**William Benton**

**Lab 3: Optical Character Recognition with Thinning**

**11/04/2019**

**Background:**

The purpose of this lab was to perform active contouring on an image. The initial contour points were given and an image of a hawk on a tree branch was used. The contour points roughly surrounded the hawk and the active contouring was intended to surround the outline of the hawk. These contouring points move towards each other as well as toward the edge of the hawk. The edge was detected using a Sobel filter on the input image and the contour points were moved together based on distance.

**Implementation:**

To decide where to move to, several “energies” were calculated. Around each contour point, a 7x7 window was taken. Using this 7x7 window, 2 internal energies were calculated, and an external energy was calculated. The internal energies are about the formation of the points and the external energy is about the edges of the image. The first internal energy was calculated by simply calculating the distance to the next contour point. The second internal energy does the same thing, except it subtracts the average distance between points from the distance to the next point and squares the result. This leaves the 7x7 window in a state where each window has a minimum point. This minimum point represents the closest point to the next point. The external energy uses the result of the Sobel filter. In order to get an edge to represent a lower value, the inverse of the Sobel filter is used. The Sobel filter result of each point in the 7x7 window is squared and that is what is used for the external energy. The resulting 7x7 windows of energy values are used to calculate the total energy around that contour point. The minimum value of the total 7x7 array is used to set where the next set of contour points are supposed to move to. An array temporarily holds that value until calculations have been performed on all contour points. After all contour points have been used for calculations, the contour points are set to their new positions. Then the calculations are performed again. This happens 30 times. After this is done, plus signs are drawn around the final contour points to show where they have moved to. The results below will show the final contour points listed out, the initial image with the initial contour points, the Sobel filter output, and the final output image.

**Results:**

|  |  |
| --- | --- |
| Contour Point 0: 166 275  Contour Point 1: 169 274  Contour Point 2: 180 270  Contour Point 3: 191 266  Contour Point 4: 203 261  Contour Point 5: 218 254  Contour Point 6: 220 257  Contour Point 7: 232 251  Contour Point 8: 236 239  Contour Point 9: 242 228  Contour Point 10: 251 223  Contour Point 11: 264 217  Contour Point 12: 267 208  Contour Point 13: 263 196  Contour Point 14: 254 195  Contour Point 15: 245 187  Contour Point 16: 238 177  Contour Point 17: 237 175  Contour Point 18: 234 174  Contour Point 19: 222 176  Contour Point 20: 211 180 | Contour Point 21: 199 181  Contour Point 22: 187 183  Contour Point 23: 175 184  Contour Point 24: 163 185  Contour Point 25: 151 187  Contour Point 26: 140 191  Contour Point 27: 129 195  Contour Point 28: 117 199  Contour Point 29: 106 211  Contour Point 30: 99 224  Contour Point 31: 87 237  Contour Point 32: 84 248  Contour Point 33: 87 258  Contour Point 34: 96 264  Contour Point 35: 104 266  Contour Point 36: 115 272  Contour Point 37: 126 276  Contour Point 38: 137 278  Contour Point 39: 141 278  Contour Point 40: 152 279  Contour Point 41: 163 276 |

Final Contour Points

A bird perched on a tree branch

Description automatically generated

Figure 1: Initial Image with Initial Contour



Figure 2: Output of Sobel Filter

A bird perched on a tree branch

Description automatically generated

Figure 3: Final Result

**Code:**

ActiveContours.h

void ActiveContour(float \*sobel\_image, int \*contour\_row, int \*contour\_col, int ROWS, int COLS);

void InvertSobel(float \*sobel\_image, int ROWS, int COLS);

void Min(int \*min\_row, int \*min\_col, float \*temp);

int Distance(int x1, int x2, int y1, int y2);

void InternalEnergy1(float \*energy\_array, int \*contour\_row, int \*contour\_col, int i);

void InternalEnergy2(float \*energy\_array, int \*contour\_row, int \*contour\_col, int avg, int i);

void ExternalEnergy(float \*energy\_array, float \*sobel\_image, int \*contour\_row, int \*contour\_col, int COLS, int i);

void SobelFilter(unsigned char \*image, float \*sobel\_image, int ROWS, int COLS);

void Normalize(float \*temp, int ROWS, int COLS, int range);

void MakeFinal(unsigned char \*image, int \*contour\_row, int \*contour\_col, int ROWS, int COLS);

void Float2Unsigned(float \*input, unsigned char \*output, int ROWS, int COLS);

const int F1[9] = {-1, 0, 1, -2, 0, 2, -1, 0, 1};

const int F2[9] = {-1, -2, -1, 0, 0, 0, 1, 2, 1};

ActiveContours.c

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

#include "ActiveContour.h"

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

int main()

{

FILE \*fpt;

unsigned char \*image;

unsigned char \*sobel\_char;

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("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);

// Read beginning contour points

if ((fpt=fopen("intialcontour.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];

MakeFinal(final, contour\_row, contour\_col, ROWS, COLS);

fpt = fopen("init.ppm", "w");

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));

SobelFilter(image, sobel\_image, ROWS, COLS);

Normalize(sobel\_image, ROWS, COLS, 255);

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

Float2Unsigned(sobel\_image, sobel\_char, ROWS, COLS);

fpt = fopen("sobelout.ppm", "w");

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

fwrite(sobel\_char, COLS\*ROWS, 1, fpt);

fclose(fpt);

ActiveContour(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];

MakeFinal(final, contour\_row, contour\_col, ROWS, COLS);

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

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);

}

void MakeFinal(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]] = 255;

}

for (c = -3; c <= 3; c++)

{

image[contour\_row[i]\*COLS+(contour\_col[i]+c)] = 255;

}

}

return;

}

void ActiveContour(float \*sobel\_image, int \*contour\_row, int \*contour\_col, int ROWS, int COLS)

{

int i, j, k;

float \*internal1;

float \*internal2;

float \*external;

float \*TotalEnergy;

int \*temp\_row;

int \*temp\_col;

int min\_row;

int min\_col;

float average\_distance;

internal1 = (float \*)calloc(7\*7, sizeof(float));

internal2 = (float \*)calloc(7\*7, sizeof(float));

external = (float \*)calloc(7\*7, sizeof(float));

TotalEnergy = (float \*)calloc(7\*7, sizeof(float));

temp\_row = (int \*)calloc(42, sizeof(int));

temp\_col = (int \*)calloc(42, sizeof(int));

Normalize(sobel\_image, ROWS, COLS, 1);

InvertSobel(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], contour\_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++)

{

InternalEnergy1(internal1, contour\_row, contour\_col, j);

InternalEnergy2(internal2, contour\_row, contour\_col, average\_distance, j);

ExternalEnergy(external, sobel\_image, contour\_row, contour\_col, COLS, j);

Normalize(internal1, 7, 7, 1);

Normalize(internal2, 7, 7, 1);

for (k = 0; k < 7\*7; k++)

{

TotalEnergy[k] = 2\*internal1[k] + internal2[k] + external[k];

}

Min(&min\_row, &min\_col, TotalEnergy);

temp\_row[j] = contour\_row[j]+min\_row;

temp\_col[j] = contour\_col[j]+min\_col;

}

for (j = 0; j < 42; j++)

{

contour\_row[j] = temp\_row[j];

contour\_col[j] = temp\_col[j];

}

}

return;

}

void InvertSobel(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 Min(int \*min\_row, int \*min\_col, float \*temp)

{

int i;

float min = 20000.00;

for (i = 0; i < 7\*7; i++)

{

if (temp[i] < min)

{

min = temp[i];

\*min\_row = (i/7) - 3;

\*min\_col = (i%7) - 3;

}

}

}

int Distance(int x1, int x2, int y1, int y2)

{

int ret\_val;

ret\_val = sqrt(SQR(x2-x1)+SQR(y2-y1));

return(ret\_val);

}

void InternalEnergy1(float \*energy\_array, int \*contour\_row, int \*contour\_col, int i)

{

int r, c;

for (r = -3; r <= 3; r++)

{

for (c = -3; c <= 3; c++)

{

if (i != 41)

{

energy\_array[(r+3)\*7+(c+3)] = Distance(contour\_row[i+1], contour\_row[i]+r, contour\_col[i+1], contour\_col[i]+c);

}

else

{

energy\_array[(r+3)\*7+(c+3)] = Distance(contour\_row[0], contour\_row[i]+r, contour\_col[0], contour\_col[i]+c);

}

}

}

return;

}

void InternalEnergy2(float \*energy\_array, int \*contour\_row, int \*contour\_col, int avg, int i)

{

int r, c;

for (r = -3; r <= 3; r++)

{

for (c = -3; c <= 3; c++)

{

if (i != 41)

{

energy\_array[(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

{

energy\_array[(r+3)\*7+(c+3)] = SQR(avg - Distance(contour\_row[0], contour\_row[i]+r, contour\_col[0], contour\_col[i]+c));

}

}

}

return;

}

void ExternalEnergy(float \*energy\_array, float \*sobel\_image, int \*contour\_row, int \*contour\_col, int COLS, int i)

{

int r, c;

for (r = -3; r <= 3; r++)

{

for (c = -3; c <= 3; c++)

{

energy\_array[(r+3)\*7+(c+3)] = SQR(sobel\_image[(contour\_row[i]+r)\*COLS+(contour\_col[i]+c)]);

}

}

return;

}

void SobelFilter(unsigned char \*image, float \*sobel\_image, int ROWS, int COLS)

{

int r, c, r2, c2;

float sumF1, sumF2;

for (r = 3; r < ROWS - 3; r++)

{

for (c = 3; c < COLS - 3; c++)

{

sumF1 = 0;

sumF2 = 0;

for (r2 = -1; r2 <= 1; r2++)

{

for (c2 = -1; c2 <= 1; c2++)

{

sumF1 += image[(r+r2)\*COLS+(c+c2)] \* F1[(r2+1)\*3+(c2+1)];

sumF2 += image[(r+r2)\*COLS+(c+c2)] \* F2[(r2+1)\*3+(c2+1)];

}

}

sobel\_image[r\*COLS+c] = sqrt(SQR(sumF1) + SQR(sumF2));

}

}

return;

}

void Normalize(float \*temp, int ROWS, int COLS, int range)

{

int i;

int max, min;

// Determine Max and Min

max = 0;

min = 2000;

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

{

if (temp[i] < min)

{

min = temp[i];

}

if (temp[i] > max)

{

max = temp[i];

}

}

// Function for nomalization found online

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

{

temp[i] = (temp[i] - min)\*range/(max-min);

}

return;

}

void Float2Unsigned(float \*input, unsigned char \*output, int ROWS, int COLS)

{

int i;

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

{

output[i] = input[i];

}

return;

}