ECE 6310 Introduction to Computer Vision
Lab #5 – Active Contours

Christin Wilson
CUID: C21515896

November 5, 2019

1 Assignment

In this project each student must implement the active contour algorithm. 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.

You must write a brief report that includes the code. The report should show your result for the Sobel edge gradient magnitude image. The program must be tested on the image hawk.ppm and hawk_init.txt files at the course website. The image with the intial contour should be displayed, along with the image with the final contour for comparison. The image coordinates of the final contour should be provided in a list or table.

2 Images

Figure 1: Initial contour points

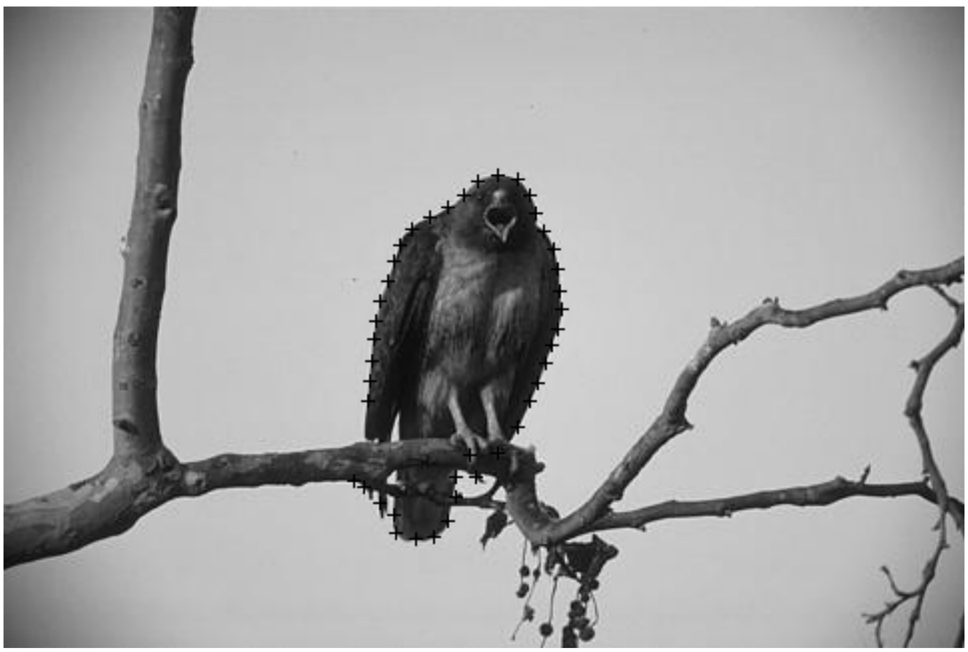

Figure 2: Final contour points

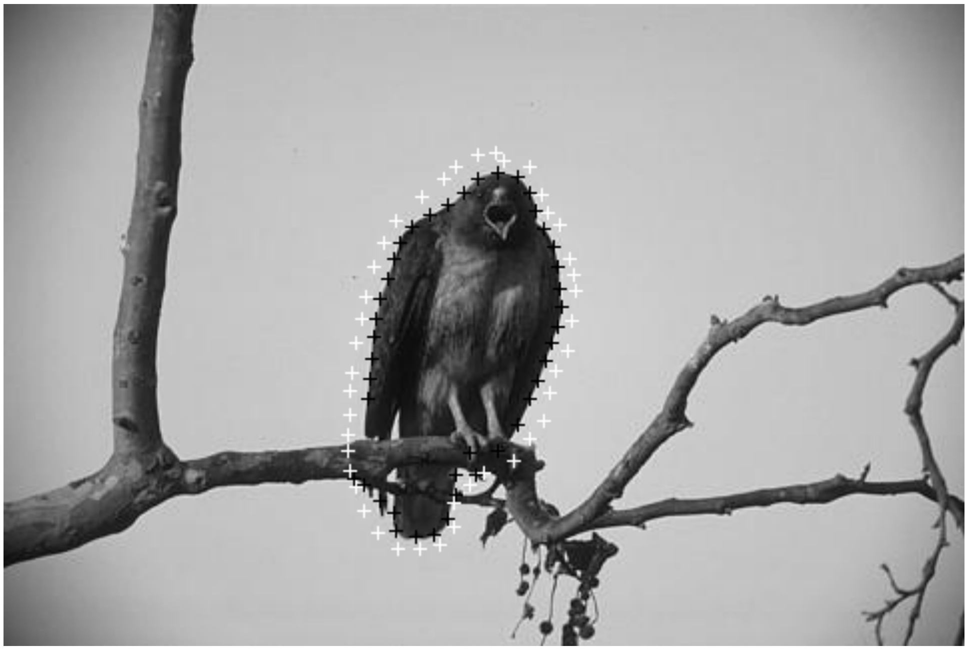

Figure 3: Initial (white) and Final (black) contour points

Figure 4: convolution with sobel filter and normalised image

The energy values used were the same as the one mentioned in the assignment and no other energy functions were used. The sobel filters used were

gx=

-1	0	1
-2	0	2
-1	0	1

and gy=

-1	-2	-1
0	0	0
1	2	1

3 Final Points

Row	Col
103	266
112	270
121	275
131	277
142	278
151	279
161	277
169	274
178	271
188	267
197	263
210	257
223	247
224	233
235	236
235	226
245	226
257	222
265	215
266	206
263	196

Row	Col
254	195
248	188
240	180
237	175
197	182
187	183
177	184
166	185
157	186
147	188
137	192
127	195
119	198
111	204
105	213
100	222
94	230
87	237
84	247
86	257
94	263

4 Source code

```
/*
** This program implements Optical character Recognition
**.
*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include<math.h>
int main()

 {
 FILE    *fpt1,*fpt2;
 unsigned char *image,*initial,*sobel_norm;
 float *ie1,*ie2,*ee,*sobel,*energy;
 int iter=0;
 unsigned char *final;
 char    header[320];
 char    header2[320];
 int   ROWS,COLS,BYTES;
 int   rows[42],cols[42],temp_rows[42],temp_cols[42],initr[42],initc[42];
 int   r1,c1,r2,c2,r,k;
 float max1=0,max2=0,max3=0,min1=0,min2=0,min3=0;
 float max=0,min=0;
 int row,col;
 FILE *fpt3;
 float avg=0;
 int gx[9]={-1,0,1,-2,0,2,-1,0,1};
 int gy[9]={-1,-2,-1,0,0,0,1,2,1};
 int xsum,ysum;
 int pos;
 /* read image */
 if ((fpt1=fopen("hawk.ppm","rb")) == NULL)
 {
   printf("Unable to open parenthood.ppm for reading\n");
   exit(0);
 }
 fscanf(fpt1,"%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);
 }

 image=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
 header[0]=fgetc(fpt1);  /* read white-space character that separates header */
 fread(image,1,COLS*ROWS,fpt1);
 fclose(fpt1);
 /* allocate memory for final version of image */
 final=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
 initial=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
 ie1=(float *)calloc(7*7,sizeof(float));
```

```
 ie2=(float *)calloc(7*7,sizeof(float));
 ee=(float *)calloc(7*7,sizeof(float));
 energy=(float *)calloc(7*7,sizeof(float));
 sobel=(float *)calloc(ROWS*COLS,sizeof(float));
 sobel_norm=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));

 int m=0,i,j=0,l;
 //Find sobel image
 for(r1=1;r1<ROWS-1;r1++)
 {
  for(c1=1;c1<COLS-1;c1++)
  {
   xsum=0;
   ysum=0;
   for(r2=-1;r2<=1;r2++)
   {
    for(c2=-1;c2<=1;c2++)
    {
      xsum+=image[(r1+r2)*COLS+(c1+c2)]*gx[(r2+1)*3+(c2+1)];
      ysum+=image[(r1+r2)*COLS+(c1+c2)]*gy[(r2+1)*3+(c2+1)];
    }
   }
   sobel[r1*COLS+c1]=sqrt(pow(xsum,2)+pow(ysum,2));
   if(sobel[r1*COLS+c1]>max1)
   {
    max1=sobel[r1*COLS+c1];
   }
   else if (sobel[r1*COLS+c1]<min1)
   {
    min1=sobel[r1*COLS+c1];
   }
  }
 }
k=0;

 //normalise sobel image
  while(k<ROWS*COLS)
   {
    sobel_norm[k]= (sobel[k]-min1)*255/(max1-min1);
    if(sobel_norm[k]>max)
    {
    max=sobel_norm[k];
    }
     else if(sobel_norm[k]<min)
     {
    min=sobel_norm[k];
    }
    k++;
   }
fpt1=fopen("sobel_norm.ppm","w");
fprintf(fpt1,"P5 %d %d 255\n",COLS,ROWS);
fwrite(sobel_norm,COLS*ROWS,1,fpt1);
fclose(fpt1);
```

```
    fpt3 = fopen("hawk_init.txt" , "r");
    r=fscanf(fpt3,"%d %d\n",&col,&row);
    for(i=0;i<ROWS*COLS;i++)
    {
      initial[i]=image[i];
    }
    while(m != EOF)
    {
    //printf("%c\n",alphabet );

      rows[j]=row;
      initr[j]=row;
      cols[j]=col;
      initc[j]=col;
        initial[row*COLS+col]=0;
        initial[(row+1)*COLS+col]=0;
        initial[(row+2)*COLS+col]=0;
        initial[(row+3)*COLS+col]=0;
        initial[(row-1)*COLS+col]=0;
        initial[(row-2)*COLS+col]=0;
        initial[(row-3)*COLS+col]=0;
        initial[row*COLS+(col+1)]=0;
        initial[row*COLS+(col+2)]=0;
        initial[row*COLS+(col+3)]=0;
        initial[row*COLS+(col-1)]=0;
        initial[row*COLS+(col-2)]=0;
        initial[row*COLS+(col-3)]=0;
        m=fscanf(fpt3,"%d %d\n",&col,&row);
        j++;

    }

        fpt1=fopen("initial_with_points.ppm","w");
        fprintf(fpt1,"P5 %d %d 255\n",COLS,ROWS);
        fwrite(initial,COLS*ROWS,1,fpt1);
        fclose(fpt1);
while(iter<30)
{
  //printf("iter:%d\n",iter);
  avg=0;
  for(i=0;i<42;i++)
  {
    if(i==0)
      {
        avg+=sqrt((rows[0]-rows[41])*(rows[0]-rows[41]) + (cols[0]-cols[41])*(cols[0]-cols[41]));
      }
    else
      {
        avg+=sqrt((rows[i]-rows[i-1])*(rows[i]-rows[i-1]) + (cols[i]-cols[i-1])*(cols[i]-cols[i-1]));
      }
  }
  avg=avg/42;
```

```
iter++;
for(i=0;i<42;i++)
{ max1=max2=max3=0;
  min1=min2=min3=1500;
  r1=rows[i];
  c1=cols[i];
  //Calculate Internal energy 1
  for(r2=-3;r2<=3;r2++)
    {
    for(c2=-3;c2<=3;c2++)
      {
      if(i<41)
        {
        ie1[(3+r2)*7+(3+c2)]=(rows[i+1]-(r1+r2))*(rows[i+1]-(r1+r2)) + (cols[i+1]-(c1+c2))*(cols[i+1]-(c1+c2));
      }
      else
      {
        ie1[(3+r2)*7+(3+c2)]=pow((rows[0]-(r1+r2)),2) + pow((cols[0]-(c1+c2)),2);
      }
      if(ie1[(3+r2)*7+(3+c2)]>max1)
          {
          max1=ie1[(3+r2)*7+(3+c2)];
          }
      if(ie1[(3+r2)*7+(3+c2)]<min1)
          {
          min1=ie1[(3+r2)*7+(3+c2)];
          }
    }
    }

  //Normalize Internal energy 1
  k=0;
  max=0;
  min=255;
  while(k<49)
    {
      ie1[k]= (ie1[k]-min1)/(max1-min1);
      if(ie1[k]>max)
      {
      max=ie1[k];
      }
       else if(ie1[k]<min)
       {
      min=ie1[k];
      }
      k++;
    }
  //Calculate Internal energy 2
  for(r2=-3;r2<=3;r2++)
    {
    for(c2=-3;c2<=3;c2++)
      {
        if(i<41)
```

```
        {
       ie2[(3+r2)*7+(3+c2)]=pow(sqrt((rows[i+1]-(r1+r2))*(rows[i+1]-(r1+r2)) + (cols[i+1]-(c1+c2))*(cols[i+1]-(c1+c2)))-
avg,2);
       }
       else
       {
       ie2[(3+r2)*7+(3+c2)]=pow(sqrt((rows[0]-(r1+r2))*(rows[0]-(r1+r2)) + (cols[0]-(c1+c2))*(cols[0]-(c1+c2)))-avg,2);
      }

      if(ie2[(3+r2)*7+(3+c2)]>max2)
        {
          max2=ie2[(3+r2)*7+(3+c2)];
        }
      if(ie2[(3+r2)*7+(3+c2)]<min2)
        {
          min2=ie2[(3+r2)*7+(3+c2)];
        }
    }
    }

   //Normalize Internal energy 2
   k=0;
   max=0;
   min=255;
   while(k<49)
    {
      ie2[k]= (ie2[k]-min2)/(max2-min2);
      if(ie2[k]>max)
      {
      max=ie2[k];
      }
       else if(ie2[k]<min)
       {
      min=ie2[k];
      }
      k++;
    }

   //Calculate external energy
   for(r2=-3;r2<=3;r2++)
    {
    for(c2=-3;c2<=3;c2++)
      {
       ee[(3+r2)*7+(3+c2)]=sobel[(r1+r2)*COLS+(c1+c2)];
       if(ee[(3+r2)*7+(3+c2)]>max3)
         {
           max3=ee[(3+r2)*7+(3+c2)];
         }
       if(ee[(3+r2)*7+(3+c2)]<min3)
         {
           min3=ee[(3+r2)*7+(3+c2)];
         }
```

```
    }
    }
    //Normalize External energy
   k=0;
   max=0;
   min=255;
   while(k<49)
    {
      ee[k]= (ee[k]-min3)/(max3-min3);
      ee[k]=1-ee[k];
      if(ee[k]>max)
      {
      max=ee[k];
      }
       else if(ee[k]<min)
       {
      min=ee[k];
      }
      k++;
    }

    //Calculate energy
    k=0;
    min=255;
    pos=0;
    int e=0;

    while(k<49)
    {
     energy[k]=ie1[k]+ie2[k]+ee[k];
     if(energy[k]<min)
     {
       min=energy[k];
       pos=k;
     }
     k++;
    }
   temp_rows[i]=rows[i]+(pos/7-3);
   temp_cols[i]=cols[i]+(pos%7-3);

  //printf("\n");
 }
 for(i=0;i<42;i++)
 {
  rows[i]=temp_rows[i];
  cols[i]=temp_cols[i];
 }
}
for(i=0;i<ROWS*COLS;i++)
   {
     final[i]=image[i];
   }
```

```
for(i=0;i<42;i++)
{
  row=rows[i];
    col=cols[i];
  printf("%d, %d \n",row,col);
        final[row*COLS+col]=0;
        final[(row+1)*COLS+col]=0;
        final[(row+2)*COLS+col]=0;
        final[(row+3)*COLS+col]=0;
        final[(row-1)*COLS+col]=0;
        final[(row-2)*COLS+col]=0;
        final[(row-3)*COLS+col]=0;
        final[row*COLS+(col+1)]=0;
        final[row*COLS+(col+2)]=0;
        final[row*COLS+(col+3)]=0;
        final[row*COLS+(col-1)]=0;
        final[row*COLS+(col-2)]=0;
        final[row*COLS+(col-3)]=0;
}
/* write out final image to see result */

fpt1=fopen("final.ppm","w");
fprintf(fpt1,"P5 %d %d 255\n",COLS,ROWS);
fwrite(final,COLS*ROWS,1,fpt1);
fclose(fpt1);

for(i=0;i<42;i++)
{
  row=initr[i];
    col=initc[i];
  printf("%d, %d \n",row,col);
        final[row*COLS+col]=255;
        final[(row+1)*COLS+col]=255;
        final[(row+2)*COLS+col]=255;
        final[(row+3)*COLS+col]=255;
        final[(row-1)*COLS+col]=255;
        final[(row-2)*COLS+col]=255;
        final[(row-3)*COLS+col]=255;
        final[row*COLS+(col+1)]=255;
        final[row*COLS+(col+2)]=255;
        final[row*COLS+(col+3)]=255;
        final[row*COLS+(col-1)]=255;
        final[row*COLS+(col-2)]=255;
        final[row*COLS+(col-3)]=255;
}

/* write out final image to see initial and final points */

fpt1=fopen("finalwithinitial.ppm","w");
fprintf(fpt1,"P5 %d %d 255\n",COLS,ROWS);
fwrite(final,COLS*ROWS,1,fpt1);
fclose(fpt1);
}
```