Multimedia Systems 10: 144–163 (2004)
Digital Object Identifier (DOI) 10.1007/s00530-004-0147-2

Multimedia Systems
© Springer-Verlag 2004

Anatomy of a multicamera video surveillance system

Long Jiao[1], Yi Wu[2], Gang Wu[2], Edward Y. Chang[2], Yuan-Fang Wang[1]

[1] Computer Science, University of California, Santa Barbara, CA 93106, USA
(e-mail: longjiao@cs.ucsb.edu, yfwang@cs.ucsb.edu)
[2] Electrical & Computer Engineering, University of California, Santa Barbara, CA 93106, USA
(e-mail: gwu@engineering.ucsb.edu, {wuyi, echang}@ece.ucsb.edu

Published online: 11 October 2004

Abstract. We present a framework for multicamera video surveillance. The framework consists of three phases: *detection*, *representation*, and *recognition*. The detection phase handles multisource spatiotemporal data fusion for efficiently and reliably extracting motion trajectories from video. The representation phase summarizes raw trajectory data to construct hierarchical, invariant, and content-rich descriptions of the motion events. Finally, the recognition phase deals with event classification and identification on the data descriptors. Through empirical study in a parking-lot-surveillance setting, we show that our spatiotemporal fusion scheme and biased sequence-data learning method are highly effective in identifying suspicious events.

Keywords: Video – Image – Motion – Surveillance – Recognition

1 Introduction

United States policymakers, especially in security and intelligence services, are increasingly turning to video surveillance as a means to combat terrorist threats and increase public security. With the proliferation of inexpensive cameras and the availability of high-speed, broadband wired/wireless networks, it has become economically and technically feasible to deploy a large number of cameras for security surveillance [38,40]. However, several important research questions must be addressed before we can rely upon video surveillance as an effective tool for crime prevention.

A surveillance task can be divided into three phases: *event detection*, *event representation*, and *event recognition* [13]. The detection phase handles multisource spatiotemporal data fusion for efficiently and reliably extracting motion trajectories from video. The representation phase summarizes raw trajectory data to construct hierarchical, invariant, and content-rich representations of the motion events. Finally, the recognition phase deals with event recognition and classification. The research challenges of the three phases are summarized as follows:

- *Event detection from multiple cameras*. Objects observed from multiple cameras should be integrated to build spatiotemporal patterns that correspond to three-dimensional viewing. Such integration must handle spatial occlusion and temporal shift (e.g., camera recording without timing synchronization and videotaping with differing frame rates). In addition, a motion pattern should not be affected by varying camera positions/poses and incidental environmental factors that can alter object appearance.
- *Hierarchical and invariant event description*. Invariant descriptions are those that are not affected by incidental change of environmental factors (e.g., lighting) and sensing configuration (e.g., camera placement). The concept of invariancy is applicable at multiple levels of event description. We distinguish two types of invariancy: *fine-grained* and *coarse-grained*. Fine-grained invariancy captures the characteristics of an event at a detailed, numeric level. Fine-grained invariant descriptors are therefore suitable for "intraclass" discrimination of similar event patterns (e.g., locating a particular event among multiple events depicting the same circling behavior of vehicles in a parking lot). Coarse-grained invariancy captures the motion traits at a concise, semantic level. Coarse-grained invariant descriptions are thus suitable for "interclass" discrimination, e.g., discriminating a vehicle's circling behavior from, say, its parking behavior. These two types of descriptors can be used synergistically to accomplish a recognition task.
- *Event recognition*. Event characterization deals with mapping motion patterns to semantics (e.g., benign and suspicious events). Traditional machine learning algorithms such as SVMs and decision trees cannot be directly applied to such infinite-dimensional data, which may also exhibit temporal ordering. Furthermore, positive events (i.e., the sought-for hazardous events) are always significantly outnumbered by negative events in the training data. In such an imbalanced set of training data, the class boundary tends to skew toward the minority class and becomes very sensitive to noise. (An example is presented in Sect. 4.2 to illustrate this problem.)

For event detection and representation, we configure two-level Kalman filters to fuse multisource video data, and we employ a variant of dynamic programming to construct

event descriptors. For effective event recognition, we first discretize a continuous spatiotemporal sequence. We propose a sequence-alignment kernel, which we show to be a legitimate *kernel function*, to discriminate events. For tackling the imbalanced-training-data problem, we propose the kernel-boundary-alignment (KBA) algorithm, which adaptively modifies the kernel matrix according to the training-data distribution. One particular application scenario we utilize to evaluate our algorithms is detecting suspicious activities in a parking lot. Our empirical study shows (1) that our spatiotemporal fusion scheme can efficiently and reliably reconstruct scene activities even when individual cameras may have spatial or temporal lapses and (2) that our sequence-alignment kernel and KBA algorithm are highly effective in identifying suspicious events.

The rest of the paper is organized as follows. In Sect. 2 we discuss related work. Section 3 addresses our sensor-data-fusion scheme and sequence-data representation. Section 4 presents event characterization and recognition methods. Section 5 contains empirical results. Finally, in Sect. 6 we offer some concluding remarks.

2 Related work

We divide our discussion of related work into two parts. The first part surveys related work in fusing data from multiple cameras. The second part discusses related research in sequence-data learning.

2.1 Sensor-data analysis and fusion

Sensor-data fusion from multiple cameras is an important problem with many potential applications. The work in multisource fusion can be divided into two categories based on camera configurations: spatially nonoverlapping and spatially overlapping. The study of [30] attempts to fuse data from several nonoverlapping cameras using a Bayesian reasoning approach. Since there might be significant gaps between the fields of view of the cameras, the precision in prediction may suffer. Most fusion algorithms assume an overlapping camera configuration and concentrate on fusing local coordinate frames of multiple cameras into one global coordinate system. For example, [32] assumes that only the intrinsic camera parameters are known, and the cameras are registered into a common frame of reference by aligning moving objects observed in multiple cameras spatially and temporally [29]. The DETER project [37] also uses an overlapping camera configuration. The homography between images from two cameras is computed, the images are mosaiced together to perform seamless tracking, and all data processing is then performed in the synthesized image plane. In this paper, we use a two-level hierarchy of Kalman filters for trajectory tracking and data fusion from multiple cameras. The advantage of our formulation is that it enables both bottom-up fusion and top-down guidance and hence is robust even with partial occlusion.

The Kalman filter is an important theoretical development for data smoothing and prediction. The traditional Kalman filter is a linear algorithm that operates under a prediction-correction paradigm. The quantities of a system to be estimated are summarized in an internal state vector, which is constrained by the observation of the system's external behavior. The prediction mechanism is used for propagating the system's internal state over time, and the correction mechanism is for fine-tuning the state propagation with external observations [47].

The Kalman filter and its variants are powerful tools for tracking inertial systems. Generally speaking, the standard Kalman filter performs satisfactorily for tracking the position of a moving object. However, for tracking the orientation of an object, where the governing equations in state propagation and observation may be nonlinear, the extended Kalman filter (e.g., nonlinear variants of the Kalman filter) must be used [34, 35]. For example, [2] uses a standard Kalman filter to predict the head position and an extension of it to estimate the head orientation of a user in a virtual reality system. In this paper, we are interested in summarizing the trajectory of a vehicle, and hence only the position, not the orientation, of a vehicle is needed. We have thus employed the traditional Kalman filter for the efficient tracking purpose.

In addition to the Kalman filter, the hidden Markov model (HMM) has been used for object tracking [4]. Both the Kalman filter and HMM can be used to estimate the internal states of a system. However, HMM is not an attractive online tracking method due to its high computational intensity with respect to the number of states. For tracking objects, where the number of possible locations (number of states) of the tracked objects is theoretically infinite, the Kalman filter is the popular choice [4]. In general, HMM is more suitable for recognition tasks [42,5], where the number of states is relatively small.

2.2 Sequence-data modeling and learning

Generative and discriminative models have been proposed to classify sequence data. The hidden Markov model (HMM) is the most widely used generative model [3] for describing sequence data. The HMM requires building a model for each pattern family. When an unknown pattern is to be classified, the Bayes rule returns the most likely model to depict it. For surveillance, however, building an HMM for each motion pattern, benign or suspicious, may not be realistic because of the scarcity of positive training data. (Learning an HMM, even with a moderate number of states, requires a large number of training instances [10].)

SVMs [44] are the most popular discriminative models; they directly estimate a discriminant function for each class. SVMs have been proven superior to generative models in classification problems for both effectiveness and efficiency. SVMs are applied to training data that reside in a vector space. The basic form of an SVM kernel function that classifies an input vector $\mathbf{x}$ is expressed as

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i y_i \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}) + b = \sum_{i=1}^{N} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\,, \quad (1)$$

where ϕ is a nonlinear mapping function that maps input vectors into the feature space, operator "$\cdot$" denotes the inner product operator, and $\mathbf{x}_i$ is the ith training sample. Parameters y_i

and α_i are class label and Lagrange multiplier of $\mathbf{x}_i$, respectively. K is a kernel function, and b is the bias.

For sequence data, in particular variable-length sequences, the basis function ϕ does not exist, nor does the feature space. Several attempts (e.g., [9,33]) have been made to convert sequence data to fixed-length segments and represent them in vector spaces, but these forced mappings of variable-length to fixed-length segments often result in loss of information. Furthermore, these models cannot capture the temporal relationship between spatial instances, nor do they consider secondary variables (e.g., velocity can be a secondary variable when traveling direction is the one).

Another vector-space approach is the SVM–Fisher kernel. The SVM–Fisher kernel [25,26] computes features from probabilistic models $p(x|\theta)$, in which θ is learned from the HMM training. It then uses the tangent vector of the log marginal likelihood $\log p(x|\theta)$ as the feature vectors. The SVM–Fisher kernel considers only positive training instances and does not take advantage of negative training instances in learning.

Fortunately, kernel methods (SVMs are a member of the kernel method family) require only a positive definite kernel matrix $K(\mathbf{x}_i, \mathbf{x}_j)$, which consists of a similarity measure between all pairs of training instances, without explicitly representing individual instances in a vector space [27]. More specifically, we can treat each sequence instance $\mathbf{x}_i$ as a random variable z_i. All we need is a kernel function that generates pairwise similarity between all pairs of z_i and z_j and that can ensure that the generated similarity matrix is positive definite. Put another way, as long as we have a similarity function that can produce a positive definite kernel matrix, we can use the kernel method to conduct sequence-data learning. Hence, our design task is reduced to formulating a function that can characterize the similarity between sequences, provided that the pairwise similarity matrix is positive definite. In this paper, we borrow an idea from sequence-data (DNA, RNA, and protein) analysis in biological science to measure sequence similarity based on alignment. We show in Sect. 4.1 the flexibility and effectiveness of this similarity function. Furthermore, we propose the *sequence-alignment* kernel, which fuses symbolic summarizations (in a nonvector space) with numeric descriptions (in a vector space). Because its ability to fuse primary structures with secondary structures that do or do not have a vector-space representation, our proposed kernel features great modeling flexibility.

Finally, to tackle the imbalanced-data-learning problem, we propose the kernel-boundary-alignment (KBA) scheme. For related work of imbalanced-data learning, please refer to [48].

3 Event detection and representation

The particular application scenario is that of automated detection and classification of suspicious activities (vehicles, humans) from surveillance video of a parking lot or a parking structure. This type of application requires solutions to all the above-mentioned difficulties: the optimal fusion of information from multiple cameras in both spatial and temporal domains, the ability to describe objects and motions in a way that is unaffected by camera placement and environmental variance, and the articulation of the query concept (suspicious vehicles) from highly skewed training datasets with high-dimensional description vectors.

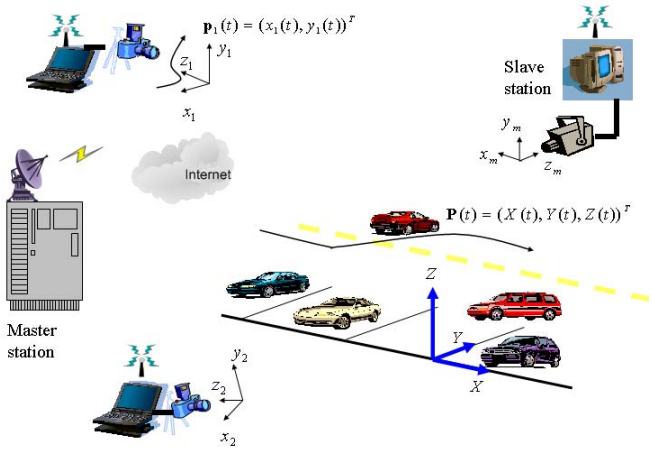


Fig. 1. Surveillance scenario configuration

The experimental scenario is depicted in Fig. 1. Multiple slave surveillance stations, each comprising a video camera connected to a host PC, are positioned at different locations to monitor the ground activities in a parking lot. There is no restriction on the number of cameras used, their brand names, or their capability. The camera aim can be stationary, follows a fixed sweep pattern, or is remotely controlled by a human operator. The view volumes of different cameras can be totally disjoint or can overlap partially. Furthermore, the clocks on different slave stations need not be synchronized. A single master station communicates with slave stations over wireless links for information gathering and fusion.

3.1 Event detection

The event-detection stage aims at achieving optimal fusion of multisensor data spatially and temporally to derive a hierarchical and invariant description of the scene activities. The sensor-data-fusion algorithm should address both the bottom-up data-integration problem and the top-down information-dissemination problem in a coherent framework. Several issues are addressed in our event-detection and sensor-fusion framework:

- *Variability in spatial coverage*: Multiple surveillance cameras have different fields of view. While it is not unreasonable to assume some overlap in the fields of view, the placement of cameras is to provide as wide a coverage of the surveillance area as possible. Hence, congruent activities must be identified and merged from multiple cameras.
- *Misalignment of temporal time stamp*: Often surveillance tapes do not come with time stamps. Even with time stamps, they are not accurate enough to allow recordings from multiple cameras to be synchronized for analysis. For example, a fast moving car can easily move out of view in as short as a few seconds. Hence, misalignment by even a few seconds temporally can have an impact on the analysis algorithm. Temporal registration is therefore as important as spatial registration in fusing multiple sensor data.

• *Occlusion and missing data*: Even with careful camera placement, occlusion of scene objects by environmental fixtures is often unavoidable. This implies that trajectories can easily break and remedial actions are needed to link and repair broken trajectories.

In the following subsections, we will discuss our solutions to these problems by proposing algorithms for spatial and temporal registration and for sensor-data fusion using a prediction-correction framework. Before proceeding any further, we will make clear the notational convention used in the ensuing discussion. Matrices and vectors are denoted by boldface characters; scalar quantities by plain-faced characters. Two-dimensional quantities will be in lowercase and 3D quantities in uppercase letters. We will use the tilde symbol to denote vectors in the homogeneous coordinates [17,18], i.e., appending an extra 1 as the last component of a vector. Hence, $\mathbf{p}$ and $\mathbf{P}$ denote 2D and 3D point coordinates and $\tilde{\mathbf{p}}$ and $\tilde{\mathbf{P}}$ the same coordinates in the homogeneous form.

3.1.1 Spatial registration

This step is for determining the essential camera parameters and the pose and aim of a camera in the environment. This is an essential step in video interpretation (e.g., to extract metric information from video). Camera calibration and pose registration is a topic that has been studied extensively in computer vision, remote sensing, and photogrammetry [17,22,49]. It is not our intention to reinvent the wheel by coming up with yet another calibration or pose-registration algorithm.[1] Here, for the sake of completeness, we provide a short description of the algorithms that we will use and motivate and justify our choices in this particular application domain.

The process of forming an image, using an *overly* simplified explanation, involves the determination of two things: *where* an object will appear (the *geometry* process) and *how* an object will appear (the *appearance* process) in an image. Camera calibration and pose registration are for answering the *where* (or the *geometry*) question. The geometry of image formation can be expressed mathematically as a concatenation of several coordinate transformations and projections that bring a 3D coordinate $\mathbf{P} = [X, Y, Z]^T$, specified in some global reference frame, to a 2D coordinate $\mathbf{p} = [x, y]^T$, specified in some camera coordinate frame. The process can be broken down into three stages:

1. $\tilde{\mathbf{T}}_{\mathbf{camera \leftarrow world}}$ A *world* to *camera* coordinate transform: This is specified as a 4×4 matrix in homogeneous coordinates that maps 3D coordinates (in a homogeneous form) specified in a global reference frame to 3D coordinates (again in a homogeneous form) in a camera- (or viewer-) centered reference frame. $\tilde{\mathbf{T}}_{\text{camera}\leftarrow\text{world}}$ is uniquely determined by a rotation plus a translation.

2. $\tilde{\mathbf{T}}_{\mathbf{ideal \leftarrow camera}}$ A *camera* to *ideal* image projective transform: This is specified as a 3×4 matrix in homogeneous coordinates that projects 3D coordinates (in a homogeneous form) in the camera reference frame onto the image plane of an ideal (a pinhole) camera. The model can be perspective, paraperspective, or weak perspective [22], which models the image formation process with differing degrees of fidelity. We will use the full perspective model. This matrix is of a fixed form with no unknown parameter, depending only on the projection model used.

3. $\tilde{\mathbf{T}}_{\mathbf{real \leftarrow ideal}}$ An *ideal* image to *real* image transform: This is specified as a 3×3 matrix in homogeneous coordinates that takes ideal projection coordinates into real projection coordinates. This process accounts for physical camera parameters (e.g., the center location of the image plane, the scale factors in the x and y directions, etc.) and lens distortion (e.g., spherical and achromatic aberration) in the real image formation process [41,49]. The combination of the above is

$$\tilde{\mathbf{p}}_{\text{real}} = \tilde{\mathbf{T}}_{\text{real}\leftarrow\text{ideal}}\tilde{\mathbf{T}}_{\text{ideal}\leftarrow\text{camera}}\tilde{\mathbf{T}}_{\text{camera}\leftarrow\text{world}}\tilde{\mathbf{P}}_{\text{world}} . \quad (2)$$

The parameters to be determined in the third step ($\tilde{\mathbf{T}}_{\text{real}\leftarrow\text{ideal}}$) are called the *intrinsic* camera parameters. They are affected only by the physical components of a camera (lens and CCD array used) and not by the physical placement of the camera in the surrounding environment. The process of determining these intrinsic parameters is called *calibration* and is usually performed once and offline. For this, we will use the algorithm of [51], which requires only two views of a planar calibration pattern, posed in relatively arbitrary orientations.

The parameters determined in the first step ($\tilde{\mathbf{T}}_{\text{camera}\leftarrow\text{world}}$) are the translation and rotation to align the camera and the world coordinate frames. This process is called *pose registration* and may need to be performed continuously online (e.g., if the camera is mounted on a swivel base). The performance of pose registration is thus more critical and time sensitive. If the camera's intrinsic parameters have been determined using offline calibration, then $\tilde{\mathbf{T}}_{\text{real}\leftarrow\text{ideal}}$ and $\tilde{\mathbf{T}}_{\text{ideal}\leftarrow\text{camera}}$ are known, and the three matrices in Eq. 2 can be combined into a single one: $\tilde{\mathbf{T}}_{\text{real}\leftarrow\text{world}}$:

$$\tilde{\mathbf{p}}_{\text{real}} = \tilde{\mathbf{T}}_{\text{real}\leftarrow\text{world}}\tilde{\mathbf{P}}_{\text{world}} . \quad (3)$$

Theoretically, $\tilde{\mathbf{T}}_{\text{real}\leftarrow\text{world}}$ in Eq. 3 is a 3×4 matrix with 12 unknowns. Now each landmark with known 3D world coordinates $\tilde{\mathbf{P}}_{\text{world}}$ and its projection in the real camera coordinate frame $\tilde{\mathbf{p}}_{\text{real}}$ provides two linear constraints using Eq. 3. Hence, a total of six point correspondences are needed for a linear closed-form solution to pose registration.

However, such a solution is not satisfactory for both theoretical and practical reasons. Theoretically, the method is computationally expensive, for it requires the inversions of a 12×12 matrix. Furthermore, the coefficient matrix may be ill-conditioned, depending on the configuration of the landmarks in space. Experience from implementing this algorithm shows that to avoid degeneration in the computation, one has to make sure that the six observed landmarks do not lie in the same plane (e.g., ground).

In the real-world scenario, given a surveillance tape taken of a parking lot at an arbitrarily chosen location, finding six landmarks that (1) have known world coordinates, (2) are present in the fields of view of multiple cameras, and (3) remain visible even with dynamic camera aims is a nontrivial task. Hence, alternative pose registration methods that require fewer landmarks for registration are important. However, it can be proven that with fewer than six landmarks, it is not possible

[1] A paper published back in 1989 [20] cited a German dissertation from 1958 that surveyed nearly 80 different solutions for camera pose registration from the photogrammetry literature.

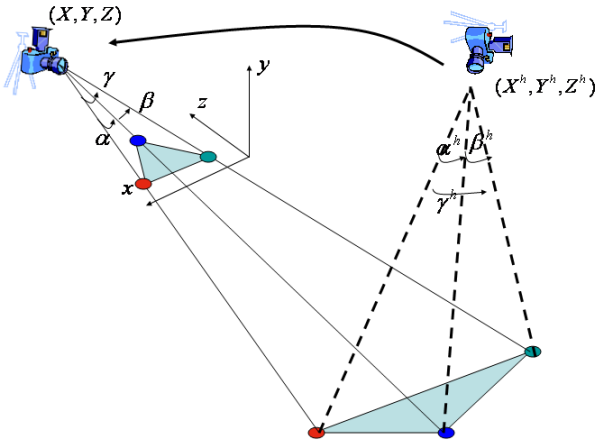


Fig. 2. Illustration of Church's algorithm

to have a linear closed-form solution to the pose-registration problem. DeMenthon and Davis [14] and Horaud et al. [22] present iterative solutions using four 3D landmarks of known world coordinates and their image coordinates. However, these four landmarks should be noncoplanar for high calibration accuracy, and this again presents a problem. (How do we know how tall a lamp post or some other fixture is at an arbitrarily chosen parking lot?)

Another alternative is to use an algorithm first developed by Earl Church back in 1945 for aerial photogrammetry. Church's algorithm [11] is an iterative, nonlinear optimization technique that requires only *three* landmarks for pose registration. Referring to Fig. 2, the solution is based on the condition that the face angle subtended by any two landmarks in space is equal to the face angle subtended at their corresponding image locations. The face angles (α, β, and γ in Fig. 2) formed by the camera origin (focal point) and the image projection of the landmarks can be calculated directly in the camera coordinate frame. However, because the location of the camera focal point [(X, Y, Z) in Fig. 2] in the world coordinate system is unknown, the face angles subtended by the landmarks in space are unspecified.

Church's method starts by guessing a value for the camera origin in the world coordinate system. With this hypothesized camera origin position [(X^h, Y^h, Z^h) in Fig. 2], we can form three face angles in the world coordinate system (α^h, β^h, and γ^h in Fig. 2). These face angles will in general not match those computed from the images unless the hypothesized camera position is correct. By partial differentiation of the difference between the two sets of face angles with respect to the current hypothesized position (X^h, Y^h, Z^h) we can get an adjustment $(\triangle X^h, \triangle Y^h, \triangle Z^h)$ that is added back to the hypothesized position. The process is iterated until the adjustment becomes insignificant and the hypothesized position converges to the correct position. The advantages of this algorithm are many:

- Since only three points are needed and they are necessarily coplanar, we can use the ground plane (Earth) for registration. It is not necessary to rely on environment fixtures such as a lamppost to establish the height information.
- For most parking lots, the marking of parking stalls is regular with a fixed spacing. Hence, it is relatively easy to establish a global reference frame by (1) selecting an origin in the repetitive parking stall marking (this can be chosen arbitrarily) and (2) extending the coordinate system coverage to a wide area based on the fixed-stall-spacing assumption. This allows the creation of a multitude of registration landmarks for wide spatial coverage.
- Therefore, no physical dimension measurements of environment fixtures are needed. Measurements can be expressed in terms of the "stall-marking unit." By extending the coordinate system coverage using the "stall-marking unit" it is now possible to register multiple cameras in the same global reference frame, even though the fields of view of the cameras have little overlap.
- Occlusion of landmarks by parked cars is less of a problem now as very few (a minimum of three) landmarks are needed for pose registration and it is not necessary to always use the same three landmarks.

Our experience with Church's algorithm is that it is very accurate and converges to the correct pose solution extremely fast with few iterations (less than five in most cases). The algorithm is very efficient to implement, and, with the current PC technology, it is possible to achieve thousands of pose updates per second.

Even though there is no universally applicable theoretical analysis on the accuracy and the rate of convergence of iterative nonlinear optimization techniques, fast and accurate convergence is often possible if a good initial guess to the final solution can be obtained. A good initial guess ensures the success of even simple and naive gradient descent optimization without expensive exhaustive search. For estimating time-varying camera aim and pose, it is always possible to obtain a good initial guess on the camera pose by using the camera pose at the previous frame. Given that the video frame rate (30 frames/s) is much faster than the servoing speed of a surveillance camera (panning of less than a few degrees/second), the initial guess from the previous frame is almost always close to the true solution. Hence, Church's algorithm is readily applicable in this application scenario.

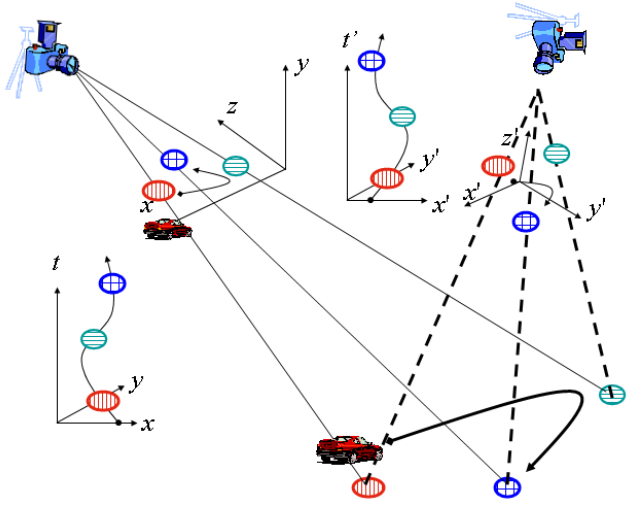


Fig. 3. Illustration of the temporal alignment algorithm

3.1.2 Temporal alignment

Our temporal alignment algorithm is illustrated graphically in Fig. 3. The same 3D trajectory is observed by multiple cameras. The same trajectory appears differently in different cameras' images because of the projective distortion. If we can somehow derive a unique, or invariant, "signature" of a 3D trajectory from its 2D projections, regardless of the particular way an image is generated, then we can correlate these invariant trajectories from different sensors to establish the time shift and, hence, solve the temporal registration problem.

More specifically, let $\mathbf{C}(t) = [x(t), y(t), z(t)]^T$ be a 3D motion trajectory, which is observed as $\mathbf{c}(t) = \mathbf{P}\mathbf{C}(t) = [x(t), y(t)]$ (where $\mathbf{P}$ denotes the projection matrix and, to simplify the math, we adopt the parallel projection model). The same motion, captured from a different viewpoint, is expressed as

$$\mathbf{c}'(t) = \mathbf{P}(\mathbf{R}\mathbf{C}(t - t_0) + \mathbf{T}) \,, \tag{4}$$

where $\mathbf{R}$ and $\mathbf{T}$ denote the rotation and translation resulting from a different camera placement, and t_o denotes the change in the camera's clock.[2]

The motion curve $\mathbf{c}'(t)$ can be recognized as the same as $\mathbf{c}(t)$ if we can derive the same signature for both in a way that is insensitive to changes in $\mathbf{R}$, $\mathbf{T}$, and t_o.

Under the parallel projection model and the far field assumption (where the object size is small relative to the distance to the camera, an assumption that is generally true for outdoor surveillance applications), it can be easily shown that

$$\mathbf{c}'(t) = \mathbf{A}\begin{bmatrix} x(t - t_o) \\ y(t - t_o) \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \mathbf{A}\mathbf{c}(t - t_o) + \mathbf{t} \,, \tag{5}$$

where $\mathbf{A}$ represents an affine transform and $\mathbf{t}$ the image-position shift. To derive a signature for a motion trajectory, we will extend the 2D image trajectory into the space by appending a third component, time, as $[\mathbf{c}, t]^T = [x, y, t]^T$ (Fig. 3). One can imagine that appending a third component t is like placing a slinky toy flat on the ground (the x-y plane) and then pulling and extending it up in the third (height) dimension. Now, it is well known in differential geometry [43] that a 3D curve is uniquely described (up to a rigid motion) by its curvature and torsion vectors with respect to its intrinsic arc length, where curvature and torsion vectors are defined as

$$\kappa(t) = \ddot{\mathbf{C}}(t), \qquad \tau(t) = \frac{d}{dt}(\dot{\mathbf{C}}(t) \times \ddot{\mathbf{C}}(t)) \tag{6}$$

or curvature and torsion vectors form a locally defined signature of a space curve. In computer vision jargon, (κ, τ) form an invariant parameter space – or a Hough transform space – and local structures are "hashed" into such a space independently of variations in the object's placement and rigid motion. Such a mapping is also insensitive to variation in the temporal shift, as the same pattern will show up sooner or later. It is also tolerant to occlusion, as the signature is computed locally, and the signature for the part of the trajectory that is not occluded will remain invariant. Hence, the recording using (κ, τ) in Eq. 6 is then insensitive to time shifts or partial occlusion. It is a simple invariant signature one can define on a motion trajectory.

To make such a hashing process invariant to difference in camera poses ($\mathbf{A}$ and $\mathbf{t}$), more processing of such trajectories is needed. In particular, variation in camera parameters has a tendency to change the magnitude of *area* quantities of a projection, or

$$\frac{d^n \mathbf{c}'(t)}{dt^n} = \mathbf{A}\frac{d^n \mathbf{c}(t - t_o)}{dt^n} \quad n \geq 1 \,. \tag{7}$$

By massaging the derivatives, we can derive many invariant expressions that do not depend on the camera pose ($\mathbf{A}$ and $\mathbf{t}$). For example,

$$U'(t) = \frac{||[\frac{d^{n+3}\mathbf{c}'(t)}{dt^{n+3}} \frac{d^{n+1}\mathbf{c}'(t)}{dt^{n+1}}]||}{||[\frac{d^{n+2}\mathbf{c}'(t)}{dt^{n+2}} \frac{d^{n}\mathbf{c}'(t)}{dt^{n}}]||}$$

$$= \frac{||[\mathbf{A}\frac{d^{n+3}\mathbf{c}(t-t_o)}{dt^{n+3}} \mathbf{A}\frac{d^{n+1}\mathbf{c}(t-t_o)}{dt^{n+1}}]||}{||[\mathbf{A}\frac{d^{n+2}\mathbf{c}(t-t_o)}{dt^{n+2}} \mathbf{A}\frac{d^{n}\mathbf{c}(t-t_o)}{dt^{n}}]||} = U(t - t_o) \,, \tag{8}$$

which forms an invariant local expression insensitive to affine pose change, as $\mathbf{A}$ and $\mathbf{t}$ do not appear in Eq. 8. The expression in Eq. 8 represents the ratio of the determinants of two 2×2 matrices, and the columns of the matrices are the derivatives of varying degrees of the observed curves. In our experiment, we used $n = 0$.

3.1.3 Sensor-data fusion

Here we present the methods we employ for sensor-data fusion. We use a hierarchical master-slave sensing configuration as shown in Fig. 4. At the bottom level, each slave station tracks the movements of scene objects semi-independently. The local trajectories (each represented as a state vector comprising the

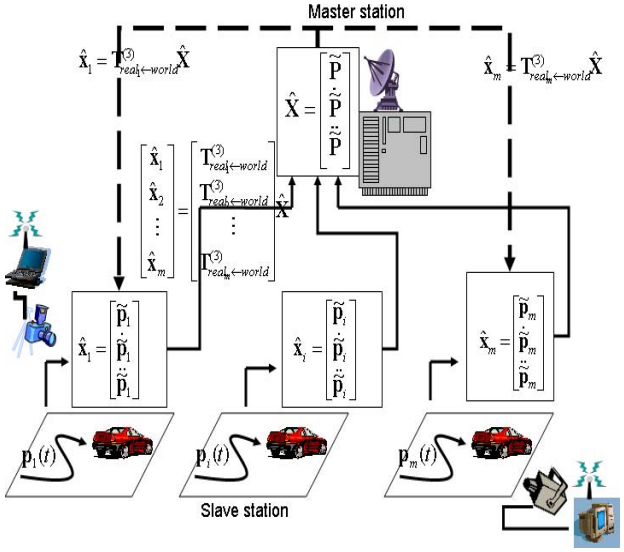


Fig. 4. Two-level hierarchical Kalman filter configuration

[2] It is assumed that video cameras have the same sampling rate, e.g., 30 frames/s for the NTSC standard. If the sampling rates are different, then it is necessary to incorporate a factor α in Eq. 4 to account for that, $\mathbf{c}'(t) = \mathbf{P}(\mathbf{R}\mathbf{C}(\frac{t-t_0}{\alpha}) + \mathbf{T})$. It is still possible to derive unique signatures in this case [50], but the formulation will not be presented here.

estimated position, velocity, and acceleration of the tracked object) are then relayed to a master station for fusing into a consistent, global representation. This represents a "bottom-up" analysis paradigm. Furthermore, as each individual camera has a limited field of view, and occlusion occurs due to scene clutter, we also employ a "top-down" analysis module that disseminates fused information from the master station to slave stations that might lose track of an object.

At slave stations we employ two different mechanisms for event detection and tracking, both based on the powerful hypothesis-and-verification paradigm. The difference is in the number of hypotheses maintained. When the state prior and noise processes can both be modeled as unimodal Gaussian processes, a single state is maintained. The Kalman filter has proven to be very effective in such situations. If the state prior or the noise processes have a more complicated form, then it is necessary to sample the state space in a more elaborate manner. We employ a formulation that is similar to the importance-based condensation algorithm [23] in this case. We describe these two mechanisms in more detail below.

Kalman-filter-based tracking

We use the Kalman filter [6,36] as the tool for fusing information spatially and temporally from multiple cameras for motion event detection. The Kalman filter is an optimal linear data-smoothing and prediction algorithm. It has been applied extensively in control, signal processing, and navigation applications since its introduction in 1960. Our contribution is in using two-level Kalman filters to fuse data from multiple sources.

The Kalman filter has been widely used to estimate the internal state of a system based on the observation of the system's external behavior [6,36]. Furthermore, a system's state estimate can be computed and then updated by incorporating external measurements iteratively – *without* recomputing the estimate from scratch each time a new measurement becomes available. Such an iterative process is optimal in the sense that the Kalman filter incorporates all available information from past measurements, weighted by their precision. Optimal information fusion is achieved by combining three factors: (1) knowledge of the system and measurement device dynamics; (2) the statistical description of the system noises, measurement errors, and uncertainty in the system model; and (3) relevant initial-state description [36]. While the Kalman filter is optimal only among *linear* estimators, and when certain assumptions about the noise processes are valid, it is easy to implement and is efficient at runtime. Work has also been done on relaxing some of the assumptions, such as the Gaussian noise assumption and the linearity assumption [28].

Suppose that a vehicle is moving in a parking lot. Its trajectory is described in the global reference system by

$$\mathbf{P}(t) = [X(t), Y(t), Z(t)]^T.$$

The trajectory may be observed in camera i as

$$\mathbf{p}_i(t) = [x_i(t), y_i(t)]^T,$$

where $i = 1, \cdots, m$ (m is the number of cameras used). The question is then how to best estimate $\mathbf{P}(t)$ given $\mathbf{p}_i(t), i = 1, \cdots, m$.

We formulate the solution as a two-level hierarchy of the Kalman filters. Referring to Fig. 4, at the bottom level of the hierarchy we employ for each camera a Kalman filter to estimate, independently, the position $\mathbf{p}_i(t)$, velocity $\dot{\mathbf{p}}_i(t)$, and acceleration $\ddot{\mathbf{p}}_i(t)$ of the vehicle, based on the tracked image trajectory of the vehicle in the *local camera reference frame*. Or in Kalman filter jargon, the position, velocity, and acceleration vectors establish the "state" of the system while the image trajectory serves as the "observation" of the system state. At the top level of the hierarchy, we use a single Kalman filter to estimate the vehicle's position $\mathbf{P}(t)$, velocity $\dot{\mathbf{P}}(t)$, and acceleration $\ddot{\mathbf{P}}(t)$ in the *global world reference frame* – this time using the estimated positions, velocities, and accelerations from multiple cameras ($\mathbf{p}_i(t)$, $\dot{\mathbf{p}}_i(t)$, $\ddot{\mathbf{p}}_i(t)$) as observations (solid feed-upward lines in Fig. 4). This is possible because we address the spatial and temporal registration problems in Sects. 3.1.1 and 3.1.2. We also allow dissemination of fused information to individual cameras (dashed feed-downward lines in Fig. 4) to help to guide image processing.

Specifically, for each camera we create a system state vector for each observed vehicle as $\hat{\mathbf{x}}_i(t) = \begin{bmatrix} \hat{\mathbf{p}}_i(t) \\ \hat{\dot{\mathbf{p}}}_i(t) \\ \hat{\ddot{\mathbf{p}}}_i(t) \end{bmatrix}$, where we use the hat notation $\hat{\mathbf{x}}$ to denote our best estimate to the true state $\mathbf{x}$.[3] The goal is then to generate and update this state estimate $\hat{\mathbf{x}}_i(t)$ recursively over time. The Kalman filter answers this question: Supposing that we already have an estimate of the system state at time $t - \Delta t$, what can we say about the state at time t? The question is answered using the following two mechanisms: (1) By extrapolating the state from $t - \Delta t$ to t (accomplished by a state prediction equation) and (2) by incorporating new observation of the system's behavior at time t. This is accomplished by an observation equation and an update mechanism.

By using Taylor series expansion, we can write the state prediction equation as

$$\begin{aligned} \hat{\mathbf{x}}_i(t^-) &= \mathbf{A}_i \hat{\mathbf{x}}_i(t - \Delta t) + \mathbf{w}_i(t) \\ \begin{bmatrix} \hat{\mathbf{p}}_i(t^-) \\ \hat{\dot{\mathbf{p}}}_i(t^-) \\ \hat{\ddot{\mathbf{p}}}_i(t^-) \end{bmatrix} &= \begin{bmatrix} \mathbf{I}_3 & \Delta t \mathbf{I}_3 & \frac{\Delta t^2}{2}\mathbf{I}_3 \\ \mathbf{0}_3 & \mathbf{I}_3 & \Delta t \mathbf{I}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{p}}_i(t - \Delta t) \\ \hat{\dot{\mathbf{p}}}_i(t - \Delta t) \\ \hat{\ddot{\mathbf{p}}}_i(t - \Delta t) \end{bmatrix} + \mathbf{w}_i(t), \end{aligned} \tag{9}$$

where $\mathbf{0}_3$ and $\mathbf{I}_3$ represent 3×3 null and identity matrices, respectively. The minus sign in $\hat{\mathbf{x}}_i(t^-)$ implies that the state at time t is estimated by extrapolating the state estimate at time $t - \Delta t$, before any new measurements at time t are incorporated. $\mathbf{w}_i(t)$ represents the model prediction uncertainty (using a few terms in the Taylor series expansion). We assume that $\mathbf{w}_i(t)$ is a random variable with a zero mean and a covariance matrix $\mathbf{Q}_i(t) = E[\mathbf{w}_i(t)\mathbf{w}_i^T(t)]$.

Now when a new measurement (a new image) is acquired and the vehicle's trajectory is computed, we incorporate this new observation to update our estimate of the system state. This update requires two equations: (1) a measurement equa-

[3] To avoid complicating the notation, we will use a single subscript i to denote the camera number, instead of double subscript i_j to denote the jth vehicle in the ith camera. The Kalman estimation mechanism is the same for all vehicles.

tion that relates the system's external behavior to its internal state and (2) an update equation that properly weighs the state estimates from the state extrapolation process (Eq. 9) and the external observation process (Eq. 10) to generate a new state estimate. The measurement equation in our case is simple: We estimate a vehicle's position, velocity, and acceleration by taking equivalent finite difference approximations from the corresponding trajectory:

$$\mathbf{z}_i(t) = \mathbf{H}_i\hat{\mathbf{x}}_i(t) + \mathbf{v}_i(t)$$
$$\begin{bmatrix} \hat{\tilde{\mathbf{P}}}_i(t) \\ \frac{\hat{\tilde{\mathbf{P}}}_i(t)-\hat{\tilde{\mathbf{P}}}_i(t-\Delta t)}{\Delta t} \\ \frac{\hat{\tilde{\mathbf{P}}}_i(t)-2\hat{\tilde{\mathbf{P}}}_i(t-\Delta t)+\hat{\tilde{\mathbf{P}}}_i(t-2\Delta t)}{\Delta t^2} \end{bmatrix} = \begin{bmatrix} \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 \end{bmatrix} \begin{bmatrix} \hat{\tilde{\mathbf{P}}}_i(t) \\ \dot{\hat{\tilde{\mathbf{P}}}}_i(t) \\ \ddot{\hat{\tilde{\mathbf{P}}}}_i(t) \end{bmatrix} + \mathbf{v}_i(t), \quad (10)$$

where $\mathbf{z}_i(t)$ represents the external observation (i.e., vehicle's image trajectory) and $\mathbf{v}_i(t)$ represents the uncertainty in such an observation. We assume that $\mathbf{v}_i(t)$ is zero mean with a covariance matrix $\mathbf{R}_i(t) = E[\mathbf{v}_i(t)\mathbf{v}_i^T(t)]$. To merge the estimates from state extrapolation (Eq. 9) and external observation (Eq. 10), we use an update equation:

$$\hat{\mathbf{x}}_i(t^+) = \hat{\mathbf{x}}_i(t^-) + \mathbf{K}_i(t)(\mathbf{z}_i(t) - \mathbf{H}_i\hat{\mathbf{x}}_i(t^-)) . \quad (11)$$

This implies that the state estimate after incorporating the new measurement $[\hat{\mathbf{x}}_i(t^+)]$ is a weighted sum of the state estimate from the extrapolation process $[\hat{\mathbf{x}}_i(t^-)$ in Eq. 9] and the "novel" or "original" part of the current observation $[\mathbf{z}_i(t)]$ that cannot be explained by the state extrapolation $[\mathbf{H}_i\hat{\mathbf{x}}_i(t^-)]$ (called "innovation" in Kalman filter jargon).

The only thing left, then, is to determine the weighting factor $\mathbf{K}_i(t)$ (or the Kalman gain in Kalman filter jargon), which is summarized in the following three equations:

$$\mathbf{K}_i(t) = \mathbf{E}_i(t^-)\mathbf{H}_i^T(t) \times [\mathbf{H}_i(t)\mathbf{E}_i(t^-)\mathbf{H}_i^T(t) + \mathbf{R}_i(t)]^{-1} , \quad (12)$$
$$\mathbf{E}_i(t^-) = \mathbf{A}_i\mathbf{E}_i(t-\Delta t)\mathbf{A}_i^T + \mathbf{Q}_i(t) , \quad (13)$$
$$\mathbf{E}_i(t^+) = [\mathbf{I} - \mathbf{K}_i(t)\mathbf{H}_i(t)]\mathbf{E}_i(t^-) , \quad (14)$$

where $\mathbf{E}_i(t) = E[(\mathbf{x}_i(t) - \hat{\mathbf{x}}_i(t))(\mathbf{x}_i(t) - \hat{\mathbf{x}}_i(t))^T]$ represents the error covariance matrix in the state estimation process. Equation 12 computes the weighting factor by properly weighting the uncertainty (or error) in the observation $[\mathbf{R}_i(t)]$ and the state propagation $[\mathbf{E}_i(t^-)]$. Equations 13 and 14 show how this error covariance can be propagated from time $t - \Delta t$ to t before the new measurement is incorporated (Eq. 13) and after the new measurement is incorporated (Eq. 14). The six Eqs. 9–14 summarize the recursive operations of the Kalman filter. Interested readers can consult books [6,36], survey papers, and online tutorials (e.g., [46]) for details.

For the global trajectory, we create a global state vector for each observed vehicle as $\hat{\mathbf{X}}(t) = \begin{bmatrix} \hat{\tilde{\mathbf{P}}}(t) \\ \dot{\hat{\tilde{\mathbf{P}}}}(t) \\ \ddot{\hat{\tilde{\mathbf{P}}}}(t) \end{bmatrix}$. Then we can write the state prediction equation for the global trajectory as follows (this is identical to the individual camera case, in which we propagate the state vector forward using the Taylor series expansion):

$$\hat{\mathbf{X}}(t + \Delta t) = \mathbf{A}\hat{\mathbf{X}}(t) + \mathbf{W}(t)$$
$$\begin{bmatrix} \hat{\tilde{\mathbf{P}}}(t+\Delta t) \\ \dot{\hat{\tilde{\mathbf{P}}}}(t+\Delta t) \\ \ddot{\hat{\tilde{\mathbf{P}}}}(t+\Delta t) \end{bmatrix} = \begin{bmatrix} \mathbf{I}_4 & \Delta t\mathbf{I}_4 & \frac{\Delta t^2}{2}\mathbf{I}_4 \\ \mathbf{0}_4 & \mathbf{I}_4 & \Delta t\mathbf{I}_4 \\ \mathbf{0}_4 & \mathbf{0}_4 & \mathbf{I}_4 \end{bmatrix} \begin{bmatrix} \hat{\tilde{\mathbf{P}}}(t) \\ \dot{\hat{\tilde{\mathbf{P}}}}(t) \\ \ddot{\hat{\tilde{\mathbf{P}}}}(t) \end{bmatrix} + \mathbf{W}(t) , \quad (15)$$

where $\mathbf{0}_4$ and $\mathbf{I}_4$ represent 4×4 null and and identity matrices, respectively. For writing the global measurement equation, we use the state vectors estimated from individual cameras as the "observables" to constrain the global state. Through camera calibration, we should have derived the $\mathbf{T}_{\text{image}\leftarrow\text{world}}$ matrix, which allows $\tilde{\mathbf{p}}$ from an individual camera to be related to $\tilde{\mathbf{P}}$ in the global reference system. Hence the measurement equation can be written as

$$\mathbf{Z}(t) = \mathbf{H}\hat{\mathbf{X}}(t) + \mathbf{V}(t)$$
$$\begin{bmatrix} \hat{\mathbf{x}}_1(t) \\ \hat{\mathbf{x}}_2(t) \\ \cdots \\ \hat{\mathbf{x}}_m(t) \end{bmatrix} = \begin{bmatrix} \mathbf{T}^{(3)}_{\text{image}_1\leftarrow\text{world}} \\ \mathbf{T}^{(3)}_{\text{image}_2\leftarrow\text{world}} \\ \cdots \\ \mathbf{T}^{(3)}_{\text{image}_m\leftarrow\text{world}} \end{bmatrix} \hat{\mathbf{X}}(t) + \mathbf{V}(t) , \quad (16)$$

where

$$\mathbf{T}^{(3)}_{\text{image}_i\leftarrow\text{world}} = \begin{bmatrix} \mathbf{T}_{\text{image}_i\leftarrow\text{world}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{T}_{\text{image}_i\leftarrow\text{world}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{T}_{\text{image}_i\leftarrow\text{world}} \end{bmatrix}, \quad (i = 1, \cdots, m) \quad (17)$$

where $\mathbf{T}_{\text{image}_i\leftarrow\text{world}}$ is the ith camera's calibration matrix. Some discussion points are in order:

- First, why is a two-stage hierarchical approach taken? Why not just combine image trajectories from all cameras directly in a single filter to estimate the global state? There are many reasons, but most notably it has to do with the efficiency in processing and communication. The state prediction and measurement equations (Eqs. 9 and 10) are much smaller for individual cameras than for their global counterparts (Eqs. 15 and 16). In particular, $\mathbf{H}$ in Eq. 16 can be as large as $9m \times 12$. Hence, it would be best to do some processing locally and relay only essential information (instead of raw data) to a centralized fusion point to reduce the communication and computation load on the single server. (For example, if local processing can determine that no movement is observed in the ith camera's field of view, $\hat{\mathbf{x}}_i$ does not have to be used in Eq. 16.)

  Furthermore, we envision future surveillance systems will be fairly "compartmentalizable." A surveillance station can be made up of a camera and associated mounting device, controlled by an inexpensive PC with a digitizer card and disk storage. A complete surveillance system then comprises many such individualized stations to achieve scalability. In this scenario, we should take advantage of the processing power of individual surveillance stations to parallel process video footage instead of using one big server to achieve integration from raw data.
- Second, the hierarchy of Kalman filters allows for two-way communication. What we illustrated above is for fusing information from multiple cameras into a global state estimate

(a bottom-up process). It is also possible to employ a top-down process for information dissemination from the global Kalman filter to individual cameras. This process is useful, say, in a scenario where a vehicle is circling a parking lot and may move in and out of the view of a surveillance camera. When a vehicle moves out of a camera's view, the error in measurement (Eq. 10) increases quickly to infinity and the future state estimates will rely on the state extrapolation (Eq. 9). However, according to the theory of Taylor series expansion [7], the error in such extrapolation grows with time [in our case, the error grows $O(\Delta t^3)$]. Hence, soon the state estimate will become highly unreliable without observation of the vehicle's movement (e.g., without valid video footage, we will not know if a car made a turn outside the field of view of the camera).

Much more reliable information on the vehicle's movement can be gathered from other cameras whose fields of view still include the vehicle of interest. Since the information is fused at the global Kalman filter, that information can be relayed to the camera that has lost sight of the vehicle. The following equation (similar to Eq. 16)

$$\hat{\mathbf{x}}_i(t) = \mathbf{T}^{(3)}_{\text{image}_i \leftarrow \text{world}} \hat{\mathbf{X}}(t) \tag{18}$$

can be used for this purpose, with a prediction error of $\mathbf{E}_i(t)$

$$\begin{aligned}
&= E[(\mathbf{x}_i(t) - \hat{\mathbf{x}}_i(t))(\mathbf{x}_i(t) - \hat{\mathbf{x}}_i(t))^T] \\
&= \mathbf{T}^{(3)}_{\text{image}_i \leftarrow \text{world}} E[(\mathbf{X}(t) - \hat{\mathbf{X}}(t))(\mathbf{X}(t) - \hat{\mathbf{X}}(t))^T] \\
&\quad \times \mathbf{T}^{(3)^T}_{\text{image}_i \leftarrow \text{world}} \\
&= \mathbf{T}^{(3)}_{\text{image}_i \leftarrow \text{world}} \mathbf{E}(t) \mathbf{T}^{(3)^T}_{\text{image}_i \leftarrow \text{world}} \cdot
\end{aligned} \tag{19}$$

This time, $\hat{\mathbf{X}}(t)$ acts as the "observation" to constrain $\hat{\mathbf{x}}_i(t)$, the state of the camera that loses its view of the vehicle.

Multihypothesis tracking

While the Kalman filter is a simple and powerful mechanism for state estimation, its validity is challenged if the assumption on the prior and noise is not valid. Furthermore, there are situations where multiple hypotheses have to be kept until a later time when more visual evidence is gathered to validate some and discredit others. For example, if two or more persons enter the field of view of a camera in such a way that their silhouettes overlap, the tracking algorithm will not know in general whether such a moving region corresponds to a single person or multiple persons. Only when the group of people split later and head in different directions can the single-person hypothesis be safely discarded.

Our approach here is to employ a robust, yet still real-time, control and fail-over mechanism – on top of low-level frame-differencing- and correlation-based tracking – to deal with noise, scene clutter, short periods of absence and merging of silhouettes, and long periods of occlusion of activities from a camera's field of view, situations that can easily fail simple Kalman-filter-based tracking. Our formulation is based on the powerful hypothesis-and-verification paradigm, which has been alternatively christened as Monte Carlo filtering [31], particle filtering [39], genetic algorithms [21], condensation (conditional density propagation) [23], and Icondensation (importance-based condensation) [24]. The utility of such a hypothesis-verification formulation over traditional linear state estimation algorithms such as Kalman filtering is that the noise processes do not have to be Gaussian and state propagation does not have to be unimodal. This allows multiple competing hypotheses to be maintained and contribute to the state estimation process. If we denote sensor data as $\mathbf{z}$, then multiple hypotheses allow us not to assume a particular parametric form of $p(\mathbf{x}|\mathbf{z})$. Instead, $p(\mathbf{x}|\mathbf{z})$ can be learned by sampling with multiple hypotheses using Bayesian rule $[p(\mathbf{x}|\mathbf{z}) \propto p(\mathbf{z}|\mathbf{x})p(\mathbf{x})]$. In this sense, hypothesis verification is akin to a Bayesian estimator instead of a maximum likelihood estimator [15].

Different incarnations of the same hypothesis-verification principle all comprise the following essential components:[4] (1) A collection of candidate states and their likelihood estimates that are initialized, propagated, and updated over time; (2) a state-propagation mechanism; (3) a state-verification mechanism using sensor feedback; and (4) a renormalization process to refresh and/or regenerate the collection of candidate states and their associated likelihood. Step 1 represents sampling the prior distribution $p(\mathbf{x})$, step 2 is for evolving the prior distribution over time, step 3 is for computing associated conditional $p(\mathbf{z}|\mathbf{x})$, and, finally, step 4 is used for computing $p(\mathbf{x}|\mathbf{z}) \propto p(\mathbf{z}|\mathbf{x})p(\mathbf{x})$ and regenerating state vectors based on updated $p(\mathbf{x}|\mathbf{z})$. We describe below our formulation in relation to these four steps.

State vector: We employ simple frame differencing and region growing for the initial detection of presence of activities. When a moving region is identified in a camera's image, multiple hypotheses (up to the number allowed for real-time execution) are postulated. A single-object hypothesis is to represent such a region as a single moving object, characterized by a state comprising the estimated position, velocity, and acceleration of the moving region. Alternatively, the region might represent a group of up to n objects, with their silhouettes merged into a single composite region in an image. Hence, for a camera, multiple state vectors $[\hat{\mathbf{p}}_{ij}(t), \dot{\hat{p}}_{ij}(t), \ddot{\hat{p}}_{ij}(t)]^T, 0 \le j < i \le n$ are created. The likelihood of the hypothesis $\pi_i, 1 \le i \le n$ is initialized as some fixed constant or dependent on the size or color of the region.

State propagation: Given a state vector at time $t - \Delta t$, we propagate the state vector to time t as

$$\begin{aligned}
\hat{\mathbf{p}}^-(t) &= \hat{\mathbf{p}}(t - \Delta t) + \dot{\hat{p}}(t - \Delta t)\Delta t + \frac{1}{2}\ddot{\hat{p}}(t - \Delta t)\Delta t^2 \\
&\quad + \mathbf{N}(\mathbf{0}, \mathbf{\Sigma_p}), \\
\dot{\hat{p}}^-(t) &= \dot{\hat{p}}(t - \Delta t) + \ddot{\hat{p}}(t - \Delta t)\Delta t + \mathbf{N}(\mathbf{0}, \mathbf{\Sigma_v}), \quad \text{and} \\
\ddot{\hat{p}}^-(t) &= \ddot{\hat{p}}(t - \Delta t) + \mathbf{N}(\mathbf{0}, \mathbf{\Sigma_a}),
\end{aligned} \tag{20}$$

where $\mathbf{N}$ is Gaussian noise processes of a zero mean and suitable covariance matrices $\mathbf{\Sigma_p}$, $\mathbf{\Sigma_v}$, and $\mathbf{\Sigma_a}$. The minus superscript denotes the state before correction or validation using sensor data. Again, how many such $\hat{\mathbf{x}}^-$ states we can generate depends on the real-time processing requirement.

[4] The difference is in the exact mechanism for realizing these. For example, standard condensation algorithms generate and update candidate states based strictly on the Bayesian formula, while importance-based condensation allows auxiliary information (e.g., from sensors) to be brought in to better position candidate states for efficiently exploring the state space.

State validation: We employ a validation mechanism in the spirit of Icondensation and regularization, which allows multiple sources of information to be used. Specifically, to update a state, i.e., to go from $\hat{\mathbf{x}}^-$ to $\hat{\mathbf{x}}^+$, where "+" denotes states after sensor information is incorporated, we employ two sources of information: (1) sensor data from the particular sensor and (2) sensor data from other sensors. Both information sources can be exploited in a high-resolution or a low-resolution form.

Referring to Fig. 4, if we are successful in locating the object at time t in the kth camera's image, then the tracked object position is recovered as $\mathbf{z}_t^{(k)} = (x_t^{(k)}, y_t^{(k)})^T$, where the subscript *t* indicates "tracked" position. However, the camera might lose track of the object under two scenarios: (1) the tracked object is still in the field of view of the camera, but its movement is so swift or jerky as to make its position fall outside the "attention window" of the tracking algorithm; or (2) the tracked object is simply not identifiable, either because of a high degree of occlusion and scene clutter or because the object has moved out of the camera's field of view. What we need here are mechanisms to either reacquire the object (scenario 1 above) or to solicit help from other sensors (scenario 2 above).

To efficiently reacquire the object, it is necessary to enlarge the attention window of the tracking algorithm to search a bigger neighborhood for the presence of the object. We accomplish this by downsampling the current frame, looking for telltale signs of the tracked object in the downsampled image (based on color and texture similarity with the tracked object) and remapping potential locations back to the original resolution. We denote candidate locations found through this reacquisition process as $\mathbf{z}_r^{(k)}$ (subscript *r* represents reacquired position) to distinguish them from $\mathbf{z}_t^{(k)}$, candidate locations found by successful tracking.

Simultaneously, the master fusion station shown in Fig. 4 is queried. The fused, consensus object location (from the most recent time instance) is projected back into the particular camera's frame of reference to generate candidate object positions, using

$$\tilde{\mathbf{z}}_f^{(k)} = \mathbf{T}_{\text{image}_k \leftarrow \text{global}} \tilde{\mathbf{P}}\,, \tag{21}$$

where subscript *f* represents positions predicted by the consensus through information fusion and $\mathbf{T}_{\text{image}_k \leftarrow \text{global}}$ is a matrix that relates global coordinate and local camera coordinate (estimated through the camera calibration process).

State normalization: The available computing power at a slave station allows updating and tracking only a finite number of states in real time. Hence, the available computing power must be distributed judiciously among all moving regions with some reserved for detecting possible appearance (or reappearance) of new moving objects. For a particular moving region, its allocated share of computing resources must be used in an optimal way, which, according to Bayesian rule, means generating tracking hypotheses based on the posterior. If we are successful in maintaining tracking, then we can update the confidence as

$$\begin{aligned}\pi(t) &= p(\hat{\mathbf{x}}^-(t)|\mathbf{z}_t(t)) \\ &\propto p(\mathbf{z}_t(t)|\hat{\mathbf{x}}^-(t))\pi(t-\Delta t)\,.\end{aligned} \tag{22}$$

Here, we assume that $p(\hat{\mathbf{x}}(t)|\mathbf{z}_t(t), \mathbf{z}_t(t-\Delta t), \cdots) = p(\hat{\mathbf{x}}(t)|\mathbf{z}_t(t))$. By collecting π from multiple hypotheses of the same object, we generate a discrete, sampled version of $p(\mathbf{x}|\mathbf{z})$. We then integrate the density function to generate the mass function $P(\mathbf{x}|\mathbf{z}) = \int p(\mathbf{x}|\mathbf{z})d\mathbf{x}$ and redistribute states ($\hat{\mathbf{x}}^+$) based on golden-rule sampling (i.e., more states for places of the state spaces with higher densities).

However, if the camera somehow loses track of the object, such a simple Bayesian formula is not applicable. We have to refresh candidate states using $\mathbf{z}_r$ and $\mathbf{z}_f$ as well. This is to refocus the processing power on the parts of the state space that are likely to contain valid candidates, based on information gathered from a cursory search of the current image ($\mathbf{z}_r$) and that supplied by other sensors ($\mathbf{z}_f$).

Hence, the decision for regenerating/refreshing state vectors is made as follows: (1) If the low-level tracker gives a high-confidence measurement in tracking, the states are refreshed mostly based on $\mathbf{z}_t$. Additional states (based on $\mathbf{z}_r$) can be generated if π calculated based on the above equation is low. (2) Otherwise, the states are regenerated based on $\mathbf{z}_r$ and $\mathbf{z}_f$.

3.2 Hierarchical, invariant representation

The raw trajectory data derived above are in terms of either local or global Cartesian coordinates. Such a representation suffers from at least two problems: (1) the same motion trajectory observed by different cameras will have different representations and (2) the representation is difficult for a human operator to understand. Our solution is to summarize such raw trajectory data using syntactic and semantic descriptors that are not affected by incidental changes in environmental factors and camera poses. We briefly describe our semantic descriptors here.

We first segment a raw trajectory fused from multiple cameras into fragments. Using a constrained optimization approach under the EM (expectation-maximization) framework, we then label these fragments semantically (e.g., a fragment representing a left turn action). We approximate the acceleration trajectory of a vehicle as a piecewise constant (zeroth-order) or linear (first-order) function in terms of its direction and its magnitude. When the magnitude of acceleration is first order [$\mathbf{r}(t) = \mathbf{r}_o + t\mathbf{r}_1$ in Eq. 23], it gives rise to a motion trajectory that is a concatenation of piecewise polynomials that can be as high as third order (cubic). This is often considered sufficient to describe a multitude of motion curves in the real world (e.g., in computer-aided design [16] and computer graphics [18], piecewise third-order Hermite, Beźier, and B-spline curves are universally used for design and manufacturing). We chop the whole acceleration trajectory $\ddot{\mathbf{P}}(t)$ from $t = t_{\min}$ to $t_{\max}$ (where $[t_{\min}, t_{\max}]$ is the time interval that a vehicle is observed by one or more of the surveillance cameras) into, say, k pieces such that $t_o \le t_1 \le \cdots \le t_k$ and $t_o = t_{\min}, t_k = t_{\max}$:

$$\begin{aligned}&\ddot{\mathbf{P}}(t) = \mathbf{r}(t)e^{i\theta(t)}, \quad \text{where} \\ &\begin{cases}\mathbf{r}(t) = \mathbf{r}_o^{(i)} \;\; or \;\; \mathbf{r}_o^{(i)} + t\mathbf{r}_1^{(i)} \\ \theta(t) = \theta_o^{(i)} \;\; or \;\; \theta_o^{(i)} + t\theta_1^{(i)}\end{cases} \\ &t_i \le t < t_{i+1}, i = 0, \cdots, k-1.\end{aligned} \tag{23}$$

We use lowercase **r** here to avoid confusion because uppercase **R** was used earlier to denote a rotation matrix. While one might

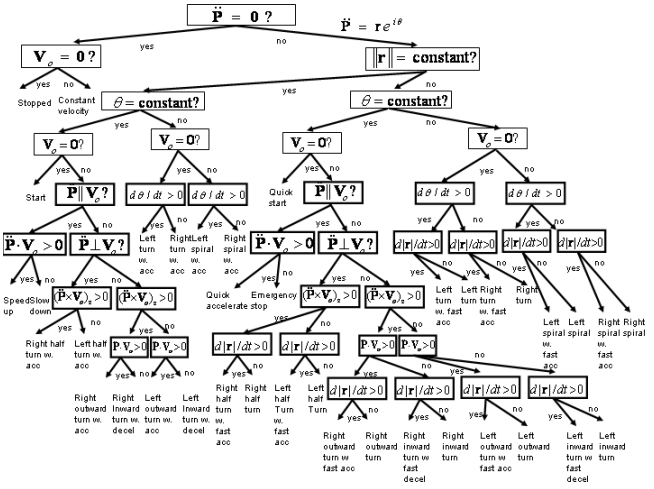


Fig. 5. An example motion pattern classification

argue that this assumption is restrictive or that more general acceleration patterns exist, we made this assumption for the following reasons:

We employ an iterative expectation and maximization (EM) algorithm [15] to segment trajectories. The EM algorithm consists of two stages: (1) The E-stage hypothesizes the number of segments with their start and stop locations, and (2) the M-stage optimizes the fitting parameters based on the start and stop locations and the number of segments derived from the E-stage. These two steps iterate until the solution converges. Table 1 sketches the pseudocode of the algorithm (using fitting $\theta(t)$ as an illustration).

We label each segmented fragment based on its acceleration and velocity statistics. More specifically, we denote the initial vehicle velocity when each segment starts as $\mathbf{V}_o$, which can be either zero or nonzero. The acceleration (Eq. 23) can be either of a constant or linearly varying magnitude and/or of a constant or a linearly varying direction. For example, if $|\mathbf{r}| \approx 0$, the motion pattern is either "constant speed" or "stop." Segmentation based on θ is meaningful and necessary only when $|\mathbf{r}| > 0$. If $|\mathbf{r}| > 0$, possible motion patterns include "speed up," "slow down," "left turn," and "right turn." "Speed up" and "slow down" can be determined by the sign of $\ddot{\mathbf{P}} \cdot \dot{\mathbf{P}}$. "Left turn" and "right turn" are determined by the sign of $(\ddot{\mathbf{P}} \times \dot{\mathbf{P}})_z$. If $(\ddot{\mathbf{P}} \times \dot{\mathbf{P}})_z > 0$, it is a right turn; otherwise, it is a left turn.

In Fig. 5, we sketch different motion patterns based on the assumption made on the acceleration ($\ddot{\mathbf{P}}$ in Eq. 23) and $\mathbf{V}_o$. For example, the `stop` condition is identified as zero acceleration and zero initial velocity. If the initial velocity is zero, if the magnitude of the acceleration is not zero (constant or changes linearly), and if the direction of acceleration is constant, we label such an action a `start` action. A `half turn` (a vehicle making a turn of approximately $90°$) is often observed at a crossroad or when a vehicle is pulling into a parking stall. It can be modeled as an acceleration of a constant direction that is roughly perpendicular to the direction of the initial velocity.

It should be noted that Fig. 5 does not sketch out all possible scenarios. For example, when the angle of the acceleration is linear $\theta = \theta_1 t + \theta_o$, the possible motion trajectories have many variations, depending on the relative orientation and size of θ_1, $\mathbf{r}_o$, and $\mathbf{V}_o$. Some of them are quite counterintuitive with unnatural twists and bents. Another example is that when $\mathbf{V}_o$ and $\ddot{\mathbf{P}}$ are antiparallel and $\dddot{\mathbf{P}}$ is a constant, the velocity will decrease to zero and then increase again, but in the opposite direction. We label this a "slow-down" action, not a "slow-down-then-speed-up-in-the-other-direction" action. This is because for a vehicle to accelerate in the opposite direction, it is necessary to shift gears, which often breaks the continuum in motion. Hence, we consider only the cases that make sense semantically. For example, if $\mathbf{r}$ is in the direction of $\mathbf{V}_o$ and $|\mathbf{r}|$ is increasing, it indicates either a fast acceleration ($\mathbf{r}$ and $\mathbf{V}_o$ are parallel) or an emergency stop ($\mathbf{r}$ and $\mathbf{V}_o$ are antiparallel).

4 Event recognition

Let us recap our discussion in Sect. 3. Our descriptors summarize a motion trajectory as an ordered sequence of labels, and each label corresponds to a motion segment with a humanly understandable action. Recognizing and identifying events on such descriptors must handle the ordered nature of the descriptors. Furthermore, in a surveillance setting, positive (suspicious) events are always significantly outnumbered by negative events. As we will explain shortly through an example, this imbalanced training-data situation can skew the decision boundary toward the minority class and hence cause high rates of false negatives (i.e., failure to identify suspicious events). In this section, we first design a sequence-alignment kernel function to work with SVMs for correlating events. We then propose using kernel-boundary alignment (KBA) to deal with the imbalanced-training-data problem.

4.1 Sequence-alignment learning

In the previous section, we labeled each segmented fragment of a trajectory with a semantic label and its detailed attributes including velocity and acceleration statistics. For convenience, we use a symbol to denote the semantic label. We use "C" for "**C**onstant speed," "D" for "slow **D**own," "L" for "**L**eft turn," "R" for "**R**ight turn," and "U" for "speed **U**p." We label each segment with a two-level descriptor: a primary segment symbol and a set of secondary variables (e.g., velocity and acceleration). We use $\mathbf{s}$ to denote a sequence that comprises the concatenation of segment symbols $s_i \in \mathcal{A}$, where $\mathcal{A}$ is the legal symbol set. We use $\mathbf{v}_i$ to denote the vector of the ith secondary variable.

The following example depicts a sequence with this two-level descriptor. Sequence $\mathbf{s}$ denotes the segmented trajectory with $\mathbf{v}_1$ representing the velocity and $\mathbf{v}_2$ the acceleration. For velocity and acceleration, we use their average values in a segment.

$\mathbf{s}$:	C	D	U	C	L	R	R	L
$\mathbf{v}_1$:	0.7	0.5	0.8	0.8	0.7	0.8	0.6	0.5
$\mathbf{v}_2$:	0.0	-0.2	0.3	0.0	-0.1	0.1	-0.2	-0.1

Now, the trajectory-learning problem is converted to the problem of sequence-data learning with secondary variables.

Table 1. The motion event segmentation process

(1) Initialization. Compute a linear fit to the $\theta(t)$ curve between the specified end points, denoted as $[t_{\min}, t_{\max})$. Using the notation $\theta_{\max} = \theta(t_{\max})$ and $\theta_{\min} = \theta(t_{\min})$, we have

$$(\theta_{\max} - \theta_{\min})t + (t_{\min} - t_{\max})\theta + (\theta_{\min} - \theta_{\max})t_{\min} + (t_{\max} - t_{\min})\theta_{\min} = 0\,. \tag{24}$$

(2) Refinement. Compute location t_{maxdev} between $t_{\min}$ and $t_{\max}$ as the largest deviation of the true acceleration curve from the fitting

$$t_{\text{maxdev}} = argmax_t|(\theta_{\max} - \theta_{\min})t + (t_{\min} - t_{\max})\theta(t) + (\theta_{\min} - \theta_{\max})t_{\min} + (t_{\max} - t_{\min})\theta_{\min}|/\Delta\,, \tag{25}$$

$$maxdev = |(\theta_{\max} - \theta_{\min})t_{\text{maxdev}} + (t_{\min} - t_{\max})\theta(t_{\text{maxdev}}) + (\theta_{\min} - \theta_{\max})t_{\min} + (t_{\max} - t_{\min})\theta_{\min}|/\Delta\,, \tag{26}$$

where $\Delta = \sqrt{(\theta_{\max} - \theta_{\min})^2 + (t_{\max} - t_{\min})^2}$.

(3) Iteration. If $maxdev$ is above a preset threshold, break the curve into two sections $[t_{\min}, t_{\text{maxdev}})$ and $[t_{\text{maxdev}}, t_{\max})$ and repeat the first two steps using these two new intervals.

For this purpose, we construct a new *sequence-alignment kernel* that can be applied to measure the pairwise similarity between sequences with secondary variables.

4.1.1 Tensor product kernel

The sequence-alignment kernel will take into consideration both the degree of conformity of the symbolic summarizations and the similarity between the secondary numerical descriptions (i.e., velocity and acceleration) of the two sequences. Two separate kernels are used for these two criteria and are then combined into a single sequence-alignment kernel through a *tensor product*. These are explained below.

Let $\mathbf{x} \in \mathcal{X}$ be a composite structure and $x_1, \ldots, x_N$ be its "parts," where $x_n \in \mathcal{X}_n$, $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_N$, and N is a positive integer. For our sequence data, $\mathbf{x}$ is a sequence with both primary segment symbols and secondary variables. Let x_1 denote its primary symbol sequence, and every other x_i be its $(i-1)$th secondary vector. Assume that $\mathcal{X}$, $\mathcal{X}_1$,…,$\mathcal{X}_N$ are nonempty sets. We define the *tensor product kernel* as follows:

Definition 1. Tensor product kernel. *Given* $\mathbf{x} = (x_1, \ldots, x_N) \in \mathcal{X}$ *and* $\mathbf{x}' = (x'_1, \ldots, x'_N) \in \mathcal{X}$. *If* K_1,…,K_N *are (positive definite) kernels defined on* $\mathcal{X}_1 \times \mathcal{X}_1$,…, $\mathcal{X}_N \times \mathcal{X}_N$ *respectively, then their tensor product,* $K_1 \otimes \cdots \otimes K_N$, *defined on* $\mathcal{X} \times \mathcal{X}$, *is*

$$K_1 \otimes \cdots \otimes K_N(\mathbf{x}, \mathbf{x}') = K_1(x_1, x'_1) \cdots K_N(x_N, x'_N).\square$$

Since kernels are closed under product [45], it is easy to see that the tensor product kernel is positive definite if each individual kernel is positive definite.

4.1.2 Sequence-alignment kernel

To measure the similarity between two sequences, our idea is to first compare their similarity at the symbol level. After the similarity is computed at the primary level, we consider the similarity at the secondary variable level. We then use the tensor product kernel to combine the similarity at the primary and secondary level.

At the primary (segment-symbol) level, we use kernel $K_s(\mathbf{s}, \mathbf{s}')$ to measure symbol-sequence similarity. We define $K_s(\mathbf{s}, \mathbf{s}')$ as a joint probability distribution (p.d.) that assigns a higher probability to more similar sequence pairs. We employ pairs-HMM (PHMM) [45], a generative probability model, to model the joint p.d. of two symbol sequences. (Note that PHMM is different from HMM, which aims to model the *evolution* of individual sequence data. PHMM is one of many *dynamic-programming*-based methods that one can employ to perform string alignment.)

A realization of PHMM is a sequence of states, starting with **START** and finishing with **END**; and in between there are three possible states: **AB**, **A**, and **B**. State **AB** emits two symbols, state **A** emits one symbol for sequence $\mathbf{a}$ only, and state **B** emits one symbol for sequence $\mathbf{b}$ only. State **AB** has an emission probability distribution $p_{\mathbf{a}_i\mathbf{b}_j}$ for emitting an aligned $\mathbf{a}_i : \mathbf{b}_j$, and states **A** and **B** have distributions $q_{\mathbf{a}_i}$ and $q_{\mathbf{b}_j}$, respectively, for emitting a symbol against a gap, such as $\mathbf{a}_i$: '$-$' and '$-$': $\mathbf{b}_j$. Parameter δ denotes the transition probability from **AB** to an insert gap state **A** or **B**, ε the probability of staying in an insert state, and τ the probability of a transition into the **END** state. Any particular pair of sequences $\mathbf{a}$ and $\mathbf{b}$ may be generated by exponentially many different realizations. The dynamic programming algorithms can sum over all possible realizations to calculate the joint probability of any two sequences. The overall computational complexity is $O(mn)$, in which m and n are the lengths of the two sequences respectively.

To compute the similarity at the secondary level, we can concatenate all variables into one vector and employ a traditional vector-space kernel such as an RBF function. Let $K_v(\mathbf{v}, \mathbf{v}')$ denote such a kernel measuring the distance between $\mathbf{v}$ and $\mathbf{v}'$. (Notice that vectors $\mathbf{v}$ and $\mathbf{v}'$ may differ in length since $\mathbf{s}$ and $\mathbf{s}'$ may have different lengths. We will discuss shortly how we align two vectors into the same length via an example.) Finally, we define the tensor product on $(\mathcal{S} \times \mathcal{S}) \times (\mathcal{V} \times \mathcal{V})$ as

$$(K_s \otimes K_v)((\mathbf{s}, \mathbf{v}), (\mathbf{s}', \mathbf{v}')) = K_s(\mathbf{s}, \mathbf{s}')K_v(\mathbf{v}, \mathbf{v}'). \tag{27}$$

In the following discussion we present an example to show the steps of computing similarity between two sequences using our sequence-alignment kernel.

Example 1. Suppose we have two sequences $(\mathbf{s}, \mathbf{v})$ and $(\mathbf{s}', \mathbf{v}')$ as depicted next. The similarity between the sequences is computed in the following three steps:

$\mathbf{s}:$	C	D	U	C	L	R	R	L
$\mathbf{v}:$	0.7	0.5	0.8	0.8	0.7	0.8	0.6	0.5
$\mathbf{s}':$	C	U	C	L	R	L	C	
$\mathbf{v}':$	0.5	0.4	0.4	0.5	0.6	0.6	0.6	

Step 1. Primary symbol-level similarity computation: $K_s(\mathbf{s}, \mathbf{s}')$.
By using PHMM, we can obtain the joint p.d. $K_s(\mathbf{s}, \mathbf{s}')$ between symbol sequences $\mathbf{s}$ and $\mathbf{s}'$. As a part of the PHMM computation, two sequences are aligned as follows:

C	D	U	C	L	R	R	L	$-$
C	$-$	U	C	L	R	$-$	L	C

Step 2. Secondary variable-level similarity computation: $K_v(\mathbf{v}, \mathbf{v}')$.
The unaligned positions in $\mathbf{v}$ and $\mathbf{v}'$ are padded by zero. We obtain two equal-length vectors and can compute their similarity by using a traditional SVM kernel, e.g., an RBF function. Note that the unaligned positions will not be counted in the calculation since we do not want to double penalize the unaligned symbols (we already penalize them at the first step of symbolic-level similarity measurement).

0.7	0.5	0.8	0.8	0.7	0.8	0.6	0.5	**0.0**
0.5	**0.0**	0.4	0.4	0.5	0.6	**0.0**	0.6	0.6

Step 3. Tensor fusion: $(K_s \otimes K_v)((\mathbf{s}, \mathbf{v}), (\mathbf{s}', \mathbf{v}'))$. □

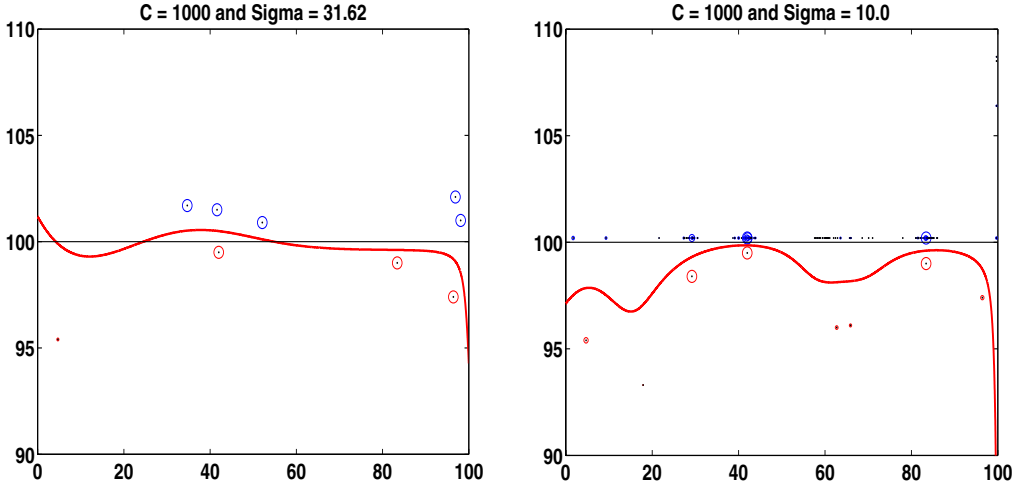
There are three advantages to the above *sequence-alignment kernel*. First, it can use any sequence-alignment algorithms to obtain a pairwise probability distribution for measuring variable-length sequence similarity. (Again, we employ PHMM to perform the measurement.) Second, the kernel considers not only the alignment of symbol strings but also secondary variables, making the similarity measurement between two sequences more informative. Third, compared with the SVM–Fisher kernel (discussed in Sect. 2), our sequence-alignment kernel adds the ability to learn from negative training instances as well from positive training instances.

4.2 Imbalanced learning via kernel-boundary alignment

Skewed class boundary is a subtle but severe problem that arises in using an SVM classifier – in fact in using *any* classifier – for real-world problems with imbalanced training data. To understand the nature of the problem, let us consider it in a binary (positive vs. negative) classification setting. Recall that the Bayesian framework estimates the posterior probability using the class conditional and the prior [19]. When the training data are highly imbalanced, it can be inferred that the state of the nature favors the majority class much more than the other. Hence, when ambiguity arises in classifying a particular sample because of similar class conditional densities for the two classes, the Bayesian framework will rely on the large prior in favor of the majority class to break the tie. Consequently, the decision boundary will skew toward the minority class.

To illustrate this skew problem graphically, we use a 2D checkerboard example. The checkerboard divides a 200×200 square into four quadrants. The top-left and bottom-right quadrants contain negative (majority) instances while the top-right and bottom-left quadrants are occupied by positive (minority) instances. The lines between the classes are the "ideal" boundary that separates the two classes. In the rest of the paper, we will use *positive* when referring to minority instances and *negative* when referring to majority instances.

Figure 6 exhibits the boundary distortion between the two left quadrants in the checkerboard under two different negative/positive training-data ratios, where a black dot with a circle represents a support vector, and its radius represents the weight value α_i of the support vector. The bigger the circle, the larger the α_i. Figure 6a shows the SVM class boundary when the ratio of the number of negative instances (in the quadrant above) to the number of positive instances (in the quadrant below) is $10:1$. Figure 6b shows the boundary when the ratio increases to $10,000:1$. The boundary in Fig. 6b is much more skewed toward the positive quadrant than the boundary in Fig. 6a and hence causes a higher incidence of false negatives.



(a) 10:1 (b) 10,000:1

Fig. 6. Boundaries of different ratios

While the Bayesian framework gives the optimal results (in terms of the smallest average error rate) in a theoretical sense, one has to be careful in applying it to real-world applications. In a real-world application such as security surveillance, the risk (or consequence) of mispredicting a positive event (a false negative) far outweighs that of mispredicting a negative event (a false positive). It is well known that in a binary classification problem, Bayesian risks are defined as

$$R(\alpha_p|\mathbf{x}) = \lambda_{pp}P(\omega_p|\mathbf{x}) + \lambda_{pn}P(\omega_n|\mathbf{x})\,,$$
$$R(\alpha_n|\mathbf{x}) = \lambda_{np}P(\omega_p|\mathbf{x}) + \lambda_{nn}P(\omega_n|\mathbf{x})\,, \quad (28)$$

where p and n refer to the positive and negative events, respectively, λ_{np} refers to the risk of a false negative, and λ_{pn} refers to the risk of a false positive. Which action (α_p or α_n) to take – or which action has a smaller risk – is affected not just by the event likelihood (which directly influences the misclassification error), but also by the risk of mispredictions (λ_{np} and λ_{pn}).

For security surveillance, positive (suspicious) events often occur much less frequently than negative (benign) events. This fact causes imbalanced training data and thereby results in higher incidence of false negatives. To remedy this boundary-skew problem, we propose an adaptive conformal transformation algorithm. In the remainder of this section, we first outline how our prior work [48] deals with the problem in a vector space (Sect. 4.2.1). We then present our solution to sequence-data learning where a discretized variable-length sequence may not have a vector-space representation (Sect. 4.2.2).

4.2.1 Conformally transforming K

In [48], we proposed feature-space adaptive conformal transformation (ACT) for imbalanced-data learning. We showed that conducting conformal transformation adaptively to data distribution and adjusting the degree of magnification based on feature-space distance (rather than on input-space distance as proposed by [1]) can remedy the imbalanced-data learning problem.

A conformal transformation, also called a conformal mapping, is a transformation T that takes the elements $X \in D$ to elements $Y \in T(D)$ while preserving the local angles between the elements after mapping, where D is a domain in which the elements X reside [12].

Kernel-based methods, such as SVMs, introduce a mapping function Φ that embeds the input space I into a high-dimensional feature space F as a curved Riemannian manifold S where the mapped data reside [1,8]. A Riemannian metric $g_{ij}(\mathbf{x})$ is then defined for S, which is associated with the kernel function $K(\mathbf{x}, \mathbf{x}')$ by

$$g_{ij}(\mathbf{x}) = \left(\frac{\partial^2 K(\mathbf{x}, \mathbf{x}')}{\partial x_i \partial x'_j}\right)_{\mathbf{x}'=\mathbf{x}}. \quad (29)$$

The metric g_{ij} shows how a local area around $\mathbf{x}$ in I is magnified in F under the mapping of Φ. The idea of conformal transformation in SVMs is to enlarge the margin by increasing the magnification factor $g_{ij}(\mathbf{x})$ around the boundary (represented by support vectors) and to decrease it around the other points. This could be implemented by a conformal transformation of the related kernel $K(\mathbf{x}, \mathbf{x}')$ according to Eq. 29, so that the spatial relationship among the data would not be affected too much [1]. Such a conformal transformation can be depicted as

$$\tilde{K}(\mathbf{x}, \mathbf{x}') = D(\mathbf{x})D(\mathbf{x}')K(\mathbf{x}, \mathbf{x}'). \quad (30)$$

In the above equation, $D(\mathbf{x})$ is a properly defined positive conformal function. $D(\mathbf{x})$ should be chosen in such a way that the new Riemannian metric $\tilde{g}_{ij}(\mathbf{x})$ associated with the new kernel function $\tilde{K}(\mathbf{x}, \mathbf{x}')$ has larger values near the decision boundary. Furthermore, to deal with the skew of the class boundary caused by imbalanced classes, we magnify $\tilde{g}_{ij}(\mathbf{x})$ more in the boundary area close to the minority class. In [48], we demonstrate that an RBF distance function such as

$$D(\mathbf{x}) = \sum_{k \in \mathbf{SV}} \exp(-\frac{|\mathbf{x} - \mathbf{x}_k|}{\tau_k^2}) \quad (31)$$

is a good choice for $D(\mathbf{x})$.

In Eq. 31, we can see that, if τ_k^2's are fixed for all support vectors $\mathbf{x}_k$'s, $D(\mathbf{x})$ would be very dependent on the density of support vectors in the neighborhood of $\Phi(\mathbf{x})$. To alleviate this problem, we adaptively tune τ_k^2 according to the spatial distribution of support vectors in F [48]. This goal can be achieved by the following equation:

$$\tau_k^2 = \mathrm{AVG}_{i \in \{\|\Phi(\mathbf{x}_i) - \Phi(\mathbf{x}_k)\|^2 < M,\ y_i \neq y_k\}} \times \left(\|\Phi(\mathbf{x}_i) - \Phi(\mathbf{x}_k)\|^2\right). \quad (32)$$

In this equation, the average on the right-hand side comprises all support vectors in $\Phi(\mathbf{x}_k)$'s neighborhood within the radius of M but having a different class label. Here, M is the average distance of the nearest and the farthest support vectors from $\Phi(\mathbf{x}_k)$. Setting τ_k^2 in this way takes into consideration the spatial distribution of the support vectors in F. Although the mapping Φ is unknown, we can play the kernel trick to calculate the distance in F:

$$\|\Phi(\mathbf{x}_i) - \Phi(\mathbf{x}_k)\|^2 = K(\mathbf{x}_i, \mathbf{x}_i) + K(\mathbf{x}_k, \mathbf{x}_k) - 2 * K(\mathbf{x}_i, \mathbf{x}_k). \quad (33)$$

Substituting Eq. 33 into Eq. 32, we can then calculate the τ_k^2 for each support vector, which can adaptively reflect the spatial distribution of the support vector in F, not in I.

When the training dataset is very imbalanced, the class boundary would be skewed toward the minority class in the input space I. We hope that the new metric $\tilde{g}_{ij}(\mathbf{x})$ would further magnify the area far away from a minority support vector $\mathbf{x}_i$ so that the boundary imbalance could be alleviated. Our algorithm thus assigns a multiplier for the τ_k^2 in Eq. 32 to reflect the boundary skew in $D(\mathbf{x})$. We tune $\tilde{\tau}_k^2$ as $\eta_p\tau_k^2$ if $\mathbf{x}_k$ is a minority support vector; otherwise, we tune it as $\eta_n\tau_k^2$. Examining Eq. 31, we can see that $D(\mathbf{x})$ is a monotonously increasing function of τ_k^2. To increase the metric $\tilde{g}_{ij}(\mathbf{x})$ in an area that is not very close to the support vector $\mathbf{x}_k$, it would be better to choose a larger η_p for the τ_k^2 of a minority support vector. For a majority support vector, we can choose a smaller η_n so as to minimize influence on the class boundary. We empirically demonstrate that η_p and η_n are proportional to the skew of support vectors, or η_p as $O(\frac{|\mathbf{SV}^-|}{|\mathbf{SV}^+|})$, and η_n as $O(\frac{|\mathbf{SV}^+|}{|\mathbf{SV}^-|})$, where $|\mathbf{SV}^+|$ and $|\mathbf{SV}^-|$ denote the number of minority and majority support vectors, respectively. (Please see [48] for the details of ACT.)

4.2.2 Modifying $\boldsymbol{K}$

For data that do not have a vector-space representation (e.g., sequence data), ACT may not be applicable. In this work we thus propose KBA, which modifies kernel matrix $\mathbf{K}$ based on training-data distribution. Kernel matrix $\mathbf{K}$ contains the pairwise similarity information between all pairs of instances in a training dataset. Hence, in kernel-based methods, all we need is a kernel matrix to learn the classifier; even the data do not reside in a vector space. Notice that KBA is certainly applicable to data that *do* have a vector-space representation since $\mathbf{K} = (k_{\mathbf{x}\mathbf{x}'}) = K(\mathbf{x}, \mathbf{x}')$.

Now, since a training instance $\mathbf{x}$ might not be a vector, in this paper we introduce a term, *support instance*, to denote $\mathbf{x}$ if its embedded point via $\mathbf{K}$ is a support vector.[5] In this situation, we cannot choose $D(\mathbf{x})$ as in Eq. 31. (It is impossible to calculate the Euclidean distance $|\mathbf{x}-\mathbf{x}_i|$ for nonvector data.) In Sect. 4.2.1, we show that $D(\mathbf{x})$ should be chosen in such a way that the spatial resolution of the manifold S would be magnified around the support instances. In other words, if $\mathbf{x}$ is close to a support instance $\mathbf{x}_k$ in F (or in its neighborhood), we hope that $D(\mathbf{x})$ would be larger so as to achieve a greater magnification. In KBA, we use the pairwise similarity $k_{\mathbf{x}\mathbf{x}_k}$ to measure the distance of $\mathbf{x}$ from $\mathbf{x}_k$ in F. Therefore, we choose $D(\mathbf{x})$ as

$$D(\mathbf{x}) = \sum_{k \in \mathbf{SI}} \exp\left(-\frac{\frac{1}{k_{\mathbf{x}\mathbf{x}_k}} - 1}{\tau_k^2} \right), \tag{34}$$

where $\mathbf{SI}$ denotes the support-instance set and τ_k^2 controls the magnitude of $D(\mathbf{x})$.

Figure 7 illustrates a $D(\mathbf{x})$ for a given support instance $\mathbf{x}_k$, where we can see that $D(\mathbf{x})$ (y-axis) becomes larger when an instance $\mathbf{x}$ is more similar to $\mathbf{x}_k$ (a larger $k_{\mathbf{x}\mathbf{x}_k}$ on the x-axis), so that there would be more magnification on the spatial resolution around the support vector embedded by $\mathbf{x}_k$ in F. Notice in the figure that $D(\mathbf{x})$ can be shaped very differently with different τ_k^2. We thus need to adaptively choose τ_k^2 as

$$\tau_k^2 = \mathrm{AVG}_{i \in \{Dist^2(\mathbf{x}_i, \mathbf{x}_k) < M,\ y_i \neq y_k\}} \times \left(Dist^2(\mathbf{x}_i, \mathbf{x}_k) \right), \tag{35}$$

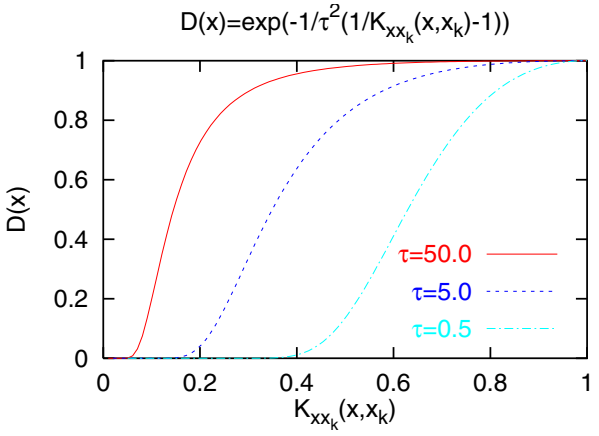


Fig. 7. $D(\mathbf{x})$ with different τ_k^2

[5] In the KBA algorithm, if $\mathbf{x}$ is a support instance, we call both $\mathbf{x}$ and its embedded support vector via $\mathbf{K}$ in F a *support instance*.

where the distance $Dist^2(\mathbf{x}_i, \mathbf{x}_k)$ between two support instances $\mathbf{x}_i$ and $\mathbf{x}_k$ is calculated via the kernel trick as

$$Dist^2(\mathbf{x}_i, \mathbf{x}_k) = k_{\mathbf{x}_i\mathbf{x}_i} + k_{\mathbf{x}_k\mathbf{x}_k} - 2 * k_{\mathbf{x}_i\mathbf{x}_k}. \tag{36}$$

The neighborhood range M in Eq. 35 is chosen as the average of the minimal distance $Dist^2_{\min}$ and the maximal distance $Dist^2_{\max}$ from $\mathbf{x}_k$. In addition, τ_k^2 is scaled in the same way as in Sect. 4.2.1 for dealing with the imbalanced-training-data problem.

Figure 8 summarizes the KBA algorithm. We apply KBA on the training dataset X_{train} until the testing accuracy on X_{test} cannot be further improved. In each iteration, KBA adaptively calculates τ_k^2 for each support instance (step 10), based on the distribution of support instances in feature space F. KBA scales the τ_k^2 according to the negative-to-positive support-instance ratio (steps 11 to 14). Finally, KBA updates the kernel matrix and performs retraining on X_{train} (steps 15 to 18).

Input:
X_{train}, X_{test}, $\mathbf{K}$;
θ; /* *stopping threshold* */
T; /* *maximum running iterations* */
Output:
$\mathcal{C}$; /* *output classifier* */
Variables:
$\mathbf{SI}$; /* *support-instance set* */
M; /* *neighborhood range* */
$\mathbf{s}$; /* *a support instance* */
$\mathbf{s}.\tau$; /* *parameter of* $\mathbf{s}$ */
$\mathbf{s}.y$; /* *class label of* $\mathbf{s}$ */
Function Calls:
SVMTrain(X_{train}, $\mathbf{K}$); /* *train classifier* $\mathcal{C}$ */
SVMClassify(X_{test}, $\mathcal{C}$); /* *classify* X_{test} *by* $\mathcal{C}$ */
ExtractSI($\mathcal{C}$); /* *obtain* $\mathbf{SI}$ *from* $\mathcal{C}$ */
ComputeM($\mathbf{s}$, $\mathbf{SI}$); /* *compute* M */
Begin
1) $\mathcal{C} \leftarrow$ SVMTrain(X_{train}, $\mathbf{K}$);
2) $\varepsilon_{\text{old}} \leftarrow \infty$;
3) $\varepsilon_{\text{new}} \leftarrow$ SVMClassify(X_{test}, $\mathcal{C}$);
4) $t \leftarrow 0$;
5) **while** $((\varepsilon_{\text{old}} - \varepsilon_{\text{new}} > \theta) \&\& (t < T))$ {
6) $\mathbf{SI} \leftarrow$ ExtractSI($\mathcal{C}$);
7) $\eta_p \leftarrow O(\frac{|\mathbf{SI}^-|}{|\mathbf{SI}^+|})$, $\eta_n \leftarrow O(\frac{|\mathbf{SI}^+|}{|\mathbf{SI}^-|})$;
8) **for** each $\mathbf{s} \in \mathbf{SI}$ {
9) $M \leftarrow$ ComputeM($\mathbf{s}$, $\mathbf{SI}$);
10) $\mathbf{s}.\tau \leftarrow \sqrt{\mathrm{AVG}_{i \in \{Dist^2(\mathbf{s}_i, \mathbf{s}) < M,\ \mathbf{s}_i.y \neq \mathbf{s}.y\}} (Dist^2(\mathbf{s}_i, \mathbf{s}))}$;
11) **if** $\mathbf{s} \in \mathbf{SI}^+$ **then** /* *a minority* */
12) $\mathbf{s}.\tau \leftarrow \sqrt{\eta_p} \times \mathbf{s}.\tau$;
13) **else** /* *a majority* */
14) $\mathbf{s}.\tau \leftarrow \sqrt{\eta_n} \times \mathbf{s}.\tau$;}
15) $D(\mathbf{x}) = \sum_{\mathbf{s} \in SI} \exp\left(-\frac{\frac{1}{k_{\mathbf{x}\mathbf{s}}} - 1}{\mathbf{s}.\tau^2} \right)$
16) **for** each k_{ij} in $\mathbf{K}$ {
17) $k_{ij} \leftarrow D(\mathbf{x}_i) \times D(\mathbf{x}_j) \times k_{ij}$;}
18) $\mathcal{C} \leftarrow$ SVMTrain(X_{train}, $\mathbf{K}$);
19) $\varepsilon_{\text{old}} \leftarrow \varepsilon_{\text{new}}$;
20) $\varepsilon_{\text{new}} \leftarrow$ SVMClassify(X_{test}, $\mathcal{C}$);
21) $t \leftarrow t + 1$;}
22) **return** $\mathcal{C}$;
End

Fig. 8. The KBA algorithm

5 Experimental results

We have conducted experiments on detecting suspicious events in a parking-lot setting to validate the effectiveness of our proposed methods. We recorded 1.5 h of video at parking lot 20 on the UCSB campus using two cameras. We collected trajectories depicting five motion patterns: *circling*, *zigzag pattern* or *M pattern*, *go-straight*, *back and forth*, and *parking*. We classified these events into benign and suspicious categories. The benign-event category consists of patterns *go-straight* and *parking*, and the suspicious-event category consists of the other three patterns. We are most interested in detecting suspicious events accurately. Specifically, we would like to answer the following three questions:

1. Can the use of the two-level Kalman filter successfully reconstruct motion patterns?
2. Can our sequence-data characterization and learning methods (in particular, the tensor product kernel) work effectively to fuse the degree of conformity of the symbolic summarizations and the similarity between the secondary descriptions?
3. Can KBA reduce the incidence of false negatives while maintaining a low incidence of false positives?

We use specificity and sensitivity as the evaluation criteria. We define the *sensitivity* of a learning algorithm as the ratio of the number of true positive (TP) predictions over the number of positive instances (TP+FN) in the test set, or Sensitivity = TP/(TP+FN). The *specificity* is defined as the ratio of the number of true negative (TN) predictions over the number of negative instances (TN+FP) in the test set. For surveillance applications, we care more about the sensitivity and at the same time hope that the specificity will not suffer too much from the other side.

Table 2 depicts the two datasets, balanced and skewed, that we used to conduct the experiments. The balanced dataset was produced from the recorded video. We then added synthetic trajectories to produce the skewed dataset. For each experiment, we chose 60% of the data as the training set and the remaining 40% as our testing data. We used PHMM for sequence alignment and selected an RBF function for $K_v(\mathbf{v}, \mathbf{v}')$ that works the best on the dataset. (The kernel and the parameter-selection processes are rather routine, so we do not report them here.) We employed the best parameter settings obtained through running a fivefold cross validation and report average class-prediction accuracy.

Here, we describe our experimental procedures on sensor registration and sensor data fusion and present some preliminary results.

Experiment $\#1$: *Spatial registration*.

After an initial, one-time camera calibration to determine the intrinsic camera parameters, we performed continuous spatial registration using Church's algorithm with three calibration points to update $\tilde{\mathbf{T}}_{\text{real}\leftarrow\text{world}}$. These three landmarks were preselected, so their positions in the world coordinate system were known. As mentioned before, Church's algorithm performs iterative nonlinear optimization for pose estimation. It usually took more iterations to establish an initial pose estimate (or this initial value could be estimated using a closed-form linear algorithm with more landmarks). For continuous pose update, movement of the camera (pan and tilt) between adjacent frames taken at the video frame rate (30 frames/s) was usually small. Hence, a good initial guess as to the current camera pose was that at the previous frame. The convergence was fast and our experiments indicated that the algorithm converged usually after a single iteration, or a few at most. The most time-consuming part of spatial registration was spent on tracking the registration landmarks. Because of the slow and predictable camera movement, the search area for the current location of a landmark could be quite small (in our experiments, we found the maximum displacement was less than eight pixels). Hence, we were able to perform spatial registration in real time.

The accuracy of spatial registration was affected by the precision of camera calibration and that of the tracking algorithm. To verify the accuracy of spatial registration, we used another known landmark (one that was not used in the algorithm) as a testing point. That is, we computed this new landmark's projected location in the image plane using the calibration matrix, $\tilde{\mathbf{T}}_{\text{real}\leftarrow\text{world}}$, established using Church's algorithm. We then compared this predicted position with that established through manual tracking. Figure 9 shows a sample error curve in this validation process for a 20-s video. The discrepancy in a landmark's predicted positions and those from manually tracking was computed once every 30 frames. As can be seen that the maximum error was less than two pixels, which is quite acceptable in our applications. Figure 10 shows yet another test of spatial registration: Three image frames with different spatial coverage were used with the transformation among them established using our algo-

Table 2. Datasets

Motion pattern	Balanced dataset # of instances	Skewed dataset # of instances
Circling	22	30
M pattern	19	22
Back and forth	38	40
Benign event	41	3,361

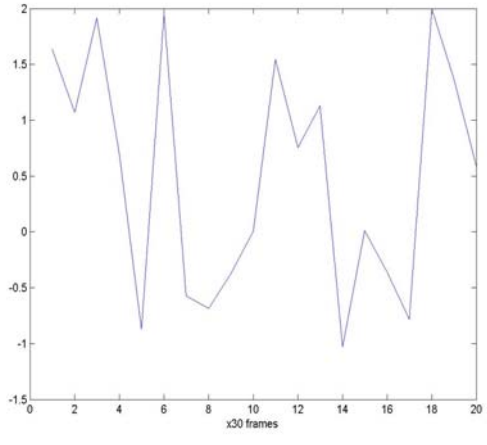


Fig. 9. Typical error in spatial registration. The *horizontal axis* is time. The *vertical axis* is the discrepancy in a landmark's predicted position (estimated using Church's algorithm) and its true location (established through manual tracking)

Fig. 10. Image mosaic. The *upper row* shows three input video frames of different spatial coverages. The *bottom row* shows the mosaic image constructed from the three input images that covers the whole parking lot

rithm. We then stitched these three frames together to obtain a mosaic image that covered the whole parking lot. This image mosaic provided a seamless background for the panning camera to aid the tracking process (discussed later).

Experiment $\#2$: *Temporal registration*.

Figure 11 shows the aligned invariant signatures of the same trajectory observed by two different cameras, the expression of which is given in Eq. 8 with $n = 0$. Although the observed trajectories from different cameras were distorted by the projection and image formation process and appeared different, their invariant signatures were quite similar to each other. Hence the invariant signatures could be used for aligning trajectories temporally.

Experiment $\#3$: *Sensor-data fusion*.

We use both the Kalman filter and the more general multihypothesis tracking algorithm for object tracking and sensor-data fusion. The running time of the Kalman filter was constant. Our experience also indicated that the precision of

a Camera 1 **b** Camera 2 **c** Invariant signature

Fig. 11. Using invariant signatures for temporal registration. **a** Trajectory as observed from camera 1. **b** Trajectory as observed from camera 2. **c** Invariant signatures, *green* for the trajectory from camera 1 and *red* for the trajectory from camera 2

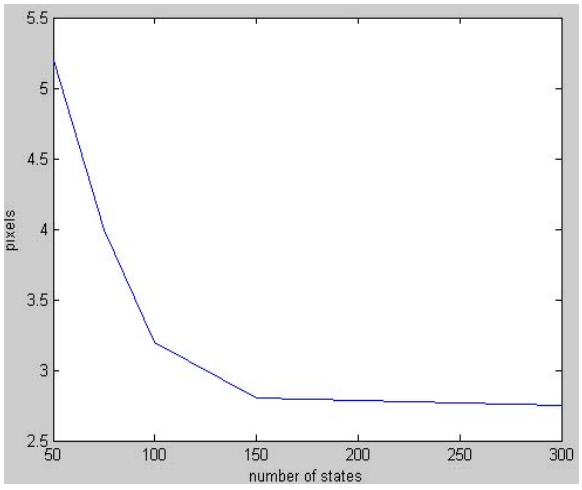


Fig. 12. Precision results of the multihypothesis tracking algorithm. The x-axis represents the number of states kept, and the y-axis represents the average discrepancy between multihypothesis tracking results and the ground truth (manual tracking)

the Kalman filter was acceptable, barring serious occlusion and scene clutter. For example, results of the Kalman filter tracking were within four pixels of the ground truth (manual tracking) in a short video of about 1 min during which a vehicle made a full circle in the parking lot. The Kalman filter tracking added only about 3% overhead in the running time (most time was spent on image processing).

The precision and running time of the multihypothesis tracking algorithm were determined by the number of candidate states kept. Figure 12 shows the tracking precision of the multihypothesis algorithm using different numbers of states, where the x-axis represents the number of states kept and the y-axis represents the average discrepancy between multihypothesis tracking results and the ground truth (manual tracking). It can be seen that the tracking precision increased quickly with the increase in the number of states used, but the effect tapered off when the number of states became large

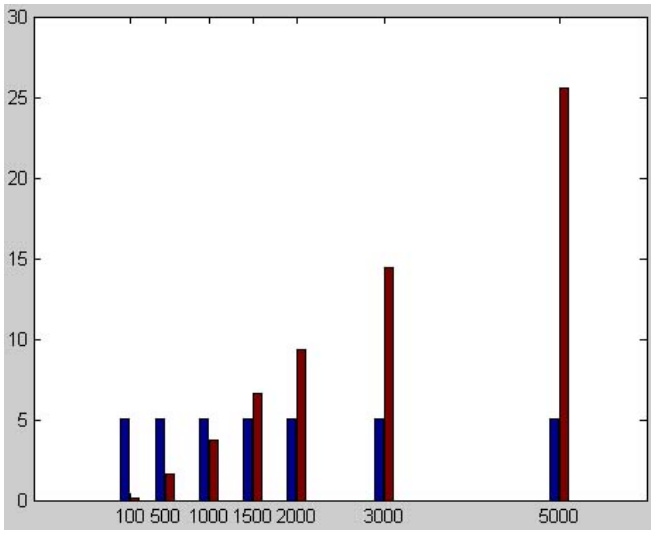


Fig. 13. Timing results of the multihypothesis tracking algorithm

(over 100 in our experiments). Figure 13 shows the overhead in keeping and updating multiple candidate states. In each group of bars, the first bar was the constant overhead in video processing, which did not change regardless of the number of states kept. The second bar was the overhead of the multihypothesis algorithm. From Fig. 13 we see that the overhead of keeping multiple candidate states was almost linear in the number of states used. Considering both the precision and the running time, using 150 states seemed to be a good compromise with an expected tracking error of about 2.8 pixels from the ground truth and a running time overhead of about 7%.

Experiment #4: *Feature respresentation*.
Figure 14 shows a sample parking pattern and its fused, invariant representation derived from multiple sensors. We used figure-background separation to detect moving objects. For a fixed camera, the background was treated as stationary. If the camera was moving, we registered and mapped time-varying video content into the mosaic image (e.g., Fig. 10) and then performed figure-background separation. Figures 14a,–c show raw vehicle trajectories from two cameras. Figure 14d is the fused and segmented trajectory and its semantic representation. Figure 14e depicts the θ, $|\mathbf{r}|$, and and $(\ddot{\mathbf{P}} \times \dot{\mathbf{P}})_z$ curves used in segmentation. The θ and $|\mathbf{r}|$ curves estimated from Kalman filter are shown in black, while the piecewise linear approximations of those using EM algorithm described earlier are shown in red. Vertical lines show the beginning and end of each segment. The result demonstrates that our tracking and segmentation algorithms worked properly.

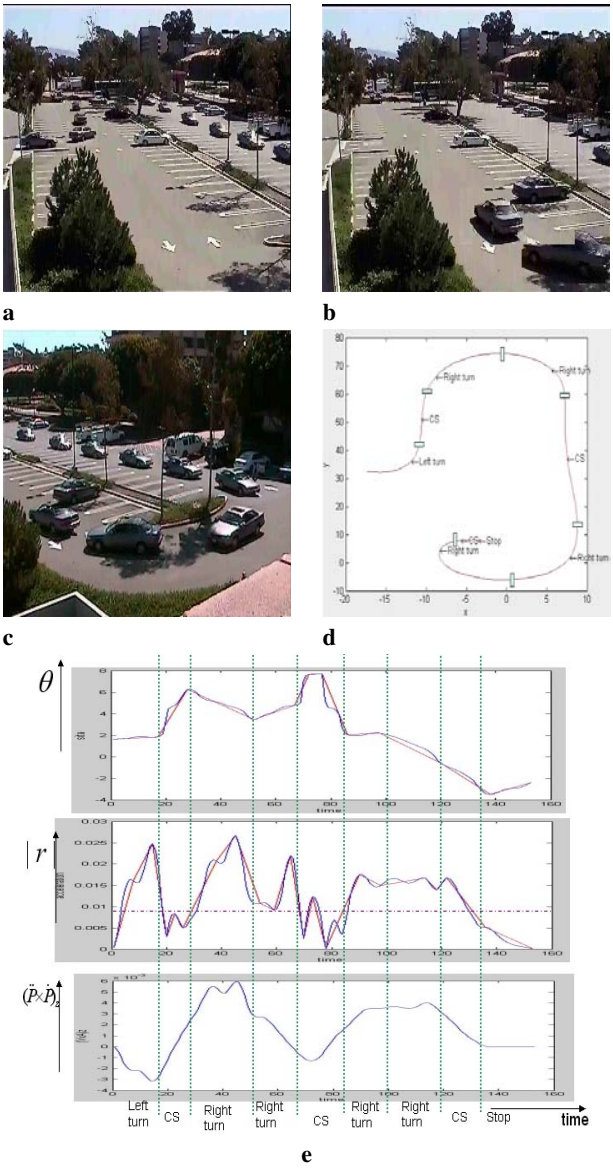


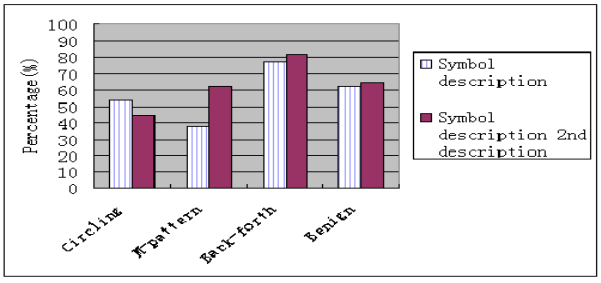
Fig. 14. A parking pattern. **a**, **b** Condensed video footage from left camera. **c** Condensed video footage from right camera. **d** Trajectory with segment boundaries and labels. **e** Acceleration curves used in segmentation

Experiment #5: *Sequence-alignment kernel evaluation*.
We used the balanced dataset to conduct this experiment. We compared the classification accuracy between when we used the primary segment symbols and when we also considered secondary description *velocity*. Figures 15a and b show that when the secondary structure was considered, both sensitivity and specificity were improved. The improvement is marked (about 6%) in sensitivity. In the remaining experiments, we thus considered both the primary and secondary information.

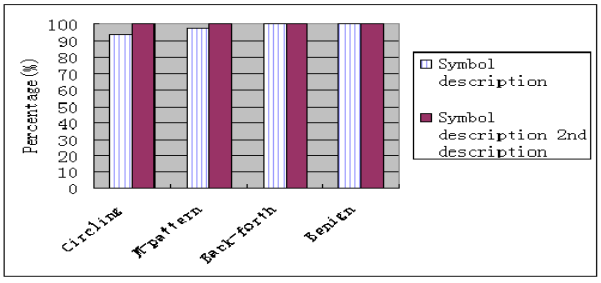
Experiment #6: KBA *evaluation*.
In this experiment, we examined the effectiveness of KBA on two datasets of different benign/suspicious ratios. The balanced dataset (second column in Table 2) has a benign/suspicious ratio of about 50%. Figures 15c and d show that the employment of KBA improves sensitivity significantly by 39%, whereas it degrades specificity by just 4%. Next, we repeated the KBA test on the skewed dataset (third column in Table 2), where the benign/suspicious ratio is less than 3%. Figures 15e and f show that the average sensitivity suffers a drop of 68% to 35%. After applying KBA, the average sensitivity improved to 70% by giving away just 3% in specificity.

6 Conclusions

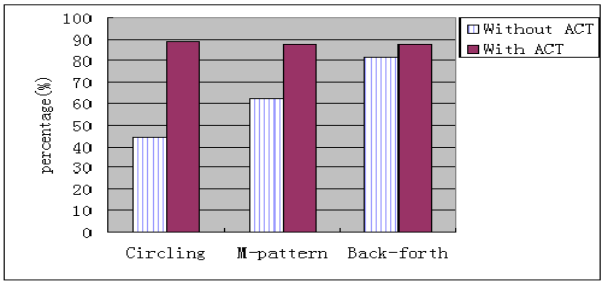
In this paper, we have presented methods for (1) fusing multicamera surveillance data, (2) characterizing motion patterns and their secondary structure, and (3) conducting statistical learning in an imbalanced-training-data setting for detecting rare events. For fusing multisource data from cameras with overlapping spatial and temporal coverage, we proposed using a two-level hierarchy of Kalman filters. For efficiently summarizing motion events, we studied hierarchical and invariant descriptors. For characterizing motion patterns, we proposed our sequence-alignment kernel, which uses tensor product to fuse a motion sequence's symbolic summarizations (e.g., left turn and right turn, which cannot be represented in a vector space) and its secondary numeric characteristics (e.g., velocity, which can be represented in a vector space). When the positive training instances (i.e., suspicious events) are significantly outnumbered by the negative training instances, we showed that kernel methods can suffer from high event-detection errors. To remedy this problem, we proposed an adaptive conformal transformation algorithm to work with



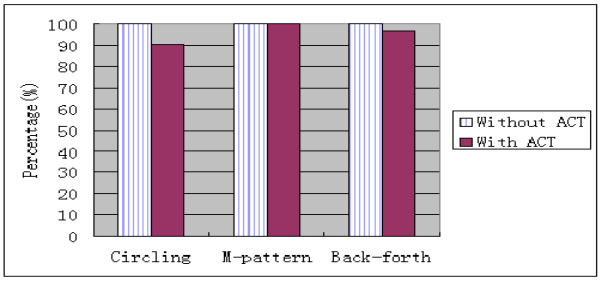
a Sensitivity (Kernel Test)



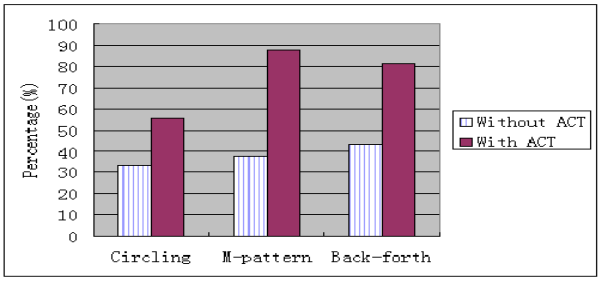
b Specificity (Kernel Test)



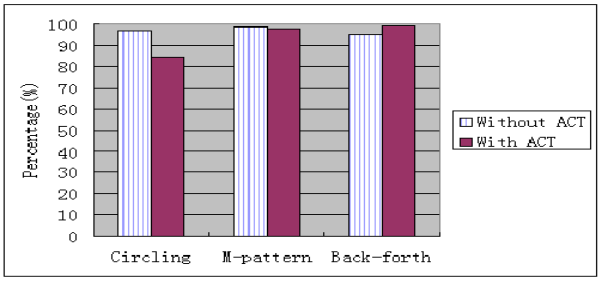
c Sensitivity (KBA Test – Balanced Data)



d Specificity (KBA Test – Balanced Data)



e Sensitivity (KBA Test – Skewed Data)



f Specificity (KBA Test – Skewed Data)

Fig. 15. Sensitivity and specificity of three test cases

our sequence-alignment kernel. Through extensive empirical study in a parking-lot surveillance setting, we showed that our system is highly effective in identifying suspicious events.

We are currently building a surveillance system with low-resolution Web cams and high-resolution zoom/tilt/pan cameras. We are particularly interested in testing the scalability of our multicamera fusion scheme (the hierarchical Kalman filter scheme) with respect to both the number of cameras and the number of objects that are simultaneously tracked. We also plan to investigate more robust parameter-tuning methods for enhancing our kernel-boundary-alignment (KBA) algorithm.

Acknowledgements. We would like to acknowledge the support of NSF Grants IIS-0133802 (CAREER), IIS-0219885, IIS-9908441, and EIA-0080134.

References

1. Amari S, Wu S (1999) Improving support vector machine classifiers by modifying kernel functions. Neural Netw 12(6):783–789
2. Azuma R (1995) Predicative tracking for augmented reality. University of North Carolina-Chapel Hill, Department of Computer Science, TR-95-007
3. Bengio Y (1998) Markovian models for sequential data. Neural Comput Surv 2:129–162
4. Boyd JE, Hunter E, Kelly PH, Tai LC, Phillips CB, Jain RC (1998) Mpi-video infrastructure for dynamic environments. In: IEEE international conference on multimedia systems '98, June 1998
5. Bozkaya T, Ozsoyoglu M (1997) Distanced-based indexing for high-dimensional metric spaces. In: Proc. ACM SIGMOD, pp 357–368
6. Brown RG (1983) Introduction to random signal analysis and Kalman filtering. Wiley, New York
7. Burden RL, Faires JD (eds) (1993) Numerical analysis, 5th edn. PWS, New York
8. Burges CJC (1999) Geometry and invariance in kernel based methods. In: Schlkopf B, Burges CJC, Smola AJ (eds) Advances in kernel methods: support vector learning. MIT Press, Cambridge, MA
9. Christin L, Eskin E, Cohen A, Westo J, Noble WS (2002) Mismatch string kernels for svm protein classification. Neural Inf Process Syst 15:1441–1448
10. Chudova D, Smyth P (2002) Pattern discovery in sequences under a markov assumption, ACM SIGKDD
11. Church E (1945) Revised geometry of the aerial photograph. Bull Aerial Photogrammetry 15
12. Cohn H (1980) Conformal mapping on Riemann surfaces. Dover, Mineola, NY

13. Collins RT, Lipton AJ (2000) A system for video surveillance and monitoring (vsam project final report). CMU Technical Report CMU-RI-TR-00-12
14. DeMenthon DF, Davis LS (1995) Model-based object pose in 25 lines of code. Int J Comput Vision 15:123–141
15. Duda RO, Hart RE, Stork DG (2001) Pattern classification, 2nd edn. Wiley, New York
16. Farin G (1997) Curves and surfaces for computer aided geometric design, 4th edn. Academic, San Diego
17. Faugeras O (1993) Three-dimensional computer vision. MIT Press, Cambridge, MA
18. Foley JD, van Dam A, Feiner SK, Hughes JF (1990) Computer graphics: principles and practice, 2nd edn. Addison-Wesley, Reading, MA
19. Fukunaga K (1990) Introduction to statistical pattern recognition, 2nd edn. Academic, Boston
20. Haralick R, Joo H, Lee C, Zhuang X, Vaidya V, Kim M (1989) Pose estimation from corresponding point data. IEEE Trans Syst Man Cybern 19:1426–46
21. Haykin S (1999) Neural networks, 2nd edn. Prentice Hall, Englewood Cliffs, NJ
22. Horaud R, Dornaika F, Lamiroy B, Christy S (1997) Object pose: the link between weak perspective, paraperspective and full perspective. Int J Comput Vision 22:173–189
23. Isard M, Blake A (1998) Condensation – conditional density propagation for visual tracking. Int J Comput Vision 29:5–28
24. Isard M, Blake A (1998) ICONDENSATION: Unifying low-level and high-level tracking in a stochastic framework. Lecture notes in computer science vol 1406. Springer, Berlin Heidelberg New York, pp 893–908. Int J Comput Vis 29(1):5–28
25. Jaakkola TS, Diekhans M, Haussler D (1999) Using the fisher kernel method to detect remote protein homologies. In: Proc. 7th international conference on intelligent system for molecular biology
26. Jaakkola TS, Haussler D (1998) Exploiting generative models in discriminative classifiers. In: Proceedings of the conference on advances in neural information processing systems II, pp 487–493
27. Jaakola TS, Haussler D (1999) Probabilistic kernel regression models. In: Conference on AI and statistics
28. Julier SJ, Uhlmann JK, Durrant-Whyte HF (1995) A new approach for filtering nonlinear systems. In: Proc. American Control Conference, Seattle
29. Kanatani K (1998) Optimal homography computation with a reliability measure. In: Proc. IAPR workshop on machine vision applications, November 1998
30. Kettnaker V, Zabih R (1999) Bayesian multi-camera surveillance. In: CVPR
31. Kitagawa G (1996) Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. J Comput Graph Stat 5:1–25
32. Lee L, Romano R, Stein G (2000) Monitoring activities from multiple video streams: establishing a common coordinate system. IEEE Trans PAMI 22(8):758–767
33. Leslie C, Eskin E, Noble WS (2002) The spectrum kernel: a string kernel for svm protein classification. In: Proc. Pacific symposium on biocomputing. World Scientific, Singapore
34. Lou J, Yang H, Hu W, Tan T (2002) Visual vehicle tracking using an improved ekf. In: ACCV
35. Maybank SJ, Worrall AD, Sullivan GD (1996) Filter for car tracking based on acceleration and steering angle. In: Proc. British Machine Vision Conference
36. Maybeck PS (1979) Stochastic models, estimation, and control, vol 1. Academic, New York
37. Pavlidis I, Morellas V (2001) Two examples of indoor and outdoor surveillance systems: motivation, design, and testing. In: Proc. 2nd European workshop on advanced video-based surveillance
38. Pavlidis I, Morellas V, Tsiamyrtzis P, Harp S (2001) Urban surveillance systems: from the laboratory to the commercial world. Proc IEEE 89(10):1478–1497
39. Pitt MK, Shephard N (1999) Filtering via simulation: Auxiliary particle filters. J Am Stat Assoc 94:590–599
40. Regazzoni C, Varshney PK (2002) Multisensor surveillance systems based on image and video data. In: Proc. IEEE conference on image processing
41. Sears FW (1958) Optics, 3rd edn. Addison-Wesley, Reading, MA
42. Starner T, Pentland A (1994) Visual recognition of American sign language using hidden Markov models. Technical Report Master's thesis, MIT, February 1995, Program in Media Arts & Sciences, MIT Media Laboratory. Also Media Lab VISMOD TR 316. http://www-white.media.mit.edu/vismod/people/publications/publications.html [860 Kb]
43. Struik DJ (1961) Differential geometry, 2nd edn. Addison-Wesley, Reading, MA
44. Vapnik V (1995) The Nature of statistical learning theory. Springer, Berlin Heidelberg New York
45. Watkins C (1999) Dynamic alignment kernels. Technical Report, Department of Computer Science, University of London
46. Welch G, Bishop G (2002) http://www.cs.unc.edu/welch/kalman
47. Welch G, Bishop G (2002) An introduction to the Kalman filter. University of North Carolina-Chapel Hill, TR 95-041
48. Wu G, Chang E Adaptive feature-space conformal transformation for learning imbalanced data. In: International conference on machine learning (ICML), August 2003
49. Xu G, Zhang Z (1996) Epipolar geometry in stereo, motion and object recognition. Kluwer, Dordrecht
50. Wu Y, Jiao L, Wu G, Wang YF, Chang E (2003) Invaraint feature extraction and biased statistical inference for video surveillance. In: Proc. IEEE conference on advanced video and signal-based surveillance, Miami, FL
51. Zhang Z (2000) A flexible new technique for camera calibration. IEEE Trans Pattern Anal Mach Intell 22:1330–4