An Improved Real-time Blob Detection for Visual Surveillance

Thanh Binh Nguyen
School of Electronics Engineering
University of Soongsil
Seoul, South Korea

Sun Tae Chung
School of Electronics Engineering
University of Soongsil
Seoul, South Korea

***Abstract*—Blob detection is an essential ingredient process in some computer applications such as intelligent visual surveillance. However, previous blob detection algorithms are still computationally heavy so that supporting real-time multi-channel intelligent visual surveillance in a workstation or even one-channel real-time visual surveillance in an embedded system using those turns out prohibitively difficult. Blob detection in visual surveillance goes through several processing steps: foreground mask extraction, foreground mask correction, and blob segmentation through connected component labeling. Foreground mask correction necessary for a precise detection is usually accomplished using morphological operations like opening and closing. Morphological operations are computationally expensive and moreover, they are difficult to run in parallel with connected component labeling routine since they need quite different type of processing from what connected component labeling does. In this paper, we first develop a fast and precise foreground mask correction method utilizing on neighbor pixel checking which is also employed in connected component labeling so that it can be incorporated into and run together with connected component labeling routine. Through experiments, it is verified that our proposed blob detection algorithm based on the foreground mask correction method developed in this paper shows better processing speed and more precise blob detection.**

Keywords-blob detection; connected component labeling; visual surveillance; union-find; morphology

I. INTRODUCTION

Blob detection is an important process in various computer vision applications such as intelligent visual surveillance [1][2][3]. Intelligent visual surveillance requires analyzing interesting object's activities. Interesting objects are derived from blob detection. It is now well known that the performance of intelligent visual surveillance heavily depends on how precisely and rapidly interesting objects are detected [1].

Blob detection is usually processed through several steps: foreground mask extraction, foreground mask correction, and blob segmentation through connected component labeling [1][2][3]. Foreground mask is a set of extracted foreground pixels. During foreground mask extraction, some foreground pixels may not be extracted so that holes can appear in the foreground mask. Also, some background pixels may be mistakenly extracted as foreground pixels which usually appear as jutted pixels or isolated small pixel regions in the foreground mask. This foreground mask needs to be corrected through filling up holes and eliminating jutted pixels or isolated pixels. Foreground mask correction has usually been accomplished using morphological operations like opening and closing as a preprocessing step [4][5][6]. However, morphological operations are computational expensive. Moreover, they are difficult to be computed in parallel with connected component labeling operations since they need quite different type of processing from what connected component labeling does.

In this paper, we propose an improved real-time blob detection algorithm for visual surveillance which is faster and more precise than the conventional ones using morphological operations. We first develop a blob correction method which can be efficiently processed and incorporated into the connected component labeling procedure. NFPP (Neighbor Foreground Pixel Propagation), a blob correction method developed in this paper, utilizes 8-neighbors checking which is also employed in connected component labeling procedure (CCL) so that NFPP can be incorporated into CCL processing routine. Then, foreground mask correction is accomplished while CCL is processing, which can save the processing time much more compared with conventional blob correction methods using morphology operations, which need to be processed separately before CCL processing. Through the experiments, it is shown that the proposed blob detection algorithm based on NFPP performs better than the conventional blob detection algorithm with respect to the processing time and the preciseness in blob detection.

The rest of the paper is organized as follows. Section 2 introduces blob detection, morphology and connected component labeling, which are the basic technical background necessary for understanding the works of the paper. Section 3 describes our proposed blob detection algorithm. There, we first explain our developed blob correction method, NFPP which is essential in achieving faster and more precise blob detection processing. Experiment results are discussed in Section 4, and finally the conclusion is presented in Section 5.

II. BACKGROUNDS

A. Blob Detection in Motion Analysis of Visual Surveillance

All Blob detection algorithms in intelligent visual surveillance usually go through the following steps [1][2][3]: foreground mask extraction, foreground mask correction, blob

978-1-4244-4131-0/09/$25.00 ©2009 IEEE

segmentation through connected component labeling, and region calculation as shown in Fig. 1.

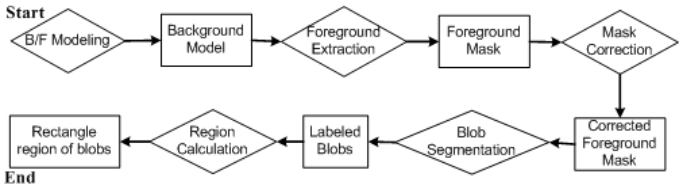

Figure 1. General work flow of blob detection

Fig. 2 shows example images related with blob detection steps.

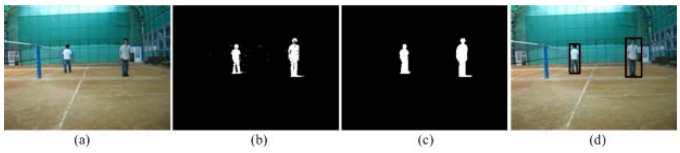

Figure 2. (a) Original frame image, (b) Foreground binary image, (c) Foreground image after correction, and (d) Labeled foreground regions

In this process, foreground masks means the set of all foreground pixels, and are represented as binary images where white (value 1) pixels are foreground pixels and black (value 0) pixels are background pixels. Foreground pixels are extracted from the incoming current image frame through matching it with the background model [2]. For foreground mask correction which may need to fill holes or eliminate jutted pixels or a small pixel region, morphological operations such as opening and closing are usually employed as preprocessing procedures. To detect interesting objects, one needs to segment blobs that are defined as a set of all connected foreground pixels. After separate blobs are segmented, rectangular region enclosing each blob tightly are calculated for later processing like object tracking.

B. Morphology

The basic idea in binary morphology is to probe an image with a simple, pre-defined shape, drawing conclusions on how this shape fits or misses the shapes in the image [4][5][6]. This simple 'probe' is called an structuring element, and is itself a binary image. The popular structure elements include disk, square, and cross-shaped element of 3x3 sizes.

The basic operations of binary morphology are dilation, erosion, closing, and opening. A dilation operation enlarges a region, while erosion makes it smaller. A closing operation, defined as an operation of applying erosion after dilation, can close up internal holes in a region and eliminate 'bays' along the boundary and a opening, defined as an operation of dilation after erosion, can get rid of small portions of the region that jut out from the boundary into the background region.

In blob detection processing, some foreground pixels are not extracted and omitted in foreground masks, and a hole can be observed in the foreground regions. On the other hand, some background pixels are mistakenly extracted as foreground pixels and appear as jutted pixels from foreground region boundaries or isolated small pixel regions in the foreground masks. The holes can be made to be filled up by closing operation and the jutted pixels or small isolated pixel regions are eliminated by opening operation in the preprocessing stage in the blob detection.

C. Connected Component Labeling with Union-Find Structure (CCLUF)

Connected components labeling scans an image and groups its pixels into components based on pixel connectivity. In a foreground mask image, the connectivity exists between foregrounds pixel by 8-neighborhood relation of the pixel.

Once all connected components are determined, each pixel is assigned the label of the connected component it belongs to. Extracting and labeling of various disjoint and connected components in an image is central of many automated computer vision applications including intelligent visual surveillance. Many algorithms have been proposed to deal with connected component labeling problem [7][8][9][10][11][12]. The promising efficient algorithms usually employ two-pass procedures utilizing union-find structure [10][11][12][13] which is also adopted in this paper. The following explanation is based on [13].

The union-find algorithm dynamically constructs and manipulates the equivalence classes efficiently by tree structures. The addition of this data structure is a useful improvement to the classical algorithm. The purpose of the union-find data structure is to store a collection of disjoint sets and to efficiently implement the operations of union (merging two sets into one) and find (determining which set a particular element is in). Each set is stored as a tree structure in which a node of the tree represents a label and points to its one parent node. This is accomplished with only a vector array PARENT whose subscripts are the set of possible labels and whose values are the labels of the parent nodes. A parent value of zero means that this node is the root of the tree. The find procedure is a given a label X and the parent array PARENT. It merely follows the parent pointers up the tree to find the label of the root node of the tree that X is in. The union procedure is given two labels X and Y and the parent array PARENT. It modifies the structure (if necessary) to merge the set containing X with the set containing Y. It starts at labels X and Y and follows the parent pointers up the tree until it reaches the roots of the two sets. If the roots are not the same, one label is made the parent of the other.

Fig. 3 shows the union-find structure for two sets of labels.

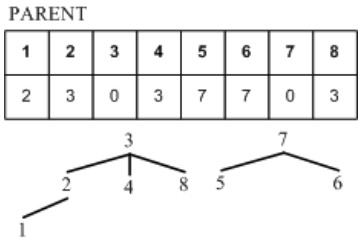

Figure 3. The union-find structure for two sets of labels

In the below, we briefly explain the two-pass connected component labeling with union-find structure.

In the first pass, the algorithm scans through the binary image from left to right and row by row. If a foreground pixel

is met during scanning, the algorithm checks whether it has a foreground pixel neighbor among already scanned pixels or not. If not, the foreground pixel is assigned the next larger label than the assigned largest label. If it has, then it is assigned the smallest one among those of the scanned foreground pixel neighbors, and each such equivalence class found is entered in the union-find structure. At the end of the first pass, each equivalence class has been completely determined and has a unique label, which is the root of its tree in the union-find structure. The second pass through the image then performs a replacement of the temporary label into a final label which is done by assigning to each pixel the label of its equivalence class.

Fig. 4 shows an example input binary image where f means foreground pixel.

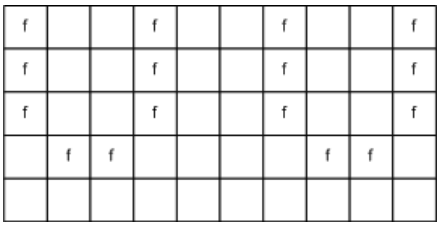

Figure 4. Foreground mask where f means foreground pixel

Fig. 5 shows the results of the 1st pass.

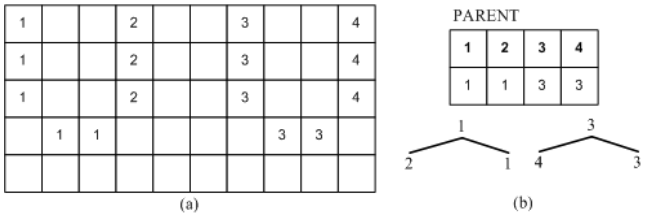

Figure 5. The result of the 1st pass

Fig. 6 shows the final labeling after 2nd pass.

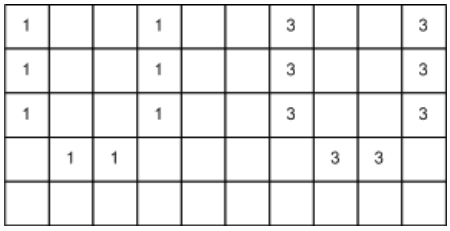

Figure 6. Final result

III. THE PROPOSED BLOB DETECTION ALGORITHM: CLNF APPROACH

From the background reviewed in Section 2, one can easily understand that it is difficult to process morphological operations like closing and opening in parallel with connected component labeling algorithm. Usual way to do processing both operations is to process sequentially: morphological operations, first and then connected component labeling, next. But, morphological operations are computationally expensive.

Thus, we first develop a foreground mask correction method which can be incorporated into a connected component labeling routine so that mask correction operations like filling up holes or eliminating jutted pixels or isolated pixels can be processed while connected component labeling routine is processing.

A. Neighbor Foreground Pixel Propagation (NFPP)

A hole in the foreground region is highly likely to be a set of missed foreground pixels region and jutted pixels or isolated pixels in the foreground mask is also highly likely to be wrongly extracted as foreground pixels. One can observe in the foreground mask that a pixel belonging to a hole is likely to have more foreground pixel neighbors and on the other hand, a jutted pixel or an isolated pixel in a foreground mask is likely to have less foreground pixel neighbors.

Here, we define the foreground neighborhood probability of a pixel X, P(X) as the ratio of the number of the foreground neighbor pixels over the number of all neighbors. In this paper, we use 8-neighborhoods as in Fig. 7.

Figure 7. 8-Neighbors

From this observation, we develop a foreground mask correction method, "Neighbor Foreground Pixel Propagation (NFPP)".

Neighbor Foreground Pixel Propagation method

In binary mask image where pixels have two states, value 1 (foreground pixel) and value 0 (background pixel), Pixel state is propagated to its neighbors to the right and below, depending on its foreground neighborhood probability. That is, when one scans the binary image from left to right in row by row, a state of a pixel X will be changed depending on its foreground neighborhood probability as (1).

$$state(X) = \begin{cases} 1; & P(X) \geq \frac{1}{2} \\ 0; & others \end{cases} \qquad (1)$$

Processing of NFPP method is graphically illustrated Fig. 8.

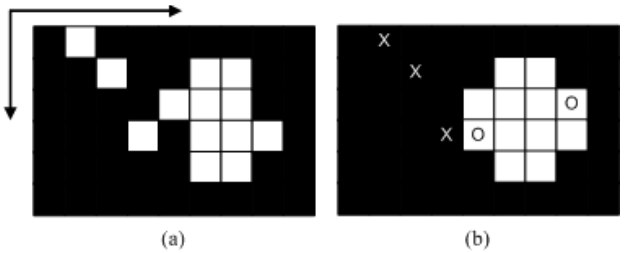

Figure 8. (a) Foreground mask, (b) The result after propagation where X are pixels that were changed to background (Black), O are pixels that were turned into foreground (White)

Fig. 9 shows the result of applying NFPP method to a foreground mask which was actually obtained using Gaussian Mixture Background Model [2][3]. One can see that the blob after NFPP looks clearer and plumper than before.

Figure 9. (a) Foreground mask, (b) Result after NFPP

NFPP is simple but shows a good result in blob correction. In addition to a good blob correction property, another more important property of NFPP method is that it can be processed while CCL algorithm is being processed since both NFPP and CCL utilize pixel neighbor checking. This property can save computational time a lot in blob detection. Based on this property, we propose CLNF (CCLUF with NFPP) approach in blob detection, which leads to a faster and more precise blob detection than conventional blob detection algorithms.

B. CLNF Approach (CCLUF with NFPP)

1) Outline: In this paper, we propose a blob detection algorithm, CLNF which combines CCLUF (Connected Component Labeling with Union and Find Structure) and NFPP (Neighbor Foreground Pixel propagation). Work flow of CLNF approach is shown in Fig. 10 with two passes (scans).

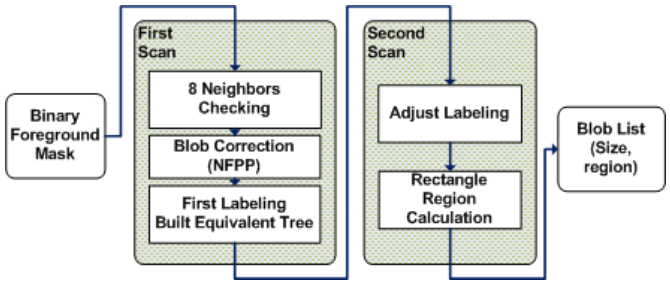

Figure 10. Work flow of CLNF

While first labeling and making Union-Find structure is doing in the 1st pass (scan), a blob correction using NFPP is also processed. In the 2nd pass (scan), temporarily assigned labels of the pixels are adjusted into a correct label using the accomplished union-find structure and finally rectangle blob regions are calculated using the labeled connected component.

2) Pseudo Code: For clearer explanation of our proposed blob detection algorithm, CLNF, the pseudo codes of CLNF are sketched in the below.

Pseudo code for 1st scan

```
-ParentLabel: the label value should be assigned to current pixel
-ChildLabel: the label which will be replaced by ParentLabel
-Image[]: binary image
-Labels[]: label map of Image[]
-LabelTree[]: PARENT array in Union-Find Structure explained in II.C
-NewLabel: =0;                          // the new value of label

With each Image[X]
Start
    With each of 8 neighbors of a pixel Image[X]
    Start
        - Count Foreground Pixel
        - Assign ParentLabel as the smallest label among Image[X]'s left or
        above foreground pixel neighbors' labels
        - In case Image[X] has 2 different neighbor labels, assign
        ChildLabel as the bigger label
    End
    If foreground neighborhood probability >=1/2
    Then
        -Image[X] = Foreground;          // Foreground = 1;
    Else
        -Image[X] = Background;          // Background = 0;
    End
    If Image[X] is foreground
    Then
        If ParentLabel and ChildLabel are not assigned
        Then
            - Increase NewLabel by 1;
            - Labels[X] = NewLabel;
        Else
            - Recalculate new ParentLabel (parent of ChildLabel) rely on
            LabelTree[];
            - LabelTree[ChildLabel] = ParentLabel;
            - Labels[X] = ParentLabel;
        End
    End
End
```

Pseudo code for 2nd scan

```
-SizeBlobs[]: list of blob size
-BlobRectangles[]: list of blob rectangle regions

With each Labels[X]
Start
        -Relabeling Labels[X] by its parent note in tree LabelTree[];
        -Increase SizeBlobs[Labels[X]] by 1;
        -Calculate BlobRectangles[Labels[X]];
End
```

IV. EXPERIMENTAL RESULTS

A. Comparison between NFPP and Morphological operations about preciseness

We compare NFPP with morphology with respect to blob correction ability. Fig. 11 shows the result that NFPP is simpler but more effective in foreground blob correction than morphological operations.

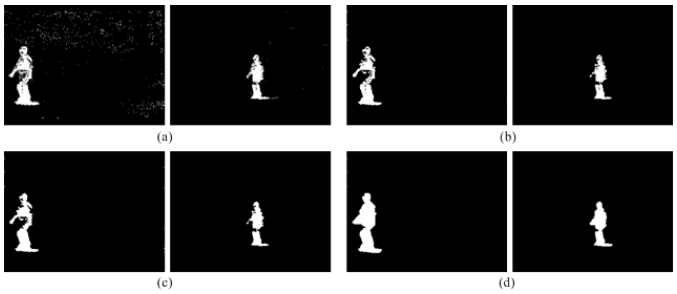

Figure 11. (a) Foreground mask, (b) The result after morphology (Open), (c) After morphology (Open + Close), (d) After NFPP

Foreground mask in Fig. 11(a) was obtained using Gaussian Mixture Background Model, and morphology results in Fig. 11(b) and 11(c) are processed using common 3x3 kernels. Because the precise details of the effect of these operators can be determined by used structuring element, one may also get a similar clear blob as NFPP in this test case if one chooses a kernel with more suitable parameters and size. But, one may have to pay higher computational cost for that. On the contrary, NFPP can be applied naturally without predefined parameters.

B. Comparison about processing time

In order to compare the computational cost between the conventional blob detection algorithm using morphological operations (CONV, hereafter) and the proposed blob detection algorithm, CLNF, we did two experiments.

First experiment calculates the processing time for blob detection in one frame, foreground mask of Fig. 11(a) for both CONV and CLNF. And the second experiment calculates the processing time for continuous blob detection in movies. Both experiments have been done using a Windows PC with a 3GHz Pentium 4 Core of 3GHz and 3GB main memory. And the resolution of each image frame is 320x240 and sizes of movie1 and movie2 are 4.67 MB (221 frames) and 245MB (1117 frames). Table I and Table II show the results for the first and second experiment, respectively.

TABLE I. COMPARISON OF THE PROCESSING TIME BETWEEN THE CONVENTIONAL BLOB DETECTION ALGORITHM (CONV) AND THE PROPOSED ONE (CLNF) FOR ONE FRAME OF FIG. 11(A)

	time(sec)
CONV	0.017714
CLNF	0.003722

TABLE II. COMPARISON OF THE PROCESSING TIME BETWEEN THE CONVENTIONAL BLOB DETECTION ALGORITHM (CONV) AND THE PROPOSED ONE (CLNF) FOR TWO MOVIES

	movie1 (sec)	**movie2 (sec)**
CONV	4.297988	21.053498
CLNF	0.874097	4.111587

The results of Table I and Table II certainly show that the proposed blob detection algorithm is faster than the conventional blob detection algorithm using morphological operations for foreground mask correction.

V. CONCLUSION

In this paper, we proposed a fast and precise real-time blob detection algorithm for visual surveillance. Blob detection is to segment separated but clear blob regions for which foreground mask correction and connected component labeling procedures are required. The main idea of the proposed blob detection algorithm is to develop a blob correction method which can be efficiently processed together with the connected component labeling algorithm. NFPP, developed in this paper as a foreground mask correction method, utilizes 8-neighbors checking which is also employed in CCL so that NFPP can be incorporated into CCL processing, which can save the processing time much more than when using a conventional foreground mask correction method using morphological operations like opening and closing. The experiment results show the effectiveness of the proposed blob detection algorithm with respect to the processing time and the preciseness in blob detection.

ACKNOWLEDGMENT

The authors would like to acknowledge the following support for the accomplishment of this work: Soongsil University Research Fund and BK21 of Korea.

REFERENCES

[1] T. Chen, H. Haussecker, A. Bovyrin, R. Belenov, K. Rodyushkin, A. Kuranov, V. Eruhimov, "Computer Vision Workload Analysis: Case Study of Video Surveillance Systems", Intel Technology Journal, May 2005.

[2] Z. Zivkovic, "Improved adaptive Gaussian mixture model for background subtraction", ICPR 2004, pp. 28-31.

[3] C. Stauffer C, W.E.L Grimson, "Adaptive background mixture models for real-time tracking", IEEE CVPR, 1999, pp. 244 - 252.

[4] P. Soille, Morphological Image Analysis: Principles and Applications, Springer-Verlag Telos, Berlin, 2003.

[5] R. C. Gonzalez, R. E. Woods, and S. L. Eddins, Digital Image Processing using MATLAB, Prentice Hall, Upper Saddle River, New Jersey, 2003.

[6] E. R. Dougherty and R. A. Lotufo, Hands-on Morphological Image Processing, SPIE Press, Bellingham, 2003.

[7] A. Rosenfeld, J.L. Pfaltz, "Sequential Operations in Digital Picture Processing", JACM, 1966, pp. 417-494.

[8] M. Onoe, Real Time Parallel Computing Image Analysis, Plenum Press, New York, 1981.

[9] R. Lumia, L. Shapiro, and O. Zuniga, "A New Connected Components Algorithm for Virtual Memory Computers", Computer Vision and Image Processing, 1983, pp. 287-300.

[10] C. Fiorio and J. Gustedt, "Two linear time Union-Find strategies for image processing", Theoretical Computer Science, Elsevier Science Publishers Ltd, 1996, pp. 165-181.

[11] R. E. Tarjan, "Efficiency of a Good But Not Linear Set Union Algorithm", Journal of the ACM, ACM, 1975, pp. 215-225.

[12] M. B. Dillencourt, H. Samet, and M. Tamminen, "A General Approach to Connected-Component Labeling for Arbitrary Image Representations", JACM, ACM, 1992, pp. 253-280.

[13] L. Shapiro and G. Stockman, Computer Vision, Prentice Hall, Upper Saddle River, New Jersey, 2001.