ECE 637 Lab 3 - Neighborhoods and Connected Components

Yifan Fei

Electrical Engineering
Purdue University

Instructor: Prof. Charles A. Bouman

Spring 2018

Section 1: Area Fill

1: A print out the image img22gd2.tif.

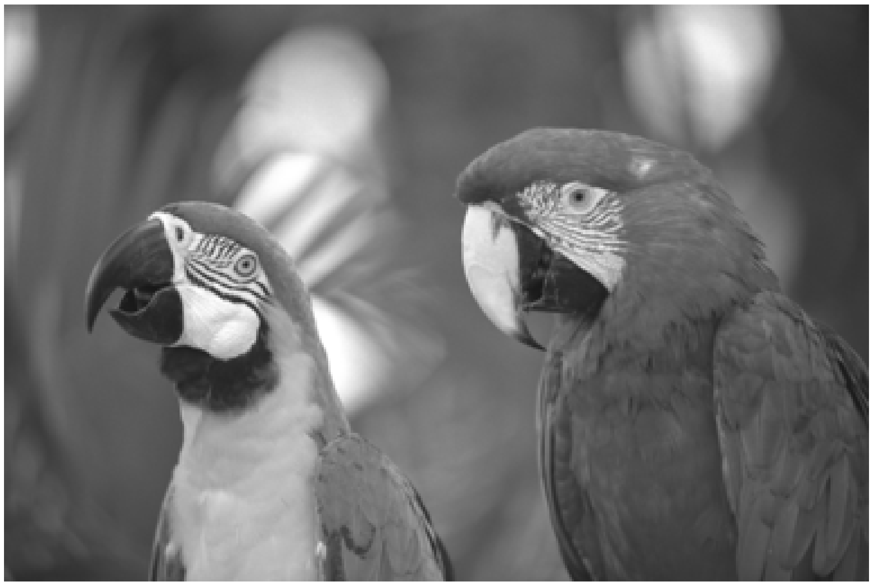


Figure 1: image img22gd2.tif

2: image with T = 2

The Connected number is 36317.

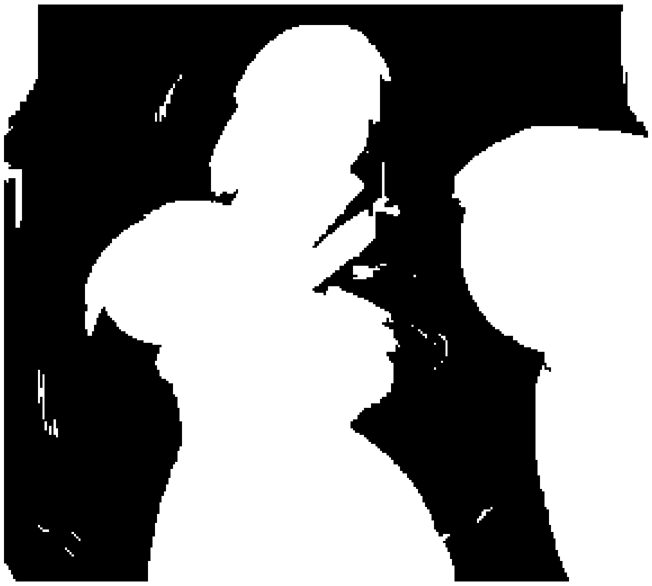


Figure 2: The connected set with T = 2

3: image with T = 1

It is clear that when T = 1, there is only a few points connected with s = (67,45). The Connected number is 3. You can see 3 black pixeles there.

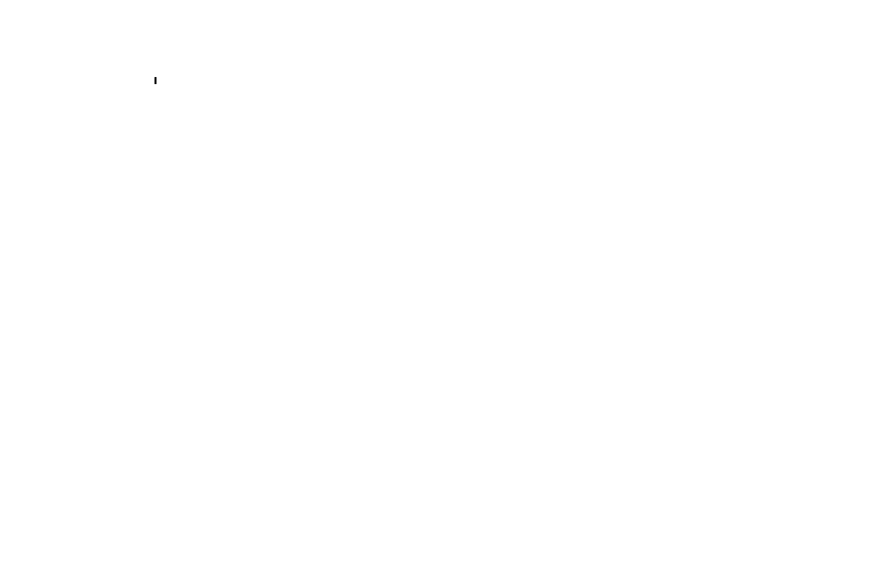


Figure 3: The connected set with T = 1

4: image with T = 3

The Connected number is 48926.

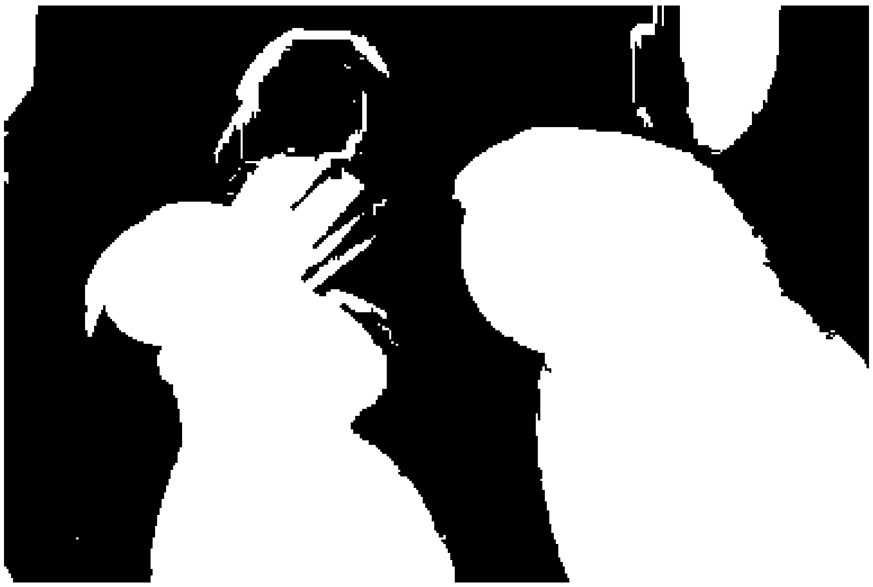


Figure 4: The connected set with T = 3

5: C code

See the attachment.

Section2: Image Segmentation

0: Without filter that requires sets containing greater than 100 pixels

Notice that in this part, the size of most connected sets for this image will be small, resulting in a large number of connected sets in the segmentation. The 3 plots show that too much noise exists.

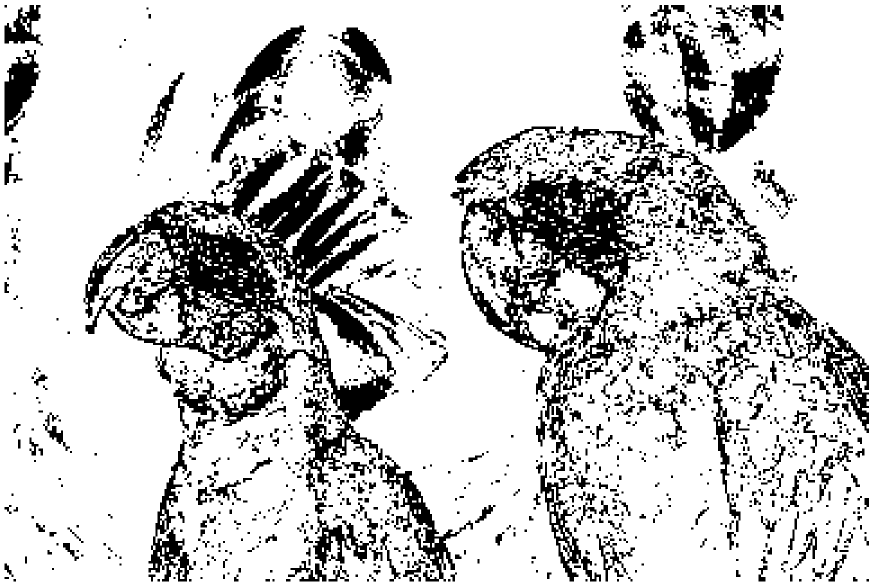


Figure 5: The connected set with T = 1, without filter

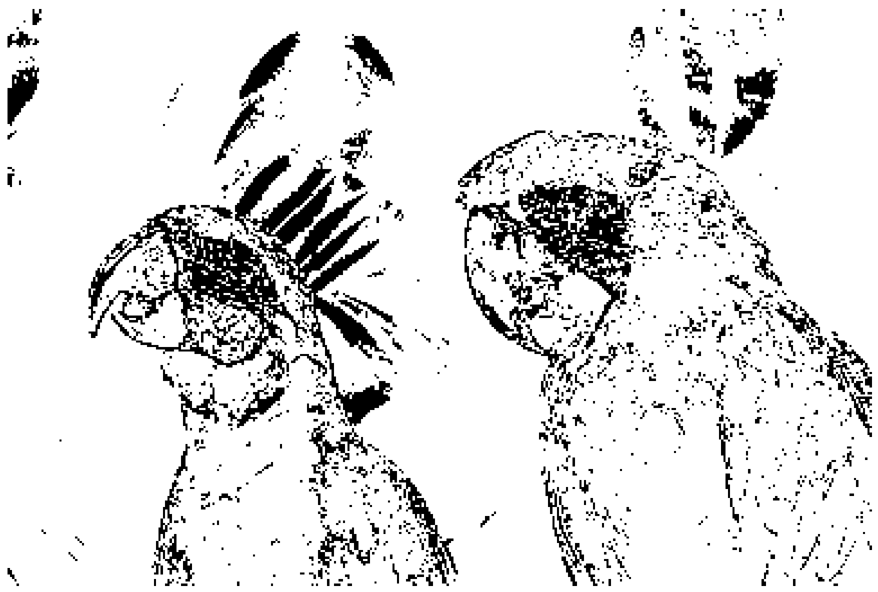


Figure 6: The connected set with T = 2,without filter

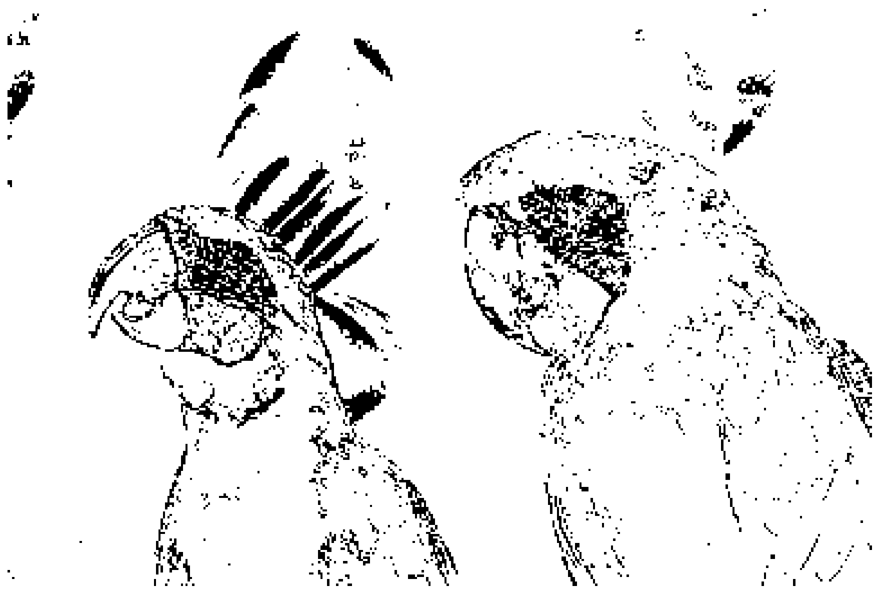


Figure 7: The connected set with T = 3,without filter

1: randomly colored segmentations

Before inputting the images into the matlab, they are all almost black. The reason is that the group numbers start at 0 to about 40, in which range the plots is black. After colored randomly, they look like:

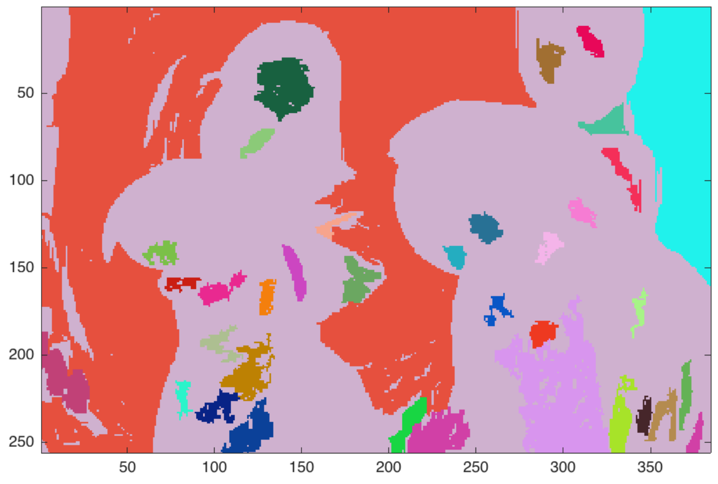


Figure 8: The colored connected set with T = 1

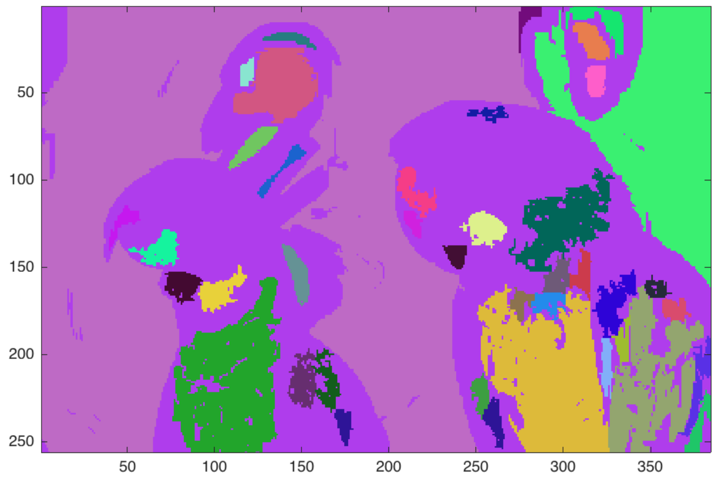


Figure 9: The colored connected set with T = 2

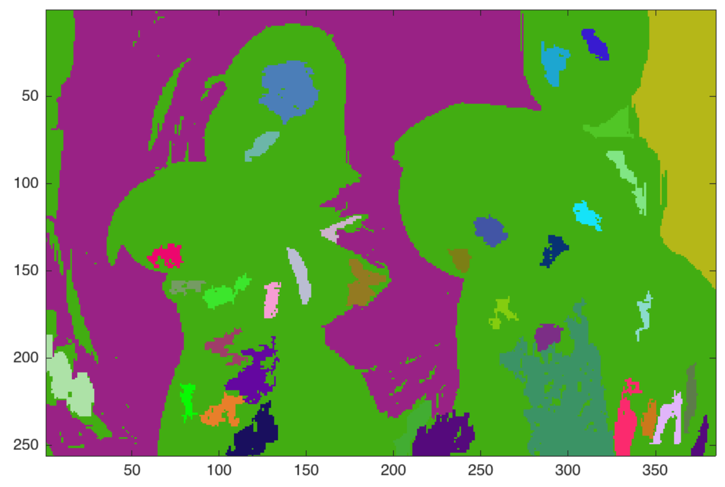
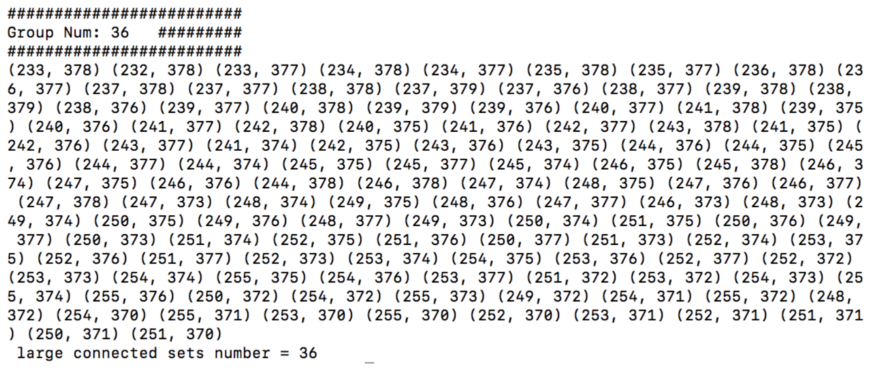


Figure 10: The colored connected set with T = 3

2: number of regions

The number of regions generated for T = 1 is 36. For T = 2, it is 41, and for T =3, it is 23. The code can print out the group number and details about the pixels in each region. The plot is shown below.



```
##########################
Group Num: 36   #########
##########################
(233, 378) (232, 378) (233, 377) (234, 378) (234, 377) (235, 378) (235, 377) (236, 378) (23
6, 377) (237, 378) (237, 377) (238, 378) (237, 379) (237, 376) (238, 377) (239, 378) (238,
379) (238, 376) (239, 377) (240, 378) (239, 379) (239, 376) (240, 377) (241, 378) (239, 375
) (240, 376) (241, 377) (242, 378) (240, 375) (241, 376) (242, 377) (243, 378) (241, 375) (
242, 376) (243, 377) (241, 374) (242, 375) (243, 376) (243, 375) (244, 376) (244, 375) (245
, 376) (244, 377) (244, 374) (245, 375) (245, 377) (245, 374) (246, 375) (245, 378) (246, 3
74) (247, 375) (246, 376) (244, 378) (246, 378) (247, 374) (248, 375) (247, 376) (246, 377)
 (247, 378) (247, 373) (248, 374) (249, 375) (248, 376) (247, 377) (246, 373) (248, 373) (2
49, 374) (250, 375) (249, 376) (248, 377) (249, 373) (250, 374) (251, 375) (250, 376) (249,
 377) (250, 373) (251, 374) (252, 375) (251, 376) (250, 377) (251, 373) (252, 374) (253, 37
5) (252, 376) (251, 377) (252, 373) (253, 374) (254, 375) (253, 376) (252, 377) (252, 372)
(253, 373) (254, 374) (255, 375) (254, 376) (253, 377) (251, 372) (253, 372) (254, 373) (25
5, 374) (255, 376) (250, 372) (254, 372) (255, 373) (249, 372) (254, 371) (255, 372) (248,
372) (254, 370) (255, 371) (253, 370) (255, 370) (252, 370) (253, 371) (252, 371) (251, 371
) (250, 371) (251, 370)
 large connected sets number = 36
```

Figure 11: The printout details about T =1, 36 groups

Attachments: C code

1:Code for section 1

Code for section 1 contains 3 parts, the .h file called mysubroutine.h, the .c file called mysubroutine.c, and the main function called Section1.c.

mysubroutine.h

```
#ifndef _MYSUBROUTINE_H_
#define _MYSUBROUTINE_H_

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


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

typedef struct node {
    struct pixel val;
    struct node * next;
} node_t;


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, uint8_t **seg,
                int *NumConPixels);
#endif
```

mysubroutine.c

```
# include "mysubroutine.h"

/*   connected pixels   */
/* c[0]:left, c[1]:up, c[2]:right, c[3]:down  */
void ConnectedNeighbors(struct pixel s,double T,
                        unsigned char **img,int width,
                        int height,int *M,struct pixel c[4]){
*M = 4;
if (s.m == 0) {
        c[0].m = -1;
        c[0].n = -1;
        (*M)--;
} else {
        if (abs(img[s.m][s.n] - img[s.m-1][s.n])<=T){
                c[0].m = s.m - 1 ;
                c[0].n = s.n;
        }else{
                c[0].m = -1;
                c[0].n = -1;
                (*M)--;}
        }

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

if (s.m == height - 1){
  c[2].m = -1;
  c[2].n = -1;
  (*M)--;

} else{
        if (abs(img[s.m][s.n] - img[s.m+1][s.n])<=T){
                c[2].m = s.m + 1 ;
                c[2].n = s.n;
        } else{
                c[2].m = -1;
                c[2].n = -1;
                (*M)--;}
        }

if (s.n == width - 1){
  c[3].m = -1;
  c[3].n = -1;
  (*M)--;
} else{
        if (abs(img[s.m][s.n] - img[s.m][s.n+1])<=T){
                c[3].m = s.m;
                c[3].n = s.n + 1;
        }else{
                c[3].m = -1;
                c[3].n = -1;
                (*M)--;}
        }
}


/* http://www.learn-c.org/en/Linked_lists */
/* Adding an item to the end of the list */

void push(node_t ** tail, struct pixel val) {
        node_t * newtail = NULL;
        (*tail)->next = malloc(sizeof(node_t));
        newtail = (*tail)->next;
        newtail->val = val;
        newtail->next = NULL;
        // free(*tail);
        *tail = newtail;
}

/* Removing the first item */
/* Delete an item at the head of the list */
void pop(node_t ** head) {
        node_t * next_node = NULL;
        struct pixel retval;
        next_node = (*head)->next;
        retval = (*head)->val;
        free(*head);
        *head = next_node;
}

void ConnectedSet(struct pixel s,double T,
                unsigned char **img,int width,int height,
                int ClassLabel,uint8_t **seg,
                int *NumConPixels){
struct pixel c[4];
int neighbor_num; // M \leq 4 here
int i, j;
int loop = 1; //
struct pixel retval;

/* use linked list to store, initialization of linked list */
node_t * head = NULL;
node_t * tail = NULL;
head = malloc(sizeof(node_t));
head->val = s;
head->next = NULL;
tail = head;

*NumConPixels = 0;
j = 0;
while (head!= NULL || loop ==1 ){
    loop = 0;
    ConnectedNeighbors (head->val,T,img,width,height, &neighbor_num, c);
    for (i =0; i < 4; i++){
        if (c[i].m!=-1 && c[i].n!=-1){
            if(seg[c[i].m][c[i].n]!= ClassLabel){
                                seg[c[i].m][c[i].n] = ClassLabel;
                push(&tail,c[i]);
                printf("push(%d,%d)",c[i].m,c[i].n);
                (*NumConPixels)++;
                                j++;
            }
        }
    }
    pop(&head);

}
}
```

Section1.c

```
// gcc -o Section1 Section1.c allocate.c randlib.c solve.c tiff.c mysubroutine.c
// ./Section1 img22gd2.tif 2

#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"
#include "mysubroutine.h"

void error(char *name);

int main (int argc, char **argv)
{
  FILE *fp;
  struct TIFF_img input_img, seg_img;
  struct pixel s;
  struct pixel c[4];
  double **img1;
  double T;
  int connect_num;
  int ClassLabel = 0;
  int neighbor_num;
  int32_t i,j;

  if ( argc != 3 ) error( argv[0] );

  T = atof(argv[2]); // third input
  /* 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 );

  if ( input_img.TIFF_type != 'g' ) {
    fprintf ( stderr, "error:  image must be monochrome\n" );
    exit ( 1 );
  }
```

```
printf("T: %f\n", T);
/* Allocate image of double precision floats
Extend the image for filtering*/
img1 = (double **)get_img(input_img.width,input_img.height,sizeof(double));

/* to allocate a full color image use type 'c' */
get_TIFF ( &seg_img, input_img.height, input_img.width, 'g' );

for(i = 0; i<input_img.height; i++)
  for(j = 0; j<input_img.width; j++){
    // seg_img.mono[i][j] = 0;
    seg_img.mono[i][j] = 255; // set to be white
  }
  printf("image dimensions: height:%d  width: %d\n",input_img.height, input_img.width );

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

  ConnectedSet(s,T,input_img.mono,input_img.width,input_img.height,
    ClassLabel,seg_img.mono, &connect_num);
  printf("Connected number: %d\n",connect_num);

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

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

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

/* de-allocate space which was used for the images */
  free_TIFF ( &(input_img) );
  free_TIFF ( &(seg_img) );
  free_img( (void**)img1 );

  return(0);
}

void error(char *name)
{
  printf("usage:  %s  image.tiff \n\n",name);
  printf("this program reads in a 24-bit color TIFF image.\n");
  printf("It then horizontally filters the green component, adds noise,\n");
  printf("and writes out the result as an 8-bit image\n");
  printf("with the name 'green.tiff'.\n");
  printf("It also generates an 8-bit color image,\n");
```

```
    printf("that swaps red and green components from the input image");
    exit(1);
  }
```

2:Code for section 2

Code for section 1 contains 3 parts, the .h file called mysubroutine2.h, the .c file called mysubroutine2.c, and the main function called Section2.c. Notice that to deal with cutting off the group numbers, some fuctions and more input and output than that in section 1.

mysubroutine2.h

```
#ifndef _MYSUBROUTINE2_H_
#define _MYSUBROUTINE2_H_

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"
#include "mysubroutine2.h"


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

/*create linked list structure*/
typedef struct node {
    struct pixel val;
    struct node * next;
} node_t;


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, uint8_t **seg, uint8_t **seg_original,
                int *NumConPixels, int *large_area_number);
/*seg_original store segmentation without filter
seg store segmentation with filter(pixel > 100)
NumConPixels store all connected points for one particular set,
*large_area_number store number of sets that have more than 100 pixel*/
#endif
```

mysubroutine2.c

```
# include "mysubroutine2.h"
```

```
/*    connected pixels    */
/* c[0]:left, c[1]:up, c[2]:right, c[3]:down  */
void ConnectedNeighbors(struct pixel s,double T,
                        unsigned char **img,int width,
                        int height,int *M,struct pixel c[4]){
*M = 4;
if (s.m == 0) {
  c[0].m = -1;
  c[0].n = -1;
  (*M)--;
} else {
  if (abs(img[s.m][s.n] - img[s.m-1][s.n])<=T){
    c[0].m = s.m - 1 ;
    c[0].n = s.n;
  }else{
    c[0].m = -1;
    c[0].n = -1;
    (*M)--;}
  }

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

if (s.m == height - 1){
  c[2].m = -1;
  c[2].n = -1;
  (*M)--;
} else{
  if (abs(img[s.m][s.n] - img[s.m+1][s.n])<=T){
    c[2].m = s.m + 1 ;
    c[2].n = s.n;
  } else{
    c[2].m = -1;
    c[2].n = -1;
    (*M)--;}
  }

if (s.n == width - 1){
  c[3].m = -1;
  c[3].n = -1;
  (*M)--;
} else{
  if (abs(img[s.m][s.n] - img[s.m][s.n+1])<=T){
    c[3].m = s.m;
    c[3].n = s.n + 1;
  }else{
    c[3].m = -1;
    c[3].n = -1;
    (*M)--;}
  }
}

/*    a->b = (*a).b    */
/* http://www.learn-c.org/en/Linked_lists */
/* Adding an item to the end of the list */

void push(node_t ** tail, struct pixel val) {
  node_t * temp = NULL;
  (*tail)->next = malloc(sizeof(node_t)); // memory allocation
  temp = (*tail)->next;
  temp->val = val;
  temp->next = NULL;
  *tail = temp;
}

/* Delete an item at the head of the list */
void pop(node_t ** head) {
  node_t * next_node = NULL;
  struct pixel retval;
  next_node = (*head)->next;
  retval = (*head)->val;
  free(*head);
  *head = next_node;
}

void ConnectedSet(struct pixel s,double T,
                unsigned char **img,int width,int height,
                int ClassLabel,uint8_t **seg, uint8_t **seg_original,
                int *NumConPixels, int *large_area_number){
struct pixel c[4];
int neighbor_num;
int i, j, x, y;
int loop = 1; // run for 1 time, set to 0 later
struct pixel retval;
int *expandedArray;
/* use linked list to store */
node_t * head = NULL;
node_t * tail = NULL;
head = malloc(sizeof(node_t));
head->val = s;
head->next = NULL;
tail = head;
expandedArray = malloc(sizeof(int) * width * height);
memset(expandedArray, -1, sizeof(int) * width * height); // expandedArray initialization
// Sets the first n bytes of the block of memory pointed by ptr to the specified value
*NumConPixels = 0;
```

```
j = 0;
while (head!= NULL || loop ==1 ){
  loop = 0;
  ConnectedNeighbors (head->val,T,img,width,height, &neighbor_num, c);
    for (i =0; i < 4; i++){
      if (c[i].m!=-1 && c[i].n!=-1){
        if(seg_original[c[i].m][c[i].n]!= ClassLabel){
          seg_original[c[i].m][c[i].n] = ClassLabel;
          expandedArray[j] = c[i].m * width + c[i].n;
          // expand each satisfied set to one dimention, c[i].m leq height, c[i].n leq wid
          /*printf("push c(%d,%d)",c[i].m,c[i].n); */
          push(&tail,c[i]);
          (*NumConPixels)++;
          j++;
        }
      }
    }
  // retval = pop(&head);
  pop(&head);
}

if(*NumConPixels > 100) {
  j = 0;
  (*large_area_number)++;
  printf("\n########################\n");
  printf("Group Num: %d   #########\n", *large_area_number);
  printf("########################\n");
  /* store the label for each pixel as a 2-D unsigned character array */
  while (expandedArray[j] != -1) {
    y = expandedArray[j] % width;
    x = (expandedArray[j] - y)/width;
    printf("(%d, %d) ", x, y);
    // seg[x][y] = ClassLabel;
    seg[x][y] = *large_area_number;
    j++;
  }
}
free(expandedArray);

}
```

Section2.c

```
// gcc -o Section2 Section2.c allocate.c randlib.c solve.c tiff.c mysubroutine2.c -g
// ./Section2 img22gd2.tif 2

#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"
#include "mysubroutine2.h"
```

```
void error(char *name);

int main (int argc, char **argv)
{
  FILE *fp;
  struct TIFF_img input_img, seg_img, seg_original;
  struct pixel s;
  struct pixel c[4];
  double T;
  int ClassLabel = 255;
  int connect_num, neighbor_num, large_connect_num = 0;
  int32_t i,j;

  if ( argc != 3 ) error( argv[0] );
  /* input lamda*/
  T = atof(argv[2]);
  /* 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 );

  if ( input_img.TIFF_type != 'g' ) {
    fprintf ( stderr, "error:  image must be monochrome\n" );
    exit ( 1 );
  }

  /* to allocate a image use type 'g' */
  get_TIFF ( &seg_img, input_img.height, input_img.width, 'g' );
  get_TIFF( &seg_original, input_img.height, input_img.width, 'g');

  /* Initialization of seg */
  for(i = 0; i<input_img.height; i++)
    for(j = 0; j<input_img.width; j++){
      seg_img.mono[i][j] = 0;
      seg_original.mono[i][j] = 0;
    }

    printf("T: %f\n", T);

  /* Get large connected sets number  */
    for (i = 0; i<input_img.height; i++)
      for (j = 0; j<input_img.width; j++) {
        s.m = i;
        s.n = j;
        if (seg_original.mono[i][j] != ClassLabel) {
          ConnectedSet(s, T, input_img.mono, input_img.width, input_img.height,
            ClassLabel, seg_img.mono, seg_original.mono, &connect_num, &large_connect_num)
        }
      }
      printf("\n large connected sets number = %d \n", large_connect_num);

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

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

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

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

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

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

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

      return(0);
    }

    void error(char *name)
    {
      printf("usage:  %s  image.tiff \n\n",name);
      printf("this program reads in a 24-bit color TIFF image.\n");
      printf("It then horizontally filters the green component, adds noise,\n");
      printf("and writes out the result as an 8-bit image\n");
      printf("with the name 'green.tiff'.\n");
      printf("It also generates an 8-bit color image,\n");
      printf("that swaps red and green components from the input image");
    exit(1);
}
```