ECE 6310 Introduction to Computer Vision

Lab 5 – Active contours

Prajval Vaskar

C20664702

October 27, 2020

The program must load a grayscale PPM image and a list of contour points. The contour points must be processed through the active contour algorithm using the options given below.

The program must output a copy of the image with the initial contour drawn on top of it, and a second image with the final contour drawn on top of it.

The program must also output a list of the final contour pixel coordinates. The program does not need to have a graphical user interface. It can run entirely from a command line and find the file names as either command line arguments or via string prompts to the user. In the output images, each contour point should be drawn using a "+" shape that is 7x7 pixels with a grayscale value of 0 so they can be clearly seen.

The file containing the list of contour points is a text file, with column and row coordinates separated by spaces, and each point ending with a carriage return.

An example is given at the course website. The active contour should use 2 internal energy terms and 1 external energy term. The internal energy terms are the square of the distances between points, and the square of the deviation from the average distance between points. The former term is detailed in the lecture notes. The latter term can be found by first calculating the average distance between all contour points, and then taking the square of the difference between that average and the distance between the current contour point and the next contour point.

It can be assumed that the contour encloses an area, so that the last contour point can be connected to the first contour point to calculate internal energy terms.

The external energy term is the square of the image gradient magnitude and should be calculated using convolution with a Sobel template.

The window around each contour point should be 7x7 pixels. Each energy term should be normalized by rescaling from min-max value to 0-1. Each energy term should be weighted equally. The active contour algorithm should run for 30 iterations.

Solution:

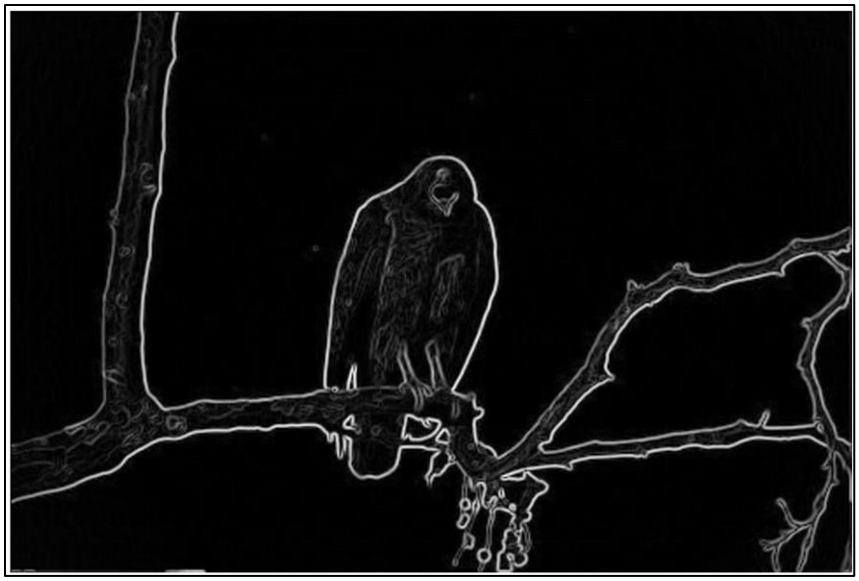

Figure 1 Result of Sobel edge gradient magnitude image

Figure 1 is the result after applying the Sobel filter in vertical as well as horizontal direction.

Sobel operators:

$$f1 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$f2 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Where, f1 and f2 computes the horizontal and vertical gradient.

Final gradient (G) can be calculated by

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * I$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * I$$

$$G = \sqrt{G_x^2 + G_y^2}$$

Where I is the original image and '*' represents the convolution.

Figure 2 The image with the initial contour points

For plotting contour points, hawk_init file is used to locate the contour points and loops are used to mark it with '+' sign.

Several results are shown below based in the weights of each energy term.

Figure 3 Result after 30 iteration with equal weights of 1

Weights used for above

1. Internal Energy – 1
2. Internal Energy 2 – 1
3. External Energy - 1

For calculating the total energy term, 3 different energy is used i.e. 2 internal energy and one external energy. External energy is calculated by taking inverse of Sobel gradient image which

was calculated above. The internal energies are calculated according the formulae given in the lab question.

Figure 4 Result after iteration 30.

Weights used for above result.

4. Internal Energy – 1
5. Internal Energy 2 – 1.5
6. External Energy - 1

Figure 5 Result after iteration 30.

Weights used for above result.

1. Internal Energy – 1
2. Internal Energy 2 – 2

3. External Energy - 1

List of Contour points after 30 iteration.

Contour point	Column, Row coordinates
1	272,115
2	274,121
3	276,129
4	277,140
5	278,151
6	273,168
7	269,179
8	266,187
9	261,199
10	258,206
11	254,213
12	250,221
13	238,235
14	229,231
15	224,240
16	222,247
17	212,265
18	201,266
19	194,259
20	194,250
21	193,244
22	186,244
23	185,241
24	179,239
25	178,233
26	179,209
27	182,187
28	183,174
29	184,162
30	187,150
31	191,138
32	194,126
33	199,114
34	210,105
35	220,100
36	229,93
37	235,87
38	244,83
39	253,84
40	262,93
41	265,101

42	267,109

Appendix

C code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>


#define SQR(x) ((x)*(x))
void Sobelfilter(unsigned char *image, float *sobel_image,int ROWS, int COLS)
{
  int r,c,r1,c1;
  float sum1,sum2;
  int sobel_h[3][3] = {-1,0,1,-2,0,2,-1,0,1};
  int sobel_v[3][3] = {-1,-2,-1,0,0,0,1,2,1};

  for (r=1; r<=ROWS-1; r++){
  for (c=1; c<=COLS-1; c++){
  sum1 = 0;
  sum2 = 0;
    for (r1=0; r1<=2; r1++){
      for (c1=0; c1<=2; c1++){
        sum1+=  image[(r+r1)*COLS+(c+c1)] * sobel_v[r1][c1];
        sum2+=  image[(r+r1)*COLS+(c+c1)] * sobel_h[r1][c1];
      }
    }
    sobel_image[r*COLS+c] = sqrt(SQR(sum1)+SQR(sum2)) ;
  }
}
return;
}

void normalization(float *sobel_image,unsigned char *normalized_image,int ROWS, i
nt COLS)
{
  float max_old = 0;
  float min_old = 0;
  float new_min = 0;
  float new_max =255;
  int r,c,i;

  for(i=0;i<ROWS*COLS;i++){
```

```
    if (sobel_image[i] > max_old){
      max_old = sobel_image[i];
    }
    if (sobel_image[i] < min_old){
      min_old = sobel_image[i];
    }
  }

  for (r=0; r<ROWS; r++){
  for (c=0;c<COLS;c++){
    normalized_image[r*COLS+c] = ((sobel_image[r*COLS+c]-min_old)*((new_max-
new_min)/(max_old-min_old)))+new_min;
  }
  }
}

float distance(int y1, int y2, int x1, int x2){
  float dis;
  dis = sqrt(SQR(y1-y2) + SQR(x1-x2));
  return dis;
}


FILE    *fpt, *file;
unsigned char *hawk_image,*hawk_init,*hawk_final,*hawk_sobel_final,*normalized;
char    header[320];
int   ROWS,COLS,BYTES;
int   con_y,con_x,r,c,i,r2,c2;
float *hawk_sobel_v;


int main()

{
  /* read input image */
  if ((fpt=fopen("hawk.ppm","rb")) == NULL)
    {
    printf("Unable to open hawk.ppm for reading\n");
    exit(0);
    }
  fscanf(fpt,"%s %d %d %d",header,&COLS,&ROWS,&BYTES);
  if (strcmp(header,"P5") != 0  ||  BYTES != 255)
    {
    printf("Not a greyscale 8-bit PPM image\n");
    exit(0);
```

```
    }

  /* Allocate dynamic memory*/
  hawk_image=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
  hawk_init=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
  hawk_sobel_v = (float *)calloc(ROWS*COLS,sizeof(float));
  hawk_final = (unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));



  header[0]=fgetc(fpt);           /* read white-
space character that separates header */
  fread(hawk_image,1,ROWS*COLS,fpt);
  fclose(fpt);

  for (i=0;i<ROWS*COLS;i++){
    hawk_init[i] = hawk_image[i];
  }

  /* Reading hawk_init file */
  fpt = fopen("hawk_init.txt","rb");

  /* Plotting initial contour points */
  while(fscanf(fpt, "%d %d\n", &con_x,&con_y) != EOF)
  {
    for (r=-3; r<=3; r++){
      hawk_init[(con_y+r)*COLS+con_x] = 0;
    }
    for (c=-3; c<=3; c++){
      hawk_init[(con_y)*COLS+con_x+c] = 0;
    }
  }

  fclose(fpt);

  Sobelfilter(hawk_image,hawk_sobel_v,ROWS,COLS);
  normalized =(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
  normalization(hawk_sobel_v, normalized,ROWS,COLS);

  int contour_x[42];
  int contour_y[42];

  fpt = fopen("hawk_init.txt", "r");
  int i=0;
```

```
  /* Storing initial contour points in array */
  while(fscanf(fpt, "%d %d", &con_x,&con_y) == 2) {
      contour_x[i] = con_x;
      contour_y[i] = con_y;
      i++;
  }
  fclose(file);

int iteration;
int window = 7;
float internal_energy1[window*window];
float internal_energy2[window*window];
float external_energy[window*window];
float total_energy[window*window];
float w1=1,w2=1,w3=1;

for (iteration = 0; iteration <=30; iteration++){
  float dist = 0;
  float avg_distance;

  for (i=0; i<=41; i++){
    if (i!=41){
      dist += distance(contour_y[i],contour_y[i+1], contour_x[i],contour_x[i+1]);
    }
    else{
      dist += distance(contour_y[i],contour_y[0], contour_x[i],contour_x[0]);
    }
  }
    avg_distance = dist/(42);

    for (i=0; i<=41;i++){

      int location = 0, location1 = 0, location2 = 0, location3 = 0, location4 =
0, location5 = 0, location6 = 0,location7=0,location8=0;
      int min_r,min_c;
      float new_min = 0;
      float new_max = 1;
      float max_energy1;
      float min_energy1;
      float max_energy2;
      float min_energy2;
      float max_energy_ext;
      float min_energy_ext;
      float max_total;
      float min_total;
```

```
      for (r = -(window/2); r<= (window/2); r++ ){
        for (c =-(window/2); c<=(window/2); c++ ){

          if (i!=41){
            internal_energy1[(r+(window/2))*window+(c+(window/2))] = w1*SQR(dista
nce(contour_y[i+1],(contour_y[i]+r),contour_x[i+1],(contour_x[i]+c)));
            internal_energy2[(r+(window/2))*window+(c+(window/2))] = w2*(SQR(avg_
distance - distance(contour_y[i+1],contour_y[i]+r,contour_x[i+1],contour_x[i]+c))
);
            external_energy[(r+(window/2))*window+(c+(window/2))] =  w3*(0 - norm
alized[(contour_y[i]+r)*COLS+(contour_x[i]+c)]);
            total_energy[(r+(window/2))*window+(c+(window/2))] = internal_energy1
[(r+(window/2))*window+(c+(window/2))]  + internal_energy2[(r+(window/2))*window+
(c+(window/2))] +external_energy[(r+(window/2))*window+(c+(window/2))];
          }
          else{
            internal_energy1[(r+(window/2))*window+(c+(window/2))]  = w1*SQR(dist
ance(contour_y[0],(contour_y[i]+r), contour_x[0],(contour_x[i]+c)));
            internal_energy2[(r+(window/2))*window+(c+(window/2))]  = w2*(SQR(avg
_distance - distance(contour_y[0],contour_y[i]+r,contour_x[0],contour_x[i]+c)));
            external_energy[(r+(window/2))*window+(c+(window/2))]  =  w3*(0 - nor
malized[(contour_y[i]+r)*COLS+(contour_x[i]+c)]);
            total_energy[(r+(window/2))*window+(c+(window/2))] = internal_energy1
[(r+(window/2))*window+(c+(window/2))]  + internal_energy2[(r+(window/2))*window+
(c+(window/2))] +external_energy[(r+(window/2))*window+(c+(window/2))];
          }
          if (internal_energy1[(r+(window/2))*window+(c+(window/2))] < internal_e
nergy1[location1]){
            min_energy1 = internal_energy1[(r+(window/2))*window+(c+(window/2))];
            location1 = (r+(window/2))*window+(c+(window/2));
          }
          if (internal_energy1[(r+(window/2))*window+(c+(window/2))] > internal_e
nergy1[location2]){
            max_energy1 = internal_energy1[(r+(window/2))*window+(c+(window/2))];
            location2 = (r+(window/2))*window+(c+(window/2));
          }
          if (internal_energy2[(r+(window/2))*window+(c+(window/2))] < internal_e
nergy2[location3]){
            min_energy2 = internal_energy2[(r+(window/2))*window+(c+(window/2))];
            location3 = (r+(window/2))*window+(c+(window/2));
          }
          if (internal_energy2[(r+(window/2))*window+(c+(window/2))] > internal_e
nergy2[location4]){
            max_energy2 = internal_energy2[(r+(window/2))*window+(c+(window/2))];
```

```
            location4 = (r+(window/2))*window+(c+(window/2));
          }
        if (external_energy[(r+(window/2))*window+(c+(window/2))] < external_ener
gy[location5]){
            min_energy_ext = external_energy[(r+(window/2))*window+(c+(window/2))
];
            location5 = (r+(window/2))*window+(c+(window/2));
          }
        if (external_energy[(r+(window/2))*window+(c+(window/2))] > external_ener
gy[location6]){
            max_energy_ext = external_energy[(r+(window/2))*window+(c+(window/2))
];
            location6 = (r+(window/2))*window+(c+(window/2));
          }

        if (total_energy[(r+(window/2))*window+(c+(window/2))] < total_energy[loc
ation7]){
          min_total = total_energy[(r+(window/2))*window+(c+(window/2))];
          location7 = (r+(window/2))*window+(c+(window/2));
          }
        if (total_energy[(r+(window/2))*window+(c+(window/2))] > total_energy[loc
ation8]){
          max_total = total_energy[(r+(window/2))*window+(c+(window/2))];
          location8 = (r+(window/2))*window+(c+(window/2));
          }
        }
      }
      float normalized_energy1[window*window], normalized_energy2[window*window],
 normalized_energy_ext[window*window],normalized_total[window*window];

      for (r = -(window/2); r<= (window/2); r++ ){
        for (c =-(window/2); c<=(window/2); c++ ){
          normalized_energy1[(r+(window/2))*window+(c+(window/2))] = ((internal_e
nergy1[(r+(window/2))*window+(c+(window/2))]-min_energy1)*((new_max-
new_min)/(max_energy1-min_energy1)))+new_min;
          normalized_energy2[(r+(window/2))*window+(c+(window/2))] = ((internal_e
nergy2[(r+(window/2))*window+(c+(window/2))]-min_energy2)*((new_max-
new_min)/(max_energy2-min_energy2)))+new_min;
          normalized_energy_ext[(r+(window/2))*window+(c+(window/2))] =((external
_energy[(r+(window/2))*window+(c+(window/2))]-min_energy_ext)*((new_max-
new_min)/(max_energy_ext-min_energy_ext)))+new_min;
          normalized_total[(r+(window/2))*window+(c+(window/2))] = ((total_energy
[(r+(window/2))*window+(c+(window/2))]-min_total)*((new_max-new_min)/(max_total-
min_total)))+new_min;
```

```
          if (normalized_total[(r+(window/2))*window+(c+(window/2))] < normalized
_total[location]){
            location = (r+(window/2))*window+(c+(window/2));
            min_r = r;
            min_c = c;
          }
        }
      }

      //printf("%d %d are inital\n", contour_y[i],contour_x[i],i);
      contour_y[i] = contour_y[i]+ min_r;
      contour_x[i] = contour_x[i]+ min_c;
    // printf("%d %d %d %d are final\n", contour_y[i],contour_x[i],iteration,i);
    }
    iteration++;
  }

  for (i=0;i<ROWS*COLS;i++){
    hawk_final[i] = hawk_image[i];
  }

    for (i=0;i<=41;i++){
      for (r=-3; r<=3; r++){
        hawk_final[(contour_y[i]+r)*COLS+(contour_x[i])] = 0;
      }
      for (c=-3; c<=3; c++){
        hawk_final[(contour_y[i])*COLS+(contour_x[i]+c)] = 0;
      }
    }

  fpt=fopen("final_contourpoints.txt","wb");
    if (fpt == NULL)
    {
    printf("Unable to open final_contourpoints.txt for writing\n");
    exit(0);
    }

  for (i=0; i<=41;i++){
    fprintf(fpt, "%d,%d\n",contour_x[i],contour_y[i]);
  }
  fclose(fpt);


  fpt=fopen("hawk_initial.ppm","wb");
```

```
  fprintf(fpt,"P5 %d %d 255\n",COLS,ROWS);
  fwrite(hawk_init,COLS*ROWS,1,fpt);
  fclose(fpt);

  fpt=fopen("hawk_original.ppm","wb");
  fprintf(fpt,"P5 %d %d 255\n",COLS,ROWS);
  fwrite(hawk_image,COLS*ROWS,1,fpt);
  fclose(fpt);

  fpt=fopen("hawk_sobel.ppm","wb");
  fprintf(fpt,"P5 %d %d 255\n",COLS,ROWS);
  fwrite(normalized,COLS*ROWS,1,fpt);
  fclose(fpt);

  fpt=fopen("hawk_final.ppm","wb");
  fprintf(fpt,"P5 %d %d 255\n",COLS,ROWS);
  fwrite(hawk_final,COLS*ROWS,1,fpt);
  fclose(fpt);
}
```