Proceedings of the IEEE
International Conference on Information and Automation
Ningbo, China, August 2016

An Efficient Watershed Algorithm For Preprocessed Binary Image

Qingyi Gu
Institute of Automation
Chinese Academy of Sciences
Beijing, China
qingyi.gu@ia.ac.cn

Jun Chen, Tadayoshi Aoyama, and Idaku Ishii
Graduate School of Engineering
Hiroshima University
Higashi-Hiroshima, Hiroshima, Japan
{*j-chen, aoyama, iishii*}*@rbotics.hiroshima-u.ac.jp*

***Abstract*— Over-segmentation of a grayscale image is a typical problem in existing watershed algorithms. To overcome this problem, preprocessing is mainly applied to the grayscale image before performing the watershed transformation to generate a gradient or binary image. In this paper, a novel watershed algorithm based on the concept of connected-component labeling and chain code is proposed, which generates a final label map in just four scans of a preprocessed binary image. The low memory consumption, low complexity, and simple data structure of the algorithm make it highly suitable for hardware implementation. Evaluation results showed that the proposed algorithm decreases the average running time by more than 39% without loss of accuracy.**

***Index Terms*— watershed transform, image segmentation, connected component labeling, chain code**

I. Introduction

Watershed-based image segmentation algorithms typically produce an over-segmentation of the image. Preprocessing and post-processing of the input image are thus performed to overcome this problem. Preprocessing is mainly applied to the image before performing the watershed segmentation to generate an edge image. As shown in Fig. 1, preprocessing involves three stages: 1) noise reduction, 2) gradient calculation, and 3) thresholding of the gradient image. The classical approach for obtaining an edge image is to threshold the gradient image. The main objective of this third stage is to reduce the over-segmentation of an image as much as possible. The optimal threshold value can be selected by an iterative method for connected components based on watershed image segmentation. The number of segments can be controlled by varying the threshold value, which allows to obtain an optimum segmentation result.

In recent years, many watershed algorithms have been developed for the segmentation of grayscale images. In this paper, an efficient watershed algorithm for preprocessing of binary images is described based on the concept of connected-component labeling and chain code. The proposed method is an image complexity-invariant watershed algorithm, which can perform watershed transformation on an input binary image in just four scans.

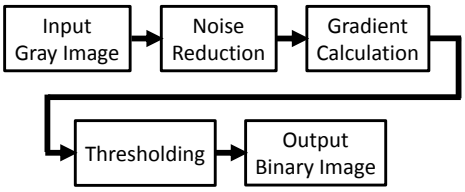

Fig. 1. Block diagram of pre-processing.

II. Related Works

In the past 20 years, two kinds of watershed algorithms have been developed.

1. Flooding watershed algorithms

In the traditional flooding-based approach, the image is considered as a topographic surface. Imagine water flooding from the bottom and going into holes at each regional minimum at a constant rate. When there is a potential for rising water from different catchment basins to merge with water from nearby catchment basins, it is necessary to build a dam to prevent the merging of the water streams. Eventually, the flood water will reach a point where only the top of the dam is visible above the water line. These contiguous dam boundaries represent the watershed lines. There are several algorithms that implement this technique [1], [2], [3], [4], and they are even implemented in hardware [5], [6]. A common problem with flooding-based watershed algorithms is the one-pixel width of the watershed lines, which tends to slow down the merging process that usually follows the calculation of the watershed transformation [7], [8], [9].

2. Rainfalling watershed algorithms

In the rainfalling watershed algorithm, raindrops fall on the mountain (topographic surface) and move in a descending direction because of gravity until they reach the local minimum surface. The algorithm tracks the path of the raindrops at each point on the surface toward the local minimum, if the raindrops pass through these points or fall on any of these points. All the points make a segment when raindrops related to them flow downward to the same deepest location. If a point has more than one path toward different

978-1-5090-4102-2/16/$31.00 ©2016 IEEE

Authorized licensed use limited to: University of New South Wales. Downloaded on July 09,2020 at 03:32:27 UTC from IEEE Xplore. Restrictions apply.

steepest surfaces, then it can be allocated to any one of the local minimum surfaces. This method has been implemented in [10] and improved in [11], [12], [13], [14], [15]. There are also other studies that focus on the hardware adaptation of this algorithm [16], and there are those that make use of parallel processing [11], [12], [13], [14], [15]. Compared to the flooding watershed algorithm, the techniques based on rainfalling simulation do not generate any watershed line because they label all the points as belonging to a basin.

Bieniek and Moga developed an efficient watershed algorithm based on connected components [11]. A disadvantage of this algorithm when implemented in hardware is the need to use a first-in-first-out (FIFO) queue and a last-in-first-out stack, which introduces memory access overhead. This algorithm was slightly modified by Maruyama and Trieu [6] to simplify the memory access. However, the number of scans is not constant but depends on the complexity of the input image. In recent years, algorithms using just four scans [11], [12] and two scans [13], [15] have been proposed. However, these algorithms use several FIFO queues and the depth of these queues depends on the complexity of the input image. As a result, the processing time increases greatly for large-sized images. Moreover, the memory consumption of FIFO queues is unpredictable.

III. Algorithm

In this paper, to meet the need for a faster way of performing watershed transformation on a preprocessed binary image, we proposed a novel method that uses the concept of rainfalling watershed, which involves four steps. In steps 1 and 2 of our algorithm, an improved connected-component labeling algorithm is used to assign numeric labels for all catchment basin pixels. In steps 2 to 4, two concepts of the rainfalling watershed algorithm are used to assign numeric labels to all non-minima plateau pixels.

A. Improved connected-component labeling

Recently, an efficient labeling algorithm suitable for hardware implementation with low memory consumption has been reported [17], [18]. However, this algorithm has two problems when detecting different regional minima. The first problem is that it uses a four-connected-component labeling algorithm. The second problem is that the algorithm is designed only for feature extraction; it does not generate a final label map. To solve the first problem, we slightly modified and improved the algorithm to use eight-connected-component labeling. In the original algorithm, the labeling function can only handle two neighbors ($left$ and up) of the $focus$ pixels. In the eight-connected-component labeling algorithm, four pixels should be considered for the $focus$ pixel in the labeling function. As shown in Fig. 2, we designed a strategy to generate $newLeft$ and $newUp$ for the $focus$ pixels in the labeling function. The improved eight-connected-component labeling algorithm can generate a label map in two raster scans. In step 1 of our algorithm, a tentative label map is generated. The final label assignment of each catchment basin is combined in step 2 to solve the second problem of the algorithm.

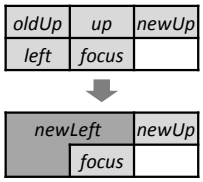

Fig. 2. New concept of eight connected component labeling.

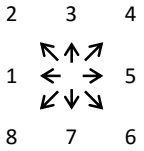

Fig. 3. Chain code.

B. Two concepts of the rainfalling watershed algorithm

The most difficult task in rainfalling watershed is how to assign labels to non-minima plateaus [13]. In this paper, we combined the concept of geodesic distance and chain code to solve this problem using only two raster scans based on the tentatively labeled image generated in step 1.

1. Geodesic distance The geodesic distance is the shortest distance to a regional minimum, which is used to solve the problem of non-minima plateaus in images [6]. In steps 2 and 3 of our proposed algorithm, the geodesic distance is used to solve the aforementioned problem in a binary image.

2. Chain code The chain code concept is used for the efficient computation of the shortest paths [13], as shown in Fig. 3. In this algorithm, the pixels of an image are accessed in an irregular way to allow the assignment of a chain code; this greatly increases the size of the FIFO queues as the image size increases. As a result, it is not suitable for hardware implementation. When generating a chain code in our algorithm, no FIFO queues are used and only two raster scans are required. The data structure is simple, and the complexity is independent of the input image.

C. Algorithm description

In our proposed algorithm, f is the input preprocessed binary image; l is the segmented, labeled image; p represents a pixel; $f(p)$ represents the gray level value of p; n is the neighbor pixel of p; and $f(n)$ represents the gray level value of the respective neighbor pixel. To save memory space, l is reused in all steps of our algorithm. In step 1, l is used to store a tentative label map of the input binary image f. After this step, all pixels belonging to the catchment basins (=1) are assigned a numeric label; the other pixels belonging

Authorized licensed use limited to: University of New South Wales. Downloaded on July 09,2020 at 03:32:27 UTC from IEEE Xplore. Restrictions apply.

to non-minima plateaus are assigned as background pixel (BG=0). The relationship between the labels is stored in a relabel map r, whose size is *LMAX*($= W \times H/4$), where W and H denote the width and height of the input image, respectively. *LMAX* defines the maximum possible labels in an image when an eight-connected-component labeling is performed. In this step, an array $prevLine$ with size $W+1$ is used to save memory in the labeling process [17]. In step 2, l is used to store a tentative geodesic distance for all non-minima plateaus. In this step, the final numeric label of each catchment basin is also updated in l by searching the relabeling table r generated in step 1. In step 3, l is used to store the chain code of non-minima plateaus to the catchment basins. In this step, the local geodesic distance of p is used to determine the chain code of p. To save memory, the array $prevLine$ used in step 1 is reused in this step to store the geodesic distance of the previous line. A variable pv is used to store the geodesic distance value of the previous pixel to the right of p. In step 4, a final label map is generated according to the chain code of non-minima plateaus. A FIFO queue $qDescending$ is used to store the steepest descending paths of non-minima plateaus, whose size is *VMAX*. *VMAX* ($= max(W, H)$) defines the maximum geodesic distance in an image. *BASIN* (=0) defines the value of the catchment basins in the input image f. *BG* (=0) defines the non-minima plateau regions in l after step 1. *UNVISITED* (=0) defines the unvisited non-minima plateau pixels in steps 2 and 3. The value range of l was well designed to avoid confusion. The range of the numeric label is [*VMAX*, *LMAX*] in all steps. The range of the geodesic distance is [1, *VMAX*), in steps 2 and 3. In step 3, the chain code of the non-minima plateaus of l is updated, whose range is $[1, 8]$.

Three functions are used in our algorithm. The $labeling()$ function is used to generate a numeric label for the focus (current) pixel. The input of this function includes the value of the focus pixel and the label of the left and up pixels. The $getChainCode()$ function is used to generate a chain code from point $pFrom$ to point pTo, where $pFrom$ and pTo are neighbors. The $getNextPoint()$ function is used to get the next point by using $pFrom$ and its chain code.

The first two scans are forward raster scans. The third scan is a backward raster scan. The last scan is a forward raster scan with a FIFO queue operation. All the non-minima plateau pixels are accessed only once. The algorithm is implemented as follows:

Step 0: Initialization

Define *BASIN* $=$ *BG* $=$ *UNVISITED* $= 0$
  VMAX $= max(W, H)$
  LMAX $= W \times H/4$
$next\_no =$ *VMAX*
r(*BG*) $=$ *BG* // size is *LMAX*
Create queue: $qDescending$ // size is *VMAX*
Initial all elements of $prevLine$ to *BG* // size is $W+1$

Step 1: Connected component labeling

```
Loop (y from 0 to H − 1)
  newLeft = up = r(prev_line(0))
  Loop (x from 0 to W − 1)
    newUp = r(prevLine(x + 1))
    focus = labeling(f(p), newLeft, newUp)
    If focus ≠ BG Then
      newLeft = focus
      up = (newUp ≠ BG) ? focus : newUp
    Else
      newLeft = (up ≠ BG) ? up : newUp
      up = newUp
    End If
    prevLine(x) = focus
  End Loop
  Loop (x from W − 1 to 0)        // relabeling in reversed order
    If prevLine(x) ≠ BG Then
      finalLabel = r(r(prevLine(x)))
      r(prevLine(x)) = finalLabel
      l(p) = prevLine(x) = finalLabel
    Else
      l(p) = BG
    End If
  End Loop
End Loop
```

Step 2: Generate tentative geodesic distance for non-minima plateaus

```
Forward Scan (p)
  If l(p) = UNVISITED Then
    minV = VMAX
    ∀n belonging to N(p){
      If l(n) ≥ VMAX Then          // local minimum existed
        l(p) = 1      // edge pixel, distance to local minimum is 1
      Else If l(n) ≠ UNVISITED and l(n) < minV Then
        minV = l(n)                // inner pixel
      End If
    }
    If l(p) ≠ 1 and minV ≠ VMAX Then
      l(p) = minV + 1
    End If
  Else
    While (l(p) ≠ relabel[l(p)])   // find final label of p
      l(p) = relabel[l(p)]
    End While
  End If
End Forward Scan (p)
```

Step 3: Generate chain code for non-minima plateaus

Initial pv and all elements of $prevLine$ to *UNVISITED*
Loop (y from $H-1$ to 0)

Authorized licensed use limited to: University of New South Wales. Downloaded on July 09,2020 at 03:32:27 UTC from IEEE Xplore. Restrictions apply.

```
  Loop (x from W − 1 to 0)
    If (l(p) < VMAX) Then
      minV = VMAX
      Loop (y′ from −1 to 1)
        Loop (x′ from −1 to 1)
          // position of p is (x, y), n is (x + x′, y + y′)
          If l(n) ≥ VMAX Then
            min = n
          End If
          // l′(n) is the geodesic distance of p's neighbourhood n
          If y′ = 1 Then
            l′(n) = prevLine(x + x′)
          Else If y′ = 0 and x′ = 1 Then
            l′(n) = pv
          Else
            l′(n) = l(n)
          End If
          If l′(n) ≠ UNVISITED and l′(n) < minV Then
            minV = l′(n)
            minv = n
          End If
        End Loop
      End Loop
      prevLine((x + 1) mod W) = pv
      pv = l(p)
      If l(p) = 1 and ∃min Then    // point to catchment basin
        l(p) = getChainCode(p, min)
      Else If ∃minv Then    // point to minimum neighbourhood
        pv = minV + 1
        l(p) = getChainCode(p, minv)
      End If
    End If
  End Loop
End Loop
```

Step 4: Label assignment of non-minima plateaus to catchment basins

```
Scan (p)
  p′ = p
  While (l(p′) ∈ [1, 8])    // it is not a minimum (chain code)
    qDescending.put(p′)
    p′ = getNextPoint(p′, l(p′))
  End While
  While (qDescending ≠ ∅)
    p″ = qDescending.get()
    l(p″) = l(p′) // assign label of catchment basin to chain elements
  End While
End Scan (p)
```

Function *labeling*()

```
In parameters: focusPixel, left, up
Out parameter: focus
Function:
  If focusPixel = BASIN Then
    If left = BG and up = BG Then
      focus = next_no
      next_no = next_no + 1
      r(focus) = focus
    Else If left ≠ BG Then
      If up ≠ BG and left ≠ up Then
        dst_no = (left > up) ? up : left
        src_no = (left > up) ? left : up
        focus = dst_no
        r(src_no) = dst_no
      Else
        focus = left
      End If
    Else
      focus = up
    End If
  Else
    focus = BG
  End If
```

Function *getChainCode*()

```
In parameters: pFrom, pTo
Out parameter: chainCode
Function:
  If pFrom.y = pTo.y Then
    chainCode = (pFrom.x > pTo.x) ? 1 : 5
  Else If pFrom.x = pTo.x Then
    chainCode = (pFrom.y > pTo.y) ? 3 : 7
  Else If pFrom.x > pTo.x Then
    chainCode = (pFrom.y > pTo.y) ? 2 : 8
  Else
    chainCode = (pFrom.y > pTo.y) ? 4 : 6
  End If
```

Function *getNextPoint*()

```
In parameters: pFrom, chainCode
Out parameter: pTo
Function:
  Switch (chainCode)
    Case 1 : pTo.x = pFrom.x − 1, pTo.y = pFrom.y
    Case 2 : pTo.x = pFrom.x − 1, pTo.y = pFrom.y − 1
    Case 3 : pTo.x = pFrom.x    , pTo.y = pFrom.y − 1
    Case 4 : pTo.x = pFrom.x + 1, pTo.y = pFrom.y − 1
    Case 5 : pTo.x = pFrom.x + 1, pTo.y = pFrom.y
    Case 6 : pTo.x = pFrom.x + 1, pTo.y = pFrom.y + 1
    Case 7 : pTo.x = pFrom.x    , pTo.y = pFrom.y + 1
    Case 8 : pTo.x = pFrom.x − 1, pTo.y = pFrom.y + 1
  End Switch
```


Authorized licensed use limited to: University of New South Wales. Downloaded on July 09,2020 at 03:32:27 UTC from IEEE Xplore. Restrictions apply.

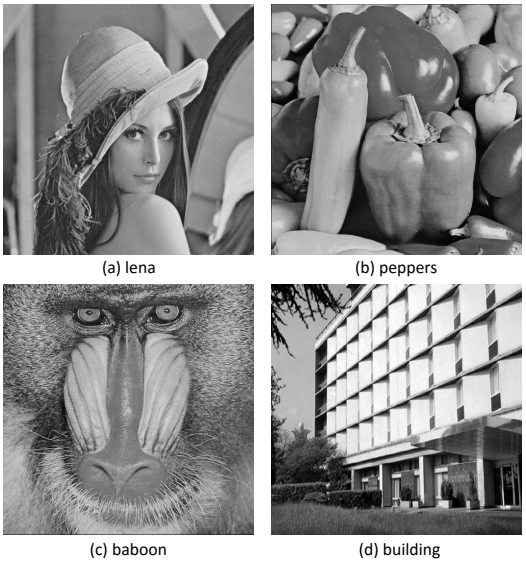

Fig. 4. Source images for speed and accuracy evaluation.

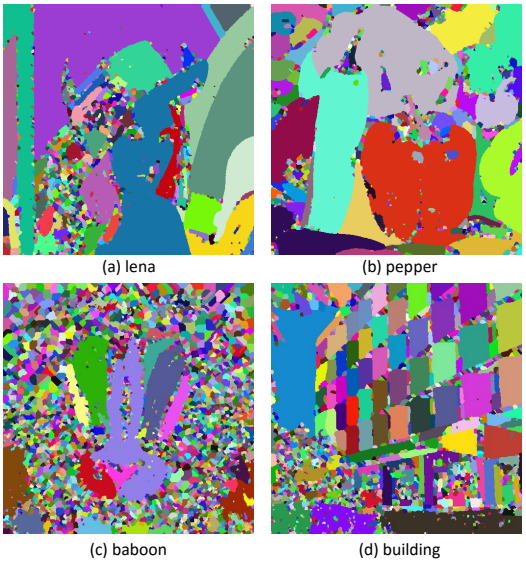

Fig. 5. Results for speed and accuracy evaluation.

IV. Evaluation

We evaluated the computation cost and accuracy of our proposed algorithm on a PC (CPU: Intel i7 3930K 3.8 GHz; memory: 8 GB DDR3; motherboard: ASUS P9X79-E WS), using the images in Fig. 4. The running time and accuracy were compared with those obtained using the algorithms previously developed by Maruyama and Trieu [6] and by Victor et al. [13]. For the accuracy evaluation, we confirmed that the number of catchment basins obtained for each algorithm was always the same even when using different-sized images.

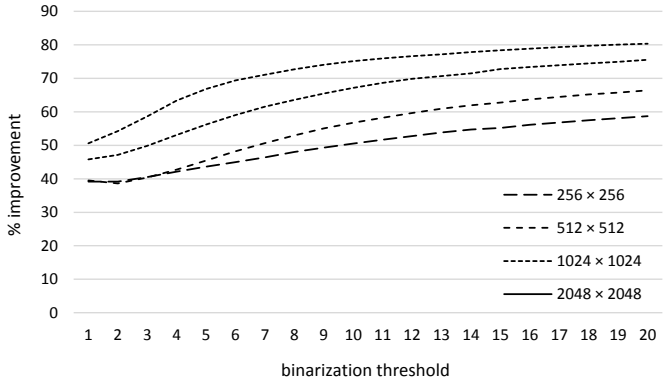

Fig. 6. Speed improvement in different binarization threshold for peppers image.

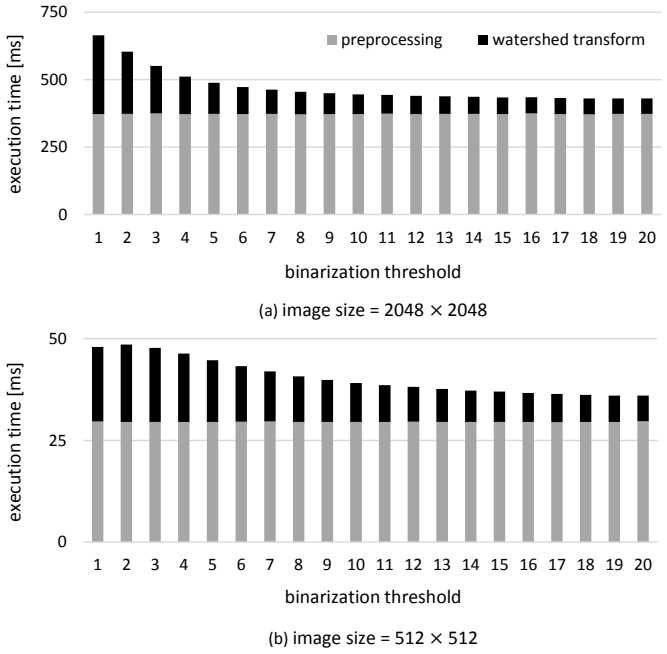

Fig. 7. Execution time in different binarization threshold for peppers image.

For the speed evaluation, all illustrated execution times were the average value of 1000 runs on the PC. Here, we show the execution time of our algorithm in detail by selecting the peppers image in Fig. 4. Fig. 6 shows the speed improvement of our algorithm compared with the other two algorithms using different-sized images and different binarization threshold values. There were two tendencies noted: 1) speed improvement increased with the increase in binarization threshold value; 2) speed improvement also increased with the increase in image size. The reason for the first tendency is the decrease in the number of non-minima plateaus ($O(N_{NMP})$) along with the increase in binarization threshold values, which decreased the complexity of our algorithm. The reason for the second tendency is the simple

Authorized licensed use limited to: University of New South Wales. Downloaded on July 09,2020 at 03:32:27 UTC from IEEE Xplore. Restrictions apply.

data structure and constant scan times of our algorithm. Compared to the algorithm of Victor et al., our algorithm does not require managing huge queues and their operations, which becomes a crucial problem especially for huge-sized images. Compared to Maruyama and Trieu's algorithm, our algorithm can perform watershed transformation in just four scans, independent of image complexity. We had confirmed that the speed evaluation results of all the other images in Fig. 4 had the same tendency as that of the peppers image.

We also evaluated the execution time of our algorithm by considering the preprocessing stage. Fig. 7(a) shows the total running time of our algorithm for the peppers image with a resolution of 2048×2048. The total running time decreased from 664.28 ms to 430.13 ms along with increasing binarization threshold value from 1 to 20. The major time cost was the preprocessing stage, which was about 372 ms. The running time of our proposed watershed algorithm decreased sharply from 292.18 ms to 56.81 ms along with increasing binarization threshold value from 1 to 20. Fig. 7(b) shows the same tendency as Fig. 7(a). It can be seen that our algorithm can work at 25 fps for a 512×512 image when the binarization threshold value is 10. Fig. 5 shows the segmentation results of our algorithm with preprocessing for the four images in Fig. 4.

V. Conclusion

In this paper, we introduced a fast watershed algorithm for preprocessing of binary images to solve the problem of over-segmentation. Our algorithm employs the connected-component labeling algorithm to assign a label to each catchment basin and uses the shortest path concept to assign a chain code to non-minima plateaus. As a result, our algorithm can perform watershed transformation on a binary image in just four scans. We verified its execution time and accuracy on a PC and compared these with those of conventional watershed algorithms. The average running time improved by more than 39% without any loss of accuracy. The low complexity, simple data structure, and constant scan times of our algorithm make it faster than existing algorithms. Moreover, the low memory consumption, simple data structure, and constant scan times of our algorithm make it suitable for hardware implementation in low-end devices. Our concept will be especially effective when performing real-time processing at high speed.

References

[1] L. Vincent and P. Soille, "Watersheds in digital spaces: an efficient algorithm based on immersion simulations," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 13, no. 6, pp. 583–598, Jun 1991.

[2] S. Beucher and F. Meyer, *The morphological approach to segmentation: the watershed transformation*, Mathematical Morphology in Image processing, Marcel Dekker Inc, 1993.

[3] S.Y. Chien, Y.W. Huang, and L.G. Chen, "Predictive watershed: a fast watershed algorithm for video segmentation," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 5, pp. 453–461, May 2003.

[4] C. Rambabu and I. Chakrabarti, "An efficient immersion-based watershed transform method and its prototype architecture," *Journal of Systems Architecture*, vol. 53, no. 4, pp. 210 – 226, 2007.

[5] C.J. Kuo, S.F. Odeh, and M.C. Huang, "Image segmentation with improved watershed algorithm and its fpga implementation," in *Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium on*, May 2001, vol. 2, pp. 753–756 vol. 2.

[6] D.B.K. Trieu and T. Maruyama, "Implementation of a parallel and pipelined watershed algorithm on fpga," in *Field Programmable Logic and Applications, 2006. FPL'06. International Conference on*. IEEE, 2006, pp. 1–6.

[7] K. Haris, S.N. Efstratiadis, N. Maglaveras, and A.K. Katsaggelos, "Hybrid image segmentation using watersheds and fast region merging," *Image Processing, IEEE Transactions on*, vol. 7, no. 12, pp. 1684–1699, Dec 1998.

[8] S.E. Hernandez and K.E. Barner, "Joint region merging criteria for watershed-based image segmentation," in *Image Processing, 2000. Proceedings. 2000 International Conference on*, Sept 2000, vol. 2, pp. 108–111.

[9] Luis Patino, "Fuzzy relations applied to minimize over segmentation in watershed algorithms," *Pattern Recognition Letters*, vol. 26, no. 6, pp. 819 – 828, 2005.

[10] A. Bleau and L.J. Leon, "Watershed-based segmentation and region merging," *Computer Vision and Image Understanding*, vol. 77, no. 3, pp. 317 – 370, 2000.

[11] A. Bieniek and A. Moga, "An efficient watershed algorithm based on connected components," *Pattern Recognition*, vol. 33, no. 6, pp. 907 – 916, 2000.

[12] H. Sun, J. Yang, and M. Ren, "A fast watershed algorithm based on chain code and its application in image segmentation," *Pattern Recognition Letters*, vol. 26, no. 9, pp. 1266 – 1274, 2005.

[13] O.R. Vctor, I.G.L. Juan, S.L. Nicols, and G.V. Pedro, "An improved watershed algorithm based on efficient computation of shortest paths," *Pattern Recognition*, vol. 40, no. 3, pp. 1078 – 1090, 2007.

[14] J. Cousty, G. Bertrand, L. Najman, and M. Couprie, "Watershed cuts: Thinnings, shortest path forests, and topological watersheds," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 32, no. 5, pp. 925–939, May 2010.

[15] Suphalakshmi. A and Anandhakumar P, "Article: An improved fast watershed algorithm based on finding the shortest paths with breadth first search," *International Journal of Computer Applications*, vol. 47, no. 11, pp. 1–9, June 2012, Full text available.

[16] C. Rambabu, T.S. Rathore, and I. Chakrabarti, "A new watershed algorithm based on hillclimbing technique for image segmentation," in *TENCON 2003. Conference on Convergent Technologies for the Asia-Pacific Region*, Oct 2003, vol. 4, pp. 1404–1408 Vol.4.

[17] Q. Gu, T. Takaki, and I. Ishii, "A fast multi-object extraction algorithm based on cell-based connected components labeling," *IEICE Trans. Inf. Syst.*, vol. 95, no. 2, pp. 636–645, 2012.

[18] Q. Gu, T. Takaki, and I. Ishii, "Fast fpga-based multiobject feature extraction," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 1, pp. 30–45, 2013.

Authorized licensed use limited to: University of New South Wales. Downloaded on July 09,2020 at 03:32:27 UTC from IEEE Xplore. Restrictions apply.