ECE637 Lab report 3
Neighborhoods and Connected Components

Name:Chengzhang Zhong

Section 1. Area Fill

Part 1. The gray scale image *img22gd2.tif*

Part 2. image showing the connected set for s = (67, 45), and T = 2,T = 1,T=3

Figure 1. s = (67, 45), and T = 2

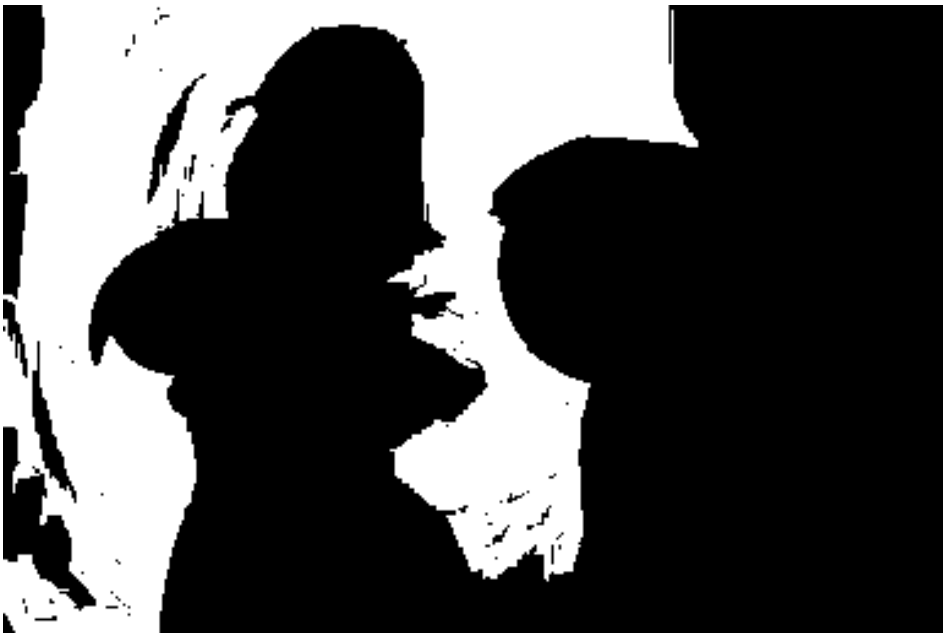

Figure 2. s = (67, 45), and T = 1

Figure 3. s = (67, 45), and T = 3

Part 3. Code for *fill in area*

```
#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"
#include <stdio.h>
#include <stdlib.h>




struct pixel
{
    int m,n; /* m=row, n=col */
};
void ConnectedSet(struct pixel s, double T, unsigned char **img, int height, int
width, int ClassLabel, unsigned int **seg, int *NumConPixels);
void ConnectedNeighbors(struct pixel s, double T, unsigned char **img, int height,
int width, int *M, struct pixel *c);

void ConnectedNeighbors(
    struct pixel s,
    double T,
    unsigned char **img,
    int height,
    int width,
    int *M,
    struct pixel *c)
{
    //s.m only affect (x,y-1), (x,y+1) components.
    //s.n affects (x-1,y) and (x+1,y) components.

    int ind=0;
    //struct pixel cc;
    //cc.m = 3;
    //cc.n = 2;

    int x_comp[4] = {1,1,1,1}; // 1 means this component exists, 0 means to eliminate
this component
    int y_comp[4] = {1,1,1,1}; //4 component follow clockwise direction, [0] is top,
[1] is right,[2] is bottom, [3] is right
```

```
int x_y[4];

int x=0;
int y=0;
int count=0;

if (s.m == 0) {y_comp[0] = 0;}

else if (s.m == height-1) {y_comp[2] = 0;}

if (s.n == 0) {x_comp[3] = 0;}

if (s.n == width-1) {x_comp[1] = 0;}

for (ind=0; ind<4; ind++) //check the 4 neighbors arround pixel s
{
    x_y[ind] = x_comp[ind]*y_comp[ind];
    if (x_y[ind] == 1){
        switch(ind){
            case 0:{
                x = s.n;
                y = s.m - 1;
                break;
                   }
            case 1:
            {
            x = s.n+1;
            y = s.m;
            break;
            }
            case 2:
            {
            x = s.n;
            y = s.m + 1;
            break;
            }
            case 3:
            {
            x = s.n-1;
            y = s.m;
            break;
            }
        }
       if(abs(img[y][x]-img[s.m][s.n])<=T)
```

```
            {
                c[count].m = y;
                c[count].n = x;
                count++;
            }
        }
    }
    *M = count;
}

void ConnectedSet(struct pixel s, double T, unsigned char **img, int height, int
width, int ClassLabel, unsigned int **seg, int *NumConPixels)

{
    struct pixel *B = (struct pixel*)malloc(height*width*sizeof(struct pixel));
//dynamic size of B, use malloc

    struct pixel current;
    struct pixel c_c[4]; //input to connect_neighbor
    int start = 0; //double pointer idea to controll the size of B
    int end = 0 ;


    B[start] = s; // serve s as the first seed

    int ind,M_M;//index and the input to connect_neighbor

    seg[s.m][s.n] = ClassLabel;// initialize the first y

    while(start <= end)
    {
        M_M=0;
        ind=0;

        current = B[start];
        start++;

        ConnectedNeighbors(current, T, img, height, width, &M_M, c_c);

        for(ind=0; ind<M_M; ind++)
        {
              if(seg[c_c[ind].m][c_c[ind].n] == 0) //check the neighbors with those
locations at Y
              {
```

```
                end++;
                B[end] = c_c[ind];
                seg[c_c[ind].m][c_c[ind].n] = ClassLabel;
            }
            c_c[ind].m = 0;
            c_c[ind].n = 0;
        }

    }

    *NumConPixels = end;
    free(B);
}



int main (int argc, char **argv)
{
    FILE *fp;
    struct TIFF_img input_img, output_img;
    struct pixel s;
    double T = 3;
    int width;
    int height;
    int ClassLabel=1;
    int i,j;
    unsigned int **seg;
    unsigned char **img;
    int NumConPixels=0;

    int M = 0;
    struct pixel c[4];




    if ( ( fp = fopen ( "/home/min/a/zhongc/Desktop/ece637_lab3/img22gd2.tif",
"rb" ) ) == NULL ) {
        fprintf ( stderr, "cannot open file img22gd2.tif\n"  );
        exit ( 1 );
    }


    if ( read_TIFF ( fp, &input_img ) ) {
```

```
        fprintf ( stderr, "error open file img22gd2.tif\n" );
        exit ( 1 );
    }


    fclose ( fp );

    img = (unsigned char**) get_img(input_img.height, input_img.width,
sizeof(double));
    seg = (unsigned int**) get_img(input_img.height, input_img.width,
sizeof(double));
    for(i=0; i<input_img.height; i++)
    for(j=0; j<input_img.width; j++)
    {
        img[i][j] = input_img.mono[i][j];//for gray scale image
        seg[i][j] = 0;
    }
    width = input_img.width;
    height = input_img.height;
    s.m = 67;
    s.n = 45;
    ConnectedSet(s, T, img, height, width, ClassLabel, seg, &NumConPixels);


    get_TIFF ( &output_img, input_img.height, input_img.width, 'g' );

    for(i=0; i<input_img.height; i++)
    {
        for(j=0; j<input_img.width; j++)
        {
            output_img.mono[i][j] = seg[i][j]*255;
        }
    }

    if ( ( fp = fopen ( "output_image_T=3.tif", "wb" ) ) == NULL ) {
    fprintf ( stderr, "cannot open file color.tif\n");
    exit ( 1 );
        }


    if ( write_TIFF ( fp, &output_img ) ) {
      fprintf ( stderr, "error writing TIFF file\n" );
      exit ( 1 );
    }
```

```
    fclose ( fp );
    free_img( (void**)img );
    free_img( (void**)seg );
    free_TIFF ( &(input_img) );
    free_TIFF ( &(output_img) );
    return(0);
}
```

Section 2. Image Segmentation

Part 1. randomly colored segmentation for T = 1, T = 2, and T = 3

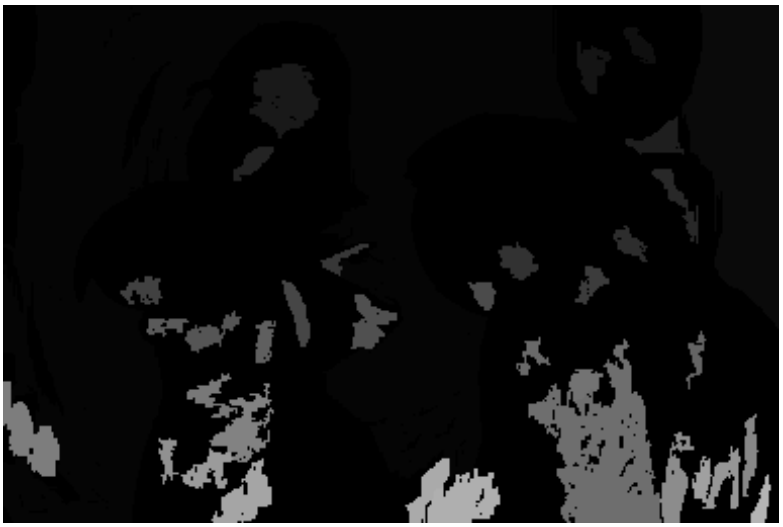

Figure 1. Gray-image, T = 1

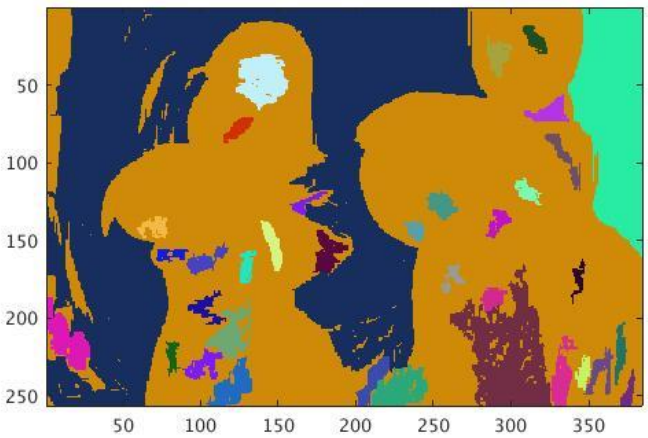

Figure 2. Random colored image, T = 1

Figure 3. Gray-image, T = 2

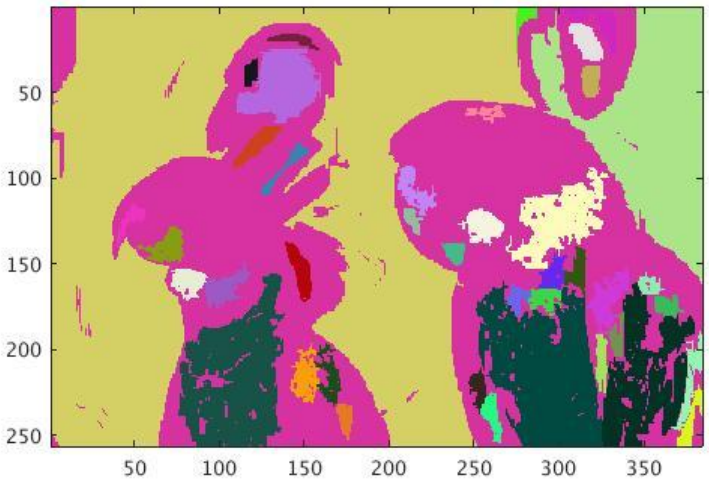

Figure 4. Random colored image, T = 2

Figure 5. Gray-image, T = 3

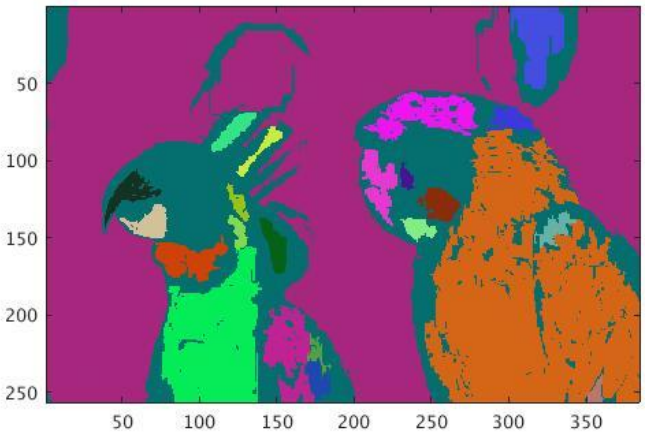

Figure 6. Random colored image, T = 3

Part 2. A listing of the number of regions generated for each of the values of T = 1, T = 2, and T = 3.

	Number of regions
T =1	36
T =2	41
T =3	23

Part 3. A list of code

```
#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"
#include <stdio.h>
#include <stdlib.h>




struct pixel
{
    int m,n; /* m=row, n=col */
};
void ConnectedSet(struct pixel s, double T, unsigned char **img, int height, int
width, int ClassLabel, unsigned int **seg, int *NumConPixels);
void ConnectedNeighbors(struct pixel s, double T, unsigned char **img, int height,
int width, int *M, struct pixel *c);

void ConnectedNeighbors(
    struct pixel s,
    double T,
    unsigned char **img,
    int height,
    int width,
    int *M,
    struct pixel *c)
{
    //s.m only affect (x,y-1), (x,y+1) components.
    //s.n affects (x-1,y) and (x+1,y) components.

    int ind=0;
    //struct pixel cc;
    //cc.m = 3;
    //cc.n = 2;

    int x_comp[4] = {1,1,1,1}; // 1 means this component exists, 0 means to eliminate
this component
    int y_comp[4] = {1,1,1,1}; //4 component follow clockwise direction, [0] is top,
[1] is right,[2] is bottom, [3] is right
```

```
int x_y[4];

int x=0;
int y=0;
int count=0;

if (s.m == 0) {y_comp[0] = 0;}

else if (s.m == height-1) {y_comp[2] = 0;}

if (s.n == 0) {x_comp[3] = 0;}

if (s.n == width-1) {x_comp[1] = 0;}

for (ind=0; ind<4; ind++) //check the 4 neighbors arround pixel s
{
    x_y[ind] = x_comp[ind]*y_comp[ind];
    if (x_y[ind] == 1){
        switch(ind){
            case 0:{
                x = s.n;
                y = s.m - 1;
                break;
                   }
            case 1:
            {
            x = s.n+1;
            y = s.m;
            break;
            }
            case 2:
            {
            x = s.n;
            y = s.m + 1;
            break;
            }
            case 3:
            {
            x = s.n-1;
            y = s.m;
            break;
            }
        }
       if(abs(img[y][x]-img[s.m][s.n])<=T)
```

```
            {
                c[count].m = y;
                c[count].n = x;
                count++;
            }
        }
    }
    *M = count;
}

void ConnectedSet(struct pixel s, double T, unsigned char **img, int height, int
width, int ClassLabel, unsigned int **seg, int *NumConPixels)

{
   struct pixel *B = (struct pixel*)malloc(height*width*sizeof(struct pixel));
//dynamic size of B

    struct pixel current;
    struct pixel c_c[4]; //input to connect_neighbor
    int start = 0; //double pointer idea to controll the size of B
    int end = 0 ;

    *NumConPixels = 0;//make a clearance

    B[start] = s; // serve s as the first seed

    int ind,M_M;//index and the input to connect_neighbor

    seg[s.m][s.n] = ClassLabel;// initialize the first y

    while(start <= end)
    {
        M_M=0;
        ind=0;

        current = B[start];
        start++;

        ConnectedNeighbors(current, T, img, height, width, &M_M, c_c);

        for(ind=0; ind<M_M; ind++)
        {
              if(seg[c_c[ind].m][c_c[ind].n] == 0) //check the neighbors with those
locations at Y
```

```
            {
                end++;
                B[end] = c_c[ind];
                seg[c_c[ind].m][c_c[ind].n] = ClassLabel;
            }
            c_c[ind].m = 0;
            c_c[ind].n = 0;
        }

    }

    if (end<100)
    {
       for (ind=0; ind<=end;ind++)
       {
           seg[B[ind].m][B[ind].n] = 0; //Don't need region less than 100pixels to
be marked on image
       }
    }

    *NumConPixels = end;
    free(B);
}



int main (int argc, char **argv)  //implemented on Linux, windows has unpredicted
bug
{
    FILE *fp;
    struct TIFF_img input_img, output_img;
    struct pixel s;
    double T = 3;
    int width;
    int height;
    int ClassLabel=1;
    int i,j,u1,u2;
    unsigned int **seg;
    unsigned char **img;
    int NumConPixels=0;

    int M = 0;
    struct pixel c[4];
```

```
    if ( ( fp = fopen ( "/home/min/a/zhongc/Desktop/ece637_lab3/img22gd2.tif",
"rb" ) ) == NULL ) {
        fprintf ( stderr, "cannot open file img22gd2.tif\n"  );
        exit ( 1 );
    }


    if ( read_TIFF ( fp, &input_img ) ) {
        fprintf ( stderr, "error open file img22gd2.tif\n" );
        exit ( 1 );
    }


    fclose ( fp );

    img = (unsigned char**) get_img(input_img.height, input_img.width,
sizeof(double));
    seg = (unsigned int**) get_img(input_img.height, input_img.width,
sizeof(double));
    for(i=0; i<input_img.height; i++)
    for(j=0; j<input_img.width; j++)
    {
        img[i][j] = input_img.mono[i][j];//for gray scale image
        seg[i][j] = 0;
    }

    width = input_img.width;
    height = input_img.height;
      get_TIFF ( &output_img, input_img.height, input_img.width, 'g' );

    for(i=0; i<input_img.height; i++)
    for(j=0; j<input_img.width; j++)
    {
        NumConPixels = 0;//make a clearance
        if (seg[i][j] == 0){
        s.m = i;
        s.n = j;
        ConnectedSet(s, T, img, height, width, ClassLabel, seg, &NumConPixels);
                }
        if (NumConPixels > 100){
           for(u1=0; u1<input_img.height; u1++)
```

```
            {
               for(u2=0; u2<input_img.width; u2++)
                {
                    output_img.mono[u1][u2] = seg[u1][u2]*5;
                }
            }
            ClassLabel++;//Number each of these large connected sets sequentially
starting at 1
        }
    }


    //get_TIFF ( &output_img, input_img.height, input_img.width, 'g' );//for gray
scale image

    /*for(i=0; i<input_img.height; i++)
    {
        for(j=0; j<input_img.width; j++)
        {
            output_img.mono[i][j] = seg[i][j]*255;
        }
    } */

    if ( ( fp = fopen ( "segmentation_T=3.tif", "wb" ) ) == NULL ) {
    fprintf ( stderr, "cannot open file color.tif\n");
    exit ( 1 );
        }


    if ( write_TIFF ( fp, &output_img ) ) {
      fprintf ( stderr, "error writing TIFF file\n" );
      exit ( 1 );
    }
    fclose ( fp );

    free_img( (void**)img );
    free_img( (void**)seg );
    free_TIFF ( &(input_img) );
    free_TIFF ( &(output_img) );

    return(0);
}
```