Tarea 3

Angel Manrique Pozos Flores; N.C M07211505
Instituto Tecnologico Nacional de Mexico, Blvd. Industrial, Mesa de Otay, 22430 Tijuana, B.C., Mexico.
(Dated: 5 de Febrero del 2016)

En el presente trabajo se implementa un algoritmo para binarizar una imagen, para aumentar el contraste y para la ecualizacion del histograma.

I. INTRODUCCIÓN

El procesamiento de imagenes es de gran utilidad en la mayoria de las areas de investigacion ya que todo lo que captamos en el mundo la gran mayoria de los datos provienen de nuestros sentidos en especial la vista.

Por ello el estudio de la vision computacional es de gran importancia, el presente trabajo se hace un acercamiento a esta area, utilizando las librerias de vision open source OpenCV y el software libre CodeBlocks el cual se configuro para que pudiera ser capaz de reconocer las librerias de OpenCV.

II. THRESHOLD

El thresholding es un metodo de segmentacion simple, la aplicacion que tiene es el separar en distintas regiones una imagen con el fin de poder analizarla, esta separacion se basa en la variacion de la intensidad entre los pixeles del objeto y los pixeles del fondo.

Para poder diferenciar los pixeles que nos interesan del resto de estos, se realiza una comparacion entre los valores de intensidad asociados al pixel respecto a un threshold el cual se determina de acuerdo a las necesidades del problema que se este analizando.

Una vez separados se les asigna un determinado valor con el fin de poder identificarlos por ejemplo cuando se hace una binarizacion y los valores correspondientes a nuestros pixeles se localizan entre los rangos de 0 y 255.

Figura 1: imagen con un thresholding aplicado.

Dentro de las librerias de OpenCV existe la funcion *"threshold"* la cual nos permite desarollar estas operaciones.

Donde podemos efectuar 5 tipos distintos de thresholding con esta funcion.

II.1. Fixed thresholding

Si consideramos que tenemos una imagen con valores de intensidad en sus pixeles $src(x, y)$ donde la linea horizontal mostrada en la Figura 2 representa el threshold (thresh fixed).

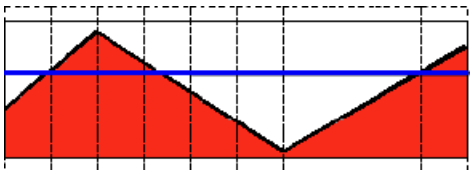

Figura 2: imagen representativa con un thresholding *fixed* aplicado.

II.2. Threshold binary

Esta operacion de thresholding puede expresarse matematicamente como:

$$dst(x, y) = \begin{cases} maxVal & \text{if src(x,y)} > \text{thresh} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Donde si la intensidad del pixel es $src(x, y)$ es mucho mayor que $thresh$, entonces la intensidad del pixel sera puesta a $maxVal$ de lo contrario sera puesta a 0.

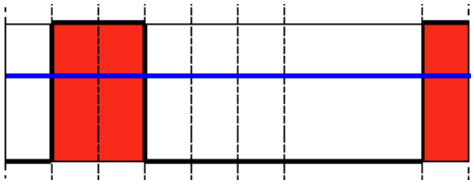

Figura 3: imagen representativa con un thresholding *binary* aplicado.

II.3. Threshold binary, inverted

Este thresholding puede expresarse como:

$$dst(x,y) = \begin{cases} 0 & \text{if src(x,y)} > \text{thresh} \\ maxVal & \text{otherwise} \end{cases} \quad (2)$$

Si la intensidad del pixel *src(x,y)* es mayor que *thresh* entonces el nuevo pixel sera puesto a 0, de lo contrario se asigana el valor a *maxVal*.

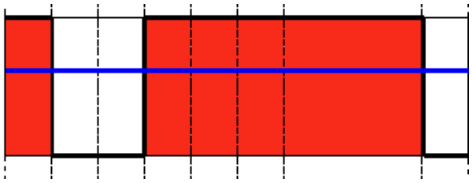

Figura 4: imagen representativa con un thresholding *binary inverted* aplicado.

II.4. Truncate

Esta operacion de thresholding puede expresarse como:

$$dst(x,y) = \begin{cases} threshold & \text{if src(x,y)} > \text{thresh} \\ src(x,y) & \text{otherwise} \end{cases} \quad (3)$$

Donde la intensidad maxima para los valores de los pixeles es *thresh*, si *src(x,y)* es mayor, entonces su valor es truncado como se muestra en la Figura 5.

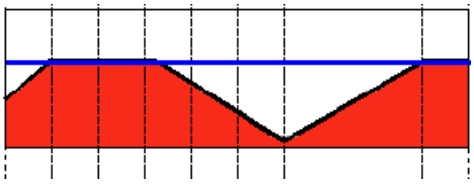

Figura 5: imagen representativa con un thresholding *truncated* aplicado.

II.5. Threshold to Zero

Esta operacion puede ser expresada como:

$$dst(x,y) = \begin{cases} src(x,y) & \text{if src(x,y)} > \text{thresh} \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

si *src(x,y)* es menor que *tresh* entonces el nuevo valor del pixel es puesto a 0.

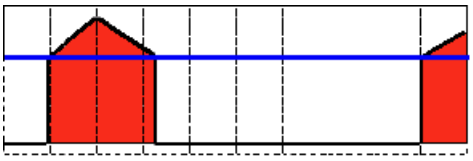

Figura 6: imagen representativa con un thresholding *truncated* aplicado.

II.6. Threshold to Zero, inverted

Esta operacion puede ser expresada como:

$$dst(x,y) = \begin{cases} 0 & \text{if src(x,y)} > \text{thresh} \\ src(x,y) & \text{otherwise} \end{cases} \quad (5)$$

si *src(x,y)* es mayor que *thresh*, el nuevo valor del pixel es puesto a 0.

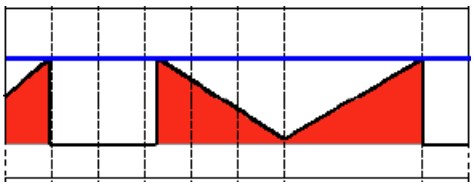

Figura 7: imagen representativa con un thresholding *truncated* aplicado.

II.7. Desarrollo

Se utilizo la imagen de lena para hacer las pruebas requeridas en la practica esta se muestra a continuacion.

Figura 8: Imagen lena utilizada comunmente en pruebas de imagen en opencv.

En esta ocasion se requiere aplicarle a la imagen una binarizacion asi como los distintos tipos de threshol-

ding, por ello se desarrollo un codigo en C++ con las librerias de vision de OpenCV para la realizacion de la practica, el siguiente codigo se muestra a continuacion.

```
#include <stdlib.h>
#include <cv.hpp>
#include <cxcore.hpp>
#include <highgui.h>
#include <iostream>

using namespace std;
using namespace cv;

int threshold_value = 0;
int threshold_type = 3;
int const max_value = 255;
int const max_type = 4;
int const max_BINARY_value = 255;
Mat img, img_gray, dst;


void Threshold(int, void*);

int main()
{

cout << "Threshold Types: \n 0: Binary \
    n 1: Binary Inverted \n 2: Truncate \
    n 3: To Zero \n 4: To Zero Inverted"
    << endl;
cout << "\n Press ENTER over the image
    to Exit \n" << endl;

  img = imread("1.jpg", 1);
  namedWindow("Original Image",
      CV_WINDOW_AUTOSIZE);
  imshow("Original Image", img);

  cvtColor(img, img_gray, COLOR_RGB2GRAY
      );
  namedWindow("Threshold Window",
      WINDOW_AUTOSIZE);
  //imshow("Gray Image", img_gray);

  // Create Trackbar to choose type of
      Threshold
  createTrackbar("Threshold: ", "
      Threshold Window",
                 &threshold_type,
                     max_type, Threshold);
  createTrackbar("Value: ", "Threshold
      Window",
                 &threshold_value,
                     max_value, Threshold)
                     ;

  // Call the function to initialize
  Threshold( 0, 0 );

waitKey(0);
return 0;
}


void Threshold( int, void* )
{
  /* 0: Binary
     1: Binary Inverted
     2: Threshold Truncated
     3: Threshold to Zero
     4: Threshold to Zero Inverted
   */

  threshold(img_gray, dst,
      threshold_value, max_BINARY_value,
      threshold_type);

  imshow("Threshold Window", dst );
}
```

Obteniendo los siguientes datos:

III. MODIFICACION DEL CONTRASTE DE MANERA UNIFORME.

Se desarrollo el siguiente codigo para modificar el contraste de manera no uniforme.

```
#include <stdlib.h>
#include <cv.hpp>
#include <cxcore.hpp>
#include <highgui.h>
#include <iostream>

using namespace std;
using namespace cv;

int main()

    {

    Mat img;
    img = imread("lena.jpg",
        CV_LOAD_IMAGE_COLOR);

    double alpha = 1;
    int beta = 90;


        for (int i = 200; i < 300; i++)

            {

            for (int j = 200; j < 400; j
                ++)
```

(a) Threshold binary.

(b) Threshold binary, inverted.

(c) Truncated.

Figura 9: Imagenes con thresholding, binario, binario invertido y truncado.

(a) Threshold to Zero.

(b) Threshold to Zero, inverted.

Figura 10: Imagenes con thresholding, a zero, a zero invertido

```
{
                    for (int k = 0; k <
                        3; k++)
                    {
                    img.at<Vec3b>(i, j)[k
                        ] = saturate_cast<
                        uchar>(alpha*(img.
                        at<Vec3b>(i, j)[k
                        ]) + beta);
                    }
                }
            }
        imshow("Image", img);
```

```
    waitKey(0);
    return 0;
    }
```

En la Figura 11 se muestra la imagen obtenida del codigo desarrollado.

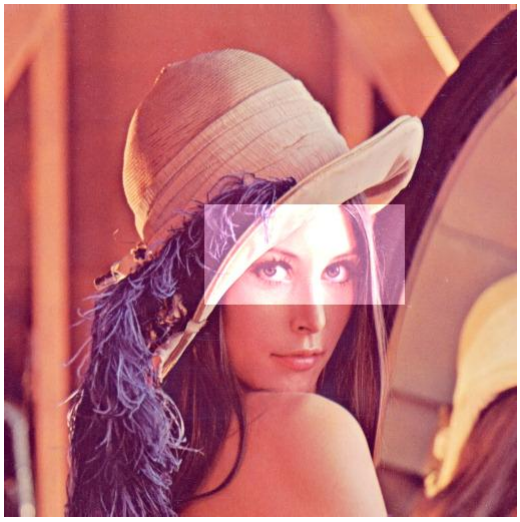

Figura 11: Imagen con un valor de brillo ($alpha = 1$) y contraste ($beta = 90$).

IV. ECUALIZACION DEL HISTOGRAMA

Y por ultimo se muestra la funcion de OpenCV para equalizar un histograma, para ello fue desarrollado el siguiente codigo.

```
#include <stdlib.h>
#include <cv.hpp>
#include <cxcore.hpp>
#include <highgui.h>
#include <iostream>

using namespace std;
using namespace cv;


int main()

{

    Mat img, gray, dst;
    char* source_window = "Source Image"
        ;
    char* equalized_window = "Equalized
        Image";
    img = imread("lena.jpg",
        CV_LOAD_IMAGE_COLOR);
    cvtColor(img, gray, CV_BGR2GRAY);
    equalizeHist(gray, dst);

    namedWindow(source_window,
        CV_WINDOW_AUTOSIZE);
    namedWindow(equalized_window,
        CV_WINDOW_AUTOSIZE);

    imshow(source_window, gray);
    imshow(equalized_window, dst);

    waitKey(0);
    return 0;

}
```

(a) Imagen lena sin equalizar el histograma.

(b) Imagen de lena con el histograma equalizado.

Figura 12: Imagen con un valor de brillo ($alpha = 1$) y contraste ($beta = 90$).

V. BIBLIOGRAFÍA

- OpenCV, Basic Thresholding Operations. (*http* : *//tinyurl.com/gmssgbs*)
- OpenCV, Changing the contrast and brightness of an image. (*http* : *//tinyurl.com/h7nphcg*)
- OpenCV, Histogram Equalization. (*http* : *//tinyurl.com/gl7yeqw*)