

ECE 637: Lab 3

Rahul Deshmukh

February 10, 2021

Section 1 Report

1. Print out of img22gd2.tif



Figure 1: Input gray scale image

2. Print out of image showing connected set for $s = (67, 45)$ and $T = 2$



3. Print out of image showing connected set for $s = (67, 45)$ and $T = 1$

!

4. Print out of image showing connected set for $s = (67, 45)$ and $T = 3$



5. For C-code of function *ConnectedSet_LL()* and *ConnectedNeighbors()* refer to Listing 2 at page 8.

Section 2 Report

1. Print out of randomly colored segmentation to $T = 1$, $T = 2$ and $T = 3$.

Segmentation Image: Threshold=1 NumRegions=36

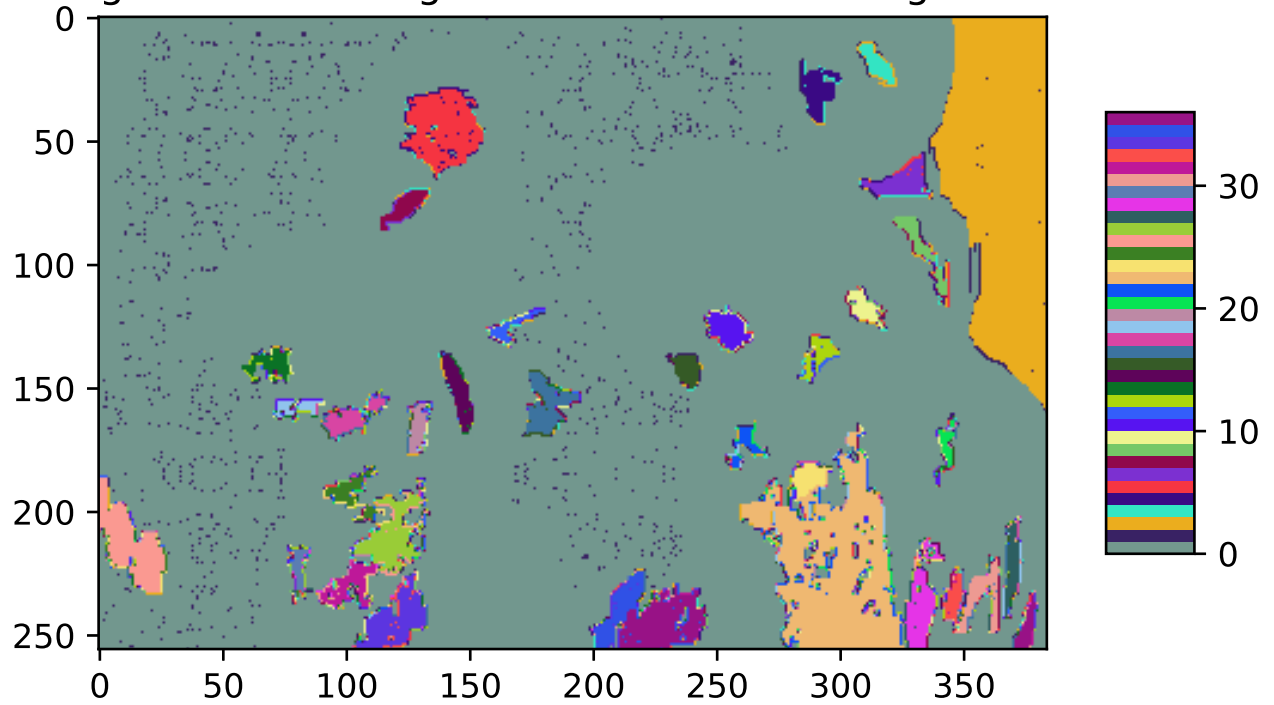


Figure 2: segmentation result for $T=1$

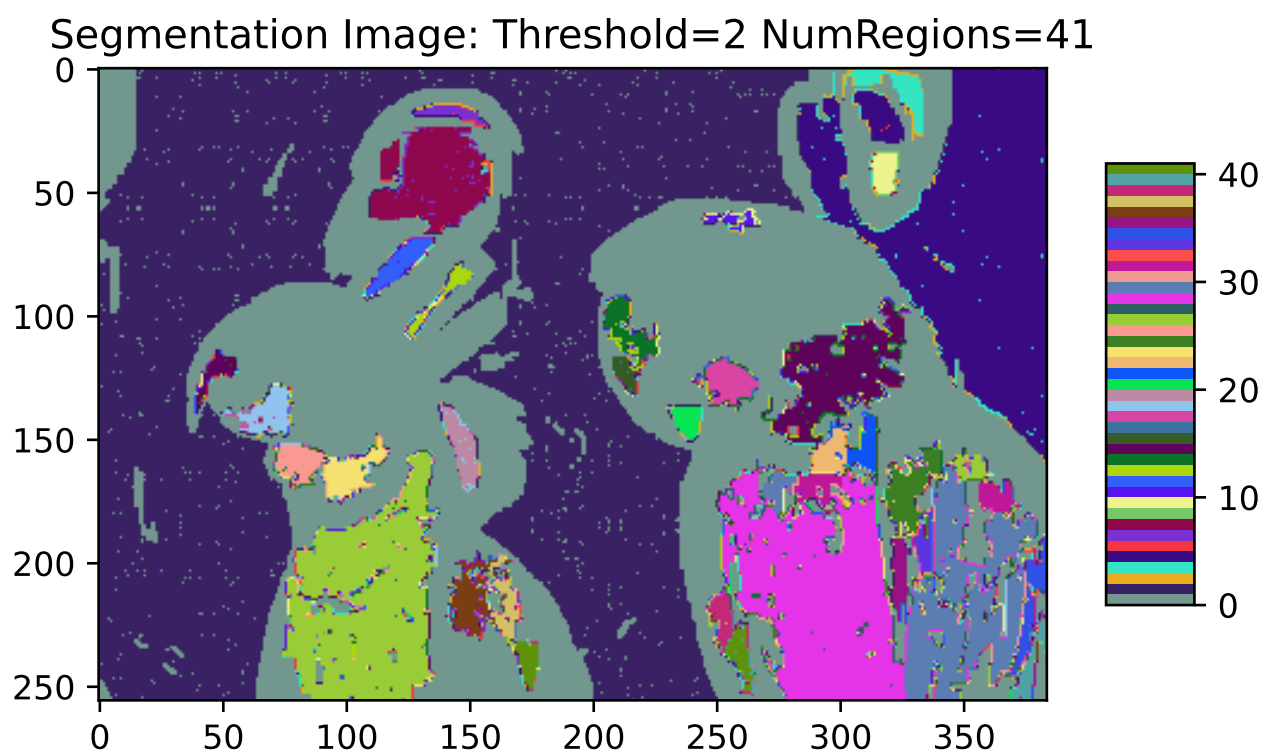


Figure 3: segmentation result for $T=2$

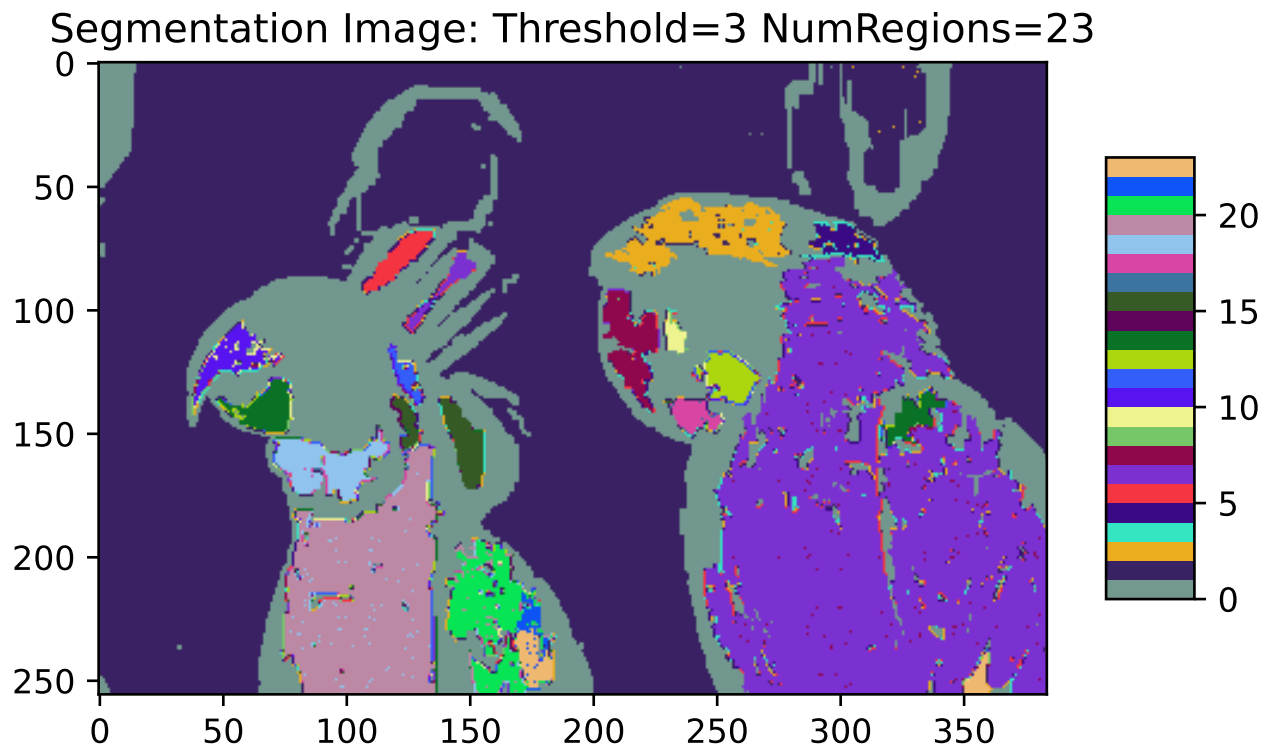


Figure 4: segmentation result for $T=3$

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

Listing 1: Text output

```

1 T = 2
2 Number of pixels in connected set: 36317
3 Number of regions in connected components: 41
4 T = 1
5 Number of pixels in connected set: 3
6 Number of regions in connected components: 36
7 T = 3
8 Number of pixels in connected set: 48926
9 Number of regions in connected components: 23
10 done

```

3. For C-code of function *ConnectedComponents()* refer to Listing 2 at page 8.

Source Code

Listing 2: C-Code

```
1
2 #include <math.h>
3 #include <stdlib.h>
4 #include <list>
5 #include "tiff.h"
6 #include "allocate.h"
7 #include "randlib.h"
8 #include "typeutil.h"
9 using namespace std;
10
11 #define MinPixCount 100
12
13 struct pixel{int m, n; /*m= row, n= col*/};
14
15 void ConnectedNeighbors(
16 struct pixel s,
17 double threshold,
18 unsigned char **img,
19 int width,
20 int height,
21 int *M,
22 struct pixel c[4]);
23
24 void ConnectedSet(
25 struct pixel s,
26 double threshold,
27 unsigned char **img,
28 int width,
29 int height,
30 int ClassLabel,
31 unsigned char **seg,
32 int *NumConPixels);
33
34 void ConnectedSet_LL(
35 struct pixel s,
36 double threshold,
37 unsigned char **img,
38 int width,
39 int height,
40 int ClassLabel,
41 unsigned char **seg,
42 int *NumConPixels);
43
44 void ConnectedComponents(
45 double threshold,
46 unsigned char **img,
47 int width,
48 int height,
49 unsigned char **seg_out,
50 unsigned char **connected_set,
51 int *NumRegions
52 );
53
54 void error(char *name);
55
```



```

56 int main (int argc, char **argv)
57 {
58     FILE *fp;
59     struct TIFF_img input_img, seg_img, cs_img, temp_img;
60     int32_t i,j,NumConPixels=0, ClassLabel=1, NumRegions=0;
61     double threshold;
62     struct pixel s0;
63
64     if ( argc != 5) error( argv[0] );
65     s0.m = atof(argv[2]); //row
66     s0.n = atof(argv[3]); //col
67     threshold = atof(argv[4]);
68
69     /* open image file */
70     if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
71         fprintf ( stderr, "cannot open file %s\n", argv[1] );
72         exit ( 1 );
73     }
74
75     /* read image */
76     if ( read_TIFF ( fp, &input_img ) ) {
77         fprintf ( stderr, "error reading file %s\n", argv[1] );
78         exit ( 1 );
79     }
80
81     /* close image file */
82     fclose ( fp );
83
84     /* check the type of image data */
85     if ( input_img.TIFF_type != 'g' ) {
86         fprintf ( stderr, "error: image must be 8-bit monochrome\n" );
87         exit ( 1 );
88     }
89
90     /* set up structure for output achromatic image */
91     /* to allocate a full color image use type 'c' */
92     get_TIFF ( &cs_img, input_img.height, input_img.width, 'g' );
93     get_TIFF ( &seg_img, input_img.height, input_img.width, 'g' );
94     get_TIFF ( &temp_img, input_img.height, input_img.width, 'g' );
95
96     /* initialize segmentation map as 0 */
97     for ( i = 0; i < input_img.height; i++ ){
98         for ( j = 0; j < input_img.width; j++ ) {
99             cs_img.mono[i][j] = 0;
100             temp_img.mono[i][j] = 0;
101             seg_img.mono[i][j]=0;
102         }}
103
104     /* find connected set*/
105     // ConnectedSet(s0,threshold,input_img.mono,input_img.width,input_img.height,\
106     //             ClassLabel,cs_img.mono,&NumConPixels);
107
108
109     ConnectedSet_LL(s0,threshold,input_img.mono,input_img.width,input_img.height,\
110                   ClassLabel,cs_img.mono,&NumConPixels);
111     fprintf(stdout, "Number of pixels in connected set: %d\n", NumConPixels);
112
113     /* change 1-black 0-white */
114     for ( i = 0; i < input_img.height; i++ ){

```

```

115 for ( j = 0; j < input_img.width; j++ ) {
116     if(cs_img.mono[i][j]==1){
117         cs_img.mono[i][j] = 0;//black
118     }
119     else{
120         cs_img.mono[i][j] = 255;//white
121     }
122 }
123 }
124
125 /*Connected Components based Segmentation */
126 ConnectedComponents(threshold, input_img.mono, input_img.width, input_img.height,\
127                     seg_img.mono,temp_img.mono, &NumRegions);
128 fprintf(stdout, "Number of regions in connected components: %d\n", NumRegions);
129
130 /* open grayscale image file */
131 if ( ( fp = fopen ( "connected_set.tif", "wb" ) ) == NULL ) {
132     fprintf ( stderr, "cannot open file connected_set.tif\n");
133     exit ( 1 );
134 }
135
136 /* write grayscale image */
137 if ( write_TIFF ( fp, &cs_img ) ) {
138     fprintf ( stderr, "error writing TIFF file connected_set.tif\n" );
139     exit ( 1 );
140 }
141
142 /* close image file */
143 fclose ( fp );
144
145 /* open grayscale image file */
146 if ( ( fp = fopen ( "segmented.tif", "wb" ) ) == NULL ) {
147     fprintf ( stderr, "cannot open file segmented.tif\n");
148     exit ( 1 );
149 }
150
151 /* write grayscale image */
152 if ( write_TIFF ( fp, &seg_img ) ) {
153     fprintf ( stderr, "error writing TIFF file segmented.tif\n" );
154     exit ( 1 );
155 }
156
157 /* close image file */
158 fclose ( fp );
159
160 /* de-allocate space which was used for the images */
161 free_TIFF ( &(input_img) );
162 free_TIFF ( &(cs_img) );
163 free_TIFF ( &(seg_img) );
164 free_TIFF ( &(temp_img) );
165
166 return(0);
167 }
168
169 void error(char *name)
170 {
171     printf("usage: %s input_image.tif output.tif s0.row s0.col Threshold\n\n",
172           name);
173     printf("this program reads in a 8-bit grayscale TIFF image.\n");

```

```

173     printf("It then computes segmentation mask of 4-connected component of s0,\n");
174     printf("and writes out the result as an 8-bit image\n");
175     exit(1);
176 }
177
178
179 void ConnectedNeighbors(
180 struct pixel s,
181 double threshold,
182 unsigned char **img,
183 int width,
184 int height,
185 int *M,
186 struct pixel c[4]){
187     int i,row,col, count = 0;
188     int row_offset[4]={0,0,-1,1};
189     int col_offset[4]={-1,1,0,0};
190     //iterate through neighbors
191     for(i=0; i<4; i++){
192         row = s.m + row_offset[i];
193         col = s.n + col_offset[i];
194         if ( ((row>=0) && (row<height)) &&\
195             ((col>=0) && (col<width)) ){
196             //check if in c(s)
197             if( abs(img[s.m][s.n] - img[row][col])<= threshold ){
198                 c[count].m = row; c[count].n = col;
199                 count+=1;
200             }
201         }
202     }
203     *M = count;
204     return;
205 }
206
207 void ConnectedSet(
208 struct pixel s,
209 double threshold,
210 unsigned char **img,
211 int width,
212 int height,
213 int ClassLabel,
214 unsigned char **seg,
215 int *NumConPixels){
216     struct pixel c[4];
217     int M=0, i;
218     bool flag_all_seg = true;
219     seg[s.m][s.n] = 1;
220     *NumConPixels+=1;
221     ConnectedNeighbors(s, threshold, img, width, height, &M, c);
222     /* base case */
223     if(M==0){return;} // no neighbors
224     for(i=0; i<M; i++){
225         if (seg[c[i].m][c[i].n]==0) { // i'th CN not accounted
226             flag_all_seg = false;
227             break;
228         }
229     }
230     if(flag_all_seg){return;}
231     else{

```

```

232     for(i=0; i<M; i++){
233         if (seg[c[i].m][c[i].n]==0){
234             ConnectedSet(c[i], threshold, img, width, height, ClassLabel, seg, NumConPixels);
235         }
236     }
237 }
238 }
239
240
241
242 void ConnectedSet_LL(
243 struct pixel s,
244 double threshold,
245 unsigned char **img,
246 int width,
247 int height,
248 int ClassLabel,
249 unsigned char **seg,
250 int *NumConPixels){
251     struct pixel c[4], i_pixel;
252     int M=0, i=0, k=0, count=0;
253     // char temp;
254     list <struct pixel> CN_LL;
255
256     //B<-{s0}
257     CN_LL.push_back(s);
258     while(!CN_LL.empty()){
259         i_pixel = CN_LL.front();
260         CN_LL.pop_front();
261         if(seg[i_pixel.m][i_pixel.n] == 0){//not labelled
262             seg[i_pixel.m][i_pixel.n] = ClassLabel;
263             count+=1;
264             ConnectedNeighbors(i_pixel, threshold, img, width, height, &M, c);
265             if(M>0){
266                 for(i=0; i<M; i++){//push not-visited pixels into list
267                     if(seg[c[i].m][c[i].n] == 0){CN_LL.push_back(c[i]); k++;}
268                 }
269                 k=0;
270             }
271         }
272         *NumConPixels = count;
273         return;
274     }
275
276
277 void ConnectedComponents(
278 double threshold,
279 unsigned char **img,
280 int width,
281 int height,
282 unsigned char **seg_out,
283 unsigned char **connected_set,
284 int *NumRegions
285 ){
286     int ClassLabel = 1, i_row, i_col, i, j;
287     int NumConPixels=0;
288     struct pixel seed;
289     for(i_row=0; i_row<height; i_row++){
290         for(i_col=0; i_col<width; i_col++){

```

```

291     if(seg_out[i_row][i_col]==0){//Not labelled
292         seed.m = i_row; seed.n = i_col;
293         ConnectedSet_LL(seed, threshold, img, width, height, 1, connected_set, &
NumConPixels);
294         //set connected_set back to zeros
295         for(i=0; i<height; i++){for(j=0; j<width; j++){connected_set[i][j]=0;}}
296         if(NumConPixels>MinPixCount){
297             //Label seg_out
298             ConnectedSet_LL(seed, threshold, img, width, height, ClassLabel, seg_out, &
NumConPixels);
299             ClassLabel+=1;
300         }
301     }
302 }
303 }
304 *NumRegions = ClassLabel-1;
305 return;
306 }

```

Appendix

Got to [git repo](#) for complete code.

Listing 3: Python code for generating segmentation image

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  @author: rahul
5  course: ECE637-DIP-I
6  lab3- Connected Components
7  """
8  import sys, os
9  import numpy as np
10 from PIL import Image
11 import matplotlib.pyplot as plt
12 import matplotlib as mp
13
14 np.random.seed(seed=637)
15
16 def main(filename):
17     basename = os.path.basename(filename).split('.')[0]
18     threshold = basename.split('_')[1]
19     im = Image.open(filename)
20     img = np.array(im)
21     NumRegions = np.max(img)
22     cmap = mp.colors.ListedColormap(np.random.rand(NumRegions,3))
23     plt.imshow(img,cmap=cmap)
24     plt.title('Segmentation Image: Threshold='
25             +str(threshold)+' NumRegions='+str(NumRegions))
26     plt.colorbar(shrink=0.5, aspect=5)
27     plt.savefig(basename+'.eps',bbox_inches='tight', pad_inches = 0, format='eps')
28
29 if __name__=="__main__":
30     filename = sys.argv[1]
31     main(filename)
```