

Informe De Implementacion

Parcial 2

Manuela Gutiérrez Rodríguez

Daniela Andrea Gallego Díaz

Departamento de Ingeniería Electrónica y
Telecomunicaciones
Universidad de Antioquia
Medellín
26 de Septiembre de 2021

Índice

1. Clases implementadas	2
1.1. Función submuestreo:	2
1.2. Funcion sobremuestreo:	2
2. Esquema de clases implementadas	3
2.1. Función submuestreo:	3
2.2. Funcion sobremuestreo:	3
3. Modulos de codigo	4
4. Estructura del circuito	6
5. Problemas presentados	6

1. Clases implementadas

Entre las clases que implementamos para la solución de este problema fueron:

1.1. Función submuestreo:

Esta clase cuenta con dos atributos, el primero es un vector llamado `matrizRGB` que se encarga de almacenar los valores relacionados al RED, GREEN Y BLUE que contiene cada pixel de la imagen leída; el segundo atributo es un vector de vectores llamado `matrizPíxeles` que se encarga de almacenar el valor medio de un grupo de píxeles, guardando en cada una de sus posiciones un vector que contiene valores de RED, GREEN Y BLUE. A su vez, esta clase cuenta con cuatro métodos, el primero que es su constructor por defecto que recibe como parámetro de entrada una imagen de tipo `QImage`, en el constructor se realiza todo el proceso de submuestreo que se almacena en `matrizPíxeles`, su segundo método es una sobrecarga del constructor que recibe una imagen de tipo `QImage` y un entero que indica el método a utilizar, este constructor solo se utiliza cuando la imagen necesita submuestreo en una única dimensión, al igual que el anterior constructor almacena los píxeles luego del submuestreo en `matrizPíxeles`, su tercer método llamado `GuardarTxt` se encarga de almacenar los valores de `matrizPíxeles` en un formato específico en un archivo `.txt` para comodidad del usuario a la hora de trasladar estos valores a la plataforma tinkercad, el cuarto y último método consiste en una función que va a permitir retornar el contenido de `matrizPíxeles`.

1.2. Función sobremuestreo:

En esta función se implementan cinco métodos, el constructor por defecto el cual tiene la función de recibir la ruta escrita por el usuario y almacenarla en un atributo privado de la clase para así poder leer la imagen. También un método llamado `lectura`, que se encarga de leer cada pixel de la imagen de acuerdo al RGB y almacena cada valor en un vector de vectores (atributos privados de la clase), uno para los píxeles rojos, otro para los verdes y por último los azules. Un método sobrecargado llamado `Lectura` que recibe un vector de vectores que contiene la matriz de la imagen ya submuestreada y lo almacena en las matrices según el RGB (atributos privados). Otro método implementado fue el de `redimension`, allí se hace el proceso de redimensión de la imagen donde se itera sobre las matrices RGB y se van añadiendo píxeles teniendo en cuenta las dimensiones originales y cuantos píxeles se deben agregar para llegar a una matriz de 12x12, en este mismo método almacenamos la matriz final ya redimensionada en un archivo `.txt` para así utilizarlo como recurso para la implementación en la plataforma de Tinkercard. Por último un método sobrecargado llamado `Redimension` que hace el proceso de redimensión en filas o columnas según las dimensiones de la imagen y la matriz final la almacena en un `.txt`.

2. Esquema de clases implementadas

2.1. Función submuestreo:

- Atributos privados de la clase:

Vector matrizRGB

Vector de vectores matrizPíxeles

- Metodos de la clase:

Submuestreo(QImage): permite tomar el valor medio de un conjunto de píxeles, determinado por el tamaño de la imagen y como ésta debe dividirse para encajar en la matriz de leds de 12x12. El proceso que se realiza parte desde el origen y toma un punto aledaño en el eje y y en el eje x, para hallar el píxel que se encuentra en medio de estos 3, de esta forma se reduce la cantidad de píxeles que tiene la imagen.

Submuestreo(QImage,int): según el tamaño de la imagen permite redimensionarla en el eje x o y, hallando el valor del píxel medio del división del eje a submuestrear. Este proceso inicia en el origen del eje a submuestrear y toma dos puntos en este y toma el píxel que se encuentre en medio de los dos.

GuardarTxt: a partir de el vector de vectores matrizPíxeles, almacena sus valores en un formato adecuado para copiar el contenido almacenado en el archivo ImagenPíxeles y pegarlo adecuadamente en tinkercad.

getMatrizPíxeles: retorna el vector matrizPíxeles.

2.2. Funcion sobremuestreo:

- Atributos privados de la clase:

Variable string llamada ruta (almacena la ruta donde se encuentra la imagen seleccionada por el usuario)

Un vector que contiene vectores, estos contienen las matrices segun el RGB (tres matrices en total por cada color)

- Metodos de la clase:

Sobremuestreo: Constructor por defecto, recibe un string el cual le da al atributo ruta la direccion que escribe el usuario, donde se encuentra la imagen a procesar.

Lectura: Metodo que lee la imagen y almacena los valores de cada píxel en vectores que contienen vectores, cada uno segun el RGB.

Lectura(vector|vector|int,int,int): Metodo sobrecargado que recibe tres parametros, una matriz, y dos enteros que corresponden al ancho y alto de la imagen, con estos datos llenamos las matrices de píxeles rojos,verdes y azules para seguir con el proceso de redimension.

Redimension:Metodo que itera sobre cada una de las matrices RGB y va almacenando y agregando pixeles de acuerdo a las dimensiones de la imagen original para asi, en un archivo .txt tener todos los valores correspondientes con dimension 12x12.

Redimension(int):Metodo sobrecargado que recibe un entero que indica si hay que sobremuestrear las filas o columnas y hace el proceso de redimension para luego almacenar la matriz final de 12x12 en el archivo .txt.

3. Modulos de codigo

- Interacción de la clase QImage con la clase submuestreo y sobremuestreo:

```
QImage imagen(ruta.c_str());
if(imagen.width() != 12 and imagen.height() != 12) -relacion del tamaño de
la imagen con el método a implementar
    Sobremuestreo uno(ruta); -implementación sobremuestreo
    uno.Lectura();
    uno.Redimension();

else if((imagen.width() != 12 and imagen.height() != 12))
    Submuestreo dos(imagen); -implementacion submuestreo
    dos.GuardarTxt();
```

- Interacción de la clase QImage con la clase submuestreo:

anchoProm=imagen.width()/12; -tamaño en x en la que se debe dividir la imagen, se usa el método width() de la clase QImage

altoProm=imagen.height()/12; -tamaño en y en la que se debe dividir la imagen, se usa el método height() de la clase QImage

for(int y=0; y+altoProm != imagen.height(); y+=altoProm) -ciclo en relacion al alto de la imagen

yttotal=(2*y+altoProm)/2;

for(int x=0; x+anchoProm != imagen.width(); x+=anchoProm) -ciclo en relacion al ancho de la imagen

xttotal=(2*x+anchoProm)/2;

matrizRGB.push_back(imagen.pixelColor(xtotal,ytotal).red()); --método de la clase

QImage para obtener el valor en RED

matrizRGB.push_back(imagen.pixelColor(xtotal,ytotal).green()); --

método de la clase QImage para obtener el valor en GREEN

matrizRGB.push_back(imagen.pixelColor(xtotal,ytotal).blue()); --método de la clase

QImage para obtener el valor en BLUE

```
matrizPixeles.push_back(matrizRGB);  
matrizRGB.clear();
```

- Interacción de la clase QImage con la clase sobremuestreo:

for (int fila=0; fila<imagen.height(); fila++) -se usa el método height() de la clase QImage

filas=vacio;

for(int columna=0; columna<imagen.height(); columna++) -ciclo en relacion al alto de la imagen

filas.push_back(imagen.pixelColor(columna, fila).red()); --método de la clase QImage para obtener el valor en RED

```
matrizPixelesrojos.push_back(filas);
```

for (int fila=0; fila<imagen.height(); fila++) -ciclo en relacion al alto de la imagen

filas=vacio;

for(int columna=0; columna<imagen.height(); columna++) -ciclo en relacion al alto de la imagen

filas.push_back(imagen.pixelColor(columna, fila).green()); --método de la clase QImage para obtener el valor en GREEN

```
matrizPixelesverdes.push_back(filas);
```

for (int fila=0; fila<imagen.height(); fila++) -ciclo en relacion al alto de la imagen

filas=vacio;

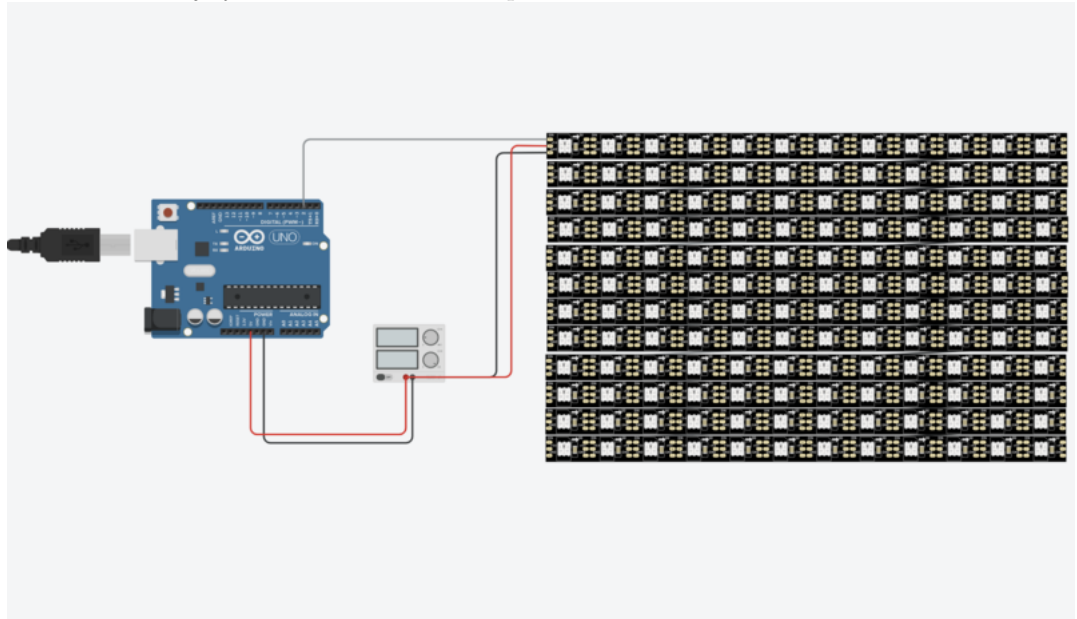
for(int columna=0; columna<imagen.height(); columna++) -ciclo en relacion al alto de la imagen

filas.push_back(imagen.pixelColor(columna, fila).blue()); --método de la clase QImage para obtener el valor en BLUE

```
matrizPixelesazules.push_back(filas);
```

4. Estructura del circuito

Para la construcción del circuito hemos utilizado 12 tiras de 12 leds neopixel, para así tener una dimensión de 12x12, conectados a una fuente de energía para evitar el retraso a la hora de ejecutar la simulación. La primera tira se encuentra conectada a un pin y a través de este se envía información al resto de tiras, debido a que se encuentran conectadas las salidas con las entradas. Las conexiones de voltaje y tierra se realizaron en paralelo.



5. Problemas presentados

- A la hora de buscar la lógica para las funciones de submuestreo y sobremuestreo, debido a que nos costaba un poco plantear el algoritmo para la redimensión.
- Cuando se buscó la forma de crear clases relacionadas a las funciones.
- Relacionado al repositorio, al principio nos costó entender como debíamos sincronizar los repositorios locales con los remotos, también al intentar deshacer algunos cambios.
- Respecto a las indicaciones, ya que se preguntó si en una imagen de 18x10 se debía hacer submuestreo en una dimensión y sobremuestreo en otra, a lo cual la respuesta obtenida fue no, el último día el profesor Augusto indicó que el programa debía tener la capacidad de submuestrear y sobremuestrear este tipo de imágenes.