**LAB 2 REPORT – Aaron Bruner**

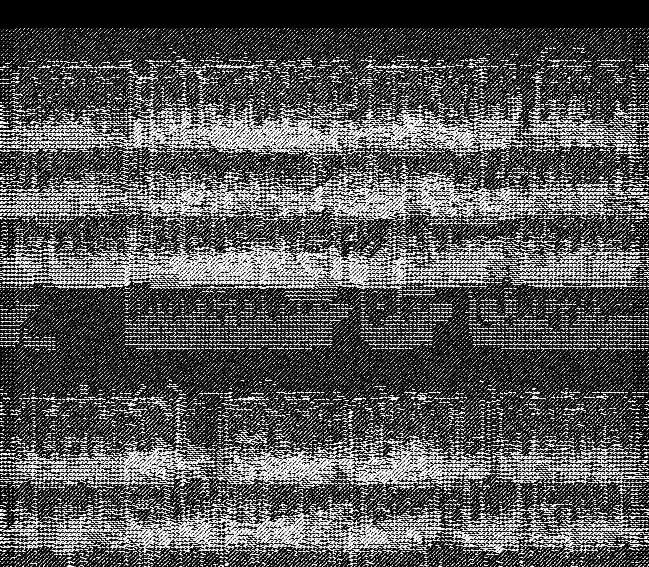
The purpose of this lab was to detect the location of characters in an image. The lab instructions laid out 4 steps for us to follow. The first step was to read the input image, template image, and ground truth files.

./lab2  
Step 1:  
Performing matched filter on images [parenthood.ppm] and [parenthood\_e\_template.ppm] using ground truth [parenthood\_gt.txt]  
 \* Reading in source image... [SUCCESS]  
 \* Reading in template image... [SUCCESS]  
 \* Opening ground truth file... [SUCCESS]  
 \* Found 1261 number of rows in the ground truth file  
 \* Allocating space for ground truth file... [SUCCESS]  
 \* Scanning in values from ground truth file... [Read in 1261 rows]

The above text is the first step output from the terminal when we execute out code. As we can see, the files parenthood.ppm, parenthood\_e\_template.ppm and parenthood\_gt.txt are used since command line arguments were not provided. We have the option of specifying which files we want to use by using the following command: ./lab2 (sourceFile.ppm) (templateFile.ppm) (groundTruth.txt). We can see that 1261 rows were read in from the ground truth and all files were successfully opened and read in.

Step 2 asked us to calculate the matched-spatial filter (MSF) image. Below is the output from the terminal and the MSF image.

Step 2:  
Calculate the mean of the template image...  
 \* Mean pixel value in the template image = 165  
 \* Generating the zero mean template image  
 \* Allocating space for template MSF image... [SUCCESS]  
 \* Allocating space for MSF image... [SUCCESS]  
 \* Convolving source and zero-mean centered image... [SUCCESS]



Step 3 asked us to normalize the MSF image to 8-bits. Below is the output from the terminal and the normalized image.

Step 3:  
Finding the minimum pixel and maximum pixel of the MSF...  
 \* Calculating the minimum and maximum pixel in MSF image... [SUCCESS]  
 \* Minimum determined to be: -128215  
 \* Maximum determined to be: 309645  
 \* Normalizing the MSF image to 8-bit...  
 \* Creating space for normalized image... [SUCCESS]

Text

Description automatically generated

The pixel values with values closer to 255 are more likely to be e.

Step 4 has us thresholding the above image for ranging values of T. The ideal value of T was determined to be 200 which yields the highest number of e’s with the lowest number of FP.

Step 4:  
Creating a binary image using the threshold...  
 \* Allocating space for result image [SUCCESS]  
 \* Loop over the MSF image with threshold values from 0 to 255 incrementing by 10...  
 \* Generating the ideal OCR image using threshold value [200]... [SUCCESS]  
 \* Sending result image to idealImage.ppm... [SUCCESS]

The full output of TP and FP values for T values ranging from 0 to 255 are on the next page along with the ideal output image at 200.

Background pattern

Description automatically generated

Threshold [ 0] : TP = 151 | FP = 1111

Threshold [ 1] : TP = 151 | FP = 1111

Threshold [ 2] : TP = 151 | FP = 1111

Threshold [ 3] : TP = 151 | FP = 1111

Threshold [ 4] : TP = 151 | FP = 1111

Threshold [ 5] : TP = 151 | FP = 1111

Threshold [ 6] : TP = 151 | FP = 1111

Threshold [ 7] : TP = 151 | FP = 1111

Threshold [ 8] : TP = 151 | FP = 1111

Threshold [ 9] : TP = 151 | FP = 1111

Threshold [ 10] : TP = 151 | FP = 1111

Threshold [ 11] : TP = 151 | FP = 1111

Threshold [ 12] : TP = 151 | FP = 1111

Threshold [ 13] : TP = 151 | FP = 1111

Threshold [ 14] : TP = 151 | FP = 1111

Threshold [ 15] : TP = 151 | FP = 1111

Threshold [ 16] : TP = 151 | FP = 1111

Threshold [ 17] : TP = 151 | FP = 1111

Threshold [ 18] : TP = 151 | FP = 1111

Threshold [ 19] : TP = 151 | FP = 1111

Threshold [ 20] : TP = 151 | FP = 1111

Threshold [ 21] : TP = 151 | FP = 1111

Threshold [ 22] : TP = 151 | FP = 1111

Threshold [ 23] : TP = 151 | FP = 1111

Threshold [ 24] : TP = 151 | FP = 1111

Threshold [ 25] : TP = 151 | FP = 1111

Threshold [ 26] : TP = 151 | FP = 1111

Threshold [ 27] : TP = 151 | FP = 1111

Threshold [ 28] : TP = 151 | FP = 1111

Threshold [ 29] : TP = 151 | FP = 1111

Threshold [ 30] : TP = 151 | FP = 1111

Threshold [ 31] : TP = 151 | FP = 1111

Threshold [ 32] : TP = 151 | FP = 1111

Threshold [ 33] : TP = 151 | FP = 1111

Threshold [ 34] : TP = 151 | FP = 1111

Threshold [ 35] : TP = 151 | FP = 1111

Threshold [ 36] : TP = 151 | FP = 1111

Threshold [ 37] : TP = 151 | FP = 1111

Threshold [ 38] : TP = 151 | FP = 1111

Threshold [ 39] : TP = 151 | FP = 1111

Threshold [ 40] : TP = 151 | FP = 1111

Threshold [ 41] : TP = 151 | FP = 1111

Threshold [ 42] : TP = 151 | FP = 1111

Threshold [ 43] : TP = 151 | FP = 1111

Threshold [ 44] : TP = 151 | FP = 1111

Threshold [ 45] : TP = 151 | FP = 1111

Threshold [ 46] : TP = 151 | FP = 1111

Threshold [ 47] : TP = 151 | FP = 1111

Threshold [ 48] : TP = 151 | FP = 1111

Threshold [ 49] : TP = 151 | FP = 1111

Threshold [ 50] : TP = 151 | FP = 1111

Threshold [ 51] : TP = 151 | FP = 1111

Threshold [ 52] : TP = 151 | FP = 1111

Threshold [ 53] : TP = 151 | FP = 1111

Threshold [ 54] : TP = 151 | FP = 1111

Threshold [ 55] : TP = 151 | FP = 1111

Threshold [ 56] : TP = 151 | FP = 1111

Threshold [ 57] : TP = 151 | FP = 1111

Threshold [ 58] : TP = 151 | FP = 1111

Threshold [ 59] : TP = 151 | FP = 1111

Threshold [ 60] : TP = 151 | FP = 1111

Threshold [ 61] : TP = 151 | FP = 1111

Threshold [ 62] : TP = 151 | FP = 1111

Threshold [ 63] : TP = 151 | FP = 1111

Threshold [ 64] : TP = 151 | FP = 1111

Threshold [ 65] : TP = 151 | FP = 1111

Threshold [ 66] : TP = 151 | FP = 1111

Threshold [ 67] : TP = 151 | FP = 1111

Threshold [ 68] : TP = 151 | FP = 1111

Threshold [ 69] : TP = 151 | FP = 1111

Threshold [ 70] : TP = 151 | FP = 1111

Threshold [ 71] : TP = 151 | FP = 1111

Threshold [ 72] : TP = 151 | FP = 1111

Threshold [ 73] : TP = 151 | FP = 1111

Threshold [ 74] : TP = 151 | FP = 1111

Threshold [ 75] : TP = 151 | FP = 1111

Threshold [ 76] : TP = 151 | FP = 1111

Threshold [ 77] : TP = 151 | FP = 1111

Threshold [ 78] : TP = 151 | FP = 1111

Threshold [ 79] : TP = 151 | FP = 1111

Threshold [ 80] : TP = 151 | FP = 1111

Threshold [ 81] : TP = 151 | FP = 1111

Threshold [ 82] : TP = 151 | FP = 1111

Threshold [ 83] : TP = 151 | FP = 1111

Threshold [ 84] : TP = 151 | FP = 1111

Threshold [ 85] : TP = 151 | FP = 1111

Threshold [ 86] : TP = 151 | FP = 1111

Threshold [ 87] : TP = 151 | FP = 1111

Threshold [ 88] : TP = 151 | FP = 1111

Threshold [ 89] : TP = 151 | FP = 1111

Threshold [ 90] : TP = 151 | FP = 1111

Threshold [ 91] : TP = 151 | FP = 1111

Threshold [ 92] : TP = 151 | FP = 1111

Threshold [ 93] : TP = 151 | FP = 1111

Threshold [ 94] : TP = 151 | FP = 1111

Threshold [ 95] : TP = 151 | FP = 1111

Threshold [ 96] : TP = 151 | FP = 1111

Threshold [ 97] : TP = 151 | FP = 1111

Threshold [ 98] : TP = 151 | FP = 1111

Threshold [ 99] : TP = 151 | FP = 1111

Threshold [100] : TP = 151| FP = 1111

Threshold [101] : TP = 151| FP = 1111

Threshold [102] : TP = 151| FP = 1111

Threshold [103] : TP = 151| FP = 1111

Threshold [104] : TP = 151| FP = 1111

Threshold [105] : TP = 151| FP = 1111

Threshold [106] : TP = 151| FP = 1111

Threshold [107] : TP = 151| FP = 1111

Threshold [108] : TP = 151| FP = 1110

Threshold [109] : TP = 151| FP = 1110

Threshold [110] : TP = 151| FP = 1110

Threshold [111] : TP = 151| FP = 1110

Threshold [112] : TP = 151| FP = 1109

Threshold [113] : TP = 151| FP = 1109

Threshold [114] : TP = 151| FP = 1108

Threshold [115] : TP = 151| FP = 1107

Threshold [116] : TP = 151| FP = 1106

Threshold [117] : TP = 151| FP = 1105

Threshold [118] : TP = 151| FP = 1105

Threshold [119] : TP = 151| FP = 1102

Threshold [120] : TP = 151| FP = 1101

Threshold [121] : TP = 151| FP = 1097

Threshold [122] : TP = 151| FP = 1097

Threshold [123] : TP = 151| FP = 1096

Threshold [124] : TP = 151| FP = 1096

Threshold [125] : TP = 151| FP = 1094

Threshold [126] : TP = 151| FP = 1091

Threshold [127] : TP = 151| FP = 1088

Threshold [128] : TP = 151| FP = 1084

Threshold [129] : TP = 151| FP = 1079

Threshold [130] : TP = 151| FP = 1073

Threshold [131] : TP = 151| FP = 1068

Threshold [132] : TP = 151| FP = 1066

Threshold [133] : TP = 151| FP = 1058

Threshold [134] : TP = 151| FP = 1053

Threshold [135] : TP = 151| FP = 1047

Threshold [136] : TP = 151| FP = 1041

Threshold [137] : TP = 151| FP = 1035

Threshold [138] : TP = 151| FP = 1026

Threshold [139] : TP = 151| FP = 1016

Threshold [140] : TP = 151| FP = 1010

Threshold [141] : TP = 151| FP = 1003

Threshold [142] : TP = 151| FP = 994

Threshold [143] : TP = 151| FP = 984

Threshold [144] : TP = 151| FP = 971

Threshold [145] : TP = 151| FP = 957

Threshold [146] : TP = 151| FP = 947

Threshold [147] : TP = 151| FP = 929

Threshold [148] : TP = 151| FP = 921

Threshold [149] : TP = 151| FP = 908

Threshold [150] : TP = 151| FP = 889

Threshold [151] : TP = 151| FP = 873

Threshold [152] : TP = 151| FP = 849

Threshold [153] : TP = 151| FP = 830

Threshold [154] : TP = 151| FP = 809

Threshold [155] : TP = 151| FP = 791

Threshold [156] : TP = 151| FP = 767

Threshold [157] : TP = 151| FP = 746

Threshold [158] : TP = 151| FP = 726

Threshold [159] : TP = 151| FP = 699

Threshold [160] : TP = 151| FP = 675

Threshold [161] : TP = 151| FP = 656

Threshold [162] : TP = 151| FP = 643

Threshold [163] : TP = 151| FP = 624

Threshold [164] : TP = 151| FP = 612

Threshold [165] : TP = 151| FP = 597

Threshold [166] : TP = 151| FP = 580

Threshold [167] : TP = 151| FP = 565

Threshold [168] : TP = 151| FP = 554

Threshold [169] : TP = 151| FP = 544

Threshold [170] : TP = 151| FP = 535

Threshold [171] : TP = 151| FP = 519

Threshold [172] : TP = 151| FP = 509

Threshold [173] : TP = 151| FP = 492

Threshold [174] : TP = 151| FP = 477

Threshold [175] : TP = 151| FP = 468

Threshold [176] : TP = 151| FP = 450

Threshold [177] : TP = 151| FP = 440

Threshold [178] : TP = 151| FP = 421

Threshold [179] : TP = 151| FP = 400

Threshold [180] : TP = 151| FP = 386

Threshold [181] : TP = 151| FP = 372

Threshold [182] : TP = 151| FP = 349

Threshold [183] : TP = 151| FP = 339

Threshold [184] : TP = 151| FP = 319

Threshold [185] : TP = 151| FP = 308

Threshold [186] : TP = 151| FP = 292

Threshold [187] : TP = 151| FP = 283

Threshold [188] : TP = 151| FP = 272

Threshold [189] : TP = 151| FP = 253

Threshold [190] : TP = 151| FP = 231

Threshold [191] : TP = 151| FP = 215

Threshold [192] : TP = 151| FP = 200

Threshold [193] : TP = 151| FP = 187

Threshold [194] : TP = 151| FP = 180

Threshold [195] : TP = 151| FP = 169

Threshold [196] : TP = 151| FP = 162

Threshold [197] : TP = 151| FP = 148

Threshold [198] : TP = 151| FP = 141

Threshold [199] : TP = 151| FP = 133

Threshold [200] : TP = 151| FP = 124

Threshold [201] : TP = 150| FP = 113

Threshold [202] : TP = 150| FP = 108

Threshold [203] : TP = 150| FP = 101

Threshold [204] : TP = 149| FP = 90

Threshold [205] : TP = 148| FP = 82

Threshold [206] : TP = 147| FP = 75

Threshold [207] : TP = 146| FP = 69

Threshold [208] : TP = 145| FP = 65

Threshold [209] : TP = 144| FP = 64

Threshold [210] : TP = 143| FP = 59

Threshold [211] : TP = 142| FP = 54

Threshold [212] : TP = 142| FP = 52

Threshold [213] : TP = 140| FP = 48

Threshold [214] : TP = 139| FP = 45

Threshold [215] : TP = 133| FP = 43

Threshold [216] : TP = 133| FP = 42

Threshold [217] : TP = 130| FP = 40

Threshold [218] : TP = 128| FP = 37

Threshold [219] : TP = 122| FP = 34

Threshold [220] : TP = 121| FP = 30

Threshold [221] : TP = 118| FP = 25

Threshold [222] : TP = 115| FP = 20

Threshold [223] : TP = 111| FP = 17

Threshold [224] : TP = 104| FP = 16

Threshold [225] : TP = 99 | FP = 14

Threshold [226] : TP = 98 | FP = 10

Threshold [227] : TP = 93 | FP = 7

Threshold [228] : TP = 89 | FP = 7

Threshold [229] : TP = 81 | FP = 5

Threshold [230] : TP = 75 | FP = 5

Threshold [231] : TP = 68 | FP = 4

Threshold [232] : TP = 63 | FP = 4

Threshold [233] : TP = 60 | FP = 2

Threshold [234] : TP = 53 | FP = 2

Threshold [235] : TP = 48 | FP = 1

Threshold [236] : TP = 43 | FP = 1

Threshold [237] : TP = 41 | FP = 0

Threshold [238] : TP = 39 | FP = 0

Threshold [239] : TP = 35 | FP = 0

Threshold [240] : TP = 31 | FP = 0

Threshold [241] : TP = 28 | FP = 0

Threshold [242] : TP = 24 | FP = 0

Threshold [243] : TP = 23 | FP = 0

Threshold [244] : TP = 20 | FP = 0

Threshold [245] : TP = 16 | FP = 0

Threshold [246] : TP = 13 | FP = 0

Threshold [247] : TP = 11 | FP = 0

Threshold [248] : TP = 8 | FP = 0

Threshold [249] : TP = 6 | FP = 0

Threshold [250] : TP = 5 | FP = 0

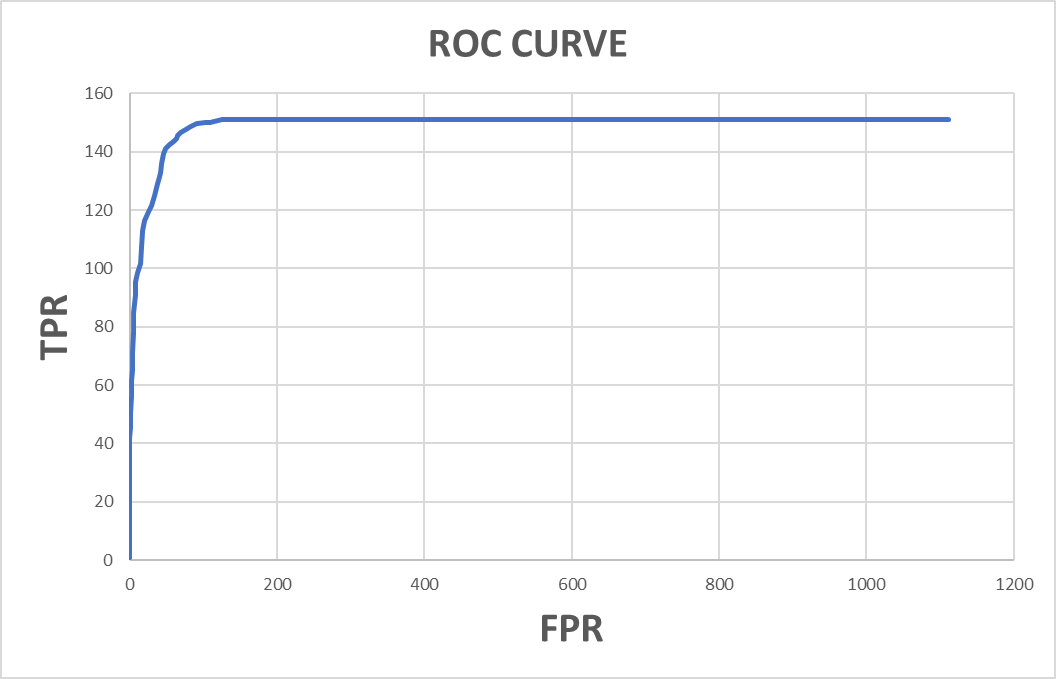
Threshold [251] : TP = 1 | FP = 0

Threshold [252] : TP = 1 | FP = 0

Threshold [253] : TP = 1 | FP = 0

Threshold [254] : TP = 1 | FP = 0

Threshold [255] : TP = 1 | FP = 0

****

Source Code:

/\* File : lab2.c

Author: Aaron Bruner

Class : ECE - 4310 : Introduction to Computer Vision

Term : Fall 2022

Description: The purpose of this lab was to design and implement a matched filter (normalized cross-correlation) to recognize letters in an image of text.

The ground truth file lists all the letters and image pixel coordinates of text in the image. The pixel coordinates are for the center point of each letter.

Required Files:

\* parenthood.ppm

\* parenthood\_e\_template.ppm

\* parenthood\_gt.txt

Bugs:

\* Currently none

\*/

#define True 1

#define False 0

#define DEBUG False

#define T 255 // Upper limit for thresholding

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

struct groundTruth {

char letter;

int x; // COLUMN

int y; // ROW

};

unsigned char\* readImage( int\* ROWS, int\* COLS, char\* source);

unsigned char\* createImage(int size);

char\* sourceImageDir = "parenthood.ppm";

char\* templateImageDir = "parenthood\_e\_template.ppm";

char\* groundTruthDir = "parenthood\_gt.txt";

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

{

unsigned char\* sourceImage, \*templateImage, \*zeroMeanImage, \*MSF\_Normalized, \*thresholdImage;

int\* templateMSF, \*MSF;

struct groundTruth\* truth;

char sourceHeader[320], templateHeader[320], temp, letter;

int temp1, temp2, fileRows = 0; // Number of rows in the ground truth file

int i = 0, j = 0, mean = 0, sourceROWS, sourceCOLS, templateROWS, templateCOLS, filterRow, filterCol;

int r, c, dr, dc, wr, wc, average, min, max, found = False, TP, FP, maxTP, maxT;

FILE\* fpt, \*TPFPfpt;

/\* ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ \*/

/\* STEP 1: Read in source, template and ground truth \*/

/\* \* User provides no arguments (argc == 1) then we default to specified files \*/

/\* \* User provides 4 arguments (argc == 4) then we open provided files \*/

/\* ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ \*/

printf("Step 1:\n");

if (argc == 1) {

printf("Performing matched filter on images [%s] and [%s] using ground truth [%s]\n", sourceImageDir, templateImageDir, groundTruthDir);

printf("\t\* Reading in source image...");

sourceImage = readImage(&sourceROWS, &sourceCOLS, sourceImageDir); printf("\t[SUCCESS]\n");

printf("\t\* Reading in template image...");

templateImage = readImage(&templateROWS, &templateCOLS, templateImageDir); printf("\t[SUCCESS]\n");

// Read in CSV/TXT file

printf("\t\* Opening ground truth file...");

fpt = fopen(groundTruthDir, "r");

fpt == NULL ? printf("Failed to open %s\n", groundTruthDir), exit(0) : printf("\t[SUCCESS]\n");

}

else if (argc == 4)

{

printf("Performing matched filter on images [%s] and [%s] using ground truth [%s]\n", argv[1], argv[2], argv[3]);

printf("\t\* Reading in source image...");

sourceImage = readImage(&sourceROWS, &sourceCOLS, argv[1]); printf("\t[SUCCESS]\n");

printf("\t\* Reading in template image...");

templateImage = readImage(&templateROWS, &templateCOLS, argv[2]); printf("\t[SUCCESS]\n");

// Read in CSV/TXT file

printf("\t\* Opening ground truth file...");

fpt = fopen(argv[3], "r");

fpt == NULL ? printf("Failed to open %s\n", argv[3]), exit(0) : printf("\t[SUCCESS]\n");

}

else

{

printf("Incorrect number of arguments...\nUsage: ./lab2 (sourceImage.ppm) (templateImage.ppm) (groundTruth.txt)\n");

exit(0);

}

while ((i = fscanf(fpt, "%c %d %d\n", &temp, &temp1, &temp2)) && !feof(fpt))

if (i == 3) fileRows += 1;

printf("\t\* Found %d number of rows in the ground truth file\n", fileRows);

printf("\t\* Allocating space for ground truth file...");

truth = calloc(fileRows, sizeof(struct groundTruth)); printf("\t[SUCCESS]\n");

rewind(fpt); // Return to the beginning of the file

printf("\t\* Scanning in values from ground truth file...");

for (i = 0; i <= fileRows && !feof(fpt); i++)

{

fscanf(fpt, "%c %d %d\n", &truth[i].letter, &truth[i].x, &truth[i].y);

}

fclose(fpt);

printf("\t[Read in %d rows]\n", i - 1);

/\* ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ \*/

/\* STEP 2: Calculate the matched-spatial filter (MSF) image. \*/

/\* a) Zero-Mean Center the template \*/

/\* b) Convolve with image \*/

/\* ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ \*/

printf("Step 2:\n"); printf("Calculate the mean of the template image...\n");

for (i = 0; i < templateCOLS \* templateROWS; i++) // Sum all pixels

mean += templateImage[i];

mean /= templateCOLS \* templateROWS;

printf("\t\* Mean pixel value in the template image = %d\n", mean);

// Zero Mean Template Image

printf("\t\* Generating the zero mean template image\n");

printf("\t\t\* Allocating space for template MSF image...");

templateMSF = (int\*)calloc(templateCOLS \* templateROWS, sizeof(int)); printf("\t[SUCCESS]\n");

for (i = 0; i < templateCOLS \* templateROWS; i++)

templateMSF[i] = templateImage[i] - mean;

// MSF[r,c] = SIG(+Wr/2 -> dr=Wr/2) SIG(+Wc/2 -> dc=Wc/2)[ I[r + dr,c + dc] \* T[dr + Wr/2,dc + Wc/2] ]

printf("\t\t\* Allocating space for MSF image...");

MSF = (int\*)calloc(sourceCOLS \* sourceROWS, sizeof(int)); printf("\t[SUCCESS]\n");

printf("\t\t\* Convolving source and zero-mean centered image...");

wr = templateROWS; wc = templateCOLS; dr = wr/2; dc = wc/2;

for (r = dr; r < sourceROWS - dr; r++)

{

for (c = dc; c < sourceCOLS - dc; c++, average = 0)

{

for (filterRow = -dr; filterRow < templateROWS - dr; filterRow++)

{

for (filterCol = -dc; filterCol < templateCOLS - dc; filterCol++)

{

average += sourceImage[(r + filterRow) \* sourceCOLS + (c + filterCol)] \*

templateMSF[(filterRow + dr) \* templateCOLS + (filterCol + dc)];

}

}

MSF[r \* sourceCOLS + c] = (int)average;

}

}

printf("\t[SUCCESS]\n");

/\* ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ \*/

/\* STEP 3: Normalize the MSF image to 8-bit \*/

/\* a) Find the min and max of MSF \*/

/\* b) Create the 8-bit representation of the MSF \*/

/\* ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ \*/

printf("Step 3:\n"); printf("Finding the minimum pixel and maximum pixel of the MSF...\n");

min = max = MSF[0];

printf("\t\* Calculating the minimum and maximum pixel in MSF image...");

for (i = 1; i < sourceROWS \* sourceCOLS; i++)

{

if (MSF[i] > max)

max = MSF[i];

if (MSF[i] < min)

min = MSF[i];

}

printf("\t[SUCCESS]\n");

printf("\t\t\* Minimum determined to be: %d\n\t\t\* Maximum determined to be: %d\n", min, max);

printf("\t\* Normalizing the MSF image to 8-bit...");

printf("\n\t\t\* Creating space for normalized image...");

MSF\_Normalized = createImage(sourceCOLS \* sourceROWS);

printf("\t[SUCCESS]\n");

for (i = 0; i < sourceROWS \* sourceCOLS; i++)

{

// https://en.wikipedia.org/wiki/Normalization\_(image\_processing)

MSF\_Normalized[i] = (MSF[i] - min) \* 255 / (max - min);

}

/\* ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ \*/

/\* STEP 4: Looping through the following steps for a range of T \*/

/\* a) Threshold at T the normalized MSF image to create a binary image \*/

/\* b) Loop through the ground truth letter locations \*/

/\* i. Check a 9 x 15 pixel area centered at the ground truth location. If \*/

/\* any pixel in the MSF image is greater than the threshold, consider \*/

/\* the letter “detected”.If none of the pixels in the 9 x 15 area are \*/

/\* greater than the threshold, consider the letter “not detected” \*/

/\* c) Categorize and count the detected letters as FP (“detected” but the letter is \*/

/\* not ‘e’) and TP (“detected” and the letter is ‘e’) \*/

/\* d) Output the total FP and TP for each T \*/

/\* ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ \*/

printf("Step 4:\n"); printf("Creating a binary image using the threshold...");

printf("\n\t\* Allocating space for result image");

thresholdImage = createImage(sourceCOLS \* sourceROWS); printf("\t[SUCCESS]\n");

char outStr[20];

TPFPfpt = fopen("TPFP.txt", "w"); TPFPfpt == NULL ? (printf("Failed to open TPFP.txt.\n"), exit(0)) : TPFPfpt;

printf("\t\* Loop over the MSF image with threshold values from 0 to %d incrementing by 10...\n", T);

for (i = 0; i <= T; i++, TP = 0, FP = 0)

{

// Part a: Generate binary image using threshold

for (int pixel = 0; pixel < sourceCOLS \* sourceROWS; pixel++)

{

thresholdImage[pixel] = MSF\_Normalized[pixel] >= i ? (unsigned char)255 : (unsigned char)0;

}

// Part b: Looping through the ground truth letter locations

for (j = 0; j <= fileRows; j++, found = False)

{

for (filterRow = truth[j].y - dr; filterRow <= truth[j].y + dr; filterRow++)

{

for (filterCol = truth[j].x - dc; filterCol <= truth[j].x + dc; filterCol++)

{

// i) found a 255 pixel within a 9x15 area of x,y | Check to see if it's already true

found = (thresholdImage[filterRow \* sourceCOLS + filterCol] == 255) ? True : (found == True) ? True : False;

}

}

// Part c: If we find a 255 pixel and it's actually e then TP. Otherwise we find something that's not e it's a FP

truth[j].letter == 'e' ? (found == True ? TP++ : TP) : (found == True ? FP++ : FP);

}

// Part d: Outputting the total FP and TP for each T value ranging from 0 to 255

fprintf(TPFPfpt, "Threshold [%3d] : TP = %4d\t| FP = %4d\n", i, TP, FP);

// Find the threshold that gives us the larges amount of TP

if (T == 0)

{

maxTP = TP;

maxT = i;

}

else if (TP >= maxTP)

{

maxTP = TP;

maxT = i;

}

}

// Generating the ideal image using the threshold value with the most TP

printf("\t\* Generating the ideal OCR image using threshold value [%d]...", maxT);

for (int pixel = 0; pixel < sourceCOLS \* sourceROWS; pixel++)

{

thresholdImage[pixel] = MSF\_Normalized[pixel] >= maxT ? (unsigned char)255 : (unsigned char)0;

}

printf("\t[SUCCESS]\n");

printf("\t\* Sending result image to idealImage.ppm...");

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

fprintf(fpt, "P5 %d %d 255\n", sourceCOLS, sourceROWS);

fwrite(thresholdImage, sourceCOLS \* sourceROWS, 1, fpt);

fclose(fpt);

printf("\t[SUCCESS]\n");

/\* ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ \*/

}

/// <summary>

/// The readImage function is designed to take a file name as the source and reads all of the data into a new image.

/// </summary>

/// <param name="ROWS"> Number of rows in the source image </param>

/// <param name="COLS"> Number of columns in the source image </param>

/// <param name="source"> File name that we're needing to open and read data from </param>

/// <returns> The function returns an array of values which makes up our image </returns>

unsigned char\* readImage(int\* ROWS, int\* COLS, char\* source)

{

int BYTES, readHeaderReturn;

static char header[80];

// Open image for reading

FILE \*fpt = fopen(source, "rb");

if (fpt == NULL) {

printf("Failed to open file (%s) for reading.\n", source);

exit(0);

}

/\* read image header (simple 8-bit greyscale PPM only) \*/

if (fscanf(fpt, "%s %d %d %d\n", header, &\*COLS, &\*ROWS, &BYTES) != 4 || strcmp(header, "P5") != 0 || BYTES != 255)

{

fclose(fpt);

printf("Image header corrupted.\n");

exit(0);

}

unsigned char\* destination = createImage((\*ROWS)\*(\*COLS)); // Create an empty image that is large enough for ROWS x COLS bytes

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

fclose(fpt);

return destination;

}

/// <summary>

/// createImage allocates memory for our image array.

/// </summary>

/// <param name="size"> Number of bytes that are needing to be allocated for our image </param>

/// <returns> An array with 'size' number of bytes allocated for our image use</returns>

unsigned char\* createImage(int size)

{

unsigned char\* newImage = (unsigned char\*)calloc(size, sizeof(unsigned char));

if (newImage == NULL) {

printf("Unable to allocate %d bytes of memory.\n", size);

exit(0);

}

return newImage;

}