CORRELATION EVALUATION OF INTENSITY AND COLOR BAND IMAGES

JANEEN M. CARTER
MORGAN STATE UNIVERSITY

RECEIVED

JAN 23 1995

OSTI

1. EXECUTIVE SUMMARY

The purpose of this project is to determine which of the three color bands - red, green, or blue - to use in providing the best possible correlation and to determine the accuracy with which these color bands correlate in comparison with the correlation of the three color bands with the intensity model.

To fulfill this purpose, the correlation technique of template matching is implemented using a correlator. Correlations are implemented with each of the individual color bands and also with the corresponding intensity model. The correlation coefficient resulting from a successful correlation ranges from 0.9 to 1. A coefficient of 1 demonstrates that the feature information varies identically.

When analyzing the data collected from the correlations, the following results are obtained. The color band recommended for the most accurate correlation is the green color band. The correlation of the color bands with the intensity model was not as successful in determining the better color band because the correlation coefficients were very low in comparison to the correlation of the individual color bands.

2. INTRODUCTION

When capturing full color microscopic images, the complete full color digital image cannot be captured. Therefore, overlapping segments are captured. These overlapping segments are correlated using the technique of template matching and then they are mapped together to obtain the high resolution full color digital image.

When the image is captured, it is captured in the three primary colors: red, green, and

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED
NIA

blue. From the three primary colors, the intensity model is developed. The intensity model comprises the average of the feature information found in the three color bands. The objective of this project is to determine if the individual red, green, and blue color bands can be used when correlating full color digital image segments, using template matching, or whether it is necessary to develop and use the intensity model.

3. TECHNICAL APPROACH

In carrying out the objective stated above, the correlation technique of template matching was implemented. Template matching is the technique which determines the closest match between an unknown image and a set of known images. This technique is fundamental in the function of the correlator, which is a tool used to correlate two adjacent image segments. The correlation process entails the correlation of a frame and a window. The frame is a 150 x 150 pixel area from a given image segment and the window is a 90 x 90 pixel area from an adjacent overlapping image segment.

Using the specified parameters to locate the frame and the window, the window will contain feature information present in the frame. Therefore, when correlating the frame and window using the correlation equation, a similarity in feature information will exist. The correlation equation is shown below:

$$R_{(u,v)} = \frac{F_{(i,j)} W_{(i+u,j+v)}}{[F_{(i,j)}^2]^{1/2} [W_{(i,j)}^2]^{1/2}}$$

where $R_{(u,v)}$ is the correlation coefficient, $F_{(i,j)}$ is the pixel value of the frame and $W_{(i+u,j+v)}$ is the pixel value of the window.

This correlation equation calculates the correlation coefficient, $R_{(u,v)}$. The calculation starts with the first pixel value in the frame and the first pixel value in the window and

calculates their product. This process is repeated a given number of times and then the sum of the products is calculated. This series of calculations comprises the numerator of the correlation equation. Each frame pixel value up to a designated position inside the frame is squared, then each squared pixel value is summed and the square root is taken of the sum. At the same time, this process is carried out for the window pixel values; then the product of the square roots is taken. This series of calculations comprises the denominator of the correlation equation. This process is used to correlate the individual color bands and an analysis is taken to determine the degree of accuracy in using the correlation process.

When feature information is identical within the frame and window, the correlation coefficient will be 1. A value of one denotes an exact match in the feature information. Also values between 0.9 and 1 denote that the feature information is very similar, which means that the correlation is successful.

The correlation process is also implemented in correlating the individual color bands with the intensity model, which is an average of the pixel values existing in each of the color bands. The equation for the intensity model is shown below:

$$I = 1/3(R + G + B)$$

where **I** is the intensity model pixel value, **R** is the pixel value in the red color band, **G** is the pixel value in the green color band, and **B** is the pixel value in the blue color band.

When correlating the individual color bands with the intensity model, the same correlation coefficient values are obtained. Theoretically, the correlations of the intensity model with the individual color bands are as accurate as the correlations within the individual color bands.

4. DISCUSSION OF RESULTS

In collecting data, there are a total of 24 correlations within the red, green, and blue color bands. These correlations were implemented using overlapping, adjacent image segments. There are also a total of 12 correlations with the individual color bands and the intensity model. The intensity model has been correlated with the corresponding image segments.

In analyzing the data collected, the individual correlations are ranked from highest to lowest. When correlating the individual color bands, it is discovered that, overall, the red color band has the lowest correlation coefficients, the green color band has the highest correlation coefficients, and the coefficients found when correlating the blue color band fall in between those of the red and green color bands. So, in determining which color band to use for the most accurate correlation, the green color band is recommended.

When correlating the individual color bands with the intensity model, the same system of ranking is used. Overall, the correlations with the intensity model are very low. This fact alone proves that there is an error in the code for calculating the intensity model because these correlations should be as accurate as those correlations of the individual color bands. Despite this fact, the green color band shows the highest correlation coefficients.

5. CONCLUSION

This research assignment has proven to be extremely challenging and, at the same time, a great learning experience. With this particular project, further research can be done. Other aspects that could not be explored - due to lack of time - can be looked into by one who would like to continue research in this area.

6. ACKNOWLEDGEMENTS

Becoming a student intern at the Lister Hill Center has been a new and challenging

experience. It provided me with the opportunity to work in a professional atmosphere among other engineers as well as work on a specified task on an individual basis.

For this opportunity I have to thank Mr. Earl Henderson, my principal investigator during my stay at the Lister Hill Center. I would also like to thank Dr. Susan Hauser for her efforts in teaching the fundamentals of image processing. Also, thanks goes to Jim Seamans for being there to help with any problems I may have encountered. I would also like to thank Paulette Shelton for her endless efforts in providing us with thing we needed. Last but not least, I would like to thank my school preceptor, Dr. Pamela Mack, for her time and effort and also for introducing me to such a wonderful opportunity.

7. REFERENCES

Gonzalez, Rafael C. and Richard E. Woods. Digital Image Processing. Addison-Wesley Publishing Company. Reading, 1992.

Myler, Harley R. and Arthur R. Weeks. Computer Imaging Recipes in C. PTR Prentice Hall. Englewood Cliffs, 1993.

Waite, Mitchell and Stephen Prata. The Waite Group's New C Primer Plus. Carmel, 1990.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

APPENDIX

/*Program CORRELATE.c

This program correlates two overlapping, adjacent image segments. It extracts a frame and a window from the same image and correlates the two. The result is the correlation coefficient which determines the accuracy of the correlation. A coefficient of .9 to 1 indicates a successful correlation. Any values lower than this indicate an unsuccessful correlation.

Command line - correlate.c <image file> <outfile_1> <outfile_2>
*/

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
```

```
#define OFFSET1 286 /*column offset*/
#define OFFSET2 316 /*column offset*/
#define ROW1 150 /*row offset*/
#define ROW2 180 /*row offset*/
int image1[150]; /*normalized frame pixel values*/
unsigned char image2[90]; /*normalized window pixel values*/
unsigned char image_f[150];
unsigned char image_w[90];
unsigned char fram[150];
unsigned char win[90];
unsigned char fram_buf[90]; /*stores the changing frame pixel values*/
int buf_size;
```

```
FILE *infile, *fram_outfile, *win_outfile, *fram_file, *win_file;
```

```
void read_buf(FILE *in_buf, int start_row, int start_col, int buf_size, FILE *outfile)
```

```
/******BEGIN MAIN PROGRAM*****/
```

```
int main(int argc, char *argv[])
{
    int count, i, j, f, t;
    long fram_pix_sum=0, win_pix_sum=0;
    float fram_avg=0, win_avg=0;
    long read_byte, prod=0;
    int square_frame=0, square_win=0;
    int sum=0, sum_frame=0, sum_win=0;
    float sq_frame=0, sq_win=0;
    long current_pos=0, window_pos=0, frame_pos=0;
    long frame_loc=0, win_loc=0, pos1=0, pos2=0;
```

```
int num, m, n, w, window;  
int row, col, x, y, z, g;
```

```
if(argc < 4)  
{  
    printf("You have not provided the name of the image file.\n");  
    printf("Please re-enter the command with the file name.\n");  
    exit(1);  
}  
if ((infile = fopen(argv[1], "rb")) == NULL)  
{  
    fprintf(stderr, "Cannot open file %s.\n", argv[1]);  
    exit(1);  
}  
if((fram_outfile = fopen(argv[2], "w")) == NULL)  
{  
    fprintf(stderr, "Output file cannot be created.\n");  
    exit(1);  
}  
if((win_outfile = fopen(argv[3], "w")) == NULL)  
{  
    fprintf(stderr, "Output file cannot be created.\n");  
    exit(1);  
}
```

```
read_buf(infile, ROW1, OFFSET1, 150, fram_outfile);  
read_buf(infile, ROW2, OFFSET2, 90, win_outfile);
```

```
printf("The frame and window have been read into buffers.\n");
```

```
fclose(infile);
```

```
        /*****NORMALIZATION*****/
```

```
for(f=0; f<60; f++)  
{  
    frame_loc = (long)(f*150);  
    if(fseek(fram_outfile, frame_loc, SEEK_SET)!=0) /*Seeks to position in fram_outfile*/  
    {  
        printf("Error in fseek: fram_outfile.\n");  
        exit(1);  
    }  
    fread(image_f, sizeof(unsigned char), 150, fram_outfile); /*Reads fow to image_f*/  
  
    for(g = 0; g<90; g++)  
    {
```

```

    fram_pix_sum += image_f[g]; /*Sums the frame pixel values*/
}
}
fram_avg = fram_pix_sum/(150*150); /*Finds the avg of the frame pixel values*/

for(m=0; m<150; m++)/*Loops through array to subtract avg from each pixel*/
{
    pos1 = (long)(m*150); /*Calculates row location in frame*/
    if(fseek(fram_outfile, pos1, SEEK_SET)!=0) /*Seeks to calculated position*/
    {
        printf("Error in fseek: fram_outfile.\n");
        exit(1);
    }
    fread(image_f, sizeof(unsigned char), 150, fram_outfile);

    for(n=0, n<150, n++)
    {
        image1[n] = image_f[n] - fram_avg; /*Subtracts the avg from each pixel*/
    }
    ff = fopen("fram_file", "w");
    fwrite(image1, sizeof(int), 150, fram_file);
}
fclose(fram_file);
fram_file = fopen("frame", "rb"); /*Reopens fram_file to reset file pointer*/
if(fram_file == NULL)
{
    printf("fram_file is NULL.\n");
}

for(w=0; w<90; w++)
{
    win_loc = (long)(w*90);
    if(fseek(win_outfile, win_loc, SEEK_SET) !=0)
    {
        printf("Error in fseek: win_outfile.\n");
        exit(1);
    }

    fread(image_w, sizeof(unsigned char), 90, win_outfile);
    for(y=0, y<90; y++)
    {
        win_pix_sum += image_w[y]; /*Sums the window pixel values*/
    }
}
win_avg = win_pix_sum/(90*90); /*Finds the avg of the indow pixel values*/

```



```

for(x=0; x<90; x++)
{
    pos2 = (long)(x*90);
    if(fseek(win_outfile, pos2, SEEK_SET) !=0)
    {
        printf("Error in fseek: win_outfile.\n");
        exit(1);
    }

    fread(image_w, sizeof(unsigned char), 90, win_outfile);

    for(z=0; z<90; z++)
    {
        image2[z] = image_w[z] - win_avg; /*Subtracts the avg from each pixel*/
    }
}
wf = fopen("winfile", "w");
fwrite(image2, sizeof(unsigned char), 90, win_file);
fclose(win_file);
win_file = fopen("window", "rb"); /*Reopens win_file to reset file pointer*/
if(win_file == NULL)
{
    printf("win_file is NULL.\n");
}
printf("The normalization is complete.\n");

```

/*****END OF NORMALIZATION*****/

/*****BEGIN CORRELATION CALCULATIONS*****/

```

for(window=0; window<90; window++) /*Moves through window to obtain pixel values*/
{
    window_pos = (window * 90); /*Determines row location in window*/
    if(fseek(win_file, window_pos, SEEK_SET) !=0) /*Seeks to position in window*/
    {
        print("Error in fseek: win_file.\n");
        exit(1);
    }
    fread(win, sizeof(unsigned char), 90, win_file); /*Reads window to the win array*/

    sq_win = win>window] * win>window]; /*Squares window pixel values*/
    sum_win += square_win; /*Sums the squares*/
    sq_win += sqrt(sum_win);
}
printf("The window calculations have been completed.\n");

```

```

for(i=0; i<60; i++) /*Moves through frame to obtain pixel values*/
{
    for(j=0; j<60; j++)
    {
        frame_pos = 150*i+j; /*Calculates position in frame*/
        if(fseek(fram_file, frame_pos, SEEK_SET) !=0)
        {
            printf("Error in fseek: fram_file.\n");
            exit(1);
        }
        for(row=i; row<90+i; row++)
        {
            fread(fram, sizeof(unsigned char), 150, fram_file);
            for(col=j; col<90+j; col++)
            {
                /*Numerator calculations*/

                sum += (long)(fram[col] * win[col]); /*Sums the product of the frame and */
                                                    /*window pixel values */
                printf("The numerator calculations have been completed.\n");

                /*Denominator calculations*/

                square_frame += fram[col] * fram[col]; /*squares frame pixel values*/
                sum_frame += square_frame; /*Sums the squares of the frame pixel values*/
                sq_frame = sqrt(sum_frame); /*Takes the sqrt of the sums and multiplies the sqrts*/
                prod = sq_frame * sq_win;

                if(prod == 0)
                {
                    printf("Error: Division by zero.\n");
                }

                coeff[col] = sum/prod; /*Calculating correlation coefficient*/
                printf("Coefficient = %d row %d col %d\n", coeff[col], row, col);
                if(coeff[col] > coeff[col--]) /*Stores the highest coeff*/
                {
                    coeff1[col] = coeff[col];
                }
            }
        }
    }
}

printf("The denominator calculations have been completed.\n");
/******END OF CORRELATION CALCULATIONS*****/
fclose(fram_file);

```

```

fclose(win_file);

if((fclose(infile) !=0) && (fclose(fram_outfile) !=0)) && (fclose(win_outfile)!=0))
{
    printf("Error in closing file %s\n", argv[1]);
}
}

        /*****END OF MAIN PROGRAM*****/

/*This function reads the frame and window into their respective buffers*/

void read_buf(FILE *in_buf, int start_row, int start_col, int buf_size, FILE *outfile)
{
    unsigned char *rbuf = NULL;
    int row=0;
    long pos=0;

    if(fseek(in_buf, 0L, SEEK_SET)!=0) /*Resets pinter to beginning of file*/
    {
        printf("Error in seek to beginning of file.\n");
        fclose(in_buf);
        fclose(outfile);
        exit(1);
    }
    /*Allocates memory for rbuf*/
    if((rbuf = (unsigned char *)calloc(buf_size, sizeof(unsigned cah))) == NULL)
    {
        printf("Function read_buf: rbuf cannot be opened.\n");
        exit(1)
    }
    for(row = start_row; row < start_row + buf_size; row++)
    {
        pos = (row *512) + start_col; /*Calculates starting position*/
        if(fseek(in_buf, pos, SEEK_CUR) !=0) /*Seeks to that position*/
        {
            printf("Error in fseek.\n");
            free(rbuf);
            exit(1);
        }

        fread(rbuf, sizeof(unsigned char), buf_size, in_buf); /*Reads input into a rbuf*/
        fwrite(rbuf, sizeof(unsigned char), outfile); /*Writes buffer to outfile*/
    }
    free(rbuf);
    fclose(outfile);
}

```

```
/*Program INTENSITY.c
```

This program develops the intensity model which calculates the average of the pixel values contained in the red, green, and blue color bands.

```
Command line - intens <file.r__> <file.g__> <file.b__>
*/
```

```
#include <stdio.h>
```

```
FILE *red_file, *green_file, *blue_file;
FILE *r_file, *g_file, *b_file;
```

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

```
{
    int count, num;
    unsigned char sum[512];
    long float avg[512];
```

```
    red_file = fopen(argv[1], "rb");
    green_file = fopen(argv[2], "rb");
    blue_file = fopen(argv[3], "rb");
```

```
    for(count=0; count<512; count++)
    {
        fread(r_file, sizeof(unsigned char), 512, red_file);
        fread(g_file, sizeof(unsigned char), 512, green_file);
        fread(b_file, sizeof(unsigned char), 512, blue_file);
```

```
    for(num=0; num<512; num++)
    {
        sum[num] = r_file[num] + g_file[num] + b_file[num];
        avg[num] = sum[num]/3;
    }
```

```
    fwrite(avg, sizeof(unsigned char), 512, intens);
```

```
}
```