Support Vector Tracking

Shai Avidan

MobilEye Vision Technologies,
24 Mishol Hadkalim,
Jerusalem, Israel,

E-mail: avidan@mobileye.com

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 a homogeneous quadratic polynomial kernel-***SVT** *for vehicle tracking in image sequences.*

1 Introduction

Tracking algorithms find how does an image region move 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. This error function is the result of making the "constant brightness assumption", i.e. pixel values does not change from frame to frame. This paradigm makes no assumptions about the nature 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 did the object move to and the classifier will give it a score. This scheme will repeat itself until the classification score falls 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.

Black and Jepson [6] integrated eigen-images into an optic-flow 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, eigen-images 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 [12], face detection [11], text classification [13] and medical applications [14] to name a few.

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 Figure 1). Vehicles cannot be well spanned by eigen-images and therefor the detection module of the application (which is not described in this paper) relies on a kernel-**SVM** that was trained on thousands of images of vehicles and non-vehicles. Once detected, the system tracks the vehicle over time. Naturally, we would like to leverage the power of the classifier for tracking.

Optic-flow [5] is the particular tracker we will use here. It assumes that pixel values do not change from frame to frame ("constant brightness constraint"), and that transformations such as a 2D affine transformation are sufficient for tracking purposes. To account for large motion between successive frames it uses a coarse-to-fine framework in which a rough alignment is achieved in the coarser levels of the pyramid and a refined solution is obtained in the fine levels.

For classification we use the **SVM** technique [1]. **SVM** is a general classification scheme that has been successfully used in the past [10, 11, 12]. Given a set of positive and negative examples, **SVM** finds the best separating hyperplane between

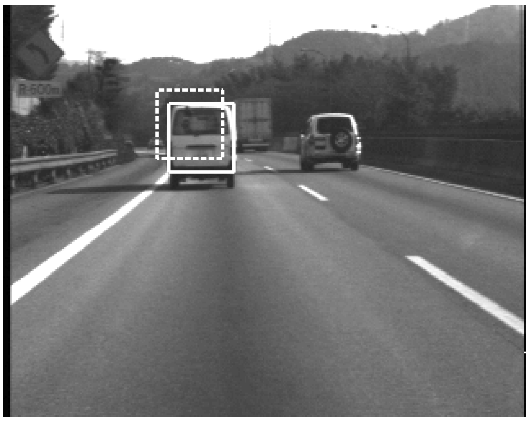


Figure 1. Illustration of SVT operation. SVT takes as input the initial guess of the position of the vehicle (dashed rectangle) and finds the position with the highest SVM score (solid rectangle).

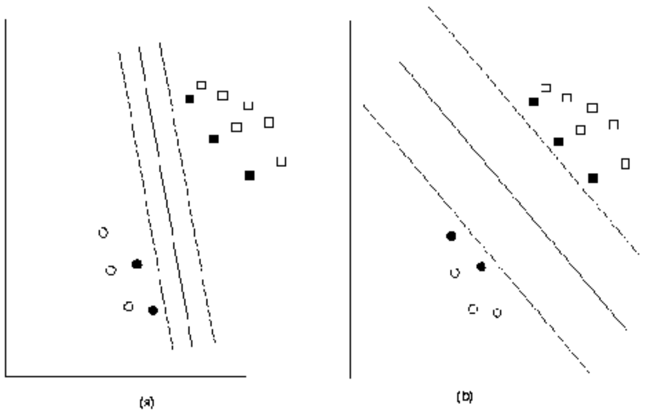


Figure 2. (a) A separating hyperplane with small margin. (b) A separating hyperplane with a large margin. A better generalization capability is expected from (b). The filled squares and circles are termed "support vectors".

the two classes. The *support vectors* are the examples closest to the separating hyperplane. Test images are classified based on their signed distance to the separating hyperplane.

In machine learning the problem tackled by the **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.* [7] 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 non-linear 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, Nuno and Lippman [8] worked on face images (that are much larger than character images) by extending "tangent distance" to work in a multi-resolution framework. Finally, Scholkopf *et al.* [9] 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 multi-resolution treatment can be incorporated into their scheme.

Our approach is similar to that of [6] 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 multi-resolution framework.

2 Support Vector Machine

For the paper to be self contained we give a brief description of **SVM**. The interested reader is referred to [1, 3] for a more detailed description. Consider the simple case of two linearly separable classes. Given a data set $\{\mathbf{x_i}, y_i\}_{i=1}^{l}$ of l examples $\mathbf{x_i}$ with labels $y_i \in \{-1, +1\}$, we wish to find a separating hyperplane between the two classes. Formally, we consider the family of decision functions

$$f(\mathbf{x}) = sgn(\mathbf{w}^T\mathbf{x} + b) \qquad (1)$$

and wish to find $\mathbf{w}, b$ such that $sgn(\mathbf{w}^T\mathbf{x_i} + b) = sgn(y_i)$. This problem is in general ill-posed because there might be an infinite number of separating hyperplanes. The question is which one has a low generalization error (i.e. which one will do a good job in classifying new examples). It was shown by Vapnik [1] that choosing the hyperplane with the minimal norm of $\mathbf{w}$ minimizes the "Structural Risk" which is an upper bound on the generalization error. Intuitively, this $\mathbf{w}$ is the one to maximize the margin between the two classes (See Figure 2). Practically, this amounts to solving the following quadratic optimization problem (QP)

$$\begin{array}{l} \min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} \\ subject\ \ to\ \ y_i(\mathbf{w}^T\mathbf{w} + b) \geq 1, \\ \qquad\quad i = 1...l \end{array} \qquad (2)$$

that can be solved quite efficiently. The example vectors closest to the separating hyperplane are called "support vectors". The classification itself is performed by measuring the signed distance of the test image from the separating hyperplane.

But how can the **SVM** be extended to handle decision functions that are not linear in the data? The answer is to use a nonlinear mapping Φ of the input data and map it to some high-dimensional (possibly even with infinite dimensions) *feature space* F. The linear **SVM** is then performed in F and will therefor be nonlinear in the original input data. Formally, let

$$\Phi : \mathcal{R}^n \rightarrow F \qquad (3)$$

be a nonlinear mapping from input space to feature space and the decision functions we deal with becomes

$$f(\mathbf{x}) = sgn(\sum_{j=1}^{l} y_j\alpha_j\Phi(\mathbf{x})^T\Phi(\mathbf{x_j}) + b). \qquad (4)$$

where α_j is a set of parameters computed by solving the QP problem. However, working in feature space can be prohibitively expansive to compute. Therefor we use Mercer kernels on the input data to avoid computing the dot products in feature space. Mercer kernels $k(\mathbf{x}, \mathbf{x_j})$ satisfy that $k(\mathbf{x}, \mathbf{x_j}) = \Phi(\mathbf{x})^T \Phi(x_j)$. Thus, in kernel-**SVM** we use the following decision functions

$$\begin{aligned} f(\mathbf{x}) &= sgn(\textstyle\sum_{j=1}^{l} y_j \alpha_j \Phi(\mathbf{x})^T \Phi(\mathbf{x_j}) + b) \\ &= sgn(\textstyle\sum_{j=1}^{l} y_j \alpha_j k(\mathbf{x}, \mathbf{x_j}) + b) \end{aligned} \tag{5}$$

and the quadratic programming problem becomes:

$$\begin{aligned} &maximize \\ &\quad W(\alpha) = \textstyle\sum_{i=1}^{l} \alpha_j - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j k(\mathbf{x_i}, \mathbf{x_j}) \\ &subject \;\; to \\ &\qquad\qquad \alpha_i \geq 0, \quad i = 1...l, \\ &\qquad and \quad \textstyle\sum_{i=1}^{l} \alpha_i y_i = 0. \end{aligned} \tag{6}$$

It turns out that α_j is equal to 1 for examples on the border between the two classes and zero otherwise. In typical applications about 10% of the examples have α_j equal to 1 and these examples are called *support vectors*. The rest of the examples are not important because they do not help separate between the two classes. The only difference between kernel and linear **SVM** is that the dot product of linear **SVM** is replaced with a kernel function.

Typical kernels used in the **SVM** literature include $k(\mathbf{x}, \mathbf{x_j}) = exp(- \ \mathbf{x} - \mathbf{x_j}||)^2$ which leads to a Gaussian RBF, $k(\mathbf{x}, \mathbf{x_j}) = (\mathbf{x}^T \mathbf{x_j} + 1)^d$ which represent polynomial of degree d and $k(\mathbf{x}, \mathbf{x_j}) = tanh(\mathbf{x}^T \mathbf{x_j} - \Theta)$ which leads to multi-layer perceptron. Extension to non-separable classification problem exist [2], where the idea is that a penalty term is used to govern the price we are willing to pay for misclassified examples.

3 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. This is usually done by exhaustively searching the image for candidates and classifying them as objects or non objects. 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 it is tracked over time. Since much effort was already put into building the detection module it is only natural to try and 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 and so on. In this work we will concentrate on the tracking module alone and will not describe the detection module.

We introduce the notations and 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.

As for notations, we denote vectors using bold face fonts $\mathbf{x}$ to distinguish them from scalars x. When using the **SVT** equations we assume the image data to be vectorized.

3.1 Support Vector Tracking

Kernel-**SVM** is given by

$$\sum_{j=1}^{l} y_j \alpha_j k(\mathbf{I}, \mathbf{x}_j) + b \tag{7}$$

where $\mathbf{x}_j$ are the l support vectors, y_j are their sign and α_j are their distance from the hyperplane. $k(\mathbf{I}, \mathbf{x}_j)$ is the kernel we choose to use, and I is the image region we wish to test.

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 $\mathbf{I_{final}}$ (Figure 1). Using first-order Taylor expansion we have that

$$\mathbf{I_{final}} = \mathbf{I_{init}} + u\mathbf{I_x} + v\mathbf{I_y} \tag{8}$$

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

$$\sum_{j=1}^{l} y_j \alpha_j k(\mathbf{I_{final}}, \mathbf{x}_j) = max\{\mathbf{I} | \sum_{j=1}^{l} y_j \alpha_j k(\mathbf{I}, \mathbf{x}_j)\} \tag{9}$$

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).

Plugging equation 8 in equation 7 we get

$$\sum_{j=1}^{l} y_j \alpha_j k(\mathbf{I} + u\mathbf{I_x} + v\mathbf{I_y}, \mathbf{x}_j) \tag{10}$$

which we want to maximize. 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 non-linear) equations to solve. This equations will depend on the particular kernel we use.

3.1.1 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

$$\begin{aligned} E(u,v) &= \textstyle\sum_{j=1}^{l} y_j\alpha_j k(\mathbf{I}+u\mathbf{I_x}+v\mathbf{I_y}, \mathbf{x}_j) \\ &= \textstyle\sum_{j=1}^{l} y_j\alpha_j((\mathbf{I}+u\mathbf{I_x}+v\mathbf{I_y})^T\mathbf{x}_j)^2 \end{aligned} \tag{11}$$

and taking the u and v derivatives gives:

$$\frac{\partial E}{\partial u} = \sum_{j=1}^{l} y_j\alpha_j \mathbf{I_x}^T\mathbf{x}_j(\mathbf{I}+u\mathbf{I_x}+v\mathbf{I_y})^T\mathbf{x}_j \tag{12}$$
$$= 0$$
$$\frac{\partial E}{\partial v} = \sum_{j=1}^{l} y_j\alpha_j \mathbf{I_y}^T\mathbf{x}_j(\mathbf{I}+u\mathbf{I_x}+v\mathbf{I_y})^T\mathbf{x}_j \tag{13}$$
$$= 0$$

and after rearranging terms we arrive at the following equation:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \tag{14}$$

Where

$$\begin{aligned} A_{11} &= \sum_{j=1}^{l} \alpha_j y_j (\mathbf{x_j}^T\mathbf{I_x})^2 \\ A_{12} &= A_{21} = \sum_{j=1}^{l} \alpha_j y_j (\mathbf{x_j}^T\mathbf{I_x})(\mathbf{x_j}^T\mathbf{I_y}) \\ A_{22} &= \sum_{j=1}^{l} \alpha_j y_j (\mathbf{x_j}^T\mathbf{I_y})^2 \\ b_1 &= -\sum_{j=1}^{l} \alpha_j y_j (\mathbf{x_j}^T\mathbf{I_x})(\mathbf{x_j}^T\mathbf{I}) \\ b_2 &= -\sum_{j=1}^{l} \alpha_j y_j (\mathbf{x_j}^T\mathbf{I_y})(\mathbf{x_j}^T\mathbf{I}) \end{aligned}$$

This equations resembles the standard optic-flow equations with the support vectors replacing the role of the second image in the equation. This means that all computations are done on a single frame each time and not on a pair of successive frames.

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 equation 14, 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 the next subsection.

3.2 Pyramid SVT

Each test image that needs to be classified is first sub-sampled 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 sub-sampled version of the test image. The misalignments must be small so that the first-order Taylor approximation, used by **SVT**, will suffice to lock 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.

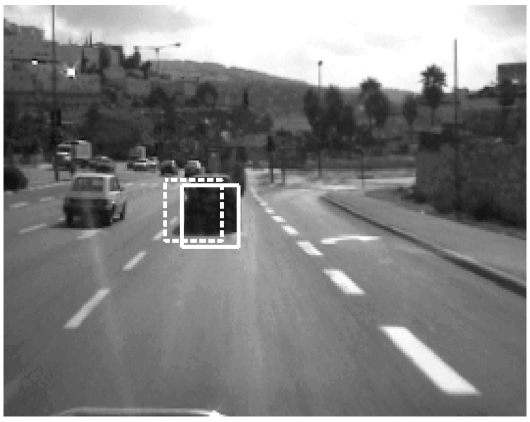
So far we have treated the support vectors as vectors, not as images. But recall that the support vectors are a sub-sampled image of the most problematic images in the learning set and therefor we can smooth and subsample them to create a support vector pyramid for each support vector. In the classification 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 algorithm that uses Newton's iterations. The experiments reveal how robust is **SVT** to transformations in the image space.

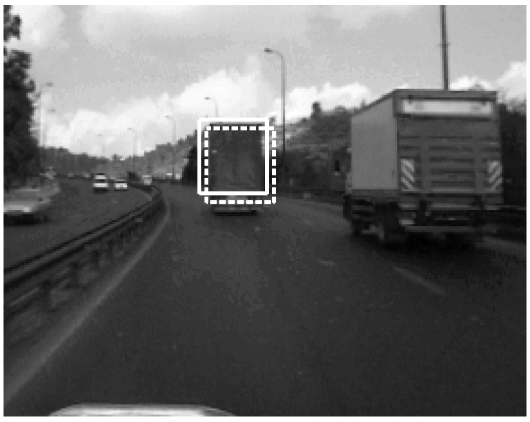
4 Experiments

The classification engine was trained on a set of approximately 10000 images of vehicles and non-vehicles. 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 $\times$ 240 pixels and at 30 frames per second. Typical vehicle size is about 50 $\times$ 50 pixels. The vehicles and non-vehicles were manually selected and reduced to the size of 20 $\times$ 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 an 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% for the learning set, with about 2000 support vectors. A similar classification rate was obtained for the testing set that contained approximately 10000 images as well.

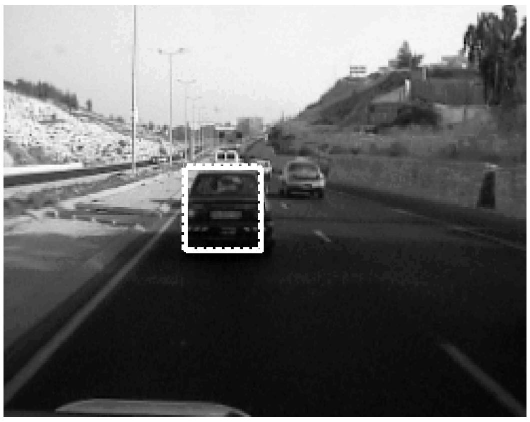
To speed up the classification phase we used the Reduced Set Method [4] to reduce the number of support vectors from



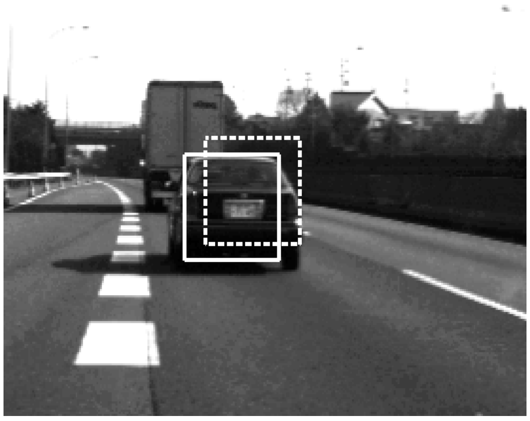
(a) Init: -8.3 Final: 2.9 Motion: (8.8,2.8)



(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: (4.4,-8.6)

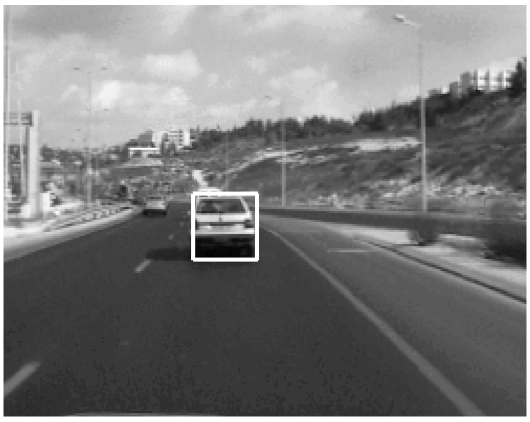
Figure 3. Examples of the initial guess (dashed line) and the final position (solid line). The image size in all cases is 320×240. Typical object size is 40×40 pixels. The SVM score of the initial guess and the final position as well as the amount of motion between the initial and final position are shown for every example. In example (c) a small change in the position (about one pixel) changed the SVM score from negative to positive.

2000 to 400. The Reduced Set Method shows that for 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. For each of the 50 support vectors we created a 2 level Gaussian pyramid by performing a 2D smoothing followed by sub-sampling. Each test image was sub-sampled 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 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.

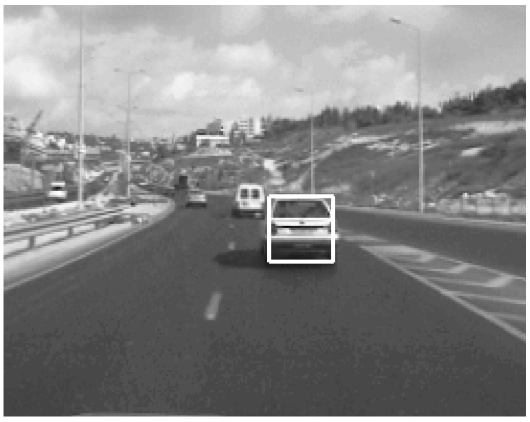
We conducted four tests on a wide variety of vehicles of different type, size, color and shape. No parameter was changed from test to test. In all cases the initial guess in the first image was supplied manually. In the real system the initial guess will be supplied by the detection module.

In the first test we supplied the algorithm with a rough initial guess and tested how well it maximized the **SVM** score (Figure 3). For each image in the figure we show how much did the **SVM** score improved and what was the motion from the initial guess to the final position. We found that translations up to about 10% of the vehicle size were handled by the algorithm. More importantly, in some cases a motion of about one pixel changed the **SVM** score from negative to positive (see image (c) in Figure 3).

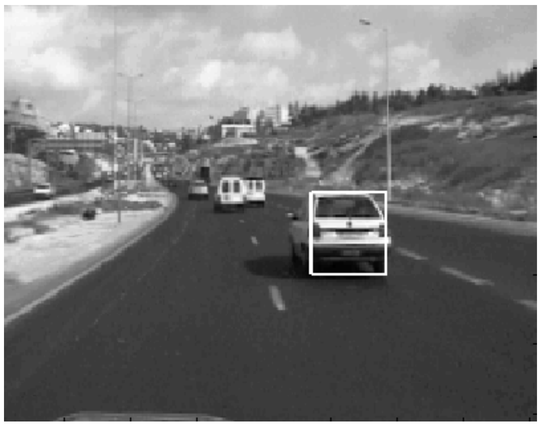
The second case test how the $2D$ translation motion model we use handles an approaching vehicle. In this sequence of 101 frames our car is approaching a slower moving white car (Figure 4). Note also that the view point changes (we can see the side of the car as we approach it) but still **SVT** manages to keep track of the rear of the car with high **SVM** scores.



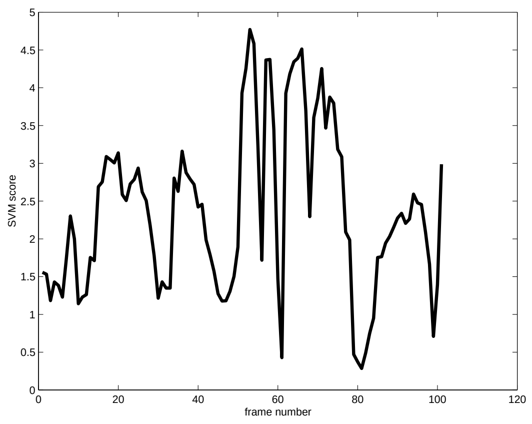
(a) Frame #1



(b) Frame #51



(c) Frame #91



(d) SVM score

Figure 4. SVT tracking. (a),(b),(c) are three frames from a 101 frames sequence. (d) show 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.

In the third test (Figure 5) we compare the **SVT** against a simple **SSD** tracker. The **SSD** tracker works by minimizing the **SSD** error between successive frames. It uses pyramids to account for large motions and it uses the following optic flow equation to estimate the motion parameters:

$$\begin{pmatrix} \sum \mathbf{I_x}^T\mathbf{I_x} & \sum \mathbf{I_x}^T\mathbf{I_y} \\ \sum \mathbf{I_x}^T\mathbf{I_y} & \sum \mathbf{I_y}^T\mathbf{I_y} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} -\sum \mathbf{I_t}^T\mathbf{I_x} \\ -\sum \mathbf{I_t}^T\mathbf{I_y} \end{pmatrix} \tag{15}$$

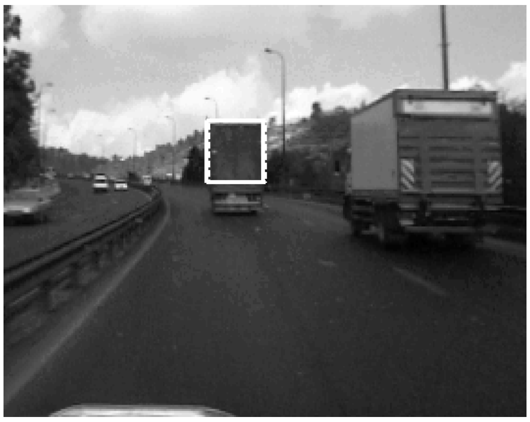
where $\mathbf{I_x}, \mathbf{I_y}$ are the x, y derivatives of the first image and $\mathbf{I_t}$ is the time derivative between the two frames. The sequence shows a truck changing lanes. As we can see, up to frame 10 both trackers perform comparably. However at frame 10, probably due to some background 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.

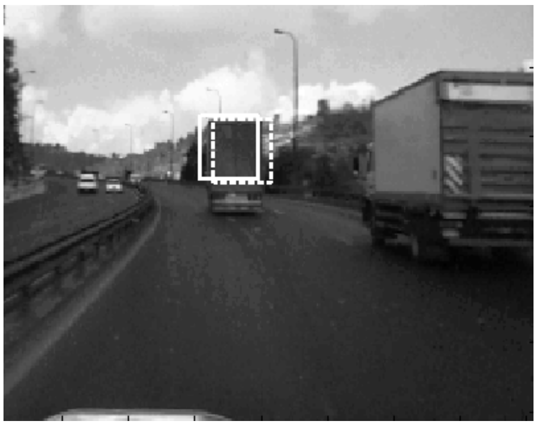
The last test shows a challenging case of changing illumination (Figure 6). The sequence is taken on a bridge covered with steel poles. The sun illuminates the vehicle in front of us through the poles, causing strong illumination artifacts. The **SSD** tracker fails to track the vehicle and drifts away after 50 frames without getting 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**.

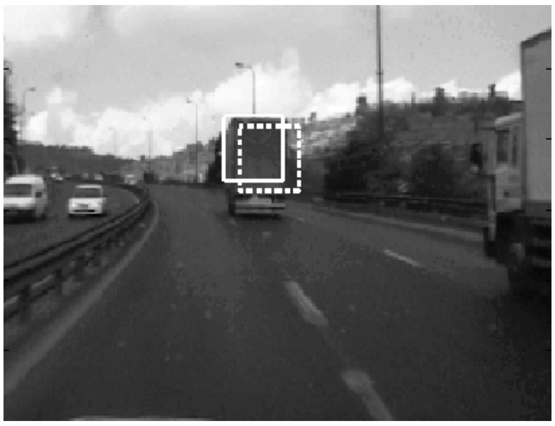
5 Conclusions

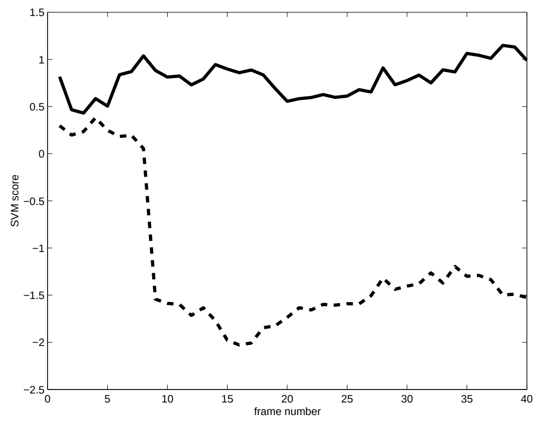
We integrated the **SVM** classifier and the optic-flow tracker to give a Support Vector Tracking (**SVT**) mechanism. **SVT** works by maximizing the **SVM** classification score, instead of minimizing an intensity difference function between successive frames. In addition we created a Gaussian pyramid from every support vector, terming it “Support Vector Pyramid”, that allows **SVT** to handle large motions in the image plane. The algorithm was tested on real video sequences for the purpose of vehicle tracking.

The **SVT** paradigm can work with various motion models


(a) Frame #1


(b) Frame #10


(c) Frame #37



(d) SVM score

Figure 5. (a),(b),(c) are three frames from a 40 frames sequence. The solid rectangle denotes the SVT tracking result, the dashed rectangle denotes a simple SSD tracker. (d) show the SVM score of the tracked region using the SVT tracker (solid line) and SSD tracker (dashed line). As can be seen, SVT is far superior to the SSD tracker. The x-axis of the graph is the frame number, the y-axis is the SVM score.

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 equations will be linear. However, other kernels can be used as well at the cost of having to solve a non-linear 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.

References

[1] V. Vapnik. *The Nature of Statistical Learning Theory.* Springer, N.Y., 1995.

[2] C. Cortes and V. Vapnik. Support Vector Networks. *Machine Learning,* 20:273-297, 1995.

[3] C.J.C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery,* 2(2):121-167, 1998.

[4] C.J.C. Burges. Simplified support vector decision rules. In L. Saitta, editor, *Proceedings, 13th Intl. Conf. on Machine Learning,* pages 71-77, San Mateo, Ca, 1996.

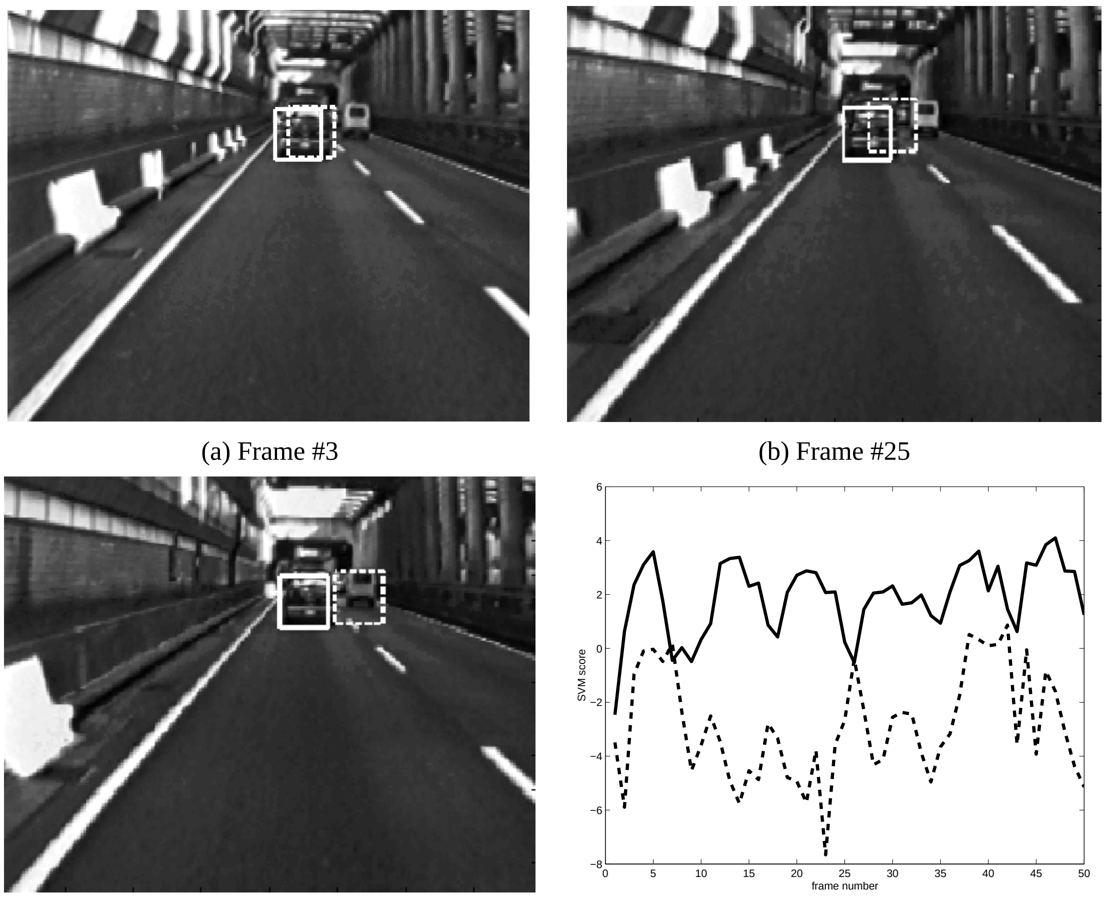
[5] P. Anandan, J. Bergen, K. Hanna and R. Hingorani. Hierarchical Model-Based Motion Estimation. In M. Sezan and R. Lagendijk, editors, *Motion Analysis and Image Sequence Processing,* chapter 1, Kluwer Academic Press, 1993.

[6] Michael J. Black and Allan D. Jepson. EigenTracking: Robust Matching and Tracking of Articulated Objects Using a View-Based Representation. In Bernard Buxton and Roberto Cipolla, editors, *European Conference on Computer Vision,* Cambridge, UK, April 1996.

[7] Simard, P. Y., LeCun, Y. and Denker, J. Efficient pattern recognition using a new transformation distance. In *Advances in Neural Information Processing System,* Morgan Kaufman, San Mate, CA, pp. 50-58, 1993.

[8] Nuno Vasconcelos and Andrew Lippman. Multiresolution tangent distance for affine-invariant classification. In *Advances in Neural Information Processing Systems,* volume 10, pages 843-849, Morgan Kaufman Publishers, 1998.

[9] B. Scholkopf, P. Y. Simard, A. J. Smola, and V. N. Vapnik.



(a) Frame #3 (b) Frame #25

(c) Frame #50 (d) SVM score

Figure 6. (a),(b),(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) show the SVM score of the tracked region using the SVT tracker (solid line) and SSD tracker (dashed line). As can be seen, the SSD tracker is fooled by the changing illumination while the SVT tacker is much more stable. The x-axis of the graph is the frame number, the y-axis is the SVM score.

Prior knowledge in support vector kernels. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural information processings Systems*, volume 10, pages 640-646, Cambridge, MA, 1998. MIT Press.

[10] L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, 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. In *Proceedings of the 12th International Conference on Pattern Recognition and Neural Networks*, Jerusalem, pages 77-87, IEEE Computer Society Press, 1994.

[11] E. Osuna, R. Freund and F. Girosi. Training Support Vector Machines: An Application to Face Detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, Puerto Rico, pages 130-136, 1997.

[12] B. Scholkopf. Support Vector Learning. R. Oldenbourg Verlag, Munich, 1997.

[13] Thorsten Joachims. Transductive Inference for Text Classification using Support Vector Machines. In *International Conference on Machine Learning*, (ICML), 1999.

[14] K. Morik, P. Brockhausen, and T. Joachims. Combining statistical learning with a knowledge-based approach - A case study in intensive care monitoring. In *Proc. 16th Int'l Conf. on Machine Learning*, (ICML-99), 1999.