ECE 637: Lab 3

Rahul Deshmukh

February 10, 2021

Section 1 Report

1. Print out of img22gd2.tif

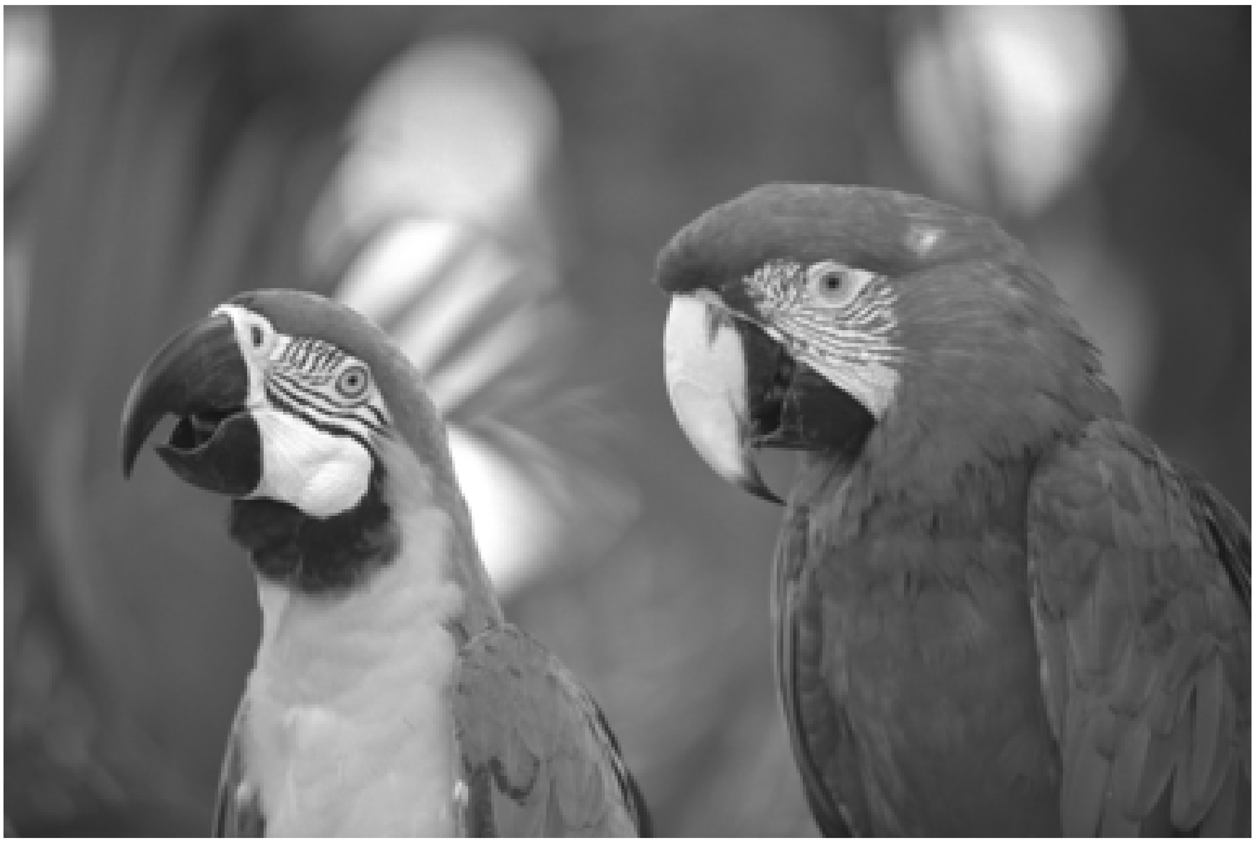
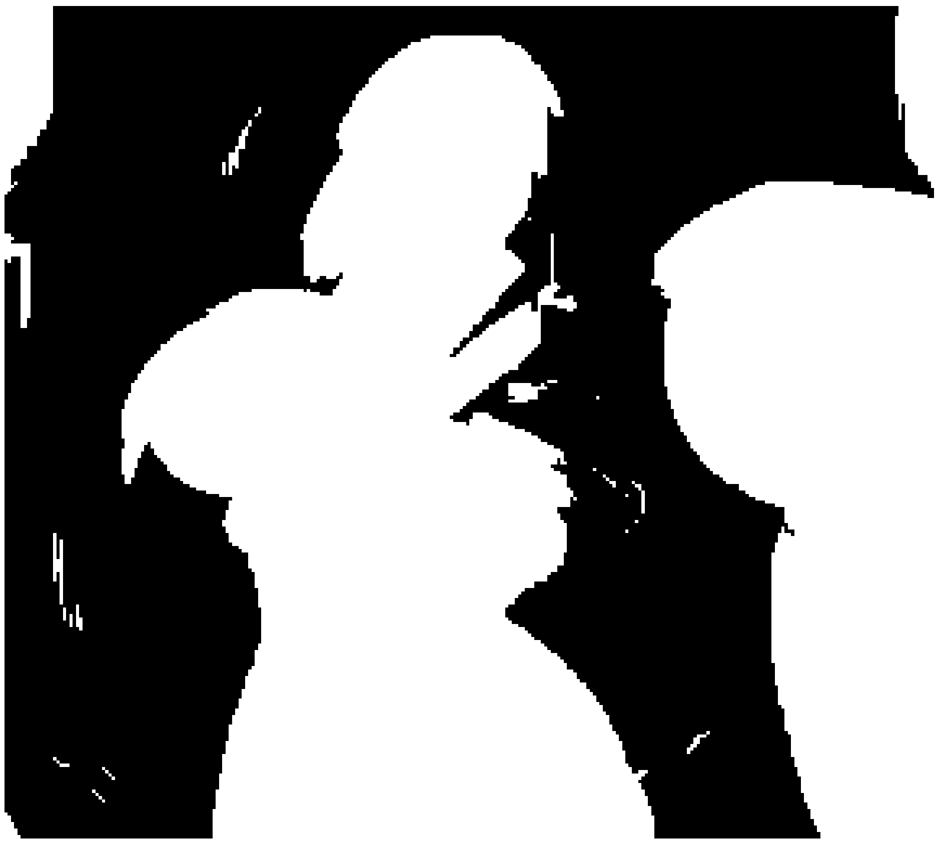


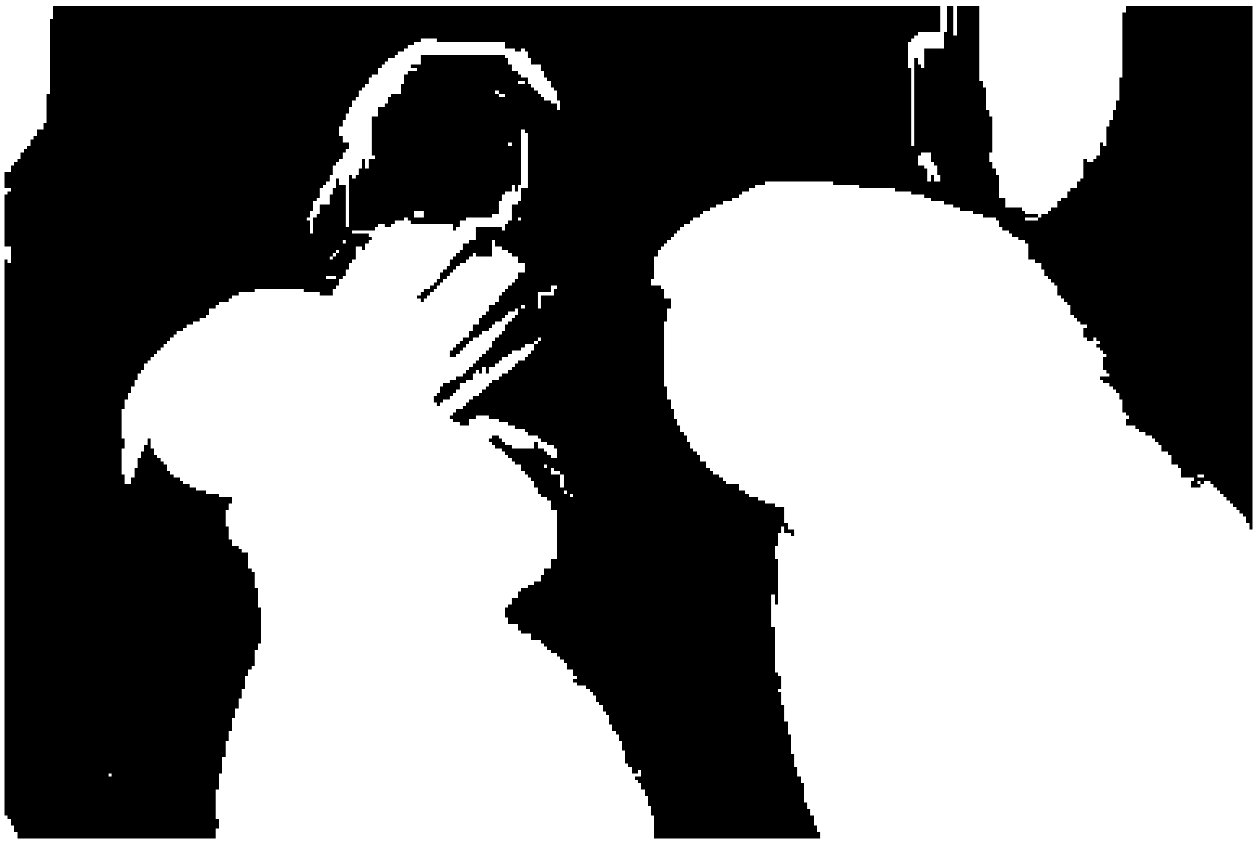
Figure 1: Input gray scale image

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 $T = 3$.

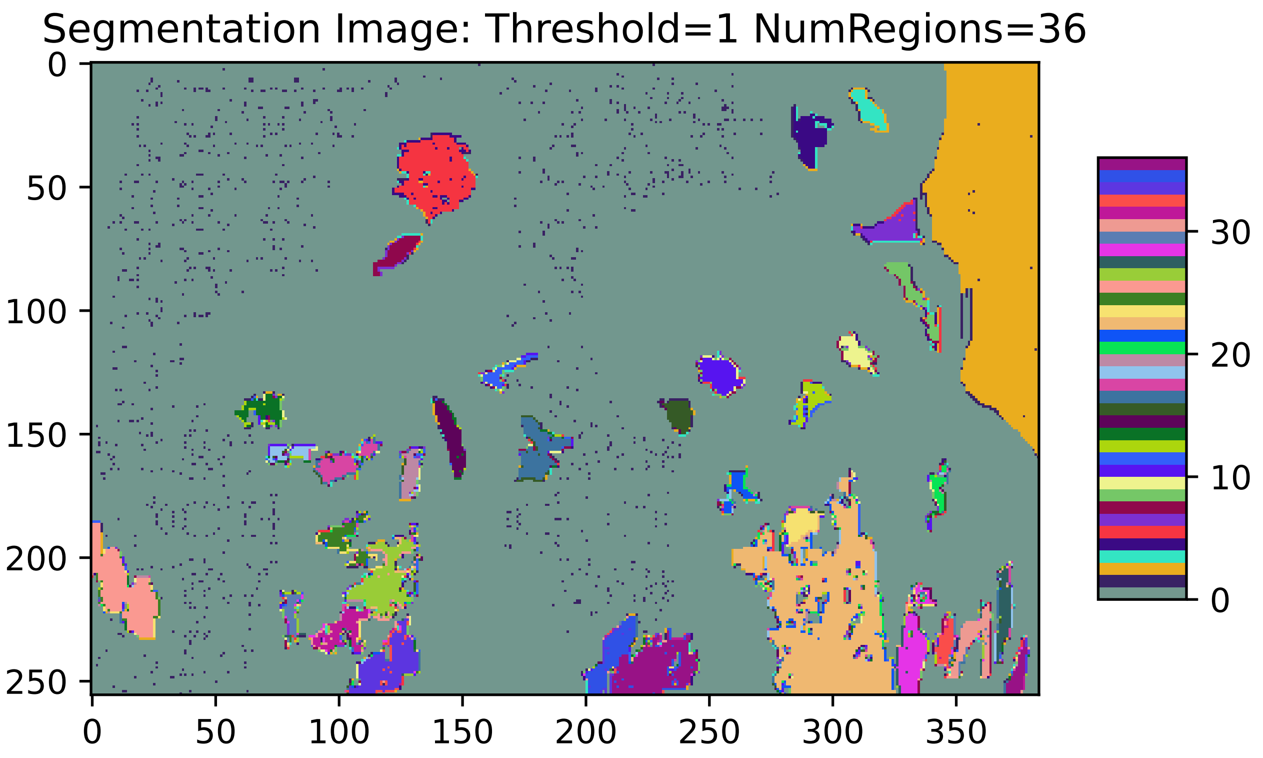


Figure 2: segmentation result for T=1

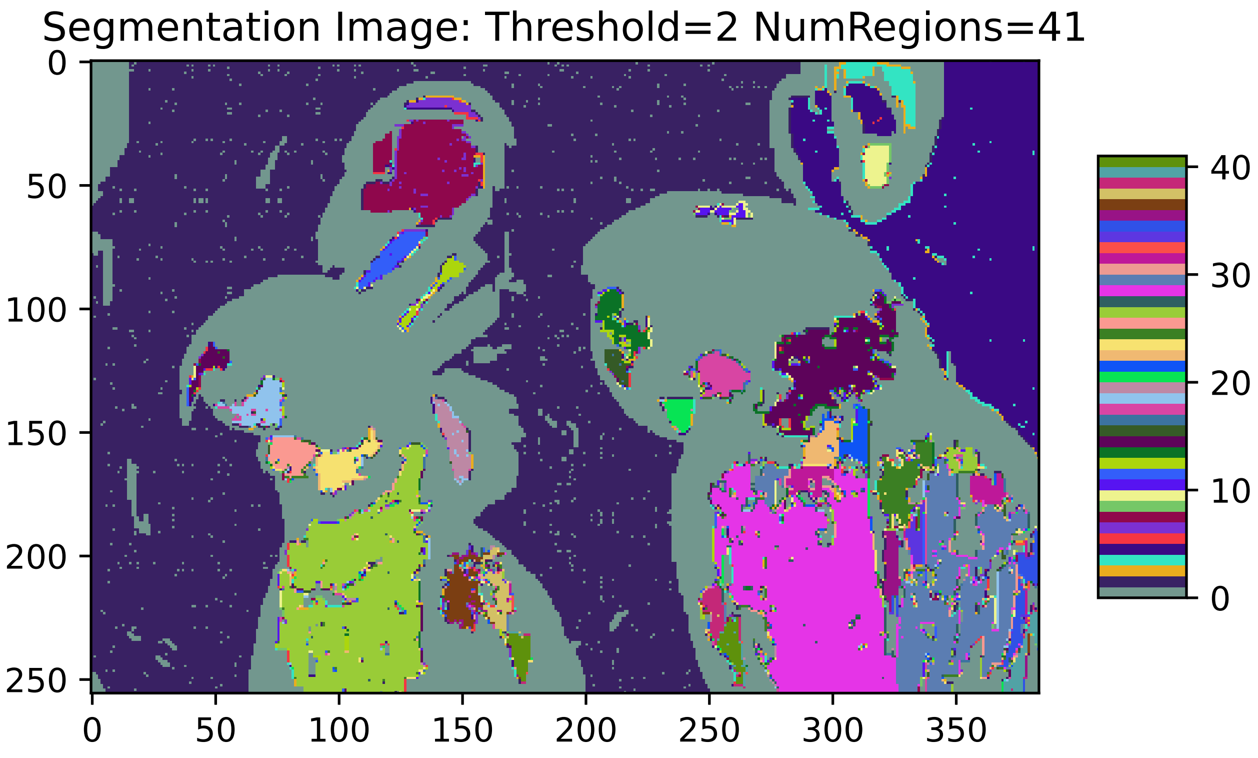


Figure 3: segmentation result for T=2

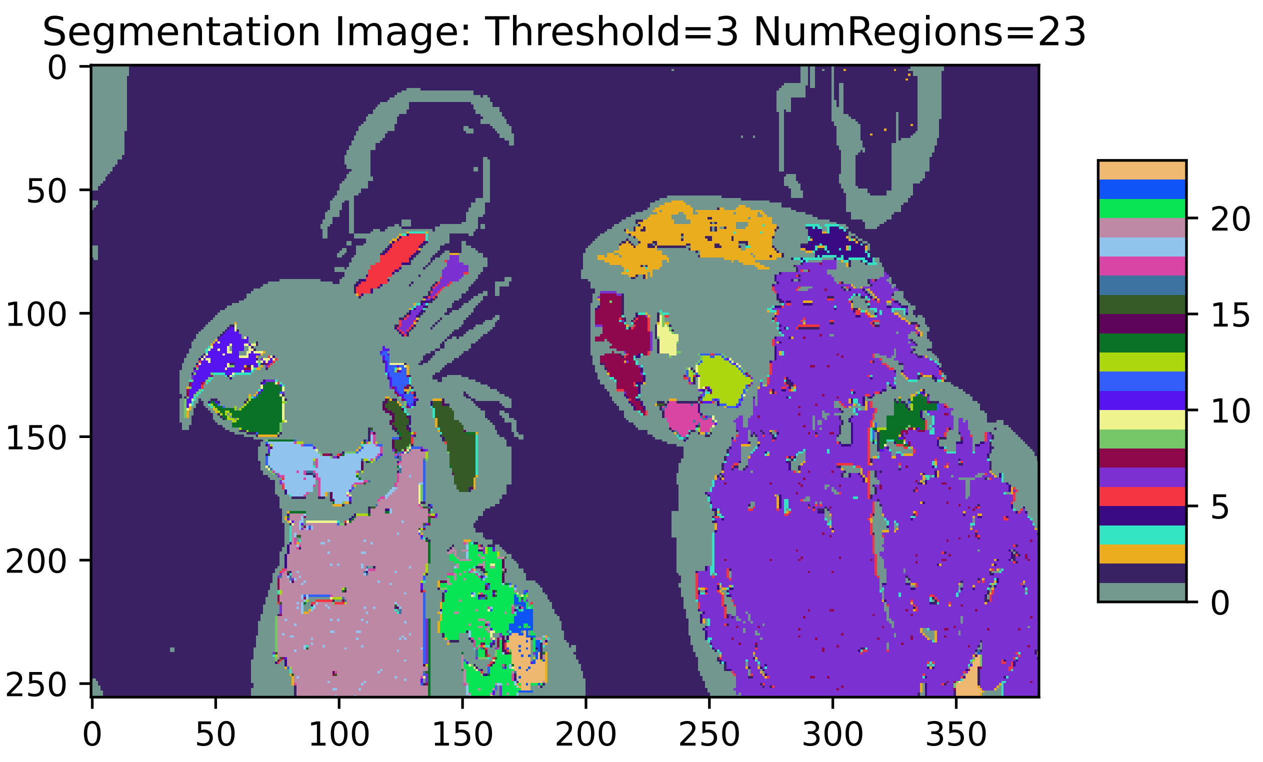


Figure 4: segmentation result for T=3

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

```
T =  2
Number of pixels in connected set: 36317
Number of regions in connected components: 41
T =  1
Number of pixels in connected set: 3
Number of regions in connected components: 36
T =  3
Number of pixels in connected set: 48926
Number of regions in connected components: 23
done
```

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

Source Code

Listing 2: C-Code

```

#include <math.h>
#include <stdlib.h>
#include <list>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"
using namespace std;

#define MinPixCount 100

struct pixel{int m, n; /*m= row, n= col*/};

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

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

void ConnectedSet_LL(
struct pixel s,
double threshold,
unsigned char **img,
int width,
int height,
int ClassLabel,
unsigned char **seg,
int *NumConPixels);

void ConnectedComponents(
double threshold,
unsigned char **img,
int width,
int height,
unsigned char **seg_out,
unsigned char **connected_set,
int *NumRegions
);

void error(char *name);

```

```c
int main (int argc, char **argv)
{
  FILE *fp;
  struct TIFF_img input_img, seg_img, cs_img, temp_img;
  int32_t i,j,NumConPixels=0, ClassLabel=1, NumRegions=0;
  double threshold;
  struct pixel s0;

  if ( argc != 5) error( argv[0] );
  s0.m = atof(argv[2]);//row
  s0.n = atof(argv[3]);//col
  threshold = atof(argv[4]);

  /* 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 8-bit monochrome\n" );
    exit ( 1 );
  }

  /* set up structure for output achromatic image */
  /* to allocate a full color image use type 'c' */
  get_TIFF ( &cs_img, input_img.height, input_img.width, 'g' );
  get_TIFF ( &seg_img, input_img.height, input_img.width, 'g' );
  get_TIFF ( &temp_img, input_img.height, input_img.width, 'g' );

  /* initialize segmentation map as 0 */
  for ( i = 0; i < input_img.height; i++ ){
  for ( j = 0; j < input_img.width; j++ ) {
    cs_img.mono[i][j] = 0;
    temp_img.mono[i][j] = 0;
    seg_img.mono[i][j]=0;
  }}

  /* find connected set*/
//  ConnectedSet(s0,threshold,input_img.mono,input_img.width,input_img.height,\
//        ClassLabel,cs_img.mono,&NumConPixels);


  ConnectedSet_LL(s0,threshold,input_img.mono,input_img.width,input_img.height,\
        ClassLabel,cs_img.mono,&NumConPixels);
  fprintf(stdout, "Number of pixels in connected set: %d\n", NumConPixels);

  /* change 1-black 0-white */
  for ( i = 0; i < input_img.height; i++ ){
```

```
  for ( j = 0; j < input_img.width; j++ ) {
    if(cs_img.mono[i][j]==1){
    cs_img.mono[i][j] = 0;//black
    }
    else{
      cs_img.mono[i][j] = 255;//white
    }
    }
  }

  /*Connected Components based Segmentation */
  ConnectedComponents(threshold, input_img.mono, input_img.width, input_img.height,\
                  seg_img.mono,temp_img.mono, &NumRegions);
  fprintf(stdout, "Number of regions in connected components: %d\n", NumRegions);

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

  /* write grayscale image */
  if ( write_TIFF ( fp, &cs_img ) ) {
      fprintf ( stderr, "error writing TIFF file connected_set.tif\n" );
      exit ( 1 );
  }

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

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

  /* write grayscale image */
  if ( write_TIFF ( fp, &seg_img ) ) {
      fprintf ( stderr, "error writing TIFF file segmented.tif\n" );
      exit ( 1 );
  }

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

  /* de-allocate space which was used for the images */
  free_TIFF ( &(input_img) );
  free_TIFF ( &(cs_img) );
  free_TIFF ( &(seg_img) );
  free_TIFF ( &(temp_img) );

  return(0);
}

void error(char *name)
{
    printf("usage:  %s  input_image.tiff output.tiff s0.row s0.col Threshold\n\n",
   name);
    printf("this program reads in a 8-bit grayscale TIFF image.\n");
    printf("It then computes segmentation mask of 4-connected component of s0,\n");
    printf("and writes out the result as an 8-bit image\n");
    exit(1);
}


void ConnectedNeighbors(
struct pixel s,
double threshold,
unsigned char **img,
int width,
int height,
int *M,
struct pixel c[4]){
  int i,row,col, count = 0;
  int row_offset[4]={0,0,-1,1};
  int col_offset[4]={-1,1,0,0};
  //iterate through neighbors
  for(i=0; i<4; i++){
    row = s.m + row_offset[i];
    col = s.n + col_offset[i];
    if ( ((row>=0) && (row<height)) &&\
       ((col>=0) && (col<width )) ){
      //check if in c(s)
      if( abs(img[s.m][s.n] - img[row][col])<= threshold ){
        c[count].m = row; c[count].n = col;
        count+=1;
      }
    }
  }
  *M = count;
  return;
}

void ConnectedSet(
struct pixel s,
double threshold,
unsigned char **img,
int width,
int height,
int ClassLabel,
unsigned char **seg,
int *NumConPixels){
  struct pixel c[4];
  int M=0, i;
  bool flag_all_seg = true;
  seg[s.m][s.n] = 1;
  *NumConPixels+=1;
  ConnectedNeighbors(s,threshold,img,width,height,&M,c);
  /* base case */
  if(M==0){return;}// no neighbors
  for(i=0; i<M; i++){
    if (seg[c[i].m][c[i].n]==0) {// i'th CN not accounted
      flag_all_seg = false;
      break;
    }
  }
  if(flag_all_seg){return;}
  else{
    for(i=0; i<M; i++){
      if (seg[c[i].m][c[i].n]==0){
        ConnectedSet(c[i],threshold,img,width,height,ClassLabel,seg,NumConPixels);
      }
    }
  }
}



void ConnectedSet_LL(
struct pixel s,
double threshold,
unsigned char **img,
int width,
int height,
int ClassLabel,
unsigned char **seg,
int *NumConPixels){
  struct pixel c[4], i_pixel;
  int M=0,i=0,k=0, count=0;
//  char temp;
  list <struct pixel> CN_LL;

  //B<-{s0}
  CN_LL.push_back(s);
  while(!CN_LL.empty()){
    i_pixel = CN_LL.front();
    CN_LL.pop_front();
    if(seg[i_pixel.m][i_pixel.n] == 0){//not labelled
    seg[i_pixel.m][i_pixel.n] = ClassLabel;
    count+=1;
    ConnectedNeighbors(i_pixel,threshold,img,width,height,&M,c);
    if(M>0){
    for(i=0;i<M;i++){//push not-visited pixels into list
    if(seg[c[i].m][c[i].n] == 0){CN_LL.push_back(c[i]);k++;}
    }}
    k=0;
    }
    }
  *NumConPixels = count;
  return;
}


void ConnectedComponents(
double threshold,
unsigned char **img,
int width,
int height,
unsigned char **seg_out,
unsigned char **connected_set,
int *NumRegions
){
  int ClassLabel = 1,i_row,i_col, i,j;
  int NumConPixels=0;
  struct pixel seed;
  for(i_row=0; i_row<height; i_row++){
    for(i_col=0; i_col<width; i_col++){
      if(seg_out[i_row][i_col]==0){//Not labelled
        seed.m = i_row; seed.n = i_col;
        ConnectedSet_LL(seed, threshold, img, width, height, 1, connected_set, &
  NumConPixels);
        //set connected_set back to zeros
        for(i=0; i<height; i++){for(j=0; j<width; j++){connected_set[i][j]=0;}}
        if(NumConPixels>MinPixCount){
        //Label seg_out
        ConnectedSet_LL(seed, threshold, img, width, height, ClassLabel, seg_out, &
  NumConPixels);
        ClassLabel+=1;
        }
      }
    }
  }
  *NumRegions = ClassLabel-1;
  return;
}
```

Appendix

Got to git repo for complete code.

Listing 3: Python code for generating segmentation image

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: rahul
course: ECE637-DIP-I
lab3- Connected Components
"""
import sys, os
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib as mp

np.random.seed(seed=637)

def main(filename):
    basename = os.path.basename(filename).split('.')[0]
    threshold = basename.split('_')[1]
    im = Image.open(filename)
    img = np.array(im)
    NumRegions = np.max(img)
    cmap = mp.colors.ListedColormap(np.random.rand(NumRegions,3))
    plt.imshow(img,cmap=cmap)
    plt.title('Segmentation Image: Threshold='
              +str(threshold)+' NumRegions='+str(NumRegions))
    plt.colorbar(shrink=0.5, aspect=5)
    plt.savefig(basename+'.eps',bbox_inches='tight', pad_inches = 0, format='eps')

if __name__=="__main__":
    filename = sys.argv[1]
    main(filename)
```