Computer Vision
Lab3
ECE5470

Yuqing Shou
ys895@cornell.edu

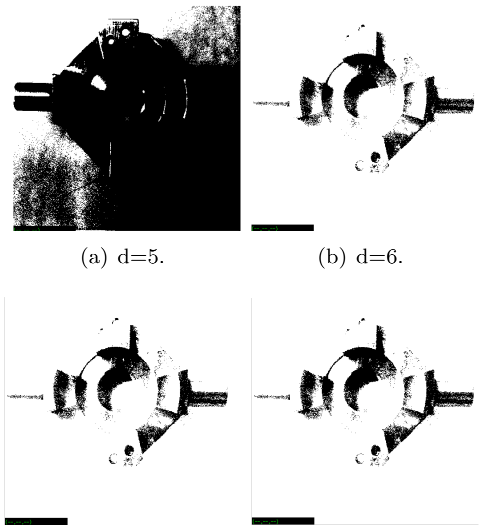
9/27/2018

Instructor: Professor Reeves, A

Section2 Peakiness Detection

The output pictures are shown in the below, it shows that the Peakiness algorithm faces a lot of disadvantages in the field of image thresholding. The conclusion is below: The Peakiness Algorithm has the advantages that it can threshold the image pretty easy without those time consuming operation doing on the image, however, it is also one of the disadvantages in a different view. From the Figure2, we get that when d=6 and d=5, there is a big gape between the quality of the images.When d=6 the pixels in low part of the object disappeared, when d=6, the low part of the objects become too dark to be observed, it reveals one of the most important disadvantages of Peakiness Algorithm: people have to exert time to find the most suitable threshold, which is time consuming.

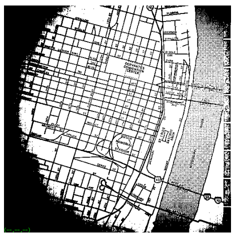
From the Figure 3,4,5, we can get that the Peakiness Algorithm's threshold influenced the clearness of images significantly, different pictures shows different suitable thresholds. Some of pictures shows that the thresholds also influence the people's perception.

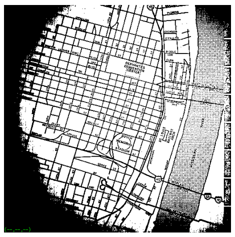


(a) d=5. (b) d=6.

(c) d=7. (d) d=10.

Figure 1: Picture of mp


(a) d=6.


(b) d=10.

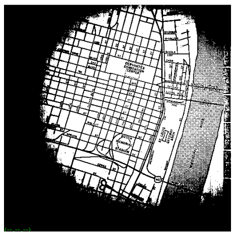
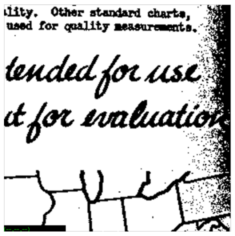
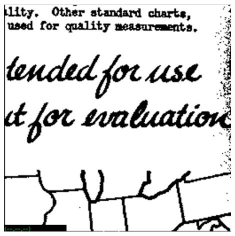

(c) d=30.

Figure 2: Picture of map



(a) d=5.



(b) d=8.

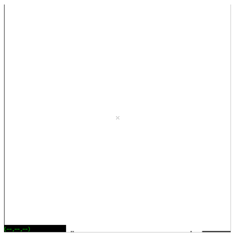

(c) d=10.

Figure 3: Picture of facsimile

(a) d=10. (b) d=20.

Figure 4: Picture of shtl

Section3 Iterative Threshold Selection

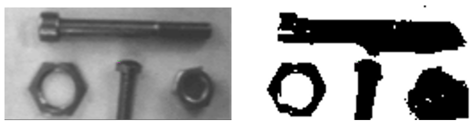
The algorithm uses a while loop with for loop in it, and the behind algorithm is relatively simple that it does require any complex operation or algorithm, it basically computes two average value of a picture which is divided into two parts: one part with the pixel value bigger than the threshold and the other with pixel value smaller than the threshold value.

The code of vits.c:

```
#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 "},
{  "d=",     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  DVAL   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 hist[256];                  /* histogram bins
*/
    int thresh=10;                     /* threshold
*/
    int maxbin;                     /* maximum histogram bin
*/
```

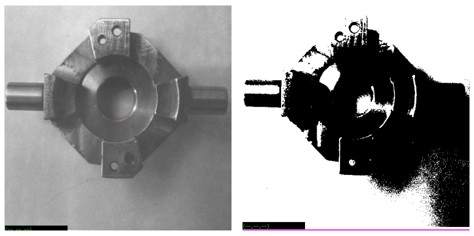
```
    int nxtbin;                         /* second maximum bin
*/
    int minbin;                         /* minumim histogram bin
*/
    int maxa, maxb;          /* second maximum bin above/below maxbin
*/
    int dist;                       /* minimum distance between maxima
*/
    unsigned int sum1=0;
    unsigned int sum2=0;
    unsigned int counter1=0;
    unsigned int counter2=0;
    int counter=0;
    VXparse(&argc, &argv, par);      /* parse the command line
*/

    dist = 10;                         /* default dist */
    if (DVAL) dist = atoi(DVAL);   /* if d= was specified, get value */
    if (dist < 0 || dist > 255) {
        fprintf(stderr, "d=␣must␣be␣between␣0␣and␣255\nUsing␣d=10\n");
        dist = 10;
    }

    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[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]]++;

         /* find maximum bin for the entire histogram */
         //maxbin = 0;
         //for (i = 0; i <= 255; i++)
         //     if (hist[i] > hist[maxbin]) maxbin = i;

           /* compute the threshold */
        while(counter<20)
        {
             int i;
```

```
                for(i=0;i<=255;i++)
                {
                        if(i>thresh)
                                {
                                sum1=sum1+i;
                                counter1=counter1+1;
                                }
                        else
                                {
                                sum2=sum2+i;
                              counter2=counter2+1;
                                }
//fprintf(stderr, "sum1 = %d\n", sum1);
                thresh=(sum1/counter1+sum2/counter2)/2;
                sum1=0;
                sum2=0;
                counter1=0;
                counter2=0;
                counter++;
              }

        }
        //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);
}
```



(a) original (b) after thresholding

Figure 5: vits with small image

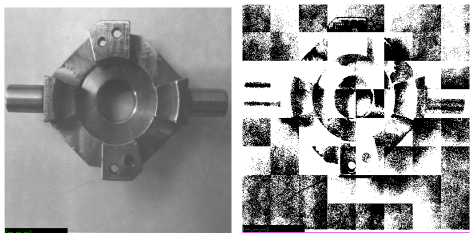


(a) original (b) after thresholding

Figure 6: vits with normal image

Section4 Adaptive Thresholding

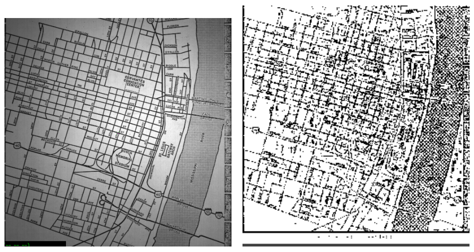
thresholding is used to segment an image by setting all pixels whose intensity values are above a threshold to a foreground value and all the remaining pixels to a background value. Whereas the conventional thresholding operator uses a global threshold for all pixels, adaptive thresholding changes the threshold dynamically over the image. This more sophisticated version of thresholding can accommodate changing lighting conditions in the image, e.g. those occurring as a result of a strong illumination gradient or shadows.



(a) original (b) mp

Figure 7: Adaptive thresholding

The best value found in the exmaple of mp.vx is: patch value:9, overlap value:8



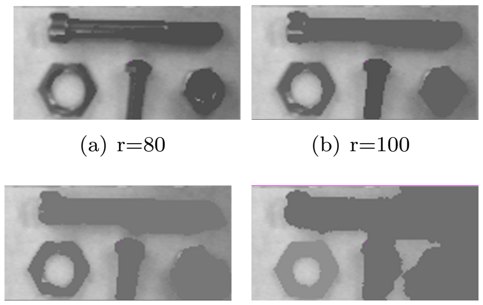
(a) original (b) mp.sq

Figure 8: Choosing the best value

Section5 Region Growing

5.1 Region growing on small images

The program of region growing is mainly about recursive application, finding the condition to continue the region grow unless the conditions are not satisfied, then return. My program update the first pixel value in the iteration operation did on everyone pixel of the input image and then use recursive operation to update the label of the input image, which was done without redundant operation of coding and is pretty easy. When it comes to parser, I finish it by writing more parameters into the data structure in the vision called VXparam_t.



(a) r=80

(b) r=100

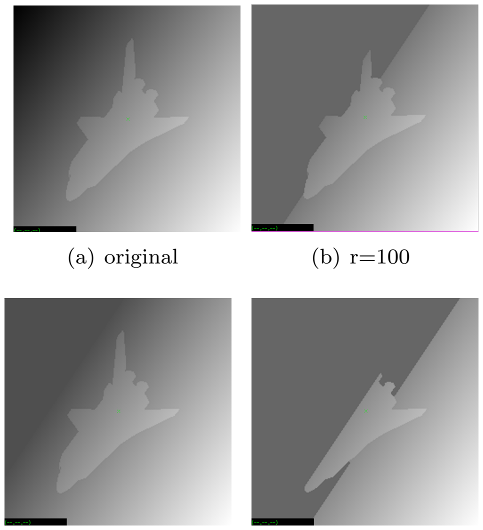
(c) r=120

(d) r=150.

Figure 9: Region growing of nb.vx

5.2 Region growing on image shtl.vx

From the Figure below, we can easily get the best value r for different pictures and how the Adaptive Thresholding works in different pictures. For the shtl.vx image, due to the color of the background is not static and varies in a linear way, so when applying Adaptive Thresholding, you can see that the background of the image was divided into several different color parts, such as the picture when r=120 and r=140.



(a) original (b) r=100

(c) r=120 (d) r=140

Figure 10: Region growing of shtl.vx