Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Pattern Recognition Letters 24 (2003) 1533–1544

Pattern Recognition Letters

www.elsevier.com/locate/patrec

Real-time thresholding with Euler numbers

L. Snidaro *, G.L. Foresti

Department of Mathematics and Computer Science (DIMI), University of Udine, Via delle Scienze 208, 33100 Udine, Italy

Received 1 May 2002

Abstract

The problem of finding an automatic thresholding technique is well known in applications involving image differencing like visual-based surveillance systems, autonomous vehicle driving, etc. Among the algorithms proposed in the past years, the thresholding technique based on the stable Euler number method is considered one of the most promising in terms of visual results. Unfortunately its high computational complexity made it an impossible choice for real-time applications. The implementation here proposed, called *fast Euler numbers*, overcomes the problem since it calculates all the Euler numbers in just one single raster scan of the image. That is, it runs in O(hw), where h and w are the image's height and width, respectively. A technique for determining the optimal threshold, called *zero crossing*, is also proposed.

Keywords: Automatic thresholding; Euler number; Real-time; Optimal threshold; Video surveillance

1. Introduction

Image differencing is a straightforward and intuitive method for change detection. The absolute value of the difference between a reference image and the current one gives the changes occurred in the time interval separating the two frames. The problem is deciding which values of the difference image are to be considered information (i.e., indicating moving objects in the scene) and which are not (i.e., noise generated by the camera). A binarization of the difference map has to be done and a threshold is needed.

In the past years, several methods for automatic thresholding have been proposed. A survey of the most effective techniques has been made by Rosin and Ellis (1995) and Rosin (1998). The works of Kapur et al. (1985), Tsai (1985) and Otsu (1979) are well-known in the literature. Among the recent developments are the results of Sezgin and Taaltin (2000) and Friel and Molchanov (1999).

The *stable Euler number* technique is very promising in terms of visual results and stands out as one of the most effective algorithms as pointed out by Rosin and Ellis (1995) and by Rosin (1998). Unfortunately, its high computational complexity has always precluded its employment in real-time applications, and thus relegated it to off-line processing and laboratory experiments.

The new generation of visual-based real-time applications, such those described by Collins et al. (2001) and Foresti (1999), could improve their performance by applying the stable Euler number method for image thresholding.

* Corresponding author. Fax: +39-432-558499.
E-mail addresses: snidaro@dimi.uniud.it (L. Snidaro), foresti@dimi.uniud.it (G.L. Foresti).

0167-8655/03/$ - see front matter
PII: S0167-8655(02)00392-6

Following the classification made by Rosin (1998), the stable Euler number algorithm falls in the category of methods describing the spatial distribution of the signal. In particular, the Euler number gives a measure of the connectivity of the image and can be preferred to region counting since it is locally countable (as indicated in the work of Gray (1971)) and it yields very similar results as suggested by Rosin and Ellis (1995).

The stable Euler number method consists in thresholding the difference image at g different levels, computing the Euler number for each binarization, and choosing the "optimal" threshold value that better separates signal from noise.

Leaving out for the moment the last step, let us consider the computational complexity of the former ones. These require the computation of the binary image and its Euler number for each of the g thresholds. This means an $O(pg)$ complexity, where p is the number of pixels constituting the image. But since p does not change linearly, that is, image dimensions vary adding a row or a column at time, a better understanding of the complexity may be achieved writing $O(hwg)$, where h and w are respectively image's height and width. If square images are considered, the complexity is obviously $O(n^2g)$, were n is the image's side length. The computational effort quashes every hope to make it run in real-time even for small images with 256 grayscale levels.

The fast Euler numbers (FEN) algorithm here proposed calculates the Euler number for every possible threshold with a single raster scan of the image thus running in $O(hw)$.

Therefore, the output of the algorithm is an array of Euler numbers: one for each threshold value. The quest for the optimal threshold can now be started. The problem is discussed in Section 4. A new method, called *zero crossing* (ZC) and based on the *corner* technique proposed by Rosin (1998), has been developed during the experiments and is presented in this paper.

2. Stable Euler number

For two-dimensional spaces the Euler number simply gives the number of distinct objects minus the number of holes inside objects (see Gray (1971) for details). Finding an image's Euler number involves the computation of the number of the q_1, q_3 and q_d quads contained in the image (Fig. 1). A quad is a 2×2 mask of bit cells. The q_1 and q_3 quads differ from the number of lit pixels: 1 and 3 respectively. The q_d is a diagonal quad, that is, the lit bits are diagonally disposed. Obviously, there are four possible configurations for the q_1 and q_3 quad types and two for the q_d.

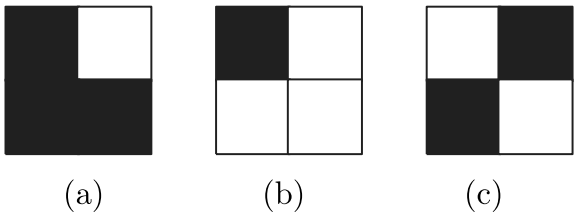


Fig. 1. Bit quads. A possible configuration for (a) q_1, (b) q_3, and (c) q_d is shown.

The Euler number $E(t)$ for a threshold t can be calculated using the following formula:

$$E(t) = \tfrac{1}{4}[q_1(t) - q_3(t) - 2q_d(t)] \quad (1)$$

as indicated by Gray (1971). The number of regions (and therefore the Euler number) in the binarized difference map is expected to diminish as the threshold increases. In fact, a high threshold generally cuts off all the spurious bits given by random noise. The classic downside is that valuable information may be eliminated as well.

Rosin (1998) points out that the graph of Euler numbers plotted against thresholds (Eq. (1)) resembles a decaying exponential. A good threshold is hence to be found in the plateau of the graph, where the Euler number is varying by small amounts and the number of regions is almost stable. This indicates that those regions are actually (or most probably) moving objects in the scene.

The method used to decide which threshold should be elected as optimal is now irrelevant. What should be apparent at this point is that the Euler number of each binarization (that is, for all possible values of the threshold) should be known to start deciding which one is corresponding to the optimal threshold.

Any attempt to sub-sample the Euler number graph with only a few threshold's values (therefore calculating the Euler number for only those values) will inevitably yield a coarse, unreliable threshold. In addition, unless parallel processing is employed, it will be still too computationally expensive for real-time applications.

3. Fast Euler thresholding

The method proposed is based on the observation that the information needed to compute the Euler number for all possible thresholds is already all contained in the difference map. As seen in Eq. (1), the Euler number depends on the counters q_1, q_3 and q_d. Each quad of the difference map will contribute to the counters according to its current binarization given by the threshold t.

Thus, by making few simple tests, it can be determined how a quad contributes to the counters. Suppose that binarization is performed setting to white (object) a pixel *pix* if $value(pix) > t$, where t is the threshold, and setting it to black (background) if $value(pix) \leqslant t$ instead.

The quad in Fig. 2 is a q_3 for $1 \leqslant t < 3$, is a q_d for $3 \leqslant t < 5$, and is a q_1 for $5 \leqslant t < 8$. If the elements of the quad are labelled with the letters a, b, c, d for increasing values so that a is the minimum and d is the maximum, the quad will contribute to the q_1, q_3 and q_d counters according to the following intervals of the value of the threshold t, if the indicated conditions are satisfied:

$$q_1 : t \in [c, d-1] \quad c < d-1 \tag{2}$$

$$q_3 : t \in [a, b-1] \quad a < b-1 \tag{3}$$

$$q_d : t \in [b, c-1] \quad b < c-1;\ c < d-1; \tag{4}$$

c and d are disposed on a diagonal of the quad

3	5
8	1

Fig. 2. A quad of the difference map before binarization.

So, the intervals are easily determined by checking the above conditions. If a condition is not met then the quad will never contribute to the corresponding counter since it will never assume that configuration. To build the Euler numbers' array $\boldsymbol{E}$ (Euler number calculated for each possible threshold), several working vectors must be constructed first.

Using three arrays, one for each counter, would solve the problem but would also be a complexity trap. To understand the reason, a closer look at the implementation is necessary. Having a vector for each counter, with a number of elements equal to the number of possible thresholds, $\boldsymbol{E}$ can be simply computed applying Eq. (1) for each element:

$$\boldsymbol{E}[i] = \tfrac{1}{4}(\boldsymbol{q_1}[i] - \boldsymbol{q_3}[i] - 2\boldsymbol{q_d}[i]) \tag{5}$$

With a single raster scan of the difference image, the contributing intervals can be determined for each quad. Although, incrementing by one unit each element of the counter arrays in the intervals specified would be a mistake. Taking as example the quad of Fig. 2, the elements of $\boldsymbol{q_1}$, $\boldsymbol{q_3}$ and $\boldsymbol{q_d}$ should be modified as shown in Fig. 3.

This must be done for the contributing intervals of every quad. The for loops needed to accomplish the operation will make the computational requirements rise dramatically. Since for every quad g updates could be necessary at worst case (where g is the number of possible thresholds), it can easily be seen that an O(hwg) complexity is obtained once again.

The problem can be overcome using two arrays for each counter: one keeps track of the starting points of the intervals and the other records the ending points for each threshold's value t. So each

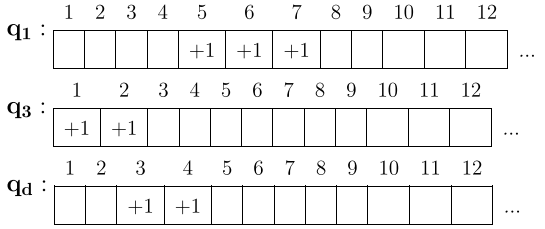


Fig. 3. How the quad of Fig. 2 affects the $\boldsymbol{q_1}$, $\boldsymbol{q_3}$ and $\boldsymbol{q_d}$ arrays of counters.

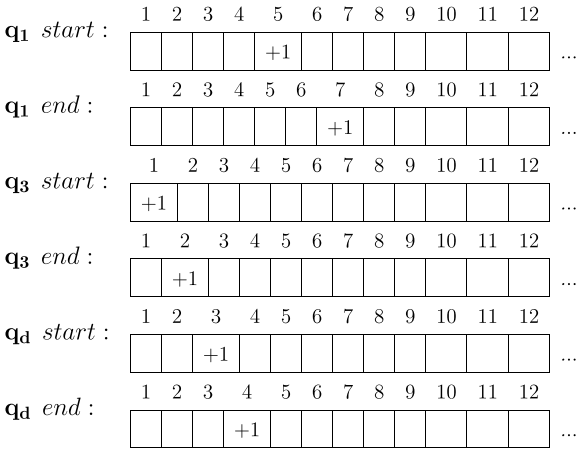


Fig. 4. How the information about the contributing intervals of the quad in Fig. 2 should be efficiently stored.

```
q1[0]= q1start[0];
q3[0]= q3start[0];
qd[0]= qdstart[0];
E[0] = (q1[0]+ q3[0] + 2*qd[0])/4;

for ( i=1; i<=maxThresholdValue; i++)
{
   q1[i]= q1[i-1] + q1start[i] - q1end[i];
   q3[i]= q1[i-1] + q3start[i] - q3end[i];
   qd[i]= q1[i-1] + qdstart[i] - qdend[i];
   E[i] = (q1[i]+ q3[i] + 2*qd[i])/4;
}
```

Fig. 5. The *C* code used to compute the *E* array of the Euler numbers.

cell contains the number of intervals starting at t and ending at t for the "start" and "end" arrays respectively. The previous example would be treated as indicated in Fig. 4.

In this way only a fixed amount of operations is necessary for each quad (six updates at the most). Now, with a single for loop, the $\boldsymbol{q_1}$, $\boldsymbol{q_3}$, $\boldsymbol{q_d}$, and $\boldsymbol{E}$ arrays can be computed.

Let us take $\boldsymbol{q_1}$ as example: $\boldsymbol{q_1}[i]$ is the number of q_1 quads when the threshold $t = i$ and can be computed considering the number of intervals spanning through $t = i$ (started at $t < i$), adding the intervals starting at $t = i$ and subtracting those ending at $t = i$. $\boldsymbol{q_3}$ and $\boldsymbol{q_d}$ can be obtained in the same way. The Euler number is given by (5). The *C* code reported in Fig. 5 will hopefully clarify the procedure.

The for loop may be interrupted as soon as $\boldsymbol{E}[i] = 0$. The six counter arrays are constructed on the fly during the raster scan of the difference map and this, as seen, runs in $O(hw)$ and dominates the $O(\max \mathit{ThresholdValue})$ operations required by the for loop above. The overall computational complexity is therefore $O(hw)$.

4. Optimal threshold

The discussion on how the optimal threshold should be chosen is beyond the purpose of this paper and is left to further research. Nonetheless are here described and discussed the techniques used during the experiments.

Two methods were tested: the *corner* method (presented by Rosin (1998)) and a heuristically modified version of it, called *ZC* method, which is here presented.

Suppose to draw the graph of the Euler numbers against thresholds. The point of the curve at maximum distance from the straight line passing between the end points of the curve is the *corner* of the graph. The corresponding threshold is chosen.

As can be seen from Fig. 6, the point *C* is the *corner* and resides just before the plateau of the curve. The corresponding threshold is t_c and is chosen as optimal value.

This technique is based on the intuitive argument that the *corner* threshold gives the best signal to noise ratio. During the experiments the tech-

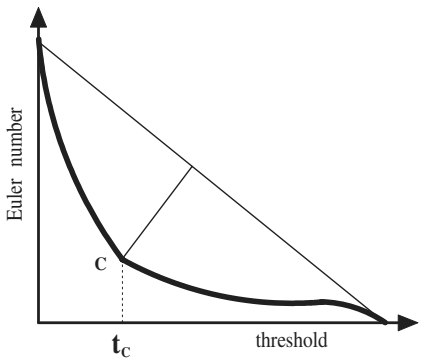


Fig. 6. The graph of the Euler numbers against thresholds.

nique usually yielded a threshold that was lower than the desiderated one. This may depend on the filtering steps carried out before the computation of the Euler numbers. Although, the *corner* threshold was always lower than expected.

Observing more in detail the curve of Euler numbers, some considerations can be done. When processing noisy video sequences (i.e., outdoor image sequences for visual surveillance), the *corner* of the curve may be in correspondence of still a great number of regions and the slope of the curve is probably far from stabilizing. Fig. 7 shows the Euler curve for an IR image. The *corner* threshold is 11.

Rosin (1998) correctly pointed out that when the Euler number is almost stable (the plateau region, generally corresponding to high thresholds) its small variations are hardly due to noise. That is, small variations mean that—most probably—the noise has been wiped out. As seen in noisy sequences, although, the *corner* technique fails to detect the starting point of the stable region.

As a matter of fact, in these conditions, the curve will be very unstable. That is, zooming in the plateau region it may look like a teeth saw. The fact remains that a more effective threshold can be found downwards from the *corner* point.

Now the problem is deciding when the Euler number is getting stable. Obviously, introducing a tolerance parameter should be avoided as it would severely affect generality and performance.

4.1. The ZC technique

The ZC heuristic tries to find a solution analyzing the slope of the Euler numbers curve. Fig. 8(a) shows the graph of the derivative computed for the curve in Fig. 7(a). It can be seen that the slope tends to zero in correspondence of the plateau region of the Euler curve. Zooming in, the behavior of the slope is not so smooth as it may seem. Fig. 8(b) reports the variations of the slope for the plateau region.

Thus, when the Euler curve is becoming smoother (after the *corner* point), its derivative is approaching the X-axis. The plateau region of the Euler curve corresponds to small fluctuations (positive and negative) of the slope along the X-axis.

The ZC technique detects the first zero crossing of the slope after the *corner* point. In fact, the slope can cross the zero various times before the *corner* point due to the instability of the Euler curve at low thresholds. Either the first zero value or zero crossing of the derivative, after the *corner* point, can be considered a good index of the stability of the Euler curve. That point is therefore chosen as threshold value.

If $\boldsymbol{de}$ is the array of the slope values against thresholds, the optimal threshold may be found when the following inequality is satisfied:

$$\boldsymbol{de}[i] \cdot \boldsymbol{de}[i-1] \leqslant 0 \quad (i > i_{\text{corner}}) \tag{6}$$

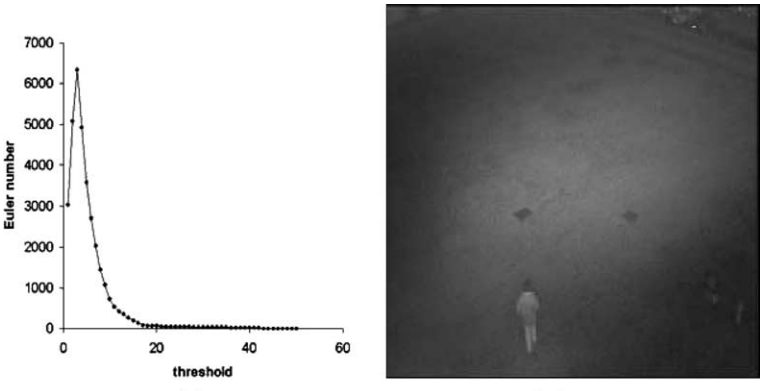


Fig. 7. Euler curve (a) for an IR image (b).

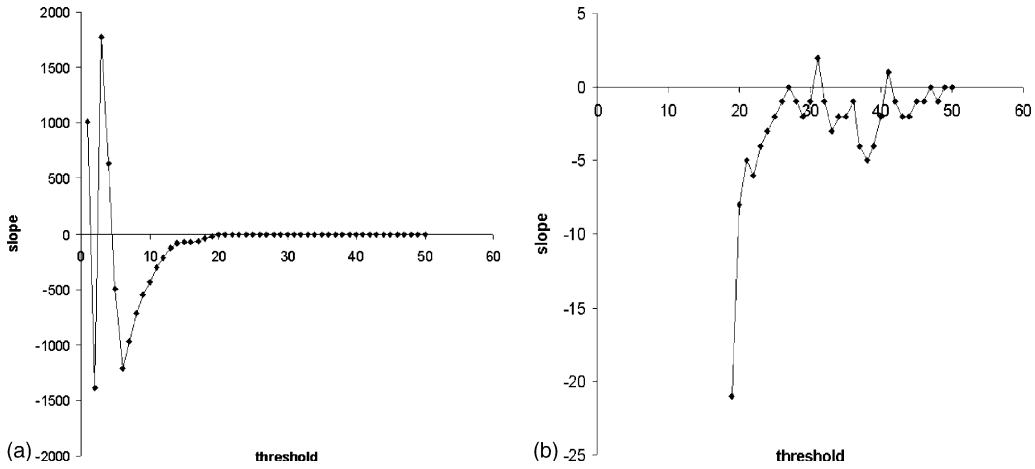


Fig. 8. Slope graphs for the Euler curve in Fig. 7. (a) Slope, (b) slope of the plateau.

where i_{corner} is the value found by the *corner* method.

As mentioned before, the *corner* threshold for Fig. 7 is 11. As can be seen in Fig. 8(b), the ZC method yields a considerably higher value: 26. The next section explains the effects of these values.

What can be observed at this point is that the ZC technique fully exploits the concept of stable Euler number. The algorithm finds a threshold as soon as the Euler curve stabilizes after the *corner* point. Either a zero value or zero crossing of the derivative stops the search.

The worst case may be an ever decaying Euler curve: in this case, the returned value of the ZC is the highest possible threshold (the intersection point of the Euler curve with the X-axis). This worst case, in practice, never occurs. The ZC will almost ever return a value at the beginning of the plateau region of the Euler curve.

The computational complexity of the ZC technique is $O(g)$, where g is the number of possible thresholds. The computation of the *corner* method is included. In practice, the ZC finds the optimal threshold with very few steps. For a 256 gray-scale image, the Euler graph generally does not span longer than $256/2$; the *corner* point approximately splits the Euler graph at $1/5$. Therefore, the ZC searches at worst between $(4/5)(256/2) \approx 100$ values. During the experiments, the ZC yielded thresholds which were at max 20 values above the *corner* point.

$O(g)$ is dominated by the complexity of the FEN algorithm. The overall computational complexity of the FEN with the ZC thresholding method is therefore $O(hw)$, where h, w are the image's dimensions.

This method gave good results and proved to be sufficiently reliable as shown in the following section.

5. Experimental results

The algorithms proposed in this paper were employed and tested in a multisensor real-time system for video-surveillance. The system, which is currently being developed at the AVIRES [1] laboratory of the University of Udine, is able to manage heterogeneous sensors (e.g., optical, infrared, radar, etc.) to operate during night and day and in presence of different weather conditions (e.g., fog, rain, etc.).

Due to the different working conditions and nature of the sensors, video signals must be processed using automatic techniques. A color camera and a b/w camera with near infrared response have been employed for the experiments. Image grabbing was performed at 256×256 pixels resolution.

[1] Artificial vision and real-time systems.

The tests were conducted primarily using outdoor sequences, which are the most appropriate test bench for automatic thresholding algorithms. The methods tested for both computational performance and visual results were the following:

- Kapur thresholding
- Euler thresholding with both *corner* and *ZC* techniques
- FEN with both *corner* and *ZC* techniques

A PC with a 1 GHz CPU and with no dedicated hardware for video processing was used. The image grabber employed was a Matrox MeteorII board. Image grabbing was performed at 25 fps. The system performance was measured for each thresholding algorithm: the Kapur and FEN methods sustained the grabbing frame rate running at 25 fps. The classic Euler algorithm performed at 2 fps.

Visual results are presented comparing the Kapur threshold against the *corner* and *ZC* techniques. The Euler numbers were computed using the fast Euler algorithm for speed's sake, but the classical Euler method would have produced the same results.

A background image, updated with the Kalman filter technique (see Foresti (1998) for details), was used as reference for the three thresholding techniques.

For each frame, the obtained threshold *th* is reported.

5.1. First IR sequence

The frames shown in Figs. 9, 13 and 17 are taken from an infrared video sequence shot with the b/w camera. An infrared beacon was used to obtain near IR response from the camera. The sequence was shot at night in presence of a dense fog bank. The camera was placed to monitor the front courtyard at Rizzi building of the University of Udine.

Nobody is present in the frames of Fig. 9, only the fog bank is visible in the low part of the image.

Fig. 10 shows the source frames of Fig. 9 thresholded with the Kapur method. The threshold is clearly too low, ranging from 6 to 7 through the four frames, and the images are heavily affected by the noise generated by the fog. False alarms (the system detects as moving object something that is part of the scenery) easily arise in this situation.

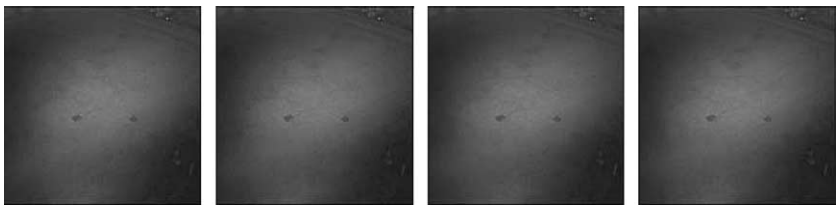
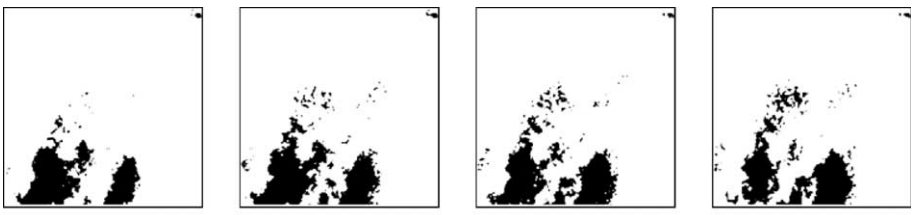


Fig. 9. Source frames from the 1st IR sequence.



(a) $th = 7$ (b) $th = 6$ (c) $th = 6$ (d) $th = 6$

Fig. 10. Kapur method applied to the frames of Fig. 9.

Fig. 11 shows the images obtained with the *corner* technique. The threshold is slightly higher, [8,10], and the noise has been reduced.

In Fig. 12 it can be seen how the ZC method cuts off almost all the noise yielding a sufficiently high threshold: [18,27]. This technique exhibits the desired behavior for this situation.

5.2. Second IR sequence

In the source frames of Fig. 13 a pedestrian is walking into the scene. The fog is still present.

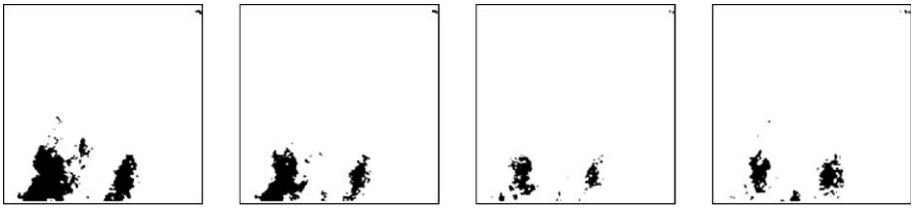
The Kapur method in Fig. 14 correctly separates the pedestrian blob from the noise generated by the fog bank. The blob is broken in two regions, but this was the best result achievable since a lower threshold would have let pass the fog noise.

The *corner* technique produces the images shown in Fig. 15. The threshold is too low, [9,10]. The fog noise has completely occluded the pedestrian blob.

In Fig. 16 the ZC technique gives a result very similar to Kapur's. The threshold, ranging from 22 to 30, is correct because the noise has been successfully cut off and the blob—although separated in two regions—is clearly visible.

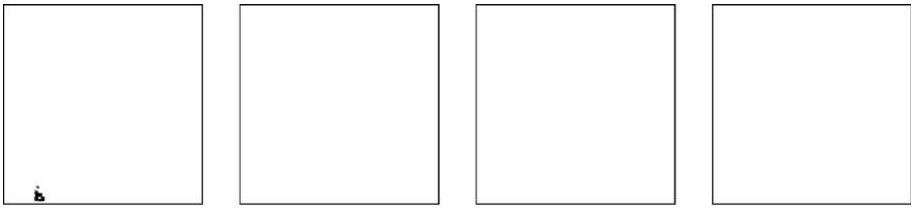
5.3. Third IR sequence

In Fig. 17 a third set of infrared source frames is shown. This time the pedestrian is far away,



(a) $th = 8$ (b) $th = 9$ (c) $th = 10$ (d) $th = 9$

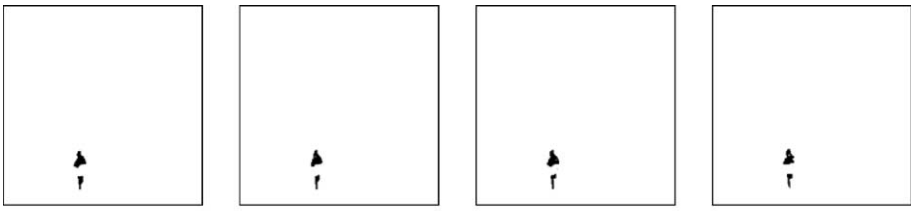
Fig. 11. *Corner* method applied to the frames of Fig. 9.



(a) $th = 27$ (b) $th = 23$ (c) $th = 20$ (d) $th = 18$

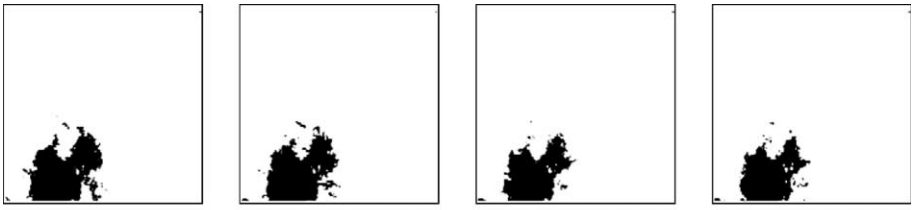
Fig. 12. ZC method applied to the frames of Fig. 9.

Fig. 13. Source frames from the 2nd IR sequence.



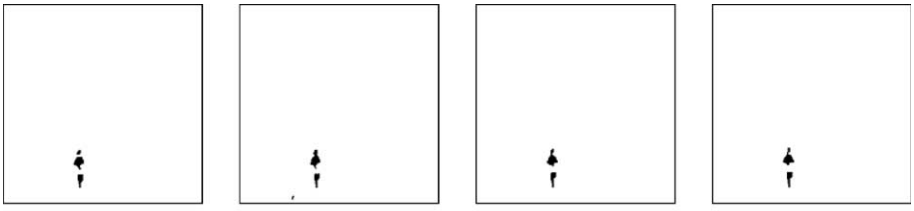
(a) $th = 17$ (b) $th = 17$ (c) $th = 17$ (d) $th = 18$

Fig. 14. Kapur method applied to the frames of Fig. 13.



(a) $th = 9$ (b) $th = 9$ (c) $th = 10$ (d) $th = 10$

Fig. 15. *Corner* method applied to the frames of Fig. 13.



(a) $th = 26$ (b) $th = 23$ (c) $th = 30$ (d) $th = 22$

Fig. 16. *ZC* method applied to the frames of Fig. 13.

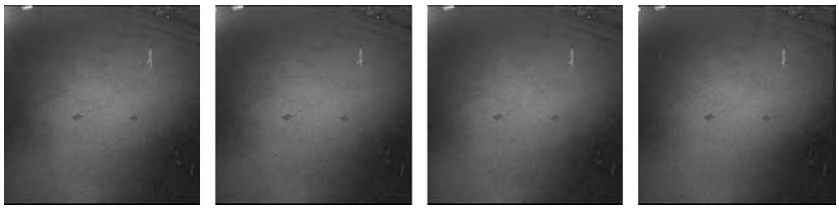
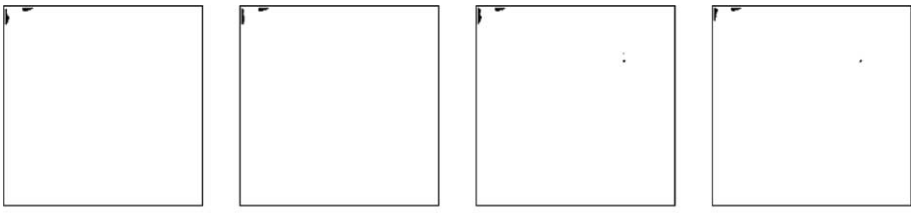


Fig. 17. Source frames from the 3rd IR sequence.

leaving the scene. Additional noise is generated by car headlights in the top-left corner of the image.

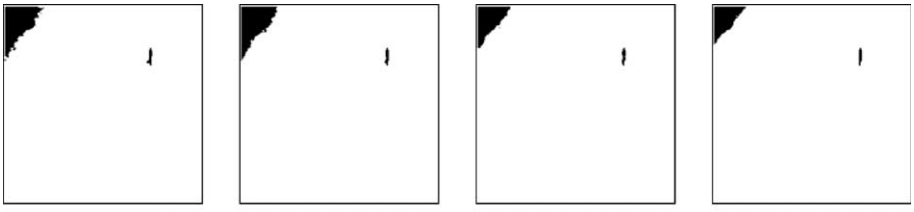
The threshold produced by the Kapur method in Fig. 18 is too high: [46,54]. The noise has been almost eliminated, but the pedestrian blob has been cut off as well.

The *corner* technique in Fig. 19 gives a satisfactory result: the pedestrian is detected, but the



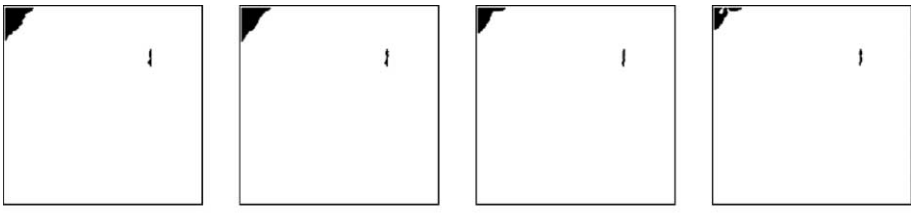
(a) $th = 54$ (b) $th = 55$ (c) $th = 46$ (d) $th = 48$

Fig. 18. Kapur method applied to the frames of Fig. 17.



(a) $th = 19$ (b) $th = 19$ (c) $th = 19$ (d) $th = 19$

Fig. 19. *Corner* method applied to the frames of Fig. 17.



(a) $th = 28$ (b) $th = 27$ (c) $th = 29$ (d) $th = 29$

Fig. 20. ZC method applied to the frames of Fig. 17.

threshold is low (19 for all the four frames) and the noise in the top-left corner (caused by headlights) can generate false alarms.

The result in Fig. 20 is good. The ZC technique produces a threshold higher than the corner's, [27,29]. The pedestrian is still visible and the noise in the top-left part of the image has been reduced (roughly 50% less).

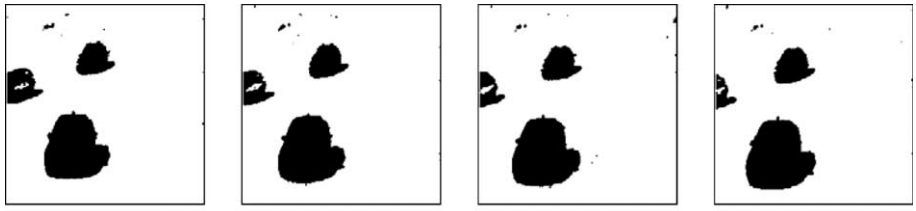
In the noisy infrared video-sequence, the overall behavior of the ZC technique has been certainly good and gave the best results if compared to the other methods.

5.4. Highway sequence

The three algorithms were also tested in an application for traffic monitoring. Fig. 21 shows three vehicles approaching a highway toll gate. The quality of the video signal is good. Noise is primarily generated by illumination variations.

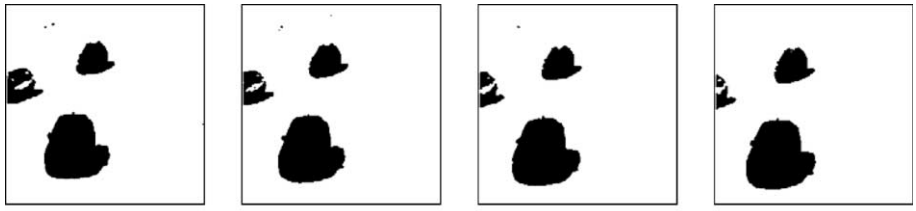
The images in Figs. 22–24 have been obtained by binarizing the difference maps with the thresholds produced by the three algorithms. No additional filtering has been performed (e.g., erosion, median filter, etc.).

Fig. 21. Source frames for the highway sequence.



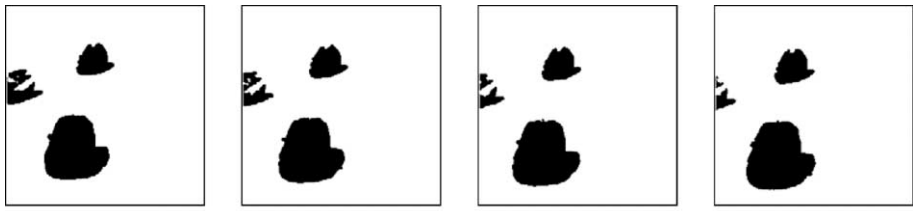
(a) $th = 6$ (b) $th = 6$ (c) $th = 6$ (d) $th = 6$

Fig. 22. Kapur method applied to the frames of Fig. 21.



(a) $th = 8$ (b) $th = 8$ (c) $th = 9$ (d) $th = 9$

Fig. 23. *Corner* method applied to the frames of Fig. 21.



(a) $th = 14$ (b) $th = 14$ (c) $th = 15$ (d) $th = 16$

Fig. 24. *ZC* method applied to the frames of Fig. 21.

The Kapur method Fig. 22 gives too a low threshold (6). An additional filtering process would be necessary to remove the spurious spots generated by the noise.

The *corner* technique does a little better (Fig. 23): the threshold is slightly higher, [8,9], and the result is a more filtered image.

The best result is achieved by the ZC method (Fig. 24). All the noise has been removed and no additional filtering is necessary.

6. Conclusions

A fast implementation of the stable Euler number thresholding method has been presented. The computational complexity has been considerably reduced from $O(hwg)$ to $O(hw)$, where h, w are the image's height and width and g is the number of possible thresholds. Only a single raster scan of the difference map is now required to determine all possible Euler numbers. The algorithm is therefore suitable for real-time applications.

A heuristic technique, called ZC, has also been devised to find out the optimal threshold value. Although further research is necessary to verify the robustness of the heuristic, it performed well in a video surveillance application for outdoor environments and in a system for traffic monitoring.

ZC has proven to be very effective for the tested conditions and to be considerably resilient to noise.

The experiments gave good results: real-time performance and good quality thresholding were achieved.

References

Collins, R., Lipton, A., Fujiyoshi, H., Kanade, T., 2001. A system for video surveillance and monitoring. Proc. IEEE 89, 1456–1477.

Foresti, G., 1998. Object detection and tracking in time-varying and badly illuminated outdoor environments. Opt. Eng. 37 (9), 2550–2564.

Foresti, G.L., 1999. Real-time detection of multiple moving objects in complex image sequences. Internat. J. Imaging Systems Technol. 10, 305–317.

Friel, N., Molchanov, I., 1999. A new thresholding technique based on random sets. Pattern Recognition 32 (9), 1507–1517.

Gray, S., 1971. Local properties of binary images in two dimensions. IEEE Trans. Comput. 20, 551–561.

Kapur, J., Sahoo, P., Wong, A., 1985. A new method for grey-level picture thresholding using the entropy of the histogram. Comput. Vision Graphics Image Process. 29, 273–285.

Otsu, N., 1979. A threshold selection method from gray-level histograms. IEEE Trans. System Man Cybernet. SMC 9 (1), 62–66.

Rosin, P., 1998. Thresholding for change detection. In: ICCV98 Proceedings.

Rosin, P., Ellis, T., 1995. Image difference threshold strategies and shadow detection. In: Proceedings of the 6th British Machine Vision Conference, pp. 347–356.

Sezgin, M., Taaltin, R., 2000. A new dichotomization technique to multilevel thresholding devoted to inspection applications. Pattern Recognition Lett. 21 (2), 151–161.

Tsai, W.-H., 1985. Moment preserving thresholding: A new approach. Comput. Vision Graphics Image Process. 29, 377–393.