EE5470 Computer Vision Lab 3

Yuzhuo Sun NetId: ys684

Binary Image Processing

Lab 3 Questions

1, Your observations on each of the images tested in section 2 that indicate strengths and weaknesses of the Peakiness Algorithm.

A: Strength: (1) the Peakiness Algorithm is easy and simple to implement (2) Good performance on simple setting images. Weakness: (1), If the histogram computed is not ideally consisted by 2 peaks and 1 valley in the middle, it won't work properly. (2), If the image to be processed contains multiple objects with different graylevels, the whole image will be highly affected by the major object and the minor details will be severely lost.

2, A written description of how your Iterative Threshold program, vits, developed in section 3, works.

A: The whole skeleton code is from vtpeak (the Peakiness Algorithm), we first use the average graylevel of the whole image as the threshold. And then compute the average graylevel of the "object" avg1 and "background" avg2 from the threshold that we computed, and then compute the threshold again from the average of avg1 and avg2. I defined the program to do an iterative operation until the average value of avg1 and avg2 equals the threshold.

3, The listing of your program vits.c.

```
/*****************************************************************/
/* vits:     Threshold image using iterative threshold    */
/*****************************************************************/

#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,    " threshold value"},
{  "-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,j;

int hist[256];                  /* histogram bins                */
int thresh;                     /* threshold                     */
int avg1 = 0;
int avg2 = 0;
int sum1 = 0;
int sum2 = 0;
int cnt1 = 0;
int cnt2 = 0;

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

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

if (TVAL) thresh = atoi(TVAL);  /* if thresh= was specified, get value */
    else{
    int sum = 0;
    for (y = im.ylo ; y <= im.yhi ; y++) {
        for (x = im.xlo; x <= im.xhi; x++)  {
        sum += im.u[y][x];
        }
    }
    thresh = sum / ((im.yhi - im.ylo + 1) * (im.xhi - im.xlo + 1));
    }
if (thresh < 0 || thresh > 255) {
    fprintf(stderr, "thresh= must be between 0 and 255\nUsing d=10\n");
    thresh = 10;
}

    /* clear the histogram */
    for (i = 0; i < 256; i++) hist[i] = 0;

    /* compute the histogram */
    for (y = im.ylo; y <= im.yhi; y++)
        for (x = im.xlo; x <= im.xhi; x++)
             hist[im.u[y][x]]++;

     /* compute the threshold */

for(i = 0; i < 10000; i++){
    sum1 = 0;
    sum2 = 0;
    cnt1 = 0;
    cnt2 = 0;
    for(j = 0; j < thresh; j++){
    sum1 += j * hist[j];
    cnt1 += hist[j];
    }
    for(j = 255; j >= thresh; j--){
    sum2 += j * hist[j];
    cnt2 += hist[j];
    }
    if(cnt1 == 0)   avg1 = 0;
    else    avg1 = sum1/cnt1;
    if(cnt2 == 0)   avg2 = 0;
    else    avg2 = sum2/cnt2;
    if(thresh == (avg1 + avg2) /2){
```

```
            fprintf(stderr, "Break at %d,%d\n", i, thresh);
        break;
        }
        else    thresh = (avg1 + avg2) / 2;
    }

        /* 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);
}
```

4, A copy of the typescript that shows your vits program working for a small test image. This typescript must show both the input image and the output image.

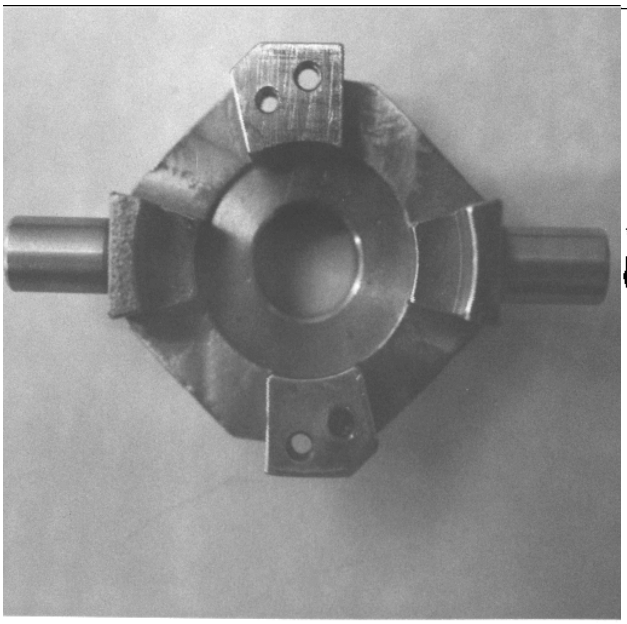
Input image:

```
   0  1  2  3  4  5  6

 5  0   255 255  0    0    0    0

 4 255 128 128 255  0    0    0

 3  0   255 128 255 255  255  0

 2  0    0   255 255 128 128 255

 1  0    0   255 128 128 128 255

 0  0    0    0  255  255 255 128
```

Output image

```
   0    1    2    3    4    5    6

 5  0  255 255  0    0    0    0

 4 255 255 255 255  0    0    0

 3  0  255 255 255 255 255  0

 2  0   0  255 255 255 255 255

 1  0   0  255 255 255 255 255

 0  0   0   0  255 255 255 255
```

5, A page with image figures and comments that shows that your program works on a full size image (show several result images).

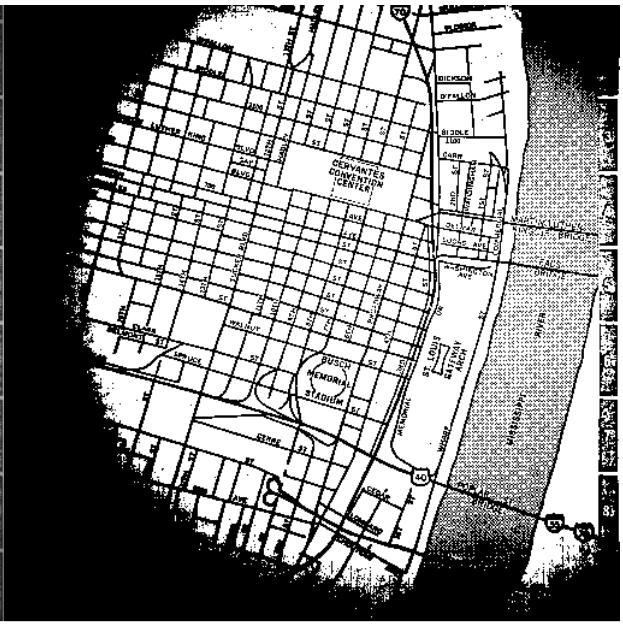


Input image mp.vx

Iterative thresholding processed image of mp.vx

The Iterative Threshold process demonstrates good performance on mechanical part image shown above. Since the object part have a good contrast with the background setting. The flaw on the image above is the lower right corner where there is shadow of the mechanical part. The shadow actually is considered to be in the object since the graylevel is higher compared with the other part of the image.

Input image of map.vx



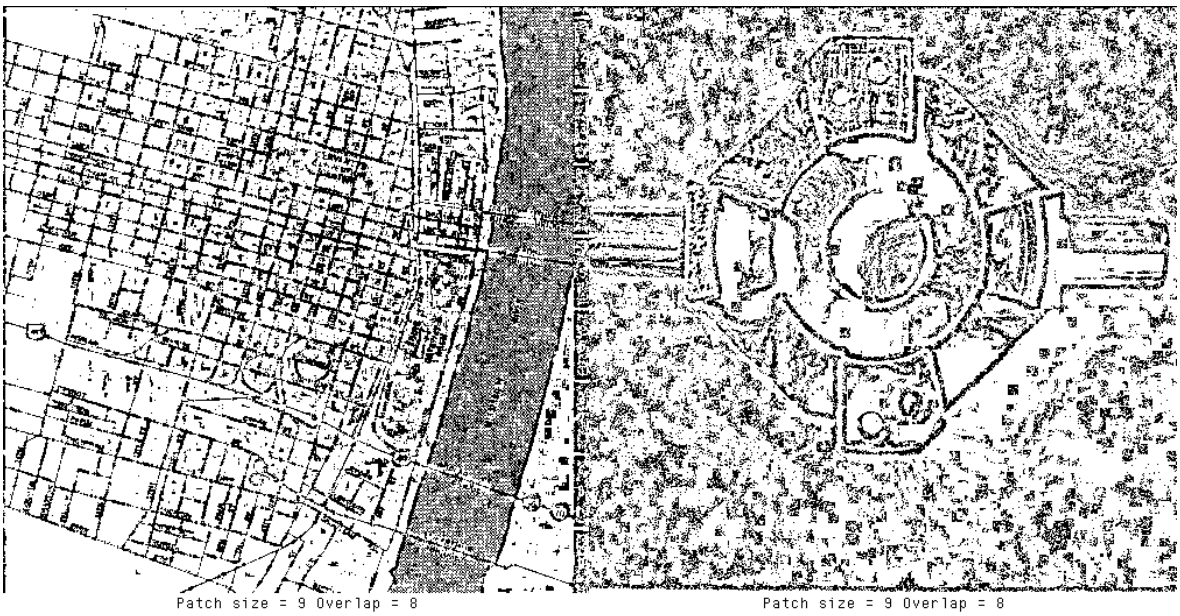
Iterative thresholding processed image of map.vx

The Iterative Threshold process works on the map image too, but due to this process only separates the image into background and object based on the average graylevel calculated from histogram. The dark corners on the processed image is due the unbalance of brightness.

6, A brief description of how the process of Adaptive Thresholding works.

A: The vpatch command will decompose and image into a set of overlapping patches, and we could specify the patches' dimensions by passing value to p variable, and also specify the overlapping between adjacent patches by passing value to l variable. The vits will do iterative thresholding to each patch from from vpatch command. Then vquilt command will Construct an image from patches. So, the basic idea of this Adaptive Thresholding process is to using the divider and conquer strategy to split the image into several patches. Then use basic thresholding method like Iterative Threshold or Peakiness Algorithm to process each of the patches. Finally, merge them back into one image.

7, Your observations on the results of the Adaptive Thresholding algorithms on map and mp.vx as described in section 4.



A: The images shown above are the best result computed from Adaptive Thresholding with Peakiness Algorithm. As the patch size grows, the original image will be separated into bigger patches, and if the overlap is set higher value, the constructed image will be smoother in the boarder of adjacent patches. As the peakiness algorithm is not so accurate in calculating complex setting images, it's better to split the original image into small patches to make the peakiness algorithm more accurate. Also we should set higher overlap value to make the boarder between patches smoother. So, if we set the parameter patch size to be 9 and overlap to be 8, though it will have some noise on the image. But it is relatively the best option.

8, A written description of how your Region Growing program, vgrow, developed in section 5, works.

A: The skeleton code is from cclabel program. The idea to implement this algorithm is that we will use DFS strategy to find each no-zero graylevel pixel to find the region it belongs to. So, I implemented the program as following: first, I construct a duplicate of the input image; set all pixel of input image to 0, and if a pixel on input image and duplicated image are both 0, means it's unlabeled, and a recursive setlabel function will be called; in the setlabel function, I will calculate each of it's 4 neighbors to see if the difference between them are within the range given, if so, I will mark them belongs to the same object.

9, The listing of your program vgrow.c.

```
/*****************************************************************/
/* vgrow          Region growing on a single byte image          */
/*****************************************************************/

#include "VisXV4.h"          /* VisionX structure include file       */
#include "Vutil.h"           /* VisionX utility header files         */
VisXfile_t *VXin,            /* input file structure                 */
           *VXout;           /* output file structure                */
VisXelem_t *VXlist,*VXptr;   /* VisionX data structure               */
VXparam_t par[] =            /* command line structure               */
{
{  "if=",    0,    " input file, vgrow: "},
{  "of=",    0,    " output file "},
{  "r=",     0,    " set the region pixel range"},
{  "-p",     0,    " value of the first pixel in the region"},
{   0,       0,    0} /* list termination */
};
#define  IVAL   par[0].val
#define  OVAL   par[1].val
#define  RANGE  par[2].val
#define  PFLAG  par[3].val

int first;

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

main(argc, argv)
int argc;
char *argv[];
{
    int start;
    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  */
    int label = 1;                        /* region sequence              */
    first = im.u[im.ylo][im.xlo];

    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(im.u[y][x] == 0 && tm.u[y][x] != 0){
        first = tm.u[y][x];
        if(PFLAG){
            setlabel(x, y, first);
        }
        else{
            setlabel(x, y, label);
            if(label == 255){
            label = 1;
            }else
            label++;
        }
        }
        }
    }

    Vfwrite(&im, OVAL);             /* write image file              */
    exit(0);
}

/* function to set the label */
void setlabel(int x, int y, int l)
{
    im.u[y][x] = l;
    if((tm.u[y][x + 1] != 0 && im.u[y][x + 1] == 0) && abs(tm.u[y][x + 1] - first) <
atoi(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) <
atoi(RANGE))
    setlabel(x - 1, y, l);
    if((tm.u[y + 1][x] != 0 && im.u[y + 1][x] == 0) && abs(tm.u[y + 1][x] - first) <
atoi(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) <
atoi(RANGE))
    setlabel(x, y - 1, l);
}
```

10, A copy of the typescript that shows that your program vgrow works for a small test image.

Input image

Command used: vppr small.mx

```
   0   1   2   3   4   5   6

 5 0 255 255 0   0   0   0

 4 255 128 128 255 0   0   0

 3 0 255 128 255 255 255 0

 2 0   0   255 255 128 128 255

 1 0   0   255 128 128 128 255
```

0 0 0 0 255 255 255 128

Output image set the range to be 128

Command used: vgrow small.mx r=128 | vppr

```
   0  1  2  3  4  5  6
5  0  1  1  0  0  0  0
4  1  1  1  1  0  0  0
3  0  1  1  1  1  1  0
2  0  0  1  1  1  1  1
1  0  0  1  1  1  1  1
0  0  0  0  1  1  1  1
```

Output image set the range to be 100

Command used: vgrow small.mx r=100 | vppr

```
   0  1  2  3  4  5  6
5  0  9  9  0  0  0  0
4  8  7  7  3  0  0  0
3  0  6  7  3  3  3  0
2  0  0  3  3  4  4  5
1  0  0  3  4  4  4  5
0  0  0  0  1  1  1  2
```

11, A page with image figures and comments that shows that your program vgrow works on the images nb.vx and shtl.vx. Indicate the range value used for each.

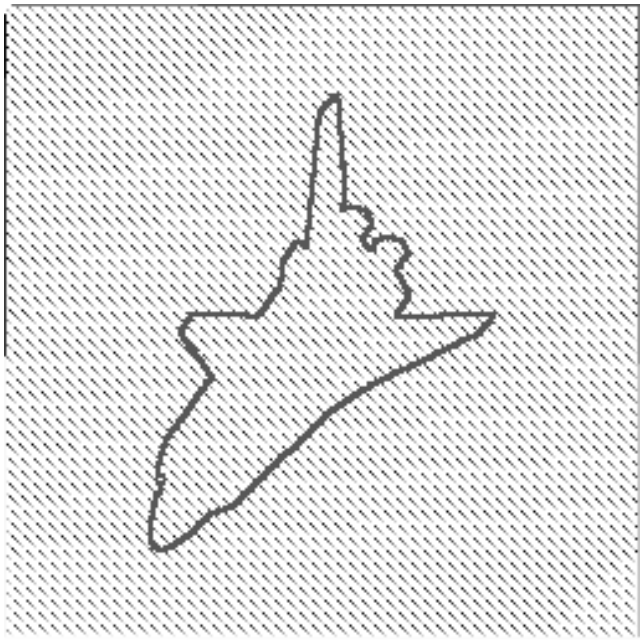
Region grow processed of nb.vx with range = 43:

range = 43

Region grow processed of shtl.vx with range = 15:

range = 15

12, A page with image figures and comments showing the segmented shtl image after executing vgrow on the edge detected image. Describe, in general terms, how this segmented image was generated.



range = 30

A: The vsobel operation did the edge detection. This operation detects the edge of the space shuttle, but due to the step by step change in the background, it also detects small bounds and marked them as 1. As we use vgrow operation afterwards, it detects the edge of the space shuttle is in one region. But for the small bounds with 1, as they are diagonal adjacent, the vgrow operation will mark each of them as different regions. And as label value is changing, we got the dots besides the original edge of the space shuttle.