Digital Signal Processing I
Purdue University
Spring 2021

Lab 3 Report

Jerry Wang

1 Area Fill

1. A Print out the image img22gd2.tif.

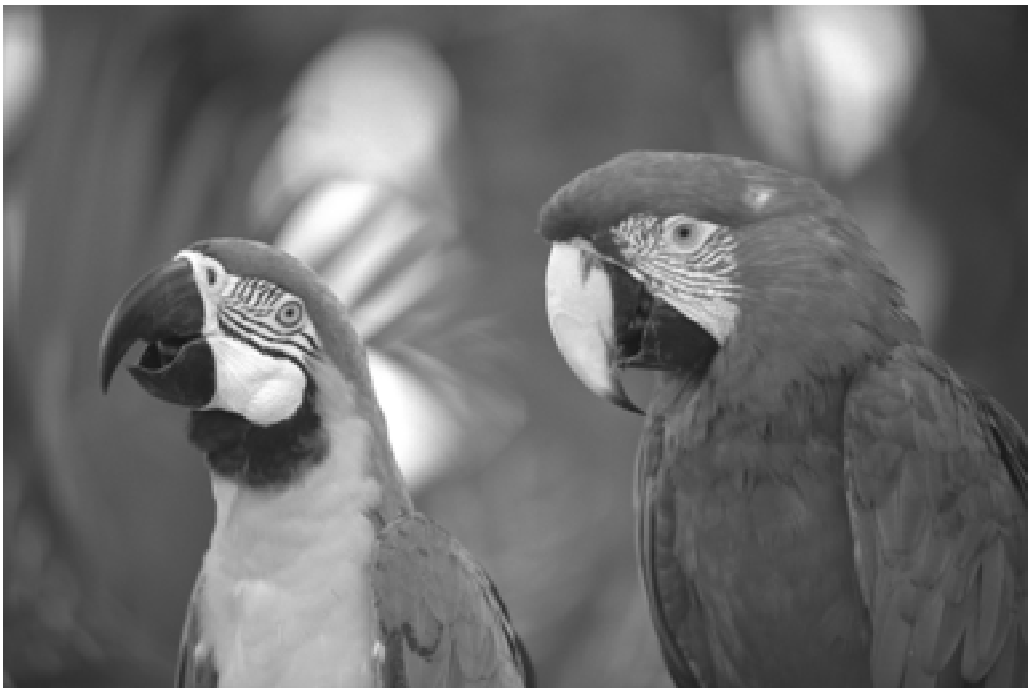

Figure 1: Print out of the original image

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

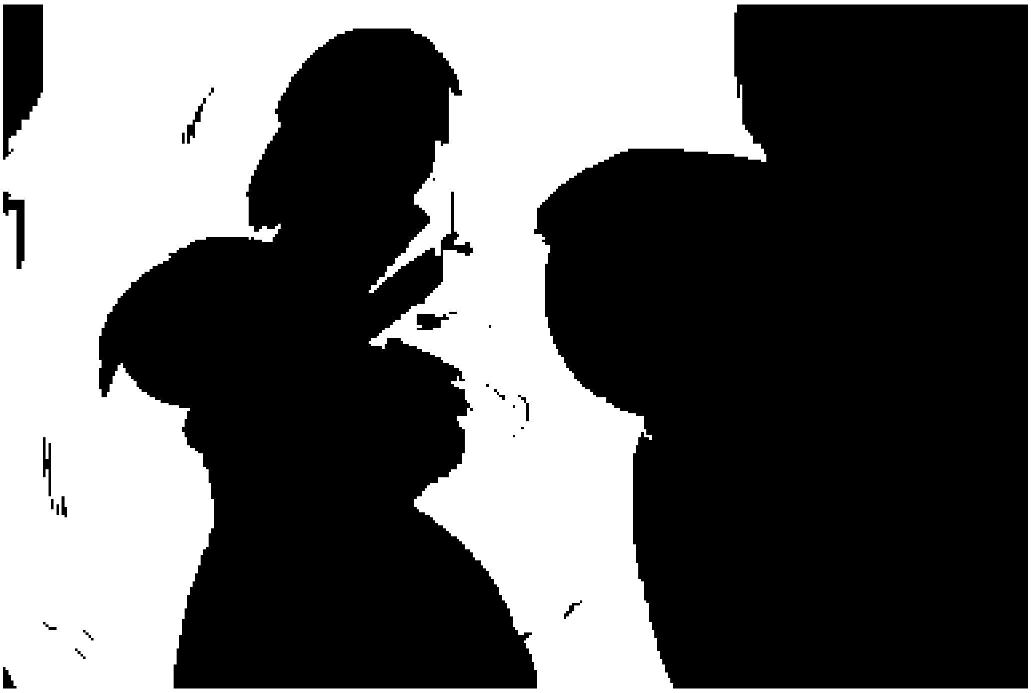

Figure 2: Connected Pixels from $(67, 45)$ (in white) when $T = 2$

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

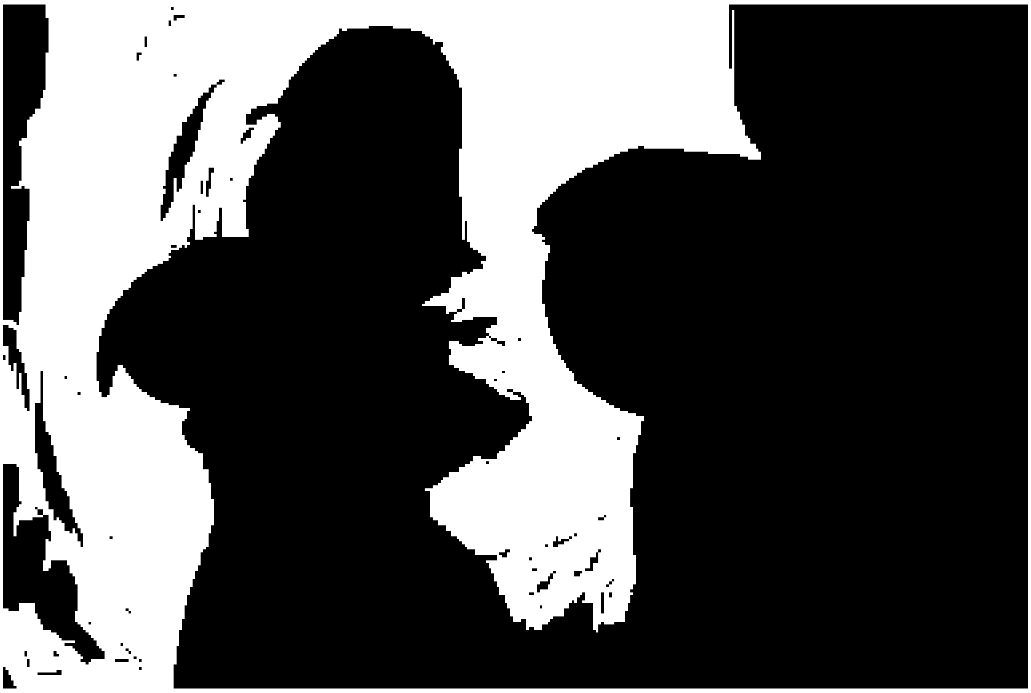

Figure 3: Connected Pixels from $(67, 45)$ (in white) when $T = 1$

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

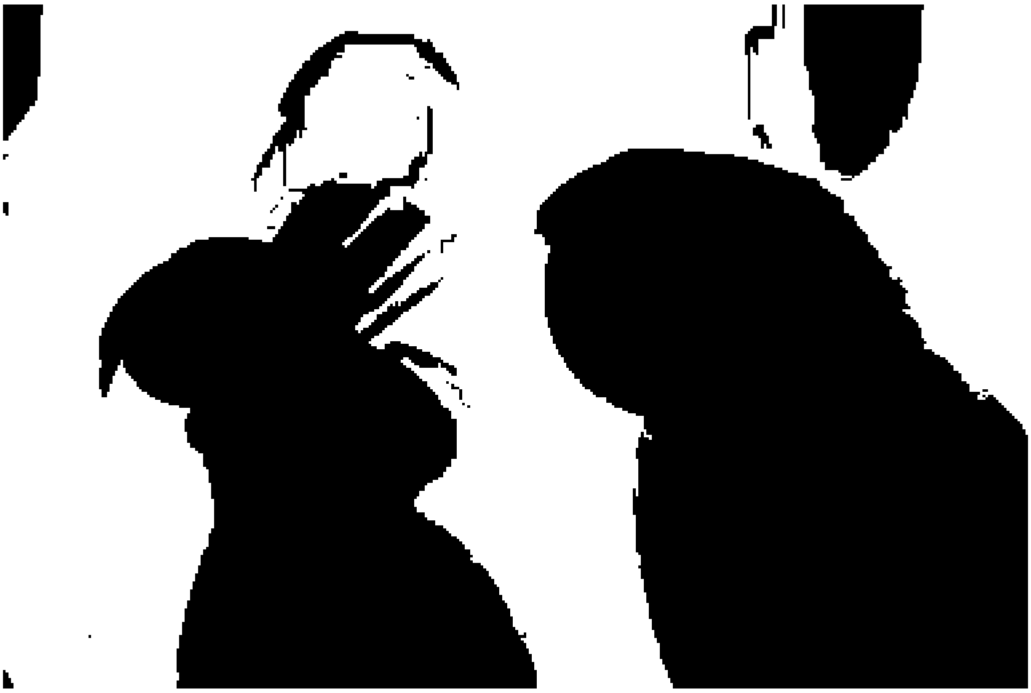

Figure 4: Connected Pixels from $(67, 45)$ (in white) when $T = 3$

5. A listing of the C code

Listing 1: Segmentation.h

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

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
);
```

Listing 2: Segmentation.c

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

void ConnectedNeighbors(struct pixel s, double T, unsigned char **img,
  int width, int height, int *M, struct pixel c[4])
{
  *M = 0;
  if (s.n - 1 >= 0) {
    if (abs(img[s.m][s.n] - img[s.m][s.n-1]) <= T) {
      c[*M].m = s.m;
      c[*M].n = s.n - 1;
      *M = *M + 1;
    }
  }

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

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

void ConnectedSet(struct pixel s, double T, unsigned char **img, int w
  int height, int ClassLabel, unsigned int **seg, int *NumConPixels)
{
  int M = 0;
  seg[s.m][s.n] = ClassLabel;
  struct pixel c[4];
  ConnectedNeighbors(s, T, img, width, height, &M, c);
  if (M == 0) {
    return;
  }
  else {
    for (int i = 0; i < M; i++) {
      struct pixel s_check = c[i];
      if (seg[s_check.m][s_check.n] != ClassLabel) {
        seg[s_check.m][s_check.n] = ClassLabel;
        *NumConPixels = *NumConPixels + 1;
        ConnectedSet(s_check, T, img, width, height, ClassLabel, seg, N
      }
    }
  }
}
```

Listing 3: areafill.c

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

int main(int argc, char **argv)
{
  FILE *fp;
  struct TIFF_img input_img, seg_img;
  unsigned int **seg;
  struct pixel s;
  double T;
  int label;
  int NumOfPixel = 1;

  if ( argc != 6 ) {
    fprintf( stderr, "Missing Argument\n");
    exit(1);
  }

  if ((fp = fopen(argv[1], "rb")) == NULL) {
    fprintf(stderr, "cannot open file %s\n", argv[1]);
    exit(1);
  }

  if (read_TIFF(fp, &input_img)) {
    fprintf(stderr, "error reading file %s\n", argv[1]);
    exit(1);
  }

  fclose(fp);

  s.m = atoi(argv[2]);
  s.n = atoi(argv[3]);
  T = atof(argv[4]);
  label = atoi(argv[5]);

  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, label, seg, &NumOfPixel);

  printf("Found␣%d␣pixels\n", NumOfPixel);

  get_TIFF(&seg_img, input_img.height, input_img.width, 'g');
  for (int i = 0; i < seg_img.height; i++) {
    for (int j = 0; j < seg_img.width; j++) {
      if (seg[i][j]) {
        seg_img.mono[i][j] = 255;
      }
    }
  }

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

  if (write_TIFF(fp, &seg_img)) {
    fprintf(stderr, "cannot␣write␣to␣file␣output.tif\n");
    exit(1);
  }

  fclose(fp);

  free_img((void**)seg);
  free_TIFF(&(input_img));
  free_TIFF(&(seg_img));
}
```

2 Image Segmentation

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

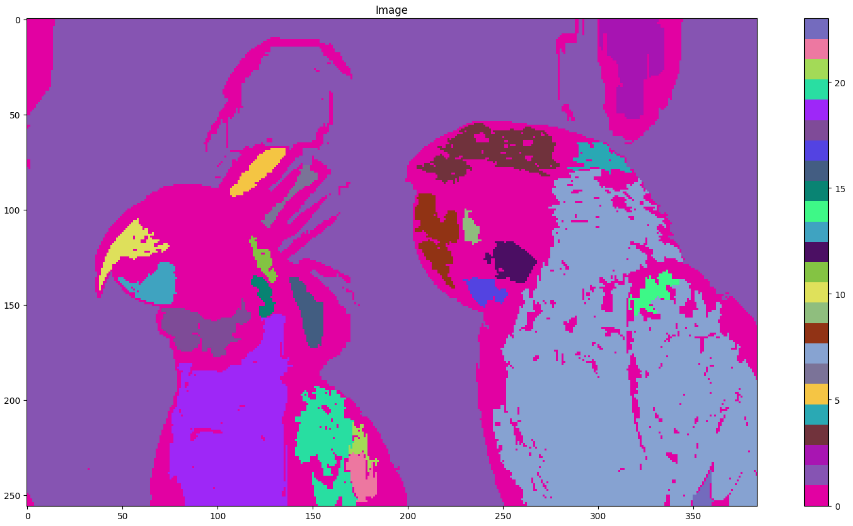

Figure 5: Randomly Colored Segmentation Image for $T = 1$

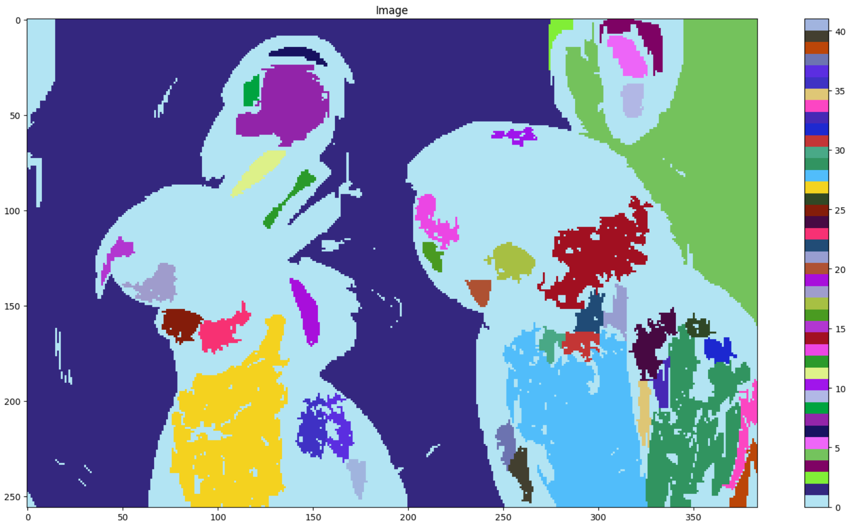

Figure 6: Randomly Colored Segmentation Image for $T = 2$

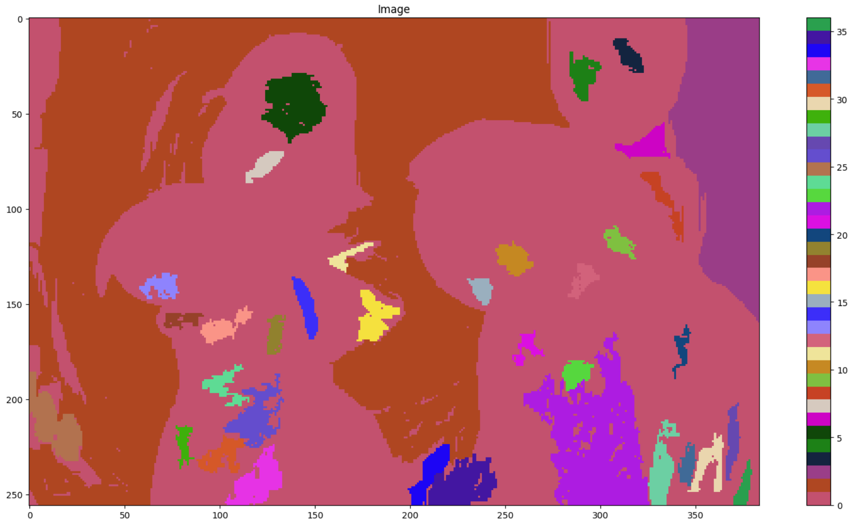

Figure 7: Randomly Colored Segmentation Image for $T = 3$

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

T	Number of Regions
1	37
2	42
3	24

3. A list of the C code.
   The code below uses the functions in *segmentation.h* and *segmentation.c* which are shown in Section 1.

Listing 4: Image Segmentation Code

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

void RemoveClassLabel(struct pixel s, int width, int height,
                      int ClassLabel, unsigned int **seg);

int main(int argc, char **argv)
{
  FILE *fp;
  struct TIFF_img input_img, seg_img;
  unsigned int **seg;
  struct pixel s;
  double T;
```

```
int NumofRegion = 1;
int NumofPixel = 1;

if ( argc != 3) {
  fprintf( stderr, "Missing Argument\n");
  exit(1);
}

if ((fp = fopen(argv[1], "rb")) == NULL) {
  fprintf(stderr, "cannot open file %s\n", argv[1]);
  exit(1);
}

if (read_TIFF(fp, &input_img)) {
  fprintf(stderr, "error reading file %s\n", argv[1]);
  exit(1);
}

fclose(fp);

T = atof(argv[2]);

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

for (int i = 0; i < input_img.height; i++) {
    for (int 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, NumofRegion++, seg,
                       &NumofPixel);
          if (NumofPixel <= 100) {
            RemoveClassLabel(s, input_img.width, input_img.height,
                             --NumofRegion, seg);
          }
          NumofPixel = 1;
        }
    }
}

printf("Found %d regions\n", NumofRegion);

get_TIFF(&seg_img, input_img.height, input_img.width, 'g');
for (int i = 0; i < seg_img.height; i++) {
  for (int j = 0; j < seg_img.width; j++) {
      // printf("%d, %d\n", i,j);
      seg_img.mono[i][j] = seg[i][j];
    }
  }

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

  if (write_TIFF(fp, &seg_img)) {
    fprintf(stderr, "cannot␣write␣to␣file␣output.tif\n");
    exit(1);
  }

  fclose(fp);

  free_img((void**)seg);
  free_TIFF(&(input_img));
  free_TIFF(&(seg_img));
}

void RemoveClassLabel(struct pixel s, int width, int height,
                        int ClassLabel, unsigned int **seg)
{
  // int M = 0;
  seg[s.m][s.n] = 0;
  struct pixel s_next;
  if (s.n - 1 >= 0) {
    if (seg[s.m][s.n-1] == ClassLabel) {
      seg[s.m][s.n-1] = 0;
      s_next.m = s.m;
      s_next.n = s.n - 1;
      RemoveClassLabel(s_next, width, height, ClassLabel, seg);
    }
  }

  if (s.m - 1 >= 0) {
    if (seg[s.m-1][s.n] == ClassLabel) {
      seg[s.m-1][s.n] = 0;
      s_next.m = s.m - 1;
      s_next.n = s.n;
      RemoveClassLabel(s_next, width, height, ClassLabel, seg);
    }
  }
  if (s.n + 1 < width) {
    if (seg[s.m][s.n+1] == ClassLabel) {
      seg[s.m][s.n+1] = 0;
      s_next.m = s.m;
      s_next.n = s.n + 1;
      RemoveClassLabel(s_next, width, height, ClassLabel, seg);
    }
  }

  if (s.m + 1 < height) {
    if (seg[s.m+1][s.n] == ClassLabel) {
      seg[s.m+1][s.n] = 0;
      s_next.m = s.m + 1;
      s_next.n = s.n;
      RemoveClassLabel(s_next, width, height, ClassLabel, seg);
    }
  }

}
```