Support Vector Tracking

Shai Avidan

Abstract—Support Vector Tracking (**SVT**) integrates the Support Vector Machine (**SVM**) classifier into an optic-flow-based tracker. Instead of minimizing an intensity difference function between successive frames, **SVT** maximizes the **SVM** classification score. To account for large motions between successive frames, we build pyramids from the support vectors and use a coarse-to-fine approach in the classification stage. We show results of using **SVT** for vehicle tracking in image sequences.

Index Terms—Support vector machines, optic-flow, visual tracking.

1 Introduction

Tracking algorithms find how an image region moves from one frame to the next. This implies the existence of an error function to be minimized, such as the sum of squared differences (**SSD**) between the two image regions. The **SSD** error function makes the "constant brightness assumption," i.e., the brightness of the object does not change from frame to frame. This paradigm makes no assumptions about the class of the tracked object. Yet, quite often, we are interested in tracking a particular class of objects such as people or vehicles. In this case, we can train a classifier in advance to distinguish between an object and the background. The question then is how to integrate the tracker and the classifier. One approach is to use the tracker and the classifier sequentially. The tracker will find where the object moved to and the classifier will give it a score. This scheme will repeat until the classification score will fall below some predefined threshold. The disadvantage of such an approach is that the tracker is not guaranteed to move to the best location (the location with the highest classification score) but rather find the best matching image region. Furthermore, such an approach relies heavily on the first frame. Even if a better image for classification purposes will appear later in the sequence, the tracker will not lock on it as it tries to minimize the **SSD** error with respect to the first image which might have a low classification score. Our solution is to replace the error function of the tracker. Instead of minimizing the **SSD** error, the tracker will try to maximize the classification score. This way, all the prior knowledge, captured by the classifier, is integrated directly into the tracking process.

Our system detects and tracks the rear-end of vehicles from a video sequence taken by a forward looking camera mounted on a moving vehicle (see Fig. 1). Vehicles cannot be well-spanned by eigenimages and, therefore, the detection module of the application (which is not described in this paper) relies on an **SVM** that was trained on thousands of images of vehicles and nonvehicles. Once detected, the system tracks the vehicle over time. Naturally, we would like to leverage the power of the classifier for tracking.

• *The author is with MobilEye Vision Technologies LTD, 10 Hartom Street, Jerusalem, Israel. E-mail: avidan@mobileye.com.*

Manuscript received 17 Jan. 2003; revised 25 Aug. 2003; accepted 19 Dec. 2003.
Recommended for acceptance by S. Soatto.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 118158.

In this paper, we fuse together an optic-flow-based tracker and an **SVM** classifier. Recall that an optic-flow-based tracker [1] is a gradient-descent method to search for transformation parameters that minimize the intensity difference between a pair of successive frames. **SVM** [21] is a general classification scheme that has been successfully used in the past [3], [12], [17]. Independently, both the tracker and the classifier are sensitive to viewpoint changes. As an object moves in the image plane, some parts of it might appear, disappear, or merge with the background, making it hard for the tracker to keep tracking it. The classifier, on the other hand, might be sensitive to small transformations in the image plane so that even a small misalignment of the tracker might cause the classifier to reject the tracked vehicle.

SVT combines the computational efficiency of optic-flow-based tracking with the power of a general classifier **SVM**, extending the power of both the tracker and the classifier. The new tracker now relies on the power of the classifier to determine the position of the object in the next frame, even if some parts of the vehicle appear or disappear. This is because now the new frame is not matched against the previous frame, but against all the patterns that the classifier was trained on. The classifier, on the other hand becomes much more robust to small transformations in the image plane, as these parameters are estimated efficiently in runtime as part of the testing process. The result is a real-time tracking algorithm that we show is capable of tracking vehicles over long periods of time.

2 Related Work

The problem of fusing detection and tracking has been investigated in the past, with approaches ranging from pure tracking, without any aid from a classifier, to pure detection methods that avoid tracking all together.

Smith and Brady [18] developed a system for vehicle tracking, among other things, that works by segmenting the optic-flow field into independently moving regions and tracking each region separately. The system runs in real time, but does not use any classifier to validate that the tracked object is indeed a vehicle.

Gong et al. [7] extend the eigenfaces approach to recognize a person from a video sequence. However, tracking and detection are separated. The detection involves

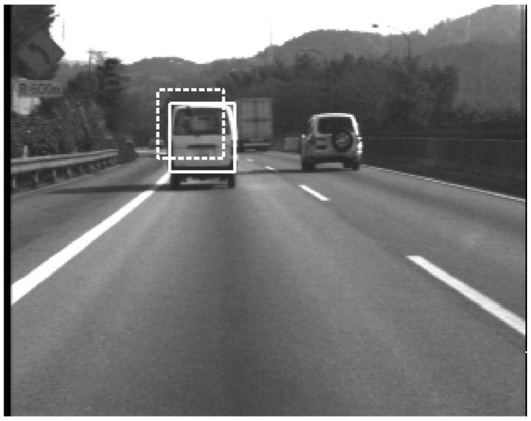

Fig. 1. Illustration of **SVT** operation. **SVT** takes as input the initial guess of the position of the vehicle (dashed rectangle) and finds the position with the highest **SVM** score (solid rectangle).

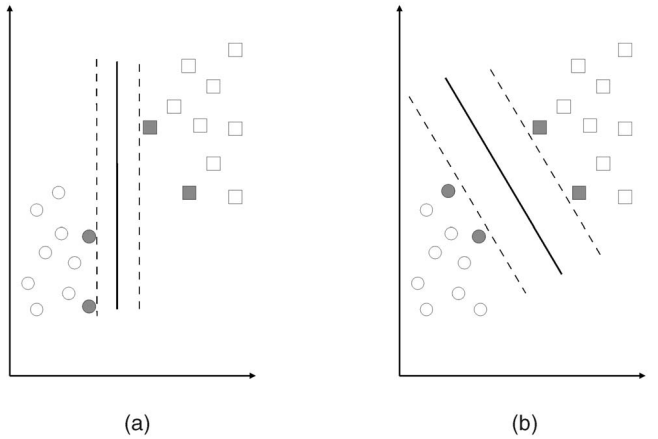

Fig. 2. (a) A separating hyperplane with small margin. (b) A separating hyperplane with a large margin. A better generalization capability is expected from (b). The filled circles and rectangles are termed "support vectors."

spatio-temporal image filters, followed by ellipse fitting, while tracking consists of Kalman filtering. The detected face in each frame is projected on eigenface subspace and this "temporal signature" is fed to a neural-network that determines the identity of the person. Note that the tracking process is *not* guaranteed to find the best head position for the purpose of person recognition.

Black and Jepson [2] integrated eigenimages into an optic-flow-based tracker calling it Eigentracking. Eigentracking works by minimizing the distance between the image region and a predefined set of eigenvectors (instead of minimizing the **SSD** error between successive frames). To account for larger motions, the entire scheme is implemented in a coarse-to-fine manner, using *EigenPyramids*. However, eigenimages are not a general classification scheme and so far they have been applicable mainly for faces. Support Vector Machines (**SVM**), on the other hand, was proven useful in a wide variety of applications, including character recognition [17], face detection [12], text classification [8], and medical applications [10] to name a few.

In machine learning, the problem tackled by **SVT** is often referred to as transformation invariance (i.e., how to make the classification process insensitive to transformations in the image plane). Simard et al. [19] used the nearest-neighbor classifier with a "tangent distance" metric, the key idea being that the set of all images of a transformed image forms a nonlinear manifold in some high-dimensional space that can be locally approximated by a "tangent plane." The distance between an example image and test image is then defined as the distance between their tangent planes and not as the distance between the two images. This approach was successfully applied for character recognition purposes. Later, Vasconcelos and Lippman [11] worked on face images (that are much larger than character images) by extending "tangent distance" to work in a multiresolution framework. Scholkopf et al. [16] introduced "tangent distance" to **SVM** by deriving a kernel that enforces a local transformation invariance. This has the advantage of keeping the classification speed high at the cost of complicating the learning stage. Moreover, it is not clear how multiresolution treatment can be incorporated into their scheme.

Finally, several researchers [22], [9], [13], [14], [15], [20] perform object detection on every frame, ignoring object tracking all together. These methods are extremely efficient in rejecting nonfaces but are computationally expansive in regions that contain faces.

Our approach is similar to that of [2] in that we find the transformation parameters as part of the classification stage. We extend their work by using **SVM** instead of eigenvectors. This does not complicate the learning phase to achieve transformation invariance and falls naturally within a multiresolution framework.

3 Support Vector Machines

We denote vectors using bold face fonts $\mathbf{x}$ to distinguish them from scalars x. In the equations, we assume the image data to be vectorized.

SVM classifiers find a separating hyperplane that maximizes the margin between two classes (see Fig. 2), where the margin is defined as the distance of the closest point, in each class, to the separating hyperplane. This is equivalent to performing structural risk minimization to achieve good generalization. See [21], [4] for a detailed description.

Given a data set $\{\mathbf{x_i}, y_i\}_{i=1}^{l}$ of l examples $\mathbf{x_i}$ with labels $y_i \in \{-1, +1\}$, finding the optimal hyperplane implies solving a constrained optimization problem using quadratic programming, where the optimization criterion is the width of the margin between the classes. The separating hyperplane can be represented as a linear combination of the training examples and classifying a new test pattern $\mathbf{x}$ is done using the following expression:

$$f(\mathbf{x}) = \sum_{i=1}^{l} \alpha_i y_i k(\mathbf{x}, \mathbf{x_i}) + b,$$

where $k(\mathbf{x}, \mathbf{x_i})$ is a kernel function and the sign of $f(\mathbf{x})$ determines the class membership of $\mathbf{x}$. Constructing the optimal hyperplane is equivalent to finding the nonzero α_i. Any data point $\mathbf{x_i}$ corresponding to nonzero α_i is

termed "support vector." Support vectors are the training patterns closest to the separating hyperplane and the kernel function extends SVM to handle nonlinear separating hyperplanes. Popular kernel functions include $k(\mathbf{x}, \mathbf{x_j}) = exp(-\frac{\|\mathbf{x}-\mathbf{x_j}\|^2}{2\sigma^2})$ which leads to a Gaussian RBF and $k(\mathbf{x}, \mathbf{x_j}) = (\mathbf{x}^T\mathbf{x_j} + 1)^d$ which represent polynomial of degree d.

4 Support Vector Tracking

Tracking algorithms assume some divine intervention to supply them with an initial guess as to the position of the object to be tracked. But, a working system must take care of the detection step as well. We take advantage of the fact that vehicles are manmade objects that contain strong horizontal and vertical edges. We convert the input image to an edge map and search for regions that have strong edges. This search produces a list of candidate image patches that are further analyzed. The edge information is not enough to distinguish between vehicles and nonvehicles, so we use an **SVM** classifier, on the input image, to determine if the candidate image patch is indeed a vehicle. The combination of the fast edge-based attention mechanism followed by the **SVM** validation step gives a fast and accurate detection module.

Our system uses a forward looking camera, mounted on a moving vehicle to detect and track the rear end of moving vehicles. Once the vehicle is detected and approved by the **SVM** classifier, it is tracked over time. Since much effort was already put into training the classifier, it is only natural to try to use it for other tasks as well, tasks such as tracking.

The framework in which **SVT** will work is as follows: The detection module will detect possible candidates in the current frame and hand them over to the **SVT**. The **SVT** will refine their position so that a local maximum of **SVM** score is achieved. If the score is positive, the candidate will be declared a vehicle and a tracking process will start. The refined position in the current frame will serve as the initial guess in the next frame, etc.

We develop **SVT** for the simple case of 2D translation model. Then, we extend **SVT** to work on pyramids by introducing the Support Vector Pyramid. Extensions to a more general transformation model (such as 2D affine transformation) are straightforward and will not be presented here.

4.1 Support Vector Tracking

SVM classification is given by

$$\sum_{j=1}^{l} y_j \alpha_j k(\mathbf{I}, \mathbf{x_j}) + b, \tag{1}$$

where $\mathbf{x}_j$ are the support vectors, y_j are their sign, and α_j are their Lagrange multipliers. $k(\mathbf{I}, \mathbf{x}_j)$ is the kernel we choose to use and $\mathbf{I}$ is the image region we wish to test. If the above expression is positive, then the image region $\mathbf{I}$ is considered as a vehicle and, if the expression is negative, then $\mathbf{I}$ is considered as a nonvehicle.

Now, let $\mathbf{I_{init}}$ represent some initial guess of the position of the object in a given image. Furthermore, let us assume that the initial guess is not too distant from the correct position of the object, defined as $\mathbf{I_{final}}$ (Fig. 1). Using first-order Taylor expansion, we have that

$$\mathbf{I_{final}} = \mathbf{I_{init}} + u\mathbf{I_x} + v\mathbf{I_y}, \tag{2}$$

where $\mathbf{I_x}, \mathbf{I_y}$ are the x and y derivatives of (sub)image $\mathbf{I_{init}}$ and u, v are the motion parameters. By assumption, we have that the SVM score of $\mathbf{I_{final}}$ is higher than that of $\mathbf{I_{init}}$. In fact, we assume the SVM score of $\mathbf{I_{final}}$ to be a local maximum. Put formally,

$$\sum_{j=1}^{l} y_j \alpha_j k(\mathbf{I_{final}}, \mathbf{x}_j) = max\left\{ \mathbf{I} | \sum_{j=1}^{l} y_j \alpha_j k(\mathbf{I}, \mathbf{x}_j) \right\}, \tag{3}$$

where $\mathbf{I}$ are all possible (sub)images (in vector form) in the neighborhood of (sub)image $\mathbf{I_{final}}$ (we drop the constant b as it does not affect the solution).

4.1.1 Problem Definition

Plugging (2) into (1), we obtain the formal problem we are trying to solve: *Find the motion parameters u, v to maximize the following expression*

$$\max_{u,v} \sum_{j=1}^{l} y_j \alpha_j k(\mathbf{I} + u\mathbf{I_x} + v\mathbf{I_y}, \mathbf{x}_j), \tag{4}$$

Solving this problem, in the context of tracking, means that, in each frame, we will look for the image region with the highest **SVM** score and *not* the image region most similar to the previous frame.

For readability, we will denote $\mathbf{I_{init}}$ as $\mathbf{I}$ from now on. Taking the derivatives with respect to u and v and setting them to zero will give us a set of (possibly nonlinear) equations to solve. These equations will depend on the particular kernel we use.

4.1.2 Homogeneous Quadratic Polynomials

In this work, we use homogeneous quadratic polynomial given by the kernel $k(\mathbf{x}, \mathbf{x}_j) = (\mathbf{x}^T\mathbf{x}_j)^2$, i.e., take the square of the dot product of the test vector $\mathbf{x}$ and the support vector $\mathbf{x}_j$. The function to be maximized is

$$\begin{aligned} E(u,v) &= \sum_{j=1}^{l} y_j \alpha_j k(\mathbf{I} + u\mathbf{I_x} + v\mathbf{I_y}, \mathbf{x}_j) \\ &= \sum_{j=1}^{l} y_j \alpha_j ((\mathbf{I} + u\mathbf{I_x} + v\mathbf{I_y})^T \mathbf{x}_j)^2 \end{aligned} \tag{5}$$

and taking the u and v derivatives gives:

$$\begin{aligned} \frac{\partial E}{\partial u} &= \sum_{j=1}^{l} y_j \alpha_j \mathbf{I_x}^T \mathbf{x}_j (\mathbf{I} + u\mathbf{I_x} + v\mathbf{I_y})^T \mathbf{x}_j = 0 \\ \frac{\partial E}{\partial v} &= \sum_{j=1}^{l} y_j \alpha_j \mathbf{I_y}^T \mathbf{x}_j (\mathbf{I} + u\mathbf{I_x} + v\mathbf{I_y})^T \mathbf{x}_j = 0 \end{aligned}$$

and, after rearranging terms, we arrive at the following equation:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}, \tag{6}$$

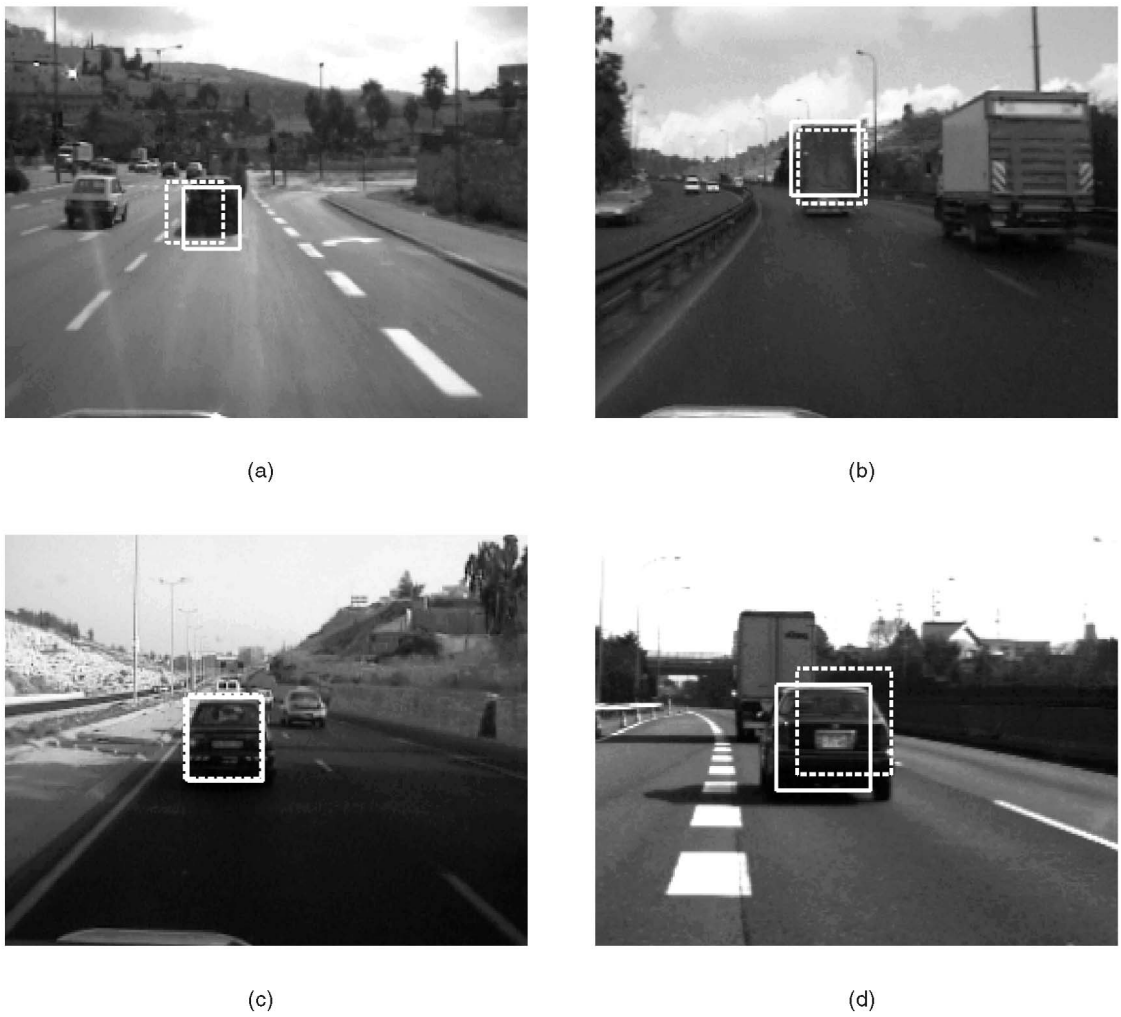

Fig. 3. Examples of the initial guess (dashed line) and the final position (solid line). The image size in all cases is 320×240. Typical object size is 40×40 pixels. The SVM score of the initial guess and the final position as well as the amount of motion between the initial and final position are shown for every example. In example (c), a small change in the position (about one pixel) changed the SVM score from negative to positive. (a) Init: -8.3 Final: 2.9 Motion: (8.8,2.8). (b) Init: -2.1 Final: 2.5 Motion: (-4.5,-2.7). (c) Init: -0.5 Final: 1.0 Motion: (-1.13,1.2). (d) Init: -3.7 Final: 0.2 Motion: (-8.6,4.4).

where

$$A_{11} = \sum_{j=1}^{l} \alpha_j y_j (\mathbf{x_j}^T \mathbf{I_x})^2$$

$$A_{12} = A_{21} = \sum_{j=1}^{l} \alpha_j y_j (\mathbf{x_j}^T \mathbf{I_x})(\mathbf{x_j}^T \mathbf{I_y})$$

$$A_{22} = \sum_{j=1}^{l} \alpha_j y_j (\mathbf{x_j}^T \mathbf{I_y})^2$$

$$b_1 = -\sum_{j=1}^{l} \alpha_j y_j (\mathbf{x_j}^T \mathbf{I_x})(\mathbf{x_j}^T \mathbf{I})$$

$$b_2 = -\sum_{j=1}^{l} \alpha_j y_j (\mathbf{x_j}^T \mathbf{I_y})(\mathbf{x_j}^T \mathbf{I}).$$

These equations resemble the standard optic-flow equations with the support vectors replacing the role of the second image. This means that all computations are done on a single frame each time and not on a pair of successive frames. It is important to emphasize that the particular choice of the homogeneous quadratic polynomial kernel led to these optic-flow look-alike equations. Choosing another kernel, even another polynomial kernel, will lead to a different set of motion equations that might not be linear in the motion parameters.

Using **SVT** is very similar to using optic-flow, with one major difference. In **SVT**, the image region to be tracked must be rescaled to the size of the support vectors. Once the image region is rescaled to the proper size we perform a number of **SVT** iterations, using (6), to maximize the **SVM** score. The iterations stop when no improvement in the score is achieved or after a predefined number of iterations is reached. This approach can handle small motions in the image plane. Larger motions must be handled in a coarse-to-fine manner, as described in Section 4.2 and as described in Section 4.3.

4.2 Pyramid SVT

Each test image that needs to be classified is first subsampled to the size of the support vectors and then its **SVM** score is computed. The goal of **SVT** is to ensure that the test image is aligned such that the maximum **SVM** score is achieved. Thus, **SVT** works on the subsampled version of the test image. The misalignments must be small so that the first-order Taylor approximation, used by **SVT**, will suffice to lock

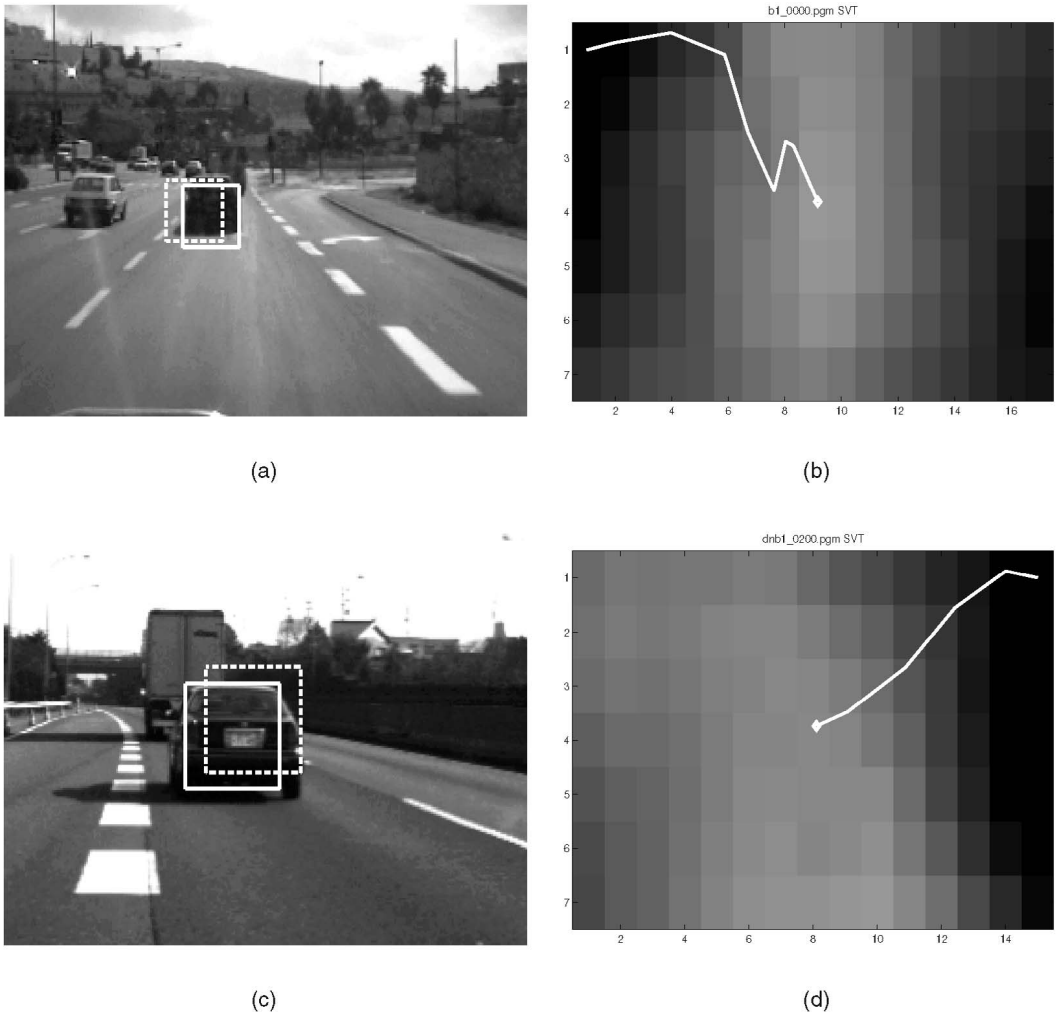

Fig. 4. **SVT** trajectory in parameter space. (a) and (b) correspond to Figs. 3a and 3d where the dashed rectangle marks the initial guess and the solid rectangle shows the final position. (c) and (d) show the trajectory of the **SVT** algorithm in parameter space, (black means lower **SVM** score, the x and y axis are measured in pixels). These plots show how the **SVT** algorithm moves toward the local maxima of the **SVM** score. The final position of the **SVT** algorithm is marked with a diamond. Note how smooth the error surface is, making it suitable for a gradient descent algorithm such as **SVT**.

on the best position. However, if the motions are large, we can no longer hope for the approximation to work and pyramids must be used.

So far, we have treated the support vectors as vectors, not as images. But, recall that the support vectors are a subsampled image of the most problematic images in the learning set and, therefore, we can smooth and subsample them to create a support vector pyramid for each support vector. In the classification stage, we create a pyramid of the same size as the support vector pyramids from the test image. Now, we run **SVT** on successive levels of the pyramids. First, we run **SVT** on the top level of the support vector pyramid and the top level of the test image pyramid. The recovered motion parameters are at the position with the best **SVM** score. This position serves as the initial guess for the **SVT** on the next level of the pyramid and so on until the motion parameters are recovered. The **SVM** score is the score of the bottom level of the pyramid. Ideally, the position with the highest **SVM** score should be the same for all levels, but this is not so because of sampling errors and noise. Nevertheless, the best position in the coarser level serves as a good starting point for the next level.

It is important to emphasize that the pyramid **SVT** does not guarantee that the **SVM** is transformation invariant. All it does is to perform a search for the local maximum in parameter space. This can cause the tracker to get stuck in local maxima, just like any other gradient descent algorithm. The experiments reveal how robust **SVT** is to transformations in the image plane.

4.3 SVT and Optic-Flow-Based Tracking

In case **SVT** uses the homogenous quadratic polynomial kernel, it is possible to combine it with standard optic-flow-based tracking in a straightforward manner. Recall that the standard optic-flow equations are given by:

$$\begin{pmatrix} \sum \mathbf{I_x}^T \mathbf{I_x} & \sum \mathbf{I_x}^T \mathbf{I_y} \\ \sum \mathbf{I_x}^T \mathbf{I_y} & \sum \mathbf{I_y}^T \mathbf{I_y} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} -\sum \mathbf{I_t}^T \mathbf{I_x} \\ -\sum \mathbf{I_t}^T \mathbf{I_y} \end{pmatrix}, \qquad (7)$$

where $\mathbf{I_x}, \mathbf{I_y}$ are the x, y derivatives of the first image and $\mathbf{I_t}$ is the temporal derivative between the two frames.

If we want the motion parameters u, v to balance between similarity to the previous frame (as is done with optic-flow-based tracking) and maximizing the SVM score (as is done with SVT), we can combine (6) and (7) to form an

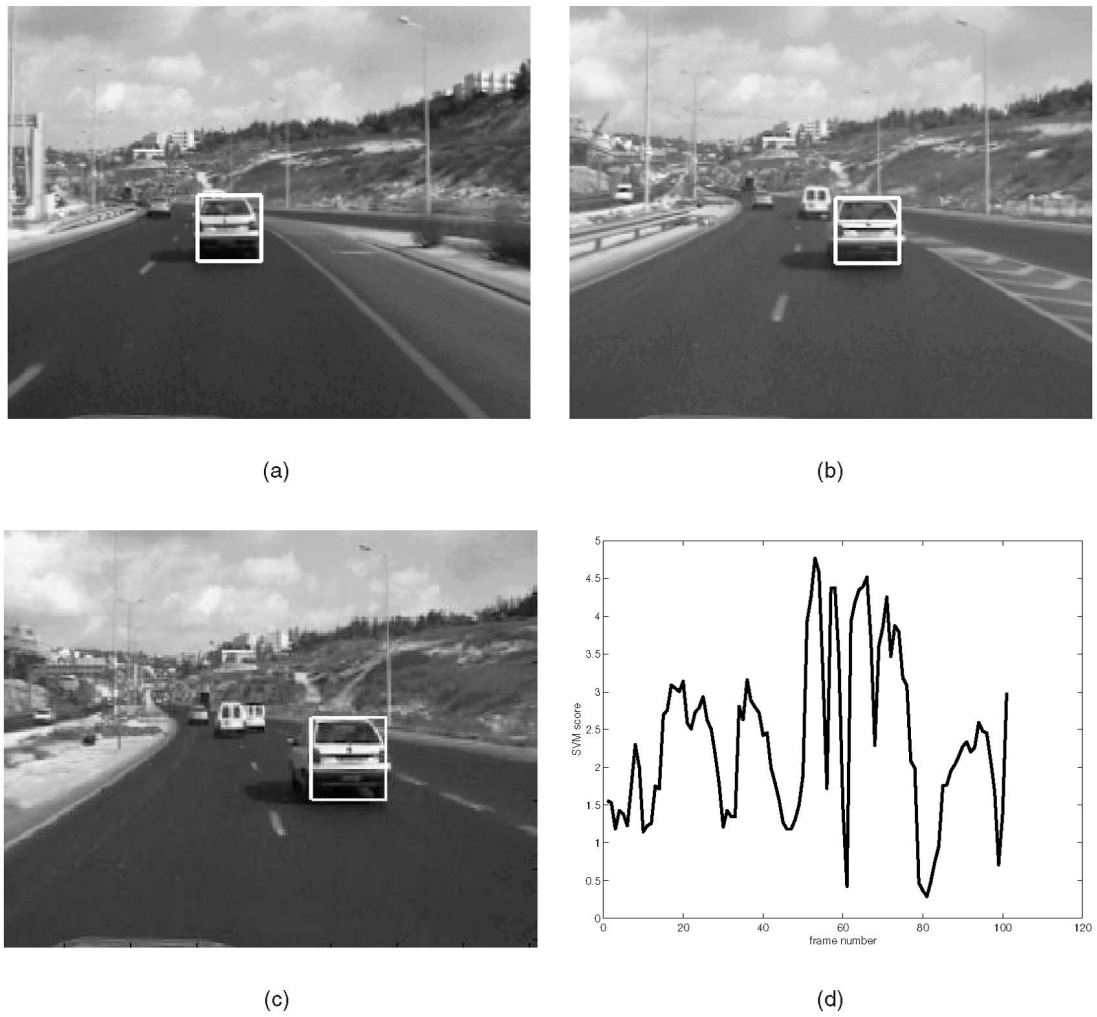

Fig. 5. **SVT**tracking. (a), (b), and (c) are three frames from a 101 frame sequence. (d) shows the **SVM** score of the tracked region using the **SVT** tracker. The x-axis of the graph is the frame number, the y-axis is the **SVM** score. (a) Frame #1. (b) Frame #51. (c) Frame #91. (d) **SVM** score. **SVT** tracking.

overdetermined system of equations and solve it to obtain motion parameters that are consistent with both constraints.

Furthermore, the relative weights of the two constraints can vary over time as the tracking proceeds. For example, when a vehicle is first detected, we might rely on **SVT** in the first couple of frames and, over time, increase the weight of the optic-flow-based tracking, using the information gathered in the first couple of frames. If we were to use a different kernel with **SVT**, then the combination with optic-flow-based tracking was still possible, but not as elegant.

We have not experimented with this extension to the **SVT** framework and leave it for further research.

5 Experiments

The classification engine was trained on a set of approximately 10,000 images of vehicles and nonvehicles. Vehicles include cars, SUVs, and trucks in different colors and sizes. The images were digitized from a progressive scan video at a resolution of 320×240 pixels and at 30 frames per second. Typical vehicle size is about 50×50 pixels. The vehicles and nonvehicles were manually selected and reduced to the size of 20×20 pixels. Their mean intensity value was shifted to the value 0.5 (in the range $[0..1]$) to help reduce the effect of variations in vehicle color. We used a homogeneous quadratic polynomial kernel given by $k(\mathbf{x}, \mathbf{x_j}) = (\mathbf{x}^T \mathbf{x_j})^2$ to perform the learning phase. The classification rate was about 92 percent for the learning set, with about $2,000$ support vectors. A similar classification rate was obtained for the testing set that contained approximately $10,000$ images as well.

To speed up the classification phase, we used the Reduced Set Method [5] to reduce the number of support vectors from $2,000$ to 400. The Reduced Set Method shows that, for the homogeneous quadratic polynomial kernel, the number of support vectors does not have to exceed the dimensionality of the input space. The number of support vectors can be reduced, through Principal Component Analysis on the support vectors in feature space, to a number bounded by the dimensionality of the input space, which is 400 in our case. In practice, we found that the 50 support vectors with the largest eigenvalues are sufficient for classification. We created a two-level Gaussian pyramid for every one of the 50 support vectors by treating every support vector as a 20×20 image. Each test image was converted to a Gaussian pyramid with the bottom level of the pyramid being 20×20 pixels. We used **SVT** with a $2D$ translation model on the pyramid to refine the image position. The score of the

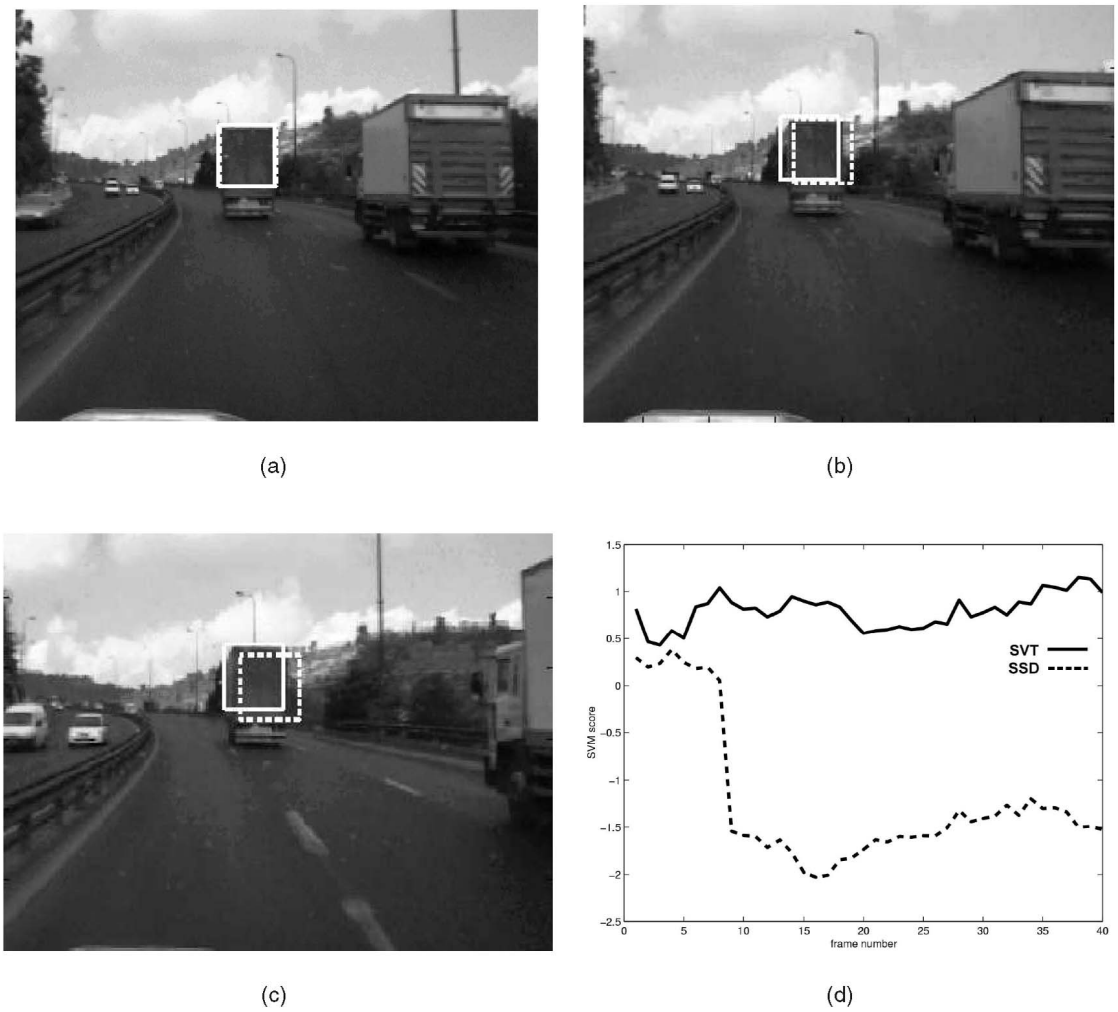

Fig. 6. (a), (b), and (c) are three frames from a 40 frame sequence. The solid rectangle denotes the **SVT** tracking result, the dashed rectangle denotes a simple **SSD** tracker. (d) shows the **SVM** score of the tracked region using the **SVT** tracker (solid line) and **SSD** tracker (dashed line). The x-axis of the graph is the frame number, the y-axis is the **SVM** score. In the **SSD** case, we tracked using the **SSD** tracker and then used the **SVM** classifier to give the tracked region a score. As can be seen, even though the **SSD** tracker managed to track the vehicle, its **SVM** score is negative and, so, the system will stop tracking this object. (a) Frame #1. (b) Frame #10. (c) Frame #37. (d) **SVM** score.

test image was taken to be the **SVM** score of the bottom level. **SVT** takes as input an initial position of the object in the first frame. It then applies pyramid **SVT** and outputs the position in the image with the highest **SVM** score. This position then serves as the initial guess for the next frame. The running time of **SVT** depends on the kernel used and the number of support vectors. In the case of a homogenous quadratic kernel, for example, **SVT** was slower than **SSD** tracking because, in each iteration, we had to perform dot-product between the test pattern and all the support vectors.

We conducted four experiments on a wide variety of vehicles of different type, size, color, and shape. No parameter was changed from experiment to experiment. In all cases, the initial guess in the first image was supplied manually. In the real system, the initial guess will be supplied by the detection module.

In the first experiment, we supplied the algorithm with a rough initial guess and tested how well it maximized the **SVM** score (Fig. 3). For each image in the figure, we show how much the **SVM** score improved and what the motion from the initial guess to the final position was. Note that, in some cases, a motion of only one pixel can change the **SVM** score from negative to positive (see Fig. 3c).

We also analyzed the error surface of **SVT** (Fig. 4). This was done by exhaustively computing the **SVM** score in the vicinity of the initial guess and computing the trajectory of the **SVT** algorithm against it. One can see that the error surface seems to be smooth, making it suitable for gradient-descent algorithms such as the **SVT**.

The average image-plane motion that was correctly recovered by **SVT**, across the sequences presented here (about 400 frames), was two pixels. The maximum was 10 pixels.

The second experiment tested how the $2D$ translation motion model we use handles an approaching vehicle. In this sequence of 101 frames, our car is approaching a slower moving white car (Fig. 5). Note also that the viewpoint changes (we can see the side of the car as we approach it) but **SVT** still manages to keep track of the rear of the car with high **SVM** scores.

In the third experiment (Fig. 6), we compare the **SVT** against a simple **SSD** tracker. In the **SSD** case, we use an **SSD**-based tracker and score the tracked region using **SVM** (not

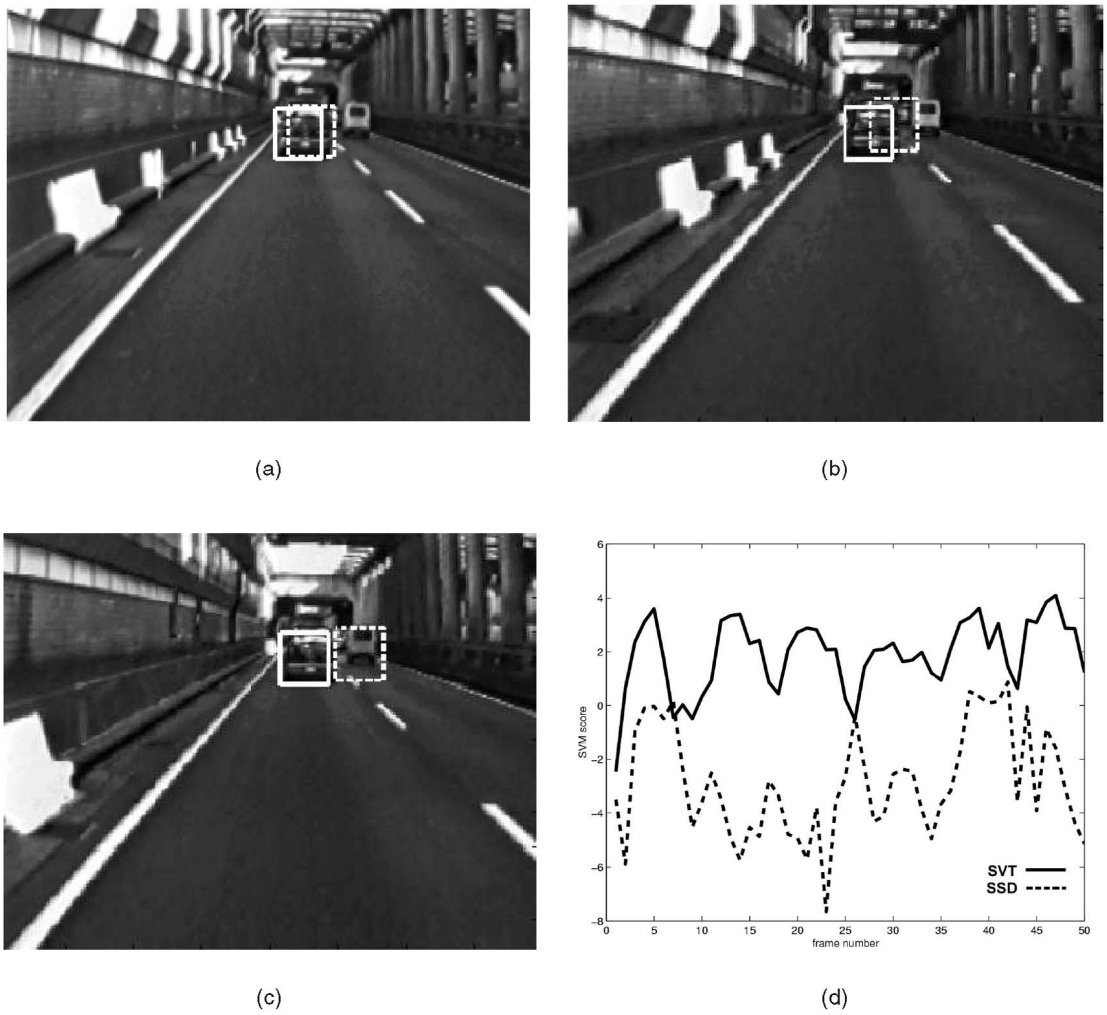

Fig. 7. (a), (b), and (c) are three frames from a 50 frames sequence. The solid rectangle denotes the **SVT** tracking result, the dashed rectangle denotes a simple **SSD** tracker. (d) shows the **SVM** score of the tracked region using the **SVT** tracker (solid line) and **SSD** tracker (dashed line). The x-axis of the graph is the frame number, the y-axis is the **SVM** score. In the **SSD** case, we tracked using the **SSD** tracker and then used the **SVM** classifier to give the tracked region a score. As can be seen, the **SSD** tracker is fooled by the changing illumination, while the **SVT** tacker is much more stable. (a) Frame #3. (b) Frame #25. (c) Frame #50. (d) **SVM** score.

SVT). This experiment shows that the error function minimized by the **SSD** tracker can result in disastrous **SVM** scores that might reduce our confidence that we are still tracking a vehicle.

The **SSD** tracker works by minimizing the **SSD** error between successive frames. It uses pyramids to account for large motions and it uses (7) to estimate the motion parameters. The sequence shows a truck changing lanes. As we can see, up to frame 10, both trackers perform comparably. However, at frame 10, probably due to some background pixels, the two trackers diverge by five pixels. This is enough to drastically lower the **SVM** score obtained by the **SSD** tracker. As can be seen, it never recovers from this miss and it keeps drifting away. The **SVT** tracker on the other hand keeps getting high **SVM** scores throughout the sequence.

The last experiment shows a challenging case of changing illumination (Fig. 7). The sequence is taken on a bridge covered with steel poles. The sun illuminates the vehicle in front of us through the poles, causing strong illumination artifacts. The **SSD** tracker fails to track the vehicle and drifts away after 50 frames without getting a positive **SVM** score along the way. The **SVT**, on the other hand, remains attached to the vehicle while maintaining a positive **SVM** score. It is interesting to see that the **SVM** is somewhat sensitive to the illumination changes as is evident from the jigsaw shape of the **SVM** score of the **SVT**.

6 Limitations and Future Research

We integrated the **SVM** classifier and the optic-flow-based tracker to give a Support Vector Tracking (**SVT**) mechanism. **SVT** works by maximizing the **SVM** classification score instead of minimizing an intensity difference function between successive frames. In addition, we created a Gaussian pyramid from every support vector, terming it "Support Vector Pyramid," that allows **SVT** to handle large motions in the image plane. We found that **SVT** gives good results for tracking vehicles over long periods of time. Moreover, the **SVM** score returned by **SVT** gives useful confidence measure that can help us determine if we are still tracking the vehicle or not.

However, **SVT** suffers from several limitations. First, the method cannot handle partial occlusions. Second, it is not designed to handle momentary disappearance and

reappearance. Third, there is no guarantee that **SVT** will not switch from one vehicle to another in case of two nearby vehicles.

The **SVT** paradigm can work with various motion models up to a 2D affine transformation and various kernels such as polynomial or RBF. Here, we used **SVT** with a homogenous quadratic polynomial kernel so that the motion equations will be linear. However, other kernels can be used as well at the cost of having to solve a nonlinear set of equations for the motion parameters. While this might slow down the computations for real-time tracking, it offers a principled manner to align a test image with the **SVM** support vectors; thus **SVT** can be viewed as extending **SVM** to have some inherent invariance to image transformations.

Finally, **SVT**, in its current form, does not integrate temporal information to improve tracking/detection performance. However, initial results suggest that it can be combined with optic-flow-based tracking to balance between different constraints. Moreover, temporal information can be used to improve the detection/tracking process as additional information about the tracked object is gathered.

REFERENCES

[1] P. Anandan, J. Bergen, K. Hanna, and R. Hingorani, "Hierarchical Model-Based Motion Estimation," *Motion Analysis and Image Sequence Processing,* Sezan and R. Lagendijk, eds., chapter 1, Kluwer Academic, 1993.

[2] M.J. Black and A.D. Jepson, "EigenTracking: Robust Matching and Tracking of Articulated Objects Using a View-Based Representation," *Proc. European Conf. Computer Vision,* B. Buxton and R. Cipolla, eds., Apr. 1996.

[3] L. Bottou, C. Cortes, J.S. Denker, and H. Drucker, I. Guyon, and L.D. Jackel, Y. LeCun, U.A. Muller, E. Sackinger, P. Simard, and V. Vapnik, "Comparison of Classifier Methods: A Case Study in Handwritten Digit Recognition," *Proc. 12th Int'l Conf. Pattern Recognition and Neural Networks,* pp. 77-87, 1994.

[4] C.J.C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," *Data Mining and Knowledge Discovery,* vol. 2, no. 2, pp. 121-167, 1998.

[5] C.J.C. Burges, "Simplified Support Vector Decision Rules," *Proc. 13th Int'l Conf. Machine Learning,* L. Saitta, ed., pp. 71-77, 1996.

[6] C. Cortes and V. Vapnik, "Support Vector Networks," *Machine Learning,* vol. 20, pp. 273-297, 1995.

[7] S. Gong, A. Psarrou, I. Katsoulis, and P. Palavouzis, "Tracking and Recognition of Face Sequences," *Proc. European Workshop Combined Real and Synthetic Image Processing for Broadcast and Video Production,* year?

[8] T. Joachims, "Transductive Inference for Text Classification Using Support Vector Machines," *Proc. Int'l Conf. Machine Learning,* 1999.

[9] S.Z. Li, L. Zhu, Z.Q. Zhang, A. Blake, H. J. Zhang, and H. Shum, "Statistical Learning of Multi-View Face Detection," *Proc. Seventh European Conf. Computer Vision,* May 2002.

[10] K. Morik, P. Brockhausen, and T. Joachims, "Combining Statistical Learning with a Knowledge-Based Approach—A Case Study in Intensive Care Monitoring," *Proc. 16th Int'l Conf. Machine Learning,* 1999.

[11] N. Vasconcelos and A. Lippman, "Multiresolution Tangent Distance for Affine-Invariant Classification," *Advances in Neural Information Processing Systems,* vol. 10, pp. 843-849, 1998.

[12] E. Osuna, R. Freund, and F. Girosi, "Training Support Vector Machines: An Application to Face Detection," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* pp. 130-136, 1997.

[13] C. Papageorgiou and T. Poggio, "A Trainable System for Object Detection," *Int'l J. Computer Vision,* vol. 38, no. 1, pp. 15-33, 2000.

[14] B. Heisele, T. Serre, M. Pontil, T. Vetter, and T. Poggio, "Categorization by Learning and Combining Object Parts," *Advances in Neural Information Processing System,* vol. 14, 2002.

[15] S. Romdhani, P. Torr, B. Scholkopf, and A. Blake, "Computationally Efficient Face Detection," *Proc. Int'l Conf. Computer Vision,* 2001.

[16] B. Scholkopf, P.Y. Simard, A.J. Smola, and V.N. Vapnik, "Prior Knowledge in Support Vector Kernels," *Advances in Neural Pnformation Processing Systems,* M.I. Jordan, M.J. Kearns, and S.A. Solla, eds., vol. 10, pp. 640-646, Cambridge, Mass.: MIT Press, 1998.

[17] B. Scholkopf, *Support Vector Learning.* R. Oldenbourg Munich: Verlag, 1997.

[18] S.M. Smith and J.M. Brady, "ASSET-2: Real-Time Motion Segmentation and Shape Tracking," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 17, no. 8, pp. 814-820, Aug. 1995.

[19] P.Y. Simard, Y. LeCun, and J. Denker, "Efficient Pattern Recognition Using a New Transformation Distance," *Advances in Neural Information Processing System,* pp. 50-58, San Mateo, Calif: Morgan Kaufman, 1993.

[20] K.-K. Sung and T. Poggio, "Example-Based Learning for View-Based Human Face Detection," *Proc. Image Understanding Workshop,* Nov. 1994.

[21] V. Vapnik, *The Nature of Statistical Learning Theory.* New York: Springer, 1995.

[22] P. Viola and M. Jones, "Rapid Object Detection Using a Boosted Cascade of Simple Features," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2001.

Shai Avidan received the BSc degree in mathematics and computer science from Bar-Ilan University, Ramat-Gan, Israel, in 1993, and the PhD degree from the Hebrew University, Jerusalem, Israel, in 1999. He was a postdoctoral fellow in the Vision Group at Microsoft Research, Redmond, Washington. He joined MobilEye in 2000 and, in addition, became a faculty member at the Interdisciplinary Center, Herzeliya, Israel. His research interests are in computer vision, machine learning, and computer graphics. His previous work included work on multiview geometry and image-based rendering.