**ECE 6310**

**Introduction to Computer Vision**

**Fall 2022**

Lab 4

Active Contours

Harshal B. Varpe

**Introduction**

This lab is concerned with the implementation of active contouring of an image based on normalized internal and external energies. Initial contour points were given. The image used for the purpose of this lab is shown below:

A bird sitting on a tree branch

Description automatically generated with medium confidence  
Figure 1: Original Image

**Implementation / Methods**

Initially, the original image in the ppm format was loaded. This image was then operated on with a Sobel operator. The output of this operation was used to calculate external energy. The external energy is mostly concerned with edges.

As opposed to external energy, internal energy is concerned with contour points. For the internal energy calculation, the contour points were considered. Around each of the contour points, a window of nxn was considered. First internal energy is simply the distance to the next contour point from the current one. The second internal energy is the square of the difference between the average distance and the next contour point.

The total energy is then calculated by adding internal energies and subtracting external energy. It is also important to apply proper weights to each of the energies, as the output varies drastically.

A bird sitting on a branch

Description automatically generated with medium confidence  
Figure 2: Original Contour

A bird sitting on a tree branch

Description automatically generated with medium confidence

Figure 3: Final Contour for weights 100, 120, 0.001 for internal energy 1, internal energy 2 and external energy respectively

|  |  |  |
| --- | --- | --- |
| 1 | 259 | 77 |
| 2 | 265 | 91 |
| 3 | 268 | 100 |
| 4 | 274 | 106 |
| 5 | 279 | 123 |
| 6 | 281 | 131 |
| 7 | 282 | 139 |
| 8 | 280 | 154 |
| 9 | 278 | 169 |
| 10 | 272 | 179 |
| 11 | 269 | 190 |
| 12 | 266 | 204 |
| 13 | 259 | 213 |
| 14 | 251 | 224 |
| 15 | 236 | 230 |
| 16 | 229 | 236 |
| 17 | 221 | 245 |
| 18 | 221 | 257 |
| 19 | 214 | 266 |
| 20 | 204 | 268 |
| 21 | 193 | 268 |
| 22 | 183 | 260 |
| 23 | 176 | 249 |
| 24 | 172 | 237 |
| 25 | 169 | 229 |
| 26 | 168 | 219 |
| 27 | 168 | 211 |
| 28 | 169 | 201 |
| 29 | 169 | 189 |
| 30 | 170 | 180 |
| 31 | 172 | 165 |
| 32 | 175 | 153 |
| 33 | 177 | 142 |
| 34 | 181 | 127 |
| 35 | 186 | 115 |
| 36 | 192 | 104 |
| 37 | 205 | 92 |
| 38 | 216 | 83 |
| 39 | 222 | 77 |
| 40 | 233 | 71 |
| 41 | 242 | 70 |

Unfortunately I could not contour the hawk perfectly.

**Code**

/\*

  \*\* Harshal Varpe Lab 5

  \*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#define SQR(x) ((x)\*(x))

int main(int argc, char \*argv[])

{

FILE  \*fpt;

char header[320];

int px[100],py[100];

int i,total\_points,ROWS,COLS,BYTES;

unsigned char \*Input, \*init\_Contour;

int x,y,x1,y1,window;

float sob\_x, sob\_y, \*sobel,max,min,max\_prev,min\_prev;

unsigned char \*norm\_sob;

double  distance,avgdist;

if (argc != 3)

  {

  printf("Usage:  Lab5 [contour points] [window]\n");

  exit(0);

  }

window=atof(argv[2]);

if (window < 3  ||  window > 19)

  {

  printf("3 <= window <= 19\n");

  exit (0);

  }

// Open hawk.ppm

if ((fpt=fopen("hawk.ppm","rb")) == NULL)

  {

  printf("Unable to open hawk.ppm for reading\n");

  exit(0);

  }

//Read image header

i = fscanf(fpt,"%s %d %d %d",header,&COLS,&ROWS,&BYTES);

if(i != 4 || strcmp(header,"P5") != 0 || BYTES != 255){ printf("Not a grayscale image \n");

exit(0);}

// Memory allocation

Input = (unsigned char \*) calloc (ROWS\*COLS,sizeof(unsigned char));

init\_Contour = (unsigned char \*) calloc (ROWS\*COLS,sizeof(unsigned char));

fread(Input,1,COLS\*ROWS,fpt);

fclose(fpt);

// Create a copy of the input image

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

{

  init\_Contour[i] = Input[i];

}

//   open hawk\_contour

fpt = fopen("hawk\_init.txt","r");

if(fpt==NULL){

printf("Unable to open hawk\_init.txt for reading\n");

exit(0);

}

// total\_points = 0;

fscanf(fpt,"%d %d",&px[total\_points],&py[total\_points]);

while( fscanf(fpt,"%d %d",&px[total\_points],&py[total\_points]) != EOF ){

  total\_points++;

}

fclose(fpt);

// To draw initial contour of 7x7

for(i=0;i<total\_points;i++){

  for(x=-3;x<=3;x++){init\_Contour[(py[i]+x)\*COLS+px[i]] = 0;}

  for(y=-3;y<=3;y++){init\_Contour[(py[i])\*COLS+(px[i]+y)] = 0;}

}

// write out initial contour

fpt = fopen("Initial\_Contour.ppm","w");

if(fpt==NULL){printf("Unable to open initial contour");exit(0);}

fprintf(fpt,"P5 %d %d 255\n",COLS,ROWS);

fwrite(init\_Contour,COLS\*ROWS,1,fpt);

fclose(fpt);

// Let us use sobel filter for external energy term.

sobel = (float \*)calloc(ROWS\*COLS,sizeof(float));

norm\_sob = (unsigned char \*)calloc(ROWS\*COLS,sizeof(unsigned char));

int sob\_fx[9]= {1,0,-1,2,0,-2,1,0,-1};

int sob\_fy[9] = {1,2,1,0,0,0,-1,-2,-1};

printf("Good till here 1 \n");

for (x=1;x<ROWS-1;x++){

  for(y=1;y<COLS-1;y++){

    sob\_x = 0.0;

    sob\_y = 0.0;

    for(x1=-1; x1<=1; x1++){

      for(y1=-1; y1<=1; y1++){

        sob\_x += Input[(x+x1)\*COLS+(y+y1)] \* sob\_fx[(x1+1)\*3+(y1+1)];

        sob\_y += Input[(x+x1)\*COLS+(y+y1)] \* sob\_fy[(x1+1)\*3+(y1+1)];

      }

    }

    sobel[x\*COLS+y] = sqrt( SQR(sob\_x) + SQR(sob\_y) );

  }

}

printf("Good till here 2");

// Let is find the min and max intensities of the sobel filtered output

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

  if(max < sobel[i]){max = sobel[i];}

  if(min > sobel[i]){min = sobel[i];}

}

// printf("max: %f\tmin: %f",max,min);

// //Let us now normalize the min and max

for(x=0;x<ROWS;x++){

  for(y=0;y<COLS;y++){

    norm\_sob[x\*COLS+y] = 255 \* ((sobel[x\*COLS+y] - min) / (max - min)) ;

  }

}

fpt = fopen("sobel\_output.ppm","wb");

if(fpt==NULL){printf("Unable to open sobel output");exit(0);}

fprintf(fpt,"P5 %d %d 255\n",COLS,ROWS);

fwrite(norm\_sob,COLS\*ROWS,1,fpt);

fclose(fpt);

// code for contouring begins here

// We will have two internal energies and one external energy.

// Let us first calculate external energy.

int l,m,n; // counters

int c1,c2,c3; // for energy loops

int iter = 2;

double ext\_E[window\*window], int\_E1[window\*window],int\_E2[window\*window],total\_E[window\*window];

// printf("\n %d", window);

for(l=0;l<iter;l++){

  distance = 0;

  for(m=0;m<42;m++){

    c1 = 0 ;

    for (x= (-window/2); x <=window/2; x++){

        for (y = (-window/2); y <=window/2; y++){

          ext\_E[c1] = (double)SQR(norm\_sob[(py[m]+x)\*COLS+(px[m]+y)]);

          // printf("%f ", ext\_E[c1]);

          c1++;

        }

    } //  external energy

    // printf("External energy is %f \n",ext\_E);

    c2 = 0;

    for (x= (-window/2); x <=window/2; x++){

        for (y = (-window/2); y <=window/2; y++){

          if(m == 41){int\_E1[c2] =(double) SQR((py[m]+x)-(py[0]))  +  SQR((px[m]+y)-(px[0]));}

          else{int\_E1[c2] = (double)SQR((py[m]+x)-(py[0]))  +  SQR((px[m]+y)-(px[0]));}

          // printf("%f ", int\_E1[c2]);

          c2++;

        }

    } // internal energy 1

    distance = 0.0;

    if(m == 41){distance += sqrt(SQR(py[m]-py[0]) + SQR(px[m]-px[0]));}

    else{distance += sqrt(SQR(py[m]-py[m+1]) + SQR(px[m]-px[m+1]));}

    avgdist = (double)(distance/total\_points);

    // printf("\n%f\n",avgdist);

    c3 = 0;

    for (x= (-window/2); x <=window/2; x++){

        for (y = (-window/2); y <=window/2; y++){ int temp = 0;

          if(m == 41){

            temp = sqrt(SQR(py[m]+x-py[0])+SQR(px[m]+y-px[0]));

            int\_E2[c3] = (double)SQR(avgdist-temp);

            // printf("Here! \n");

            // printf("%d \n",temp);

            // printf("%d \n",avgdist);

            }

          else{temp = sqrt(SQR(py[m]+x-py[m+1])+SQR(px[m]+y-px[m+1]));

            int\_E2[c3] = (double)SQR(avgdist-temp);}

            // printf("%f ", int\_E2[c3]);

          c3++;

        }

      }// internal energy 2

    // All the energies need to be in normalised before they could be added.

    int min1;

    int max1;

    int min2 ;

    int max2 ;

    int min3 ;

    int max3 ;

    for (n = 0; n < (window\*window); n++){

        if(max1 < ext\_E[n]){max1 = ext\_E[n];}

        if(min1 > ext\_E[n]){min1 = ext\_E[n];}

    }

    // printf("%d %d \n",min1,max1);

    // for(x=0;x<50;x++){

    //   printf("%f  ",ext\_E[x]);

    // }

    for (n = 0; n < (window\*window); n++){

      ext\_E[n] = 1\*((ext\_E[n]-min1)/(max1-min1));

    }

    // for(x=0;x<50;x++){

    //   printf("External %f  ",ext\_E[x]);

    // }

    for (n = 0; n < (window\*window); n++){

        if(max2 < int\_E1[n]){max2 = int\_E1[n];}

        if(min2 > int\_E1[n]){min2 = int\_E1[n];}

    }

    // printf("%d %d \n",min2,max2);

    for (n = 0; n < (window\*window); n++){

      int\_E1[n] = 1\*((int\_E1[n]-min2)/(max2-min2));

    }

    for (n = 0; n < (window\*window); n++){

        if(max3 < int\_E2[n]){max3 = int\_E2[n];}

        if(min3 > int\_E2[n]){min3 = int\_E2[n];}

    }

    // printf("%d %d \n",min3,max3);

    for (n = 0; n < (window\*window); n++){

      int\_E2[n] = 1\*((int\_E2[n]-min3)/(max3-min3));

    }

    for (n = 0; n < window\*window;n++){

      total\_E[n] = (double) (100 \*(int\_E1[n]) + 120 \*(int\_E2[n]) - 0.001\*(ext\_E[n])) ;

      // printf("%f \n",total\_E[n]);

    }

    // for(x=0;x<50;x++){

    //   printf("%f  ",total\_E[x]);

    // }

    // We have to move the point to location with min energy. Hence -

    int min\_idx = 0;

    for (n = 0; n < window\*window;n++){

      if(min > total\_E[n]){min = total\_E[n]; min\_idx = n ;}

    }

    px[m] = px[m] + (min\_idx % window) - (window/2) ;

    py[m] = py[m] + (min\_idx / window) - (window/2) ;

  } // contour points loop

  printf("Im here. %d \n", l);

}//iter loop

// printf("%d",total\_points);

fpt = fopen("hawk\_finalpts.txt","wb");

for(n=0;n<total\_points;n++){

  printf("%d %d\n",px[n],py[n]);

  fprintf(fpt,"%d %d\n",px[n],py[n]);

  for(x=-3;x<=3;x++){Input[(py[n]+x)\*COLS+px[n]] = 0;}

  for(y=-3;y<=3;y++){Input[(py[n])\*COLS+(px[n]+y)] = 0;}

}

fclose(fpt);

printf("%d",total\_points);

fpt=fopen("final\_output.ppm","w");

fprintf(fpt,"P5 %d %d 255 \n",COLS,ROWS);

fwrite(Input,ROWS\*COLS,1,fpt);

fclose(fpt);

}