ECE 637 Lab 3 Report

Neighborhoods and Connected Components

Xihui Wang

Section 1 – Area Fill

In this section, I think the description of the problem is not clear enough. At the beginning, it said that m represents the row index, n represents the column index. Then in the middle, the pixels are represented as img[m][n]. Then in the end of this problem, it said that for s = (67, 45), 67 is the column index, and 45 is the row index. Therefore, I think s=(67,45) should be img[45][67]. But in this section, I just provide the two possible results.

1.1 The Image *img22gd2.tif*

Fig 1-1 img22gd2.tif

1.2 The Image Showing the Connected Set of and T = 2

Fig 1-2-1 The image showing the connected set for row=45, col=67

Fig 1-2-2 The image showing the connected set for row=67, col=45

1.3 The Image Showing the Connected Set of T = 1

Fig 1-3-1 The image showing the connected set for row=45, col=67

Fig 1-3-2 The image showing the connected set for row=67, col=45

1.4 The Image Showing the Connected Set of s = (67,45), and T = 3

Fig 1-4-1 The image showing the connected set for row=45, col=67

Fig 1-4-2 The image showing the connected set for row=67, col=45

1.5 A Listing of C Code

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

/*gcc -ansi -Wall -std=c99 lab3_sec1.c tiff.c allocate.c randlib.c*/

struct pixel    {
    int m,n;
};

void ConnectedNeighbors(
    struct pixel s,
    double T,
    unsigned char ** img,
    int width,
    int height,
    int *M,
    struct pixel c[4]);

void ConnectedSet(
    struct pixel s,
    double T,
    unsigned char **img,
    int width,
    int height,
```

```
    int ClassLabel,
    unsigned char **seg,
    int *NumConPixels);

int main(int argc, char const *argv[])
{
    /* code */
    FILE *fp;
    struct TIFF_img input_img, output;
    struct pixel s;
    double T;
    int ClassLabel = 0;
    int NumConPixels = 0;

    s.m = 45;
    s.n = 67;
    T = 3.0;

    /* open image file */
    if ( ( fp = fopen ( "img22gd2.tif", "rb" ) ) == NULL ) {
        fprintf ( stderr, "cannot open file %s\n", argv[1] );
        exit ( 1 );
    }

    /* read image */
    if ( read_TIFF ( fp, &input_img ) ) {
        fprintf ( stderr, "error reading file %s\n", argv[1] );
        exit ( 1 );
    }
```

```
    /* close image file */
    fclose ( fp );

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

    int i, j;
    for(i=0; i<input_img.height; i++)
    {
        for(j=0; j<input_img.width; j++)
        {
            output.mono[i][j] = 255;
        }
    }
    output.mono[s.m][s.n] = ClassLabel;

    ConnectedSet(s, T, input_img.mono, input_img.width,
input_img.height, ClassLabel, output.mono, &NumConPixels);

    /* open output image file */
    if ( ( fp = fopen ( "output.tif", "wb" ) ) == NULL ) {
        fprintf ( stderr, "cannot open file output.tif\n");
        exit ( 1 );
    }

    /* write output image */
    if ( write_TIFF ( fp, &output ) ) {
        fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
        exit ( 1 );
    }
```

```
    /* close output image file */
    fclose ( fp );

    free_TIFF ( &(input_img) );
    free_TIFF ( &(output) );
    return 0;
}
void ConnectedNeighbors(struct pixel s, double T, unsigned char ** img,
int width, int height, int *M, struct pixel c[4])
{
    if(((s.m-1)>=0)&&(abs(img[s.m][s.n]-img[s.m-1][s.n])<=T))
    {
        c[*M].m = s.m-1;
        c[(*M)++].n = s.n;
    }

    if(((s.m+1)<height)&&(abs(img[s.m][s.n]-img[s.m+1][s.n])<=T))
    {
        c[*M].m = s.m+1;
        c[(*M)++].n = s.n;
    }

    if(((s.n-1)>=0)&&(abs(img[s.m][s.n]-img[s.m][s.n-1])<=T))
    {
        c[*M].m = s.m;
        c[(*M)++].n = s.n-1;
    }

    if(((s.n+1)<width)&&(abs(img[s.m][s.n]-img[s.m][s.n+1])<=T))
    {
```

```
        c[*M].m = s.m;
        c[(*M)++].n = s.n+1;
    }
    return;
}

void ConnectedSet(struct pixel s, double T, unsigned char **img, int width,
int height, int ClassLabel, unsigned char **seg, int *NumConPixels)
{
    int M = 0;
    struct pixel c[4];

    ConnectedNeighbors(s, T, img, width, height, &M, c);

    while(M>0)
    {
        if (seg[c[M-1].m][c[M-1].n] != ClassLabel)
        {
            seg[c[M-1].m][c[M-1].n] = ClassLabel;
            (*NumConPixels)++;
            ConnectedSet(c[M-1], T, img, width, height, ClassLabel, seg,
NumConPixels);
        }
        M--;
    }
    return;
}
```

Section 2 – Image Segmentation

2.1 Randomly Colored Segmentation for T = 1, T = 2, and T = 3

Fig 2-1-1-1 segmentation.tif for T = 1

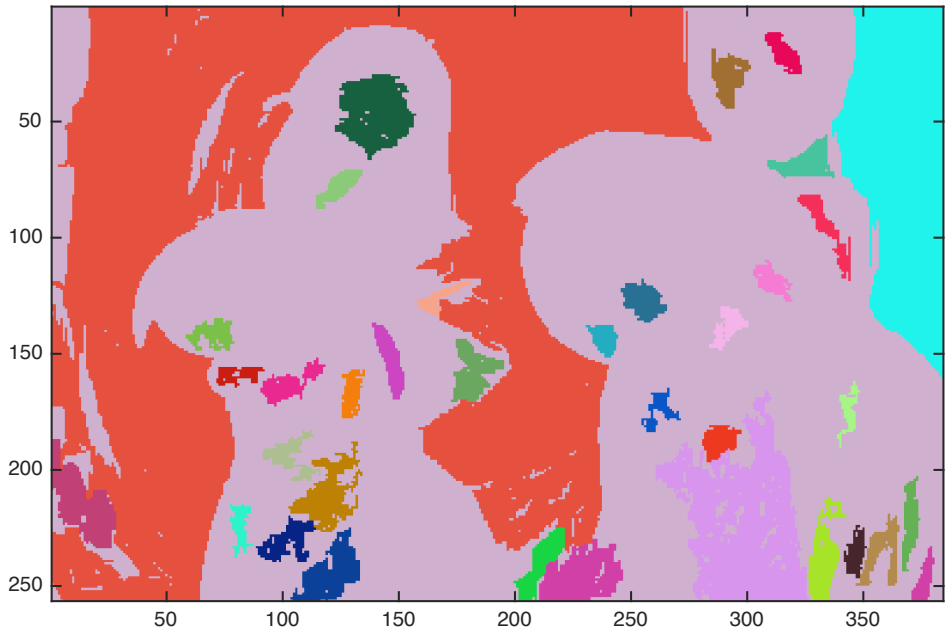

Fig 2-1-1-2 Randomly colored segmentation for T = 1

Fig 2-1-2-1 segmentation.tif for T = 2

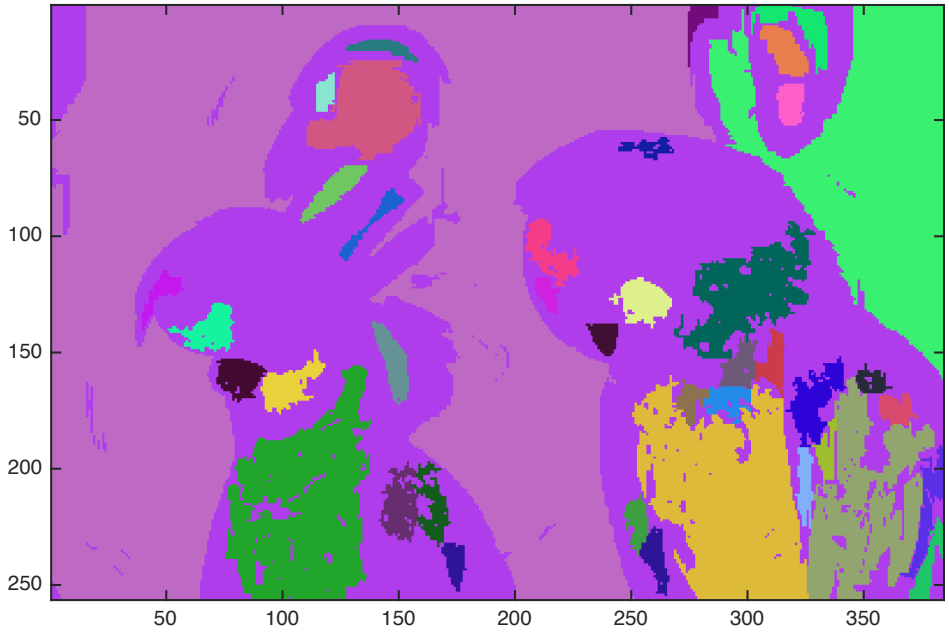

Fig 2-1-2-2 Randomly colored segmentation for T = 2

Fig 2-1-3-1 segmentation.tif for T = 3

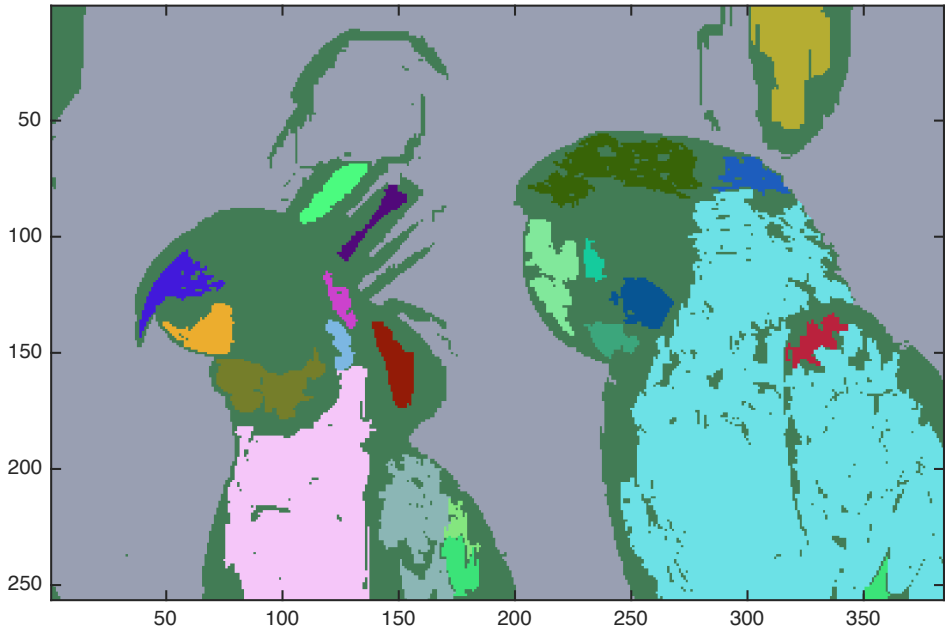

Fig 2-1-3-2 Randomly colored segmentation for T = 3

2.2 A Listing of the Number of Regions Generated for Each of the Values of T = 1, T = 2, and T = 3

Threshold	Number of Regions
1	36
2	41
3	23

2.3 A Listing of C Code

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

struct pixel    {
    int m,n;
};

void ConnectedNeighbors(
    struct pixel s,
    double T,
    unsigned char ** img,
    int width,
    int height,
    int *M,
    struct pixel c[4]);
```

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

int main(int argc, char const *argv[])
{
    /* code */
    FILE *fp;
    struct TIFF_img input_img, output;
    struct pixel s;
    double T;
    int ClassLabel = 1;
    int NumConPixels = 0;

    T = 1.0;

    /* open image file */
    if ( ( fp = fopen ( "img22gd2.tif", "rb" ) ) == NULL ) {
       fprintf ( stderr, "cannot open file %s\n", argv[1] );
       exit ( 1 );
    }

    /* read image */
    if ( read_TIFF ( fp, &input_img ) ) {
```

```
        fprintf ( stderr, "error reading file %s\n", argv[1] );
        exit ( 1 );
    }

    /* close image file */
    fclose ( fp );

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

    int i, j;
    for(i=0; i<input_img.height; i++)
    {
        for(j=0; j<input_img.width; j++)
        {
            output.mono[i][j] = 0;
        }
    }

    for(i=0; i<input_img.height; i++)
    {
        for(j=0; j<input_img.width; j++)
        {
            if (output.mono[i][j] == 0)
            {
                s.m = i;
                s.n = j;
                output.mono[i][j] = ClassLabel;
                ConnectedSet(s, T, input_img.mono, input_img.width,
input_img.height, ClassLabel, output.mono, &NumConPixels);
                if(NumConPixels > 100)
```

```
                {
                    ClassLabel++;
                    NumConPixels = 0;
                }
                else
                {
                    output.mono[i][j] = 255;
                    NumConPixels = 0;
                    ConnectedSet(s, T, input_img.mono, input_img.width,
input_img.height, 255, output.mono, &NumConPixels);
                }
            }
        }
    }

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

    /* open output image file */
    if ( ( fp = fopen ( "segmentation.tif", "wb" ) ) == NULL ) {
        fprintf ( stderr, "cannot open file output.tif\n");
        exit ( 1 );
```

```
    }

    /* write output image */
    if ( write_TIFF ( fp, &output ) ) {
        fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
        exit ( 1 );
    }

    /* close output image file */
    fclose ( fp );

    free_TIFF ( &(input_img) );
    free_TIFF ( &(output) );
    return 0;
}

void ConnectedNeighbors(struct pixel s, double T, unsigned char ** img,
int width, int height, int *M, struct pixel c[4])
{

    if(((s.m-1)>=0)&&(abs(img[s.m][s.n]-img[s.m-1][s.n])<=T))
    {
        c[*M].m = s.m-1;
        c[(*M)++].n = s.n;
    }

    if(((s.m+1)<height)&&(abs(img[s.m][s.n]-img[s.m+1][s.n])<=T))
    {
        c[*M].m = s.m+1;
        c[(*M)++].n = s.n;
```

```
    }

    if(((s.n-1)>=0)&&(abs(img[s.m][s.n]-img[s.m][s.n-1])<=T))
    {
        c[*M].m = s.m;
        c[(*M)++].n = s.n-1;
    }

    if(((s.n+1)<width)&&(abs(img[s.m][s.n]-img[s.m][s.n+1])<=T))
    {
        c[*M].m = s.m;
        c[(*M)++].n = s.n+1;
    }

    return;
}

void ConnectedSet(struct pixel s, double T, unsigned char **img, int width,
int height, int ClassLabel, unsigned char **seg, int *NumConPixels)
{
    int M = 0;
    struct pixel c[4];

    ConnectedNeighbors(s, T, img, width, height, &M, c);

    while(M>0)
    {
        if (seg[c[M-1].m][c[M-1].n] == 0)
        {
            seg[c[M-1].m][c[M-1].n] = ClassLabel;
```

```
            (*NumConPixels)++;
            ConnectedSet(c[M-1], T, img, width, height, ClassLabel, seg,
NumConPixels);
        }
        M--;
    }
    return;
}
```