HOMEWORK 3

DIGITAL IMAGE PROCESSING LABORATORY

ALEXANDRE OLIVE PELLICER

Section 1

1. A print out the image img22gd2.tif.

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

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

The T=1 image appears blank for s= (67, 45) when considering 67 as the column and 45 as the row, as specified in the lab manual. However, if 67 is interpreted as the row and 45 as the column, the output will resemble the previous. This is the output obtained when flipping the indexes:

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

5. A listing of your C code.

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

struct pixel {
    int m;
    int 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 int **seg,
    int *NumConPixels);

int main (int argc, char **argv) {
    FILE *fp;
    struct TIFF_img input_img;
    struct pixel s;
    int i, j;
    int ClassLabel = 1;
    int NumConPixels = 0;

    /*pixel and threshold selection*/
    s.m = 45;
    s.n = 67;
    double T = 1;
```

```
53.
54.     /* open image file */
55.     if ( ( fp = fopen ( "img22gd2.tif", "rb" ) ) == NULL ) {
56.         fprintf ( stderr, "cannot open file %s\n", argv[1] );
57.         exit ( 1 );
58.     }
59.
60.     /* read image */
61.     if ( read_TIFF ( fp, &input_img ) ) {
62.         fprintf ( stderr, "error reading file %s\n", argv[1] );
63.         exit ( 1 );
64.     }
65.
66.     /* close image file */
67.     fclose ( fp );
68.
69.     /* check the type of image data */
70.     if ( input_img.TIFF_type != 'g' ) {
71.         fprintf ( stderr, "error:  image must be 24-bit color\n" );
72.         exit ( 1 );
73.     }
74.
75.     /* allocate segmentation */
76.     unsigned int **seg = (unsigned int **)get_img(input_img.width,
   input_img.height, sizeof(unsigned int*));
77.
78.     /* initialize segmentation */
79.     for (i=0; i<input_img.height; i++){
80.         for (j=0; j<input_img.width; j++) {
81.             seg[i][j] = 0;
82.         }
83.     }
84.
85.     ConnectedSet(s, T, input_img.mono, input_img.width,
   input_img.height, ClassLabel, seg, &NumConPixels);
86.
87.     for (i = 0; i < input_img.height; i++) {
88.         for (j = 0; j < input_img.width; j++) {
89.             if (seg[i][j] == ClassLabel) {
90.                 input_img.mono[i][j] = 0;
91.             }
92.             else {
93.                 input_img.mono[i][j] = 255;
94.             }
95.         }
96.     }
97.
98.     /* open output image file */
99.     if ( ( fp = fopen ( "out_T1.tif", "wb" ) ) == NULL ) {
```

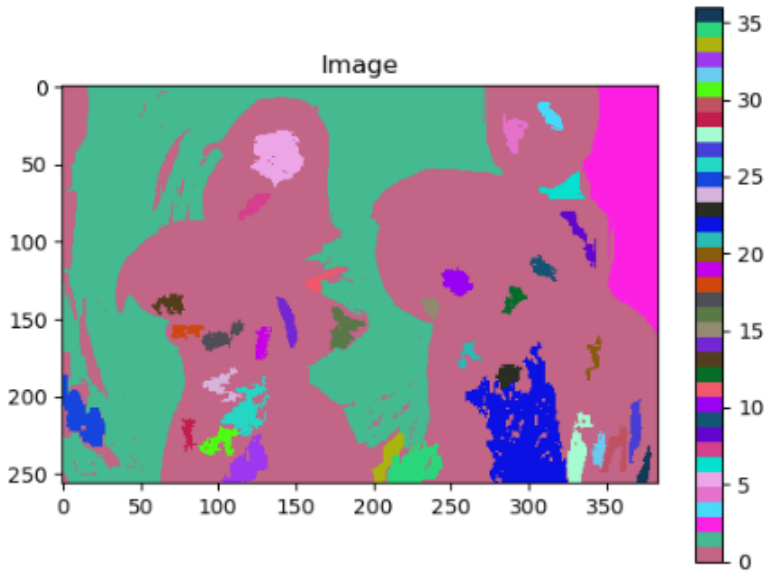
```
100.          fprintf ( stderr, "cannot open file out_T1.tif\n");
101.          exit ( 1 );
102.          }
103.
104.          /* write output image */
105.          if ( write_TIFF ( fp, &input_img ) ) {
106.          fprintf ( stderr, "error writing TIFF file out_T1.tif
   \n");
107.          exit ( 1 );
108.          }
109.
110.          /* close output image file */
111.          fclose ( fp );
112.
113.          /* de-allocate memory */
114.          free_img((void *)seg);
115.          free_TIFF ( &(input_img) );
116.
117.          return(0);
118.      }
119.
120.      void ConnectedNeighbors(
121.          struct pixel s,
122.          double T,
123.          unsigned char **img,
124.          int width,
125.          int height,
126.          int *M,
127.          struct pixel c[4])
128.      {
129.          *M = 0;
130.
131.          if ((s.n-1) >= 0 && abs(img[s.m][s.n] - img[s.m][s.n-1])
   <= T) {
132.              c[*M].m = s.m;
133.              c[*M].n = s.n - 1;
134.              (*M)++;
135.          }
136.          if ((s.n+1) < width && abs(img[s.m][s.n] -
   img[s.m][s.n+1]) <= T) {
137.              c[*M].m = s.m;
138.              c[*M].n = s.n + 1;
139.              (*M)++;
140.          }
141.          if ((s.m-1) >= 0 && abs(img[s.m][s.n] - img[s.m-1][s.n])
   <= T) {
142.              c[*M].m = s.m - 1;
143.              c[*M].n = s.n;
144.              (*M)++;
```

```
145.         }
146.         if ((s.m+1) < height && abs(img[s.m][s.n] -
   img[s.m+1][s.n]) <= T) {
147.             c[*M].m = s.m + 1;
148.             c[*M].n = s.n;
149.             (*M)++;
150.         }
151.     }
152.
153.     void ConnectedSet(
154.         struct pixel s,
155.         double T,
156.         unsigned char **img,
157.         int width,
158.         int height,
159.         int ClassLabel,
160.         unsigned int **seg,
161.         int *NumConPixels)
162.     {
163.         *NumConPixels = 0;
164.
165.         struct pixel *B = malloc(width * height * sizeof(struct
   pixel));
166.         int pos = 0, i, M;
167.         struct pixel c[4];
168.
169.         /* unlabeled  and connected pixels and the given pixel
   will be appended to B at position pos*/
170.         B[pos] = s;
171.         pos = pos + 1;
172.
173.         while (pos > 0) {
174.             /* every time a pixel is labeled, position pos will
   be reduced by 1 and NumConPixels incremented by 1*/
175.             pos = pos -1;
176.             if (seg[B[pos].m][B[pos].n] == 0) {
177.                 seg[B[pos].m][B[pos].n] = ClassLabel;
178.                 (*NumConPixels)++;
179.
180.                 /* get connected neighbors */
181.                 ConnectedNeighbors(B[pos], T, img, width, height,
   &M, c);
182.
183.                 /* append the connected and without label pixels
   to B */
184.                 for (i = 0; i < M; i++) {
185.                     if (seg[c[i].m][c[i].n] == 0) {
186.                         B[pos] = c[i];
187.                         pos = pos + 1;
```

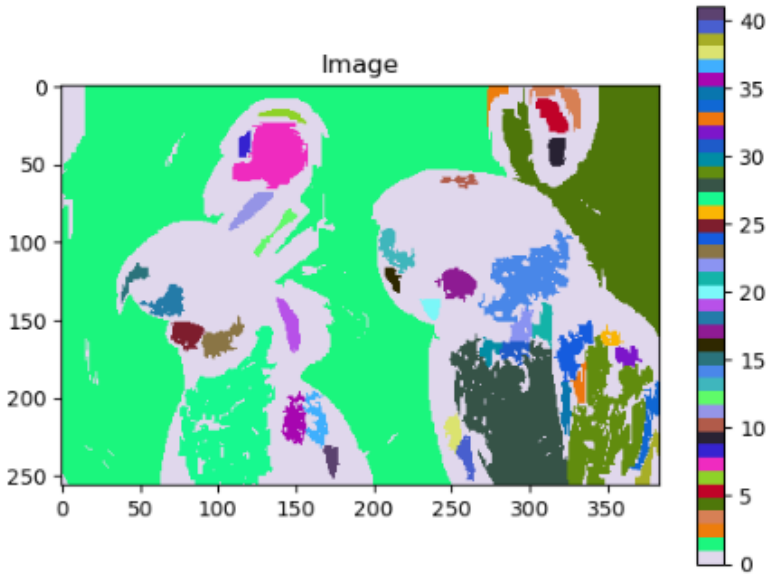
```
188.                     }
189.                 }
190.             }
191.         }
192.
193.         free(B);
194.     }
195.
```

Section 2

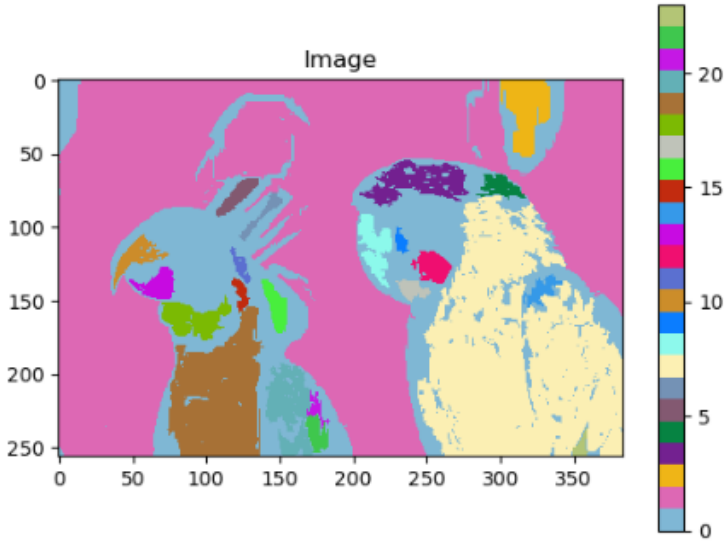
1. Print outs of the randomly colored segmentation for T = 1, T = 2, and T = 3.
   T=1:



   T=2:



T=3:



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

Value of T	Num of regions
1	36
2	41
3	23

3. A listing of your C code.

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

struct pixel {
    int m;
    int n;
};

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

```
28.
29.void ConnectedSet(
30.     struct pixel s,
31.     double T,
32.     unsigned char **img,
33.     int width,
34.     int height,
35.     int ClassLabel,
36.     unsigned int **seg,
37.     int *NumConPixels);
38.
39.int main (int argc, char **argv)
40.{
41.
42.     FILE *fp;
43.     struct TIFF_img input_img;
44.     struct pixel s;
45.     int i, j, a, b;
46.     int ClassLabel = 1;
47.     int NumConPixels = 0;
48.
49.     double T = 1;
50.
51.     /* open image file */
52.     if ( ( fp = fopen ( "img22gd2.tif", "rb" ) ) == NULL ) {
53.         fprintf ( stderr, "cannot open file %s\n", argv[1] );
54.         exit ( 1 );
55.     }
56.
57.     /* read image */
58.     if ( read_TIFF ( fp, &input_img ) ) {
59.         fprintf ( stderr, "error reading file %s\n", argv[1] );
60.         exit ( 1 );
61.     }
62.
63.     /* close image file */
64.     fclose ( fp );
65.
66.     /* check the type of image data */
67.     if ( input_img.TIFF_type != 'g' ) {
68.         fprintf ( stderr, "error:  image must be 24-bit color\n" );
69.         exit ( 1 );
70.     }
71.
72.     /* allocate segmentation */
73.     unsigned int **seg = (unsigned int **)get_img(input_img.width,
   input_img.height, sizeof(unsigned int*));
74.
75.     /* initialize segmentation */
```

```
76.     for (i=0; i<input_img.height; i++){
77.         for (j=0; j<input_img.width; j++) {
78.             seg[i][j] = 0;
79.         }
80.     }
81.
82.     for (i=0; i<input_img.height; i++){
83.         for (j=0; j<input_img.width; j++) {
84.             if (seg[i][j] == 0) {
85.                 s.m = i;
86.                 s.n = j;
87.                 ConnectedSet(s, T, input_img.mono, input_img.width,
    input_img.height, ClassLabel, seg, &NumConPixels);
88.                 /* If the number of labeled pixels is higher than
    100, we move to the next class. If not, we replace tha label added
    to the pixels and put the value back to 0 */
89.                 if (NumConPixels > 100) {
90.                     ClassLabel ++;
91.                 }
92.                 else {
93.                     for (a = 0; a < input_img.height; a++){
94.                         for (b = 0; b < input_img.width; b++){
95.                             if (seg[a][b] == ClassLabel){
96.                                 seg[a][b] = 0;
97.                             }
98.                         }
99.                     }
100.                      }
101.                 }
102.             }
103.         }
104.
105.         for (i = 0; i < input_img.height; i++){
106.             for (j = 0; j < input_img.width; j++) {
107.                 input_img.mono[i][j] = seg[i][j];
108.             }
109.         }
110.
111.         /* open output image file */
112.         if ( ( fp = fopen ( "segT1.tif", "wb" ) ) == NULL ) {
113.         fprintf ( stderr, "cannot open file segT1.tif \n");
114.         exit ( 1 );
115.         }
116.
117.         /* write output image */
118.         if ( write_TIFF ( fp, &input_img ) ) {
119.         fprintf ( stderr, "error writing TIFF file segT1.tif
    \n");
120.         exit ( 1 );
```

```
121.          }
122.
123.          /* close output image file */
124.          fclose ( fp );
125.
126.          /* de-allocate memory */
127.          free_img((void *)seg);
128.          free_TIFF ( &(input_img) );
129.
130.          printf("Num of segs: %d\n", ClassLabel - 1);
131.
132.          return(0);
133.      }
134.
135.      void ConnectedNeighbors(
136.          struct pixel s,
137.          double T,
138.          unsigned char **img,
139.          int width,
140.          int height,
141.          int *M,
142.          struct pixel c[4])
143.      {
144.          *M = 0;
145.
146.          if ((s.n-1) >= 0 && abs(img[s.m][s.n] - img[s.m][s.n-1])
    <= T) {
147.              c[*M].m = s.m;
148.              c[*M].n = s.n - 1;
149.              (*M)++;
150.          }
151.          if ((s.n+1) < width && abs(img[s.m][s.n] -
    img[s.m][s.n+1]) <= T) {
152.              c[*M].m = s.m;
153.              c[*M].n = s.n + 1;
154.              (*M)++;
155.          }
156.          if ((s.m-1) >= 0 && abs(img[s.m][s.n] - img[s.m-1][s.n])
    <= T) {
157.              c[*M].m = s.m - 1;
158.              c[*M].n = s.n;
159.              (*M)++;
160.          }
161.          if ((s.m+1) < height && abs(img[s.m][s.n] -
    img[s.m+1][s.n]) <= T) {
162.              c[*M].m = s.m + 1;
163.              c[*M].n = s.n;
164.              (*M)++;
165.          }
```

```
166.     }
167.
168.     void ConnectedSet(
169.         struct pixel s,
170.         double T,
171.         unsigned char **img,
172.         int width,
173.         int height,
174.         int ClassLabel,
175.         unsigned int **seg,
176.         int *NumConPixels)
177.     {
178.         *NumConPixels = 0;
179.
180.         struct pixel *B = malloc(width * height * sizeof(struct
    pixel));
181.         int pos = 0, i, M;
182.         struct pixel c[4];
183.
184.         /* unlabeled  and connected pixels and the given pixel
    will be appended to B at position pos*/
185.         B[pos] = s;
186.         pos = pos + 1;
187.
188.         while (pos > 0) {
189.             /* every time a pixel is labeled, position pos will
    be reduced by 1 and NumConPixels incremented by 1*/
190.             pos = pos -1;
191.             if (seg[B[pos].m][B[pos].n] == 0) {
192.                 seg[B[pos].m][B[pos].n] = ClassLabel;
193.                 (*NumConPixels)++;
194.
195.                 /* get connected neighbors */
196.                 ConnectedNeighbors(B[pos], T, img, width, height,
    &M, c);
197.
198.                 /* append the connected and without label pixels
    to B */
199.                 for (i = 0; i < M; i++) {
200.                     if (seg[c[i].m][c[i].n] == 0) {
201.                         B[pos] = c[i];
202.                         pos = pos + 1;
203.                     }
204.                 }
205.             }
206.         }
207.
208.         free(B);
209.     }
```