

.....

# CENG-566 IMAGE PROCESSING PROJECT

## HW-4



*By: Lütfi İteris ÖNEY  
Number: 1305937  
Submitted to: Dr. Sibel TARI*

## **ABSTRACT**

This project report's aim is to express ideas about HW4 from its theory to implementation. So we can cover all in these sub groups

1. Theory
2. Some Considerations on Feature Extraction
3. Co – occurrence Matrix
4. Features
5. Classification
6. Future Work
7. Source Codes

## THEORY

It is written that there is no formal definition of the texture in literature. This lack of satisfactory formal definition is due to the nature of the term, evading such a definition. Nevertheless the problem of retrieving similar textures falls within the category of statistical pattern recognition, and as idiosyncratic of problems within this category, any content based image retrieval system can be considered as rough consisting of two major subsystems. Texture features have been used to identify contents of images. It also describes the contents of images such as bricks, hairs, fabrics. So we can call texture as logical-neighborhood property. When a human being sees a texture it automatically describes its relatives connections.

Even at this stage I will describe what is in my mind, and what can be done depending on the project. First in order to start project you have to do some research. Homework was open end so doing some research will make sense. Exploring on some while I found that there are lots of techniques to determine texture. They are *Feature Extraction subsystem* and *Similarity Measurement subsystem*.

### Some Considerations on Feature Extraction

We are doing some experiment on a image. Gathering some information on it and trying to find out what the texture is like. In image processing lesson we have seen that there is a well known technique Fourier transform. So I thought about how can we get benefit from Fourier transform. Fourier transform has a power spectrum. That spectrum is much more concerned with periodicity and directionality of its texture. For instance an image of coarse texture would have a tendency towards low frequency components in its power spectrum, whereas another image with finer texture would have higher frequency components. Stripes in one direction would cause the power spectrum to concentrate near the line through the origin and perpendicular to the direction as you have drawn at class.

So if we have opportunity of seeing the image before, and selecting the technique for that texture as long as a given strong periodicity and movement (directions) Fourier analysis would be the right choice for that image.

The second technique for retrieving the information from the image is surely will be offline and detection must not be computational complex. This is a little bit different from our homework approach but giving more accurate results having the strong passes (high frequency variables). In order to compute the co-occurrence matrix, you find out some histogram so every computation is depending on that histogram.

## Co-occurrence Matrix

Co-occurrence matrix is based on the image, first you have to obtain some patches from the image. I have selected 30 patches from the image. First problem was to detect random number. In previous project I was assigning a `srand(1)` initially but (if you don't C automatically assigns it as 1) no idea what was it about. After reading some while I found that it is assigning some random values. But these values have the same order of sequence. So I will not give proper solution or it will give all the same solution as the program works. This is what I really don't, so I included `<time.h>` and assigned time variable to `srand`, so that whenever the `rand` function calls `srand` it generates random number.

Here below I included the code of co-occurrence matrix generation. Co-occurrence matrix have the form occurrences of the pairs. For to detect boundaries I preferred to choose not to take in account that is in those regions. But after project evolves for a period of days, I found that those values can be important to deal with. So I added `continue` statement whenever the pointer comes at a point at can not access. But this idea will give inner point more importance depending on the other patches. At those areas effect images energy, contrast etc. more than the pixels that are in the other patches.

```
for (patches=0 ; patches < 30 ; patches++)
{
for(y=0;y<255;y++)
    {
    for(x=0;x<255;x++)
    {
        co_occurence[x][y] = 0;
    }
}

srand((unsigned int) time(NULL));

randnumberwidth = (width * (rand() % 100)) /100;
randnumberheight = (height * (rand() % 100))/100;

for(y=0;y<32;y++)
{
    if(picture[randnumberheight+y][randnumberwidth] > height-2)
        continue;
    else
        for(x=0;x<32;x++)
        {
            if(picture[randnumberheight+y][randnumberwidth+x+1] < width-2)
                co_occurence
[picture[randnumberheight+y][randnumberwidth+x]][picture[randnumberheight+y][randnumberwidth+
x+1]]++;
            else
                continue;;
        }
}
```

## Features

I started with features Energy Entropy and Inverse difference moment , but while programming these features , I realized some notes, here they are

Energy : When all the matrix elements are almost equal , when and the values are very close to each other , the value of energy is small.

Entropy : It is the opposite of energy , thus it has a lower value when the co-occurrence is irregular. It has its highest peak when the co-occurrence is uniform. But because of it is opposite of energy , after testing some while I found that there is no need to keep entropy and energy together. Also another problem was in math.h log function was not working properly. So I removed Entropy from the code and added Contrast.

Contrast : Measures the difference moment of co-occurrence matrix. The value will be high if the image has high local variation.

Inverse Difference Moment: It is (I hope so) the opposite of contrast . If the co-occurrence matrix values are at the diagonal , the value function of Inverse difference moment is high.

After computing the features , I build a avgEnergy , avgContrast , avgInverse arrays to store distinct values. Then compute the average of these vectors and form a class defining the image training. This training image vector will be used by test image feature vector to determine its relation with its energy contrast etc.

Processing the same steps for the given image is the same as above. You just call the function with an initial parameter -test, and find out co-occurrence matrix , compute the energy , contrast and inverse difference moments , take de average of the values and have the feature vector.

## Classification

In this part of the project was an unsuccessful attempt. After gathering the value which have the similar values I used Euclian distance formula to compute test images feature vector which class it must be in.

The formula is given below:

$$D(x) = \text{test} * \text{class}(i) - \frac{1}{2}(\text{class}(i)^T * \text{class}(i))$$

So I thought that it will get the right classification in order to this formula. As in our text book covers in chapter 9 , decision –theoretic methods , I tested this techniques for proper output but could not have the correct output for that given test image. Especially there are some area which are

clearly not relevant with that image but it marks as it is in that classification.

## Future Work

Maybe we can again check whether the Energy and Contrast really have the information. Take out the partial information from the boundary parts and needs to be re-calculated again.

Leave the dynamic array type (maybe there is some pointer assignment fault).

## Source - Code

```
/* Lütfi İteris ONEY  
  
Bilgisayar Mühendisliği Tezsiz Yüksek Lisans  
  
1305937  
  
HW-2.C  
  
Subject : Basic Convolution Operations  
Subject To: Sibel Tari  
  
*/  
  
#include <stdio.h>  
  
#include <stdlib.h>  
  
#include <string.h>  
  
#include <time.h>  
  
#include <math.h>  
  
//prototypes  
  
void datas (char * , char * , char *);  
  
void test (char * , char * , char *);  
  
//global values  
  
char *filename , *word , *outfilename1 , *outfilename2;  
  
int width=4,height=4,MaxValue,i,arraysize,Max=0,Min , selection;  
  
FILE *f;  
  
int **picture, **noisepicture , **solution , **co_occurence;  
  
double a = 1.0;
```

```

int x=0,y=0,randnumberint=0,randnumberwidth=0,randnumberheigth=0, tempvalue1=0 , tempvalue2=0
,array[8] ={0},array2[24]={0},l;

float percentage=0.0,randnumberfloat=0.0;

int patches=0;

double Contrast[30]={0},avgContrast=0,Entropy[30]={0},avgEntropy=0
,Inverse[30]={0},avgInverse={0},Energy[30]={0},avgEnergy=0;

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

{

if (argc < 4)

{

printf("Parameter Required!.... it's usage is input.ppm output1.ppm dummy.ppm -parameter\n");

}

filename = (char *) malloc(sizeof(char));

word = (char *) malloc(sizeof(char));

outfilename1 = (char *) malloc(sizeof(char));

outfilename2 = (char *) malloc(sizeof(char));

if (!strcmp(argv[4],"-datas"))

grad(argv[1],argv[2],argv[3]);

else if (!strcmp(argv[4],"-test"))

median(argv[1],argv[2],argv[3]);

else

{

printf("Not a valid expression...\n");

}
}

```

```

}

void grad(char *argv1 , char *argv2 , char *argv3)
{
filename = argv1;
outfilename1 = argv2;
outfilename2 = argv3;

if ((f = fopen(filename, "r")) == NULL)
    fprintf(stderr, "Cannot open %s\n", "f");
else
{
printf("File Opened....\n");
fscanf(f,"%s \n",word);
fscanf(f,"%d %d \n",&width,&heighth);
fscanf(f,"%d",&MaxValue);
//malloc for original picture
picture =malloc(heighth * sizeof(int *));
for (i=0 ; i <= heighth ;i++)
{
    picture[i]=malloc(width * sizeof(int));
}
//malloc for solution
solution =malloc(heighth * sizeof(int *));

for (i=0 ; i <= heighth ;i++)
{
    solution[i]=malloc(width * sizeof(int));
}
for(y=0;y<heighth;y++)
    for(x=0;x<width;x++)
        {

```

```

        solution[y][x]=0;

    }

co_occurrence =malloc(heigth * sizeof(int *));

for (i=0 ; i <= heigth ;i++)

{

    co_occurrence[i]=malloc(width * sizeof(int));

}

for(y=0;y<255;y++)

    for(x=0;x<255;x++)

        {

            co_occurrence[y][x]=0;

        }

//reading the original picture

for(y=0;y<heigth;y++)

    for(x=0;x<width;x++)

        {

            fscanf(f,"%d",&picture[y][x]);

            fscanf(f,"\n");

        }

}

fclose(f);

//randnumberwidth = (width * (rand() % 100))/100 ;

//randnumberheigth = (heigth * (rand() % 100))/100;

//Where my program starts

for(patches=0 ; patches < 30 ; patches++)

{

for(y=0;y<255;y++)

    {

        for(x=0;x<255;x++)

            {

                co_occurrence[x][y] = 0;

```

```

}
}

srand((unsigned int) time(NULL));

randnumberwidth = (width * (rand() % 100)) /100;
randnumberheight = (height * (rand() % 100))/100;
for(y=0;y<32;y++)
{
    if (picture[randnumberheight+y][randnumberwidth] > height-2)
        continue;
    else
        for(x=0;x<32;x++)
        {
            if (picture[randnumberheight+y][randnumberwidth+x+1] < width-2)
                co_occurrence
[picture[randnumberheight+y][randnumberwidth+x]][picture[randnumberheight+y][randnumberwidth+x+1]]++;

            else
                continue;;
        }
}

for(y=0;y<255;y++)
{
    for(x=0;x<255;x++)
    {
        Energy[patches] += co_occurrence[y][x] * co_occurrence[y][x];
// Entropy[patches] += (double) co_occurrence[y][x] *
log((double)co_occurrence[y][x]);

        if (y > x)
        {
            Inverse[patches] += (co_occurrence[y][x] * co_occurrence[y][x]) / (y-x);
            Contrast[patches] += (y-x) * (co_occurrence[y][x] * co_occurrence[y][x]);
        }
        else if(x-y !=0){

```

```

Inverse[patches] += (co_occurrence[y][x] * co_occurrence[y][x]) / (x-y);

Contrast[patches] += (x-y) * (co_occurrence[y][x] * co_occurrence[y][x]);

}

}

}

}

for(i=0;i<30;i++)

{

    avgEnergy +=Energy[i];

    avgContrast +=Contrast[i];

    avgInverse +=Inverse[i];

}

avgEnergy = avgEnergy/30.0;

avgEntropy = avgContrast/30.0;

avgInverse = avgInverse/30.0;

printf("%f\n",avgInverse);

printf("%f\n",avgEnergy);

printf("%f\n",avgContrast);

//writing image to a file

if ((f = fopen(outfilename1, "w")) == NULL)

    fprintf(stderr, "Cannot open %s\n", "f");

else

{

    printf("File Opened for writing.....\n");

    fprintf(f, "%f\n",avgEnergy);

    fprintf(f, "%f\n",avgContrast);

    fprintf(f, "%f\n",avgInverse);

    printf("\nFile %s written....\n",outfilename1);

    fclose(f);

}

```

```

}

void test(char *argv1 , char *argv2 , char *argv3)

{

int bigx=0 , bigy=0;

double avgEnergytrainings[5]={0} ,avgContrasttrainings[5]={0},avgInversetrainings[5]={0};

    filename = argv1;

outfilename1 = argv2;

outfilename2 = argv3;

    if ((f = fopen(filename, "r")) == NULL)

        fprintf(stderr, "Cannot open %s\n", "f");

    else

    {

printf("File Opened.....\n");

fscanf(f,"%s \n",word);

fscanf(f,"%d %d \n",&width,&heighth);

fscanf(f,"%d",&MaxValue);

//malloc for original picture

picture =malloc(heighth * sizeof(int *));

for (i=0 ; i <= heighth ;i++)

{

    picture[i]=malloc(width * sizeof(int));

}

//malloc for noise picture

noisepicture =malloc(heighth * sizeof(int *));

for (i=0 ; i <= heighth ;i++)

{

    noisepicture[i]=malloc(width * sizeof(int));

}

//malloc for solution

solution =malloc(heighth * sizeof(int *));

```

```

for (i=0 ; i <= heigth ;i++)
{
    solution[i]=malloc(width * sizeof(int));
}
for(y=0;y<heigth;y++)
    for(x=0;x<width;x++)
    {
        solution[y][x]=0;
    }
//malloc for noise solution
co_occurence =malloc(heigth * sizeof(int *));
for (i=0 ; i <= heigth ;i++)
{
    co_occurence[i]=malloc(width * sizeof(int));
}
for(y=0;y<255;y++)
    for(x=0;x<255;x++)
    {
        co_occurence[y][x]=0;
    }
//reading the original picture
for(y=0;y<heigth;y++)
    for(x=0;x<width;x++)
    {
        fscanf(f,"%d",&picture[y][x]);
        fscanf(f,"\n");
    }
}
fclose(f);
//Where my program starts
for(bigx=32 ; bigx < 255-32 ;bigx++)

```

```

for(bigy=32 ; bigy < 255-32 ;bigy++)
{
for(y=0;y<255;y++)
{
for(x=0;x<255;x++)
{
co_occurence[x][y] = 0;
}
}
srand((unsigned int) time(NULL));
randnumberwidth = (width * (rand() % 100)) /100;
randnumberheigth = (heigth * (rand() % 100))/100;
for(y=0;y<32;y++)
{
if((picture[randnumberheigth+y][randnumberwidth] > heigth-2)
continue;
else
for(x=0;x<32;x++)
{
if((picture[randnumberheigth+y][randnumberwidth+x+1] < width-2)
co_occurence
[picture[randnumberheigth+y][randnumberwidth+x]][picture[randnumberheigth+y][randnumberwidth+x+1]]++;
else
continue;;
}
}
for(y=0;y<255;y++)
{
for(x=0;x<255;x++)
{
Energy[0] += co_occurence[y][x] * co_occurence[y][x];
}
}
}
}

```

```

//          Entropy[patches] += (double) co_occurence[y][x] *
log((double)co_occurence[y][x]);

        if (y > x)

            {

                Inverse[0] += (co_occurence[y][x] * co_occurence[y][x]) / (y-x);

                Contrast[0] += (y-x) * (co_occurence[y][x] * co_occurence[y][x]);

            }

        else if(x-y !=0)

            {

                Inverse[0] += (co_occurence[y][x] * co_occurence[y][x]) / (x-y);

                Contrast[0] += (x-y) * (co_occurence[y][x] * co_occurence[y][x]);

            }

        }

    }

if ((f = fopen("b1.txt", "r")) == NULL)

    fprintf(stderr, "Cannot open %s\n", "f");

else

{

    fscanf(f,"%d\n",avgEnergytrainings[0]);

    fscanf(f,"%d\n",avgContrasttrainings[0]);

    fscanf(f,"%d\n",avgInversetrainings[0]);

fclose(f);

}

if ((f = fopen("b2.txt", "r")) == NULL)

    fprintf(stderr, "Cannot open %s\n", "f");

else

{

    fscanf(f,"%d\n",avgEnergytrainings[1]);

    fscanf(f,"%d\n",avgContrasttrainings[1]);

    fscanf(f,"%d\n",avgInversetrainings[1]);

```

```

fclose(f);
}
if ((f = fopen("b3.txt", "r")) == NULL)
    fprintf(stderr, "Cannot open %s\n", "f");
else
{
    fscanf(f, "%d\n", avgEnergytrainings[2]);
    fscanf(f, "%d\n", avgContrasttrainings[2]);
    fscanf(f, "%d\n", avgInversetrainings[2]);
fclose(f);
}
if ((f = fopen("b4.txt", "r")) == NULL)
    fprintf(stderr, "Cannot open %s\n", "f");
else
{
    fscanf(f, "%d\n", avgEnergytrainings[3]);
    fscanf(f, "%d\n", avgContrasttrainings[3]);
    fscanf(f, "%d\n", avgInversetrainings[3]);
fclose(f);
}
if(Energy[0] > avgEnergytrainings[0])
    Energy[1] = Energy[0] - avgEnergytrainings[0];
else
    Energy[1] = avgEnergytrainings[0] - Energy[0];
}
}

```