Práctica 4: Spatial Transformation

Visión por Computadora

Elaborado por:
José Baltazar Ramírez Rodríguez

Dra María Susana Ávila García

26 de febrero del 2019

I. DESCRIPCIÓN DEL PROBLEMA

La transformación espacial define una relación entre cada punto de la imagen de entrada y la imagen de salida, aplicándoles una ecuación con parámetros distintos para obtener un cambio en la imagen de entrada. Puede ser aplicada una transformación lineal a una imagen para obtener su negativo, así como funciones logarítmicas y de potencia. Alguna de estas aplicaciones puede tener fin médico. El ejercicio consiste en implementar estas funciones aplicadas a una imagen en color.

II. ALGORITMO UTILIZADO PARA RESOLVER EL PROBLEMA

El algoritmo que se utilizó para resolver estos ejercicios consiste en cargar la imagen a una matriz. Esta será nuestra imagen de entrada.

```
// Leer la imagen
Mat imagen = imread("C:/Users/HOLA/Desktop/Balta/Visual Studio/HolaMundo/

if (imagen.empty()) // Verificar que se haya cargado la imagen
{
    cout << "No se pudo abrir o encontrar la imagen." << endl;
    system("pause");
    return -1;
}

String windowName = "My HelloWorld Window"; //Nombre de la ventana
namedWindow(windowName, WINDOW_AUTOSIZE); // Crear la ventana
imshow(windowName, imagen); // Mostrar la imagen dentro de la ventana
waitKey(0);
//destroyWindow(windowName); //destruir la ventana creada
```

Se comienza por obtener su número de filas y columnas de esta imagen de entrada para construir las nuevas matrices de salida, a las que se aplicará la transformación espacial. Estos pasos se muestran en la siguiente figura

```
int filas = imagen.rows;
int columnas = imagen.cols;
Mat matrizGrises = Mat::zeros(filas, columnas, CV_64FC1);
cvtColor(imagen, matrizGrises, COLOR_BGR2GRAY); //Para convertir la imagen a escala de grises
Mat matrizGrises1 = Mat::zeros(filas, columnas, CV_64FC1);
cvtColor(imagen, matrizGrises1, COLOR_BGR2GRAY);    //Para convertir la imagen a escala de grises
Mat matrizGrises2 = Mat::zeros(filas, columnas, CV_64FC1);
cvtColor(imagen, matrizGrises2, COLOR_BGR2GRAY);    //Para convertir la imagen a escala de grises
```

Se decide convertir estas matrices a escala de grises para trabajar con un solo canal y trabajar de una manera más clara y sencilla.

La función **linear transformation** lo que realiza es un inverso de los valores de la imagen (matriz) en cada uno de sus pixeles. Es necesario definir los valores de la ecuación a utilizar que es:

$$s = L - 1 - r$$

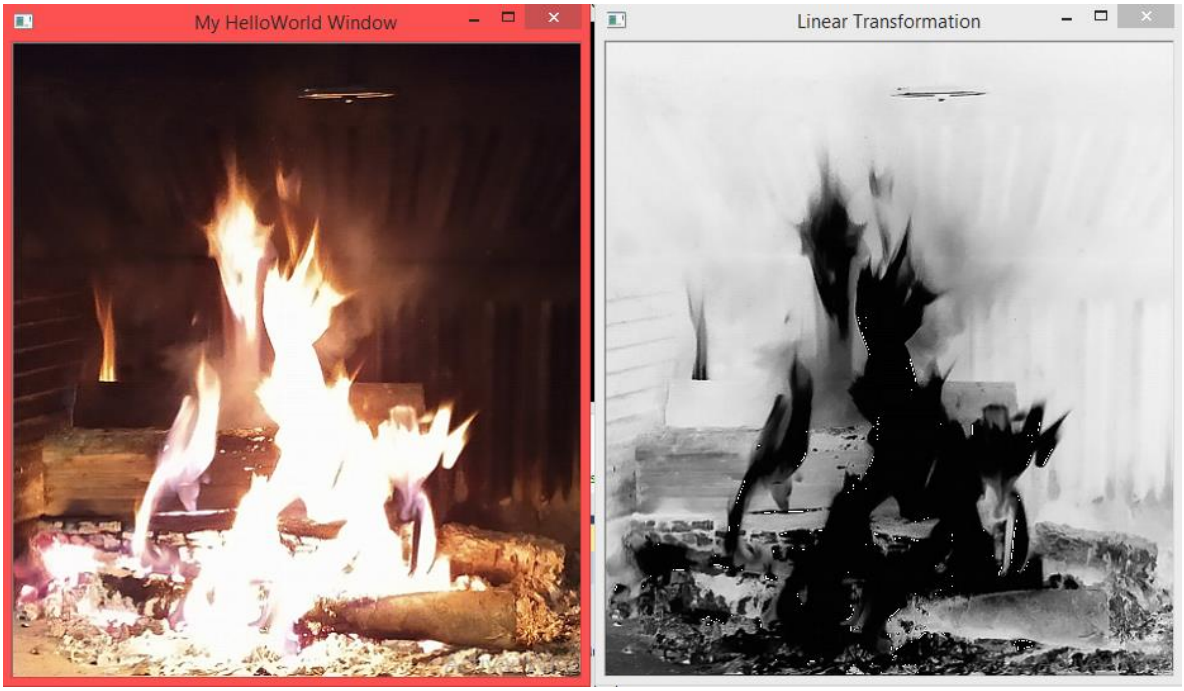
Donde **r** es la intensidad de entrada, **s** es la intensidad de salida y **L-1** es el valor máximo de **r**.

```
double L = 255; //Maximo valor de r
double s = 0;   //intensidad de salida

//Para Linear Transformation
for (int x = 0; x < matrizGrises.rows; x++) {
    for (int y = 0; y < matrizGrises.cols; y++) {
        Scalar intensity = matrizGrises.at<uchar>(x, y);
        double r = intensity.val[0];
        s = L - 1 - r;
        matrizGrises.at<uchar>(x, y) = s;
    }
}

String windowName1 = "Linear Transformation"; //Nombre de la ventana
namedWindow(windowName1, WINDOW_AUTOSIZE); // Crear la ventana
imshow(windowName1, matrizGrises); // Mostrar la imagen dentro de la ventana
waitKey(0);
//destroyWindow(windowName);//destruir la ventana creada
```

Se define L como 255 y se declara r. Se recorre la matriz de acuerdo a la cantidad de filas y columnas. Después se define un Scalar llamado *intensidad* para obtener el valor de cada posición que se va recorriendo con los ciclos anidados. Posteriormente se asigna como r el valor de esta intensidad, la cual será la intensidad de entrada y se realiza la operación. Por último se asigna a la matriz de grises que había creado el valor de la ecuación y se muestra esta imagen ya con las operaciones realizadas. El resultado es el siguiente:



La imagen primero se convirtió a escala de grises y sobre esa, se operó.

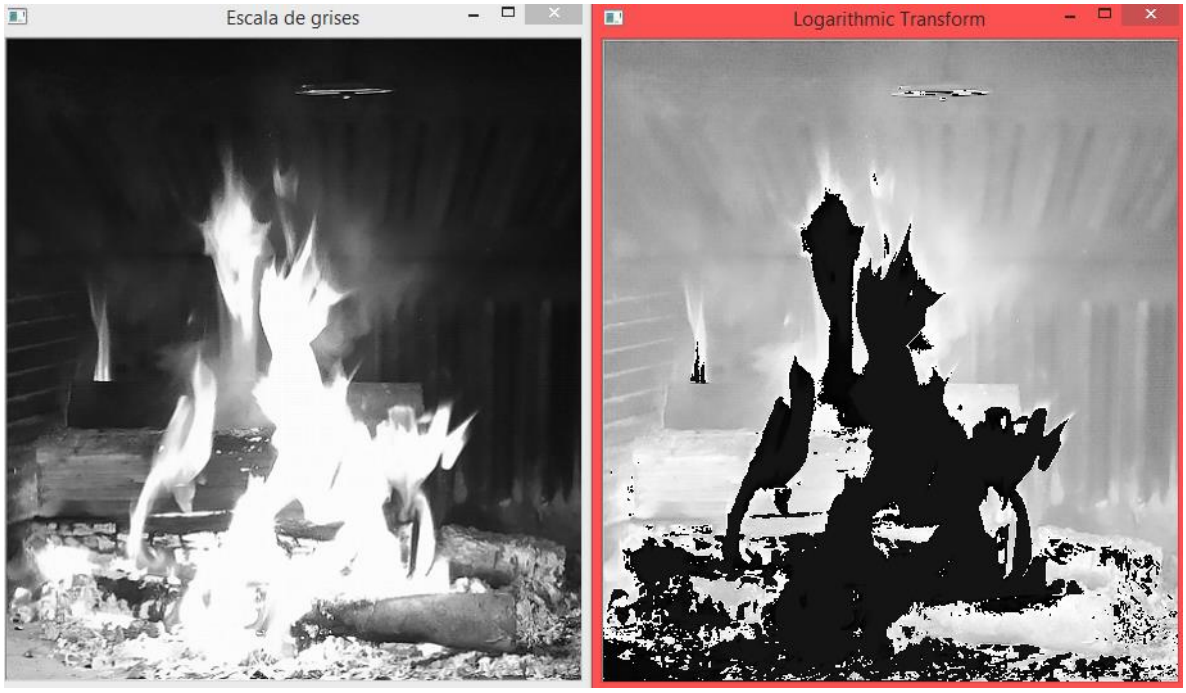
La siguiente función en ser utilizada fue **logarithm transform**. Al igual que para la transformación lineal se definió una matriz en escala de grises y se operó sobre ella.

Se utiliza una **c** igual a 50 y simplemente se cambia la fórmula en la función. Y se muestra la salida. La fórmula a seguir es $s = c * \log(1 + r)$;

```
double c = 50;
//Logarithmic Transform
for (int x = 0; x < matrizGrises1.rows; x++) {
    for (int y = 0; y < matrizGrises1.cols; y++) {
        Scalar intensity = matrizGrises1.at<uchar>(x, y);
        double r = intensity.val[0];
        s = c * log(1 + r);
        matrizGrises1.at<uchar>(x, y) = s;
    }
}

String windowName3 = "Logarithmic Transform"; //Nombre de la ventana
namedWindow(windowName3, WINDOW_AUTOSIZE); // Crear la ventana
imshow(windowName3, matrizGrises1); // Mostrar la imagen dentro de la ventana
waitKey(0);
//destroyWindow(windowName);//destruir la ventana creada
```

La imagen de salida que obtuve fue la siguiente:



El resultado que se obtuvo fue correcto, exceptuando la parte central de la imagen. Duda que será resuelta en el salón de clases.

La última operación fue **power law transform**. Los valores que se definen para la ecuación $s = c * r^{\beta}$ donde c es una cte y beta otra constante. Los primeros valores que se definen son c = 200 y beta = 0.5;

Se sigue el mismo comportamiento que en lo anterior. Definiendo una matriz para asignar los nuevos valores. El resultado fue el siguiente:

```
//Power Law Transform
double gamma = 2.5;
c = 0.001;
for (int x = 0; x < matrizGrises2.rows; x++) {
    for (int y = 0; y < matrizGrises2.cols; y++) {
        Scalar intensity = matrizGrises2.at<uchar>(x, y);
        double r = intensity.val[0];
        s = c * pow(r, gamma);
        matrizGrises2.at<uchar>(x, y) = s;
    }
}

String windowName4 = "Power Law Transform"; //Nombre de la ventana
namedWindow(windowName4, WINDOW_AUTOSIZE); // Crear la ventana
imshow(windowName4, matrizGrises2); // Mostrar la imagen dentro de la ventana
waitKey(0);
//destroyWindow(windowName);//destruir la ventana creada


return 0;
```