Motion Detection in Real-Time Video Streams Using Moved Frame Background

Amina A. Dawood **Balasim A. Hussean**
University of Babylon

1- Abstract

There are many approaches for motion detection in a real-time video stream. All of them are based on comparing of the current video frame with one from the previous frames or with background.

Another approach is to compare the current frame not with the previous one but with the first frame in the video sequence. So, if there were no objects in the initial frame, comparison of the current frame with the first one will give us the whole moving object independently of its motion speed. But, the approach has a big disadvantage - what will happen, if there was, for example, a person on the first frame, but then he is gone? Yes, we'll always have motion detected on the place, where the person was. Of course, we can renew the initial frame sometimes, but still it will not give us good results in the cases where we can not guarantee that the first frame will contain only static background. But, there can be an inverse situation.

The most efficient algorithms are based on building the so called background of the scene and comparing each current frame with the background. There are many approaches to build the scene, but most of them are too complex. Our approach for building the background is to get the first frame of the video stream as the background frame. And then we'll always compare the current frame with the background one. Our approach is to "move" the background frame to the current frame on the specified amount (we used 1 level per frame). We move the background frame slightly in the direction of the current frame - we are changing colors of pixels in the background frame by one level per frame. To build the background we use the Morph filter, because the implementation of this filter is more efficient, so the filter produce better performance. The idea of the filter is to preserve specified percentage of the source filter and to add missing percentage from overlay image.

الخلاصة

توجد هنالك عدة طرق لتحسس الحركة في تدفقات فيديو الوقت الحقيقي. تعتمد جميعها على مقارنة اطار الفيديو الحالي مع اطار من الاطارت السابقة او مع خلفية الصورة.

طريقة اخرى، هي بمقارنة الاطار الحالي، ليس مع الاطار السابق وانما مع الاطار الاول في السلسة. لذا، اذا لم تكن هنالك اي كيانات في الاطار الاولي، فان مقارنة الاطار الحالي مع الاطار الاول سوف تعطينا الكائن المتحرك باكمله بغض النضر عن سرعته. لكن هذه الطريقة لها مشكلة كبيرة – ماذا سوف يحدث، مثلا، اذا كان هنالك شخص في الاطار الاول، لكنه اختفى؟ في هذه الحالة سوف يكون لدينا دائما تحسس بحركة في ذلك المكان الذي كان يتواجد به الشخص. يمكننا بالطبع تحديث الاطار الابتدائي بعض المرات، لكن النتيجة تبقى غير جيدة في الحالات التي لانستطيع ضمان ان الاطار الاولي سوف يحتوي فقط على كيانات الخلفية الثابتة. لكن، يمكن ان توجد هنالك حالة معاكسة.

تعتمد الخوارزميات الاكثر فعالية على بناء ما يسمى بخلفية المشهد ومقارنة كل اطار حالي مع هذه الخلفية. توجد هنالط عدة طرق لبناء المشهد، لكن اغلبها معقدة جدا. الطريقة المقترحة لبناء الخلفية هي باخذ الاطار الاول من تدفق الفيديو كاطار خلفية. ومقارنة الاطار الحالي مع اطار الخلفية، لكن يتم "تحريك" اطار الخلفية باتجاه الاطار الحالي بمقدار معين (قمنا باستخدام مستوى واحد لكل اطار). القيام بتحريك اطار الخلفية بصورة قليلة باتجاه الاطار الحالي – اي القيام بتغيير الوان بكسلات اطار الخلفية بمستوى واحد لكل اطار. لبناء الخلفية قمنا باستخدام Morph filter، لان تطبيق هذا الفلتر يكون ذو اداء اكبر. فكرة الفلتر هي بالحفاظ على نسبة معينة من الفلتر المصدر واضافة النسبة المتبقية من الصورة الفوقية.

2- Introduction

In the last few years, visual surveillance has become one of the most active research areas in computer vision, especially due to the growing importance of visual surveillance for security purposes. Visual surveillance is a general framework that groups a number of different computer vision tasks aiming to detect, track, and classify objects of interest from image sequences, and on the next level to understand and describe these objects behavior. The ultimate goal in designing smart visual

surveillance systems is to replace the existing passive surveillance and to remove, or at least, minimize the need for a human observer to monitor and analyze the visual data. [Mohamed, 2006].

The increasing availability of video sensors and high performance video processing hardware opens up exciting possibilities for tackling many video understanding problems [Alan, 2000]. It is important to develop robust real-time video understanding techniques which can process the large amounts of data attainable. In our paper we take the motion detection problem, we assumed an input video stream from a web cam or any other type of digital video cameras.

3- The detection process

The detection process is accomplished by:

- **Background modeling**, which resembled by a morph filter combines the background as an overlay image, and the current frame to decrease the difference with the background, which can be taken as updating the background [Hu, 2004].
- **Temporal variance**, which is accomplished by a Connected Component Labeling Algorithm [Jung ,2001]. That takes connected labeled pixels, which assembles a region in the image, and combines them into object.

The final step is to count those object and calculate a rectangle to surround their area and draw that rectangle in the screen.

By conquest processing of the incoming video frames in real-time we end with a moving triangles around the detected objects motion.

4- Motion Detection Algorithm

4-1 Getting the initial background image

As a first step we prepare the background to be the first frame we received, as that we now have no motion at all, we further process the background by applying a Grayscale filter and a Pixellate Filter. The pixellete filter here used to reduce the pixels count and emphasize the over all color distribution of the image; then we extract the image dimensions to use in further processing. So we now have the initial background image [Collins, 2000] (Fig. 1).

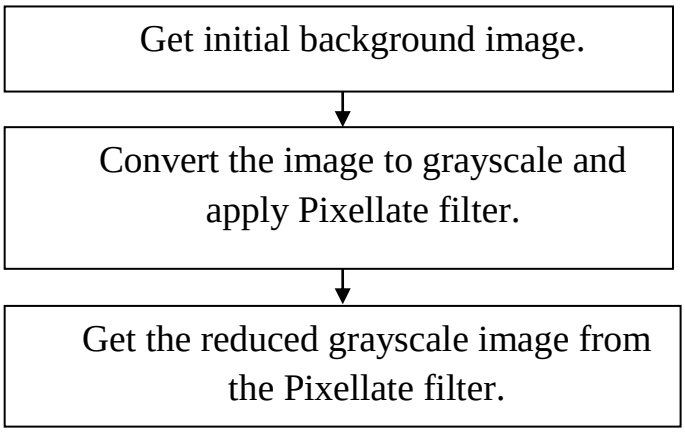


Fig 1. Setting the initial background image.

4-2 Updating the background image

From the steps in (4-1) we get a frame and called it the current frame, we first apply the same filters as we did with the background image. That means we make the current frame as the same as the background image in structure and format.

Update the background image by moving the pixels intensity towards the pixels intensity of the current frame by one level, to decrease diffirence with overlay image - source image is moved towards overlay image. The update equation is defined in the next way:

*Result = src + Min(Abs(ovr - src), step) * Sign(ovr - src)*
[Mohamed, 2006]

Where :
Result is the updatd background image, which will be the background for the next frame.
Src is the curent frame image.
Over is the curent background image.
Step defines the maximum amount of changes per pixel in the source image.

The bigger is step size value the more resulting image will look like overlay image. For example, in the case if step size is equal to 255, the resulting image will be equal to overlay image regardless of source image's pixel values. In the case if step size is set to 1, the resulting image will very little differ from the source image. But, in the case if the filter is applied repeatedly to the resulting image again and again, it will become equal to overlay image in maximum 255 iterations. In our case we repeatedly applies the filter to the updated background overlayed on the curent frame, which in result will be counted as applying the filter for the first time.

The value, of step per pixel, we take is 1, because if we increase the moving steps, we make the background image more similar to the current frame, with this small amount of movement we prevent the background image from becoming less sensitive to the changes of the upcoming frames, and also reduce the number of iterations that will be made on the background and the current frame, which yields more speed in processing the frames which is a crucial criteria in real-time processing.

4-3 Blob extraction and counting

Detection of connected components between pixels in binary images is a fundamental step in segmentation of an image objects and regions, or *blob.* Each blob is assigned a unique label to separate it from other blobs. All the pixels within a blob of spatially connected 1's are assigned the same label. It can be used to establish boundaries of objects, components of regions, and to count the number of blobs in an image [Gonzalez ,1992]. Its applications can be found in automatic inspection, optical character recognition, robotic vision, etc. [Ronson, 1954].

The original algorithm was developed by Rosenfeld and Pfaltz [Rosenfeld, 1966] in 1966. It performs two passes through the image. In the first pass, the image is processed from left to right and top to bottom to generate labels for each pixel and all of the equivalent labels are stored in a pair of arrays. In the second pass, each label is replaced by the label assigned to its equivalence class. Several papers [Lumia, 1983], [Lumia, 1983], [Manohar, 1989] pointed out the problems in the second pass for large images because the equivalence arrays can become unacceptably large [Lumia, 1983]. The way in which label equivalences are resolved can have a dramatic effect upon the running, time of this algorithm.

Modifications include one proposed by Haralick that does not use air equivalence array [Jung, 2001] and a small equivalence table by Lumia, Shapiro, and Zinup [Lumia, 1983] that is reinitialized for each line. The latter paper makes comparison runs between these three algorithms. Another solution uses a bracket table [Yang, 1989] to associate equivalent groups. Its pushdown stack data structure that implemented in hardware. Our approach computes the connected components of binary image in real-time without any hardware support. Instead it applies the power and efficiency of the divide-and-conquer technique.

4-3-1 The Basics

A pixel p at coordinate (x, y) has four direct neighbors, $N_4(p)$ and four diagonal neighbors, $N_D(p)$. Eight-neighbors, $N_8(p)$of pixel p consist of the union of $N_4(p)$ and $N_D(p)$ [Mohamed, 2006].

To establish connectivity for pixels p and q can be considered :

1- 4-connectivity-connected if q is in $N_4(p)$;
2- 8-connectivity-connected if q is in $N_8(p)$;
3- m-connectivity-connected if q is in $N_4(P)$, or if q is in $N_D(P)$ and $N_4(p) \cap N_4(q)=\phi$;

4-3-2 A Connected Component Labeling Algorithm

The labeling algorithm is described below based on 8-connectivity.

Step 1*: Initial labeling.*

Scan the image pixel by pixel from left to right and top to bottom. Let p denote the current pixel in the scanning process and 4-nbr denote four neighbor pixels in N, NW. NE and W direction of p. If p is 0, move on to the next scanning position If p is 1 and all values in 4-nbrs are 0, assign a new label to p. If only one value in 4-nbrs is not 0, assign its values to p. If two or more values in 4-nbrs are not 0, assign one of the labels to p and mark labels in 4-nbrs as equivalent.

Step 2: *Resolve equivalences (This is developed as fallows).*

The equivalent relations are expressed as a binary matrix. For example, if label 1 is equivalent to 2, label 3 is equivalent to 4, label 4 is equivalent to 5, and label 1 is, equivalent to 6 then the manta: L is that shown in Figure 2.a. Equivalence relations satisfy reflexivity, symmetry and transitive [Gonzales, 1992]. To add reflexivity in matrix L. all main diagonals are set to 1. To obtain transitive closure the Floyd_Warshall (F-W) algorithm [Mohamed, 2006] is *used.*

```
  for j = 1 to n
  for i = 1 to n
if L[i,j] = 1 then
for k = 1 to n
L[I,k] = L[1, k] OR L[j,.k];
```

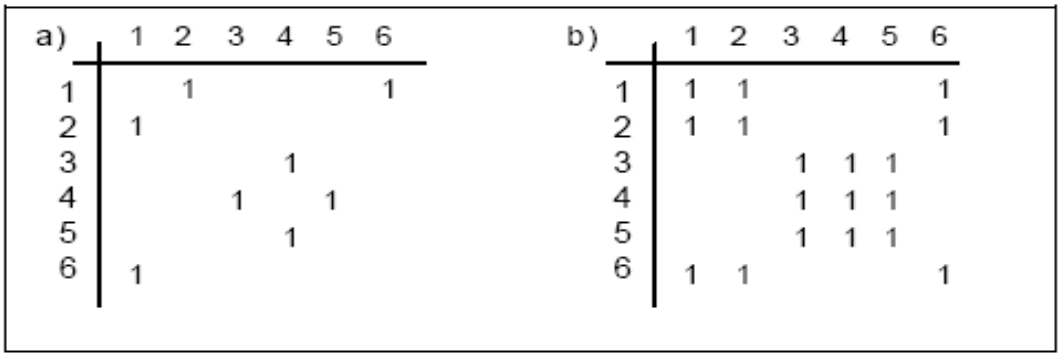


Fig 2. Equivalence relations in terms of binary matrix.
a) Matrix before applying the F-W algorithm. b) Matrix after applying reflexivity and the F-W algorithm.

After applying reflexivity and the F-W algorithm, the matrix L is that shown in Fig. 2.b. This algorithm, can be performed in $O(n^3)$ ***OR*** operations. After calculating the transitive closure, each label value is recalculated to resolve equivalences. The image is scanned again and each label is replaced by the label assigned to its equivalence class.

4-3-3 A Fast Connected Component Labeling Algorithm

The main idea in this algorithm is to divide the image into NxM small regions (we use NxN here for simplicity). The large equivalence array is the main bottleneck in the original algorithm, but NxN small equivalence arrays can be found in greatly reduced time. Figure 3 shows that an image divided into 3x3 small regions for labeling independently. Then we connect each region with its neighbor regions to generate the actual label within the entire image. We use NxN pointers Label_Llst[i] to point to arrays that maintain the *global labels* with respect to the entire image. Label_List[i] points to the array for Region[i] where each array element is the global label within the entire image and the index for each array element is the local label within Region[i]. Memory allocation for each array pointed to by Label_List[i] can be done dynamically according to the maximum local label in Region[i]. Figure 4 depicts these lists. The example of Figure 5 shows that local label 1, 2 and 6 are equivalent and their global label within the entire image is 8; local label 3, 4, and 5 are equivalent and their global label is 9. The Total_Index equals 7 at the end of Region[i-1], which is kept in the list at index 0.

5- The *Fast Labeling algorithm*

Our *fast labeling algorithm* (based on 8-connectivity) is described below. The other connectivity differs only in its neighboring checking.

5-1 *Fast Labeling algorithm*

Region[1]	Region[2]	Region[3]
Region[4]	Region[5]	Region[6]
Region[6]	Region[7]	Region[8]

Fig 3. Division of original image into 3x3 regions.

Step 1- Divide the given image into NxN small regions and set Total_ Index = 0
Step 2: For each region i = 1 to NxN
1- apply *Step 1*of the original algorithm in Section 3-3-2;
2- allocate memory for the array pointed to by Label_List[i] as maximum no. of labels for Region[i];
3- use F-W algorithm in Section 3-3-2 to resolve the equivalences within Region[i].
4- for j=1 to size of an array for Region[i] do
Label_List[i][j] = Total_Index + Ibl
// lbl is a label to its equivalence class after equiv. resolution (see Figure 5).
5- Total_index= Total_index + maximum{lbl}
6- if (i > 1) then call *Merge(i):*
// to update labels in bordering area between regions.
Step3: For each region i = l toNxN do
scan image in Region[i] from left to right, top to bottom and replace all local label value k with Label_List[i][k];

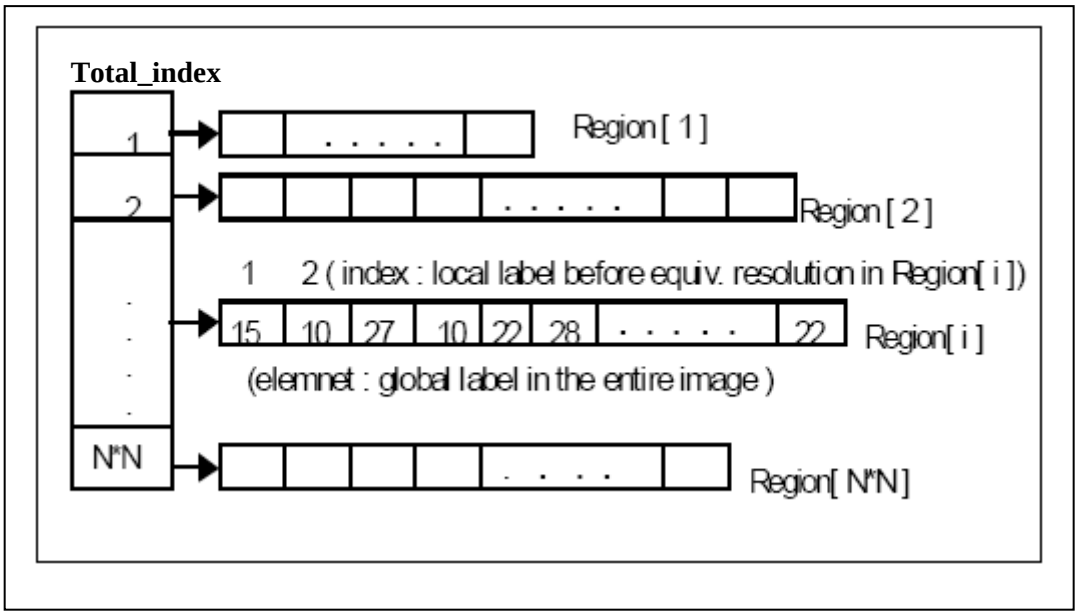


Fig 4. The Label_List structure.

5-2 The Merge(*i*) Function (resolve equivalences of pixels in bordering area between regions).
Step 1: select first pixel p in Region[i];
If (label (p)>0) then
for each pixel q in N_8 (p) intersects other regions
//see figure 6.a
if (label(q) > 0) then
call Resolve_Equivalence(p,q,i);
Step 2: for each pixel p in the first column in Region[i]
if (label (p) > 0) then
for each pixel q in N_8 (p) intersects Region [i-1]
//see Figure 6.b
if (label (q) > 0) then
call Resolve_Equivalence(p,q,i);
Step 3: for each pixel p in the first row in Region[i]

if label(p) > 0 then
for each pixel q in $N_8(p)$ intersects Region [i-N]
//see Figure 6-c
if (label (q) > 0) then
call Resolve_Equivalence(p,q,i);

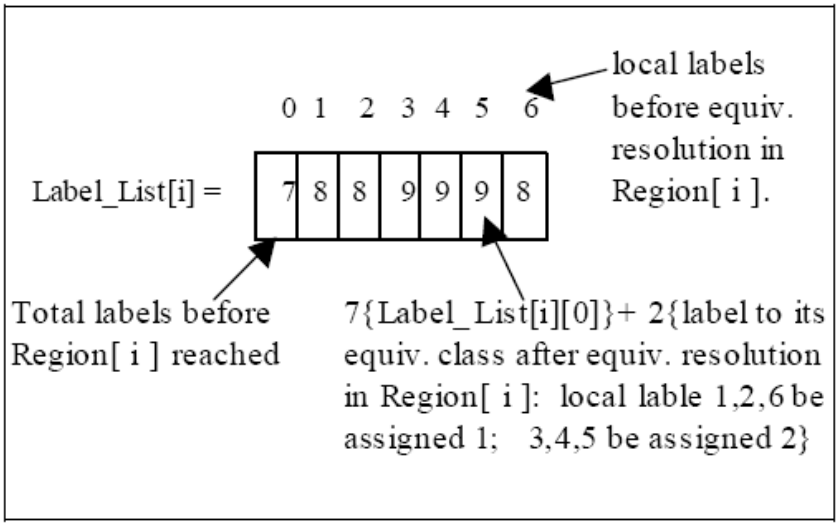


Fig 5. The Label_List[i] example.

5-3 The Resolve_Equivalence(*p,q,i*) Function.

Step1*:* Index1 = Label_List[region no. of q][label(q)];
Index2 = Label_list[i][label(p)] ;
if(Index1 not equal to Index2) then
do Step 2.

Step2*:* Small_Lb1 = min{indexl, index2};
Large_Lbl = max{indexl, index2};
for k = 1 to i do
for j = l to size of any array for Region[k]
if (Label_List[k][j] > Large_Lbl) then
Label_List[k][j] = Label_List[k][j] -1;
else if (Label_List[k][j] = large_Lbl) then
Label_List[k][j] = small_lb1;
Total_Index = Total_Index-1;

5-4 Final Steps

As the previous steps completed, the result is a binary image with objects represents the moving objects in the current frame, so we now:

For each object[i] in current frame
$rect[i].Max_x$ = max_x(object[i])
$rect[i].Min_x$ = min_x(object[i])
$rect[i].Max_y$ = max_y(object[i])
$rect[i].Min_y$ = min_y(object[i])
draw_rect(rect[i])

The above processes will effect all detected objects in the frame and drawing rectangles over them, which result in drawing attention over any movement happing in the frame, as a result, for the whole input video stream, these rectangles will move with the object that they surround.

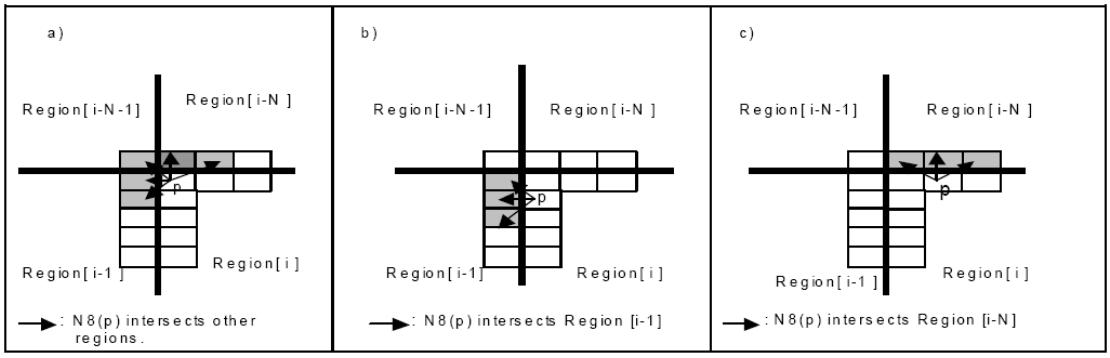


Fig 6. Merge taken place in three ways at Region[i]. a) Merge at the first pixel in Region[i] b) Merge at pixels in the first column in Region[i] and the last column in Region[i-1] c) Merge at pixels in the first row in Region[i] and the last row in Region[i-N].

6- Conclusions

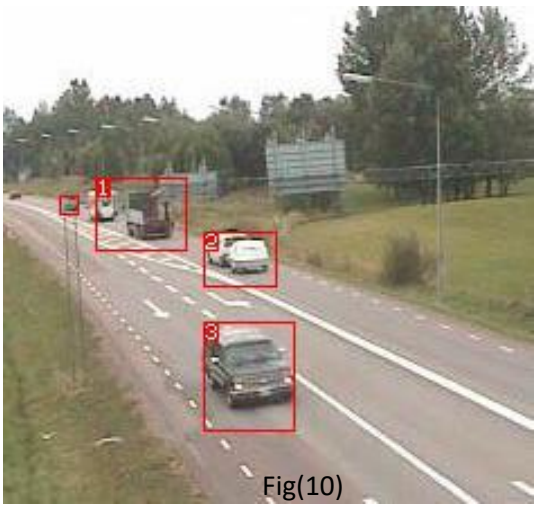
In live video streams any process on the stream must be as fast, and accurate as possible. So we do here develop the accuracy in the moving background, to, always, update the background and make it move towards the movement of the frame pixels by morphing the frame with the background, which slightly change the background pixels from the current frame to prepare it to be the new background for the next frame. The fast Blob algorithm will detect the movement in a rapid fashion by eliminating the unnecessary loops over the image and converging the data structures involved in it, we eliminate hare these steps to increase the speed of detection and to be able to process as many as the input stream delivers frames from the scene we observe.

In the simplest way, it takes the difference between two frames and highlight that difference, this method is poor in object recognition because the difference don't emphasize the object shape as in Fig(7). The second best approach is based on edge detection, in spite of its good shape recognition, Fig(8), but it's lake in speed and takes a lot of hardware resources. So by a little enhancement in the previous algorithm by adding a pixellete filter we obtain a good representation of the object and a fast performance, due to the reduced number of pixels blocks to process as seen by fig(9). After all we eliminate the bounders and replace them by a surrounding rectangle applied to each detected object, because the idea is to detect and track motion not to recognize object shapes.

Fig(7)

Fig(8)

Fig(9)



Fig(10)

References

Alan, J. Lipton, 2000, Moving target classification and tracking from real-time video, The Robotics Institute., Carnegie Melon University.

Collins R. T., 2000, A system for video surveillance and monitoring, Carnegie Mellon Univ., Pittsburgh.

Hu W., 2004, A survey on visual surveillance of object motion and behaviors, IEEE Trans. on systems, man, and cybernetics part C: Applications and Reviews, Vol.34, NO.3, August.

Jung-Me Park, 2001, Fast Connected Component Labeling Algorithm Using A Divide and Conquer Technique., Computer Science Dept. University of Alabama.

Lumia, R., Shapiro, L. ,1983, A New Connected Components Algorithm for Virtual Memory Computers, Computer Vision, Graphics, and Image Processing. 22. 287-300.

Lumia R., 1983, A New Three-dimensional connected components Algorithm, Computer Vision, Graphics, and Image Processing, 23. 207-217.

Mohamed, F., 2006, Integrated Motion Detection and Tracking for Visual Surveillance Center for Automation Research (CfAR) UoM.

Manohar, M., 1989, Ramapriyan, H.,K., Connected Component Labeling of Binary Image on a Mesh connected Massively Parallel Processor, Vision, Computer Graphics and Image processing, 45, 133149.

Ronson, 1954, Connected components in Binary images: The Detection Problem, Research Stitches Press.

Rosenfeld, A., Pfaltz, J.,L., 1966, Sequential Operations in digital Processing, JACM. 13. 471-494.

Woods, R.,E., 1992; Gonzalez, R., C.; Addison Wesley, Digital Image Processing.

Yang, X., D., 1989, An Improved Algorithm for Labeling Connected Components in a Binary Image, TR 89-981, March.