

$$E[\mathbf{n}(\mathbf{x})] = 0; \tag{2.4.4a}$$

$$E[n(\mathbf{x}_1)n(\mathbf{x}_2)] = \sigma_n^2\delta(\mathbf{x}_1 - \mathbf{x}_2) \tag{2.4.4b}$$

2.4.1 Kernels

There are many ways to construct $\mathbf{k}$. One may try to solve it directly via equation (2.4.2), by either differential equations on the functional domain or matrix manipulation on discrete samples. Here we present a more constrained approach. $\mathbf{k}$ is to be derived by a linear combination of kernels $K_i(\mathbf{x})$, $i = 1, \ldots, m_K$, each of them a function $\mathbb{R}^n \to \mathbb{R}^r$. Write $\mathbf{K}(\mathbf{x})$ as $[K_1(\mathbf{x}), \ldots, K_{m_K}(\mathbf{x})]^T$ so that $\mathbf{K}$ has a range of m_K by r matrices.

The square difference between the observation $\mathbf{g}$ and the transformed original signal $\mathbf{f}(\mathbf{T}(\mathbf{x}; \hat{\tau}))$ convolved by the kernels $\mathbf{K}$ can be approximated to first order:

$$\begin{aligned} e_K &= \left\| \int_{\mathbf{x}\in\mathbf{P}} \mathbf{K}(\mathbf{x}) \left(\mathbf{g}(\mathbf{x}) - \mathbf{f}(\mathbf{T}(\mathbf{x}; \hat{\tau}))\right)\, \mathrm{d}\mathbf{x} \right\|^2 \\ &\approx \left\| \int_{\mathbf{x}\in\mathbf{P}} \mathbf{K}(\mathbf{x}) \left(\mathbf{g}(\mathbf{x}) - \mathbf{f}(\mathbf{x})\right)\, \mathrm{d}\mathbf{x} + \mathbf{D}(\hat{\tau} - \tau_0) \right\|^2 \end{aligned} \tag{2.4.5}$$

where $\mathbf{D}$ is

$$\mathbf{D} = \frac{\mathrm{d}}{\mathrm{d}\tau} \int_{\mathbf{x}\in\mathbf{P}} \mathbf{K}(\mathbf{x})\mathbf{f}(\mathbf{T}(\mathbf{x}; \tau))\, \mathrm{d}\mathbf{x} \Big|_{\tau=\tau_0} \tag{2.4.6}$$

To find $\hat{\tau}$ that minimizes equation (2.4.5), we need

$$\frac{\mathrm{d}}{\mathrm{d}\hat{\tau}} e_K = 0 \tag{2.4.7}$$

which leads to

$$\mathbf{D}(\hat{\tau} - \tau_0) = \int_{\mathbf{x}\in\mathbf{P}} \mathbf{K}(\mathbf{x}) \left(\mathbf{g}(\mathbf{x}) - \mathbf{f}(\mathbf{x})\right)\, \mathrm{d}\mathbf{x} \tag{2.4.8}$$

so that

$$\hat{\tau} = \int_{\mathbf{x}\in\mathbf{P}} \mathbf{D}^{-}\mathbf{K}(\mathbf{x}) \left(\mathbf{g}(\mathbf{x}) - \mathbf{f}(\mathbf{x})\right)\, \mathrm{d}\mathbf{x} + \tau_0 \tag{2.4.9}$$

where $\mathbf{D}^{-}$ is a pseudo-inverse of the matrix $\mathbf{D}$.

Compare equation (2.4.9) to equation (2.2.2),

$$\mathbf{k}(\mathbf{x}) = \mathbf{D}^{-}\mathbf{K}(\mathbf{x}) \tag{2.4.10}$$

Now as the form of $\mathbf{k}$ is constrained, K_i is picked from a family of kernels to optimize on equation (2.4.2).

The purpose of introducing a kernel is to take the derivative of the kernels instead of the derivative of the signal itself, since the former is usually smoother for better convergence and easier to differentiate by construction. This is achieved by change of integration variable:

$$\mathbf{D} = \frac{\mathrm{d}}{\mathrm{d}\tau} \int_{\mathbf{x}\in\mathbf{T}(\mathbf{P};\tau)} \left|\frac{\partial \mathbf{T}^{-1}}{\partial \mathbf{x}}\right| \mathbf{K}(\mathbf{T}^{-1}(\mathbf{x};\tau))\mathbf{f}(\mathbf{x})\ \mathrm{d}\mathbf{x}\Bigg|_{\tau=\tau_0} \tag{2.4.11}$$

where $\mathbf{T}^{-1}$ is the inverse of $\mathbf{T}$.

Note that the range of integration is also a function of τ, so that there shall be an extra term when moving the derivative sign into the integration sign. If we mandate the boundary condition of the kernel:

$$\forall_{\mathbf{x}\in\mathbf{B_P}} \mathbf{K}(\mathbf{x}) = 0 \tag{2.4.12}$$

where $\mathbf{B_P}$ is the boundary of P under the transformation $\mathbf{T}$ at $\tau = \tau_0$, then this extra term is approximately equal to 0 to the first order. Appendix A.1 details the proof of this condition. Thus we can write:

$$\mathbf{D} \approx \int_{\mathbf{x}\in\mathbf{P}} \left[\frac{\partial}{\partial\tau}\left|\frac{\partial \mathbf{T}^{-1}}{\partial \mathbf{x}}\right| \mathbf{K}(\mathbf{x}) + \left|\frac{\partial \mathbf{T}^{-1}}{\partial \mathbf{x}}\right| \frac{\partial \mathbf{T}^{-1}}{\partial \tau}\mathbf{K}'(\mathbf{x})\right] \mathbf{f}(\mathbf{x})\ \mathrm{d}\mathbf{x}\Bigg|_{\tau=\tau_0} \tag{2.4.13}$$

where

$$\mathbf{K}'(\mathbf{x}) = \frac{\mathrm{d}}{\mathrm{d}\mathbf{x}}\mathbf{K}(\mathbf{x}) \tag{2.4.14}$$

Equations (2.4.9) and (2.4.13) are the formulas for estimating transformation parameters for a given set of kernels. The real problem lies in how to find the set of kernels that has the minimum estimation error as defined in equation (2.4.2) given the signal (e.g. image) being tracked. We can further limit the possible choices of the optimal kernels within a given family of kernels. So is the topic for the rest of the chapter.

Also, for the rest of the chapter we shall limit ourselves in single-band signals (grey-level images for example). In other words, r=1 in section 2.2.

2.4.2 Connection to SSD

Suppose $\mathbf{x}$ takes values over a grid, i.e. a subset of $\mathbb{Z}^n$. Let $\mathbf{x}^{(i)}$ be points on the grid. If we choose kernels $K_i(\mathbf{x})$ as

$$K_i(\mathbf{x}) = \delta(\mathbf{x} - \mathbf{x}^{(i)}) \tag{2.4.15}$$

then the objective function to minimize in equation (2.4.5) turns into the classic SSD formulation for tracking[4].

$$\begin{aligned} e_K &= \sum_{i=1}^{m_K} \left(\int_{\mathbf{x}\in\mathbf{P}} K_i(\mathbf{x})(\mathbf{g}(\mathbf{x}) - \mathbf{f}(\mathbf{T}(\mathbf{x};\hat{\tau}))) \, \mathrm{d}\mathbf{x} \right)^2 \\ &= \sum_{i=1}^{m_K} \left(\mathbf{g}(\mathbf{x}) - \mathbf{f}(\mathbf{T}(\mathbf{x}^{(\mathbf{i})};\hat{\tau})) \right)^2 \end{aligned} \tag{2.4.16}$$

However, as a result the SSD method takes the derivative of the signal itself when making the first order approximation in its next step.

2.5 One-Dimensional Translation Kernels

In this section we focus on the one dimensional translation kernels, both to give a concrete example and to develop some interesting results. This implies $n = 1$ and $m = 1$ (and $r = 1$ too as previously assumed). Notation-wise, many of the vectors in the previous section are now scalars so that they are no longer written in bold font.

2.5.1 General Error Analysis

The transformation T has only one single parameter u for shifting:

$$T(x) = x - u \tag{2.5.1}$$

with the initial condition $u_0 = 0$.

Substituting into equation (2.2.1), we have

$$g(x) = f(x - u) + b(x - u) + n(x); \tag{2.5.2}$$

For now, only one kernel ($m_K = 1$) is to be used. Denote this kernel simply as K. Letting $P = [a, b]$ $(a < b)$, equation (2.4.6) becomes

$$\begin{aligned} D &= \frac{\mathrm{d}}{\mathrm{d}u} \int_a^b K(x) f(x - u) \, \mathrm{d}x \Big|_{u=0} \\ &= \frac{\mathrm{d}}{\mathrm{d}u} \int_{a-u}^{b-u} K(x + u) f(x) \, \mathrm{d}x \Big|_{u=0} \\ &= \int_a^b K'(x) f(x) \, \mathrm{d}x + (K(b)f(b) - K(a)f(a)) \end{aligned} \tag{2.5.3}$$

where $K'(x) = \frac{\mathrm{d}}{\mathrm{d}x} K(x)$.

[4]The usual way to write SSD is to take the difference between the original and inverse-transformed observed signal instead of the one between the transformed and the observed signal as shown here, however, the two objective functions are equivalent under a change of variables.

Now equation (2.4.10) becomes

$$k(x) = D^{-1}K(x); \tag{2.5.4}$$

and the estimator equation (2.2.2) then becomes

$$\hat{u} = D^{-1}\int_a^b K(x)(g(x;u) - f(x))\,\mathrm{d}x \tag{2.5.5}$$

Substituting back into equation (2.4.2), and assuming independence between u, b and n, and without loss of generality assuming the zero-mean of b,[5] we have

$$\begin{aligned} e &= E\left[\left(D^{-1}\int_a^b K(x)(f(x-u) - f(x) + b(x-u) + n(x))\,\mathrm{d}x - u\right)^2\right] \\ &\triangleq e_f + e_b + e_n \end{aligned} \tag{2.5.6}$$

where

$$e_f = E\left[\left(D^{-1}\int_a^b K(x)(f(x-u) - f(x))\,\mathrm{d}x - u\right)^2\right] \tag{2.5.7a}$$

$$e_b = E\left[\left(D^{-1}\int_a^b K(x)b(x-u)\,\mathrm{d}x\right)^2\right] \tag{2.5.7b}$$

$$e_n = E\left[\left(D^{-1}\int_a^b K(x)n(x)\,\mathrm{d}x\right)^2\right] \tag{2.5.7c}$$

The three error terms represent the estimation errors caused by the signal itself, background, and noise, respectively. Let us look at the background error e_b first. Since $b(x) = 0$ for all x that[6] $a \le x \le b$, we have

$$e_b = \begin{cases} E\left[\left(D^{-1}\int_a^{a+u} K(x)b(x-u)\,\mathrm{d}x\right)^2\right] & u \ge 0 \\ E\left[\left(D^{-1}\int_{b+u}^{b} K(x)b(x-u)\,\mathrm{d}x\right)^2\right] & u < 0 \end{cases} \tag{2.5.8}$$

Now if we mandate $K(a) = K(b) = 0$, plus the continuity of K and a small variation of u, we can expect the e_b to be approximately zero so that it does not become the primary consideration when looking for an optimal kernel. The side-benefit of such mandate is that equation (2.5.3) is simplified to

$$D = \int_a^b K'(x)f(x)\,\mathrm{d}x \tag{2.5.9}$$

[5] Because the estimator in equation (2.5.5) does not change if both $g(x)$ and $f(x)$ increase by a constant factor, we can always subtract the mean of $b(x)$ from everything.

[6] Although the same symbol is used, the number b of boundary $[a, b]$ and the function of background $b(x)$ should be distinguishable according to their context.

The noise error e_n is simple given the zero-mean white Gaussian assumption in the previous section:

$$e_n = \sigma_n^2 D^{-2} \int_a^b K^2(x)\ \mathrm{d}x \tag{2.5.10}$$

Now comes the signal estimation error itself:

$$\begin{aligned} e_f &= E\left[\left(D^{-1}\int_{a-u}^{b-u} K(x+u)f(x)\ \mathrm{d}x - D^{-1}\int_a^b K(x)f(x)\ \mathrm{d}x - u\right)^2\right] \\ &= E\left[\left(D^{-1}\int_a^b (K(x+u)-K(x))f(x)\ \mathrm{d}x - u + \Delta_f\right)^2\right] \end{aligned} \tag{2.5.11}$$

where

$$\Delta_f = D^{-1}\int_{a-u}^{a} K(x+u)f(x)\ \mathrm{d}x - D^{-1}\int_{b-u}^{b} K(x+u)f(x)\ \mathrm{d}x \tag{2.5.12}$$

This can be similarly approximated as in the case of background error under the condition of $K(a) = 0$ and $K(b) = 0$ so $\Delta_f \approx 0$:

$$\begin{aligned} e_f &\approx E\left[\left(D^{-1}\int_a^b (K(x+u)-K(x))f(x)\ \mathrm{d}x - u\right)^2\right] \\ &= E\left[\left(D^{-1}\int_a^b (K(x+u)-K(x)-K'(x)u)f(x)\ \mathrm{d}x\right)^2\right] \end{aligned} \tag{2.5.13}$$

If $K'(x)$ is also differentiable,

$$\begin{aligned} e_f &\approx E\left[\left(D^{-1}\int_a^b \frac{1}{2}u^2 K''(x)f(x)\ \mathrm{d}x\right)^2\right] \\ &= \frac{1}{4}D^{-2}\left(\int_a^b K''(x)f(x)\ \mathrm{d}x\right)^2 E\left[u^2\right] \end{aligned} \tag{2.5.14}$$

where $K''(x) = \frac{\mathrm{d}}{\mathrm{d}x}K'(x)$

Ignoring the presumably small background error if there is no suitable model for the background, equations (2.5.14) and (2.5.10) can be used to optimize the kernel $K(x)$ being used. Note that D also depends on $K(x)$.

2.5.2 Roof Kernels

The estimates on estimation error terms are derived for general kernels in the previous section. However, it is hard to select a general optimal kernel based on these estimates directly since the domain is so big. Rather, we want to select the optimal function within a given class of functions. In this section, we choose the roof kernel as the class, because its first order derivative only changes at three points (two of them fixed at the boundary), which is the minimal. This provides great

potential for minimizing the estimation error because the error is exactly caused by varying first order derivative. In addition, as we can see later in the section, it only takes linear time to evaluate the error estimate for all roof peak positions.

A one-dimensional "roof" kernel on the interval $[a, b]$ with peak position c $(a < c < b)$ and height h is defined as follows:

$$R(x; c, h) = \begin{cases} \frac{h}{c-a}(x-a) & a \le x \le c \\ \frac{h}{b-c}(b-x) & c < x \le b \\ 0 & \text{otherwise} \end{cases} \tag{2.5.15}$$

Obviously all errors calculated from equations (2.5.14), (2.5.8) and (2.5.10) are independent of the choice of h,[7] so we may as well let $h = 1$ in equation (2.5.15). Note that the interval $[a, b]$ is the same one for the signal f, in other words, the roof kernel covers the entire length of the signal. Only c is the variable to be optimized on.

The derivative of the (normalized) roof kernel is even simpler:

$$R'(x; c) = \begin{cases} \frac{1}{c-a} & a < x < c \\ -\frac{1}{b-c} & c < x < b \\ 0 & x < a \text{ or } x > b \end{cases} \tag{2.5.16}$$

Note that $R'(x; c)$ is undefined on points a, c and b.

If roof kernels are to be used, the equation (2.5.3) is

$$D = \frac{1}{c-a}\int_a^c f(x)\,\mathrm{d}x - \frac{1}{b-c}\int_c^b f(x)\,\mathrm{d}x \tag{2.5.17}$$

In other words, D measures the difference between the averages of the two partitions of the signal.

The noise error term e_n is easy following equation (2.5.10):

$$\begin{aligned} e_n &= \sigma_n^2 D^{-2}\left(\int_a^c \frac{1}{(c-a)^2}(x-a)^2\,\mathrm{d}x + \int_c^b \frac{1}{(b-c)^2}(b-x)^2\,\mathrm{d}x\right) \\ &= \sigma_n^2 D^{-2}\left(\frac{1}{3}(c-a) + \frac{1}{3}(b-c)\right) \\ &= \frac{1}{3}\sigma_n^2 D^{-2}(b-a) \end{aligned} \tag{2.5.18}$$

To minimize e_n is to maximize $|D|$.

As $R''(x; c) = 0$ everywhere except undefined on points a, b and c, the signal error term e_f cannot be calculated simply from equation (2.5.14). However, it can still be easily derived directly from equation (2.5.11) as

$$e_f \approx \frac{1}{4}D^{-2}S^2E[u^4] \tag{2.5.19}$$

[7] More generally, multiplying the kernel by a constant factor has no effect on the estimator.

1. Let $F(x^0) = 0$, and for all $i = 1, \ldots, N_s$, $F(x^{(i)}) = F(x^{(i-1)}) + f(x^{(i)})$
2. Find the $\hat{c} \in x^{(i)}$ that minimizes $e_f + e_n$ as in equations (2.5.19) and (2.5.18), with D computed by equation (2.5.22).
3. The optimal roof kernel for the given signal $f(x)$ is $R(x; \hat{c})$ as in equation(2.5.15).

Figure 2.1: The Algorithm to Select Peak Position of 1-D Roof Kernel

where

$$S = \frac{f(c) - f(a)}{c - a} + \frac{f(c) - f(b)}{c - b} \tag{2.5.20}$$

The details of the deduction is in appendix A.5.

To minimize e_f is to minimize $|S/D|$.

Combining equations (2.5.19) and (2.5.18), and ignoring[8] e_b, we can search for the best c to minimize e given the foreground signal $f(x)$ and noise strength σ_n. Practically, the signal $f(x)$ is given as N samples $f(x_1), \ldots, f(x_N)$, so finding the best c is $O(N)$ even using brute-force search.

Some comments about D and S: if we define

$$F(x') = \int_a^{x'} f(x)\ \mathrm{d}x \tag{2.5.21}$$

another way to write equation (2.5.17) is

$$D = \frac{F(c) - F(a)}{c - a} - \frac{F(c) - F(a)}{b - c} \tag{2.5.22}$$

The resemblance between this and equation (2.5.20) for S is interesting.

Equations (2.5.19) and (2.5.18) give the foreground and noise error estimates. Ignoring the background error as usual, if both the noise strength and the distribution of the translation parameter are known, we can search for the best roof kernel that minimizes the total error. Limiting ourselves to kernels that cover the entire signal range, i.e. a and b are fixed, the only kernel parameter to choose is the location of the peak of the kernel, c. If both the signal and the kernel are represented in N_s discrete data points (pixels) $x^{(i)}$, $i = 1, \ldots, N_s$, a straightforward algorithm to find the optimal 1-D roof kernel is shown in figure 2.1. The cost of both time and memory for the algorithm is $\Theta(N_s)$.

2.5.3 "Zigzag" Kernels

Rearranging the equations in section 2.5.2, we have

$$\int_a^b R(x; c)(f(x + u) - f(x))\ \mathrm{d}x \approx Du - \frac{1}{2}Su^2 \tag{2.5.23}$$

[8] Actually, if we make the boundary of the roof kernel "inside" the boundary of the signal by enough margin, i.e. replace a and b in equation (2.5.15) for a^* and b^* that $a^* > a$ and $b^* < b$, the term e_b can be completely ignored given small u no matter what $b(x)$ is.

where D and S are given in equations (2.5.17) and (2.5.20).

Now if we have two roof kernels $R(x;c_1)$ and $R(x;c_2)$ with corresponding D_1, S_1 and D_2, S_2 such that

$$S_1 D_1 S_2 D_2 < 0 \tag{2.5.24}$$

we can combine the two kernels into one "zigzag" kernel:

$$Z(x;c_1,c_2) = S_2 R(x;c_1) - S_1 R(x;c_2) \tag{2.5.25}$$

so that

$$\int_a^b Z(x;c_1,c_2)(f(x+u) - f(x))\,\mathrm{d}x \approx (S_2 D_1 - S_1 D_2)u \tag{2.5.26}$$

The estimator for this zigzag kernel is accordingly

$$\hat{u} = \frac{1}{S_2 D_1 - S_1 D_2} \int_a^b Z(x;c_1,c_2)(g(x) - f(x))\,\mathrm{d}x \tag{2.5.27}$$

Obviously, it has a lower order of foreground error e_f than either of its component roof kernels. The approximation or omission for background error e_b is still valid as $Z(a) = Z(b) = 0$. In the terms of noise error e_n:

$$e_{n,Z} = \sigma_n^2 (S_2 D_1 - S_1 D_2)^{-2} \int_a^b (S_2 R(x;c_1) - S_1 R(x;c_1))^2\,\mathrm{d}x \tag{2.5.28}$$

A rough calculation shows that, under the condition of equation (2.5.24),

$$\begin{aligned} e_{n,Z} &\le \sigma_n^2 (S_2 D_1 - S_1 D_2)^{-2} \int_a^b 2(S_2^2 R^2(x;c_1) + S_1^2 R^2(x;c_1))\,\mathrm{d}x \\ &< S_2^{-2} D_1^{-2} \int_a^b 2 S_2^2 R^2(x;c_1)\,\mathrm{d}x + S_1^{-2} D_2^{-2} \int_a^b 2 S_1^2 R^2(x;c_2)\,\mathrm{d}x \\ &= 2e_{n,1} + 2e_{n,2} \end{aligned} \tag{2.5.29}$$

If an extra condition of $S_1 S_2 < 0$ is satisfied, since $R(x;c) \ge 0$

$$\begin{aligned} e_{n,Z} &\le \sigma_n^2 (S_2 D_1 - S_1 D_2)^{-2} \int_a^b (S_2^2 R^2(x;c_1) + S_1^2 R^2(x;c_1))\,\mathrm{d}x \\ &< S_2^{-2} D_1^{-2} \int_a^b S_2^2 R^2(x;c_1)\,\mathrm{d}x + S_1^{-2} D_2^{-2} \int_a^b S_1^2 R^2(x;c_2)\,\mathrm{d}x \\ &= e_{n,1} + e_{n,2} \end{aligned} \tag{2.5.30}$$

We can see the noise error is linearly bounded above.

For fixed a and b, the zigzag kernel has two free parameters c_1 and c_2, corresponding to the two peak positions of its component roof kernels. The algorithm to find their best values is similar to the one described in figure 2.1. However its time complexity is $\Theta(N_s^2)$.

2.5.4 Multiple Kernels

Still limiting ourselves in the one-parameter transformation ($m = 1$) case, we now analyze whether using multiple kernels ($m_K > 1$) can help us. Note that both $\mathbf{D}$ and $\mathbf{K}$ in equations (2.4.10) and (2.4.6) are column vectors. Although there can be many pseudo-inverses of a column vector $\mathbf{D} = [D_1, \ldots, D_{m_K}]^T$, a general one is

$$\mathbf{D}^- = \frac{\mathbf{D}^T}{\mathbf{D}^T\mathbf{D}} \tag{2.5.31}$$

so that equation (2.4.10) becomes

$$k(x) = \frac{\mathbf{D}^T\mathbf{K}(x)}{\mathbf{D}^T\mathbf{D}} \tag{2.5.32}$$

Each component kernel $K_i(x)$, if used individually, should produce $k_i(x) = D_i^{-1}K_i(x)$. That is,

$$k(x) = \frac{\sum_{i=1}^{m_K} D_i^2 k_i(x)}{\sum_{i=1}^{m_K} D_i^2} \tag{2.5.33}$$

We see that $k(x)$ is a weighted average of $k_i(x)$. Furthermore, the weights D_i^2 are (among other factors) inversely proportional to the errors caused by k_i, as in equations (2.5.14) and (2.5.10). So roughly speaking, the more accurate the individual kernel is, the more weight is assigned to it.

Now let us analyze the errors from $k(x)$ in detail. Ignoring e_b as usual, first e_f:

$$\begin{aligned}
e_f &= E\left[\left(\int_a^b k(x)(f(x-u) - f(x))\,\mathrm{d}x - u\right)^2\right] \\
&= E\left[\left(\sum_{i=1}^{m_K} D_i^2\right)^{-2}\left(\sum_{i=1}^{m_K} D_i\left(\int_a^b K_i(x)(f(x-u) - f(x))\,\mathrm{d}x - D_i u\right)\right)^2\right] \\
&\le E\left[\left(\sum_{i=1}^{m_K} D_i^2\right)^{-1}\sum_{i=1}^{m_K}\left(\int_a^b K_i(x)(f(x-u) - f(x))\,\mathrm{d}x - D_i u\right)^2\right] \\
&= \left(\sum_{i=1}^{m_K} D_i^2\right)^{-1}\sum_{i=1}^{m_K} D_i^2 e_{f,i}
\end{aligned} \tag{2.5.34}$$

So the foreground signal error e_f is bounded above by a weighted average of $e_{f,i}$ which are the foreground errors if the component kernels were used individually. This implies that the foreground error cannot be worse than that of the worst of the kernels.

Then e_n:

$$\begin{aligned} e_n &= \sigma_n^2 \left(\sum_{i=1}^{m_K} D_i^2 \right)^{-2} \int_a^b \left(\sum_{i=1}^{m_K} D_i K_i(x) \right)^2 \, \mathrm{d}x \\ &\leq \sigma_n^2 \left(\sum_{i=1}^{m_K} D_i^2 \right)^{-2} \int_a^b \left(\sum_{i=1}^{m_K} D_i^2 \right) \left(\sum_{i=1}^{m_K} K_i^2(x) \right) \, \mathrm{d}x \\ &= \sigma_n^2 \left(\sum_{i=1}^{m_K} D_i^2 \right)^{-1} \int_a^b \sum_{i=1}^{m_K} K_i^2(x) \, \mathrm{d}x \\ &= \left(\sum_{i=1}^{m_K} D_i^2 \right)^{-1} \sum_{i=1}^{m_K} D_i^2 e_{n,i} \end{aligned} \tag{2.5.35}$$

Same as the case of e_f, the noise error e_n is bounded above by a weighted average of e_{n_i} which are the noise errors if the component kernels were used individually. This implies that the noise error cannot be worse than that of the worst of the kernels, either.

If the noise errors of the individual kernels only depend on D in the form of $e_{n,i} = \sigma_n^2 D_i^{-2} C$ where C is a constant, as in the case of roof kernels, the bound in equation (2.5.35) can be rearranged into

$$\begin{aligned} e_n &\leq m_K \sigma_n^2 \left(\sum_{i=1}^{m_K} D_i^2 \right)^{-1} C \\ &< m_K \sigma_n^2 D_i^{-2} C \\ &= m_K e_{n,i} \end{aligned} \tag{2.5.36}$$

so that the noise error cannot be worse than m_K times of the noise error of the best of the kernels, either, in this case.

In another special case, if the kernels $K_i(x)$ are not overlapping with each other, i.e. for any x that $a \leq x \leq b$, there is at most one i that $K_i(x) \neq 0$, then we have

$$\begin{aligned} e_n &= \sigma_n^2 \left(\sum_{i=1}^{m_K} D_i^2 \right)^{-2} \int_a^b \left(\sum_{i=1}^{m_K} D_i K_i(x) \right)^2 \, \mathrm{d}x \\ &= \sigma_n^2 \left(\sum_{i=1}^{m_K} D_i^2 \right)^{-2} \int_a^b \sum_{i=1}^{m_K} D_i^2 K_i^2(x) \, \mathrm{d}x \\ &= \left(\sum_{i=1}^{m_K} D_i^2 \right)^{-2} \sum_{i=1}^{m_K} D_i^4 e_{n,i} \end{aligned} \tag{2.5.37}$$

If we further assume that all D_is and e_is are close in value, $e_n \approx e_{n,i}/m_K$, which matches our intuition.

2.5.5 Effectiveness of Kernel Selection

In the section 2.5.2 we discussed the estimation of two kinds of errors given the position of the roof kernel on the known signal and outlined an algorithm to select optimal roof kernels. In this

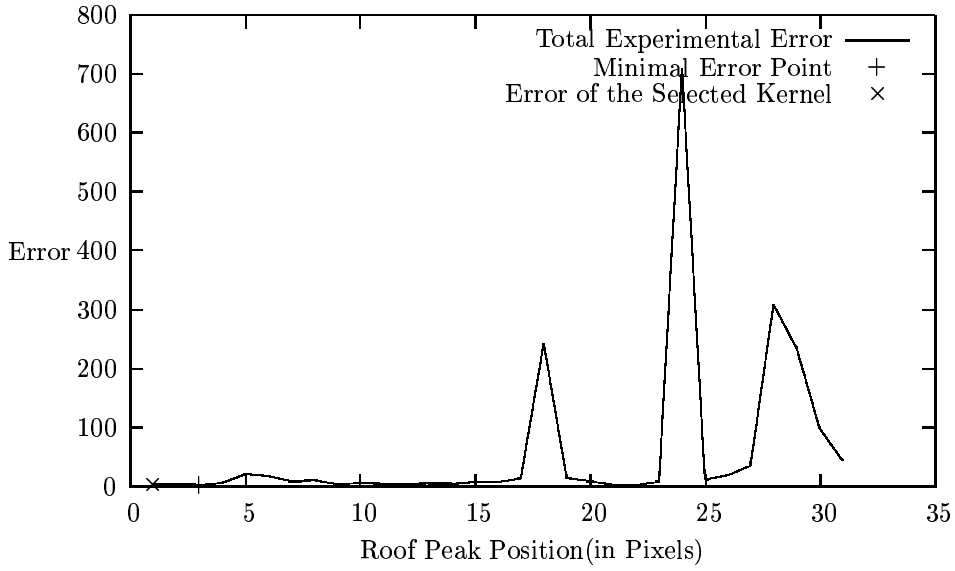


Figure 2.2: The graph of total (except background) estimation error against the peak position of the one-dimension roof kernel on a smoothed discrete random signal, with white Gaussian additive noise of 5 percent of the signal strength. Kernel width is 32 points and the translation parameter is uniformly distributed in $[-5, 5]$. All quantities are in pixels.

subsection we verify the effectiveness of the kernel selection algorithm by simulation experiments.

Figure (2.2) gives a typical graph of total (except background) error against the peak position of the roof kernel on a smoothed random signal, as well as the result from the kernel selected by the algorithm in figure 2.1. As we can see, there are several 'peaks' in the graph, corresponding to the points of c where D is very small.

In the simulation, the distribution of u is fixed as $p(u)$. Every signal $f^{(i)}$, $i = 1, \ldots, N$ is a smoothed (colored) Gaussian random process. For each $f^{(i)}$, an estimator $\hat{u}$ is generated from the roof kernel, and the estimation error is defined as[9]

$$e^{(i)} = \int p(u)(\hat{u}(u) - u)^2 \, \mathrm{d}u \tag{2.5.38}$$

The average error and the standard deviation of the error are defined as

$$\bar{e} = \frac{1}{N} \sum_i e^{(i)} \tag{2.5.39a}$$

$$s_e = \frac{1}{N} \sum_i \left(e^{(i)} - \bar{e} \right)^2 \tag{2.5.39b}$$

Table 2.5.5 gives the average error $\bar{e}$ and the standard deviation s_e of the error of roof kernels in the simulation. In the table, "Selected Kernels" are kernels selected by the algorithm in section 2.5.2; "Best Kernels" are the best performed roof kernels that have minimum error as

[9] Obviously, discrete samples are used to compute the integral in the experiment.

Type of Roof Kernel	Average Error $\bar{e}$ (in points)	Standard Deviation s_e (in points)
Selected Kernels	2.48	1.56
Best Kernels	1.18	0.69
Multiple Kernels	3.92	1.47
Average Kernels	350.68	7843.60
SSD (not roof kernels)	3.16	0.07

Table 2.1: The average error and standard deviation of one-dimension roof kernels over 1000 smoothed discrete random signals with white Gaussian additive noise of 5 percent of the signal strength. Kernel width is 32 and the translation parameter is uniformly distributed in $[-5, 5]$. Note that the "SSD" entry is the result of the SSD method, which is not really a roof kernel.

in equation (2.5.38) in the simulation; The results of "Multiple Kernels" as in section (2.5.4) are computed from weighted error of average kernels as in equations (2.5.34) and (2.5.35). The result of the SSD method in section 2.3 is also listed for comparison purpose.

From the table, we can see that the selected kernels perform statistically better then SSD. They are still not as good as the best kernels since the equations (2.5.19) and (2.5.18) the algorithm relies on are only approximations. Multiple kernels are not performing very well as they are usually worse than the best of their composite kernels.

2.6 More General Kernels

It is possible to analyze kernels for more general transformation by the similar method used in the previous section, however, for certain types of common transformation, we can reduce to 1-D translation thus reuse the existing result.

2.6.1 Two-dimensional Translation

The selection method for roof kernels can be used to track translation of 2-D images:

$$g(x, y) = f(x - u, y - v) + b(x - u, y - v) + n(x, y) \tag{2.6.1}$$

We shall devise two 2-D kernels, K_x and K_y, each for estimating one translation parameter u and v, respectively:

$$\hat{x} = D_x^{-1} \int_{a_y}^{b_y} \int_{a_x}^{b_x} K_x(x, y)[g(x, y) - f(x, y)] \, \mathrm{d}x\mathrm{d}y \tag{2.6.2a}$$

$$\hat{y} = D_y^{-1} \int_{a_y}^{b_y} \int_{a_x}^{b_x} K_y(x, y)[g(x, y) - f(x, y)] \, \mathrm{d}x\mathrm{d}y \tag{2.6.2b}$$

where

$$D_x = \int_{a_y}^{b_y} \int_{a_x}^{b_x} \frac{\partial K_x}{\partial x}(x,y) f(x,y) \, \mathrm{d}x\mathrm{d}y \tag{2.6.3a}$$

$$D_y = \int_{a_y}^{b_y} \int_{a_x}^{b_x} \frac{\partial K_y}{\partial y}(x,y) f(x,y) \, \mathrm{d}x\mathrm{d}y \tag{2.6.3b}$$

with the requirement that

$$0 = \int_{a_y}^{b_y} \int_{a_x}^{b_x} \frac{\partial K_x}{\partial y}(x,y) f(x,y) \, \mathrm{d}x\mathrm{d}y \tag{2.6.4a}$$

$$0 = \int_{a_y}^{b_y} \int_{a_x}^{b_x} \frac{\partial K_y}{\partial x}(x,y) f(x,y) \, \mathrm{d}x\mathrm{d}y \tag{2.6.4b}$$

The above can be easily deduced similarly as in the 1-D case by first-order approximation:

$$\begin{aligned}
&\int_{a_y}^{b_y} \int_{a_x}^{b_x} K_x(x,y) f(x-u, y-v) \, \mathrm{d}x\mathrm{d}y \\
\approx &\int_{a_y}^{b_y} \int_{a_x}^{b_x} K_x(x+u, y+v) f(x,y) \, \mathrm{d}x\mathrm{d}y \\
\approx &\int_{a_y}^{b_y} \int_{a_x}^{b_x} K_x(x,y) f(x,y) \, \mathrm{d}x\mathrm{d}y \\
&+u \int_{a_y}^{b_y} \int_{a_x}^{b_x} \frac{\partial K_x}{\partial x}(x,y) f(x,y) \, \mathrm{d}x\mathrm{d}y \\
&+v \int_{a_y}^{b_y} \int_{a_x}^{b_x} \frac{\partial K_x}{\partial y}(x,y) f(x,y) \, \mathrm{d}x\mathrm{d}y
\end{aligned} \tag{2.6.5}$$

and the same goes for K_y.

To apply our method of selecting 1-D roof kernels, each 2-D kernel K is composed by two separate 1-D roof kernels R (or zigzag kernels), one on each direction:

$$K_x(x,y) = R_{xx}(x) R_{xy}(y) \tag{2.6.6a}$$

$$K_y(x,y) = R_{yx}(x) R_{yy}(y) \tag{2.6.6b}$$

Ignoring the interaction between x- and y-translation, which is a second order error term and harder to estimate, the criterion of best $K_x(x)$ that also satisfies the constraint of equation (2.6.4a) can be separated into:

- minimize the error from estimator

$$\hat{x} = D_x^{-1} \int_{a_x}^{b_x} R_{xx}(x) f_x(x) \, \mathrm{d}x \tag{2.6.7}$$

  where

$$D_x = \int_{a_x}^{b_x} R'_{xx}(x) f_x(x) \, \mathrm{d}x \tag{2.6.8}$$

  and

$$f_x(x) = \int_{a_y}^{b_y} R_{xy}(y) f(x,y) \, \mathrm{d}y \tag{2.6.9}$$

1. Initialize (u_0, v_0) to a reasonable value (e.g. the center of the image region).

2. For $t = 1, \ldots, T$, where T is the maximum number of iterations, do

   (a) Let u_t be the result of the optimal 1-D kernel selection algorithm in figure 2.1 on the signal $f_x(x) = \int_{a_y}^{b_y} R_{xy}(y; v_{t-1}) f(x, y)\ \mathrm{d}y$.

   (b) Exit the loop if $\exists_{0<t'<t}$ that $u_{t'} = u(t)$

   (c) Let v_t be the result of a modified version of the optimal 1-D kernel selection algorithm in figure 2.1 on the signal $f_y(y) = \int_{a_x}^{b_x} R_{xy}(x; u_t) f(x, y)\ \mathrm{d}x$. The objective to minimize in the algorithm is modified to be D as in equation (2.5.17) instead of $e_f + e_n$.

   (d) Exit the loop if $\exists_{0<t'<t}$ that $v_{t'} = v(t)$

3. The optimal 2-D kernel for x-translation is $K_x(x, y) = R_{xx}(x; u_t) R_{xy}(y_t)$.

Figure 2.3: The Algorithm to Construct 2-D Roof Kernel for X-translation

- minimize D_y which is

$$D_y = \int_{a_y}^{b_y} R'_{xy}(y) f_x(y)\ \mathrm{d}y \tag{2.6.10}$$

  where

$$f_y(y) = \int_{a_x}^{b_x} R_{xx}(x) f(x, y)\ \mathrm{d}x \tag{2.6.11}$$

However, term (2.6.9) contains $R_{xy}(y)$ while term (2.6.11) contains $R_{xx}(x)$, so they can't be directly computed at once. Nevertheless, they can be computed iteratively by fixing one while searching for the other. In practice, it usually only takes a few iterations before they converge. The algorithm to find the optimal $K_x(x, y)$ can be described in figure 2.3. The same algorithm except x and y swapped is to be used on selection of K_y.

To verify the effctiveness of the 2-d kernels constructed by the algorithm, experiment is conducted using real image and simulated translation. Figure (2.4) shows the image used for tracking experiments. Figure (2.5) shows the magnitude of estimation error over the two translation parameters on simulated translations for both composed roof kernel and SSD. We can see that the error of SSD increases more quickly than that of the composed roof kernel for bigger translations.

Figure (2.6) from the same experiment further illustrates the difference of the convergence ranges between the two methods.

2.6.2 One-Dimensional Scaling

In this section, we focus on the kernels that can estimate scaling. Denote s as the scaling parameter, now we assume

$$T(x; s) = \frac{x}{1 + s} \tag{2.6.12}$$

Figure 2.4: The image used for tracking 2-D translation using composed roof kernels, boxed area (60 by 60 pixels) is the template.

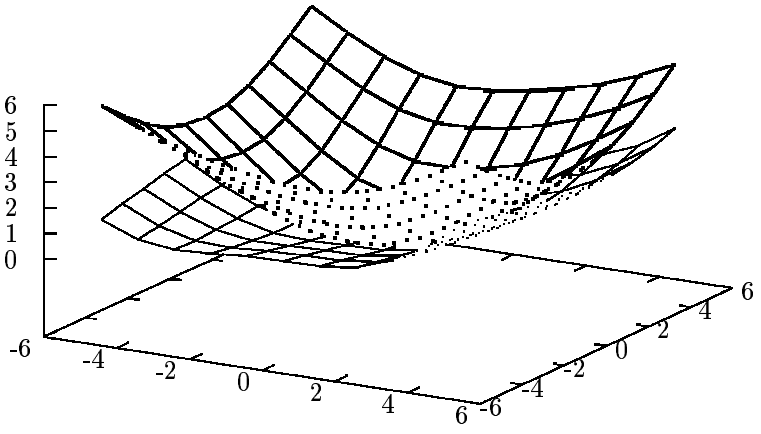


Figure 2.5: The magnitude ($\sqrt{(\hat{x}-x)^2+(\hat{y}-y)^2}$) of estimation error over the two translation parameters x and y on simulated translations. The upper mesh is from the SSD method and the bottom mesh is from the composed roof kernel. All quantities are in pixels.

then

$$T^{-1}(x;s) = (1+s)x \tag{2.6.13}$$

Substituting into equations (2.4.9) and (2.4.13) and letting $[a,b]$ be the interval, the estimate is

$$\hat{s} = D^{-1}\int_a^b K(x)(g(x)-f(x))\,\mathrm{d}x \tag{2.6.14}$$

where $g(x)$ is the input scaled signal and

$$D = \int_a^b (K(x)+xK'(x))f(x)\,\mathrm{d}x \tag{2.6.15}$$

Here $K'(x)$ stands for $\frac{\mathrm{d}}{\mathrm{d}x}K(x)$ as usual.

Constructing the estimator from any given kernel is easy as above, however, the question is to find an optimal kernel for the purpose. Since optimal translation kernels have already been discussed in previous sections, especially optimal roof kernels in section 2.5.2, we can simply exploit existing conclusions under a different coordinate system, where scaling now appears as translation.

Define

$$f_e(x) = kae^{k(x-a)}f(ae^{k(x-a)}) \tag{2.6.16a}$$

$$K_e(x) = K(ae^{k(x-a)}) \tag{2.6.16b}$$

It is easy to verify for $k=(b-a)^{-1}$ that both $f_e(x)$ and $K_e(x)$ are well defined on interval $[a,b]$ if

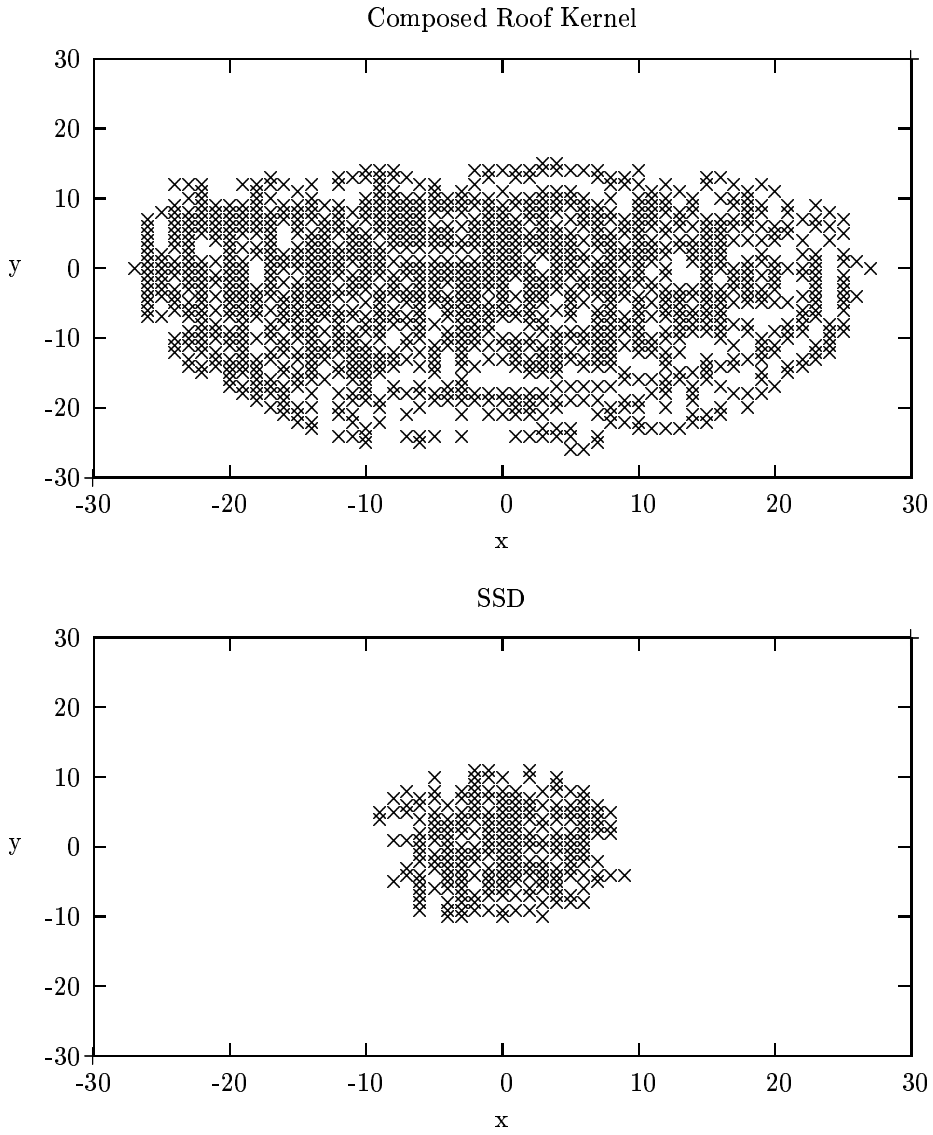


Figure 2.6: Each cross is marked for being able to estimate this pair (x,y) of 2D-translation parameters within error ($\sqrt{(\hat{x}-x)^2+(\hat{y}-y)^2}$) of 0.5 after 3 iterations. All quantities are in pixels.

1. Calculate $f_e(x)$ from known $f(x)$ by equation 2.6.16a.
2. Compute optimal translation kernel $K_e(x)$ by the algorithm in figure 2.1.
3. Calculate $K(x)$ from the optimal $K_e(x)$ by equation 2.6.16b.
4. Calculate D by equation (2.6.15).
5. Construct the scaling kernel $k(x) = D^{-1}K(x)$ (as in equation (2.4.10)).

Figure 2.7: The Algorithm to Construct 1-D Scaling Kernel

both $f(x)$ and $K(x)$ are well defined on $[a, b]$. Further, the following holds for small s:

$$\begin{aligned}\int_a^b K(x)f(\frac{x}{1+s})\,\mathrm{d}x &= \int_a^b kae^{k(x-a)}K(e^{k(x-a)})f(e^{k(x-a)-\ln(1+s)})\,\mathrm{d}x \\ &\approx \int_a^b kae^{k(x-a)}K(e^{k(x-a)})f(e^{k(x-a)-s})\,\mathrm{d}x \\ &= \int_a^b K_e(x)f_e(x-k^{-1}s)\,\mathrm{d}x\end{aligned} \tag{2.6.17}$$

Now right-hand side is exactly in the form of translation, and the new parameter for the translation $k^{-1}s$ is proportional to the original scaling parameter s. Gaussian additive noise is still Gaussian additive noise under the new coordinates, thus an optimal $K_e(x)$ for it can be found by one of the methods described in previous sections. More specifically, the algorithm for constructing a scaling kernel is shown in figure 2.7.

This coordination transform method requires that $0 \notin [a, b]$. However, it is the nature of scaling that there is little change for the points around $x = 0$, thus their values are not going to contribute to the estimation anyway.

Figure 2.8 compares the estimation errors of the one-dimension transformed roof kernel for scaling as described above and the estimation errors of the SSD method in a simulated experiment. The signals are randomly generated and smoothed. As it shows, the transformed roof kernel does not have an advantage on accuracy against SSD for very small s, probably because it is indirectly constructed, but its error does not rise as quickly for relatively greater s, fulfilling its purpose.

2.6.3 Two-Dimensional Scaling and Rotation

For a two-dimensional image, we need to deal with rotation and scaling simultaneously. The transformation $T(\mathbf{x}; \tau)$ is [10]

$$T_x(x, y; \psi, s) = \frac{x\cos\psi + y\sin\psi}{1+s} \tag{2.6.18a}$$

$$T_y(x, y; \psi, s) = \frac{y\cos\psi - x\sin\psi}{1+s} \tag{2.6.18b}$$

[10] Please be aware that $\mathbf{x}$ is now the vector (x, y) and τ is (ψ, s).

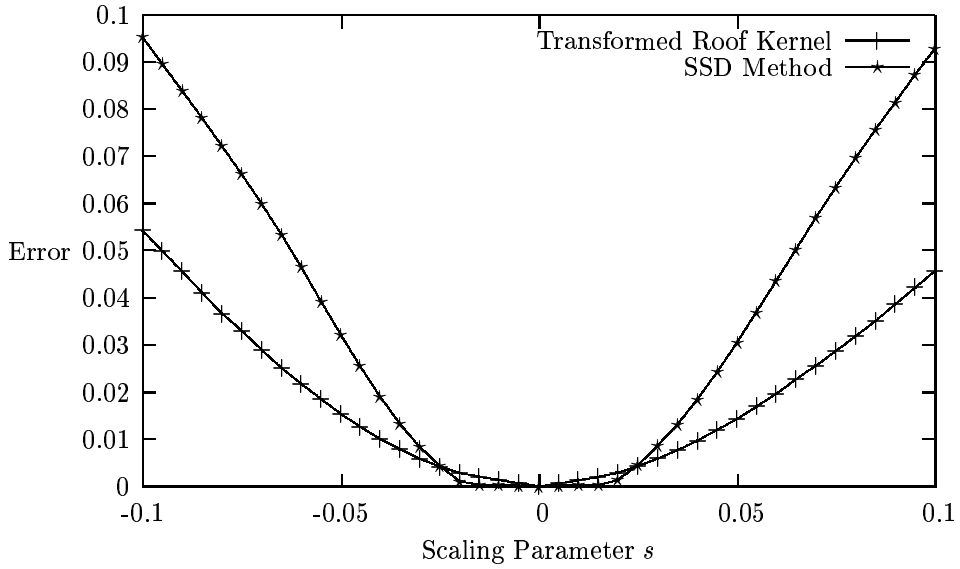


Figure 2.8: The graph of estimation errors of the one-dimension transformed roof kernel for scaling and estimation errors of the SSD method over the scaling parameter on a smoothed discrete random signal. Kernel width is 64 points. The errors are averages of absolute errors over 1000 runs. All quantities are unitless.

Therefore

$$T_x^{-1}(x, y; \psi, s) = (1 + s)(x \cos\psi - y \sin\psi) \tag{2.6.19a}$$

$$T_y^{-1}(x, y; \psi, s) = (1 + s)(y \cos\psi + x \sin\psi) \tag{2.6.19b}$$

so

$$\frac{\mathbf{T}^{-1}(\mathbf{x}; \tau)}{\partial \mathbf{x}} = \begin{pmatrix} (1+s)\cos\psi & (1+s)\sin\psi \\ -(1+s)\sin\psi & (1+s)\cos\psi \end{pmatrix} \tag{2.6.20a}$$

$$\frac{\mathbf{T}^{-1}(\mathbf{x}; \tau)}{\partial \tau} = \begin{pmatrix} x\cos\psi - y\sin\psi & y\cos\psi + x\sin\psi \\ (1+s)(-y\cos\psi + x\sin\psi) & (1+s)(x\cos\psi - y\sin\psi) \end{pmatrix} \tag{2.6.20b}$$

Substituting into equation (2.4.13) we have

$$D_\psi = \int \left[xK'_y(x, y) - yK'_x(x, y) \right] f(x, y)\, \mathrm{d}x\mathrm{d}y \tag{2.6.21a}$$

$$D_s = \int \left[K(x, y) + xK'_x(x, y) + yK'_y(x, y) \right] f(x, y)\, \mathrm{d}x\mathrm{d}y \tag{2.6.21b}$$

where

$$K'_x(x, y) = \frac{\mathrm{d}K(x, y)}{\mathrm{d}x} \tag{2.6.22a}$$

$$K'_y(x, y) = \frac{\mathrm{d}K(x, y)}{\mathrm{d}y} \tag{2.6.22b}$$

so for two kernels $K^{(1)}(x,y)$ and $K^{(2)}(x,y)$ we can compute their corresponding $(D_\psi^{(1)}, D_s^{(1)})$ and $(D_\psi^{(2)}, D_s^{(2)})$. The estimators can be derived from equation (2.4.9):

$$\hat{\psi} = h^{-1} \int \left[D_s^{(2)} K^{(1)}(x,y) - D_s^{(1)} K^{(2)}(x,y) \right] (g(x,y) - f(x,y)) \, \mathrm{d}x\mathrm{d}y \tag{2.6.23a}$$

$$\hat{s} = h^{-1} \int \left[D_\psi^{(1)} K^{(2)}(x,y) - D_\psi^{(2)} K^{(1)}(x,y) \right] (g(x,y) - f(x,y)) \, \mathrm{d}x\mathrm{d}y \tag{2.6.23b}$$

where

$$h = D_\psi^{(1)} D_s^{(2)} + D_\psi^{(2)} D_s^{(1)} \tag{2.6.24}$$

Again, we are facing the problem of choosing kernels $K^{(1)}$ and $K^{(2)}$, and we shall reduce it into translation by changing variables x, y to r, θ:

$$x = a e^{k(r-a)} \cos\theta \tag{2.6.25a}$$

$$y = a e^{k(r-a)} \sin\theta \tag{2.6.25b}$$

and accordingly map the kernel and the signal onto the new coordinates:

$$f_e(r,\theta) = kae^{k(r-a)} f(ae^{k(r-a)} \cos\theta, ae^{k(r-a)} \sin\theta) \tag{2.6.26a}$$

$$K_e(r,\theta) = K(ae^{k(r-a)} \cos\theta, ae^{k(r-a)} \sin\theta) \tag{2.6.26b}$$

$$\tag{2.6.26c}$$

Now since

$$\begin{aligned} & kae^{k(r-a)} f(T_x(x,y;\psi,s), T_y(x,y;\psi,s)) \\ =& kae^{k(r-a)} f(ae^{k(r-a)-\log(1+s)} \cos(\theta-\psi), ae^{k(r-a)-\log(1+s)} \sin(\theta-\psi)) \\ =& f_e(r - k^{-1}\log(1+s), \theta - \psi) \\ \approx& f_e(r - k^{-1}s, \theta - \psi) \end{aligned} \tag{2.6.27}$$

we have

$$\begin{aligned} & \int K(x,y) f(T_x(x,y;\psi,s), T_y(x,y;\psi,s)) \, \mathrm{d}x\mathrm{d}y \\ =& \int kae^{k(r-a)} K(x,y) f(T_x(x,y;\psi,s), T_y(x,y;\psi,s)) \, \mathrm{d}r\mathrm{d}\theta \\ \approx& \int K_e(r,\theta) f_e(r - k^{-1}s, \theta - \psi) \, \mathrm{d}r\mathrm{d}\theta \end{aligned} \tag{2.6.28}$$

Now right-hand side is again exactly in the form of 2-D translation. An optimal $K_e(x,y)$ for it can be found by the algorithm described in figure 2.3. Similar to the 1-D scaling case, the algorithm for constructing the 2-D scaling and rotation kernel for equation (2.6.23) is shown in figure 2.9.

Figure 2.10 shows the parameter estimate of the transformed roof kernel for rotation as described above and the estimate of the SSD method in a simulated experiment. The image rotated is in figure 2.4, in which the boxed area is the template. As the graph shows, for greater rotation parameter ψ, the transformed roof kernel gives more accurate result than the SSD method as expected.

1. Calculate $f_e(r,\theta)$ from known $f(x,y)$ by equation 2.6.26a.
2. Compute optimal translation kernel $K_e^{(r)}(r,\theta)$ and $K_e^{(\theta)}(r,\theta)$ respectively for r and θ translation, by the algorithm in section 2.6.1.
3. Calculate $K^{(r)}(x,y)$ and $K^{(\theta)}(x,y)$ from the optimal $K_e^{(r)}(r,\theta)$ and $K_e^{(\theta)}(r,\theta)$ by equation 2.6.26b.
4. Calculate elements of D by equation (2.6.21).
5. Construct the kernels by equation (2.4.10).

Figure 2.9: The Algorithm to Construct 2-D Scling and Rotation Kernels

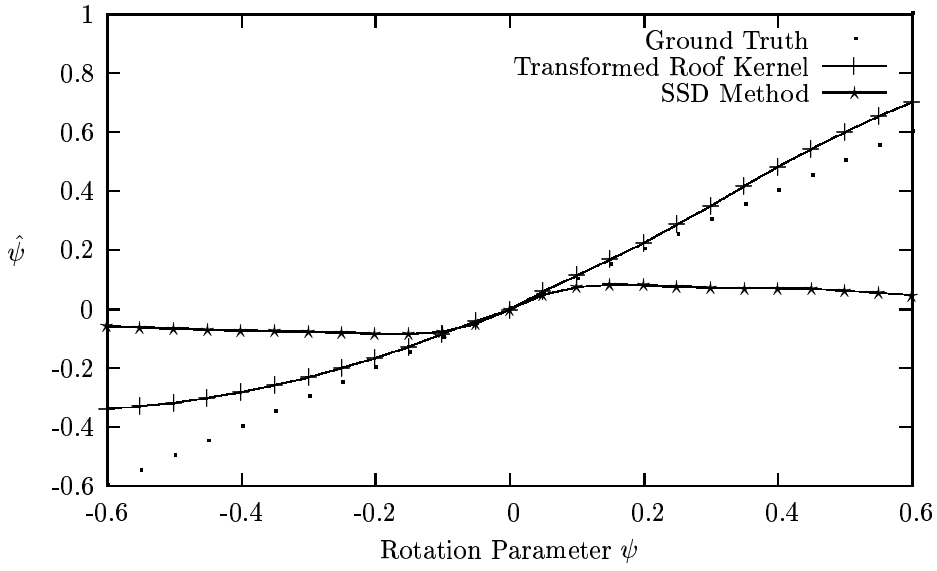


Figure 2.10: The graph of both the parameter estimate from the transformed roof kernel for rotation and the estimate from the SSD method over the rotation parameter on the image in figure 2.4. Dots are ground truth. All quantities are in radians (0.6 rad = 34.4°).

Chapter 3

The Tracking Framework

3.1 Rationale

The aim of component-based tracking is to provide a general framework to utilize all types of information from

- all kinds of image-level detectors,
- internal constraints of visual or physical components of the target,
- prior knowledge of the system dynamics, and
- the observation mode that relates the states of the target to visual observations.

We base this framework on the probabilistic graphical models not only because they provide a means to fuse the above sources of information in a mathematically unified form, but also because they are a natural and intuitive way to present this information.

We use undirected graphs in our framework, for the following reasons:

- The constraint between different parts of the target is more naturally described by an undirected edge than a directed one.
- Although a directed edge is the natural choice for an observation mode between a state and an observation, it is equivalent to an undirected edge if the observation only depends on this particular state.
- A directed graph can be always converted to an undirected one by creating new nodes that are collections of existing nodes.

Furthermore, Gaussian mixture models are used as the basis of our computation for their nice properties of easy manipulation and suitability to model multi-mode probability densities.

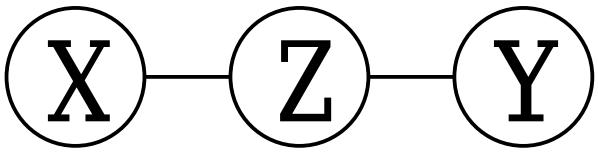


Figure 3.1: A simple graph example. In the graph, random variables X and Y are generally not independent; however, since they are only connected by random variable Z, X and Y are independent given Z.

3.2 The Graphical Model

In this framework, an undirected graph is used to represent the relational structure for all information sources of the tracking process. The graph G consists of a set of vertexes or nodes V and a set of undirected edges E. Vertices are continuous random variables X_i, representing tracking components. The entire graph represents the joint probability distribution[1] of all variables, in the sense that its structure matches the probabilistic dependencies between node variables. This can be roughly described as independence between variables given a subset of variables that are on every path of the original two. For example, in the simple graph illustrated in figure 3.1, random variables X and Y are generally not independent; however, since they are only connected by random variable Z, X and Y are independent given Z.

For a loopless graph, if we associate each edge with a local potential function ψ of the two variables that the edge connects, then the joint probability density function of all variables can be dissected into a product of these potential function and a constant normalization factor called the "partition function". Returning to the example in figure 3.1:

$$p_{xyz}(x,y,z) = Z_{xyz}^{-1}\psi_{xz}(x,z)\psi_{yz}(y,z) \tag{3.2.1}$$

where

$$Z_{xyz} = \int_z \int_y \int_x \psi_{xz}(x,z)\psi_{yz}(y,z)\ \mathrm{d}x\mathrm{d}y\mathrm{d}z \tag{3.2.2}$$

For a graph with loops, the joint probability density function is still determined by local potential functions on all cliques, but no longer as a straightforward product of them. Nevertheless, we[2] still treat it as if it were loopless and use the same message-passing algorithm presented in chapter 4 for inference.

From the joint probability density function of all variables, the marginal probability density function of one variable can be easily deduced by integrating out all other variables. For example,

[1] Strictly speaking, all the probability functions here are the posterior probabilities given the observation. This "given clause" is always omitted to simplify expression.

[2] In fact, almost everyone does.

if x is our variable of interest,

$$p_x(x) = \int_z \int_y p_{xyz}(x, y, z) \, \mathrm{d}y\mathrm{d}z \tag{3.2.3}$$

One of the simplest estimators of x would be its expectation

$$\hat{x} = E[x] = \int_x x p_z(x) \, \mathrm{d}x \tag{3.2.4}$$

3.3 Message-passing for Inference

One of the algorithms to infer the marginal probabilistic distribution of any component is called message-passing. Since the marginal probability of one variable is the integral of the joint probability density function over the rest of variables in the graph, and in the loopless graph the joint probability density is proportional to the product of local potentials, the marginal probability is computed from the multiplication and integration of local potentials. These operations can be done in a stepwise and localized manner by executing one or more iterations of passing "messages". Messages are partial results from the corresponding parts of the graph. They originate from "leaf" nodes, which have only one edge associated with; they are processed on their way to the destination node. Continuing with the example of the simple graph of figure 3.1: the message from Y to Z would be simply $\psi_{yz}(y, z)$; Z receives this message and integrates it over y, then multiplies it by $\psi_{xy}(x, y)$, so the message from Z to X is $\psi_{xy}(x, y) \int_y \psi_{yz}(y, z) \, \mathrm{d}y$; finally X receives this message and integrates it over z, and finally normalizes to get its marginal distribution.

Algorithmically, message-passing can be described as follows: the message between node i and node j can be written as

$$m_{ij}(x_j) = \int \psi_{ij}(x_i, x_j)\psi_i(x_i) \prod_{k \in N(i) \setminus j} m_{ki}(x_i) \, \mathrm{d}x_i \tag{3.3.1}$$

where $\psi_i(x_i)$ is the node potential that only applies to leaf nodes in our framework and the $N(i)$ are the neighbor nodes of node i.

The marginal probability of a node is proportional to the product of all its incoming messages:

$$p(x_i) \propto \prod_{k \in N(i)} m_{ki}(x_i) \tag{3.3.2}$$

A loopless graph is a tree, so the algorithm can run in one serial pass, provided that messages are passed in the correct order. Let the node being estimated be the root of the tree. Then messages must originate from leaf nodes and be passed all the way up to the root. However, for a general graph the algorithm would have to run on all nodes iteratively. Generally the convergence is not guaranteed in theory but observed in practice. However, if everything is Gaussian, the convergence is proven[55].

Type of Visual Clue	Possible States	More Possible States
Corner Point	Position	
Edge	Position and Orientation	Length
Homogeneous Region (Blobs)	Position	Size
Heterogeneous Region	Transformation Parameters	

Table 3.1: Examples of Types of Observation Components

3.4 Component-based Tracking

Everything discussed so far in the previous sections of this chapter is existing work upon which the proposed component-based tracking framework is based. In our framework, nodes are thought of as components and edges as constraints. Together they intuitively and mathematically model the tracking target.

3.4.1 Components

Components are classified into three classes: observation (visual clues), history (historical information), and parts (target parts). Observation and history components are leaf components while part components are not.

Observation components directly operate on the input images. They have node potentials computed from the underlying image-processing. These potentials are likelihood probabilities based on the image. Different kinds of observations rely on different kinds of visual clues and image models, they all generate a function of the state vector from the image. As we shall see later in section 3.5, the function will be in form of sum of Gaussians. In addition, as we shall see later in section 3.7, it can also carry an optional term indicating the reliability of the visual detector itself. Table 3.4.1 lists a few examples of the observations. The individual tracking methods of heterogeneous regions discussed in chapter 2 can be used to generate the node potential functions, which are discussed in detail later in section 3.4.1.

History components are actually a simplification on the temporal structure of the graph under the following two assumptions:

1. Only observations on time t' where $t' \leq t$ are used for estimation of state at time t. In other words, only previous and current observations are used.

2. Time is discrete, and the process is first order Markovian, which means that the conditional probability of the state at time t given states at all other times is same as that given just the state at $t-1$.[3]

[3]Strictly speaking, since we are using undirected graph (Markovian field), the conditional probability of the state at time t given states at all other times is same as that given state at $t-1$ and $t+1$. However, this makes no actual difference since future observations are not used anyway.

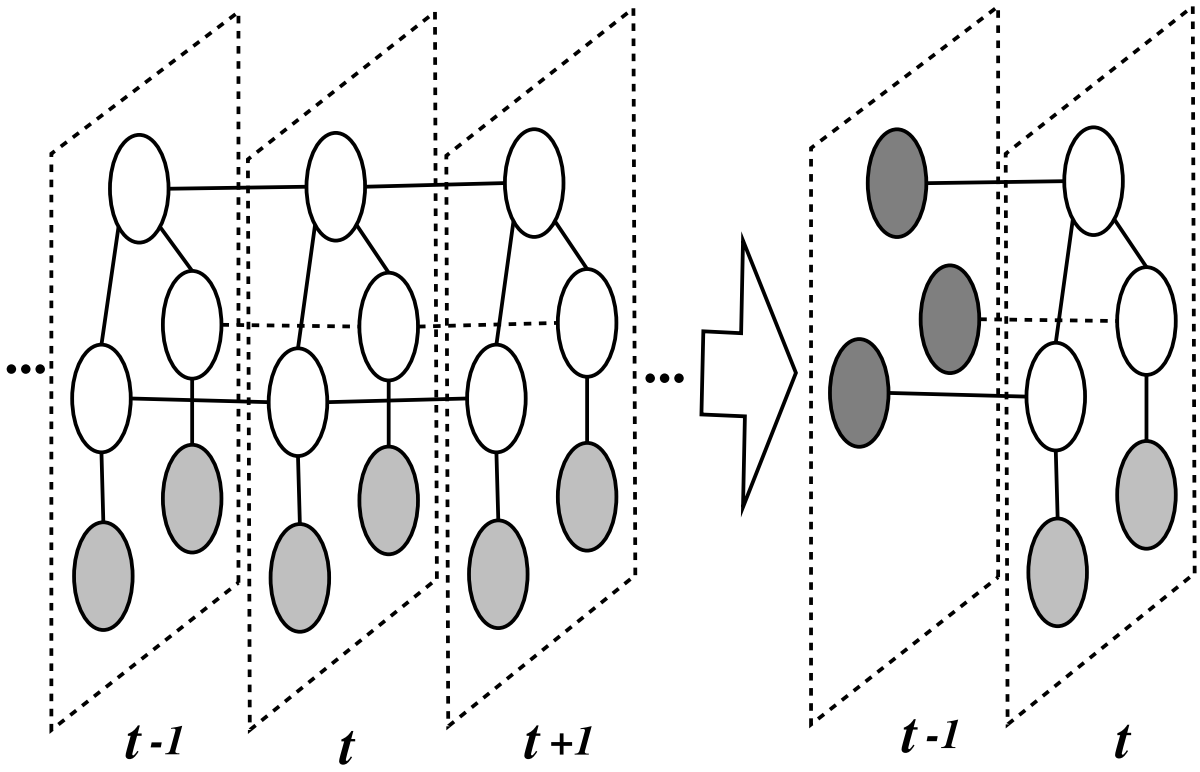


Figure 3.2: History components are actually a simplification of the temporal structure of the graph under certain assumptions. Note that white circles denote part component, gray circles denote observation components and dark circles denote history components. This notation is used for all figures in the rest of the chapter.

Type of Constraint	Possible Potential Expression
Overlapping	$\exp\left\{-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^T A(\mathbf{x}_i - \mathbf{x}_j)\right\}$
Linear	$\exp\left\{-\frac{1}{2}(A\mathbf{x}_i + B\mathbf{x}_j + \mathbf{u})^T (A\mathbf{x}_i + B\mathbf{x}_j + \mathbf{u})\right\}$
Distance	$\exp\left\{-\frac{1}{2}\left(\sqrt{(A\mathbf{x}_i - B\mathbf{x}_j)^T(A\mathbf{x}_i - B\mathbf{x}_j)} - d\right)^2\right\}$

Table 3.2: Examples of Types of Observation Components

Figure 3.2 shows this simplification on the temporal structure of the graph. The right graph is equivalent to the left one given the marginal probability of nodes in time $t-1$ has already been computed as of in time $t-1$ when computing the marginal probability of nodes in t. The node potential of the history component simply takes the marginal probability of the corresponding part component of the previous time-step.

Part components represent the states of visually identifiable parts of the tracking target or that of the whole object itself. They collectively model the conceptual structure of the tracking target. Usually the result of the tracking is a state estimate of one or a few of the part components.

3.4.2 Constraints

The concept of constraints generalizes three types of knowledge about the target. They are

- the internal constraint of the target parts, as an edge between two part components;
- the prior knowledge of the system dynamics, as an edge between a part component and a history component; and
- the observation model, as an edge between a part component and an observation component.

The constraints between components are described in the form of the potential function associated with the edge connecting the components. The function takes variables of states of both components. The value of the function is the relative probability that these two states meet the constraints, without regard to other considerations. Thus, all constraints are inherently "soft", meaning that violation is only penalized, but not totally prohibited. They could be made "harder" by increasing the penalties.

Table 3.4.2 lists a few examples of constraints classified in mathematical form.

3.5 Gaussian Mixtures

Equation (3.3.1) and (3.3.2) are in mathematical terms. The functions involved must be represented in some way before meaningful computation can be performed. If the variables were discrete and took only a finite number of possible values, their functions could be faithfully

represented as vectors or tables. However, for variables taking continuous values in our framework, absent a totally symbolic computation which is not usually feasible, some kind of approximation must be used.

One possible way is to represent the function by a collection of samples, as in particle filtering, however, we are looking for a more effective representation in the sense of fewer numbers are used to describe the distribution. In our framework, we approximate all density and potential functions involved as multivariate Gaussians or weighted sums of multivariate Gaussians (also known as Mixtures of Gaussians, hereby referred as MoGs). They can be computed effectively and exactly, and the results would be multivariate Gaussians or MoGs as well. Furthermore, this representation has an intuitive meaning: each of the single Gaussians in an MoG can be thought of as a weighted hypothesis of the state of the corresponding component.

If all functions are approximated as MoGs, the rules for their multiplication and integration must be established. Letting $\mathbf{x}$ denote $(x_1, \ldots, x_{|V|})^T$, an MoG function can be written as:[4]

$$f(\mathbf{x}) = \sum_{i=1}^{n} w_i \exp\left\{-\frac{1}{2}(\mathbf{x}-\mathbf{u}_i)^T A_i (\mathbf{x}-\mathbf{u}_i)\right\} \tag{3.5.1}$$

where w_i, u_i and A_i are weight, center and inverse of the covariance matrix of respective Gaussians. In case any of the Gaussians is only a function of part of the variables x_j, the rows and columns of unrelated variables in corresponding A_is are zeros, and they are to be ignored when computing the inverse.

It is easy to see the multiplication of two such functions and integration of the product results in the same form provided that the product of any two weighted Gaussians is still a weighted Gaussian and the integral of a weighted Gaussian is also a weighted Gaussian. For Gaussians, it is easy to deduce:

$$\begin{aligned} & w_1 \exp\left\{-\frac{1}{2}(\mathbf{x}-\mathbf{u}_1)^T A_1 (\mathbf{x}-\mathbf{u}_1)\right\} \cdot w_2 \exp\left\{-\frac{1}{2}(\mathbf{x}-\mathbf{u}_2)^T A_2 (\mathbf{x}-\mathbf{u}_2)\right\} \\ =& w \exp\left\{-\frac{1}{2}(\mathbf{x}-\mathbf{u})^T A (\mathbf{x}-\mathbf{u})\right\} \end{aligned} \tag{3.5.2}$$

where

$$w = w_1 w_2 \exp\left\{-\frac{1}{2}(\mathbf{u}_1-\mathbf{u}_2)^T A_1 (A_1+A_2)^{-1} A_2 (\mathbf{u}_1-\mathbf{u}_2)\right\} \tag{3.5.3a}$$

$$\mathbf{u} = (A_1+A_2)^{-1}(A_1\mathbf{u}_1 + A_2\mathbf{u}_2) \tag{3.5.3b}$$

$$A = A_1 + A_2 \tag{3.5.3c}$$

[4] $\exp\{x\} \equiv e^x$

And

$$\begin{aligned}&\int w \exp\left\{-\frac{1}{2}\begin{pmatrix}\mathbf{x}_{\bar{j}}-\mathbf{u}_{\bar{j}}\\ x_j-u_j\end{pmatrix}^T\begin{pmatrix}A & B\\ B^T & C\end{pmatrix}\begin{pmatrix}\mathbf{x}_{\bar{j}}-\mathbf{u}_{\bar{j}}\\ x_j-u_j\end{pmatrix}\right\}\mathrm{d}x_j\\ =&w'\exp\left\{-\frac{1}{2}(\mathbf{x}_{\bar{j}}-\mathbf{u}_{\bar{j}})^T A'(\mathbf{x}_{\bar{j}}-\mathbf{u}_{\bar{j}})\right\}\end{aligned} \tag{3.5.4}$$

where

$$w' = w(2\pi)^{\frac{m}{2}}\|C\|^{-\frac{1}{2}} \tag{3.5.5a}$$

$$A' = A - BC^{-1}B^T \tag{3.5.5b}$$

and m is the number of dimensions in x_j, and $x_{\bar{j}}$ stands for the column vector of all other variables that are not x_j.

Appendix A.2 and A.3 list the derivation details for equations (3.5.2) and (3.5.4).

Also, an estimator based on an MoG probability density function is easy to compute. Take the example of equation (3.2.4), if

$$p(\mathbf{x}) = \sum_{i=1}^{n} \frac{w_i}{(2\pi)^{-m/2}\|A_i\|^{-1}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\mathbf{u}_i)^T A_i(\mathbf{x}-\mathbf{u}_i)\right\} \tag{3.5.6}$$

where m is the number of dimensions of $\mathbf{x}$, the estimator becomes

$$\begin{aligned}\hat{\mathbf{x}} &= \int_{\mathbf{x}} \mathbf{x}p(\mathbf{x})\ \mathrm{d}\mathbf{x}\\ &= \sum_{i=1}^{n} w_i\mathbf{u}_i\end{aligned} \tag{3.5.7}$$

3.6 Linearization of Potentials

There are actually two sources of approximation when representing all functions as MoGs: one is to approximate the leaf potentials (likelihood functions) from visual clues as a single Gaussian or an MoG. For example, if the visual clue is a color blob, positions of connected components from the hue-segmented image can be used as centers of Gaussians, although the weights and covariances can be tricky to assign. On the other hand, the potential function $\psi_i j(x_i, x_j)$ on edges mathematically models the physical or logical constraint between components i and j, which is not necessarily naturally in a Gaussian form. In the case it is not, the potential function has to be approximated. One obvious way is to fit the MoG on the function to minimize their difference. However, we use a more efficient approximation called linearization here. It linearizes the square root of the exponent to make the function a Gaussian, as we shall see later in this chapter.

For the purpose of simplicity let us assume for the moment both x_i and x_j are scalars. In the case of one or both of them being vectors, it is but a simple task to adapt the discussion here

accordingly. Since the scale of the potential function does not matter, let $0 < \psi_{ij}(x_i, x_j) \leq 1$, so that we can define

$$\gamma_{ij}(x_i, x_j) = \sqrt{-2 \ln \psi_{ij}(x_i, x_j)} \tag{3.6.1}$$

such that

$$\psi_{ij}(x_i, x_j) = \exp\left\{-\frac{1}{2}\gamma_{ij}^2(x_i, x_j)\right\} \tag{3.6.2}$$

First order Taylor expansion of $\gamma_{ij}(x_i, x_j)$ on the point $(x_i, x_j) = (u, v)$ leads to

$$\begin{aligned}\psi_{ij}(x_i, x_j) &\approx \exp\left\{-\frac{1}{2}\left(\gamma_{ij}(u,v) + (x_i - u)\frac{\partial \gamma_{ij}}{\partial x_i}(u,v) + (x_j - v)\frac{\partial \gamma_{ij}}{\partial x_j}(u,v)\right)^2\right\} \\ &\triangleq \psi'_{ij}(x_i, x_j; u, v)\end{aligned} \tag{3.6.3}$$

Obviously, this is in the form of a Gaussian of variables (x_i, x_j). Now the question is, how to pick the best (u, v). Equation (3.6.3) is a good approximation only for (x_i, x_j) which is in a small neighborhood of (u, v). Recall that $\psi'_{ij}(x_i, x_j)$ is used in the equation (3.3.1) for multiplication with a function of x_i (and other variables) then the product is integrated. The integral is later used in equation (3.3.1) again and (3.3.2) for another multiplication with functions of x_j (and other variables). If both the former functions and at least one of the later functions are in the form of MoG, we can use their peaks (centers of individual Gaussians) as u and v.

Formally, the sub-expression of $m_{jk}(x_k)$ at question is:

$$E(x_j, x_k) = F(x_j, x_k) \int \psi_{ij}(x_i, x_j) G(x_i)\, \mathrm{d}x_i \tag{3.6.4}$$

where both F and G are functions. If both of them are MoGs such that

$$F(x_j, x_k) = \sum_l f_l(x_j, x_k) \tag{3.6.5a}$$

$$G(x_i) = \sum_h g_h(x_i) \tag{3.6.5b}$$

and f_l and g_h are centered at $x_j = v_l$ and $x_i = u_h$ respectively, equation 3.6.4 becomes

$$\begin{aligned} & E(x_j, x_k) \\ = & \sum_l \sum_h f_l(x_j, x_k) \int \psi_{ij}(x_i, x_j) g_h(x_i)\, \mathrm{d}x_i \\ \approx & \sum_l \sum_h f_l(x_j, x_k) \int \psi'_{ij}(x_i, x_j; u_h, v_l) g_h(x_i)\, \mathrm{d}x_i \end{aligned} \tag{3.6.6}$$

Thus we impose the following restriction from the linearization of the constraint: all components shall have at least one natural Gaussian constraint. For example, the graph shown in figure 3.3 is not computable in our framework if both potentials $\psi(x, z)$ and $\psi(y, z)$ are non-Gaussian, since Z does not have any natural point of linearization for processing messages. This restriction should not be a problem in practice since there is always the option to attach a history component.

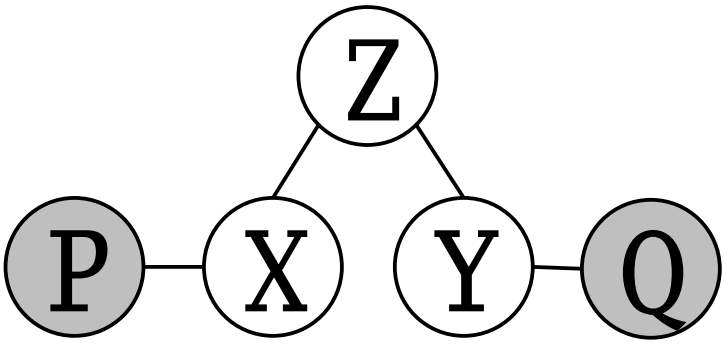


Figure 3.3: A Forbidden graph. The graph is not computable if both potentials $\psi(x,z)$ and $\psi(y,z)$ are non-Gaussian.

A common type of non-linear constraints is the "distance" constraint. Suppose it is between two components i and j whose variables are $\mathbf{x}_i$ and $\mathbf{x}_j$ respectively, the original constraint equation is

$$(A\mathbf{x}_i - B\mathbf{x}_j)^T(A\mathbf{x}_i - B\mathbf{x}_j) = d^2 \tag{3.6.7}$$

where A and B are matrices and d is the scalar which represents the "distance" between $\mathbf{x}_i$ and $\mathbf{x}_j$.

A nature construction for the potential function between components i and j is

$$\psi_{ij}(\mathbf{x}_i, \mathbf{x}_j) = \exp\left\{-\frac{1}{2}\left(\sqrt{(A\mathbf{x}_i - B\mathbf{x}_j)^T(A\mathbf{x}_i - B\mathbf{x}_j)} - d\right)^2\right\} \tag{3.6.8}$$

as listed in table 3.4.2.

The linearization of equation (3.6.8) at point $(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{u}, \mathbf{v})$ as previously described in this section is

$$\psi'_{ij}(\mathbf{x}_i, \mathbf{x}_j; \mathbf{u}, \mathbf{v}) = \exp\left\{-\frac{1}{2}\left(\frac{2}{k}(A\mathbf{u} - B\mathbf{v})^T(A\mathbf{x}_i - B\mathbf{x}_j) - k - d\right)^2\right\} \tag{3.6.9}$$

where

$$k = \sqrt{(A\mathbf{u} - B\mathbf{v})^T(A\mathbf{u} - B\mathbf{v})} \tag{3.6.10}$$

Appendix A.6 lists the derivation details for equation (3.6.9) and (3.6.10).

In the case that both A and B can be further simplified to the identity matrix I multiplied by a scalar factor, d can "absorb" the scalar factor and equation (3.6.8) through equation (3.6.10) become[5]

$$\psi_{ij}(\mathbf{x}_i, \mathbf{x}_j) = \exp\left\{-\frac{1}{2\sigma^2}\left(\sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_j)} - d\right)^2\right\} \tag{3.6.11}$$

[5]Note that the definition of d is no longer same as in equation (3.6.7).

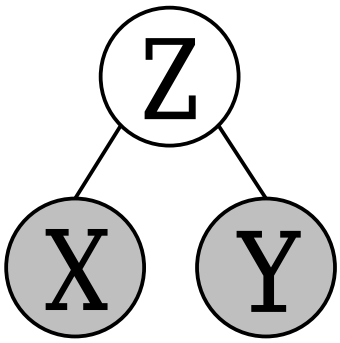


Figure 3.4: A simple graph with two leaf components X and Y and a part component Z as the result.

where σ represents the "elasticity" of the constraint and

$$\psi'_{ij}(\mathbf{x}_i, \mathbf{x}_j; \mathbf{u}, \mathbf{v}) = \exp\left\{-\frac{1}{2\sigma^2}\left(\frac{2}{k}(\mathbf{u}-\mathbf{v})^T(\mathbf{x}_i - \mathbf{x}_j) - k - d\right)^2\right\} \tag{3.6.12}$$

where

$$k = \sqrt{(\mathbf{u}-\mathbf{v})^T(\mathbf{u}-\mathbf{v})} \tag{3.6.13}$$

3.7 Null Hypotheses

3.7.1 Purpose

The purpose of introducing the so called "null hypothesis" in this component-based tracking framework is to account for the possibility that some of the leaf nodes, thus part of the graph, provides no useful information at all. For example, figure 3.4 shows a simple graph with three nodes, two of them leaves.

If both nodes X and Y are observed with corresponding potential function $\psi_x(x)$ and $\psi_y(y)$, and the constraints between X and Z and Y and Z are $\psi_{xz}(x,z)$ and $\psi_{yz}(y,z)$, respectively, the joint probability density function of the entire graph is

$$p_{xyz}^{(xyz)}(x,y,z) = Z_{xyz}^{-1}\psi_x(x)\psi_{xz}(x,z)\psi_y(y)\psi_{yz}(y,z) \tag{3.7.1}$$

where

$$Z_{xyz} = \int_z\int_y\int_x \psi_x(x)\psi_{xz}(x,z)\psi_y(y)\psi_{yz}(y,z)\,\mathrm{d}x\mathrm{d}y\mathrm{d}z \tag{3.7.2}$$

The superscript (xyz) is used above to indicate the situation that all three nodes X, Y and Z are present.

We only care about Z, so the marginal probability density for Z is

$$p_z^{(xyz)}(z) = \int_y\int_x p_{xyz}^{(xyz)}(x,y,z)\,\mathrm{d}x\mathrm{d}y \tag{3.7.3}$$

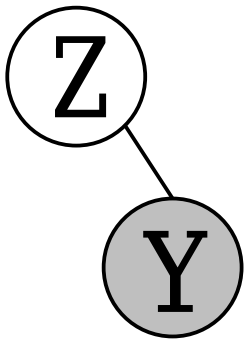


Figure 3.5: A Possible Situation with Node X Missing.

Alternatively, $p(z)$ can be expressed in the form of product of messages from nodes X and Y:

$$p_{xyz}^{(xyz)}(z) = Z_{xyz}^{-1} m_{xz}(z) m_{yz}(z) \tag{3.7.4}$$

where

$$m_{xz}(z) = \int_x \psi_x(x)\psi_{xz}(x,z)\ \mathrm{d}x \tag{3.7.5a}$$

$$m_{yz}(z) = \int_y \psi_y(y)\psi_{yz}(y,z)\ \mathrm{d}y \tag{3.7.5b}$$

Equation (3.7.3) and equation (3.7.4) are equivalent, however, the latter is more graphically intuitive and it is the way our component-based tracking framework works.

Now consider the possible situation that for some reason, the sensor corresponding to the leaf node X does not provide any useful information at all. In other words, the leaf node X is not present in the graph in this situation and only the nodes Y and Z exist, as shown in figure 3.5.

In this case, the joint probability density function of the entire graph is

$$p_{yz}^{(yz)}(y,z) = Z_{yz}^{-1} \psi_y(y)\psi_{yz}(y,z) \tag{3.7.6}$$

where

$$Z_{yz} = \int_z \int_y \psi_y(y)\psi_{yz}(y,z)\ \mathrm{d}y\mathrm{d}z \tag{3.7.7}$$

and

$$p_z^{(yz)}(z) = \int_y p_{yz}^{(yz)}(y,z)\ \mathrm{d}y \tag{3.7.8}$$

The superscript (yz) is used above to indicate the situation that only two nodes Y and Z are present.

In the term of messages, this can be expressed as

$$p_z^{(yz)}(z) = Z_{yz}^{-1} m_{yz}(z) \tag{3.7.9}$$

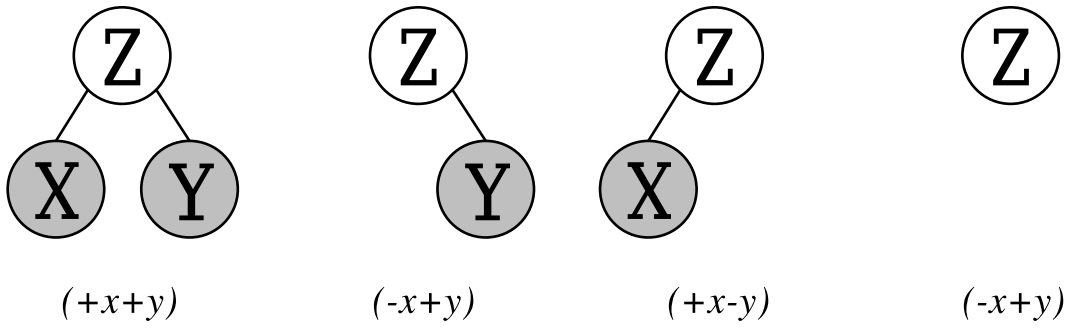


Figure 3.6: The Ensemble for All Four Possible Situations

Note that the message $m_{yz}(z)$ remains the same in both situations.

Call the original graph in figure 3.4 the whole graph, and let us formally define a "situation" of a whole graph as a connected subset of the whole graph. Obviously, the whole graph itself is a situation, too. If we assign probability $w^{(+x)}$ to the situation in figure 3.4 and probability $w^{(-x)}$ to the situation in figure 3.5, where $w^{(+x)} + w^{(-x)} = 1$, the overall (mean) marginal probability distribution of Z with regard to the set of these two situations is

$$\begin{aligned} p_z(z) &= w^{(+x)} p_z^{(xyz)}(z) + w^{(-x)} p_z^{(yz)}(z) \\ &= w^{(+x)} Z_{xyz}^{-1} m_{xz}(z) m_{yz}(z) + w^{(-x)} Z_{yz}^{-1} m_{yz}(z) \\ &= Z_{xyz}^{-1} (w^{(+x)} m_{xz}(z) + w^{(-x)} \frac{Z_{xyz}}{Z_{yz}}) m_{yz} \end{aligned} \tag{3.7.10}$$

Define

$$m'_{xz}(z) = w^{(+x)} m_{xz}(z) + w^{(-x)} \frac{Z_{xyz}}{Z_{yz}} \tag{3.7.11}$$

then

$$p_z(z) = Z_{xyz}^{-1} m'_{xz}(z) m_{yz}(z) \tag{3.7.12}$$

which is in the same form as equation (3.7.4).

Thus, we can compute the overall $p_z(z)$ without going through both situations, instead, we only need to do the computation as in the full graph, with corresponding graph branches sending messages which have an additional constant term to indicate the possibility that this particular branch is missing.

Furthermore, we want to consider the possibilities that the branch with leaf node Y is present or absent as well. Define an "ensemble" to be the set of all situations that contains a node of interest, which is Z here. In our case the ensemble as shown in figure 3.6 contains 4 situations, each with its probability: $w^{(+x+y)}$, $w^{(-x+y)}$, $w^{(+x-y)}$, and $w^{(-x-y)}$. Assuming the presence or absence

of X and Y are independent of each other, we have

$$w^{(+x+y)} = w^{(+x)}w^{(+y)} \tag{3.7.13a}$$

$$w^{(-x+y)} = w^{(-x)}w^{(+y)} \tag{3.7.13b}$$

$$w^{(+x-y)} = w^{(+x)}w^{(-y)} \tag{3.7.13c}$$

$$w^{(-x-y)} = w^{(-x)}w^{(-y)} \tag{3.7.13d}$$

where $w^{(+y)}$ and $w^{(-y)}$ are the respective possibilities for whether node Y is present or absent, and $w^{(+y)} + w^{(-y)} = 1$.

At this moment, let us ignore the fact that the single Z node under $w^{(-x-y)}$ is a rather trivial situation and cannot be used for interference by itself. The overall marginal probability distribution of Z now becomes

$$\begin{aligned} p_z(z) =& w^{(+x+y)}p_z^{(xyz)}(z) + w^{(-x+y)}p_z^{(yz)}(z) \\ &+ w^{(+x-y)}p_z^{(xz)}(z) + w^{(-x-y)}p_z^{(z)}(z) \\ =& w^{(+x)}w^{(+y)}Z_{xyz}^{-1}m_{xz}(z)m_{yz}(z) + w^{(-x)}w^{(+y)}Z_{yz}^{-1}m_{yz}(z) \\ &+ w^{(-x)}w^{(+y)}Z_{xz}^{-1}m_{yz}(z) + w^{(-x)}w^{(-y)}Z_z^{-1} \\ =& Z_{xyz}^{-1}(w^{(+x)}m_{xz}(z) + w^{(-x)}\frac{Z_{xyz}}{Z_{yz}})(w^{(+y)}m_{yz}(z) + w^{(-y)}\frac{Z_{xyz}}{Z_{xz}}) \\ &+ Z_{xyz}^{-1}(\frac{Z_{xyz}}{Z_z} - \frac{Z_{xyz}}{Z_{yz}}\frac{Z_{xyz}}{Z_{xz}}) \end{aligned} \tag{3.7.14}$$

In addition to equation (3.7.11), we define

$$m'_{yz}(z) = w^{(+y)}m_{yz}(z) + w^{(-y)}\frac{Z_{xyz}}{Z_{xz}} \tag{3.7.15}$$

and

$$\Delta = Z_{xyz}^{-1}(\frac{Z_{xyz}}{Z_z} - \frac{Z_{xyz}}{Z_{yz}}\frac{Z_{xyz}}{Z_{xz}}) \tag{3.7.16}$$

then

$$p_z(z) = Z_{xyz}^{-1}m'_{xz}(z)m'_{yz}(z) + \Delta \tag{3.7.17}$$

which is again in the same form as equation (3.7.4) if we ignore Δ, which will be discussed in section 3.7.2 later.

More generally, we can adjust every message to include an additional weighted term to indicate the possibility of the absence of the node which the message is originated from. We call this additional term (not including its weight) a "null hypothesis", because each of the Gaussians that forms the original message is thought of as a hypothesis, thus the absence of the node and the corresponding lack of information is a "null" hypothesis. The later section 3.7.3 explains why the null hypothesis is not the constant number 1. We use symbol ζ for the null hypothesis.

As we can see from our example above, by using a null hypothesis adjusted message in place of the original messages when inferring on the graph, we can directly compute the overall

(mean) marginal probability of our node of interest over the ensemble without going over each of the situations individually. Presumably, the ensemble is a better model for the reality than any of its member situations if the probabilities of each of situations are accurate. Then, by using null-hypothesis adjusted messages we have the following advantage:

- The result is more robust than the result from only one situation, for example the whole graph, because the ensemble covers all situations.
- The computational cost is significantly lower than computing the results for all situations individually, because the number of the situations grows exponentially with the number of nodes in the graph.
- The computation is confined to be local. Naturally, only the sensor itself can estimate the probability that it is not functional.

3.7.2 Weight Adjustment

All the above is well except for two issues: first, we do not know the ratio between partition functions, yet. The value of the partition function itself is not needed at all if only one situation is of concern, since we can normalize the result in the end. If we are to compute its value, we have to wait till the computation of the whole graph is completed – this defeats the purpose of locality; if we are to compute the values of both partition functions, we have to actually go through both graphs – this defeats the purpose of saving computation cost.

The second problem appears in the form of Δ as in equation (3.7.16) when more than one branch may be missing. Although the probabilities over all the situations in the ensemble is calculated correctly by using the null hypothesis provided that the missing probabilities of branches are independent of each other, the (yet) unknown ratio of partition functions is not a simple product:

$$\frac{Z_{xyz}}{Z_z} \neq \frac{Z_{xyz}}{Z_{yz}} \cdot \frac{Z_{xyz}}{Z_{xz}} \tag{3.7.18}$$

In other words, $\Delta \neq 0$, so even if we get both $\frac{Z_{xyz}}{Z_{yz}}$ and $\frac{Z_{xyz}}{Z_{xz}}$ correct, their product is still not the right $\frac{Z_{xyz}}{Z_z}$ which is what we want.

Here we can pinpoint the exact problem: we want to estimate the impact of a missing branch on the partition function of the graph. As shown in figure 3.7, the branch X in question may be missing, and the (unmodified) message it passes to node Z is $g(z)$. Assume the product of the messages from rest of the graph to Z is $f(z)$,[6] then the ratio we want to know is

$$R = \frac{\int_z f(z)g(z)\,\mathrm{d}z}{\int_z f(z)\,\mathrm{d}z} \tag{3.7.19}$$

[6]It does not matter whether node Z is really the variable we care about or not, since the value of the partition function is a constant given the graph and can be computed using any node as the "final" node.

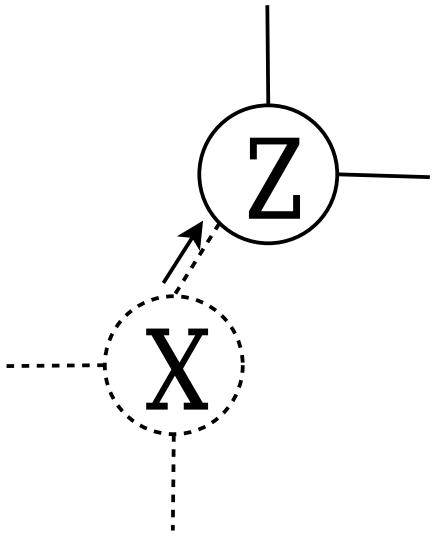


Figure 3.7: A general graph focused on the message from X to Z, where X has the possibility to be missing.

However, so far we only know $g(z)$ but not $f(z)$. Consider the fact that both are Mixture of Gaussians, what is our estimate of the ratio R?

One possible solution is to simply normalize $g(z)$, which is the actual method we used in the implementation:

$$R \approx \int_z g(z)\, \mathrm{d}z \tag{3.7.20}$$

If we take this simple normalization approach, the problem of Δ can be solved as well, since in this approximation,

$$\Delta \approx 0 \tag{3.7.21}$$

This normalization is done after the messages are computed but before they are sent. This effectively adjusts the relative weights of the null hypotheses.

3.7.3 Operation Rules

Recall that we represent all messages in the form of mixture of Gaussians in the form of equation (3.5.1). However, as we can see from equations (3.7.11) and (3.7.15), the modified messages are no longer in such form due to the null hypothesis. Furthermore, while this additional term acts like a constant term when calculating the marginal distribution, i.e. the products of incoming messages, it does not act so when calculating the outgoing message:

$$m_{ij}(x_j) = \int \psi_{ij}(x_i, x_j) \prod_{k \in N(i) \setminus j} m_{ki}(x_i)\, \mathrm{d}x_i \tag{3.7.22}$$

Here, before the integration there is a final multiplication between the potential function and the partial product of the incoming messages, in addition to the multiplication to form the latter. For this final multiplication, the null hypothesis from the partial product cannot be treated as a constant term. The reason is that the null hypothesis here indicates that none of the underlying branches provide any information, thus the whole sub-tree shall not provide any information in this case either, regardless of the value of potential function. The null hypothesis itself is the only meaningful result for this final multiplication in the case of one of the operand is a null hypothesis.

In summary, the general form of MoG in equation (3.5.1) is modified to

$$f(\mathbf{x}) = \sum_{i=1}^{n} w_i \exp\left\{-\frac{1}{2}(\mathbf{x}-\mathbf{u}_i)^T A_i (\mathbf{x}-\mathbf{u}_i)\right\} + w_o\zeta \tag{3.7.23}$$

where ζ is the symbol denoting the null hypothesis, and w_0 is the weight of null hypothesis.

Equations (3.5.2) and (3.5.3) still apply for Gaussian terms, however, when the computation involves the null hypothesis term ζ, the different sets of rules as below apply instead.

When multiplying the null hypothesis with other messages:

$$\zeta \cdot f(x) = f(x) \tag{3.7.24a}$$

$$\zeta \cdot \zeta = \zeta \tag{3.7.24b}$$

When multiplying the null hypothesis with the potential function and integrating:

$$\zeta \cdot f(x) = \zeta \tag{3.7.25a}$$

$$\zeta \cdot \zeta = \zeta \tag{3.7.25b}$$

$$\int \zeta = \zeta \tag{3.7.25c}$$

Last but no least, we need an estimator based on the following density that includes null hypothesis:

$$p(\mathbf{x}) = \sum_{i=1}^{n} \frac{w_i}{(2\pi)^{-m/2}\|A_i\|^{-1}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\mathbf{u}_i)^T A_i (\mathbf{x}-\mathbf{u}_i)\right\} + w_0\zeta \tag{3.7.26}$$

where m is the number of dimensions of $\mathbf{x}$.

Since the null hypothesis itself does not confer any useful information but only the lack of it, the estimator in equation (3.5.7) becomes:

$$\hat{\mathbf{x}} = \frac{\sum_{i=1}^{n} w_i \mathbf{u}_i}{\sum_{i=1}^{n} w_i} \tag{3.7.27}$$

3.8 Pruning and Merging Hypotheses

Since all our messages take the form of sums of Gaussians, a problem naturally arises when the messages are multiplied together: the number of Gaussian terms increases exponentially. Although we can definitely prune the lowest weighted terms to keep the number below an upper

limit, which we do in the end, it is always better to find another more delicate method to decrease the number of terms before the brutal purge.

One way to do this is to merge the similar Gaussian terms, each of which represents a hypothesis. If two Gaussian functions are very close to each other, we may as well use one to represent both.[7] The weight of merged hypothesis is the sum of both hypotheses.

Now the question is what would be a good metric[8] for the Gaussian functions. Assuming the two Gaussians in question are $G_1(\mathbf{x})$ and $G_2(\mathbf{x})$, that

$$G_1(\mathbf{x}) = (2\pi)^{-\frac{m}{2}} \|A_1\|^{\frac{1}{2}} \exp\{-\frac{1}{2}(\mathbf{x}-\mathbf{u}_1)^T A_1 (\mathbf{x}-\mathbf{u}_1)\} \tag{3.8.1a}$$

$$G_2(\mathbf{x}) = (2\pi)^{-\frac{m}{2}} \|A_2\|^{\frac{1}{2}} \exp\{-\frac{1}{2}(\mathbf{x}-\mathbf{u}_2)^T A_2 (\mathbf{x}-\mathbf{u}_2)\} \tag{3.8.1b}$$

where m is the number of dimensions of $\mathbf{x}$ and $\|M\|$ stands for the absolute value of determinant of matrix M, we define distance D as

$$D(G_1, G_2) = \int_{\mathbf{x}} (G_1(\mathbf{x}) - G_2(\mathbf{x}))^2 \, \mathrm{d}\mathbf{x} \tag{3.8.2}$$

Allow the same deduction as in equations (3.5.2) and (3.5.2), we have

$$\int_{\mathbf{x}} G_1(\mathbf{x}) G_2(\mathbf{x}) \, \mathrm{d}\mathbf{x} = (2\pi)^{-\frac{m}{2}} \|A_1\|^{\frac{1}{2}} \|A_2\|^{\frac{1}{2}} \|A_1 + A_2\|^{-\frac{1}{2}} R \tag{3.8.3}$$

where

$$R = \exp\{-\frac{1}{2}(\mathbf{u}_1 - \mathbf{u}_2)^T A_1 (A_1 + A_2)^{-1} A_2 (\mathbf{u}_1 - \mathbf{u}_2)\} \tag{3.8.4}$$

therefore

$$\begin{aligned} D &= \int_{\mathbf{x}} (G_1(\mathbf{x}) - G_2(\mathbf{x}))^2 \, \mathrm{d}\mathbf{x} \\ &= \int_{\mathbf{x}} \left[(G_1(\mathbf{x}))^2 + (G_2(\mathbf{x}))^2 - 2G_1(\mathbf{x})G_2(\mathbf{x}) \right] \mathrm{d}\mathbf{x} \\ &= (2\pi)^{-\frac{m}{2}} \left[\frac{\sqrt{2}}{2} \|A_1\|^{\frac{1}{2}} + \frac{\sqrt{2}}{2} \|A_2\|^{\frac{1}{2}} - 2\|A_1\|^{\frac{1}{2}} \|A_2\|^{\frac{1}{2}} \|A_1 + A_2\|^{-\frac{1}{2}} R \right] \end{aligned} \tag{3.8.5}$$

With the metric D defined, the criterion of whether G_1 and G_2 are close enough is

$$D(G_1, G_2) < D_{\text{th}} \tag{3.8.6}$$

where D_{th} is a pre-given threshold.

However, there is still one catch for the above similarity metric D: the value of D depends on the unit of $\mathbf{x}$. More specifically, D is inversely proportional to the unit of $\mathbf{x}$. This is because the probability density function is of a dimension inverse to that of $\mathbf{x}$. It may not be a problem as long as it has been taken into account when choosing the threshold D_{th}, but it would be better if we can

[7] Or better yet, replace both with a new Gaussian which is a weighted combination of the originals. However, if the original Gaussians are already very close, this would not make much difference anyway.

[8] Actually, being a true metric is not a requirement, just our choice happens to be one.

have a universal threshold without considering the situational choice of unit every time, and also for all variables of the entire graph. The following normalization would make the metric dimensionless:

$$D^*(G_1, G_2) = \frac{\int_{\mathbf{x}} (G_1(\mathbf{x}) - G_2(\mathbf{x}))^2 \ \mathrm{d}\mathbf{x}}{\int_{\mathbf{x}} (G_1(\mathbf{x}) + G_2(\mathbf{x}))^2 \ \mathrm{d}\mathbf{x}} \tag{3.8.7}$$

Using the same deduction as before, we have

$$D^*(G_1, G_2) = \frac{\sqrt{2}\|A_1\|^{\frac{1}{2}} + \sqrt{2}\|A_2\|^{\frac{1}{2}} - 4\|A_1\|^{\frac{1}{2}}\|A_2\|^{\frac{1}{2}}\|A_1 + A_2\|^{-\frac{1}{2}}R}{\sqrt{2}\|A_1\|^{\frac{1}{2}} + \sqrt{2}\|A_2\|^{\frac{1}{2}} + 4\|A_1\|^{\frac{1}{2}}\|A_2\|^{\frac{1}{2}}\|A_1 + A_2\|^{-\frac{1}{2}}R} \tag{3.8.8}$$

and the criterion under this normalized metric is

$$D^*(G_1, G_2) < D^*_{\text{th}} \tag{3.8.9}$$

where D^*_{th} is a pre-given threshold.

Furthermore, if one or both of the hypotheses are not Gaussians but null hypotheses, the following rules apply:

$$D^*(G, \zeta) = \infty \tag{3.8.10a}$$

$$D^*(\zeta, G) = \infty \tag{3.8.10b}$$

$$D^*(\zeta, \zeta) = 0 \tag{3.8.10c}$$

To give an intuitive example on reasonable choices of the threshold, assume $A_1 = A_2 = A$ for a particular case. Then in such a case,

$$D^*(G_1, G_2) = \frac{1 - R}{1 + R} \tag{3.8.11}$$

where

$$R = \exp\{-\frac{1}{4}(\mathbf{u}_1 - \mathbf{u}_2)^T A^{-1}(\mathbf{u}_1 - \mathbf{u}_2)\} \tag{3.8.12}$$

To cover the situations that two hypotheses are within the area of equivalent 1σ of each other,

$$D^*_{\text{th},1\sigma} = \frac{1 - e^{0.25}}{1 + e^{0.25}} = 0.124 \tag{3.8.13}$$

Nevertheless, 1σ is a pretty loose region. If the two are within the area of equivalent 0.1σ of each other,

$$D^*_{\text{th},0.1\sigma} = \frac{1 - e^{0.0025}}{1 + e^{0.0025}} = 0.00125 \tag{3.8.14}$$

Another example is the case of $\mathbf{u}_1 = \mathbf{u}_2$, but $A_1 = r^2 A_2$. In this case,

$$D^*(G_1, G_2) = \frac{(r+1)\sqrt{2(r^2+1)} - 4r}{(r+1)\sqrt{2(r^2+1)} + 4r} \tag{3.8.15}$$

For the ratio $r = 1.1$, $D^*_{\text{th}} = 0.0170$.

3.9 Observations

Observations are results generated by image-level sensors. They are leaf potentials of observation components in the framework. The potential function is in the form of MoG with the null hypothesis. In this section we briefly talk about some of the observations in table 3.4.1, and focus on how to fit the region trackers in chapter 2 into the component-based tracking framework.

The general form of the leaf potentials generated by the sensors from the image is

$$p(\mathbf{x}) = \sum_{i=1}^{n} w_i G(\mathbf{x}; \mathbf{u}_i, A_i) + w_0 \zeta \tag{3.9.1}$$

where $G(\mathbf{x}; \mathbf{u}, A)$ is the Gaussian

$$G(\mathbf{x}; \mathbf{u}, A) = \frac{1}{(2\pi)^{\frac{m}{2}} \|A\|^{-1}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\mathbf{u})^T A(\mathbf{x}-\mathbf{u})\right\} \tag{3.9.2}$$

and m is the number of dimensions of $\mathbf{x}$. The questions for each type of observations are:

1. How to generate the peak x_i and the covariance A_i for each hypothesis.
2. How to assign the weight w_i, $i = 1, \ldots, n$ to corresponding hypothesis.
3. How to assign the weight w_0 to the null hypothesis ζ.

Question 3 is easy. Because the weights are only relative, we can simply assign a constant number to w_0, as long as the weights w_i,$i = 1, \ldots, n$ are properly assigned. However, questions 1 and 2 are specific to individual sensors.

3.9.1 Corners

Corner points are one of the most used features in computer vision. The state vector consists of the x and y coordinates of the corner points. Let $I(x, y)$ be the image, the Harris corner detector defines a matrix for a point (x, y):

$$M = \begin{pmatrix} A & B \\ B & C \end{pmatrix} = \begin{pmatrix} I'_x(x,y)I'_x(x,y) & I'_x(x,y)I'_y(x,y) \\ I'_x(x,y)I'_y(x,y) & I'_y(x,y)I'_y(x,y) \end{pmatrix} \tag{3.9.3}$$

where $I'_x(x, y)$ and $I'_y(x, y)$ are gradients on x and y directions, smoothed over a local neighborhood of (x, y). The corner "strength" is defined as

$$R = \det(M) - k \cdot \mathrm{trace}(M) = (AC - B^2) - k(A + C)^2 \tag{3.9.4}$$

where k is an empirical constant. Given a threshold R_{th}, if

$$R \geq R_{th} \tag{3.9.5}$$

and R is local maximum at (x, y), we take (x, y) as a corner point.

The above can be easily implemented in an algorithm that generates the set of corner points $\{(x_i^*, y_i^*),\ i = 1, \ldots, n\}$ from the image. So it would be easy to fit the corner detector into the component-based tracking framework: for $i = 1, \ldots, n$,

$$\mathbf{u}_i = \begin{pmatrix} x_i^* \\ y_i^* \end{pmatrix} \tag{3.9.6a}$$

$$A_i = \frac{1}{\sigma^2} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{3.9.6b}$$

$$w_i = \omega R \tag{3.9.6c}$$

where σ and ω are constants.

3.9.2 Blobs

Blobs are regions that are homogeneous or otherwise statistically distinguishable on some feature space. To make it simple, in this section we assume the state vector be the x and y coordinates of the center of the blob. A blob consists of a set of pixels. These pixels are segmented from the rest of the image using a binary function on certain characteristic of the pixel and maybe its neighbors. For example, the segmentation can be based on intensity, hue or texture. The resulting positive pixels are classified into different blobs according to some criteria, for example, connectivity, distance or result of some clustering algorithm. Usually the blob itself must meet some extra requirements to be considered, for example, the area must be greater than a certain number of pixels.

Assuming there are n blobs $B_i = \{(x_{ij}^*, y_{ij}^*),\ j = 1, \ldots, m_i\},\ i = 1, \ldots, n$ detected in a particular image I, we can let

$$\mathbf{u}_i = \begin{pmatrix} \frac{1}{m_i} \sum_{j=1}^{m_i} x_{ij}^* \\ \frac{1}{m_i} \sum_{j=1}^{m_i} y_{ij}^* \end{pmatrix} \tag{3.9.7a}$$

$$A_i = \begin{pmatrix} \frac{1}{m_i - 1} \sum_{j=1}^{m_i} (x_{ij}^* - u_i)^2 & \frac{1}{m_i - 1} \sum_{j=1}^{m_i} (x_{ij}^* - u_i)(y_{ij}^* - v_i) \\ \frac{1}{m_i - 1} \sum_{j=1}^{m_i} (x_{ij}^* - u_i)(y_{ij}^* - v_i) & \frac{1}{m_i - 1} \sum_{j=1}^{m_i} (y_{ij}^* - v_i)^2 \end{pmatrix}^{-1} \tag{3.9.7b}$$

$$w_i = \frac{1}{m_i} \sum_{j=1}^{m_i} \rho(x, y; B_i) \tag{3.9.7c}$$

where $\rho(x, y; B_i)$ is the "fittest" measure for pixel (x, y) to blob B_i. If a prior model of the characteristics of the blob is known beyond simple screening requirement, for example on its area or shape, this knowledge can be incorporated in w_i as an additional multiplier.

3.9.3 Region Tracker and Sensor Hints

For heterogeneous regions that are identifiable by their internal spatial structure, the region trackers in chapter 2 can be used. The state vector is the transformation parameters. The estimators

of both the SSD tracker and the optimal kernel tracker have the same form:

$$\hat{\tau}(\tau_0) = \tau_0 + \sum_{\mathbf{x}} k(\mathbf{x}) \left(I(T(\mathbf{x}; \tau_0)) - I_0(\mathbf{x}) \right) \tag{3.9.8}$$

In addition, we define a verifier

$$E(\hat{\tau}) = \frac{1}{2\pi\sigma_z^2} \exp\left\{ -\frac{1}{2\sigma_z^2 s} \sum_{\mathbf{x}} \left(I(T(\mathbf{x}; \hat{\tau})) - I_0(\mathbf{x}) \right)^2 \right\} \tag{3.9.9}$$

where σ_z is a constant and s is the area of the template image I_0. Now, we can almost fit the region trackers into the component-based tracking framework:

$$\mathbf{u}_i = \hat{\tau}(\mathbf{u}_i^-) \tag{3.9.10a}$$

$$A_i = \frac{1}{\sigma^m} I_d(m) \tag{3.9.10b}$$

$$w_i = E(\mathbf{u}_i) \tag{3.9.10c}$$

where σ is a constant, m is the number of dimensions of the parameters, and $I_d(m)$ refers to the m by m identity matrix.

However, there is still one thing missing from the above equations (thus the "almost"). We need the initial values $\mathbf{u}_i^-$ to estimate the new parameters. A stand-alone region tracker can simply use the value of the previous estimate. In this framework, we have a better solution.

Ignoring its own potential function, the observation component in question receives messages from the rest of the graph, too. The marginal density as the product of these messages presents the information from the rest of the graph about the observation. Same as everything else in the graph, the density $p'(\mathbf{x})$ is in the form of MoG:

$$p'(\mathbf{x}) = \sum_{i=1}^{n} w_i' G(\mathbf{x}; \mathbf{u}_i', A_i') + w_0' \zeta \tag{3.9.11}$$

Now we can use the peak positions of these individual Gaussians as the initial values for the parameters:

$$\mathbf{u}_i^- = \mathbf{u}_i' \tag{3.9.12}$$

Since $p'(\mathbf{x})$ is what the rest of the graph uses to tell the sensor where to look, we call it sensor hints.

3.10 Relationship with Established Works

The component-based tracking framework is built upon probabilistic graphical models; however, instead of using the usual sampling approach on messages for continuous variables, a sum of Gaussian representation is used and all computations are essentially manipulating the mean and covariance of the Gaussian. In fact, the algorithm is deterministic even the framework is probabilistic. This is the fundamental difference with all other existing works using probabilistic graphical models

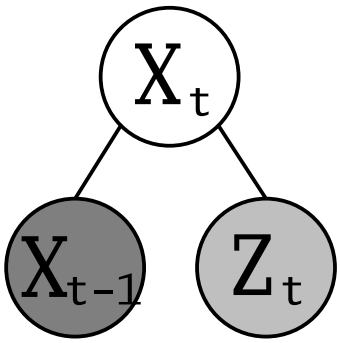


Figure 3.8: A simple graph is equivalent to the standard Kalman filter if the observation Z_t is a single Gaussian without null hypothesis.

on visual tracking. Furthermore, the null hypothesis is a new introduction to address the robustness issue.

The standard Kalman filter can be thought as a special case of the component-based tracking framework. Figure 3.8 shows a simple graph with a history component X_{t-1}, an observation component Z_t, and a part component X_t. It is equivalent to the standard Kalman filter if the observation Z_t is a single Gaussian without null hypothesis. The system (dynamic) equation in the Kalman filter is incorporated as the constriant between components X_{t-1} and X_t, while the observation (measurement) equation is incorporated as the constraint between components X_t and Z_t. By assigning the measurment to the sensor of Z_t and infering on X_t, the exact equations of the Kalman filters can be derived. For a detailed proof please see appendix A.4.

If we ignore the variance of the Gaussian term, each hypothesis is essentially a weighted sample in the particle filters. However, there is still difference: in our framework the hypotheses only get verified but not resampled. In this sense it is much like the bottom-up approaches like RANSAC.

Chapter 4

Algorithm and Simulation

4.1 Algorithm Description

Chapter 3 provides the mathematical basis for the component-based tracking framework, however, an actual algorithm must be implemented based on this framework. This section details the algorithm used in the implementation.

As stated in section 3.2, a graph represents our probabilistic model. Our algorithm runs on this graph as described in section 3.3. The graph consists of a set of vertices ($v_i \in V$), or nodes, and a set of edges ($e_{ij} \in E$). Each node is associated with a set of variables x_i. In addition, each leaf node is associated with a fixed "sensor" module, which provides the observed likelihood for each time-step. Each edge is associated with a fixed potential module, which either is already in the form of mixture of Gaussians ("linear"), or can be linearized at given points ("non-linear").

On the variable side, since our algorithm only accesses local[1] information for all nodes in the graph at each iteration, the result of previous iteration is always used for calculation of the value of the current iteration. In this section we use superscript $^-$ to denote the value of previous iteration. Each vertex v_i is associated with a marginal probability p_i, and each edge is associated with a pair of messages m_{ij} and m_{ji}. These messages are either the true messages to be sent out if the potential of the edge is linear, or the partial product of incoming messages of v_i before multiplication with the potential and integration over the variables, if the potential of the edge is non-linear. The reason for the latter is that the linearization of potential ψ_{ij} needs points of both x_i and x_j, and the latter is unavailable until the message is used in another multiplication operation itself.

Note that m_{ij} (and m_{ij}^-) is function of x_i and x_j and p_i (and p_i^-) is function of x_i. In the computation these variables x_i and y_i are never actually evaluated but are kept as mere symbols. Messages and potentials are both in the form of a collection of weighted Gaussian terms, each of them is represented by an exponent matrix and a vector of symbols for the corresponding

[1] Here "local" mean the node itself and its neighboring edges.

1. Properly initialize p_i^- and m_{ij} for all vertices and edges.

2. Repeat for L times, in each iteration:

   (a) Set $p_i^- = p_i$ for all non-leaf vertices v_i.

   (b) Set $m_{ij}^- = m_{ij}$ for all edges e_{ij}

   (c) Set $p_i = \prod_k m_{ik}^-$

   (d) For all edges e_{ij},

   - if v_i is leaf, set $m_{ij} = p_i^-$;
   - otherwise set $m_{ij} = \int \psi_{ij} \prod_{k \neq j} m_{ik}^- \, \mathrm{d}x_i$

Figure 4.1: The Standard Algorithm of Running the Graph for Each Time-step

variables. Section 3.5 details symbolic multiplication and integration for this representation. In the implementation, however, since the multiplication of two Gaussians is the most common operation, it is computationally more effective to combine the mean vector and inverse covariance matrix of the Gaussian into one larger matrix so the multiplication operation is simplified to a single matrix addition. We call this form of the Gaussian the "homogeneous form" because it resembles the homogeneous coordinates used in computer graphics and vision. For details on the homogeneous form and on the rules of calculations of Gaussians under the homogeneous form, please refer to appendix B. In addition, rules regarding null hypotheses are listed in section 3.7.3.

If we do not consider all the issues about potential linearization, null hypothesis, pruning and merging of hypotheses, and sensor hints, the standard message-passing inference algorithm can be applied on the graph, as shown in Figure 4.1.

However, since we do need to consider these issues, the actual algorithm is much more complex. Figure 4.2 shows the algorithm for running the graph at each time-step. To allow the messages propagate from one end of the graph to another, the number of iterations L is at least the diameter[2] of the graph plus 1.[3]

Note that a few important details are missing from the algorithm description in figure 4.2. One is the merging and pruning of Gaussian terms as described in section 3.8. This happens to every product computed, as well as to every message calculated, at steps 4(c)iiB and 4(c)iiC, and 4(c)vB and 4(c)vC of the algorithm. Figure 4.3 and 4.4 describe this process. The exact parameters for the algorithms are dictated by the policy provided by the application.

Another important detail absent in figure 4.2 is the delayed linearization of potentials described in section 3.6. This is described in figure 4.5.

[2]The diameter of a connected graph is the length of a longest shortest path between two vertices.

[3]This "plus 1" is caused by the way the algorithm works: the first iteration only in effect lets all leaf emit their messages.

Variables (in the programming sense) of the graph:

- Each vertex v_i has densities p_i^- and p_i;
- Each edge e_{ij} has messages m_{ij}^- and m_{ij} for direction i to j, and m_{ji}^- and m_{ji} for direction j to i.

For each time-step,

1. Generate p_i^- for all leaf vertices v_i from their sensor modules using p_i as hints.
2. Set $p_i = \zeta$ for all vertices v_i.
3. Set $m_{ij}^- = \zeta$ and $m_{ij} = \zeta$ for all edges e_{ij}.
4. Repeat for L times, in each iteration:
   (a) Set $p_i^- = p_i$ and initialize $p_i = \zeta$ for all non-leaf vertices v_i.
   (b) Set $m_{ij}^- = m_{ij}$ and initialize $m_{ij} = \zeta$ for all edges e_{ij}
   (c) For all vertices v_i, do the following:
      i. Sort all incoming edges e_{ji} in the order of
         A. with linear potential from leaf vertex;
         B. with non-linear potential from leaf vertex;
         C. from non-leaf vertex.

         Call the sorted incoming edges e_k, $k = 1, \ldots, K$, where K is the degree of the v_i, and their corresponding messages m_k and potentials ψ_k.
      ii. Let $M_0 = 1$ and for $k = 1, \ldots, K$ do
         A. Compute $M_k = M_{k-1} m_k^-$.
         B. Merge M_k as in figure 4.3.
         C. Prune the number of Gaussian terms in M_k as in figure 4.4. .
      iii. Let $p_i = M_K$.
      iv. if v_i is leaf, let $m_{ij} = \int \psi_{ij} p_i^- \, \mathrm{d}x_i$ for all edges e_{ij};
      v. if v_i is not leaf, do
         A. Compute $m_{ij} = \int \psi_{ij} (M_{k-1} \prod_{s=k+1}^{K} m_s^-) \, \mathrm{d}x_i$, where e_{ji} is e_k.
         B. Merge m_{ij} as in figure 4.3.
         C. Prune the number of Gaussian terms in m_{ij} as in figure 4.4.
         D. Adjust the null weight of m_{ij} as in figure 4.6.

Figure 4.2: The Algorithm of Running the Graph for Each Time-step

For a collection of terms $\{(w_i, G_i),\ i = 1, \ldots, n\}$, where G_i can be either a Gaussian or the null hypothesis ζ, the objective is to merge the similar terms according to a given precision D^*_{th}.
In the following algorithm, the merging is done "in-place". $\{(w_i, G_i),\ i = 1, \ldots, q-1\}$ is always the set of merged hypotheses at the end of loop in step 2a, while $\{(w_i, G_i),\ i = q, \ldots, p\}$ is the set of yet-to-be-merged hypotheses.

1. Let $q = 1$.
2. For $p = 1, \ldots, n$,
   (a) Find the minimum r that $1 \leq r < p$, and $D^*(G_r, G_p) < D^*_{th}$ where D^* is defined in equations (3.8.8) and (3.8.10).
   - If such r exists, let $w_r = w_r + w_p$;
   - Otherwise,
     i. Let $G_q = G_p$.
     ii. Let $w_q = w_p$.
     iii. Let $q = q + 1$.
3. Return the collection $\{(w_i, G_i),\ i = 1, \ldots, q-1\}$.

Figure 4.3: The Algorithm of Merging Gaussian Terms

For a collection of terms $\{(w_i, G_i),\ i = 1, \ldots, n\}$, where G_i can be either a Gaussian or the null hypothesis ζ, the objective is to select at most N terms carrying at most P of original weights:

1. Sort the terms according their weights in the descending order: $\{(w'_i, G'_i),\ i = 1, \ldots, n\}$ that $\forall_{i<j} : w'_i \geq w'_i$.
2. Normalize the weights: $w''_i = \left(\sum_{i=1}^{n} w'_i\right)^{-1} w'_i$
3. Search for the maximum k that $k \leq N$ and $\sum_{i=1}^{k} w''_i \leq P$
4. Return the normalized collection $\{(w^*_i, G'_i),\ i = 1, \ldots, n\}$, where $w^*_i = \left(\sum_{i=1}^{k} w''_i\right)^{-1} w''_i$.

Figure 4.4: The Algorithm of Pruning the Number of Gaussian Terms

Two kinds of operations are involved in handling delayed linearization of potentials (all numbers refer to the main time-step algorithm in figure 4.2):

- When doing last multiplication and integration in 4(c)vA, $m_{ij} = \int \psi_{ij} f_{\text{input}} \, \mathrm{d}x_i$
  - if the potential ψ_{ij} is linear, process normally;
  - otherwise just store the f_{input} as m_{ij}.
- When doing multiplications between messages in 4(c)iiA and 4(c)vA, which is actually done in the accumulative way, $f_{\text{output}} = m_{ij} f_{\text{input}}$,
  - if the potential ψ_{ij} is linear, process normally;
  - otherwise, suppose $f_{\text{input}} = \sum_s w_s G_s$ and $m_{ij} = \sum_k w'_k G'_k$, where G_s and G'_k are Gaussians with respective centers c_s and c'_k:
    1. For all s and k linearize ψ_{ij} at $x_i = c_s$ and $y_i = c'_k$. Denote the result as ψ'_{sk}.
    2. Let $f_{\text{output}} = \sum_s \sum_k w_s w'_k G_s \int \psi'_{sk} G_k \, \mathrm{d}x_i$

  The sorting in 4(c)i ensures that as long as at least one of the potentials is linear, the multiplication of them can be done.

Figure 4.5: Handling of Delayed Linearization of Potentials

For a collection of terms $\{(w_i, G_i),\ i = 1, \ldots, n\}$, where G_i can be either a Gaussian or the null hypothesis ζ, the objective is to adjust the weight of null hypothesis.

1. Calculate $w_a = \sum_i w_i$ for all $G_i \neq \zeta$.
2. Calculate $w_b = \sum_i w_i$ for all $G_i = \zeta$.
3. Let $w'_i = (w_a + w_b) w_a^{-1} (w_a + 2w_b)^{-1} w_i$ for all $G_i \neq \zeta$.
4. Let $w'_i = (w_a + 2w_b)^{-1} w_i$ for all $G_i = \zeta$.
5. Return the collection $\{(w'_i, G_i),\ i = 1, \ldots, n\}$.

Figure 4.6: Adjusting the Weights of the Null Hypotheses

1. Set up the graph structure.
2. Initialize image-level sensors.
3. Repeat the following:
   (a) Acquire a new image or set of images.
   (b) Run the graph for this time-step.
   (c) Retrieve the p_i for all nodes of interest v_i.
   (d) Compute the estimators $\hat{x}_i$ from p_i, for example, by equation 3.2.4.

Figure 4.7: Structure of a Typical Application

The last detail needing more explanation is the adjustment of the weight of null-hypothesis for each message computed as described in section 3.7.2. This happens before each message is sent at step 4(c)vD of the algorithm in figure 4.2. Figure 4.6 shows the algorithm for adjusting null weights.

Finally, the application would run through many time-steps, turning the stream of input into stream of output. The intended way of doing this is described in figure 4.7.

4.2 The Implementation

A software system has been developed for the purpose of implementing and verifying ideas and algorithms proposed in this thesis. With that in mind, the design emphasizes the following properties of the system, in their order:

1. Flexible structure to allow fast prototyping. The ideas and algorithms themselves need to be tested and refined in the process.
2. Extensible modules. The framework itself is supposed to work with all kinds of components and constraints.
3. Reuse of existing code when suitable. In fact, this system relies on
   - the XVision2[20] library, which is developed by Computational Interaction and Robotics Lab at Johns Hopkins University, for image data structures and input and output;
   - the LAPACK[2] and BLAS[36, 9] routines for linear algebra operations.
4. Reusability of the code. Many implemented image processing and vision functionalities should be beneficial to other projects on these areas.

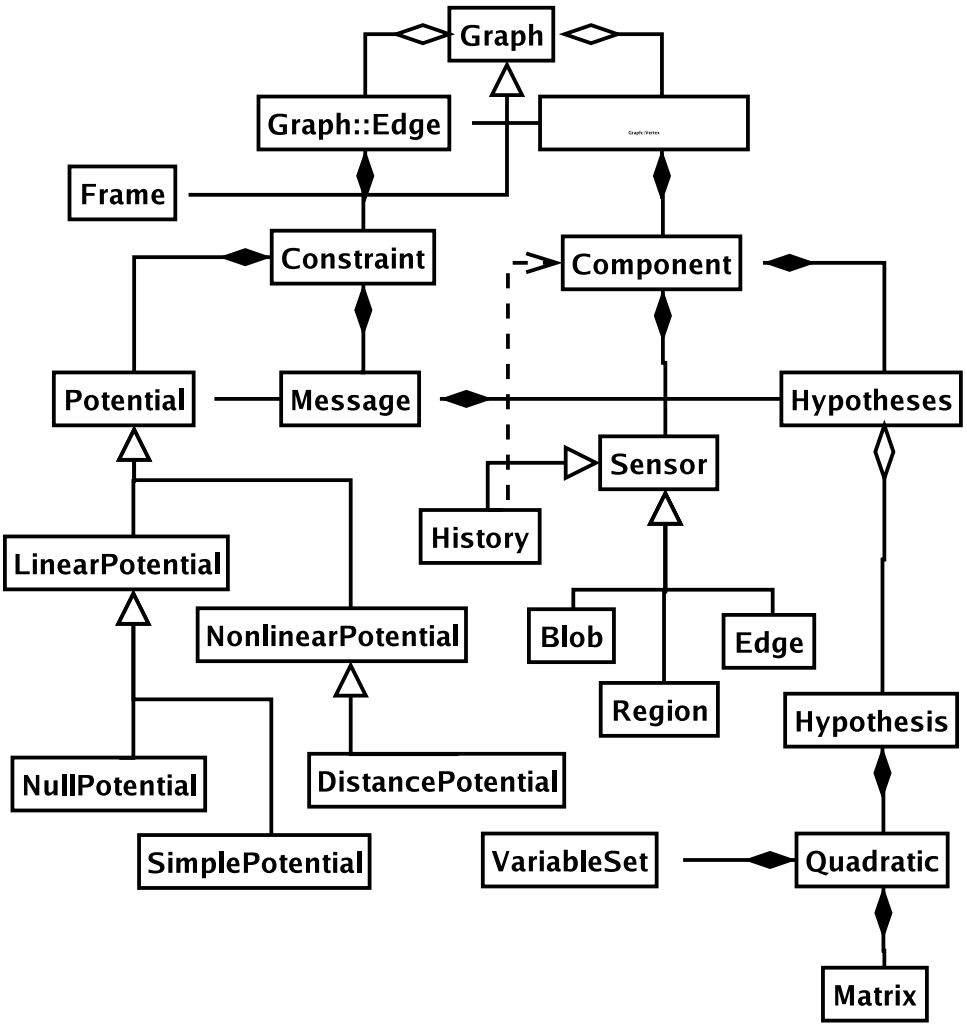
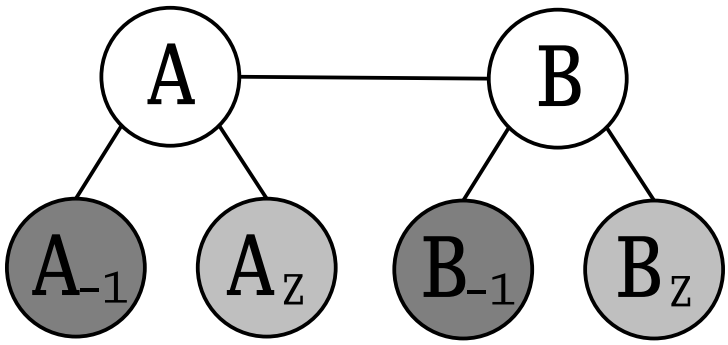


Figure 4.8: Simplified Class Diagram of the System



The graphic model for the tracking target in simulation. A and B are target parts; A_{-1} and B_{-1} are history components, which take values of their corresponding part component of the previous time step. A_Z and B_Z are the observations. The edge between A and B is a distance constraint; all other edges are overlapping constraints.

Figure 4.9: The Graph for Simulated Experiment

The system is written in C++. Figure (4.8) shows a much simplified version of the class diagram of the system in UML (Unified Modeling Language). For a detailed description of the software system, please refer to the source code. Appendix C lists a sample of the actual source code used in the experiments.

4.3 Simulation

The purpose of the simulation is to verify the theory in chapter 3 and its corresponding algorithm in section 4.1. The observations are simulated: there is no actual stream of input video; otherwise the experiment is as same as an actual application.

4.3.1 Setup

In our settings, the 2-D tracking target is composed of two identical and observable parts A and B, softly constrained by an unobservable bar with fixed length. The target is shifting and (in-plane) rotating over the time, but the application has no knowledge of its dynamics. The application has the knowledge of the constraint, as well as the initial position of target parts A and B at time $t = 0$. Also in the scene are random distractors, all identical to A and B.

Mathematically, the state vector $\mathbf{x}$ has components (x, y). All $\mathbf{x}_i^*$, $i = 1, \ldots, n$ are distractors except for $\mathbf{x}_a^*$ is the true position of A and $\mathbf{x}_b^*$ is the true position of B.

The graphic model for the target described above is shown in figure 4.9. A and B are target parts constrained by their fixed Euclidean distance. Each of them connects a history component,

respectively A_{-1} and B_{-1}, and an observation component, respectively A_Z and B_Z. Overlapping constraints are between the parts and leaves.

In addition to the graph, the model needs to be described by its potential functions. For the observations A_Z and B_Z,

$$\psi_{A_Z}(\mathbf{z}_A) = \begin{cases} w_a G(\mathbf{z}_A - \mathbf{x}_a^*, \sigma_z) + w_0\zeta & t = 0 \\ \sum_{i=0}^{n} w_i G(\mathbf{z}_A - \mathbf{x}_i^*, \sigma_z) + w_0\zeta & t > 0 \end{cases} \tag{4.3.1a}$$

$$\psi_{B_Z}(\mathbf{z}_B) = \begin{cases} w_b G(\mathbf{z}_B - \mathbf{x}_b^*; \sigma_z) + w_0\zeta & t = 0 \\ \sum_{i=0}^{n} w_i G(\mathbf{z}_B - \mathbf{x}_i^*; \sigma_z) + w_0\zeta & t > 0 \end{cases} \tag{4.3.1b}$$

where $G(\mathbf{x}; \sigma)$ is the Gaussian:

$$G(\mathbf{x}; \sigma) = \frac{1}{2\pi\sigma^2} \exp\left\{-\frac{1}{2\sigma^2}\mathbf{x}^T\mathbf{x}\right\} \tag{4.3.2}$$

The potentials between parts and observations are

$$\psi_{AA_Z}(\mathbf{x}_A, \mathbf{z}_A) = G(\mathbf{x}_A - \mathbf{z}_A; \sigma_x) \tag{4.3.3a}$$

$$\psi_{BB_Z}(\mathbf{x}_B, \mathbf{z}_B) = G(\mathbf{x}_B - \mathbf{z}_B; \sigma_x) \tag{4.3.3b}$$

and the potentials between parts and observations are

$$\psi_{AA_{-1}}(\mathbf{x}_A, \mathbf{y}_A) = G(\mathbf{x}_A - \mathbf{y}_A; \sigma_y) \tag{4.3.4a}$$

$$\psi_{BB_{-1}}(\mathbf{x}_B, \mathbf{y}_B) = G(\mathbf{x}_B - \mathbf{y}_B; \sigma_y) \tag{4.3.4b}$$

Finally, the potential between two parts is

$$\psi_{AB}(\mathbf{x}_A, \mathbf{x}_B) = \exp\left\{-\frac{1}{2\sigma_d^2}\left(\sqrt{(\mathbf{x}_A - \mathbf{x}_B)^T(\mathbf{x}_A - \mathbf{x}_B)} - d\right)^2\right\} \tag{4.3.5}$$

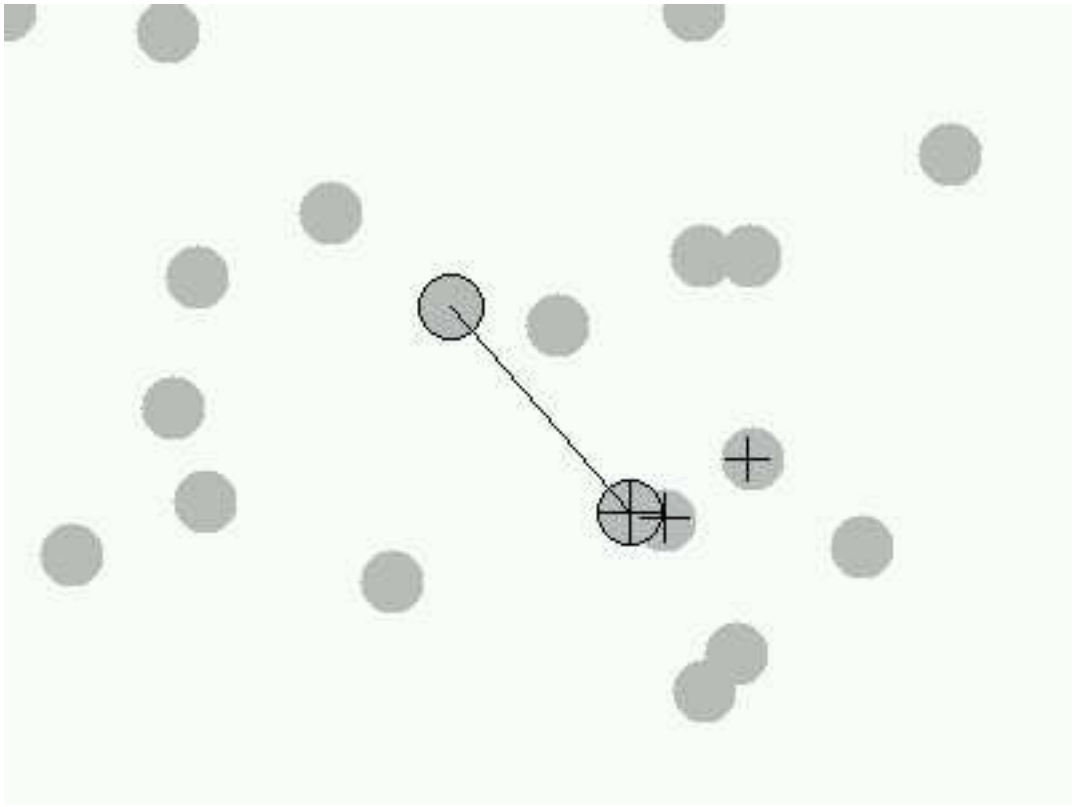
From section 3.6, we know their linearized forms at $\mathbf{x}_A = \mathbf{u}_A$, $\mathbf{x}_B = \mathbf{u}_B$ are

$$\psi'_{ij}(\mathbf{x}_A, \mathbf{x}_B; \mathbf{u}_A, \mathbf{u}_B) = \exp\left\{-\frac{1}{2\sigma_d^2}\left(\frac{2(\mathbf{u}_A - \mathbf{u}_B)^T}{\sqrt{(\mathbf{u}_A - \mathbf{u}_B)^T(\mathbf{u}_A - \mathbf{u}_B)}}(\mathbf{x}_A - \mathbf{x}_B) - \sqrt{(\mathbf{u}_A - \mathbf{u}_B)^T(\mathbf{u}_A - \mathbf{u}_B)} - d\right)^2\right\} \tag{4.3.6}$$

In the simulation, we take $d = 30$, $\sigma_x = \sigma_y = \sigma_z = \sigma_d = 1$, $w_i = 1$ for $i = 1, \ldots, n$, and $w_0 = 0.1$.

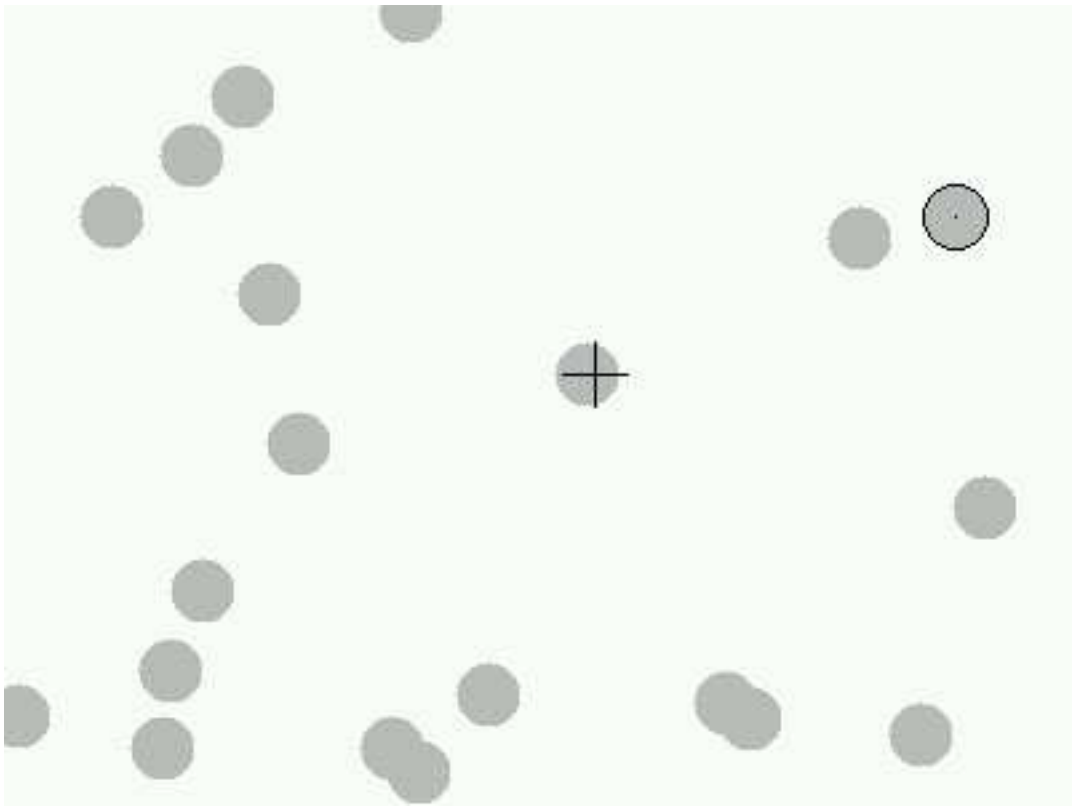
4.3.2 Results

Figure 4.10 shows a typical tracking result for part A in the simulation. In the figure, the center of the largest cross which indicates the estimate from the best hypothesis is almost coincident



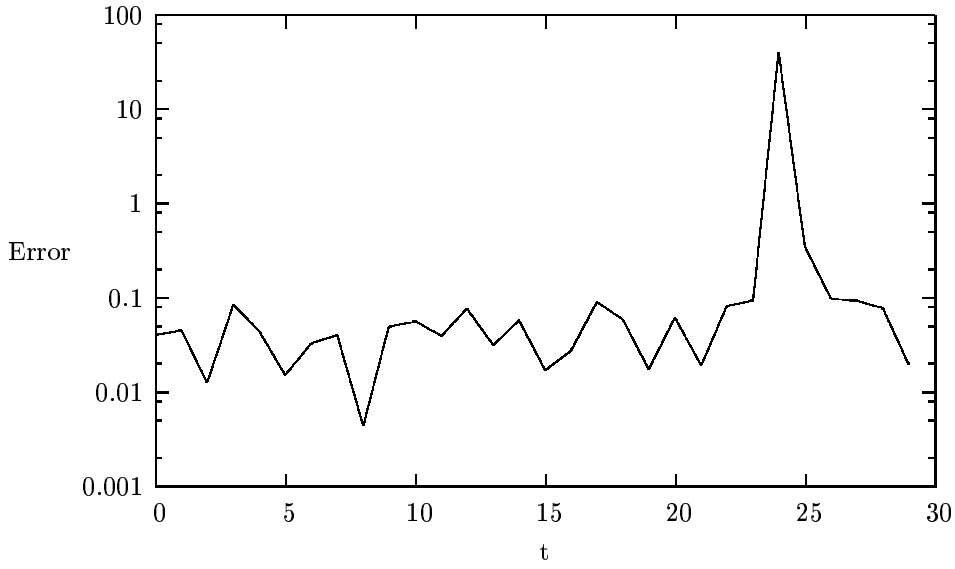
At $t = 10$, the tracking results for part A. Distractors are shown by unbounded grey circles and true parts are shown by bounded grey circles with a line connecting them. The centers of the circles are the actual positions while the sizes of the circles are only for illustration purpose. Each hypothesis as the tracking result of part A is shown by a cross, with the center of the cross indicating the position and the size of the cross linear with the logarithm of its weight. The tracking result of part B is not shown.

Figure 4.10: Result from a Typical Frame in Simulation



At $t = 24$, the tracking results for part A. Distractors are shown by unbounded grey circles and part B is shown by a bounded grey circle. Part A is actually missing from the scene. The centers of the circles are the actual positions while the sizes of the circles are only for illustration purpose. The hypothesis as the tracking result of part A is shown by a cross. The tracking result of part B is not shown.

Figure 4.11: Result from Missing Object Part in Simulation



Tracking error ($|\hat{\mathbf{x}} - \mathbf{x}|$) for the estimate of the best hypothesis of part A. The error is plotted in logarithmic scale.

Figure 4.12: Tracking Error for Part A in Simulation

with the center of one of the bounded circle which indicates the true position of the part. Nearby distractors are all covered by hypotheses as well but with much smaller weights.

Figure 4.11 shows the tracking result for part A for a scene from which it is actually missing. Obviously, one of the nearby distractors is mistakenly taken as part A. However, when the true part A emerge in the next frame, the tracker catches it up immediately. This shows the robustness of the tracker.

Figure 4.12 shows the tracking error for part A over time. The peak at $t = 24$ is caused by missing of the part from the scene.

On the other hand, figure 4.13 shows the relative weight of the best hypothesis of part A over time. The dip around $t = 11$ is caused by very close distractors as in figure 4.10. Another dip at $t = 24$ is caused by missing of the part as in figure 4.11.

It is worth noting that the simulation performs almost identically except for the first few frames even if the tracker is not initialized, i.e. there is no special case for $t = 0$ in equation (4.3.1). This is not surprising given the bottom-up approach of the framework. This shows the framework's auto-reinitialization potential.

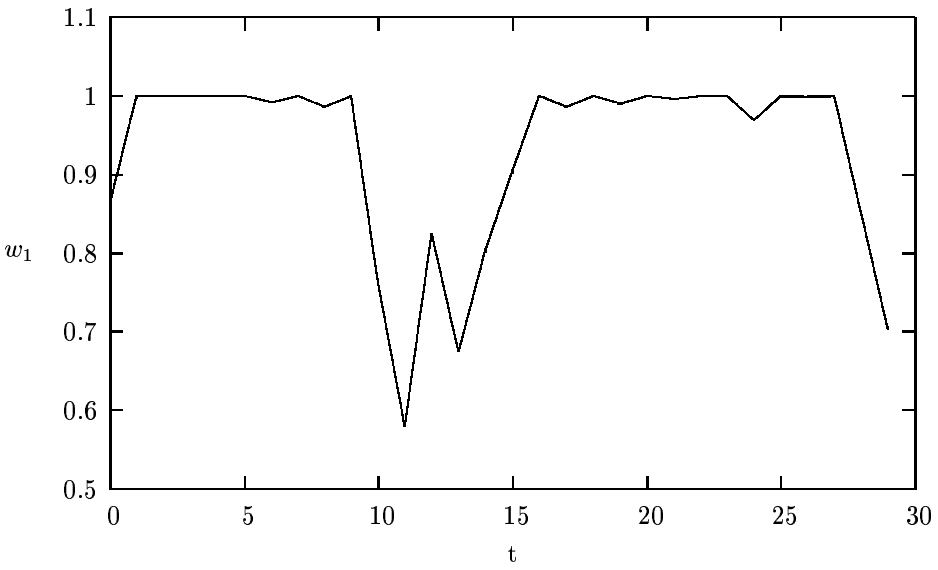


Figure 4.13: Weight of the Best Hypothesis for Part A in Simulation

Chapter 5

Experiments

5.1 Setup

In this chapter, we focus on an experiment that uses real-world data. Specifically, we are to track the 2-D positions of both eyes and the mouth of a human face on the image.[1] All three of target parts (the eyes and the mouth) are tracked as heterogeneous regions under 2-D translations by the methods described in chapter 2. Figure 5.1 shows the first image of the video sequence. Figure 5.2 shows the three image regions cut from this image, which are used as templates for tracking the target parts.

Besides the cluttered background, the image sequence is not always as "friendly" as the image shown in figure 5.1. For example, figure 5.3 lists a few difficult scenes in the video sequence. For them, the deformation of the eyes and the mouth in the images is obviously beyond the assumption of simple transformations made in chapter 2, so the tracker would have a hard time to track the targets. Nevertheless, even the tracker is completely lost on these scenes, its performance can be still measured on how it recovers from these occasions. In other words, by not using perfect low-level detectors on target regions, the robustness from combining these detectors and exploring the relationship between the target parts can be shown.

For the same purpose, the constraints between the target parts are simply set to be distance constraints. For the images in the video sequence where the face rotates out-of-plane, the distances between the parts changes, however, since the constraints have inherent flexibility, we expect the construct would still work to a certain degree.

The graphical model for the tracking target is shown in figure 5.4. L, R, and M are target parts for the left eye, the right eye, and the mouth, respectively. They are all connected with

[1] If both the 3-D model of relative positions of the eyes and the mouth on the face and the camera parameters are known, the 3-D position and orientation of the face can be deduced from these 2-D positions. However, since 2-D tracking already serves the purpose of verifying and evaluating the component-based tracking framework, we are not going to explore the 3-D implication in this chapter.

Figure 5.1: An Image from the Video Sequence

Figure 5.2: Templates of Tracked Regions

Figure 5.3: Difficult Scenes in the Video Sequence

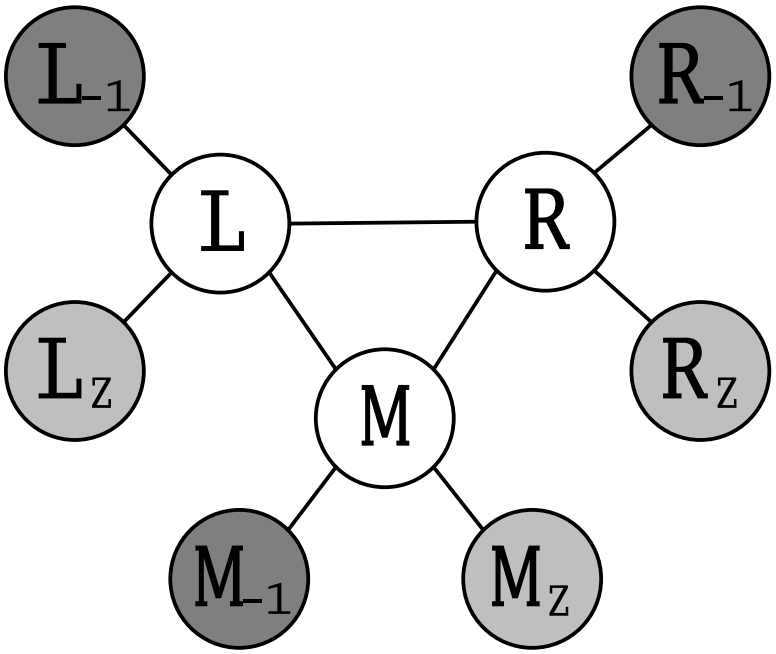


Figure 5.4: The graphic model for the tracking target in experiment. L, R, and M are target parts for the left eye, the right eye, and the mouth, respectively; L_{-1}, R_{-1}, and $M_{-1}$are history components; L_Z, R_Z and M_Z are the observations. The edge between L and R, the edge between L and M, and the edge between R and M are distance constraints; all other edges are overlapping constraints.

each other by distance constraints. Each of them connects a history component and an observation component by overlapping constraints.

All the node variables here are 2-D vectors of x and y coordinates. The position of a region is defined as the center of the region. Denote the estimates of new target parts positions for the eyes and the mouth to be $\hat{\mathbf{z}}_{L,i}$ $(i = 1 \ldots n_L)$, $\hat{\mathbf{z}}_{R,i}$ $(i = 1 \ldots n_R)$, and $\hat{\mathbf{z}}_{M,i}$ $(i = 1 \ldots n_M)$, respectively, leaf potential functions for the observations are

$$\psi_{L_Z}(\mathbf{z}_L) = \sum_i^{n_L} w_{L,i} G(\mathbf{z}_L - \hat{\mathbf{z}}_{L,i}; \sigma_z) + w_0 \zeta \tag{5.1.1a}$$

$$\psi_{R_Z}(\mathbf{z}_R) = \sum_i^{n_R} w_{R,i} G(\mathbf{z}_R - \hat{\mathbf{z}}_{R,i}; \sigma_z) + w_0 \zeta \tag{5.1.1b}$$

$$\psi_{M_Z}(\mathbf{z}_M) = \sum_i^{n_M} w_{M,i} G(\mathbf{z}_M - \hat{\mathbf{z}}_{M,i}; \sigma_z) + w_0 \zeta \tag{5.1.1c}$$

where $G(\mathbf{x}; \sigma)$ is the Gaussian:

$$G(\mathbf{x}; \sigma) = \frac{1}{2\pi\sigma^2} \exp\left\{-\frac{1}{2\sigma^2}\mathbf{x}^T\mathbf{x}\right\} \tag{5.1.2}$$

Denote $\mathbf{h}$ for image pixel locations within the template images relative to the image centers and $T(\mathbf{h}; \mathbf{x})$ for the geometry transformation function. Since we use 2-D translation, the latter actually is

$$T(\mathbf{h}; \mathbf{x}) = \mathbf{h} + \mathbf{x}; \tag{5.1.3}$$

The weights in equations (5.1.1) are given as in section 3.4.1:

$$w_{L,i} = \frac{1}{\sqrt{2\pi}\sigma_s} \exp\left\{-\frac{1}{2\pi\sigma_s^2 s_L} \sum_{\mathbf{h}} \left(I(T(\mathbf{h}; \hat{\mathbf{z}}_{L,i})) - I_L(\mathbf{h})\right)^2\right\} \tag{5.1.4a}$$

$$w_{R,i} = \frac{1}{\sqrt{2\pi}\sigma_s} \exp\left\{-\frac{1}{2\pi\sigma_s^2 s_R} \sum_{\mathbf{h}} \left(I(T(\mathbf{h}; \hat{\mathbf{z}}_{R,i})) - I_L(\mathbf{h})\right)^2\right\} \tag{5.1.4b}$$

$$w_{M,i} = \frac{1}{\sqrt{2\pi}\sigma_s} \exp\left\{-\frac{1}{2\pi\sigma_s^2 s_M} \sum_{\mathbf{h}} \left(I(T(\mathbf{h}; \hat{\mathbf{z}}_{M,i})) - I_L(\mathbf{h})\right)^2\right\} \tag{5.1.4c}$$

where I is the input image at the current time-step, I_L, I_R, and I_M are image templates and s_L, s_R, and s_M are their size (area in pixels). All images are grey-level: only the intensity of the original images from the video sequence is used.

The potentials between parts and observations are

$$\psi_{LL_Z}(\mathbf{x}_L, \mathbf{z}_L) = G(\mathbf{x}_L - \mathbf{z}_L; \sigma_x) \tag{5.1.5a}$$

$$\psi_{RR_Z}(\mathbf{x}_R, \mathbf{z}_R) = G(\mathbf{x}_R - \mathbf{z}_R; \sigma_x) \tag{5.1.5b}$$

$$\psi_{MM_Z}(\mathbf{x}_M, \mathbf{z}_M) = G(\mathbf{x}_M - \mathbf{z}_M; \sigma_x) \tag{5.1.5c}$$

and the potentials between parts and histories are

$$\psi_{LL_{-1}}(\mathbf{x}_L, \mathbf{y}_L) = G(\mathbf{x}_L - \mathbf{y}_L; \sigma_y) \tag{5.1.6a}$$

$$\psi_{RR_{-1}}(\mathbf{x}_R, \mathbf{y}_R) = G(\mathbf{x}_R - \mathbf{y}_R; \sigma_y) \tag{5.1.6b}$$

$$\psi_{MM_{-1}}(\mathbf{x}_M, \mathbf{y}_M) = G(\mathbf{x}_M - \mathbf{y}_M; \sigma_y) \tag{5.1.6c}$$

Finally, the potential between parts are

$$\psi_{LR}(\mathbf{x}_L, \mathbf{x}_R) = \exp\left\{-\frac{1}{2\sigma_d^2}\left(\sqrt{(\mathbf{x}_L - \mathbf{x}_R)^T(\mathbf{x}_L - \mathbf{x}_R)} - d_{LR}\right)^2\right\} \tag{5.1.7a}$$

$$\psi_{LM}(\mathbf{x}_L, \mathbf{x}_M) = \exp\left\{-\frac{1}{2\sigma_d^2}\left(\sqrt{(\mathbf{x}_L - \mathbf{x}_M)^T(\mathbf{x}_L - \mathbf{x}_M)} - d_{LM}\right)^2\right\} \tag{5.1.7b}$$

$$\psi_{RM}(\mathbf{x}_R, \mathbf{x}_M) = \exp\left\{-\frac{1}{2\sigma_d^2}\left(\sqrt{(\mathbf{x}_R - \mathbf{x}_M)^T(\mathbf{x}_R - \mathbf{x}_M)} - d_{RM}\right)^2\right\} \tag{5.1.7c}$$

From section 3.6, we know their linearized forms at $\mathbf{x}_L = \mathbf{u}_L$, $\mathbf{x}_R = \mathbf{u}_R$, $\mathbf{x}_M = \mathbf{u}_M$ are

$$\psi'_{ij}(\mathbf{x}_L, \mathbf{x}_R; \mathbf{u}_L, \mathbf{u}_R) = \exp\left\{-\frac{1}{2\sigma_d^2}\left(\frac{2(\mathbf{u}_L - \mathbf{u}_R)^T}{\sqrt{(\mathbf{u}_L - \mathbf{u}_R)^T(\mathbf{u}_L - \mathbf{u}_R)}}(\mathbf{x}_L - \mathbf{x}_R) - \sqrt{(\mathbf{u}_L - \mathbf{u}_R)^T(\mathbf{u}_L - \mathbf{u}_R)} - d_{LR}\right)^2\right\} \tag{5.1.8}$$

and $\psi'_{ij}(\mathbf{x}_L, \mathbf{x}_M; \mathbf{u}_L, \mathbf{u}_M)$ and $\psi'_{ij}(\mathbf{x}_R, \mathbf{x}_M; \mathbf{u}_R, \mathbf{u}_M)$ are similar.

In the experiment, we take $\sigma_x = \sigma_y = \sigma_z = 1$, $\sigma_d = \sqrt{10}$, $\sigma_s = 10$. These values should be close to the actual variances in the video sequence. $w_0 = 0.01$. d_{LR}, d_{LM}, and d_{RM} are their corresponding distances (between left and right eye, between the left eye and the mouth, and between the right eye and the mouth, respectively) in the first frame of the video sequence.

5.2 Particle Filtering

Before comparing the result of our component-based tracking to that of particle filtering, the technical detail of the latter is briefly introduced in this section.

Particle filtering, or Condensation, assumes a Markovian state and observation model as same as that of Kalman filtering: state $\mathbf{x}_t$ only depends on $\mathbf{x}_{t-1}$ and observation $\mathbf{z}_t$ only depends on $\mathbf{x}_t$. However, unlike Kalman filtering, there is no restriction on the linearity of either the state equation between $\mathbf{x}_t$ and $\mathbf{x}_{t-1}$ or the observation equation between $\mathbf{z}_t$ and $\mathbf{x}_t$. Instead, the only additional assumption the particle filter relies on is that the state dynamic $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ can be sampled and the likelihood $p(\mathbf{z}_t|\mathbf{x}_t)$ can be evaluated. These are very weak restrictions that can be satisfied in almost all cases.

Each of the particles is a state vector $\mathbf{x}^{(i)}$ associated with a weight $w^{(i)}$. The algorithm starts with an initial set of particles $\{(\mathbf{x}_0^{(i)}, w_0^{(i)}),\ i = 1, \ldots, n\}$, where n is the number of particles.

For each time-step t, a new set of particle $\{(\mathbf{x}_t^{(i)}, w_0^{(i)}),\ i = 1, \ldots, n\}$ is generated from the set of particles $\{(\mathbf{x}_{t-1}^{(i)}, w_0^{(i)})\},\ i = 1, \ldots, n\}$ of the previous time-step $t-1$:

1. For all $j = 1, \ldots, n$, independently generate $\mathbf{x}_t^{(j)}$ and $\tilde{w}_t^{(j)}$ by the following procedure:
   (a) Draw $\tilde{\mathbf{x}}_{t-1}^{(j)}$ as a sample from probability density function $s(\mathbf{x}) = \sum_{i=1}^{n} w_i \delta(\mathbf{x} - \mathbf{x}_{t-1}^{(i)})$ where δ is the Dirac delta function. In other words, with probability w_i let $\tilde{\mathbf{x}}_{t-1}^{(j)} = \mathbf{x}_{t-1}^{(i)}$.
   (b) Draw $\mathbf{x}_t^{(j)}$ as a sample from probability density function $p(\mathbf{x}_t | \mathbf{x}_{t-1} = \tilde{\mathbf{x}}_{t-1}^{(j)})$.
   (c) Evaluate $\tilde{w}_t^{(j)} = p(\mathbf{z}_t | \mathbf{x}_t = \mathbf{x}_t^{(j)})$.
2. For all $j = 1, \ldots, n$, let $w_t^{(j)} = \left(\sum_{i=1}^{n} \tilde{w}_t^{(i)} \right)^{-1} \tilde{w}_t^{(j)}$

It can be proved that with this algorithm the set of particles asymptotically converges to the posterior probability of the state as $n \to \infty$:

$$\{(\mathbf{x}_t^{(i)}, w_0^{(i)})\} \to p(\mathbf{x}_t | \mathbf{Z}_t) \tag{5.2.1}$$

where $\mathbf{Z}_t = \{\mathbf{z}_\tau, \tau = 1, \ldots, t\}$. Let the estimator be the weighted average of the particles:

$$\hat{\mathbf{x}}_t = \sum_{i=1}^{n} w_t^{(i)} \mathbf{x}_t^{(i)} \tag{5.2.2}$$

then

$$\lim_{n \to \infty} \hat{\mathbf{x}}_t = E\left[p(\mathbf{x}_t | \mathbf{Z}_t) \right] \tag{5.2.3}$$

In our experiment, the forms of both the state dynamic and the likelihood function are invariant to time-step t, so in the rest of the section the subscript t is dropped from the equations. Instead, the subscripts L, R, and M are used as in the previous section to identify the left eye, the right eye, and the mouth, respectively, and $\mathbf{y}$ is used for the state vector for the previous time-step.

The state vector for the experiment consists of three sub-vectors $(\mathbf{x}_L, \mathbf{x}_R, \mathbf{x}_M)$. The state dynamic is on par with equations (5.1.6):

$$p(\mathbf{x}_L, \mathbf{x}_R, \mathbf{x}_M | \mathbf{y}_L, \mathbf{y}_R, \mathbf{y}_M) = G(\mathbf{x}_L - \mathbf{y}_L; \sigma_y') G(\mathbf{x}_R - \mathbf{y}_R; \sigma_y') G(\mathbf{x}_M - \mathbf{y}_M; \sigma_y') \tag{5.2.4}$$

and the likelihood is on par with equations (5.1.4) and (5.1.7):

$$p(I | \mathbf{x}_L, \mathbf{x}_R, \mathbf{x}_M) = W_L(I, \mathbf{x}_L) W_R(I, \mathbf{x}_R) W_M(I, \mathbf{x}_M) D_{LR}(\mathbf{x}_L, \mathbf{x}_R) D_{LM}(\mathbf{x}_L, \mathbf{x}_M) D_{RM}(\mathbf{x}_R, \mathbf{x}_M) \tag{5.2.5}$$

where

$$W_L(I, \mathbf{x}_L) = \frac{1}{\sqrt{2\pi}\sigma_s} \exp\left\{ -\frac{1}{2\sigma_s^2 s_L} \sum_{\mathbf{h}} \left(I(T(\mathbf{h}; \mathbf{x}_L)) - I_L(\mathbf{h}) \right)^2 \right\} \tag{5.2.6a}$$

$$W_R(I, \mathbf{x}_R) = \frac{1}{\sqrt{2\pi}\sigma_s} \exp\left\{ -\frac{1}{2\sigma_s^2 s_R} \sum_{\mathbf{h}} \left(I(T(\mathbf{h}; \mathbf{x}_R)) - I_R(\mathbf{h}) \right)^2 \right\} \tag{5.2.6b}$$

$$W_M(I, \mathbf{x}_M) = \frac{1}{\sqrt{2\pi}\sigma_s} \exp\left\{ -\frac{1}{2\sigma_s^2 s_M} \sum_{\mathbf{h}} \left(I(T(\mathbf{h}; \mathbf{x}_M)) - I_M(\mathbf{h}) \right)^2 \right\} \tag{5.2.6c}$$

and

$$D_{LR}(\mathbf{x}_L, \mathbf{x}_R) = \frac{1}{2\pi\sigma_d^2} \exp\left\{ -\frac{1}{2\sigma_d^2} \left(\sqrt{(\mathbf{x}_L - \mathbf{x}_R)^T(\mathbf{x}_L - \mathbf{x}_R)} - d_{LR} \right)^2 \right\} \tag{5.2.7a}$$

$$D_{LM}(\mathbf{x}_L, \mathbf{x}_M) = \frac{1}{2\pi\sigma_d^2} \exp\left\{ -\frac{1}{2\sigma_d^2} \left(\sqrt{(\mathbf{x}_L - \mathbf{x}_M)^T(\mathbf{x}_L - \mathbf{x}_M)} - d_{LM} \right)^2 \right\} \tag{5.2.7b}$$

$$D_{RM}(\mathbf{x}_R, \mathbf{x}_M) = \frac{1}{2\pi\sigma_d^2} \exp\left\{ -\frac{1}{2\sigma_d^2} \left(\sqrt{(\mathbf{x}_R - \mathbf{x}_M)^T(\mathbf{x}_R - \mathbf{x}_M)} - d_{RM} \right)^2 \right\} \tag{5.2.7c}$$

Here all symbols not defined otherwise are as same as in section 5.1. Also, the same constant parameters take the same value for both the component-based tracking and particle filtering. Since there is no σ_x or σ_z for the particle filtering setup, we let $\sigma_y' = 2$.

5.3 Results

To validate and evaluate the performance of our component-based tracking, and to compare it against particle filtering, some form of ground truth must be established. We manually labeled centers of both eyes and the mouth in the video sequence as the "true" values of the state vectors. In this section these values are denoted by $\mathbf{x}_L^*$, $\mathbf{x}_R^*$, and $\mathbf{x}_M^*$. We estimate the accuracy of these "true" values to be within about 2 pixels. The estimation error from one of the tracking methods is defined as[2]

$$e = \frac{1}{\sqrt{6}} \sqrt{(\hat{\mathbf{x}}_L - \mathbf{x}_L^*)^T(\hat{\mathbf{x}}_L - \mathbf{x}_L^*) + (\hat{\mathbf{x}}_R - \mathbf{x}_R^*)^T(\hat{\mathbf{x}}_R - \mathbf{x}_R^*) + (\hat{\mathbf{x}}_M - \mathbf{x}_M^*)^T(\hat{\mathbf{x}}_M - \mathbf{x}_M^*)} \tag{5.3.1}$$

where $\hat{\mathbf{x}}_L$, $\hat{\mathbf{x}}_R$, and $\hat{\mathbf{x}}_M$ are estimates from the tracking method. All estimates are expectations as in equation (3.7.27) or (5.2.2).

Firstly we are to verify the effectiveness of the constraints in equations (5.1.7). Figure 5.5 shows the estimation errors for component-based tracking with and without constraint. Note that the actual distance between the target parts on the image in in the range of 80 to 100 pixels, so an error greater than 20 pixels can be thought as a lost track. From the plots the constraints do help the tracker to avoid losing track in many periods of time. For example, the subject begins to turn the head to his right (left in the image) with the eyes closed around frame 140, in which case the constraints prevent the region tracker for the right eye (left in the image) from losing track and slipping to the left of the image, as demonstrated in figure 5.6.

The next step is to compare our component-based tracking against particle filtering. Figure 5.7 shows the estimation error of the two tracking methods. Figure 5.8 reveals more details by showing only the first 100 frames. With more particles, the particle filter performs better, but generally still not as good as component-based tracking for this video sequence; For instance, figure 5.9 shows the tracking result for frame 440, which is in one of its error peaks. On the other hand, the

[2]The factor $\frac{1}{\sqrt{6}}$ in the equation is because of the number of total dimensions in all state vectors is 6.

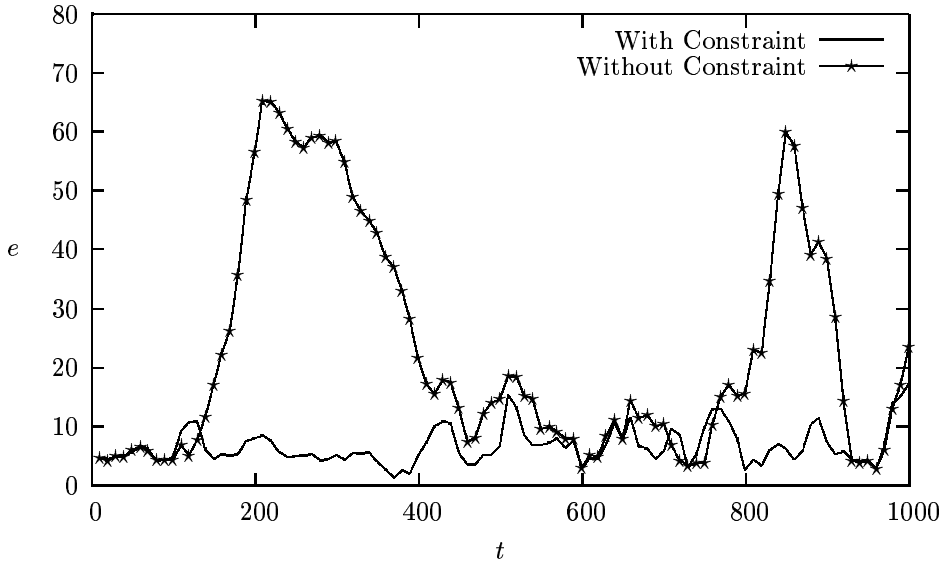


Figure 5.5: The estimation errors of component-based tracking, with or without constraints between object parts. The upper limit of the number of hypotheses is 20. The errors e are in pixels and the time t is the frame number.

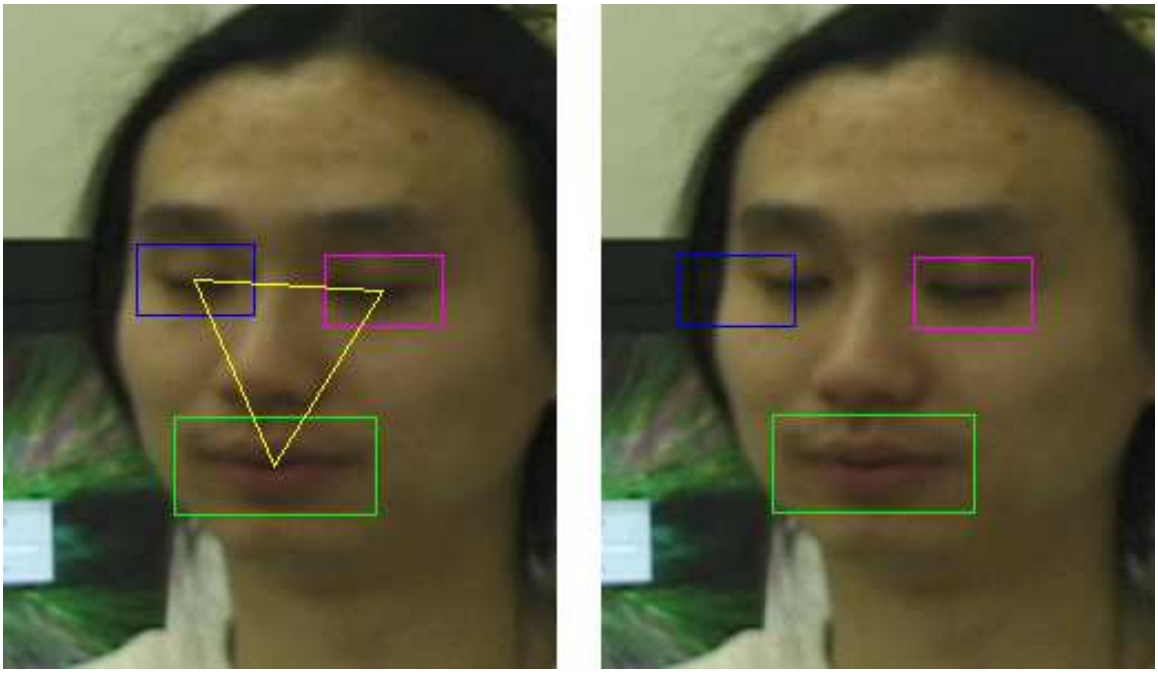


Figure 5.6: The tracking results at frame 143 by component-based tracking. The left image is the result with constraints between object parts while the right image is the result without these constraints. The boxes are regions indicated by the state estimates; the triangle represents the constraints between object parts.

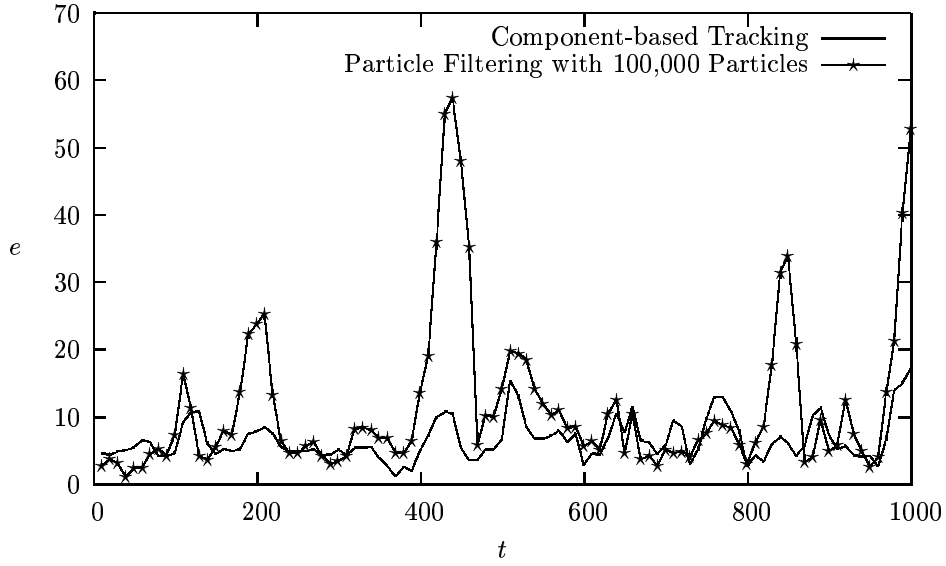


Figure 5.7: The estimation errors of component-based tracking and particle filtering. The upper limit of the number of hypotheses is 20 for component-based tracking; the number of particles is 100,000 for particle filtering. The errors e are in pixels and the time t is the frame number.

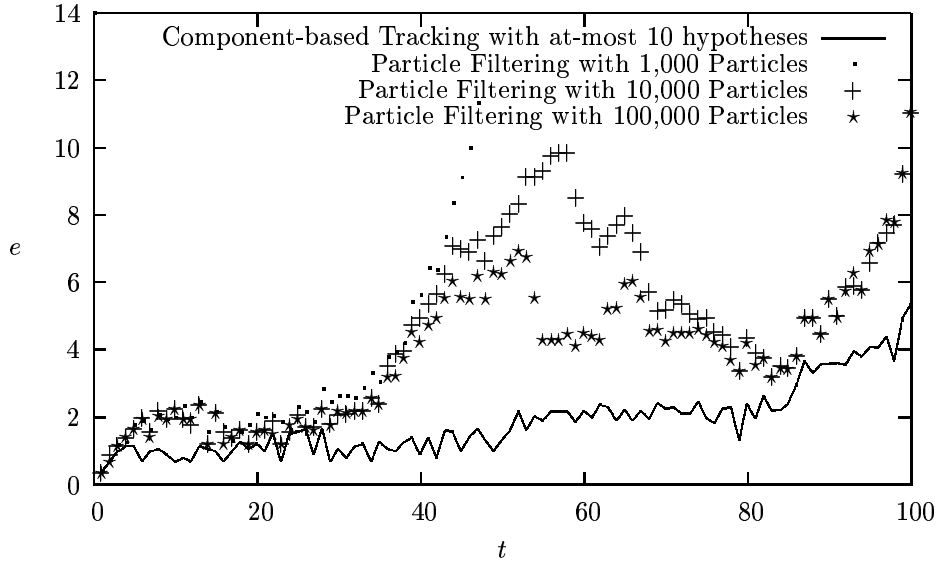


Figure 5.8: The estimation errors of component-based tracking and particle filtering. The upper limit of the number of hypotheses is 20 for component-based tracking; the number of particles is either 1,000, 10,000, or 100,000 for particle filtering. The errors e are in pixels and the time t is the frame number. Note that the results of particle filtering with 1,000 particles are only shown for $t < 50$, since the rest of the data are out of scale.

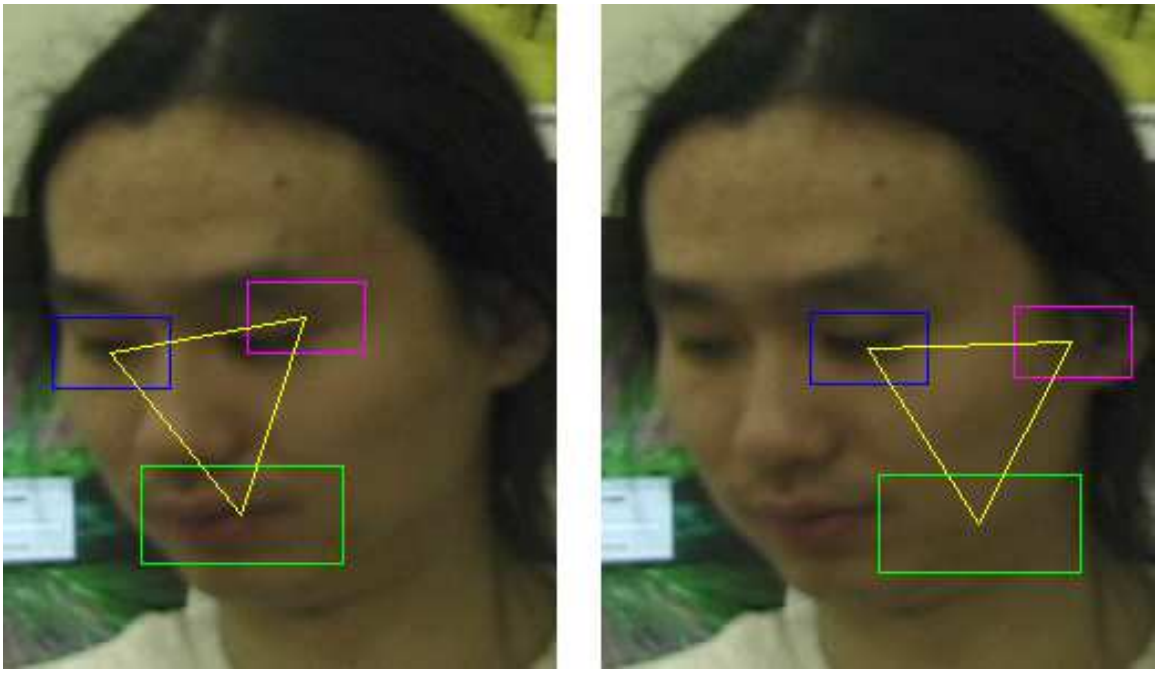


Figure 5.9: The tracking results for comparison at frame 440. The left image is the result from component-based tracking; the right image is the result from particle filtering. The boxes are regions indicated by the state estimates; the triangle represents the constraints between the object parts.

Figure 5.10: An Image from Another Video Sequence

performance of component-based tracking only requires a limited number of hypotheses to perform reasonably well. In this experiment, the resulting difference from choosing the upper limit as either 10 or 20 is too small to show in the plot.

Figure 5.10 shows the first image from another video sequence. The same program is used to track their positions of both the eyes and the mouth of the subject on this video sequence. The results are shown in figure 5.11. Again, the component-based tracker outperforms the particle filter according to the data.

Table 5.1 summarizes the result from both methods with different parameters on both image sequences. The methods are either component-based tracking (CBT) or particle filtering (PF). The parameter is either the upper limit of the number of hypotheses for the former, or the number of particles for the latter. The average error is the sum of estimation errors over time divided by the total number of frames. The average computation time per frame is from the implementation running on an Athlon64 3000+ processor. From the table we can conclude that for this particular settings and video sequence our component-based tracker performs better than the established particle filter.

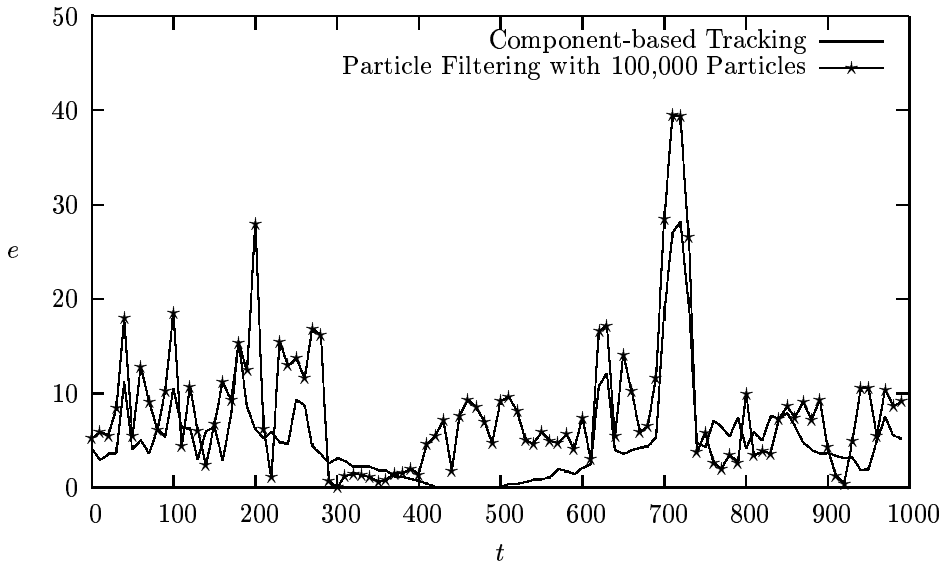


Figure 5.11: The estimation errors of component-based tracking and particle filtering on the second video sequence. The upper limit of the number of hypotheses is 20 for component-based tracking; the number of particles is 100,000 for particle filtering. The errors e are in pixels and the time t is the frame number.

Table 5.1: Result summary of tracking methods. "CBT" stands for component-based tracking and "PF" stands for particle filtering; "Parameter" is either the upper limit of the number of hypotheses for the component-based tracker, or the number of particles in the particle filter; "Error" is the average estimation error in pixels; "Time" is the average computation time in milliseconds per frame; (1) and (2) are results for the first and the second video sequences.

Tracker	Parameter	Error (1)	Time (1)	Error (2)	Time (2)
CBT	2	18.4	59	16.5	62
CBT	10	6.7	65	4.8	75
CBT	20	6.6	66	4.8	77
PF	1,000	33.2	116	34.8	121
PF	10,000	12.8	833	10.1	860
PF	100,000	11.6	7692	8.2	7945

Chapter 6

Conclusion

6.1 Contributions

This thesis has covered two closely related topics: a low-level heterogeneous region tracker by optimal kernels and a high-level component-based tracking framework combining visual clues.

In some sense, optimal kernel tracking is a generalization of sum-of-square difference (SSD) tracking. Although its final mathematical form on estimating the transformation parameters bears similarity with SSD and other related region tracking methods such as hyperplane tracking and kernel tracking, the philosophy behind the optimal kernel tracking is essentially different than these existing works. In optimal kernel tracking,

- A general structure is formulated for estimating of the transformation parameters by using kernels.
- The estimation error of roof kernels on 1-D translation is mathematically studied and an algorithm is developed accordingly for selecting the optimal kernel.
- Some other transformations like 2-D translation, 1-D and 2-D scaling, and rotation are all reduced to 1-D translation. Thus algorithms are developed for selecting the optimal kernel to minimize the estimation error under these transformations.

In summary, optimal kernel tracking is a novel method to track heterogeneous regions under certain types of transformations. By using kernels which have more predictable gradient structure, it attains both accuracy and a great convergence range.

On the other hand, component-based tracking is a general framework based on a probabilistic graphical model. By approximating all functions in the form of a weighted sum of Gaussians, each of which can be thought as a hypothesis, this framework significantly differs from sampling-based probabilistic approaches, such as particle filtering. In component-based tracking,

- A systematic method is proposed to combine and evaluate hypotheses from low-level visual detectors.
- A single intuitive representation is presented for all kinds of constraints, which include internal relationship of target parts, system dynamics, and the observation model.
- A null hypothesis is introduced for missing information to increase the robustness of tracking.
- A message-passing-based algorithm is developed to run the graph for tracking.

Overall, component-based tracking provides an intuitive and modular way to model the tracking target. It works both effectively and efficiently in spite of unreliable low-level visual detectors and approximate modeling of the system dynamics.

6.2 Future Work

The algorithms for optimal kernel tracking are not perfect. In particular, the kernel generated for 2-D scaling is still sensitive to the rotation of the image even though it should not be in theory. The reason is that the process of reducing the problem into 2-D translation results in an image that has overwhelming gradients in one direction. Although the translation in both directions on the image can still be detected separately, the translation of the direction with strong gradients affects the detector of the other direction with weak gradients. It is not a problem in practice, as one can always detect the rotation first. Nevertheless, solving this issue would further improve the accuracy and convergence range when both scaling and rotation are significant.

From a higher point of view, the algorithms of optimal kernel tracking are designed individually for each kind of transformations. Although they include the most common types of transformations, a general formulation to minimize estimation error for an arbitrary transformation would be desirable for a wider range of application.

Both the algorithm of the component-based tracking framework itself and the low-level visual detectors have some parameters to be tuned. The choices of values for some of them are straightforward, for example, the weights of non-null hypotheses from a region tracker, or the covariance of a distance constraint; the choices of some other parameters are not that obvious but reasonable values can still be easily decided by experiments, for example, the maximum number of hypotheses allowed for messages. The problem lies in the rest of the parameters that can be neither chosen in a straightforward way nor easily decided by experiments. The weights of null hypotheses from some low-level detectors are such examples. Although their current choices of values seem reasonable and work in experiments, the tracker could have performed better if there was a systematic way to tune these parameters.

To apply the component-based tracking framework in a more complex setting, more types of visual detectors and constraints are needed. This also relates to another problem that some of

constraints in reality are not easy to approximate as a weighted sum of Gaussians, for example, the constraint that one object part must always be on the left side of another. Its potential is naturally a step function, but the linearization result of the step function does not make much sense,[1] and a functional fitting would generate an infinite number of Gaussians.[2] Of course, for this particular case in practice we do not have to choose the step function as the constraint. If we limit how far the left part can go, the potential can be approximated by a few Gaussians. However, a more general solution to this issue would allow the algorithm to work on more tracking situations.

For an actual application, the current implementation of the component-based tracking algorithm is far from optimized. Some inefficiencies are due to historical reasons as the framework itself was still evolving at the time these codes were written. For example, each hypothesis carries the list of the names of the variables it associates with, and the contents of the list are compared and merged for every single operation on the hypothesis. But it is in fact totally unnecessary since the variables are fixed for any message between two particular nodes in a given graph. The computational performance can be further increased by reducing these inefficiencies.

[1] Because the derivative of the step function is zero everywhere except undefined on the origin.
[2] Because the area under the step function is infinite while the area under any Gaussian is finite.

Bibliography

[1] D. L. Alspach and H. W. Sorerison. Nonlinear Bayesian estimation using Gaussian sum approximation. *IEEE Transactions on Automatic Control*, 17(4):439–448, 1972.

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, third edition, 1999.

[3] Y. Bar-Shalom and T. Fortmann. *Tracking and Data Association*. Academic Press, 1993.

[4] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 679–698, November 1986.

[5] C. K. Chui and G. Chen. *Kalman Filtering with Real Time Applications*. Springer-Verlag, third edition, 1999.

[6] D. Comaniciu, V. Ramesh, and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[7] I. J. Cox and S. L. Hingorani. An efficient implementation and evaluation of Reid's multiple hypothesis tracking algorithm for visual tracking. In *Proceedings of International Conference on Pattern Recognition*, volume A, pages 437–442, 1994.

[8] D. P. Dee and A. M. Da Silva. Data assimilation in the presence of forecast bias. *Quarterly Journal of the Royal Meteorological Society*, 124:169–295, 1998.

[9] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson. An extended set of FORTRAN basic linear algebra subprograms. In *ACM Transcations on Mathematical Software*, pages 1–17, 1988.

[10] A. Doucet, S. Godsill, and C. Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10:197–208, 2000.

[11] G. Evensen. Sequential data assimilation with non-linear quasi-geostrophic model using Monte Carlo methods to forecast error statistics. *Journal of Geophysical Research*, 99(C5):10143–10162, 1994.

[12] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with application to image analysis and automated cartography. *Communications of the Association for Computing Machinery*, 24:381–395, 1981.

[13] D. Fox. KLD-sampling: Adaptive particle filters. In *Proceedings of Neural Information Processing Systems*, 2001.

[14] B. Friedland. Treatment of bias in recursive filtering. *IEEE Transaction on Automatic Control*, 14(4):359–367, 1969.

[15] R. Frühwirth. Track fitting with non-Gaussian noise. *Computer Physics Communications*, page 1, 1997.

[16] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, November 1984.

[17] G. D. Hager and P. N. Belhumeur. Efficient region tracking with parameteric models of geometry and illumination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(10):1125–1139, 1998.

[18] G. D. Hager and M. Dewan. Multiple kernel tracking with ssd. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 790–797, 2004.

[19] G. D. Hager and C. Rasmussen. Joint probabilistic techniques for tracking multi-part objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6):560–576, 2001.

[20] G. D. Hager and K. Toyama. The XVision system: A general-purpose substrate for portable real-time vision applications. *Computer Vision and Image Understanding*, (1):23–37, 1998.

[21] C. J. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings of fourth Alvey Vision Conference*, pages 147–151, 1988.

[22] P. V.C. Hough. Method and means for recognizing complex patterns. United States Patent and Trademarks Office, 03069654, 1962.

[23] M. Isard and A. Blake. CONDENSATION — conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998.

[24] M. Isard and A. Blake. ICONDENSATION: Unifying low-level and high-level tracking in a stochastic framework. In *Proceedings of 5th European Conference on Computer Vision*, pages 893–908, 1998.

[25] M. Isard and J. MacCormick. BraMBLe: A bayesian multiple-blob tracker. In *Proceedings of International Conference on Computer Vision*, pages 34–41, 2001.

[26] F. V. Jensen. *An Introduction to Bayesian Networks.* Springer, 1996.

[27] M. I. Jordan and Y. Weiss. Graphical models: Probabilistic inference. *The Handbook of Brain Theory and Neural Networks, 2nd edition, M. Arbib(Ed.)*, 2002.

[28] S. Julier and J. Uhlmann. A new extension of the Kalman filter to nonlinear systems. In *Proceedings of AeroSense: The 11th International Symposium on Aerospace/Defense Sensing, Simulation and Controls*, 1997.

[29] F. Jurie and M. Dhome. Hyperplane approximation for template matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):996–1000, 2002.

[30] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[31] H. Kalviainen, P. Hirvonen, L. Xu, and E. Oja. Comparisons of probabilistic and non-probabilistic hough transforms. In *Proceedings of 3rd European Conference on Computer Vision*, pages 351–360, 1994.

[32] O. King and D. Forsyth. How does CONDENSATION behave with a finite number of samples. In *Proceedings of 6th European Conference on Computer Vision*, pages 695–709, 2000.

[33] G. Kitagawa. Monte Carlo filter and smoother for nonlinear non-Gaussian state space models. *Journal of Computational and Graphical Statistics*, 5:1–25, 1996.

[34] J. H. Kotecha and P. M. Djurìc. Gaussian sum particle filtering. *IEEE Transactions on Signal Processing*, pages 2602–2612, 2003.

[35] S. Lauritzen. *Graphical Models.* Oxford Press, 1996.

[36] C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for FORTRAN usage. In *ACM Transcations on Mathematical Software*, pages 308–323, 1979.

[37] L. Lu, X. Dai, and G. D. Hager. A particle filter without dynamics for robust 3d face tracking. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, 2004.

[38] L. Lu, X. Dai, and G. D. Hager. Efficient particle filtering using ransac with application to 3d face tracking. In *Proceedings of Image and Vision Computing*, 2005.

[39] S. Thrun M. Montemerlo. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI)*, pages 593–598, 2002.

[40] J. MacCormick and M. Isard. Partitioned sampling, articulated objects, and interface-quality hand tracking. In *Proceedings of European Conference on Computer Vision*, pages 3–19, 2000.

[41] K. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of Uncertainty in AI*, pages 467–475, 1999.

[42] K. G. Murty. An algorithm for ranking all the assignments in order of increasing cost. *Operations Research*, pages 682–687, 1968.

[43] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Network of Plausible Inference*. Morgan Haufmann, San Mateo, CA, 1988.

[44] P. Pérez, J. Vermaak, and A. Blake. Data fusion for visual tracking with particles. In *Proceedings of IEEE*, number 3, pages 495–513, March 2004.

[45] M. K. Pitt and N. Shephard. Filtering via simulation: Auxiliary particle filters. *Journal of the American Statistical Association*, 94(446):590–599, 1999.

[46] D. B. Reid. An algorithm for tracking multiple targets. *IEEE Transactions on Automatic Control*, 24(6):843–854, 1979.

[47] H. S. Richardson. Multiple multistage hypothesis tests: A sequential detection approach to target tracking. Master's thesis, Queen's University, Kingston, Ontario, Canada, 1992.

[48] J. Shi and C. Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.

[49] P. Smyth. Belief networks, hidden Markov models, and Markov random fields:a unifying view. *Pattern Recognition*, 18(11):1261–1268, 1997.

[50] H. W. Sorenson. *Kalman Filtering: Theory and Application*. IEEE Press, New York, 1985.

[51] H. W. Sorenson and D. L. Alspach. Recursive Bayesian estimation using Gaussian sums. *Automatica*, 7:465–479, 1971.

[52] E. B. Sudderth, A. T. Ihler, W. T. Freeman, and A. S. Willsky. Nonparametric belief propagation. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 605–612, 2003.

[53] K. Toyama and G. D. Hager. Incremental focus of attention for robust visual tracking. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, page 189, 1996.

[54] R. van der Merwe, N. de Freitas, A. Doucet, and E. Wan. The unscented particle filter. *Technical Report CUED/F-INFENG/TR 380*, 2000.

[55] Y. Weiss and W. T. Freeman. Correctness of belief propagation in Gaussian graphical models of arbitrary topology. *Neural Computation*, 13(10):2173–2200, 2001.

[56] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder: Real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):780–785, 2005.

[57] L. Xu, E. Oja, and P. Kultanen. A new curve detection method: Randomized Hough transform (rht). *Pattern Recognition Letters*, 11(5):331–338, 1990.

[58] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Generalized belief propagation. *Advances in Neural Information Processing Systems*, pages 689–695, 2000.

Appendix A

Proofs

A.1 Equation (2.4.13) by Equation (2.4.12)

For the sake of simplicity, define

$$\mathbf{F}(\mathbf{x};\tau) = \left|\frac{\partial \mathbf{T}^{-1}}{\partial \mathbf{x}}\right| \mathbf{K}(\mathbf{T}^{-1}(\mathbf{x};\tau))\mathbf{f}(\mathbf{x}) \tag{A.1.1}$$

so equation (2.4.11) becomes

$$\mathbf{D} = \frac{\mathrm{d}}{\mathrm{d}\tau} \int_{\mathbf{x}\in\mathbf{T}(\mathbf{P};\tau)} \mathbf{F}(\mathbf{x};\tau)\ \mathrm{d}\mathbf{x}\Big|_{\tau=\tau_0} \tag{A.1.2}$$

Now at $\tau = \tau_0$

$$\Delta\left(\int_{\mathbf{x}\in\mathbf{T}(\mathbf{P};\tau)} \mathbf{F}(\mathbf{x};\tau)\ \mathrm{d}\mathbf{x}\right) \approx \int_{\mathbf{x}\in\mathbf{T}(\mathbf{P};\tau_0)} \Delta\mathbf{F}(\mathbf{x};\tau)\ \mathrm{d}\mathbf{x} + \int_{\mathbf{x}\in\mathbf{B_P}} \mathbf{F}(\mathbf{x};\tau_0)\ \mathrm{d}\mathbf{x} \tag{A.1.3}$$

Since $\mathbf{T}(\mathbf{x};\tau_\mathbf{0}) = \mathbf{x}$ (see equation (2.2.3)), if

$$\forall_{\mathbf{x}\in\mathbf{B_P}}\mathbf{K}(\mathbf{x}) = 0 \tag{A.1.4}$$

we have

$$\int_{\mathbf{x}\in\mathbf{B_P}} \mathbf{F}(\mathbf{x};\tau_0)\ \mathrm{d}\mathbf{x} = 0 \tag{A.1.5}$$

thus

$$\frac{\mathrm{d}}{\mathrm{d}\tau}\int_{\mathbf{x}\in\mathbf{T}(\mathbf{P};\tau)} \mathbf{F}(\mathbf{x};\tau)\ \mathrm{d}\mathbf{x}\Big|_{\tau=\tau_0} \approx \int_{\mathbf{x}\in\mathbf{P}} \frac{\mathrm{d}}{\mathrm{d}\tau}\mathbf{F}(\mathbf{x};\tau)\ \mathrm{d}\mathbf{x}\Big|_{\tau=\tau_0} \tag{A.1.6}$$

Substitue back $\mathbf{F}$,

$$\begin{aligned}\mathbf{D} &\approx \int_{\mathbf{x}\in\mathbf{P}} \frac{\partial}{\partial\tau}\left(\left|\frac{\partial \mathbf{T}^{-1}}{\partial \mathbf{x}}\right| \mathbf{K}(\mathbf{T}^{-1}(\mathbf{x};\tau))\mathbf{f}(\mathbf{x})\right)\ \mathrm{d}\mathbf{x}\Big|_{\tau=\tau_0} \\ &= \int_{\mathbf{x}\in\mathbf{P}} \left[\frac{\partial}{\partial\tau}\left|\frac{\partial \mathbf{T}^{-1}}{\partial \mathbf{x}}\right| \mathbf{K}(\mathbf{x}) + \left|\frac{\partial \mathbf{T}^{-1}}{\partial \mathbf{x}}\right| \frac{\partial \mathbf{T}^{-1}}{\partial\tau}\mathbf{K}'(\mathbf{x})\right] \mathbf{f}(\mathbf{x})\ \mathrm{d}\mathbf{x}\Big|_{\tau=\tau_0}\end{aligned} \tag{A.1.7}$$

which is equation (2.4.13).

A.2 Equation (3.5.2) and (3.5.3)

For symmetric, inversible matrice A_1 and A_2 of the same dimension:

$$\begin{aligned} & A_1(A_1+A_2)^{-1}A_2(A_1^{-1}+A_2^{-1}) \\ = \; & A_1(A_1+A_2)^{-1}A_2A_1^{-1}+A_1(A_1+A_2)^{-1} \\ = \; & A_1(A_1+A_2)^{-1}A_2A_1^{-1}+A_1(A_1+A_2)^{-1}A_1A_1^{-1} \\ = \; & A_1(A_1+A_2)^{-1}(A_1+A_2)A_1^{-1} \\ = \; & I \end{aligned} \tag{A.2.1}$$

Thus we have

$$\begin{aligned} & A_1(A_1+A_2)^{-1}A_2 \\ = \; & (A_1^{-1}+A_2^{-1})^{-1} \\ = \; & A_2(A_1+A_2)^{-1}A_1 \end{aligned} \tag{A.2.2}$$

Now is deduction process of equation (3.5.2) and (3.5.3):

$$\begin{aligned} & w_1\exp\left\{-\frac{1}{2}(\mathbf{x}-\mathbf{u}_1)^TA_1(\mathbf{x}-\mathbf{u}_1)\right\}\cdot w_2\exp\left\{-\frac{1}{2}(\mathbf{x}-\mathbf{u}_2)^TA_2(\mathbf{x}-\mathbf{u}_2)\right\} \\ = \; & w_1w_2\exp\left\{-\frac{1}{2}(\mathbf{x}^T(A_1+A_2)\mathbf{x}-(\mathbf{u}_1^TA_1+\mathbf{u}_2^TA_2)\mathbf{x}-\mathbf{x}^T(A_1\mathbf{u}_1^T+A_2\mathbf{u}_2^T)+\mathbf{u}_1^TA_1\mathbf{u}_1+\mathbf{u}_2^TA_2\mathbf{u}_2)\right\} \\ = \; & w_1w_2\exp\left\{-\frac{1}{2}(\mathbf{x}-(A_1+A_2)^{-1}(A_1\mathbf{u}_1+A_2\mathbf{u}_2))^T(A_1+A_2)(\mathbf{x}-(A_1+A_2)^{-1}(A_1\mathbf{u}_1+A_2\mathbf{u}_2))\right\} \\ & \cdot\exp\left\{-\frac{1}{2}(-(A_1\mathbf{u}_1+A_2\mathbf{u}_2)^T(A_1+A_2)^{-1}(A_1\mathbf{u}_1+A_2\mathbf{u}_2)+\mathbf{u}_1^TA_1\mathbf{u}_1+\mathbf{u}_2^TA_2\mathbf{u}_2)\right\} \\ = \; & w_1w_2\exp\left\{-\frac{1}{2}(\mathbf{x}-(A_1+A_2)^{-1}(A_1\mathbf{u}_1+A_2\mathbf{u}_2))^T(A_1+A_2)(\mathbf{x}-(A_1+A_2)^{-1}(A_1\mathbf{u}_1+A_2\mathbf{u}_2))\right\} \\ & \cdot\exp\left\{-\frac{1}{2}(-\mathbf{u}_1^TA_1(A_1+A_2)^{-1}(A_1\mathbf{u}_1+A_2\mathbf{u}_2)-\mathbf{u}_2^TA_2(A_1+A_2)^{-1}(A_1\mathbf{u}_1+A_2\mathbf{u}_2))\right\} \\ & \cdot\exp\left\{-\frac{1}{2}(\mathbf{u}_1^TA_1(A_1+A_2)^{-1}(A_1+A_2)\mathbf{u}_1+\mathbf{u}_2^TA_2(A_1+A_2)^{-1}(A_1+A_2)\mathbf{u}_2)\right\} \\ = \; & w_1w_2\exp\left\{-\frac{1}{2}(\mathbf{x}-(A_1+A_2)^{-1}(A_1\mathbf{u}_1+A_2\mathbf{u}_2))^T(A_1+A_2)(\mathbf{x}-(A_1+A_2)^{-1}(A_1\mathbf{u}_1+A_2\mathbf{u}_2))\right\} \\ & \cdot\exp\left\{-\frac{1}{2}(\mathbf{u}_1^TA_1(A_1+A_2)^{-1}A_2(\mathbf{u}_1-\mathbf{u}_2)-\mathbf{u}_2^TA_2(A_1+A_2)^{-1}A_1(\mathbf{u}_1-\mathbf{u}_2))\right\} \\ = \; & w_1w_2\exp\left\{-\frac{1}{2}(\mathbf{u}_1-\mathbf{u}_2)^TA_1(A_1+A_2)^{-1}A_2(\mathbf{u}_1-\mathbf{u}_2)\right\} \\ & \cdot\exp\left\{-\frac{1}{2}(\mathbf{x}-(A_1+A_2)^{-1}(A_1\mathbf{u}_1+A_2\mathbf{u}_2))^T(A_1+A_2)(\mathbf{x}-(A_1+A_2)^{-1}(A_1\mathbf{u}_1+A_2\mathbf{u}_2))\right\} \end{aligned} \tag{A.2.3}$$

Note that the last step above uses equation (A.2.2).

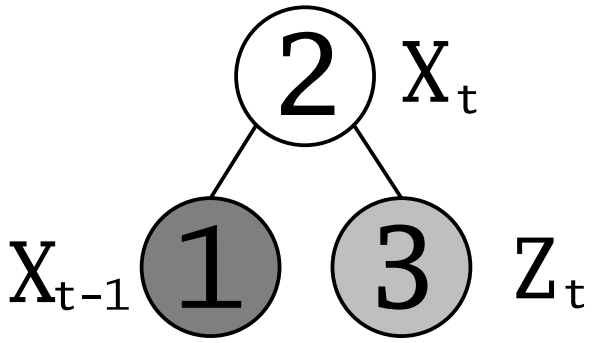


Figure A.1: Labelled Kalman Filter Equivalence Graph

A.3 Equation (3.5.4) and (3.5.5)

Let m be the length of vector x_j or 1 in case it is a scalar, then for any v which is not a function of x_j and symmetric, positive definite matrix W:

$$\int \exp\left\{-\frac{1}{2}(x_j - v)^T W (x_j - v)\right\} \mathrm{d}x_j = (2\pi)^{\frac{m}{2}} \|W\|^{-\frac{1}{2}} \tag{A.3.1}$$

where $\|M\|$ stands for the absolute value of determinant of matrix M.

Now we have

$$\begin{aligned}
& \int w \exp\left\{-\frac{1}{2}\begin{pmatrix}\mathbf{x}_{\bar{j}} - \mathbf{u}_{\bar{j}} \\ x_j - u_j\end{pmatrix}^T \begin{pmatrix} A & B \\ B^T & C\end{pmatrix} \begin{pmatrix}\mathbf{x}_{\bar{j}} - \mathbf{u}_{\bar{j}} \\ x_j - u_j\end{pmatrix}\right\} \mathrm{d}x_j \\
= \; & w \exp\left\{-\frac{1}{2}(\mathbf{x}_{\bar{j}} - \mathbf{u}_{\bar{j}})^T A (\mathbf{x}_{\bar{j}} - \mathbf{u}_{\bar{j}})\right\} \\
& \cdot \int \exp\left\{-\frac{1}{2}((\mathbf{x}_{\bar{j}} - \mathbf{u}_{\bar{j}})^T B (\mathbf{x}_j - \mathbf{u}_j) + (\mathbf{x}_j - \mathbf{u}_j)^T B^T (\mathbf{x}_{\bar{j}} - \mathbf{u}_{\bar{j}}) + (\mathbf{x}_j - \mathbf{u}_j)^T C (\mathbf{x}_j - \mathbf{u}_j))\right\} \mathrm{d}x_j \\
= \; & w \exp\left\{-\frac{1}{2}((\mathbf{x}_{\bar{j}} - \mathbf{u}_{\bar{j}})^T A (\mathbf{x}_{\bar{j}} - \mathbf{u}_{\bar{j}}) - (\mathbf{x}_{\bar{j}} - \mathbf{u}_{\bar{j}})^T B C^{-1} B^T (\mathbf{x}_{\bar{j}} - \mathbf{u}_{\bar{j}}))\right\} \\
& \cdot \int \exp\left\{-\frac{1}{2}(((\mathbf{x}_j - \mathbf{u}_j) - C^{-1} B^T (\mathbf{x}_{\bar{j}} - \mathbf{u}_{\bar{j}}))^T C ((\mathbf{x}_j - \mathbf{u}_j) - C^{-1} B^T (\mathbf{x}_{\bar{j}} - \mathbf{u}_{\bar{j}})))\right\} \mathrm{d}x_j \\
= \; & w (2\pi)^{\frac{m}{2}} \|C\|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(\mathbf{x}_{\bar{j}} - \mathbf{u}_{\bar{j}})^T (A - B C^{-1} B^T)(\mathbf{x}_{\bar{j}} - \mathbf{u}_{\bar{j}})\right\}
\end{aligned} \tag{A.3.2}$$

Note that the last step above uses equation (A.3.1).

A.4 Equivalence of Figure 3.8 and the Kalman Filter

To simplify the notation, let us use numbers to label the nodes in figure 3.8, resulting in figure A.1.

The standard (discrete) Kalman filter setup can be described as follows:

$$X_t = AX_{t-1} + \omega_t \tag{A.4.1a}$$

$$Z_t = HX_t + \nu_t \tag{A.4.1b}$$

where X_t is the state (vector) at time-step t and Z_t is the observation (vector) at time-step t. ω_t and ν_t are white Gaussian noises with covariance Q and R, respectively. Now we are going to see that in this case our graphical framework without null hypothesis generates the same result as the Kalman filter.

Assume P_{t-1} be the covariance and X_{t-1}, the potential functions are listed as follows:

$$\psi_1(X_{t-1}) = \exp\left\{\{-(X_{t-1} - \hat{X}_{t-1})^T P_{t-1}^{-1}(X_{t-1} - \hat{X}_{t-1})\}\right\} \tag{A.4.2a}$$

$$\psi_{12}(X_{t-1}, X_t) = \exp\left\{\{-(X_t - AX_{t-1})^T Q^{-1}(X_t - AX_{t-1})\}\right\} \tag{A.4.2b}$$

$$\psi_{23}(X_t, Z_t) = \exp\left\{\{-(Z_t - HX_t)^T R^{-1}(Z_t - HX_t)\}\right\} \tag{A.4.2c}$$

From the equations above, we can deduce the following:

$$\begin{aligned} m_{12}(X_t) &= \int \psi_1(X_{t-1})\psi_{12}(X_{t-1}, X_t)\,\mathrm{d}X_{t-1} \\ &= \exp\left\{\{-(X_t - AX_{t-1})^T (A^T P_{t-1} A + Q)^{-1}(X_t - AX_{t-1})\}\right\} \end{aligned} \tag{A.4.3a}$$

$$\begin{aligned} m_{23}(X_t) &= \psi_{23}(X_t, Z_t) \\ &= \exp\left\{\{-(Z_t - HX_t)^T R^{-1}(Z_t - HX_t)\}\right\} \end{aligned} \tag{A.4.3b}$$

Now define

$$P_t^- = A^T P_{t-1} A + Q \tag{A.4.4}$$

we have

$$m_{12} = \exp\left\{\{-(X_t - AX_{t-1})^T (P_t^-)^{-1}(X_t - AX_{t-1})\}\right\} \tag{A.4.5}$$

and

$$\begin{aligned} p(X_t) &= m_{12}(X_t) m_{23}(x_t) \\ &= \exp\left\{\{(X_t - \hat{X}_t)^T P_t^{-1}(X_t - \hat{X}_t)\}\right\} \end{aligned} \tag{A.4.6}$$

where

$$\hat{X}_t = (P_t^{-^{-1}} + H^T R^{-1} H)^{-1}(P_t^{-^{-1}} A\hat{X}_{t-1} + H^T R^{-1} Z_t) \tag{A.4.7a}$$

$$P_t = (P_t^{-^{-1}} + H^T R^{-1} H)^{-1} \tag{A.4.7b}$$

Consider the fact that

$$(P_t^{-^{-1}} + H^T R^{-1} H)^{-1} = P_t^- - P_t^- H^T (H P_t^- H^T + R)^{-1} H P_t^- \tag{A.4.8}$$

we have got

$$\hat{X}_t = A\hat{X}_{t-1} + P_t^- H^T (HP_t^- H^T + R)^{-1}(Z_t - HA\hat{X}_{t-1}) \tag{A.4.9a}$$

$$P_t = (I - P_t^- H^T (HP_t^- H^T + R)^{-1} H) P_t^- \tag{A.4.9b}$$

If we define $K_t = P_t^- H^T (HP_t^- H^T + R)^{-1}$, the above becomes the familiar Kalman filter equations.

A.5 Equation (2.5.19)

Denote

$$\begin{aligned} d_f \triangleq & \int_a^b (K(x+u) - K(x) - K'(x)u) f(x)\, \mathrm{d}x \\ & + \int_{a-u}^{a} K(x+u) f(x)\, \mathrm{d}x - \int_{b-u}^{b} K(x+u) f(x)\, \mathrm{d}x \end{aligned} \tag{A.5.1}$$

so from equation (2.5.11)

$$e_f = E\left[\left(D^{-1} d_f\right)^2\right] \tag{A.5.2}$$

For $u \geq 0$, assuming small u that $u < \min(c-a, b-c)$ and $f(x+\epsilon) \approx f(x)$ for $0 < |\epsilon| < |u|$ based on the assumed continuity of $f(x)$:

$$\begin{aligned} & \int_a^b (R(x+u;c) - R(x;c) - R'(x;c)u) f(x)\, \mathrm{d}x \\ = & -\frac{(b-a)}{(c-a)(b-c)} \int_{c-u}^{c} (x+u-c) f(x)\, \mathrm{d}x + \frac{1}{b-c} \int_{b-u}^{b} (x+u-b) f(x)\, \mathrm{d}x \\ \approx & -\frac{(b-a) f(c)}{(c-a)(b-c)} \frac{u^2}{2} + \frac{f(b)}{(b-c)} \frac{u^2}{2} \end{aligned} \tag{A.5.3}$$

$$\begin{aligned} \int_{a-u}^{a} R(x+u;c) f(x)\, \mathrm{d}x &= \frac{1}{c-a} \int_{a-u}^{a} (x+u-a) f(x)\, \mathrm{d}x \\ &\approx \frac{f(a)}{(c-a)} \frac{u^2}{2} \end{aligned} \tag{A.5.4}$$

$$\int_{b-u}^{b} R(x+u) f(x)\, \mathrm{d}x = 0 \tag{A.5.5}$$

Overall, for $u \geq 0$,

$$\begin{aligned} d_f &= \left(\frac{f(a)}{(c-a)} + \frac{f(b)}{(b-c)} - \frac{(b-a) f(c)}{(c-a)(b-c)} \right) \frac{u^2}{2} \\ &= -\left(\frac{f(c) - f(a)}{c-a} + \frac{f(c) - f(b)}{c-b} \right) \frac{u^2}{2} \end{aligned} \tag{A.5.6}$$

and the exact same formula can be deduced in the case of $u < 0$ under the same assumptions. Let

$$S = \frac{f(c) - f(a)}{c-a} + \frac{f(c) - f(b)}{c-b} \tag{A.5.7}$$

and from equation (A.5.2) we have

$$e_f \approx \frac{1}{4} D^{-2} S^2 E[u^4] \tag{A.5.8}$$

A.6 Equation (3.6.9) and (3.6.10)

Let

$$\gamma_{ij}(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(A\mathbf{x}_i - B\mathbf{x}_j)^T(A\mathbf{x}_i - B\mathbf{x}_j)} - d \tag{A.6.1}$$

so that

$$\psi_{ij}(\mathbf{x}_i, \mathbf{x}_j) = \exp\left\{-\frac{1}{2}\gamma_{ij}^2(\mathbf{u}, \mathbf{v})\right\} \tag{A.6.2}$$

Now since[1]

$$\frac{\partial \gamma}{\partial \mathbf{x}_i} = \frac{2(A\mathbf{x}_i - B\mathbf{x}_j)^T A}{\sqrt{(A\mathbf{x}_i - B\mathbf{x}_j)^T(A\mathbf{x}_i - B\mathbf{x}_j)}} \tag{A.6.3a}$$

$$\frac{\partial \gamma}{\partial \mathbf{x}_j} = -\frac{2(A\mathbf{x}_j - B\mathbf{x}_j)^T B}{\sqrt{(A\mathbf{x}_i - B\mathbf{x}_j)^T(A\mathbf{x}_i - B\mathbf{x}_j)}} \tag{A.6.3b}$$

and

$$\psi'_{ij}(\mathbf{x}_i, \mathbf{x}_j; \mathbf{u}, \mathbf{v}) = \exp\left\{-\frac{1}{2}\left(\gamma_{ij}(\mathbf{u}, \mathbf{v}) + \frac{\partial \gamma_{ij}}{\partial \mathbf{x}_i}(\mathbf{u}, \mathbf{v})(\mathbf{x}_i - \mathbf{u}) + \frac{\partial \gamma_{ij}}{\partial \mathbf{x}_j}(\mathbf{u}, \mathbf{v})(\mathbf{x}_j - \mathbf{v})\right)^2\right\} \tag{A.6.4}$$

we have

$$\begin{aligned}
&\psi'_{ij}(\mathbf{x}_i, \mathbf{x}_j; \mathbf{u}, \mathbf{v}) \\
= \quad &\exp\left\{-\frac{1}{2}\left(\sqrt{(A\mathbf{u} - B\mathbf{v})^T(A\mathbf{u} - B\mathbf{v})} - d + \frac{2(A\mathbf{u} - B\mathbf{v})^T A(\mathbf{x}_i - \mathbf{u})}{\sqrt{(A\mathbf{u} - B\mathbf{v})^T(A\mathbf{u} - B\mathbf{v})}} - \frac{2(A\mathbf{u} - B\mathbf{v})^T B(\mathbf{x}_j - \mathbf{v})}{\sqrt{(A\mathbf{u} - B\mathbf{v})^T(A\mathbf{u} - B\mathbf{v})}}\right)^2\right\} \\
= \quad &\exp\left\{-\frac{1}{2}\left(\sqrt{(A\mathbf{u} - B\mathbf{v})^T(A\mathbf{u} - B\mathbf{v})} - d + \frac{2(A\mathbf{u} - B\mathbf{v})^T(A\mathbf{x}_i - B\mathbf{x}_j)}{\sqrt{(A\mathbf{u} - B\mathbf{v})^T(A\mathbf{u} - B\mathbf{v})}} - \frac{2(A\mathbf{u} - B\mathbf{v})^T(A\mathbf{u} - B\mathbf{v})}{\sqrt{(A\mathbf{u} - B\mathbf{v})^T(A\mathbf{u} - B\mathbf{v})}}\right)^2\right\} \\
= \quad &\exp\left\{-\frac{1}{2}\left(\frac{2(A\mathbf{u} - B\mathbf{v})^T(A\mathbf{x}_i - B\mathbf{x}_j)}{\sqrt{(A\mathbf{u} - B\mathbf{v})^T(A\mathbf{u} - B\mathbf{v})}} - \sqrt{(A\mathbf{u} - B\mathbf{v})^T(A\mathbf{u} - B\mathbf{v})} - d\right)^2\right\}
\end{aligned} \tag{A.6.5}$$

If we define

$$k = \sqrt{(A\mathbf{u} - B\mathbf{v})^T(A\mathbf{u} - B\mathbf{v})} \tag{A.6.6}$$

then equation (A.6.5) can be written as

$$\psi'_{ij}(\mathbf{x}_i, \mathbf{x}_j; \mathbf{u}, \mathbf{v}) = \exp\left\{-\frac{1}{2}(\frac{2}{k}(A\mathbf{u} - B\mathbf{v})^T(A\mathbf{x}_i - B\mathbf{x}_j) - k - d)^2\right\} \tag{A.6.7}$$

[1] The (partial) derivative of a scalar with respect to a column vector is presented as a row vector.

Appendix B

Two Forms of the Multivariate Gaussian

A form of a function is just a regularized way to present the given function. Different forms of a function are still the same function. On the other hand, when implementing symbolic manipulation of the function, one form may be more convenient or has better computational efficiency than another in spite of the same result. In our framework, mixture of Gaussians (MoG) is the most frequently used form of function so choosing the right form for Gaussian functions is important.

This section lists two forms of multivariate Gaussian and their corresponding rules for multiplication and (indefinite) integration. The proofs are either trivial or similar to those in appendix A.

B.1 Natural Form

The natural form of a multivariate Gaussian function is

$$G(\mathbf{x};\mathbf{u},W) = (2\pi)^{-\frac{m}{2}}\|W\|^{\frac{1}{2}}\exp\left\{-\frac{1}{2}(\mathbf{x}-\mathbf{u})^T W(\mathbf{x}-\mathbf{u})\right\} \tag{B.1.1}$$

where m is the number of dimensions of $\mathbf{x}$.

- Rule of multiplication:

$$\begin{aligned} & G(\mathbf{x};\mathbf{u}_1,W_1)\cdot G(\mathbf{x};\mathbf{u}_2,W_2) \\ = \; & \left(\frac{\|W_1^{-1}+W_2^{-1}\|}{\|W_1^{-1}\|\cdot\|W_2^{-1}\|}\right)^{\frac{1}{2}}\exp\left\{-\frac{1}{2}(\mathbf{u}_1-\mathbf{u}_2)^T(W_1^{-1}+W_2^{-1})^{-1}(\mathbf{u}_1-\mathbf{u}_2)\right\} \\ & \cdot G(\mathbf{x};(W_1+W_2)^{-1}(W_1\mathbf{u}_1+W_2\mathbf{u}_2),W_1+W_2) \end{aligned} \tag{B.1.2}$$

- Rule of integration:

$$\begin{aligned} & \int G(\begin{pmatrix}\mathbf{x}\\ \mathbf{y}\end{pmatrix};\begin{pmatrix}\mathbf{u}\\ \mathbf{v}\end{pmatrix},\begin{pmatrix}A & B\\ B^T & C\end{pmatrix})\,\mathrm{d}\mathbf{y} \\ = \quad & (\frac{\|(A-BC^{-1}B^T)^{-1}\|\cdot\|C^{-1}\|}{\left\|\begin{pmatrix}A & B\\ B^T & C\end{pmatrix}^{-1}\right\|})^{\frac{1}{2}}G(\mathbf{x};\mathbf{u},A-BC^{-1}B^T) \end{aligned} \tag{B.1.3}$$

B.2 Homogeneous Form

The homogeneous form of a multivariate Gaussian function is (note that the symmetric matrix A has one more row than the column vector $\mathbf{x}$ has):

$$H(\mathbf{x};A)=\exp\left\{-\frac{1}{2}\begin{pmatrix}\mathbf{x}^T & 1\end{pmatrix}A\begin{pmatrix}\mathbf{x}\\ 1\end{pmatrix}\right\} \tag{B.2.1}$$

The homogeneous form can convert to the natural form and vice versa:

$$G(\mathbf{x};\mathbf{u},W)=(2\pi)^{-\frac{m}{2}}\|W\|^{\frac{1}{2}}H(\mathbf{x};\begin{pmatrix}W & -W\mathbf{u}\\ -\mathbf{u}^TW & \mathbf{u}^TW\mathbf{u}\end{pmatrix}) \tag{B.2.2a}$$

$$H(\mathbf{x};\begin{pmatrix}W & B\\ B^T & c\end{pmatrix})=(2\pi)^{\frac{m}{2}}\|W\|^{-\frac{1}{2}}\exp\left\{-\frac{1}{2}(c-B^TW^{-1}B)\right\}G(\mathbf{x};-W^{-1}B,W) \tag{B.2.2b}$$

where m is the number of dimensions of $\mathbf{x}$.

- Rule of multiplication:

$$H(\mathbf{x};W_1)\cdot H(\mathbf{x};W_2)=H(\mathbf{x};W_1+W_2) \tag{B.2.3}$$

- Rule of integration (note that symmetric matrix A has one more row than column vector $\mathbf{x}$ has while symmetric matrix C has same number of rows as column vector $\mathbf{y}$ has):

$$\begin{aligned} & \int H(\begin{pmatrix}\mathbf{y}\\ \mathbf{x}\end{pmatrix};\begin{pmatrix}C & B^T\\ B & A\end{pmatrix})\,\mathrm{d}\mathbf{y} \\ = \quad & (2\pi)^{\frac{m_y}{2}}\|C\|^{-\frac{1}{2}}H(\mathbf{x};A-BC^{-1}B^T) \end{aligned} \tag{B.2.4}$$

  where m_y is the number of dimensions of $\mathbf{y}$.

Appendix C

A Sample of the Source Code

The code listed here is the actual code used in the face tracker in chapter 5. The code is a function to construct the graph as in figure 5.4, and to initialize the underlying region trackers by a face model which contains the template images and their positions in the first frame.

```
Ref<Frame> build_graph( GrayImage& input, const FaceModel& face ) {

  Ref<ImageInput<Image> > source = newImageInput(input) ;
  Ref<Frame> graph = new Frame ;

  // left eye and its constraints
  // make names of variables
  VariableSet vl = makeVariableSet( "xl", "yl" ) ;
  VariableSet vlh = makeVariableSet( "xlh", "ylh" ) ;
  // create a region tracker using roof kernels for 2-D translation
  cbt::Region * l_eye = new cbt::Region
    ( vl, face.l_eye, source, new cbt::Region::TransRoof() );
  // initialize the region tracker
  l_eye->init( vector<double>( face.l_pos, face.l_pos+2 ) );
  // add part component to the graph
  Ref<Node> nl = graph->add( 0, vl, (string)"l_eye" );
  // add observation component to the graph, which is the region tracker
  Ref<Node> nlo = graph->add( l_eye, (string)"l_eye_o" );
  // add history component to the graph
  Ref<Node> nlh = graph->add( new History( **nl, vl, vlh ), vlh,
                              (string)"l_eye_h" );
  // add a simple constriant between part and observation to the graph
  graph->add( nl, nlo, new SimplePotential( vl, 0, .01 ) );
  // add a simple constraint between part and history to the graph
  graph->add( nl, nlh, new SimplePotential( vl, vlh, .01 ) );

  // right eye and its constraints
  // make names of variables
  VariableSet vr = makeVariableSet( "xr", "yr" ) ;
  VariableSet vrh = makeVariableSet( "xrh", "yrh" ) ;
```

```
  // create a region tracker using roof kernels for 2-D translation
  cbt::Region * r_eye = new cbt::Region
    ( vr, face.r_eye, source, new cbt::Region::TransRoof() );
  // initialize the region tracker
  r_eye->init( vector<double>( face.r_pos, face.r_pos+2 ) );
  // add part component to the graph
  Ref<Node> nr = graph->add( 0, vr, (string)"r_eye" );
  // add observation component to the graph, which is the region tracker
  Ref<Node> nro = graph->add( r_eye, (string)"r_eye_o" );
  // add history component to the graph
  Ref<Node> nrh = graph->add( new History( **nr, vr, vrh ), vrh,
                              (string)"r_eye_h" );
  // add a simple constriant between part and observation to the graph
  graph->add( nr, nro, new SimplePotential( vr, 0, .01 ) );
  // add a simple constraint between part and history to the graph
  graph->add( nr, nrh, new SimplePotential( vr, vrh, .01 ) );

  // mouth and its constraints
  // make names of variables
  VariableSet vm = makeVariableSet( "xm", "ym" ) ;
  VariableSet vmh = makeVariableSet( "xmh", "ymh" ) ;
  // create a region tracker using roof kernels for 2-D translation
  cbt::Region * mouth = new cbt::Region
    ( vm, face.mouth, source, new cbt::Region::TransRoof() );
  // initialize the region tracker
  mouth->init( vector<double>( face.m_pos, face.m_pos+2 ) );
  // add part component to the graph
  Ref<Node> nm = graph->add( 0, vm, (string)"mouth" );
  // add observation component to the graph, which is the region tracker
  Ref<Node> nmo = graph->add( mouth, (string)"mouth_o" );
  // add history component to the graph
  Ref<Node> nmh = graph->add( new History( **nm, vm, vmh ), vmh,
                              (string)"mouth_h" );
  // add a simple constriant between part and observation to the graph
  graph->add( nm, nmo, new SimplePotential( vm, 0, .01 ) );
  // add a simple constraint between part and history to the graph
  graph->add( nm, nmh, new SimplePotential( vm, vmh, .01 ) );

  // constraints between the three parts
  // add a distance constraint between left and right eyes to the graph
  graph->add( nl, nr, new DistancePotential
      ( vl, vr, .1, distance( face.l_pos, face.r_pos ) ));
  // add a distance constraint between the left eye and the mouth to the graph
  graph->add( nl, nm, new DistancePotential
      ( vl, vm, .1, distance( face.l_pos, face.m_pos ) ));
  // add a distance constraint between the right eye and the mouth to the graph
  graph->add( nr, nm, new DistancePotential
      ( vr, vm, .1, distance( face.r_pos, face.m_pos ) ));

  return graph ;
} // build_graph
```

Vita

Xiangtian Dai was born in September, 1976 in Suzhou, China. He received his Bachelor of Engineering in Electronic Engineering at Tsinghua University[1], Beijing, China in 1999. After that, he entered the Ph.D. program of Department of Computer Science at Johns Hopkins University, and received his Master of Science in Engineering there in 2001. He tried to defend his Ph.D. thesis on August 25th, 2005.

[1] That you can probably deduce from the style of this vita.