Learning Patterns of Activity Using Real-Time Tracking

Chris Stauffer, *Member*, *IEEE*, and
W. Eric L. Grimson, *Member*, *IEEE*

Abstract—Our goal is to develop a visual monitoring system that passively observes moving objects in a site and learns patterns of activity from those observations. For extended sites, the system will require multiple cameras. Thus, key elements of the system are motion tracking, camera coordination, activity classification, and event detection. In this paper, we focus on motion tracking and show how one can use observed motion to learn patterns of activity in a site. Motion segmentation is based on an adaptive background subtraction method that models each pixel as a mixture of Gaussians and uses an on-line approximation to update the model. The Gaussian distributions are then evaluated to determine which are most likely to result from a background process. This yields a stable, real-time outdoor tracker that reliably deals with lighting changes, repetitive motions from clutter, and long-term scene changes. While a tracking system is unaware of the identity of any object it tracks, the identity remains the same for the entire tracking sequence. Our system leverages this information by accumulating joint co-occurrences of the representations within a sequence. These joint co-occurrence statistics are then used to create a hierarchical binary-tree classification of the representations. This method is useful for classifying sequences, as well as individual instances of activities in a site.

Index Terms—Real-time visual tracking, adaptive background estimation, activity modeling, co-occurrence clustering, object recognition, video surveillance and monitoring (VSAM).

1 Introduction

THE goal of this project is a vision system that monitors activity in a site over extended periods of time, i.e., that detects patterns of motion and interaction demonstrated by objects in the site. The system:

- Should provide statistical descriptions of typical activity patterns, e.g., normal vehicular volume or normal pedestrian traffic paths for a given time of day;
- Should detect unusual events, by spotting activities that are very different from normal patterns, e.g., unusual volumes of traffic, or a specific movement very different from normal observation; and
- Should detect unusual interactions between objects, e.g., a person parking a car in front of a building, exiting the car, but not entering the building.

Because a site may be larger than can be observed by a single camera, our system observes activities with a "forest of sensors" distributed around the site. Ideally, each sensor unit would be a compact packaging of camera, on-board computational power, local memory, communication capability, and possibly locational instrumentation (e.g., GPS). Example systems exist [10], [11], [17] and more powerful systems will emerge as technology in sensor design, DSP processing, and communications evolves. In a forest, many such sensor units would be distributed around the site. For outdoor settings, this would involve attaching them to poles, trees, and buildings.[1] For indoor settings, this would involve attaching to walls and furniture for indoor sites, such as the Intelligent Room.[2] For this article, we explore the monitoring of an outdoor site by connecting a set of video cameras to an interconnected suite of PCs, with each camera looking out a different window of a building, i.e., our focus is on the algorithmic processing of the data, rather than on the specific sensor packages.

The forest should learn patterns of activities in a site, then monitor and classify activities based on these learned patterns. A coordinated sensor forest needs:

- *Self-calibration*—determine the positions of all the cameras relative to one another;
- *Construction of rough site models*—determine the ground plane and mark occupied areas;
- *Detect objects* in the site—extract information about all moving objects in the site;
- *Classify detected objects*—label detected objects by common shape, appearance, or motion;
- *Learn from extended observation* (e.g., over a period of weeks)—what are the common activity patterns; and
- *Detect unusual events* in the site—mark activities that don't fit common patterns.

Our hypothesis is that these tasks can be accomplished simply by observing moving objects. To verify this hypothesis, we need a robust tracker that can reliably detect moving objects and return an accurate description of the observed object, both its motion parameters and its intrinsic parameters, such as size and shape, and methods that can use such tracking data to accomplish the tasks listed above. In the following sections, we describe our tracking method [23], then outline our system for monitoring activities over extended time periods by simply observing object motions. Calibration of cameras and extraction of ground plane information are covered separately in [18].

2 Building a Robust Motion Tracker

A robust video surveillance and monitoring system should not depend on careful placement of cameras. It should also be robust to whatever is in its visual field or whatever lighting effects occur. It should be capable of dealing with movement through cluttered areas, objects overlapping in the visual field, shadows, lighting changes, effects of moving elements of the scene (e.g., swaying trees), slow-moving objects, and objects being introduced or removed from the scene. Thus, to monitor activities in real outdoor settings, we need robust motion detection and tracking that can account for such a wide range of effects.

Traditional approaches based on backgrounding methods typically fail in these general situations. Our goal is to create a robust, adaptive tracking system that is flexible enough to handle variations in lighting, moving scene clutter, multiple moving objects, and other arbitrary changes to the observed scene. The resulting tracker is primarily geared toward scene-level video surveillance applications.

2.1 Previous Work and Current Shortcomings of Motion Tracking

Most researchers have abandoned nonadaptive methods of backgrounding because of the need for manual initialization. Without reinitialization, errors in the background accumulate over time, making this method useful only in highly supervised, short-term tracking applications without significant changes in the scene. It is possible to use a maximum interframe difference [19], but this

• *The authors are with the Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139.*
E-mail: {stauffer, welg}@ai.mit.edu.

Manuscript received 21 Apr. 1999; revised 25 Feb. 2000; accepted 28 Mar. 2000.
Recommended for acceptance by R. Collins.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 109663.

1. See http://www.ai.mit.edu/projects/vsam..
2. See http://www.ai.mit.edu/projects/hci..

leaves "ghosts" where the object was and leaves large regions of the object undetected unless the object undergoes significant motion each frame.

Most backgrounding methods involve continuously estimating a statistical model of the variation for each pixel. A common method of adaptive backgrounding is averaging the images over time, creating a background approximation which is similar to the current static scene except where motion occurs. While this is effective in situations where objects move continuously and the background is visible a significant portion of the time, it is not robust to scenes with many moving objects, particularly if they move slowly. It also cannot handle bimodal backgrounds, recovers slowly when the background is uncovered, and has a single, predetermined threshold for the entire scene. One interesting attempt to meet these difficulties is W^4 [9], which combined its estimates of the minimum value, maximum value, and maximum interframe difference per pixel.

Ivanov et al. [12] used disparity verification to determine moving regions in a scene. This showed invariance to lighting variations, but involved a costly, off-line initialization. Its primary application is for geometrically static backgrounds. Recently, an eigenvector approximation of the entire image was used to model the background in outdoor scenes [20].

Changes in scene lighting can cause problems for many backgrounding methods. Ridder et al. [21] modeled each pixel with a Kalman Filter, which made their system more robust to lighting changes in the scene. While this method does have a pixel-wise automatic threshold, it still recovers slowly and does not handle bimodal backgrounds well. Koller et al. [16] have successfully integrated this method in an automatic traffic monitoring application.

Pfinder [24] uses a multiclass statistical model for the foreground objects, but the background model is a single Gaussian per pixel. After an initialization period where the room is empty, the system reports good results. There have been no reports on the success of this tracker in outdoor scenes.

Friedman and Russell [5] have recently implemented a pixel-wise EM framework for detection of vehicles that bears the most similarity to our work. Their method attempts to explicitly classify the pixel values into three separate, predetermined distributions corresponding to the road color, the shadow color, and colors corresponding to vehicles. Their attempt to mediate the effect of shadows appears to be somewhat successful, but it is not clear what behavior their system would exhibit for pixels which did not contain these three distributions. For example, pixels may present a single background color or multiple background colors resulting from repetitive motions, shadows, or reflectances.

2.2 Our Approach to Motion Tracking

Rather than explicitly modeling the values of all the pixels as one particular type of distribution, we simply model the values of a particular pixel as a mixture of Gaussians. Based on the persistence and the variance of each of the Gaussians of the mixture, we determine which Gaussians may correspond to background colors. Pixel values that do not fit the background distributions are considered foreground until there is a Gaussian that includes them with sufficient, consistent evidence supporting it to convert it to a new background mixture.

Our system adapts to deal robustly with lighting changes, repetitive motions of scene elements, tracking through cluttered regions, slow-moving objects, and introducing or removing objects from the scene. Slowly moving objects take longer to be incorporated into the background because their color has a larger variance than the background. Also, repetitive variations are learned and a model for the background distribution is generally maintained, even if it is temporarily replaced by another distribution which leads to faster recovery when objects are removed.

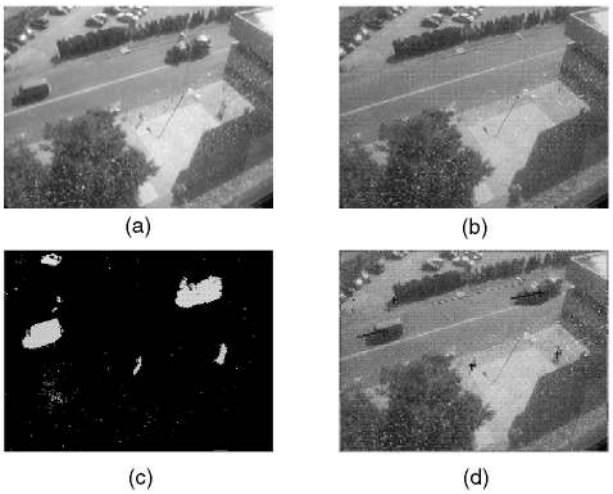


Fig. 1. The execution of the program. (a) the current image, (b) an image composed of the means of the most probable Gaussians in the background model, (c) the foreground pixels, (d) the current image with tracking information superimposed. Note: While the shadows are foreground in this case, if the surface was covered by shadows a significant portion of the time, a Gaussian representing those pixel values may be significant enough to be considered background.

Our backgrounding method contains two significant parameters—α, the learning constant, and T, the proportion of the data that should be accounted for by the background. Without any alteration of parameters, our system has been used in an indoor, human-computer interface application and, since October 1997, has been continuously monitoring outdoor scenes.

3 Adaptive Backgrounding for Motion Tracking

If each pixel resulted from a single surface under fixed lighting, a single Gaussian would be sufficient to model the pixel value while accounting for acquisition noise. If only lighting changed over time, a single, adaptive Gaussian per pixel would be sufficient. In practice, multiple surfaces often appear in the view frustum of a particular pixel and the lighting conditions change. Thus, multiple, adaptive Gaussians are required. We use an adaptive mixture of Gaussians to approximate this process.

Each time their parameters are updated, the Gaussians are evaluated using a simple heuristic to hypothesize which are most likely to be part of the "background process." Pixel values that do not match one of the pixel's "background" Gaussians are grouped using connected components. Finally, the connected components are tracked across frames using a multiple hypothesis tracker. The process is illustrated in Fig. 1.

3.1 Online Mixture Model

We consider the values of a particular pixel over time as a "pixel process," i.e., a time series of scalars for grayvalues or vectors for color pixel values. At any time, t, what is known about a particular pixel, $\{x_0, y_0\}$, is its history

$$\{X_1, \ldots, X_t\} = \{I(x_0, y_0, i) : 1 \leq i \leq t\}, \tag{1}$$

where I is the image sequence. Some "pixel processes" are shown by the (R,G) scatter plots in Fig. 2 which illustrate the need for adaptive systems with automatic thresholds. Fig. 2b and Fig. 2c also highlight a need for a multimodal representation. In each case, the ideal distribution of values should be a tight, Gaussian-like cluster around some point. The fact that the cluster can shift dramatically over a period of a few minutes or that two or more

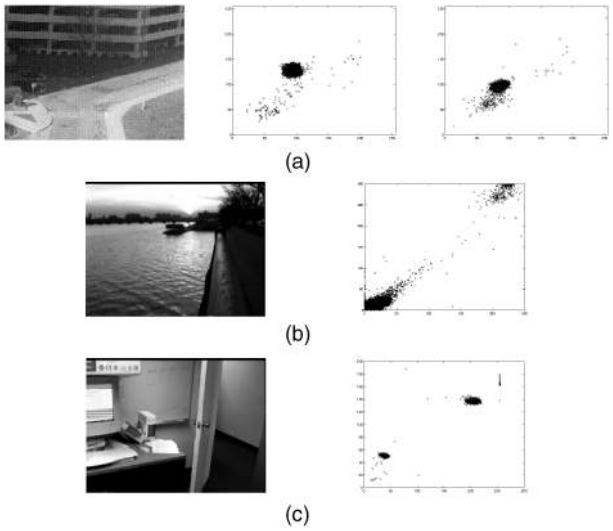


Fig. 2. This figure contains images and scatter plots of the red and green values of a single pixel from the image over time. It illustrates some of the difficulties involved in real environments. (a) shows two scatter plots from the same pixel taken 2 minutes apart. This would require two thresholds. (b) shows a bi-model distribution of a pixel values resulting from specularities on the surface of water. (c) shows another bi-modality resulting from monitor flicker.

processes at the same pixel can result in several distinctive clusters illustrates the need for an adaptive, multimodal representation.

We chose to model the recent history of each pixel, $\{X_1, \ldots, X_t\}$, as a mixture of K Gaussian distributions. The probability of observing the current pixel value is

$$P(X_t) = \sum_{i=1}^{K} \omega_{i,t} * \eta(X_t, \mu_{i,t}, \Sigma_{i,t}), \tag{2}$$

where K is the number of distributions, $\omega_{i,t}$ is an estimate of the weight (the portion of the data accounted for by this Gaussian) of the ith Gaussian in the mixture at time t, $\mu_{i,t}$ and $\Sigma_{i,t}$ are the mean value and covariance matrix of the ith Gaussian in the mixture at time t, and where η is a Gaussian probability density function

$$\eta(X_t, \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(X_t - \mu_t)^T \Sigma^{-1} (X_t - \mu_t)}. \tag{3}$$

K is determined by the available memory and computational power. Currently, from three to five are used. Also, for computational reasons, the covariance matrix is assumed to be of the form:

$$\Sigma_{k,t} = \sigma_k^2 \mathbf{I}. \tag{4}$$

This assumes that the red, green, and blue pixel values are independent and have the same variances. While this is certainly not the case, the assumption allows us to avoid a costly matrix inversion at the expense of some accuracy.

Thus, the distribution of recently observed values of each pixel in the scene is characterized by a mixture of Gaussians. A new pixel value will, in general, be represented by one of the major components of the mixture model and used to update the model.

If the pixel process could be considered a stationary process, a standard method for maximizing the likelihood of the observed data is *expectation maximization* [4]. Because there is a mixture model for every pixel in the image, implementing an exact EM algorithm on a window of recent data would be costly. Also, lighting changes and the introduction or removal of static objects suggest a decreased dependence on observations further in the past. These two factors led us to use the following on-line K-means approximation to update the mixture model.

Every new pixel value, X_t, is checked against the existing K Gaussian distributions until a match is found. A match is defined as a pixel value within 2.5 standard deviations of a distribution.[3] This threshold can be perturbed with little effect on performance. This is effectively a per pixel/per distribution threshold. This is extremely useful when different regions have different lighting (see Fig. 2a) because objects which appear in shaded regions do not generally exhibit as much noise as objects in lighted regions. A uniform threshold often results in objects disappearing when they enter shaded regions.

If none of the K distributions match the current pixel value, the least probable distribution is replaced with a distribution with the current value as its mean value, an initially high variance, and low prior weight.

The prior weights of the K distributions at time t are adjusted as follows:

$$\omega_{k,t} = (1 - \alpha)\omega_{k,t-1} + \alpha(M_{k,t}), \tag{5}$$

where α is the learning rate[4] and $M_{k,t}$ is 1 for the model which matched and 0 for the remaining models. After this approximation, the weights are renormalized. $1/\alpha$ defines the time constant which determines change. $\omega_{k,t}$ is effectively a causal low-pass filtered average of the (thresholded) posterior probability that pixel values have matched model k given observations from time 1 through t. This is equivalent to the expectation of this value with an exponential window on the past values.

The μ and σ parameters for unmatched distributions remain the same. The parameters of the distribution which matches the new observation are updated as follows:

$$\mu_t = (1 - \rho)\mu_{t-1} + \rho X_t \tag{6}$$

$$\sigma_t^2 = (1 - \rho)\sigma_{t-1}^2 + \rho(X_t - \mu_t)^T (X_t - \mu_t), \tag{7}$$

where

$$\rho = \alpha \eta(X_t | \mu_k, \sigma_k) \tag{8}$$

is the learning factor for adapting current distributions.[5] This is effectively the same type of causal low-pass filter as mentioned above, except that only the data which matches the model is included in the estimation.

One of the significant advantages of this method is that, when something is allowed to become part of the background, it doesn't destroy the existing model of the background. The original background color remains in the mixture until it becomes the Kth most probable and a new color is observed. Therefore, if an object is stationary just long enough to become part of the background and then it moves, the distribution describing the previous background still exists with the same μ and σ^2, but a lower ω, and will be quickly reincorporated into the background.

3.2 Background Model Estimation

As the parameters of the mixture model of each pixel change, we would like to determine which of the Gaussians of the mixture are most likely produced by background processes. Heuristically, we are interested in the Gaussian distributions which have the most supporting evidence and the least variance.

3. Depending on the kurtosis of the noise, some percentage of the data points generated by a Gaussian will not "match." The resulting random noise in the foreground image is easily ignored by neglecting connected components containing only a few pixels.

4. While this rule is easily interpreted an an interpolation between two points, it is often shown in the equivalent form: $\omega_{k,t} = \omega_{k,t-1} + \alpha(M_{k,t} - \omega_{k,t-1})$.

5. In high dimensional spaces with full covariance matrices, it is sometimes advantageous to use a constant ρ to reduce computation and provide faster Gaussian tracking.

To understand this choice, consider the accumulation of supporting evidence and the relatively low variance for the "background" distributions when a static, persistent object is visible. In contrast, when a new object occludes the background object, it will not, in general, match one of the existing distributions, which will result in either the creation of a new distribution or the increase in the variance of an existing distribution. Also, the variance of the moving object is expected to remain larger than a background pixel until the moving object stops. To model this, we need a method for deciding what portion of the mixture model best represents background processes.

First, the Gaussians are ordered by the value of ω/σ. This value increases both as a distribution gains more evidence and as the variance decreases. After reestimating the parameters of the mixture, it is sufficient to sort from the matched distribution toward the most probable background distribution because only the matched models relative value will have changed. This ordering of the model is effectively an ordered, open-ended list, where the most likely background distributions remain on top and the less probable transient background distributions gravitate toward the bottom and are eventually replaced by new distributions.

Then, the first B distributions are chosen as the background model, where

$$B = argmin_b \left(\sum_{k=1}^{b} \omega_k > T \right), \tag{9}$$

where T is a measure of the minimum portion of the data that should be accounted for by the background. This takes the "best" distributions until a certain portion, T, of the recent data has been accounted for. If a small value for T is chosen, the background model is usually unimodal. If this is the case, using only the most probable distribution will save processing.

If T is higher, a multimodal distribution caused by a repetitive background motion (e.g., leaves on a tree, a flag in the wind, a construction flasher, etc.) could result in more than one color being included in the background model. This results in a transparency effect which allows the background to accept two or more separate colors.

3.3 Connected Components

The method described above allows us to identify foreground pixels in each new frame while updating the description of each pixel's process. These labeled foreground pixels can then be segmented into regions by a two-pass, connected components algorithm [8].

Because this procedure is effective in determining the whole moving object, moving regions can be characterized not only by their position, but size, moments, and other shape information. Not only can these characteristics be useful for later processing and classification, but they can aid in the tracking process.

3.4 Multiple Hypothesis Tracking

Establishing correspondence of connected components between frames is accomplished using a linearly predictive multiple hypotheses tracking algorithm which incorporates both position and size. We have implemented an online method for seeding and maintaining sets of Kalman filters.

At each frame, we have an available pool of Kalman models and a new available pool of connected components that they could explain. First, the models are probabilistically matched to the connected regions that they could explain. Second, the connected regions which could not be sufficiently explained are checked to find new Kalman models. Finally, models whose fitness (as determined by the inverse of the variance of its prediction error) falls below a threshold are removed.

Matching the models to the connected components involves checking each existing model against the available pool of connected components which are larger than a pixel or two. All matches with relatively small error are used to update the corresponding model. If the updated models have sufficient fitness, they will be used in the following frame. If no match is found, a "null" match can be hypothesized which propagates the model as expected and decreases its fitness by a constant factor. If the object reappears in a predictable region of uncertainty shortly after being lost, the model will regain the object. Because our classification system requires tracking sequences which consist of representations of a single object, our system generally breaks tracks when objects interact rather than guessing at the true correspondence.

The unmatched models from the current frame and the previous two frames are then used to hypothesize new models. Using pairs of unmatched connected components from the previous two frames, a model is hypothesized. If the current frame contains a match with sufficient fitness, the updated model is added to the existing models. To avoid possible combinatorial explosions in noisy situations, it may be desirable to limit the maximum number of existing models by removing the least probable models when excessive models exist. In noisy situations (e.g., ccd cameras in low-light conditions), it is often useful to remove the short tracks that may result from random correspondences. Further details of this method can be found at http://www.ai.mit.edu/projects/vsam/.

4 Performance of the Tracker

On an SGI O2 with an R10000 processor, this method can process 11 to 13 frames a second (frame size 160×120 pixels). The variation in the frame rate is due to variation in the amount of foreground present. Our tracking system has been effectively storing tracking information for five scenes since 1997 [7]. Fig. 3 and Fig. 4 show accumulated tracks in two scenes over the period of a day. While quick changes in cloud cover (relative to α, the learning rate) can sometimes necessitate a new set of background distributions, it will stabilize within 10-20 seconds and tracking will continue unhindered.

The tracking system has the most difficulty with scenes containing high occurrences of objects that visually overlap. The multiple hypothesis tracker is not extremely sophisticated about reliably disambiguating objects which cross. Adding more complex dynamics or appearance templates [9] could help in this regard. This problem can be compounded by long shadows, but, for our applications, it was much more desirable to track an object and its shadow and avoid cropping or missing dark objects than it was to attempt to remove shadows. In our experience, on bright days when the shadows are the most significant, both shadowed regions and shady sides of dark objects are black (not dark green, not dark red, etc.).

The tracker was robust to all but relatively fast lighting changes (e.g., flood lights turning on and partly cloudy, windy days). It successfully tracked outdoor scenes in rain, snow, sleet, hail, overcast, and sunny days. It has also been used to track birds at a feeder, mice at night using Sony NightShot, fish in a tank, people in a lab environment, and objects in outdoor scenes. In these environments, it reduces the impact of repetitive motions from swaying branches, rippling water, specularities, slow moving objects, and acquisition noise. The system has proven robust to day/night cycles and long-term scene changes. More recent results and project updates are available at http://www.ai.mit.edu/projects/vsam/.

5 Improving the Tracker

Although we find the results of the tracker encouraging, there are still opportunities for improvement.

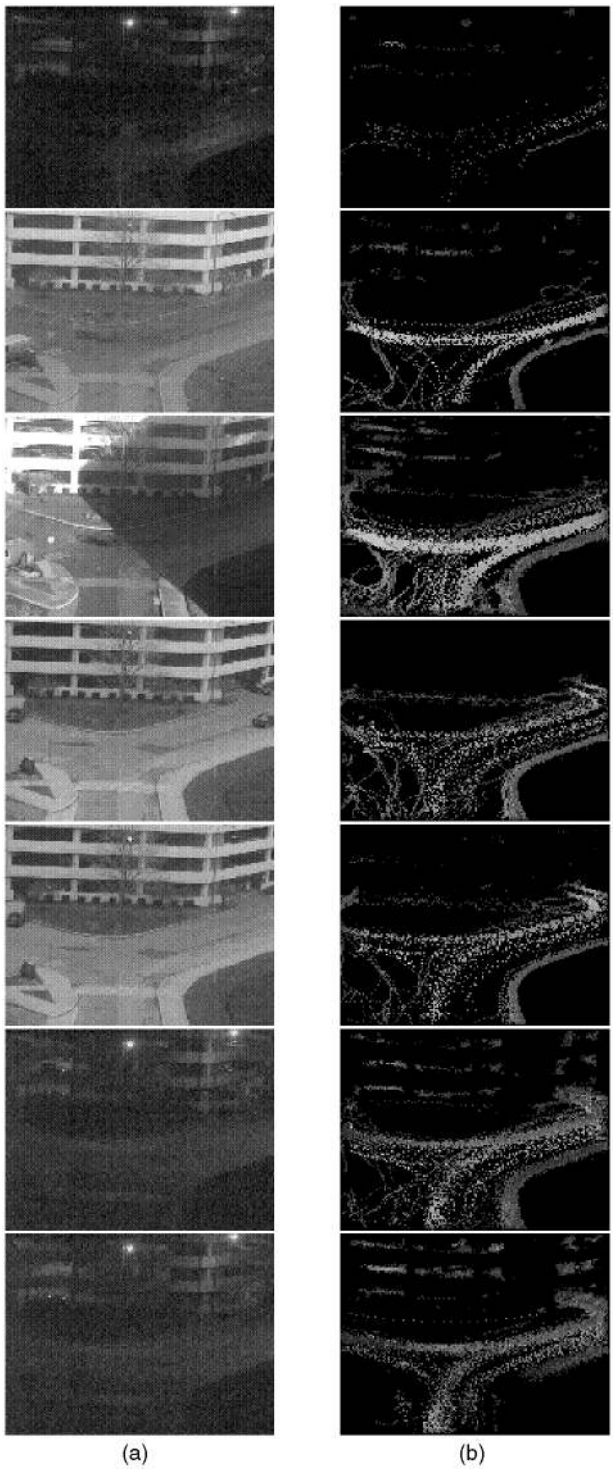


Fig. 3. This figure shows consecutive hours of tracking from 6am to 9am and 3pm to 7pm. (a) shows the image at the time the template was stored and (b) shows the accumulated tracks of the objects over that time. Color encodes object direction and intensity encodes object size. The consistency of the colors within particular regions reflects the consistency of the speed, direction, and size parameters which have been acquired.

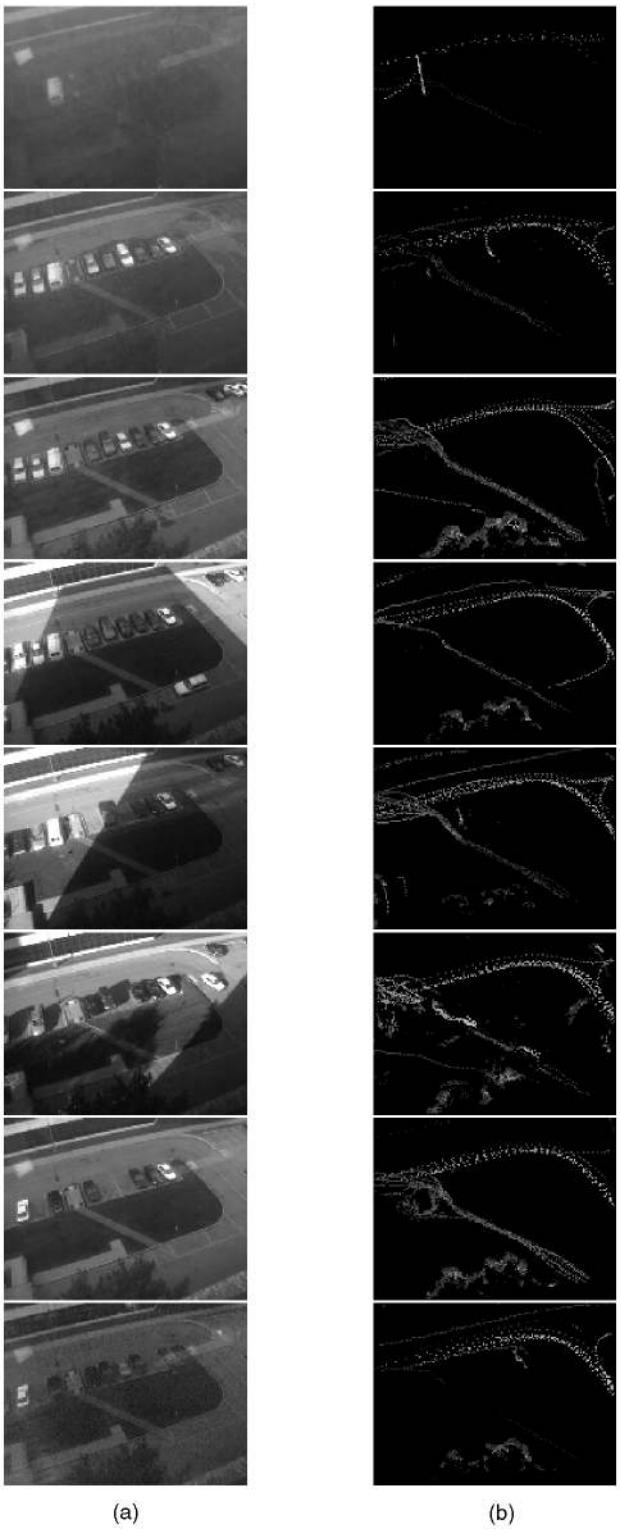


Fig. 4. This figure shows consecutive intervals of tracking on a different scene than previous figure. Also, this particular day was foggy, then clear, then overcast. As the templates show, the tracking was relatively unaffected.

As computers improve and parallel architectures are investigated, this algorithm can be run faster, on larger images, and using a larger number of Gaussians in the mixture model. All of these factors will increase performance. A full covariance matrix would further improve performance. Adding prediction to each Gaussian (e.g., the Kalman filter approach) may also lead to more robust tracking of lighting changes.

Beyond these obvious improvements, we are investigating modeling some of the interdependencies of the pixel processes. Relative values of neighboring pixels, correlations with neighboring pixel's distributions, and simple texture measures may be useful in this regard. This would allow the system to model changes in occluded pixels by observations of some of its neighbors.

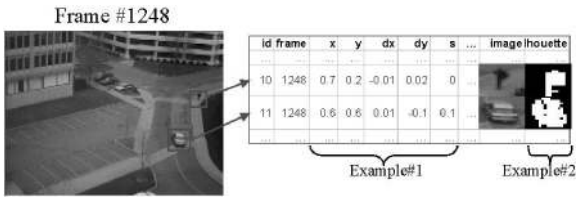


Fig. 5. This figure shows a single frame from a typical scene and the information which recorded for the two moving objects. The fields which are used for the two classification examples are labeled.

Our method has been used on grayscale, RGB, HSV, and local linear filter responses. But, this method should be capable of modeling any streamed input source in which our assumptions and heuristics are generally valid. We are investigating use of this method with frame-rate stereo, IR cameras, and including depth as a fourth channel (R,G,B,D). Depth is an example where multimodal distributions are useful because, while disparity estimates are noisy due to false correspondences, those noisy values are often relatively predictable when they result from false correspondences in the background.

6 Interpreting the Motion Tracks

Our simple adaptive background tracker has tracked over 10 million objects since 1997. As shown in Fig. 5, for every frame that an object is tracked, its location (x, y), speed/direction (dx, dy), and size are recorded. Also, an image of the object and a binary motion silhouette are cropped from the original image and the binary difference image, respectively.

Because of the stability and completeness of the representation, it is possible to do some simple classification based on aspect ratio or size. Of more interest is classification based on the actual movement or shape of the object. The two sets of experiments discussed below perform classification based on the {x, y, dx, dy, size} representation and the binary motion silhouette representation using literally millions of training examples. Rather than using sequences to create a sequence classifier (as is most common), we are using the sequences to create an instance classifier.

Our method involves developing a codebook of representations using an on-line Vector Quantization (VQ) on the entire set of representations acquired by the tracker. Second, we accumulate joint co-occurrence statistics over the codebook by treating the set of representations in each sequence as an equivalency multiset. Finally, we perform hierarchical classification using only the accumulated co-occurrence data.

6.1 Previous Work in Classification

There are countless examples of tracking systems that perform predetermined classification tasks on tracked data, e.g., human vs. vehicle or walking vs. running [2]; walking, marching, line-walking, and kicking [3]; etc.

We are not interested in predetermined classification tasks. Our method is most similar to the work of Johnson and Hogg [13]. They begin their process by on-line Vector Quantization on the input space. They then quantize again into a predetermined number of probability distribution functions (pdfs) over their discrete states. While a significant number of these pdfs will result in tight clusters of activity, it is unclear how to relate two inputs that are grouped into separate pdfs or to select the proper number of pdfs.

Our hierarchical classification involves a step that has the flavor of Normalized Cuts and its many derivatives (see [22]). It has discrete nodes (defined by the codebook). It has edges which represent pairwise distances (or dissimilarities or costs) between them. In addition, the goal is to determine two sets of nodes that are dissimilar. However, that is the extent of the similarity. Our "costs" are probabilities, not "distances." Those similarities are not directly related to the coordinates or properties of the nodes, but rather are measured empirically from the data. Our "cut" does not produce two discrete sets that minimize the cut "similarities." It produces two distributions that both explain the observed joint statistics and are relatively dissimilar and match the co-occurrence data.

The following sections describe the method, show two sets of results, discuss ways of improving this method, and draw conclusions.

7 The Classification Method

We assume that the tracker will produce a sequence of representations of the same object. For example, a person who is tracked through the scene for N frames will produce N images, N binary silhouettes, N positions, N velocities, etc. Unless the tracker makes a mistake in establishing correspondence, every representation in a sequence should correspond to the same underlying object. When developing a tracker for this type of application, it is important to avoid tracking errors involving false correspondences.

The following sections outline the basic process for classification. First, a codebook of prototype representations is generated using on-line Vector Quantization(VQ). Second, the automatically tracked sequences are used to define a co-occurrence matrix over the prototypes in the codebook. Finally, the co-occurrence data is used to probabilistically break apart the prototypes in the codebook into a binary tree representation. The result is a hierarchical classifier which can classify any individual representation or sequences of representations.

7.1 Codebook Generation

A codebook is a set of prototype representations which approximate the density of the input representations. There are many methods of developing codebooks of prototypes (see [6] for a discussion).

For the quantity of data and the number of prototypes we use, an off-line method, such as K-means, is not an option. The simplest method of on-line Vector Quantization is to initialize the codebook randomly with K prototypes centered at existing data points. Then, take single data points, find the closest prototype in the codebook, and adapt that prototype toward the data point using a learning factor, α. This process is repeated for millions of data points as the α value is slowly decreased until the prototypes are stable and represent an equal amount of data. The input spaces we dealt with did not require complex annealing strategies.

We occasionally encountered an initialization problem. Prototypes seeded on outliers may be stranded, representing only that data point. We circumvented this problem with a method used by Johnson and Hogg [13] which enforces that each prototype represents the same amount of data. Over time, stranded data points account for larger regions of the input space until they represent new data points. The prototypes are then adapted toward the new data points until they represent as much data as all the other points.

Once a codebook is generated, it is used as a lookup table for incoming values, i.e., new values are represented by labels of nearby prototypes. Given the desired size of the codebook, the goal of quantizing is to determine a set of prototypes which best represents the dataset. Our results were produced with codebooks of 400 prototypes. More complex spaces (e.g., color image space) would necessitate either more prototypes or more complex prototypes.

Depending on the complexity of the input space, it may be difficult to create an effective codebook of representations. If all the representations in the codebook are equally likely to result from all the underlying classes, this system will fail. For example, if none of the representations in your codebook is more likely to result from a person than a vehicle, there will be no possibility of using those representations to differentiate people and vehicles without additional information.

While this may seem unsettling, we are encouraged by our ability to generate large codebooks. Large codebooks are usually troublesome because, as the size of the codebook, K, increases, the amount of data needed for effective codebook generation increases on the order of K. Also, the amount of data needed for co-occurrence statistics accumulation increases on the order of K^2. Since our system automatically collects and processes data, we have hundreds of gigabytes of tracking data for future processing steps. And, our method converges as the amount of data increases rather than suffering from over-fitting.

An area of high data point density may accumulate a large portion of the prototypes, leaving few prototypes for the rest of input space. In some cases, it may be desirable to have a large number of prototypes in the high-density areas because those regions may be the most ambiguous regions of the input space (e.g., traffic at an intersection). In other cases, the areas of high density may arise from uninteresting, repetitive input data (e.g., scene clutter) and there is no benefit to wasting a large portion of your prototypes in that region. We currently filter most of the sequences which are less than a few seconds in duration. This filters most of the repetitive motions in the scene before the learning process.

7.2 Accumulating Co-Occurrence Statistics

Once the codebook has been generated, the input space is no longer considered. Every input data point is labeled as the most representative prototype—the one that is nearest to it. So, rather than considering a sequence of images, binary silhouettes, positions, or histograms, we convert to the codebook labels, then only consider sequences of symbols, s_1 through s_K, corresponding to the K prototypes.

Further, our method disregards the order of the sequence and considers them as multisets of symbols. A multiset is a set which can contain multiple instances of the same element. Each pair within a sequence (excluding pairing a prototype label with itself) is evidence that those two prototypes' appearances resulted from the same underlying class.

The goal of this system is to produce a classification system which can be given one or more observations (e.g., an image, a silhouette, etc.) of a particular object and classify it into a set of classes such that the same type of object tends to be put in the same class. This is in contrast to systems that are specifically designed to recognize sequences (e.g., Hidden Markov Models). When the system has learned to classify an object based on its motion silhouette, color histogram, or size, it should be capable of doing so with a single example. Of course, the system should perform better if given multiple examples, but it should not rely on seeing a complete sequence.

Our model for the production of the sequences is simple. There are N underlying classes, each of which occurs with some prior probability, π_c. A class c, when observed, has some probability distribution, $p_c()$, of producing each of the prototype's symbols. As long as the object is observed, it will produce symbols given the same distribution. This model reflects our assumption of the independence of samples in a sequence discussed earlier.

The multisets of prototypes are used to estimate a co-occurrence matrix, C where $c_{i,j}$ is the estimated probability that a sequence from the training sequences will contain an input represented by the ith prototype and a separate input represented by the jth prototype.

First, a matrix of the accumulated co-occurrences, $C_{i,j}^{total}$, is initialized to zeros or a prior joint distribution (see Section 10). Given a multiset, each possible pair (excluding pairing symbols with themselves) is added to C^{total}, weighted inversely by the number of pairs in that sequence. Given a sequence, $S = \{S_1, S_2, \ldots\}$, for each pair $\{S_i, S_j \; where \; i \neq j\}$

$$C_{i,j}^{total} = C_{i,j}^{total} + 1/P, \tag{10}$$

where $P = |S|^2 - |S|$ is the number of valid pairs in this sequence. Then, the current joint co-occurrence estimate, C, is C^{total} normalized

$$C = C^{total}/Z, \tag{11}$$

where Z is the number of sequences currently used to estimate C^{total}.

If there was a single underlying class and infinite sequences to train, $C_{i,j}$ would converge to $p_1(i) * p_1(j)$. In such a case, nothing can be said about the relative relationships of the prototypes. With N underlying classes,

$$\lim_{Z \to \infty} C_{i,j} = \sum_{c=1}^{N} \pi_c * p_c(i) * p_c(j). \tag{12}$$

Given enough synthetically produced data from a system for which each class has one prototype for which it is the sole producer, it is possible to solve for all parameters of the model. Since this is a restrictive case, we will not pursue it here. The next section outlines how our system extracts a hierarchical approximation to these classes.

7.3 Hierarchical Classification

Our classification method takes the entire set of prototypes and the co-occurrence matrix and attempts to determine two distributions, or probability mass functions(pmfs), across the prototypes of the codebook that best explain the co-occurrence matrix. Once these distributions are determined, each distribution is treated as another set of prototypes and their co-occurrence matrix is estimated. The process is repeated until a stopping criterion is reached.

The root of the tree represents the universal pmf, including every prototype in proportion to how often it occurred. At each branch in the tree, the pmf is broken into two separate pmfs that are relatively dissimilar. This process does not necessarily guarantee that the two pmfs will sum to the parent pmf.

At each branch, we initialize two random pmfs with two priors, π_1 and π_2, and use the pmfs and priors to create an estimate of the co-occurrence matrix,

$$\hat{C}_{i,j} = \sum_{c=1}^{N} \pi_c * p_c(i) * p_c(j), \tag{13}$$

and iteratively reestimate the parameters to minimize the sum squared error[6]

$$E = \sum_{i,j} (C_{i,j} - \hat{C}_{i,j})^2. \tag{14}$$

The update rules that minimize the error function with respect to our model parameters are

6. Arguably, the Kullback-Leibler (KL) distance, $(\sum_{i,j}(C_{i,j} ln \frac{C_{i,j}}{\hat{C}_{i,j}}))$ would be more appropriate in comparing distributions. We are currently investigating this and other error functions and the update rules which result.

$$\pi_c = (1 - \alpha_\pi) * \pi_c + (\alpha_\pi) * \sum_{i,j} (C_{i,j} - \hat{C}_{i,j}) p_c(i) * p_c(j) \quad (15)$$

and

$$p_c(i) = (1 - \alpha_p) * p_c(i) + (\alpha_p) * \sum_{j} (C_{i,j} - \hat{C}_{i,j}) * p_c(j), \quad (16)$$

where c is the class ($c \in \{0, 1\}$) and the learning factor for the priors, α_π, is higher than the learning factor for the pmfs, α_p. It is sometimes useful to put soft constraints on the priors to insure both distributions represent significant portions of the co-occurrence data.

At each branch, the parent distribution is used to estimate the co-occurrences that result from that class. The co-occurrence for the left branch subproblem would be derived from the original co-occurrence, C, and the left pmf, $p_0(.)$, as follows:

$$C^0_{i,j} = C_{i,j} * p_0(i) * p_0(j). \quad (17)$$

C^0 is used to determine the children pmfs of $p_0(.)$, $p_{00}(.)$, and $p_{01}(.)$. For example, if a pmf was uniform over half the prototypes, the co-occurrence matrix used for its children would include only the co-occurrences between those prototypes. If this was not done, every branch may result in the same pmfs as the initial branch.

Once the parameters for the pmfs have been determined, any exclusive set of them can be used as classifiers. An exclusive set of prototypes can be determined by using the leaf nodes of any pruned version of the binary tree. We prune after any node whose children's distributions similarity exceeds a threshold, although Fig. 7 shows a complete tree before the pruning for evaluation purposes. All leaf nodes are associated with a probability distribution across the prototypes that can now be used to classify sequences.

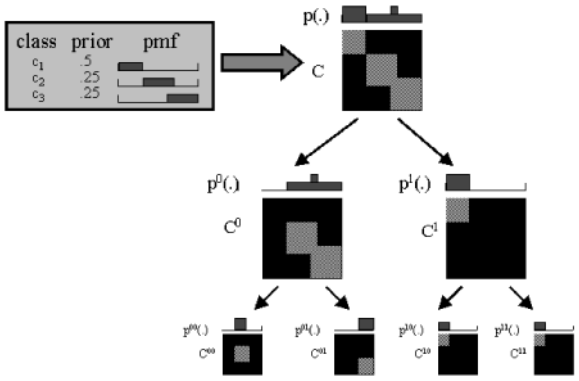


Fig. 6. This figure shows a synthetic classification example with three underlying classes shown in the upper left. The first branch separates the class whose pmf doesn't have any overlap from the other two, p^1. That separable class cannot be further separated. The other two class pmfs are separated at the next branch (into p^{00} and p^{01}).

7.4 Classifying a Sequence

Each observation in a sequence is treated as an independent observation. Thus, the probability of a particular class is the product of the probabilities of that class producing each of the observations in the sequence. This can be computed by using the dot product of the log of the pmfs (with prior) with the accumulated prototype histogram from the sequence. Note that if the prototypes were split into two distinct classes, even observations which mapped to extremely ambiguous prototypes would count toward one class or the other in equal proportion to the definitive examples.

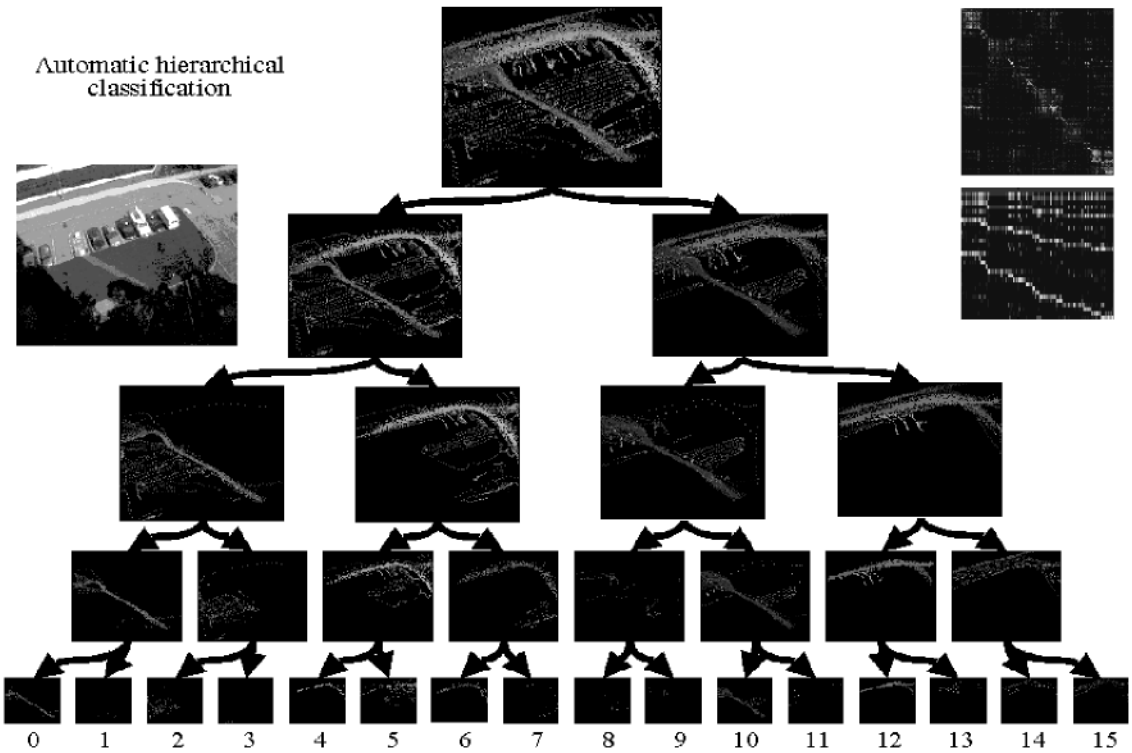
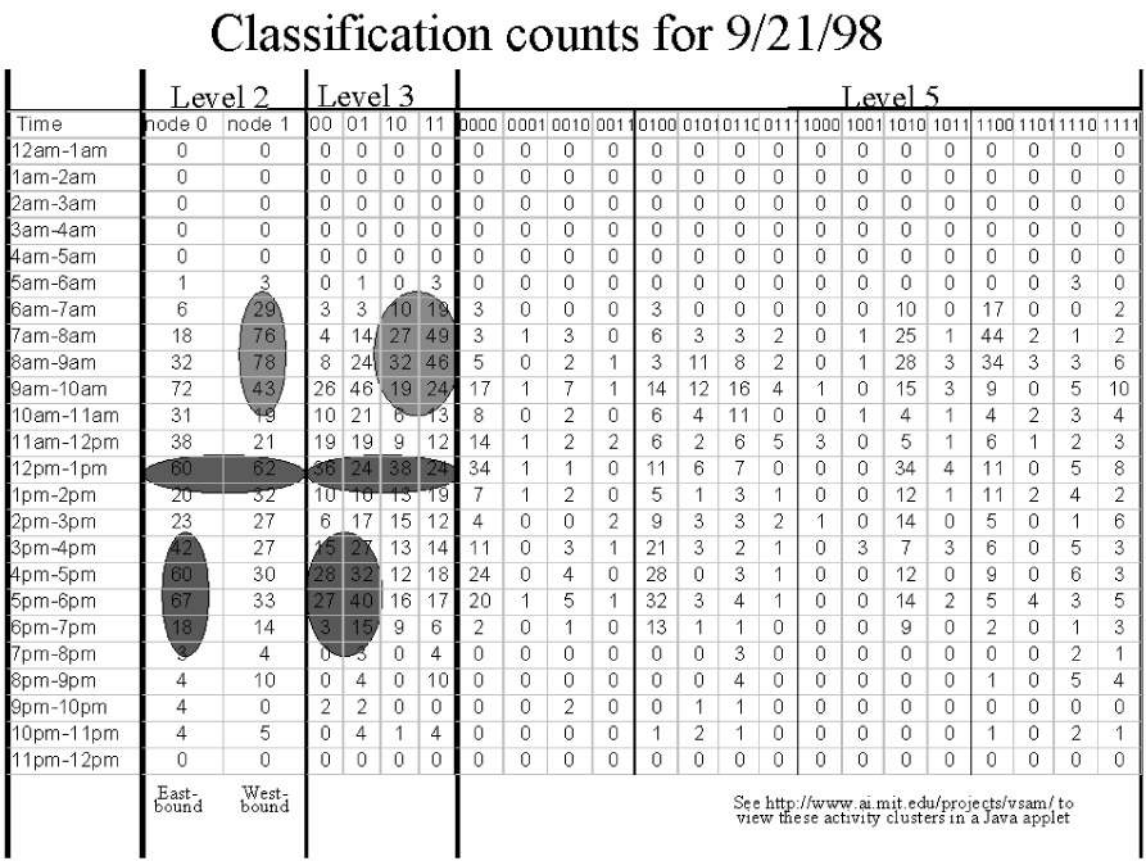


Fig. 7. This figure shows an image of the scene (upper left), the classification hierarchy (center), and the co-occurrence matrix and normalized pmfs (upper right) for each element of the tree. The scene contains a road with adjacent parking spots and a path through the grass near the loading bay of our building. The binary tree shows accumulated motion templates for each node of the tree. And the co-occurrence matrix and normalized pmfs show which prototypes occurred within the same sequences and the probability distributions for each node in the tree (ordered breadth-first). The final level of the tree specific classes including: pedestrians on the path (one class in each direction), pedestrians and lawn-mowers on the lawn, activity near the loading dock, cars, trucks; etc. These classes can be viewed in a Java 1.1 compatible browser at: http://www.ai.mit.edu/projects/vsam/Classification/. Note: the columns and rows of the co-occurrence matrix have been ordered to make some of its structure more apparent.



Classification counts for 9/21/98

	Level 2		Level 3				Level 5															
Time	node 0	node 1	00	01	10	11	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
12am-1am	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1am-2am	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2am-3am	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3am-4am	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4am-5am	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5am-6am	1	3	0	1	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0
6am-7am	6	29	3	3	10	19	3	0	0	0	3	0	0	0	0	0	10	0	17	0	0	2
7am-8am	18	76	4	14	27	49	3	1	3	0	6	3	3	2	0	1	25	1	44	2	1	2
8am-9am	32	78	8	24	32	46	5	0	2	1	3	11	8	2	0	1	28	3	34	3	3	6
9am-10am	72	43	26	46	19	24	17	1	7	1	14	12	16	4	1	0	15	3	9	0	5	10
10am-11am	31	19	10	21	6	13	8	0	2	0	6	4	11	0	0	1	4	1	4	2	3	4
11am-12pm	38	21	19	19	9	12	14	1	2	2	6	2	6	5	3	0	5	1	6	1	2	3
12pm-1pm	60	62	36	24	38	24	34	1	1	0	11	6	7	0	0	0	34	4	11	0	5	8
1pm-2pm	20	32	10	10	13	19	7	1	2	0	5	1	3	1	0	0	12	1	11	2	4	2
2pm-3pm	23	27	6	17	15	12	4	0	0	2	9	3	3	2	1	0	14	0	5	0	1	6
3pm-4pm	42	27	15	27	13	14	11	0	3	1	21	3	2	1	0	3	7	3	6	0	5	3
4pm-5pm	60	30	28	32	12	18	24	0	4	0	28	0	3	1	0	0	12	0	9	0	6	3
5pm-6pm	67	33	27	40	16	17	20	1	5	1	32	3	4	1	0	0	14	2	5	4	3	5
6pm-7pm	18	14	3	15	9	6	2	0	1	0	13	1	1	0	0	0	9	0	2	0	1	3
7pm-8pm	8	4	0	8	0	4	0	0	0	0	0	0	3	0	0	0	0	0	0	0	2	1
8pm-9pm	4	10	0	4	0	10	0	0	0	0	0	0	4	0	0	0	0	0	1	0	5	4
9pm-10pm	4	0	2	2	0	0	0	0	2	0	0	1	1	0	0	0	0	0	0	0	0	0
10pm-11pm	4	5	0	4	1	4	0	0	0	0	1	2	1	0	0	0	0	0	1	0	2	1
11pm-12pm	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	East-bound	West-bound																				

Fig. 8. This figure shows how many of the activities were detected on a particular day. The first two columns correspond to the initial branch. The following four columns correspond to the next level of the binary classification tree. The last eight columns are the leaf nodes of the classification tree. Below some of the columns the primary type of activity for that node is listed. Morning rush hour is highlighted in green (light gray) and shows traffic moving mostly in one direction. The lunch-time pedestrian traffic is highlighted in red (gray). The evening rush hour is highlighted in blue (dark gray) and shows more movement in the opposite direction as the morning rush hour.

7.5 A Simple Example

Fig. 6 shows a synthetic example. Using the predefined classes and priors, a root co-occurrence matrix can be formed. At each branch, the pmf is broken into two pmfs which best explain the observed joint co-occurrences. The classification hierarchy behaves as would be expected, first breaking apart the class which never presents like the other two classes, then breaking remaining two classes.

8 Results

The following two examples involve creating a classification hierarchy using the same number of prototypes, the same learning parameters, and the same sequences produced by our tracking system. The only difference is that they use different representations. The first example classifies activity based on a 5-tuple (image position, speed, direction, and size). The second example classifies shape based on a 1,024-tuple (32×32 binary silhouettes).

8.1 Classifying Activities

This example classifies objects based on a representation of their position, speed, direction and size (x, y, dx, dy, s). First, 400 representative prototypes are determined. Each prototype represents all the objects of a particular size that are seen in a particular area of a scene moving in a particular direction. Cooccurrences are accumulated using 24 hours of sequences from that scene. Finally, the universal pmf (the true pmf of the entire set of sequences) is probabilistically broken into two pmfs.

The process is repeated to produce a binary tree of height 4 detailed in Fig. 7. Fig. 8 shows the history of one particular day.

Note that the scene contains a road with adjacent parking spots and a path through the grass near the loading bay of our building. The binary tree shows accumulated motion templates for each node of the tree. The first break separates traffic moving in one direction around the building and traffic moving in the other direction because objects in this scene did not generally change their direction. The second break for both branches separates traffic on the road and traffic on the path. While there are some prototype states which are common to both activities, these two activities were significantly different and accounted for a significant amount of the data. Further bifurcations result in classes for: pedestrians on the path, pedestrians and lawn-mowers on the lawn, activity near the loading dock, cars, trucks, etc. These classes can be viewed in a Java 1.1 compatible browser at: http://www.ai.mit.edu/projects/vsam/Classification/.

Fig. 10 shows the distribution of events over a 24 hour period, highlighting the changes in density of pedestrian and vehicular traffic as a function of time.

8.2 Classifying Motion Silhouettes

While this example results in a rather simple classification, it illustrates an intended use for this type of classification. VQ resulted in 400 silhouettes of vehicles and people. The first break broke the silhouettes into two relatively discrete classes, people, and vehicles. Some of the more blurry prototypes remained ambiguous because they matched both vehicles and people. These prototypes were shared between the two classes. Fig. 9 shows the co-occurrence matrix, the pmfs, and some examples of prototypes from both classes.

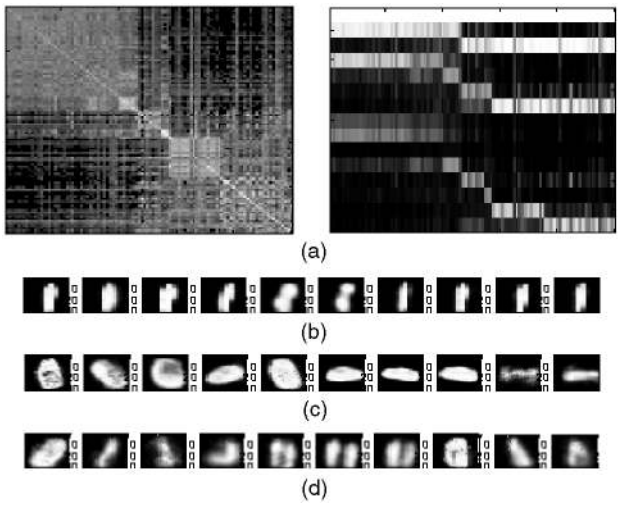


Fig. 9. (a) shows the co-occurrence matrix and resulting pmfs. Some of the prototypes from the person class (b), vehicle class (c), and some prototypes which were significantly ambiguous (d). In C, the upper left corresponds to silhouettes of people and the lower right corresponds to silhouettes of vehicles. The vehicles show less statistical independence because vehicles in this particular scene were only seen as they passed through particular orientations. If the scene contained vehicles driving in circles, the corresponding prototypes would exhibit more independence. Note: the co-occurrence matrix has been ordered to make some of its structure more apparent.

Fig. 10 shows classification of a day of silhouette sequences. After setting the similarity parameter for pruning, the resulting classifier first separated vehicles as they were decisively different from the other silhouettes. This means that while vehicles appeared at many different angles within their sequences, few sequences contained both vehicles and people. The next break was individual pedestrians. Then, the last break removed groups of pedestrians from clutter and lighting effects.

The daily activity histograms show some interesting facts. The highest occurrences of people and cars was in the morning and evening as expected. Groups of people tended to occur most shortly after noon. The clutter was primarily trees, garbage, and lighting effects on the sides of buildings. The histogram and images show that it was a very windy morning and the lighting effects occurred near dusk.

9 Detecting Unusual Events

Often, a particular scene will contain events which have never occurred or occur so rarely that they are not represented in the clustered activities. In many cases, it is these events that are of most interest.

Because we can build representations of common patterns in a site, we are able to use that information to detect uncommon patterns. We have done some preliminary work on determining unusual activities as they occur. Our system measures two aspects of how usual each track is. First, it measures the typicality of each of the instantaneous states using the codebook as a density approximator. Second, it looks at the co-occurrences exhibited by the sequence in relation to the accumulated co-occurrence statistics. Both measures can provide evidence in support of an unusual event and we are currently developing this work and determining methods by which to evaluate them.

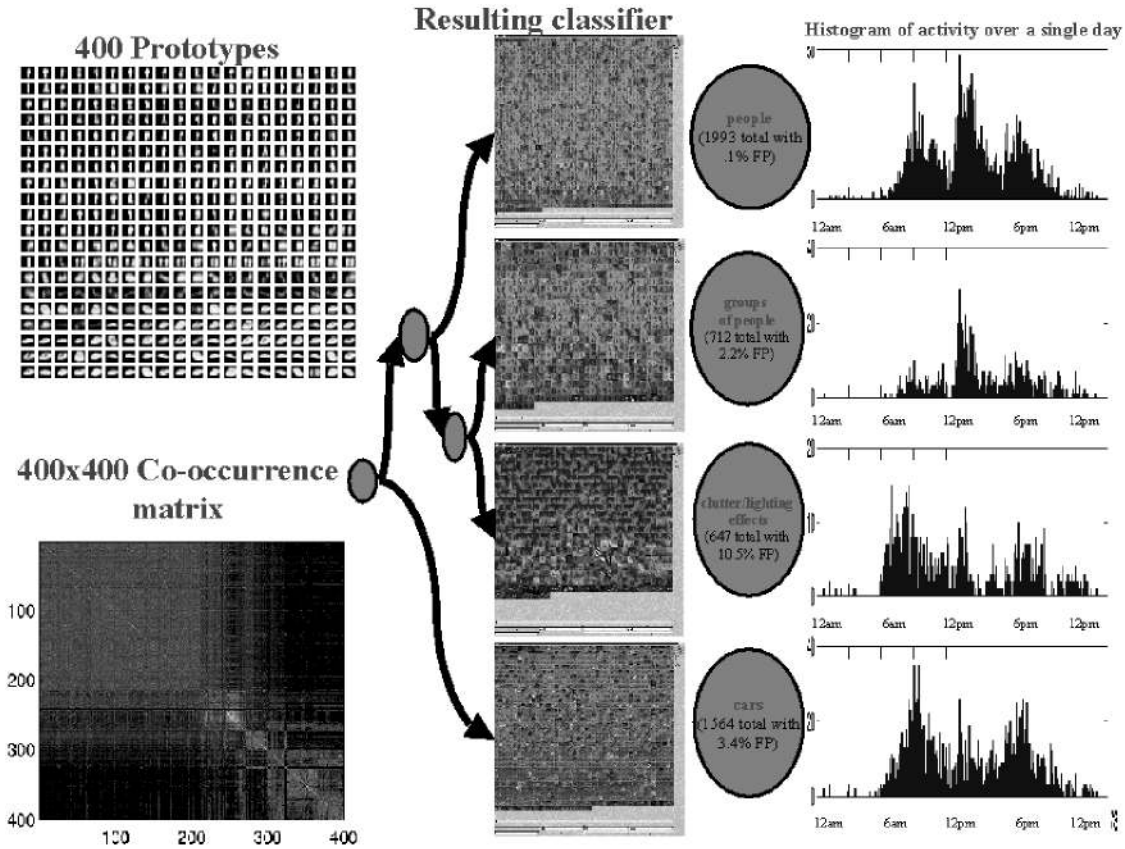


Fig. 10. On the left are the 400 silhouette prototypes and the co-occurrence matrix that resulted from a day's worth of tracking sequences. In the middle is the classification hierarchy which resulted, images of all occurrences of each class, and description of the classes, as well as their performance relative to those descriptions. On the right are 24-hour histograms of the occurrences of each class. See web page for more higher quality images.

10 Classification Shortcomings and Future Work

Admittedly, the scene that these classifiers were derived from was well-suited to this problem. Some scenes containing the same types of objects would have resulted in classification hierarchies without as much structure. For example, if cars drove straight through the scene on two separate roads, there may be no sequences of cars moving from one road to the other. Without such evidence, there is no reason to expect that the two resulting classes would be near each other in the hierarchy. Unless the extended scene with multiple cameras shows that two representations are similar, it will have to be told by a short supervision process following training.

The most obvious weakness of this algorithm is the need to discretize complex input spaces. We are currently investigating automatically deriving local feature sets using VQ on subimages and learning those features similarities using local (in time and space) co-occurrence measurements. Doing this hierarchically holds promise for learning useful feature sets and better prototypes.

This could also be useful for texture segmentation. For example, create 10,000 texture prototypes and define their similarity based on which prototypes occur near other prototypes (spatially and temporally). Learning similarities this way, rather than attempting to assert a prior for which textures are similar, takes advantage of domain specific regularities and could define regularities in domains where it is not certain how similar two textures are.

Of course, assumed similarities are useful, particularly in cases where there is not enough data. In such cases, the C^{ttotal} can be seeded with a co-occurrence matrix. Hence, prototypes without sufficient representation will assume the similarities they are given, while the similarities of the prototypes which are observed often are determined by the data.

Finally, we are investigating using both the prototypes and the co-occurrences to detect outliers. If many data points in a sequence are not represented by a prototype, it may be an unusual event. Also, if a sequence's co-occurrences are very unlikely given the joint co-occurrences, it is likely to be unusual.

Anomaly detection and classification in general would be greatly enhanced by learning context cycles. If we could learn a traffic light cycle, we could detect that cars running the light are unusual, even though their pattern of activity was not unusual. If we could learn daily cycles, our models could contain specific prototypes for day and night (e.g., headlights vs. full vehicles). Also, only deliveries made at night may be unusual.

11 Conclusions

This paper has shown a novel, probabilistic method for background subtraction. It involves modeling each pixel as a separate mixture model. We implemented a real-time approximate method which is stable and robust. The method requires only two parameters, α and T. These two parameters are robust to different cameras and different scenes.

This method deals with slow lighting changes by slowly adapting the values of the Gaussians. It also deals with multimodal distributions caused by shadows, specularities, swaying branches, computer monitors, and other troublesome features of the real world which are not often mentioned in computer vision. It recovers quickly when background reappears and has an automatic pixel-wise threshold. All these factors have made this tracker an essential part of our activity and object classification research.

This system has been successfully used to track people in indoor environments, people and cars in outdoor environments, fish in a tank, ants on a floor, and remote control vehicles in a lab setting. All these situations involved different cameras, different lighting, and different objects being tracked. This system achieves our goals of real-time performance over extended periods of time without human intervention.

We have also motivated and implemented a new approach to automatic object classification. This approach has shown promise with two contrasting classification problems. In one case, it produced a nonparametric activity classifier. In the other case, it produced an binary image-based classifier. We are currently investigating many other possible uses for this method.

Acknowledgments

This research is supported in part by a grant from the US Department of Army Research Projects Agency (DARPA) under contract N00014-97-1-0363 administered by the US Office of Naval Research (ONR) and in part by a grant jointly administered by DARPA and ONR under contract N00014-95-1-0600.

References

[1] D. Beymer, P. McLauchlan, B. Coifman, and J. Malik, "A Real-Time Computer Vision System for Measuring Traffic Parameters," *Proc. Computer Vision and Pattern Recognition,* June 1997.

[2] R. Collins, A. Lipton, and T. Kanade, "A System for Video Surveillance and Monitoring," *Proc. Am. Nuclear Soc. (ANS) Eighth Int'l Topical Meeting Robotic and Remote Systems,* Apr. 1999.

[3] L. Davis et al., "Visual Surveillance of Human Activity," *Proc. Asian Conf. Computer Vision (ACCV98),* Jan. 1998.

[4] A Dempster, N. Laird, and D. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *J. Royal Statistical Soc.,* vol. 39 (Series B), pp. 1-38, 1977.

[5] N. Friedman and S. Russell, "Image Segmentation in Video Sequences: A Probabilistic Approach," *Proc. 13th Conf. Uncertainty in Artificial Intelligence (UAI),* Aug. 1997.

[6] A. Gersho and R.M. Gray, *Vector Quantization and Signal Compression.* Kluwer Academic, 1991.

[7] W.E.L. Grimson, C. Stauffer, R. Romano, and L. Lee, "Using Adaptive Tracking to Classify and Monitor Activities in a Site," *Computer Vision and Pattern Recognition (CVPR 98),* June 1998.

[8] B.K.P. Horn, *Robot Vision,* pp. 66-69, 299-333. MIT Press, 1986.

[9] I. Haritaoglu, D. Harwood, and L.S. Davis, "W^4: Who? When? Where? What? A Real Time System for Detecting and Tracking People," *Proc. Third Int'l Conf. Automatic Face and Gesture Recognition,* Apr. 1998.

[10] I. Horswill and M. Yamamoto, "A $1000 Active Stereo Vision System," *Proc. IEEE/IAP Workshop Visual Behaviors,* Aug. 1994.

[11] I. Horswill, "Visual Routines and Visual Search: A Real-Time Implementation and Automata-Theoretic Analysis," *Proc. Int'l Joint Conf. AI,* 1995.

[12] Y. Ivanov, A. Bobick, and J. Liu, "Fast Lighting Independent Background Subtraction," Technical Report no. 437, MIT Media Laboratory, 1997.

[13] N. Johnson and D.C. Hogg, "Learning the Distribution of Object Trajectories for Event Recognition," *Proc. British Machine Vision Conf.,* D. Pycock, ed., pp. 583-592, Sept. 1995.

[14] N. Johnson and D.C. Hogg, "Learning the Distribution of Object Trajectories for Event Recognition," *Image and Vision Computing,* vol. 14, no. 8, pp. 609-615, Aug. 1996.

[15] T. Kanade, "A Stereo Machine for Video-Rate Dense Depth Mapping and Its New Applications," *Proc. Image Understanding Workshop,* pp. 805-811, Feb. 1995.

[16] D. Koller, J. Weber, T. Huang, J. Malik, G. Ogasawara, B. Rao, and S. Russel, "Towards Robust Automatic Traffic Scene Analysis in Real-Time," *Proc. Int'l Conf. Pattern Recognition,* Nov. 1994.

[17] K. Konolige, "Small Vision Systems: Hardware and Implementation," *Proc. Eighth Int'l Symp. Robotics Research,* Oct. 1997.

[18] L. Lee, R. Romano, and G. Stein, "Monitoring Activities from Multiple Video Streams: Establishing a Common Coordinate Frame," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 22, no. 8, pp. 758-767, Aug. 2000.

[19] A. Lipton, H. Fujiyoshi, and R.S. Patil, "Moving Target Classification and Tracking from Real-Time Video," *Proc. IEEE Workshop Applications of Computer Vision (WACV),* pp. 8-14, Oct. 1998.

[20] N. Oliver, B. Rosario, and A. Pentland, "A Bayesian Computer Vision System for Modeling Human Interactions," *Proc. Int'l Conf. Vision Systems '99,* Jan. 1999.

[21] C. Ridder, O. Munkelt, and H. Kirchner, "Adaptive Background Estimation and Foreground Detection Using Kalman-Filtering," *Proc. Int'l Conf. Recent Advances in Mechatronics, ICRAM '95,* pp. 193-199, 1995.

[22] J. Shi and J. Malik, "Normalized Cuts and Image Segmentation," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* June 1997.

[23] C. Stauffer and W.E.L. Grimson, "Adaptive Background Mixture Models for Real-Time Tracking," *Proc. Computer Vision and Pattern Recognition 1999 (CVPR '99),* June 1999.

[24] C.R. Wren, A. Azarbayejani, T. Darrell, and A. Pentland, "Pfinder: Real-Time Tracking of the Human Body," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 19, no. 7, pp. 780-785, July 1997.