Lab 3 Segmentation, Thresholding and Region Growing Report

ECE 5470

Computer Vision

Name: Yanling Wu

NetID: yw996

Section 2: Peakiness Detection

Description of the vtpeak.c program:

1. The program will acquire the names of input and output file and the parameter, dist, of minimum move distance.
2. Compute the histogram of the picture and find the maximum bin for the entire histogram as maxbin.
3. Find the next maximum bin at dist below maxbin as maxb and above maxbin as maxa and compare maxb with maxa to get the second maximum bin as nxtbin.
4. Find the minimum between peaks and use it as threshold.
5. Set all pixel values which are less than threshold to 0 and set all pixel values which are more than threshold to 255.

Strengths and Weaknesses:

Strengths:

The contrast of some processed pictures will become more noticeable and some details could become clearer. And the calculation of this method is easy.

Weaknesses:

1. Like the figure 1.1 shown, this method will ignore important information when the pixel values' distribution of original picture is very uneven. Because sometimes the pixels that can express the information only account for tinny percent of whole pixels and when computed the histogram, this part will be small, and this method will ignore it. Like the facsimile picture, its histogram distribution is the figure 1.2, we can see this distribution is very uneven. Most of the pixel values are

in the high values and few are in low area. When I tried to change the value of d, it is very difficult to detect the low pixel value. So, after it was processed, it will become almost totally white.

2. For gray pictures, we find the threshold using this method, it will segment the picture only use one criterion. And it will lose important edge information Like the figure 1.3 shown, this shuttle gray picture almost loses all features after processed by vtpeak.c. Because it only pick one threshold and set the lower pixel value to 0 but for this picture, most edge information are in area where pixel values are lower.

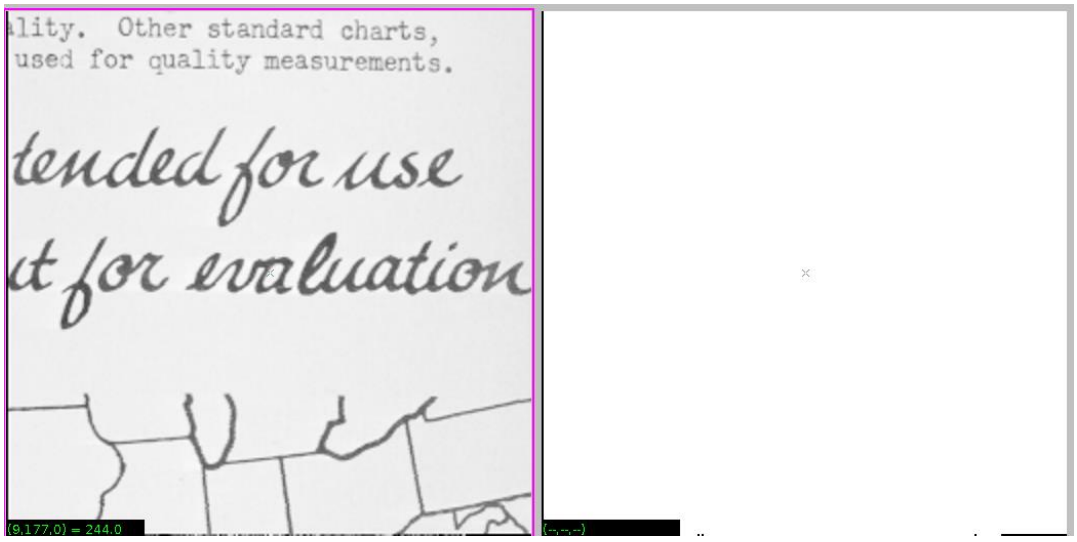


Figure 1.1 The comparison of facsimile picture and the processed picture

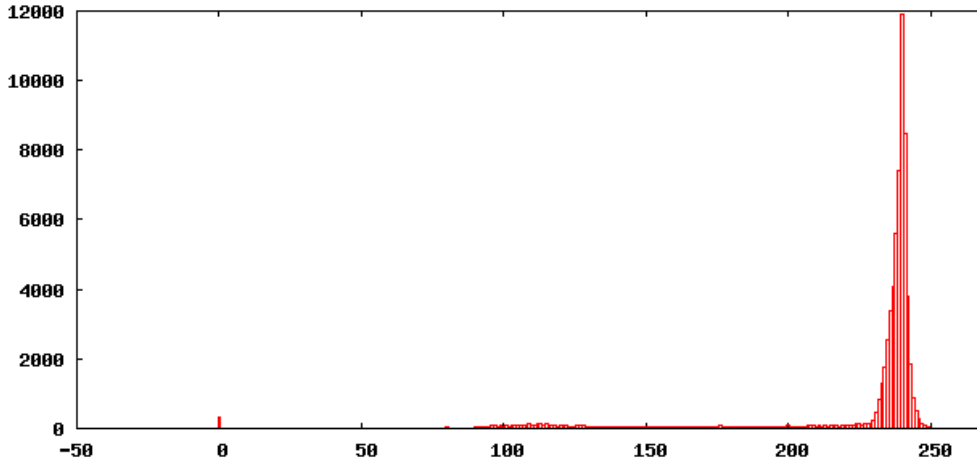


Figure 1.2 The histogram distribution of facsimile

Figure 1.3 The comparison of shuttle picture and the processed shuttle picture.

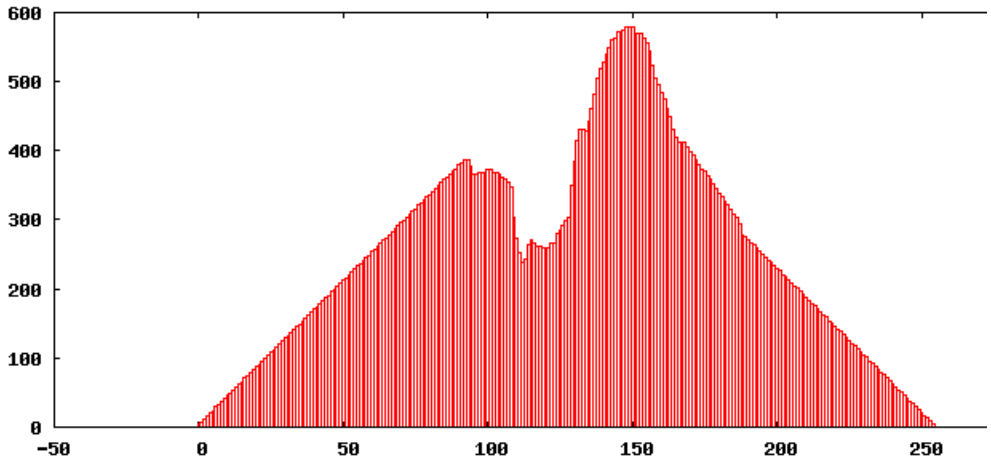


Figure 1.4 The histogram distribution of shuttle

Section 3: Iterative Threshold Selection

Description of the vits.c program:

1. The program will acquire the names of input and output file and the parameter, orithresh, the original threshold, if there is no command to set the original thresh, then set the original threshold to 127.
2. Computer the average pixel values (avg1, avg2) for R1 and R2 and R1 is the region where pixel values are above the threshold and R2 is the region where pixel values are below the threshold.
3. Compute a new threshold using: thresh = (avg1 + avg2)/2.
4. Repeat step 2-4 until avg1 and avg2 do not change between successive iterations.
5. Apply the threshold to the image (values above thresh set to 255 otherwise 0).

Code is attached in Appendix.

Result:

Debug using small test images:

1. Test image1 (I create one small test image whose size is 12 * 12):

Threshold = 18

The left picture of figure 2.0 is the original picture and the right one is processed image. The threshold of this image is 18, so we can clearly see the right image is segmented by this criterion.

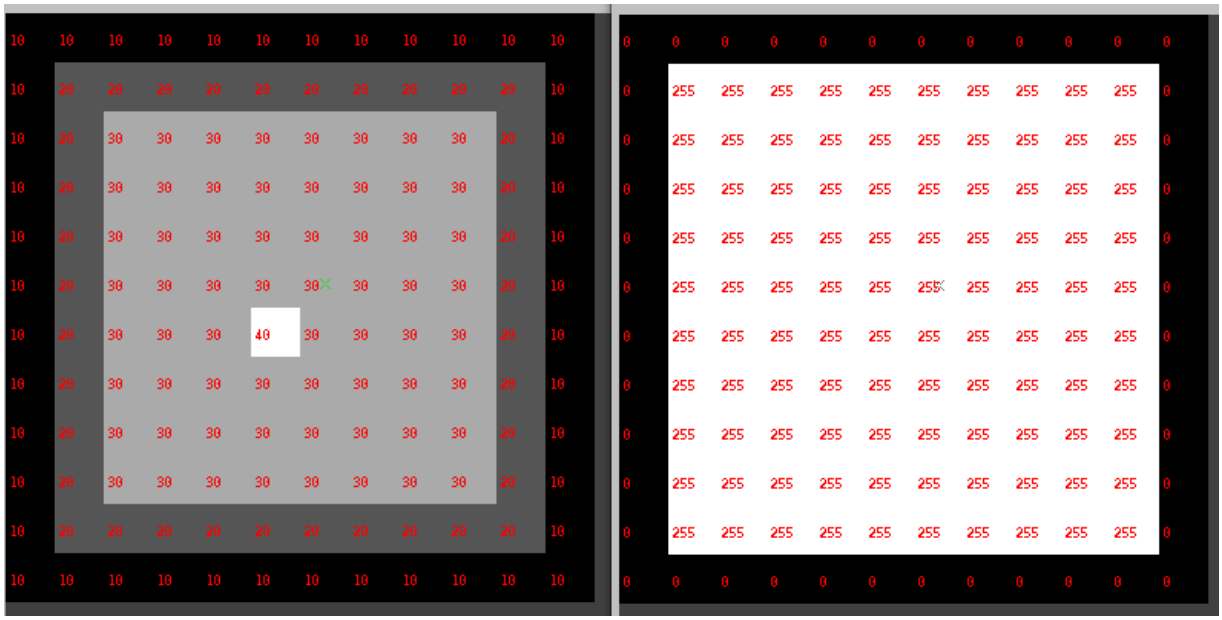


Figure 2.0 The comparison picture of my designed test image

2. Test image 2 (128 * 64)

Threshold = 127

Figure 2.1 is the comparison picture of a small test picture. The left picture is input picture and the right one is the output processed by vits.c. The calculated threshold is 127 and from this comparison, we can see processed picture still keep the features of the original picture.

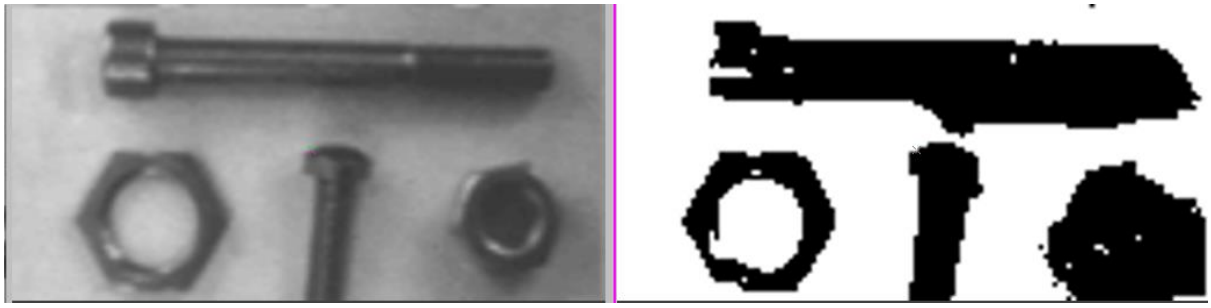


Figure 2.1 the comparison pictures of test nb.vx

Testing the full size image:

1. facsimile:

Threshold = 178;

From figure 2.2, we can see the improvement compared to the Peakiness Algorithm. This picture almost disappears after processed by vtpeak. But after the vits processing, it still can be read. Although some details of this picture become blur and hard to recognize, most of it keeps the original feature.

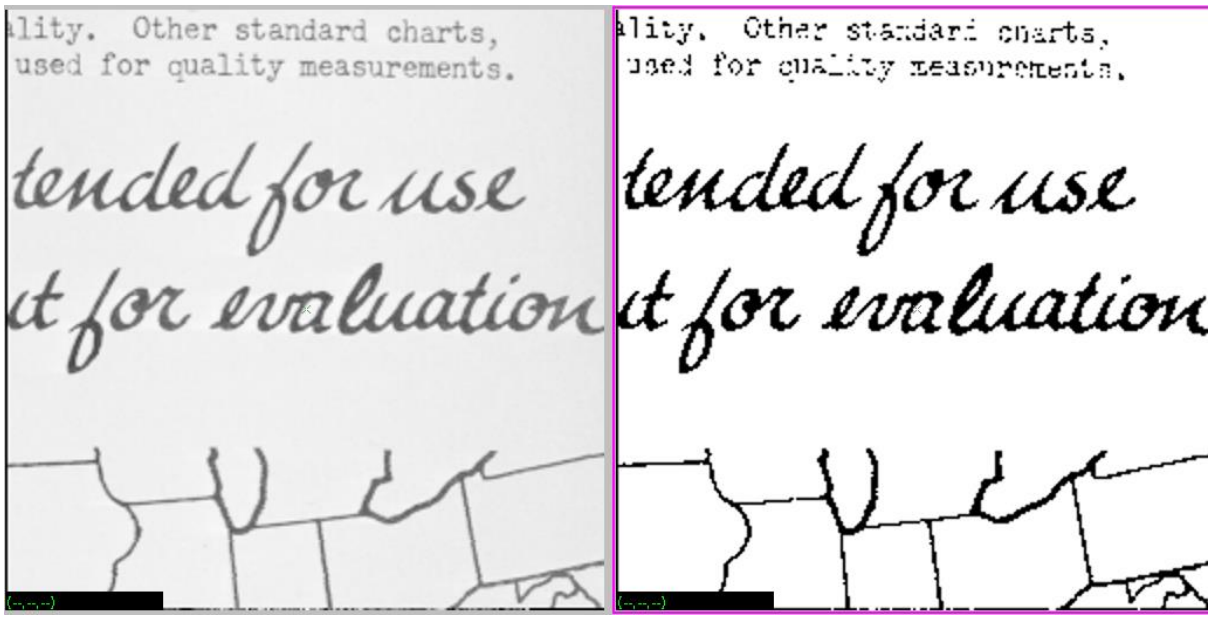


Figure 2.2 The comparison picture of the first full-size image-- facsimile

2. mp.vx:

Threshold = 141;

From figure 2.3, we notice that the vits process for this picture is not so much successful, although we still can recognize the contour of this devise, it loses a lot of details.

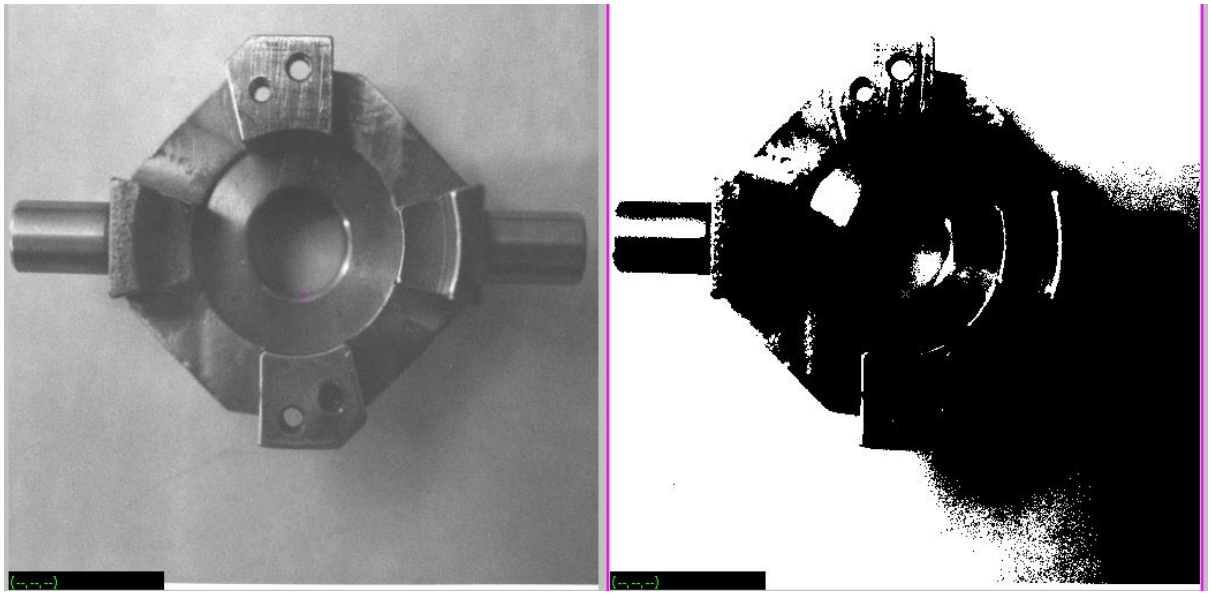


Figure 2.3 The comparison picture of the second full-size image—mp.vx

3. map:

Threshold = 132;

From figure 2.4, we can know that this method could enhance the contrast degree. But still, it will lose some details of the picture.

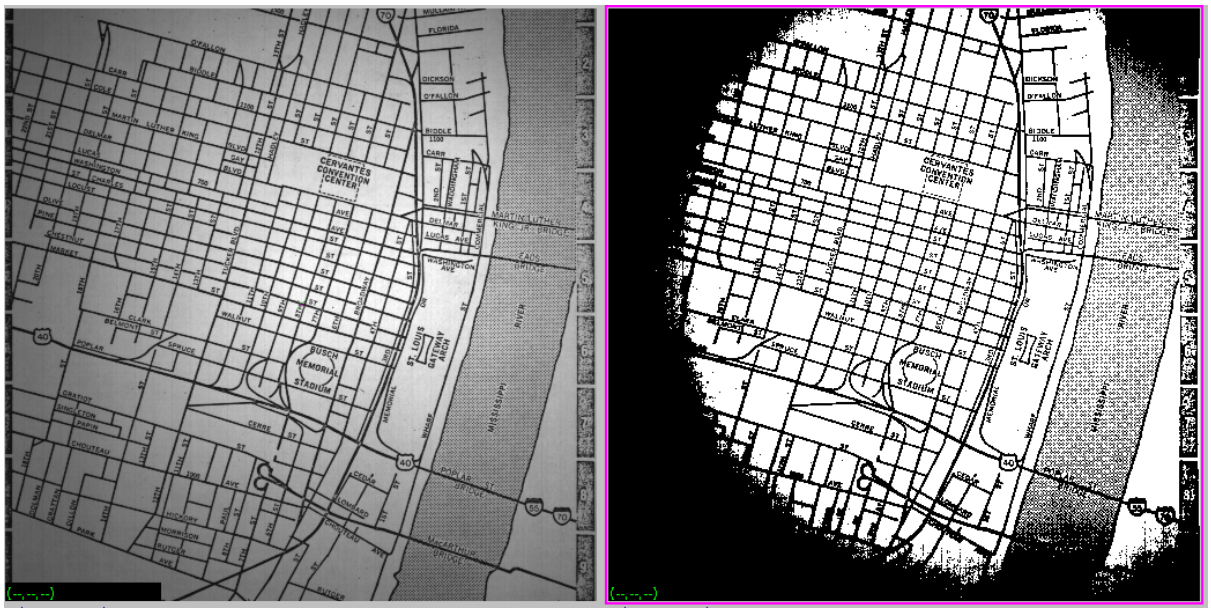


Figure 2.4 The comparison picture of the third full-size image -- map

4. shtl.vx:

Threshold = 122;

From the figure 2.5, we can see that this method is not efficient for this kind of picture.

There are some difficulties to decide which threshold is the best to segment this picture.

But this method is better than the peakiness for processing this shtl.vx picture.

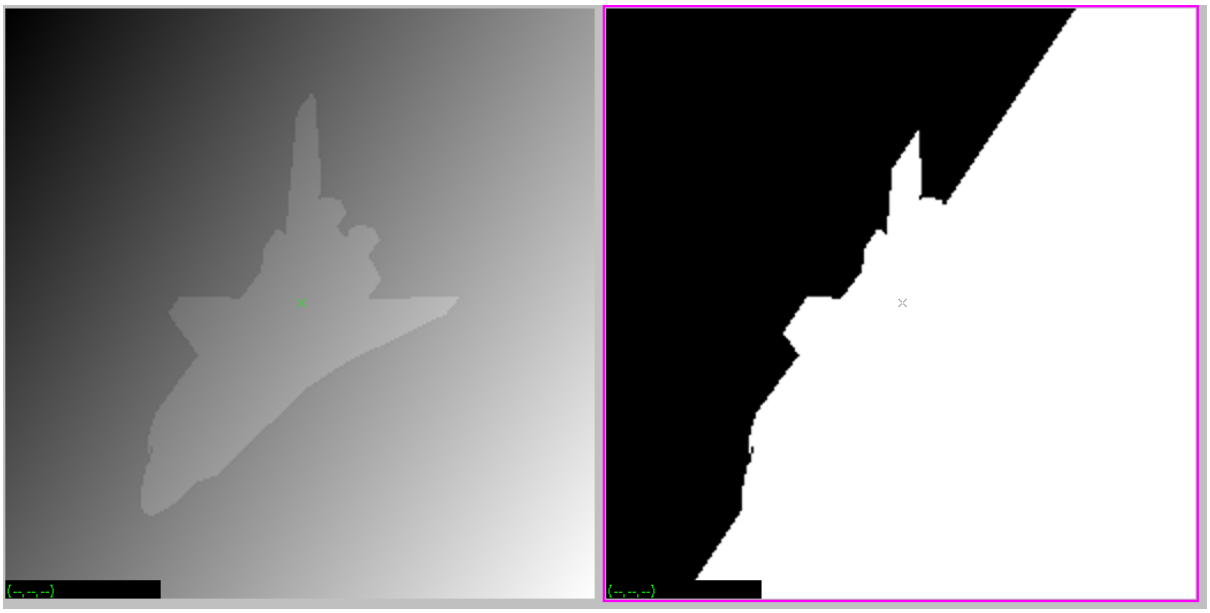


Figure 2.5 The comparison picture of the fourth full-size image – shtl.vx

All in all, this iterative threshold can decide a better threshold than peakiness algorithm

but still, it is not perfect and has limitation.

Section 4: Adaptive Thresholding

Description of the Adaptive Thresholding program:

Firstly, '*vpatch*' command decomposes an image into a set of overlapping rectangular image regions. The parameter of '*p*' means the patch size and the '*l*' means the values of overlap between adjacent patches.

Secondly, after decomposing the image into a set of regions, the peakiness algorithm will calculate the threshold in each region and segment this region according to the threshold. In this way, we can get different thresholds that fit in different regions. Different patch size and overlapping values can affect the quality of image segment.

Thirdly, we need to do parameter optimization to find the best parameters to get the best segment.

Observation on the results of this algorithms on map and mp.vx

1. If the patch size and overlap are too small or too big, the threshold of segmentation will be too low or too high and the processed picture will lose a lot of details and the noise is very high. As the figure 3.1 shows.

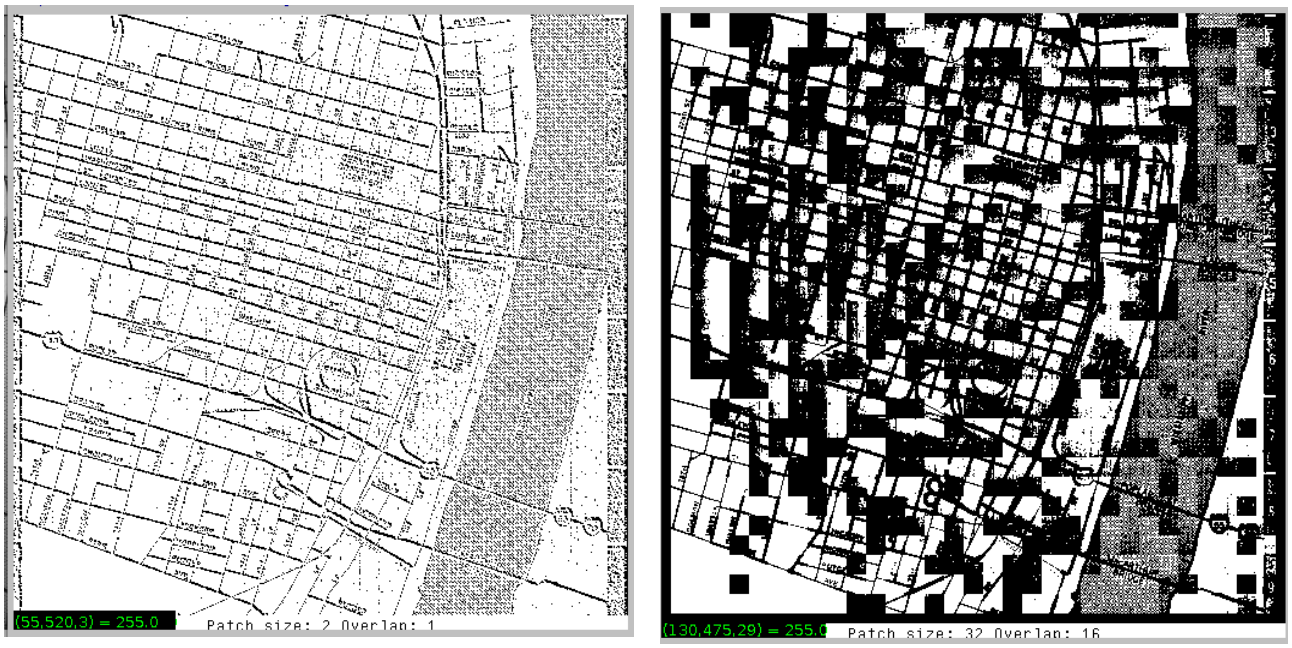


Figure 3.1 the left image is patch and overlap are too low and the right one is they are too high

2. I think the best parameters are patch size equals to 9 and the overlap equals to 8. Because I can see more details and edge features that original pictures own from the pictures under these parameters.

Figure 3.2 The comparison map image with 'best' parameters

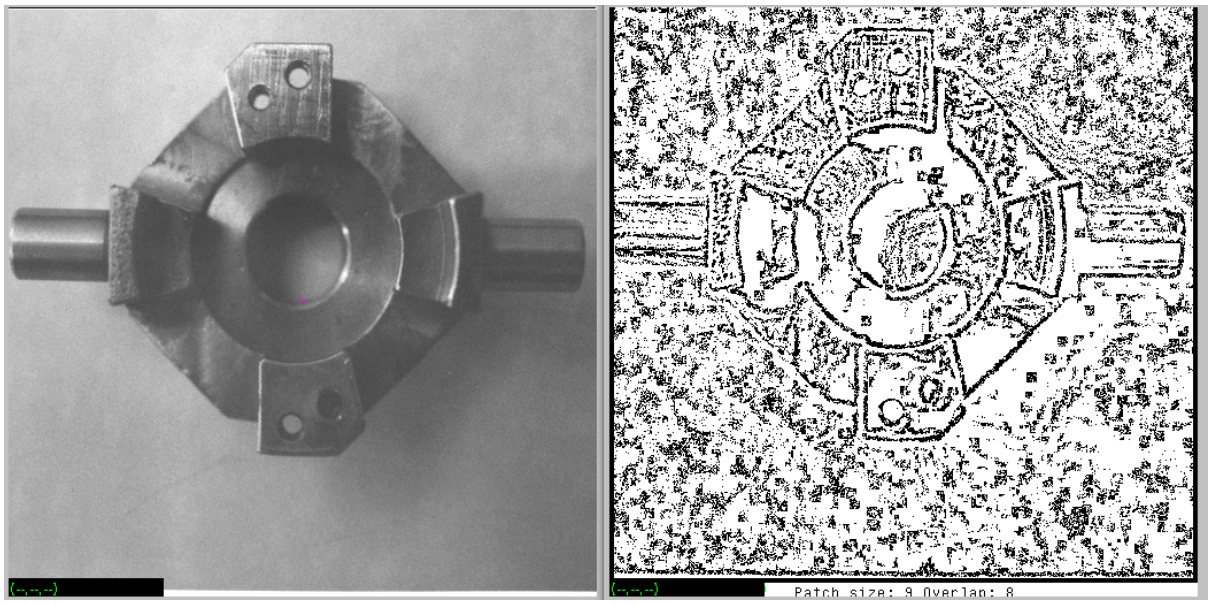


Figure 3.3 The comparison mp image with ‘best’ parameters

Section 5: Region Growing

Description of the vgrow.c program:

1. The program will acquire the names of input and output files and some parameters including 'r=' to set the region pixel range and '-p' to select the labeling scheme, if there is no 'r=' command, assign r=10;
2. Define some global variables, such as im, tm, range, first and embed the image im into the image tm and add a one pixel border into the image.
3. Set all pixels' labels to 0 and label variable n = 1.
4. Scan the input image, tm, if the image pixel is not labeled that means the values of im equal to 0, then, set the global variable, first, to this pixel value, and if '-p' flage is set, call *setlabel(x, y, first)*, else call *setlabel(x, y, n)* and judge whether n is more than 255, if it does, do n = n % 256 to roll over the value of n;
5. Repeat step 3 for all unlabeled pixels.
6. Recursive function *setlabel(x, y, L);*
   a) Set the image pixel at (x, y) to L (im.u[y][x] = L);
   b) Scan its four neighbors to check the following three criteria:
      i. Whether the image pixel, tm, is non-zero (we assume that we only have zero pixel at the image border, this check prevent us from leaving the bounds of im);
      ii. And whether the pixel is unlabeled (whether the value of im equals to zero);

iii. And whether the pixel is in range, which means the difference between this pixel and the *first* pixel for this region is less than *range*;

c) If its neighbors satisfy these three criteria, then call *setlabel(x, y, L)* (x, y) represents the neighbors' coordinate values).

Code is attached in Appendix.

Result:

Debug using small test images:

Test image1 (I create one small test image whose size is 12 * 12):

The following figures are the test image, the figure 4.1 is the comparison images whose test image is no '-p' flag and range = 10 and this *vgrow.c* can successfully label different regions with sequence. The test images of figure 4.2 and figure 4.3 are both with '-p' flag but the former one's range is 10, the latter one's range is 20. And this program works well. It can label the same part according our criteria.

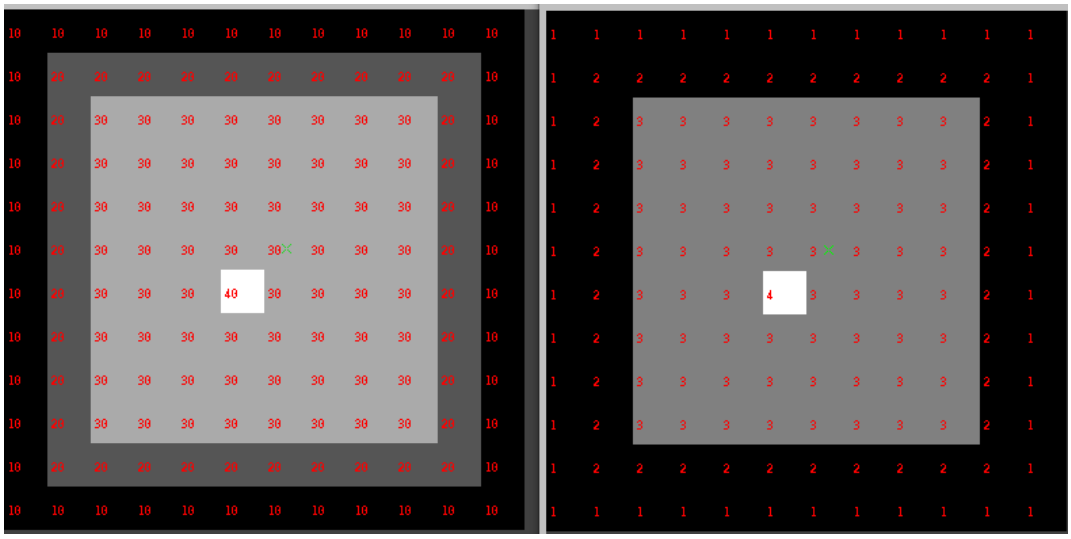


Figure 4.1 Test image with no '-p' flag and range = 10

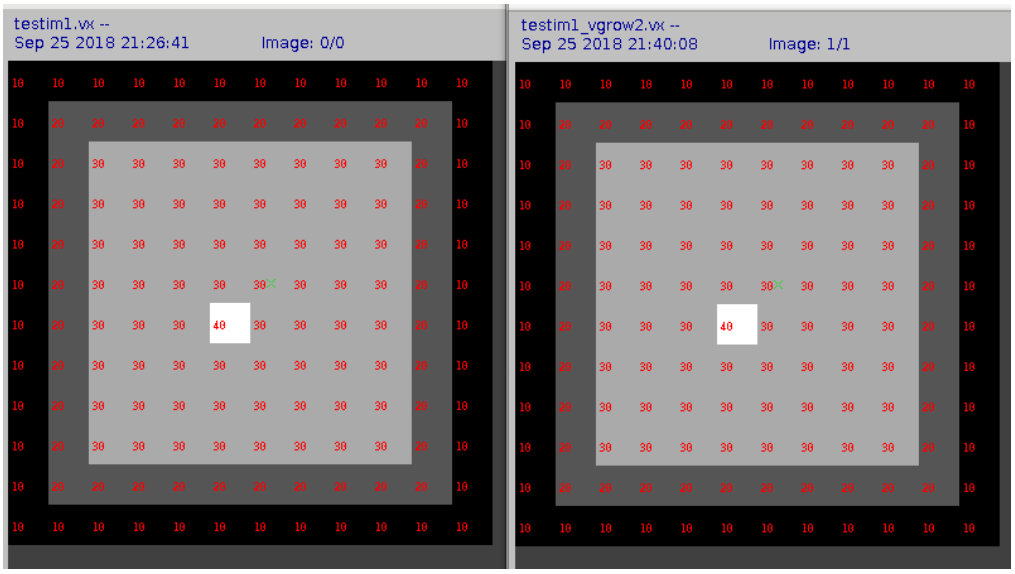


Figure 4.2 Test image with '-p' flag and range = 10

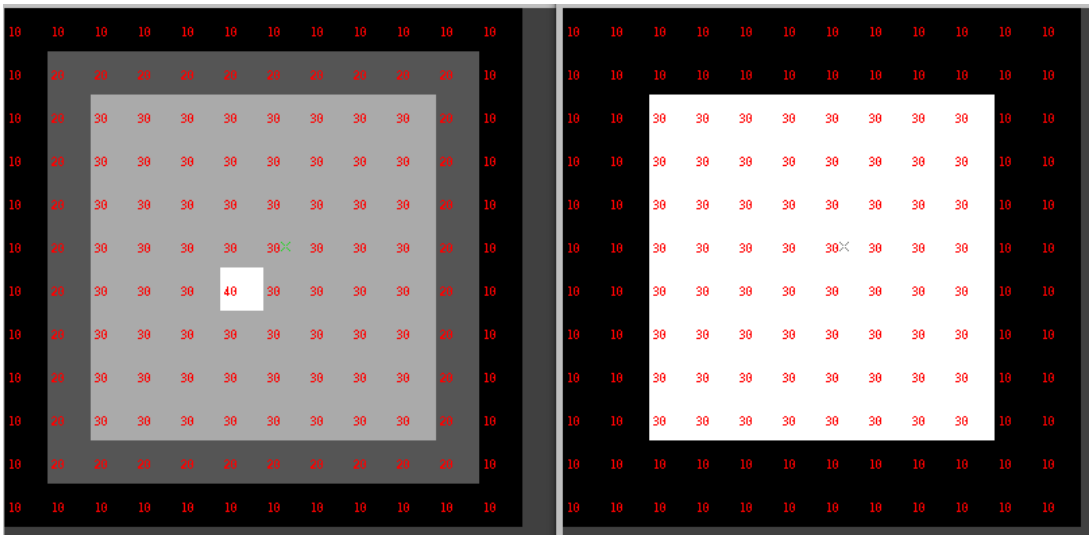
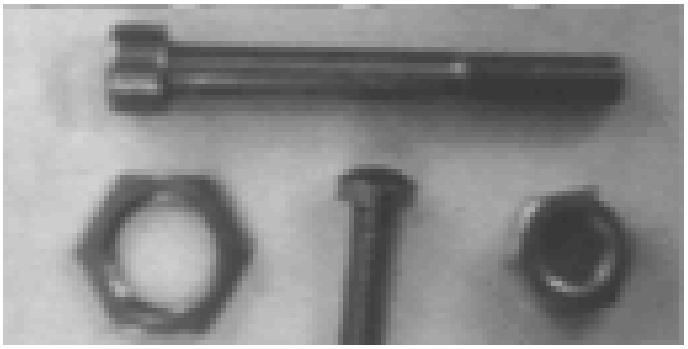


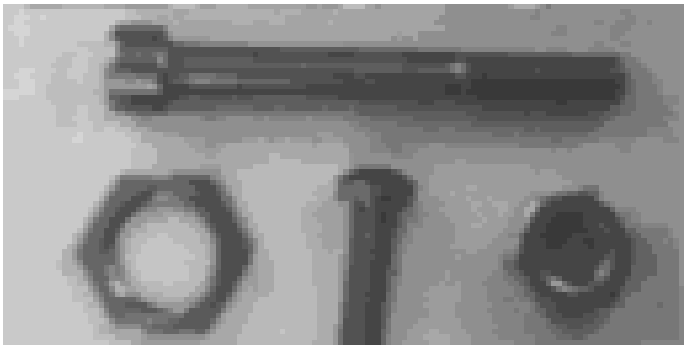
Figure 4.3 Test image with '-p' flag and range = 20

Testing the full-size image:

We used the *vgrow* to process the *nb.vx* and *shtl.vx* and the *range* equals to 10. The following images are the input and output images.



nb.vx



vgrow processed

shtl.vx

vgrow processed shtl.vx, r=10

Executing vgrow on the edge detected image :

As figure 5.1 shows, the left image is the one that we execute the *sobel* edge operator on *shtl.vx* image and then use *vgrow* to process the edge image to generate. But in this way, there will be some strange things, which is there are many shade in the background. This is because there are a lot of pixels whose pixel values equal to 0 in the original image, and when *vgrow* executing, it will recognize these

pixels as border So it will segment these pixels that should be considered as one region.

However, the right image is the one that I, firstly, invert the *shtl.vx* to *shtl_inv.vx* using command " *vpix -neg shtl.vx of=shtl_inv.vx* " and then execute the *sobel* edge operator to inverted image by using "*vsobel shtl.vx of=edge.vx*" and use vgrow to process it by using "*vgrow r=10 edge.vx of=edge_10.vx*".

We can avoid shadow problem in this way and the 'best' *range* that I use is 10. Because when range equals to 10, the edge of the plane is clearest and the segment is the best.

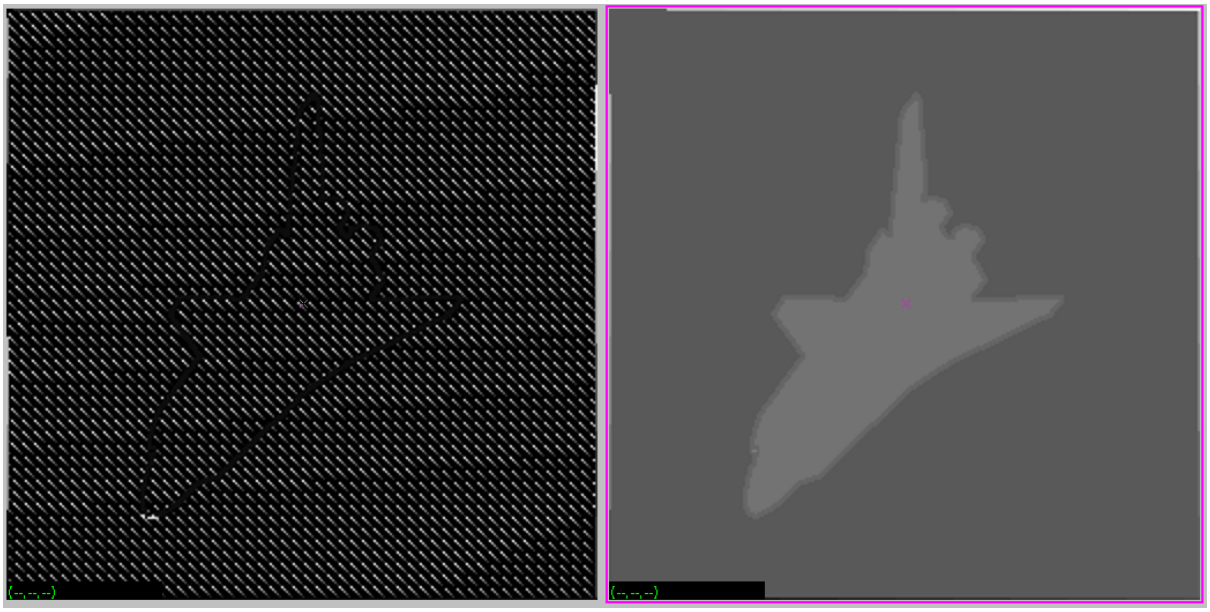


Figure 5.1 The comparison images of processed edge images by two different ways

Appendix

Code one:

vits.c

```
/*****************************************************************/
/* vits:     Iterative Threshold Selection Algorithm              */
/*@ Copyright                                                    */
/*  Name: Yanling Wu                                              */
/*  NetID: yw996                                                  */
/*  ECE 5470 Lab3                                                 */
/*****************************************************************/

#include "VisXV4.h"          /* VisionX structure include file      */
#include "Vutil.h"           /* VisionX utility header files        */

VXparam_t par[] =            /* command line structure              */
{
{  "if=",   0,   " input file, vtpeak: threshold between hgram peaks"},
{  "of=",   0,   " output file "},
{  "t=",    0,   " the initial threshold (default the average of all pixel values)"},
{  "-v",    0,   "(verbose) print threshold information"},
{   0,      0,   0} /* list termination */
};
#define  IVAL   par[0].val
#define  OVAL   par[1].val
#define  TVAL   par[2].val
#define  VFLAG  par[3].val

main(argc, argv)
int argc;
char *argv[];
{

    Vfstruct (im);               /* input image structure         */
    int y,x;                     /* index counters                */
    int i;
    long sum1 = 0;
    long sum2 = 0;
    int count1 = 0;
    int count2 = 0;
    int hist[256];               /*histogram bins                 */
    int thresh;                  /* threshold                     */
```

```
int orithresh = 127;              /* the original threshold   */
long sum = 0;               /*the sum of the all pixel values    */
int avg1 = 0;               /*the average of R1 (pixel above threshold) */
int avg2 = 0;               /*the average of R2 (pixel below threshold) */
int lasavg1 = 0;
int lasavg2 = 0;
int temp;
int count = 0;

VXparse(&argc, &argv, par);    /* parse the command line        */

if (TVAL) orithresh = atoi(TVAL);  /* if d= was specified, get value */
if (orithresh < 0 || orithresh > 255) {
fprintf(stderr, "d= must be between 0 and 255\nUsing d=10\n");
    orithresh = 143;
//orithresh = temp;

}
fprintf(stderr, "orithresh = %d\n", orithresh);

while ( Vfread( &im, IVAL) ) {
    if ( im.type != VX_PBYTE ) {
          fprintf (stderr, "error: image not byte type\n");
          exit (1);
    }
/*clear the histogram"*/
for (i=0; i <256; i++) hist[0]=0;
    for(y=im.ylo; y<=im.yhi; y++)
      for(x=im.xlo; x<=im.xhi; x++)
        hist[im.u[y][x]++;
  /*computer the threshold*/
while(1){
  for(i = 0; i < orithresh; i++){
    sum1 += i * hist[i];
    count1 += hist[i];
  }
  for(i = orithresh; i <256; i++){
    sum2 += i*hist[i];
    count2 += hist[i];
  }

 if(count1 == 0)
    avg1 = 0;
 else
```

```
        avg1 = sum1 / count1;
     if(count2 == 0)
        avg2 = 0;
     else
        avg2 = sum2 / count2;
     orithresh = (avg1 + avg2) / 2;
     if(lasavg1 == avg1 && lasavg2 == avg2){
        fprintf(stderr, "avg1 = %d\n", avg1);
        fprintf(stderr, "avg2 = %d\n", avg2);
        break;
    }
     lasavg1 = avg1;
     lasavg2 = avg2;
    fprintf(stderr, "avg1 = %d\n", avg1);
    fprintf(stderr, "avg2 = %d\n", avg2);
    fprintf(stderr, "orithresh = %d\n", orithresh);

    }

    thresh = orithresh;
    fprintf(stderr, "thresh = %d\n", thresh);

    if(VFLAG)
         fprintf(stderr, "thresh = %d\n", thresh);

    /* apply the threshold */
    for (y = im.ylo; y <= im.yhi; y++) {
        for (x = im.xlo; x <= im.xhi; x++) {
             if (im.u[y][x] >= thresh) im.u[y][x] = 255;
             else                      im.u[y][x] = 0;
        }
    }

    Vfwrite( &im, OVAL);
    } /* end of every frame section */
    exit(0);
}
```

Code two:

Vgrow.c

```
*****************************************************************/
/*vgrow      Compute label operation on a single byte image      */
/*****************************************************************/
```

```
/*@copyright                                                    */
/*Name: Yanling Wu                                             */
/*NetID: yw996                                                 */
/*LAB Number: LAB 2                                            */
/*************************************************************/

#include "VisXV4.h"          /* VisionX structure include file    */
#include "Vutil.h"           /* VisionX utility header files      */

VXparam_t par[] =            /* command line structure           */
{ /* prefix, value,   description                        */
{    "if=",    0,   " input file  vtemp: local max filter "},
{    "of=",    0,   " output file "},
{    "r=",     0,   " set the region pixel range"},
{    "-p",     0,   " select the labeling scheme"},
{     0,       0,   0}  /* list termination */
};
#define  IVAL   par[0].val
#define  OVAL   par[1].val
#define  RVAL   par[2].val
#define  PVAL   par[3].val

void setlabel(int ,int ,int );
Vfstruct (im);                    /* i/o image structure         */
Vfstruct (tm);                    /* temp image structure        */
int range;                       /*global varibale           */
int first;

main(argc, argv)
int argc;
char *argv[];
{

  int y,x;                       /* index counters              */
  VXparse(&argc, &argv, par);      /* parse the command line      */
  Vfread(&im, IVAL);              /* read image file             */
  Vfembed(&tm, &im, 1,1,1,1);      /* image structure with border  */
  if (RVAL)
      range = atoi(TVAL);     /* if t= was specified, get value */
  else
      range = 10;                /*set default range value     */
  if ( im.type != VX_PBYTE ) {      /* check image format          */
     fprintf(stderr, "vtemp: no byte image data in input file\n");
     exit(-1);
```

```
  }
  //set all pixels (labels) to 0
  for (y = im.ylo ; y <= im.yhi ; y++) {
         for (x = im.xlo; x <= im.xhi; x++)
            im.u[y][x] = 0;
    }
  int n =1;
/*  Scan the image to label every unlabeled point */
  for (y = im.ylo ; y <= im.yhi ; y++) {
     for (x = im.xlo; x <= im.xhi; x++)  {
        if(tm.u[y][x] != 0 && im.u[y][x] == 0){
            first = tm.u[y][x];
            if (PVAL)
                setlabel(x,y,first);
            else {
                setlabel(x,y,n);
                n++;
                if(n > 255)
                    n = n % 256;  /*in case the label over the ranger of pixel value
*/
            }
        }
     }
   }

   Vfwrite(&im, OVAL);            /* write image file  */
   exit(0);
}
/* setlabel function   */
void setlabel(int x,int y,int L){
    im.u[y][x] = L;
    if (tm.u[y+1][x] != 0 && im.u[y+1][x] == 0 && abs(tm.u[y+1][x] – first) < range )
        setlabel(x,y+1,L);
    if( tm.u[y-1][x] != 0 && im.u[y-1][x] == 0 && abs(tm.u[y-1][x] – first) < range )
        setlabel(x,y-1,L);
    if(tm.u[y][x-1] != 0 && im.u[y][x-1] == 0 && abs(tm.u[y][x-1] – first) < range )
        setlabel(x-1,y,L);
    if(tm.u[y][x+1] != 0 && im.u[y][x+1] == 0 && abs(tm.u[y][x+1] – first) < range )
        setlabel(x+1,y,L);
}
```