Digital Image Processing Laboratory 3

Neighborhoods and Connected Components

Praneet Singh

100

February 7, 2020

Note: The **ConnectedSet()** and **ConnectedNeighbors()** functions can be found in **utils.c** which has been appended along with its header at the end of this document.

1 Area Fill

In this section, we've written a C program that fills in an area of connected pixels in an image. The first index in every coordinate is the column and the second index is the row.

1.1 The Original image *img22gd2.tif*

1.2 Image showing the connected set for s = (67,45), and T = 2

1.3 Image showing the connected set for s = (67,45), and T = 1

1.4 Image showing the connected set for s = (67,45), and T = 3

1.5 Area Fill C-Code

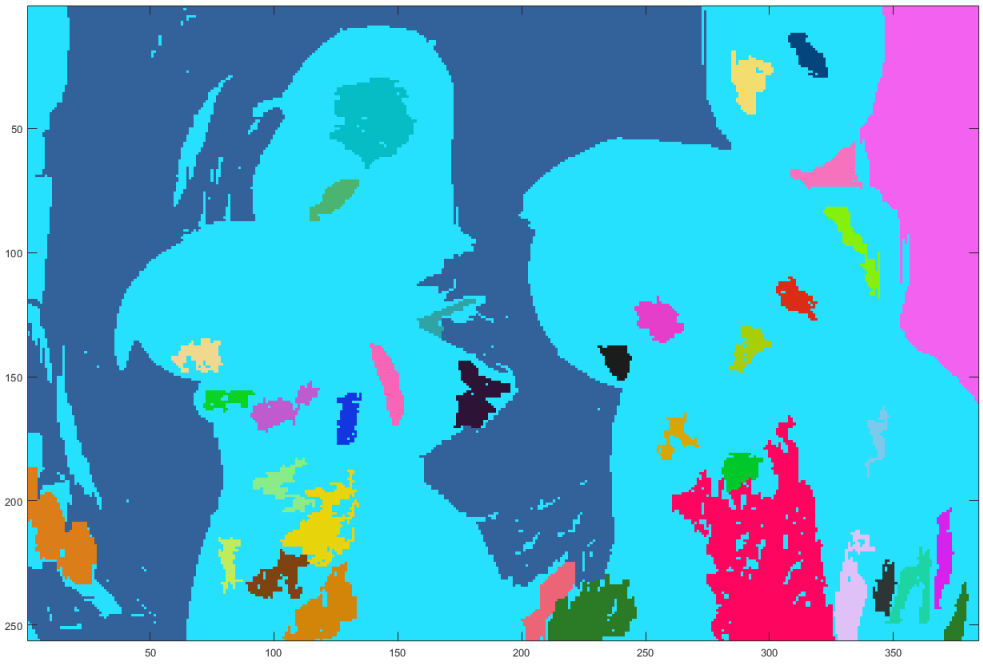
```c
#include <stdio.h>
#include "utils.h"


void error(char *name);

int main (int argc, char **argv) {
  FILE *fp;
  struct TIFF_img input_img,output_img;
  struct pixel s;
  int i, j, numcon;
  int ClassLabel = 1;
  double T;

  if (argc != 6) error(argv[0]);

  /* 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 grayscale\n");
```

```
    exit(1);
  }

  sscanf(argv[2], "%d", &(s.n));
  sscanf(argv[3], "%d", &(s.m));
  sscanf(argv[4], "%lf", &T);

  unsigned int **seg = (unsigned int **)get_img(input_img.width,
                                                input_img.height,
                                                sizeof(unsigned int));

  ConnectedSet(s, T, input_img.mono, input_img.width, input_img.height,ClassLabel, seg
    , &numcon);

  get_TIFF(&output_img,input_img.height,input_img.width,'g');

  for (i = 0; i < input_img.height; i++) {
    for (j = 0; j < input_img.width; j++) {
      if (seg[i][j] == ClassLabel) {
        output_img.mono[i][j] = 0;
      } else {
        output_img.mono[i][j] = 255;
      }
    }
  }

  /* open output image file */
  if ((fp = fopen(argv[5], "wb")) == NULL) {
    fprintf(stderr, "cannot open file output.tif\n");
    exit(1);
  }

  /* write output image */
  if(write_TIFF(fp, &output_img)) {
    fprintf(stderr, "error writing TIFF file %s\n", argv[5]);
    exit(1);
  }

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

  /* de-allocate space which was used for the images */
  free_TIFF(&(input_img));
  free_TIFF(&(output_img));
  free_img(*seg);

  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);
}
```

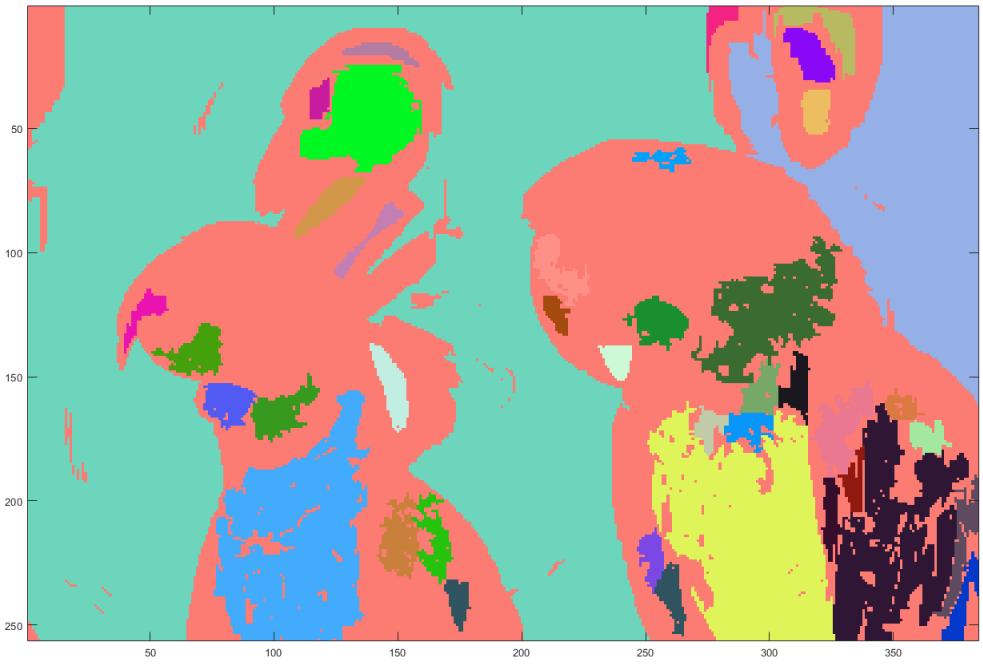
2 Image Segmentation

In this section, we have used the subroutines for region filling to segment the image into connected components.

2.1 Segmented Image for T = 1



2.2 Segmented Image for T = 2



2.3 Segmented Image for T = 3

2.4 Number of Regions Generated

1. T=1, Number of Connected Sets with 100 or more members = 36
2. T=2, Number of Connected Sets with 100 or more members = 41
3. T=3, Number of Connected Sets with 100 or more members = 23

2.5 Segmentation C-Code

```
#include <stdio.h>
#include "utils.h"

void error(char *name);

int main (int argc, char **argv) {
  FILE *fp;
  struct TIFF_img input_img,output_img;
  struct pixel s;
  int i, j, numcon;
  double T;
  int segLabel=1;

  if (argc != 4) error(argv[0]);

  /* 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 grayscale\n");
    exit(1);
  }

  sscanf(argv[2], "%lf", &T);

  unsigned int **seg = (unsigned int **)get_img(input_img.width,
                                                input_img.height,
                                                sizeof(unsigned int));

  for(i=0;i<input_img.height;i++){
    for(j=0;j<input_img.width;j++){
      seg[i][j]=0;
    }
  }

  for(i=0;i<input_img.height;i++){
```

```
    for(j=0;j<input_img.width;j++){
      if (seg[i][j]==0){
        s.m=i;
        s.n=j;
        ConnectedSet(s, T, input_img.mono, input_img.width, input_img.height,segLabel,
    seg, &numcon);
        if(numcon>100){
          segLabel++;
        }
        else
        {

        ConnectedSet(s, T, input_img.mono, input_img.width, input_img.height,0, seg, &
    numcon);
        }

      }
    }
  }

  get_TIFF(&output_img,input_img.height,input_img.width,'g');

  for (i = 0; i < input_img.height; i++) {
    for (j = 0; j < input_img.width; j++) {
        output_img.mono[i][j] = seg[i][j];
    }
    }

  printf("Number of ConnectedSets for Threshold %lf is %d \n",T,segLabel-1);
  /* open output image file */
  if ((fp = fopen(argv[3], "wb")) == NULL) {
    fprintf(stderr, "cannot open file output.tif\n");
    exit(1);
  }

  /* write output image */
  if(write_TIFF(fp, &output_img)) {
    fprintf(stderr, "error writing TIFF file %s\n", argv[5]);
    exit(1);
  }

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

  /* de-allocate space which was used for the images */
  free_TIFF(&(input_img));
  free_TIFF(&(output_img));
  free_img(*seg);

  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);
}
```

3 Utility Functions

3.1 "utils.h"

```
#include <stdio.h>
#include <math.h>

#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"

struct pixel {
  int m;
  int n;
};

struct px_linked_list {
  struct pixel px;
  struct px_linked_list *next_px;
};

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);

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

3.2 "utils.c"

```
#include "utils.h"

void CheckThreshold(struct pixel s,struct pixel p,double T,unsigned char **img,int
   width,int height,int *M,struct pixel c[4]){

      if (p.n >= 0 && p.n <width && p.m>=0 && p.m<height){
                if (abs(img[s.m][s.n] - img[p.m][p.n]) <= T) {
                c[*M].m = p.m;
                c[*M].n = p.n;
                (*M)++;
                }
        }

}

void ConnectedNeighbors(struct pixel s,double T,unsigned char **img,int width,int
   height,int *M,struct pixel c[4]) {
        *M = 0;

  struct pixel p1,p2,p3,p4;

```

```
  p1.m=s.m-1;
  p1.n=s.n;

  p2.m=s.m+1;
  p2.n=s.n;

  p3.m=s.m;
  p3.n=s.n+1;

  p4.m=s.m;
  p4.n=s.n-1;

  CheckThreshold(s,p1,T,img,width,height,M,c);
  CheckThreshold(s,p2,T,img,width,height,M,c);
  CheckThreshold(s,p3,T,img,width,height,M,c);
  CheckThreshold(s,p4,T,img,width,height,M,c);


}

void ConnectedSet(struct pixel s,double T,unsigned char **img,int width,int height,int
    ClassLabel,unsigned int **seg,int *NumConPixels) {
  struct px_linked_list *head_px, *next_px, *tmp_px;
  struct pixel c[4];
  int M;
  int i;

  (*NumConPixels) = 0;

  head_px = (struct px_linked_list *)malloc(sizeof(struct px_linked_list));
  head_px->px.m = s.m;
  head_px->px.n = s.n;
  head_px->next_px = NULL;
  next_px = head_px;

  /*Checking to end loop */

  while (head_px != NULL) {
    if (seg[head_px->px.m][head_px->px.n] != ClassLabel) {

      seg[head_px->px.m][head_px->px.n] = ClassLabel;
      (*NumConPixels)++;
      ConnectedNeighbors(head_px->px, T, img, width, height, &M, c);
      for (i = 0; i < M; i++) {
        if (seg[c[i].m][c[i].n] != ClassLabel) {
          tmp_px = (struct px_linked_list *)malloc(sizeof(struct px_linked_list));
          tmp_px->px.m = c[i].m;
          tmp_px->px.n = c[i].n;
          tmp_px->next_px = NULL;
          next_px->next_px = tmp_px;
          next_px = tmp_px;
        }
      }
    }
    tmp_px = head_px->next_px;
    free(head_px);
    head_px = tmp_px;
  }
}
```