Tracking and Recognizing Actions of Multiple Hockey Players using the Boosted Particle Filter

by

Wei-Lwun Lu

B.Sc., National Tsing Hua University, 2002

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Computer Science)

The University of British Columbia

April 2007

Abstract

This thesis presents a system that can automatically track multiple hockey players and simultaneously recognize their actions given a single broadcast video sequence, where detection is complicated by a panning, tilting, and zooming camera. Firstly, we use the Histograms of Oriented Gradients (HOG) to represent the players, and introduce a probabilistic framework to model the appearance of the players by a mixture of local subspaces. We also employ an efficient offline learning algorithm to learn the templates from training data, and an efficient online filtering algorithm to update the templates used by the tracker. Secondly, we recognize the players' actions by incorporating the HOG descriptors with a pure multi-class sparse classifier with a robust motion similarity measure. Lastly, we augment the Boosted Particle Filter (BPF) with new observation model and template updater that improves the robustness of the tracking system. Experiments on long sequences show promising quantitative and qualitative results, and the system can run smoothly in near real-time.

Contents

List of Tables

List of Figures

List of Algorithms

Acknowledgements

Without encouragement from James J. Little, it would be impossible to write this thesis. It is difficult to express my gratitude for his extensive support, discussion and endless intelligent ideas.

I am grateful to Kevin Murphy and Kenji Okuma for their contribution of time and ideas to my work. I would also like to thank Brendan Little for hand annotating the hockey sequence.

Special thanks for Yu-Hsuan. Your encouragement, company, and support have made these three years in UBC a memorable and wonderful experience.

I would also like to thank my parents and many other friends in Vancouver, for helping me and enjoying our happy life together.

WEI-LWUN LU

The University of British Columbia
April 2007

To my love, Yu-Hsuan.

Chapter 1

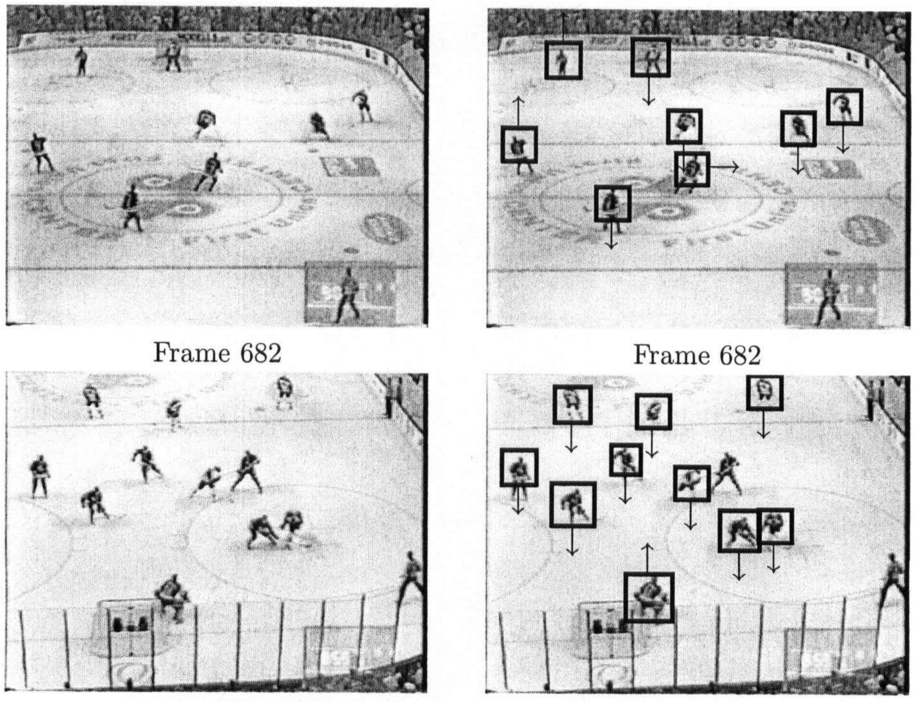
Introduction

1.1 Motivation

Visual tracking and action recognition systems have gained more and more attention in the past few years because of their potential applications in smart surveillance systems, advanced human-computer interfaces, and sport video analysis. In the past decade, there has been intensive research and giant strides in designing algorithms for tracking humans and recognizing their actions [15, 18].

Our motivation arises in the challenge of inventing a computer system that tracks multiple persons in a video sequence and simultaneously classifies their actions. Such a computer system can be used in surveillance in order to track and detect unusual activities, and it is also very useful for sport teams to analyse the movements and actions of the players in order to improve the team strategies and the skills of the players. To accomplish these tasks, this system should be able to automatically detect persons when they appear in the video sequence, track the locations and estimate the sizes of the persons, and recognize the actions of the persons. Ultimately, such a computer system should be able to track persons under occlusion and recognize the identities of the persons throughout the video sequence.

Among these major aspects of the systems, we primarily focus on the problem of tracking and recognize the actions of the *hockey players.* Specifically, we address the problems of estimating the locations and sizes of multiple hockey players given a single video sequence and classifying their moving directions. Figure 1.1 shows the examples of the input and output of the system. The input is a video sequence from a single panning, tilting, and zooming camera; the system outputs the locations and sizes of the hockey players (represented by bounding boxes) as well as their moving directions (represented by arrows). In this thesis, we do not recognize the identities of the players and leave the problem for future work.



Frame 682 Frame 682

Frame 814 Frame 814

(a) Input (b) Output

Figure 1.1: **System Input and Output:** (a) shows the input of the system. Observe that there are multiple hockey players in the scene, and they are usually very small. Players also interact with others very frequently. Thus, partial occlusions often occur. (b) shows the output of the system. The system presented in this thesis can track and recognize multiple hockey players' actions simultaneously. The location and size of the players are specified by bounding boxes, while the actions of the players are visualized as arrows.

1.2 The Problem

This thesis focuses on two major problems: tracking multiple hockey players from a single video sequence and simultaneously recognizing their actions. The video sequences are 320×240 low-resolution hockey games originally broadcast on TV with one to fifteen targets in each frame. The sizes of the players are usually very small, ranging from 15 to 70 pixels. Moreover, the camera is *not* stationary and the sequence contains fast camera motions and significant zoom in/out. Figure 1.1 (a) shows some examples of the video sequences.

The task of visual tracking is to automatically estimate the locations and sizes of the hockey players on the image coordinate system given a video sequence. Tracking typically starts when the system detects that a hockey player appears in the video sequence. The system will store the visual information of the players (e.g., shapes and colors) as a *template*. In the next frame, the system will search for a bounding box in the image that is most similar to the template; the center and size of the bounding box are thus the location and size of the player in the next frame. Figure 1.1 (b) shows some examples of tracking multiple hockey players. The tracking problem can be divided into many sub-problems. For example, how to represent the visual cues in an informative and economic way? How to update the template when the shapes of the players change? How to efficiently and reliably estimate the locations and sizes of multiple hockey players? In the following chapters, we will tackle these sub-problems one by one.

The problem of recognizing the actions of the players can be solved if the trajectories of hockey players on the rink coordinate system are available. Since the camera is not stationary, and we do not have an accurate homography between the image and rink coordinate system, it is very difficult to obtain accurate trajectories of hockey players on the rink. As a result, we seek to classify the actions of the players based on image patches of 30 to 70 pixels heights obtained from the tracking system. Figure 1.2 gives an example of the action recognizer. The input of the

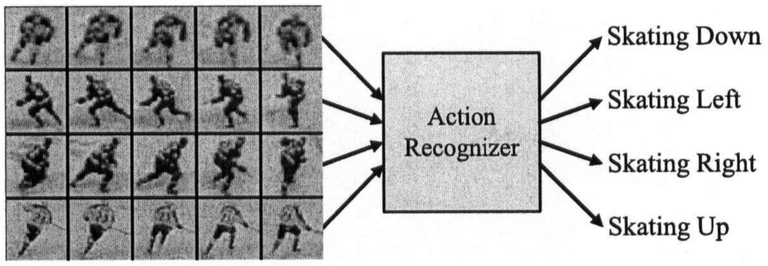


Figure 1.2: **Visual action recognizer – an example:** The input of the action recognizer is a sequence of image patches that have a single person centered in the image. The action recognizer will classify the person's action based on these image patches. The output of the action recognizer is the label of the actions. In this figure, for example, the action labels are "skating down", "skating left", "skating right", and "skating up".

action recognizer is a sequence of image patches that have a single person centered in the image. The action recognizer will utilize these image patches to classify the person's actions, and report the action labels of the person. The action labels can be long-term actions such as "shooting", "passing", and "skating", or short-term actions such as "skating left" and "skating right".

1.3 Challenges

This problem of visual tracking is challenging due to several reasons. Firstly, it is usually difficult to construct a reliable appearance model of the targets because of different viewpoints and illumination changes. The problem will be even harder if the targets are deformable objects, such as hockey players. Secondly, the scenes of the video sequences are usually very complicated. They usually contain shadows, background clutter, and other objects that may occlude the targets. Finally, the trajectories of the targets are uncertain and usually nonlinear, and thus make the prediction of the target's next location difficult. Figure 1.1 (a) illustrates the challenges of the tracking problems. We can notice that the video is low resolution, the camera is moving, the players are small and interacting, and the shapes of the players are changing.

Visual action recognition shares the same difficulties of visual tracking due to the variations of the target's appearance and the complicated scenes of the video sequences (see Figure 1.1 and 1.2 for examples). Furthermore, the actions of the targets usually have various styles and speeds, and thus make the problem of visual action recognition even harder.

1.4 Outline of Thesis

Developing an automatic tracking and action recognition system is a challenging task. In the following chapters, we show that it is, however, possible to develop an integrated system that is capable of tracking and recognizing actions of *multiple* hockey players in near real-time given a single video sequences.

In Chapter 2, we will first review some related work that tackles the problems of tracking and action recognition. We will also introduce a brief background of the template updating algorithms.

We start to present our integrated tracking and action recognition system by discussing the problem of how to represent the visual cues in an informative and economic way. In Chapter 3, we introduce the observation models used in our system: the Hue-Saturation-Value (HSV) color histogram [44] and the Histogram of Oriented Gradients (HOG) descriptor [8]. The HSV color histograms and the HOG descriptors encode the color and shape information of the image patches of hockey players, respectively. In addition, they can be very efficiently computed using integral histograms [47].

The next problem we encounter is how to update the templates when the shapes of the players change. As described in Section 1.2, the tracker searches for a bounding box in the image that is most similar to the template. Since hockey players always change their pose during a hockey game, it is impossible to track a hockey player using a fixed template. In Chapter 4, we describe a Switching

Probabilistic Principal Component Analysis (SPPCA) template updater to predict and update the templates used by the tracker. The SPPCA template updater learns the template model from training data offline. Before the tracker starts in the current frame, the SPPCA template updater will predict new templates based on previous observations. After the tracker estimates the locations and sizes of hockey players in the current frame, the image patches of hockey players will be extracted and fed back into the SPPCA template updater to update the template model. Figure 1.3 illustrates the template updating procedures.

The third problem we face is how to classify the actions of hockey players from the image patches extracted by the tracker. In Chapter 5, we present an action recognizer that takes the HOG descriptors of hockey players as input features, and classifies the HOG descriptors into action categories using a Sparse Multinomial Logistic Regression (SMLR) classifier [27]. The SMLR classifier has several desirable properties: it has a sparsity-promoted prior that increases the classification speed and reduces the generalization error, and it has no restriction on the choices of basis functions. By incorporating the SMLR classifier and the motion similarity measure introduced by Efros *et al.* [11], the action recognizer is capable of accurately and efficiently classifying players' moving direction.

The last problem is how to efficiently and reliably track the locations and sizes of multiple hockey players. In Chapter 6, we detail the Boosted Particle Filter (BPF) presented by Okuma *et al.* [44]. The BPF tracker augments the standard Particle Filter [2] by incorporating cascaded Adaboost detection [61] in its proposal distribution, and it improves the robustness of the tracker. We also describe how to combine the BPF tracker with the SPPCA template updater and the action recognizer. Figure 1.3 shows the system diagram of our algorithm.

In Chapter 7, we conclude this thesis and provide several possible research directions for future work.

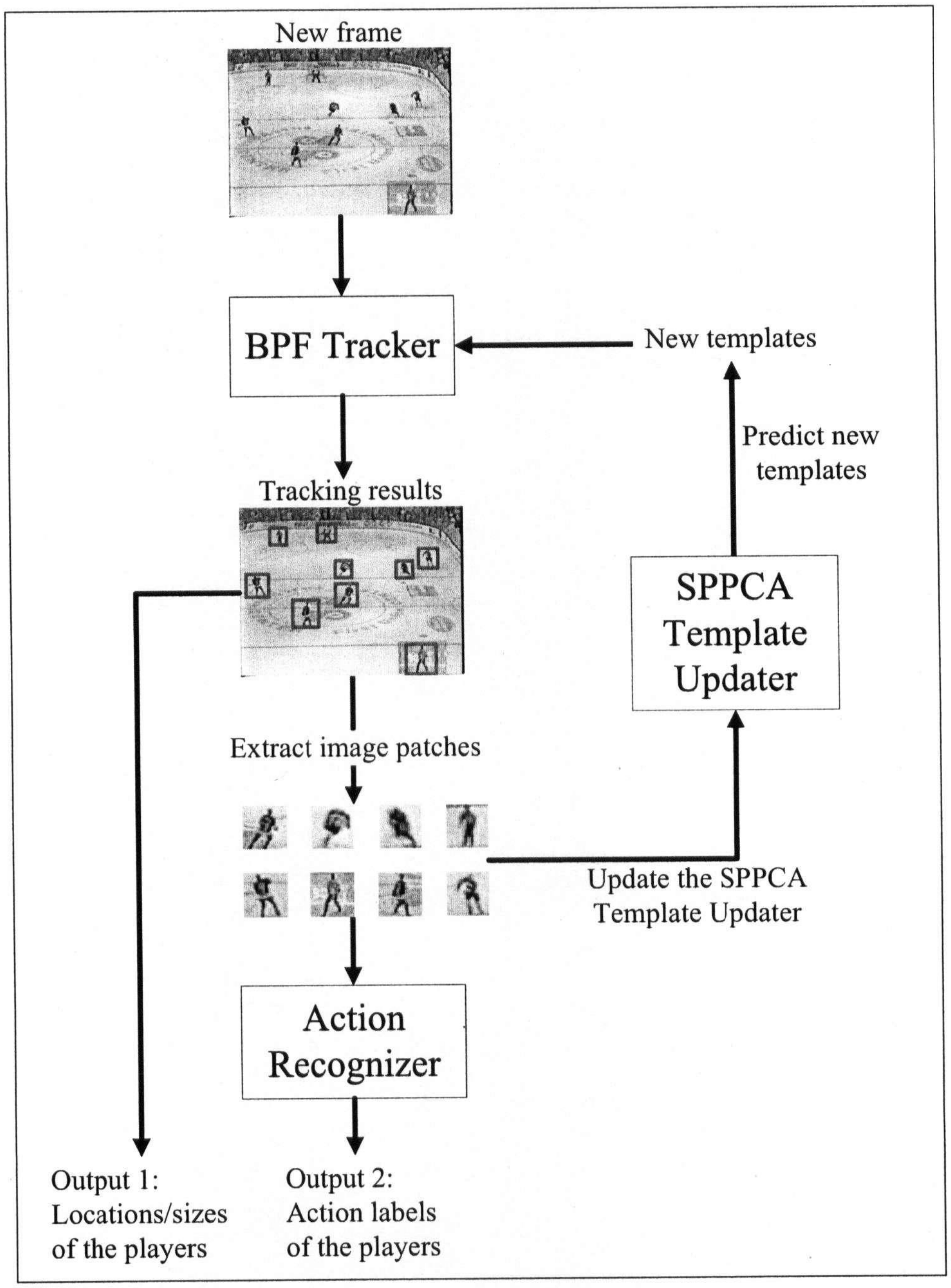


Figure 1.3: **System Diagram:** The system contains three important components: the Boosted Particle Filter (BPF) tracker, the Switching Probabilistic Principal Components Analysis (SPPCA) template updater, and the action recognizer. The following chapters will describe these three components in more detail.

Chapter 2

Related Work

2.1 Template Updating

The goal of a tracking system is to estimate the locations and sizes of the targets in a video sequence. In order to accomplish this task, the trackers have to know the appearance of the targets. A *template*, or an *exemplar*, provides the information about the appearance of the targets, and thus plays an important role in the tracking system. Unfortunately, due to the fact that the targets may be non-rigid objects and the viewpoint of the camera may change in the video, the appearance of the targets may not remain the same during tracking. Therefore, in order to reliably track the targets throughout the video sequence, a *template updating* algorithm is required to adapt the template to the newly observed appearance of the targets.

Many template updating algorithms have been developed recently. The most naïve approach uses the previous observation as the template for the tracker to find the most probable location of the target in the next frame. Though simple, this approach has problems because the estimation of the target's location inevitably has errors so that the bounding box may include the background or other objects. If we take the previous observation as the template in the next frame, the errors will accumulate and finally lead to loss of targets in the future [36]. When the target is rigid, an alternative is to first use the previous observation as the template to obtain a rough estimation of the target's location. Then, we can conduct a local search utilizing the reliable first template, and start the search from the rough estimated location in order to correct the rough estimation [36]. However, this technique does not work when the targets are deformable such as hockey players.

Toyama and Blake introduced the exemplar tracker [57]. They learned the representation and the transition of exemplars offline. During tracking, the exemplar tracker infers both the position of the target and the exemplar used by the tracker. The problem of the exemplar tracker is that the exemplars only encode a fixed number of appearance variations of the targets. In order to introduce more appearance variations, Elgammal *et al.* modeled the distribution between the exemplars and the intermediate observation using a non-parametric distribution [12].

Instead of constraining the appearance of the targets to be similar to some fixed number of templates, Black *et al.* [4] constrained the target to lie on a learned eigen-subspace. Their EigenTracking algorithm simultaneously estimates the location of the targets and the coordinates of the target's appearance in the subspace to minimize the distance between the target's appearance and the subspace. However, since the EigenTracker learns the appearance model off-line, it cannot fully capture the appearance variations of the targets. Recently, Ross *et al.* [50] and Ho *et al.* [17] proposed algorithms to efficiently update the subspace on-line.

Khan *et al.* [24] formulated the EigenTracking algorithm in a probabilistic framework, and applied it to track honey bees (Figure 2.1 (a)). Instead of using PCA, they used Probabilistic Principal Component Analysis (PPCA) [56], which is a generative model that generates the observed variable from the hidden variables by factor analysis model with isotropic Gaussian noise. They projected the appearance of the target to a single latent subspace using Probabilistic Principal Component Analysis (PPCA) [56]. Figure 2.1 (b) shows the probabilistic graphical model of their tracking system. Since the hidden variables are continuous and have high dimensionality, they used a Rao-Blackwellized particle filter [9] to infer the

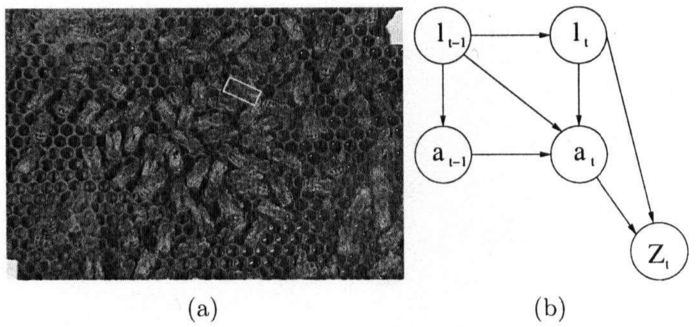


Figure 2.1: **A Rao-Blackwellized Particle Filter for EigenTracking:** (a) Khan *et al.* [24] presented the Rao-Blackwellized Particle Filter for EigenTracking to track honey bees. (b) The probabilistic graphical model of the Rao-Blackwellized Particle Filter for EigenTracking. $\boldsymbol{l}_t$ represents the location of the bee at time t, $\boldsymbol{a}_t$ represents the appearance of the bee at time t, and $\boldsymbol{Z}_t$ is the observation at time t. These figures are taken from [24].

probability of the hidden variables for greater efficiency.

Instead of using a single eigen-subspace to model the appearance of the target, researchers also tried to model the appearance by using multiple subspaces. For example, Lee *et al.* [28] presented a system that models the faces by a set of eigen-subspaces. In the learning phase, they divided the training data into groups according to the identities of the faces, and learned a single eigen-subspace for each group. During the runtime, they first recognized the identity of the face based on the history of the tracking results, and chose the eigen-subspace belonging to the recognized person. Then, they utilized the chosen eigen-subspace to generate a template for the tracker, and tracking could be performed by searching for the location whose image patch is most similar to the template.

Recently, Lim *et al.* [29] have introduced an appearance updating technique that first transforms the observation to a latent space by Locally Linear Embedding (LLE) [52], and then uses the Caratheodory-Fejer (CF) approach to predict the next position of the appearance in the latent space, and finally inversely transform the point from the latent space to the observation space by a set of Radial Basis

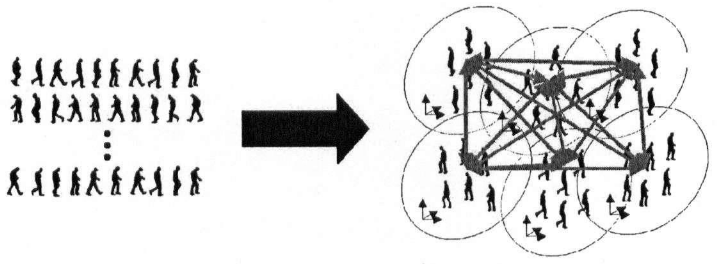
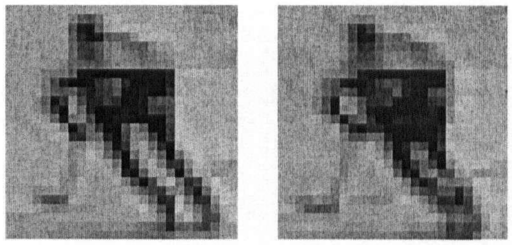
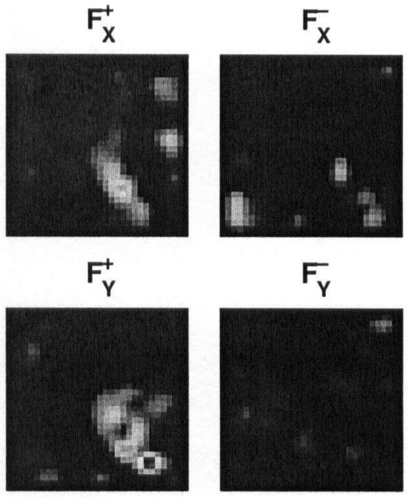


Figure 2.2: **Learning Dynamic Point Distribution Model:** A principal component analysis (PCA) is applied to in each cluster of registered shapes to obtain compact shape parameterization known as "Point Distribution Model" [7]. The transition probability between clusters is also learned. This figure is taken from [16].

functions (RBFs). The experimental results show that their approach can accurately predict the appearance of the target even under occlusions. Urtasun *et al.* [59] and Moon *et al.* [38] also transform the observation to a latent space and then predict the next appearance. Instead of using LLE and CF, they use Gaussian Process Dynamic Models (GPDM) [62] to predict and update the template of the tracker.

The work most similar to ours is Giebel *et al.* [16]. They presented a system that can track and detect pedestrians using a camera mounted on a moving car. Their tracker combines texture, shape, and depth information in their observation likelihood. The texture is encoded by the color histogram, the shape is represented by a Point Distribution Model (PDM) [7], and the depth information is provided by the stereo system. In order to capture more variations of the shape, they constructed multiple eigen-subspaces from the training data, and the transition probability between subspaces were also learned (Figure 2.2). During runtime, they used a Particle Filter to estimate the probability of the hidden variables of the tracking. To reduce the number of particles, they also used a smart proposal distribution based on the detection results. Our tracker shares the same merits. However, in the template updating part, we infer the probability of the hidden variables using the Rao-Blackwellized Particle Filter to increase the speed. In multi-target tracking, we


(a) Two consecutive frames



(b) Decomposed Optical Flow

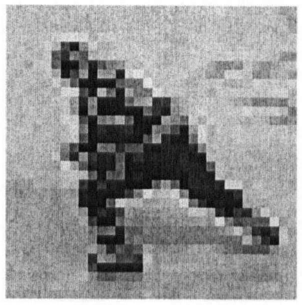
Figure 2.3: **Decomposed Optical Flow:** (a) Two consecutive frames. (b) The Decomposed Optical Flow constructed by decomposing the optical flow of (a) into four channels $(F_X^+, F_X^-, F_Y^+, F_Y^-)$, where F_X^+, F_X^-, F_Y^+, and F_Y^- represent the optical flow along the X^+, X^-, Y^+, and Y^- directions, respectively.

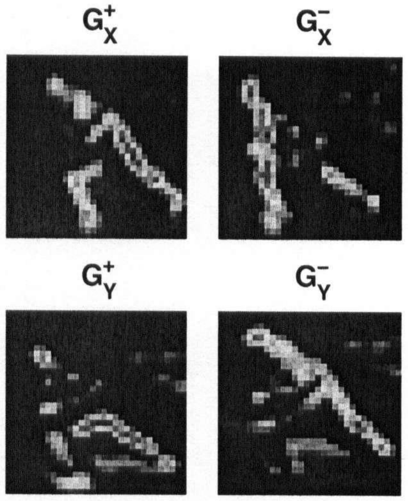
use the Boosted Particle Filter that incorporates the cascaded Adaboost detector to obtain fast and reliable detections.

2.2 Visual Action Recognition

The goal of visual action recognition is to classify the actions of persons based on a video sequence. Figure 1.2 shows an example input/output of a visual action recognition system. In this section, we briefly review the literature related to our visual action recognition system. For a more complete survey, please refer to the reviews of Gavrila [15] and Hu *et al.* [18].

Freeman *et al.* [14] utilized global orientation histograms to encode the shapes of the hands, and used a nearest neighbor classifier to determine the gesture of the hands. In [13], they further divides the images into cells, and compute the orientation histograms of all cells. However, their approach determines the gesture of the


(a) Original image



(b) Decomposed Image Gradients

Figure 2.4: **Decomposed Image Gradients:** (a) The original image. (b) The Decomposed Image Gradient constructed by decomposing the image gradients of (a) into four channels $(G_X^+, G_X^-, G_Y^+, G_Y^-)$, where G_X^+, G_X^-, G_Y^+, and G_Y^- represent the image gradient along the X^+, X^-, Y^+, and Y^- directions, respectively.

target only by the current posture of the person. No previous posture information is used.

Efros *et al.* [11] employed a motion descriptor, the Decomposed Optical Flow (DOF). The DOF descriptor can be constructed by decomposing the optical flow of two consecutive frames into four channels $(F_X^+, F_X^-, F_Y^+, F_Y^-)$, where F_X^+, F_X^-, F_Y^+, and F_Y^- represent the optical flow along the X^+, X^-, Y^+, and Y^- directions, respectively. They also presented a novel motion-to-motion similarity measure that can handle actions of different speeds. A nearest-neighbor classifier was used to determine the person's actions. Figure 2.3 shows an example of the DOF descriptor of two consecutive frames. We can see clear flow information on the legs of the hockey player.

Wu [64] extended Efros *et al.* [11] by introducing another motion descriptor, the Decomposed Image Gradients (DIG). The DIG descriptor can be constructed

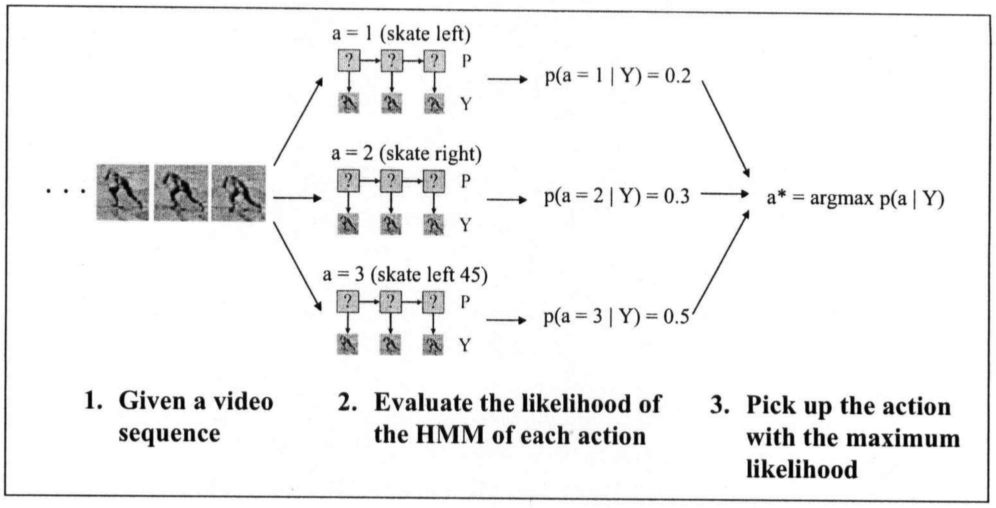


Figure 2.5: **The HMM Action Recognizer:** This figure illustrates the procedures of classifying actions of a video sequence using the HMM action recognizer. The HMM action recognizer consists of multiple HHMs, and each of them models the appearance variations of a single action. The action is determined by finding the HMM that has the maximum likelihood $p(a|\boldsymbol{Y})$ where a is the action label and $\boldsymbol{Y}$ is a video sequence. The hidden state $\boldsymbol{P}$ represents the appearance variation of the target.

by first computing the image gradients of the image, and then decomposing the image gradients into four channels $(G_X^+, G_X^-, G_Y^+, G_Y^-)$, where G_X^+, G_X^-, G_Y^+, and G_Y^- represent the image gradient along the X^+, X^-, Y^+, and Y^- directions, respectively. He also used the motion-to-motion similarity measure similar to Efros *et al.* A nearest-neighbor classifier was also used to determine the person's actions. Wu compared his action recognizer with Efros *et al.* in the hockey domain, and showed that the DIG descriptors outperform the DOF descriptors in classification accuracy. Figure 2.4 shows an example of the DIG descriptor of a single image.

The problem of action recognition can be also formulated in a generative probabilistic model. For example, [12, 66] used Hidden Markov Models (HMMs) to recognize the target's action. In their system, they trained separate HMMs for each action. The hidden state of the HMM represents the appearance variations of the target and the observation is either raw images or the target's contour. During recognition, they fed the entire video sequence to all HMMs and the actions of the target is determine by the HMM having the maximum likelihood (see Figure 2.5 for an example). In our previous work, we also employed HMMs to recognize the target's actions [32, 33]. Instead of using the entire video sequence, we used a fixed-length sliding window to determine the target's actions.

2.3 Object Tracking

Automated tracking of multiple objects is still an open problem in many settings, including car surveillance [26], sports [37, 42] and smart rooms [20] among many others [19, 22, 35]. In general, the problem of tracking visual features in complex environments is fraught with uncertainty [20]. It is therefore essential to adopt principled probabilistic models with the capability of learning and detecting the objects of interest.

Over the last few years, particle filters, also known as condensation or sequential Monte Carlo, have proved to be powerful tools for image tracking [10, 21, 46, 51]. The strength of these methods lies in their simplicity, flexibility, and systematic treatment of nonlinearity and non-Gaussianity.

Various researchers have attempted to extend particle filters to multi-target tracking. Among others, Hue *et al.* [19] developed a system for multi-target tracking by expanding the state dimension to include component information, assigned by a Gibbs sampler. They assumed a fixed number of objects. To manage a varying number of objects efficiently, it is important to have an automatic detection process. The Bayesian Multiple-BLob tracker (BraMBLe) [22] is an important step in this direction. BraMBLe has an automatic object detection system that relies on modeling a fixed background. It uses this model to identify foreground objects (targets). With the Boosted Particle Filter (BPF) in [44], we can relax this assumption of a fixed background in order to deal with realistic TV video sequences, where the background changes.

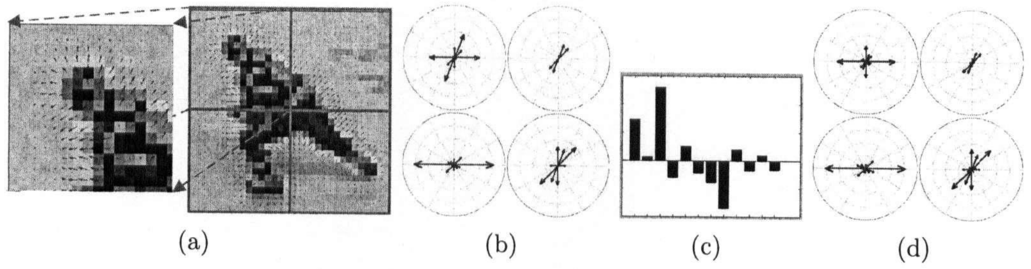
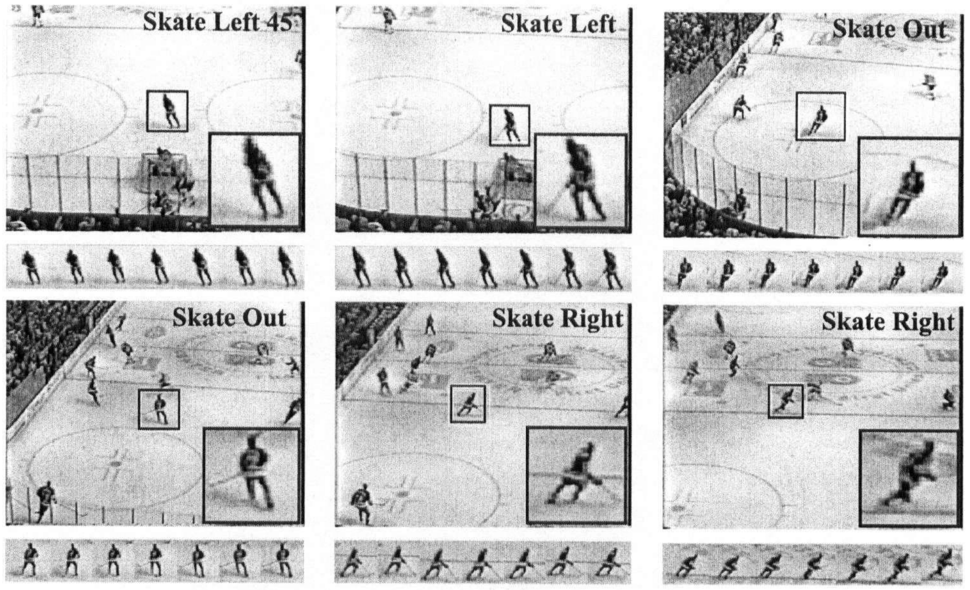
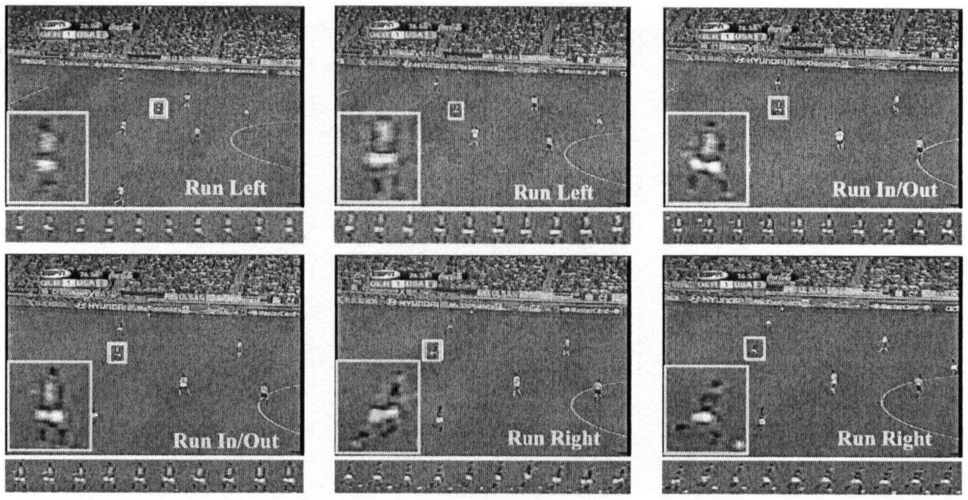


Figure 2.6: **The PCA-HOG descriptor:** (a) The image gradient. (b) The HOG descriptor with a 2×2 grid and 8 orientation bins. (c) The PCA-HOG descriptor with 12 principal components. (d) The reconstructed HOG descriptor.

In our previous work [32, 33], we presented a system that can simultaneously track and recognize a single target's action. The observation model we used is either the Histograms of Oriented Gradients (HOG) descriptors [33], or the PCA-HOG descriptor constructed by applying the Principal Component Analysis to the HOG descriptors [32]. Figure 2.6 shows an example of the PCA-HOG descriptor. Tracking was performed by a particle filtering tracker similar to [46]. After estimating the location of the target, we extracted the image patch at the estimated location and fed the new observation to a HMM action recognizer described in Section 2.2. After estimating the target's action, we updated the template of the tracker by using a HMM template updater similar to the exemplar tracker [57]. Unlike the exemplar tracker, we divided the templates of the target into groups according to their actions, and trained a single HMM template updater for each group. Then, we could utilize the estimated action to select the HMM template updater in the next frame and thus reduced the number of possible templates in the next frame. Figure 2.7 shows some experimental results of [32] in tracking and recognizing a single player's action using both hockey and soccer sequences.



(a) Hockey Player



(b) Soccer Player

Figure 2.7: **Experimental results of tracking and recognizing a single player:** The upper parts of the images are the video frames and the lower parts are the most recent observations used to classify the player's actions. (a) Tracking hockey players with changing poses and actions. (b) Tracking the soccer player who is running with a ball.

Chapter 3

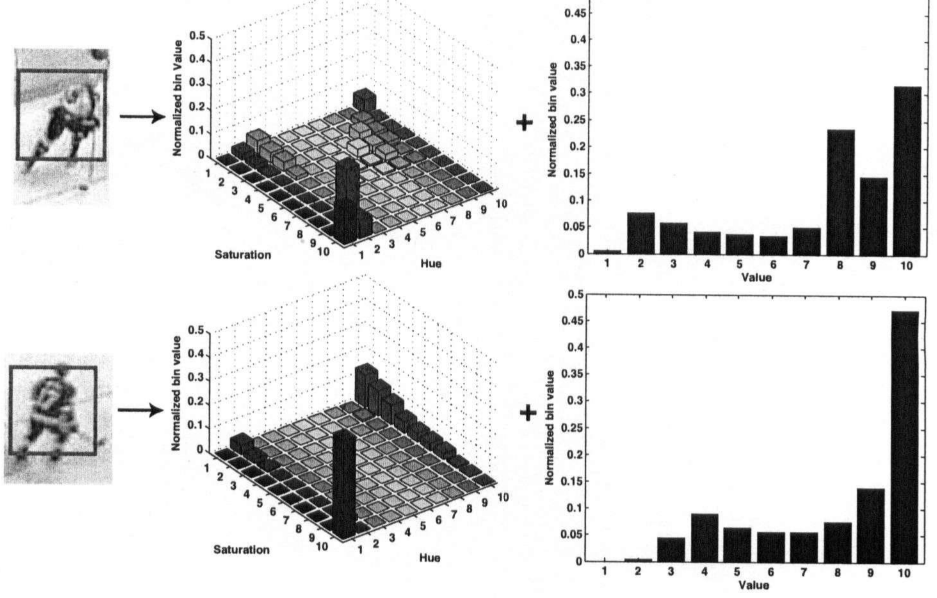
Observation Models

Observation models encode the visual information of the target's appearance. Since a single cue does not work in all cases, many researchers have combined multiple cues for robust tracking [3, 16, 65, 67]. In this thesis, we utilize the Hue-Saturation-Value (HSV) color histogram to capture the color information of the target, and the Histogram of Oriented Gradients (HOG) descriptors [8] to encode the shape information. The following sections describe the color and shape observation models in more detail.

3.1 Color

We encode the color information of the targets by a two-part color histogram based on the Hue-Saturation-Value (HSV) color histogram used in [44, 46]. We use the HSV color histogram because it decouples the intensity (i.e., value) from color (i.e., hue and saturation), and it is therefore more insensitive to illumination effects than using the RGB color histogram. The exploitation of the spatial layout of the color is also crucial due to the fact that the jersey and pants of hockey players usually have different colors [44, 46].

Our color observation model is composed of 2-D histogram based on Hue and Saturation and 1-D histogram based on value. We assign the same number



Color histogram of a player (TOP: white uniform BOTTOM: red uniform)

Figure 3.1: **HSV Color histograms:** This figure shows two different color histograms of selected rectangular regions. The first 2D histograms are Hue and Saturation histograms. The other 1D histograms are Value histograms. Both 2D and 1D histograms have Z axis and Y axis respectively for the normalized bin value. The player on top has uniform whose color is the combination of dark blue and white and the player on bottom has a red uniform. Although one can clearly see concentrations of color bins due to limited number of colors, this figure shows a clear color distinction between two players.

of bins for each color component, i.e., $N_h = N_s = N_v = 10$, and it results in a $N_h \times N_s + N_v = 10 \times 10 + 10 = 110$ dimension HSV histogram. Figure 3.1 shows two instances of the HSV color histograms.

3.2 Shape

We apply the Histograms of Oriented Gradient (HOG) descriptor [8] to encode the shape information of the targets. The HOG descriptor is computed by sampling a set of the SIFT descriptors [31] with a fixed spacing over the image patches.

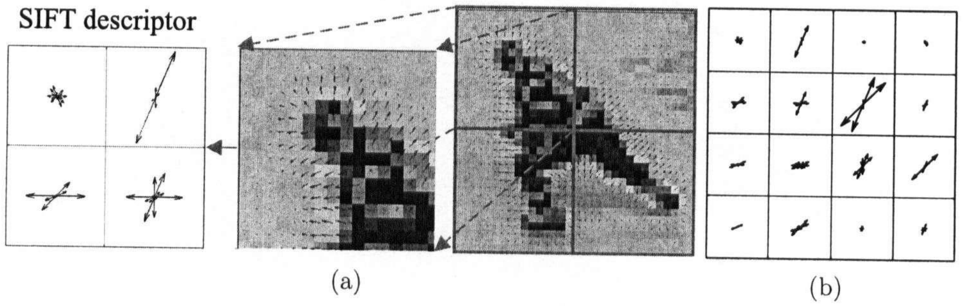


Figure 3.2: **The HOG descriptor:** (a) RIGHT: The image gradient of a 32×32 image. CENTER: A block of size 16×16. LEFT: The SIFT descriptor of the block with $n_w = n_h = 2$, $n_b = 8$. (b) The HOG descriptor of the image computed by sampling SIFT descriptors of size 16×16 with a 16-pixels spacing.

Incorporating with a Support Vector Machine (SVM) classifier, the HOG descriptor has been shown to be very successful in the state-of-the-art pedestrian detection system [8]. In this thesis, we employ the HOG descriptor because it is robust under viewpoint and lighting changes, possesses good discriminative power, and can be efficiently computed.

The SIFT descriptor was originally introduced by Lowe [31] to capture the appearance information centered on the detected SIFT features. To compute the SIFT descriptor, we first smooth the image patch by a Gaussian low-pass filter and compute the image gradients using a $[-1, 0, 1]$ kernel. The original SIFT descriptor implementation [31] rotated the directions of the gradients to align the dominating orientation of the SIFT features in order to have a rotation-invariant local descriptor. In our case, however, we do not rotate the directions of the gradients because the dominating orientation provides crucial information for the tracking and action recognition system. After computing the image gradients, we divide the image patch into small spatial regions ("cells"), for each cell accumulating a local 1-D histogram of gradient directions over the pixels of the cell. In this thesis, we use the *unsigned* image gradient, and the orientation bins are evenly spaced over 0°–180° to make the descriptor more invariant to the color of the players' uniforms. For better invariance to lighting changes, we normalize the local response by the total histogram energy accumulated over all cells across the image patch (the L_1 norm).

The HOG descriptor is constructed by uniformly sampling the SIFT descriptor of the same size over the image patch with a fixed spacing ("stride"). There is no constraint on the spacing between the SIFT descriptors and therefore these SIFT descriptors may be overlapped. The aggregation of all these SIFT descriptors forms the HOG descriptor of the image patch.

In summary, the HOG descriptor is constructed by densely sampling the SIFT descriptors of the same size $\eta \times \eta$ over a image patch of size $p_w \times p_h$ (η, p_w, p_h are measured in number of pixels). We divide each SIFT descriptor into $n_w \times n_h$ cells, in which an n_b histogram of oriented gradients is computed. Figure 3.2 shows an example of the HOG descriptor.

3.3 Efficient Implementation

Since the observations are all histogram-based features, and we do not apply any spatial weighting function to the tracking region, we can use Integral Histograms [47] to compute the observation very efficiently.

Suppose we want to compute an N bins histogram, the Integral Histograms technique constructs N Integral Images [61] for each bin of the histogram of the entire image. After constructing those integral images, the histogram of a given rectangle region can be computed in constant time.

The original integral histogram implementation is designed for object detection [47], and it computes integral histograms of the entire image because the object can be detected in any position. However, in the case of object tracking, the targets are supposed to appear in a small portion of the image, they usually around the locations of the targets in the previous frame. Therefore, it is uneconomic to compute the integral histogram of the entire image. In this thesis, we propose to use *local integral histograms* which constructs integral histograms around a small region around the targets. Experimental results show that the local integral histograms work 10 times faster than global integral histograms.

Chapter 4

Template Updating

4.1 Introduction

Tracking is usually performed by searching for the location in the image that is similar to a given *template*. If the target is a rigid object, i.e., the shape of the target remains the same over time, we can use the image patch of the target in the first frame as the template. Then, either a deterministic or a probabilistic algorithm can be employed to search for the location in the image that is similar to the image patch of the target in the first frame.

However, when the targets are non-rigid objects, the problem becomes more challenging because the shape of the targets changes constantly, and a template updating mechanism is needed to adapt the template of the tracker over time.

The most naïve method to update the template is to use the image patch of the previous estimated location of the target. Unfortunately, this method usually causes problems because the location estimates of the targets inevitably contain some errors. Thus, the image patch of the previous estimated location of the target may contain only a part of the targets, background pixels, or even other objects. As a result, when we trust the previous estimated location and use the image patch of the previous estimates as the template, the current location estimates of the targets will also inevitably have errors. More severely, these errors will accumulate quickly

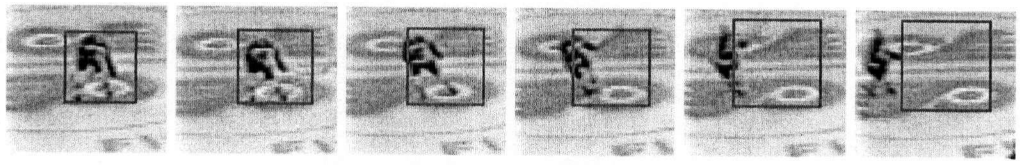


Figure 4.1: **Tracking with a naïve template updater:** This figure shows the results of tracking a single hockey player with a naïve template updater, i.e., update the template using the image patch of the previous location estimates. Since the location estimates inevitably have errors, the template is gradually polluted by the background in (c),(d),(e), and the tracker finally loses its targets in (f).

and finally lead to loss of targets. Fig 4.1 gives an example of applying the naïve updating method in tracking. We can observe that the tracker quickly fails to track the targets.

In this thesis, we introduce the Switching Probabilistic Principal Component Analysis (SPPCA) model to update the templates of the tracker. In particular, we utilize the SPPCA model to update the templates of the HOG descriptors because the HOG descriptors of the hockey players change constantly due to the changes of poses. Notice that we do not apply the SPPCA model to update the templates of the color histograms in this thesis because the colors of the hockey players usually do not change over time. In other applications, however, the SPPCA model could be used for generating templates for color histograms or even raw images.

The main idea of the SPPCA template updater is to update the template such that the new template will be similar to the image patch of the previous estimated location of the target, but it is also restricted to be similar to the training data. This can be done by EigenTracking [4], which learns a eigen-space of the image patches of the targets off-line, and then forces the template to lie on the eigen-space during runtime [4]. Khan *et al.* have introduced a probabilistic extension of the EigenTracking [24], where they learn a single linear latent space from the training data. However, the actual distribution of the data usually lies on a nonlinear subspace, and cannot be fully captured by a single linear subspace. In this thesis,

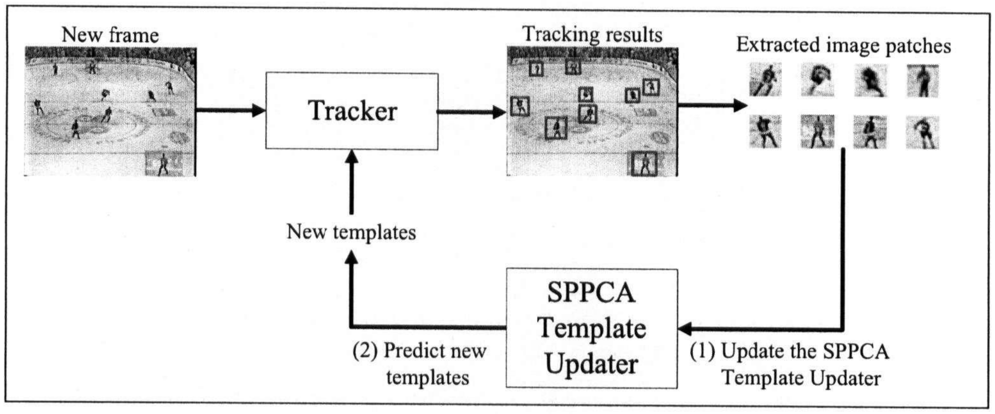


Figure 4.2: **The SPPCA Template Updater:** This figure visualizes the operations of the SPPCA template updater. During tracking, the SPPCA template updater has two operations: *updating* and *prediction*. After the locations and sizes of the targets are estimated, the tracker will extract image patches centered in the estimated locations. These image patches are used to *update* the template models of the SPPCA template updater. Then, the *prediction* operation generates a set of new templmates based on the current observations. These templates will be utilized by the tracker in the next frame to search for the locations and sizes of the targets.

we use a set of linear subspaces to approximate the nonlinear one by introducing an additional "switch" to select the subspace. The idea can be seen as an extension of a mixture of PPCA [55] and a variant of the the Switching Linear Dynamical System (SLDS) [40].

The SPPCA template updater consists of three major operations: learning, updating, and prediction. Before discussing the details of the three operations, we will first describe the probabilistic graphical model of the SPPCA in Section 4.2,. The *learning* operation learns the template models off-line from a set of training data. Section 4.3 will present the Expectation-Maximization (EM) algorithm [41] that learns the parameters of the SPPCA model. During tracking, the SPPCA template updater performs two operations: *updating* and *prediction*. As shown in Figure 4.2, the tracker extracts image patches centered in the estimated locations after tracking the locations and sizes of the targets. These image patches are used to *update* the template models of the SPPCA template updater. Section 4.4 will detail the Rao-Blackwellized Particle Filtering algorithm that is utilized to update the template models. The *prediction* operation presented in Section 4.5 generates a set of new templates based on the current observations, and these templates will be utilized by the tracker in the next frame to search for the locations and sizes of the targets. Experimental results are shown in Section 4.6.

4.2 Probabilistic Graphical Model

Let $s_t \in \{1, \ldots, n_s\}$ be a discrete random variable representing the subspace we use at time t, $\boldsymbol{y}_t \in \mathbb{R}^{n_y}$ be a continuous random variable representing the observation at time t, and $\boldsymbol{z}_t \in \mathbb{R}^{n_z}$ be a continuous random variable representing the coordinate of the observation on the subspace. The probabilistic graphical model of an SPPCA model is shown in Figure 4.3.

When $t > 1$, the dynamics between s_t and s_{t-1} is defined as

$$p(s_t|s_{t-1}) = \boldsymbol{\Phi}(s_t, s_{t-1}) \tag{4.1}$$

where $\boldsymbol{\Phi}$ is a $n_s \times n_s$ transition matrix where $\boldsymbol{\Phi}(i,j) = p(s_{t+1} = j|s_t = i)$ denoting the probability transition from s_{t-1} to s_t. When $t = 1$, s_1 is generated from a initial distribution

$$p(s_1) = \boldsymbol{v}_s(s_1) \tag{4.2}$$

where $\boldsymbol{v}_s(s_1)$ is the initial distribution.

The dynamics between $\boldsymbol{z}_t$ and $\boldsymbol{z}_{t-1}$ can be divided into two cases. When we update the template using the same subspace, i.e., $s_t = s_{t-1}$, we can generate $\boldsymbol{z}_t$ according to

$$\boldsymbol{z}_t = \boldsymbol{A}_{s_t}\boldsymbol{z}_{t-1} + \boldsymbol{Q}_{s_t} \tag{4.3}$$

$\boldsymbol{A}_{s_t}$ is the system matrix parameterized by s_t, and $\boldsymbol{Q}_{s_t}$ is a zero-mean Gaussian noise such that $\boldsymbol{Q}_{s_t} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{V}_{s_t})$ where $\boldsymbol{V}_{s_t}$ is the system covariance matrix parameterized by s_t.

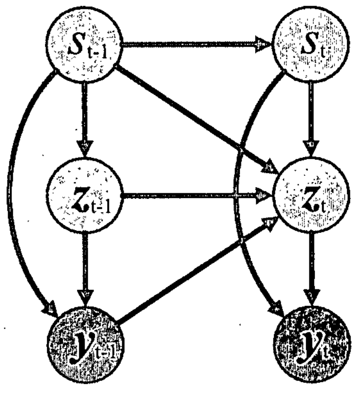


Figure 4.3: **Probabilistic Graphical Model of a SPPCA Model** The probabilistic graphical model of a Switching Probabilistic Principal Component Analysis (SPPCA) of time $t-1$ and t. The continuous random variable $\boldsymbol{y}$ is the observation, while s is a discrete random variable representing the subspace and $\boldsymbol{z}$ is a continuous random variable representing the coordinates of $\boldsymbol{y}$ on the subspace.

When we switch from one subspace to another, i.e., $s_t \neq s_{t-1}$, we re-initialize $\boldsymbol{z}_t$ by projecting $\boldsymbol{y}_{t-1}$ into s_t's subspace

$$\boldsymbol{z}_t = \boldsymbol{\Gamma}_{s_t} \boldsymbol{y}_{t-1} + \boldsymbol{\Lambda} \tag{4.4}$$

where $\boldsymbol{\Gamma}_{s_t}$ is the inverse observation matrix parameterized by s_t, and $\boldsymbol{\Lambda}$ is a Gaussian noise such that $\boldsymbol{\Lambda} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ where $\boldsymbol{I}$ is an identity matrix. When $t = 1$, $\boldsymbol{z}_1$ is generated from a initial Gaussian distribution

$$p(\boldsymbol{z}_1) = \mathcal{N}(\boldsymbol{z}_0, \boldsymbol{\Sigma}_0) \tag{4.5}$$

where $\boldsymbol{z}_0$ and $\boldsymbol{\Sigma}_0$ is the mean and covariance of the initial Gaussian distribution, respectively.

The current observation $\boldsymbol{y}_t$ can be generated from $\boldsymbol{z}_t$ using the following equation:

$$\boldsymbol{y}_t = \boldsymbol{C}_{s_t} \boldsymbol{z}_t + \boldsymbol{\mu}_{s_t} + \boldsymbol{R}_{s_t} \tag{4.6}$$

$\boldsymbol{C}_{s_t}$ is the observation matrix parameterized by s_t, $\boldsymbol{\mu}_{s_t}$ is the mean value of the subspace, and $\boldsymbol{R}_{s_t}$ is the Gaussian noise such that $\boldsymbol{R}_{s_t} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{W}_{s_t})$ where $\boldsymbol{W}_{s_t}$ is the observation covariance matrix parameterized by s_t.

4.3 Learning

The expectation-maximization (EM) algorithm [41] can be used to learn the maximum-likelihood parameters $\hat{\boldsymbol{\Omega}} = \{\hat{\boldsymbol{\Phi}}, \hat{\boldsymbol{v}}_s, \hat{\boldsymbol{A}}, \hat{\boldsymbol{C}}, \hat{\boldsymbol{\Gamma}}, \hat{\boldsymbol{V}}, \hat{\boldsymbol{W}}, \hat{\boldsymbol{\mu}}, \hat{z}_0, \hat{\boldsymbol{\Sigma}}_0\}$ by maximizing the likelihood $p(\boldsymbol{y}_{1:T}|\boldsymbol{\Omega})$

$$\hat{\boldsymbol{\Omega}} = \underset{\boldsymbol{\Omega}}{arg\,max}\ p(\boldsymbol{y}_{1:T}|\boldsymbol{\Omega}) \tag{4.7}$$

where $\boldsymbol{y}_{1:T}$ is the training sequence of length T.

The EM algorithm iteratively performs the following two procedures:

- E-step: The E-step computes the expectation of the the hidden states $s_{1:T}$ and $\boldsymbol{z}_{1:T}$ given the observation $\boldsymbol{y}_{1:T}$ and the parameters $\boldsymbol{\Omega}^i$ in the i-th iteration, i.e.,

$$f^i(s_{1:T}, \boldsymbol{z}_{1:T}) = p(s_{1:T}, \boldsymbol{z}_{1:T}|\boldsymbol{y}_{1:T}, \boldsymbol{\Omega}^i) \tag{4.8}$$

- M-step: The M-step maximizes the expected log likelihood with respect to the parameters $\boldsymbol{\Omega}$, i.e.,

$$\boldsymbol{\Omega}^{i+1} = \underset{\boldsymbol{\Omega}}{arg\,max}\ \langle log\ p(s_{1:T}, \boldsymbol{z}_{1:T}, \boldsymbol{y}_{1:T}|\boldsymbol{\Omega}^i)\rangle_{f^i(s_{1:T}, \boldsymbol{z}_{1:T})} \tag{4.9}$$

  $\langle\cdot\rangle_p$ denotes the expectation of a function $(\cdot)$ under a distribution p.

To avoid the problem of sticking into poor local optimum, we perform the EM algorithm in the same training data for 10 times, and the parameters with the maximum likelihood will be stored. In the following, we will describe the initialization procedures, E-step, and M-step in more detail.

4.3.1 Initialization

To initialize the parameters for the EM algorithm, we apply k-means clustering [23] to partition the training data $\boldsymbol{y}_{1:T}$ into n_s groups. For each group, we employ PPCA as described in Section 4.3.3 to estimate the initial value of $\boldsymbol{C}_i$, $\boldsymbol{u}_i$, $\boldsymbol{\Gamma}_i$, and $\boldsymbol{R}_i$ for $i = 1$ to n_s. $\boldsymbol{A}$, $\boldsymbol{V}$, and $\boldsymbol{\Sigma}_0$ are initialized as identical matrices, and $\boldsymbol{\mu}_0$ is initialized as a zero vector. The transition matrix $\mathbf{\Phi}$ and the initial distribution $\boldsymbol{v}_s(s_0)$ are initialized by a uniform distribution.

4.3.2 E-step

Exact inference in SPPCA is intractable. Therefore, people seek an approximation algorithm to tackle this problem [9, 40, 45]. In this thesis, we use Viterbi Inference (VI) [45] because it is simple and fast, and it usually has acceptable quality of estimation [43].

In the i-th iteration of the EM algorithm, the Viterbi inference approximates the joint posterior over the hidden state $s_{1:T}$ and $\boldsymbol{z}_{1:T}$ by a peaked posterior over $\boldsymbol{z}_{1:T}$ with an obtained pseudo-optimal label sequence $\hat{s}^i_{1:T}$:

$$\begin{aligned} p(s_{1:T}, \boldsymbol{z}_{1:T}|\boldsymbol{y}_{1:T}, \mathbf{\Omega}^i) &= p(\boldsymbol{z}_{1:T}|s_{1:T}, \boldsymbol{y}_{1:T}, \mathbf{\Omega}^i)\ p(s_{1:T}|\boldsymbol{y}_{1:T}, \mathbf{\Omega}^i) \\ &\approx p(\boldsymbol{z}_{1:T}|s_{1:T}, \boldsymbol{y}_{1:T}, \mathbf{\Omega}^i)\ \delta(\hat{s}^i_{1:T}) \end{aligned} \tag{4.10}$$

where $\delta(\cdot)$ is the Dirac-delta function. The pseudo-optimal sequence $\hat{s}^i_{1:T}$ can be computed exactly and efficiently using the Viterbi algorithm [45]. Algorithm 1 summarizes the procedures of the E-step.

Algorithm 1 E-step using the Viterbi Inference

```
Input: $\boldsymbol{y}_{1:T}$, $\boldsymbol{\Omega}$
Output: $\hat{s}_{1:T}$, $\bar{z}_{1:T}$, $\bar{\boldsymbol{\Sigma}}_{1:T}$
for $i = 1$ to $n_s$ do
    $[\bar{z}_1^i, \bar{\boldsymbol{\Sigma}}_1^i, L_1^i] = KalmanInitialize(\boldsymbol{y}_1, \boldsymbol{z}_0(i), \boldsymbol{\Sigma}_0(i), \boldsymbol{\Theta}_i)$
    $J(1,i) = L_1^i + \log \upsilon_s(i)$
end for

for $t = 2$ to $T$ do
    for $i = 1$ to $n_s$ do
        for $j = 1$ to $n_s$ do
            if $i = j$ then
                $[\bar{z}_t^{i,j}, \bar{\boldsymbol{\Sigma}}_t^{i,j}, L_t^{i,j}] = KalmanUpdate(\boldsymbol{y}_t, \boldsymbol{z}_{t-1}^i, \boldsymbol{\Sigma}_{t-1}^i, \boldsymbol{\Theta}_j)$
            else
                $\tilde{z}_t^i = \boldsymbol{\Gamma}_j(\boldsymbol{y}_{t-1} - \boldsymbol{\mu}_j)$
                $[\bar{z}_t^{i,j}, \bar{\boldsymbol{\Sigma}}_t^{i,j}, L_t^{i,j}] = KalmanInitialize(\boldsymbol{y}_1, \tilde{z}_t^{(i)}, \boldsymbol{I}, \boldsymbol{\Theta}_j)$
            end if
            if $J(t,j) < J(t-1,i) + L_t^{i,j} + \log \boldsymbol{\Phi}(i,j)$ then
                $J(t,j) = J(t-1,i) + L_t^{i,j} + \log \boldsymbol{\Phi}(i,j)$
                $B(t,j) = i$
                set $[\boldsymbol{z}_t^i, \boldsymbol{\Sigma}_t^i] = [\bar{z}_t^{i,j}, \bar{\boldsymbol{\Sigma}}_t^{i,j}]$
            end if
        end for
    end for
end for
$\hat{s}_T = \underset{j=1\ldots n_s}{arg\,max}\ J(T,j)$
for $t = T-1$ to $1$ do
    $\hat{s}_t = arg\,max\ J(t+1, \hat{s}_{t+1})$
end for

$[\bar{z}_{1:T}, \bar{\boldsymbol{\Sigma}}_{1:T}] = KalmanSmoothing(\boldsymbol{y}_{1:T}, \hat{s}_{1:T}, \boldsymbol{z}_0, \boldsymbol{\Sigma}_0, \boldsymbol{\Theta})$
```

4.3.3 M-step

The M-step maximizes the expected log likelihood with respect to the parameters. The parameters we want to learn include the transition matrix $\boldsymbol{\Phi}$, the system matrix and covariance $\{\boldsymbol{A}, \boldsymbol{V}\}$, the observation matrix, mean, and covariance $\{\boldsymbol{C}, \boldsymbol{\mu}, \boldsymbol{W}\}$, and the initial distribution $\boldsymbol{v}_s(s_0)$, $\boldsymbol{z}_0$ and $\boldsymbol{\Sigma}_0$. In this thesis, we assume that the current template is normally distributed around the previous template with a fixed covariance matrix, i.e., $\boldsymbol{A}_i = \boldsymbol{I}$ for $i = 1$ to n_s and $\boldsymbol{V}_i = \boldsymbol{I}$ for $i = 1$ to n_s. In other words, we want the current template to be similar to the image patch of the previous location estimates of the target, and it should be also similar to the training data and thus lie on the learned subspace.

Algorithm 2 summarizes the algorithm of the M-step. Briefly, after collecting the sufficient statistics in the E-step, $\{\boldsymbol{\Phi}, \boldsymbol{v}\}$ can be estimated by counting the frequency; and $\{\boldsymbol{z}_0, \boldsymbol{\Sigma}_0\}$ can be estimated by fitting a Gaussian distribution; $\{\boldsymbol{C}, \boldsymbol{\Gamma}, \boldsymbol{\mu}, \boldsymbol{W}\}$ can be estimated by PPCA [55].

Algorithm 2 M-step

```
**Input:** $\boldsymbol{y}_{1:T}$, $\hat{s}_{1:T}$, $\bar{\boldsymbol{z}}_{1:T}$, $\bar{\boldsymbol{\Sigma}}_{1:T}$
**Output:** $\boldsymbol{\Phi}, \boldsymbol{v}, \boldsymbol{z}_0, \boldsymbol{\Sigma}_0, \boldsymbol{C}, \boldsymbol{\Gamma}, \boldsymbol{\mu}, \boldsymbol{W}$
  estimate $\boldsymbol{\Phi}$ and $\boldsymbol{v}_s$ by counting the frequencies of $\hat{s}_{1:T}$ [40]
  for $i = 1$ to $n_s$ do
    estimate $\boldsymbol{z}_0$ and $\boldsymbol{\Sigma}_0$ using the technique in [40]

    $\tilde{\pi}_i = (1/T)\sum_{t=1}^{T} \hat{s}_t^i$,   $\boldsymbol{\mu}_i = (\sum_{t=1}^{T} \hat{s}_t^i \boldsymbol{y}_t)/(\sum_{t=1}^{T} \hat{s}_t^i)$
    $\boldsymbol{S}_i = (1/(\tilde{\pi}_i T))\sum_{t=1}^{T} \hat{s}_t^i (\boldsymbol{y}_t - \boldsymbol{\mu}_i)(\boldsymbol{y}_t - \boldsymbol{\mu}_i)'$

    $d = n_y$,   $q = n_x$
    $\{\lambda_1, \ldots, \lambda_d\} = EigenValue(\boldsymbol{S}_i)$,   $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_d\} = EigenVector(\boldsymbol{S}_i)$
    $\boldsymbol{U}_q = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_q]$,   $\boldsymbol{\Lambda}_q = diag([\lambda_1, \ldots, \lambda_q])$
    $\sigma_i^2 = \frac{1}{d-q}\sum_{j=q+1}^{d} \lambda_j$

    $\boldsymbol{C}_i = \boldsymbol{U}_q(\boldsymbol{\Lambda}_q - \sigma^2 \boldsymbol{I}_q)$
    $\boldsymbol{W}_i = \sigma_i^2 \boldsymbol{I}_d$
    $\boldsymbol{M}_i = \boldsymbol{C}_i' \boldsymbol{C} + \sigma_i^2 \boldsymbol{I}_q$
    $\boldsymbol{\Gamma}_i = \boldsymbol{M}_i^{-1} \boldsymbol{C}_i'$
  end for
```

4.4 Updating

The updating operation is employed to update the template model after observing the new appearance of players. In this thesis, we utilize the Rao-Blackwellized Particle Filter (RBPF) [9, 39] to efficiently update the hidden states of the SPPCA model. In Section 4.4.1, we will briefly introduce the Particle Filtering algorithm [2]. Section 4.4.2 will describes Rao-Blackwellized Particle Filtering in more detail.

4.4.1 Particle Filtering

Particle filtering [2] has been widely applied to systems with nonlinear/non-Gaussian distributions. Since exact inference in the SPPCA model is intractable, we can apply particle filtering to estimate the posterior distribution over $\{s_t, \boldsymbol{z}_t\}$. In particle filtering, we represent the posterior distribution over $\{s_t, \boldsymbol{z}_t\}$ by a weighted set of particles $\{s_t^{(i)}, \boldsymbol{z}_t^{(i)}, w_t^{(i)}\}_{i=1}^N$. The posterior distribution can be approximated by

$$p(s_t, \boldsymbol{z}_t | s_{1:t-1}, \boldsymbol{z}_{1:t-1}, y_{1:t}) \approx \sum_{i=1}^{N} w_t^{(i)} \, \delta_{[s_t^{(i)}, \boldsymbol{z}_t^{(i)}]}([s_t, \boldsymbol{z}_t]) \tag{4.11}$$

where $w_t^{(i)}$ is the weight associated with the i-th particle, and it can be updated incrementally by

$$w_t^{(i)} = w_{t-1}^{(i)} \frac{p(\boldsymbol{y}_t | s_t^{(i)}, \boldsymbol{z}_t^{(i)}) p(s_t^{(i)}, \boldsymbol{z}_t^{(i)} | s_{t-1}^{(i)}, \boldsymbol{z}_{t-1}^{(i)})}{q(s_t^{(i)}, \boldsymbol{z}_t^{(i)} | s_{1:t-1}^{(i)}, \boldsymbol{z}_{1:t-1}^{(i)}, \boldsymbol{y}_{1:t})} \quad \text{for } i = 1 \dots N \tag{4.12}$$

The details of the particle filtering algorithm can be found in [2].

However, particle filtering is also known to be inefficient when the dimensionality of the hidden states is large due to the requirement to have a sufficient number of particles to approximate the distribution of a large feature space (the number of particles required is exponential to the feature space). Furthermore, the state estimation of a particle filter might have poor quality without a good proposal distribution $q(s_t^{(i)}, \boldsymbol{z}_t^{(i)} | s_{1:t-1}^{(i)}, \boldsymbol{z}_{1:t-1}^{(i)}, \boldsymbol{y}_{1:t})$ [2].

4.4.2 Rao-Blackwellized Particle Filtering

In this thesis, instead of using a particle filter similar to Giebel *et al.* [16], we employ the Rao-Blackwellized Particle Filter (RBPF) [9, 39] for online filtering. The key idea of the RBPF is that it factorizes the joint posterior distribution over $\{s_t, \boldsymbol{z}_t\}$ into two parts

$$p(s_t, \boldsymbol{z}_t | s_{1:t-1}, \boldsymbol{z}_{1:t-1}, \boldsymbol{y}_{1:t}) = p(\boldsymbol{z}_t | \boldsymbol{z}_{1:t-1}, s_{1:t}, \boldsymbol{y}_{1:t})\; p(s_t | s_{1:t-1}, \boldsymbol{y}_{1:t}) \tag{4.13}$$

where the posterior distribution $p(s_t | s_{1:t-1}, \boldsymbol{y}_{1:t})$ can be approximated by a typical particle filter, and the posterior distribution $p(\boldsymbol{z}_t | \boldsymbol{z}_{1:t-1}, s_{1:t}, \boldsymbol{y}_{1:t})$ can be solved analytically by a Kalman filter [63].

Assuming that we can approximate the posterior distribution over s_t by a weighted set of particles $\{s_t^{(i)}, w_t^{(i)}\}_{i=1}^N$, i.e.,

$$p(s_t | s_{1:t-1}, \boldsymbol{y}_{1:t}) \approx \sum_{i=1}^{N} w_t^{(i)}\, \delta_{s_t^{(i)}}(s_t) \tag{4.14}$$

Then, the posterior distribution over $\boldsymbol{z}_t$ given s_t can be approximated by a Gaussian mixture

$$p(\boldsymbol{z}_t | \boldsymbol{z}_{1:t-1}, s_{1:t}, \boldsymbol{y}_{1:t}) \approx \sum_{i=1}^{N} w_t^{(i)}\; p(\boldsymbol{z}_t | \boldsymbol{z}_{1:t-1}, s_{1:t}^{(i)}, \boldsymbol{y}_{1:t}) \tag{4.15}$$

Since both $p(\boldsymbol{z}_t | \boldsymbol{y}_{t-1}, \boldsymbol{z}_{t-1}, s_{t-1}, s_t)$ and $p(\boldsymbol{y}_t | \boldsymbol{z}_t, s_t)$ are Gaussian distributions in the SPPCA model, $p(\boldsymbol{z}_t | \boldsymbol{z}_{1:t-1}, s_{1:t}^{(i)}, \boldsymbol{y}_{1:t})$ can be computed efficiently by using the Kalman filter [63].

The procedures of the RBPF can be summarized as follows. Firstly, we sample the current particles $\{s_t^{(i)}\}_{i=1}^N$ from the transition prior $p(s_t^{(i)} | s_{t-1}^{(i)})$, i.e.,

$$s_t^{(i)} \sim p(s_t^{(i)} | s_{t-1}^{(i)}) \quad \text{for } i = 1 \ldots N \tag{4.16}$$

Secondly, the weights of particles can be updated according to

$$w_t^{(i)} = p(y_t | s_t^{(i)}) \quad \text{for } i = 1 \ldots N \tag{4.17}$$

where $p(y_t|s_t^{(i)})$ can be computed analytically by the Kalman filter [63]. Thirdly, we normalize the weights to make them sum to one

$$\tilde{w}_t^{(i)} = \frac{w_t^{(i)}}{\sum_{i=1}^{N} w_t^{(i)}} \quad \text{for } i = 1 \dots N \tag{4.18}$$

where $\tilde{w}_t^{(i)}$ is the weight of the i-th particle after normalization. Then, we re-sample the particles according to the normalized importance weight $\tilde{w}_t^{(i)}$ to generate a set of particles of uniform weights. Finally, we use Kalman recursion [63] to approximate the posterior distribution in Eq. (4.15). Algorithm 3 summarizes the procedures of the Rao-Blackwellized Particle Filter in the SPPCA model.

Algorithm 3 Rao-Blackwellised Particle Filter for SPPCA

```
Input: $\boldsymbol{y}_{1:T}$, $\boldsymbol{\Omega}$
Output: $s_{1:T}$, $\bar{z}_{1:T}$, $\bar{\Sigma}_{1:T}$
1: for $t = 1$ to $T$ do
2:   for all particle $i$ do
3:     if $t = 1$ then
4:       sample $s_t^i$ from $\boldsymbol{v}_s$
5:       $[\boldsymbol{z}_t^i, \boldsymbol{\Sigma}_t^i, L_t^i] = KalmanInitialize(\boldsymbol{y}_t, \boldsymbol{z}_0, \boldsymbol{\Sigma}_0, \boldsymbol{\Theta}_{s_t^i})$
6:     else
7:       sample $s_t^i$ from $\boldsymbol{\Phi}(s_{t-1}^i, :)$
8:       if $s_t^i = s_{t-1}^i$ then
9:         $[\boldsymbol{z}_t^i, \boldsymbol{\Sigma}_t^i, L_t^i] = KalmanUpdate(\boldsymbol{y}_t, \boldsymbol{z}_{t-1}^i, \boldsymbol{\Sigma}_{t-1}^i, \boldsymbol{\Theta}_{s_t^i})$
10:      else
11:        $\tilde{\boldsymbol{z}}_t = \boldsymbol{\Gamma}(s_t^i)(\boldsymbol{y}_t - \boldsymbol{\mu}(s_t^{(i)})$
12:        $[\boldsymbol{z}_t^i, \boldsymbol{\Sigma}_t^i, L_t^i] = KalmanInitialize(\boldsymbol{y}_t, \tilde{\boldsymbol{z}}_t, \boldsymbol{I}, \boldsymbol{\Theta}_{s_t^i})$
13:      end if
14:    end if
15:    $w_t^i = L_t^i$
16:  end for
17:
18:  $\{w_t^i\}_{i=1}^N = normalise(\{w_t^i\}_{i=1}^N)$
19:  $\{s_t^i, \boldsymbol{z}_t^i, w_t^i\}_{i=1}^N = Resample(\{s_t^i, \boldsymbol{z}_t^i, w_t^i\}_{i=1}^N)$
20:
21:  $s_t = histogram(\{s_t^i\}_{i=1}^N)$
22:  $\bar{\boldsymbol{z}}_t = mean(\{\boldsymbol{z}_t^i\}_{i=1}^N)$
23:  $\bar{\boldsymbol{\Sigma}}_t = covariance(\{\boldsymbol{\Sigma}_t^i\}_{i=1}^N)$
24: end for
```

4.5 Prediction

The *prediction* operation provides a new template to the tracker in the next frame. This can be done by first sampling a new set of particles $\{\tilde{s}_{t+1}^{(i)}\}_{i=1}^{N}$ from the transition prior, i.e., $\tilde{s}_{t+1}^{(i)} \sim p(\tilde{s}_{t+1}^{(i)}|s_t^{(i)})$ for $i = 1 \dots N$, and then evaluate the following probability

$$p(\boldsymbol{y}_{t+1}|s_{1:t}^{(i)}, \boldsymbol{z}_{1:t}, \boldsymbol{y}_{1:t}, \tilde{s}_{t+1}^{(i)}) \quad \text{for } i = 1 \dots N \tag{4.19}$$

Note that Eq. (4.19) can be computed efficiently by Kalman prediction [63]. Let $\bar{\boldsymbol{y}}_{t+1}^{(i)}$ be the mean of Eq. (4.19), the new template $\tilde{\boldsymbol{y}}_{t+1}$ can be computed by averaging out all $\bar{\boldsymbol{y}}_{t+1}^{(i)}$

$$\tilde{\boldsymbol{y}}_{t+1} = \frac{1}{N}\sum_{i=1}^{N} \bar{\boldsymbol{y}}_{t+1}^{(i)} \tag{4.20}$$

The new template $\tilde{\boldsymbol{y}}_{t+1}$ will be fed to the tracker for searching for the location in the image that is most similar to the template at time $t+1$.

4.6 Experiments

In this experiment, we evaluate the prediction accuracy of the SPPCA template updater. We have a collection of 5609 training images as shown in Figure 4.4. All images are square and contain a single hockey player. We divided the collection into two sets: the training set and the testing set. The training set contains 4803 images and the testing set contains 806 images. We deliberately made both the training and testing sets have the images of players from several different teams, and players that perform all kinds of actions. We transform all image patch to the HOG descriptors constructed by sampling the SIFT descriptors of size 16×16 from the image patch with a 16-pixel spacing, and every SIFT descriptor is computed with $n_w = n_h = 2$, $n_b = 8$ (yielding the HOG descriptors with $n_y = 128$). Then, we trained the SPPCA with different n_z and n_s using the training set. In the testing phase, we utilize the *prediction* operation of the SPPCA template updater to generate a new template

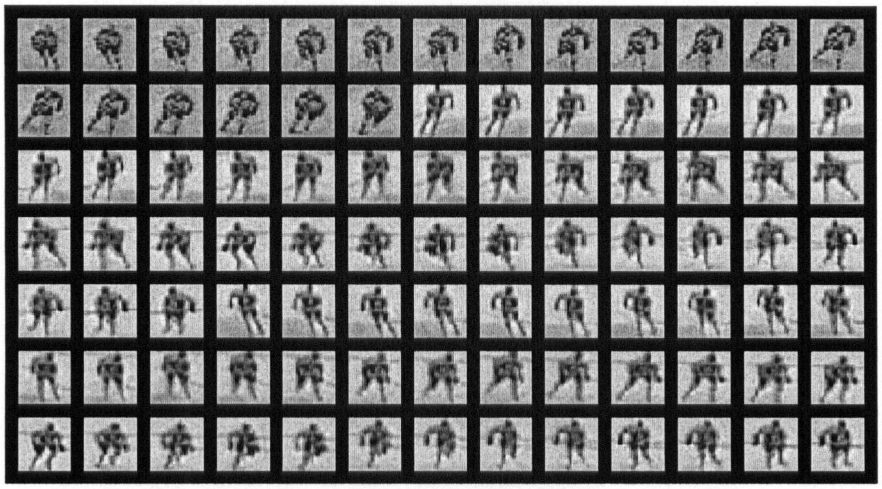


Figure 4.4: **Training set of images for hockey players:** This figure shows a part of our hand annotated training data. A total of 5609 different figures of hockey players are used for the training.

$\tilde{\boldsymbol{y}}_{t+1}$, and compare the prediction with the ground truth $\boldsymbol{y}_{t+1}$ using the relative sum-of-squares error

$$\epsilon_{t+1} = \frac{\sum_{i=1}^{n_y} (\tilde{y}_{t+1,i} - y_{t+1,i})^2}{\sum_{i=1}^{n_y} y_{t+1,i}^2} \tag{4.21}$$

where $\tilde{y}_{t+1,i}$ and $y_{t+1,i}$ is the i-th feature of vector $\tilde{\boldsymbol{y}}_{t+1}$ and $\boldsymbol{y}_{t+1}$, respectively. Then, we feed the new observation $\boldsymbol{y}_{t+1}$ back to the SPPCA template updater and call the *updating* operation to update the joint posterior distribution over $\{s_{t+1}, \boldsymbol{z}_{t+1}\}$ as described in Section 4.5. Experimental results are shown in Table 4.5, and the error is the average prediction error over all testing sequences, i.e., $E = (1/T)\sum_{t=1}^{T} \epsilon_t$ where T is the length of the testing sequence.

For comparison, we also show the predictive power of a HMM template updater in Table 4.5. The HMM template updater we use is similar to the exemplar tracker [32, 33, 57] which use a fixed number of learned templates. The hidden state of the HMM has n_s possible values, indicating that there are n_s possible templates we can use. We also learn the transition and initial distribution of the hidden state, and the parameters of the observation distribution (single Gaussian in our case).

The prediction of $\boldsymbol{y}_{t+1}$ is computed by the standard HMM prediction, which will be a mixture of Gaussians. After observing the new data, we perform one step of the standard forward algorithm [48] to update the hidden state.

From the experimental results, several observations can be made. Firstly, the predictive error of the SPPCAs are smaller than the HMMs. This is because we allow the templates to "move" on the subspace rather than using a fixed number of templates. Thus, the templates of the SPPCAs have much more variations than the HMMs. Secondly, we have better results with the increase of number of subspaces n_s. This confirms our hypothesis that the data lies on a nonlinear subspace, and thus the more local linear subspaces we have, the more we can approximate the nonlinear space. Thirdly, the predictive power improves with the increase of the number of principal components n_z due to the nature of the PPCA. An interesting observation is that when n_z is large, we do not have much accuracy gain.

We have also tried to learn the system matrix $\boldsymbol{A}$ from the training data instead of setting $\boldsymbol{A} = \boldsymbol{I}$. However, the preliminary experimental results show no improvement to the accuracy of prediction.

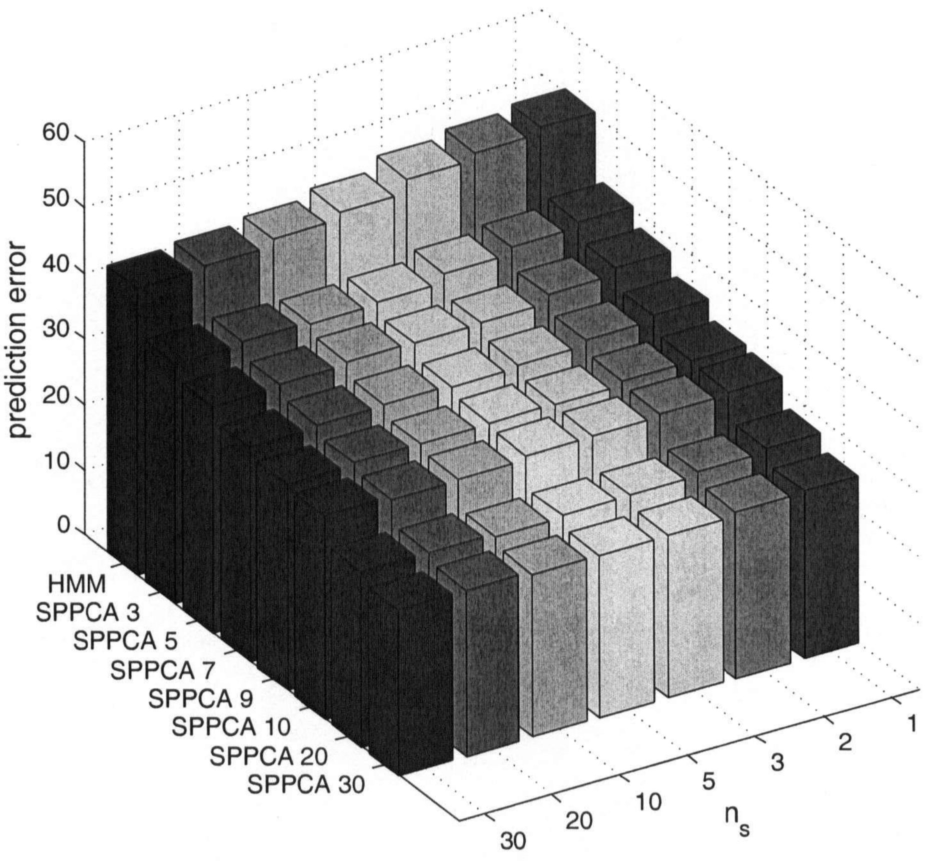


Figure 4.5: **Experimental results of the SPPCA template updater:** This figure shows the prediction error of the SPPCA template updater and the HMM template updater with different n_s and n_z. The error is measured by averaging out the criterion in Eq. (4.21), i.e., $E = (1/T)\sum_{t=1}^{T} \epsilon_t$ where T is the length of the testing sequence. The HMM template updater is abbreviated to **HMM**, and the SPPCA template updater with $n_s = k$ is abbreviated to **SPPCA k** (e.g., SPPCA template updater with $n_s = 10$ is abbreviated to **SPPCA 10**). Note that in the HMM template updater, n_s denotes the number of templates, while in the SPPCA template updater, n_s denotes the number of subspaces.

Chapter 5

Action Recognition

5.1 Introduction

The goal of the action recognizer is to classify the actions or activities of the hockey players *online* after tracking the positions of the players on the image coordinate system. The understanding of human activities is usually solved by analyzing and classifying the trajectories of people. In the hockey domain, however, the problem is more challenging because we do not know the trajectories of the players on the rink coordinate system due to the facts that the camera is not stationary, and we do not have an accurate homography between the image and the rink coordinate system. As a result, the only information we have about the hockey players is small image patches of 30 to 70 pixels height that contain the entire body of the players.

The action recognizer presented in this section is capable of classifying video clips into known categories representing the actions/activities of the hockey players. The action recognizer first transforms image patches to the HOG descriptors [8] described in Section 3.2. We use the HOG descriptors as the input feature because it summarizes the shape of the players in a discriminative and economic way, and thus it improves both the accuracy and speed. After constructing the HOG descriptors of the image patches, the action recognizer computes the frame-to-frame similarity matrix between the testing and training data. In order to aggregate temporal in-

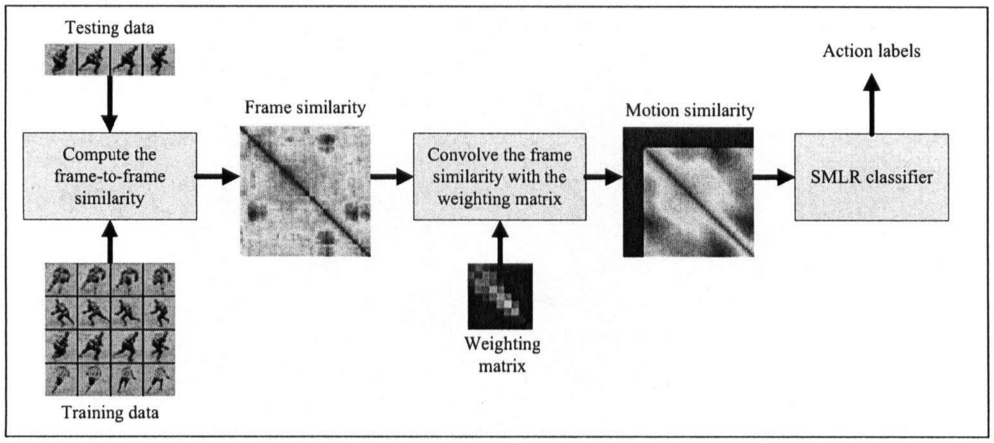


Figure 5.1: **The Action Recognizer:** This figure illustrates the procedures of the action recognizer. The action recognizer first computes the frame-to-frame similarity between the training and testing data. Then, the frame-to-frame similarity matrix is convolved with a weighting matrix to produce the motion-to-motion similarity matrix. Finally, the SMLR classifier takes the frame similarity matrix as input feature to determine the player's actions.

formation, the frame-to-frame similarity matrix is then convolved with a weighting function similar to [11] to produce the motion-to-motion similarity matrix. Finally, a Sparse Multinomial Logistic Regression (SMLR) classifier [27] is exploited and it takes the motion similarity matrix as input to determine the player's actions. Figure 5.1 illustrates the procedures of the action recognizer.

This chapter is organized as follows: Section 5.2 introduces the Sparse Multinomial Logistic Regression (SMLR) classifier. Section 5.3 describes the robust motion-to-motion similarity measure. Experimental results are presented in Section 5.4.

5.2 The Sparse Multinomial Logistic Regression Classifier

In this thesis, we use a recently developed sparse classifier, Sparse Multinomial Logistic Regression (SMLR) [27], to classify the hockey players' actions. The SMLR classifier learns weighted sums of basis functions with sparsity-promoting priors encouraging the weight estimates to be significantly large or exactly zero. The SMLR classifier has been shown to have comparable or better classification accuracy than the Support Vector Machine (SVM) and Relevance Vector Machine (RVM) [27], and has been used in the field of bio-informatics [5, 53].

We choose to use the SMLR classifier for the following reasons: (1) The SMLR classifier utilizes a Laplacian prior to promote sparsity and leads to a smaller number of basis functions. The outcome includes an increase of the classification speed and a better generalization error because it avoids over-fitting the training data. (2) Unlike SVM, SMLR has no restriction on the choices of basis functions [27]. Any kernel function can be used to transform the original feature space to a higher dimensional feature space. (3) We can always find the global optimal weights for the basis functions. This is because the Laplacian prior is a log-concave function, and when combined with a concave log-likelihood, it leads to a concave log-posterior with a unique maximum.

Let $\boldsymbol{y} = [y_1 \dots y_d]^T \in \mathbb{R}^d$ be the d observed features, and $\boldsymbol{a} = [a_1 \dots a_m]^T$ represent the class labels using a "1-of-m" encoding vector such that $a_i = 1$ if $\boldsymbol{y}$ corresponds to an example belonging to class i and $a_i = 0$ otherwise. Under a multinomial logistic regression model, the probability that $\boldsymbol{y}$ belongs to class i is written as

$$p(a_i = 1 | \boldsymbol{y}, \boldsymbol{w}) = \frac{\exp\left(\boldsymbol{w}_i^T g(\boldsymbol{y})\right)}{\sum_{j=1}^{m} \exp\left(\boldsymbol{w}_j^T g(\boldsymbol{y})\right)} \tag{5.1}$$

for $i \in \{1, \dots, m\}$, where $\boldsymbol{w}_i$ is the weight vector corresponding to class i. Note that $g(\boldsymbol{y})$ can be either the original input feature, any linear/nonlinear transformation that maps the original data to a higher dimensional space, or a kernel centered at the training samples.

There are many ways to estimate the optimal $\boldsymbol{w}$ given the training examples. In the SMLR framework, we adopt maximum a posteriori (MAP) estimation because we want to encourage the weight estimates to be significantly large or exactly zero.

Specifically, we optimize the following objective function given n labeled training points $\{\boldsymbol{y}_j, \boldsymbol{a}_j\}_{j=1}^n$

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{arg\,max} \sum_{j=1}^{n} \log p(\boldsymbol{w}|\boldsymbol{y}_j, \boldsymbol{a}_j) = \underset{\boldsymbol{w}}{arg\,max} \sum_{j=1}^{n} \log p(\boldsymbol{a}_j|\boldsymbol{y}_j, \boldsymbol{w}) p(\boldsymbol{w}) \tag{5.2}$$

The prior $p(\boldsymbol{w})$ is defined as a sparsity-promoted Laplacian prior,

$$p(\boldsymbol{w}) \propto \exp\left(-\lambda \|\boldsymbol{w}\|_1\right) \tag{5.3}$$

where $\|\boldsymbol{w}\|_1$ denotes the L_1 norm, and λ is a user-specified parameter that controls the sparsity of the classifier. Observe that the objective function Eq. (5.2) is still a concave function and therefore all local optimal estimates $\hat{\boldsymbol{w}}$ are also a global optimum.

There are many algorithms that optimizes the objective function Eq. (5.2), e.g., [5, 27, 53]. We adopt the component-wise updating procedure introduced by Krishnapuram *et al.* [27] because it converges faster than others in practice.

5.3 Motion Similarity Measure

A useful property of the SMLR classifier is that there is no restriction on the choice of basis function $g(\boldsymbol{y})$ [27]. Since human actions have various styles and speeds, we choose to use a kernel function centered at the training samples as the basis functions.

The kernel function we use is similar to the motion similarity measure presented by Efros *et al.* [11]. The difference is that our motion similarity measure is a weighted sum of past frame-to-frame similarities, while Efros *et al.* [11] summed up both past and future frame-to-frame similarities. Let $\boldsymbol{y}_t$ be the input feature at time t, and $\boldsymbol{y}_{1:t}$ represent the input feature from time 1 to time t, a naïve way to measure the motion-to-motion similarity between the testing video clip $\boldsymbol{y}_{1:i}$ and the training sample $\boldsymbol{y}^\star_{1:j}$ is to measure the frame-to-frame similarity between the last frames $\boldsymbol{y}_i$ and $\boldsymbol{y}_j^\star$, i.e.,

$$d_{\text{naïve}}\left(\boldsymbol{y}_{1:i}, \boldsymbol{y}_{1:j}^\star\right) = k(\boldsymbol{y}_i, \boldsymbol{y}_j^\star) \tag{5.4}$$

where $k(\cdot,\cdot)$ is a distance measure. Note that the similarity measure between two motions can be represented by a similarity matrix $\boldsymbol{M}$, where $\boldsymbol{M}_{i,j} = d_{\text{naïve}}\left(\boldsymbol{y}_{1:i}, \boldsymbol{y}_{1:j}^\star\right)$. Figure 5.2 (a) shows the frame-to-frame similarity matrix between two motions using the naïve approach.

However, it is not realistic to measure the similarity between two motions only by the similarity between the last frames. A more robust approach is to convolve the frame-to-frame similarity matrix with a $T \times T$ weighting matrix $\boldsymbol{K}_T$

$$d(\boldsymbol{y}_{1:i}, \boldsymbol{y}_{1:j}^\star) = \sum_{s=1}^{T}\sum_{t=1}^{T} \boldsymbol{K}_T(s,t) k(\boldsymbol{y}_{i-T+s}, \boldsymbol{y}_{j-T+t}^\star) \tag{5.5}$$

A simple choice is to use a $T \times T$ identical matrix $\boldsymbol{I}_T$ as the weighting matrix, i.e., $\boldsymbol{K}_T = \boldsymbol{I}_T$. In other words, we sum up the frame-to-frame similarities of the previous T frames to obtain the motion-to-motion similarity.

The problem of using an identical matrix as the weighting matrix is that it cannot handle actions of different speeds. Thus, we replace the identical matrix $\boldsymbol{I}_T$ in Eq. (5.5) by a $T \times T$ weighting kernel $\boldsymbol{K}_T$ defined as follows:

$$\boldsymbol{K}_T(i,j) = \sum_{r \in R} \omega(r)\ \kappa(i-T, r(j-T)) \tag{5.6}$$

for $i = 1 \ldots T$, $j = 1 \ldots T$, $R \in [1/r_{max}, r_{max}]$ where r_{max} is a user-specified constant representing the maximum speed. The function $\kappa(i-T, r(j-T))$ is defined as

$$\kappa(i-T, r(j-T)) = \begin{cases} 1 & \text{if } i - T = round(r(j-T)) \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

The weight $\omega(r)$ is defined as a function of r,

$$\omega(r) = \begin{cases} r^2 & \text{if r} \leq 1 \\ 1/r^2 & \text{otherwise} \end{cases} \tag{5.8}$$

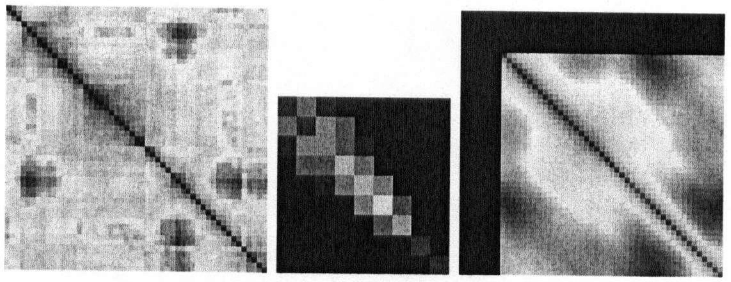


Figure 5.2: **Similarity Matrix:** (a) The frame-to-frame similarity matrix. (b) A weighting matrix with $T = 9$ and $r_{max} = 1.5$. (c) The motion-to-motion similarity matrix computed by convolving (a) with (b).

The weighting matrix $\boldsymbol{K}_T$ is visualized in Figure 5.2 (b). Observe that it gives bigger weight to the diagonal entries, and the entries that are farther from the lower-right corner are more diffused.

Figure 5.2 (c) shows the motion-to-motion similarity matrix computed from convolving the frame-to-frame similarities (Eq. (5.4)) by the new weighting matrix. Observe that the original frame-to-frame similarities in Figure 5.2 (a) are more noisy, while the motion-to-motion similarities in Figure 5.2 (c) are much smoother.

The motion similarity measure $d(\boldsymbol{y}_{1:i}, \boldsymbol{y}^\star_{1:j})$ can be easily incorporated into the SMLR classifier. Since the SMLR classifier has no restriction on the choice of basis functions, we can use the motion similarity measure as a kernel centered on the training samples and replace the basis functions $g(\boldsymbol{y}_t)$ by

$$g(\boldsymbol{y}_t) = [d(\boldsymbol{y}_{t-T+1:t}, \boldsymbol{y}^\star_{1:T}), d(\boldsymbol{y}_{t-T+1:t}, \boldsymbol{y}^\star_{2:T+1}), \ldots, d(\boldsymbol{y}_{t-T+1:t}, \boldsymbol{y}^\star_{n-T+1:n})]^T \quad (5.9)$$

where $\boldsymbol{y}^\star$ are the training samples, T is the width of the weighting matrix, and n is the length of the training video sequence.

5.4 Experiments

This experiment compares the performance between the proposed action recognizer that uses the SMLR classifier with the HOG descriptors (SMLR+HOG), and

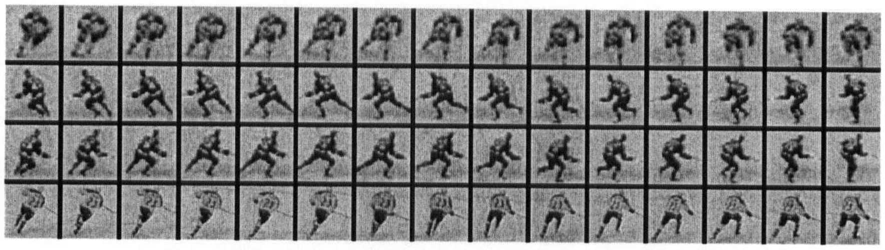


Figure 5.3: **Training data for the action recognizer:** This figure shows a part of our hand annotated training data. A total of 4295 different figures of hockey players are used as training images for the action recognizer. FIRST ROW: players skating down. SECOND ROW: players skating left. THIRD ROW: players skating right. FOURTH ROW: players skating up.

the action recognizers presented by Efros *et al.* [11] (5-NN+DOF) and Wu [64] (5-NN+DIG). We also present the results of combining the SMLR classifier with the DIG descriptor (SMLR+DIG) and the DOF descriptor (SMLR+DOF).

5.4.1 Dataset

We first manually collected a dataset consisting of 5609 32×32 gray images in which the hockey players are aligned and centered. Among these 5609 images, 4295 of them are training images and the remaining 1314 are testing images. Figure 5.3 shows some examples of the training dataset. In order to increase the diversity of the dataset, we put players with different uniforms and images with different lighting conditions into the dataset. Finally, we manually divided all images into four categories according to the direction of the hockey players' movement: skating down, up, left, and right. Note that it is also possible to partition the training images into more abstract categories, e.g., skating and shooting. Unfortunately, due to a lack of training data, we focus on classifying the moving directions of the players in this thesis.

5.4.2 Parameter Settings

In our experiments, we combine three image descriptors (HOG, DOF, and DIG) with two classifiers (SMLR and nearest neighbor). The following sections details the parameter settings of the three image descriptors and the two classifiers.

The HOG descriptor

The HOG descriptor is constructed by sampling the SIFT descriptors of size 16×16 from the image patch with a 16-pixel spacing, and every SIFT descriptor is computed with $n_w = n_h = 2$, $n_b = 8$. These parameters result in a HOG descriptor of dimensionality 128.

In the original SIFT descriptor implementation [31], Lowe smoothed the image patch with a Gaussian low-pass filter before computing the image gradients. However, Dalal *et al.* [8] suggested that the classification accuracy is better when no smoothing is used. From preliminary experiments, we also confirmed that smoothing the images before computing the image gradients decreases the classification accuracy. As a result, we decided not to apply any smoothing before computing image gradients in this thesis.

Dalal *et al.* [8] also suggested that using overlapping SIFT descriptors will improve the classification accuracy. However, in our preliminary experiments we found out that using the overlapping SIFT descriptors not only slightly decreases the classification accuracy but also increases the dimensionality of the HOG descriptor. Therefore, we decided not to overlap the SIFT descriptor in this thesis for better classification accuracy and speed.

We use the χ^2 distance to compute the frame-to-frame similarities between a pair of HOG descriptors. The χ^2 distance is defined as

$$k(\boldsymbol{y}_i, \boldsymbol{y}_j) = \chi^2(\boldsymbol{y}_i, \boldsymbol{y}_j) = \sum_{k=1}^{D} \frac{(y_{i,k} - y_{j,k})^2}{y_{i,k} + y_{j,k}}. \tag{5.10}$$

where $\boldsymbol{y}_{i,k}$ represents the k-th feature of the vector $\boldsymbol{y}_i$, and D is the size of $\boldsymbol{y}$

($D = 128$ in our case). In order to aggregate information across multiple frames, we use a 5×5 temporal kernel with $r_{max} = 1.5$ to compute the motion-to-motion similarity measure described in Section 5.3. Finally, the motion-to-motion similarity vector is used as a feature and fed into a 5-nearest-neighbor classifier (5-NN+HOG) and a SMLR classifier with $\lambda = 0.1$ to determine the player's action (SMLR+HOG). Notice that the 5-NN+HOG classifier is similar to the one introduced by Freeman *et al.* [13].

The DOF descriptor

The Decomposed Optical Flow (DOF) descriptor [11] is constructed by using the flow images computed by the Lucas-Kanade algorithm [34] and smoothed by a 3×3 Gaussian low-pass filter. Then, the flow images are decomposed into four channels $(F_X^+, F_X^-, F_Y^+, F_Y^-)$, where F_X^+, F_X^-, F_Y^+, and F_Y^- represent the optical flow along the X^+, X^-, Y^+, and Y^- directions, respectively (See Figure 2.3 for an example of the DOF descriptor). Since we use 32×32 gray images, the DOF descriptors have dimensionality $32 \times 32 \times 4 = 4096$.

We use the scalar product to compute the frame-to-frame similarities between a pair of DOF descriptors. Specifically, the frame-to-frame similarity measure is defined as $k(\boldsymbol{y}_i, \boldsymbol{y}_j) = \boldsymbol{y}_i^T \boldsymbol{y}_j$. In order to aggregate information across multiple frames, we use a 5×5 temporal kernel with $r_{max} = 1.5$ to compute the motion-to-motion similarities described in Section 5.3. Finally, the motion-to-motion similarity vector is used as a feature and fed into a 5-nearest-neighbors classifier (5-NN+DOF) or a SMLR classifier with $\lambda = 0.1$ (SMLR + DOF) to determine the player's action.

The DIG descriptor

The Decomposed Image Gradients (DIG) descriptor [64] is constructed by using the image gradients computed by using a $[-1, 0, 1]$ kernel and smoothed by a 3×3 Gaussian low-pass filter. Similar to DOF, the image gradients are decomposed into four channels $(G_X^+, G_X^-, G_Y^+, G_Y^-)$, where G_X^+, G_X^-, G_Y^+, and G_Y^- represent the image gradients along the X^+, X^-, Y^+, and Y^- directions, respectively (See Figure 2.4 for an example of the DIG descriptor). Since we use 32×32 gray images, the DIG descriptors have dimensionality $32 \times 32 \times 4 = 4096$.

We use the scalar product to compute the frame-to-frame similarities between a pair of DIG descriptors. Specifically, the frame-to-frame similarity measure is defined as $k(\boldsymbol{y}_i, \boldsymbol{y}_j) = \boldsymbol{y}_i^T \boldsymbol{y}_j$. In order to aggregate information across multiple frames, we use a 5×5 temporal kernel with $r_{max} = 1.5$ to compute the motion-to-motion similarities described in Section 5.3. Finally, the motion-to-motion similarity vector is used as a feature and fed into a 5-nearest-neighbors classifier (5-NN+DIG) or a SMLR classifier with $\lambda = 0.1$ (SMLR+DIG) to determine the player's action.

5.4.3 Results

Table 5.1 shows the accuracy and speed of the five action recognizers: 5-NN+DOF, 5-NN+DIG, SMLR+DOF, SMLR+DIG, and SMLR+HOG. The accuracy is measured by the percentage of actions that are correctly classified. The speed is measured by the average time of computing the descriptor for a single image patch (the $T1$ time), and the average time of classifying a single image patch given the descriptor (the $T2$ time). The average total time of classifying an image patch (the $T1+T2$ time) is also shown in the table.

Among all action recognizers, the SMLR+HOG classifier has the best classification accuracy, followed by SMLR+DOF and 5-NN+DIG. The classification accuracy of 5-NN+DOF, 5-NN+HOG, and SMLR+DIG is considerably worse than SMLR+HOG. An interesting observation is that the SMLR classifier has better accuracy than the 5-NN classifier when we use the DOF and HOG descriptors. In the case of the DIG descriptor, however, the accuracy of the SMLR classifier is significantly poorer than the 5-NN classifier.

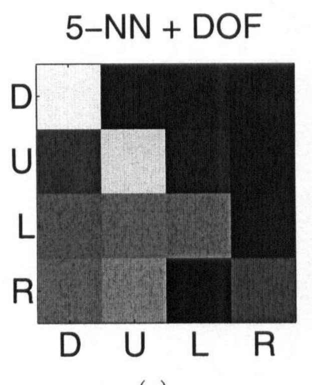
Another advantage of SMLR+HOG is its speed. From the $T1+T2$ column in

Method	*Accuracy*	$T1$	$T2$	$T1 + T2$
5-NN + DOF	62.90%	$0.830s$	$3.199s$	$4.029s$
5-NN + DIG	70.97%	**0.017s**	$3.246s$	$3.263s$
5-NN + HOG	52.42%	$0.023s$	$0.823s$	$0.846s$
SMLR + DOF	73.21%	$0.830s$	$2.758s$	$3.588s$
SMLR + DIG	59.65%	**0.017s**	$2.731s$	$2.748s$
SMLR + HOG	**76.37%**	$0.023s$	**0.183s**	**0.206s**

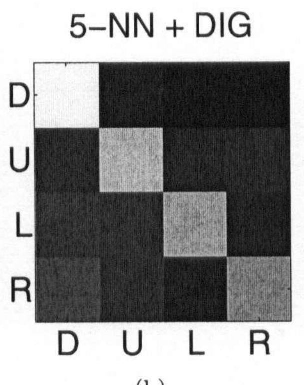
Table 5.1: **Action Recognition Results:** *Accuracy* measures the percentage of the actions that are correctly classified. $T1$ measures the average time of computing the descriptor for a image patch of size 32×32 pixels. $T2$ measures the average time of classifying a single image patch given the descriptor. $T1 + T2$ measures the average total time of classifying a image patch.

Table 5.1, we can observe that SMLR+HOG is at least 10 times faster than others. The $T1$ column in Table 5.1 shows the average time of computing the descriptor for a image patch of size 32×32 pixels. The HOG descriptors can be very efficiently constructed and the computational time for the HOG descriptor is just slightly longer than the DIG descriptor, but 40 times faster than the DOF descriptor. The $T2$ column in Table 5.1 measures the average time of classifying a single image patch given the descriptor. Since the dimensionality of the HOG descriptors (128D) is much smaller than those of the DIG and DOF descriptors (4096D), SMLR+HOG spends much less time than others computing the motion-to-motion similarities between the testing and training data. This results in a significant shorter classification time.

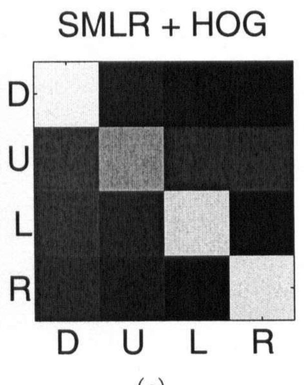
Figure 5.4 shows the confusion matrix of the three action recognizers. The diagonal of the confusion matrix represents the fraction of the actions that are correctly classified. The SMLR+HOG classifies most of the actions correctly except that it usually confuses skating up and down. The main diagonal of the SMLR+HOG classifier is $[0.94, 0.54, 0.73, 0.79]$. In contrast, the 5-NN+DIG classifier makes more mistakes in classifying skating left and right; however, it performs better in classifying skating up. The main diagonal of the 5-NN+DIG classifier is $[0.92, 0.64, 0.58, 0.51]$. The 5-NN+DOF classifier has the worst overall classification



(a)



(b)



(c)

Figure 5.4: **Confusion Matrix for the Action Recognition Results:** (a) 5-NN+DOF: Action recognition results of using the 5-nearest-neighbor classifier with the DOF descriptors. The main diagonal is: $[0.91, 0.73, 0.33, 0.19]$. (b) 5-NN+DIG: Action recognition results of using the 5-nearest-neighbor classifier with the DIG descriptors. The main diagonal is $[0.92, 0.64, 0.58, 0.51]$. (c) SMLR+HOG: Action recognition results of using the SMLR classifier on the HOG descriptor. The main diagonal is $[0.94, 0.54, 0.73, 0.79]$.

accuracy. It is very uncertain about skating left and down, and often mis-classifies these two actions into skating up and down. The main diagonal of the 5-NN+DOF classifier is $[0.91, 0.73, 0.33, 0.19]$.

Chapter 6

Multi-target Tracking

6.1 Introduction

We incorporate the template updater described in Chapter 4 and the action recognizer described in Chapter 5 with a multi-target tracking system. In this thesis, we choose the Boosted Particle Filter (BPF) introduced by Okuma *et al.* [44] because it is fully automatic and very efficient.

We augment the BPF with several extensions. Firstly, the original implementation of the BPF [44] only utilized the HSV color histogram described in Section 3.1 as its observation model. In this thesis, we use both the HSV color histogram and the HOG descriptor described in Section 3.2. The combination the color and shape cues improves the robustness of the tracker. Secondly, we employ the diffusion distance [30] instead of the Bhattacharyya coefficient to compare the difference between histograms. It has been shown by Ling *et al.* [30] that the diffusion distance is more robust to deformation and quantization effects than the Bhattacharyya coefficient that is used in [44]. Thirdly, we apply a mode-seeking algorithm similar to the mean shift [6] to generate a naïve proposal when there is no detection. This further improves the performance of the BPF regardless of occasional sparse Adaboost detections. Fourthly, we use the SPPCA template updater described in Chapter 4 to update the shape templates of the tracker, while [44] did not update their observational model and use only the initial observation as their templates. Lastly, we recognize the target's action by the action recognizer described in Chapter 5. The entire system can simultaneously track and recognize multiple targets' actions smoothly in near real-time. Figure 1.3 shows the system diagram of the tracking and action recognition system.

6.2 Statistical Model

In non-Gaussian state-space models, the state sequence $\{\boldsymbol{x}_t; t \in \mathbb{N}\}, \boldsymbol{x}_t \in \mathbb{R}^{n_{\boldsymbol{x}}}$, is assumed to be an unobserved (hidden) Markov process with initial distribution $p(\boldsymbol{x}_0)$ and transition distribution $p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$, where $n_{\boldsymbol{x}}$ is the dimension of the state vector. In our case, $\boldsymbol{x} = \{l_x, l_y, l_s\}$ where $\{l_x, l_y\}$ represents the location of the player, and l_s represents the size of the player in the image coordinate system. The observations $\{\boldsymbol{y}_t; t \in \mathbb{N}\}, \boldsymbol{y}_t \in \mathbb{R}^{n_{\boldsymbol{y}}}$, are conditionally independent given the process $\{\boldsymbol{x}_t; t \in \mathbb{N}\}$ with marginal distribution $p(\boldsymbol{y}_t|\boldsymbol{x}_t)$, where $n_{\boldsymbol{y}}$ is the dimension of the observation vector.

Letting $\boldsymbol{y}_{1:t} \triangleq \{\boldsymbol{y}_1 \dots \boldsymbol{y}_t\}$ be the observation vectors up to time t, our goal is to estimate $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})$, the probability of the current state $\boldsymbol{x}_t$ given $\boldsymbol{y}_{1:t}$, which can be solved by the following Bayesian recursion [10]:

$$\begin{aligned} p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}) &= \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})}{p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1})} \\ &= \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t)\int p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})p(\boldsymbol{x}_{t-1}|\boldsymbol{y}_{1:t-1})d\boldsymbol{x}_{t-1}}{\int p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})\, d\boldsymbol{x}_t} \end{aligned} \tag{6.1}$$

In our tracking system, this transition distribution $p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$ is a combination of a first-order dynamic model and a second-order autoregressive dynamic model (i.e., a constant acceleration) with additive Gaussian noise. The observation likelihood $p(\boldsymbol{y}_t|\boldsymbol{x}_t)$ is defined in the following section.

6.3 Observation Likelihood

As already described in Chapter 3, our observation model consists of color and shape information which is encoded by the HSV color histogram and the HOG descriptor, respectively. We compute the observation likelihood by

$$p(\boldsymbol{y}_t|\boldsymbol{x}_t) \propto p_{hsv}(\boldsymbol{y}_t|\boldsymbol{x}_t)\, p_{hog}(\boldsymbol{y}_t|\boldsymbol{x}_t) \tag{6.2}$$

For the HSV color model, we use a combination of a 2D color histogram based on Hue and Saturation and a 1D color histogram based on Value. The distribution of the color likelihood is given as follows:

$$p_{hsv}(\boldsymbol{y}_t|\boldsymbol{x}_t) \propto e^{-\lambda_c \xi[\boldsymbol{K}^*, \boldsymbol{K}(\boldsymbol{x}_t)]} \tag{6.3}$$

where $\boldsymbol{K}(\boldsymbol{x}_t)$ is the HSV color histogram computed at $\boldsymbol{x}_t$, $\boldsymbol{K}^*$ is the template for the HSV color histogram, and $\xi(\cdot,\cdot)$ is the diffusion distance [30]. We fix the scaling constant $\lambda_c = 10$ throughout our experiments.

We use a 3D histogram based on the magnitude of gradients in both x and y direction and their orientations for the HOG descriptor. Then the following likelihood distribution is given:

$$p_{hog}(\boldsymbol{y}_t|\boldsymbol{x}_t) \propto e^{-\lambda_s \xi[\boldsymbol{H}^*, \boldsymbol{H}(\boldsymbol{x}_t)]} \tag{6.4}$$

where $\boldsymbol{H}(\boldsymbol{x}_t)$ is the HOG descriptor computed at $\boldsymbol{x}_t$, $\boldsymbol{H}^*$ is the template for the HOG descriptor, and $\xi(\cdot,\cdot)$ is the diffusion distance [30]. We fix the scaling constant $\lambda_c = 10$ throughout our experiments.

6.4 Particle Filtering

Since the observation likelihood Eq. (6.3) is nonlinear and non-Gaussian, there is no analytical solution for the Bayesian recursion Eq. (6.1). Instead, we seek an approximation solution, using particle filtering [10].

In standard particle filtering, we approximate the posterior $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})$ with a Dirac measure using a finite set of N particles $\{\boldsymbol{x}_t^{(i)}, w_t^{(i)}\}_{i=1}^N$. To accomplish this, we sample candidate particles from an appropriate proposal distribution

$$\boldsymbol{x}_t^{(i)} \sim q(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{1:t-1}^{(i)}, \boldsymbol{y}_{1:t}) \quad \text{for } i = 1 \dots N \tag{6.5}$$

In the simplest scenario, it is set as $q(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{1:t-1}^{(i)}, \boldsymbol{y}_{1:t}) = p(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{t-1}^{(i)})$, yielding the bootstrap filter [10]. However, a smarter proposal distribution can be employed. The following section will discuss this issue.

The weights associated with these particles according to the following importance ratio:

$$w_t^{(i)} = w_{t-1}^{(i)} \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t^{(i)})\, p(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{t-1}^{(i)})}{q(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{1:t-1}^{(i)}, \boldsymbol{y}_{1:t})} \tag{6.6}$$

We resample the particles using their importance weights to generate an unweighted approximation of $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})$. The particles are used to obtain the following approximation of the posterior distribution:

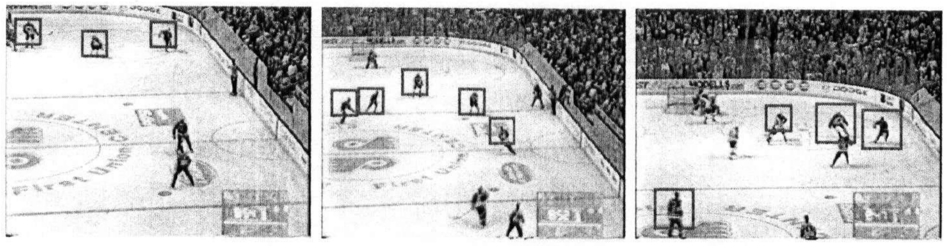
$$p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}) \approx \sum_{i=1}^{N} w_t^{(i)}\, \delta_{\boldsymbol{x}_t^{(i)}}(\boldsymbol{x}_t) \tag{6.7}$$

6.5 Boosted Particle Filter

It is widely accepted that proposal distributions that incorporate the recent observations (in our case, through the Adaboost detections) outperform naïve transition prior proposals considerably [51, 60]. In this thesis, we use the Boosted Particle Filter [44] that incorporates the current detections of hockey players to produce a better proposal distribution $q(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{1:t-1}^{(i)}, \boldsymbol{y}_{1:t})$.

6.5.1 Adaboost Detection

In order to detect hockey player in the current frame, we adopt the cascaded Adaboost algorithm of Viola and Jones [61], which was originally developed for detecting faces. In our experiments, a 23 layer cascaded classifier is trained to detect



(a) (b) (c)

Figure 6.1: **Hockey player detection results:** This figure shows results of the Adaboost hockey detector. (a), (b), and (c) show mostly accurate detections. Please note that there are players who are not detected and some of the boxes do not cover the entire figure of the player (i.e., a box is too small to cover a lower part of the body)

hockey players. In order to train the detector, a total of 5609 figures of hockey players are used. These figures are scaled to have a resolution of 24×24 pixels. We hand annotate figures of hockey players to use for the training as shown in Figure 4.4. Unlike the detector used in [44], our trained Adaboost classifier produces few false positives (i.e., a few false positives in several thousand frames) even alongside the edge of the rink where most false positives appeared in [44]. More human intervention with a larger and better training set leads to better Adaboost detection results, although localization failures would still be expected in regions of clutter and overlap. The non-hockey-player sub-windows used to train the detector are generated from over 300 images manually chosen to contain nothing but the hockey rink and audience. Since our tracker is implemented for tracking hockey scenes, there is no need to include training images from outside the hockey domain. Suffice it to say, exploiting such domain knowledge greatly reduces the false positive rate of our detector.

The results of using the cascaded Adaboost detector in our hockey dataset are shown in Figure 6.1. The cascaded Adaboost detector performs well at detecting the players but often gets confused in a cluttered region with multiple players and ignores some of players.

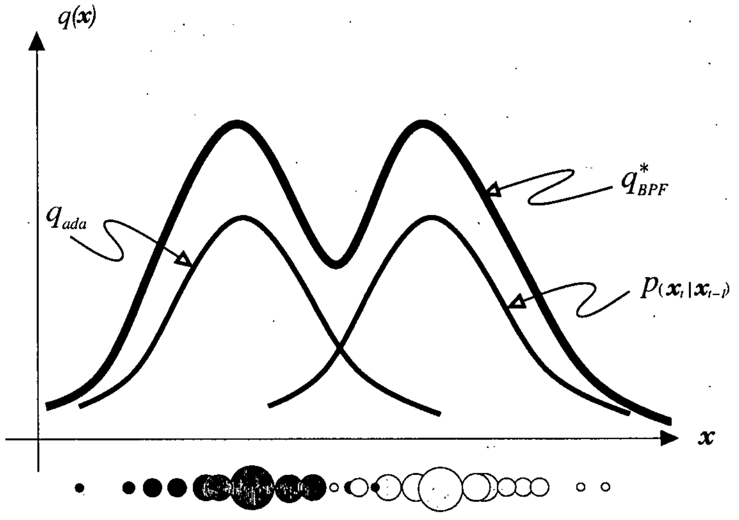


Figure 6.2: **Mixture of Gaussians for the Proposal Distribution.**

6.5.2 Proposal Distribution with the Adaboost Detections

It is clear from the Adaboost detection results that they could be improved if we considered the motion models of the players. In particular, by considering plausible motions, the number of false positives could be reduced. For this reason, the Boosted Particle Filter (BPF) incorporates the Adaboost detection in the proposal mechanism of the particle filters. The expression for the proposal distribution is given by the following mixture.

$$q^*_{BPF}(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{1:t-1}^{(i)}, \boldsymbol{y}_{1:t}) = \alpha_{ada} q_{ada}(\boldsymbol{x}_t^{(i)}|\boldsymbol{y}_t) + (1 - \alpha_{ada}) p(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{t-1}^{(i)}) \qquad (6.8)$$

where q_{ada} is a Gaussian distribution centered in the Adaboost detection with a fixed variance (See Figure 6.2). The parameter α_{ada} can be set dynamically without affecting the convergence of the particle filter (it is only a parameter of the proposal distribution and therefore its influence is corrected in the calculation of the importance weights). When $\alpha_{ada} = 0$, our algorithm reduces to the bootstrap particle filter. By increasing α_{ada} we place more importance on the Adaboost detections. We can adapt the value of α_{ada} depending on tracking situations, including cross overs, collisions and occlusions.

6.5.3 Further Boosting by a Naïve Proposal

Since there is no guarantee that the Adaboost detector detects all targets in the scene, the detection results can be sparse over time. The performance of BPF is, however, much better when there are many detections densely over time. One way to further improve the performance of BPF is to use an additional proposal mechanism other than the Adaboost detector. Thus, we use a mode-seeking algorithm similar to mean shift [6] to find a local maximum of the HSV and HOG observation likelihoods and employ a Gaussian distribution centered in the local maximum as a new proposal distribution. This proposal is not as reliable as the Adaboost detections; however, it is often better than the transition distribution which cannot accurately model the dynamics of the targets due to the moving camera.

6.6 Multi-target Tracking

Multi-target tracking is performed by running multiple independent Boosted Particle Filters for every target in the scene. Algorithm 4 summarizes our fully automatic multi-target tracking algorithm.

Briefly, the targets are detected and initialized by using the cascaded Adaboost detector described in Section 6.5.1. During the tracking at time $t+1$, we use the SPPCA template updater described in Chapter 4 to predict the new template $\tilde{\boldsymbol{y}}_{t+1}$ of the HOG descriptor for each target. The color template is not updated because there is usually no noticeable change in the colors of hockey players. Then, BPF is applied to estimate the posterior distribution over $\boldsymbol{x}_{t+1}$. To update the posterior distribution over $\{s_{t+1}, \boldsymbol{z}_{t+1}\}$, we compute the mean $\bar{\boldsymbol{x}}_{t+1}$ of the posterior distribution $p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{t+1})$, and extract the image patch $\bar{\boldsymbol{y}}_{t+1}$ located in $\bar{\boldsymbol{x}}_{t+1}$. The image patch $\bar{\boldsymbol{y}}_{t+1}$ is then fed into the SPPCA template updater to update the posterior over $\{s_{t+1}, \boldsymbol{z}_{t+1}\}$. Similarly, we give the image patch $\bar{\boldsymbol{y}}_{t+1}$ to the action

Algorithm 4 Boosted Particle Filter

```
Input: ${I_t}_{i=1}^{T}$
Output: ${\boldsymbol{x}_{m,1:T}}_{m=1,\dots,M}$
1: $M = 0$
2: for $t = 1$ to $T$ do
3:   Detect targets by the cascaded Adaboost detector
4:   if there are $M_{new}$ new targets then
5:     for $m = 1$ to $M_{new}$ do
6:       Generate $N$ particles $\{\boldsymbol{x}_{m,t}^{(i)}\}_{i=1}^{N}$ by sampling from a Gaussian distribution
         centered on the Adaboost detection
7:       Extract the image patch $\boldsymbol{y}_{m,t}$ from the Adaboost desction
8:       Initialize the SPPCA template updater $U_m$ using $\boldsymbol{y}_{m,t}$
9:     end for
10:    $M = M + M_{new}$
11:  end if
12:
13:  for $m = 1$ to $M$ do
14:    Generate a new template $\tilde{\boldsymbol{y}}_{m,t}$ by Eq. (4.19) from the SPPCA template
       updater $U_m$
15:
16:    Propose new particles $\{\boldsymbol{x}_{m,t}^{(i)}\}_{i=1}^{N}$ by Eq. (6.8)
17:    Compute the observation likelihood for $\{\boldsymbol{x}_{m,t}^{(i)}\}_{i=1}^{N}$ by Eq. (6.2)
18:    Update the importance weights $\{w_{m,t}^{(i)}\}_{i=1}^{N}$ by Eq. (6.6)
19:    Generate unweighted samples $\{\tilde{\boldsymbol{x}}_{m,t}^{(i)}\}_{i=1}^{N}$ by resampling $\{\boldsymbol{x}_{m,t}^{(i)}\}_{i=1}^{N}$ according
       to the importance weights $\{w_{m,t}^{(i)}\}_{i=1}^{N}$
20:
21:    $\bar{\boldsymbol{x}}_{m,t} = mean(\{\tilde{\boldsymbol{x}}_{m,t}^{(i)}\}_{i=1}^{N})$
22:    Extract image patch $\bar{\boldsymbol{y}}_{m,t}$ centered in $\bar{\boldsymbol{x}}_{m,t}$
23:
24:    Update $\{\boldsymbol{z}_{m,t}, s_{m,t}\}$ of the SPPCA template updater $U_m$ by $\bar{\boldsymbol{y}}_{m,t}$ using
       RBPF described in Section 4.4.2
25:    Recognize the action of the player $a_{m,t}$ by $\bar{\boldsymbol{y}}_{m,t}$ using the SMLR+HOG
       action recognizer described in Chapter 5
26:  end for
27:
28:  Remove $M_1$ targets whose AdaBoost confidence is below a threshold
29:  Merge $M_2$ pairs of targets who overlap with each others
30:  $M = M - M_1 - M_2$
31: end for
```

recognizer described in Chapter 5. The action recognizer will classify the action of the targets at time $t+1$ and return the probability $p(\boldsymbol{a}_{t+1}|\bar{\boldsymbol{y}}_{t+1}, \boldsymbol{w})$.

There are also mechanisms to remove and merge the targets. The targets will be removed either when their Adaboost confidence is lower than a threshold, or when the bounding boxes are out of the image. The merge operation is performed when there is significant overlap between two bounding boxes. The mean of the two bounding boxes will be computed and the target with the lower Adaboost confidence will be removed.

6.7 Experiments

We evaluate the tracking performance of our system using a hockey game sequence with more than 2000 frames. This sequence is very challenging due to the following reasons. Firstly, the resolution of the video sequence is only 320×240. The low resolution sequence makes tracking and action recognition challenging because the appearance of the targets cannot be clearly observed. Secondly, the camera is *not* stationary. The moving camera increases the complexity of the system because no background subtraction technique can be used. Thirdly, hockey players interact with others very often. Occlusions between two, three, or even four players occur frequently over the entire sequence. Lastly, significant lighting condition changes occasionally occur due to the flash of other cameras.

6.7.1 Parameter Settings

In all experiments, the HOG descriptor is constructed by sampling the SIFT descriptors of size 16×16 from the image patch with a 16-pixel spacing, and every SIFT descriptor is computed with $n_w = n_h = 2$, $n_b = 8$ (yielding a HOG descriptor of dimensionality 128). The color histograms are computed by setting $N_h = N_s = N_h = 10$, yielding a HSV color histogram of dimensionality 110. The SPPCA template updater is constructed by utilizing 30 subspaces ($n_s = 30$) and 30 principal components ($n_z = 30$). During the updating operation, 30 particles are used in the Rao-Blackwellized Particle Filter in the SPPCA template updater. The configurations of various tracking systems are described as follows.

The HSV tracker

This simplest setting uses only HSV color histograms as the observation model. Since there is usually no noticeable change in the colors of hockey players, we use the observation of the first frame as the color template, and we do not update the color template over time. We use 30 particles for each target and set $\alpha_{ada} = 0$ in Eq. (6.8) (thus the tracking system becomes a bootstrap particle filter).

The HOG+SPPCA tracker

This setting uses only the HOG descriptors as the observation model, and applies the SPPCA template updater described in Chapter 4 to update the HOG templates over time. We use 30 particles for each target and set $\alpha_{ada} = 0$ in Eq. (6.8) (thus the tracking system becomes a bootstrap particle filter).

The HSV+HOG+SPPCA tracker

This tracker is a combination of the HSV tracker and the HOG+SPPCA tracker. We use both the HSV color histograms and the HOG descriptor as the observation model, and the HOG templates are updated by the SPPCA template updater. We use 30 particles for each target and set $\alpha_{ada} = 0$ in Eq. (6.8) (thus the tracking system becomes a bootstrap particle filter).

The HSV+HOG+SPPCA+BPF tracker

This is the complete version of our tracking system. All parameter settings are the same as the HSV+HOG+SPPCA tracker, except that we employ multiple independent Boosted Particle Filters (BPFs) to track the locations of the targets (thus

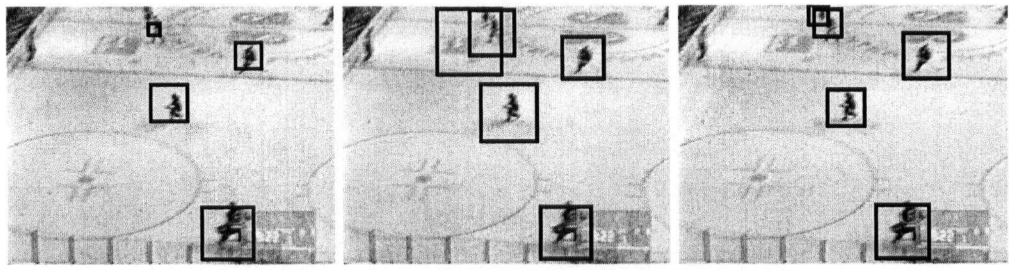

(a) HSV (b) HOG+SPPCA (c) HSV+HOG+SPPCA

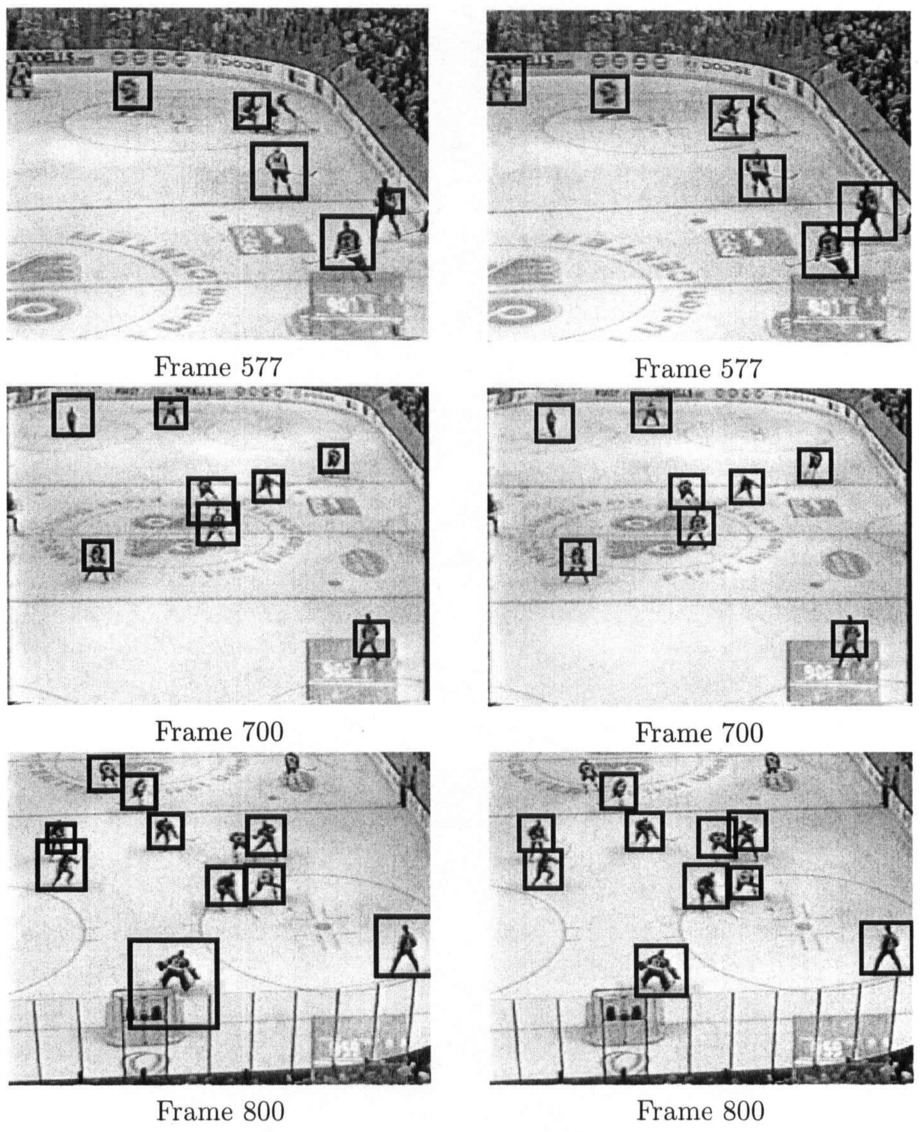
Figure 6.3: **Tracking results I:** This figure shows the comparison between HSV, HOG+SPPCA, and HSV+HOG+SPPCA. In (a), there are a few boxes that have wrong scales. In (b), there is a box that is already shifted away from the object. However, all the targets are correctly tracked in (c).

$\alpha_{ada} \neq 0$). Similar to other settings, 30 particles are used for each target.

6.7.2 Results

Firstly, we show that using two observation models (i.e., the HOG descriptors and the HSV color histograms) is better than using only either one of them alone. When we only use the HSV color histograms as the observation model as shown in Figure 6.3 (a), we can observe that the localization of the targets is correct. However, some of the estimated boxes have incorrect scale, i.e., the bounding boxes only contain a part of the body of the hockey players. When we only utilize the HOG descriptors with the SPPCA template updater as shown in Figure 6.3 (b), the scale of the estimated boxes is better than Figure 6.3 (b) because they usually contain the entire body of the hockey players. However, the localization of the estimated boxes is worse because the HOG descriptors are more sensitive to the background clutter. When combining the HSV color histograms and the HOG descriptors together as shown in Figure 6.3 (c), we achieve a better scale and localization estimation of the targets.

Secondly, we show the benefit of using the Boosted Particle Filter. In Figure 6.4, we compare the performance of the HSV+HOG+SPPCA tracker (the nonBPF



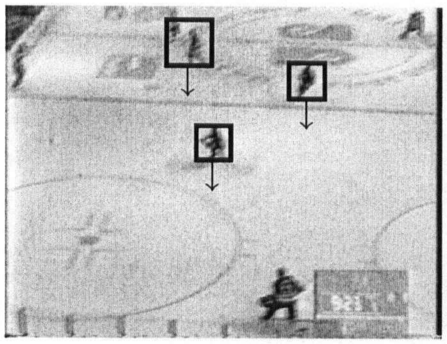
(a) The nonBPF tracker (b) The BPF tracker

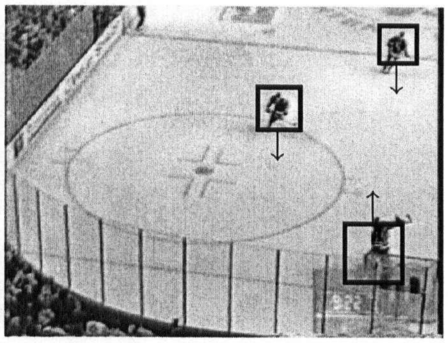
Figure 6.4: **Tracking results II:** This figure shows the comparison between HSV+HOG+SPPCA with and without BPF. The left column shows two images that are generated by not using BPF. The right column has two other images that are generated by BPF.

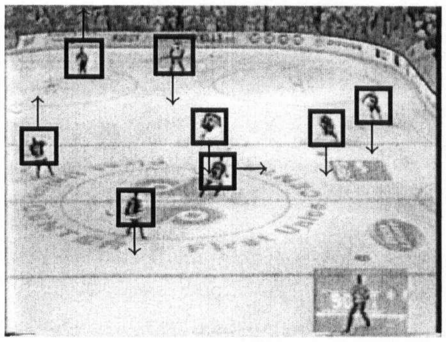
tracker) and the HSV+HOG+SPPCA+BPF tracker (the BPF tracker) in the same sequence. The difference between the two trackers is that one employs the Adaboost detection in its proposal distribution (the BPF tracker) while another is a standard bootstrap particle filter (the nonBPF tracker). In frame 577, the nonBPF tracker keeps track of fewer targets, and one of the targets at the right border of the image has already had an incorrectly sized box. In the meanwhile, BPF has no evident problems. In frame 700, the nonBPF tracker has more inaccurate sized boxes, and the most obvious one occurs in the center of the image while one player partially occludes the other. In the same time, the BPF tracker still has nearly perfect estimation of both the scales and the locations of the players. In frame 800, the nonBPF tracker assigns an incorrectly sized box to some of the hockey players and the goalie whereas the BPF tracker still has accurate estimation of the scales and the locations of the players.

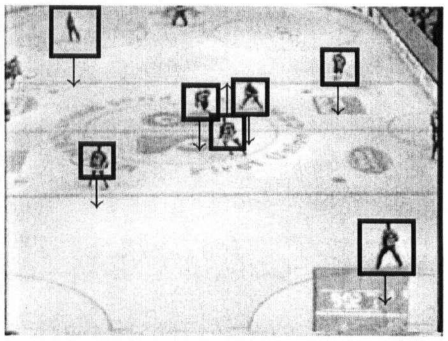
Finally, we show the experimental results of the entire system in Figure 6.5. In this experiment, we employ the Boosted Particle Filter with a joint likelihood computed from both the HSV color histograms and the HOG descriptors, and the HOG templates are updated by the SPPCA template updater (the HSV+HOG+SPPCA+BPF tracker). The action recognizer described in Chapter 5 is also employed to classify the players' actions online after we estimate the locations and sizes of the players. We can observe that the entire system can simultaneously track and recognize multiple hockey players' actions.

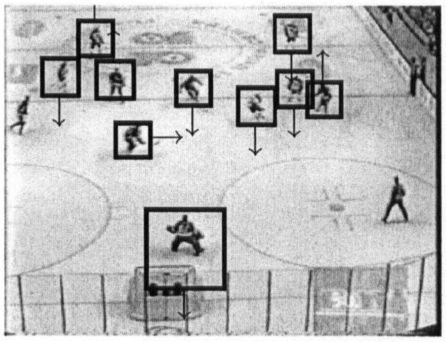
We acknowledge that it is extremely difficult to evaluate a multi-target tracking system due to complex multi-target interactions and many algorithmic elements that influence the overall performance of the tracking system. Therefore, we additionally provide a set of video sequences that contain visual tracking results with each configuration of our system. The URL to the data is given in [1].


(a) Frame 97


(b) Frame 116


(c) Frame 682


(d) Frame 710


(e) Frame 773

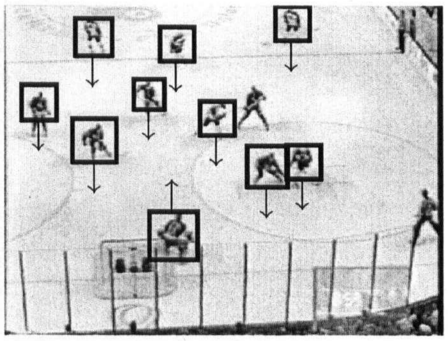

(f) Frame 814

Figure 6.5: **Tracking and action recognition results:** This figure shows the final result of our system. All figures have the size of 320 × 240. The square box represents the tracking region and arrows indicate the recognized action of the player.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis presents a system that can automatically track multiple hockey players and simultaneously recognize their actions given a single broadcast video sequence. There are three contributions. Firstly, we employ the Histograms of Oriented Gradients (HOG) and the HSV color histogram as the observation likelihood, and present Switching Probabilistic Principal Component Analysis (SPPCA) to model the appearance of the players by a mixture of local subspaces. SPPCA can be used to update the template of the HOG descriptor of the tracked targets, and we show that SPPCA has a better predictive accuracy than the HMM template updater previous presented in our early work [33]. Secondly, we recognize the players' actions by incorporating the HOG descriptors with the Sparse Multinomial Logistic Regression (SMLR) classifier. A robust motion-to-motion similarity measure is also utilized to take consideration of actions of different speed. Experimental results show that the proposed action recognizer outperforms [11, 64] in both accuracy and speed. Finally, we augment the Boosted Particle Filter (BPF) with new observation model and the SPPCA template updater and improves the robustness of the tracking system. Experimental results show that the entire system can track and recognize multiple hockey players' action robustly in near real-time.

7.2 Future Work

Although our system can detect, track, and recognize multiple hockey players' actions automatically, several extensions could be made in the future. For example, in the Switching Probabilistic Principal Component Analysis (SPPCA) framework, the observation is generated from the hidden states by a Gaussian distribution. A more sophisticated generative model such as Gaussian Processes [49] and the Binary Latent Variable model [54] can be utilized to better model the observation. The dynamics of the hidden states $\boldsymbol{z}$ can be modeled more accurately. In our system, we assume that the hidden state $\boldsymbol{z}_t$ remains the same when the switch s_t does not change. However, the dynamics could be modeled in a more complicated way. For example, Gaussian Process Regression can be used to predict the next templates of the targets [38, 59]. In these cases, the Rao-Blackwellized Particle Filter would no longer be applied because the posterior distribution over $\boldsymbol{z}_t$ can not be solved analytically by the Kalman Filter [63]. Nevertheless, particle filtering still can be used in these models.

The action recognizer and the tracker could be coupled more tightly in the future. In this thesis, the action recognizer utilizes the information provided by the tracker. However, no information is fed back to the tracker. In the future, it is possible to utilize the action information to guide the template updater as in [32, 33]. This idea shares the same merits with [28] which exploited the identities of the targets to help generate a more accurate template. In the tracking framework, we could also use the action information incorporated with the transition prior of the particle filter to obtain a more accurate estimation of the locations of the targets.

The Boosted Particle Filter (BPF) could be further improved. In this thesis, we employ multiple independent BPFs to track multiple targets. Simple target intialization, removal, and merge operations are also implemented. However, user-specified thresholds are needed to determine when these operations are triggered. In the future, it is possible to use a more sophisticated Markov Chain Monte Carlo technique to perform these tasks automatically [25]. Furthermore, the particle filtering framework requires a good transition prior in order to obtain a more accurate state estimation. In this thesis, however, we do not have an accurate transition prior due to the moving camera. If we have an accurate homography between the image and the rink coordinate system, then the locations and speed of the players in the rink coordinate system can be estimated. This information can help us to establish a better transition prior because the movement of the players in the rink coordinate system is approximated linear.

The cascaded Adaboost detector can be improved. The cascaded Adaboost detector requires user-specified thresholds for each layer. These threshold might be learned automatically. Moreover, the number of labelled training data can be considerably reduced if we could employ the active learning technique such as [58] to greedily determine which training images should be labelled.

Bibliography

[1] http://www.cs.ubc.ca/~vailen/imavis. 63

[2] Arulampalam, M. S., Maskell, S., Gordon, N., and Clapp, T. A Tutorial on Particle Filters for On-line Non-linear/Non-Gaussian Bayesian Tracking. *IEEE Transactions on Signal Processing 50*, 2 (February 2002), 174–188. 6, 32

[3] Avidan, S. Ensemble Tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence 29*, 2 (February 2007), 261–271. 18

[4] Black, M., and Jepson, A. EigenTracking: Robust Matching and Tracking of Articulated Objects Using a View-Based Representation. *International Journal of Computer Vision 26*, 1 (1998), 63–84. 9, 24

[5] Cawley, G., Talbot, N., and Girolami, M. Sparse Multinomial Logistic Regression via Bayesian L1 Regularisation. In *Advances in Neural Information Processing Systems 19* (2007). 41, 42

[6] Comaniciu, D., and Meer, P. Mean Shift: A Robust Approach Toward Feature Space Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*, 5 (2002), 603–619. 51, 57

[7] Cootes, T., Taylor, C., Cooper, D., and Graham, J. Active shape models – their training and application. *Computer Vision and Image Understanding 61*, 1 (1995), 38–59. 11

[8] Dalal, N., and Triggs, B. Histograms of Oriented Gradients for Human Detection. In *IEEE Conference on Computer Vision and Pattern Recognition* (2005), vol. 1, pp. 886–893. 5, 18, 19, 20, 39, 46

[9] Doucet, A., de Freitas, N., Murphy, K., and Russell, S. Rao-Blackwellised Filtering for Dynamic Bayesian Networks. In *Uncertainty in Artificial Intelligence* (2000), pp. 176–183. 9, 29, 32, 33

[10] Doucet, A., Freitas, N. D., and Gordon, N., Eds. *Sequential Monte Carlo Methods in Practice*. Springer, 2005. 15, 52, 53, 54

[11] Efros, A., Breg, C., Mori, G., and Malik, J. Recognizing Action at a Distance. In *International Conference on Computer Vision* (2003), pp. 726–733. 6, 13, 40, 42, 45, 47, 65

[12] Elgammal, A., Duraiswami, R., and Davis, L. S. Probabilistic Tracking in Joint Feature-Spatial Spaces. In *IEEE Conference on Computer Vision and Pattern Recognition* (2003), vol. 1, pp. 781–788. 9, 14

[13] Freeman, W., Tanaka, K., Ohta, J., and Kyuma, K. Computer vision for computer games. In *International Conference on Automatic Face and Gesture Recognition* (1996), pp. 100–105. 12, 47

[14] Freeman, W. T., and Roth, M. Orientation Histograms for Hand Gesture Recognition. In *International Workshop on Automatic Face and Gesture Recognition* (1995). 12

[15] Gavrila, D. The Visual Analysis of Human Movement: A Survey. *Computer Vision and Image Understanding 73*, 1 (January 1999), 82–98. 1, 12

[16] Giebel, J., Gavrila, D., and Schnörr, C. A Bayesian Framework for Multi-cue 3D Object Tracking. In *European Conference on Computer Vision* (2004), pp. 241–252. 11, 18, 33

[17] Ho, J., Lee, K., Yang, M., and Kriegman, D. Visual Tracking Using Learned Linear Subspaces. In *IEEE Conference on Computer Vision and Pattern Recognition* (2004), vol. 1, pp. 782–789. 9

[18] Hu, W., Tan, T., Wang, L., and Maybank, S. A survey on visual surveillance of object motion and behaviors. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews 34*, 3 (2004), 334–352. 1, 12

[19] Hue, C., Cadre, J. L., and Pérez, P. Tracking Multiple Objects with Particle Filtering. *IEEE Transactions on Aerospace and Electronic Systems 38*, 3 (2002), 791–812. 15

[20] Intille, S., David, J., and Bobick, A. Real-Time Closed-World Tracking. In *IEEE Conference on Computer Vision and Pattern Recognition* (1997), pp. 697–703. 15

[21] Isard, M., and Blake, A. CONDENSATION–Conditional Density Propagation for Visual Tracking. *International Journal of Computer Vision 29*, 1 (1998), 5–28. 15

[22] Isard, M., and MacCormick, J. BraMBLe: A Bayesian Multiple-Blob Tracker. In *International Conference on Computer Vision* (2001), vol. 2, pp. 34–41. 15

[23] Jain, A., and Dubes, R. *Algorithms for Clustering Data.* Prentice Hall, 1988. 28

[24] Khan, Z., Balch, T., and Dellaert, F. A Rao-Blackwellized Particle Filter for EigenTracking. In *IEEE Conference on Computer Vision and Pattern Recognition* (2004), vol. 2, pp. 980–986. 9, 10, 24

[25] Khan, Z., Balch, T., and Dellaert, F. MCMC-Based Particle Filtering for Tracking a Variable Number of Interacting Targets. *IEEE Transactions on Pattern Analysis and Machine Intelligence 27*, 11 (2005), 1805–1819. 67

[26] Koller, D., Weber, J., and Malik, J. Robust Multiple Car Tracking with Occlusion Reasoning. In *European Conference on Computer Vision* (1994), pp. 186–196. 15

[27] Krishnapuram, B., Carin, L., Figueiredo, M. A., and Hartemink, A. J. Sparse Multinomial Logistic Regression: Fast Algorithms and Generalization Bounds. *IEEE Transactions on Pattern Analysis and Machine Intelligence 27*, 6 (June 2005), 957–968. 6, 40, 41, 42

[28] Lee, K., Ho, J., Yang, M., and Kriegman, D. Visual tracking and recognition using probabilistic appearance manifolds. *Computer Vision and Image Understanding 99* (2005), 303–331. 10, 66

[29] Lim, H., Morariu, V. I., Camps, O. I., and Sznaier, M. Dynamic Appearance Modeling for Human Tracking. In *IEEE Conference on Computer Vision and Pattern Recognition* (2006), vol. 1, pp. 751–757. 10

[30] Ling, H., and Okada, K. Diffusion Distance for Histogram Comparison. In *IEEE Conference on Computer Vision and Pattern Recognition* (2006), vol. 1, pp. 246–253. 51, 53

[31] Lowe, D. G. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision 60*, 2 (2004), 91–110. 19, 20, 46

[32] Lu, W.-L., and Little, J. J. Simultaneous Tracking and Action Recognition using the PCA-HOG Descriptor. In *The Third Canadian Conference on Computer and Robot Vision* (2006). 15, 16, 36, 66

[33] Lu, W.-L., and Little, J. J. Tracking and Recognizing Actions at a Distance. In *ECCV Workshop on Computer Vision Based Analysis in Sport Environments* (2006), pp. 49–60. 15, 16, 36, 65, 66

[34] Lucas, B., and Kanade, T. An iterative image registration technique with an application to stereo vision. In *Proceedings of Imaging understanding workshop* (1981), pp. 121–130. 47

[35] MacCormick, J., and Blake, A. A probabilistic exclusion principle for tracking multiple objects. In *International Conference on Computer Vision* (1999), vol. 1, pp. 572–578. 15

[36] Matthews, L., Ishikawa, T., and Baker, S. The Template Update Problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 6 (2004), 810–815. 8, 9

[37] Misu, T., Naemura, M., Zheng, W., Izumi, Y., and Fukui, K. Robust Tracking of Soccer Players Based on Data Fusion. In *International Conference on Pattern Recognition* (2002), pp. 556–561. 15

[38] Moon, K., and Pavlović, V. Impact of Dynamics on Subspace Embedding and Tracking of Sequences. In *IEEE Conference on Computer Vision and Pattern Recognition* (2006), vol. 1, pp. 198–205. 11, 66

[39] Murphy, K., and Russel, S. Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks. In *Sequential Monte Carlo Methods in Practice*, A. Doucet, N. de Freitas, and N. Gordon, Eds. Springer-Verlag, 2001. 32, 33

[40] Murphy, K. P. Learning Switching Kalman Filter Models. Tech. Rep. Tech Report 98-10, Compaq Cambridge Research Lab, 1998. 25, 29, 31

[41] Neal, R. M., and Hinton, G. A new view of the EM algorithm that justifies incremental, sparse, and other variants. In *Learning in Graphical Models*, M. I. Jordan, Ed. Kluwer Academic Publishers, 1998, pp. 355–368. 25, 28

[42] Needham, C., and Boyle, R. Tracking multiple sports players through occlusion, congestion and scale. In *British Machine Vision Conference* (2001), pp. 93–102. 15

[43] Oh, S. M., Rehg, J. M., Balch, T., and Dellaert, F. Learning and Inference in Parametric Switching Linear Dynamic Systems. In *International Conference on Computer Vision* (2005), vol. 2, pp. 1161–1168. 29

[44] Okuma, K., Taleghani, A., de Freitas, N., Little, J. J., and Lowe, D. G. A Boosted Particle Filter: Multitarget Detection and Tracking. In *European Conference on Computer Vision* (2004), pp. 28–39. 5, 6, 15, 18, 51, 54, 55

[45] Pavlović, V., Rehg, J., Cham, T., and Murphy, K. A Dynamic Bayesian Network Approach to Figure Tracking Using Learned Dynamic Models. In *International Conference on Computer Vision* (1999), vol. 1, pp. 94–101. 29

[46] Pérez, P., Hue, C., Vermaak, J., and Gangnet, M. Color-Based Probabilistic Tracking. In *European Conference on Computer Vision* (2002), pp. 661–675. 15, 16, 18

[47] Porikli, F. Integral Histogram: A Fast Way to Extract Histograms in Cartesian Spaces. In *IEEE Conference on Computer Vision and Pattern Recognition* (2005), vol. 1, pp. 829–836. 5, 21

[48] Rabiner, L. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE 77*, 2 (1989), 257–286. 37

[49] Rasmussen, C. E., and Williams, C. K. I. *Gaussian Processes for Machine Learning.* The MIT Press, 2006. 66

[50] Ross, D., Lim, J., and Yang, M. Adaptive Probabilistic Visual Tracking with Incremental Subspace Update. In *European Conference on Computer Vision* (2004), pp. 470–482. 9

[51] RUI, Y., AND CHEN, Y. Better Proposal Distributions: Object Tracking Using Unscented Particle Filter. In *IEEE Conference on Computer Vision and Pattern Recognition* (2001), vol. 2, pp. 786–793. 15, 54

[52] SAUL, L. K., AND ROWEIS, S. T. Think Globally, Fit Locally: Unsupervised Learning for Low Dimensional Manifolds. *Journal of Machine Learning Research 4* (2003), 119–155. 10

[53] SHEVADE, S., AND KEERTHI, S. A simple and efficient algorithm for gene selection using sparse logistic regression. *Bioinformatics 19*, 17 (2003), 2246–2253. 41, 42

[54] TAYLOR, G., HINTON, G., AND ROWEIS, S. Modeling Human Motion Using Binary Latent Variables. In *Advances in Neural Information Processing Systems 19* (Cambridge, MA, 2007), B. Schölkopf, J. Platt, and T. Hoffman, Eds., MIT Press. 66

[55] TIPPING, M., AND BISHOP, C. Mixtures of Probabilistic Principal Component Analyzers. *Neural Computation 11* (1999), 443–482. 25, 31

[56] TIPPING, M., AND BISHOP, C. Probabilistic principal component analysis. *Journal of Royal Statistical Society B 61*, 3 (1999), 611–622. 9

[57] TOYAMA, K., AND BLAKE, A. Probabilistic Tracking with Exemplars in a Metric Space. *International Journal of Computer Vision 48*, 1 (2002), 9–19. 9, 16, 36

[58] TUR, G., HAKKANI-TÜR, D., AND SCHAPIRE, R. E. Combining active and semi-supervised learning for spoken language understanding. *Speech Communication 45* (2005), 171–186. 67

[59] URTASUN, R., FLEET, D., AND FUA, P. 3D People Tracking with Gaussian Process Dynamical Models. In *IEEE Conference on Computer Vision and Pattern Recognition* (2006), vol. 1, pp. 238–245. 11, 66

[60] van der Merwe, R., Doucet, A., de Freitas, N., and Wan, E. The Unscented Particle Filter. In *Advances in Neural Information Processing Systems 13* (2001). 54

[61] Viola, P., and Jones, M. Rapid Object Detection using a Boosted Cascade of Simple Features. In *IEEE Conference on Computer Vision and Pattern Recognition* (2001), vol. 1, pp. 511–518. 6, 21, 54

[62] Wang, J., Fleet, D., and Hertzmann, A. Gaussian Process Dynamical Models. In *Advances in Neural Information Processing Systems 18* (2006). 11

[63] Welch, G., and Bishop, G. An Introduction to the Kalman Filter. Tech. Rep. TR 95-041, Deparment of Computer Science, University of North Carolina at Chapel Hill, 1995. 33, 34, 35, 66

[64] Wu, X. Templated-based Action Recognition: Classifying Hockey Players' Movement. Master's thesis, The University of British Columbia, 2005. 13, 45, 47, 65

[65] Wu, Y., and Huang, T. Robust Visual Tracking by Integrating Multiple Cues Based on Co-Inference Learning. *International Journal of Computer Vision 58*, 1 (2004), 55–71. 18

[66] Yamato, J., Ohya, J., and Ishii, K. Recognizing Human Action in Time-Sequential Images using Hidden Markov Model. In *IEEE Conference on Computer Vision and Pattern Recognition* (1992), pp. 379–385. 14

[67] Yang, C., Duraiswami, R., and Davis, L. Fast Multiple Object Tracking via a Hierarchical Particle Filter. In *International Conference on Computer Vision* (2005), vol. 1, pp. 212–219. 18