ECE63700-Lab3

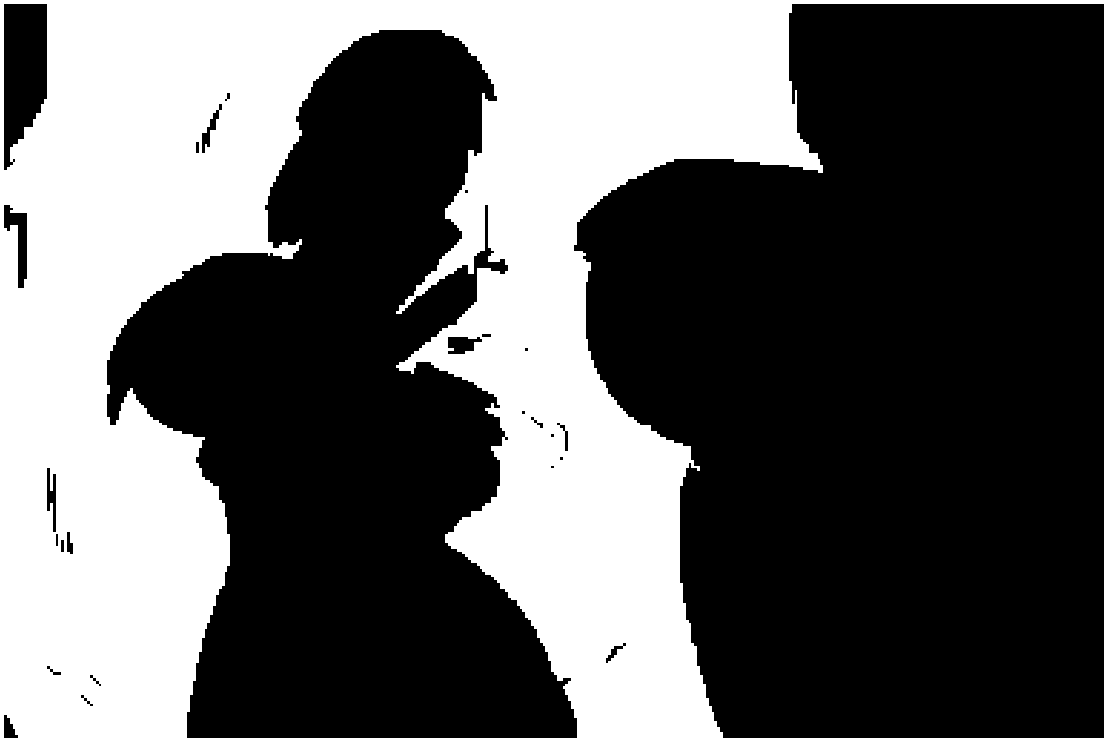
Sriranga R

January 2023

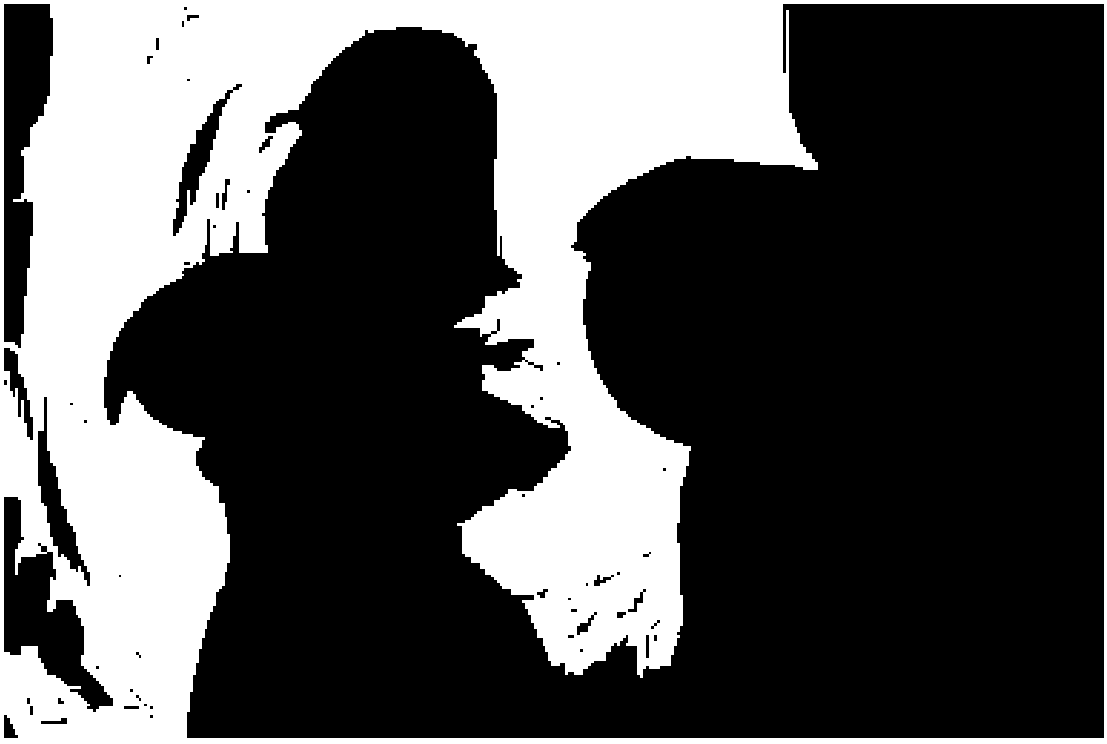
1 Area Fill

1.1 A print out the image img22gd2.tif

1.2 A print out of the image showing the connected set for s = (67, 45), and T = 2



1.3 A print out of the image showing the connected set for s = (67; 45), and T = 1



1.4 A print out of the image showing the connected set for s = (67; 45), and T = 3

1.5 C code

1.5.1 Implementation of ConnectedNeighbours() and ConnectedSet()

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "defs.h"

void ConnectedNeighbours(
    Pixel s, double T, unsigned char** img,
    int width, int height, int *M, Pixel c[4] )
{
    int neighbours = 0;
    //Calculating the neighbors of s.
    int x = s.m; // row index of image
    int y = s.n; // col index of image
    if((x - 1 >= 0) && abs(img[x][y] - img[x-1][y]) <= T)
    {
        c[neighbours].m = x - 1;
        c[neighbours].n = y;
        neighbours++;
    }
```

```
    if((x + 1 < height) && abs(img[x][y] - img[x+1][y]) <= T)
    {
        c[neighbours].m = x + 1;
        c[neighbours].n = y;
        neighbours++;
    }
    if((y - 1 >= 0) && abs(img[x][y] - img[x][y-1]) <= T)
    {
        c[neighbours].m = x;
        c[neighbours].n = y - 1;
        neighbours++;
    }
    if((y + 1 < width) && abs(img[x][y] - img[x][y+1]) <= T)
    {
        c[neighbours].m = x;
        c[neighbours].n = y + 1;
        neighbours++;
    }
    *M = neighbours;
    /*printf("Connected components of %d, %d are with neighbors = %
    d\n", s.m, s.n, neighbours);
    for(int i = 0; i < neighbours; i++)
    {
        printf("m = %d, n = %d\n", c[i].m, c[i].n);
    }*/
    return;
}

void ConnectedSet(
    Pixel s, double T, unsigned char** img, int width,
    int height, int ClassLabel,unsigned int **seg, int *
    NumConPixels )
{
    //printf("Entered connected set\n");
    int count = 0;
    int connectedPixels = 0;
    Pixel c[4];
    ConnectedNeighbours(s, T, img, width, height, &connectedPixels,
     c);
    if(connectedPixels == 0)
    {
        //printf("Returning now\n");
        return;
    }

    for(int i = 0; i < connectedPixels; i++)
    {
        if(seg[c[i].m][c[i].n] != ClassLabel)
        {
            //printf("Pixel Coordinate - %d, %d set to 1.\n", c[i].
    m, c[i].n);
            seg[c[i].m][c[i].n] = ClassLabel;
            count++;
            ConnectedSet(c[i], T, img, width, height, ClassLabel,
    seg, NumConPixels);
        }
    }
```

```
    //printf("Recursion ended\n");
    *NumConPixels += count;
    return;
}
```

```
#ifndef DEFS_H
#define DEFS_H

struct pixel
{
    int m; // row index
    int n; // col index
};

typedef struct pixel Pixel;

//Fucntions
void ConnectedNeighbours(
    Pixel s,
    double T,
    unsigned char** img,
    int width,
    int height,
    int *M,
    Pixel c[4]
);

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

#endif
```

1.5.2 Implementation of the main function

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

//clear the allocated variables as well.

void help()
{
    printf(" arg1 : Input Image, \n \
```

```
            arg2 : m-co-ordinate of img, \n \
            arg3 : n-co-ordiate of img, \n \
            arg4 : threshold T, \n \
            arg5 : ClassLabel L, \n");
}

int main(int argc, char* argv[])
{
  FILE *fp;
  struct TIFF_img input_img, output_tiff;

  if ( argc != 6 )
  {
    help();
    //error( argv[0] );
    exit(0);
  }
  //Testing the connected neighbour subroutine
    Pixel s;
    s.m = atoi(argv[2]);
    s.n = atoi(argv[3]);
    double T = atoi(argv[4]);
    int ClassLabel = atoi(argv[5]);

  /* open image file */
  if ( ( fp = fopen ( argv[1], "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 );

  /* check the type of image data */
  if ( input_img.TIFF_type != 'g' ) {
    fprintf ( stderr, "error:  image must be gray scale\n" );
    exit ( 1 );
  }


   unsigned int** seg = (unsigned int**)get_img(input_img.width,
    input_img.height, sizeof(unsigned int));
   for(int i = 0; i < input_img.height; i++)
   {
    for(int j = 0; j < input_img.width; j++)
    {
        seg[i][j] = 0;
    }
   }
   get_TIFF ( &output_tiff, input_img.height, input_img.width, 'g'
    );
```

```
71    for(int i = 0; i < input_img.height; i++)
72    {
73     for(int j = 0; j < input_img.width; j++)
74     {
75         output_tiff.mono[i][j] = 0;
76     }
77    }
78
79     int connectedPixels = 0;
80     ConnectedSet(s, T, input_img.mono, input_img.width, input_img.
       height, ClassLabel, seg, &connectedPixels);
81     printf("Connected Pixesls = %d\n", connectedPixels);
82     for(int i = 0; i < input_img.height; i++)
83     {
84     for(int j = 0; j < input_img.width; j++)
85     {
86         if(seg[i][j] == 1)
87         {
88             output_tiff.mono[i][j] = 255;
89         }
90     }
91     }
92
93     if ( ( fp = fopen ( "output.tif", "wb" ) ) == NULL ) {
94     fprintf ( stderr, "cannot open file output.tif\n");
95     exit ( 1 );
96     }
97
98     /* write green image */
99     if ( write_TIFF ( fp, &output_tiff ) ) {
100    fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
101    exit ( 1 );
102    }
103
104    fclose ( fp );
105    #if 1
106    ClassLabel = 1;
107    get_TIFF ( &output_tiff, input_img.height, input_img.width, 'g'
       );
108    for(int i = 0; i < input_img.height; i++)
109    {
110        for(int j = 0; j < input_img.width; j++)
111        {
112            output_tiff.mono[i][j] = 0;
113        }
114    }
115    for(int i = 0; i < input_img.height; i++)
116    {
117        for(int j = 0; j < input_img.width; j++)
118        {
119            seg[i][j] = 0;
120        }
121    }
122
123    for(int i = 0; i < input_img.height; i++)
124    //for(int i = 145; i < 160; i++)
125    {
```

```
126         printf("i = %d\n", i);
127         for(int j = 0; j < input_img.width; j++)
128         //for(int j = 145; j < 160; j++)
129         {
130             connectedPixels = 0;
131             if(seg[i][j] == 0)
132             {
133                 s.m = i;
134                 s.n = j;
135                 ConnectedSet(s, T, input_img.mono, input_img.width,
     input_img.height, ClassLabel, seg, &connectedPixels);
136                 if(connectedPixels > 100)
137                 {
138                     printf("Pixel : m = %d, n = %d, connected
    pixels = %d\n", s.m, s.n, connectedPixels);
139                     ClassLabel++;
140                 }
141                 else
142                 {
143                     printf("Pixel : m = %d, n = %d, connected
    pixels = %d\n", s.m, s.n, connectedPixels);
144                     ConnectedSet(s, T, input_img.mono, input_img.
    width, input_img.height, 0, seg, &connectedPixels);
145                 }
146             }
147         }
148     }
149
150     for(int i = 0; i < input_img.height; i++)
151     {
152         for(int j = 0; j < input_img.width; j++)
153         {
154             if(seg[i][j] != 0)
155             {
156                 output_tiff.mono[i][j] = seg[i][j];
157             }
158         }
159     }
160
161     if ( ( fp = fopen ( "output_segment.tif", "wb" ) ) == NULL ) {
162     fprintf ( stderr, "cannot open file output.tif\n");
163     exit ( 1 );
164     }
165
166     /* write green image */
167     if ( write_TIFF ( fp, &output_tiff ) ) {
168     fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
169     exit ( 1 );
170     }
171
172     fclose ( fp );
173     #endif
174     free_TIFF(&input_img);
175     free_TIFF(&output_tiff);
176     free_img(seg);
177
178   return 0;
```

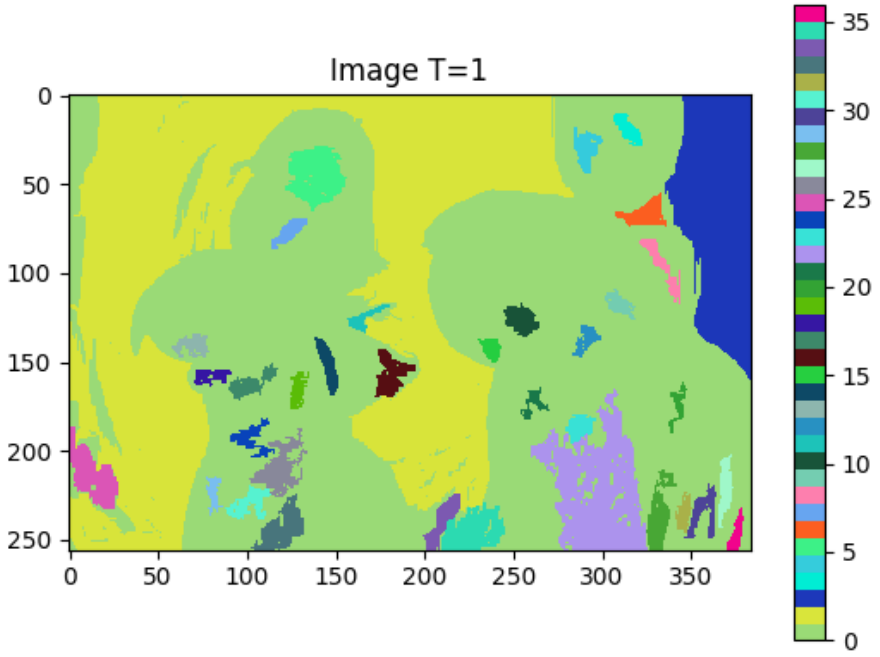
```
179 }
```

The above code contains the code for both segmentation and as well as finding the connected set for a given point in the image.

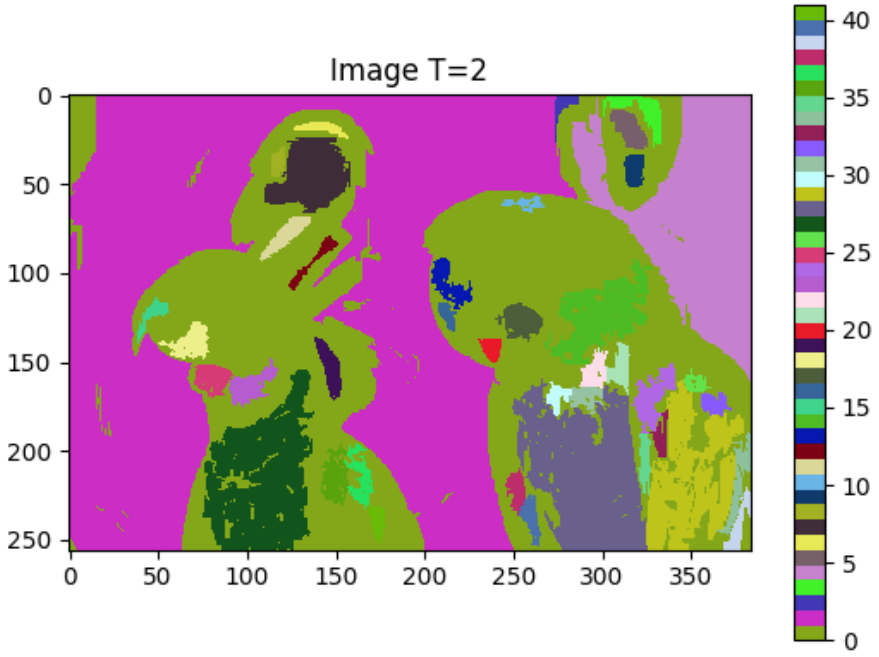
2 Image Segmentation

2.1 Print outs of the randomly colored segmentation for T = 1, T = 2, and T = 3

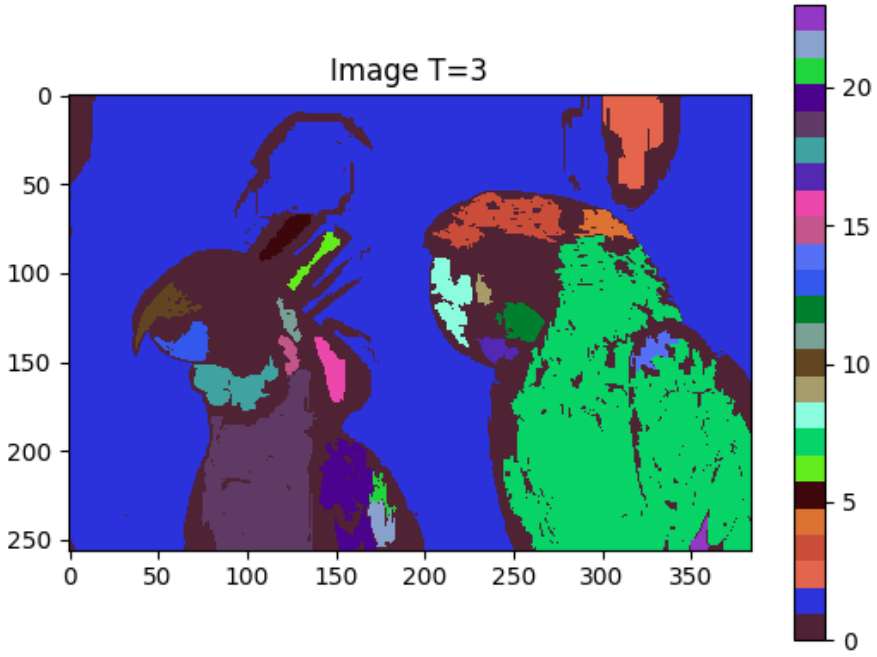
2.1.1 Segmentation image for T=1



2.1.2 Segmentation image for T=2



2.1.3 Segmentation image 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.

T	Number of segments
1	36
2	41
3	23

2.3 Python code

The C code is mentioned in section in 1.5.1 and 1.5.2. The Python function is as below.

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib as mpl

im = Image.open("output_segment_T_1.tif")
```

```
7
8  x=np.array(im)
9
10 N = np.max(x)
11
12 cmap = mpl.colors.ListedColormap(np.random.rand(N+1,3))
13 plt.imshow(x, cmap=cmap, interpolation='none')
14 plt.colorbar()
15 plt.title('Image T=1')
16 plt.show()
17
18 im = Image.open("output_segment_T_2.tif")
19
20 x=np.array(im)
21
22 N = np.max(x)
23
24 cmap = mpl.colors.ListedColormap(np.random.rand(N+1,3))
25 plt.imshow(x, cmap=cmap, interpolation='none')
26 plt.colorbar()
27 plt.title('Image T=2')
28 plt.show()
29
30 im = Image.open("output_segment_T_3.tif")
31
32 x=np.array(im)
33
34 N = np.max(x)
35
36 cmap = mpl.colors.ListedColormap(np.random.rand(N+1,3))
37 plt.imshow(x, cmap=cmap, interpolation='none')
38 plt.colorbar()
39 plt.title('Image T=3')
40 plt.show()
```