Görüntü

İşleme

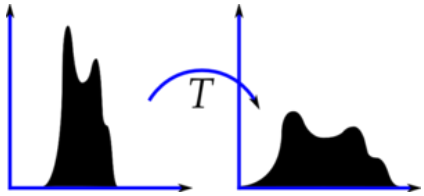
Ödev 1

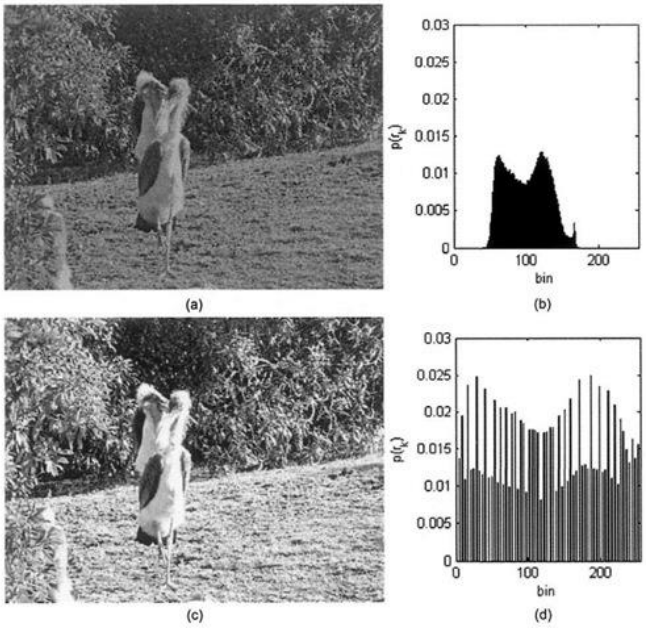
Gri resimlere Histogram Eşleme işlemi uygulanması

Sardor HAZRATOV

14011901

Yöntem: Histogram Eşleme bir resimdeki renk değerlerinin belli bir yerde kümelenmiş olmasından kaynaklanan renk dağılımı bozukluğunu gidermek için kullanılan bir yöntem.





Her pixel belli bir sayısal değere sahipdir ve o değer grinin tonlarıdır (veya rengin, RGB ise üç kanaldan oluşur). 0-sıyah ve 255-beyaz olacak şekilde en 256 ton vardır. Yalnız bu resime göre değişiklik gösterebilir. Örneğin, resim sadece 7 tontan da oluşabilir. 256 ton bu maximum değerdir.

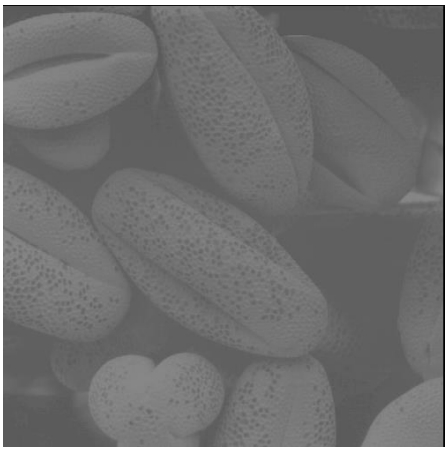
İşlem resmin pixellerinin taranmasından başlar. Değer aralığımız 0..255 ise her bir değerden kaç pixel olduğu hesaplanır ve bir pixel değerinin resimde bulunma olasılığını hesaplanır. Ardından gri tonların normalizasyonu yapılır. En önemli aşaması da son yapacağımız işlem, yani yeni değerleri belirlemek. Bunun için gri değerin ve kendisinden önceki değerlerin resimde bulunma olasılığını kümülatif toplanılır. Elde ettiğimiz değer hangi normalizasyon yapılan değere yakınsa yeni değerimiz o dur.

Örnek: 64x64 resim, Gri ton aralığı 0..7

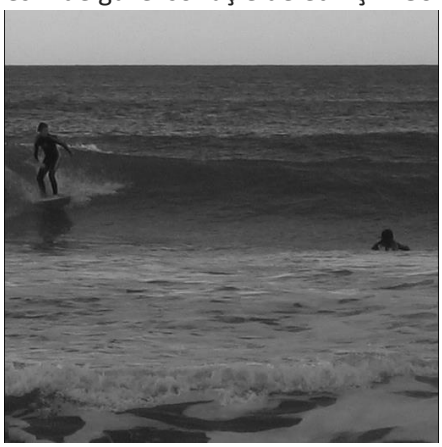
Gri Değer	Normalizasyon Değer	Resimdeki sayısı	Bulunma olasılığı	Yeni Değer
0	0/7=0	790	0.19	1
1	1/7=0.14	1023	0.25	3
2	2/7=0.29	850	0.21	5
3	3/7=0.43	656	0.16	6
4	4/7=0.57	329	0.08	6
5	5/7=0.71	245	0.06	7
6	6/7=0.86	122	0.03	7
7	7/7=1	81	0.02	7

Uygulama:

Bu resimdeki değişiklik resime contrast işlemi yapılmış gibi gözüküyor. Parlaklık artmış.



Bu resimde güzel sonuç elde etmişiz. Görüntü net ve doygun.



Bu resimde ise Histogram Eşleme işlemi sonucu 1. Resimdeki gibi
Ve aşağidaki resimde Histogram Eşleme çok kötü çıkmış. Görüntü biraz bulanık.

Sonuç: Histogram Eşleme işlemi resimlerin kalitesini ve görünebilirliğini arttıran işlemdir. Ancak bu yöntem, her resmi iyileştirmez. Yalnızca tüm pixellerin belli araklıkta renk değerine sahip olduğu resimlerde etkilidir.

Program kodu:

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

//void histogramEqu(unsigned char**,int,int,int,unsigned char*);
//void readImageHeadings(FILE *,char *,int *,int *,int *);

int main(){
   FILE *inputImage;
   FILE *outputImage;
   unsigned char **imageMatrix;
   unsigned char *newValues;
   int imageHeight=0;
   int imageWidth=0;
   int grayLevel=0;
   char pgmFormat[2]={0};
   char c;
   int i;
   int j;

   /** Opening image**/
   if((inputImage=fopen("boat256.pgm","rb"))==NULL){
      printf("\nCould not open file(image)...\n");
      return -1;
   }
   /** Creating output image**/
   if((outputImage=fopen("boat256_1.pgm","w"))==NULL){
      printf("\nCould not create file(image)...\n");
      return -2;
   }
   printf("\nImage loaded successfully...\n");

   /**
      Function reads headings from source image
   **/
   readImageHeadings(inputImage,&pgmFormat,&imageWidth,&imageHeight,&grayLevel);

   printf("\nPGM format:%s\n",pgmFormat);
```

```
printf("Image Size: %dx%d\n",imageWidth,imageHeight);
printf("Gray Level: %d\n",grayLevel);

/** Allocating memory for array of new Values after histogram equalization **/
newValues=(unsigned char*)malloc(grayLevel*sizeof(unsigned char));
if(newValues==NULL){
   printf("\nCould not allocate memory!\n");
   fclose(inputImage);
   fclose(outputImage);
   return -3;
}
/*Resetting to zero*/
for(i=0;i<=grayLevel;i++){
   newValues[i]=0;
}
/** Allocating memory for source image {pixels}**/
/* controls whether PGM with format P5 (binary version)*/
if(pgmFormat[1]=='5'){
   imageMatrix=(unsigned char**)malloc(imageHeight*sizeof(unsigned char*));
   if(imageMatrix==NULL){
      printf("\nCould not allocate memory!\n");
      free(newValues);
      fclose(inputImage);
      fclose(outputImage);
      return -4;
   }
   for(i=0;i<=imageHeight;i++){
      imageMatrix[i]=(unsigned char*)malloc(imageWidth*sizeof(unsigned char));
   }
   printf("\nMemory allocated successfully...\n");
}
else{
   printf("\nThis Program can parse only pgm images with version %s\n",pgmFormat);
   free(newValues);
   fclose(inputImage);
   fclose(outputImage);
   return -5;
}

/* Writing headings to output image */
fputc(pgmFormat[0],outputImage);
//fputc(pgmFormat[1],outputImage);
fputc('2',outputImage);  //for ascii encoding
fputc('\n',outputImage);
fprintf(outputImage,"%d",imageWidth);
```

```
   fputc(' ',outputImage);
   fprintf(outputImage,"%d",imageHeight);
   fputc('\n',outputImage);
   fprintf(outputImage,"%d",grayLevel);
   fputc('\n',outputImage);

   /**0 Start reading pixels from source image 0**/
   i=0;
   while(!feof(inputImage)){
        j=0;
        while(j<imageWidth){
             imageMatrix[i][j]=fgetc(inputImage);
             j++;
        }
       i++;
   }

   /**
      Function makes histogram equalization
   */
   histogramEqu(imageMatrix,imageHeight,imageWidth,grayLevel,newValues);

   /* Writing new values for pixels to output image */
   for(i=0;i<imageHeight;i++){
      for(j=0;j<imageWidth;j++){
         fprintf(outputImage,"%d ",newValues[imageMatrix[i][j]]); //for ascii encoding
         //fputc(newValues[imageMatrix[i][j]],outputImage);
      }
   }

   printf("\nEnded\n");
   //system("pause");
   return;
   free(newValues);
   free(imageMatrix);
   fclose(inputImage);
   fclose(outputImage);
}

/** Reads header data form image: width, height, gray level
*** Writes them to reference values
**/
void readImageHeadings(FILE *image,char *pgmFormat,int *imageWidth1,int *imageHeight1,int
*grayLevel1){
   int imageHeight=0;
```

```
   int imageWidth=0;
   int grayLevel=0;
   char commentStr[100];

   fscanf(image,"%s",pgmFormat);
   fgetc(image);//new line
   if((fgetc(image))=='#'){
      //printf("Comment detected: #");
      while((fgetc(image))!='\n'){
         fscanf(image,"%s",&commentStr);
         //printf("%s ",commentStr);
      }
   }
   else{
      fseek(image,-1,SEEK_CUR);
   }
   fscanf(image,"%d",&imageWidth);
   fscanf(image,"%d",&imageHeight);
   fgetc(image);//new line
   fscanf(image,"%d",&grayLevel);
   fgetc(image);//new line
   /* Writing to reference values */
   *imageWidth1=imageWidth;
   *imageHeight1=imageHeight;
   *grayLevel1=grayLevel;

   printf("\nReading image headers completed...\n");
}

/** Makes Histogram Equalization for image
**/
void histogramEqu(char** image, int imageHeight,int imageWidth,int grayLevel,unsigned char
newValues[]){
   long *levelsArray;
   float *probArray;
   float *rkArray;
   int i;
   int j;
   long n;
   unsigned char min;
   unsigned char max;
   float sum=0;

   /**
   levelsArray[] is an array where number of each gray level stored
```

```
*/
levelsArray=(long*)malloc(grayLevel*sizeof(long));
if(levelsArray==NULL){
   printf("There is a problem with allocating memory!\n");
   return;
}
/*Resetting to zero*/
for(i=0;i<grayLevel;i++){
   levelsArray[i]=0;
}
/*Start counting number of each level*/
for(i=0;i<imageHeight;i++){
   for(j=0;j<imageWidth;j++){
      levelsArray[image[i][j]]=levelsArray[image[i][j]]+1;
   }
}

/** Probability of each gray levels in source image */
probArray=(float*)malloc(grayLevel*sizeof(float));
if(probArray==NULL){
   printf("There is a problem with allocating memory!\n");
   return;
}
/* n is the total number of pixels in source image*/
n=imageHeight*imageWidth;

/** Calculating probabilities*/
for(i=0;i<=grayLevel;i++){
   probArray[i]=(float)levelsArray[i]/n;
}

/**
 rkArray[] is normalized grayLevels
*/
rkArray=(float*)malloc(grayLevel*sizeof(float));
if(rkArray==NULL){
   printf("There is a problem with allocating memory!\n");
   return;
}
for(i=0;i<=grayLevel;i++){
   rkArray[i]=(float)i/grayLevel;
}

/*Start calculating new values for pixels*/
for(i=0;i<=grayLevel;i++){
```

```
        sum=0;
        for(j=0;j<i;j++){
            sum+=probArray[j];
        }
        max=0;
        while(sum>=rkArray[max] && max<=grayLevel){
            max++;
        }
        //printf("sum:%f rkArrayMin:%f\n",sum,rkArray[min]);
        min=max-1;
        //printf("sum:%f rkArrayMax:%f\n",sum,rkArray[max]);
        if((sum-rkArray[min])>(rkArray[max]-sum)){
            newValues[i]=max;
            //printf("*max**Old value: %d [%d]| new Value:%d **\n",i,newValues[i],max);
        }
        else{
            newValues[i]=min;
            //printf("*min**Old value: %d [%d]| new Value:%d **\n",i,newValues[i],min);
        }
    }
    printf("Returning from histogramEqualization func...\n");
}
```