Multi-camera Spatio-temporal Fusion and Biased Sequence-data Learning for Security Surveillance

Gang Wu, Yi Wu, Long Jiao, Yuan-Fang Wang, Edward Chang
Electrical & Computer Engineering and Computer Science
University of California, Santa Barbara, CA 93106
echang@ece.ucsb.edu

ABSTRACT

In this paper, we propose a framework for multi-camera video surveillance. *Our framework addresses the detection, represenation, and recognition of motion events. The detection phase handles spatio-temporal data fusion for efficiently and reliably extracting motion trajectories from video. The representation phase summarizes raw trajectory data to construct a hierarchical, invariant, and content-rich representation of the motion events. Finally, the recognition phase deals with learning using imbalanced training datasets and infinite-dimensional data that also exhibit temporal ordering.* Due to space limit, only the following two components are discussed in the paper: fusing spatio-temporal information from multiple camera sources and characterizing and detecting suspicious surveillance events.

For fusing multi-source data from cameras with overlapping spatial and temporal coverage, we propose a novel method that uses a two-level hierarchy of Kalman filters to construct coherent and invariant event descriptors. Once the event descriptors are constructed, we devise a sequence-alignment kernel function to perform sequence data learning for detecting suspicious events. When the positive training instances (i.e., suspicious events) are significantly outnumbered by the negative training instances, SVMs can suffer from high event-detection errors. To remedy this problem, we propose an adaptive conformal transformation algorithm to work with the sequence-alignment kernel. Through empirical study in a parking-lot surveillance setting, we show that our spatio-temporal fusion scheme can efficiently and reliably reconstruct scene activities, and that our learning method is highly effective in identifying suspicious events.

1. INTRODUCTION

United States policymakers, especially in security and intelligence services, are increasingly turning toward video surveillance as a means to combat terrorist threats and a response to the public's demand for security. With the proliferation of inexpensive cameras and the availability of high-speed, broad-band network, it has become economically and technically feasible to deploy a large number of cameras for outdoor security surveillance. However, several important research questions must be addressed before we can rely upon video surveillance as an effective tool for crime prevention.

In this paper, we propose a framework for multi-camera video surveillance. Our framework addresses the detection, representation, and recogntion of motion events. The detection phase handles spatio-temporal data fusion for efficiently and reliably extracting motion trajectories from video. The representation phase summarizes raw trajectory data to construct a hierarchical, invariant, and content-rich representation of the motion events. Finally, the recognition phase deals with learning using imbalanced training datasets and infinite-dimensional data that also exhibit temporal ordering. Due to space limit, we will focus our discussion on the following core research problems of building a multi-camera surveillance system: namely, spatio-temporal data fusion, and event characterization and detection.

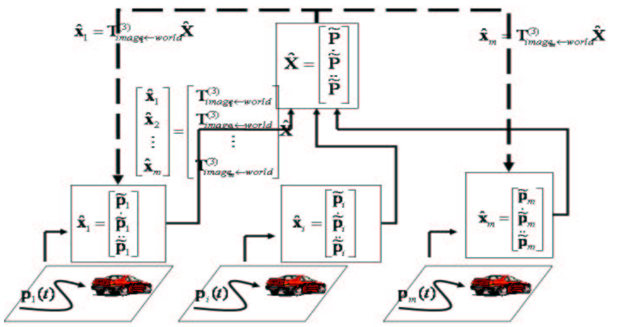
- *Spatio-temporal data fusion*. Objects observed from multiple cameras should be integrated to build spatio-temporal patterns. Such integration must handle spatial occlusion and temporal shift (e.g., camera recording without precise timing information and with different frame rates). In addition, a motion pattern should not be affected by varying camera poses and incidental environmental factors that can alter object appearance. We formulate the multi-source data-fusion solution as a two-level hierarchy of Kalman filters. At the base level of the hierarchy, each Kalman filter estimates, independently, the position, velocity, and acceleration of the target object in the *local camera reference frame*. At the top level, we use one Kalman filter to register the position, velocity, and acceleration of the vehicle in the *global camera reference frame*. An important feature in this hierarchical function is that we allow both bottom-up data fusion and top-down information dissemination to improve the robustness of the solution. We present the details in Section 2.
- *Event characterization and detection*. Event detecion 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 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 towards the minority class and becomes very sensitive to noise. (An example is presented in Section 3.3 to illustrate this problem.) For effective sequence-data matching, we first discretize a continuous sequence. We show that our sequence-alignment kernel is a legitimate *kernel function* to be used with SVMs. For tackling the imbalanced training-data problem, we propose an adaptive conformal transformation (ACT) algorithm, which conformally and adaptively spread the area around the class-boundary outward on the Riemannian manifold where all mapped data are located. We present the details in Section 3.

One particular application scenario we utilized to evaluate our algorithms is detecting suspicious activities in a parking lot. Our empirical study shows that our spatio-temporal fusion scheme can efficiently and reliably reconstruct scene activities even when individual cameras may have spatial or temporal lapses, and that our sequence-alignment kernel and ACT algorithm are highly effective in identifying suspicious events.

The rest of the paper is organized as follows. Section 2 presents the hierarchy of Kalman filters for fusing spatio-temporal data. Section 3 presents event characterization and detection methods. Section 4 presents empirical results. In Section **??** we discuss related work. Finally, in Section 5 we offer concluding remarks.

2. MULTI-SOURCE FUSION

We use the Kalman filter [2, 12] as the tool for fusing information spatially and temporally from multiple cameras. 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 novel contribution is in using two-level Kalman filters to fuse data from multiple sources.



$\hat{\mathbf{x}}_1 = \mathbf{T}^{(3)}_{image_1 \leftarrow world}\hat{\mathbf{X}}$
$\hat{\mathbf{X}} = [\tilde{\mathbf{P}}, \dot{\tilde{\mathbf{P}}}, \ddot{\tilde{\mathbf{P}}}]$
$\hat{\mathbf{x}}_m = \mathbf{T}^{(3)}_{image_m \leftarrow world}\hat{\mathbf{X}}$
$[\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \vdots, \hat{\mathbf{x}}_m] = [\mathbf{T}^{(3)}_{image_1 \leftarrow world}, \mathbf{T}^{(3)}_{image_2 \leftarrow world}, \vdots, \mathbf{T}^{(3)}_{image_m \leftarrow world}]\hat{\mathbf{X}}$
$\hat{\mathbf{x}}_1 = [\tilde{\mathbf{p}}_1, \dot{\tilde{\mathbf{p}}}_1, \ddot{\tilde{\mathbf{p}}}_1]$
$\hat{\mathbf{x}}_i = [\tilde{\mathbf{p}}_i, \dot{\tilde{\mathbf{p}}}_i, \ddot{\tilde{\mathbf{p}}}_i]$
$\hat{\mathbf{x}}_m = [\tilde{\mathbf{p}}_m, \dot{\tilde{\mathbf{p}}}_m, \ddot{\tilde{\mathbf{p}}}_m]$
$\mathbf{p}_1(t)$
$\mathbf{p}_i(t)$
$\mathbf{p}_m(t)$

Figure 1: Two-level hierarchical Kalman filter configuration

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 [2, 12]. 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 [12]. 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 run time. Work has also been done on relaxing some of the assumptions, such as the Gaussian noise assumption and the linearity assumption [11].

Suppose that a vehicle is moving in the parking lot, whose 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).[1] 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. 1, 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 the 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 (the solid feed-upward lines in Fig. 1). This is possible because camera calibration and registration [3, 7, 10, 14, 15] are used for deriving the transform matrix $\mathbf{T}_{image_i \leftarrow world}$. This matrix allows $\mathbf{p}_i$, measured in the reference frame of an individual camera, to be related to $\mathbf{P}$ in the global world system. We also allow dissemination of fused information to individual cameras (the dashed feed-downward lines in Fig. 1) to help to guide image processing.

3. EVENT REPRESENTATION AND EVENT DETECTION

In this section, we first depict how an event is characterized. We then design a sequence-alignment kernel function to work with SVMs for event detection. Finally, we propose using adaptive conformal transformation, which adaptively modifies the resolution in the metric space to deal with the imbalanced training-data problem.

3.1 Event Descriptors

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. 1), 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 [6] and computer graphics [8], 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 $t_o = t_{min}, t_k = t_{max}$

$$\ddot{\mathbf{P}}(t) = \mathbf{r}(t)e^{i\theta(t)}, where$$

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

We employ an iterative expectation and maximization (EM) algorithm [5] to segment trajectories. The EM algorithm consists of two stages: (1) The E-stage hypothesizes the number of segments and 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 from the E-stage. These two steps iterate until the solution converges. Table 1 sketches the pseudo-code of the algorithm (using fitting $\theta(t)$ as an illustration).

[1]There might be multiple moving vehicles in a busy parking lot, and it may be difficult to synchronize the activities observed in multiple cameras. The question is then how we disambiguate the correspondence of multiple trajectories both spatially and temporally. Spatial and temporal trajectory correspondence can be established through the camera registration and stereopsis correspondence processes [3, 4, 7, 9, 10, 13, 14, 15], which are well established techniques in photogrammetry and computer vision. For our discussion, we will assume that these problems can be solved and we can achieve spatial and temporal registration of vehicle trajectories.

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

$$(\theta_{max} - \theta_{min})t + (t_{min} - t_{max})\theta + (\theta_{min} - \theta_{max})t_{min} + (t_{max} - t_{min})\theta_{min} = 0 \tag{2}$$

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

$$t_{maxdev} = argmax_t|(\theta_{max} - \theta_{min})t + (t_{min} - t_{max})\theta(t) + (\theta_{min} - \theta_{max})t_{min} + (t_{max} - t_{min})\theta_{min}|/\Delta \tag{3}$$

$$maxdev = |(\theta_{max} - \theta_{min})t_{maxdev} + (t_{min} - t_{max})\theta(t_{maxdev}) + (\theta_{min} - \theta_{max})t_{min} + (t_{max} - t_{min})\theta_{min}|/\Delta \tag{4}$$

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 $[t_{min}, t_{maxdev})$ and $[t_{maxdev}, t_{max})$ and repeat the first two steps using these two new intervals.

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. 1) 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.

3.2 Sequence Alignment Learning

In the previous section, we have 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, e.g., 'R' represents "RightTurn," 'L' represents "LeftTurn," etc. 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, which 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 i^{th} 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 taken place 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. For this purpose, we construct a new *sequence-alignment kernel* that can be applied to measure pair-wise similarity between sequences with secondary variables.

3.2.1 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 *tensor product*. These are explained below.

Let $\mathbf{x} \in \mathcal{X}$ be a composite structure and $x_1, ..., x_N$ 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 each 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* $\mathbf{x} = (x_1, ..., x_N) \in \mathcal{X}$ *and* $\mathbf{x}' = (x'_1, ..., x'_N) \in \mathcal{X}$. *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 [**?**], it is easy to see that the tensor product kernel is positive definite if each individual kernel is positive definite.

3.2.2 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 fuse 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) [**?**], a generative probability model, to model the joint p.d. of two symbol sequences. (Notice that PHMM is different from HMM, which aims to model the *evolution* of individual sequence data.)

A realization of PHMM is a sequence of states, starting with **START** and finishing with **END**; and in between there are three possible states: 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 length. 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

$$(K_s \otimes K_v)((\mathbf{s}, \mathbf{v}), (\mathbf{s}', \mathbf{v}')) = K_s(\mathbf{s}, \mathbf{s}')K_v(\mathbf{v}, \mathbf{v}'). \tag{5}$$

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

Example 1:

$\mathbf{s}$:	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	

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

- **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:

$$\begin{array}{ccccccccc} C & D & U & C & L & R & R & L & - \\ C & - & U & C & L & R & - & L & C \end{array}$$

- **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.

$$\begin{array}{ccccccccc} 0.7 & 0.5 & 0.8 & 0.8 & 0.7 & 0.8 & 0.6 & 0.5 & \mathbf{0.0} \\ 0.5 & \mathbf{0.0} & 0.4 & 0.4 & 0.5 & 0.6 & \mathbf{0.0} & 0.6 & 0.6 \end{array}$$

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

There are three advantages of 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 Section **??**), our sequence-alignment kernel adds the ability to learn from negative training instances, as well from positive training instances.

3.3 Imbalanced Learning via Adaptive Conformal Transformation

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 [**?**]. 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.

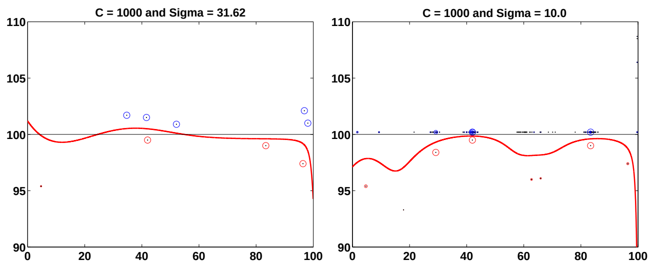
Figure 2 exhibits the boundary distortion between the two left quadrants in the checkerboard under two different negative/positive training-data ratios, where a black dot with a circle represents a support vector, and its radius represents the weight value α_i of the support vector. The bigger the circle, the larger the α_i. Figure 2(a) 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 2(b) shows the boundary when the ratio increases to $10,000 : 1$. The boundary in Figure 2(b) is much more skewed towards the positive quadrant than the boundary in Figure 2(a), and hence causes a higher incidence of false negatives.

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 outweights that of mispredicting a negative event (a false positive). It is well known that in a binary classification problem, Bayesian risks are defined as:

$$\begin{aligned} R(\alpha_p|\mathbf{x}) &= \lambda_{pp} P(\omega_p|\mathbf{x}) + \lambda_{pn} P(\omega_n|\mathbf{x}) \\ R(\alpha_n|\mathbf{x}) &= \lambda_{np} P(\omega_p|\mathbf{x}) + \lambda_{nn} P(\omega_n|\mathbf{x}) \end{aligned} \qquad (6)$$

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} 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 [**?**] deals with the problem in a vector space (Section 3.3.1). We then present our solution to sequence-data learning where a discretized variable-length sequence may not have a vector-space representation (Section 3.3.2).



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

Figure 2: Boundaries of Different Ratios.

3.3.1 Conformal Transformation in a Vector Space

In [**?**], we proposed feature-space adaptive conformal transformation (ACT) for imbalance-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 based on input-space distance proposed by [1]) can remedy the imbalance-data learning problem.

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

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

$$g_{ij}(\mathbf{x}) = \frac{1}{2}\frac{\partial^2 K(\mathbf{x}, \mathbf{x})}{\partial x_i \partial x_j} - \left(\frac{\partial^2 K(\mathbf{x}, \mathbf{x}')}{\partial x_i' \partial x_j'}\right)_{\mathbf{x}=\mathbf{x}'}. \quad (7)$$

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. 7, so that the spatial relationship between the data would not be affected much [1]. Such a conformal transformation can be depicted as

$$\tilde{K}(\mathbf{x}, \mathbf{x}') = D(\mathbf{x})D(\mathbf{x}')K(\mathbf{x}, \mathbf{x}'), \quad (8)$$

where $D(x)$ is a properly defined positive conformal function. $D(\mathbf{x})$ should be chosen in a way such that the new Remannian 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, we magnify $\tilde{g}_{ij}(\mathbf{x})$ more in the boundary area close to the minority class. An RBF distance function such as

$$D(\mathbf{x}) = \sum_{k \in SV} \exp(-\frac{|\mathbf{x} - \mathbf{x}_k|}{\tau_k^2}). \quad (9)$$

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

To reflect the spatial distribution of the support vectors in F, we hope that the τ_k (the magnification factor in the neighborhood of support vector x_k) would be smaller for the area in feature space F where the support vectors are dense, so as to get a larger magnification metric $\tilde{g}_{ij}(\mathbf{x})$; otherwise τ_k should be larger for the area in F where the support vectors are scarce. In this way, τ_k can be chosen as

$$\tau_k^2 = AVG_{i \in \{\|\Phi(\mathbf{x}_i) - \Phi(\mathbf{x}_k)\|^2 < M,\ y_i \neq y_k\}} \left(\|\Phi(\mathbf{x}_i) - \Phi(\mathbf{x}_k)\|^2\right), \quad (10)$$

where the average comprises all support vectors falling into its neighborhood with the radius of M but having different class labels with $\Phi(\mathbf{x}_i)$. Here, M is the average distance of the nearest and the farthest support vector from $\mathbf{x}_k$. Choosing 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:

$$\|\Phi(\mathbf{x}_i) - \Phi(\mathbf{x}_k)\|^2 = K(\mathbf{x}_i, \mathbf{x}_i) + K(\mathbf{x}_k, \mathbf{x}_k) - 2K(\mathbf{x}_i, \mathbf{x}_k). \quad (11)$$

Substituting Eq. 11 into Eq. 10, we can then calculate the τ_k 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 towards the minority class in the input space I. In this situation, the minority support vectors are located more closely to the class-boundary than the majority ones in I. We then 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 coefficient for the τ_k in Eq. 10 to reflect the boundary skew in $D(\mathbf{x})$. We choose the square of $\tilde{\tau}_k$ as $\eta_p \tau_k^2$ if $\mathbf{x}_k$ is a minority support vector, otherwise it is as $\eta_n \tau_k^2$. We empirically demonstrate that η_p and η_n are proportional to the skew of support vectors, or η_p as $O(\frac{|SV+|}{|SV-|})$, and η_n as $O(\frac{|SV-|}{|SV+|})$, where $|SV+|$ and $|SV-|$ denote the number of minority and majority support vectors, respectively. Please refer to [?, ?] for details.

3.3.2 Conformal Transformation in a Metric Space

The sequence data do not reside in a Euclidean vector space. In this particular situation, we cannot directly apply our adaptive conformal transformation method for sequence-data learning. Fortunately, we can embed them into a metric space via the sequence-alignment kernel $K_s \otimes K_v$ we constructed in Section 3.2. We can then apply the idea of adaptive conformal transformation by modifying the metric distance in this space.

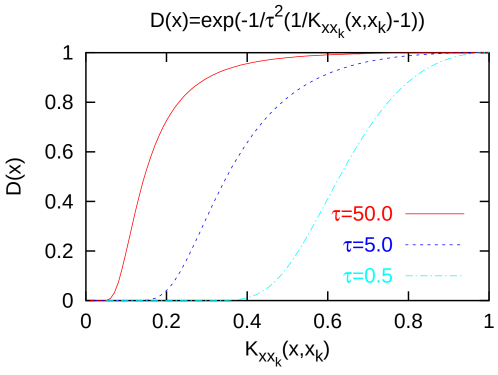


Figure 3: $D(\mathbf{x})$ with Different $\frac{1}{\tau^2}$.

Since $\mathbf{x}$ is a sequence instance, we cannot choose $D(\mathbf{x})$ as in Eq. 9. It is impossible to calculate the Euclidean distance $|\mathbf{x} - \mathbf{x}_i|$. In Section 3.3.2, 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 vectors. In other words, if a training sample $\mathbf{x}'$ is similar to a support sequence[2] $\mathbf{x}$, its embedded point via $K_{xx'}$ is close to the support vector embedded by the support sequence $\mathbf{x}$ (or in its neighborhood), $D(\mathbf{x})$ then should be larger so as to achieve a greater magnification. Since our sequence-alignment kernel $K_{xx'}(\mathbf{x}, \mathbf{x}')$ models the similarity between the sequence data $\mathbf{x}$ and $\mathbf{x}'$, we can assume that their similarity represents their distance in a metric space (and vice versa). Therefore, we choose $D(\mathbf{x})$ as

$$D(\mathbf{x}) = \sum_{k \in SQ} \exp\left(-\frac{\left(\frac{1}{K_{\mathbf{x}\mathbf{x}_k}(\mathbf{x}, \mathbf{x}_k)} - 1\right)^2}{\tau_k^2}\right), \quad (12)$$

where SQ denotes the support sequence set, and τ_k controls the magnitude of $D(\mathbf{x})$. Figure 3 illustrates a $D(x)$ for a given support sequence $\mathbf{x}_k$, where we can see $D(x)$ becomes larger when a sequence $\mathbf{x}$ is more similar to $\mathbf{x}_k$ (a larger K_{xx_k}), and is shaped very differently with different τ_k. We thus adaptively choose τ_k as

$$\tau_k^2 = AVG_{i \in \{Dist(\mathbf{x}_i - \mathbf{x}_k) < M,\ y_i \neq y_k\}} Dist(\mathbf{x}_i - \mathbf{x}_k). \quad (13)$$

In the above equation, the distance $Dist(\mathbf{x}_i - \mathbf{x}_k)$ between two sequence data $\mathbf{x}_i$ and $\mathbf{x}_k$ is calculated via the kernel trick as

$$Dist(\mathbf{x}_i - \mathbf{x}_k) = K_{x_i x_k}(\mathbf{x}_i, \mathbf{x}_i) + K_{x_i x_k}(\mathbf{x}_k, \mathbf{x}_k) - 2 * K_{x_i x_k}(\mathbf{x}_i, \mathbf{x}_k). \quad (14)$$

The threshold M is chosen as the average of the minimal distance $Dist_{min}$ and the maximal distance $Dist_{max}$. In addition, τ_k is scaled in the same way as we did in Section 3.3.1 for dealing with the imbalanced training-data problem.

Figure 4 summarizes the ACT algorithm in sequence-data learning. We apply ACT on the training dataset X_{train} until the testing accuracy on X_{test} cannot be further improved. In each iteration, ACT adaptively calculates τ_k^2 for each support sequence (step 9), based on the distribution of support vectors embedded by support sequences in feature space F. ACT scales the τ_k^2 according to the negative-to-positive support-sequence ratio (steps 10 to 13). Finally, ACT updates the kernel matrix and performs retraining on X_{train} (steps 14 to 17).

[2]Since a training sample $\mathbf{x}$ is actually not a vector, but a discrete sequence, we call it a *support sequence* if its embedded point via $K_{xx'} = K_x \otimes K_{x'}$ is a support vector in the metric space.

```
Input:
  $X_{train}$, $X_{test}$, $\theta$, $\mathbf{K}$;
Output:
  $\mathcal{C}$; /* output classifier */
Variables:
  $\mathbf{SQ}$; /* support sequence set */
  $M$; /* distance threshold */
  $\mathbf{s}$; /* a support sequence */
  $\mathbf{s}.\tau$; /* parameter of $\mathbf{s}$ */
  $\mathbf{s}.y$; /* class label of $\mathbf{s}$ */
Function Calls:
  SVMTrain($X_{train}$, $\mathbf{K}$); /* train SVM classifier $\mathcal{C}$ */
  SVMClassify($X_{test}, \mathcal{C}$); /* classify $X_{test}$ by $\mathcal{C}$ */
  ExtractSQ($\mathcal{C}$);            /* obtain SQ from $\mathcal{C}$ */
  ComputeThresh($\mathcal{C}$);      /* compute distance threshold */

Begin
1)  $\mathcal{C} \leftarrow$ SVMTrain($X_{train}$, $\mathbf{K}$);
2)  $\varepsilon_{old} \leftarrow \infty$;
3)  $\varepsilon_{new} \leftarrow$ SVMClassify($X_{test}, \mathcal{C}$);
4)  while ($\varepsilon_{new} - \varepsilon_{old} > \theta$)
5)    SQ←ExtractSQ($\mathcal{C}$);
6)    $\eta_p \leftarrow O(\frac{|SQ^-|}{|SQ^+|})$, $\eta_n \leftarrow O(\frac{|SQ^+|}{|SQ^-|})$;
7)    for each $\mathbf{s} \in \mathbf{SQ}$
8)    $M$ ←ComputeThresh($\mathbf{s}$, $\mathbf{SQ}$);
9)      $\mathbf{s}.\tau \leftarrow sqrt(AVG_{i \in \{Dist(\mathbf{s}_i - \mathbf{s}) < M,\ \mathbf{s}_i.y \neq \mathbf{s}.y\}} Dist(\mathbf{s}_i - \mathbf{s}))$;
10)     if $\mathbf{s} \in \mathbf{SQ}^+$ then /* a minority support sequence */
11)       $\mathbf{s}.\tau \leftarrow \sqrt{\eta_p} \times \mathbf{s}.\tau$;
12)     else /* a majority */
13)       $\mathbf{s}.\tau \leftarrow \sqrt{\eta_n} \times \mathbf{s}.\tau$;
14)   $D(\mathbf{x}) = \sum_{\mathbf{s} \in SQ} \exp\left(-\frac{\left(\frac{1}{K(\mathbf{x},\mathbf{x}_k)}-1\right)^2}{\mathbf{s}.\tau_k^2}\right)$
15)   for each $\mathbf{K}_{ij}$ in $\mathbf{K}$
16)     $\mathbf{K}_{ij} \leftarrow D(\mathbf{x}_i) \times D(\mathbf{x}_j) \times \mathbf{K}_{ij}$;
17)   $\mathcal{C} \leftarrow$ SVMTrain($X_{train}$,$\mathbf{K}$);
18)   $\varepsilon_{old} \leftarrow \varepsilon_{new}$;
19)   $\varepsilon_{new} \leftarrow$ SVMClassify($X_{test}, \mathcal{C}$);
20) return $\mathcal{C}$;
End
```

Figure 4: ACT Algorithm for Learning Sequence Data.

4. 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 one hour and a half's video at parking lot-20 on 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 classify these events into the 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 event 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 comformity of the symbolic symmarizations and the similarity beween the secondary descriptions?

3. Can ACT reduce the incidence of false negatives while maintaining 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, hopefully the specificity will not suffer too much from the other side.

Table 2 depicts the two datasets, a balanced and a skewed dataset, which we used to conduct the experiments. The balanced dataset was produced from the recorded video. We then added synthetic trajaecories 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 five-fold cross validation, and report average class-prediction accuracy.

$Motion\ Pattern$	*Balanced Data Set* $\#\ of\ Instances$	*Skewed Data Set* $\#\ of\ Instances$
$Circling$	22	30
$M-pattern$	19	22
$Back-and-forth$	38	40
$Benign\ event$	41	$3,361$

Table 2: Datasets.

Experiment $\#1$: *Kalman filter evaluation.*
For this experiment, two cameras were used to record the activities in the parking lot. Sample images for a circling pattern are shown in Fig. 5(a) and (b).[3] We employed a simple mechanism for figure-background separation. As in our current experiment the camera aims were fixed, we detected the presence of moving objects by performing a simple difference operation between adjacent video frames. We then extracted the moving objects by another difference operation with an adjacent video frame with no motion. The Kalman filter was used to track the moving vehicles. It helped in smoothing the trajectories, fusing the trajectories from the two cameras, and providing velocity and acceleration estimates from the raw trajectories. Fig. 5(c) shows the fused raw vehicle trajectories from the two cameras. Sample raw and filtered vehicle trajectories are shown in Fig. 5(d) where the black (dark) curve is the raw vehicle trajectory and the red (light) curve is the Kalman filtered and fused trajectory. The agreement of the two curves demonstrates the effectiveness of our fusion and trajectory reconstruction method.
For trajectory segmentation, we imposed the piecewise linearity constraint on both $|\mathbf{r}|$ and θ of the acceleration curve after the trajectory was segmented into two types: where $|\mathbf{r}| > 0$, and where $|\mathbf{r}| \approx 0$. In our experiment, the threshold for $|\mathbf{r}|$ to be considered roughly zero was 0.9. (This level is indicated as the horizontal dashed line in Figs. 5(e) and 6(c).)
In Fig. 5 (d) and (e), we show the sample results of segmenting a circling pattern. Fig. 5(e) depicts the $|\mathbf{r}|$, θ, and $(\ddot{\mathbf{P}} \times \dot{\mathbf{P}})_z$ curves used in segmentation. The θ and $|\mathbf{r}|$ trajectories estimated from the Kalman filter are shown in black while the piecewise linear approximations of these curves using the EM algorithm described before are shown in red. Vertical lines show the begin and end of each segment. For illustration, the boundaries between adjacent segments and the segment labels are shown in Fig. 5(d) as well.
Fig. 6 shows another result of segmenting an M-pattern. Fig. 6(a) depicts the raw, condensed footage from the left camera only. Fig. 6(b) depicts the raw (in black) and the Kalman filtered (in red) trajectories, and (c) the $|\mathbf{r}|$, θ, and $(\ddot{\mathbf{P}} \times \dot{\mathbf{P}})_z$ curves used in segmentation. The segment boundaries and labels are superimposed on Fig. 6(b). As can be seen from Figs. 5 and 6, the Kalman filter was able to smooth the noisy raw trajectories and arrived at reasonable velocity

[3]To conserve space and to better illustrate the motion trajectories, we superimposed multiple video frames into a single picture for display.

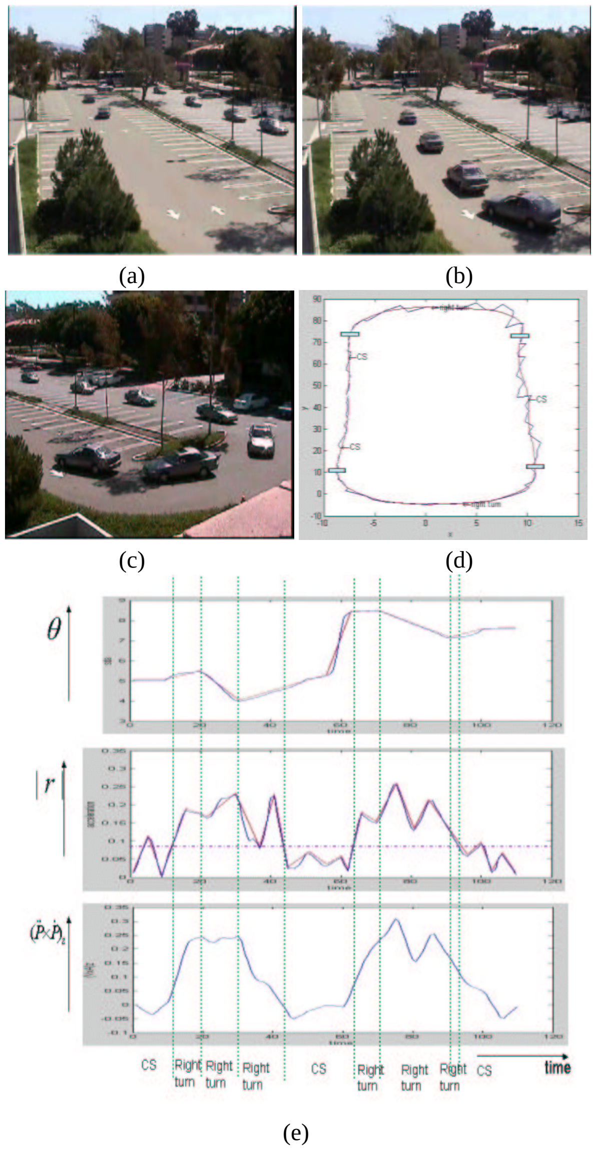


Figure 5: A Circling Pattern. (a) and (b) condensed video footages from the left camera, (c) condensed video footage from the right camera, (d) raw (black or dark) and the Kalman filtered (red or light) trajectories with segment boundaries and labels, and (e) acceleration curves used in segmentation.

and acceleration estimates. And our EM segmentation algorithm was able to segment the trajectories into pieces that conformed to the intuitive notion of a human observer. These results demonstrate that our tracking and segmentation algorithms work correctly.

Experiment $\#2$: *Sequence-alignment kernel evaluation*.
We used the balanced dataset to conduct this experiment. We compared the classification accuracy between when we only have the primary segment symbols and when we also take secondary description *velocity* into consideration. Figures 7(a) and 7(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 rest of the experiments, we thus considered both the primary and secondary information.

Experiment $\#3$: *ACT evaluation*.
In this experiment, we examined the effectiveness of ACT on two datasets of different benign/suspicious ratios. The balanced dataset (the second column in Table 2) has a benign/suspicious ratio of about

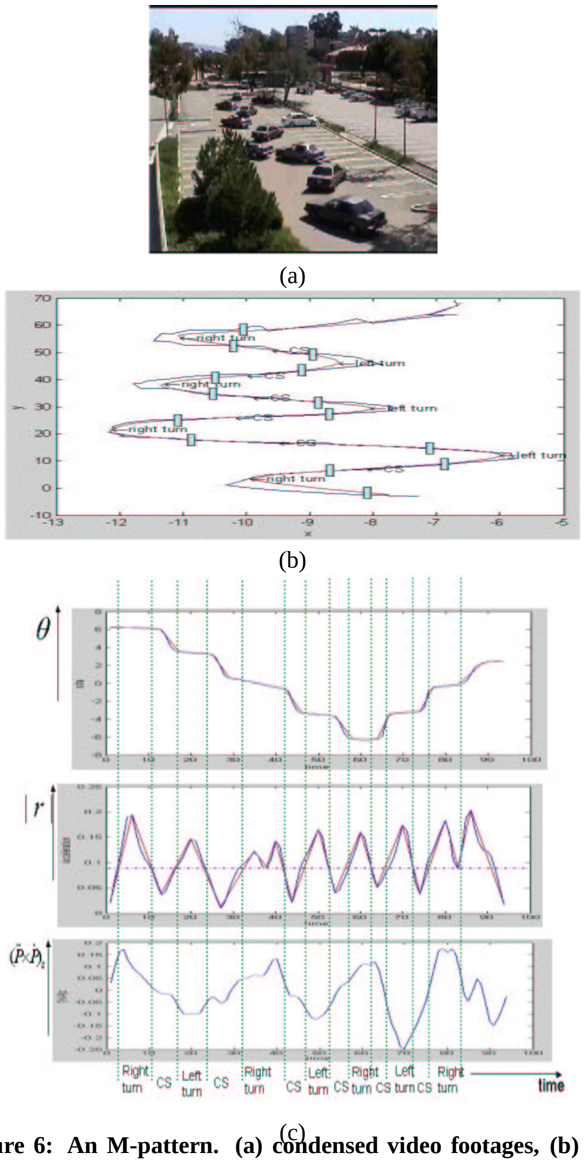
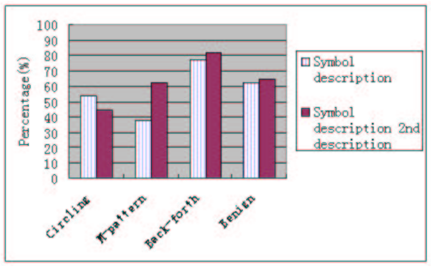


Figure 6: An M-pattern. (a) condensed video footages, (b) raw (black or dark) and the Kalman filtered (red or light) trajectories with segment boundaries and labels, and (c) acceleration curves used in segmentation.

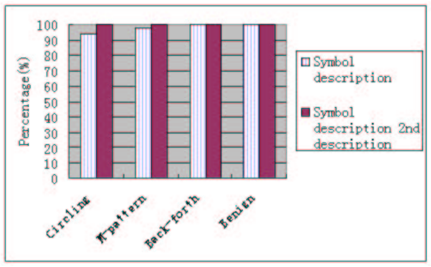
50%. Figures 7(c) and 7(d) show that the employment of ACT improves sensitivity significantly by 39%, whereas it degrades specificity by just 4%. Next, we repeated the ACT test on the skewed dataset (the third column in Table 2), where the benign/suspicious ratio is less than 3%. Figures 7(e) and 7(f) show that the average sensitivity suffers from a drop from 68% to 35%. After applying ACT, the average sensitivity improved to 70% by giving away just 3% in specificity.

5. CONCLUSIONS

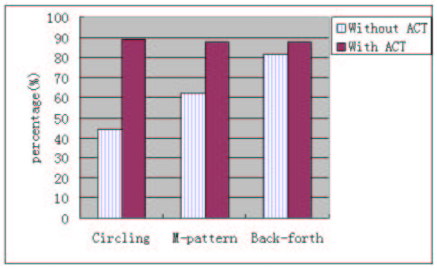
In this paper, we have presented methods for 1) fusing multi-camera surveillance data, 2) characterizing motion patterns and their secondary structure, 3) and conducting statistical learning in an imbalanced training-data setting for detecting rare events. For fusing multi-source data from cameras with overlapping spatial and temporal coverage, we proposed using a two-level hierarchy of Kalman filters. 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



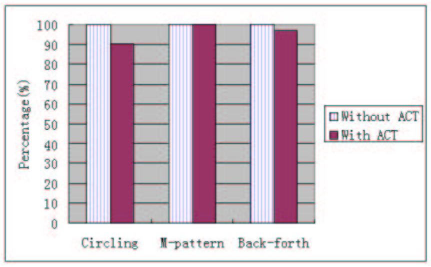
(a) Sensitivity (Kernel Test)



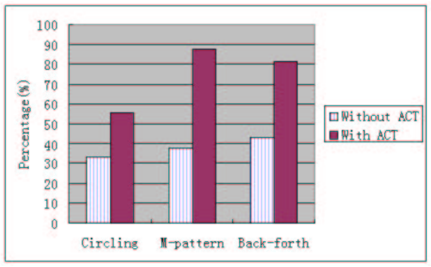
(b) Specificity (Kernel Test))



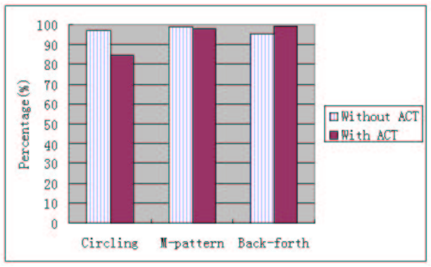
(c) Sensitivity (ACT Test — Balanced Data)



(d) Specificity (ACT Test — Balanced Data)



(e) Sensitivity (ACT Test — Skewed Data)



(f) Specificity (ACT Test — Skewed Data)

Figure 7: Sensitivity and Specificity of Three Test Cases.

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 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.

6. REFERENCES

[1] S. Amari and S. Wu. Improving support vector machine classifiers by modifying kernel functions. *Neural Networks*, 1999.

[2] R. G. Brown. *Introduction to Random Signal Analysis and Kalman filtering*. Wiley, New York, NY, 1983.

[3] E. Church. Revised Geometry of the Aerial Photograph. *Bulletin of Aerial Photogrammetry*, 15, 1945.

[4] D. F. DeMenthon and L. S. Davis. Model-based object pose in 25 lines of code. *Int. J. Comput. Vision*, 15:123–141, 1995.

[5] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification, 2nd Ed.* Wiley, New York, 2001.

[6] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, San Diego, CA, 4 edition, 1997.

[7] O. Faugeras. *Three-Dimensional Computer Vision*. MIT Press, Cambridge, MA, 1993.

[8] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice, 2nd ed.* Addison-Wesley, Reading, MA, 1990.

[9] R. Haralick, H. Joo, C. Lee, X. Zhuang, V. Vaidya, and M. Kim. Pose estimation from corresponding point data. *IEEE Trans. Syst., Man, Cybern.*, 19:1426–46, 1989.

[10] R. Horaud, F. Dornaika, B. Lamiroy, and S. Christy. Object pose: The link between weak perspective, paraperspective and full perspective. *Int. J. Comput. Vision*, 22:173–189, 1997.

[11] S. J. Julier, J. K. Uhlmann, and H. F. Durrant-Whyte. A New Approach for Filtering Nonlinear Systems. In *Proc. Amiercan Control Conference*, Seattle, WA, 1995.

[12] P. S. Maybeck. *Stochastic Models, Estimation, and Control, vol. 1*. Academic Press, New York, NY, 1979.

[13] F. W. Sears. *Optics, 3rd Ed.* Addison-Wesley, Reading, MA, 1958.

[14] G. Xu and Z. Zhang. *Epipolar Geometry in Stereo, Motion and Object Recognition*. Kluwer Academic Publishers, The Netherlands, 1996.

[15] Z. Zhang. A flexible new technique for camera calibration. *IEEE Trans. Pattern Analy. Machine Intell.*, 22:1330–4, 2000.