EE392j Project Report

Segmentation of Vehicles in Traffic Video

Tun-Yu Chiang, Wilson Lau
Stanford University
Email: {tchiang, wslau}@stanford.edu

Abstract— **Segmentation of moving objects in a scene is often desired in applications such as video surveillance, where our interest is in monitoring for example, cars and people. In the project, we look at traffic videos and investigate two approaches that can be used to segment vehicles from the background. From the results, we explored the pros and cons of each method.**

Our work is separated as the following: Tun-Yu Chiang worked on the background mixture model segmentation, and Wilson Lau worked on the motion based segmentation.

I. Introduction

Closed-Circuit Television Cameras are becoming increasingly common on freeways. Used for traffic management, the cameras allow operators to monitor traffic conditions visually. The large number of cameras make it impractical for each to be monitored at all times by an operator, and as such the videos are usually only monitored after an event of interest (e.g. an accident) has been known to occur within a particular camera's field of view.

With suitable processing and analysis it is possible to extract a lot of useful information on traffic from the videos, e.g. the number, type, and speed of vehicles using the road. In order to do this, one might try to segment the video into foreground objects of interest (the vehicles) and the background (road, trees). Segmenting the video into foreground and background can also reduce the data rate required to transmit live video as it may not be necessary to transmit the background as frequently as the foreground vehicles.

In the project, we investigate methods that can be used to extract moving vehicles from the background in such videos.

II. Method

We used a Sony Digital 8 DCR-TRV320 camcorder to shoot our own test sequences of traffic scenes. To simplify the task of motion segmentation, the shot angle was carefully chosen to minimize perspective distortions. The video sequences were downloaded to a computer using an IEEE-1394 cable. From the raw footage clips of interest (approximately 5 s or 150 frames @30 fps) were edited out. The location shown in the video is around the intersection of Page Mill Road and I-280, Palo Alto, CA.

We implemented two algorithms based on two completely different approaches to the problem. The first attempts to adaptively model each pixel as a statistical process and identify foreground objects by deviations from the usual parameters. The other tries to segment out the vehicles in the scene by grouping together regions undergoing a coherent motion.

We will describe each algorithm in detail and show the corresponding results for each in this report.

III. Adaptive background Mixture Model Based Segmentation

In the possible applications of vehicle extraction, such as traffic surveillance, real-time processing is often desired due to the amount of accumulating video data. From the literatures reviewed, the adaptive background mixture model matching method is a robust and also computationally inexpensive approach.

In this part of our experiment we implemented the background mixture model algorithms proposed by Stauffer [7] and Harville [5]. Our goal is to apply the mixture model to the simple traffic sequences with translational motion, and hope we can learn how different parameters affect the segmentation results qualitatively. One particular problem that arose from the experiment is then global motion produced by the hand-held camera. The global motion causes several obviously visible artifacts in the segmentation results. In the previous works this problem does not exist simply because in the surveillance context the video source is usually a fixed camera. To estimate and compensate the global motion, we tried a simple phase correlation motion estimation, which helped to eliminated some of the artifacts. We will discuss the experiments and results in the second part of this section.

A. The Method

This algorithm models a specific pixel value as a mixture of weighted K 3-D Gaussian distributions in the color space:

$$\textbf{pixel value} \quad \mathbf{X_t} = \begin{bmatrix} X_{r,t} \\ X_{g,t} \\ X_{b,t} \end{bmatrix}$$

$$\textbf{probability} \quad P(\mathbf{X_t}) = \sum_{i=1}^{K} w_{i,t} * \eta_{i,t}(\mathbf{X_t}, \mu_{\mathbf{i,t}}, \Sigma_{i,t})$$

The method is equivalently a on-line K-means clustering of the new observations. The adaptive method guarantees the background model is built with the recent history of observations. In each time step t, we try to match the new pixel observation $X_{i,t}$ with the K distributions in the mixture model. If we find a match, the weightings w_k and also the mean $\mu_{\mathbf{k}}$ and variance σ_k^2 of the match distribution is updated accordingly. If no match was found, we replace the the least-weighted distribution in the mixture with the new observation. In this phase it is important to specify the matching criterion and also the adaptive learning rate α in the process. We will discuss them in detail later.

For the background segmentation, after adapting the model with the observation, we sort the distributions according to the values w_k/σ_k. The w/σ value increases as the weight increases (more observation is matched to this distribution) and as the variance decreases(the distribution is more stable). The value is thus a good index to distinguish the background values from the K distributions. Also, we will assume that during the whole process, the pixel represents the background for T portion of the time. We pick the first B distributions from the sorting according to the following criterion:

$$B = \arg\min_b \left(\sum_{k=1}^{b} w_k > T \right)$$

For the foreground extraction, we classify the new observation as foreground if either (i) no match is found or (ii) the matched distribution does not belong to the B background distributions. In these cases, after replace the least possible distribution with the new observation, we take weighted average of the remaining $T-B$ distributions as the foreground value.

B. Implementation

To simplify the model, Both Stauffer [7] and Harville [5] assume that each of the K 3-D Gaussian distributions at time step t is of the form:

$$\eta_{k,t} = N(\mu_{\mathbf{k,t}}, \Sigma_{k,t})$$

$$\mu_{\mathbf{k,t}} = \begin{bmatrix} \mu_{r,t} \\ \mu_{g,t} \\ \mu_{b,t} \end{bmatrix}, \Sigma_{K,t} = \sigma^2 I$$

Note that the above assumption of the 1-D distributions being independent with same variance values is appropriate depending on the color space chosen. However in our experiment we only worked in the RGB space, further exploration of the difference caused by the color spaces is not included in this discussion.

1) The matching criterion: **The two key parts of the algorithm are the matching criterion and the adpative process. For the matching criterion, both [7] and [5] suggest that we compare the newly observed pixel value with the mean μ_k of the *kth* distribution. If the observation is within $\beta\sigma$ from the mean μ_k we declare a match found. There are two ways to compare the 'distance' from the observation to the mean vector with the $\beta\sigma$ bound the 3-D space: (i) compare the Gaussian probaility value [7] (note: the definition of β in the two formulas are different):**

A match is found with $\eta_k$when:

$$P(\mathbf{X_t}) > P(\mu_{\mathbf{k,t}} + \beta\sigma)$$

(ii)calculate the Euclidean distance [5]):

A match is found with $\iota_k$when:

$$(\mathbf{X_t} - \mu_{\mathbf{k,t}})^2 > \beta\sigma_k^2$$

At a preliminary stage of the experiment, we implemented both (i) and (ii) in MATLAB. We found that qualatatively the results does are not very different from each other, whereas there is an obvious efficiency gap [1]. Profiling results showed the bottleneck was the Gaussian probability calculation. The efficiency issue is important to the experment because this adaptive method takes several iterations (depending on the parameter chosen, usually 15-20 from our observation)to establish a stable background model. This is the major reason we chose to closely follow the formulae from [5] instead of from [7].

2) The adaptive process: **In each time step, we need to update the weights w_k's, the means μ_k's, and the variances σ_k's according to the matching result. The update formulae proposed by [7] are based on the learning rate α and also the the gaussian probability value. As mentioned before, the computations for Gaussian probability slows down the processing speed to an unacceptable level. So we used the updating formulae listed below, which is an simplified version of those in [5] [2].**

If the a match is found with the mth distribution, the mean and variance are updated with:

$$\mu_{\mathbf{m,t+1}} = (1-\alpha)\cdot\mu_{\mathbf{m,t}} + \alpha\cdot\mathbf{X_t}$$
$$\sigma_{m,t+1} = (1-\alpha)\sigma_{m,t} + \alpha(\mathbf{X_t} - \mu_{\mathbf{m,t}})^T(\mathbf{X_t} - \mu_{\mathbf{m,t}})$$

The weights of all η_k's are updated with:

$$w_k = (1-\alpha)w_k + \alpha M_k$$
$$M_k = \begin{cases} 1 & \textbf{match found}, \\ 0 & \textbf{otherwise}. \end{cases}$$

The weights are re-normalized after the update.

In addition, in order to improve the program efficiency we also incorporated a thresholding method. The idea of is simply that if a pixel value did not change over a comparatively long time period, we should consider it as static. So we compare the current frame with a previous reference frame 20 time steps ahead. We then threshold on the difference values. If the similarity of the pixels is above the threshold we would directly conclude the current pixel as part of the

[1] If use gaussian probability in the matching and updating functions, it took 380 to 1300sec. to process a frame. If simply $alpha$ is used, it took only about 38sec. on average

[2] They used a scene activity measure to modulate α in each step

background and skip the adpative process. This thresholding is applied before the Gaussian mixture process in each iteration. Since the scheme is more or less adhoc, we tried several different threshold values trying to find the most appropriate. The results show that the values does affects the quality of the output in the sense that the background model will be established more slowly when the threshold is low. This is because in that case, fewer gaussian mixture processing would be performed. Based on the trial-and-erorr's we fixed the threshold value of the minimum difference between the current and the reference frame to $minDiff = 20$**.**

C. Experiments and results

Throughout the experiment we fixed parameters $\beta = 2.5$ **as suggested in [7] and the threshold value min Diff** $= 20$ **as mentioned above. To further simplifiy matters we only chose The distribution with the highest weight** w/σ **as the background pixel value, instead of using the** T **criterion before mentioned. In the experiment the variables are parameters** α **and** K**.**

We tested the algorithms on three 5-seccond long video sequces with temporal sampling rate 15 frames/sec. The algorithm worked for all three sequecnes. Each of the segmented sequences shows some specific characteristics of the model, details are explained in *fig.1, fig.2 and fig3*. In general, the vehicles would disappear within 15-20 time steps (and equivalently in the foreground the vehicle would appear without significantly visible holes). In this section we will focus on some of the artifacts we observed in the segmentation results. We will discuss the possible causes of these artifacts, and also how to compensate them.

1) residue in the segmented background and the learning rate: **We observed residues of the vehicles the bakcground segmentations. The time for the residues to disappear is related to the learning rate** α**. In other words, the faster the model is adapted to the newly observed vehicle value, the more residue we would see in the background, because the interior pixel values of the vehicles are accepted as a higly weighted memeber of the K gaussian mixtures. The resutls from different** α **values are shown in *fig.4*.**

2) blurring of the background: **We observed that when there is apparent camera motion between two time steps, the extracted background will become blurred after in several following time steps. The reason of this artifact is that the background model was built before the camera motion took place, and so the new observations contains a spatially shifted version of the background. The particular Gaussain mixture algorithm applied here is literally producing an average from the background model and the new observation. Therefore in the presence of global motion between consecutive observations the ouput is a spatially averaged version of the original and shifted background.**

Theoretically, the blurring artifact could be eliminated with global motion compensation. However, as we can see later in the report that it is difficult to achieve the global motion compensation with enough subpixel precision. Even after motion compensation, the blurring effect is still visible in the regions with many details, such as the trees and the lawn in our example.

Fig. 4. Background of frame 38 from *Pagemill01*. Top: $\alpha = 0.05$, bottom: $\alpha = 0.95$.

The other easier way to improve the sharpness is simply to increase K**, the number of possible distributions in the mixture. This is by the assumption that the global motion from the hand-held camera is randowm and small in value. By increasing K we can include more of the possible neighborhood values in the distribution mixture, and thus build a better model.**

In the experiment sequence pagemill01.avi, there are some pronounced horizontal global motion around the 100**th frame. *fig.5* includes pictures of the extracted background with** $\alpha = 0.05$ **and** $K = 3, 4, 5$**. We can see the sharpness of the tree region increases with** K**.**

3) Vertical and horizontal global motion: **The global motion is readily observable from both the background and the foreground sequences. Two particularly good examples are the disppearing and reappearing of the electrial pole in the background, and the existence of the lane marks in the foreground. From the pagemill01 sequence we can see that the electrical pole would partially disappear and reappear. This is caused by the comparatively rigorous horizontal global motion between those frames.**

The other more apparent example are the horizontal-line shaped lane marks. We expect these marks to be part of the background. But it was nearly absent (only small parts appeared) in the background and always present in the foreground. This is because it is actually 'moving' if we don't compensate the globale motion. We can see the vertical motions from one of the "lane mark traces" over time in *fig.6*.

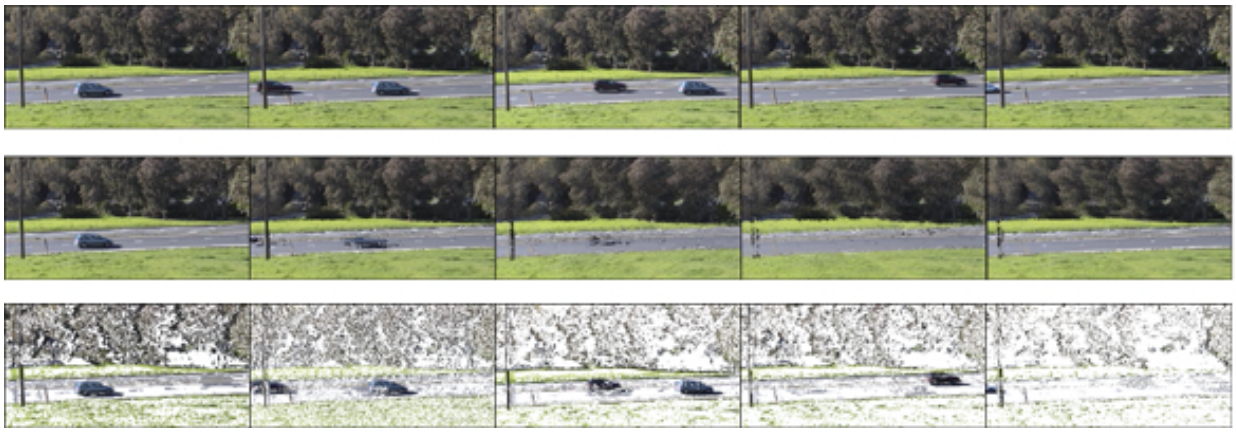


Fig. 1. Segmentation of pagemill01.avi. Left to right: frames 2, 26,52,78,104. Top to bottom: Original, Background, Foreground. Note that in the the last 3 frames of the background the electrical pole starts to disappear. Also, the trees are blurred in the background.

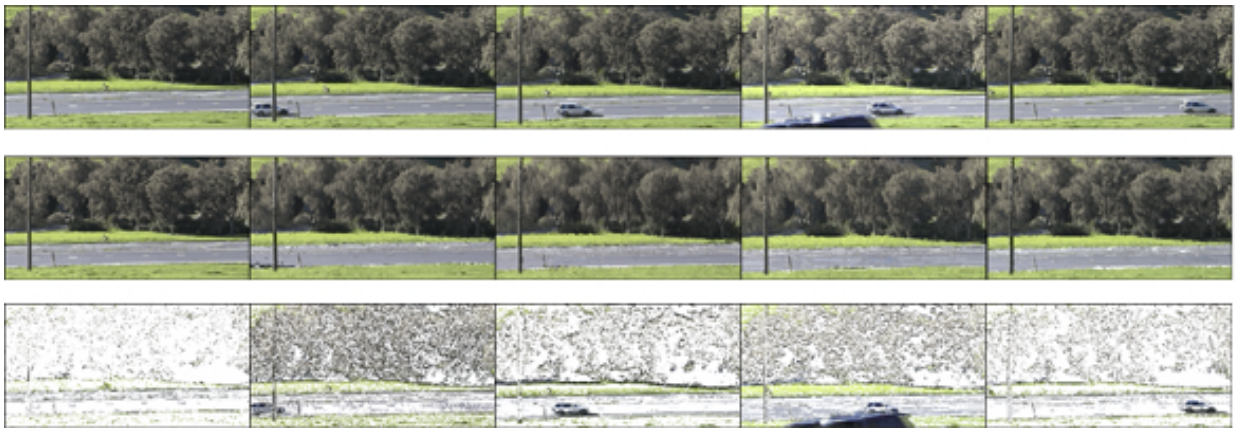


Fig. 2. Segmentation of pagemill02.avi. Left to right: frames 2, 26,52,78,104. Top to bottom: Original, Background, Foreground. Note that the background sequence has less residue comparing to that of Pagemill01. The starting frames of the sequence does not contain vehicles, so the background model became stable more quickly.

The situation here is similar to a rotating fan. However, with the omnipresent, jittering global motion, the Gaussian mixture model failed to segment these details. This is because (i)the contrast between the mark and the road is large (ii) this global motion is repetitive and small. We tested this phenomenon by applying the algorithm on a synthetic sequence with a red box on top of a black background. The box moves vertically with ± 1 pixel from frame to frame. The result showed that the part of the box is always existing in the foreground.

When we check the mean values of the K distributions in the model for pixels near the box region, we found that it is composed of interleaving extreme values. Because of the nature of the synthetic motion, at regions near the top and bottom of the box the new observation will match one of the distributions, but never the most possible (highest w/σ) in the background model. According to the algorithm, the rest of the distributions are averaged to represent the foreground, so it would always present in the extracted foreground sequence. In the case of the pagemill01 sequence, around one of the lanemarks the mixture model contains distributions of interleaving mean values with differences of about 100, which coincides the above explanation.

To eliminate the artifacts produced by the global motion, we performed a simple motion compensation method to be discussed in the next section.

D. Global Motion Compensation

1) Global motion estimation using phase correlation method: **The motion estimation method used here is phase correlation estimation. The basic idea of this block-based estimation is to extract the motion vector from the change of phase in the frequency domain. Assume that two blocks**

Fig. 3. Segmentation of freeway.avi. Left to right: frames 2, 26,52,78,104. Top to bottom: Original, Background, Foreground. This sequence has less global motion, so the extracted foreground appear to be clearner than those of the previous sequences.

are related by purely a translation $\mathbf{d}$):

$$\psi_1(\mathbf{x}) = \psi_2(\mathbf{x}+\mathbf{d})$$
$$\Psi_1(\mathbf{f}) = \Psi_2(\mathbf{f}) \cdot e^{j2\pi \mathbf{d}^T \mathbf{f}}$$

The normalized cross-power spectrum between the blocks can be calculated as:

$$\tilde{\Psi}(\mathbf{f}) = \frac{\psi_1(\mathbf{x}) \cdot \Psi_2^*(\mathbf{f})}{|\psi_1(\mathbf{x}) \cdot \Psi_2^*(\mathbf{f})|} = e^{j2\pi \mathbf{d}^T \mathbf{f}}$$

After the Fourier inversion we obtain the *phase correlation function*(PCF):

$$PCF(\mathbf{x}) = \mathcal{F}^{-1}(\tilde{\Psi}(\mathbf{f})) = \delta(\mathbf{x}+\mathbf{d})$$

The normalized correlation is used to compensate for the possible change in luminance and the noise. The algorithm simply picks the peak from the PCF as the 2-D translational motion vectors.

We chose phase correlation estimation because it is fast comparing to the block-matching estimation. The efficiency is desirable for the real-time application context. In our experiment we used four $32x32$ windows near the corners from each frame (using the Y values only), and the average of the motion vector obtained from the four blocks would be our esitmate. In order to get subpixel precision, we zero-padded the spectrum to $64x64$ before the inverse FFT. For the pagemill01 sequence the resulting estimated global motion vectors are shown in *fig.8*.

2) Experiments and Results: **After obtaining the motion vectors we tried to align the frames with the first frame. We generated a series of $340x160$ frames (originally $360x180$) frames from the original sequence according to the motion vectors. Because the global motion vectors estimated have subpixel precision, we used `interp2` in MATLAB to generate the compensated frames. We then input the compensated sequence to the segmentation algorithm we developed in the previous parts. The results are shown together with the non-compensated results in *fig.9*.**

We observed from this segmentation that some obvious global-motion related problems are eliminated, for example the eletrical pole is now always present in the background. Also, the blurring of the background is less obvious, and the foreground is cleaner. However, the artifacts did not completely go away; we can still see the lane marks from the foreground segmentation. Moreover, from the segmented background we can see the model converges slower than the non-compensated case. We suspect the non-ideal result comes from the lack of accuracy in the motion estimation phase, and also the interpolation in the motion compensation phase.

E. Future Improvements

We can try to apply some filtering technique on the phase correlation estimation, since it nees a big enough block size yet also should be able to capture the details. Also, we can use pixel-based methods such as optical flow estimation to get more accurate global motion vectors before the compensation.

IV. Motion-based Segmentation

Objects moving in a 3D scene are projected onto the image plane as regions undergoing motion. Depending on the geometry of the camera with respect to the scene, this motion can be described by a parametric motion model. The

Fig. 9. Segmentation of pagemill01.avi. Left to right: frames 2, 26,52,78,104. Top to bottom: Original, Background (without motion compensation), Background(with compensation), Foreground(with compensation). With compensation, the electrical pole constantly appear in the backgrond, and the foreground is cleaner than before compensation.

idea of motion-based segmentation is to extract the moving objects by identifying these regions of coherent motion.

Our approach consists of first estimating the optical flow for every pixel in the image. We then attempt to group these pixels into regions by performing clustering in the motion parameter space. In our case a translational motion model is sufficient to describe the motions present in the image and we can directly use the optic flow estimates as our motion parameters. For more complex models an additional step is required to estimate the motion parameters from the optic flow estimates, and it may be necessary to iterate between the motion parameter and region estimation to find the solution.

We have thus broken down the motion segmentation problem into two sub-problems: (i) optic flow estimation, and (ii) clustering of motion parameters.

A. Optic Flow Estimation

We estimate the motion vectors (v_x, v_y) at each pixel location by solving the Optical Flow Equation (OFE):

$$I_x v_x + I_y v_y + I_t = 0$$

where I_x, I_y and I_t are the spatial-temporal derivatives of intensities we can compute for each frame in the image sequence.

The OFE is under-constrained (1 equation with 2 unknowns), but we can solve it by imposing additional constraints, for example by assuming neighbouring pixels have the same motion. We used a 5×5 window, and use least-squares to solve for the motion vectors:

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum -I_x I_t \\ \sum -I_y I_t \end{bmatrix}$$

where the summation is taken over a 5x5 pixel Gaussian window with σ of 1 pixel. Summing over a Gaussian window gives more weight to the center pixels and less to those at the peripheral, and the result is a weighted least squares solution [8].

The validity of the OFE rests upon two assumptions: constant intensity along motion trajectory and 'small' motion (where 'small' in practice means $\sim$1 pixel/frame). The small motion assumption is often violated — for example, in the video sequences we used, even the slower moving cars had velocities of 3-4 pixels/frame. This leads us to a multi-resolution or hierarchical approach.

In a hierarchical framework, we first generate a Gaussian pyramid of images by iteratively filtering and subsampling by a factor of 2. We used a 5-by-5 separable filter (or generating kernel) defined by

$$w(m,n) = \hat{w}(m)\hat{w}(n)$$

with

$$\begin{aligned} \hat{w}(0) &= a \\ \hat{w}(-1) = \hat{w}(1) &= 1/4 \\ \hat{w}(-2) = \hat{w}(2) &= 1/4 - a/2 \end{aligned}$$

where a is set to 0.6 as suggested by Burt [3].

At each level of the pyramid, starting from the highest (lowest resolution) we

(i) **project and interpolate estimates from previous level**

(ii) **pre-warp the neighbourhood window according to the motion estimates computed from the previous level before performing least squares. In our implementation, we assume the entire window has the same**

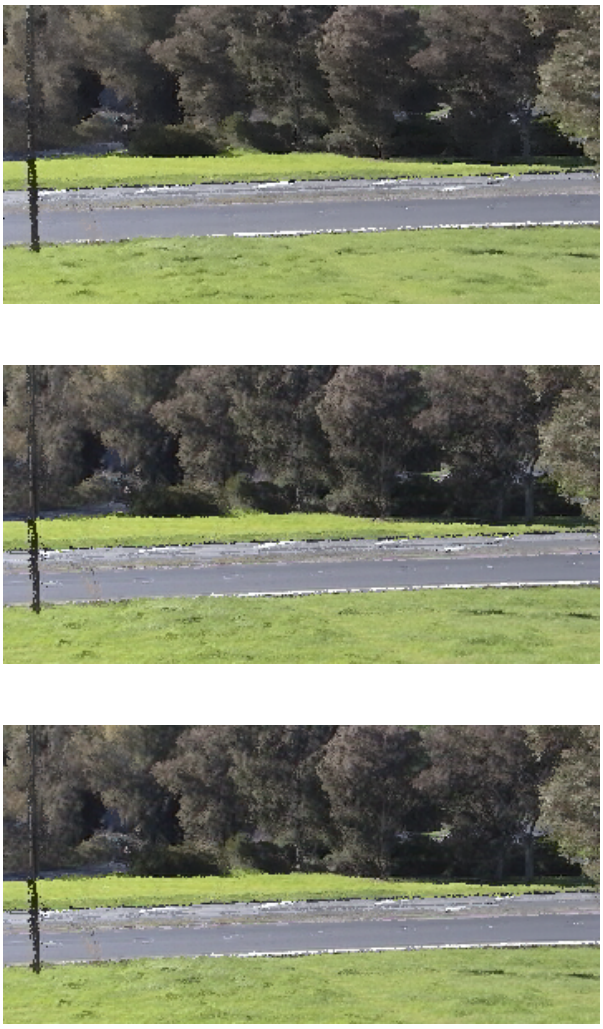


Fig. 5. Background of frame 100 from *Pagemill01*.$K = 3, 4, 5$ The sharpness of the tree area increases with K.

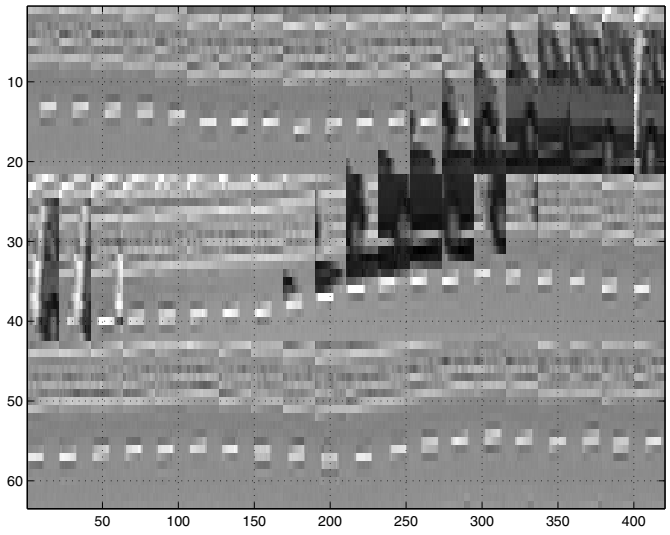


Fig. 6. 'Trace' of the one lane mark with time Top: frames 1 to 40, middle: frames 41 to 80, bottom: frames 81 to 120.

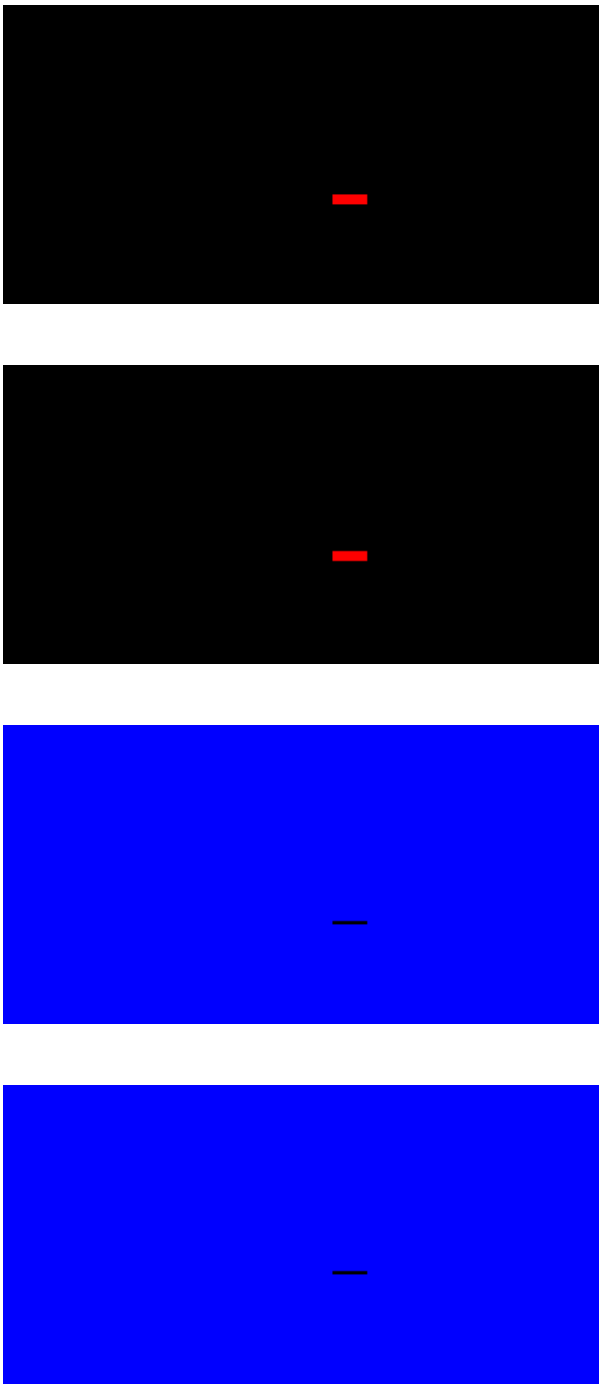


Fig. 7. Top: orignal frames of synthetic sequence. Bottom: extracted foreground of synthetic sequence.

motion estimate as the center pixel, and simply shift the entire window by the nearest integer number of pixels to reduce the interpolation operations required. The spatial and temporal derivatives are then calculated on the shifted window.

(iii) **the least squares solution is added to the initial estimates to give give a refined estimate for that level.**

In this fashion as we descend the pyramid, our motion

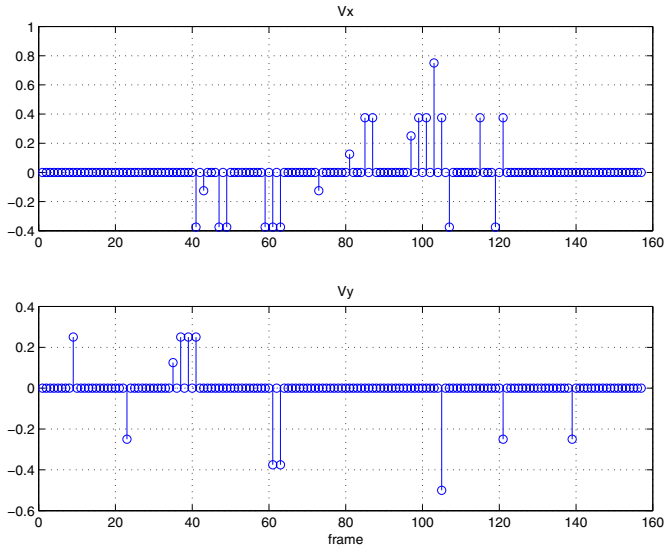


Fig. 8. Estimated global motion vectors. Top: horizontal, bottom: vertical

vector estimates become more and more accurate. The initial estimates at the highest level are set to zero.

At each level, the algorithm only computes the optic flow for rectangular areas where the motions exceed a certain threshold. The threshold was set to 2 pixels/frame at the highest resolution level, corresponding to a threshold 1 and 0.5 pixels/frame at levels 1 and 2 respectively. This threshold value is not sensitive as long as the vehicle motions are significantly larger than the global motion caused by the camera. For our video sequences where the moving vehicles only occupy a small area we were able to significantly reduce the computational load by this method.

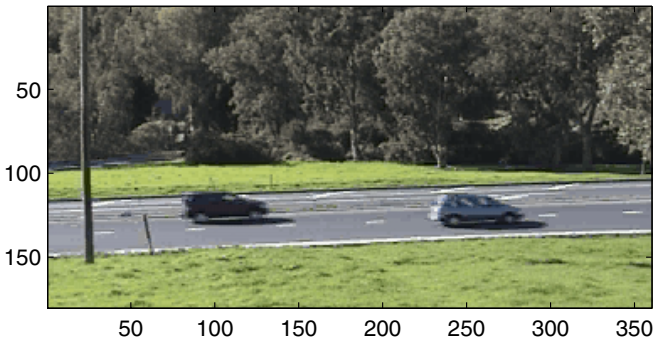
The number of levels used in the pyramid determines the largest motion that we can resolve accurately by solving the OFE. We used a 3-level pyramid, and assuming the OFE is applicable for motions up to 1.5 pixels/frame, the largest motion that can be validly computed is 6 pixels/frame. For our application here this is adequate but we note that the range can easily be doubled by adding another level.

1) Computational issues: **The importance of numerical differentiation and spatiotemporal smoothing was emphasized by Barron [1], and our experiences corroborates this observation. In particular we found that temporal smoothing improved the robustness of the direction of the velocity estimates at the cost of the magnitude. To compute the spatial derivatives, we used a 3-point central difference (filter coefficients $[-0.5, 0, 0.5]$) computed on the average of the two frames.**

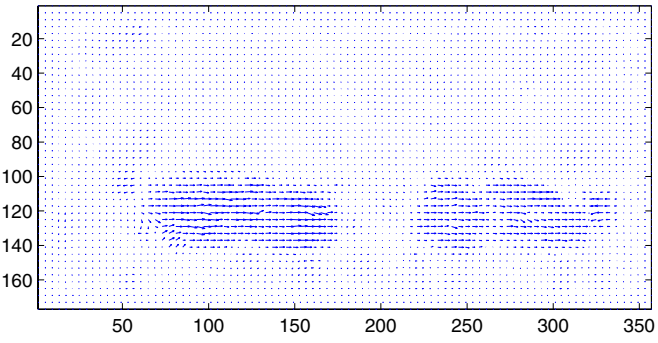
2) Error propagation: **With a multi-resolution approach, errors in the velocity estimates in the higher levels will propagate to a larger and larger spatial area as we descend the pyramid. The sub-sampling performed also means that the magnitude of error is doubled as it is projected and interpolated to the next level. It is therefore crucial that the estimates in the higher levels be accurate, or it will be impossible for the lower levels to recover from such an error. In our implementation, at the lowest resolution level, we used a Gaussian filter with a relatively large σ of 1.5 frame periods (requiring a support of 15 frames) to pre-smooth the image pairs before taking the temporal derivative (frame difference). As we descend the pyramid, we gradually reduce σ to 1.0 and finally 0.5 at the original resolution to reduce the amount of blurring caused at the higher resolutions. We stress here the importance of temporal pre-smoothing, without which the least squares solution gives almost random results.**

3) Edge effects: **Using a 5x5 neighbourhood window, we are unable to calculate velocity estimates for a border region 2 pixels wide without assuming the values outside the boundary given for the image. With a 3-level pyramid, the 2 pixel border at level 2 projects to an 8 pixel border region with no velocity estimates (and hence no segmentation). In our case we have a reasonably large frame size and can afford to lose nearly 10 pixels at each side. However for smaller video sizes this could be a concern and more elaborate schemes for dealing with edges may have to be considered especially if a multi-resolution approach is to be used.**

4) Confidence measure of estimates: **We determine whether to accept or reject the weighted LS solution by simple thresholding. If either the x or y component of the velocity exceeds 1.5, the small motion assumption of the optical flow equation is violated. The LS solution is discarded but we retain the initial estimates from the higher level for that pixel.**



(a) original image



(b) estimated optical flow

Fig. 10. Optical flow estimation (frame 39, pagemill001.avi)

B. Clustering of motion parameters

Object motion in the image sequences used can be modelled by a translational motion model, and thus we are able to use the optic flow estimates v_x and v_y directly as our motion parameters. We used k-means clustering with $k = 5$, with a feature vector

$$\mathbf{x} = [v_x, v_y, i, j, R, G, B]'$$

where i, j are the pixel locations to add spatial information to the vector, and R, G, B are the intensities of the color components at that point. The distance between two feature vectors was measured using a Mahalanobis distance, defined by

$$d(\mathbf{x_1}, \mathbf{x_2}) = (\mathbf{x_1} - \mathbf{x_2})\Sigma(\mathbf{x_1} - \mathbf{x_2})^T$$

where $\Sigma = diag(1, 1, 1, 1, 2, 2, 2)$. The purpose of the matrix Σ is to weight the the color intensities since they are of a different scale (and units) from the other 4 components in the feature vector After some trial-and-error we found 2 to give good results.

The clustering algorithm is implemented as follows:

(i) **threshold magnitude of velocity to determine approximately the regions of interest (vehicle locations). The threshold was set to 2 (pixels/frame).**

(ii) **perform k-means clustering on the entire region extracted in (1).**

(iii) **for each region segmented by k-means clustering, further segment by connectivity and remove unconnected regions with size less than a threshold size, set to 200 pixels.**

C. Results

1) Optic flow estimation: **The accuracy of the optic flow estimation has a significant impact on the quality of the eventual segmentation. Figure 11 shows a typical result of the optic flow estimation algorithm. In the frame, the car is moving towards the right.**

We can see that the horizontal motion vectors are not confined to the car but leaks out considerably into the neighbouring 'road' pixels. This is a consequence of the method we used — by performing a least-squares fit over a 5x5 neighbourhood window, we are implicity smoothing the motion across motion boundaries. The multi-resolution approach further exacerbates this problem as every motion vector found at the n-th level in the pyramid eventually projects onto an area of $2^n \times 2^n$ pixels at the original resolution. The increased leakage of motion vectors along the trajectory of the car is due to temporal pre-smoothing. Low pass filtering in time reduces the sensitivity of our optic flow estimates to noise. We experimented with several different values of the σ used in the Gaussian temporal filter at the lowest resolution image, and found that $\sigma = 1.5$, as suggested by Barron [1] offers a good trade-off between robustness and resolution.

Our implementation also leads to numerous invalid motion vectors near motion boundaries: e.g. the edges of the cars. Various methods have been proposed to identify these invalid estimates, for example by examining the eigenvalues of the A-matrix in the LS formulation [1] — the larger the minimum eigenvalue, the more confidence we should have in the estimate. We investigated this approach but perplexingly the motion estimates for the interior pixels of cars at the lowest level turned out consistently to be very unreliable according to this criterion. We conjecture that the car interior may have insufficient texture at the highest resolution images for the least squares solution to be well-conditioned In our particular case where a translational model is used, these randomly pointing vectors near the object edge may actually work to our favor making the clustering procedure easier. However in the more general case where we have to iteratively estimate the motion model parameters and the motion region these invalid motion vectors may be a more serious problem.

2) Segmentation results: **Initially we used a feature vector containing only the 2 components of the translational velocity and the pixel locations. Figure 12 shows the result of segmentation in this 4-space. As expected from our previous discussion of the smoothing effects of our optical flow estimates, the segmented car includes a considerable amount of road pixels, in particular along the direction of motion.**

To improve the segmentation R,G,B color intensities were added to the feature vector With this extension we were able to get much 'cleaner' segmentations. For comparison, figure 13 shows the results of segmenting the same frame in figure 12 but with the new color information added included. There are some undesirable effects, such as the windows being segmented off because of the high contrast between it and the the white car body. Particularly good results were obtained for vehicles with colors having high contrast with

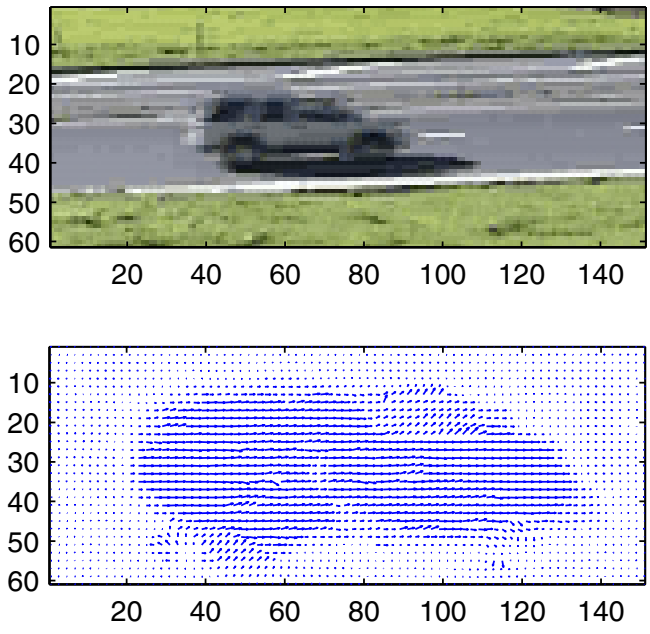


Fig. 11. Optical flow estimated around car (frame 60, pagemill002.avi)

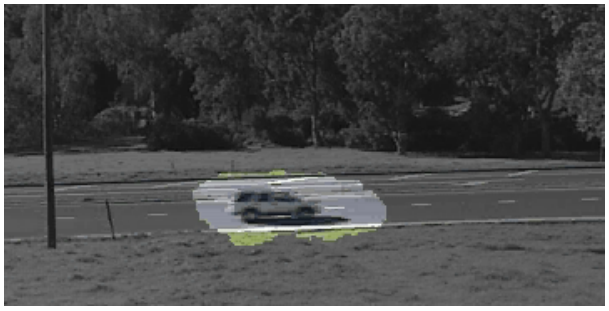
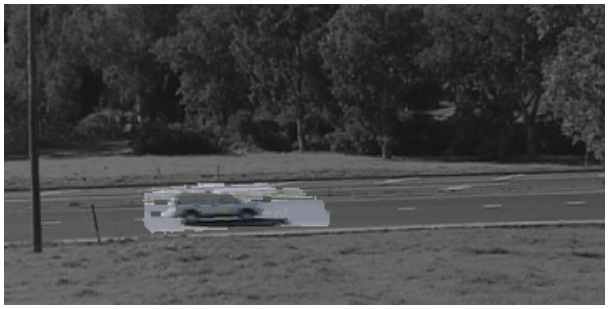
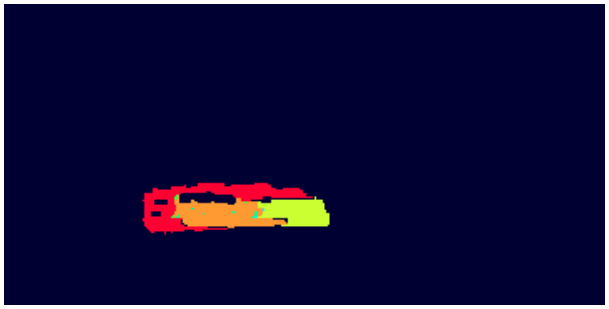


Fig. 12. Segmentation without color information (frame 92, pagemill002.avi)

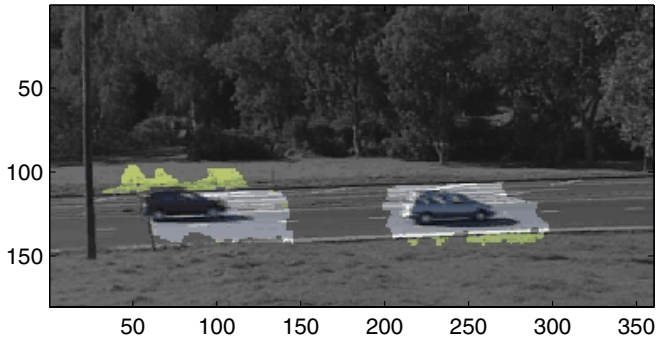


(a) original image (segmented regions in color)

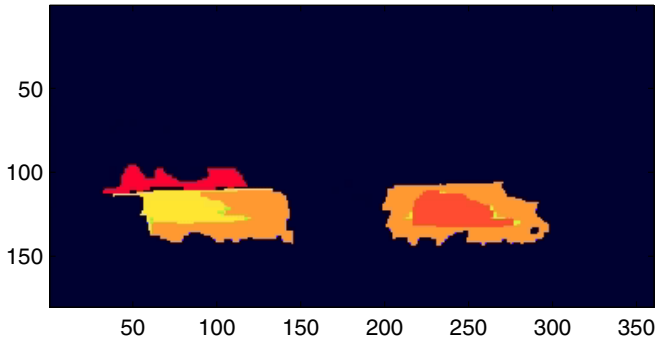


(b) segmentation map

Fig. 13. Segmentation with color information (frame 92, pagemill002.avi)



(a) original image (segmented regions in color)



(b) segmentation map

Fig. 14. Segmentation results (frame 26, pagemill001.avi)

the road (figure 14). Nevertheless, having an accurate optic flow estimate remains the key to a good segmentation.

As currently implemented in Matlab, our algorithm can process a frame of 180x360 pixels in approximately a minute. About two-thirds of the processing time is taken up by optic flow estimation. Further work will investigate methods to identify and track the cars in the regions segmented.

V. Conclusions

From the two different methods implemented in the project, we would like to make the following comments:

(i) **Adaptive filtering method such as the background mixture model is easy to implement, and can give sub-optimal segmentation results. The basic idea of the method is simple: properly take data from the current input and place it either in the background or the foreground. So the results is always not too far away from what is expected. The key is to understand how different parameters affects the different artifacts in the segmentation, and also to understand in what situation the method would fail.**

(ii) **Accurate sub-pixel levelmotion estimation is difficult to achieve and also difficult to evaluate. This is clearly exemplfied in the amount of efforts we spent on the dense field optical flow estimation. However, accurate motion estimation is important not only for motion-based segmentation, but also for compensating the non-ideal camera motion. Also, only with motion estimation we are able to extract informations such as car speed. It is the most important but difficult stage for motion-based processing.**

Comparing the two methods we would say that they are suitable for different purposes. Although easy to implement, the background mixture model can gave us only the segmentation. The motion-based method allows more possible applications, such as measuring car speed or l ayered processing, it is on the other hand harder to implement.

As the project was originally aimed at separating layers from the traffic video sequence, we hoped that we could somehow combine the efforts of the two widely different

methods to achieve the goal. Although it turned out in the given time we were unable to reach the stage of integrating the efforts, through the discussions in our individual problem-solving process, we both learned a lot from each other.

REFERENCES

[1] J.L. Barron, D.J. Fleet, S.S. Beauchemin, and T.A. Burkitt. Performance of optical flow techniques. *CVPR*, 92:236–242, 1992.

[2] S. S. Beauchemin and J. L. Barron. The computation of optical flow. *ACM Computing Surveys*, 27(3):433–467, 1995.

[3] Peter J. Burt and Edward H. Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, COM-31,4:532–540, 1983.

[4] F. Dufaux, F. Moscheni, and A. Lippman. Spatio-temporal segmentation based on motion and static segmentation, 1995.

[5] M. Harville, G. Gordon, and J. Woodfill. Foreground segmentation using adaptive mixture models in color and depth. In *Proceedings of the IEEE Workshop on Detection and Recognition of Events in Video*, 2001.

[6] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys (CSUR)*, 31(3):264–323, 1999.

[7] C. Stauffer and W. Grimson. Adaptive background mixture models for real-time tracking. In *Proc. CVPR*, pages 246–252, 1999.

[8] J. Wang and E. Adelson. Spatio-temporal segmentation of video data, 1994.

[9] Y. Wang, J. Ostermann, and Y.Q. Zhang. *Video Processing and Communications*. Prentice Hall, 2002.

APPENDIX

Matlab code: Adaptive background mixture model segmentation

```
%%============================================================
%%Main code for Adaptive Gaussian Mixture Model Segmentation
%%
%%written by Tun-Yu Chiang, last modified date: 03/12/2002
%%============================================================
% Notes:
% 1. Input/output data are tiff files
% 2. Outputs both foreground and background sequences



clear all;
close all;

%==============================
% Import video data into memory
%==============================

filename='newPM';
totalframe=60;
tempframe= imread([filename '001.tif'],'tif');
height=size(tempframe,1);
width=size(tempframe,2);
frames=zeros(size(tempframe,1),size(tempframe,2),3,totalframe);
for i=1:totalframe
   j=2*i;
   if (j<10)
      name = [filename '00' int2str(j) '.tif'];
   elseif (j>=10 & j<100)
      name = [filename '0' int2str(j) '.tif'];
   else
      name = [filename int2str(j) '.tif'];
   end;
frames(:,:,:,i+1)=imread(name,'tif');

end;




%====================================
% Set up parameters and empty matrices
%====================================


%setup parameters: K= #of guassians in the mixture
%  alpha = learning constant
%  T = min. portion of background
%  initVar = initial variance
%  minDiff = min. pixel value diff for thresholding

K=5;
alpha=0.05;
T=0.6;
initVar=125;
```

```
minDiff=30;


%setup matrices
Bframe=zeros(height,width,3);
Fframe=zeros(height,width,3);

frameNow=zeros(height,width,3);
framePrev=zeros(height,width,3);
frameDiff=zeros(height,width,3);

map=ones(height,width);
pixel=zeros(3,1);
miu=zeros(3,1,K);
sigma=ones(1,K);
weight=zeros(1,K);
bgweight=zeros(1,K);

%setup parameters for the guassian mixture model:
w=ones(height,width,K)/K;
Gmiu=zeros(height,width,3,K);
Gsigma=ones(height,width,K);




%=======================================================
%Extract Foreground and Background by K-mixture model
%=======================================================

%initialize g-mixture model
Gsigma = Gsigma*initVar;

for k=1:K
   Gmiu(:,:,1,k)=tempframe(:,:,1);
   Gmiu(:,:,2,k)=tempframe(:,:,2);
   Gmiu(:,:,3,k)=tempframe(:,:,3);
end;


%adaptive g-mixture background segmentation

for frnum=2:totalframe
   %get current frame and the refernece frame
   frameNow=frames(:,:,:,frnum);
   if (frnum<=20)
      tempframe=frames(:,:,:,1);
   else
      tempframe=frames(:,:,:,frnum-20);
   end;


   %thresholding to get the map for pixels to perform the process
   frameDiff(:,:,1)=abs(frameNow(:,:,1)-tempframe(:,:,1));
   frameDiff(:,:,2)=abs(frameNow(:,:,2)-tempframe(:,:,2));
frameDiff(:,:,3)=abs(frameNow(:,:,3)-tempframe(:,:,3));
   map=ones(height,width);
   map(frameDiff(:,:,1)>minDiff)=0;
```

```
map(frameDiff(:,:,2)>minDiff)=0;
map(frameDiff(:,:,3)>minDiff)=0;


%extract the parts considered "stable background" from current frame
Bframe(:,:,1)=map.*frameNow(:,:,1);
Bframe(:,:,2)=map.*frameNow(:,:,2);
Bframe(:,:,3)=map.*frameNow(:,:,3);

%reset foreground frame
Fframe(:,:,:)=0;
Fframe(:,:,1)=map.*zeros(size(frameNow,1),size(frameNow,2));
Fframe(:,:,2)=map.*zeros(size(frameNow,1),size(frameNow,2));
Fframe(:,:,3)=map.*ones(size(frameNow,1),size(frameNow,2))*255;


%gaussian mixture matching & model updating
[i,j]=find(map(:,:)==0);
for k=1:size(i,1)

   pixel=reshape(frameNow(i(k),j(k),:),3,1);
   miu=reshape(Gmiu(i(k),j(k),:,:),3,1,K);
   sigma=reshape(Gsigma(i(k),j(k),:),1,K);
   weight=reshape(w(i(k),j(k),:),1,K);

      %update gaussian mixture according to new pix value
      match=findMatch2(miu,sigma,pixel);
      wmask=zeros(1,K);
      if(match(1)==1)
         [tempmiu, tempsig]=updateMatched2(miu(:,:,match(2)),...
            sigma(match(2)), pixel,alpha);
         miu=tempmiu;
         sigma=tempsig;
         Gmiu(i(k),j(k),:,match(2))=miu;
         Gsigma(i(k),j(k),match(2))=sigma;
         wmask(match(2))=1;
         weight=updateWeightRGB(weight,wmask,alpha);

         bgweight=weight./sqrt(sigma);
         [maxW,maxWIdx]=max(bgweight);

         if (maxWIdx==match(2))
             Fframe(i(k),j(k),:)=[0;0;255];
         else
              Fframe(i(k),j(k),:)=miu;
         end;

      else %no match found
         bgweight=weight./sqrt(sigma);
         [minW,minWIdx]=min(bgweight);
         Gmiu(i(k),j(k),:,minWIdx)=pixel;
         Gsigma(i(k),j(k),minWIdx)=initVar;
         %consider this pixel as part of the foreground image
         Fframe(i(k),j(k),:)=pixel;

      end;
      w(i(k),j(k),:)=weight;
```

```
          %decide background distribution and assign background value
          bgweight=weight./sqrt(sigma);
          [bgSort,bgIdx]=max(bgweight);
          miu=reshape(Gmiu(i(k),j(k),:,:),3,1,K);
          Bframe(i(k),j(k),:)=miu(:,:,bgIdx(1));
       end;

        Bframe(:,:,1)=Bframe(:,:,1)./max(max(Bframe(:,:,1)));
Bframe(:,:,2)=Bframe(:,:,2)./max(max(Bframe(:,:,2)));
          Bframe(:,:,3)=Bframe(:,:,3)./max(max(Bframe(:,:,3)));

        Fframe(:,:,1)=Fframe(:,:,1)./max([1,max(max(Fframe(:,:,1)))]);
Fframe(:,:,2)=Fframe(:,:,2)./max([1,max(max(Fframe(:,:,2)))]);
          Fframe(:,:,3)=Fframe(:,:,3)./max([1,max(max(Fframe(:,:,3)))]);

         imwrite(Bframe,['Bg' int2str(frnum) '.tif'],'tiff');
         imwrite(Fframe,['Fg' int2str(frnum) '.tif'],'tiff');
end;



%%=============================================================
%% function updateMacthed2 for the main code
%%
%% written by Tun-Yu Chiang
%%=============================================================
%% update the mean and variance value of the matched distribution
%% according to the learning rate alpha

function [miu,sigma]=updateMatched2(oldmiu,oldsigma,pix,alpha)

miu=(1-alpha)*oldmiu + alpha*pix;
sigma=(1-alpha)*oldsigma+ alpha*(pix-miu)'*(pix-miu);
%%=============================================================
%%Code for Generating Global Motion Compensated Frames
%%
%%written by Tun-Yu Chiang
%%=============================================================


clear all;
close all;


totalframe=79;
filename='pagemill';

wsize=31;
margin=10;
winNum=4;
n1=180;
n2=360;
Vx=zeros(1,totalframe);
Vy=zeros(1,totalframe);
```

```
name=[filename '001.tif'];
tempframe=double(imread(name,'tif'));
prevFrm=RGB2YUV(tempframe(:,:,1),tempframe(:,:,2),tempframe(:,:,3));
prevFrmY=prevFrm(:,:,1);


for frnum=1:2:totalframe*2-1

   if (frnum<10)
      name = [filename '00' int2str(frnum) '.tif'];
   elseif (frnum>=10 & frnum<100)
      name = [filename '0' int2str(frnum) '.tif'];
   else
      name = [filename int2str(frnum) '.tif'];
   end;

   tempframe=double(imread(name,'tif'));
   currFrm=RGB2YUV(tempframe(:,:,1),tempframe(:,:,2),tempframe(:,:,3));
   currFrmY=currFrm(:,:,1);

   cwindow=zeros(wsize,wsize,winNum);
   pwindow=zeros(wsize,wsize,winNum);
   cwindow(:,:,1)=currFrmY(margin:margin+wsize-1,margin:margin+wsize-1);
   cwindow(:,:,2)=currFrmY(margin:margin+wsize-1,n2-margin-wsize+1:n2-margin);
   cwindow(:,:,3)=currFrmY(n1-margin-wsize+1:n1-margin,margin:margin+wsize-1);
   cwindow(:,:,4)=currFrmY(margin:margin+wsize-1,n2-margin-wsize+1:n2-margin);
   pwindow(:,:,1)=prevFrmY(margin:margin+wsize-1,margin:margin+wsize-1);
   pwindow(:,:,2)=prevFrmY(margin:margin+wsize-1,n2-margin-wsize+1:n2-margin);
   pwindow(:,:,3)=prevFrmY(n1-margin-wsize+1:n1-margin,margin:margin+wsize-1);
   pwindow(:,:,4)=prevFrmY(margin:margin+wsize-1,n2-margin-wsize+1:n2-margin);

   mVec = zeros(4,2);

   for i=1:4
      oldblock=zeros(63,63);
      oldblock(17:47,17:47)=cwindow(:,:,i);
      newblock=zeros(63,63);
      newblock(17:47,17:47)=pwindow(:,:,i);
      ftold=fft2(oldblock);
      ftnew=fft2(newblock);
      corltn = fft2(oldblock).*conj(fft2(newblock));

      nCorr= corltn./abs(corltn);
      PCF=fftshift(ifft2(nCorr));


      tempPCF = reshape(PCF,1,size(PCF,1)*size(PCF,2));
      [peak,idx]=max(abs(tempPCF));
      mvx=floor(idx/size(PCF,1));
      mvy=mod(idx,size(PCF,1))-1;

      mvx=mvx-floor(size(PCF,2)/2);
      mvy=mvy-floor(size(PCF,1)/2);
      mVec(i,1)= mvx;
      mVec(i,2)= mvy;

   end;
```

```
    Vx((frnum-1)/2+1)=mean(mVec(:,1))/2;
    Vy((frnum-1)/2+1)=mean(mVec(:,2))/2;

    dx=sum(Vx(1:(frnum-1)/2+1));
    dy=sum(Vy(1:(frnum-1)/2+1));

    [xi,yi]=meshgrid(margin+1+2*dx:1:n2-margin+2*dx,margin+1+2*dy:1:n1-margin+2*dy);
    [x,y]=meshgrid(1:360,1:180);
    newframe=zeros(n1-margin*2,n2-margin*2,3);
    newframe(:,:,1)=interp2(x,y,tempframe(:,:,1),xi,yi);
    newframe(:,:,2)=interp2(x,y,tempframe(:,:,2),xi,yi);
    newframe(:,:,3)=interp2(x,y,tempframe(:,:,3),xi,yi);

    newframe(:,:,1)=newframe(:,:,1)./max([1,max(max(newframe(:,:,1)))]);
    newframe(:,:,2)=newframe(:,:,2)./max([1,max(max(newframe(:,:,2)))]);
    newframe(:,:,3)=newframe(:,:,3)./max([1,max(max(newframe(:,:,3)))]);

    imwrite(newframe,['mc' name(length(filename)+1:length(name))],'tiff');

prevFrmY=curnFrmY;
end;


figure;
quiver(Vx, Vy);
```

APPENDIX

MATLAB code: Motion segmentation

```
% main.m

startFrame = 39;    % starting frame to process
endFrame = 39;      % end frame to process
buffSize = 15;

video = 'pagemill001';
file = strcat('avi\', video, '.yuv');


% Initialize buffer
[frameBuffer, rows, cols] = InitFrameBuffer(video, startFrame, buffSize);

minRegionSize = 200; % at level 0
minSpeed = 2;        % pixels/frame at level 0

tic;

for frame=startFrame:endFrame
   fprintf('\n\nProcessing frame %.3g (start/end - %.3g/%.3g)', frame, startFrame, endFrame');
   fprintf('\n-----------------------------------------');
   fprintf('\n-Elapsed time: %.3g minutes\n', toc/60);

   % Compute OF for contiguous regions with motion
   [Vx, Vy, regionMap] = ComputeOF(frameBuffer, rows, cols, minRegionSize, minSpeed);

   % color info added
   colorIm = double(imread(strcat('tifs\', sprintf('\\%.3d.tif', frame))));
   classMap = newCSegmenter2(colorIm, Vx, Vy, regionMap, minRegionSize);

   fileOut = strcat('newmaps\', video,sprintf('\\%.3d.tif',frame));
   imwrite(uint8(classMap), fileOut, 'tif');

   % update buffer
   frameBuffer(:,1) = [];
   im = read_yuv(file, frame+ceil(buffSize/2));
   frameBuffer = [frameBuffer im(:)];
end

fprintf('\nTotal elapsed time: %.3g minutes', toc/60);
fprintf('\n');
```

```
function [Vx, Vy, regionMap] = ComputeOF(frameBuffer, rows, cols, minRegionSize, minSpeed)

% COMPUTE OPTIC FLOW (Vx,Vy) IN IMAGE
fprintf('\nComputing optic flow');

% LEVEL 2 - top level
fprintf('\n-Level 2\n');
[im1L2 im2L2] = genPyramidImages(frameBuffer, 2, rows, cols, 1);
[Vx2, Vy2] = EstimateOFInit(im1L2, im2L2);

% LEVEL 1
fprintf('\n-Level 1\n');
level = 1;
[r c] = size(im1L2);
```

```
Vx1Est = 2*interp2(1:c, [1:r]', Vx2, .75:.5:c+.25, [.75:.5:r+.25]');
Vy1Est = 2*interp2(1:c, [1:r]', Vy2, .75:.5:c+.25, [.75:.5:r+.25]');

speed = sqrt(Vx1Est.^2 + Vy1Est.^2);
regionMap1 = segmentRegions(speed, minSpeed/(2^level), minRegionSize/(2^level));

[im1L1 im2L1] = genPyramidImages(frameBuffer, 1, rows, cols, 1);
[Vx1, Vy1] = EstimateOpticFlow(im1L1, im2L1, Vx1Est, Vy1Est, regionMap1);


% LEVEL 0
fprintf('\n-Level 0\n');
[r c] = size(im1L1);
Vx0Est = 2*interp2(1:c, [1:r]', Vx1, .75:.5:c+.25, [.75:.5:r+.25]');
Vy0Est = 2*interp2(1:c, [1:r]', Vy1, .75:.5:c+.25, [.75:.5:r+.25]');

speed = sqrt(Vx0Est.^2 + Vy0Est.^2);
regionMap0 = segmentRegions(speed, minSpeed, minRegionSize);

[im1L0 im2L0] = genPyramidImages(frameBuffer, 0, rows, cols, 1);
[Vx0, Vy0] = EstimateOpticFlow(im1L0, im2L0, Vx0Est, Vy0Est, regionMap0);

Vx = Vx0;
Vy = Vy0;
regionMap = regionMap0;

function [Vx, Vy] = EstimateOpticFlow(im1, im2, VxEst, VyEst, regionMap)

[rows, cols] = size(im1);

% window function used in LS solution to give
% more influence to constraints at center of neighbourhood than
% those at periphery.
% weighting from Barron, et al. Performance of Optical Flow Techniques.
W = [.0625, .25, .375, .25, .0625]';
W = W*W';
W = W(:);
W = diag(W);

% generate LPF gaussian filter used in gradients2 function for efficiency
sigma = 1.5;
LPF = fspecial('gaussian', ceil(6*sigma), sigma);


% initialize Vx, Vy with estimates from previous levels
Vx = VxEst;
Vy = VyEst;

% translation in pixels for each location
deltaX = round(VxEst);
deltaY = round(VyEst);

fprintf('  Processing region (total %0.2g): ', max(regionMap(:)));

for region = 1:max(regionMap(:))
   [regionY regionX] = find(regionMap==region);

   fprintf('%0.2g.', region);
```

```
   for x=min(regionX):max(regionX)
      for y=min(regionY):max(regionY)

         % select window to compute derivatives on
         % window size is 1 larger than required since 2-pt difference used in
         % computing gradients
         win1 = im1(y-3:y+3, x-3:x+3);
         win2 = im2(y-3+deltaY(y,x):y+3+deltaY(y,x), x-3+deltaX(y,x):x+3+deltaX(y,x));

         [Ixc, Iyc, Itc] = gradients2(win1, win2, LPF);

         Ixc = Ixc(2:6, 2:6);
         Iyc = Iyc(2:6, 2:6);
         Itc = Itc(2:6, 2:6);

         % calculate optic flow (over entire frame)
         A = W*[Ixc(:) Iyc(:)];
         b = -W*Itc(:);

         v = A\b;

         if (max(v)<1.5)
            Vx(y,x) = deltaX(y,x) + v(1);
            Vy(y,x) = deltaY(y,x) + v(2);
         end

      end
      if (mod(x,32)==0)
         fprintf('.');
      end
   end
end
fprintf('done');

function newRegionMap = newCSegmenter2(colorIm, Vx, Vy, regionMap, minRegionSize)

[rows cols] = size(Vx);

% get x,y coordinates of region
[y2 x2] = find(regionMap);
idx2 = find(regionMap(:));
regionSize = size(idx2,1);

red = 2*colorIm(:,:,1);
green = 2*colorIm(:,:,2);
blue = 2*colorIm(:,:,3);

% if no regions of interest then return
if (isempty(idx2))
   newRegionMap = zeros(rows, cols);
   fprintf('-No regions of interest for segmentation');
   return;
end

featureMatrix = [Vx(idx2) Vy(idx2) x2 y2 red(idx2) green(idx2) blue(idx2)];

k=5;      % number of clusters to use in k-means clustering
```

```
N = 500;  % training data points to use
class = qkmeans(featureMatrix, k, N);

% produce initial segmentation map
initMap = zeros(rows,cols);
for n=1:regionSize
   initMap(y2(n),x2(n)) = class(n);
end


% remove isolated regions and write final result to newRegionMap
newRegionMap = zeros(rows, cols);

for r = 1:max(initMap(:))
   regionR = bwlabel(initMap==r);  % label current region

   for subregion = 1:max(regionR(:))
      if (sum(regionR(:)==subregion)>minRegionSize)
         newRegionMap = newRegionMap + (r*5+subregion)*double((bwmorph(regionR==subregion, 'clos
      end
   end
end


function [Vx, Vy] = EstimateOFInit(im1, im2)

[rows, cols] = size(im1);

border = 3;

% first calculate gradients
[Ix Iy It] = gradients(im1, im2);

[r c] = size(im1);

Vx = zeros(r,c);
Vy = zeros(r,c);

% window function used in LS solution to give
% more influence to constraints at center of neighbourhood than
% those at periphery.
% weighting from Barron, et al. Performance of Optical Flow Techniques.
W = [.0625, .25, .375, .25, .0625]';
W = W*W';
W = W(:);
W = diag(W);

fprintf('  ');

for x=1+border:cols-border
   for y=1+border:rows-border
      xRange = x-2:x+2;
      yRange = y-2:y+2;

      Ixc = Ix(yRange, xRange);
      Iyc = Iy(yRange, xRange);
      Itc = It(yRange, xRange);
```

```
      % calculate optic flow
      A = W*[Ixc(:) Iyc(:)];
      b = -W*Itc(:);

      v = A\b;

      if (max(v)<1.5)
         Vx(y,x) = v(1);
         Vy(y,x) = v(2);
      end

      %      end
   end
   if (mod(x,32) == 0)
   fprintf('%0.2g%%..', (100*(x-1)/(cols-border)))
   end
end
fprintf('done');
```

```
function [im1, im2] = genPyramidImages(frameBuffer, level, rows, cols, LPF)

taps = 13;  % # taps used by temporal LP gaussian filter - should be odd
ctap = ceil(13/2);  % center tap

if (LPF)
   switch level
   case 0,
      sigma = .5;  % set sigma <.2 for no temporal filtering at level 0
   case 1,
      sigma = 1;
   case 2,
      sigma = 1.5;
   otherwise,
      error('only 2 levels supported');
   end
   g = fspecial('gaussian', [taps 1], sigma);

   im1 = reshape(frameBuffer(:, 2:taps+1)*g, rows, cols);
   im2 = reshape(frameBuffer(:, 3:taps+2)*g, rows, cols);
else
   im1 = reshape(frameBuffer(:,ctap+2),rows,cols);
   im2 = reshape(frameBuffer(:,ctap+3),rows,cols);
end

for n=1:level
   im1 = reduce(im1);
   im2 = reduce(im2);
end
```

```
function [frameBuffer, rows, cols] = InitFrameBuffer(video, frameNum, buffSize)

frameBuffer = [];

center = ceil(buffSize/2);

file = strcat('avi\', video, '.yuv');

% fill frame buffer
for n=1:buffSize
```

```
   im = read_yuv(file, frameNum-center+n);
   frameBuffer(:,n) = im(:);
end

[rows cols] = size(im);
```

```
function length = kDistance(x, dest)
% FUNCTION
% length = kDistance(x, dest)
%
% Computes Euclidean distance from point x to
%  the set of points in matrix dest.
%  (specified by the rows)
%

length = []; % initialize length vector


% convert x into column vector if not already so
if (size(x,1)<size(x,2))
   x = x'; % origin vector;
end

% if dest is a vector (not a matrix)
% then make sure it is a row vector
if (sum(size(dest)==1)>0)
   if size(dest,1)<size(x,2)
      dest = dest';
   end
end

for n=1:size(dest)
   d = dest(n,:);
   d = d';

   length = [length; sqrt((x-d)'*(x-d))];
end
```

```
function [class, CC] = kmeans(featureMatrix, k)

% FUNCTION
%  [class, CC] = kmeans(featureMatrix, k)
%
% Performs kmeans clustering given the affine
%  parameters as rows in featureMatrix
% k is the number of initial cluster centers to use.
%
% class is the class number assigned to each row of the featureMatrix
% CC are the cluster center parameters

% Parameters
initMinSep = 1;  % min. initial separation of cluster centers
maxIterations = 20; % maximum number of iterations to do classification


% ============================
% STAGE ONE
%  Initialize cluster centers
% ============================
```

```
cc_idx = kmeansInit(featureMatrix, k, initMinSep);

% form matrix whose rows are the cluster centers
CC = [];

k = size(cc_idx);  % actual number of clusters used
                   % (could be < k because not enough centers
                   %  satisfying min. separation could be found)

for n=1:k
   CC = [CC; featureMatrix(cc_idx(n),:)];
end

% ============================
% STAGE TWO
%  Classify iteratively
% ============================

% for debugging %
%hold off;
%points = ['ox+sd^ph'];
%color = ['ymcrgbwk'];
%pn = 1;
% end debugging %

% do classification
fprintf('\n-Locating cluster centers');
for iteration = 1:maxIterations
   fprintf('.');
   distanceC = []; % distances from centers

   for n=1:k
      distanceC = [distanceC kDistance(CC(n,:), featureMatrix)];
   end

   % classify
   [tmp newClass] = min(transpose(distanceC));

   % return if converged
   if (iteration~=1)
      if (sum(newClass'~=class)==0)
         break;
      end
   end

   class = newClass';

   % re-number classes (some may have been eliminated)
   maxValidClass = max(class);
   for n=1:max(class)
      if (isempty(find(class==n)))
         [tmp idx] = find(class==maxValidClass);
         class(idx) = n;
         maxValidClass = maxValidClass-1;
      end
   end
```

```
   % calculate new centers
   for n=1:max(class)
      CC(n,:) = mean(featureMatrix(find(class==n),:),1);
   end
end
```

```
function class = qkmeans(featureMatrix, k, N)

% quick k-means
% for large feature matrices with rows > 500
% Uses random subset (N) of feature vectors (rows of featureMatrix) to
%  find cluster centers, then classify remaining points
%  to the closest centers.


[rows cols] = size(featureMatrix);

% if rows of feature matrix < N then no need to train on subset of data
if (size(featureMatrix,1)<N)
   [class cc] = kmeans(featureMatrix, k);
   return;
end

% use random subset of feature points as training data to build cluster centers
idx = randperm(size(featureMatrix,1));
trainingData = featureMatrix(idx(1:N), :);
[class cc] = kmeans(trainingData, k);


% classify all points using the found cluster centers
fprintf('\n-Classifying points to clusters');
cDistance = [];
for n=1:k
   q = (featureMatrix - ones(rows,1) * cc(n,:)).^2;
   q = sum(q,2);
   cDistance = [cDistance q];
end

[tmp class] = min(cDistance');
```

```
function centers = kmeansInit(featureMatrix, k, minSeparation)

% FUNCTION
%  centers = kmeansInit(featureMatrix, k, minSep)
%
% Initializes the center locations of the k-means
%  clustering algorithm.
% The rows of featureMatrix contain the affine parameters
%  of each block;
%
% k is the initial number of cluster centers desired; and
%
% minSep is minimum separation between the initial cluster centers
%
% centers is a vector of the row # in featureMatrix of the
%  selected cluster centers.
%

blocks = size(featureMatrix,1); % # blocks
```

```
% choose centers
centers = [];
possibleCs = 1:size(featureMatrix,1);

for m=1:k
   idx = ceil(rand * (size(possibleCs,1)-1)) + 1;
   c = possibleCs(idx);

   % add selected index location to centers
   centers = [centers; c];

   for c_idx = 1:size(centers,1)
      x = featureMatrix(centers(c_idx), :);
      distFromX = kDistance(x, featureMatrix);
      possibleCs = intersect(find(distFromX > minSeparation), possibleCs);
   end

   if (isempty(possibleCs))
      break;
   end
end
```

```
function [y,u,v] = read_yuv(filename,framenum)

% MODIFIED FROM read_frame_qcif.
%
% Reads a designated frame from a qcif sequence and outputs
% the y, u, and v components for that frame.  For the provided
% qcif sequences, all of the data for all of the frames is
% contained in a single file.
%
% [y,u,v] = read_frame_qcif(filename,framenum)
%
% Arguments:
%     filename is a string, e.g. 'foreman.qcif' (including quotes)
%     framenum is an integer (the first frame is frame 0)
%
% John Apostolopoulos (EE392J)

rows = 180;   % qcif resolution (quarter-CIF)
cols = 360;

fid = fopen(filename,'r');
fseek(fid,(framenum*rows*cols*1.5),-1);     % Jumps to the desired frame
[temp,count] = fread(fid,[cols,rows],'uchar');
y = temp';
[temp2,count] = fread(fid,[cols/2,rows/2],'uchar');
u = temp2';
[temp2,count] = fread(fid,[cols/2,rows/2],'uchar');
v = temp2';
fclose(fid);
```

```
function [xgrad, ygrad, tgrad] = gradients(im1, im2);
%
% FUNCTION
%  [xgrad, ygrad, tgrad] = gradients(im1, im2);
%
```

```
% Computes the spatial and temporal gradients for the current
%  frame im1 given the next frame im2.
%

sigma = 1.5; % std dev of the gaussian filter used to smooth image

LPF = fspecial('gaussian', ceil(6*sigma), sigma); % gaussian 2D low pass filter

% use average of two images to compute spatial gradients
LPimage = conv2(.5*(im1+im2), LPF, 'same');

xderiv = [.5 0 -.5];
yderiv = [.5 0 -.5]';

xgrad = conv2(LPimage, xderiv, 'same');
ygrad = conv2(LPimage, yderiv, 'same');

% need to lpf time gradient to reduce 'noise' in Vx and Vy estimates
tgrad = conv2(im2-im1, LPF, 'same');
```

```
function [xgrad, ygrad, tgrad] = gradients2(im1, im2, LPF);
%
% FUNCTION
%  [xgrad, ygrad, tgrad] = gradients(im1, im2);
%
% Computes the spatial and temporal gradients for the current
%  frame im1 given the next frame im2.
%
% Same as gradients except need to specify LPF

% use average of two images to compute spatial gradients
LPimage = conv2(.5*(im1+im2), LPF, 'same');

xderiv = [.5 0 -.5];
yderiv = [.5 0 -.5]';

xgrad = conv2(LPimage, xderiv, 'same');
ygrad = conv2(LPimage, yderiv, 'same');

% need to lpf time gradient to reduce 'noise' in Vx and Vy estimates
tgrad = conv2(im2-im1, LPF, 'same');
```

```
function newIm = reduce(im)
% FUNCTION REDUCE
%  computes the 'reduced' version of the input image
%  by a factor of 2 in both dimensions as in
%  Burt & Adelson, The Laplacian Pyramid as a Compact Image Code, 1983

a = 0.6;  % parameter given in paper to give optimal performanace

[rows, cols] = size(im);

% generating kernel (size 5x5)
w = [1/4-a/2, 1/4, a, 1/4, 1/4-a/2]';

W = w*w';

% low pass filter image
LPFIm = conv2(im, W, 'same');
```

```
%newIm = newIm(2:2:rows, 2:2:cols);

% subsample image by 2.
newIm = interp2( [1:cols]', 1:rows, LPFIm, [1.5:2:cols]', 1.5:2:rows);
```

```
function regionMap = segmentRegions(speed, speedThresh, minRegionSize)

[L N] = bwlabel(speed>speedThresh);

[rows cols] = size(L);

for n=1:N
   L = L(:);
   regionSize = sum(L==n);
   if (regionSize<minRegionSize)
      idx = find(L==n);
      L(idx) = 0;
   end
   L = reshape(L, rows, cols);
end

[L N] = bwlabel(L);

regionMap = L;
```

```
function newimage = subsample(image, border, samprate)
% SUBSAMPLE Subsamples an image
%   NEWIMAGE = SUBSAMPLE(IMAGE, M, N) removes a border of M pixels then
%   takes every N'th row then every N'th column.
[rows, cols] = size(image);
newimage = image(1+border:samprate:rows-border, 1+border:samprate:cols-border);
```