Edge-end Pixel Extraction for Edge-based Image Segmentation

Mahinda P. Pathegama and Özdemir Göl

Abstract—Extraction of edge-end-pixels is an important step for the edge linking process to achieve edge-based image segmentation. This paper presents an algorithm to extract edge-end pixels together with their directional sensitivities as an augmentation to the currently available mathematical models. The algorithm is implemented in the Java environment because of its inherent compatibility with web interfaces since its main use is envisaged to be for remote image analysis on a virtual instrumentation platform.

Keywords—edge-end pixels, image processing, image segmentation, pixel extraction

I. INTRODUCTION

Image segmentation is a central problem in image processing. Various techniques have been proposed to cope with the problem. These popularly include edge-based methods. Unfortunately, edge-based approaches can yield inaccurate results if broken boundaries are present in the image. Such boundaries need to be linked for reliable results.

This paper presents an algorithm to reliably extract the edge-end pixels for use in edge-based image segmentation in image processing. The extracted pixels can then be drawn into one of the several techniques available in image processing. Edge-linking technique is one of these, which is capable of proceeding with contour filling with its embedded directional sensitivity functions [1]. The proximity selectiveness feature of the edge-linking model allows the appropriate pixels to be selected for linking. In addition to the embedded functions, the algorithm presented here also nominate directions for each edge-end pixel. This direction nomination assigned at the same time of edge-end pixel extraction aims at enhancing the acuity of the edge-linking technique.

II. ALGORITHM OVERVIEW

The algorithm is implemented in Java with two major subroutines (classes). The first class compiles the image whereas the second class does householding.

The algorithm aims at performing two tasks: extraction of edge-end pixels and recognition of the associated directions. At the outset, the algorithm reads all the pixels in a 2-D image row by row by a speedy process. The search first identifies each pixel located at edge-ends by seeking the neighbouring pixel values. If the target pixel is found it records the coordinates of the recognised pixel to complete the extraction.

To perform the task of recognition, the numeric scheme indicated in Fig. 1 is used to represent the eight ‘directions’ of the target pixel. Similar schemes have been used for image processing in the past. One method is the so-called Chain Coding [2][3][4], which traces a pixel-wide line, using the scheme of Fig. 1.

The direction of each of the detected pixels takes into account the known alignment of neighbouring pixels.

III. ALGORITHM EXECUTION

The execution starts with the creation of a two dimensional grid using a two dimensional array of a certain number of columns and rows. In the following, Java code for the salient operations will be given in the interest of transparency.

```
private Pix[][] pixArray;
pixArray = new Pix[rows][cols];
```

The array is then filled with new Pix objects, where the
value of the x and y coordinates is set using the Pix class constructor.

```java
for(int i=0;i<rows;i++)
{
    for(int j=0;j<cols;j++)
    {
        pixArray[i][j] = new Pix(j,i);
    }
}
```

Once this is complete, specified Pix objects have their filled variable set to represent an actual pixel. Then an ASCII representation of the grid with the filled pixels is printed out. One test sample supplied is given in Fig. 2 and corresponding input image is shown in Fig. 3.

The Pix.java class represents a pixel in the grid and stores information pertaining to that pixel space. This includes variables which store the x coordinate, the y coordinate, whether the pixel space is 'filled' and its direction. Each variable has Accessor and Mutator stages.

The algorithm searches all Pix objects in the array using the search() method for any that have their filled variable set to 'true'. For example, the code below checks the right neighbouring object.

```java
if(pixArray[i+1][j].getFilled())
{
    count++;
    dir=directionarray[2];
    checkdiag=false;
}
```

Should it find any objects, the searchNeighbours(i,j) method is then called to examine all other eight Pix objects surrounding the discovered Pix object located at row i, column j. In this way, every pixel is investigated separately by searching every direction circularly in counter clockwise sense.

In the searchNeighbours stage, an incremented variable is used to count the number of neighbouring Pix objects which are filled. First, the horizontal and vertical neighbours are examined. If none of them is filled, the diagonal Pix objects are checked. This step sets the direction of the incomplete pixel to be the opposite to that of the neighbouring cell direction. Should the count value of neighbouring filled pixels equal 1, then the direction of the original Pix object is set such that the value is opposite to that of the neighbouring filled object. The Pix object is then stored in an ArrayList and has the variable isSet assigned ‘true’ in the method set():

```java
if(count==1){
    set(pixArray[i][j],dir);
}
```

If the count is greater than one, then the search continues; the direction remains unassigned.

### IV. RESULTS

A new ASCII grid representation is constructed showing only the extracted edge-end pixels, as shown in Fig. 4.

The target pixels are detected and extracted from the image pixels of Fig. 3. The outcome is the extracted edge-end pixels as in Fig. 5. A comparison with Fig. 3 shows that the isolated pixels located at (1,1) and (1,8) are also deleted by the
algorithm as constituting noise.

Figure 5: Output image given by the algorithm: Extracted edge-end pixels

If the size of the ArrayList is greater than 0 (i.e. it contains one or more Pix objects), then the x coordinate, the y coordinate and the direction of the Pix objects stored in the ArrayList are displayed. The result shown in Fig. 6 is produced for the input data shown in Fig. 2. Directions are assigned in accordance with the scheme depicted in Fig. 1.

| x coord: 4, y coord: 2, direction: 5 |
| x coord: 6, y coord: 2, direction: 7 |
| x coord: 2, y coord: 4, direction: 1 |
| x coord: 8, y coord: 4, direction: 3 |
| x coord: 4, y coord: 8, direction: 0 |
| x coord: 6, y coord: 8, direction: 4 |

Figure 6: Test results produced for the input data in Fig. 2: Locations and directions for the edge-end pixels

Figure 7: Composite image comprising extracted edge-end pixels, input image and directional sensitivities

Fig. 7 presents the output image representing the extracted edge-end pixels superimposed on the input image. The directional sensitivities are now assigned for each pixel as indicated by the arrows in the composite image of Fig. 7. The outcomes from this step can be applied to surface perception or edge-linking processes when they utilised directional sensitivities.

V. APPLICATIONS

A. Neural Model Enhancement

Artificial neural activities can be considerably enhanced by the directional sensitivity manipulation performed by the direction-assigning algorithm presented in this paper. The extracted edge-end pixels - with the aid of neighbouring cell-responses - provide the basis for surface creation or edge-linking. The process uses as input the edge-end pixels of each fragment along with directional information obtained from direction sensitive neurons. Similarity- and proximity-based selection is used during the process.

B. Modelling of Biological Systems

Several recent studies have confirmed the presence of direction-sensitive cells in the visual cortex. Most cells in layer IV have spatiotemporally oriented receptive fields in which gradients of response time across the field convey a direction [5]. Linear summation of these responses across the receptive field, followed by a static nonlinear amplification, has been shown to account for directional tuning in layer IV. Most neurons in area VI also manifest some directional tuning and spatiotemporal orientation [5].

A study of visually responsive neurons in the superficial layers of the rat brain found that cell responses within the superior colliculus respond to direction by evoking direction-biased cells [6]. Almost every cell in the middle temporal area (MT) is sensitive to direction of movement [6]. This has provided the impetus for many vision researchers to turn their attention to motion rather than 'directional-sensitivity'.

The algorithm presented in this paper can be used in the modelling of the above biological phenomena. The technique used in the algorithm intrinsically reflects the tuning of cells in opposing directions, as illustrated in Fig. 7. This is also in agreement with a laboratory study on neurotransmitter [7] which revealed that directionally sensitive ganglion cells become equally responsive to opposite directions when a visual stimulus is present.

C. Medical Image Analysis

Medical image analysis is accomplished by applying a number of image processing techniques sequentially. These may include smooth filtering, edge detection, thresholding along with morphological operations including the removal of small features and thinning operation.

The application of edge detection and thresholding steps invariably reveal discontinuities in cell boundaries. The algorithm presented in this paper is suitable in closing gaps without distorting the object boundaries in medical images.
D. Virtual Instrumentation Platform for Remote Clients

A virtual instrumentation (VI) platform is ideally suited to image analysis by remote clients via the Internet. The inbuilt functions of a VI can be used to extract and label the objects during the edge-based image segmentation process. Additional advantages of using the VI platform are user friendliness, interactive use and suitability for use on the web. The algorithm satisfies the requirements for remote delivery due to being compiled in Java.

VI. CONCLUSION

The algorithm presented successfully extracts edge-end pixels in their entirety. The simplicity of the proposed algorithm should make it an attractive tool for edge-based image segmentation; essential in biological cell image analysis and indeed in any image processing task.

REFERENCES


Mahinda P. Pathegama holds a Bachelor of Engineering degree with First Class Honours in Electrical and Mechatronic Engineering from the University of South Australia, Australia. He is currently a PhD candidate at the University of South Australia.

He has been passionately interested in being able to model the human vision and the brain, particularly with the aid of artificial neural networks. His PhD research has applied engineering skills crucial to medicine, developing artificial intelligence techniques to enhance diagnostic accuracy in medical practice.

Mr. Pathegama has gained much recognition for his academic achievements by winning prestigious awards and prizes which have included the Sir William Goodman Electrical Engineering Prize, the Australian Postgraduate Award, the Chancellor’s Award and the Dean’s Merit Award of the University of South Australia and the South Australian EDSA Utilities Prize. He is a Member of The Institution of Engineers, Australia (MIEAust), and a Member of the Institution of Electrical Engineers (MIEE), UK.

Özdemir Göl has well recognised expertise in the advanced application of engineering techniques to problems ranging from test automation in manufacturing to medical applications. He holds the degrees of Master of Engineering Science from the Istanbul Technical University, Turkey, Master of Engineering from the University of Melbourne, Australia, and PhD from the University of Adelaide, Australia; all in electrical engineering.

He is currently Head of Electrical Engineering at the University of South Australia, and Director of the Electrical Machines and Drives Research Group which he has founded. Previously he worked in industry and academia in Europe.