ECE 6310 – INTRODUCTION TO COMPUTER VISION

Lab 5 – Active Contour

C14109603

Rahil Modi
rmodi@g.clemson.edu

1. **Introduction**

In this lab we were expected to develop active contour on a PPM Image. We were given a hawk bird PPM image for the input and the initial contour points were given in a text file. The counter points around the image try to move towards each other and should stop at the edges of the hawk. Edges were to be determined with the help of Sobel Filter.

2. **Implementation**

We had to calculate three energies to see where the contour points need to move. Two internal energies and 1 external energy had to be calculated using a 7x7 window around every contour point. First internal energy is calculated based on the formula given in the notes which is the square of the distance between the current and the next contour points. Second internal energy is calculated based on the instructions given in the question, first the average distance between the all the contour points is calculated and square of the difference between the current contour point and the next contour point is calculated. For the external energy we need to have Sobel filter output which has the detected edges. Inverse of the Sobel filter is considered for edge to represent a lower value. Every pixel's sober filter output in the 7x7 window is squared and used for the external energy. Total energy is calculated by summing all the energies and then the minimum energy in the 7x7 window is determined and it is telling us in which direction the contour point must move. These minimum values are stored in an array and then these minimum values are added to the existing contour points and the new contour points are established. This is carried out for 30 iterations. Once 30 iterations are completed and the final contour points are displayed in the hawk.ppm with the help of the '+' symbol generated by making the middle row and column of the 7x7 window to black.

3. **Results**

Original Image:

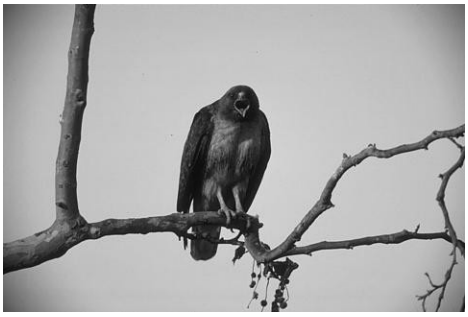

Figure 1 Original Image

Image with initial contour points:

Figure 2 Initial Image

Sobel Filter Output:

Figure 3 Sobel Filter Output

Active Contour Image:

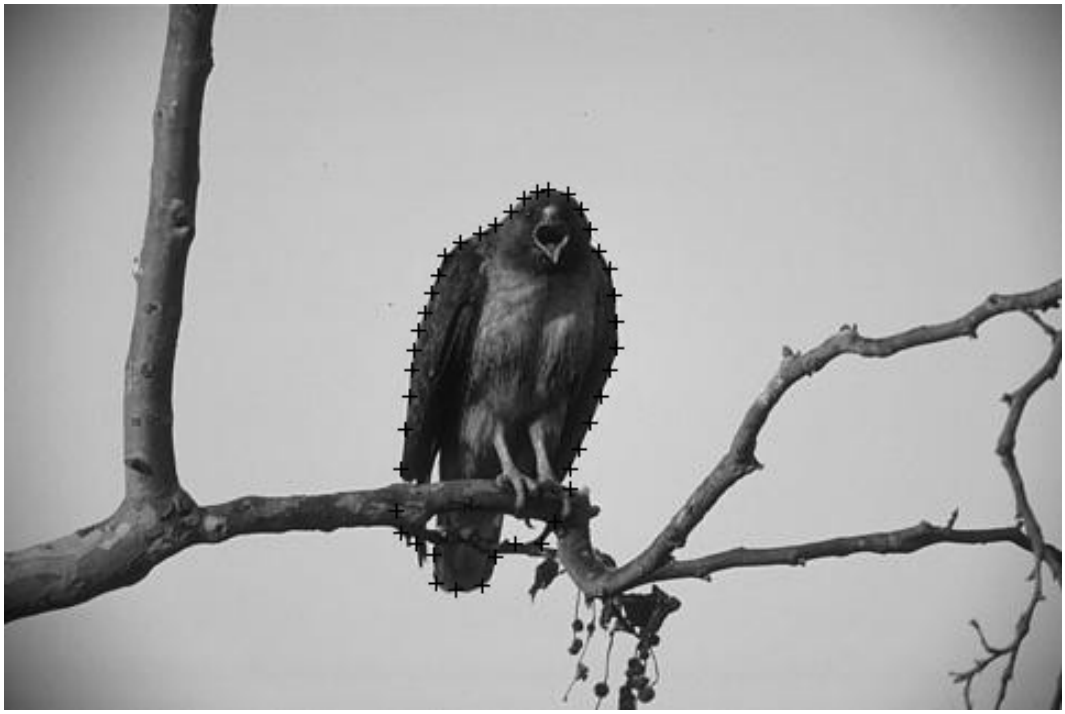

Figure 4 Active contour image

Active contour image (255 value for contour points as some points are not clear)

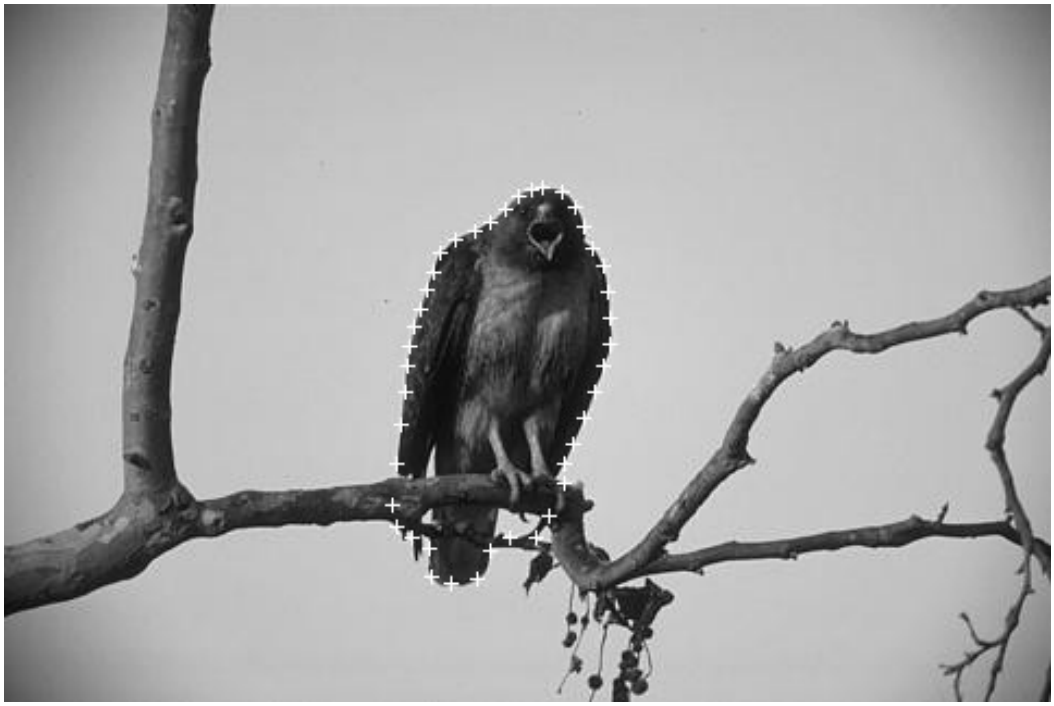

Figure 5 Active contour image (White contour point)

Result 1: (Multiplying Internal Energy 2 by 2)

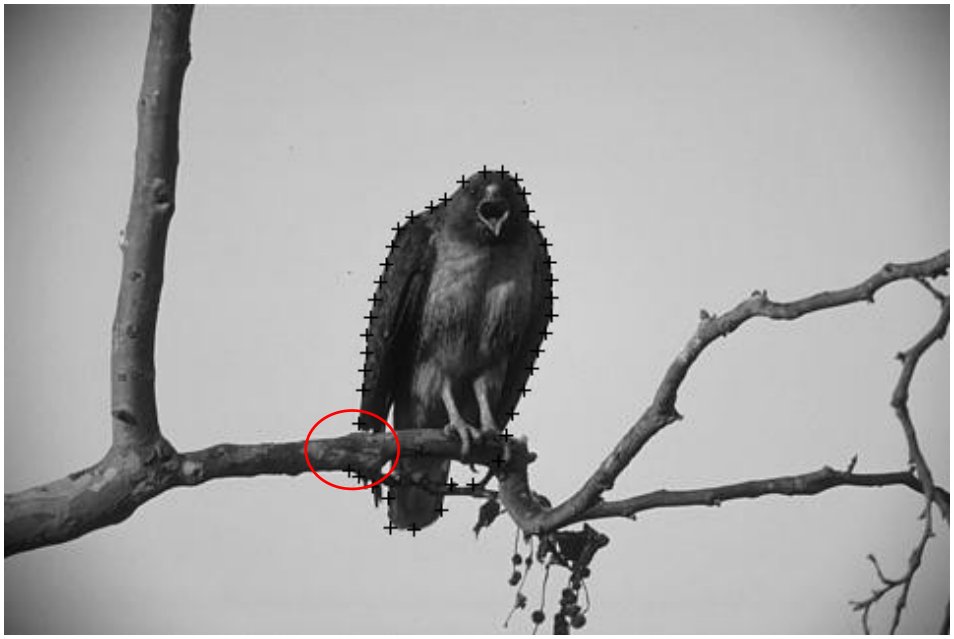

Figure 6

It can be observed that the contour points are not appearing on the tree branch causing break in the active contour.

Result 2: (Multiplying Internal Energy 1 by 2)

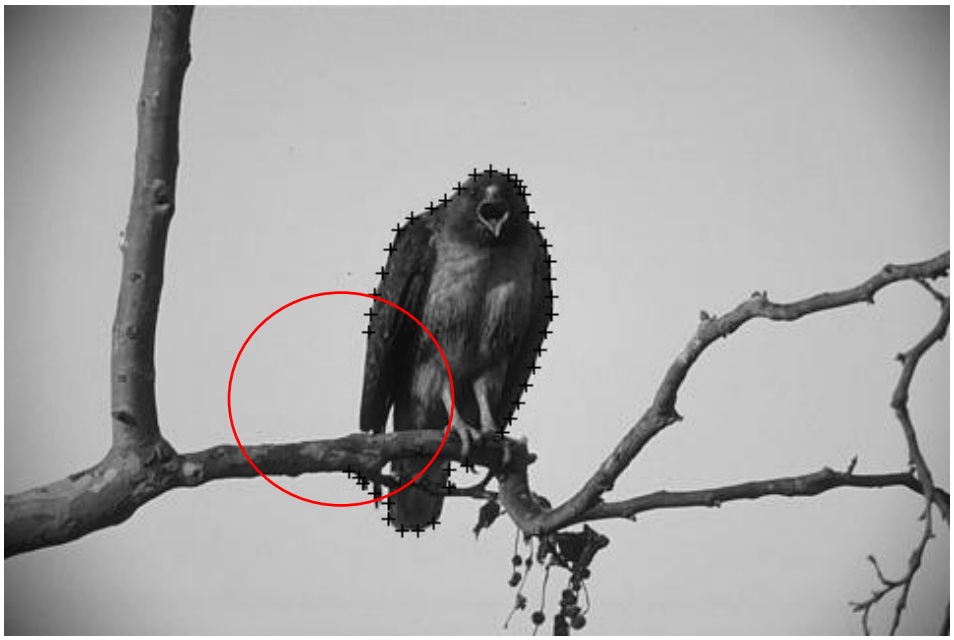

Figure 7

It can be observed that there is a very big break in the active contour but are a little evenly spaced.

Result 3: (Multiplying External Energy by 2)

Figure 8

It can observed in this image too that spacing is improper as well as there is a big break in the active contour points.

Contour points table:

Point Number	Row	Column
Contour Point 0:	112	270
Contour Point 1:	120	275
Contour Point 2:	132	277
Contour Point 3:	144	278
Contour Point 4:	156	278
Contour Point 5:	166	275
Contour Point 6:	178	271
Contour Point 7:	190	266
Contour Point 8:	202	261
Contour Point 9:	211	257
Contour Point 10:	220	257
Contour Point 11:	235	250
Contour Point 12:	245	244
Contour Point 13:	245	232
Contour Point 14:	251	223
Contour Point 15:	264	217
Contour Point 16:	266	205
Contour Point 17:	263	196

Contour Point 18:	250	195
Contour Point 19:	245	187
Contour Point 20:	240	180
Contour Point 21:	230	178
Contour Point 22:	211	180
Contour Point 23:	193	182
Contour Point 24:	178	184
Contour Point 25:	166	185
Contour Point 26:	157	186
Contour Point 27:	148	188
Contour Point 28:	140	191
Contour Point 29:	131	194
Contour Point 30:	123	197
Contour Point 31:	114	200
Contour Point 32:	108	207
Contour Point 33:	104	216
Contour Point 34:	100	223
Contour Point 35:	94	230
Contour Point 36:	88	236
Contour Point 37:	85	242
Contour Point 38:	84	247
Contour Point 39:	86	256
Contour Point 40:	93	262
Contour Point 41:	102	266

Conclusion:

In this lab we learned to implement active contour on an image, we understood the concept of Sobel Filter and energy calculations. How these different energies affect the direction of the contour points in which direction will they be moving.

Appendix:

Code is attached over here as per the question.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#define SQR(x) ((x)*(x))
/* Submitted by Rahil Modi C14109603 */

/*------------------------------------------------------------------------------
----------------------*/
```

```
void initial_image(unsigned char *image, int *contour_row, int *contour_col, int
ROWS, int COLS)
{
    int i, r, c;

    for (i = 0; i < 42; i++)
    {
        for (r = -3; r <= 3; r++)
        {
            image[(contour_row[i]+r)*COLS+contour_col[i]] = 0;
        }
        for (c = -3; c <= 3; c++)
        {
            image[contour_row[i]*COLS+(contour_col[i]+c)] = 0;
        }
    }
    return;
}

/*-------------------------------------------------------------------------------
------------------------*/

#define SQR(x) ((x)*(x))
void sobel_filter(unsigned char *image, float *sobel_image, int ROWS, int COLS)
{
    int         r, c, r2, c2;
    float       sum_gx, sum_gy;
    int         gx[9] = {-1, 0, 1, -2, 0, 2, -1, 0, 1};
    int         gy[9] = {-1, -2, -1, 0, 0, 0, 1, 2, 1};

    for (r = 3; r <ROWS-3; r++)
    {
        for (c = 3; c <COLS-3; c++)
        {
            sum_gx = 0;
            sum_gy = 0;
            for (r2 = -1; r2 <= 1; r2++)
            {
                for (c2 = -1; c2 <= 1; c2++)
                {
                    sum_gx += image[(r+r2)*COLS+(c+c2)] * gx[(r2+1)*3+(c2+1)];
                    sum_gy += image[(r+r2)*COLS+(c+c2)] * gy[(r2+1)*3+(c2+1)];
                }
            }
```

```
            sobel_image[r*COLS+c] = sqrt(SQR(sum_gx) + SQR(sum_gy));
        }
    }

    return;
}

/*-----------------------------------------------------------------------------
----------------------*/

void normalize(float *dummy_image, int ROWS, int COLS, int range)
{
  int i,j;
  float max, min;

  max  = dummy_image[0];
  min  = dummy_image[0];

  for (i = 0; i < ROWS * COLS; i++)
  {
    if (dummy_image[i] > max)
    {
      max = dummy_image[i];
    }
    if (dummy_image[i] < min)
    {
      min = dummy_image[i];
    }
  }

  for (i = 0; i < ROWS*COLS; i++)
  {
    dummy_image[i] = (dummy_image[i] - min)*range/(max-min);
  }
  return;
}

/*-----------------------------------------------------------------------------
----------------------*/

void invert_sobel(float *sobel_image, int ROWS, int COLS)
{
    int i;
     for (i = 0; i < ROWS*COLS; i++)
     {
```

```
         sobel_image[i] = 1 - sobel_image[i];
     }

     return;
}

/*-------------------------------------------------------------------------
-----------------------*/

void minimum(int *min_row, int *min_col, float *dummy_image)
{
    int i;
    float min = dummy_image[0];

    for (i = 0; i < 7*7; i++)
    {
        if (dummy_image[i] < min)
        {
            min = dummy_image[i];
            *min_row = (i/7) - 3;
            *min_col = (i%7) - 3;
        }
    }

}

/*-------------------------------------------------------------------------
------------------------*/

int distance(int x1, int x2, int y1, int y2)
{
    int euclidean_distance;

    euclidean_distance = sqrt(SQR(x2-x1)+SQR(y2-y1));

    return(euclidean_distance);
}

/*-------------------------------------------------------------------------
-------------------------*/

void internal_energy_1(float *total_int_energy, int *contour_row, int *contour_co
l, int i)
{
    int r, c;
```

```
    for (r = -3; r <= 3; r++)
    {
        for (c = -3; c <= 3; c++)
        {
            if (i != 41)
            {
                total_int_energy[(r+3)*7+(c+3)] = SQR(distance(contour_row[i+1],
contour_row[i]+r, contour_col[i+1], contour_col[i]+c));
            }
            else
            {
                total_int_energy[(r+3)*7+(c+3)] = SQR(distance(contour_row[0], co
ntour_row[i]+r, contour_col[0], contour_col[i]+c));
            }
        }
    }
    return;
}

/*-------------------------------------------------------------------------------
-----------------------------*/

void internal_energy_2(float *total_int2_energy, int *contour_row, int *contour_c
ol, int avg, int i)
{
    int r, c;

    for (r = -3; r <= 3; r++)
    {
        for (c = -3; c <= 3; c++)
        {
            if (i != 41)
            {
                total_int2_energy[(r+3)*7+(c+3)] = SQR(avg - distance(contour_row
[i+1], contour_row[i]+r, contour_col[i+1], contour_col[i]+c));
            }
            else
            {
                total_int2_energy[(r+3)*7+(c+3)] = SQR(avg - distance(contour_row
[0], contour_row[i]+r, contour_col[0], contour_col[i]+c));
            }
        }
    }
    return;
```

```
}

/*----------------------------------------------------------------------------
--------------------------------------*/

void external_energy(float *total_ext_energy, float *sobel_image, int *contour_ro
w, int *contour_col, int COLS, int i)
{
    int r, c;

    for (r = -3; r <= 3; r++)
    {
        for (c = -3; c <= 3; c++)
        {
            total_ext_energy[(r+3)*7+(c+3)] = SQR(sobel_image[(contour_row[i]+r)*
COLS+(contour_col[i]+c)]);
        }
    }
    return;
}

/*----------------------------------------------------------------------------
--------------------------------------*/

void active_contour(float *sobel_image, int *contour_row, int *contour_col, int R
OWS, int COLS)
{
    int i, j, k, l;

    float   *internal_1;
    float   *internal_2;
    float   *external;
    float   *total_energy;
    int     *dummy_row;
    int     *dummy_col;
    int     min_row;
    int     min_col;
    float   average_distance;

    internal_1   = (float *)calloc(7*7, sizeof(float));
    internal_2   = (float *)calloc(7*7, sizeof(float));
    external     = (float *)calloc(7*7, sizeof(float));
    total_energy = (float *)calloc(7*7, sizeof(float));
    dummy_row    = (int *)calloc(42, sizeof(int));
    dummy_col    = (int *)calloc(42, sizeof(int));
```

```
    normalize(sobel_image,  ROWS, COLS, 1);
    invert_sobel(sobel_image, ROWS, COLS);

    for (i = 0; i < 31; i++)
    {
        average_distance = 0;
        for (j = 0; j < 41; j++)
        {
            average_distance += distance(contour_row[j+1], contour_row[j], contou
r_col[j+1], contour_col[j]);
        }
        average_distance += distance(contour_row[0], contour_row[j], contour_col[
0], contour_col[j]);
        average_distance = average_distance/42;

        for (j = 0; j < 42; j++)
        {
            internal_energy_1(internal_1, contour_row, contour_col, j);
            internal_energy_2(internal_2, contour_row, contour_col, average_dista
nce, j);
            external_energy(external, sobel_image, contour_row, contour_col, COLS
, j);
            normalize(internal_1, 7, 7, 1);
            normalize(internal_2, 7, 7, 1);

            for (k = 0; k < 7*7; k++)
            {
                total_energy[k] =  internal_1[k] + internal_2[k] + external[k];
            }

            minimum(&min_row, &min_col, total_energy);

            dummy_row[j] = contour_row[j]+min_row;
            dummy_col[j] = contour_col[j]+min_col;
        }
        for (j = 0; j < 42; j++)
        {
            contour_row[j] = dummy_row[j];
            contour_col[j] = dummy_col[j];
        }
    }
    return;
}
```

```
/*-------------------------------------------------------------------------------
-------------------------------------*/

int main()
{
    FILE            *fpt;
    unsigned char   *image;
    unsigned char   *sobel_final;
    unsigned char   *final;
    float           *sobel_image;
    int             *contour_row;
    int             *contour_col;
    char            header[320];
    int             ROWS,COLS,BYTES;
    int             i;

    /* read image */
    if ((fpt=fopen("D:/Computer_Vision/Lab5/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);
    }
    image=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
    header[0]=fgetc(fpt);   /* read white-
space character that separates header */
    fread(image,1,COLS*ROWS,fpt);
    fclose(fpt);

    if ((fpt=fopen("D:/Computer_Vision/Lab5/hawk_init.txt", "rb")) == NULL)
    {
        printf("Unable to open intial contours file\n");
        exit(0);
    }

    contour_row = (int *)calloc(42, sizeof(int));
    contour_col = (int *)calloc(42, sizeof(int));

    i = 0;
    while(!feof(fpt))
```

```
    {
        fscanf(fpt, "%d %d\n", &contour_col[i], &contour_row[i]);
        i++;
    }

    final = (unsigned char *)calloc(ROWS*COLS, sizeof(unsigned char));
    for (i = 0; i < ROWS*COLS; i++)
        final[i] = image[i];

    initial_image(final, contour_row, contour_col, ROWS, COLS);
    fpt = fopen("D:/Computer_Vision/Lab5/initial.ppm", "wb");
    fprintf(fpt, "P5 %d %d 255\n", COLS, ROWS);
    fwrite(final, COLS*ROWS, 1, fpt);
    fclose(fpt);

    sobel_image = (float *)calloc(ROWS*COLS, sizeof(float));
    sobel_final = (unsigned char *)calloc(ROWS*COLS, sizeof(unsigned char));

    sobel_filter(image, sobel_image, ROWS, COLS);
    normalize(sobel_image, ROWS, COLS, 255);
    for (i = 0; i < ROWS*COLS; i++)
    {
    sobel_final[i] = sobel_image[i];
    }
    fpt = fopen("D:/Computer_Vision/Lab5/sobel.ppm", "wb");
    fprintf(fpt, "P5 %d %d 255\n", COLS, ROWS);
    fwrite(sobel_final, COLS*ROWS, 1, fpt);
    fclose(fpt);

    active_contour(sobel_image, contour_row, contour_col, ROWS, COLS);
    final = (unsigned char *)calloc(ROWS*COLS, sizeof(unsigned char));
    for (i = 0; i < ROWS*COLS; i++)
        final[i] = image[i];

    initial_image(final, contour_row, contour_col, ROWS, COLS);

    fpt = fopen("D:/Computer_Vision/Lab5/final.ppm", "wb");
    fprintf(fpt, "P5 %d %d 255\n", COLS, ROWS);
    fwrite(final, COLS*ROWS, 1, fpt);
    fclose(fpt);

    for (i = 0; i < 42; i++)
    {
        printf("Contour Point %d: %d %d\n", i, contour_row[i], contour_col[i]);
    }
```

```
    return(0);

}
```