On Local Detection of Moving Edges

Ten-lee Hwang James J. Clark

Division of Applied Sciences
Harvard University
Cambridge, MA 02138

Abstract

Optical flow is traditionally found by working on two or more input images in a time sequence. On the other hand, most edge detectors locate edges only on static images. In this paper, we therefore propose a detection framework with multiple velocity channels for moving edges based on a generalization of Canny's edge detector[6]. Finite state machines (FSM's) are set up at discrete lattice points in the image plane and operate based on the outputs of all velocity channels. The outputs of the FSM's denote whether there are edges at their corresponding positions and their states record the edge velocities. In the temporal dimension, statistics are attached to the edges to aid in removing phantom edges.

1 Introduction

People have proposed various algorithms in edge detection. Most of them operate on a single static image or a snapshot only. Another important low-level vision task is motion detection. The proposed algorithms mainly operate on two images sampled at different but close time epochs. On the other hand, biological motion models are basically built on the spatio-temporal domain, not just on two image snapshots only. As we possess more computational power, it is natural to consider edge detection and velocity flow extraction in the spatio-temporal domain and to develop computational algorithms compatible with the spatio-temporal biology models.

Assuming that the input images are so densely sampled in both the spatial and the temporal domains that the sampling artifacts are smoothed out in the spatio-temporal filtering, there are three different kinds of algorithms estimating the velocity flow of the moving edges. The first kind employs Gabor-like filters where the velocities of the image patches are extracted without regard to the image features, e.g., [5]. The second kind deals with the velocities of the zero crossings which can be considered as special edges, e.g., [2]. The third kind specifically extracts the velocities of the moving edges. For example, Haynes and Jain[4] employ the edge detection in one image and used two consecutive images to determine the edge motion. Kahn[8] proposed a moving edge detection mechanism based on a fixed triad tessellation and a careful spatio-temporal arrangement. Crowley et al.[3] track the movement of edge lines in the two dimensional image plane by Kalman filtering. All these three algorithms of the third kind use temporal information to calculate the velocity of the image edges, but none of them explicitly use temporal information to aid edge detection.

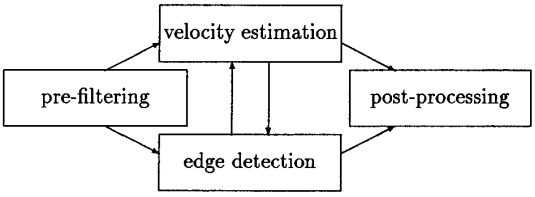

Figure 1: The data-fusion framework detecting moving edges consists of four modules conceptually.

We propose a data-fusion framework solving edge detection and velocity estimation together using temporal information. As shown in Figure 1, the framework *conceptually* consists of four modules. The pre-filtering module uses the generalized Canny's edge detectors[6] on two independent two-dimensional spatio-temporal planes (computationally more efficient than using the three-dimensional spatio-temporal filters directly) and then provides relevant spatio-temporal information to both the edge detection module and the velocity estimation module. The generalized Canny's edge detectors are optimal only when they are exactly tuned to the velocity of the one-dimensional edge. Therefore, several edge detectors tuned to different velocities and employed at the same time in the pre-filtering module appear necessary to detect the edges moving with different unknown velocities. Because the causal optimal filters are recursively realizable in the temporal domain, *two*-dimensional spatio-temporal *causal* edge detectors are used.

Conceptually, the edge detection module has to signal the velocity estimation module where the edges are approximately located. A moving real (versus phantom) edge should move consistently, so the velocity estimation module can predict the location of the moving edge in the next time epoch. By using this information, the edge detection module is able to throw away the phantom edges. As shown in Figure 1, both the edge detection module and the velocity estimation module share information and interact with each other. We are able to remove unnecessary assump-

tions made by each module and increase the confidence of the solutions to both problems. We will use finite state machines (FSM's) for the edge detection module and the velocity estimation module to clarify the algorithm and promote the expandability of the framework. The inputs to the FSM's come from the pre-filtering module. The post-processing module combines the results on two independent two-dimensional spatio-temporal planes and calculates the location and the normal velocity of the moving image edges. We leave the aperture problem unsolved as it is persistent unless some sort of global operations are used.

Because hypothesis testing is used in both the edge detection module and the velocity estimation module as an estimation technique, our framework employs both linear filtering and statistical classification approaches in the three-dimensional spatio-temporal domain. In contrast to our approach, Bouthemy modeled the moving edges in an image sequence as planar patches in the three-dimensional spatio-temporal domain[1]. A likelihood ratio test was used to detect edges and to simultaneously estimate the velocity components perpendicular to the edges. Hence, his approach uses both surface fitting and statistical classification techniques.

2 A FSM Model

For a perfect one-dimensional step edge located at $x = 0$ moving with a velocity v_0 at some time epoch, the convolutional outputs from N edge detectors tuned to N different velocities, $v_1 < v_2 < ... < v_N$, are maximized at $x = 0$ in the spatial domain if there is no noise. Figure 2 shows an example where $N = 3$. As the edge moves from $x = 0$ to $x = x_1$ the outputs are also shifted x_1 to the right. As a result, the output shape around $x = x_0$ changes from a basically monotonically decreasing curve, to a convex hill, and then to a basically monotonically increasing curve. These three different shapes suggest three different edge (closest to x_0) positions at different time epochs, that is, the edge position was less than x_0, around x_0, and then larger than x_0. At any discrete position, say x_0, we can then utilize the computed filter outputs tuned to the N different velocities and send a signal to the corresponding finite state machine (FSM) which records the transition of the N convolutional output curves at x_0.

Specifically, the local curve shape of the filtered outputs at each sampled point, say x_0, could be a $HILL$ (a convex hill), a DEC (a monotonically decreasing curve), a INC (a monotonically increasing curve), or a $VALLEY$ (a concave valley). It is locally determined by the average slopes at the right and the left sides of the point x_0, within a window of size $2w_c + 1$ centered at x_0 [7]. Other shapes of the outputs, in-between DEC and INC, or not strong enough to be $HILL$ or $VALLEY$, would be claimed as $TRAN$'s (transitions). Within the window of size $2w_c + 1$ centered at x_0, there would be $N \cdot (2w_c + 1)$ local output shapes determined from the outputs of N velocity channels. The final curve shape used as the input to the FSM at x_0 is decided by a winner-take-all scheme[7]. Notice that we report only a $HILL$ or a $VALLEY$ within the window to achieve the best edge localization. If the final curve shape at x_0 is a $HILL$ or a $VALLEY$ and v_k is the velocity channel with the largest

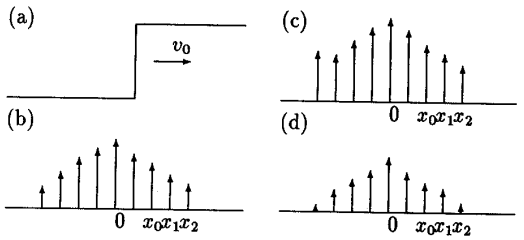

Figure 2: (a) A noise-free step edge moving with $v_0 > 0$; Illustrations on the expected outputs of the edge detectors tuned to three different velocities (b) $v_1 > v_0$; (c) $v_2 > v_0$; (d) $v_3 < v_0$.

output, the input to the FSM at x_0 would be $(HILL, k)$ or $(VALLEY, k)$. Otherwise, the final curve shapes, e.g., DEC, would be the inputs to the corresponding FSM's.

Recall that a FSM is characterized by $M = (K, Y, Z, q_0, \delta, \varphi)$ where K is a finite set of states; Y is a finite set of input symbols; Z is a finite set of output symbols; $q_0 \in K$ is the initial state; δ, the state transition function, is a function from $K \times Y$ to K; and φ, the output function, is a function of its next states only, i.e., from K to Z. In this context, let us define the elements of a FSM at x_0 as $K = \{Q_{any}, Q_{01}, .., Q_{0N}, Q_{11}, .., Q_{1N}, Q_{3l}, Q_{5l}, Q_{23l}, Q_{45l}, Q_{3d}, Q_{5d}, Q_{23d}, Q_{45d}\}$; $Y = \{(HILL, 1), ..., (HILL, N), (VALLEY, 1), (VALLEY, N), DEC, INC, TRAN\}$; $Z = \{0, 1\}$; and $q_0 = Q_{any}$. The states in the set K represent different temporal situations. For example, Q_{any} is an initial state and the resting state when there is not enough information. $Q_{0j}, j = 1, ..., N$, means there is a light edge (the intensity increases as x increases) at x_0 and its velocity is closest to v_j. Please see [7] for detailed explanations of the other states and the state transition function δ. Notice that we are conservative in most situations, e.g., the machine goes to the state Q_{any} if the current inputs are very unlikely to happen. Also, we would rather report phantom edges than miss any possible edges, because it is easier to remove extra edges than to add new edges in later processing. As far as φ, the machine outputs 1 if and only if the next state is Q_{0j} or Q_{1j} (dark edges), $j = 1, ..., N$. In other words, the machine at a particular position outputs 1, if and only if there would be an edge at that position.

There are many different ways in setting up the structure of the above state machine. We will mention two possible modifications toward two different directions corresponding to two different design philosophies. In the first modification, we suggest a simpler and faster realization. The state machine M can be simplified to $M' = (K', Y', Z, q'_0, \delta', \varphi')$ where $K' = \{Q'_{any}, Q_{01}, ..., Q_{0N}, Q_{11}, ..., Q_{1N}\}$ and $Y' = \{(HILL, 1), ..., (HILL, N), (VALLEY, 1), ..., (VALLEY, N), OTHERS\}$. $q'_0 = Q'_{any}$ and $Z = \{0, 1\}$. The state Q'_{any} therefore includes the situations which were represented by Q_{3l}, Q_{5l}, Q_{23l}, Q_{45l}, Q_{3d}, Q_{5d}, Q_{23d}, Q_{45d} in addition to Q_{any} in M. Evidently, $OTHERS$ in M' is equivalent to other three input symbols, DEC, INC, and $TRAN$ in M. The new transition function δ' is

δ'	$(HILL, k)$	$(VALLEY, k)$	$OTHERS$
Q'_{any}	Q_{0k}	Q_{1k}	Q'_{any}
Q_{0j}	Q_{0k}	Q'_{any}	Q'_{any}
Q_{1j}	Q'_{any}	Q_{1k}	Q'_{any}

φ' is 1 if and only if the next state is Q_{0j} or Q_{1j}, $j = 1, ..., N$. In the second modification, the FSM's can be extended to stochastic FSM's (SSM's). Because the SSM's capture stochastic temporal transition information, they are better test beds for investigating the temporal behaviors of edges. It is very likely that the outputs from the pre-filtering module presented to the SSM's and the states of the SSM's would be defined differently. Actually, it is possible to use other pre-filters, depending on the defined edge shape and other proposed criteria.

Once an edge **e** with a velocity v_1 is claimed by the FSM at some point x_1, we will check if **e** is also claimed as an edge at the FSM at $x_1 + v_1 \cdot T$ at next time epoch, where T is the sampling period of the image sequence. If **e** does, it is exactly tracked. Due to measurement noise and quantization errors on the input images, if **e** appears at $x_1 + v_1 \cdot T \pm 1$ ($x_1 + v_1 \cdot T \pm 2$), it is partially (weakly) tracked. If not so, we will say **e** is lost. In this case, **e** disappears from the image plane; errors in x_1 or v_1 are too large; or **e** is just a phantom edge. **e** would be considered as a real edge if **e** is either exactly or partially (weakly) tracked until it is lost. An improvement is to use $E[v] = \frac{\sum_{i=1}^{n} v_i}{n}$, instead of v_i, if **e** is tracked for consecutive n images to reduce the velocity estimation errors. Initially, we give **e** a confidence measure **Conf** = 1.0. At each time epoch that follows, **Conf** is increased by 0.2 if **e** is exactly tracked, decreased by 0.2 (0.4) if **e** is partially (weakly) tracked. If there is a consistent velocity estimation error, **Conf** would finally fall below 0.5 and then we would claim **e** is lost. Edges have to be tracked at least for N_1 consecutive times to be reported as edges. When n is larger than another threshold, say N_2, we will set n to be N_1 with $E[v]$ unchanged. This hysteresis would accommodate the velocity changes while keeping reporting the existence of the edges.

To extend the edge detection and velocity estimation from one spatial dimension to two spatial dimensions, let us pick up two independent vectors in the two-dimensional image plane as two coordinate axes. If the proposed algorithm in the previous section is employed along these two axes independently, we would have two FSM's running at each lattice point in the image plane. If we define the edge points are located at the lattice points where the outputs of two FSM's are both 1, the edges points would form more straight but less uniformly distributed lines. If we claim the lattice points as the edge points when either of the FSM outputs or both are 1, the edge points form less straight but more continuous lines[7]. Suppose the edge velocities detected in two independent axes u_1, u_2 at a particular point are v_1 and v_2 respectively, then the final velocity of the point is $v = \frac{v_1 v_2 \sin\phi}{\sqrt{v_1^2 + v_2^2 - 2v_1 v_2 \cos\phi}}$, where ϕ is the angle between u_1 and u_2. The angle between the normal velocity v and the axis u_1 is $\cos^{-1}\frac{v}{v_1}$[7].

3 Implementation

We choose the optimal filter[6] with a spatial size of 11, $c = 1$, $\alpha = 3$ and $\tau = 1$. Five optimal filters are used simultaneously, tuned to -2 (pixels/time frame), -1, 0, 1, and 2 respectively, namely $N = 5$. The output of any optimal filter has to be greater than $\mathbf{thr} = \frac{a_0}{2\alpha^2\tau}$ to be claimed as a real edge. This means that a_0 is the minimum absolute abrupt intensity change to be an edge[6]. We set $w_c = 3$ in detecting the output curve shape.

First, we ran the algorithm over one dimensional synthetic edges corrupted by white noise with different maximum amplitude[7]. Let γ be the ratio between the edge intensity and the maximum noise amplitude. We found that as γ decreases, more phantom edges are detected, even if we use a larger a_0. By checking consistency ($N_1 = 2$ and $N_2 = 5$), most spurious edges are removed and the locations and velocities of the real edges are more accurate.

We tested the algorithm on both synthetic and real images with a_0 fixed at 80. The lattice points are claimed as edge points when either of the FSM outputs, or both, are 1. The edge velocities are collected in a 3×3 window. Each of the synthetic one-byte (0-255) images is of size 150×150, corrupted by uniformly random noise with a maximum intensity 10. The real images are taken by a Panasonic WV-CD50 camera. The original 485×512 one-byte images are blown down to 242×256 for faster processing.

As shown in Figure 3, four blocks moves with different velocities. Between two consecutive frames, the upper-left small block does not move at all; the upper-right small block moves one pixel to the left; the central block moves one pixel to the right and one pixel to the bottom; the bottom-right block moves two pixels to the left. The intensity of the background, the upper-left, upper-right, central, and the bottom-right block are 10, 170, 180, 230, and 110 respectively. The detected edge positions shown by little boxes are accurate within one pixel. The estimated displacements shown by lines are five times of the average velocities between two consecutive frames so far. Due to the aperture problem, the normal velocities are correctly recovered. As another example, a sequence of synthetic images with a moving circle are created, as shown in (d) in Figure 3. The circle moves one pixel to the right and one pixel to the bottom between two consecutive image frames. The intensity of the background and the circle are 140 and 250 respectively. The detected edge positions are accurate within one pixel. Due to the aperture problem, the bottom-right and the upper-left parts of the circle exhibit the largest normal velocities; the bottom-left and the upper-right parts have a zero velocity; and some local movements different from the global velocities are reported at some parts of the circle.

We moved the camera towards a telephone and took a sequence of real images, as shown in (e) and (f) in Figure 3. The bottom and the left parts of the scene go outward, so do the recovered velocities. If we overlap the pictures, we will see the edge velocities are correctly estimated. As another example, we took the pictures of the Harvard Head, as shown in Figure 4. The camera was moved to the right, so the recovered estimated displacements are running to the left. Some horizontal edges on the bottom-left of the input pictures are reported by Canny's edge detector but not by our algorithm, because they are too noisy and not straight enough (the translation is horizontal). The smaller a_0 is, the more detailed the recovered edges are. We are currently tracing the edge points based on a double thresholding scheme in order to improve the results.

4 Summary

In [6], a set of optimal (under Canny's original criteria) edge detectors are derived in the spatio-temporal domain. Assuming that densely sampled images are available, a flexible framework with multiple velocity channels are then pro-

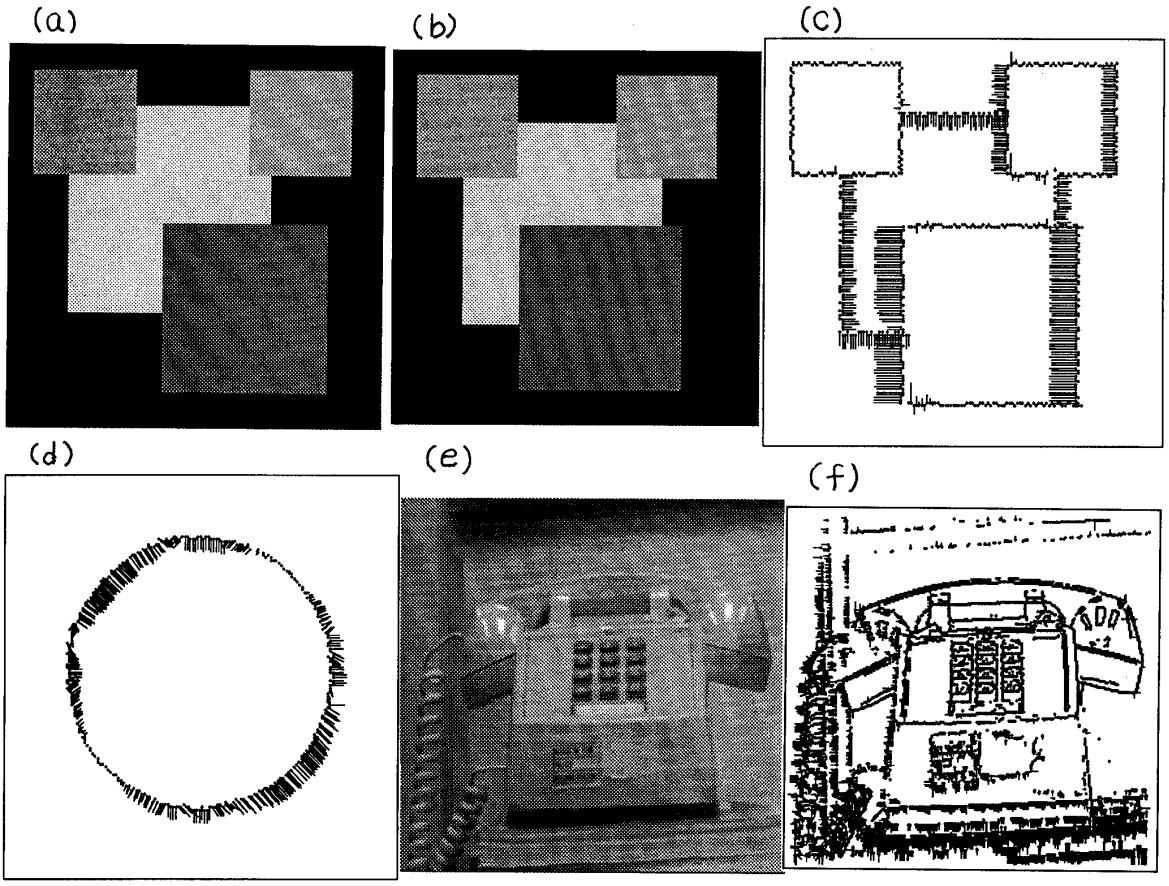

Figure 3: (a) and (b) are the 5th and the 10th frames of the input four-block synthetic images respectively. The detected edge positions and the estimated displacements for the four-block [one-circle] image sequence for the next 5 frames at the 10th frame are shown in (c) [(d)]. (e) is the 10th frame of a real image sequence and (f) shows the the detected edge positions and the estimated displacements at the 10th frame for the next 5 frames.

posed to fuse the edge detection and edge velocity estimation which are traditionally separately dealt with. The correspondence problem is inherently solved but only the velocities perpendicular to the edges are recovered. We describe how to set up state transition machines at discrete lattice points and how the state machines operate based on the outputs of all velocity channels. The outputs of the state machines denote whether there are edges at their corresponding positions and the states of the state machines record the edge velocities. To record real edges (moving or not) and to remove spurious edges more confidently, the mean of the edge velocity within a certain period of time is attached to the edges. If the edges move according to their associated velocities, they are considered as real edges, otherwise, they are treated as phantom edges. This is found to be very effective. Some implementation results on synthetic and real images are shown and compared with Canny's edge detectors. Last but not the least, the whole algorithm is uniform, local, and therefore implementable in SIMD machines or VLSI technologies.

5 Acknowledgement

Comments from Alan Yuille are appreciated. This work was supported by the Brown/Harvard/MIT Center for Intelligent Control Systems under Army Research Office grant DAA103-86-K-0171.

References

[1] P. Bouthemy, *A Maximum Likelihood Framework for Determining Moving Edges*, IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. PAMI-11, No. 5, pp. 499-511, May. 1989.

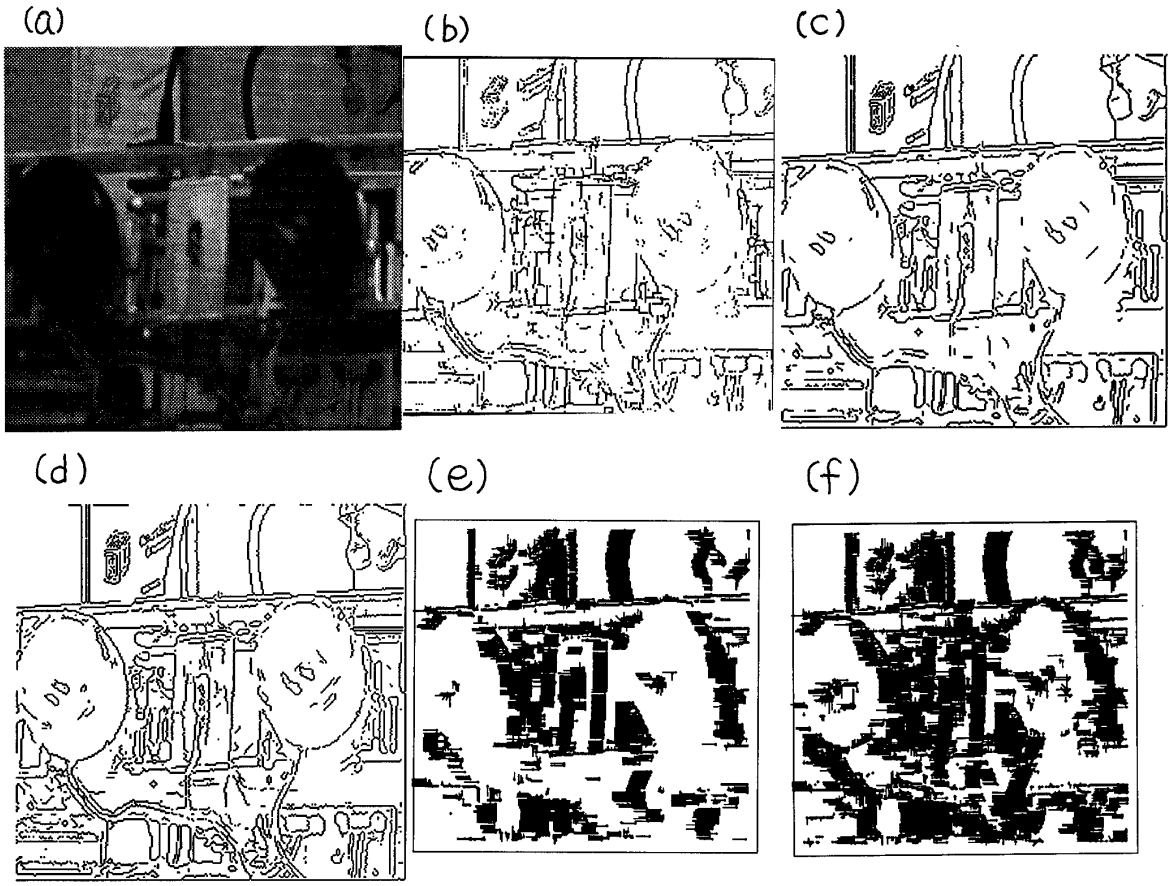

Figure 4: (a) is the 10th frame of the Harvard Head image. (b) shows the detected edges up to the 10th frame by the algorithm using $a_0 = 15$. (c) and (d) are the detected edge maps by Canny's edge detector on the 10th frame only, with a noise margin 50% and 40% respectively. (e) and (f) show the detected edge positions and the estimated displacements for the next 5 frames at the 10th frame with $a_0 = 30$ and $a_0 = 15$ respectively.

[2] B. F. Buxton and H. Buxton, *Computation of Optical Flow from the Motion of Edge Features in Image Sequences*, Image and Vision Computing, vol. 2, No. 2, pp. 59-75, May 1984.

[3] J. L. Crowley, P. Stelmaszyk, and C. Discours, *Measuring Image Flow by Tracking Edge-Lines*, The second International Conference on Computer Vision, pp. 658-664, 1988.

[4] S. M. Haynes and R. Jain, *Detection of Moving Edges*, Computer Graphics and Image Processing, vol. 21, pp. 345-367, 1983.

[5] D. Heeger, *Optical Flow Using Spatio-temporal Filters*, International Journal of Computer Vision, pp. 279-302, 1988.

[6] T. Hwang and J. Clark, *A Spatio-temporal Generalization of Canny's Edge Detector*, Proceedings in computer vision, ICPR, June, 1990.

[7] T. Hwang and J. Clark, *On Local Detection of Moving Edges*, TR, no. 89-6, Harvard Robotics Laboratory, 1989.

[8] P. Kahn, *Local Determination of a Moving Contrast Edge*, IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. PAMI-7, No. 4, pp. 402-409, July, 1985.