# Segundo Parcial
# Informe de implementación

Ajuste de una imagen para visualizarla en un arreglo de LEDs con menor o mayor resolución

**David Correa Ochoa**

**Y**

**Francis David Roa Bernal**

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

# Índice

# 1. Descripción del problema

Se debe extraer la información RGB de una imagen y modificar dicha información para que por medio de varios arreglos de LEDs RGB o NeoPixel en el simulador TinkerCad se pueda visualizar la misma imagen con una menor o mayor resolución.

# 2. Solución implementada

A continuación se describen las soluciones implementadas para resolver el problema descrito en la sección anterior.

- Para la reducción se implementó;
  - Recorrer la fila de la imagen mientras se copia el contenido en un vector hasta que la cantidad de elementos en el vector sea adecuada, entonces se extrae un promedio de los valores contenidos en el vector, luego se guarda el promedio como un nuevo elemento en otro vector para luego borrar la información del primer vector cambiar de fila y repetir todo lo descrito hasta que se termine de recorrer la imagen.
  - Luego de terminar de recorrer la imagen, se recorre el vector donde se guardaron los promedios y se recorre recorre de forma adecuada mientras se suman los elementos recorridos hasta alcanzar un número de elementos recorridos adecuado para calcular un promedio de los elementos recorridos y guardar el promedio como un nuevo elemento en otro vector, se repite todo lo descrito en el presente item hasta que se termine de recorrer el primer vector.
  - El item anterior da como resultado una versión rotada en sentido anti-horario dentro del último vector, por lo que se crea un nuevo vector para guardar de forma adecuada la información del vector con la imagen rotada con el propósito de que la imagen final quede bien orientada en el nuevo vector.
- Para el aumento se implementó;
  - Recorrer la fila de una imagen elemento por elemento mientras se copia el contenido en un vector y se consulta la posibilidad de clonar el presente elemento, si el elemento se puede clonar entonces se vuelve a ingresar el mismo elemento en el vector la cantidad de veces que sea necesario para luego seguir con el proceso de revisión y clonado hasta llegar al último elemento de la fila para luego cambiar de fila y repetir todo lo descrito hasta terminar de recorrer la imagen.
  - Luego de terminar de recorrer la imagen se crea un nuevo vector y se le reserva una capacidad adecuada y luego se copia en este la información contenida en el primer vector.

- Se recorre el nuevo vector mientras se evalúa la posibilidad de clonar una cierta cantidad de elementos que de ser posible, se copia cada elemento del grupo e insertan dentro del mismo vector a una distancia(en número de elementos) adecuada de los originales luego se repite todo lo descrito una cantidad de veces adecuada.

# 3. Clases implementadas

Luego de una revisión de los ejemplos proporcionados en clase y una breve discusión se optó por utilizar las clases QImage, Adafruit_NeoPixel y vector.

- La clase QImage del entorno QT se utilizará para obtener los datos de la imagen a modificar. Se utilizaran métodos como el height(), width(), pixelcolor(), red(), blue() y green().
- La clase vector del entorno QT se utilizará para guardar los datos proporcionados por QImage en vectores, generando en el algoritmo 6 vectores, 3 de uso temporal y 3 que guardaran los datos de la imagen ya modificada, estos últimos son los que se imprimirán en un archivo de texto para usarlos en Arduino.
- La clase Adafruit_NeoPixel del entorno Arduino se utilizará en el código de Arduino donde permitirá a través del método setPixelColor() para manipular de manera sencilla de las tiras de neopixel solo necesitando 4 datos; posición (x,y) del led y los valores de intensidad de cada RGB(rojo,verde,azul).
- Se creó una clase imagen que se utiliza para guardar la información de la imagen y la dirección de la misma además de métodos que hacen uso de objetos de las clases QImage, vector y ofstream.
- La clase ofstream permite, leer, escribir, crear y modificar archivos de texto.

# 4. Estructura de las clases implementadas

Imagen: la clase imagen contiene una serie métodos que, en conjunto se encargan de;

- obtener los datos RGB de una imagen
- ajustar las dimensiones de la imagen para que se pueda mostrar en las tiras de LEDs Neo Píxel del circuito de la figura (1)
- escribir la información de la imagen redimensionada en un archivo de texto para su posterior uso.

## 5. Módulos de código

```
//estos son fragmentos del codigo implementado

/*
Fragmento donde se crea una instancia de la clase imagen y se pasa la
direccion de la imagen parametro a su constructor, se crea una
instancia de la clase QImage, se usa un metodo de la clase imagen que
accede a un atributo que tiene almacenado la direccion de la imagen,
se abre la imagen con un metodo de la clase QImage y se guarda en la
instancia de QImage
*/

imagen pic(a);
QImage im;
im=pic.dar_imagen();

/*
Fragmento donde se llama un metodo de una instancia de la clase imagen
para crear y/o abrir un archivo de texto
*/

pic.abrir_archivo();

/*
Fragmento donde se llama un metodo de una instancia de la clase imagen
que reduce el ancho de una imagen para un solo color varias veces para
reducir el ancho de todos los colores de la imagen, se le pasan como
parametros un char (controla la informacion de que color se va a usar) y
una instancia de la clase QImage (tiene guardada la informacion de una
imagen)
*/
pic.reduccionc(im,r);
pic.reduccionc(im,g);
pic.reduccionc(im,b);

/*
Fragmento donde se llama un metodo de una instancia de la clase imagen
que aumenta el ancho de una imagen para un solo color varias veces para
asi aumentar el ancho de todos los colores de la imagen, se le pasan
como parametros un char (controla la informacion de que color se va a
usar) y una instancia de la clase QImage (tiene guardada la informacion
de una imagen)
*/
```

```
pic.aumentoc(im,r);
pic.aumentoc(im,g);
pic.aumentoc(im,b);

/*
Fragmento donde se llama un metodo de una instancia de la clase imagen
que aumenta el largo de una imagen para un solo color varias veces para
asi aumentar el largo de todos los colores de la imagen, se le pasan
como parametros un char (controla la informacion de que color se va a
usar) y una instancia de la clase QImage (tiene guardada la informacion
de una imagen)
*/
pic.aumentof(im,r);
pic.aumentof(im,g);
pic.aumentof(im,b);

/*
Fragmento donde se llama un metodo de una instancia de la clase imagen
que reduce el largo de una imagen para un solo color varias veces para
reducir el largo de todos los colores de la imagen, se le pasan como
parametros un char (controla la informacion de que color se va a usar) y
una instancia de la clase QImage (tiene guardada la informacion de una
imagen)
*/
pic.reduccionf(im,r);
pic.reduccionf(im,g);
pic.reduccionf(im,b);

/*
Metodo que se utiliza para generar el fragmento de codigo para el proyecto de ti
*/
void escribir(vector<int>,char);

Fragmento donde se llama un metodo de una instancia de la clase imagen
para crear y/o abrir un archivo de texto a partir de la clase ofstream
*/
pic.cerrar_archivo();
```

## 6. Problemas presentados durante la implementación

- El montaje hecho en TinkerCad al principio funcionó solo cuando se trataba de mostrar un único color
- El montaje en TinkerCad funcionaba parcialmente cuando se retiraba de

la configuración una parte de las tiras NeoPixel

- Al aumentar la cantidad de filas de una imagen se presentaron problemas de locación de memoria cuando las imágenes presentaban cierta altura
- Al reducir las filas de una imagen luego de reducir las columnas, el resultado era la imagen completa reducida pero rotada en sentido anti-horario.

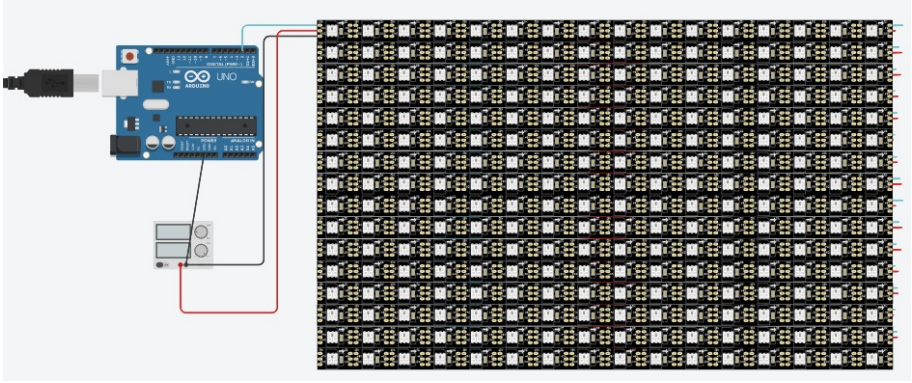

Figura 1: Montaje final del circuito