CODIFICACIÓN Y TRANSMISIÓN DE VIDEO

para

Anchos de Banda Limitados

Versión 1.0

Autores:
Ricardo Coronado
Eduardo Hopperdietzel
Diego Sandoval

Docentes:
Pablo Huijse Heise
Cristian Lazo

Asignatura:
Comunicaciones (INFO239)

Universidad Austral de Chile
29 de abril de 2021

Introducción

El presente informe describe el diseño, desarrollo e implementación de un sistema de procesamiento de imágenes hecho a medida.

Una cámara de video ubicada en Torres del Paine captura imágenes del ambiente. Nuestro sistema elimina el ruido en las imágenes, luego las comprime, las transmite a través de un canal con ancho de banda limitado y finalmente decodifica la información al otro extremo de ese canal.

El canal no siempre tiene el mismo ancho de banda, por lo que nuestro sistema posee tres perfiles de compresión, para distintos anchos de banda.

El orden en el que se presentan los contenidos de este informe intenta ser el mismo orden en el que se ejecuta cada procedimiento del codec, incluidos otros procesos como la transferencia y decodificación, para finalizar con el análisis de cada canal en específico.

Instrucciones de Uso

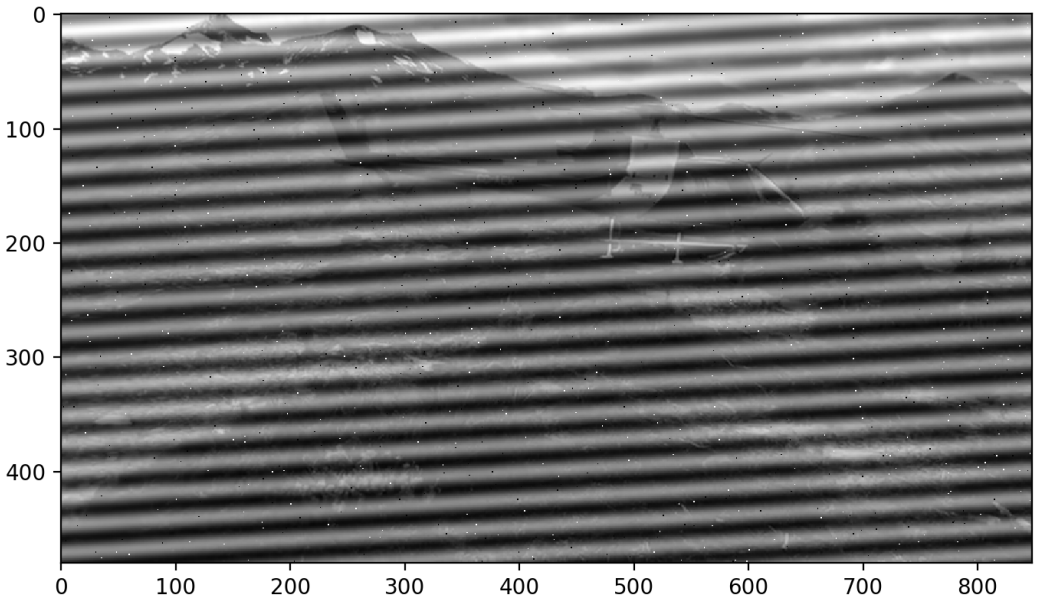
Cada perfil de compresión se implementó como un diccionario de Python.
Para seleccionar un diccionario, se debe modificar la variable **bandwidth** ubicada en el archivo **settings.py** por uno de los siguientes valores: 500, 1000 o 5000.

Cada diccionario posee los siguientes parámetros modificables:

Parámetro	Descripción
filtroSeno	Este parámetro permite seleccionar el método para eliminar el ruido sinusoidal, sus posibles valores son **FFT**, **FIR** o **IIR**.
interFrameThreshold	Este parámetro establece el umbral de Error Medio Cuadrático (EMC) entre los cuadros de 8x8 px del frame actual, respecto a los del anterior. Mientras menor sea su valor, más cuadrados serán enviados.
interFrameNeighbors	Si un cuadrado de 8x8 es enviado, este valor indica el radio de cuadrados vecinos que también lo serán.
interFrameSendAll	Este parámetro establece cada cuantos frames se sustituye el valor de **interFrameThreshold** por 1 con el fin de enviar más cuadros.
interFrameBlur	Este parámetro establece cuánto afecta el blur de los cuadrados de 8x8 px enviados por el frame actual, a los vecinos del frame almacenado en el receptor.
interFrameBlurRadius	Este parámetro establece el radio en píxeles del blur aplicado a los vecinos.
skipFrameAfter	Establece cada cuantos frames el emisor omite un envío.

postQuantization	Establece el método para reducir redundancia de las DCT luego de aplicar cuantización y Zig-Zag. Sus posibles valores son **Truncate** y **RunLength**.
dynamicDendrogram	Si su valor es True, se crea un dendrograma en base al frame actual y es enviado por el canal. Si es False utiliza uno estático y no es enviado por el canal.
allQ	Indica el porcentaje de calidad aplicados a los cuadrados que se envían según la instancia indicada en **interFrameSendAll**
maxQ	Calidad máxima con la que serán comprimidos los cuadrados, omitiendo los de la instancia de **allQ**.
minQ	Calidad máxima con la que serán comprimidos los cuadrados.
dendrograma	Ubicación del dendrograma estático a utilizar.

Análisis del Ruido



En el video pudimos identificar dos tipos de ruidos:

1. Un ruido sinusoidal de frecuencia baja y constante, con dirección diagonal, también constante, cuya fase aumenta en cada frame.
2. Ruido sal y pimienta, que satura un pixel y aparece en ubicaciones aleatorias cada frame.

Análisis en Frecuencia

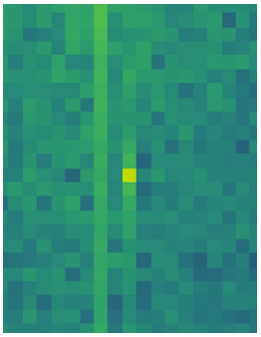
Analizando la transformada de Fourier 2D [1] pudimos localizar el ruido sinusoidal (los dos puntos amarillos alejados del centro), pero no así el ruido de sal y pimienta.

Filtrado del Ruido

Ruido Sinusoidal

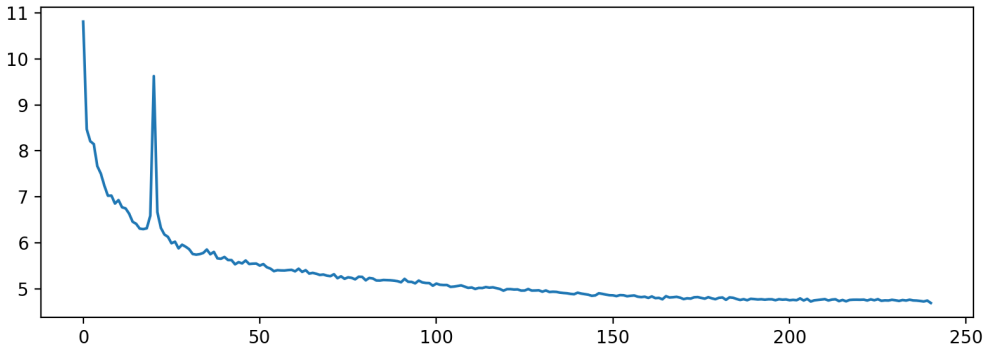
Implementamos tres métodos para filtrar el ruido sinusoidal:

1. **FFT 2D**: El primer método fue utilizar las funciones **dft** e **idct** de OpenCV para calcular la FFT 2D, eliminar los puntos previamente mencionados en el espacio de frecuencias y aplicar la transformada inversa. Pero esto no fue suficiente para eliminarlo por completo, también fue necesario reducir algunas componentes vecinas en el eje vertical (componentes sobre y bajo el punto amarillo).

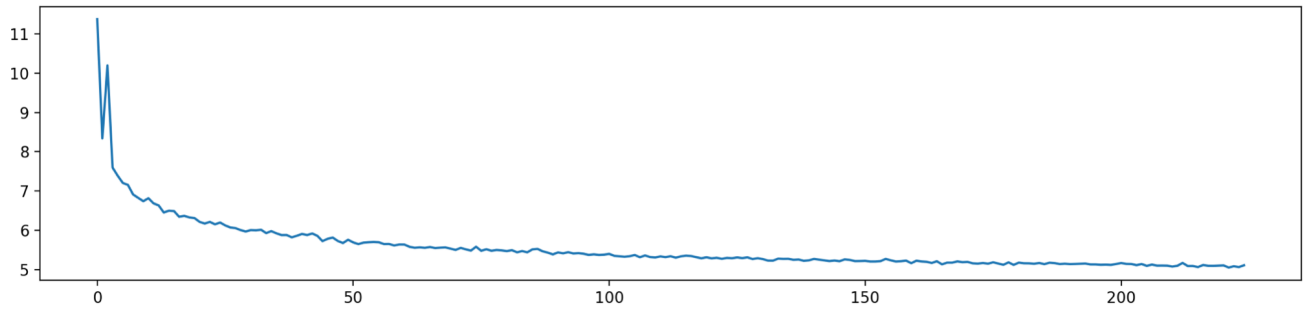


   Este método logró eliminar por completo el ruido, en un tiempo de 25 ms por frame.

2. **FIR 1D**: En un intento por mejorar el rendimiento del sistema, se nos ocurrió que sería más eficiente filtrar por filas o por columnas, ya que visto en ambos sentidos, el ruido sigue siendo sinusoidal.



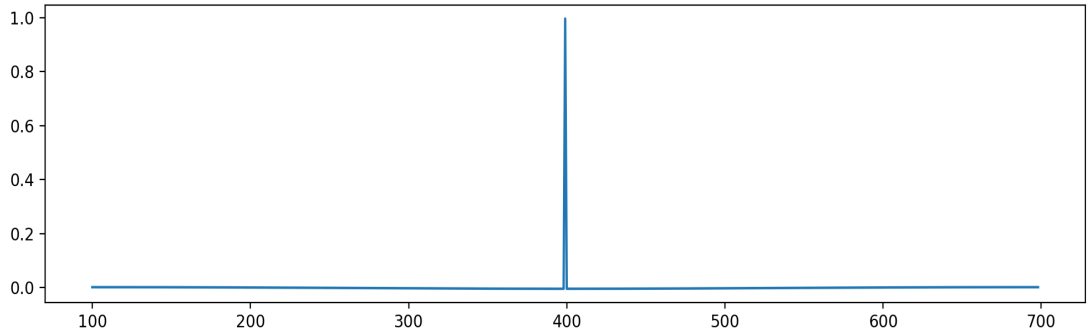
Espectro de magnitud promedio de todas las columnas de un frame.



Espectro de magnitud promedio de todas las filas de un frame.

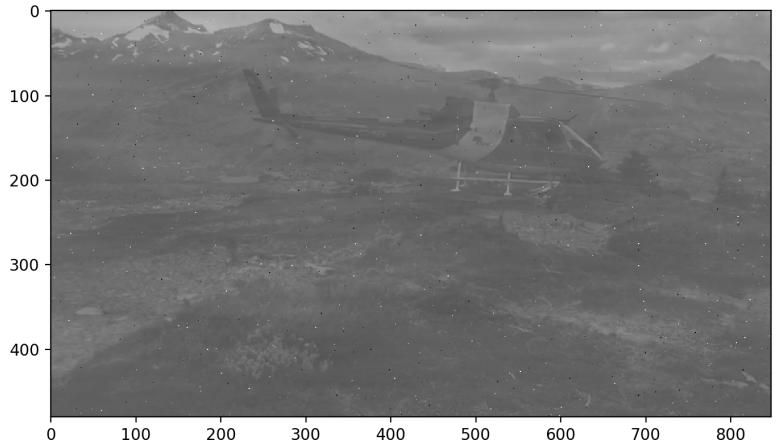
En ambos casos se aprecia claramente el componente sinusoidal.

Una vez identificada la ubicación del componente, diseñamos un filtro FIR, el cual al convolucionarlo con las filas, logra eliminar por completo el ruido.



Sin embargo, dado su gran número de coeficientes, tarda alrededor de 100 ms en filtrar cada frame.
Intentamos acotarlo utilizando diversos tipos de enventanado, pero al ser tan baja la frecuencia del ruido, una leve reducción de los coeficientes provoca que no logre eliminar el ruido por completo.

3. **IIR Notch**: Habiendo comprobado que filtrar únicamente por filas o por columnas es posible. Se nos ocurrió utilizar un filtro IIR de tipo Notch, el cual es ideal para eliminar frecuencias específicas.
   Este tipo de filtro posee un factor Q que indica la estrechez de la banda.
   Encontramos el Q óptimo haciendo un barrido lineal minimizando el error medio cuadrático entre un frame filtrado perfectamente y el mismo frame, pero filtrado por este.



En general, funciona bastante bien. Tarda aproximadamente 4 ms en filtrar cada frame, pero genera algunos artefactos, como manchas, deformaciones y contraste debido a la distorsión de fase, pero son prácticamente indetectables.

Ruido Sal y Pimienta

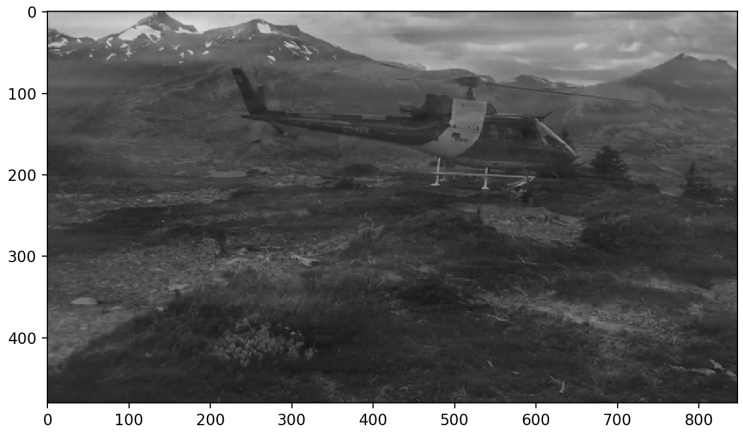
Dado que el ruido sal y pimienta genera cambios bruscos en la señal, nuestra intuición nos dice que se vería reflejado en frecuencias altas en el espectro de magnitud, por lo que intentamos eliminarlo reduciendo los bordes de la FFT 2D. Sin embargo, no tuvimos éxito.

El segundo método fue utilizar un filtro mediana, que consiste en reemplazar los píxeles saturados por la media de sus vecinos.

Para encontrar los puntos, partimos utilizando un umbral fijo, lo cual no funcionó, ya que muchos píxeles no ruidosos poseen un rango de intensidad mayor a algunos del ruido.

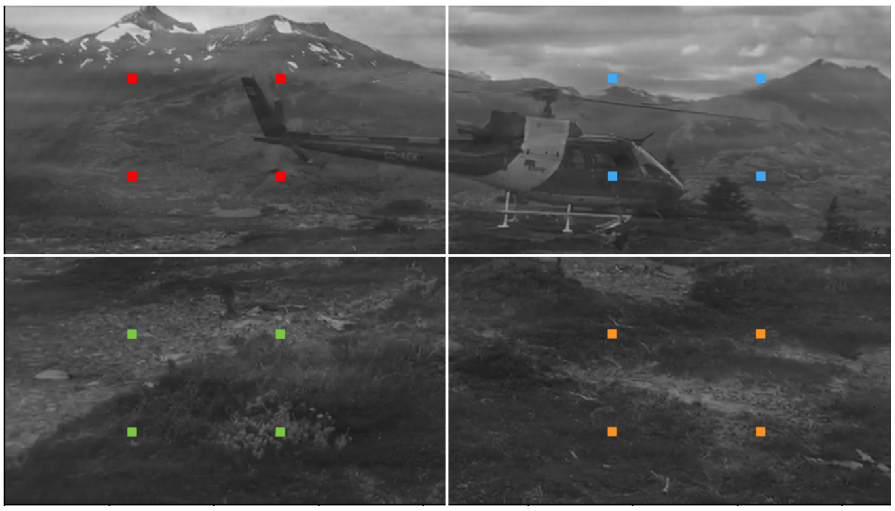
La segunda opción fue utilizar un umbral adaptativo, utilizando el método **adaptiveThreshold** de OpenCV, pero tampoco logramos buenos resultados.

Por último, comparamos cada píxel con sus vecinos no diagonales. En el caso que la diferencia de cada uno de ellos con el pixel de ruido sea mayor a un umbral (30), será considerado como ruido.
Este método logró eliminar por completo el ruido con una demora de 3 ms por frame.



Detección Cambio de Perspectiva

Entre dos imágenes consecutivas existen cambios de rotación y perspectiva. Como queremos implementar compresión entre cuadros o *Inter frame compression*, es necesario cuantificar estos cambios para corregir esas diferencias en la imagen resultante.



Para esto, se calcula el movimiento en cada esquina de la imagen seleccionando algunos pixeles equiespaciados, midiendo el EMC respecto a las mismas coordenadas del frame anterior pero entre 1 y 18 píxeles en cada dirección. De esta forma, el desfase con menor EMC equivale a la traslación de cada esquina.

Utilizamos dos enteros de 8 bits para almacenar la traslación (x,y) de cada esquina, los cuales suman 8 bytes en total y son posteriormente enviados por el canal.

En un inicio calculamos únicamente la traslación de la imágen completa, pero nos dimos cuenta que también existen cambios de rotación y de perspectiva entre frames.

Esta transformación es aplicada a una copia del frame anterior tanto en el emisor como en el receptor. Se transforma la del emisor, con el objetivo de reducir el EMC entre el nuevo frame y esta.

Utilizamos el método **warpPerspective** de OpenCV para aplicar el cambio de perspectiva utilizando el flag **BORDER_CONSTANT** en el emisor para hacer negros los bordes y generar que en el siguiente paso se envien los cuadros de esas posiciones.
En el lado del receptor utilizamos **BORDER_REPEAT**, el cual refleja la imagen en los bordes con el fin de ocultar lo mejor posible los errores.

Compresión Interframe

Luego de calcular el cambio de perspectiva, nuestro codec divide el frame actual en cuadros de 8x8 y calcula el error medio cuadrático de cada uno respecto al frame anterior (con perspectiva aplicada).
Los cuadrados con EMC menor al umbral **interFrameThreshold,** definido en el diccionario de configuraciones del ancho de banda, son enviados.
Para indicar qué cuadros son enviados y cuáles no, se crea una matriz de booleanos de 106 x 60, la cual es posteriormente comprimida en un arreglo de bits y enviada por el canal.
Cada N número de frames enviados (parámetro **interFrameSendAll** del diccionario de configuraciones) disminuye el umbral, provocando que se envíen muchos cuadros.
También se puede especificar la cantidad de vecinos que se envían de un cuadrado que será enviado, esto ayuda a que objetos que estén en movimiento no se vean “cortados” (parámetro **interFrameNeighbors**).

Por último, se puede cambiar el framerate del video, modificando la variable **skipFrameAfter**, el cual indica cada cuántos frames se omite uno.

Transformación y Cuantización

Transformación

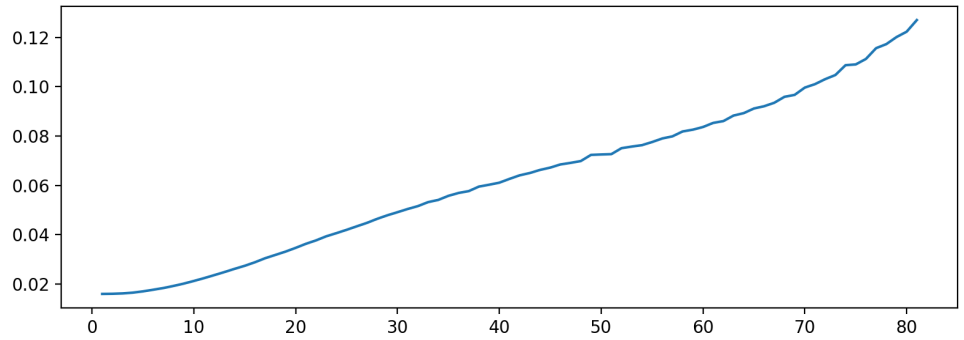
Para la transformación y cuantización decidimos basarnos en el algoritmo JPEG [2].
Transformamos cada matriz de 8x8 que será enviada, restando 128 y calculando su transformada discreta de coseno.

Cuantización

Luego aplicamos cuantización dividiendo los cuadros por la matriz estándar de JPEG [2] y aplicando redondeo. Utilizamos la matriz de cuantización de JPEG porque queremos tomar provecho de la función de porcentaje de calidad de JPEG.

Porcentaje de Calidad de Matriz Q

En vez de utilizar un porcentaje fijo, decidimos seleccionarlo dinámicamente en base a la cantidad de cuadros de 8x8 que son enviados en un frame, tomando en cuenta el ancho de banda actual y los cuadros por segundo del video o configuración seleccionada.
Para esto, medimos con anterioridad la cantidad de kbps promedio de cada cuadro de 8x8, cuantizando con un porcentaje de calidad de 1 hasta 80.



Podemos notar que el peso de la imagen aumenta de forma prácticamente lineal respecto al porcentaje de calidad.

Conociendo esta relación, podemos calcular la calidad máxima de la matriz de cuantización que nos asegure que no se sobrepasará el ancho de banda.

Este valor de porcentaje lo almacenamos en un entero sin signo de 8 bits y lo enviamos posteriormente junto a la demás información por el canal.

Redundancia Post Cuantización

Luego de cuantizar un cuadrado de 8x8, la matriz resultante queda compuesta principalmente por ceros, a excepción de la esquina superior izquierda, donde se almacenan los componentes de frecuencias bajas.

Por lo tanto, intentamos reducir esa redundancia, transformando la matriz a un arreglo de una dimensión aplicando el reordenamiento Zig-Zag [3] y utilizando el método Run Length.

Sin embargo este método no resultó ser eficiente, ya que en muchos casos los números se repetían una única vez, lo que duplicaba el tamaño de bits de esos valores.
Un método que resultó ser bastante mejor fue simplemente truncar el arreglo hasta el último valor distinto de cero y utilizar un entero de 8 bits para indicar el largo de la tira de bits.
Este valor es ubicado justo antes del arreglo truncado, con el fin de poder realizar el proceso inverso posteriormente.

Ambos métodos pueden ser seleccionados modificando el diccionario de configuraciones (parámetro **postQuantization**).

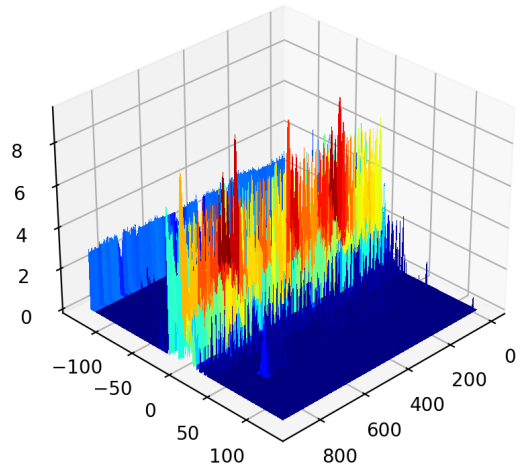
Codificación de Fuente

Decidimos utilizar el algoritmo Huffman para generar dendrogramas.

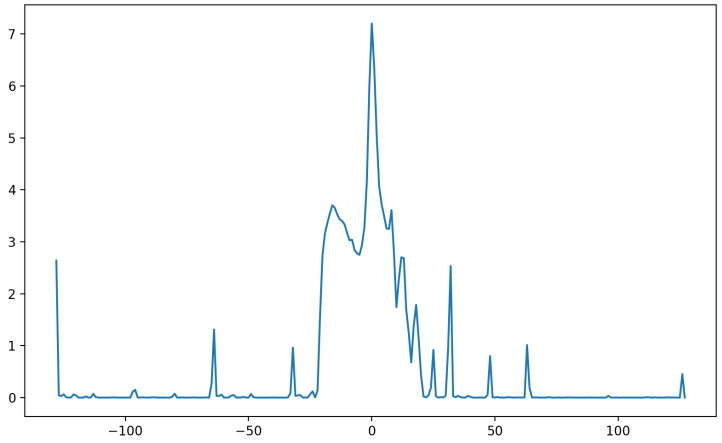
Utilizamos como fuente todos los valores posibles de 8 bits representados como enteros con signo desde -128 a 127, ya que en ese rango se encuentran los coeficientes de las DCT luego de ser cuantizadas.

Inicialmente generamos un dendrograma por frame y lo enviamos junto a este por el canal, y pese a que cada frame se codifica de forma óptima, el tamaño de cada dendograma es muy grande, consume alrededor de 75 - 200 Kbps del ancho de banda.

Por lo tanto decidimos analizar la variación del histograma de cada frame cuantizado en el tiempo.



Podemos observar que a lo largo del video suelen repetirse los mismos símbolos.



Graficando el promedio de todos los histogramas en escala logarítmica, vemos que los símbolos que más se repiten son los cercanos a cero. Esto tiene sentido, ya que la mayoría de los coeficientes de la DCT tienen estos valores luego de ser cuantizadas.

Dado esto, optamos por generar un dendrograma estático para cada ancho de banda evitando su envío por el canal.

Configuración de los Ancho de Banda

500 Kbps

Para el ancho de banda 500 Kbps, optamos por incrementar el umbral de EMC interframe para reducir el número de cuadrados a enviar (255/6360 por frame aprox), y enfatizamos en el uso de la predicción de perspectiva. Esto ocasiona que pequeños cambios en la imagen no se visualicen, como las aspas del helicóptero, pero sí objetos de mayor tamaño como la persona caminando. Al enviarse pocos cuadros, establecimos que también se envíen los vecinos en un radio de 3. Por último redujimos los fps a 24 y establecimos el rango de porcentaje de calidad entre 18 y 25 intentando maximizar el uso del ancho de banda.

El ancho de banda usado cada segundo del video varía entre 106 y 496 Kbps con una media de 311 Kbps.

El EMC entre el video filtrado y el video transmitido varía entre 7 y 52, con una media de 48 a lo largo del video.

1 Mbps

Para el siguiente caso, decidimos reducir el umbral de EMC interframe con el fin de incrementar la visualización de cambios menores como el de las aspas. Al enviar más cuadrados (550/6360 aprox), decidimos reducir el radio de vecinos a 2. Por último, incrementamos el rango de porcentaje de calidad a 20 - 30 para maximizar el uso del ancho de banda.

El ancho de banda usado en cada segundo del video varía entre 351 y 973 Kbps con una media de 632 Kbps.

El EMC entre el video filtrado y el video transmitido varía entre 2.7 y 42, con una media de 33.7 a lo largo del video.

5 Mbps

Por último, para el caso de 5 Mbps, pudimos darnos el lujo de reducir el umbral de EMC interframe a 30, por lo que cualquier ínfimo cambio entre frames es enviado, y aumentar el factor de calidad a un rango de 30 - 45.

El ancho de banda usado en cada segundo del video varía entre 1960 y 4728 Kbps con una media de 2970 Kbps.

El EMC entre el video filtrado y el video transmitido varía entre 0.8 y 13.1, con una media de 8.1 a lo largo del video.

Bibliografía

[1] P. Huijse, 'Filtrado de imágenes', 2020. [Online]. Available: https://phuijse.github.io/UACH-INFO185/clases/unidad1/06_filtrado_de_im%C3%A1genes.html [Accessed: 2021-05-01]

[2] G. K. Wallace, "The JPEG still picture compression standard," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii-xxxiv, 1992.

[3] K. Cabeen and P. Gent, "Image Compression and the Discrete Cosine Transform". [Online]. Available: https://www.math.cuhk.edu.hk/~lmlui/dct.pdf [Accessed: 2021-05-01]