Title: Lab3 – Segmentation, Thresholding and Region Growing

Name:Yuixang Long

NetID:yl3377

Part 1: The result for Section 2

A. The histogram of the image mp.vx and the thresholded version generated by the command “vplot -h mp.vx” and “vtpeak” are as followed.

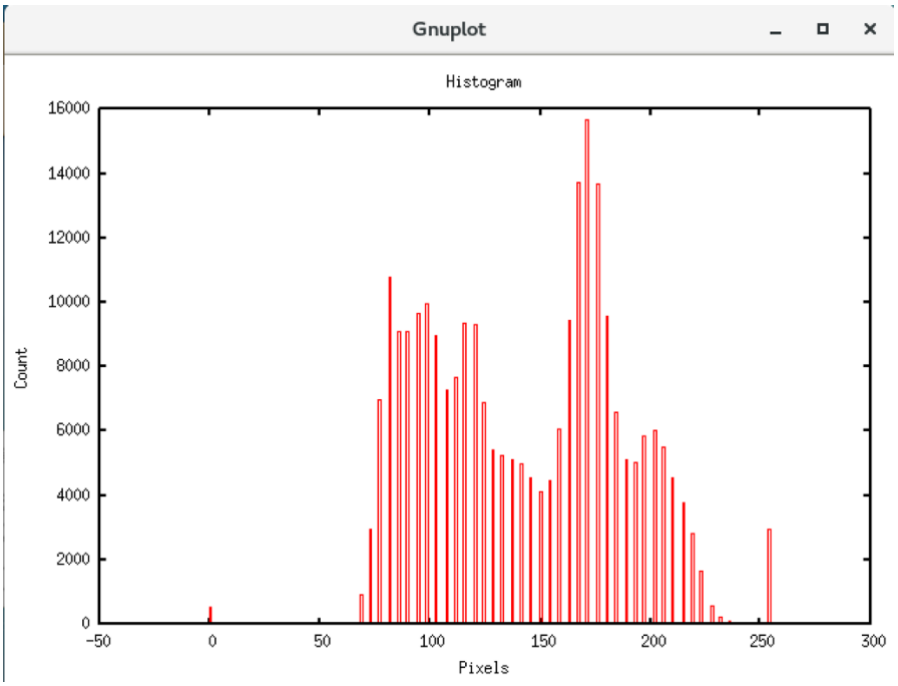


Figure 1: Histogram of mp.vx

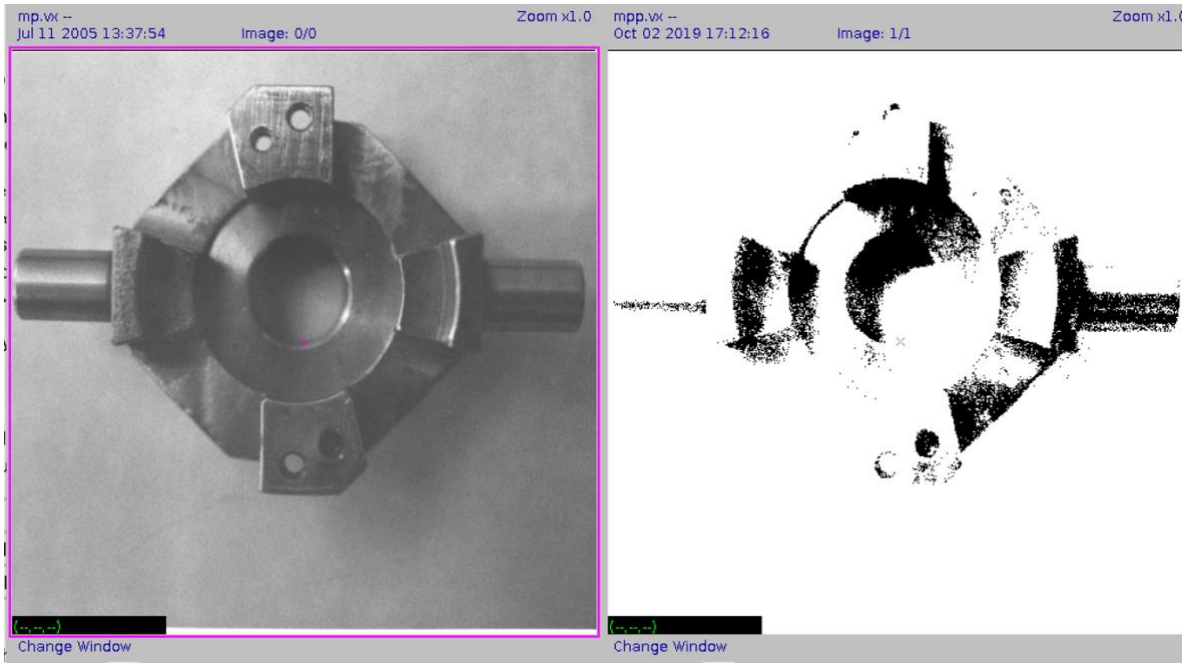


Figure 2: raw image, d=10

B. The histogram of the image facsimile and the thresholded version generated by the command “vplot -h facsimile” and “vtpeak”are as followed.

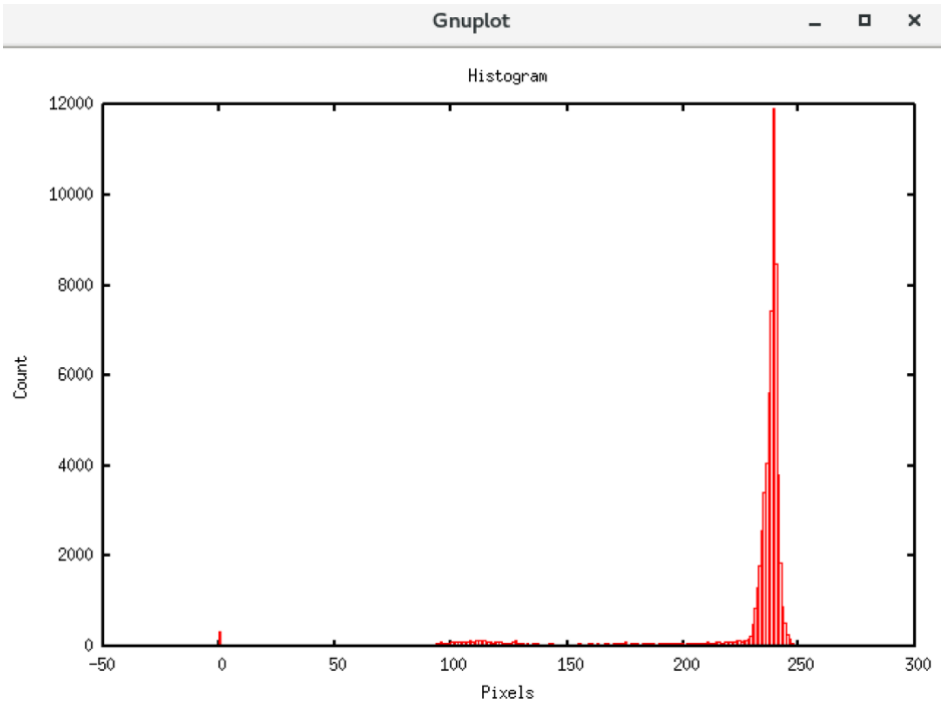


Figure 3: Histogram of facsimile

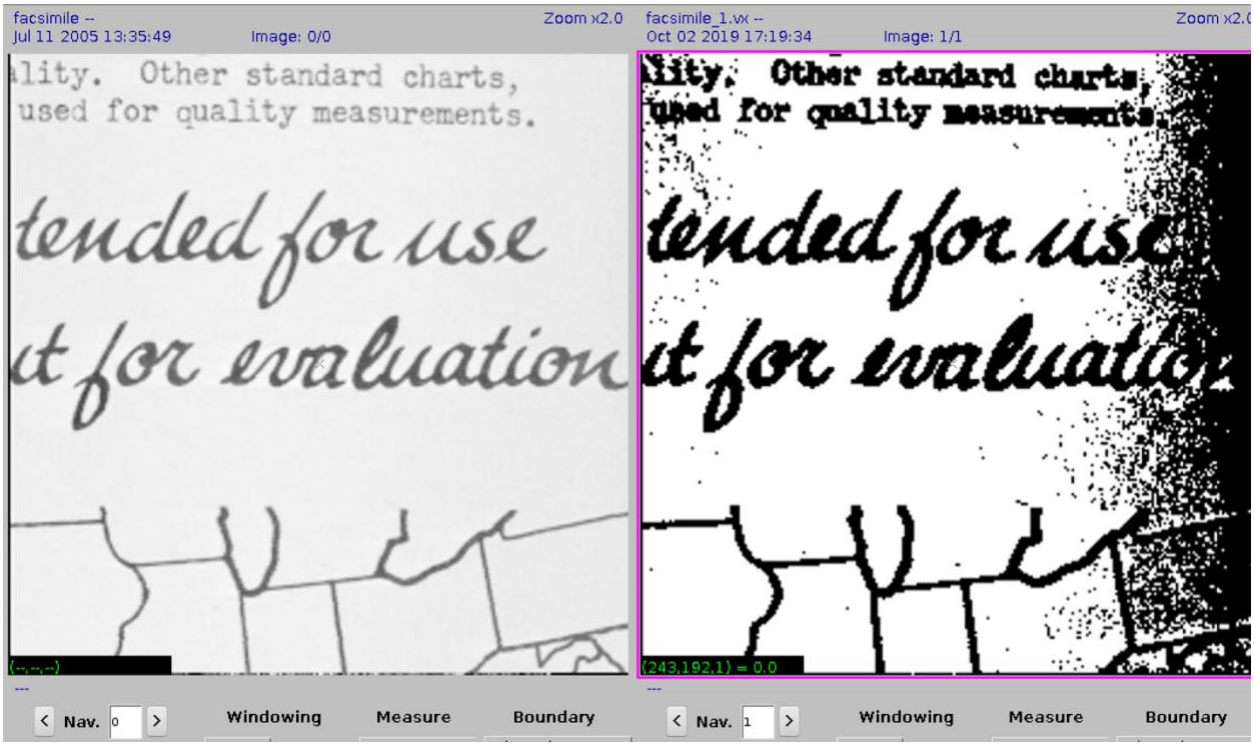


Figure 4: raw image, d=3

C. The histogram of the image map and the thresholded version generated by the command “vplot -h map” and “vtpeak” are as followed.

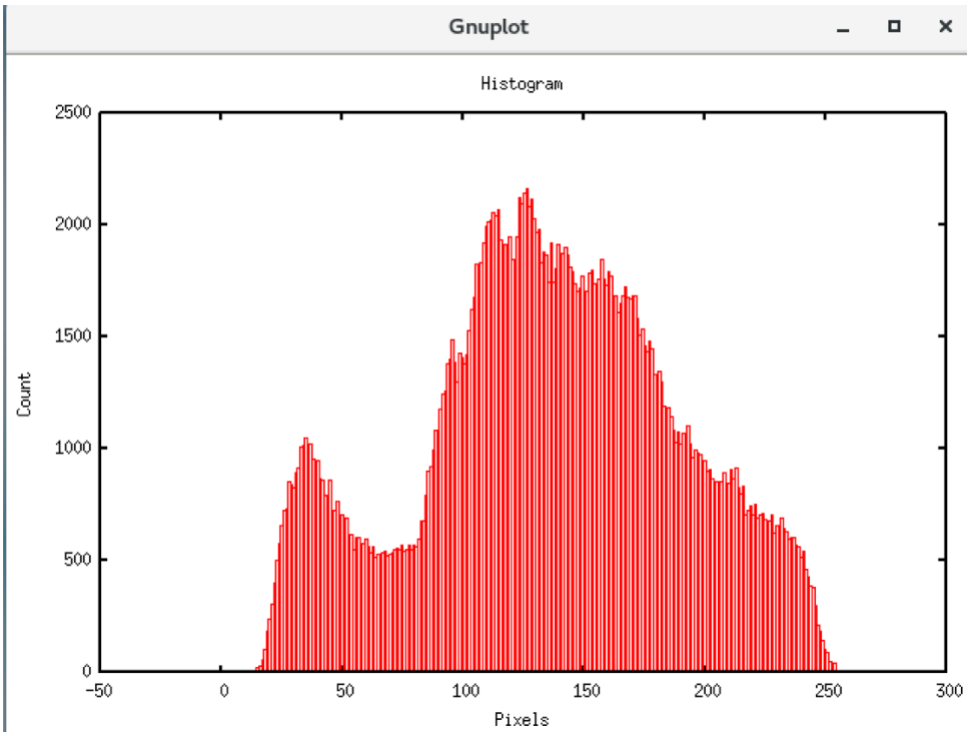


Figure 5: Histogram of map

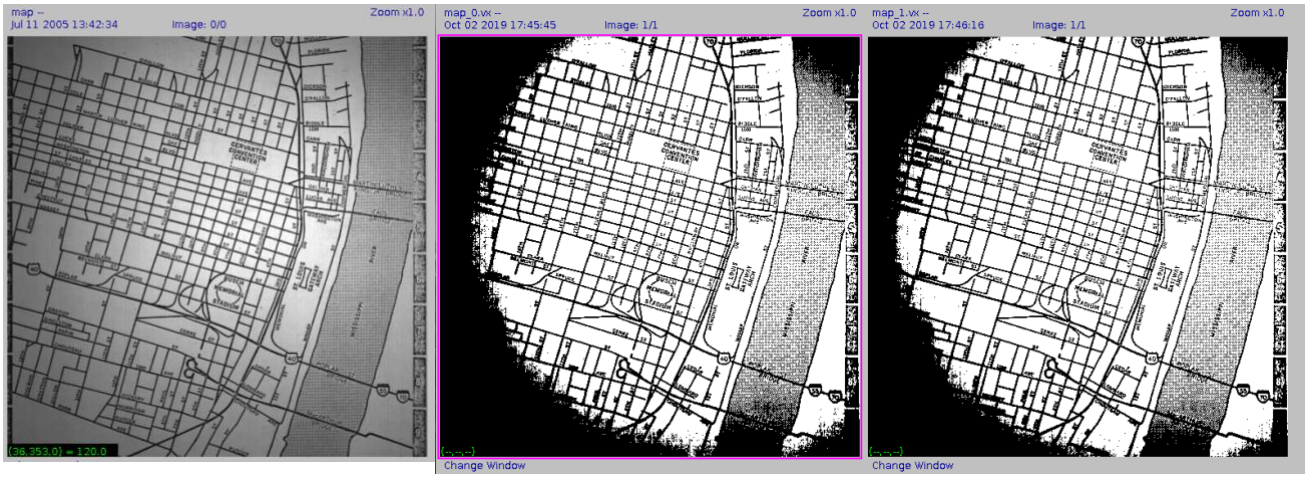


Figure 6: raw image, d=0, 10

D. The histogram of the image shtl.vx and the thresholded version generated by the command “vplot -h shtl.vx” and “vtpeak” are as followed.

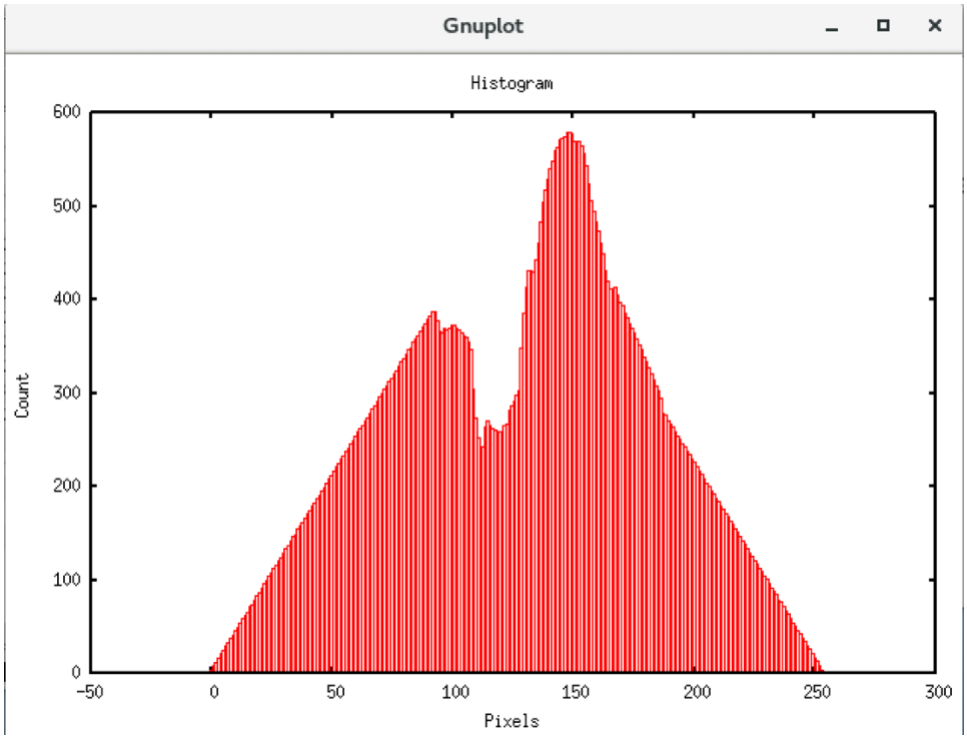


Figure 7: Histogram of shtl.vx

Figure 8: raw image, d=0, 10

E. The strength and weakness of the Peakiness Akgorithm is as followed:

Strength:

1. It is time saving because it only calculate once to generate the threshold of the image, compared with other recursive methods, it can greatly improve the efficiency.
2. The “d=” setting in the algorithm can avoid some kind of glitch, because the algorithm doesn’t need to take the value which is near the initial peak into account.

Weakness:

1. It only calculate once to generate the threshold value, as a result, this generated threshold will probably make the processed image less smooth, and maybe brings some gap to the image.
2. It can make the image unable to observe. For example, if the pixel value of the image is focused on the range [120,140], the initial peak is 130, then if we set the d to be 10, we can’t generate the nextpeak using this method, so it may make the processed image whole white. In the lab3, if we set d=10 to the image facsimile, the output is whole white because of this weakness.

Part 2: The description of the Iterative Threshold program in section 3

A. Firstly, we need to calculate all the pixel value in the image and add them together, then divide the sum with the whole pixel number, this can be achieved using two “for” cycle. The result of this part is the initial threshold.
B. Then we can separate the image into two parts R1 (pixel value less than threshold)and R2 (pixel value more than threshold). And I used sum1 to represent the sum of pixel value which belong to R1, and sum2 to represent the sum of pixel value which belong to R2. Besides, I also use pixel1 to represent pixel number of R1, and pixel2 to represent pixel value of R2. To begin with, the algorithm will detect each of the pixel from the image, if it is above the threshold, we add the value to sum2, and increment pixel2 for 1, else we will add to sum1 and pixel1. After getting the sum2, sum1 and pixel2, pixel1, the avg1 and avg2 can be calculated and new threshold is generated.
C. In this algorithm, I used the recursive method to achieve the function. after the B above, the algorithm will compare the new threshold to the older one, if they are same, break out of the circle, if not, go back to the beginning again until we get the final result (or until the count equals the set maximum count number).
D. After we generate the final threshold, we process the image with this threshold and output the final image.

Part 3. The list of the Iterative threshold program vits.c

```
/************************************************************************/
/* vtpeak:     Threshold image between two most sig. hgram peaks     */
/************************************************************************/

#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 "},
{  "count=",    0,    " min dist between hgram peaks (default 10)"},
{  "-v",    0,   "(verbose) print threshold information"},
{   0,      0,   0} /* list termination */
};
#define  IVAL    par[0].val
#define  OVAL    par[1].val
#define  COUNT   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;
    int sum;
    sum = 0;
    int pixel_sum;
    int count;
    count= 0;
    int count_set;
    int avg1, avg2;
    int sum1,sum2,pixel1,pixel2;
    int temp;
    int thresh;
    sum1 = 0;
    sum2 = 0;
    pixel1 = 0;
    pixel2 = 0;
```

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

count_set = 10000;                      /* default dist */
if (COUNT) count_set = atoi(COUNT);  /* if d= was specified, get value */

while ( Vfread( &im, IVAL) ) {
    if ( im.type != VX_PBYTE ) {
          fprintf (stderr, "error: image not byte type\n");
          exit (1);
    }

    /* compute the initial thresh */
```

```
    /* compute the initial thresh */
    for (y = im.ylo; y <= im.yhi; y++){
        for(x = im.xlo; x <= im.xhi; x++){
            sum = sum + im.u[y][x];
            pixel_sum ++;
            }
    }
    thresh = sum/pixel_sum;

     /* compute the threshold */
while(count < count_set){
    for (y = im.ylo; y <= im.yhi; y++){
        for (x = im.xlo; x <= im.xhi; x++){
            if (im.u[y][x] < thresh){
               sum1 = sum1 + im.u[y][x];
               pixel1 ++;}
            else {
               sum2 = sum2 + im.u[y][x];
               pixel2 ++;}
            }
    }
    avg1 = sum1/pixel1;
    avg2 = sum2/pixel2;
    temp = (avg1 + avg2)/2;
    if (temp == thresh) {break;}
    thresh = temp;
    count ++;
    }
```

```
        }
        avg1 = sum1/pixel1;
        avg2 = sum2/pixel2;
        temp = (avg1 + avg2)/2;
        if (temp == thresh) {break;}
        thresh = temp;
        count ++;
        }

        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);
}
```

Part 4. The vit program working for the small size image

In this part, I choose the nb.vx image as the small size test image. Firstly, I set the count_time (which controls the loop number) to be 0, the threshold calculated is 136, and the raw image and the processed image is as followed:

```
ECE5470:~/lab3 [9:58pm] 284$vits if=nb.vx of=nb_0.vx count=0 -v
thresh = 136
```

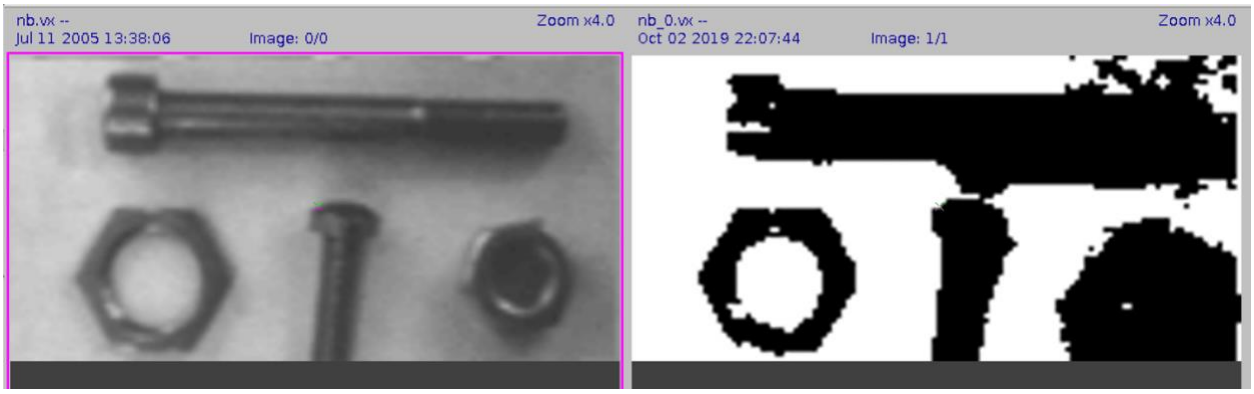


Figure 1: Raw image, count = 0

Then, I set the count_time (which controls the loop number) to be 10, the threshold calculated is 136, and the raw image and the processed image is as followed:

```
ECE5470:~/lab3 [10:13pm] 292$vits if=nb.vx of=nb_0.vx count=10 -v
thresh = 130
```

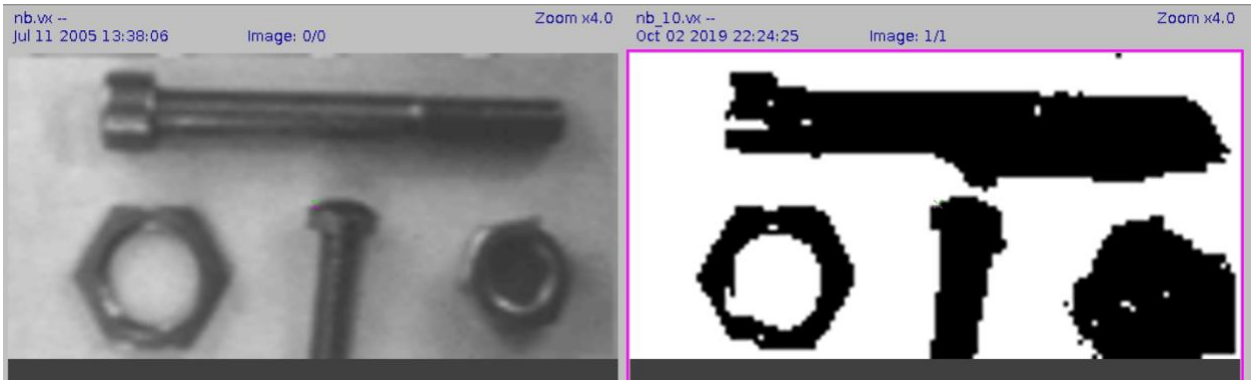


Figure 2: Raw image, count = 10

Part 5. The vit program working for the full size image

In this part, I choose image map to be the tested full size image, and the count_time is set to be 10. The calculated threshold is 134, and the input image and the output image are as followed:

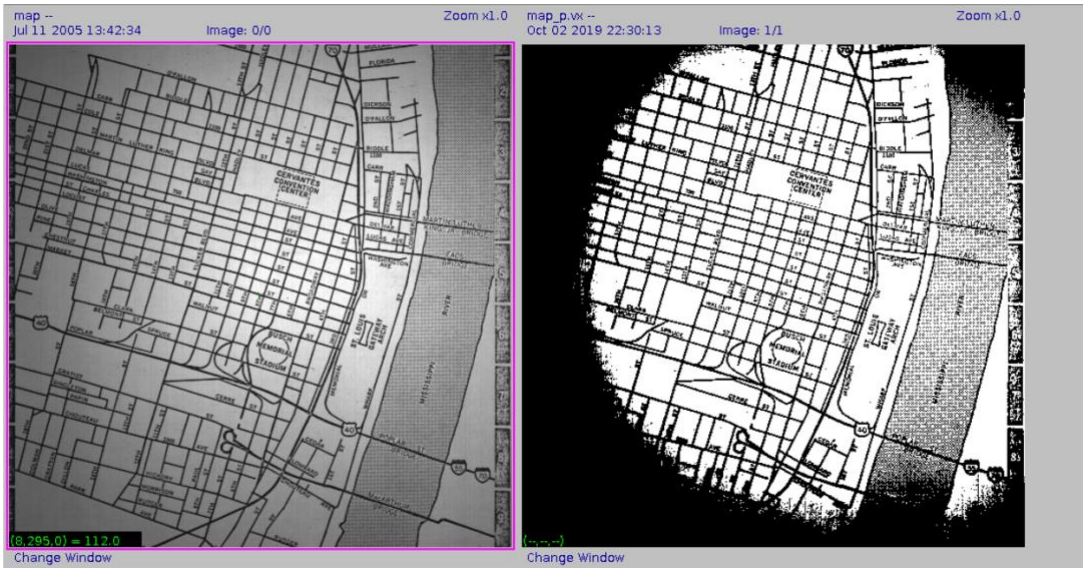


Figure 1: raw image, output image

I also tested another image – facsimilie, and the result is as followed:

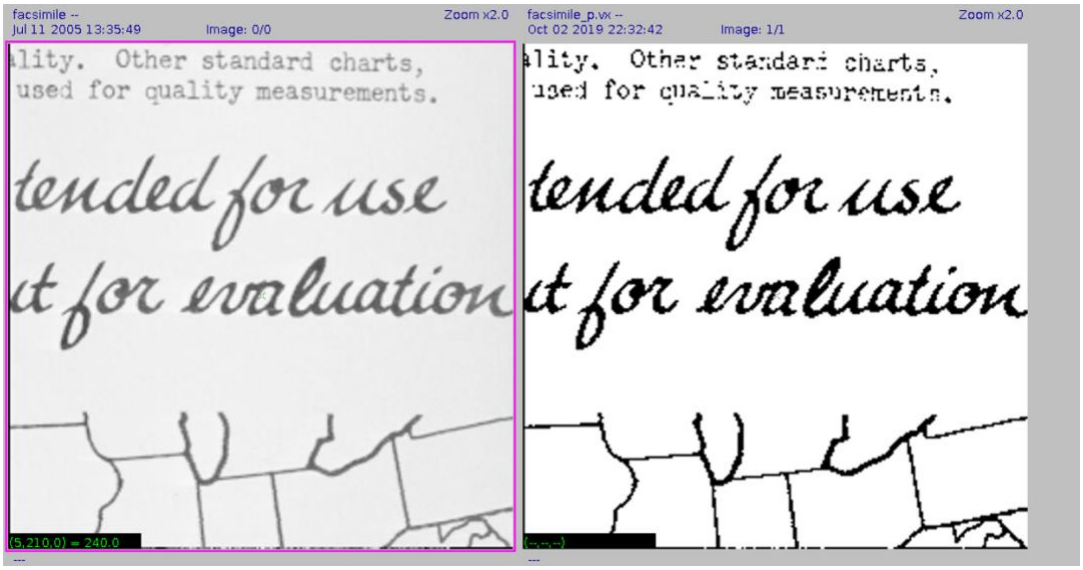


Figure 2: raw image, output image

From the results above, we can find that this vit program can make the image brighter through the function it apply. The effect of this program differs from images to images. As for some images, after the process of the program, it can show more details and the quality is improved, but for some others, it may decrease their image quality. Besides, if increase the count_time, the effect of the program can be much better, because the threshold calculated by the program can be more fitable.

Part 6. The process of Adaptive Thresholding

1. Adaptive Thresholding can take a grayscale or color image as input and outputs a binary image which represents the segmentation.

2. Firstly, the image will be decomposed into some patches, and a separate threshold will be computed through these patches. In this part, we can use the vpatch command to decompose the image into a sequence of overlapping patches.

3. Then we will use the vquilt command with appropriate parameters, which will resemble the patches back into the image.

4. In order to achieve the best algorithm performance, we need to generate the suitable parameters for the algorithm, which requires us to run a script file (which includes a set of different parameters) to generate them.

5. At last, we identify the best parameters for the images by reviewing the image set.

Part 7: The observation on the result of the Adaptive Thresholding algorithm in “map” and “mp.vx”

A. “map.vx” result

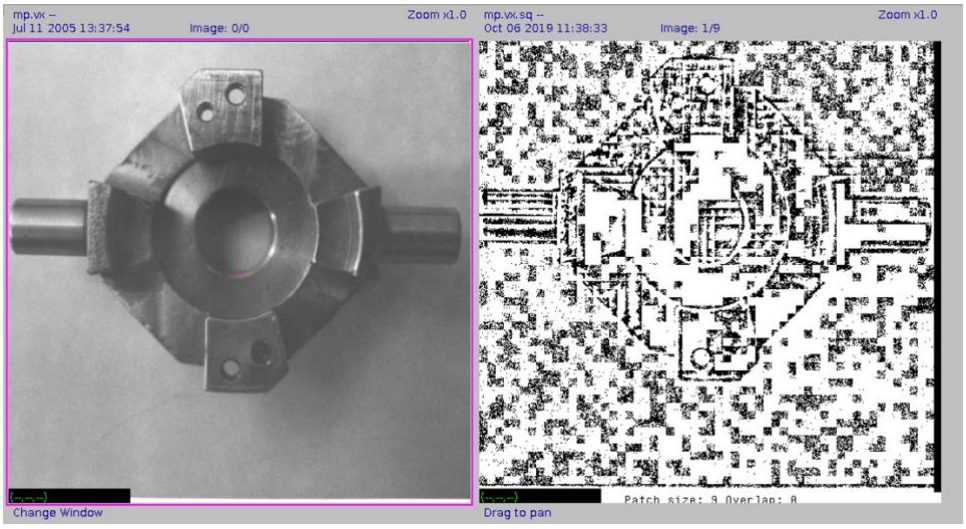
The image sequence generated by the algorithm is as followed:

1: patch size-9 overlap-0. 2: patch size-9 overlap-4. 3: patch size-9 overlap-8

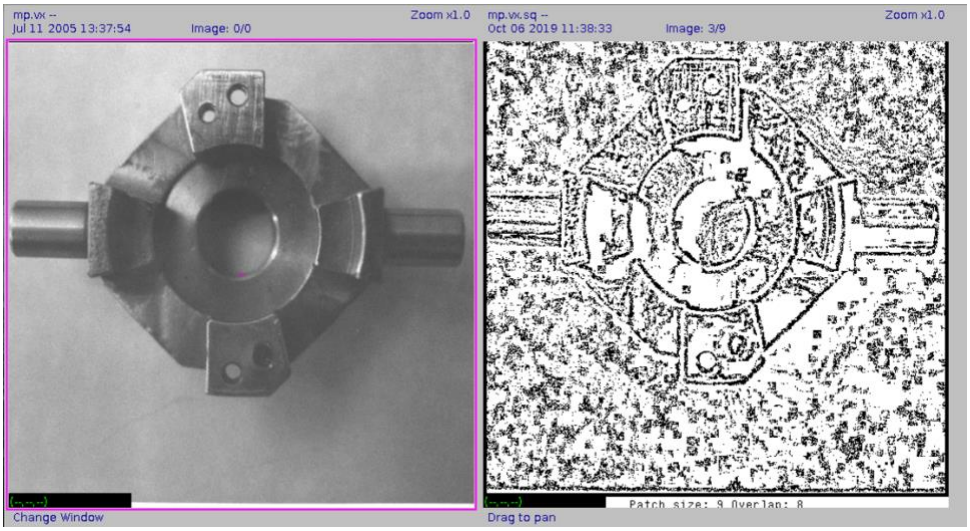
4: patch size-16 overlap-0 5: patch size-16 overlap-4. 6: patch size-16 overlap-8

7: patch size-24 overlap-0. 8: patch size-24 overlap-4. 9: patch size-24 overlap-8

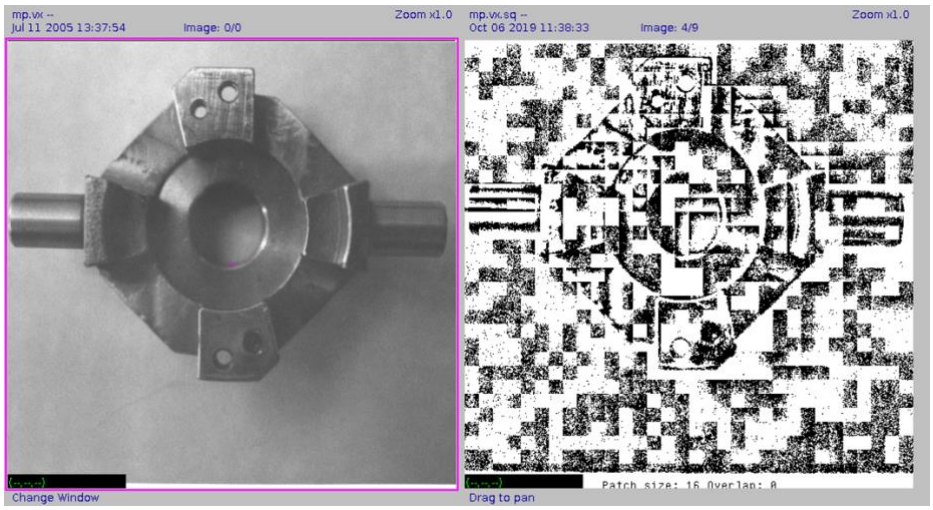
B. “mp.vx” result:



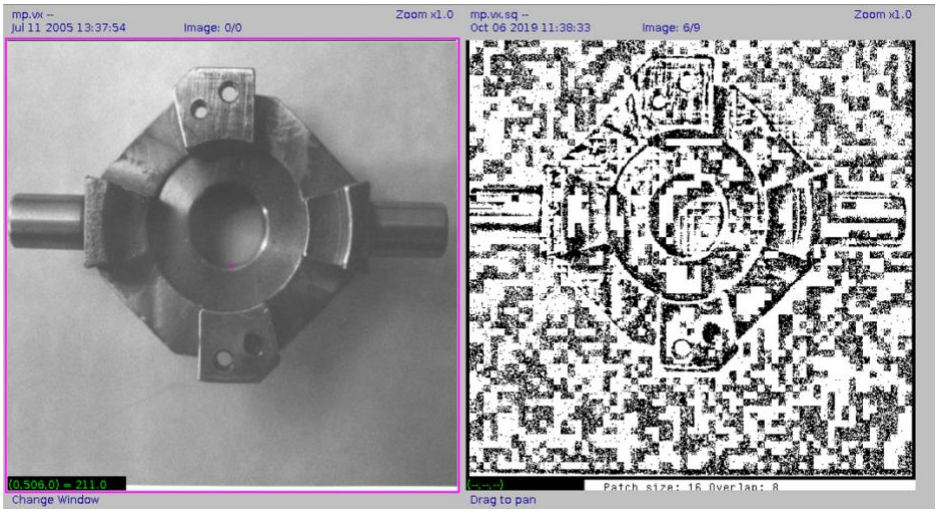
1. patch size-9, overlap-0



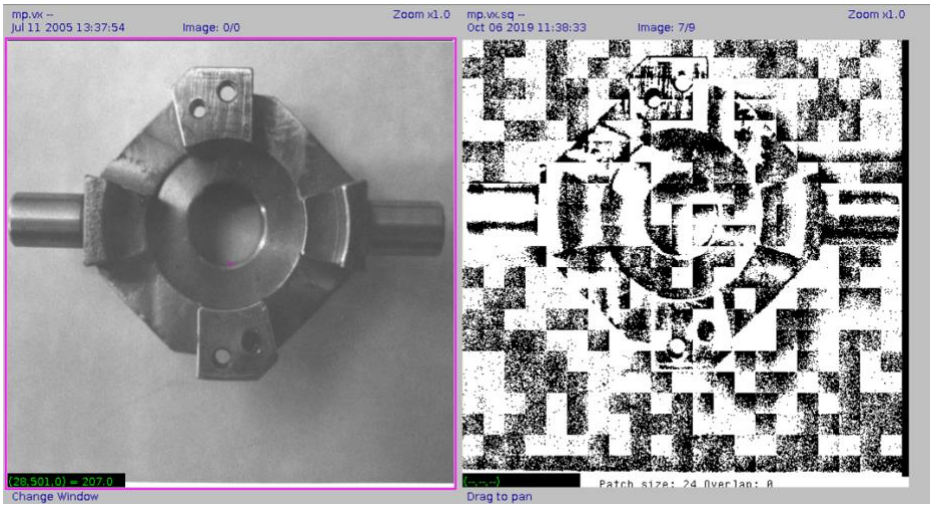
2. patch size-9, overlap-8



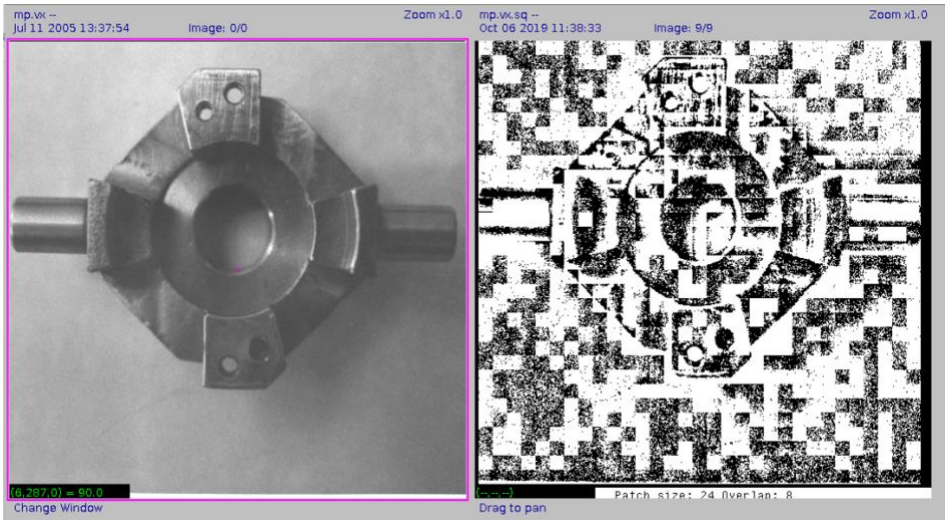
3. patch size-16, overlap-0



4. patch size-16, overlap-8



5. patch size-24, overlap-0



6. patch size-24, overlap-8

From the results above, I found that the image quality after processed is determined by the patch size and overlap. The smaller the patch size is, or the larger the overlap is, the better the image quality.

Part 8: The description of the Region Growing program on Section 5

In the Section 5, it is required that we need to write a program to generate a output image which labels the different object part with the pixel value detected first, to be more detail, the part's pixel value which located in the range is labeled as the same value. And the pixels from the same connective part have the same label. The main principle of the program is as followed:

1. Firstly, the program takes the input image and store it in "&im". Then use Vfembed command to extend the boundary of the im, and store the new image to "&tm". Now because we have the image stored in "&tm", we can set the pixels of the image stored in "&im" all to zero. The reason we do this is because we need to generate the labeled image and store it in "&im".
2. Then we search the image in "&tm", until we found the object pixel whose pixel value is not zero. Meanwhile the pixel must not be labeled. When we found that, we make another judgement, if "-p" is in the command, we call the set label function "setlabel_value (y,x,first)", "first" stand for the first pixel's value. But if "-p" is not in the command, we call the traditional set label function "setlabel_seq (y,x,L)", which labels the different parts of the image with sequencial number, the detail is the same as the lab2.
3. The "setlabel" function implements the recursive method, firstly, we set the global variable first to be the first not labeled pixel value, and search the four pixels around the pixel "tm.u[y][x]", when one of them is in the range, we call the "setlabel_value" function again until there is no surrounded pixels' value in the range. Then we begin with a new "first". This is the recursive method. After all the pixels are labeled, the program exits.

Part 9: The listing of the program vgrow.c

```
/*****************************************************************/
/* vtemp      Compute local max operation on a single byte image   */
/*****************************************************************/

#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,   " region pixel range"},
{    "-p",    0,   " flag label" },
{     0,       0,   0}  /* list termination */
};
#define  IVAL   par[0].val
#define  OVAL   par[1].val
#define  RANGE  par[2].val
#define  FLAG   par[3].val
void setlabel_seq(int, int,int);
void setlabel_value(int, int, int);
Vfstruct (im);                       /* i/o image structure          */
Vfstruct (tm);                       /* temp image structure         */
int range;
int first;
main(argc, argv)
int argc;
char *argv[];
{
  int        y,x;                     /* index counters               */
  int L = 1;
  VXparse(&argc, &argv, par);        /* parse the command line       */
  if (RANGE) range = atoi(RANGE);
  Vfread(&im, IVAL);                 /* read image file              */
  Vfembed(&tm, &im, 1,1,1,1);        /* image structure with border  */
  if ( im.type != VX_PBYTE ) {       /* check image format           */
     fprintf(stderr, "vtemp: no byte image data in input file\n");
     exit(-1);
  }

  for (y = im.ylo ; y <= im.yhi ; y++) {
     for (x = im.xlo; x <= im.xhi; x++)  {
           im.u[y][x] = 0;
     }
   }
  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(FLAG){
                        setlabel_value(y,x,first);
                                }
               else {setlabel_seq(y,x,L);
                     if(L>255){L = 1;}
                     else {L = L + 2;}
       }
     }
   }
}
   Vfwrite(&im, OVAL);             /* write image file              */
   exit(0);
}
```

```
void setlabel_value(int y, int x, int L)
{
    im.u[y][x] = first;
    if ((tm.u[y+1][x] != 0) && (abs(tm.u[y+1][x] - first) < range) && (im.u[y+1][x] == 0)){
        setlabel_value(y+1,x,first);}
    if ((tm.u[y-1][x] != 0) && (abs(tm.u[y-1][x] - first) < range) && (im.u[y-1][x] == 0)){
        setlabel_value(y-1,x,first);}
    if ((tm.u[y][x-1] != 0) && (abs(tm.u[y][x-1] - first) < range) && (im.u[y][x-1] == 0)){
        setlabel_value(y,x-1,first);}
    if ((tm.u[y][x+1] != 0) && (abs(tm.u[y][x+1] - first) < range) && (im.u[y][x+1] == 0)){
        setlabel_value(y,x+1,first);}
}
void setlabel_seq(int y, int x, int L)
{
    im.u[y][x] = L;
    if ((tm.u[y+1][x] != 0) && (abs(tm.u[y+1][x] - first) < range) && (im.u[y+1][x] == 0)){
        setlabel_seq(y+1,x,L);}
    if ((tm.u[y-1][x] != 0) && (abs(tm.u[y-1][x] - first) < range) && (im.u[y-1][x] == 0)){
        setlabel_seq(y-1,x,L);}
    if ((tm.u[y][x-1] != 0) && (abs(tm.u[y][x-1] - first) < range) && (im.u[y][x-1] == 0)){
        setlabel_seq(y,x-1,L);}
    if ((tm.u[y][x+1] != 0) && (abs(tm.u[y][x+1] - first) < range) && (im.u[y][x+1] == 0)){
        setlabel_seq(y,x+1,L);}
}
```

Part 10: The result of vgrow working on a small test image

I copied the testim1.vx from lab2 file to the lab3 file, and used this image as the test image, the input image and output image are as followed, the range is set to be 20, the first one with “-p”, second one without “-p” :

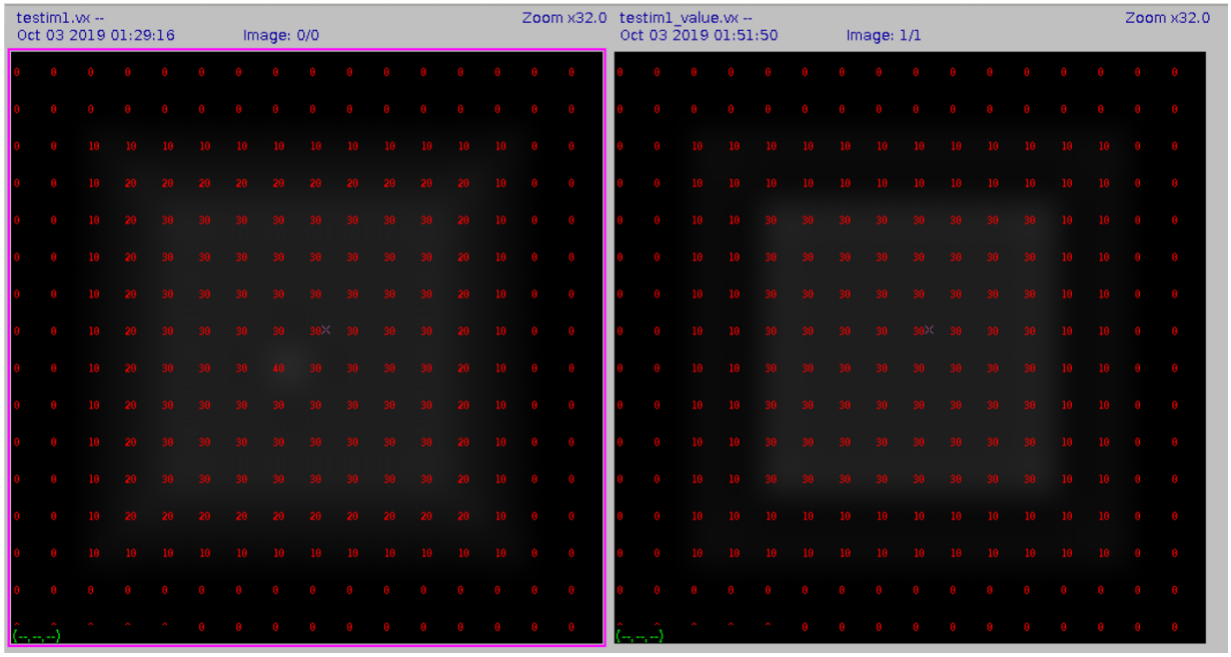


Figure1: range=20, with “-p
”

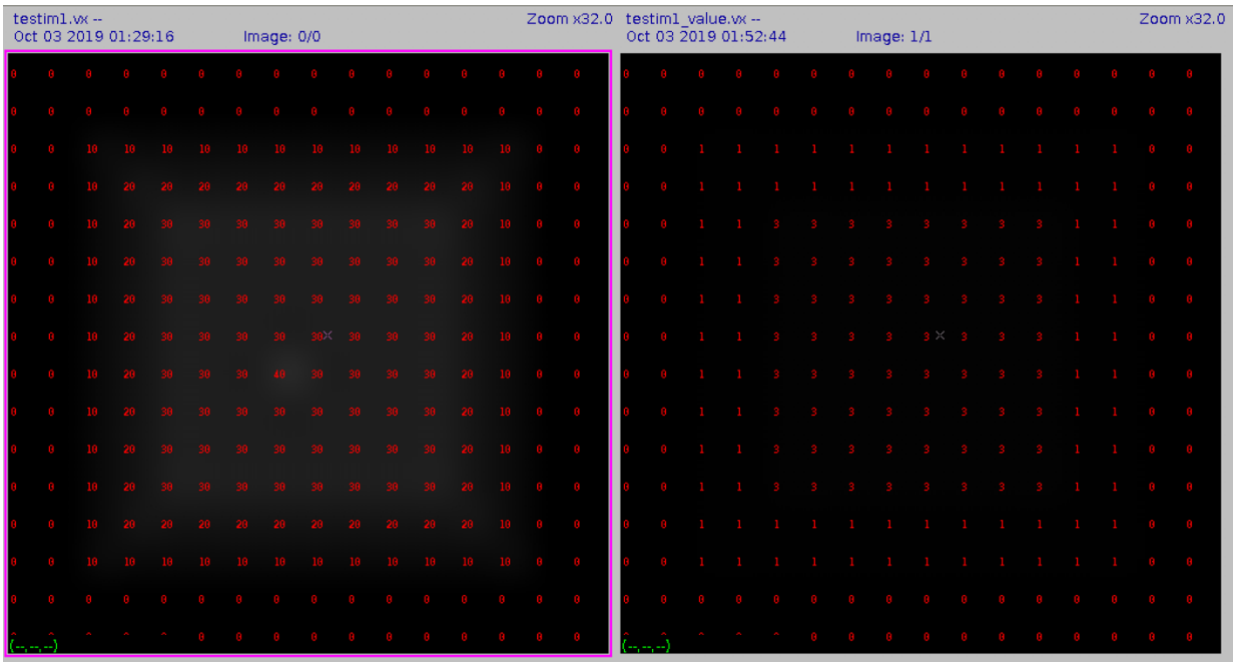
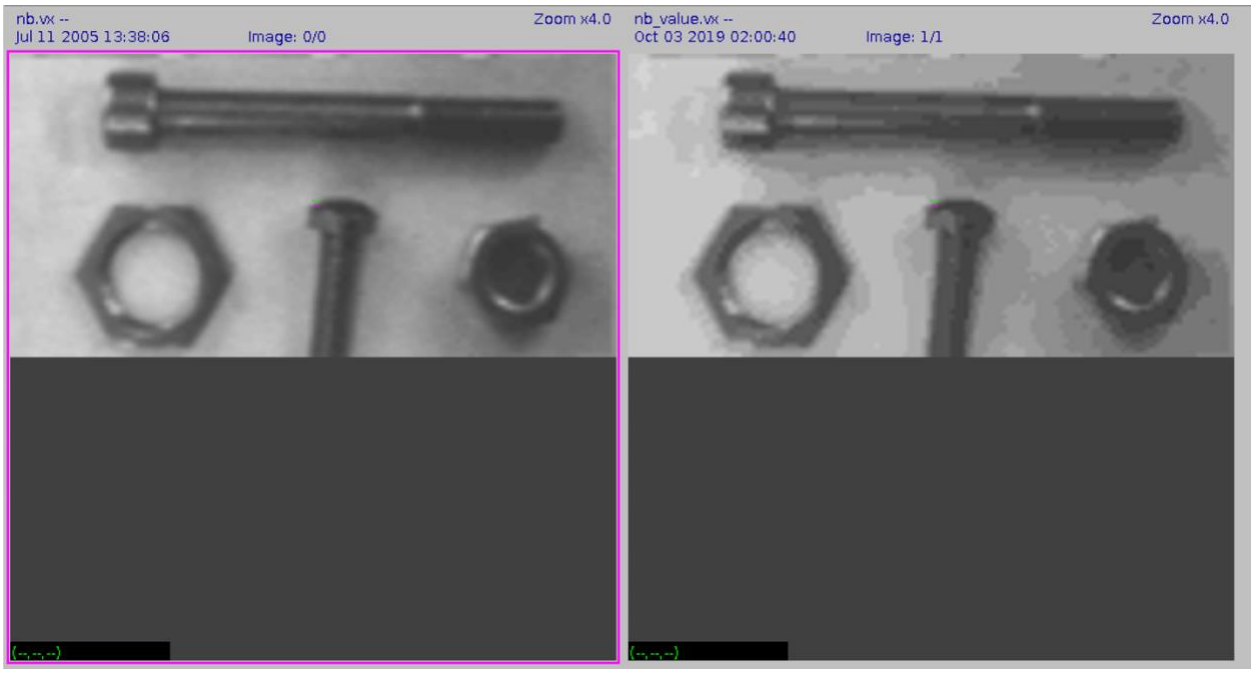


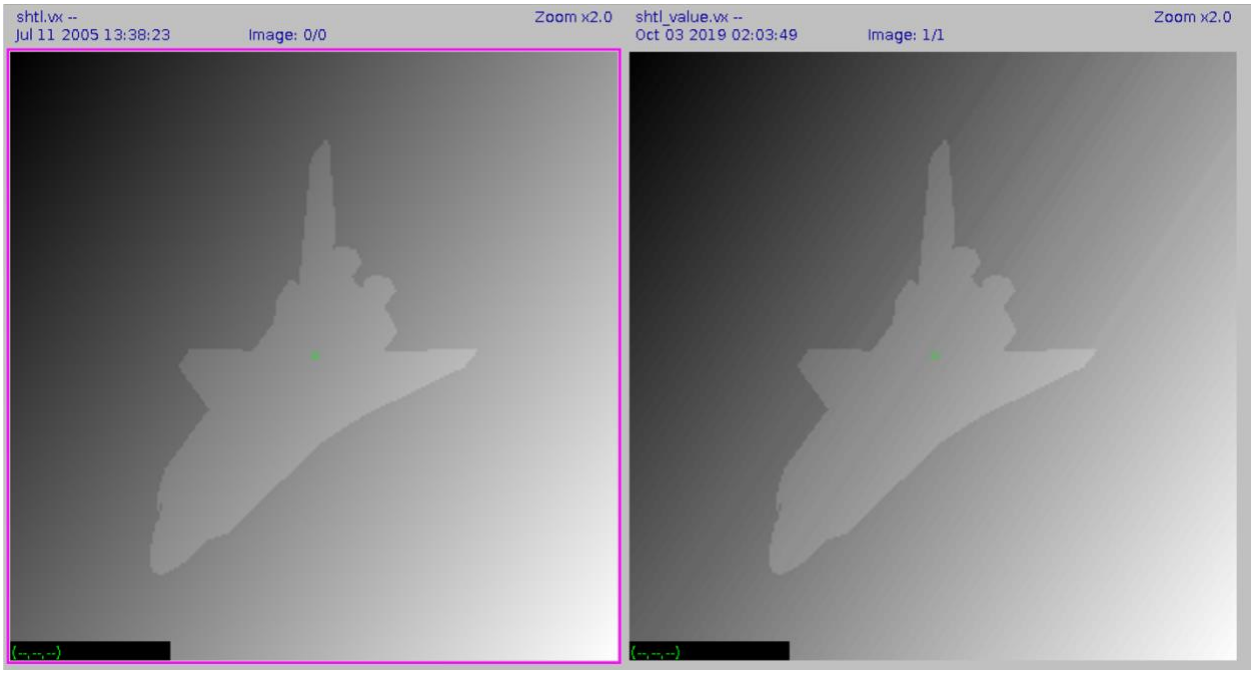
Figure 2: range=20, without “-p”

Part 11: The result of vgrow working on a image nb.vx and shtl.vx

A. input image is nb.vx, and the range is set to be 10:

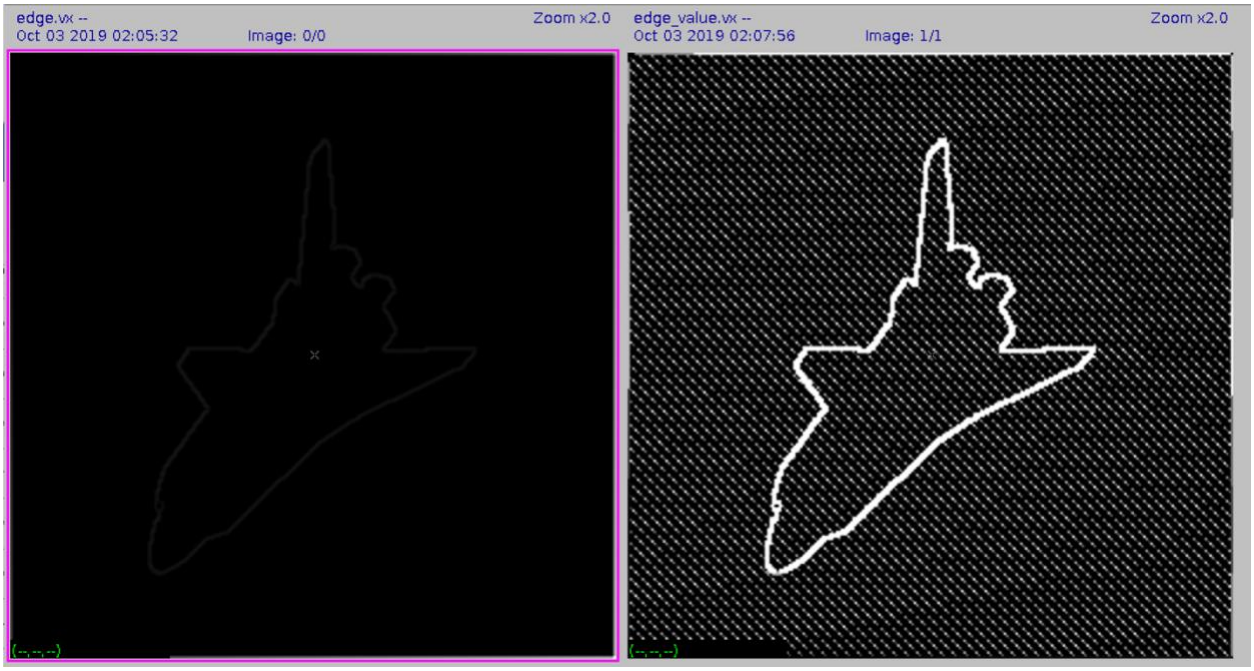


B. input image is shtl.vx, and the range is set to be 3:



Part 12: The result of vgrow working on the edge detected image edge.vx

A. Firstly, use the command " vsobel shtl.vx of=edge.vx" to generate shtl.vx 's edge detected image, and then execute vgrow on edge.vx image (without "-p"), and the best segmentation range value is chosen to be 15.



B. The generation of this segmented image:
The edge detecting command help highlight the edge of the object in the image shtl.vx. This is the initial segmentation of the image. Then we use region grow function to the image. Through the setting of an appropriate range, the edge of the object can be highlighted again with great brightness, and the region inside and outside of the edge could be set to the same label, which generated the final segmented image.