Camera Surveillance of Crowded Traffic Scenes

Benjamin Maurin
Osama Masoud
Nikos Papanikolopoulos[*]

Department of Computer Science and Engineering
University of Minnesota
4-192 EE/CS Bldg., 200 Union Street SE
Minneapolis, MN 55455-1059

Abstract

This paper deals with real-time image processing of crowded outdoor scenes with the objective of creating an effective traffic management system that monitors urban settings (urban intersections, streets after athletic events, etc.). The proposed system can detect, track, and monitor both pedestrians (crowds) and vehicles. We describe the characteristics of the tracker that is based on a new detection method. Initially, we produce a motion estimation map. This map is then segmented and analyzed in order to remove inherent noise and focus on particular regions. Moreover, tracking of these regions is obtained in two steps: fusion and measurement of the current position and velocity, and then estimation of the next position based on a simple model. The instability of tracking is addressed by a multiple-level approach to the problem. The computed data is then analyzed to produce motion statistics. Experimental results from various sites in the Twin Cities area are presented. The final step is to provide this information to an urban traffic management center that monitors crowds and vehicles in the streets.

1 Introduction

Monitoring crowded urban scenes is a difficult problem. Despite significant advances in traffic sensors, modern monitoring systems cannot handle effectively busy intersections. The reason is that there are too many moving objects (vehicles, and pedestrians). Tracking humans in outdoor scenes is also very complex. The number of pedestrians and vehicles in a scene varies and is usually unpredictable. Recently, there was progress on techniques to track an arbitrary number of pedestrians simultaneously. However, as the number of pedestrians increases, system performance degrades due to the limited processing power. When the number of pedestrians exceeds a threshold value, the processing rate drops dramatically. This makes it hard to keep accurate track of every pedestrian and vehicle because of the large displacement among pedestrian and vehicle locations in the processed frames.

We propose a vision-based system that can detect, track, and monitor busy urban scenes. Our system is not only limited to vehicles but can deal with pedestrians and crowds. Our approach is robust to a variety of weather conditions and is based on an efficient scheme

[*] Author to whom all correspondence should be addressed.

to compute and approximation of optical flow. The optical flow helps us classify the different objects in the scene while certain statistics for each object are continuously updated as more images become available.

Addressing busy intersections can have a significant impact on several traffic operations in urban areas and on some more general applications. One traffic operation which can be affected is the automated walk signal request. Currently, a pedestrian is required to press a button to request a walk signal at a crosswalk. It may be desirable to automate this process especially when there is a large number of pedestrians waiting to cross the street. Crowd detection can be used in this case. Another application is the study of flow patterns at certain intersections. It may be desirable as a city planning consideration to study the use of crosswalks at a certain intersection. Flow data of pedestrians crossing the street throughout the day can be collected and used to make decisions such as building a pedestrian bridge, etc. Finally, our proposed system may be used to monitor crowds outside schools, nursing homes, train tracks, and at athletic events. One interesting application is to compute crowd density (with obvious public safety applications). In general, monitoring areas where accidents occur often can result into early warning signals and can reduce deadly consequences.

The paper starts with a review of previous work, continues with the detection and tracking schemes and concludes with experimental results.

1.1 Previous work

Many methods have been proposed for the detection and tracking of moving objects; many of those have focused on the detection and tracking of traffic objects. The applications of this line of research are multiple: tracking of targets, vision-based robot manipulators, presence detectors, analyzers of visual data, etc.

The first generation trackers made several assumptions regarding the nature of the objects in the scene. These assumptions resulted in acceptable tracking performance but on the other hand, they were not robust to changes in the scene (e.g., movement of the camera or small changes in the shape of the object). Baumberg and Hogg in [1] introduced one of first trackers. It was based on active contours to track the silhouette of a person. Others, such as Rossi and Bozzoli in [2], addressed the problem by locating the camera above the scene, which avoids the creation of occlusions. But this kind of practice is not easy to implement especially in traffic applications. The systems described require extensive models of the objects to be tracked. They also require some knowledge of their dynamic characteristics. We are going to use only a few assumptions on the models in order not to restrict our tracker to a single class of detection. For example, no limits will be assumed for the velocities of the traffic objects, their sizes, their acceleration, or their shape.

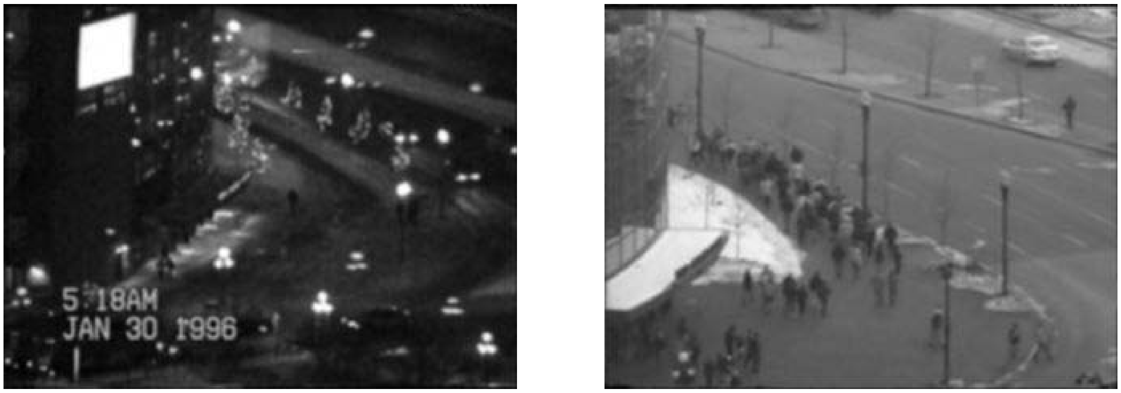
Systems based only on a background subtraction give good results but fail to detect and track traffic entities when they get close to each other. One example of such a method can be found in [3]. Our system extends that approach by utilizing motion cues.

Cutler and Tuck [4] developed a recognition and classification module for motion. It aimed to interpret gestures. Their work is based on the analysis of moving areas in a video sequence. This makes it more of a motion classifier than a tracker. Our detection method is close to [4] and to Dockstader's work [5], which addressed robust motion estimation.

2 Processing of the Image Sequences

2.1 Detection

We have developed a new method to detect blobs. Because outdoor environments cannot be modeled accurately, it is difficult to separate the traffic objects from the background. Moreover, the great differences between day and night (see Figure 1) explain why a robust detector needs to be developed. In the case of groups of people, it is difficult to decompose a group into individuals. Instead, one would attempt to track the group as one entity. A possible way to do this is to calculate the movement of each point and then group together points with similar motion characteristics.



(a) Night video (b) Day video

Figure 1. Traffic Scenes from the Xcel Energy Center in St Paul.

2.1.1 Nature of the video sequences and preprocessing

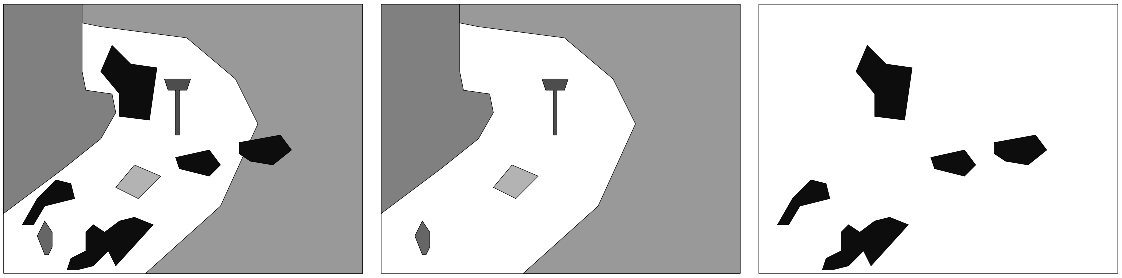
Images from the outdoor video sequences that we used are grayscale. We do not make any assumptions on the nature of the scene (night, day, cloudy, rainy, winter, summer, etc.). Elements of the scene are not known a priori and the sizes of the traffic objects (vehicles and pedestrians) are also unknown. An assumption that we make is that a crowd is a slow deformable object with a principal direction of motion.

This assumption is not restrictive for our objectives, but it can be restrictive in the case of vehicles. A traffic tracker must be able to track objects that vary from a simple pedestrian to a complex crowd.

Outdoor images can have very bad quality, especially at night. In the preprocessing phase, it is necessary to remove noise. We achieve this goal by convolving the images with a Gaussian filter. We can also apply a mask to the image to remove the undesired regions such as roads, flashing billboards, or reflective building walls.

2.1.2 Background removal

In [6], Haritaoglu *et al.* modeled the background by following a statistical approach. The detection of people is achieved by subtracting this background from the new images (see Figure 2). The parameters of the background are re-evaluated with every new image. The model uses a Gaussian distribution for each pixel and a comparison-evaluation for every frame. The classification of foreground/background is done at this time by combining all the previous results. This approach creates a very good separation between the two possible classes (foreground, background). However, no other information can be extracted from this technique. In particular, during the motion of two crowds past each other, one cannot estimate the speed or direction. This method is useful in the case of easily distinguishable objects with slow motion. In our case, a pixel by pixel scan would be relatively slow.



(a) First image (b) Background model (c) Detected blobs

Figure 2. Background creation.

2.1.3 Motion extraction

Another idea for the detection part of our approach is to consider the optical flow for a precise extraction of the motion in the scene. It is a very difficult problem to solve in real-time, however. This paper tries to address some of the computational issues related to optical flow estimation. Like Sun [7], Dockstader *et al.* [5], and other researchers, we will concentrate on the speed requirements of the algorithm, more than on the accuracy of the estimation of the optical flow. Using optical flow estimation, our method is able to distinguish objects occluding each other if part of them is still visible and has different motion characteristics (see Figure 3).

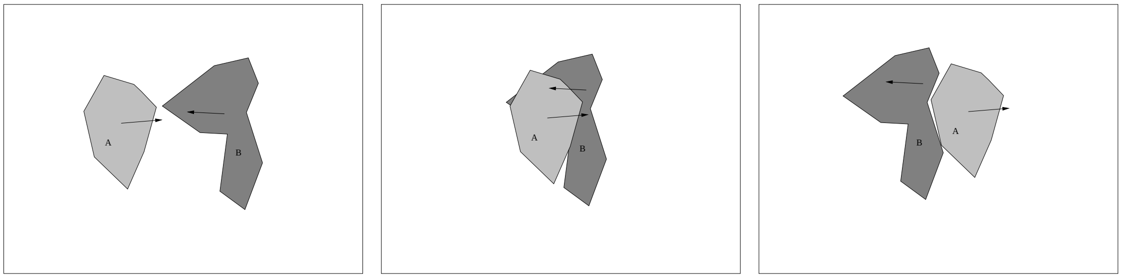


Figure 3. Two detection zones going in opposite directions.

2.2 Optical flow

In [8], Galvin *et al.* give a comparison of estimation algorithms for optical flow. Of all the algorithms presented, only two will be considered because of their simplicity and speed. The others, such as fast separation in pyramids, the robust estimation of Black and Anandan [9], or the wavelet decomposition for fast and accurate estimation by Bernard [10], are too complex to implement in real-time.

2.2.1 Definition

The optical flow represents the 2D projection of the 3D velocity map of the scene on the camera plane (see Figure 4). The motion flow (the 3D velocity vectors map) is the true motion of each object. Because we cannot measure it, we just compute the projection of it on a plane. As a result, all the movements along the z axis which are not perceived by the sensor and will not be computed.

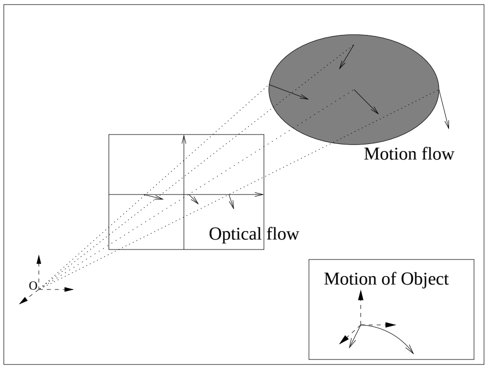


Figure 4. Optical flow and motion flow.

2.2.2 Mathematical approach

In optical flow estimation, it is generally assumed that the intensity I is constant over a displacement velocity (u,v) of an object between two frames and time interval ∂t:

$$I(x,y,t) = I(x+u\cdot\partial t, y+v\cdot\partial t, t+\partial t) \qquad (1)$$

A Taylor expansion of the right hand side of **(1)** gives:

$$I(x,y,t) + \left(\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t}\right)\cdot\partial t + O(\partial t) \qquad (2)$$

Setting **(1)**=**(2)**, we get the following equation:

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0 \qquad (3)$$

In reality, this equation is not directly applicable due to the lack of constraints on u and v.

2.2.3 Differential method

This method is explained in Barron *et al.* [11]. It was also used by Rwekamp and Peter [12] with the goal of using ASIC processors for the real-time detection of movement. The

idea is to use a minimization procedure where a spatial estimate of the derivative by a Laplacian is employed. One may use a spatial iterative filter to obtain a more precise estimation.
We have experimented with this method using a floating point implementation as well as an integer implementation (for efficiency concerns). Based on our experiments, we concluded that although this method works well with synthetic images, it is sensitive to noise which is present in our video sequences. In addition, the computational cost is prohibitive (less than 1 fps in the floating point implementation). For these reasons, we decided to explore other methods.

2.2.4 Correlation method

Principle

We look for the velocity vector of each pixel by analyzing its motion between two successive images: I_i and I_{i-1}. To know the position of the pixel in the next image I_i, we calculate a similarity value on all the possible destinations. This value can be obtained using standard sum of square differences (SSD) or sum of absolute differences (SAD). In all the cases, we have to go through a Disc (see Figure 5) in I_i around the original localization of the pixel in I_{i-1} and to compute the function for all the positions of the Disc. Then, the minimal value is sought, and thus we obtain the best position in the Disc of I_i. This position corresponds to the movement of the pixel of I_i and thus the optical flow is derived. The Disc can be also called a ‘pattern’ (which is not necessarily circular).

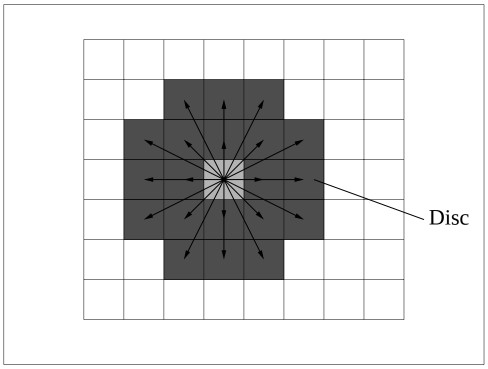


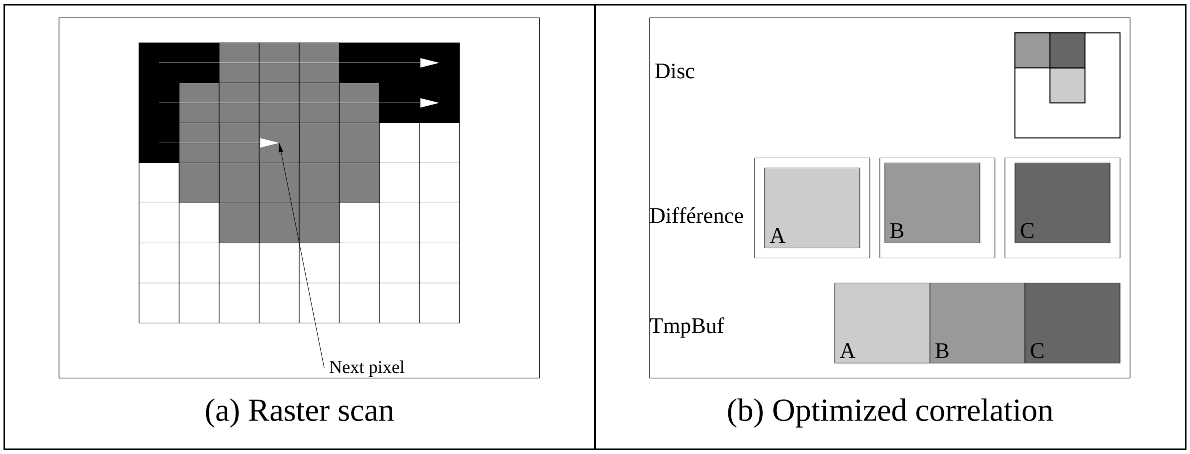
Figure 5. Correlation approach.

Simple approach

If we use SAD as the cost function, then for each point $p = (x, y)$ in the image, we calculate:

$$\forall p \in I_{i-1}, \quad \text{SAD}(p) = \min_{(u,v)\in Disc}\left(\sum_{(a,b)\in \text{Neighborhood}(u,v)} \left| I_i(x+a, y+b) - I_{i-1}(x,y) \right| \right) \qquad (4)$$

This value must give the minimum over the Disc as the sum on the neighborhood. The computation is done in a serial way for each pixel. We just do a simple raster scan (see Figure 6(a)). At the end, we have the motion map for the pixels.



(a) Raster scan

(b) Optimized correlation

Figure 6. Two ways to see the correlation.

Optimized approach

We utilized our hardware (Matrox Genesis vision board) in implementing a parallel version of this approach. The optimized approach has the following phases:

Phase 1: Parallel calculation of the cost function

Here, we calculate all the differences for a position of the Disc. All the simple arithmetic operations can be done on sub-windows (child buffers). In our case, we define as many sub-windows as the number of positions in the Disc. In other words, if the Disc contains N points, we have N sub-windows gathered in a single TmpBuf buffer array and indexed by their positions. Each one will contain the result of a subtraction between image I_i and the image I_{i-1}, but with a spatial shift corresponding to the position of the point in the Disc (see Figure 6(b)).

To do this, we use a sliding window in image I_i and a fixed window at the center of image I_{i-1}. We compute the difference between these windows, which have the same size, and then we convolve the difference image with a kernel of ones. Thus, we have the sum and therefore the value of the cost function for this position in the Disc. The result is saved in TmpBuf. Then, we move the window of image I_i to reach another position of the Disc and so on. The kernel can be of size 3x3, 5x5, …, 11x11. The larger it is, the less the noise will be. But that also means less sensitivity. A good compromise is a size of 5x5 or 7x7.

Phase 2: Retrieval of indices of minimal cost

At the end of the algorithm, we seek the minimum value of the cost function for each pixel moved. The imaging hardware can calculate the minimum M between two images

I_k and I_l by the formula:

$$M(i,j) = \min(I_k(i,j), I_l(i,j)) \tag{5}$$

We augment the cost values with indices (locations in Disc) to retrieve the flow.
A last optimization, or rather approximation, is to define a Disc where only the principal axes are used. We keep for example the horizontal and vertical axes and the diagonals. This approximation is not restrictive (see Cutler [4]). Also, the radius of Disc itself can be made small. The cameras used are usually rather far away from the scene and motion does not exceed 2 to 3 pixels per frame. Therefore, a typical Disc is defined as follows:

1		1		1
	1	1	1	
1	1	1	1	1
	1	1	1	
1		1		1

Noise removal

To remove the noise from the images during the correlation, thresholding is used. It is known (see Cutler [4]) that a video CCD camera has a Gaussian white noise of amplitude σK with $\sigma = 2.5$ and $K = 5$ to 10 according to the model of the camera. In our difference calculation, the conclusion is that an absolute difference less than 6σ is considered as noise.
As the noise is white, the calculation of the cost function at the central position gives the value of the noise at each pixel. Moreover, after convolution, this noise is averaged. A thresholding with $S = \sigma \cdot K \cdot Kernel^2$ gives a mask corresponding to the level of noise. Thus, the points that give a lower value than this level are not to be taken into our optical flow estimate. For example with a kernel 5x5, we obtain $S = 2.5 \cdot 6 \cdot 5^2 = 375$.

Results

The results are very good for the estimation part. The kernel that was used for the summation is 7x7. On 320x240 images with a Disc of ± 3 pixels, the processing rate is approximately 4 fps. On 160x120 images and by using the same Disc, we reached computational speeds of 10 fps.

Conclusion for our application

This method is very adaptable to our problem. The limitation of the size of the Disc does not prevent it from having a good separation between moving parts and noise. It is a robust and fast method: we reach the 15 fps with a Disc radius of ± 2 pixels.

2.3 Background

To detect the static traffic objects or people, we need another way to approach the problem. For these cases, we use a detection scheme that subtracts the image from the background. The idea is based on an adaptive background update. The update equations are as follows:

$$A = \mathrm{AND}\left(M_{\mathrm{bkgnd}}, I_{\mathrm{opt.\,mask}}\right) \qquad (6)$$
$$B = \mathrm{AND}\left(\frac{I_{\mathrm{current\ frame}} + 15 \cdot M_{\mathrm{bkgnd}}}{16}, \mathrm{NOT}(I_{\mathrm{opt.\,mask}})\right)$$
$$M_{\mathrm{bkgnd}} = \mathrm{OR}(A, B)$$

where M_{bkgnd} is the background, $I_{\mathrm{current\ frame}}$ is the current image, and $I_{\mathrm{opt.\,mask}}$ is the mask obtained after binarization of the optical flow. Every 16 iterations, the background is completely readjusted.

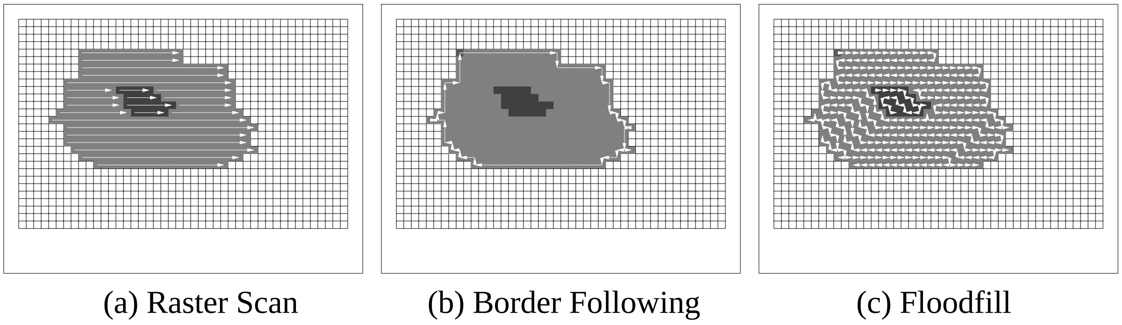
The detection is carried out at every new estimation of the optical flow by a simple subtraction of the background and thresholding with $4\sigma K$ (a value much higher than the one for the flow). Then, we merge the results obtained by these two methods by assigning the background a special value (meaning that the blob comes from the second type of detection and not from the optical flow). The estimation of the optical flow is dominating on the background (in the case of double detection). That processing leads to a unified color map mixing the two methods and having a lookup table describing the motion and the nature of each point. The points belonging to the background are associated to no motion and no position in the Disc.

2.4 Segmentation

The segmentation is the next step. In our case, the choice of an effective method is not obvious. The goal is to use some information from this stage during the tracking phase.

2.4.1 Clustering the motion map from the optical flow

For this simple stage, we have various choices: a raster scan, a border following or a floodfill algorithm. Figure 7 shows the three methods.



(a) Raster Scan (b) Border Following (c) Floodfill

Figure 7. Classic methods for segmentation.

Raster scan

This method is very effective for zone segmentation and detection in the case of large shapes that may have holes. The algorithm consists of dividing the image into runs and memorizing the similar parts in a run. It is very fast and the computational time is only dependent on the size of the image, not its content. On the other hand, it requires two passes to obtain the segmentation and it must maintain a chain-list of runs. Therefore, it has a high memory requirement that need to be appropriately managed to guarantee sufficient speed. Statistics relating to the zones (e.g., size, area, the minimal bounding rectangle, the mean velocity estimated by the flow) can be calculated on the fly and do not require additional stages. We just have to go through the chain-list of runs to gather the data.

Border following

In this case, we follow only the contours of objects. This approach dramatically reduces the time used to go through the entire area, but does not allow the detection of internal holes. If we want to gather pixels having a different value inside an region, we must enter the zone and the algorithm looses its efficiency. Furthermore, border following does not make it possible to calculate all the desirable statistics.

Floodfill

In computer graphics, this simple technique allows filling regions with a color. The basic idea is to recursively check the four different directions at every pixel (starting from a seed pixel). Each time a pixel is visited, it is marked so that it does not get visited again. Statistics are computed on the fly due to the fact that we go through all the pixels of the shape.

Conclusion on the choice of the algorithm

Comparisons are summarized in Table 1. The floodfill method offers the most advantages, is unquestionably slower than the border following and a little slower than the raster scan, but allows to segment shapes having internal variations.

Algorithm	Speed	Statistics	Comparison	Memory	Segmentation
Raster Scan	Medium	Yes	Lines	Chain-list	Yes, 2 passes
Border follow	Fast	Incomplete	Borders	Chain-list	Holes not visited
Floodfill	Slow	Yes	Local,4 dirs	Dbl. Buf	Yes, 1 pass

Table 1. Comparison of classical algorithms for segmenting an image.

2.4.2 Two features computed by the segmentation

Rotation

When an object is in slow rotational motion, the pixels describing the object move according to a local gradient. Each point has a direction slightly different from its neighbor. Therefore, if the rotation is not fast, the floodfill algorithm help us gather all these close pixels. In this method, we adjust a validation parameter: the maximum angle between two directions. To calculate the difference between two angles, the following formula is used:

$$\text{difference} = \begin{cases} 360 - \left|\text{Angle}_1 - \text{Angle}_2\right| & if \left|\text{Angle}_1 - \text{Angle}_2\right| > 180 \\ \left|\text{Angle}_1 - \text{Angle}_2\right| & otherwise \end{cases} \qquad (7)$$

This approach also helps maintain all the pixels which are static.

Shape fitting

After the segmentation, the blobs are shaped in simple geometrical shapes. At this stage, a minimum bounding box can be calculated. The floodfill algorithm makes the procedure easier since it provides all the points contained in the shape. At the end of the segmentation, the blobs are described by the minimum bounding box, the position of their centers, and the number of pixels they contain (or mass). A better adaptation to a more complex shape can be carried out at this stage (oriented bounding box) and will be explained later.

3 Tracking

Tracking is an essential component of the system. A diagram of the system is shown in Figure 8.

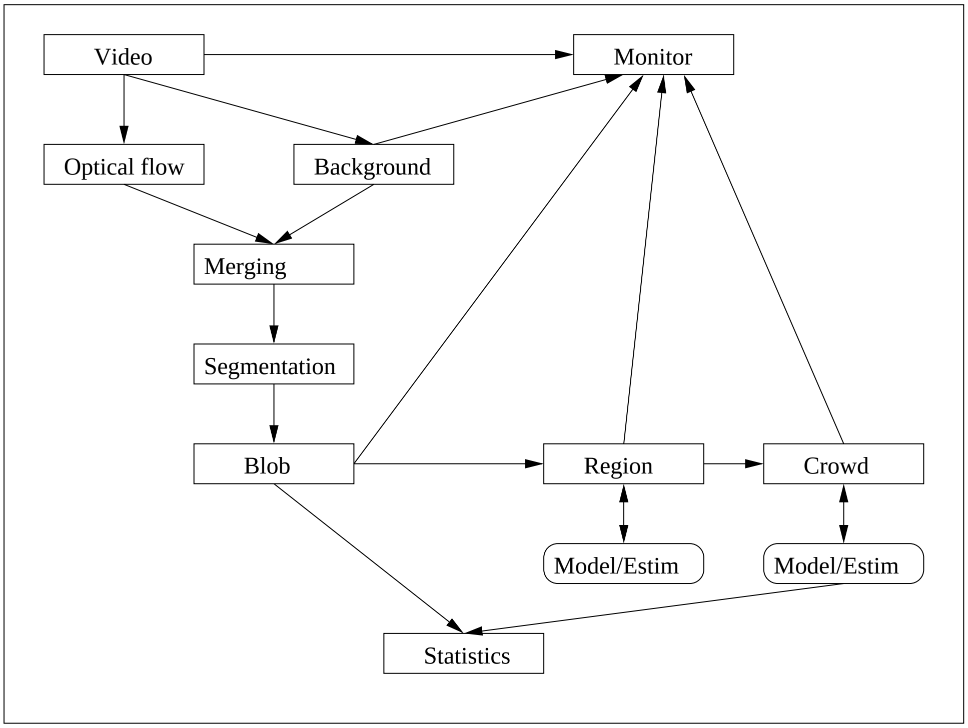


Figure 8. Tracking system.

As explained above, optical flow is estimated by the correlation method and the background is used to find motionless foreground objects. The fusion of both is then segmented into blob entities.

Blobs are used for the layers model. They serve as measurements for higher level entities. A confidence value is associated to each of the possible links between two entities. If blobs are strongly connected more than three times in a row, we create a region (higher layer). The confidence value for two entities depends on the overlap surface, the distance, and speed similarity.

3.1 Layer model

3.1.1 First level: blobs

We model a blob using its position, size, area, and velocity. The first three parameters are computed by the floodfill algorithm during segmentation. If we add a pixel, $P(x,y)$, to the blob, we have:

Size

Let $Size$ be an accumulator. So each time we add a pixel, we do: $Size_n = Size_{n-1} + 1$

Position

Let X_c, Y_c be accumulators. We define their sum by:

$$\begin{cases} X_{cn} = X_{cn-1} + x \\ Y_{cn} = Y_{cn-1} + y \end{cases} \quad (8)$$

Then at the end of the floodfill, we compute the centroid using the first moment:

$$\begin{cases} \sigma X_c = \dfrac{X_{cn}}{Size} \\ \sigma Y_c = \dfrac{Y_{cn}}{Size} \end{cases} \quad (9)$$

Velocity

Let V_x, V_y be accumulators. Since pixel velocity is stored as an index into Disc, we get its displacement D_x, D_y by looking up the pattern:

$$\begin{cases} D_x = \text{DisplacementX}[P(x,y)] \\ D_y = \text{DisplacementY}[P(x,y)] \end{cases} \quad \text{And then} \quad \begin{cases} V_{xn} = V_{xn-1} + D_x \\ V_{yn} = V_{yn-1} + D_y \end{cases} \quad (10)$$

Then, we have the global velocity of the blob:

$$\begin{cases} \sigma V_x = \dfrac{V_{xn}}{Size} \\ \sigma V_y = \dfrac{V_{yn}}{Size} \end{cases} \quad (11)$$

Area (minimum bounding box)

Let MinX,MaxX,MinY and MinY be values of the sides of the box. We have:
$\text{MinX} = \min(\text{MinX}, \text{x})$, $\text{MaxX} = \max(\text{MaxX}, \text{x})$, and similarly for MinY and MaxY.
And so Area=(MaxX-MinX) *(MaxY-MinY).

These blobs describe all zones in the optical flow. Their modeling is easy and fast. The floodfill algorithm is needed here if we want to have that kind of description.

3.1.2 Second level: regions

The intrinsic parameters are the same as for the blobs. The differences are in the way we compute their values. The regions inherit values from their parent blobs at their creation but they are adapted during the tracking. Other properties, like the time of existence or inactivity, are also used for regions.

3.2 Geometrical description of the entities

The tracking part has been improved by the introduction of new blob descriptions. Relations based on simple bounding boxes are clearly not effective for very mobile and close objects (such as distant pedestrians). Moreover, the density ($Mass/Area$) of a crowd is badly represented by this approach. Therefore, we compute oriented rectangles instead of horizontally aligned boxes. We want to preserve simplicity and efficient computation of these shapes.

3.2.1 Oriented boxes

Covariance Matrix

For a 2-D space of n points, we have a covariance matrix M between the two variables X and Y. It is defined by:

$$M = \begin{pmatrix} <X^2> & <X,Y> \\ <X,Y> & <Y^2> \end{pmatrix} \qquad (12)$$

The marks $<>$ mean that we take the variance value.

Eigenspace representation

We use a technique often associated with the principal components analysis: the diagonalization reduction. This representation results in finding maximum dispersions by analyzing eigenvalues and eigenvectors. Given M, its decomposition is written as: $M = \Delta^t D \Delta$, with Δ being the transition matrix, and D the diagonal matrix with eigenvalues. Let $\Delta = [v_1, v_2]$ and $D = \begin{pmatrix} e_1 & 0 \\ 0 & e_2 \end{pmatrix}$. We can choose to have $e_1 > e_2$, so that v_1 gives the direction of elongation and $\sqrt{e_1}$ gives the total elongation. The second axis is described by v_2 and $\sqrt{e_2}$.

For example, Figure 9 describes a two dimensional distribution. Using this scheme, the two vectors and the resulting rectangle are shown. In our algorithm, we include a multiplicative coefficient equal to 1.5 for the two sides. The elongation obtained after diagonalization is often a box smaller than the real area. For a better fit, we increase this box to include a bigger part of the distribution.

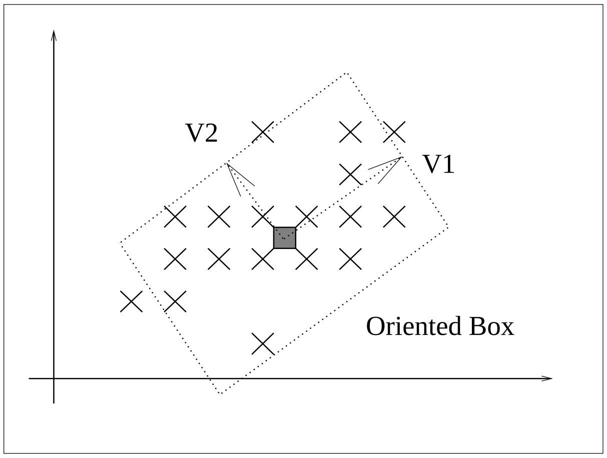


Figure 9. Oriented box on a distribution.

Overlap

The overlap is a quantity between 0 and 1 that represents the degree of intersection between two rectangles. It is equal to $\frac{\text{Overlap Area}}{\text{Maximum Area}}$. Overlap Area is the area of region of intersection between the two rectangles. It is computed by polygon clipping (see shaded areas in Figure 10). Maximum Area is the larger area of the two rectangles.

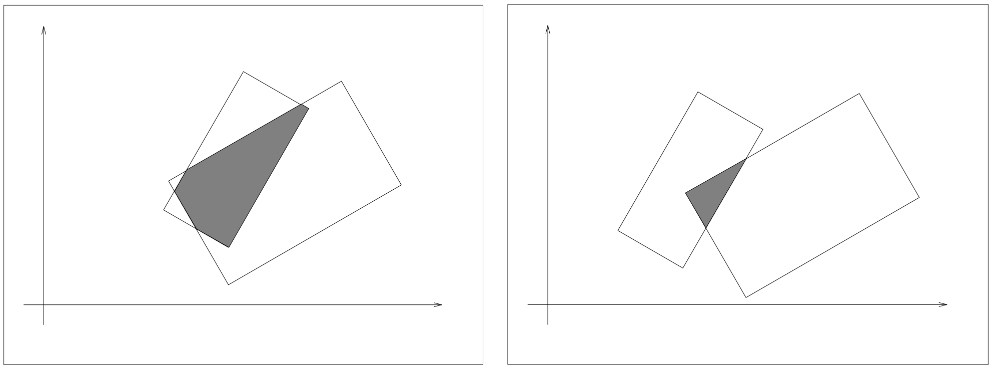


Figure 10. Overlap Area.

3.3 Details of relations

All the relations are based on confidence values (α, β) which reflect the similarity between two entities A and B. We introduce thresholds to define the presence or absence of a relation between two entities. In our tracker, the value α will be related to the position and the area of the objects while the value β will be related to the similarity in motion direction deduced from optical flow. Our merging algorithm is based on a score calculation between entities.

3.3.1 Relations based on scores

We propose to create a tracking method based on scores of each blob B_i with regard to a region A. Then, we keep only a percentage of the best related blobs that we use in a Kalman filter as measurement. The introduced score will be used as confidence value for the filtering.

Score dependent on position

During this phase, we have the positions and descriptions for blobs and regions. First, we compute the distance with a fuzzy model using the distance between the two centroids and the perimeter of each rectangle. We introduce a confidence term by normalizing the distance by a multiple of the perimeter (a factor of 2 is used in our implementation). Therefore, we deduce the fuzzy value Distance:

$$\text{Distance} = \max\left(0,1 - \frac{\|AB_i\|}{2 \cdot \text{Perimeter}(A)}\right) \quad (13)$$

A cost function is then defined on the position:

$$\alpha(A, B_i) = \frac{\text{Overlap}(A, B_i) \cdot 7 + \text{Distance} \cdot 3}{10} \quad (14)$$

In the case of blob-to-blob relation, the coefficients are (5,5) instead of (7,3). This is explained by the fact that a relationship between a region and a blob is supposed to be based more on the overlap than the position.

Score dependent on angle

We start by calculating the angle difference. We also use a fuzzy model as before. This time, we use a threshold to accept the minimal value of angle difference:

$$\beta(A, B_i) = \begin{cases} 1 & \text{if Difference} < \text{Thresh} \\ \max(0, \dfrac{180 - 2 \cdot (\text{Difference} - \text{Thresh})}{180} & \text{otherwise} \end{cases} \quad (15)$$

3.3.2 Heuristic combination

Once these two coefficients are calculated, we look for the best heuristic combination between the various objects. For that, we combine the results of all the blobs with themselves and with the regions.

First, we test if the α value found for a relation is higher than a threshold (to guarantee a minimal connection between objects). Then once this stage is validated, we add the value β to α. This last value provides the final cost function for the relations. All the entities (regions or blobs) are tested and the relationships are calculated. Then, we keep the group of the best relationships or the best relationships.

Regions creation based on blob-to-blob relationships

During each loop, we preserve the previous list of blobs to check which of the new blobs are related to the preceding ones. These relationships are one-to-one. A counter on each previous entity specifies if a link has already been created before. The counter is propagated from one entity to another and thus gives the total number of blobs related in a row. If this value exceeds a threshold (2 in our case), then we create a new region inheriting the parameters of the last linked blob. The minimal threshold is taken such that we have at least $\alpha > 50\%$ in each relation.

Merging many blobs

The relationships between blobs and regions are many-to-many. In other words, several blobs can be used as measurements for a region. A threshold, $\alpha > 10\%$, ensures a minimal relationship. Then, we create a list of the relationships ranging between 70% of the maximum value and the maximum value itself. This group of blobs will be used to estimate the location measurement of the region at the next loop. For this, we compute the centroid, the minimal bounding box, and the mean velocity of the group. Then, we use this information in the regions measurement/estimation part (filtering).

3.4 State space representation of regions

3.4.1 State vector and data splitting

The positions x and y are related to the region centroid. The other parameters are the covariances and the mass (number of pixels).

We can describe all regions by a state representation: let $\mathbf{X}$ be the state vector that describes a region:

$$\mathbf{X} = \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \\ \mathrm{cov} < x^2 > \\ \mathrm{cov} < y^2 > \\ \mathrm{cov} < x, y > \\ \mathrm{mass} \end{bmatrix} \qquad (16)$$

So our system has the following state equation:

$$\mathbf{X}' = \begin{bmatrix} 1 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \mathbf{X} + \mathbf{V} \qquad (17)$$

The parameters are only estimated and do not have real physical significance. The measurement equation is:

$$\mathbf{Y} = \mathbf{C} \cdot \mathbf{X} + \mathbf{W} \qquad (18)$$

with:

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad (19)$$

It is clear that the system can be split into smaller systems: one for coordinates and speeds, and four for each feature of the shape.

3.4.2 Position characteristics:

We use a constant velocity model for the coordinates,

$$\mathbf{X} = \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix} \qquad (20)$$

and,

$$\mathbf{X}' = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \mathbf{X} + \mathbf{V}_1 \qquad (21)$$

$\mathbf{V}_1$ is a white Gaussian noise that depends on the system. Its covariance has a matrix that has an estimate for limit. In our case:

$$\varepsilon\{\mathbf{V}_1, \mathbf{V}_1^T\} \to \mathbf{V}_{\text{position}} \cdot \mathbf{V}_{\text{position}}^T \qquad (22)$$

$$\mathbf{V}_{\text{position}} = \begin{bmatrix} \sigma_x \\ \sigma_{\dot{x}} \\ \sigma_y \\ \sigma_{\dot{y}} \end{bmatrix}$$

And $\sigma_x = \sigma_y$ shows the system's standard-deviation in position while $\sigma_{\dot{x}} = \sigma_{\dot{y}}$ shows the one in velocity.

3.4.3 Shape features

In the same way, we obtain:

$$X = (\text{param}) \qquad (23)$$
$$X' = X + V_2$$

V_2 is a white Gaussian noise that depends on the feature. Its covariance has an estimate for limit:

$$\varepsilon\{V_2\} \to V_{\text{param}} \qquad (24)$$
$$V_{\text{param}} = \sigma_{\text{param}}$$

3.5 Kalman filtering

The Kalman filter makes it possible to merge measurements assuming that we know their reliability. Furthermore, It allows to optimally estimate the next state of the system. The process to be estimated must be controlled by a differential equation, linear and stochastic. We will not extend on the subject since our study requires only a first order filtering. We have the following equations:

3.5.1 Position filtering

Measurement equations:

Kalman gain:

$$\mathbf{K}_n = \hat{\mathbf{P}}_n \cdot \mathbf{C}^T \cdot \left(\mathbf{C} \cdot \hat{\mathbf{P}}_n \cdot \mathbf{C}^T + \mathbf{R}_n\right)^{-1} \qquad (25)$$

State correction:

$$\mathbf{X}_n = \hat{\mathbf{X}}_n + \mathbf{K}_n \cdot \left(\mathbf{Y}_n - \mathbf{C} \cdot \hat{\mathbf{X}}_n\right) \qquad (26)$$

Variance:

$$\mathbf{P}_n = (\mathbf{I} - \mathbf{K}_n) \cdot \hat{\mathbf{P}}_n \qquad (27)$$

Temporal equations:

System:

$$\hat{\mathbf{X}}_{n+1} = \mathbf{A} \cdot \mathbf{X}_n \qquad (28)$$

Variance propagation:

$$\hat{\mathbf{P}}_{n+1} = \mathbf{A} \cdot \mathbf{P}_n \cdot \mathbf{A}^T + \mathbf{Q}_n \qquad (29)$$

3.5.2 Shape filtering

In the case of scalar values instead of matrices, the measurement equations do not change, except for the fact that we have real values, not vectors. In addition, A = 1. We can use the following equations:

Kalman Gain:

$$K_n = \frac{\hat{P}_n}{\hat{P}_n + R_n} \qquad (30)$$

State correction:

$$X_n = \hat{X}_n + K_n \cdot \left(Y_n - \hat{X}_n\right) \qquad (31)$$

Variance:

$$P_n = (1 - K_n) \cdot \hat{P}_n \qquad (32)$$

System:

$$\hat{X}_{n+1} = X_n \qquad (33)$$

Variance propagation:

$$\hat{P}_{n+1} = P_n + Q_n \qquad (34)$$

3.5.3 Parameter values

Now, we have to define the various parameters of the filtering that the matrices $\mathbf{Q}_n$ and $\mathbf{R}_n$, and the scalars Q_n and R_n represent. The terms depend on the kind of measurement or on the system (measurements variances or internal noise). The internal noise of the system will manage the changes of motion and shapes.

System noise parameters

The positions lie between 0 and 160 pixels for X and 0 and 120 for Y. We can consider that the two axes are independent and contain the same noise ratio. Because the intervals of values are close, we use a block diagonal matrix $\mathbf{Q}_n$. The speeds lie between 0 and some pixels, moreover acceleration will be included as noise in speed. Based on experiments, we take $\sigma_{x,y} = 0.01$, and $\sigma_{\dot{x},\dot{y}} = 0.0001$. The following assignment is used:

$$\mathbf{Q}=\begin{bmatrix}\mathbf{A} & 0\\ 0 & \mathbf{A}\end{bmatrix}, \text{and} \quad \mathbf{A}=\begin{bmatrix}\sigma_{x,y}^2 & \sigma_{x,y}\sigma_{\dot{x},\dot{y}}\\ \sigma_{x,y}\sigma_{\dot{x},\dot{y}} & \sigma_{\dot{x},\dot{y}}^2\end{bmatrix} \qquad (35)$$

The mass is regarded as a constant varying from 0 to a few thousands. It must keep a constant value and for this reason, we try to assign it very small internal noise (for example a standard-deviation of 0.01). Therefore, we take $Q=0.01^2$.

The covariance values of matrix M (see Section 3.2.1) are between 0 and a few units. The internal noise must be small to handle slow changes in shapes and to avoid continuous changes (for example a standard-deviation of 0.01). We choose $Q=0.01^2$.

Measurement noise parameters

The $\mathbf{R}_n$ matrix depends on the confidence value $\alpha+\frac{\beta}{5}$, computed previously. We use an empirical formula (deduced from experiments) to find:

$$\Sigma=\frac{1}{\alpha+\frac{\beta}{5}}-1 \qquad (36)$$

Then, for the positions, we say that X and Y measurements are not correlated and have the same noise. Therefore, $\mathbf{R}_n=\begin{bmatrix}\Sigma^2 & 0\\ 0 & \Sigma^2\end{bmatrix}$.

For the state parameters, we found that $R_n=\Sigma$ gives good results.

3.5.4 Initialization

We initialize the filtering at the current position of the region and a zero speed, because the value obtained by the optical flow is not accurate enough at this stage. The matrix $\mathbf{P}_0$ is reset to zero. The intrinsic characteristics are initialized with the current region values and P_0 is set to zero.

4 Experimental Results

We tried our algorithms in several outdoor scenes from the Twin Cities area. One area where one may find crowds is around the Xcel Energy Center in St Paul. We filmed scenes during day and night. We also applied our method on a video during winter at a road intersection to check the quality of the tracking when vehicles drive past each other in opposite directions. The optical flow and the first layer (blobs) give the required statistics on crowds: direction, size, and mean velocity.

4.1 General comments

The speed of our software depends on the quality of the detection done by the optical flow approach. In the general case, a Disc with a radius of 1 pixel is enough to detect objects on images with size of 160x120 pixels. All the results are obtained with a 5x5 Kernel, which allows a very good detection without significant corruption by noise. Then, our tracker runs at 15 fps with all the characteristics displayed on 4 different screen buffers on the same monitor: optical flow map, background, tracked regions, and most used zone. If we increase the detection with a Disc of radius 2, the frame rate drops to 10 fps. Here, we should mention the following observation for the quality of the tracking. One thing worth mentioning is that in case some frames have to be dropped, it is important to drop the same number of frames consistently. For example, the pattern would be one frame processed followed by two dropped and so on. Otherwise we get temporal instability. The reason is that optical flow estimation assumes that the time interval between two frames is always the same. For this reason, our tracker runs at speeds of either 15 fps or 10 fps, or $\frac{30}{n+1}$ fps (n being the number of skipped frames).

4.2 Statistics

The first result we present shows statistics gathered over a period of time $T = 2$ minutes. These statistics were computed from optical flow, blobs, and regions. The values are shown in Table 2. The statistics were computed by accumulating the flow, region, and blob directions during T . In the case of optical flow, the table shows the percentage of pixels moving in different directions (the values in parenthesis show the percentage out of moving pixels only). In the case of blobs (and regions), the percentages are also of pixels but the directions are of blobs (and regions) containing these pixels. The frame rate was 10 fps. Notice that blobs were not able to sufficiently capture the motion in direction 135. This is due to the fact that blobs may be accurately initialized in terms of direction when the underlying motion of pixels is not consistent. This also stresses the need for the regions level which was able to capture that direction based on blob information even though the information is not accurate. Notice also that the most frequent region direction is 0 whereas the most frequent pixel direction is –45. This is merely a quantization issue since pixel directions are quantized while region directions are not.

The density represents the ratio of the number of pixels in a blob (or region) to the area of the bounding box of the blob (or region). This value exceeds 1 because of inaccuracies of modeling the blob (or region) as a rectangle. However, it is a useful indicator to identify a crowd; a region representing a crowd would have a high density (above 0.75). Other parameters that can be used to identify a crowd are the area (sufficiently large) and the speed (sufficiently small).

Direction (degrees)	Optical Flow	Blobs	Regions
Mass (pixels)	2,019,710	1,947,316	2,255,602
Area (pixels)	N/A	1,136,066	1,433,754
Density (Mass/Area)	N/A	1.7	1.57
No Motion	39	N/A	N/A
-135	6 (10)	6	13
-90	6 (10)	1	5
-45	20 (33)	1	8
0	6 (10)	79	44
45	6 (10)	3	8

90	2 (3)	3	5
135	12 (20)	1	13
180	0 (0)	2	0

Table 2. Statistics on 821 frames during 120 seconds.

4.3 Most used area

Figure 11 is a representation of most used areas in the scene which is a direct result computed by accumulating optical flow values over a period of 2 minutes.

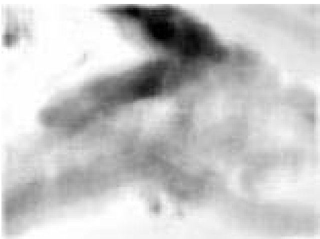


Figure 11. Most used areas during two minutes of day video.

4.4 Tracking results

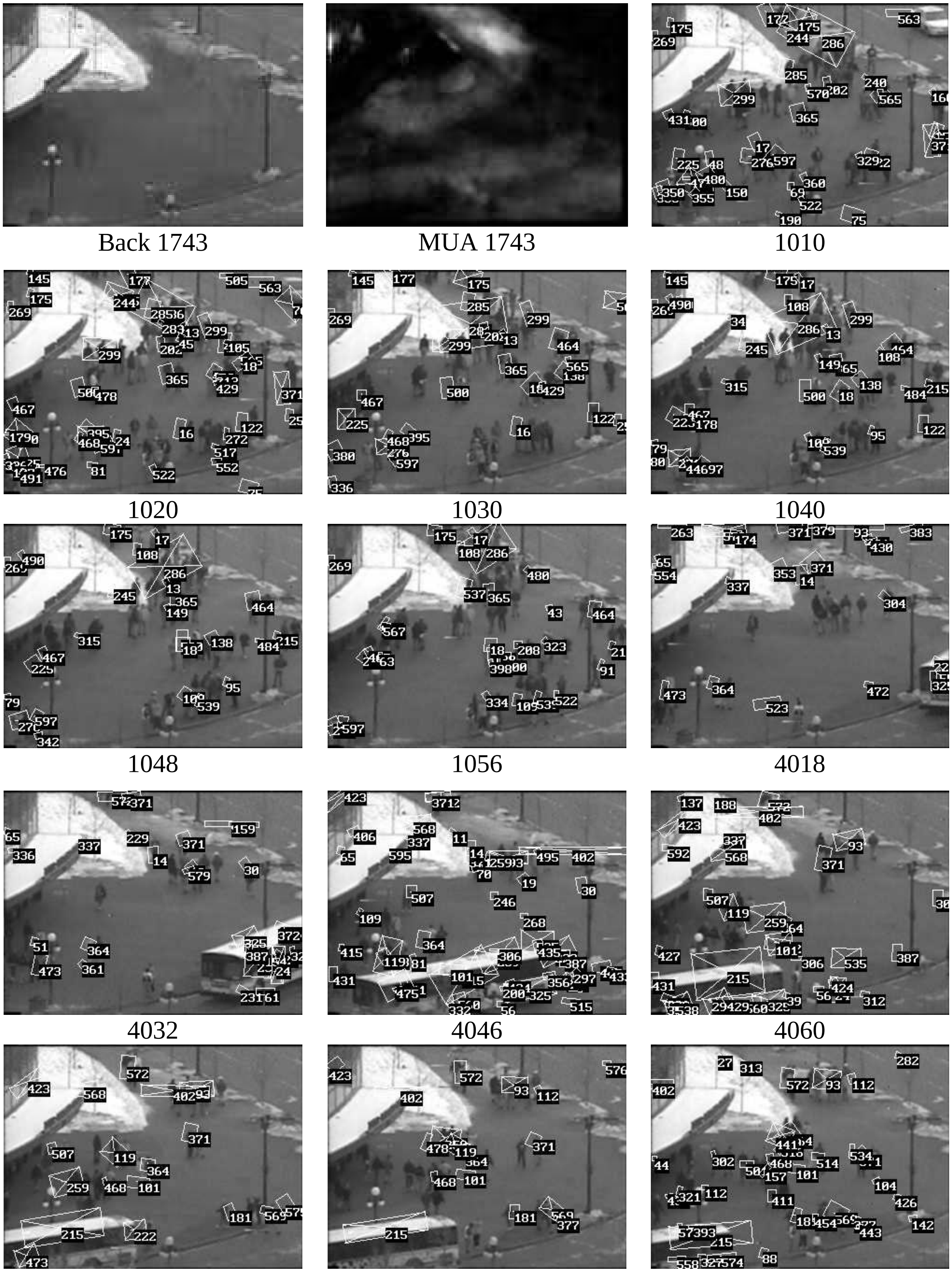
Figure 12, Figure 13, Figure 14, and Figure 15 show some of the tracking results. The results are presented as snapshots every 2 seconds. Rectangles represent regions. Regions with a cross represent detected crowds. The numbers shown are the region identifiers. Images marked ‘Back’ are background. Images marked ‘MUA’ show most used areas. Images marked ‘Opt’ show optical flow and optical flow segmentation.

5 Conclusions

This paper presents a vision-based system for monitoring crowded urban scenes. Our approach combines an effective detection scheme based on optical flow that can locate vehicles, individual pedestrians, and crowds. The detection phase is followed by the tracking phase that tracks all the detected entities. Traffic objects are not simply tracked but a wealth of information is gathered about them (position, velocity, acceleration/deceleration, bounding box, and shape features). Potential applications of our methods include intersection control, traffic data collection, and even crowd control after athletic events. Extensive experimental results for a variety of weather conditions are presented. Future work will be focused on methods to deal with shadows and occlusions.

6 Acknowledgements

This work has been supported in part by the Minnesota Department of Transportation and in part by the National Science Foundation through award #CMS-0127893.



Back 1743

MUA 1743

1010

1020

1030

1040

1048

1056

4018

4032

4046

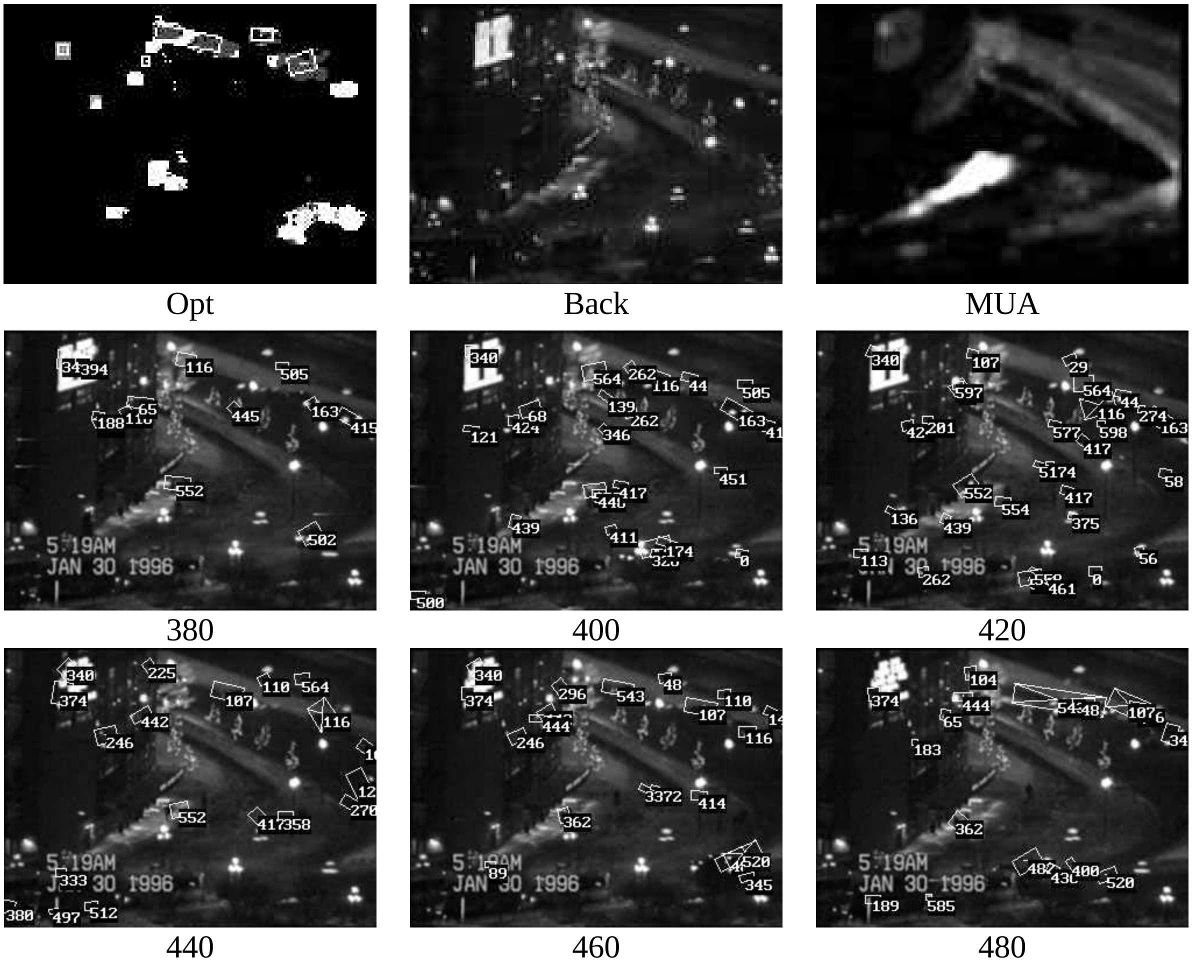
4060

4074

4088

4102

Figure 12. Results from a day video.

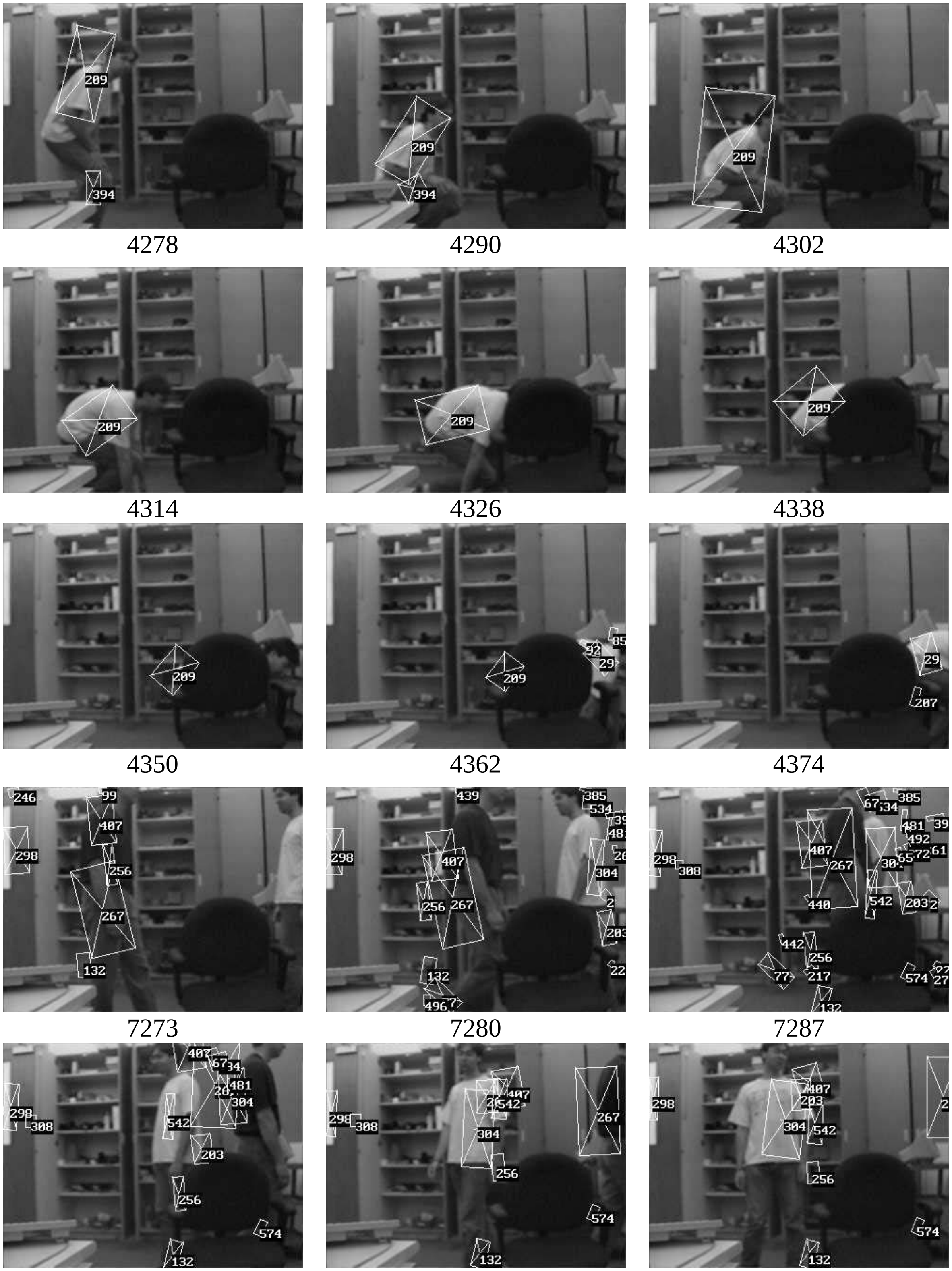


Opt Back MUA

380 400 420

440 460 480

Figure 13. Results from a night video.



4278 4290 4302

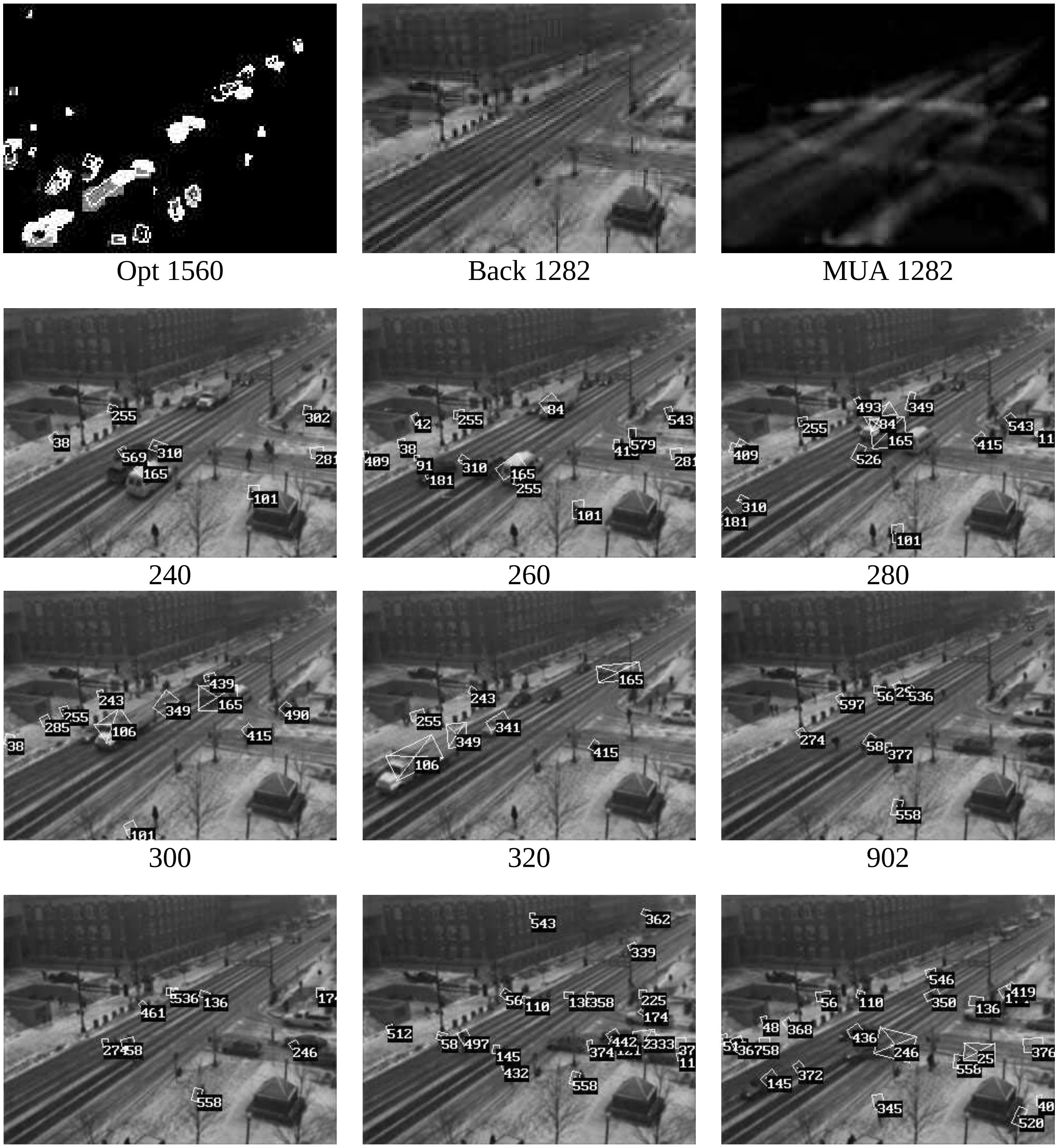
4314 4326 4338

4350 4362 4374

7273 7280 7287

7294 7301 7308

Figure 14. Results from a real-time video.



Opt 1560 Back 1282 MUA 1282

240 260 280

300 320 902

924 946 968

Figure 15. Results from a winter video.

7 Endnotes

1. A. Baumberg and D. Hogg, 'An Efficient Method for Contour Tracking Using Active Shape Models', in Proc. of IEEE Workshop on Motion of Non-rigid and Articulated Objects, pp. 195-199, IEEE Computer Society Press, November 1994.
2. M. Rossi and A. Bozzoli, 'Tracking and Counting Moving People', Proc. Second

IEEE International Conference on Image Processing, pp. 212-216, 1994.

3. O. Masoud, Ph.D. Thesis on 'Tracking and Analysis of Articulated Motion with an Application to Human Motion', University of Minnesota, Minneapolis, March 2000.
4. R. Cutler and M. Turk, 'View-based Interpretation of Real-time Optical Flow for Gesture Recognition', Third IEEE International Conference on Automatic Face and Gesture Recognition, Nara, Japan, April 14-16, 1998.
5. S. L. Dockstader, 'Motion Estimation and Tracking', University of Rochester, document on the web, http://www.ee.rochester.edu:8080/users/dockstad/research/rtop.html, May 24, 2001.
6. I. Haritaoglu, D.Harwood, and L.S.Davis, 'Real-Time Surveillance of People and Their Activities', IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 8, pp. 809-830, August 2000.
7. C. Sun, 'A Fast Stereo Matching Method', Digital Image Computing: Techniques and Applications, pp.95-100, Massey University, Auckland, New Zealand, December 10-12, 1997.
8. B. Galvin, B. McCane, K. Novins, D. Mason, and S. Mills, 'Recovering Motion Fields: An Evaluation of Eight Optical Flow Algorithms', British Conference on Computer Vision, Computer Science Department University of Otago, New Zealand.
9. M.J. Black and P. Anandan, 'A Framework for the Robust Estimation of Optical Flow', Proc. Fourth Int. Conf. on Computer Vision (ICCV'93), Berlin, Germany, May 1993.
10. C. Bernard, Ph.D. Thesis on 'Ill-conditioned Problems: Optical Flow and Irregular Interpolation', document on the web, www.cmap.polytechnique.fr/~bernard/these, 1999.
11. S.S. Beauchemin and J.L. Barron, 'The Computation of Optical Flow', Dept. of Computer Science, University of Western Ontario, ACM Computer Surveys, vol. 27, no. 3, pp. 433-467, 1995.
12. T. Rwekamp and L. Peter, 'A Compact Sensor for Visual Motion Detection', VIDERE, vol. 1, no. 2, Article 2, MIT Press, Winter 1998.

LIST OF FIGURES

LIST OF TABLES