



Tracking and recognizing actions of multiple hockey players using the boosted particle filter

Wei-Lwun Lu *, Kenji Okuma, James J. Little

University of British Columbia, Department of Computer Science, 2366 Main Mall, Vancouver, BC, Canada V6T 1Z4

ARTICLE INFO

Article history:

Received 20 March 2007
Received in revised form 20 February 2008
Accepted 22 February 2008

Keywords:

Tracking
Action recognition
Boosted particle filter

ABSTRACT

This article 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. There are three contributions. 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 off-line 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 augment the boosted particle filter (BPF) with a new observation model and a template updater that improves the robustness of the tracking system. Finally, we recognize the players' actions by combining the HOG descriptors with a pure multi-class sparse classifier with a robust motion similarity measure. Experiments on long sequences show promising quantitative and qualitative results.

© 2008 Elsevier B.V. All rights reserved.

1. Introduction

1.1. Motivation

Visual tracking and action recognition have gained more and more attention 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 [3,4].

Our motivation arises in the challenge of inventing a system that tracks multiple hockey players in a video sequence and simultaneously classifies their actions. Such a system is 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 their actions.

Fig. 1 shows the examples of the input and output of the system. The input 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. The camera is *not* stationary and the sequence contains fast camera motions and significant zoom in/out. The system outputs the locations and sizes of the hockey players (represented by bounding boxes) as well as their moving directions (represented by arrows).

1.2. The problem

This article focuses on two major problems: tracking multiple hockey players from a single video sequence and simultaneously recognizing their actions.

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 stream. Tracking typically starts when the system detects that a hockey player appears in the video stream. The system then stores 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. Fig. 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 sections, we will tackle these sub-problems one by one.

* Corresponding author. Tel.: +1 604 710 9397.

E-mail addresses: vailen@cs.ubc.ca (W.-L. Lu), okumak@cs.ubc.ca (K. Okuma), little@cs.ubc.ca (J.J. Little).

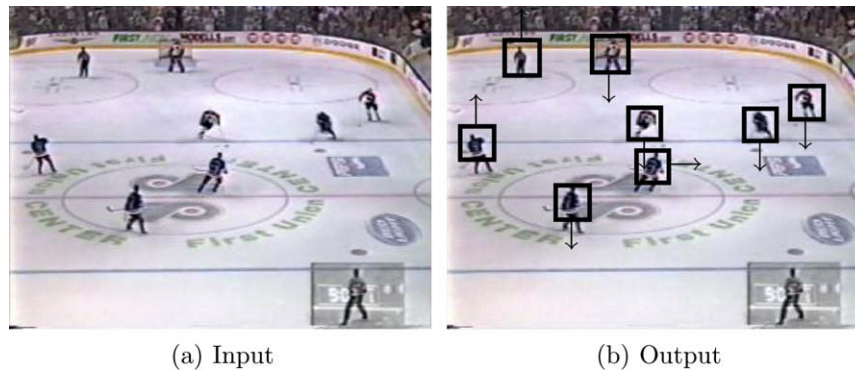


Fig. 1. System input and output.

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 30–70 pixels heights obtained from the tracking system. 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 utilize these image patches to classify the person's actions. The action labels can be long-term one such as “shooting”, “passing”, and “skating”, or short-term one such as “skating left” and “skating right”.

1.3. System outline

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

In Section 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 Section 3, we introduce the observation models used in our system: the Hue–Saturation–Value (HSV) color histogram [5] and the Histogram of Oriented Gradients (HOG) descriptor [6]. The HSV color histograms and the HOG descriptors encode the color and shape information of the image patches of hockey players, respectively.

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 Section 4, we describe a Switching Probabilistic Principal Component Analysis (SPPCA) template updater to predict and update the templates used by the tracker. The parameters of the SPPCA template updater can be learned off-line using a set of labeled training examples.

The third problem we face is how to classify the actions of hockey players from the image patches extracted by the tracker. In Section 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 [2]. By incorporating the SMLR

classifier and the motion similarity measure introduced by Efros et al. [7], 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 Section 6, we detail the boosted particle filter (BPF). The BPF tracker augments the standard Particle Filter [8] by incorporating cascaded Adaboost detection [9] 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. Fig. 2 shows the system diagram of our algorithm. Section 7 present the performance evaluation and Section 8 concludes this article.

2. Previous work

2.1. Tracking & template updating

The goal of an automatic tracking system is to estimate the locations and sizes of the targets in a video sequence, and it is still an open problem in many settings, including car surveillance [10], sports [11,12] and smart rooms [13] among many others [14–16]. 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 [17].

One alternative is the *exemplar tracker* [18], which uses a fixed number of pre-learned exemplars as templates. The problem of the exemplar tracker is that only a fixed number of examples can be used as templates to model the appearance of targets. The *EigenTracker* [19] was then introduced to tackle this problem. In *EigenTracker*, a infinite number of templates can be generated by a linear combination of Eigenfaces (principle components of training examples). Later on, Khan et al. [20] introduced a probabilistic version of the *EigenTracker* (using Probabilistic PCA [21]) and applied

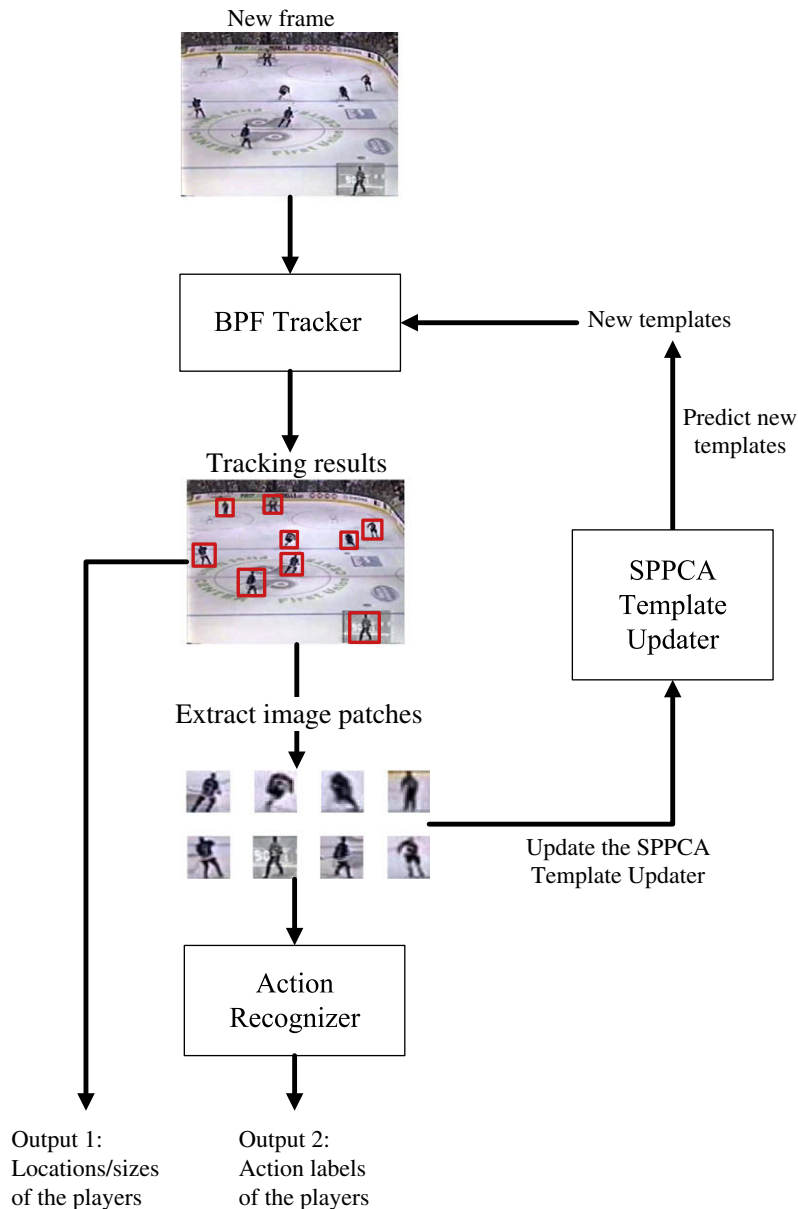


Fig. 2. System diagram: The system contains three important components: the boosted particle filter (BPF) tracker, the Switching Probabilistic Principal Components Analysis (SPPCA) template updater, and the action recognizer.

it to track honey bees. Recently, researchers also tried to use more advanced techniques such as Locally Linear Embedding (LLE) [22] and Gaussian Process Dynamic Models (GPDM) [23,24] to model the templates of the tracker.

Given the template, the next question is how to automatically estimate the location of the targets. In general, the problem of tracking visual features in complex environments is fraught with uncertainty [13]. 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 [25–28]. 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. [14] 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) [15] 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 [5], we can relax this assumption of a fixed background in order to deal with realistic TV video sequences, where the background changes.

In this article, we augment the BPF with several extensions. First our extended BPF uses both 2D and 1D color histograms as the color model and Histograms of Oriented Gradients (HOG) as the shape model whereas Okuma et al. in [5] use only a 1D color histogram for their observation model. Secondly, we use the diffusion distance to compare histograms as it is shown in [29] to be more robust to deformation and quantization effects than the Bhattacharyya coefficients that are used in [5]. Thirdly, we use a

mode-seeking algorithm similar to the mean shift [30] to generate a naïve proposal when there is no detection, which further improves the performance of the BPF regardless of occasional sparse Adaboost detections. Lastly, we use SPPCA to update the shape template of the tracker, while [5] did not update their observational model and use only the initialized model as their template.

The work most similar to ours is Giebel et al. [31]. 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) [32], 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. During run time, 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 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. 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 [3] and Hu et al. [4].

Freeman et al. [33] 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 [34], they further divided the images into cells, and computed the orientation histograms of all cells. However, their approach determines the gesture of the target only by the current posture of the person. No previous posture information is used. Recently, Wang et al. [58] have extended this approach using a hierarchical model.

Efros et al. [7] 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.

Wu [35] extended Efros et al. [7] by introducing another motion descriptor, the Decomposed Image Gradients (DIG). The DIG descriptor can be constructed 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.

The problem of action recognition can be also formulated in a generative probabilistic model. For example, [36] 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 determined by the HMM having the maximum likelihood. In our previous work, we also em-

ployed HMMs to recognize the target's actions [37,38]. Instead of using the entire video sequence, we used a fixed-length sliding window to determine the target's actions.

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 [31,39–41]. In this article, 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 [6] to encode the shape information.

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 [25,5]. 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 [25,5].

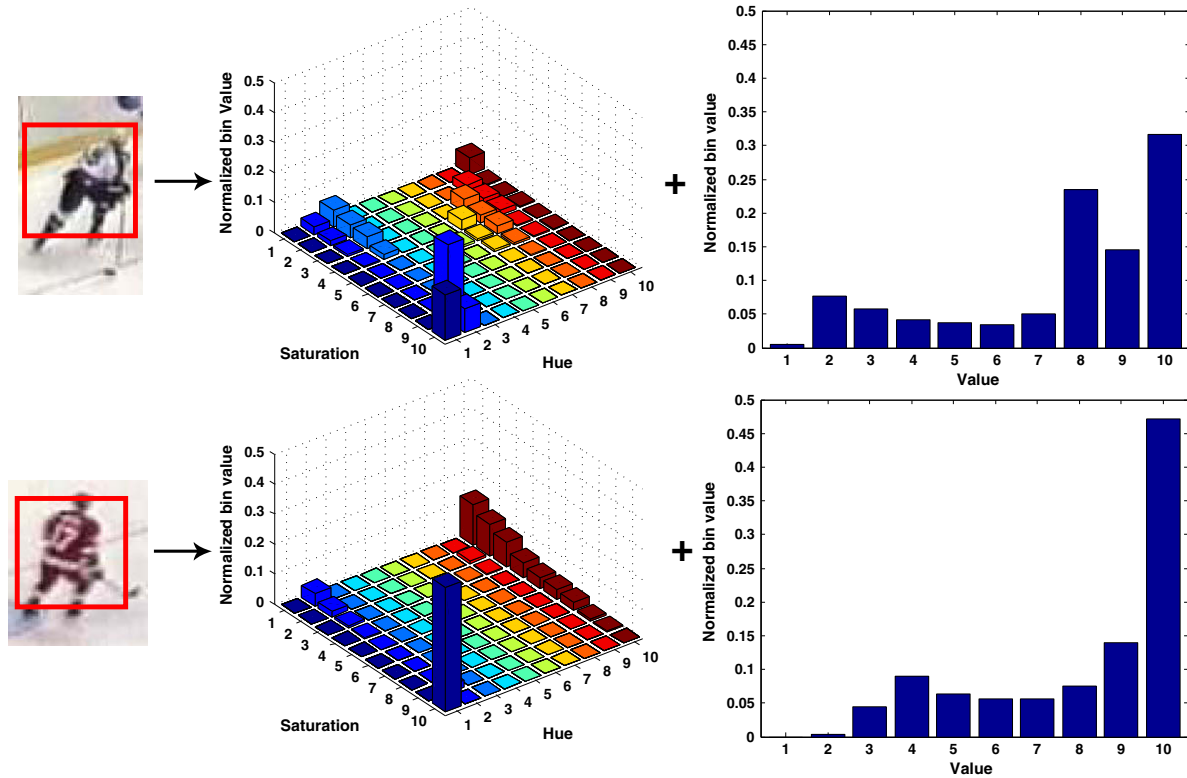
Our color observation model is composed of a 2D histogram based on Hue and Saturation and a 1D histogram based on value. Both histograms are normalized such that all bins are sum to one. We assign the same number of bins for each color component, i.e., $N_h = N_s = N_v = 10$, and it results in a $N_h \times N_s + N_v = 10 \times 10 + 10 = 110$ dimension HSV histogram. Fig. 3 shows two instances of the HSV color histograms.

3.2. Shape

We apply the Histograms of Oriented Gradient (HOG) descriptor [6] to encode the shape information of the targets. The HOG descriptor is computed by sampling a set of the SIFT descriptors [42] with a fixed spacing over the image patches. Combined 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 [6]. In this article, 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 [42] to capture the appearance information centered on the detected SIFT features. To compute the SIFT descriptor, we first resize the image patch to an $p_w \times p_h$ patch and then 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 [42] 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 1D histogram of gradient directions over the pixels of the cell. In this article, 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 HOG descriptor is constructed by uniformly sampling the SIFT descriptor of the same size over the image patch with a fixed



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

Fig. 3. 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 (both histograms are normalized such that all bins sum to one). 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.

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. Fig. 4 shows an example of the HOG descriptor.

4. Template updating

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. 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, the location estimates of the targets inevitably contain some errors, and thus the estimated bounding box in the previous frame may contain only a part of the targets, background pixels, or even other objects. As a result, if we use these “polluted” image patches as templates in the tracking system, the errors will be usually accumulated and finally lead to loss of targets. Fig. 5 gives an example of applying the naïve updating method in tracking, and we can observe that the tracker quickly fails to track the targets.

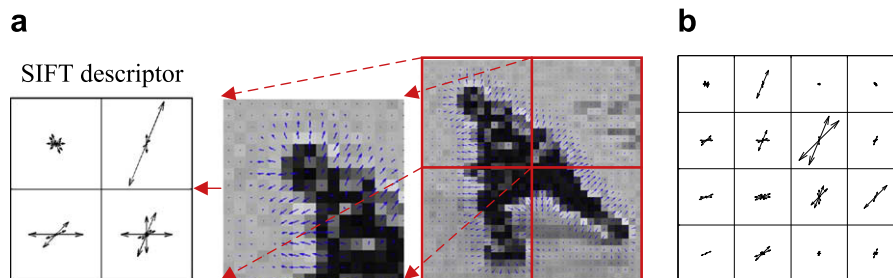


Fig. 4. The HOG descriptor: (a) Right: The image gradient of a 32×32 image. Center: A block of size 16×16 . Left: The SIFT descriptor of the block with $n_w = n_h = 2, n_b = 8$. (b) The HOG descriptor of the image computed by sampling SIFT descriptors of size 16×16 with a 16-pixels spacing.

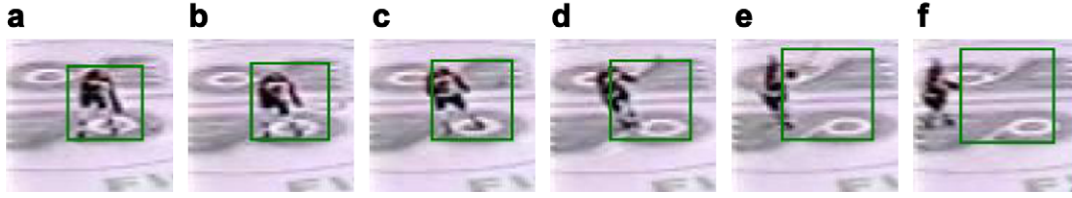


Fig. 5. Tracking with a naïve template updater: This figure shows the results of tracking a single hockey player with a naïve template updater, i.e., update the template using the image patch of the previous location estimates. Since the location estimates inevitably have errors, the template is gradually polluted by the background in (c), (d), (e), and the tracker finally loses its targets in (f).

In this article, 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 article 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. The SPPCA template updater consists of three major operations: *learning*, *updating*, and *prediction*. The *learning* operation learns the parameters of template models off-line using from a set of training examples. During tracking, the SPPCA template updater performs two operations: *updating* and *prediction*. As shown in Fig. 6, the tracker extracts image patches centered in the estimated locations after tracking the locations and sizes of the targets. The *update* operation utilizes these image patches to update the template models. The *prediction* operation 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.

4.1. Probabilistic graphical model

Let $s_t \in \{1, \dots, n_s\}$ be a discrete random variable representing the subspace we use at time t , $\mathbf{y}_t \in \mathbb{R}^{n_y}$ be a continuous random variable representing the observation at time t , and $\mathbf{z}_t \in \mathbb{R}^{n_z}$ 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 Fig. 7.

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

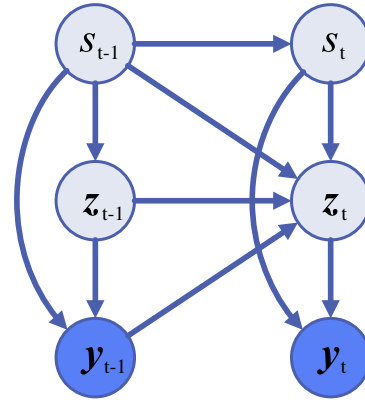
$$p(s_t | s_{t-1}) = \Phi(s_t, s_{t-1}) \quad (1)$$


Fig. 7. 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 \mathbf{y} is the observation, while s is a discrete random variable representing the subspace and \mathbf{z} is a continuous random variable representing the coordinates of \mathbf{y} on the subspace.

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

$$p(s_1) = v_s(s_1) \quad (2)$$

where $v_s(s_1)$ is the initial distribution.

The dynamics between \mathbf{z}_t and \mathbf{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 \mathbf{z}_t according to

$$\mathbf{z}_t = \mathbf{A}_{s_t} \mathbf{z}_{t-1} + \mathbf{Q}_{s_t} \quad (3)$$

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

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

$$\mathbf{z}_t = \Gamma_{s_t} \mathbf{y}_{t-1} + \Lambda \quad (4)$$

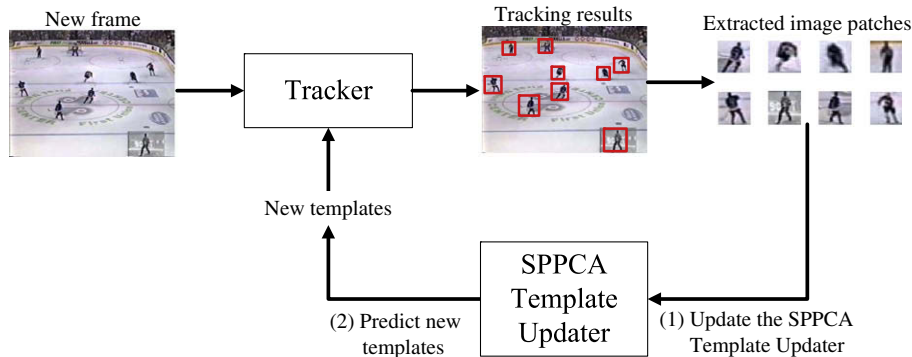


Fig. 6. The SPPCA template updater.

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

$$p(\mathbf{z}_1) = \mathcal{N}(\mathbf{z}_0, \Sigma_0) \quad (5)$$

where \mathbf{z}_0 and Σ_0 is the mean and covariance of the initial Gaussian distribution, respectively.

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

$$\mathbf{y}_t = \mathbf{C}_{s_t} \mathbf{z}_t + \mu_{s_t} + \mathbf{R}_{s_t} \quad (6)$$

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

4.2. Learning

The expectation–maximization (EM) algorithm [43] can be used to learn the maximum-likelihood parameters $\hat{\Omega} = \{\hat{\Phi}, \hat{v}_s, \hat{\mathbf{A}}, \hat{\mathbf{C}}, \hat{\Gamma}, \hat{\mathbf{V}}, \hat{\mathbf{W}}, \hat{\mu}, \hat{\mathbf{z}}_0, \hat{\Sigma}_0\}$ by maximizing the likelihood $p(\mathbf{y}_{1:T} | \Omega)$

$$\hat{\Omega} = \arg \max_{\Omega} p(\mathbf{y}_{1:T} | \Omega) \quad (7)$$

where $\mathbf{y}_{1:T}$ 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 $\mathbf{z}_{1:T}$ given the observation $\mathbf{y}_{1:T}$ and the parameters Ω^i in the i th iteration, i.e.,

$$f^i(s_{1:T}, \mathbf{z}_{1:T}) = p(s_{1:T}, \mathbf{z}_{1:T} | \mathbf{y}_{1:T}, \Omega^i) \quad (8)$$

- M-step: The M-step maximizes the expected log likelihood with respect to the parameters Ω , i.e.,

$$\Omega^{i+1} = \arg \max_{\Omega} (\log p(s_{1:T}, \mathbf{z}_{1:T}, \mathbf{y}_{1:T} | \Omega^i))_{f^i(s_{1:T}, \mathbf{z}_{1:T})} \quad (9)$$

$\langle \cdot \rangle_p$ 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.

4.2.1. Initialization

To initialize the parameters for the EM algorithm, we apply k -means clustering [44] to partition the training data $\mathbf{y}_{1:T}$ into n_s groups. For each group, we employ PPCA as described in Section 4.2.3 to estimate the initial value of \mathbf{C}_i , μ_i , Γ_i , and \mathbf{R}_i for $i = 1$ to n_s . \mathbf{A} , \mathbf{V} , and Σ_0 are initialized as identical matrices, and μ_0 is initialized as a zero vector. The transition matrix Φ and the initial distribution $v_s(s_0)$ are initialized by a uniform distribution.

4.2.2. E-step

Exact inference in SPPCA is intractable. Therefore, people seek an approximation algorithm to tackle this problem [46,47,45]. In this article, we use Viterbi Inference (VI) [46] because it is simple and fast, and it usually has acceptable quality of estimation [48].

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

$$\begin{aligned} p(s_{1:T}, \mathbf{z}_{1:T} | \mathbf{y}_{1:T}, \Omega^i) &= p(\mathbf{z}_{1:T} | s_{1:T}, \mathbf{y}_{1:T}, \Omega^i) p(s_{1:T} | \mathbf{y}_{1:T}, \Omega^i) \\ &\approx p(\mathbf{z}_{1:T} | s_{1:T}, \mathbf{y}_{1:T}, \Omega^i) \delta(\hat{s}_{1:T}^i) \end{aligned} \quad (10)$$

where $\delta(\cdot)$ is the Dirac-delta function. The pseudo-optimal sequence $\hat{s}_{1:T}^i$ can be computed exactly and efficiently using the Viterbi algorithm [46]. Algorithm 1 summarizes the procedures of the E-step.

Algorithm 1. E-step using the Viterbi Inference

Input $\mathbf{y}_{1:T}, \Omega$
Output $\hat{s}_{1:T}, \bar{\mathbf{z}}_{1:T}, \bar{\Sigma}_{1:T}$

- 1: **for** $i = 1$ to n_s **do**
- 2: $[\bar{\mathbf{z}}_1^i, \bar{\Sigma}_1^i, L_1^i] = \text{KalmanInitialize}(\mathbf{y}_1, \mathbf{z}_0(i), \Sigma_0(i), \Theta_i)$
- 3: $J(1, i) = L_1^i + \log v_s(i)$
- 4: **end for**
- 5:
- 6: **for** $t = 2$ to T **do**
- 7: **for** $i = 1$ to n_s **do**
- 8: **for** $j = 1$ to n_s **do**
- 9: **if** $i = j$
- 10: $[\bar{\mathbf{z}}_t^{ij}, \bar{\Sigma}_t^{ij}, L_t^{ij}] = \text{KalmanUpdate}(\mathbf{y}_t, \bar{\mathbf{z}}_{t-1}^i, \bar{\Sigma}_{t-1}^i, \Theta_j)$
- 11: **else**
- 12: $\bar{\mathbf{z}}_t^i = \Gamma_j(\mathbf{y}_{t-1} - \mu_j)$
- 13: $[\bar{\mathbf{z}}_t^{ij}, \bar{\Sigma}_t^{ij}, L_t^{ij}] = \text{KalmanInitialize}(\mathbf{y}_t, \bar{\mathbf{z}}_t^i, \mathbf{I}, \Theta_j)$
- 14: **end if**
- 15: **if** $J(t, j) < J(t-1, i) + L_t^{ij} + \log \Phi(i, j)$ **then**
- 16: $J(t, j) = J(t-1, i) + L_t^{ij} + \log \Phi(i, j)$
- 17: $B(t, j) = i$
- 18: **set** $[\bar{\mathbf{z}}_t^i, \bar{\Sigma}_t^i] = [\bar{\mathbf{z}}_t^{ij}, \bar{\Sigma}_t^{ij}]$
- 19: **end if**
- 20: **end for**
- 21: **end for**
- 22: **end for**
- 23: $\hat{s}_T = \arg \max_{j=1..n_s} J(T, j)$
- 24: **for** $t = T-1$ to 1 **do**
- 25: $\hat{s}_t = \arg \max J(t+1, \hat{s}_{t+1})$
- 26: **end for**
- 27:
- 28: $[\bar{\mathbf{z}}_{1:T}, \bar{\Sigma}_{1:T}] = \text{KalmanSmoothing}(\mathbf{y}_{1:T}, \hat{s}_{1:T}, \mathbf{z}_0, \Sigma_0, \Theta)$

4.2.3. 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 Φ , the system matrix and covariance $\{\mathbf{A}, \mathbf{V}\}$, the observation matrix, mean, and covariance $\{\mathbf{C}, \mu, \mathbf{W}\}$, and the initial distribution $v_s(s_0)$, \mathbf{z}_0 and Σ_0 . In this article, we assume that the current template is normally distributed around the previous template with a fixed covariance matrix, i.e., $\mathbf{A}_i = \mathbf{I}$ for $i = 1$ to n_s and $\mathbf{V}_i = \mathbf{I}$ 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, $\{\Phi, v\}$ can be estimated by counting the frequency; and $\{\mathbf{z}_0, \Sigma_0\}$ can be estimated by fitting a Gaussian distribution; $\{\mathbf{C}, \Gamma, \mu, \mathbf{W}\}$ can be estimated by PPCA [49].

Algorithm 2. M-step

Input: $\mathbf{y}_{1:T}, \hat{s}_{1:T}, \bar{\mathbf{z}}_{1:T}, \bar{\Sigma}_{1:T}$
Output: $\Phi, v, \mathbf{z}_0, \Sigma_0, \mathbf{C}, \Gamma, \mu, \mathbf{W}$

estimate Φ and v_s by counting the frequencies of $\hat{s}_{1:T}$ [47]

for $i = 1$ to n_s **do**

 estimate \mathbf{z}_0 and Σ_0 using the technique in [47]

$$\tilde{\pi}_i = (1/T) \sum_{t=1}^T \hat{s}_t^i, \quad \boldsymbol{\mu}_i = (\sum_{t=1}^T \hat{s}_t^i \mathbf{y}_t) / (\sum_{t=1}^T \hat{s}_t^i)$$

$$\mathbf{S}_i = (1/(\tilde{\pi}_i T)) \sum_{t=1}^T \hat{s}_t^i (\mathbf{y}_t - \boldsymbol{\mu}_i)(\mathbf{y}_t - \boldsymbol{\mu}_i)^T$$

$$d = n_y, \quad q = n_x$$

$$\{\lambda_1, \dots, \lambda_d\} = \text{EigenValue}(\mathbf{S}_i), \quad \{\mathbf{u}_1, \dots, \mathbf{u}_d\} = \text{Eigenvector}(\mathbf{S}_i)$$

$$\mathbf{U}_q = [\mathbf{u}_1, \dots, \mathbf{u}_q], \quad \boldsymbol{\Lambda}_q = \text{diag}([\lambda_1, \dots, \lambda_q])$$

$$\sigma_i^2 = \frac{1}{d-q} \sum_{j=q+1}^d \lambda_j$$

$$\mathbf{C}_i = \mathbf{U}_q (\boldsymbol{\Lambda}_q - \sigma_i^2 \mathbf{I}_q)$$

$$\mathbf{W}_i = \sigma_i^2 \mathbf{I}_d$$

$$\mathbf{M}_i = \mathbf{C}_i \mathbf{C}_i + \sigma_i^2 \mathbf{I}_q$$

$$\boldsymbol{\Gamma}_i = \mathbf{M}_i^{-1} \mathbf{C}_i^T$$

end for

4.3. Updating

The updating operation is employed to update the template model after observing the new appearance of players. In this article, we utilize the Rao-Blackwellized Particle Filter (RBPF) [45,50] to efficiently update the hidden states of the SPPCA model.

The key idea of the RBPF is that it factorizes the joint posterior distribution over $\{s_t, \mathbf{z}_t\}$ into two parts

$$p(s_t, \mathbf{z}_t | s_{1:t-1}, \mathbf{z}_{1:t-1}, \mathbf{y}_{1:t}) = p(\mathbf{z}_t | \mathbf{z}_{1:t-1}, s_{1:t}, \mathbf{y}_{1:t}) p(s_t | s_{1:t-1}, \mathbf{y}_{1:t}) \quad (11)$$

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

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.,

$$p(s_t | s_{1:t-1}, \mathbf{y}_{1:t}) \approx \sum_{i=1}^N w_t^{(i)} \delta_{s_t^{(i)}}(s_t) \quad (12)$$

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

$$p(\mathbf{z}_t | \mathbf{z}_{1:t-1}, s_{1:t}, \mathbf{y}_{1:t}) \approx \sum_{i=1}^N w_t^{(i)} p(\mathbf{z}_t | \mathbf{z}_{1:t-1}, s_{1:t}^{(i)}, \mathbf{y}_{1:t}) \quad (13)$$

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

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.,

$$s_t^{(i)} \sim p(s_t^{(i)} | s_{t-1}^{(i)}) \quad \text{for } i = 1, \dots, N \quad (14)$$

Secondly, the weights of particles can be updated according to

$$w_t^{(i)} = p(\mathbf{y}_t | s_t^{(i)}) \quad \text{for } i = 1, \dots, N \quad (15)$$

where $p(\mathbf{y}_t | s_t^{(i)})$ can be computed analytically by the Kalman filter [1]. Thirdly, we normalize the weights to make them sum to one

$$\tilde{w}_t^{(i)} = \frac{w_t^{(i)}}{\sum_{i=1}^N w_t^{(i)}} \quad \text{for } i = 1, \dots, N \quad (16)$$

where $\tilde{w}_t^{(i)}$ is the weight of the i th particle after normalization. Then, we resample 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 [1] to approximate the posterior distribution in Eq. (13). Algorithm 3 summarizes the procedures of the Rao-Blackwellized Particle Filter in the SPPCA model.

¹ In our implementation, we call Kevin Murphy's Kalman Filter Toolbox [51] to update the posterior distribution over \mathbf{z}_t .

Algorithm 3. Rao-Blackwellized Particle Filter for SPPCA

Input $\mathbf{y}_{1:T}, \boldsymbol{\Omega}$

Output $\mathbf{s}_{1:T}, \bar{\mathbf{z}}_{1:T}, \bar{\boldsymbol{\Sigma}}_{1:T}$

```

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

```

4.4. 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 evaluating the following probability

$$p(\mathbf{y}_{t+1} | s_{t+1}^{(i)}, \mathbf{z}_{t+1}^{(i)}, \tilde{s}_{t+1}^{(i)}) \quad \text{for } i = 1, \dots, N \quad (17)$$

Note that Eq. (17) can be computed efficiently by Kalman prediction [1]. Let $\tilde{\mathbf{y}}_{t+1}^{(i)}$ be the mean of Eq. (17). The new template $\tilde{\mathbf{y}}_{t+1}$ can be computed by averaging out all $\tilde{\mathbf{y}}_{t+1}^{(i)}$

$$\tilde{\mathbf{y}}_{t+1} = \frac{1}{N} \sum_{i=1}^N \tilde{\mathbf{y}}_{t+1}^{(i)} \quad (18)$$

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

5. Action recognition

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 30–70 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 ac-

tions/activities of the hockey players. The action recognizer first transforms image patches to the HOG descriptors [6] 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 information, the frame-to-frame similarity matrix is then convolved with a weighting function similar to [7] to produce the motion-to-motion similarity matrix. Finally, a Sparse Multinomial Logistic Regression (SMLR) classifier [2] is exploited and it takes the motion similarity matrix as input to determine the player's actions. Fig. 8 illustrates the procedures of the action recognizer.

5.1. The sparse multinomial logistic regression classifier

In this article, we use Sparse Multinomial Logistic Regression (SMLR) [2] 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) [2], and has been used in the field of bio-informatics [52,53].

We 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 [2]. (3) We can always find the unique global optimal weights for the basis functions because the log-posterior is a concave function.

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

$$p(a_i = 1 | \mathbf{y}, \mathbf{w}) = \frac{\exp(\mathbf{w}_i^T \mathbf{g}(\mathbf{y}))}{\sum_{j=1}^m \exp(\mathbf{w}_j^T \mathbf{g}(\mathbf{y}))} \quad (19)$$

for $i \in \{1, \dots, m\}$, where \mathbf{w}_i is the weight vector corresponding to class i . Note that $\mathbf{g}(\mathbf{y})$ 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 \mathbf{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 $\{\mathbf{y}_j, \mathbf{a}_j\}_{j=1}^n$

$$\hat{\mathbf{w}} = \arg \max_{\mathbf{w}} \sum_{j=1}^n \log p(\mathbf{w} | \mathbf{y}_j, \mathbf{a}_j) = \arg \max_{\mathbf{w}} \sum_{j=1}^n \log p(\mathbf{a}_j | \mathbf{y}_j, \mathbf{w}) p(\mathbf{w}) \quad (20)$$

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

$$p(\mathbf{w}) \propto \exp(-\lambda \|\mathbf{w}\|_1) \quad (21)$$

where $\|\mathbf{w}\|_1$ 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. (20) is still a concave function and therefore all local optimal estimates $\hat{\mathbf{w}}$ are also a global optimum.

There are many algorithms that optimize the objective function Eq. (20), e.g. [2,52,53]. We adopt the component-wise updating procedure introduced by Krishnapuram et al. [2] because it converges faster than others in practice.

5.2. Motion similarity measure

The most naïve way to measure motion similarity is to sum up the frame-to-frame similarity for a fixed number of frames. Let $\mathbf{y}_{1:N}^*$ be the training sequence, and $\mathbf{y}_{1:N}$ be the testing sequence. Then the naïve motion similarity is defined as:

$$\tilde{d}_M(\mathbf{y}_{1:N}, \mathbf{y}_{1:N}^*) = d_F(\mathbf{y}_1, \mathbf{y}_1^*) + \dots + d_F(\mathbf{y}_N, \mathbf{y}_N^*) = \sum_{i=1}^N d_F(\mathbf{y}_i, \mathbf{y}_i^*) \quad (22)$$

where $\tilde{d}_M(\cdot)$ is the naïve motion similarity and $d_F(\cdot)$ is the frame-to-frame similarity. However, this motion similarity has a strong assumption that the testing and training actions have the same speed, and thus fails to handle actions of various speeds.

Efros et al. [7] presented a robust motion similarity measure that can handle actions of different speed. Their measure is also a weighted sum of frame-to-frame similarity. However, their weighting matrix has diffused tails and thus off-diagonal frame-to-frame similarity also have some weights (see [7] for more details). In this article, we modify their robust motion similarity mea-

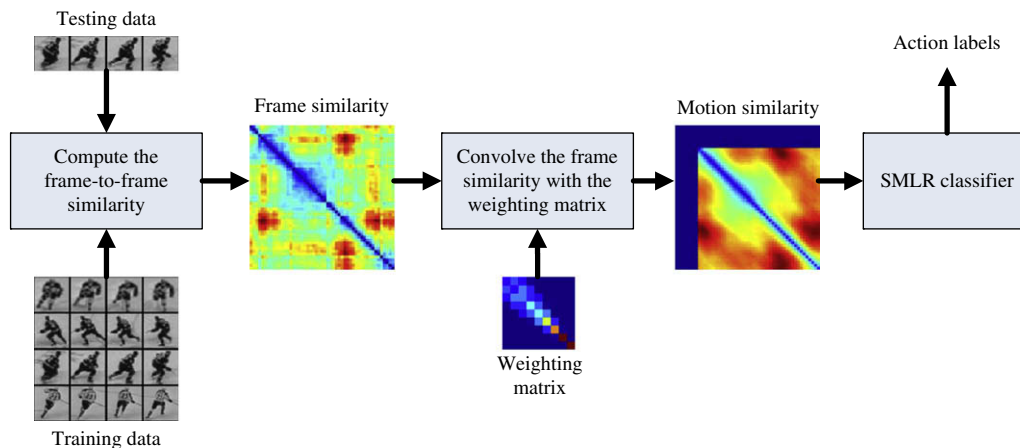


Fig. 8. The action recognizer: This figure illustrates the procedures of the action recognizer. The action recognizer first computes the frame-to-frame similarity between the training and testing data. Then, the frame-to-frame similarity matrix is convolved with a weighting matrix to produce the motion-to-motion similarity matrix. Finally, the SMLR classifier takes the frame similarity matrix as input feature to determine the player's actions.

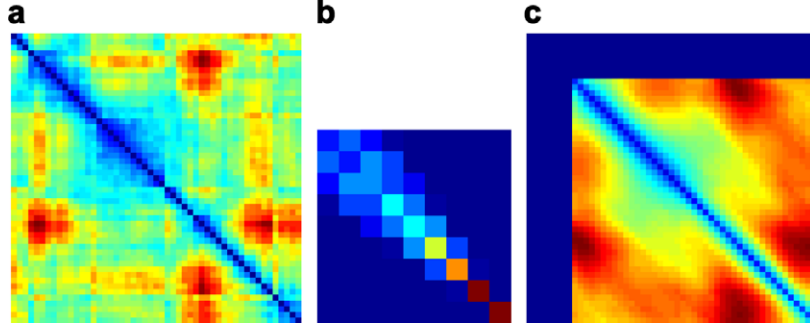


Fig. 9. Similarity matrix: (a) The frame-to-frame similarity matrix. (b) The weighting matrix. (c) The motion-to-motion similarity matrix computed by convolving (a) with (b).

sure by only considering previous frames. As a result, the weighting matrix only has one diffused tail instead of two. Fig. 9 illustrates the computation of the motion similarity measure. Fig. 9(a) shows the frame-to-frame similarity of testing sequence $\mathbf{y}_{1:N}$ and training sequence $\mathbf{y}_{1:N}^*$. Fig. 9(b) is the new weighting matrix that only consider previous frames. Fig. 9(c) is the motion similarity computed by convolving (a) with (b).

Since the SMLR classifier does not have restrictions on the choice of basis functions, we use a distance function that measures the motion similarity between the testing data and all training examples as the basis functions. Since we collect training examples of various action styles, this basis function can handle actions of different styles. This basis function can also handle actions of different speed because of the use of the robust motion similarity.

6. Multi-target tracking

We incorporate the template updater described in Section 4 and the action recognizer described in Section 5 with a multi-target tracking system, the boosted particle filter (BPF). Fig. 2 shows the system diagram of the tracking and action recognition system.

6.1. Statistical model

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

Letting $\mathbf{y}_{1:t} \triangleq \{\mathbf{y}_1 \dots \mathbf{y}_t\}$ be the observation vectors up to time t , our goal is to estimate $p(\mathbf{x}_t | \mathbf{y}_{1:t})$, the probability of the current state \mathbf{x}_t given $\mathbf{y}_{1:t}$, which can be solved by the following Bayesian recursion [26]:

$$\begin{aligned} p(\mathbf{x}_t | \mathbf{y}_{1:t}) &= \frac{p(\mathbf{y}_t | \mathbf{x}_t) p(\mathbf{x}_t | \mathbf{y}_{1:t-1})}{p(\mathbf{y}_t | \mathbf{y}_{1:t-1})} \\ &= \frac{p(\mathbf{y}_t | \mathbf{x}_t) \int p(\mathbf{x}_t | \mathbf{x}_{t-1}) p(\mathbf{x}_{t-1} | \mathbf{y}_{1:t-1}) d\mathbf{x}_{t-1}}{\int p(\mathbf{y}_t | \mathbf{x}_t) p(\mathbf{x}_t | \mathbf{y}_{1:t-1}) d\mathbf{x}_t} \end{aligned} \quad (23)$$

In our tracking system, this transition distribution $p(\mathbf{x}_t | \mathbf{x}_{t-1})$ 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 $p(\mathbf{y}_t | \mathbf{x}_t)$ is defined in the following section.

6.2. Observation likelihood

As already described in Section 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

$$p(\mathbf{y}_t | \mathbf{x}_t) \propto p_{\text{hsv}}(\mathbf{y}_t | \mathbf{x}_t) p_{\text{hog}}(\mathbf{y}_t | \mathbf{x}_t) \quad (24)$$

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:

$$p_{\text{hsv}}(\mathbf{y}_t | \mathbf{x}_t) \propto e^{-\lambda_c \xi[\mathbf{K}^*, \mathbf{K}(\mathbf{x}_t)]} \quad (25)$$

where $\mathbf{K}(\mathbf{x}_t)$ is the HSV color histogram computed at \mathbf{x}_t , \mathbf{K}^* is the template for the HSV color histogram, and $\xi(\cdot, \cdot)$ is the diffusion distance [29]. 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:

$$p_{\text{hog}}(\mathbf{y}_t | \mathbf{x}_t) \propto e^{-\lambda_s \xi[\mathbf{H}^*, \mathbf{H}(\mathbf{x}_t)]} \quad (26)$$

where $\mathbf{H}(\mathbf{x}_t)$ is the HOG descriptor computed at \mathbf{x}_t , \mathbf{H}^* is the template for the HOG descriptor, and $\xi(\cdot, \cdot)$ is the diffusion distance [29]. We fix the scaling constant $\lambda_s = 10$ throughout our experiments.

6.3. Particle filtering

Since the observation likelihood Eq. (25) is nonlinear and non-Gaussian, there is no analytical solution for the Bayesian recursion Eq. (23). Instead, we seek an approximation solution, using particle filtering [26].

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

$$\mathbf{x}_t^{(i)} \sim q(\mathbf{x}_t^{(i)} | \mathbf{x}_{t-1}^{(i)}, \mathbf{y}_{1:t}) \quad \text{for } i = 1, \dots, N \quad (27)$$

In the simplest scenario, it is set as $q(\mathbf{x}_t^{(i)} | \mathbf{x}_{t-1}^{(i)}, \mathbf{y}_{1:t}) = p(\mathbf{x}_t^{(i)} | \mathbf{x}_{t-1}^{(i)})$, yielding the bootstrap filter [26]. 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:

$$w_t^{(i)} = w_{t-1}^{(i)} \frac{p(\mathbf{y}_t | \mathbf{x}_t^{(i)}) p(\mathbf{x}_t^{(i)} | \mathbf{x}_{t-1}^{(i)})}{q(\mathbf{x}_t^{(i)} | \mathbf{x}_{t-1}^{(i)}, \mathbf{y}_{1:t})} \quad (28)$$

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

$$p(\mathbf{x}_t | \mathbf{y}_{1:t}) \approx \sum_{i=1}^N w_t^{(i)} \delta_{\mathbf{x}_t^{(i)}}(\mathbf{x}_t) \quad (29)$$

6.4. 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 [28,54]. In this article, we use the boosted particle filter [5] that incorporates the current detections of hockey players to produce a better proposal distribution $q(\mathbf{x}_t^{(i)} | \mathbf{x}_{1:t-1}, \mathbf{y}_{1:t})$.

6.4.1. Adaboost detection

In order to detect hockey player in the current frame, we adopt the cascaded Adaboost algorithm of Viola and Jones [9], which was originally developed for detecting faces. In our experiments, a 23 layer cascaded classifier is trained to detect 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 Fig. 13. Unlike the detector used in [5], 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 [5]. 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 Fig. 10. 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.

6.4.2. 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.

$$q_{BPF}^*(\mathbf{x}_t^{(i)} | \mathbf{x}_{1:t-1}, \mathbf{y}_{1:t}) = \alpha_{ada} q_{ada}(\mathbf{x}_t^{(i)} | \mathbf{y}_t) + (1 - \alpha_{ada}) p(\mathbf{x}_t^{(i)} | \mathbf{x}_{t-1}^{(i)}) \quad (30)$$

where q_{ada} is a Gaussian distribution centered in the Adaboost detection with a fixed variance (See Fig. 11). 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. We set $\alpha_{ada} = 1$ for implementing a boosted particle filter throughout our experiments.

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 [30] 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. Therefore, we use a mode-seeking proposal as an alternative whenever mixture proposals with the detection results are not available (i.e., no detections nearby a target).

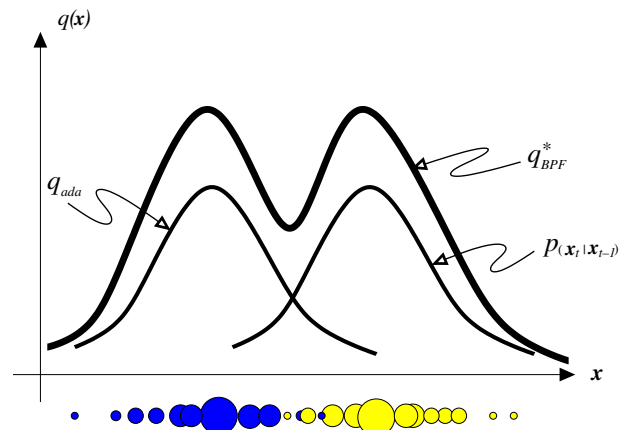


Fig. 11. Mixture of Gaussians for the proposal distribution.

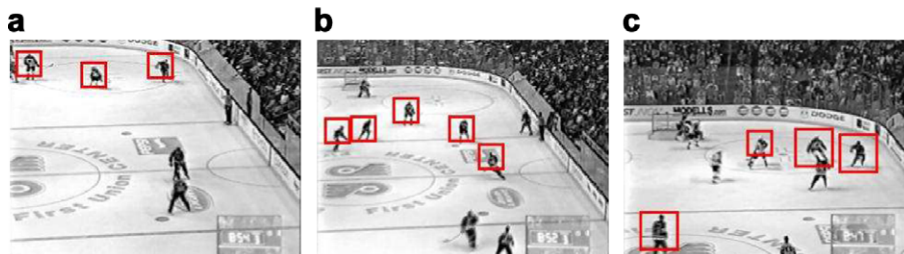


Fig. 10. Hockey player detection results: This figure shows results of the Adaboost hockey detector. (a), (b), and (c) show mostly accurate detections. Please note that there are players who are not detected and some of the boxes do not cover the entire figure of the player (i.e., a box is too small to cover a lower part of the body).

6.5. 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.

Algorithm 4. Boosted Particle Filter

Input: $\{I_t\}_{t=1}^T$
Output: $\{\mathbf{x}_{m,1:T}\}_{m=1,\dots,M}$

- 1: $M = 0$
- 2: **for** $t = 1$ to T
- 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 $\{\mathbf{x}_{m,t}^{(i)}\}_{i=1}^N$ by sampling from a Gaussian distribution centered on the Adaboost detection
- 7: Extract the image patch $\mathbf{y}_{m,t}$ from the Adaboost detection
- 8: Initialize the SPPCA template updater U_m using $\mathbf{y}_{m,t}$
- 9: **end for**
- 10: $M = M + M_{new}$
- 11: **end if**
- 12:
- 13: **for** $m = 1$ to M **do**
- 14: Generate a new template $\tilde{\mathbf{y}}_{m,t}$ by Eq. (17) from the SPPCA template updater U_m
- 15:
- 16: Propose new particles $\{\mathbf{x}_{m,t}^{(i)}\}_{i=1}^N$ by Eq. (30)
- 17: Compute the observation likelihood for $\{\mathbf{x}_{m,t}^{(i)}\}_{i=1}^N$ by Eq. (24)
- 18: Update the importance weights $\{w_{m,t}^{(i)}\}_{i=1}^N$ by Eq. (28)
- 19: Generate unweighted samples $\{\tilde{\mathbf{x}}_{m,t}^{(i)}\}_{i=1}^N$ by resampling $\{\mathbf{x}_{m,t}^{(i)}\}_{i=1}^N$ according to the importance weights $\{w_{m,t}^{(i)}\}_{i=1}^N$
- 20:
- 21: $\bar{\mathbf{x}}_{m,t} = \text{mean}(\{\tilde{\mathbf{x}}_{m,t}^{(i)}\}_{i=1}^N)$
- 22: Extract image patch $\tilde{\mathbf{y}}_{m,t}$ centered in $\bar{\mathbf{x}}_{m,t}$
- 23:
- 24: Update $\{\mathbf{z}_{m,t}, s_{m,t}\}$ of the SPPCA template updater U_m by $\tilde{\mathbf{y}}_{m,t}$ using RBPF described in Section 4.3
- 25: Recognize the action of the player $a_{m,t}$ by $\tilde{\mathbf{y}}_{m,t}$ 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**

Briefly the targets are detected and initialized by using the cascaded Adaboost detector described in Section 6.4.1. During the tracking at time $t + 1$, we use the SPPCA template updater described in Section 4 to predict the new template $\tilde{\mathbf{y}}_{t+1}$ 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 \mathbf{x}_{t+1} . To update the posterior distribution over $\{s_{t+1}, \mathbf{z}_{t+1}\}$, we compute the mean $\bar{\mathbf{x}}_{t+1}$ of the posterior distribution $p(\mathbf{x}_{t+1} | \mathbf{y}_{t+1})$, and extract the image patch $\tilde{\mathbf{y}}_{t+1}$ located in $\bar{\mathbf{x}}_{t+1}$. The image patch $\tilde{\mathbf{y}}_{t+1}$ is then fed into the SPPCA template updater to update the posterior over $\{s_{t+1}, \mathbf{z}_{t+1}\}$. Similarly, we give the image patch $\tilde{\mathbf{y}}_{t+1}$ to the action recognizer described in Section 5. The action recognizer will classify the action of the targets at time $t + 1$ and return the probability $p(a_{t+1} | \tilde{\mathbf{y}}_{t+1}, \mathbf{w})$.

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.

7. Experiments

7.1. Template updating

In this experiment, we evaluate the prediction accuracy of the SPPCA template updater. We have a collection of 5609 training images as shown in Fig. 13. 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 $\tilde{\mathbf{y}}_{t+1}$, and compare the prediction with the ground-truth \mathbf{y}_{t+1} using the relative sum-of-squares error

$$\epsilon_{t+1} = \frac{\sum_{i=1}^{n_y} (\tilde{y}_{t+1,i} - y_{t+1,i})^2}{\sum_{i=1}^{n_y} y_{t+1,i}^2} \quad (31)$$

where $\tilde{y}_{t+1,i}$ and $y_{t+1,i}$ is the i th feature of vector $\tilde{\mathbf{y}}_{t+1}$ and \mathbf{y}_{t+1} , respectively. Then, we feed the new observation \mathbf{y}_{t+1} back to the SPPCA template updater and call the *updating* operation to update the joint posterior distribution over $\{s_{t+1}, \mathbf{z}_{t+1}\}$ as described in Section 4.4. Experimental results are shown in Fig. 12, 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 Fig. 12. The HMM template updater we use is similar to the exemplar tracker [18,37,38] 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 \mathbf{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 [55] 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 when we increase the 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 better 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.

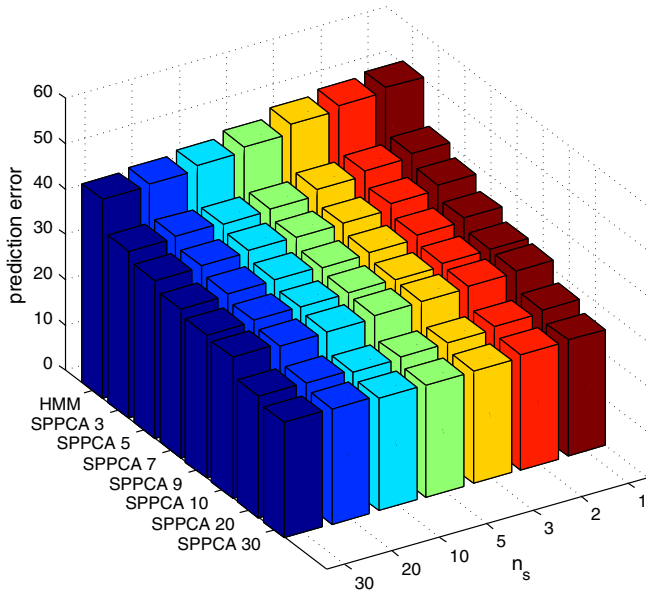


Fig. 12. Experimental results of the SPPCA template updater: This figure shows the prediction error of the SPPCA template updater and the HMM template updater with different n_s and n_t . The error is measured by averaging out the criterion in Eq. (31). The HMM template updater is abbreviated to **HMM**, and the SPPCA template updater with $n_s = k$ is abbreviated to **SPPCA k**. 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.

7.2. Action recognition

This experiment compares the performance between the proposed action recognizer that uses the SMLR classifier with the HOG descriptors (SMLR + HOG), and the action recognizers presented by Efros et al. [7] (5-NN + DOF) and Wu [35] (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).

7.2.1. 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. Fig. 13 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 article.

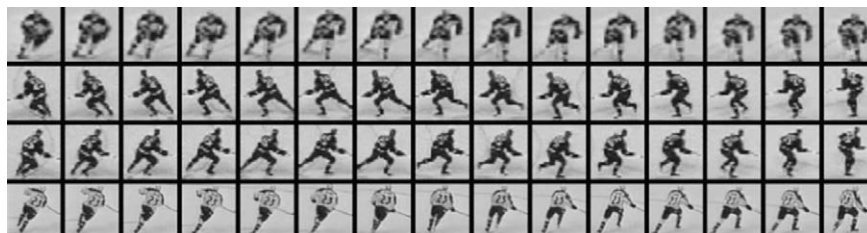


Fig. 13. Training data for the action recognizer: This figure shows a part of our hand annotated training data. A total of 4295 different figures of hockey players are used as training images for the action recognizer. First row: players skating down. Second row: players skating left. Third row: players skating right. Fourth row: players skating up.

7.2.2. Parameter settings

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

7.2.2.1. HOG descriptors. 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. The frame-to-frame distance between a pair of HOG descriptors is χ^2 distance, i.e., $k(\mathbf{y}_i, \mathbf{y}_j) = \chi^2(\mathbf{y}_i, \mathbf{y}_j)$ where $\chi^2(\cdot, \cdot)$ is the χ^2 distance between input features \mathbf{y}_i and \mathbf{y}_j .

7.2.2.2. DOF descriptors. The Decomposed Optical Flow (DOF) descriptor [7] is constructed by using the flow images computed by the Lucas–Kanade algorithm [56] and smoothed by a 3×3 Gaussian low-pass filter. Then, the flow images are decomposed into four channels (X^+ , X^- , Y^+ , Y^-) to form sparse descriptors of dimensionality 4096. The frame-to-frame distance between a pair of DOF descriptors is the scalar product, i.e., $k(\mathbf{y}_i, \mathbf{y}_j) = \mathbf{y}_i^T \mathbf{y}_j$.

7.2.2.3. DIG descriptors. The Decomposed Image Gradients (DIG) descriptor [35] 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 (X^+ , X^- , Y^+ , Y^-) to form sparse descriptors of dimensionality 4096. The frame-to-frame distance between a pair of DIG descriptors is the scalar product, i.e., $k(\mathbf{y}_i, \mathbf{y}_j) = \mathbf{y}_i^T \mathbf{y}_j$.

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 similarity described in Section 5.2. Then, the motion-to-motion similarity vector is used as a feature and fed into a 5-nearest-neighbors classifier (5-NN) and a SMLR classifier described in Section 5.1 with the regularization constant $\lambda = 1$.

7.2.3. Results

Table 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 T_1 time), and the average time of classifying a single image patch given the descriptor (the T_2 time). The average total time of classifying an image patch (the $T_1 + T_2$ 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 descrip-

Table 1
Action recognition results

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

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.

tors. 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 Table 1, we can observe that SMLR + HOG is at least 10 times faster than others. The T1 column in Table 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 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.

Fig. 14 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. 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 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].

7.3. Multi-target tracking

Evaluating the performance of a multi-target tracking system is a difficult task. In this section, we explain our evaluation criteria

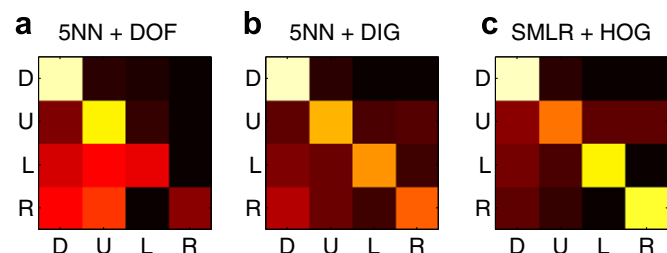


Fig. 14. Confusion matrix for the action recognition results: (a) 5-NN + DOF: action recognition results of using the 5-nearest-neighbor classifier with the DOF descriptors. The main diagonal is: [0.91, 0.73, 0.33, 0.19]. (b) 5-NN + DIG: action recognition results of using the 5-nearest-neighbor classifier 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].

and present both qualitative and quantitative experimental results. Throughout the experiments, we apply the same parameter settings for the HOG descriptor, HSV color histogram, and we also use the SPPCA template updater to update the HOG model as described in the previous sections. We use 30 particles for each target with the boosted particle filter.

7.3.1. Qualitative evaluation

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 Fig. 15(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 Fig. 15(b), the scale of the estimated boxes is better than Fig. 15(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 Fig. 15(c), we achieve a better scale and localization estimation of the targets.

Secondly, we show the experimental results of the entire system in Fig. 18. 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 [HOG + HSV, BPF] tracker). The action recognizer described in Section 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.

7.3.2. Quantitative evaluation

The previous section shows that a joint likelihood of the HSV color histogram and the HOG descriptors is better than using either one of them alone. We then evaluate our tracking system quantitatively in order to show the power of the boosted particle filter. The evaluation compares the position and size estimation of our system with the ground-truth.

Let (E_x^i, E_y^i, E_s^i) be the estimated (x, y) center position and size of a player i , respectively. Let (G_x^i, G_y^i, G_s^i) be the ground-truth center (x, y) position and size of a player i , respectively. We compute the normalized distance between the center positions of the estimation and ground-truth, i.e.,

$$Error_{center}(i) = \min_j \frac{\sqrt{(E_x^i - G_x^j)^2 + (E_y^i - G_y^j)^2}}{G_s^j} \quad (32)$$

where $Error_{center}(i)$ represents the minimum distance between the center of a player i and the ground-truth center. However, this measure does not take into account the errors made by an incorrect estimation of the size of the player. Thus, we also compute the distance between an estimated foot position (i.e., the bottom center of the bounding box) of the target and the ground-truth, i.e.,

$$Error_{foot}(i) = \min_j \frac{\sqrt{(E_x^i - G_x^j)^2 + ((E_y^i + E_s^i)/2) - (G_y^j + G_s^j/2))^2}}{G_s^j} \quad (33)$$

Figs. 16 and 17 show the results of the average error per frame measured in both Eqs. (32) and (33). Here, the average error is computed by averaging over errors made by all players who are present in a single frame. We conduct our experiments with three different types of trackers: (1) [HOG, PF] tracker uses the HOG descriptor for the observation model and regular particle filters

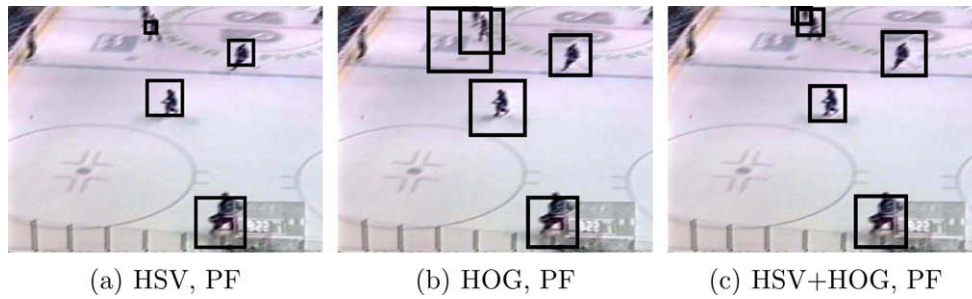


Fig. 15. Tracking results: This figure shows the comparison between [HSV, PF], [HOG, PF], and [HOG + HSV, PF]. In (a), there are a few boxes that have wrong scales. In (b), there is a box that is already shifted away from the object. However, all the targets are correctly tracked in (c).

for tracking. (2) [HOG, BPF] tracker uses the HOG descriptor for the observation model and uses boosted particle filters ($\alpha_{ada} = 1$) for tracking. (3) [HOG + HSV, BPF] tracker uses both the HOG descriptor and HSV color histogram for the observation model and boosted particle filters ($\alpha_{ada} = 1$) for tracking.

Figs. 16 and 17 clearly show that BPF generally increases the accuracy of the tracker over the entire sequence. Moreover, adding the color feature (HSV color histogram) also improves the accuracy, especially for images between frames No. 100 and No. 200.

Table 2 shows the average errors over all targets through the entire video sequence. We can also observe that [HOG, BPF] outperforms [HOG, PF], and the one using BPF with both HOG and color cues ([HOG + HSV, BPF]) performs the best.

Nevertheless, we acknowledge that it is still 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 [57].

8. Conclusion

We present a system that can automatically track multiple hockey players and simultaneously recognize their actions given a broadcast video sequence. There are three contributions. Firstly, we employ the grids of 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 our previous HMM template updater [37]. Secondly, we augment the boosted particle filter (BPF) with new observation model and SPPCA, and improve the robustness of the tracking system. Finally, 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 used to handle actions of different speed. Experimental results show that the action recognizer outperforms [7,35] in both accuracy and speed.

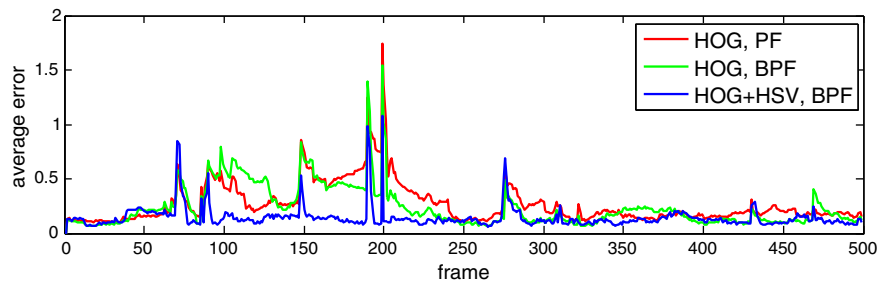


Fig. 16. Average center distance error ($Error_{center}$) per frame. The x -axis represents the frame number. The y -axis represents the average center distance ($Error_{center}$) over all targets in a single frame.

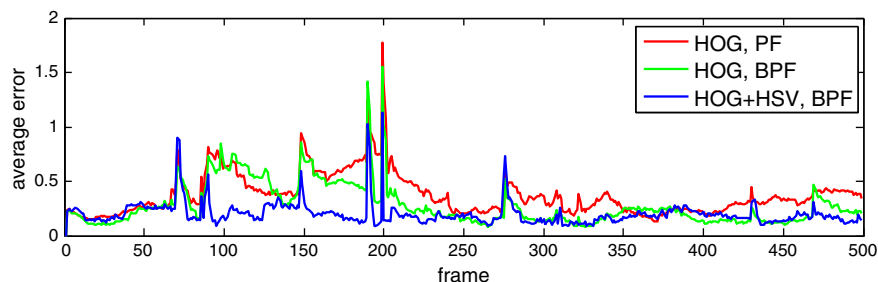


Fig. 17. Average foot distance error ($Error_{foot}$) per frame. The x -axis represents the frame number. The y -axis represents the average foot distance ($Error_{foot}$) over all targets in a single frame.

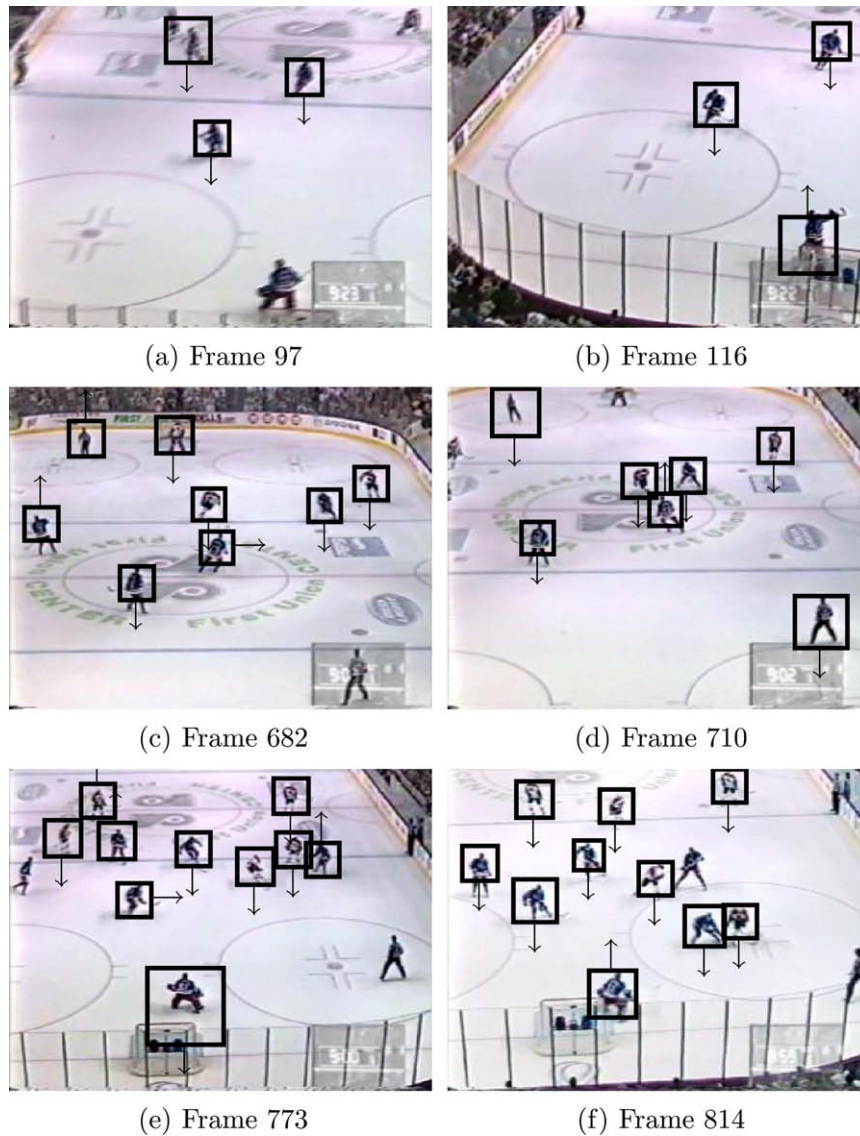


Fig. 18. Tracking and action recognition results: This figure shows the final result of our system. All figures have the size of 320×240 . The square box represents the tracking region and arrows indicate the recognized action of the player.

Table 2
Tracking accuracy

Tracker	$Error_{center}$	$Error_{foot}$
HOG, PF	0.2701	0.3748
HOG, BPF	0.2383	0.2801
HOG + HSV, BPF	0.1431	0.1998

This table shows the accuracy of three trackers in terms of average error over all targets through the entire sequence. The unit is the fraction of width of the ground-truth box.

References

- [1] G. Welch, G. Bishop, An introduction to the Kalman filter, Tech. Rep. TR 95-041, Department of Computer Science, University of North Carolina at Chapel Hill, 1995.
- [2] B. Krishnapuram, L. Carin, M.A. Figueiredo, A.J. Hartemink, Sparse multinomial logistic regression: fast algorithms and generalization bounds, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27 (6) (2005) 957–968.
- [3] D. Gavrilu, The visual analysis of human movement: a survey, *Computer Vision and Image Understanding* 73 (1) (1999) 82–98.
- [4] W. Hu, T. Tan, L. Wang, S. Maybank, A survey on visual surveillance of object motion and behaviors, *IEEE Transactions on Systems, Man and Cybernetics C: Applications and Reviews* 34 (3) (2004) 334–352.
- [5] K. Okuma, A. Taleghani, N. de Freitas, J.J. Little, D.G. Lowe, A boosted particle filter: multitarget detection and tracking, in: *European Conference on Computer Vision*, 2004, pp. 28–39.
- [6] N. Dalal, B. Triggs, Histograms of oriented gradients for human detection, in: *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, 2005, pp. 886–893.
- [7] A. Efros, C. Breg, G. Mori, J. Malik, Recognizing action at a distance, in: *International Conference on Computer Vision*, 2003, pp. 726–733.
- [8] M.S. Arulampalam, S. Maskell, N. Gordon, T. Clapp, A tutorial on particle filters for on-line non-linear/non-Gaussian Bayesian tracking, *IEEE Transactions on Signal Processing* 50 (2) (2002) 174–188.
- [9] P. Viola, M. Jones, Rapid object detection using a boosted cascade of simple features, in: *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, 2001, pp. 511–518.
- [10] D. Koller, J. Weber, J. Malik, Robust multiple car tracking with occlusion reasoning, in: *European Conference on Computer Vision*, 1994, pp. 186–196.
- [11] T. Misu, M. Naemura, W. Zheng, Y. Izumi, K. Fukui, Robust tracking of soccer players based on data fusion, in: *International Conference on Pattern Recognition*, 2002, pp. 556–561.
- [12] C. Needham, R. Boyle, Tracking multiple sports players through occlusion, congestion and scale, in: *British Machine Vision Conference*, 2001, pp. 93–102.
- [13] S. Intille, J. David, A. Bobick, Real-time closed-world tracking, in: *IEEE Conference on Computer Vision and Pattern Recognition*, 1997, pp. 697–703.

- [14] C. Hue, J.L. Cadre, P. Pérez, Tracking multiple objects with particle filtering, *IEEE Transactions on Aerospace and Electronic Systems* 38 (3) (2002) 791–812.
- [15] M. Isard, J. MacCormick, BraMBLE: A Bayesian multiple-blob tracker, in: *International Conference on Computer Vision*, vol. 2, 2001, pp. 34–41.
- [16] J. MacCormick, A. Blake, A probabilistic exclusion principle for tracking multiple objects, in: *International Conference on Computer Vision*, vol. 1, 1999, pp. 572–578.
- [17] L. Matthews, T. Ishikawa, S. Baker, The template update problem, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26 (6) (2004) 810–815.
- [18] K. Toyama, A. Blake, Probabilistic tracking with exemplars in a metric space, *International Journal of Computer Vision* 48 (1) (2002) 9–19.
- [19] M. Black, A. Jepson, EigenTracking: robust matching and tracking of articulated objects using a view-based representation, *International Journal of Computer Vision* 26 (1) (1998) 63–84.
- [20] Z. Khan, T. Balch, F. Dellaert, A Rao-Blackwellized particle filter for EigenTracking, in: *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, 2004, pp. 980–986.
- [21] M. Tipping, C. Bishop, Probabilistic principal component analysis, *Journal of Royal Statistical Society B* 61 (3) (1999) 611–622.
- [22] H. Lim, V.I. Morariu, O.I. Camps, M. Sznajder, Dynamic appearance modeling for human tracking, in: *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, 2006, pp. 751–757.
- [23] R. Urtasun, D. Fleet, P. Fua, 3D people tracking with Gaussian process dynamical models, in: *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, 2006, pp. 198–205.
- [24] K. Moon, V. Pavlović, Impact of dynamics on subspace embedding and tracking of sequences, in: *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, 2006, pp. 198–205.
- [25] P. Pérez, C. Hue, J. Vermaak, M. Gangnet, Color-based probabilistic tracking, in: *European Conference on Computer Vision*, 2002, pp. 661–675.
- [26] A. Doucet, N.D. Freitas, N. Gordon (Eds.), *Sequential Monte Carlo Methods in Practice*, Springer, 2005.
- [27] M. Isard, A. Blake, CONDENSATION – conditional density propagation for visual tracking, *International Journal of Computer Vision* 29 (1) (1998) 5–28.
- [28] Y. Rui, Y. Chen, Better proposal distributions: object tracking using unscented particle filter, in: *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, 2001, pp. 786–793.
- [29] H. Ling, K. Okada, Diffusion distance for histogram comparison, in: *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, 2006, pp. 246–253.
- [30] D. Comaniciu, P. Meer, Mean shift: a robust approach toward feature space analysis, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24 (5) (2002) 603–619.
- [31] J. Giebel, D. Gavrilu, C. Schnörr, A Bayesian framework for multi-cue 3D object tracking, in: *European Conference on Computer Vision*, 2004, pp. 241–252.
- [32] T. Cootes, C. Taylor, D. Cooper, J. Graham, Active shape models – their training and application, *Computer Vision and Image Understanding* 61 (1) (1995) 38–59.
- [33] W.T. Freeman, M. Roth, Orientation histograms for hand gesture recognition, in: *International Workshop on Automatic Face and Gesture Recognition*, 1995.
- [34] W. Freeman, K. Tanaka, J. Ohta, K. Kyuma, Computer vision for computer games, in: *International Conference on Automatic Face and Gesture Recognition*, 1996, pp. 100–105.
- [35] X. Wu, Templated-based action recognition: classifying hockey players' movement, Master's Thesis, The University of British Columbia, 2005.
- [36] J. Yamato, J. Ohya, K. Ishii, Recognizing human action in time-sequential images using hidden Markov model, in: *IEEE Conference on Computer Vision and Pattern Recognition*, 1992, pp. 379–385.
- [37] W.-L. Lu, J.J. Little, Tracking and recognizing actions at a distance, in: *ECCV Workshop on Computer Vision Based Analysis in Sport Environments*, 2006, pp. 49–60.
- [38] W.-L. Lu, J.J. Little, Simultaneous tracking and action recognition using the PCA-HOG descriptor, in: *The Third Canadian Conference on Computer and Robot Vision*, 2006.
- [39] C. Yang, R. Duraiswami, L. Davis, Fast multiple object tracking via a hierarchical particle filter, in: *International Conference on Computer Vision*, vol. 1, 2005, pp. 212–219.
- [40] Y. Wu, T. Huang, Robust visual tracking by integrating multiple cues based on co-inference learning, *International Journal of Computer Vision* 58 (1) (2004) 55–71.
- [41] S. Avidan, Ensemble tracking, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29 (2) (2007) 261–271.
- [42] D.G. Lowe, Distinctive image features from scale-invariant keypoints, *International Journal of Computer Vision* 60 (2) (2004) 91–110.
- [43] R.M. Neal, G. Hinton, A new view of the EM algorithm that justifies incremental, sparse, and other variants, in: M.I. Jordan (Ed.), *Learning in Graphical Models*, Kluwer Academic Publishers, 1998, pp. 355–368.
- [44] A. Jain, R. Dubes, *Algorithms for Clustering Data*, Prentice Hall, 1988.
- [45] A. Doucet, N. de Freitas, K. Murphy, S. Russell, Rao-Blackwellised filtering for dynamic Bayesian networks, in: *Uncertainty in Artificial Intelligence*, 2000, pp. 176–183.
- [46] V. Pavlović, J. Reh, T. Cham, K. Murphy, A dynamic Bayesian network approach to figure tracking using learned dynamic models, in: *International Conference on Computer Vision*, vol. 1, 1999, pp. 94–101.
- [47] K.P. Murphy, Learning switching Kalman filter models, Tech. Report 98-10, Compaq Cambridge Research Lab, 1998.
- [48] S.M. Oh, J.M. Reh, T. Balch, F. Dellaert, Learning and inference in parametric switching linear dynamic systems, in: *International Conference on Computer Vision*, vol. 2, 2005, pp. 1161–1168.
- [49] M. Tipping, C. Bishop, Mixtures of probabilistic principal component analyzers, *Neural Computation* 11 (1999) 443–482.
- [50] K. Murphy, S. Russel, Rao-Blackwellised particle filtering for dynamic Bayesian networks, in: A. Doucet, N. de Freitas, N. Gordon (Eds.), *Sequential Monte Carlo Methods in Practice*, Springer-Verlag, 2001.
- [51] K. Murphy, Kalman Filter Toolbox. Available from: <<http://www.cs.ubc.ca/~murphyk/Software/Kalman/kalman.html>>.
- [52] S. Shevade, S. Keerthi, A simple and efficient algorithm for gene selection using sparse logistic regression, *Bioinformatics* 19 (17) (2003) 2246–2253.
- [53] G. Cawley, N. Talbot, M. Girolami, Sparse multinomial logistic regression via Bayesian L1 regularisation, in: *Advances in Neural Information Processing Systems*, vol. 19, 2007.
- [54] R. van der Merwe, A. Doucet, N. de Freitas, E. Wan, The unscented particle filter, in: *Advances in Neural Information Processing Systems*, vol. 13, 2001.
- [55] L. Rabiner, A tutorial on Hidden Markov models and selected applications in speech recognition, *Proceedings of the IEEE* 77 (2) (1989) 257–286.
- [56] B. Lucas, T. Kanade, An iterative image registration technique with an application to stereo vision, in: *Proceedings of Imaging Understanding Workshop*, 1981, pp. 121–130.
- [57] <http://www.cs.ubc.ca/~vailen/imavis>.
- [58] Y. Wang, P. Sabzmejdani, G. Mori, Semi-latent Dirichlet allocation: A hierarchical model for human action recognition, in: *Proceedings 2nd Workshop on Human Motion Understanding, Modeling, Capture and Animation (at ICCV)*, 2007.