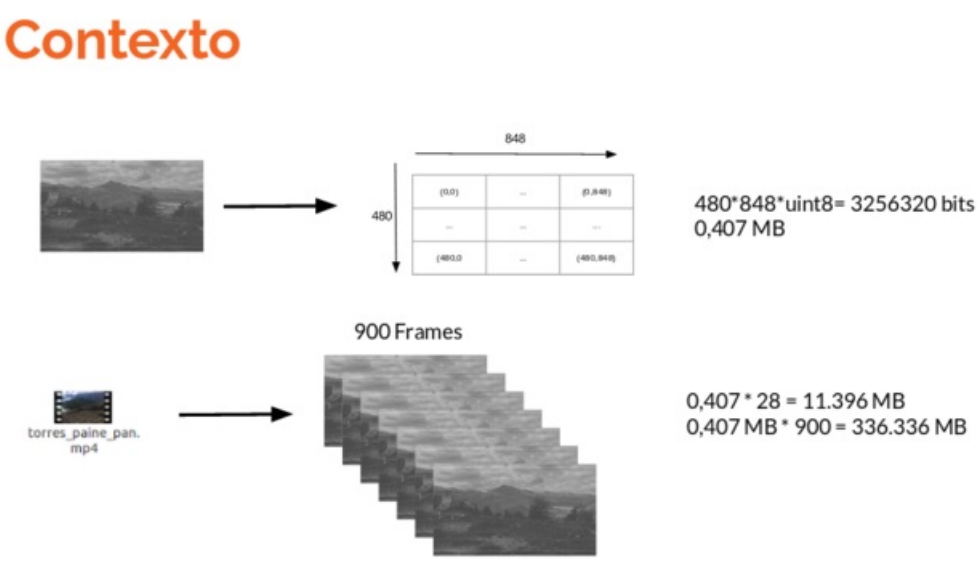
TRATAMIENTO Y TRANSFERENCIA DE IMAGENES

Intengrantes: -Felipe Saldias -Daniela Fábrega
Profesores: Pablo Huijse, Christian Lazo

Introducción

Tradicionalmente una imagen en "blanco y negro" puede ser entendida como un arreglo de `M * N` elementos llamados pixeles en donde cada uno de ellos corresponde a un valor entre 0 y 255 que representa la instensidad lumínica en esa coordenada de la imagen. Si consideramos que para poder representar 256 valores diferentes, siguiendo el razonamiento binario $log_2(256) = 8$, es necesario una palabra fija de largo 8 bits para representar cada uno de los valores de los pixeles en la imagen. Si tenemos, por ejemplo, una imagen de `480 * 848 = 407.040 pixeles` seria necesario `480 * 848 * 8 = 3.256.320 bits` o `0.407MB` para representar un frame lo que implica que para cada segundo de video se necesitan `0.407MB *28 = 11.397MB` lo que implica que para el video completo de 32 segundos (900 frames) se tiene un peso de 366.336MB.



Este proyecto consiste en visualizar la información de manera remota entregada por una cámara instalada para la detección de incendios, para lograrlo deberemos filtrar el ruido causado por las "condiciones climáticas", y considerar diversos tamaños de ancho de banda. La idea es conseguir una imagen con la mejor calidad visual posible y comprimirla en un tamaño soportado por ancho de banda de maximo 5 MB/s para conseguir una conexion estable y constante de mínimo 28 frames por segundo, para luego ser decodificado visualizado.

Para lograr enviar estos frames deberemos pasar por tres procesos claves en cada uno de ellos: Limpieza(filtrado), Cuantización y Codificación. Cada una de ellas divididas a su vez en otra serie de pasos a seguir para su correcta implementación.

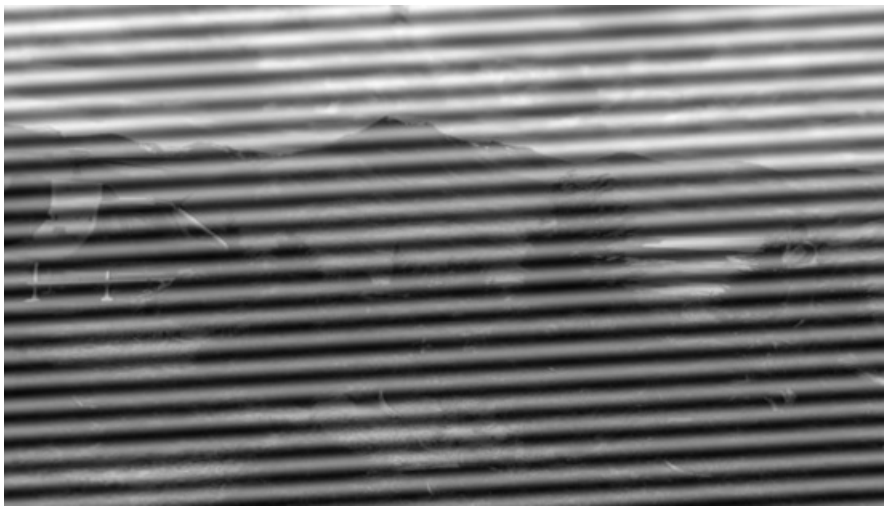
Metodologías

Analizaremos los métodos ocupados para resolver las distintas etapas del proyecto.

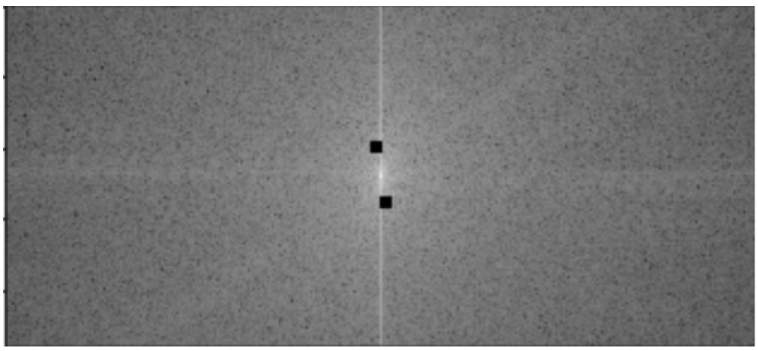
Limpieza

Existen diversas formas para filtrar una imagen con ruido y poder llegar a un estimado relativamente limpio, barajamos varias posibilidades antes de encontrar una respuesta, entre ellas usar un filtro de Wiener ya que contabamos con el video original, pero este resultaba ser algo "tramposo", además en la “vida real” lo normal sería no contar con esta información. Finalmente nos decidimos por un filtro rechaza banda, ya que al hacer una transformada de fourier podíamos visualizar la distribución del ruido. Este es un método que teníamos más o menos entendido, así que lo aplicamos para eliminar esas frecuencias “basura”, esto lo conseguimos asignándoles ceros a estas frecuencias inservibles. Ya en este punto nuestra duda era cuanto cortar, si cortabamos mucho podríamos perder información relevante del video, pero si era muy poco el ruido podría seguir siendo un factor molesto. Finalmente nos decidimos por eliminar un cuadro de 8 pixeles(8x8), por lo que fue necesario eliminar 64*2 pixeles de los 407.040 que teníamos originalmente, eliminando decentemente el ruido y no perjudicando la visualización de la imagen que queremos obtener. Aplicamos esto a todos los frames del video (900), obteniendo un resultado satisfactorio.

Frame contaminado con ruido periodico



Espectro de la imagen con el filtro rechaza banda ya aplicado



Frame limpio

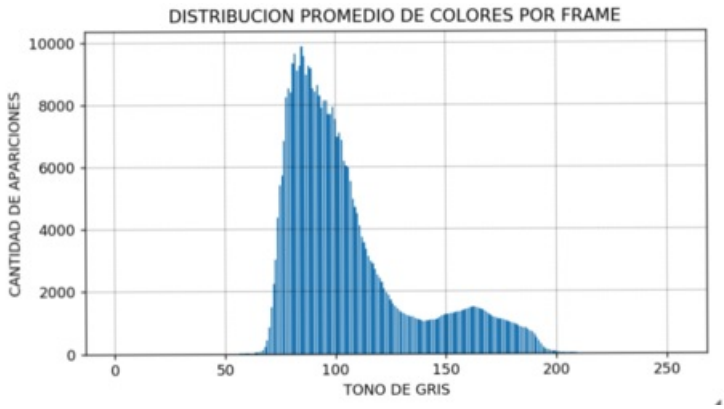
Compresión

Esta parte consistió en reducir la cantidad de datos utilizados para la representación de cada frame del video, para ello utilizamos los métodos que veremos a continuación. En particular es necesario destacar la gran ayuda de la libreria bitArray y sus metodos `encode() y decode()`

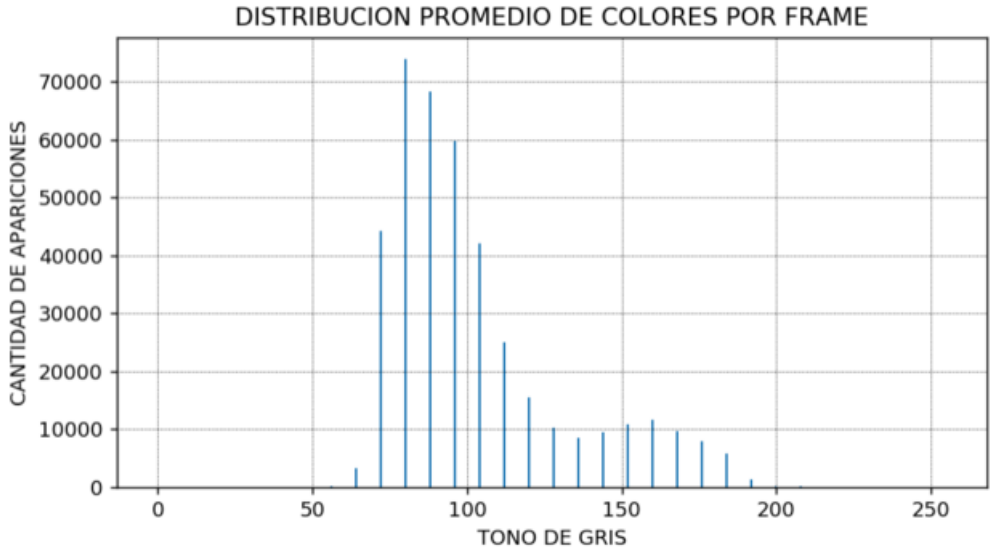
Para una mejor comprensión, esta etapa será dividida en dos secciones con su respectivo desarrollo:

Cuantización:

Este es un proceso para la compresión con pérdida de información.
Despues de explorar métodos que reducian los valores obtenidos en las transformadas(fourier y coseno) y encontrar diversos problemas en la representacion de estos mediante representativos, se opto por implementar un cuantizador en el dominio de los colores que originalmente se distribuian de la siguien manera:



Aprovechamos que el video estaba en una escala de grises para cuantizar de tal manera de no perder tanta calidad. Como el ojo humano solo puede apreciar hasta 32 tonos de grises, decidimos hacer una cuantizacion en estos 32 rangos, obteniendo un resultado que reemplaza los valores originales a un total de 32 valores representativos posibles, para esto: dividimos la matriz por 8, sacamos su parte entera, y multiplicandos la misma por 8(256 valores originales dividido los 32 valores de nuestra escala de grises), con esto conseguimos disminuir nuestra salida a un total de 21 valores(no habían más tonos aparte de ese rango, por lo que no era necesario incluir más valores).



Aplicando esto para todos los frames del video se obteniene un resultado que si bien, no se veía con la misma calidad que en un principio, disminuía su peso a menos de la mitad y se veía de manera aceptable, que era lo que queríamos lograr.

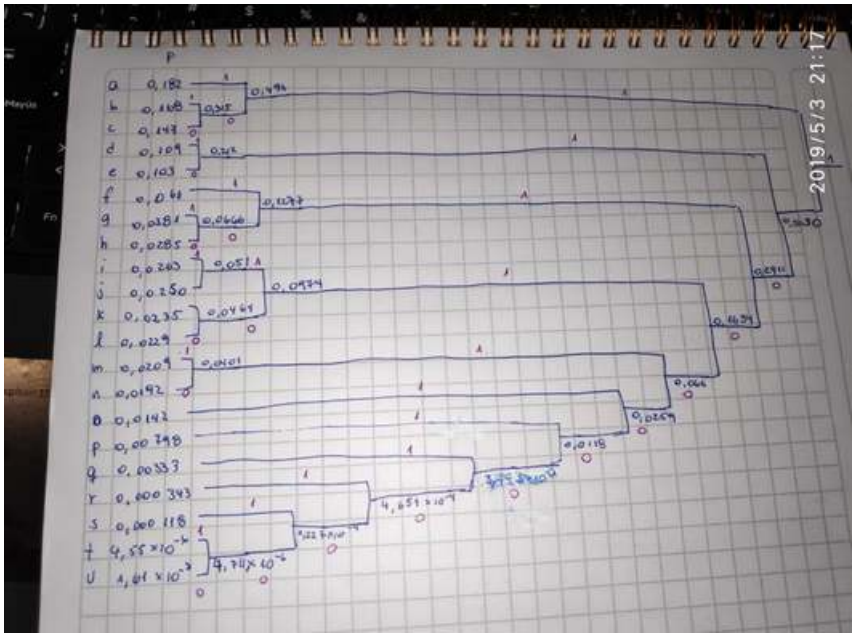
Imagen cuantizada a 16 tonos de grises:

Primero intentamos cuantizar a una menor cantidad de tonos para poder reducir aún más el peso, pero no nos dejó conforme el resultado, ya que no creímos que sea suficiente para poder detectar un foco de incendio, que era el principal objetivo del proyecto.

Imagen cuantizada a 32 tonos de grises:

Codificación:

En esta etapa se asignó un código para cada salida (color cuantizado) del video, utilizando “palabras” más cortas para aquellas salidas que se repiten más veces en el video según el gráfico de probabilidades que obtuvimos. Esta asignación de “palabras” (tira de bits) la hicimos utilizando el método de codificación de Huffman manualmente, conscientes de que esto nos limitaría funcionando solo para el video que se nos entregó, pero fue lo pudimos hacer en virtud del tiempo y de los conocimientos que tenemos. Para este método ordenamos las probabilidades de salidas de menor a mayor, resolviendo de la forma que se muestra a continuación:



De aquí obtuvimos nuestro “diccionario”, pero notamos un error grande, si bien en la parte de decodificación no hubo problemas al testear(pasar de la tira de bits a colores) al testear la codificación nos daba error, luego de investigar notamos que era porque la función encode solo acepta variables de tipo “char” para codificar y nosotros le estamos pasando una variable tipo string, lo que solucionamos utilizando una máscara, asociando cada string a un char (a,b,c…). Codificamos el video completo utilizando este método siguiendo el siguiente diccionario, máscara.

```
#CODIFICACION
huffman_dict = {
    'a': bitarray('11'), 'b': bitarray('101'),
    'c': bitarray('100'), 'd': bitarray('011'),
    'e': bitarray('010'), 'f': bitarray('0011'),
    'g': bitarray('00101'),'h': bitarray('00100'),
    'i': bitarray('000111'),'j': bitarray('000110'),
    'k': bitarray('000101'),'l': bitarray('000100'),
    'm': bitarray('000011'),'n': bitarray('000010'),
    'o': bitarray('000001'),'p': bitarray('0000001'),
    'q': bitarray('00000001'),'r': bitarray('000000001'),
    's': bitarray('0000000001'),'t': bitarray('00000000001'),
    'u': bitarray('00000000000'),
}
                #  0   1   2    3    4    5   6     7   8     9   10   11   12  13    14  15   16  17   18  19
NUMEROS = np.array([80, 88, 96, 72, 104, 112, 120, 160, 152 ,128, 168, 144, 136, 176, 184, 64, 192, 200, 56, 20
LETRAS = np.array(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i' ,'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's
```

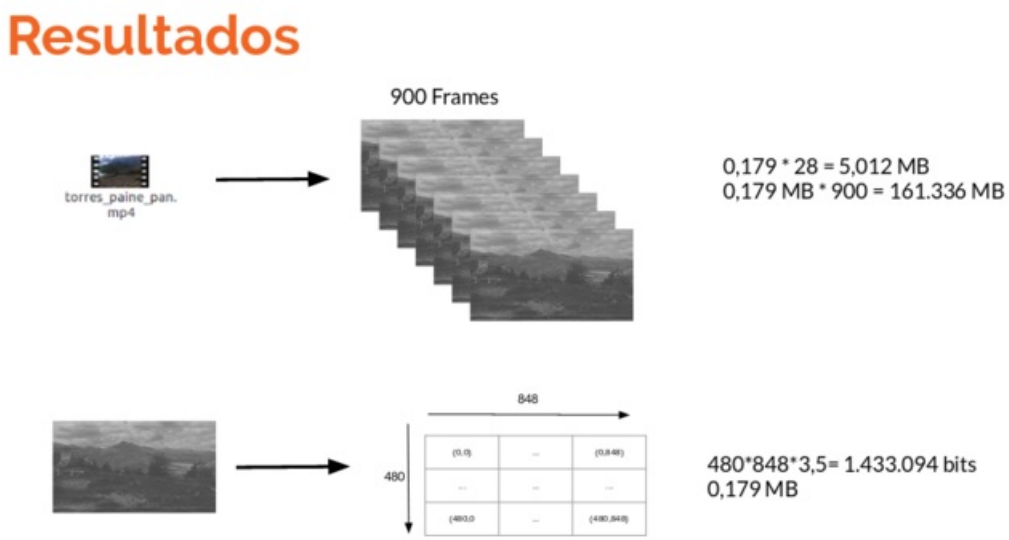
Descompresión

Decodificación:

Ya resuelto todo lo anterior y teniendo nuestro video en una tira de bits iniciamos la etapa de descompresión, que consistió en pasar los bits a nuestros colores originales (cuantizados). Usamos una función decode para esto, nos tomó un aproximado de 40 minutos descomprimir todo el video, logrando finalmente obtener nuevamente las imágenes con una calidad aceptable y a un peso ligeramente inferior a la mitad de lo que era originalmente.

Resultados

Una vez finalizados los procedimientos aplicados al video se logro obtener un largo de palabra promedio (calculado en base al largo promedio de las palabras entregadas por el compresor) de 3,5 bits por cada pixel lo que solo representa una compresion al 43.75% de su tamaño "original" de 8 bit para representar un color entre 0 y 255. A continuacion se aprecia un resumen de los resultados obtenidos tanto para cada frame, segundo de video y video completo:



Conclusiones

Es necesario recalcar que nuestro solución para el problema planteado en este proyecto va a alcanzar su máxima eficiencia solamente en este caso, ya que el diccionario fue calculado específicamente para la distribución de tonos de grises encontrados en el video analizado, siendo básicamente inútil o mucho menos eficiente en otras circunstancias. Ya que el ojo humano diferencia 32 tonalidades de grises es imposible reducir mas el peso del video de la manera estudiada en el presente informe sin perjudicar la imagen por lo que para alcanzar velocidades inferiores a 5MB de banda ancha es necesario explotar el area de la compresion basada en la transformacion o mas conocida en imagen como correlación interpixel.

Bibliografía

https://stackoverflow.com/questions/33089660/decoding-a-huffman-code-with-a-dictionary
https://riptutorial.com/es/algorithm/example/23995/codificacion-huffman
https://pypi.org/project/bitarray/

In [ ]: