

SISTEMA DE ADQUISICIÓN, COMPRESIÓN SIN PÉRDIDAS Y TRASMISIÓN DE VIDEO

Proyecto de fin de carrera de Ingeniería Eléctrica
Plan 97, opción Telecomunicaciones

Integrantes:

Gonzalo Martín Pino Capilla
Sandra Leticia Montañó Silveira
Felipe Joaquín Rivera Lordoguín

Tutor:

Ing. Ignacio Ramírez

Versión corregida por el tribunal

Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay
16 de abril de 2007

Felipe Rivera: felipejrlor@adinet.com.uy
Gonzalo Pino: gmpino@adinet.com.uy
Sandra Montaña: smont@adinet.com.uy

RESUMEN

En este documento se detalla la investigación y las contribuciones realizadas para la implementación de un “Sistema de Adquisición, Compresión sin Pérdidas y Trasmisión de Video”. La investigación consiste en una revisión del estado del arte en los métodos de compresión de imágenes y videos sin pérdidas. Como se verá la misma estará restringida a los algoritmos de compresión de imágenes sin pérdidas CALIC y LOCO-I/JPEG-LS, así como a aquellos algoritmos de compresión de video sin pérdidas que proponen una extensión a video de los mismos. El principal aporte de este proyecto es un algoritmo de compresión de video sin pérdidas que propone una extensión simple de JPEG-LS a video. El mismo logra niveles de compresión competitivos con los máximos alcanzados por los métodos estudiados, pero con un costo computacional notablemente inferior. La implementación del sistema toma la forma de aplicaciones cliente y servidor, residiendo estos en computadores tipo PC. Este servidor se encarga de la adquisición de video, la compresión del mismo y su trasmisión mediante un stream TCP. En contrapartida el cliente tiene por función la decodificación de este stream y el despliegue del video. Finalmente, es de destacar que absolutamente todas las herramientas de software utilizadas fueron libres.

Tabla de contenidos

1. INTRODUCCIÓN	3
1.1. Motivación	3
1.2. Objetivos del proyecto	4
1.3. Hipótesis de Trabajo	5
1.4. Líneas de Trabajo a Desarrollar	5
2. CONCEPTOS BÁSICOS	7
2.1. Definiciones	7
2.1.1. Imagen Digital	7
2.1.2. Video	7
2.2. Teoría de la Información	13
2.2.1. Información	14
2.2.2. Definición de información	14
2.2.3. Definición de Entropía	15
2.2.4. Definición de Longitud media del Código	16
2.2.5. Entropía y Videos Digitales	16
2.2.6. ¿Qué entendemos por compresión de video sin pérdidas?	18
3. COMPRESIÓN DE VIDEO	19
3.1. Estado del Arte en la Compresión de Video sin Pérdidas	19
3.1.1. CALIC	19
3.1.2. I-CALIC	24
3.1.3. M-CALIC	26
3.1.4. LOPT-3D	28
3.2. Comparación de Desempeños	31
3.2.1. Conjunto de Videos de Prueba	31
3.2.2. Parámetros de Evaluación	39
3.2.3. Desempeños de I-CALIC, M-CALIC y LOPT-3D	40
3.3. Comparación de Resultados y Conclusiones	41
4. LOCO-I JPEG-LS	45
4.1. El Codificador	45
4.1.1. El Modo Normal	47
4.1.2. Modelado de Contexto	49
4.1.3. Corrección Adaptativa	50
4.1.4. Reducción del rango para ϵ	53
4.1.5. Codificación	53
4.1.6. Reseteo	61

4.1.7. El Modo de Rachas	61
4.2. Resumen de la Implementación del Algoritmo: JPEG-LS	65
4.2.1. Comentarios	70
4.3. El Decodificador	70
5. ALGORITMOS PROPUESTOS PARA EL SISTEMA	75
5.1. JPEG-LS Cuadro a Cuadro	75
5.2. JPEG-LS con “Memoria” Inter-Cuadro	77
5.3. JPEG-LS + Compensación de Movimiento	79
5.3.1. Compensación de Movimiento	80
5.3.2. Métodos Considerados	81
5.3.3. Implementación	85
5.3.4. Resultados Obtenidos	85
5.4. LOVICO	93
5.4.1. Modificación del Predictor	95
5.4.2. Modificación del Cálculo del Contexto	104
5.5. Comparación de Resultados y Conclusiones	107
5.5.1. Comparación: Nivel de Compresión	107
5.5.2. Comparación: Cuadros Procesados por Segundo	109
5.5.3. Conclusiones	110
6. STREAMING DE VIDEO	117
6.1. Introducción	117
6.1.1. Diferentes Tipos de Transmisión de Video	117
6.1.2. Arquitectura de un sistema de streaming de video	118
6.1.3. Control de congestión	119
6.1.4. Acciones frente a las pérdidas en la red	119
6.2. Protocolos para streaming de video	121
6.3. Streaming de Video con TCP	121
6.3.1. Implementación	121
7. ADQUISICIÓN DE VIDEO	125
7.1. API <i>V4L2</i>	125
7.1.1. Apertura del dispositivo	126
7.1.2. Configuración del dispositivo	126
7.1.3. Negociar un tipo de datos	127
7.1.4. Negociar un método de entrada-salida	127
7.1.5. Bucle de entrada-salida	128
7.1.6. Cerrado del dispositivo	128
7.2. Implementación	129
7.3. Hardware Utilizado	129
8. SISTEMA IMPLEMENTADO	131
8.1. Hardware Utilizado	132
8.2. Software Implementado	133
8.2.1. Software servidor	133
8.2.2. Software cliente	134
8.3. Relevamiento del Sistema	136
8.3.1. Relevamiento Sobre el Conjunto de Videos de Prueba	136

8.3.2. Relevamiento Sobre Videos Caseros	139
8.4. Conclusiones	141
9. CONCLUSIONES GLOBALES	143
A. SOFTWARE DE RED	149
A.1. Los sockets de Berkeley	149
A.1.1. Cliente-Servidor	149
A.1.2. Código Implementado	150
B. CONTENIDO DEL CD	153

Índice de figuras

2.1. Numeración usual de filas y columnas	7
3.1. Píxeles involucrados en los promedios de la primera pasada	20
3.2. Ponderaciones asignadas a los píxeles involucrados y promedios de la segunda pasada	21
3.3. Ponderación de píxeles en la tercer pasada	22
3.4. Molde causal utilizado por I-CALIC	25
3.5. Enumeración 2D relativa al cuadro actual usada por LOPT-3D	29
3.6. Enumeración 3D relativa al cuadro actual y al anterior usada por LOPT-3D . . .	29
3.7. YCbCr 4:2:0. El cuadrado representa valores para cada píxel de Y, mientras que el triángulo y el círculo representan valores compartidos de Cb y Cr para cada cuatro píxeles	32
3.8. Cuadro 10 de la secuencia “Salesman”	32
3.9. Cuadro 100 de la secuencia “Salesman”	33
3.10. Cuadro 10 de la secuencia “ Mobile and Calendar”	33
3.11. Cuadro 250 de la secuencia “Mobile and Calendar”	34
3.12. Cuadro 10 de la secuencia “Container”	34
3.13. Cuadro 250 de la secuencia “Container”	35
3.14. Cuadro 10 de la secuencia “Tempete”	35
3.15. Cuadro 250 de la secuencia “Tempete”	36
3.16. Cuadro 10 de la secuencia “Kitchgrass”	36
3.17. Cuadro 99 de la secuencia “Kitchgrass”	37
3.18. Cuadro 10 de la secuencia “Sean”	37
3.19. Cuadro 150 de la secuencia “Sean”	38
3.20. Cuadro 10 de la secuencia “Silent”	38
3.21. Cuadro 100 de la secuencia “Silent”	39
3.22. Cuadro 10 de la secuencia “Foreman”	39
3.23. Cuadro 280 de la secuencia “Foreman”	40
4.1. Diagrama de bloques de JPEG-LS	46
4.2. Molde causal usado para modelado de contexto y predicción	46
4.3. El primer caso en que se elige el valor b como predicción	47
4.4. El segundo caso en que se elige b como predicción	48
4.5. Interpretación de nivel de gris como coordenada Z , para la imagen “Lena”	49
5.1. Búsqueda en tres pasos	82
5.2. Búsqueda en tres pasos rápida	83
5.3. Búsqueda logarítmica	83

5.4. Búsqueda en capas	84
5.5. Búsqueda de dirección conjugada	84
5.6. Molde causal “espacio-temporal”	94
5.7. Píxeles de la secuencia Mobile y Calendar con columna fija igual a un cuarto del total	96
5.8. Píxeles de la secuencia Mobile y Calendar con columna fija igual a un medio del total	97
5.9. Píxeles de la secuencia Mobile y Calendar con fila fija igual a un cuarto del total	98
5.10. Píxeles de la secuencia Mobile y Calendar con fila fija igual a un medio del total	98
5.11. Píxeles de la secuencia Container con columna fija igual a un cuarto del total	99
5.12. Píxeles de la secuencia Container con columna fija igual a un medio del total	99
5.13. Píxeles de la secuencia Container con fila fija igual a un cuarto del total	100
5.14. Píxeles de la secuencia Container con fila fija igual a un medio del total	100
5.15. Píxeles de la secuencia Silent con columna fija igual a un cuarto del total	101
5.16. Píxeles de la secuencia Silent con columna fija igual a un medio del total	101
5.17. Píxeles de la secuencia Silent con fila fija igual a un cuarto del total	102
5.18. Píxeles de la secuencia Silent con fila fija igual a un medio del total	102
5.19. Predictores MED utilizado por LOVICO	103
5.20. Nivel de compresión absoluto medio alcanzado por los diferentes algoritmos aplicados al conjunto de videos de prueba, en bpp.	108
5.21. Mejora en el nivel de compresión medio respecto a JPEG-LS cuadro a cuadro alcanzada por los diferentes algoritmos	108
5.22. Cuadros procesados por segundo en media para los algoritmos considerados	109
5.23. Cuadros procesados por segundo vs nivel de compresión en %, ambos en media	110
5.24. Nivel de compresión en bpp “cuadro a cuadro” para SALESMAN	111
5.25. Nivel de compresión en bpp “cuadro a cuadro” para MOBILE and CALENDAR	112
5.26. Nivel de compresión en bpp “cuadro a cuadro” para CONTAINER	112
5.27. Nivel de compresión en bpp “cuadro a cuadro” para TEMPETE	113
5.28. Nivel de compresión en bpp “cuadro a cuadro” para KITCHGRASS	113
5.29. Nivel de compresión en bpp “cuadro a cuadro” para SEAN	114
5.30. Nivel de compresión en bpp “cuadro a cuadro” para SILENT	114
5.31. Nivel de compresión en bpp “cuadro a cuadro” para FOREMAN	115
8.1. Esquema del sistema completo	131
8.2. Diagrama de flujo del servidor	133
8.3. Diagrama de flujo del cliente.	135

Índice de tablas

2.1. Algunas aplicaciones y sus características Fuente: [1]	10
2.2. Algunas tecnologías de WAN, LAN y WLAN y el bit rate que ofrecen	11
2.3. Parámetros establecidos por ITU-R BT.601-5	11
2.4. Porcentaje de resolución de cromas respecto a la luminancia en las direcciones vertical y horizontal	13
2.5. Codificación binaria de los dígitos decimales	13
2.6. Imagen de 8 bpp	17
2.7. Ocurrencias y probabilidades estimadas de los símbolos que forman la imagen en 2.6	17
2.8. Ocurrencias y probabilidades estimadas de los pares de píxeles que forman la imagen en 2.6	18
3.1. Cantidad de cuadros utilizados de cada video de prueba. Se detalla el primer cuadro utilizado, el último y la cantidad total.	41
3.2. Nivel de Compresión en bpp alcanzado por los algoritmos LOPT-3D, I-CALIC, M-CALIC para el canal G de las secuencias de prueba especificadas. Fuente: [2]	41
3.3. Tiempo de procesamiento consumido por los algoritmos LOPT-3D, I-CALIC, M- CALIC para el canal G de las secuencias de prueba especificadas, en un PC con CPU Intel Pentium IV, 2.6GHz. Fuente: [2]	42
3.4. Cuadros procesados por segundo por los algoritmos LOPT-3D, I-CALIC, M- CALIC para el canal G de las secuencias de prueba especificadas, en un PC con CPU Intel Pentium IV, 2.6GHz. Fuente: [2]	42
4.1. Código Unario de q	55
4.2. Códigos de Golomb cuando $m = 2^c$, o $GPO2$	55
4.3. Códigos de Golomb cuando $m \neq 2^c$	56
4.4. Mapeo de Rice	58
4.5. $\epsilon, M(\epsilon), M'(\epsilon), R_k(M(\epsilon))$ y $R_k(M'(\epsilon))$ para $k = 1$	60
4.6. $\epsilon, M(\epsilon), M'(\epsilon), R_k(M(\epsilon))$ y $R_k(M'(\epsilon))$ para $k = 0$	61
4.7. Tabla $kr = J(i)$	63
4.8. Tabla para determinar map . El símbolo v indica que la condición se cumple ("verdadera") y el símbolo f indica lo contrario ("falsa")	65
5.1. Nivel de compresión en bpp y tiempo de procesamiento para JPEG-LS Cuadro a Cuadro aplicado al conjunto de videos de prueba	76
5.2. Nivel de compresión en % y cuadros procesados por segundo para JPEG-LS Cua- dro a Cuadro aplicado al conjunto de videos de prueba	76

5.3. Nivel de compresión en bpp y tiempo de procesamiento para MeJPEG-LS aplicado al conjunto de videos de prueba	77
5.4. Nivel de compresión en % y cuadros procesados por segundo para MeJPEG-LS aplicado al conjunto de videos de prueba	78
5.5. Nivel de compresión en % y cuadros procesados por segundo para MeJPEG-LS aplicado al conjunto de videos de prueba, con <i>RESET</i> = 128	78
5.6. Nivel de compresión en % y cuadros procesados por segundo para MeJPEG-LS aplicado al conjunto de videos de prueba, con <i>RESET</i> = 256	79
5.7. Nivel de compresión en % y cuadros procesados por segundo para MeJPEG-LS aplicado al conjunto de videos de prueba, con diferentes valores de <i>RESET</i> . . .	79
5.8. Nivel de compresión en bpp y tiempo de procesamiento para MoJPEG-LS con $L = 16$	87
5.9. Nivel de compresión en % y cuadros procesados por segundo para MoJPEG-LS con $L = 16$	87
5.10. Nivel de compresión en bpp y tiempo de procesamiento para MoJPEG-LS con $L = 32$	87
5.11. Nivel de compresión en % y cuadros procesados por segundo para MoJPEG-LS con $L = 32$	88
5.12. Nivel de compresión en bpp y tiempo de procesamiento para MoJPEG-LS con $L = 64$	88
5.13. Nivel de compresión en % y cuadros procesados por segundo para MoJPEG-LS con $L = 64$	88
5.14. Nivel de compresión en bpp y tiempo de procesamiento para MoJPEG-LS con $L = 128$	89
5.15. Nivel de compresión en % y cuadros procesados por segundo para MoJPEG-LS con $L = 128$	89
5.16. Resumen de performance para $L = 16$, $L = 32$, $L = 64$ y $L = 128$. Tanto el nivel de compresión como la cantidad de cuadros procesados por segundo corresponden a valores medios	89
5.17. Nivel de compresión en bpp y tiempo de procesamiento para AnJPEG-LS	90
5.18. Nivel de compresión en % y cuadros procesados por segundo para AnJPEG-LS .	91
5.19. Nivel de compresión en bpp y tiempo de procesamiento para AnJPEG-LS con secuencias de video submuestreadas temporalmente con <i>SALTO</i> = 2	92
5.20. Nivel de compresión en % y cuadros procesados por segundo para AnJPEG-LS con secuencias de video submuestreadas temporalmente con <i>SALTO</i> = 2	92
5.21. Nivel de compresión en bpp y tiempo de procesamiento para AnJPEG-LS con secuencias de video submuestreadas temporalmente con <i>SALTO</i> = 3	92
5.22. Nivel de compresión en % y cuadros procesados por segundo para AnJPEG-LS con secuencias de video submuestreadas temporalmente con <i>SALTO</i> = 3	93
5.23. Resumen de performance para los algoritmos propuestos. Tanto el nivel de compresión como la cantidad de cuadros procesados por segundo corresponden a valores medios	93
5.24. Nivel de compresión en bpp y tiempo de procesamiento para LOVICO aplicado al conjunto de videos de prueba	105
5.25. Nivel de compresión en % y cuadros procesados por segundo para LOVICO . . .	105
5.26. Nivel de compresión en bpp y tiempo de procesamiento para LOVICO con secuencias de video submuestreadas temporalmente con <i>SALTO</i> = 3	106
5.27. Nivel de compresión en % y cuadros procesados por segundo para LOVICO con secuencias de video submuestreadas temporalmente con <i>SALTO</i> = 3	106

5.28. Nivel de compresión medio en bpp y % alcanzado por los algoritmos vistos. . . .	107
5.29. Cuadros procesados por segundo.	109
7.1. Dispositivos reconocidos por V4L2	126
8.1. Cuadros por segundo generados en codificación en el host servidor por JPEG-LS cuadro a cuadro, MeJPEG-LS y LOVICO para los distintos videos de prueba utilizando el canal verde.	132
8.2. Nivel de compresión, cantidad de cuadros por segundo que el sistema puede procesar y ancho de banda consumido, utilizando JPEG-LS cuadro a cuadro para los distintos videos de prueba.	137
8.3. Nivel de compresión, cantidad de cuadros por segundo que el sistema puede procesar y ancho de banda consumido, utilizando MeJPEG-LS para los distintos videos de prueba.	137
8.4. Nivel de compresión, cantidad de cuadros por segundo que el sistema puede procesar y ancho de banda consumido, utilizando LOVICO para los distintos videos de prueba.	138
8.5. Valores medios de nivel de compresión, cantidad de cuadros por segundo que el sistema puede procesar y ancho de banda consumido par los métodos de codifi- cación implementados en el sistema	138
8.6. Cuadros procesados por segundo.	138
8.7. Cantidad de cuadros por segundo y ancho de banda consumido por JPEG-LS cuadro a cuadro para los videos mencionados utilizando el canal Y.	139
8.8. Cantidad de cuadros por segundo y ancho de banda consumido por JPEG-LS cuadro a cuadro para los videos mencionados utilizando el canal verde.	139
8.9. Cantidad de cuadros por segundo y ancho de banda consumido por MeJPEG-LS para los videos mencionados utilizando el canal Y.	139
8.10. Cantidad de cuadros por segundo y ancho de banda consumido por MeJPEG-LS para los videos mencionados utilizando el canal verde.	140
8.11. Cantidad de cuadros por segundo y ancho de banda consumido por LOVICO para los videos mencionados utilizando el canal Y.	140
8.12. Cantidad de cuadros por segundo y ancho de banda consumido por LOVICO para los videos mencionados utilizando el canal verde.	140
8.13. Cantidad de cuadros por segundo promedio generados por los distintos algoritmos en los videos caseros formato CIF para el canal verde y la luminancia.	140
8.14. Cantidad de cuadros por segundo promedio generados por los distintos algoritmos en los videos caseros formato QCIF para el canal verde y la luminancia.	140

1 INTRODUCCIÓN

1.1. Motivación

Existen variedad de aplicaciones en diferentes campos de la ciencia, la industria, y las actividades empresariales en que la compresión de imágenes digitales sin pérdidas es la única forma admisible de reducir la cantidad de bits empleados para representarlas.

Esto puede deberse a razones éticas, legales, o económicas.

Como ejemplo se puede citar su aplicación a la *imagenología médica*. En la compresión de documentos médicos para su archivado o comunicación la pérdida de información podría conducir a un diagnóstico erróneo o llevar a investigadores a conclusiones equivocadas. Esta situación se reitera en otros campos de la ciencia, como la astronomía, la física o la topografía, donde imágenes adquiridas deben ser almacenadas para su posterior procesamiento sin que esto las afecte de modo alguno. También en el almacenamiento de copias digitales de diversos documentos legales, la compresión con pérdidas no está permitida.

En aplicaciones como la producción publicitaria se efectúan codificaciones y decodificaciones reiteradas sobre la misma imagen. Si una técnica de compresión con pérdidas fuese utilizada la calidad final podría ser severamente afectada. En el caso de que el proceso para la adquisición sea extremadamente costoso, por ejemplo imágenes provenientes de satélites o dispositivos enviados a otros planetas, la compresión con pérdidas es absolutamente indeseable. Otras aplicaciones en las que la compresión sin pérdidas se utiliza por motivos económicos son las que implementan algún tipo de control industrial en base al análisis de imágenes. Por ejemplo, descartar o no un producto en base al análisis computarizado de imágenes del mismo.

Se hace evidente entonces durante la última década del siglo XX un número creciente de aplicaciones demandantes de procesos de compresión que no afecten los datos originales.

Esto tiene como consecuencia, la dedicación de mucho tiempo de investigación por diversas instituciones al desarrollo de algoritmos eficientes para la compresión de imágenes digitales sin pérdidas.

Como consecuencia natural aparecen una cantidad creciente de aplicaciones que producen series de imágenes fuertemente correlacionadas temporalmente y que requieren una compresión sin pérdidas de estas secuencias.

Una de ellas es el cine digital, donde la ausencia de pérdidas debe mantenerse en el proceso de adquisición, post-producción, almacenamiento y distribución hasta el cinematógrafo. De nuevo las aplicaciones médicas son importantes, tanto para almacenamiento como para procesamiento remoto.

Sin embargo mucho menos atención se ha prestado a la compresión de video sin pérdidas en comparación a la compresión de video con pérdidas, probablemente debido al enorme potencial económico de estas últimas, teniendo en cuenta las aplicaciones involucradas, principalmente aplicaciones multimedia para el entretenimiento sobre diversos medios de comunicación. Esto es debido a que para el usuario final tanto el audio, video, o las imágenes pueden tolerar ciertas pérdidas, y en la mayoría de los casos, no ser percibidas por el oído u ojo humano.

En otros casos las pérdidas sí pueden ser apreciadas aunque esto no es determinante para la

aplicación en cuestión, como videoconferencia, o video en sistemas móviles. En general cuanto mayor la tolerancia a las pérdidas, mayor es la compresión que puede obtenerse explotando estas limitantes de la percepción humana. Ejemplos de codificación con pérdidas para audio, video e imágenes son por excelencia MP3, MPEG y JPEG respectivamente (ver [3]).

En este contexto surgen las preguntas:

- ¿Cuáles son los niveles de compresión alcanzados por los métodos actuales de codificación de video sin pérdidas?
- ¿Cuál es el costo computacional asociado a cada método? (Cuantificado a través de los retardos introducidos en los procesos de compresión y de-compresión)

Con el propósito de hallar una respuesta a estas cuestiones es que surge la idea de este proyecto, dado el interés académico en el tema del tutor del mismo.

1.2. Objetivos del proyecto

De acuerdo a lo estipulado en el Plan del Proyecto, aprobado en el segundo semestre de 2005, en este proyecto se busca obtener un sistema de compresión de video sin pérdidas, separando el equipo de captura del equipo encargado del procesamiento de datos y comunicando los mismos sobre una red de área local o *LAN* (de la sigla en inglés Local Area Network) tipo Ethernet Rápido estándar IEEE 802.3u [4]).

Por tanto, al finalizar el proyecto se tendrá implementado, en forma eficiente, un algoritmo de compresión de video sin pérdidas y un sistema de transmisión por *streaming* (ver capítulo 6) para procesamiento en hosts separados. Esta implementación tendrá la forma de software *servidor* y software *cliente*. Los *hosts* donde resida el software servidor y el software cliente serán computadores tipo *PC* y estarán conectados mediante una LAN tipo Ethernet Rápido de 100 Mbps.

Desde este momento es importante resaltar dos puntos que condicionan el desarrollo del sistema en todas sus etapas. Uno de ellos es el de la *eficiencia* nombrada en el párrafo anterior. La eficiencia a la que hacemos referencia es la rapidez con que puedan ejecutarse los procesos de compresión, streaming y de-compresión para un cuadro.

Así, una medida de la eficiencia de estos procesos, y por ende del éxito del proyecto, será la capacidad del sistema de trabajar en tiempo real para unos valores de profundidad (ver 2.1.1), tamaño y tasa de adquisición de cuadros(ver 2.1.2) determinados. Esto se traduce en que una vez implementado el sistema deberá evaluarse el rendimiento del mismo contestando las siguientes cuestiones:

- Dado un tamaño de cuadro y una profundidad, ¿Cuántos cuadros por segundo puede procesar el sistema?
- Dado un tamaño de cuadro y una cantidad de cuadros por segundo, ¿Cuál es la profundidad máxima que soporta el sistema?
- Dada una cantidad de cuadros por segundo y una profundidad, ¿Cuál es el máximo tamaño de cuadro con el que el sistema puede trabajar?

El otro punto notablemente condicionante es la necesidad de ausencia de pérdidas en el proceso de compresión y decompresión. Esto significa que píxel por píxel, cuadro por cuadro, el video decodificado por el software cliente deberá ser idéntico en sus valores al codificado por el software servidor.

Además se dotará al software cliente de una interfaz gráfica simple que permita a un usuario ajeno al desarrollo de este proyecto el uso del software cliente.

1.3. Hipótesis de Trabajo

Para la realización del proyecto se asumieron una serie de hipótesis y restricciones en lo concerniente al *marco de trabajo* (generalmente denominado *framework* en el idioma inglés) en el cual se seleccionaría el algoritmo de compresión de video sin pérdidas a implementar, así como también en lo que hace al hardware y software utilizado.

El objetivo de estas limitantes fue guiar la ejecución del proyecto de modo que pudieran cumplirse los objetivos originales, y que a la vez esto se hiciese recorriendo un camino enriquecedor en áreas del perfil Telecomunicaciones de la Ingeniería Eléctrica como la Teoría de la Información aplicada a la Compresión de Imágenes y Video, el Tratamiento de Imágenes por Computador y la Comunicación en Redes de Computadores.

Así, desde el primer momento el marco de trabajo quedó restringido a aquellos algoritmos de compresión de video sin pérdidas que proponen una extensión para video de los algoritmos de compresión de imágenes sin pérdidas LOCO-I JPEG-LS [5, 6] y CALIC [7]. Esta decisión fue tomada en acuerdo con el tutor del proyecto en base a estudios preliminares como los presentados en [8].

Además se decidió utilizar sólo Software Libre, desde el sistema operativo utilizado por los computadores hasta el programa con el cual se generó esta documentación, pasando por entornos de desarrollo, compiladores, bibliotecas de manipulación y procesamiento de imágenes, etc. Por último, el hardware utilizado se limita al que posee el Instituto de Ingeniería Eléctrica de la Facultad de Ingeniería de la Universidad de la República, los integrantes del proyecto y el tutor del mismo.

1.4. Líneas de Trabajo a Desarrollar

El documento comienza con una introducción en donde se comentan algunos conceptos básicos y definiciones. Posteriormente se presenta un resumen de la Teoría de la Información y su aplicación a la compresión de video.

En el siguiente se explora *el estado del arte* de los métodos para la compresión de video sin pérdidas en el marco de trabajo propuesto. Seguidamente se presenta una evaluación comparativa de los mismos. El objetivo fundamental de este capítulo es determinar si alguno de los métodos vistos es apto para ser implementado en un sistema con las características planteadas en los objetivos del proyecto.

En respuesta al resultado de la cuestión anterior, surge el siguiente capítulo, en donde se busca comprender el algoritmo de compresión sin pérdidas de imágenes LOCO-I/JPEG-LS [9, 5, 6] al punto de poder obtener una implementación propia del mismo a partir del estándar que lo define [6].

En el capítulo siguiente se buscará por diferentes caminos obtener una *extensión* de LOCO-I/JPEG-LS a video. Como se verá allí estas extensiones estarán orientadas a explotar la información contenida en cuadros pasados de la secuencia de video con el fin de conseguir mayores niveles de compresión. En el proceso se estudiará la *compensación de movimiento*, forma usual de lograr este objetivo entre los métodos de compresión de video con pérdidas. El capítulo concluye con el algoritmo LOVICO, un método original que logra muy buenos resultados en el

marco de trabajo propuesto.

Una vez concluída la etapa de selección de los algoritmos de compresión de video a implementar dentro del sistema, se pasa a analizar los conceptos fundamentales del *streaming* de video. Se detallan las características posibles para la transmisión de video en una red de paquetes y dentro de éstas, se listan las deseadas para el sistema a implementar, y las limitantes impuestas por un proceso de compresión sin pérdidas del video.

El siguiente capítulo comenta las características principales del software utilizado para el control del dispositivo de captura de video. Este es el último gran bloque del sistema completo, junto a los de compresión y transmisión por streaming.

La implementación de este sistema es descrita en el capítulo siguiente. En el se realiza además un relevamiento del funcionamiento del mismo.

Para finalizar, se presentan en el último capítulo conclusiones globales sobre el desarrollo del proyecto en sí.

Los apéndices presentan las características más importantes del protocolo de transporte TCP y una descripción del software de red implementado.

2 CONCEPTOS BÁSICOS

2.1. Definiciones

2.1.1. Imagen Digital

La *representación digital* de una imagen, corresponde a la discretización de una imagen I en su dominio y a una cuantización en su codominio. Además tanto dominio como codominio son cerrados y acotados. Es decir, la representación digital de una imagen puede definirse como una función $I : \Omega \rightarrow \Gamma$ donde Ω cumple:

$$\Omega = \{(i, j) : 1 \leq i \leq m ; 1 \leq j \leq n\}$$

con $n \in \mathbb{N}$, $m \in \mathbb{N}$, $\Gamma \in \mathbb{N}^b$ con $b \in \mathbb{N}^+$.

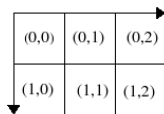
Se denominará *canal i* de la representación digital de una imagen a la i -ésima componente de I , es decir $I_i : \Omega \rightarrow \mathbb{N}$ con $1 \leq i \leq b$.

Estas representaciones son los objetos a los cuales nos referiremos de aquí en más por *imagen digital* o simplemente *imagen*. Así, una imagen digital puede ser considerada como una matriz cuyos índices de filas y columnas identifican a un punto en la imagen y el elemento de la matriz correspondiente identifica el valor de I en dicho punto. Los elementos de dicha matriz son a su vez denominados *elementos de imagen*, o *píxeles* (de, *pixels* derivado de *picture element* en el idioma inglés). Esta forma de representación digital de una imagen es sumamente útil a la hora de implementar procesamiento matemáticos sobre la misma, por lo que será repetidamente utilizada a lo largo de esta documentación. En la figura 2.1 se muestra el modo usual de numerar filas y columnas en esta representación en el campo del tratamiento de imágenes por computador, el cual es empleado en este documento.

Dado que estas imágenes son generadas por dispositivos de aritmética binaria, la cantidad de niveles de cuantización será, en general, una potencia entera de dos.

Se denomina profundidad de píxel a la mínima cantidad de bits necesaria para poder representar el rango completo de valores que un píxel puede tomar. Por ejemplo, para una imagen color de tres canales (esto es, $n = 3$) con 256 niveles de cuantización por canal (lo que corresponde a 8 bits por canal, para cada píxel) la profundidad de píxel será $3 \times 8 = 24$.

2.1.2. Video



(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)

Figura 2.1: Numeración usual de filas y columnas

Se denomina video a una secuencia de imágenes, que llamaremos cuadros, desplegadas a una determinada tasa medida en cuadros por segundo, o *fps* (por su sigla en el idioma inglés Frames Per Second) en forma tal de generar la *ilusión de animación*.

Para lograr un efecto de animación, es sus comienzos y luego de una intensa experimentación, se adoptó como 16 la tasa de refresco, es decir, la cantidad de cuadros que deben exponerse por segundo para dar idea de suavidad a los movimientos presentes en la escena. Sin embargo, cuando las películas comenzaron a mostrar acción a alta velocidad (por ejemplo películas del género *western* o eventos deportivos), la industria del cine decidió incrementar a 24 la cantidad de cuadros por segundo, una tasa que es usada aún hoy.

Desde sus comienzos este fue capturado, almacenado y transmitido exclusivamente en forma analógica, siendo aún hoy una forma común de realizar dichos procesos.

Sin embargo, con los avances recientes en la tecnología de circuitos integrados y en los algoritmos de compresión de imágenes, los sistemas de captura, almacenamiento y transmisión de video digital se han vuelto cada vez más comunes, al punto que se pronostica su predominancia en el futuro.

Como se verá más adelante estas señales consumen grandes cantidades de espacio para su almacenamiento y un amplio ancho de banda para su transmisión, lo que hace necesario la implementación de eficientes algoritmos de compresión para poder hacer económicamente rentable la producción y transmisión de video digital.

Video analógico

Se conoce por *video analógico* a aquel en el cual la información de las imágenes que lo componen está representada mediante una señal continua en amplitud y tiempo.

Una cámara de video analógica convierte la luz que incide a través de su lente en una señal eléctrica que varía con el tiempo según la intensidad y el color de la luz emitida desde los distintos objetos de la escena que se desea registrar. Cada imagen capturada es escaneada en un orden predefinido: comienza por el ángulo superior izquierdo y termina en el ángulo inferior derecho (análogamente al recorrido en la lectura del idioma español).

La elección de la cantidad de líneas escaneadas por imagen es un compromiso entre el ancho de banda, la resolución temporal y la resolución espacial. En forma similar, incrementar la cantidad de imágenes por segundo incrementará la resolución temporal.

Los tres estándares más importantes de video analógico actualmente son *PAL* (de la sigla en inglés Phase Alternating Line, en español línea alternada en fase, ver [10]), *SECAM* (de la sigla en francés Séquentiel Couleur avec Mémoire o Color Secuencial con Memoria en español) y *NTSC* (de la sigla en inglés National Television Systems Committee, Comité de Sistemas de Televisión Nacional, ver [11]).

En Europa y América del Sur los estándares de televisión empleados son PAL y SECAM, que utilizan una frecuencia de refresco de 25 cuadros por segundo, conveniente para transmitir una película en televisión. La película filmada a 24 cuadros por segundo es desplegada a 25, una diferencia indetectable. En Estados Unidos el estándar es NTSC, y la tasa de refresco es 30 cuadros por segundo, aunque cuando el color fue agregado al video esta tasa bajó a 29,97 debido a la necesidad de tener una precisa separación entre las señales de video y audio.

Existen dos tipos de escaneo para señales de video, estos son interlaceado y progresivo. El interlaceado se utiliza como una forma de obtener una buena calidad de imagen dentro de un ancho de banda angosto. Las líneas horizontales de escaneo de cada cuadro interlaceado son numeradas y repartidas en dos campos: el campo impar, constituido por las líneas impares y el campo par compuesto por las líneas pares. NTSC, PAL y SECAM son formatos interlaceados.

Así para PAL tendremos una tasa de campos por segundo de 50 y para NTSC 60.

Por otra parte, en el escaneo progresivo cada periodo de refresco actualiza todas las líneas de la imagen.

Cuando una línea es escaneada, la intensidad de la señal varía en forma continua. La imagen es desplegada línea por línea, pero cada línea es continua. Por lo tanto, la imagen desplegada es muestreada únicamente en la dimensión vertical.

Otra característica a tener en cuenta del video es su relación de aspecto. La relación de aspecto describe las proporciones de ancho y alto de las imágenes que componen el video. En una pantalla de televisión tradicional esta es de 4:3. La televisión de alta definición sin embargo posee una relación de aspecto de 16:9. Durante el escaneo, la cámara genera tres señales de colores primarios: rojo, verde y azul, éstas son las llamadas señales *RGB* (de la sigla en el idioma inglés Red Green Blue). Estas señales son posteriormente procesadas para la transmisión y se genera un nuevo espacio de color. Tanto PAL, SECAM como NTSC son estándares de codificación de color. En éstos existe una componente con información de luminancia y dos con información de color o cromatismo. NTSC utiliza componentes YIQ que están definidas como:

$$\begin{cases} Y = 0.299R + 0.587G + 0.114B \\ I = 0.596R - 0.274G - 0.322B \\ Q = 0.211R - 0.523G + 0.311B \end{cases} \quad (2.1)$$

PAL utiliza el espacio de colores YUV:

$$\begin{cases} Y = 0.299R + 0.587G + 0.114B \\ U = -0.147R - 0.289G + 0.436B \\ V = 0.615R - 0.515G - 0.100B \end{cases} \quad (2.2)$$

SECAM por su parte utiliza YDrDb:

$$\begin{cases} Y = 0.299R + 0.587G + 0.114B \\ Db = -0.450R - 0.833G + 1.333B \\ Dr = -1.333R + 1.116G - 0.217B \end{cases} \quad (2.3)$$

Luego estas tres componentes son multiplexadas en una única señal (junto con información de sincronismo). Esto se denomina *video compuesto*.

Video digital

Se entiende por *video digital* aquel compuesto por imágenes digitales, de acuerdo a cómo se definieron en 2.1.1. Al igual que para el video analógico, la tasa de cuadros así como el tamaño de los mismos dependen de la aplicación. En video digital se suma además la profundidad de píxel de los cuadros que componen el video. En la tabla 2.1 se listan a modo de ejemplo algunas aplicaciones junto con estas características.

El concepto de relación de aspecto del píxel también debe ser mencionado. Usualmente se piensa en un píxel como un punto matemático, sin dimensiones ni forma. Si bien este concepto es sumamente útil a la hora de formalizar teorías que involucran procesamiento de imágenes, esto obviamente no es cierto para los dispositivos de captura o despliegue de imágenes donde, por ejemplo, los píxeles en un monitor de computadora son cuadrados o rectangulares creando imágenes con apariencia filosa. Como se verá más adelante, muchos algoritmos de compresión de

Aplicación	Cuadros por Segundo	Tamaño de Cuadro	Profundidad de Píxel
Vigilancia	5	640 x 480	12
Video Teléfono	10	320 x 240	12
Multimedia	15	320 x 240	16
TV Analógica	25	640 x 480	16
HDTV	60	1280 x 720	24

Tabla 2.1: Algunas aplicaciones y sus características Fuente: [1]

imágenes basados en *predicción* del valor de un píxel asumen imágenes compuestas por píxeles cuadrados, pequeños en comparación a los detalles presentes, y de idénticas dimensiones entre sí.

En comparación al video analógico, el video digital presenta las siguientes ventajas:

- Es fácilmente editable. Esto es especialmente útil dentro de la cinematografía en el campo de los “efectos especiales”. Por ejemplo, imágenes generadas por computador son combinadas con imágenes de actores produciendo situaciones complejas de aspecto real. Además existe hoy en día una gran cantidad de software para computadores personales que involucra edición de video digital con diferentes motivos, como mejoramiento, restauración o mezcla.
- Es posible almacenarlo en cualquier medio digital, como discos duros, memorias extraíbles, CD-ROMs o DVDs. Métodos para la corrección o detección de errores pueden ser utilizados (de ser necesarios), lo que hace posible la copia de grandes cantidades de datos sin pérdidas. Esto se traduce en una calidad idéntica entre original y copia, a diferencia del video analógico en el cual cada copia sucesiva es de calidad inferior al original.
- Puede ser *comprimido*. Esto tiene como consecuencias aumento de la capacidad de almacenamiento y transmisión de un determinado video, lo cual abre un enorme campo de aplicaciones, como comunicaciones multimedia (video teléfono, video conferencia o televisión por Internet). Del mismo modo hace posible aumentar la cantidad de videos a transmitir por un determinado medio físico o el tamaño de cuadro de los mismos.

Si fijamos nuestra atención en el la tabla 2.1, la necesidad del punto anterior se vuelve evidente. Por ejemplo, la aplicación más simple genera $5 \times 640 \times 480 \times 12 = 18,432,000$ bits por segundo (o 2,304,000 bytes por segundo) mientras que la más compleja $60 \times 1280 \times 720 \times 24 = 1,327,104,000$ bits por segundo (o 165,888,000 bytes por segundo).

A su vez, en la tabla 2.2 se muestran las tasas de bits o *bit rates* para tecnologías de WAN, (de la sigla en inglés de Red de Área Amplia) LAN, y WLAN¹ (de la sigla en inglés Red de Área Local Inalámbrica).

Comparando la capacidad de las tecnologías enumeradas y la tasa de bits generada por la transmisión de video digital “crudo”, se comprende el por qué de la necesidad de compresión.

Formatos de video digital

Un formato ampliamente utilizado en la codificación de señales de video se define en [12]. En éste, la componente de luminancia es muestreada a 13.5 MHz y los cromas a 6.75 MHz para

¹Tasas de bits típicas y máximas dentro de la distancia adecuada

Tecnología de:	Tasa de bits
WAN	Módem V.32 9.6 kbps, V.34 28,8 kbps, V.90 56 kbps
WAN	ISDN con dos canales de 64 kbps
WAN	ADSL desde 128 kbps hasta 8 Mbps
LAN	IEEE 802.3a 10 Mbps Sobre Coaxial
LAN	IEEE 802.3i 10 Mbps Sobre Par trenzado
LAN	IEEE 802.3j 10 Mbps Sobre Fibra óptica
LAN	IEEE 802.3u 100BASE-TX,-T4,-FX Ethernet Rápido 100 Mbps
LAN	IEEE 802.3z Sobre Fibra Óptica a 1 Gbps
LAN	IEEE 802.3ab Sobre par trenzado a 1 Gbps
WLAN	IEEE 802.11b 6.5 Mbps típico, 11 Mbps máximo
WLAN	IEEE 802.11g 25 Mbps típico, 54 Mbps máximo
WLAN	IEEE 802.11n 200 Mbps típico 540 Mbps máximo

Tabla 2.2: Algunas tecnologías de WAN, LAN y WLAN y el bit rate que ofrecen

	30 fps	25 fps
Campos por segundo	60	50
Líneas por cuadro	525	625
Líneas activas	480	576
Bits por muestra	8	8
Tasa de bits total	216 Mbps	216 Mbps
Muestras por línea (Y)	720	720
Muestras por línea (Cr,Cb)	360	360

Tabla 2.3: Parámetros establecidos por ITU-R BT.601-5

producir una señal YCrCb 4:2:2 (ver tabla 2.4). Los parámetros que se establecen de la señal digital dependen de la tasa de cuadros del video (25 o 30) y se muestran en la tabla 2.3. La tasa de 25 cuadros es empleada en Europa, mientras que la de 30 cuadros por segundo es utilizada en Norte América.

Cada muestra de la luminancia tiene un rango posible de valores entre 0 y 255, sin embargo el nivel 0 y el 255 son reservados para sincronismo. Por lo tanto, la señal de luminancia es restringida a un rango de 16 (negro) a 235 (blanco).

Este formato se utiliza en imágenes con calidad necesaria para edición de video profesional, sin embargo, para otras aplicaciones puede ser necesario un menor grado de resolución. Por ejemplo, en videoconferencia o videotelefonía se requiere un menor tamaño de imagen para satisfacer las exigencias del ancho de banda.

Dado que la mayoría de la fuentes de video siguen la recomendación [12] es lógico pensar en relacionar los diferentes formatos de cuadro con el definido por [12].

Algunos de los formatos estándares de video digital a los que se hará referencia en el resto de este documento se citan a continuación:

SIF (Source Input Format)

Una forma natural de disminuir la resolución de imagen de [12] sería una secuencia de imágenes con la mitad de resolución en cada dirección. Esto es para cada estándar de ITU-R 601 (es decir el europeo de 50 Hz y el norteamericano de 60 Hz), reducir a la mitad la parte activa de la imagen en el sentido vertical, horizontal y temporal. El resultado es una imagen no interlaceada. De este modo para el estándar europeo, el formato SIF tendrá 352 píxeles por línea, 288 líneas por cuadro y 25 cuadros por segundo (12.5, 8.3 son aceptables para aplicaciones de ancho de banda reducido). Para el estándar norte americano los valores serán 352 píxeles por línea, 240 líneas por cuadro y 30 cuadros por segundo (15, 10 o 7.5 en aplicaciones de ancho de banda reducido).

Es interesante comentar que el número 352 es derivado de 360, que es a su vez la mitad de la cantidad de píxeles por línea de [12]. El porque de elegir 352 sobre 360 se halla en que 352 es múltiplo de 16, no así 360. Este hecho hace que muchos procedimientos matemáticos se simplifiquen para 352 píxeles por línea, lo que justifica su elección.

CIF (Common Intermediate Format)

Para poder realizar una videoconferencia a lo largo y ancho del mundo se entendió necesario contemplar de alguna manera tanto el estándar europeo (625 líneas, 50 Hz) como el norteamericano (525 líneas, 60 Hz). Así, las fuentes de video que cumplen con [12] se deben convertir a un formato común. Surge de este modo el formato CIF.

Tanto para el estándar europeo como el norte americano, el formato CIF presenta 352 píxeles por línea, 288 líneas por cuadro y 30 cuadros por segundo (exactamente 29.97).

SubQCIF, QCIF y QSIF

Para determinadas aplicaciones como video en redes móviles o videotelefonía, es posible reducir la tasa de cuadros por segundo. En el caso de CIF y SIF-525, son aceptables 15, 10 y 7.5 cuadros por segundo, para SIF-625 estas tasas son 12.5 y 8.3 cuadros por segundo.

Para este tipo de aplicaciones también es factible realizar una reducción en la resolución espacial del cuadro. Es así que surgen los formatos QSIF, QCIF y SubQCIF.

El cuadro SubQCIF se compone de 128 píxeles por línea, 96 líneas por cuadro, QCIF de 176 píxeles por línea, 144 líneas por cuadro y QSIF para el estándar europeo 352 píxeles por línea, 288 líneas por cuadro mientras que para el estándar americano 352 píxeles por línea y 240 líneas por cuadro.

HDTV

Este formato presenta el doble de resolución que [12]. También utiliza un ancho de banda mayor para la crominancia, pudiendo llegar a la mitad o a lo mismo que la luminancia. De este modo se utilizan formatos de imagen (ver tabla 2.4) 4:2:2 o 4:4:4. En la mayoría de los casos HDTV presenta un escaneo progresivo de la imagen para mejorar la resolución vertical.

Los formatos usuales de HDTV en modo progresivo presentan 1280 píxeles por línea, 720 líneas y 60 cuadros por segundo, mientras que en modo interlaceado poseen 1920 píxeles por línea, 1080 líneas y una tasa de 50 campos por segundo.

Formato de imagen	Horizontal(%)	Vertical (%)
4:4:4	100	100
4:2:2	50	100
4:2:0	50	50
4:1:1	25	100

Tabla 2.4: Porcentaje de resolución de cromas respecto a la luminancia en las direcciones vertical y horizontal

Dígito decimal	Representación binaria
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Tabla 2.5: Codificación binaria de los dígitos decimales

2.2. Teoría de la Información

La Teoría de la información es una rama de la matemática que se dedica al estudio de la información y todo lo relacionado con ella. Sus orígenes datan del artículo publicado en 1948 en el Bell Systems Technical Journal por Claude E. Shannon, al cual tituló “Una teoría matemática de la comunicación” [13].

En dicho trabajo Shannon extiende el trabajo de Hartley y Nyquist en busca de una teoría matemática general de la comunicación, estudiando la forma en que se deben representar los mensajes producidos por una fuente para obtener una comunicación confiable sobre un canal con limitaciones físicas.

Para introducir el concepto de teoría de la información es necesario definir algunos conceptos previos.

Se define *fente de información* a cualquier proceso que genere una secuencia de símbolos. Dados $S = s_1, s_2, \dots, s_q$, $X = x_1, x_2, \dots, x_r$ dos conjuntos de símbolos, se define un *código*, como la correspondencia de todas las secuencias posibles de símbolos de S a secuencias de símbolos de X . Se denomina S *alfabeto fuente* y X *alfabeto código*.

Veamos como ejemplo la codificación binaria de los dígitos decimales. En este caso tenemos un alfabeto fuente $S = 0, 1, \dots, 9$ y un alfabeto código $X = 0, 1$, donde a cada símbolo del alfabeto fuente S le corresponde una secuencia de símbolos del alfabeto código X . A dicha secuencia se le denomina *palabra código*.

Por otra parte se define *código bloque* como aquel donde a cada símbolo del alfabeto fuente S le corresponde una secuencia fija de símbolos del alfabeto código X .

Un código bloque se dice *no singular* si todas las palabras código son distintas. Es decir a símbolos fuentes distintos le corresponden distintas palabras código.

Es evidente que nos interesa que un código sea no singular para obtener una decodificación única del mismo. Sin embargo esta es una condición necesaria, pero no suficiente. Aún cuando todas las palabras de un código sean diferentes puede ocurrir que para dos secuencias distintas de símbolos de la fuente le corresponda la misma secuencia de símbolos del alfabeto código.

Por lo tanto, lo que se busca es que a dos secuencias cualquiera de símbolos de la fuente de igual longitud de código, le correspondan secuencias de símbolos de alfabeto código distintas. Para que esto se cumpla el código debe ser *unívocamente decodificable*. Este concepto se define a partir del de *extensión de orden n* de un código bloque.

La extensión de orden n de un código bloque que hace corresponder los símbolos s_i con las palabras código X_i , es el código bloque que hace corresponder las secuencias de símbolos de la fuente $s_{i_1}, s_{i_2}, \dots, s_{i_n}$, con las secuencias de palabras código $X_{i_1}, X_{i_2}, \dots, X_{i_n}$.

Así, un código bloque es unívocamente decodificable si y solo si su extensión de orden n es no singular.

Se define la *propiedad de prefijo* como la que establece que dada un cierto patrón de bits asignado al código de un símbolo, ningún otro código debe comenzar con ese patrón (el patrón no puede ser el prefijo de ningún otro código).

2.2.1. Información

La información que contiene un símbolo determinado, depende fuertemente tanto del interlocutor como de la probabilidad de ocurrencia del evento representado por el mismo. Supongamos que queremos transmitir el estado del tiempo clasificándolo en soleado, lluvia, nublado. Si transmitimos el mensaje lluvia, este mensaje ocasionará un efecto en un interlocutor que se encuentra en el desierto de Sahara y otro efecto totalmente distinto en un interlocutor que se encuentra en la cuenca del río Amazonas. Para el interlocutor que se encuentra en el Sahara este mensaje aporta mucha información, ya que es un evento improbable; mientras que para el interlocutor que se encuentra en el Amazonas este mensaje no aporta información alguna. En síntesis, cuanto más inesperado o improbable sea un evento, más información aporta. Sin embargo, si tenemos la certeza de la ocurrencia del mismo, la información aportada será nula.

Denotamos por $P(E)$ la probabilidad de ocurrencia de un evento e $I(E)$ la *información* aportada por el mismo. A partir del ejemplo anterior, sería deseable que $I(E)$ cumpla:

$$\begin{cases} P(E) = 0 \Rightarrow I(E) = \infty \\ P(E) = 1 \Rightarrow I(E) = 0 \\ P(E_1) < P(E_2) \Rightarrow I(E_1) > I(E_2) \end{cases} \quad (2.4)$$

2.2.2. Definición de información

Sea E un suceso que puede presentarse con una probabilidad $P(E)$. Cuando E tiene lugar, se dice que hemos recibido $I(E)$ unidades de información. De acuerdo a las condiciones deseables para $I(E)$ establecidas en 2.4, se observa que la función *logaritmo* posee las propiedades adecuadas para definir $I(E)$. Esto porque:

$$\begin{cases} \log(x) \longrightarrow \infty \text{ cuando } x \longrightarrow \infty \\ \log(1) = 0 \\ \log(x_1) < \log(x_2) \text{ cuando } x_1 < x_2 \end{cases} \quad (2.5)$$

En base a las ecuaciones 2.5, si tomamos $x = \frac{1}{P(E)}$ y recordamos la propiedad del logaritmo $\log\left(\frac{a}{b}\right) = \log(a) - \log(b)$ llegamos a que 2.6 cumple con 2.4, y por lo tanto es tomada como definición matemática de información.

$$I(E) = \log\left(\frac{1}{P(E)}\right) \quad (2.6)$$

La elección de la base del logaritmo que interviene en la definición equivale a seleccionar una determinada unidad para *medir* la cantidad de información. Si utilizamos logaritmo en base 2, la unidad correspondiente se denomina “bit”, si utilizamos logaritmo en base e la unidad de información es el “nats” y “Hartley” si el logaritmo utilizado es en base 10.

2.2.3. Definición de Entropía

Definimos una fuente discreta estacionaria y sin memoria como una fuente donde las probabilidades de los símbolos se mantienen constantes en el tiempo y los símbolos sucesivos son estadísticamente independientes. Dicha fuente puede describirse completamente mediante el alfabeto fuente $S = s_1, s_2, \dots, s_q$ y las probabilidades con que los símbolos se presentan $P(s_1), P(s_2), \dots, P(s_q)$.

La información media por símbolo de la fuente recibe el nombre de Entropía $H(S)$ y se calcula la forma siguiente:

$$H(S) = \sum_S P(S_i) I(S_i) \quad (2.7)$$

Hasta este punto se trabajó bajo la hipótesis de fuente sin memoria, es decir que cada símbolo emitido por la fuente es independiente de los anteriores. Si levantamos dicha hipótesis, surge como elemento útil el de la fuente de Markov. Se denomina fuente de Markov de orden m a aquella en la cual la presencia de un símbolo determinado depende de un número m de símbolos precedentes. Por lo tanto la probabilidad de presencia de un símbolo cualquiera está determinada por los m símbolos que lo preceden:

$$P(s_i/s_{i-m}, \dots, s_{i-1}) \text{ para } i = m + 1, \dots, \infty \quad (2.8)$$

Siendo q la cantidad de símbolos de la fuente. De esta forma la cantidad media de información por símbolo viene dada por la siguiente ecuación:

$$H(S/s_{i-m}, \dots, s_{i-1}) = \sum_S P(s_{i-m}, \dots, s_{i-1}) I(s_i/s_{i-m}, \dots, s_{i-1}) \quad (2.9)$$

Calculando el valor medio de esta cantidad se obtiene la cantidad media de información o entropía de la fuente de Markov de orden m :

$$H(S) = \sum_{S^m} P(s_{i-m}, \dots, s_{i-1}) H(S/s_{i-m}, \dots, s_{i-1}) \quad (2.10)$$

Para un alfabeto fuente, es posible elaborar más de un código instantáneo unívocamente decodificable. Entonces surge la pregunta ¿cuál de los posibles códigos es el más eficiente?. Una forma intuitiva de selección es de la longitud de las palabras. Si estamos ante una fuente cuyo

símbolos son equiprobables, es preferible utilizar un código formado por palabras cortas a uno que tenga palabras de gran longitud. Por lo tanto nos vemos en la necesidad de definir longitud media de código.

Se define redundancia de un conjunto de símbolos como la diferencia entre la entropía más grande posible y su entropía real.

$$R = \log_2 n + \sum_{i=1}^n P_i \log_2 P_i \quad (2.11)$$

Donde: n es la cantidad total de símbolos

P_i es la probabilidad del símbolo i

2.2.4. Definición de Longitud media del Código

Sea S una fuente que emite los símbolos $s_1, s_2, \dots, s_q$, con probabilidad de ocurrencia $P_{s_1}, P_{s_2}, \dots, P_{s_q}$ respectivamente. Y sean $X_1, X_2, \dots, X_q$ las palabras código asociadas a dichos símbolos, con $l_1, l_2, \dots, l_q$ las longitudes de las palabras. Definiremos longitud media del código L como:

$$L = \sum_{i=1}^q P_{s_i} l_i \quad (2.12)$$

Ahora la pregunta que surge es : Dado un alfabeto fuente y un alfabeto código de datos, ¿cuál es la menor longitud media posible?. Shannon respondió esta pregunta en su primer teorema.

Primer teorema de Shannon

$$H(S) \leq L_{min} \quad (2.13)$$

Siendo:

- L_{min} , la menor longitud media alcanzable por un código cualquiera dada la fuente.
- $H(S)$ la entropía de la fuente.

El primer Teorema de Shannon nos dice que la longitud media esta acotada por la entropía de la fuente. Es decir la entropía nos indica el límite teórico para la compresión de datos.

2.2.5. Entropía y Videos Digitales

La teoría de la información nos brinda las herramientas básicas para obtener un modelo cuantitativo de la información contenida en las secuencias de video. Esta representación nos permite hallar una estimación del valor máximo posible para la compresión dado por el primer teorema de Shannon.

En primera instancia debemos establecer un modelo probabilístico de la imagen y en base a éste modelo hallar la entropía de la fuente.

15	15	142	142	31	25	25	25
15	15	142	142	31	25	25	25
15	15	142	142	31	25	25	25
15	15	142	142	31	25	25	25
15	15	142	142	31	25	25	25

Tabla 2.6: Imagen de 8 bpp

Nivel de Gris	Cantidad	Probabilidad
15	10	0,250
25	15	0,375
31	5	0,125
142	10	0,250

Tabla 2.7: Ocurrencias y probabilidades estimadas de los símbolos que forman la imagen en 2.6

Estimación de orden cero

Asumimos que la imagen es producida por una fuente que emite símbolos secuenciales y estadísticamente independientes. En esta representación los símbolos son los niveles de gris, si asumimos que trabajamos con 8 bpp entonces el alfabeto está compuesto por 256 símbolos posibles. Por ejemplo, si suponemos que los símbolos de la fuente son equiprobables entonces utilizando la ecuación 2.7 la entropía de la fuente tiene un valor de 8 bpp. En este caso no hay compresión posible, ya que la longitud media está limitada por la entropía.

Un método para estimar la entropía es construir un modelo de la fuente basado en la cantidad de ocurrencias de un píxel de determinado nivel de gris dentro de la imagen en cuestión. Podemos modelar las probabilidades de cada símbolo fuente usando un histograma de los valores de nivel de gris en la imagen. Así, para la imagen de 8 bpp esquematizada en la tabla 2.6:

Obtendríamos la tabla 2.2.5 con los valores de cantidad de ocurrencias y probabilidades correspondientes a cada símbolo:

Esta es una estimación de orden cero y la entropía de la fuente puede ser calculada aplicando la ecuación 2.7. El resultado que así se obtiene para este ejemplo es 1,906 bpp, equivalente a un total de 77 bits. La cantidad total de bits en la imagen no comprimida es de $5 \times 8 \times 8 = 320$ bits. La reducción en el número de bits es de $320 - 77 = 243$ bits, lo que pone de manifiesto la presencia de redundancia entre los píxeles de la imagen.

Esto implica que si utilizamos un código de longitud variable, el código óptimo debería aproximarse arbitrariamente a este valor de entropía.

Estimación de orden uno

Otra estimación de la entropía de la fuente que genera la imagen puede ser hallada examinando la frecuencia de los bloques de símbolos, donde un bloque es un grupo de píxeles adyacentes.

Si se supone que la imagen se forma de línea a línea y de fin a principio, es posible hallar la frecuencia relativa de pares de píxeles, correspondiente a una fuente de Markov de orden uno.

Para la imagen ejemplo 2.6, las probabilidades correspondientes a los bloques de 2 píxeles, se

Pares de Nivel de Gris	Cantidad	Probabilidad
(15,15)	5	0,125
(15,142)	5	0,125
(142,31)	5	0,125
(142,142)	5	0,125
(31,25)	5	0,125
(25,25)	10	0,250
(25,15)	5	0.125

Tabla 2.8: Ocurrencias y probabilidades estimadas de los pares de píxeles que forman la imagen en 2.6

detallan en la tabla 2.8. El resultado de la estimación de la entropía es $1,625/2$ bit/símbolo, o lo que es lo mismo $0,8125$ bpp, donde la división entre dos viene dada por considerar símbolos de píxeles. Esta última aproximación es llamada estimación de primer orden de la entropía de la fuente.

La estimación anterior equivale a 33 bits, lo que implica una reducción respecto a la estimación de orden cero de $77 - 33 = 44$ bits.

2.2.6. ¿Qué entendemos por compresión de video sin pérdidas?

Se define compresión como la acción de codificar la información de forma tal de obtener menos bits que en su forma original.

Compresión de video se refiere a la reducción en la cantidad de bits necesaria para representar el contenido del video. En caso de ser una compresión sin pérdidas el proceso de compresión y de-compresión es reversible debido a que la información comprimida es la misma que la original. La utilidad de la compresión radica en el uso eficiente de recursos tales como espacio de almacenamiento o ancho de banda al transmitir.

La correlación existente entre píxeles de iguales coordenadas espaciales en imágenes sucesivas de una secuencia de video origina la denominada *redundancia temporal*, mientras que la existente entre píxeles cercanos de una misma imagen genera la llamada *redundancia espacial*. Además, si el video en cuestión está representado por más de un canal, existe también una correlación entre píxeles de iguales coordenadas espaciales en canales diferentes, esto produce lo que se conoce como *redundancia espectral*.

La compresión de video se logra a través de un proceso de codificación que explota estos tres tipos de redundancia. Cuando dicha compresión se realiza de modo tal que la secuencia de video decodificada es idéntica píxel por píxel, cuadro por cuadro, a la secuencia original, decimos que se ha efectuado una *compresión sin pérdidas*.

Existen otros métodos de compresión que explotan además de estas propiedades la *respuesta psico-visual* del cerebro humano, alcanzando mayores tasas de compresión. La idea tras este procedimiento es descomponer la imagen en sus distintas componentes de frecuencia, y conociendo la respuesta del ojo humano, se hace posible eliminar o codificar con menor precisión las componentes donde el ojo no es sensible. Como consecuencia, este aumento en la compresión lleva consigo una pérdida de información irre recuperable.

Como se mencionó anteriormente según cual sea el área de trabajo será el compromiso entre calidad de imagen y compresión deseado.

3 COMPRESIÓN DE VIDEO

Este capítulo está constituido por todo lo concerniente al algoritmo de compresión y de-compresión sin pérdidas de video a utilizar en el sistema a implementar. El mismo comienza naturalmente con un estudio del “estado del arte” en la compresión de video sin pérdidas dentro del marco de trabajo propuesto.

Se verá a través de la descripción de los algoritmos I-CALIC, M-CALIC [14] y LOPT-3D [15] que es un área relativamente nueva, en donde los esfuerzos de investigación buscan extender los conocidos métodos para la compresión de imágenes sin pérdidas CALIC[7, 16] y LOCO-I/JPEG-LS[6, 5, 9] a video.

Luego de comprender su funcionamiento es fundamental determinar si alguno de ellos es adecuado para ser implementado en un sistema con las características planteadas en los objetivos del proyecto. La respuesta a esta pregunta será encontrada al final del capítulo.

3.1. Estado del Arte en la Compresión de Video sin Pérdidas

En esta sección se describen las características principales de los algoritmos de compresión de video sin pérdidas I-CALIC, M-CALIC [14], y LOPT-3D [15].

Dado que los algoritmos I-CALIC y M-CALIC proponen *extensiones* a video del método de compresión de imágenes CALIC [7] es prudente analizar las características de éste en primera instancia.

3.1.1. CALIC

Este método de compresión CALIC [7] ha sido desarrollado para compresión de imágenes en escala de grises, pero puede ser extendido a imágenes color separando las componentes de dicha imagen y tratando a cada una de ellas como una imagen en escala de grises.

Su nombre *CALIC* surge de la sigla en inglés Context-based Adaptive Lossless Image Compression (Compresión de Imágenes sin Pérdidas Adaptativa basada en Contextos).

La mayoría de los métodos actuales de compresión de imágenes sin pérdidas codifican los píxeles de izquierda a derecha y de arriba a abajo. Esto lleva a que los contextos disponibles para el modelado no puedan “encerrar” al píxel actual. Sólo los píxeles arriba y a la izquierda del píxel a ser predicho son conocidos tanto por el codificador como el decodificador. Este tipo de configuración espacial del modelado de contextos es del tipo 180° , ya que los píxeles involucrados forman un semi-círculo alrededor del píxel a codificar. Muchas características de las imágenes como gradientes de intensidad, orientación de bordes y texturas, pueden ser mejor modeladas mediante contextos que rodeen al píxel. Este tipo de configuración es del tipo 360° . CALIC se basa en la utilización de tres *pasadas*, tanto en la codificación como en la decodificación, para crear un contexto simétrico (360°) alrededor del píxel actual, utilizando cuantización

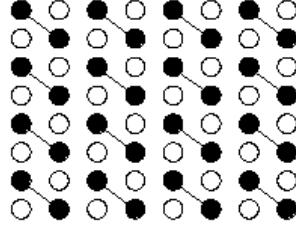


Figura 3.1: Píxeles involucrados en los promedios de la primera pasada

para reducir el número de posibles contextos a un tamaño manejable.

Descripción de las pasadas

Se considerará una imagen $I[i, j]$ constituida por H filas y W columnas de píxeles. Tanto el codificador como el decodificador realizarán tres pasadas sobre la imagen. En la primera pasada se calculan promedios de pares de píxeles. Se trabaja sólo con los píxeles $I[i, j]$ tales que i y j tengan la misma paridad, es decir, ambos sean pares o impares. La segunda pasada utiliza esos promedios para codificar cada uno de los píxeles involucrados en los promedios. La tercera pasada, utiliza los promedios más los píxeles de la segunda pasada para codificar los píxeles $I[i, j]$ tales que i y j tienen diferente paridad.

Primera pasada

En la primera pasada se calculan los $W/2 \times H/2$ valores $\mu[i, j]$ definidos como:

$$\mu[i, j] = \lfloor \frac{I[2i, 2j] + I[2i + 1, 2j + 1]}{2} \rfloor \text{ para } 0 \leq i < H/2, 0 \leq j < W/2 \quad (3.1)$$

El primer índice indica filas y el segundo columnas. Cada $\mu[i, j]$ es por tanto el promedio de dos píxeles diagonalmente adyacentes. En la figura 3.1 se muestran con círculos negros los píxeles involucrados en el cálculo para una imagen de 8×8 píxeles. Cada par utilizado para calcular un promedio se conecta con una línea.

Se debe tener en cuenta que los dos píxeles originales no se pueden reconstruir completamente con el promedio debido a 1 bit que puede perderse en la división entre 2. Los valores calculados $\mu[i, j]$ son considerados como píxeles de una nueva imagen de tamaño $W/2 \times H/2$. Esta nueva imagen es recorrida de la forma habitual y cada píxel es predicho por cuatro de sus vecinos, utilizando un modelado de 180° , tres píxeles centrados por encima y uno a la izquierda. Si $x = \mu[i, j]$ es el píxel actual, este es predicho por la cantidad:

$$\hat{x} = \frac{1}{2}\mu[i - 1, j] + \frac{1}{2}\mu[i, j - 1] + \frac{1}{4}\mu[i - 1, j + 1] - \frac{1}{4}\mu[i - 1, j - 1] \quad (3.2)$$

donde los coeficientes $1/2$, $1/4$, y $-1/4$ así como los coeficientes a ser usados en las siguientes pasadas fueron determinados previamente por una regresión lineal utilizando un conjunto de imágenes de entrenamiento. La idea de este entrenamiento es encontrar los coeficientes a_k que ofrezcan la mejor compresión para esas imágenes, luego son redondeados a un entero potencia de 2 (para simplificar los cálculos) y utilizados posteriormente en el algoritmo.

Luego de hallado el valor de $\hat{x}$ se codifica el valor del error $e = x - \hat{x}$ y se continúa con la siguiente pasada.

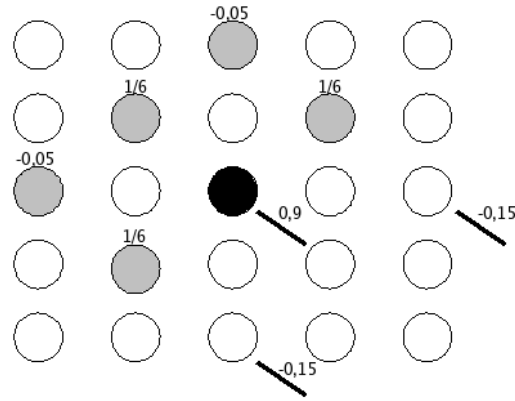


Figura 3.2: Ponderaciones asignadas a los píxeles involucrados y promedios de la segunda pasada

Segunda pasada

La segunda pasada involucra los mismos píxeles que en la primera (por lo tanto la mitad de los contenidos en la imagen original), pero en esta pasada los píxeles serán predichos individualmente a partir de los promedios.

En esta etapa son conocidos de la imagen μ los píxeles ubicados a la derecha y abajo del píxel actual, y de la etapa actual los ubicados a la izquierda y arriba, permitiendo la creación de un modelado de contexto de 360° con la información de las dos pasadas.

Asumiendo que $x = I[2i, 2j]$ indica el píxel actual, este es predicho utilizando cinco vecinos conocidos por encima de él y a la izquierda, y tres promedios μ , por debajo y a la derecha:

$$\begin{aligned} \hat{x} = & 0.9\mu[i, j] + \frac{1}{6} (I[2i + 1, 2j - 1] + I[2i - 1, 2j - 1] + I[2i - 1, 2j + 1]) \\ & - 0.05 (I[2i, 2j - 2] + I[2i - 2, 2j]) - 0.15 (\mu[i, j + 1] + \mu[i + 1, j]) \end{aligned} \quad (3.3)$$

En la figura 3.2 se muestran los cinco píxeles (círculos grises) y los tres promedios (líneas negras) involucrados en el cálculo.

El valor $I[2i + 1, 2j - 1]$ es conocido por el decodificador gracias a la ecuación 3.4 y al valor ya conocido de $I[2i, 2j - 2]$.

Los coeficientes del predictor se obtienen por regresión lineal utilizando el mismo conjunto de imágenes de entrenamiento ya mencionado en la primera pasada.

Debido a la información aportada por los promedios se tiene que $I[2i, 2j]$ puede ser predicho con una mayor exactitud que utilizando una pasada simple. Esto explica el por qué de la primera pasada.

Una vez que $I[2i, 2j]$ es reconstruido, el decodificador puede hallar:

$$I[2i + 1, 2j + 1] = 2\mu[i, j] - I[2i, 2j] \quad (3.4)$$

Se obtiene dicho valor sin necesitar más información de $I[2i + 1, 2j + 1]$ que la dada en la propia ecuación. Observando la ecuación 3.1, se revela que 3.4 puede causar un error de truncamiento de 1 si la suma $I[i, j] + I[2i + 1, 2j + 1]$ es impar.

Según [7] un bit puede ser utilizado para recuperarse de este error de truncamiento ó este “error mínimo y espacialmente aislado” puede ser ignorado.

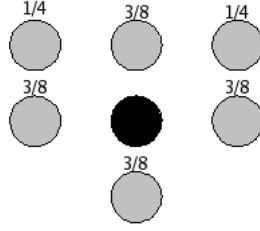


Figura 3.3: Ponderación de píxeles en la tercer pasada

Luego de hallado el valor de $\hat{x}$, se codifica el valor del error $e = x - \hat{x}$ para cada píxel x en este subconjunto.

Tercera pasada

La tercera pasada involucra a los píxeles restantes:

$$I[2i, 2j + 1] \text{ y } I[2i + 1, 2j], \text{ para } 0 \leq i < H/2, 0 \leq j < W/2 \quad (3.5)$$

Cada uno es predicho en un contexto del tipo 360° casi simétrico de seis píxeles (ver figura 3.3) constituido por los 4 vecinos del píxel actual más dos vecinos diagonales. Si $x = I[i, j]$ es el píxel actual, su predicción es:

$$\begin{aligned} \hat{x} = & \frac{3}{8} (I[i, j - 1] + I[i - 1, j] + I[i, j + 1] + I[i + 1, j]) \\ & - \frac{1}{4} (I[i - 1, j - 1] + I[i - 1, j + 1]) \end{aligned} \quad (3.6)$$

El decodificador puede reproducir esta operación ya que los píxeles por debajo y a la derecha de x son conocidos de la segunda pasada.

Nuevamente luego de hallado $\hat{x}$, se codifica el valor del error $e = x - \hat{x}$

Cuantización de los contextos

En cada una de las pasadas se codifica aritméticamente (ver [17]) el valor de los errores $e = x - \hat{x}$, lo que implica que sean asignadas probabilidades. Asumiremos que el píxel x es predicho por K píxeles vecinos $x_1, x_2, \dots, x_K$. Los valores de K para las distintas pasadas son 4, 8 y 6 respectivamente. Para asignarle la probabilidad a x el codificador debe contar el número de veces que x fue encontrado en el pasado en cada contexto posible. Si el píxel es guardado en un número de c bits (representando una escala de gris), entonces el número de contextos posibles es de $2^{K \cdot c}$. Este número aún para valores pequeños de c es demasiado grande para una realización práctica.

Por ejemplo, para $c = 4$ (solo 16 tonos de gris) la cantidad de contextos llegaría a $2^{8 \cdot 4} \approx 4.3 \times 10^9$.

De este modo, aún para imágenes muy grandes, no serían suficientes las muestras como para obtener una buena estimación de la probabilidad. Este fenómeno se conoce en el campo de la compresión de imágenes como *dilución de contextos*. Es por este motivo que CALIC reduce el número de contextos en varios pasos. Primero crea un número de K bits $t = t_K \dots t_1$ mediante:

$$t_k = \begin{cases} 0 & \text{si } x_k \geq \hat{x} \\ 1 & \text{si } x_k < \hat{x} \end{cases} \quad (3.7)$$

Esto puede verse como una forma de modelar un patrón espacial de la textura de la imagen. Además del patrón espacial de textura, la variación en $x_1, x_2, \dots, x_K$ se correlaciona fuertemente con la suavidad de la imagen alrededor del píxel x . Para modelar esta correlación con un número pequeño de parámetros y un costo computacional accesible es que se define:

$$\Delta = \sum_{k=1}^K \omega_k |x_k - \hat{x}| \quad (3.8)$$

donde los coeficientes ω_k son calculados previamente en un conjunto de imágenes de entrenamiento, de manera de minimizar la diferencia ($|e| - \Delta$), y luego introducidos al algoritmo.

La cantidad Δ es llamada *discriminante de la fuerza del error* y es cuantizada en uno de L valores enteros, donde L es típicamente 8. Llamaremos d a esta cuantización de Δ . Una vez que $\hat{x}$ y sus K vecinos $x_1, x_2, \dots, x_K$ son conocidos, pueden ser calculados tanto t como el valor cuantizado d de Δ , y se convierten en los índices del contexto.

Combinando el discriminante cuantizado de la fuerza del error Δ de L niveles y los 2^K patrones cuantizados de textura, se logra cuantizar los originales $2^{K \cdot c}$ estados en $L2^K$ estados, que para 8 bits por píxel da un valor de $8 \cdot 2^8 = 2048$. Estos estados son llamados *contextos cuantizados* y se denotarán por $C(d, t)$, $0 \leq d < L$, $0 \leq t < 2^K$.

Modelado Adaptativo del Error basado en Contextos

Aún luego de la cuantización de los contextos, los $L2^K$ diferentes contextos $C(d, t)$ son todavía demasiados como para tener una buena estimación de $p(e|C(d, t))$ obtenida a partir de la estadística de las muestras de una imagen. Para sobrellevar este inconveniente, no se estima $p(e|C(d, t))$, sino que sólo se calcula la esperanza condicional $E\{e|C(d, t)\}$ utilizando para ello la media de e en los diferentes contextos cuantizados, media que se denominará $\bar{e}(d, t)$. Dado que $\bar{e}(d, t)$ es la estimación más probable del error de predicción para un contexto cuantizado $C(d, t)$, es factible realizar una compensación del error modificando la predicción de x por medio de $\hat{x}$ a:

$$\dot{x} = \hat{x} + \bar{e}(d, t) \quad (3.9)$$

Con este nuevo predictor $\dot{x}$, se reemplaza $e = x - \hat{x}$ por $\epsilon = x - \dot{x}$ en la codificación del error. Para cerrar el bucle de realimentación del error, se modifica 3.9 a:

$$\dot{x} = \hat{x} + \bar{\epsilon}(d, t) \quad (3.10)$$

donde $\bar{\epsilon}(d, t)$ es la media de las muestras ϵ en el contexto $C(d, t)$. Esto lleva a que el modelado del contexto se base en ϵ en vez de e .

El predictor $\dot{x}$ de la ecuación 3.10 es más robusto que $\hat{x}$ ya que $\dot{x}$ puede cancelar los corrimientos de la predicción lineal de $\hat{x}$ bajo un contexto determinado.

Dado que el número de contextos posibles $C(., .)$ es demasiado grande para realizar una codificación de entropía basada en $p(e|C(., .))$, se halló la forma de unir diferentes contextos en

los cuales $p(e|C(.,.))$ estén próximos. Sabiendo que el error de predicción $|e|$, en video e imágenes, sigue empíricamente una distribución de probabilidad Laplaciana (o en su versión discreta una distribución geométrica), en lugar de calcular directamente la probabilidad $p(e|C(.,.))$, se calculan algunos parámetros de dicho tipo de distribución. Si bien esta distribución es aproximadamente Laplaciana, no necesariamente posee media cero, lo cual se logra cancelando los corrimientos. Luego de la cancelación del corrimiento, los diferentes contextos serán caracterizados por una familia de distribuciones Laplacianas de media nula teniendo diferentes varianzas. Dado que Δ es una medida de dicha varianza, luego de su cuantización, se hace posible unir diferentes contextos en un conjunto más pequeño, disminuyendo el problema de escasez de muestras.

A continuación se presenta un pseudo código para el algoritmo CALIC:

Para todas las pasadas

Inicialización: $N(d, t) := 1; S(d, t) := 0; d = 0, 1, \dots, L, t = 0, 1, \dots, 2^K;$

Parámetros: a_k y ω_k están pre-fijados en un conjuntos de imágenes de entrenamiento;

FOR todos los píxeles x en la corriente pasada DO

1. $\hat{x} = \sum_{k=1}^K a_k \cdot x_k;$
2. $\Delta = \sum_{k=1}^K \omega_k (x_k - \hat{x});$
3. $d = \text{Cuantizador}(\Delta);$
4. Calcular $t = t_K \dots t_2 t_1;$
5. $\bar{\epsilon} = S(d, t) / N(d, t);$
6. $\dot{x} = \hat{x} + \bar{\epsilon};$
7. $\epsilon = x - \dot{x};$
8. $S(d, t) := S(d, t) + \epsilon; N(d, t) := N(d, t) + 1;$
9. IF $N(d, t) \geq 128$ THEN:
 - $S(d, t) := S(d, t) / 2; N(d, t) := N(d, t) / 2;$
10. IF $S(d, t) < 0$ *codificar* $(-\epsilon, d)$ ELSE *codificar* $(\epsilon, d);$

END FOR;

END;

Siendo $S(d, t)$ la suma de los errores de predicción ϵ en ese contexto y $N(d, t)$ la cantidad de ocurrencias de ese contexto.

A continuación revisaremos las características principales de uno de los métodos que propone una extensión de CALIC a video, conocido como I-CALIC.

3.1.2. I-CALIC

Este método, considerado en [2], consiste en dotar al algoritmo CALIC (ver 3.1.1) de la posibilidad de utilizar la redundancia temporal y espacial presente en una secuencia de video.

I-CALIC presenta dos modos diferentes de operación, el principal es denominado *tono continuo* donde el codificador computa una predicción explotando la redundancia temporal o espacial,

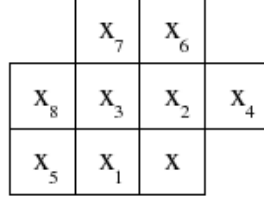


Figura 3.4: Molde causal utilizado por I-CALIC

mientras que el secundario se denomina *binario*, en el cual se entra si se cumplen ciertas condiciones de regularidad entre los píxeles del molde causal presentado en la figura 3.4.

Esta decisión se toma para cada píxel a ser codificado del siguiente modo: el codificador primero verifica si los seis píxeles que integran el molde causal de la figura 3.4 no toman más que dos diferentes valores s_1 y s_2 . Luego se codifica el símbolo T como sigue:

$$T = \begin{cases} 0 & \text{si } x(i, j) = s_1 \\ 1 & \text{si } x(i, j) = s_2 \\ 2 & \text{en otro caso} \end{cases} \quad (3.11)$$

Donde el símbolo $T = 2$ corresponde al escape para volver al modo tono continuo, lo que ocurre siempre que no se cumplan las condiciones para ingresar al modo binario.

La correlación entre imágenes $F[i, j]$ y $G[i, j]$ de tamaño $M \times N$ se define como:

$$\rho = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} F[m, n] G[i + m, j + n] \quad (3.12)$$

En el modo tono continuo el algoritmo estima el coeficiente de correlación ρ (definido en la ecuación 3.12) entre el molde causal antes mencionado, y el correspondiente en el cuadro anterior, o sea, aquel correspondiente al píxel que ocupa la misma posición en el cuadro anterior.

Si ρ resulta mayor a un umbral predefinido, la predicción se realiza *entre cuadros*, seleccionando uno de tres predictores, dependiendo de la presencia o no de bordes verticales u horizontales. Si ρ es menor que el umbral, el método decide que la redundancia temporal no es suficiente y selecciona un predictor espacial *intra cuadro* denominado *GAP* (de la sigla en inglés *Gradient Adjusted Predictor*). En este caso la predicción es formada entre cinco diferentes esquemas basados en la detección de gradientes verticales u horizontales.

Posteriormente con la predicción $\hat{x}(i, j)$ se calcula el error de predicción $e(i, j)$ como:

$$e(i, j) = x(i, j) - \hat{x}(i, j) \quad (3.13)$$

el cual es codificado luego de ser realizado un modelado de contexto y una cancelación del corrimiento. Las secuencias de video con fondos estáticos o movimientos lentos benefician considerablemente el uso del predictor *entre cuadros*, sin embargo, una amplia gama de videos no entran en estas hipótesis, limitando a I-CALIC a su predicción espacial en mayor parte.

Modelado de contexto

Dado que la predicción en general falla en remover toda la redundancia entre los píxeles, es posible mejorar la eficiencia de la codificación realizando un modelado de contexto de la predicción del error previo a realizar la codificación de entropía.

I-CALIC clasifica el error de predicción en diferentes contextos dependiendo del modo de predicción empleado (espacial o temporal).

Se desea estimar la probabilidad $P(e|C)$ la cual correspondería a un largo de código $l(e|C) = -\log(P(e|C))$ donde C es un contexto informativo para el error de predicción e .

I-CALIC toma una serie de parámetros que se consideran empíricamente que contienen información suficiente sobre e como son:

- La magnitud de los errores de predicción previamente codificados $|e(i-1, j)|, |e(i, j-1)|$
- El coeficiente de correlación ρ
- Un parámetro d el cual es un estimador en mínimos cuadrados de $|x(i, j) - x(i-1, j)| + |x(i, j) - x(i, j-1)|$
- Un coeficiente que modela la “textura” del contexto.

Claramente es necesario cuantificar la cantidad de contextos para poder obtener una mayor cantidad de ocurrencias de cada uno y así una estimación de probabilidad más confiable.

De este modo la textura es cuantizada en un patrón de bits $t = t_6 t_5 \dots t_1$ donde:

$$t_k = \begin{cases} 0 & \text{si } x_k \geq \hat{x} \\ 1 & \text{si } x_k < \hat{x} \end{cases} \quad (3.14)$$

La correlación ρ es cuantizada en dos niveles obteniendo la variable binaria $\hat{\rho}$ la cual minimiza la entropía condicional $H(e|\hat{\rho})$ en el conjunto de imágenes de entrenamiento.

Por otro lado d , $|e(i-1, j)|, |e(i, j-1)|$ se combinan en un parámetro Δ , para formar un estimador en mínimos cuadrados de $|e|$, el cual se cuantifica en δ por medio de un cuantizador escalar K de cuatro niveles. Finalmente se construye el contexto $C = (\hat{\rho}, \delta, t)$, pero aún siguen existiendo problemas para la estimación de $P(e|C)$ debido al alto número de contextos posibles, por lo que debe buscarse otra forma de encarar el problema.

Este problema es resuelto de la misma manera que CALIC. I-CALIC utiliza los contextos C para detectar y remover corrimientos en la predicción.

La cancelación del corrimiento consiste en sustraer el error esperado $E(e|C)$ (estimado en un contexto C) del error de predicción e , por lo tanto el error codificado es $\hat{e} = e - E(e|C)$. Usualmente $E(e|C)$ es estimado como el promedio de los errores de predicción ocurridos hasta ese entonces en el contexto. Luego de la cancelación del corrimiento, los diferentes contextos serán caracterizados por una familia de distribuciones Laplacianas de media nula teniendo diferentes varianzas, por lo que contextos con similar distribución podrían ser unidos. Para este propósito es que la varianza estimada se cuantiza en $K = 8$ niveles y el índice de cuantización es usado como parámetro en el codificador aritmético.

3.1.3. M-CALIC

M-CALIC propuesto en [2] plantea integrar al marco de trabajo de CALIC la *compensación de movimiento*¹ como herramienta para explotar la redundancia temporal presente en las secuencias de video.

Cuando se realiza compensación de movimiento el cuadro a ser codificado, f , se divide en bloques de tamaño $M \times N$. Para cada bloque $\bar{B}^f(p)$ en la posición $p(i, j)$, es buscado en el cuadro $f-1$ en la vecindad de $p = (i, j)$ un bloque $\bar{B}^{f-1}(p+v) = (b_1^{f-1}, \dots, b_{N^2}^{f-1})$ que minimice algún tipo de

¹La compensación de movimiento será abordada en la sección 5.3.1

medida de distancia. Las usualmente utilizadas son la distancia euclidiana entre los dos bloques o la suma de los valores absolutos de las diferencias entre píxeles.

M-CALIC calcula el residuo de la diferencia como:

$$e = \overline{B}^f(p) - \overline{B}^{f-1}(p + v) \quad (3.15)$$

donde e es el vector que contiene le residuo de las diferencias para cada píxel del bloque $\overline{B}^f(p)$ y v es el vector de movimiento que indica el desplazamiento relativo entre los dos bloques.

La ecuación 3.15 emplea únicamente al cuadro anterior como referencia, por lo que extendiendo para M cuadros se obtiene la siguiente ecuación:

$$e = \overline{B}^f(p) - \sum_{l=1}^M W_l \overline{B}^{f-l}(p + v_l) \quad (3.16)$$

Donde M es el número de cuadros anteriores utilizados para la predicción, y $W(p) = (\omega_1(p), \dots, \omega_M(p))$ es un factor de ponderación apropiado. Esto significa que para cada bloque $\overline{B}^f(p)$ es buscado un bloque con una distancia mínima en cada uno de los M cuadros pasados, y la predicción es formada como una combinación lineal ponderada de los bloques seleccionados. La ponderación se calcula de forma tal que minimice el error en mínimos cuadrados o MSE (de su sigla en inglés Minumun Squared Error) del residuo, resolviendo el sistema de ecuaciones:

$$P \bullet W = R \quad (3.17)$$

Donde:

$$P = \begin{bmatrix} b_1^{f-1}(p + v_{f-1}) & \cdots & b_1^{f-M}(p + v_{f-M}) \\ \vdots & \ddots & \vdots \\ b_{N^2}^{f-1}(p + v_{f-1}) & \cdots & b_{N^2}^{f-M}(p + v_{f-M}) \end{bmatrix} \quad (3.18)$$

es una matriz cuyas columnas están compuestas por los píxeles de cada bloque predictor y

$$R = \begin{bmatrix} b_1^f(p + v_f) \\ \vdots \\ b_{N^2}^f(p + v_f) \end{bmatrix} \quad (3.19)$$

es un vector columna que contiene el valor de los píxeles del bloque a ser predicho. Típicamente este es un sistema sobre-determinado en donde generalmente, una solución exacta no puede ser hallada, pero que puede ser fácilmente resuelto minimizando el error medio cuadrático a través de su descomposición en valores singulares o una descomposición QR .

Los factores de ponderación W no son independientes entre un bloque y otro, lo cual es razonable y esperado dado que todos los bloques tienen la misma energía habiendo sido seleccionados para minimizar el MSE respecto a $\overline{B}^f(p)$.

De este modo, en lugar de usar y transmitir W , que en general posee valores reales, se puede diseñar con un conjunto de imágenes de entrenamiento, un cuantizador óptimo (en el sentido MSE) de modo tal que sea necesario enviar solamente valores enteros de índices de cuantización al decodificador. Por esta razón, el codificador calcula los pesos óptimos para cada bloque, los cuantiza, transmite los índices de cuantización y utiliza la versión cuantizada de W en 3.16 para calcular el residuo de predicción. De este modo el decodificador puede invertir el proceso y reconstruir sin pérdidas el cuadro original.

El decodificador necesita tanto los vectores de movimiento v_j como los índices de cuantización

para W de modo tal que sea conformada la misma predicción y, agregada al residuo de predicción, se obtenga una perfecta reconstrucción del píxel.

En consecuencia la cadena de bits codificada consiste en residuos de predicción e información lateral, es decir, vectores de movimiento e índices de cuantización. Realizar la compensación de movimiento en M cuadros lleva consigo el envío de M vectores de movimiento como información de lado, lo que genera un incremento en la cadena de bits. Debido a la alta correlación existente entre los vectores de movimiento adyacentes, su entropía es muy baja comparada con la mejora inducida a la codificación del residuo.

Experimentalmente se observa que el predictor de la ecuación 3.16 tiene una performance mejor que el predictor puramente espacial GAP de I-CALIC, independientemente de la correlación existente entre los píxeles vecinos, por lo que no se utiliza el predictor espacial en M-CALIC sino únicamente el predictor dado por 3.16.

Luego de la predicción tiene lugar un modelado de contexto. El contexto C se calcula para cada píxel en un bloque y se utiliza para cancelar corrimientos y realizar una codificación de entropía similar a I-CALIC. En el modelado del contexto se incluye información proveniente de la textura como en CALIC, y una estimación cuantizada de la magnitud del error de predicción dada por:

$$\hat{e} = \alpha|e(i-1, j)| + \beta|e(i, j-1)| \quad (3.20)$$

Los factores α y β son calculados fuera de línea mediante una regresión lineal en un conjunto de imágenes de entrenamiento, luego un cuantizador escalar de $\hat{e}$ se diseña para reducir el espacio modelado. Experimentalmente se determinó que la cantidad de niveles para el cuantizador debe ser 8.

Por otra parte la textura local t se calcula del mismo modo que en CALIC.

Finalmente es determinado el contexto $C = (t, \delta)$, donde δ es un índice de cuantización para $\hat{e}$. Luego de la cancelación del corrimiento se realiza la codificación de entropía del mismo modo que I-CALIC, es decir, pasando como parámetro al codificador aritmético un índice de cuantización de la estimación de varianza $\sigma(C)$ el cual es cuantizado en 8 niveles.

Como veremos más adelante esta técnica de codificación alcanza niveles de compresión mejores que I-CALIC a expensas de un incremento en la complejidad.

3.1.4. LOPT-3D

Este algoritmo de compresión de video sin pérdidas presentado en [15] está basado en un esquema de compresión para imágenes estáticas, extendido para tener en consideración la información temporal presente en las secuencias de video. Específicamente según su autor “*la técnica propuesta es comparable a una extensión 3D del estándar JPEG-LS para imágenes estáticas*”. El codificador tiene como primer operación realizar una compensación de movimiento (ver 5.3.1); el cuadro actual es dividido en bloques B_h de dimensiones $n_b \times n_b$ píxeles buscándose un vector de movimiento (m_0, n_0) en la región $-M_v < m_0 < M_v$, $-M_v < n_0 < M_v$ de modo tal de minimizar la diferencia absoluta:

$$\hat{D} = \sum_{(m,n) \in B_h} |x_k(m, n) - x_{k-1}(m - m_0, n - n_0)| \quad (3.21)$$

Una vez que son hallados los vectores de movimiento para cada bloque, cada píxel es predicho utilizando una combinación lineal de píxeles en el cuadro actual y anteriores. Los coeficientes de la predicción lineal son calculados adaptativamente para cada píxel de forma tal de minimizar la potencia del error de predicción, como se detallará más adelante.

Para definir la estructura del predictor, se tendrán en cuenta dos tipos de enumeraciones para

			16	14	
		11	8	6	9
15	7	3	2	4	10
13	5	1	x_t		

Figura 3.5: Enumeración 2D relativa al cuadro actual usada por LOPT-3D

10	5	11		
4	1	6		
13	7	12		
			8	3
			2	x_t

Figura 3.6: Enumeración 3D relativa al cuadro actual y al anterior usada por LOPT-3D

los píxeles:

- Una enumeración 2D relativa al cuadro actual, la cual ordena los píxeles de acuerdo a su distancia d_{2D} del píxel actual $x_t = x_k(m, n)$. La distancia d_{2D} se define como:

$$d_{2D} = \sqrt{(m-p)^2 + (n-q)^2} \quad (3.22)$$

para píxeles $x_k(m, n)$, $x_k(p, q)$ pertenecientes al cuadro actual. Los píxeles con la misma distancia se ordenan en forma horaria como se muestra en la figura 3.5.

- Una enumeración 3D que tiene en cuenta píxeles en el cuadro anterior, y los ordena de acuerdo a su d_{3D} definida como:

$$d_{3D} = \sqrt{(m-p)^2 + (n-q)^2} \quad (3.23)$$

para píxeles $x_k(m, n)$, $x_k(p, q)$ pertenecientes al cuadro actual, y

$$d_{3D} = \sqrt{\frac{1}{4} + (m-p)^2 + (n-q)^2} \quad (3.24)$$

para píxeles $x_k(m, n)$, $x_{k-1}(p-m_0, q-n_0)$ pertenecientes al cuadro actual y al cuadro anterior compensado; donde (m_0, n_0) es el vector de movimiento para el bloque en el cual $x_k(m, n)$ reside. Nuevamente, los píxeles con igual distancia son ordenados en forma horaria, mostrado esta vez en la figura 3.6.

A continuación se utilizará una nomenclatura especial para indicar los píxeles. En particular, $x_{t,-i}$ representará al píxel ubicado en la posición i -ésima antes de x_t de acuerdo a la enumeración 2D. Además, $x_{t,-i}(-j)$ representará al píxel ubicado en la posición j -ésima antes de $x_{t,-i}$ de acuerdo a la enumeración 3D.

Introducida la notación a utilizar, la predicción para el píxel actual se calcula como:

$$\hat{x}_{t,0} = \sum_{j=1}^N a_j x_{t,0}(-j) \quad (3.25)$$

Donde N es el orden del predictor. Como se observa esta predicción tiene en cuenta píxeles próximos tanto en el orden temporal como espacial.

El vector de coeficientes de predicción óptimo $a_0 = [a_1, \dots, a_N]^T$, se calcula de forma tal de minimizar la medida de la potencia del error de predicción siguiente:

$$P = \sum_{i=1}^M (x_{t,-i}(0) - \hat{x}_{t,-i}(0))^2 \quad (3.26)$$

La minimización se lleva a cabo en una región en la vecindad 2D de x_t . Utilizando las ecuaciones 3.25 y 3.26 se puede escribir en forma matricial:

$$P = (Ca - X)^T(Ca - X) \quad (3.27)$$

$$C = \begin{bmatrix} x_{t,-1}(-1) & \cdots & x_{t,-1}(-N) \\ x_{t,-2}(-1) & \cdots & x_{t,-2}(-N) \\ \vdots & \ddots & \vdots \\ x_{t,-M}(-1) & \cdots & x_{t,-M}(-N) \end{bmatrix} \quad (3.28)$$

$$X = \begin{bmatrix} x_{t,-1}(0) \\ x_{t,-2}(0) \\ \vdots \\ x_{t,-M}(0) \end{bmatrix} \quad (3.29)$$

Tomando la derivada de la ecuación 3.27 respecto a a e igualándola a 0, obtenemos los coeficientes óptimos para el predictor como:

$$a_0 = (C^T C)^{-1} C^T X \quad (3.30)$$

El codificador LOPT-3D utiliza una codificación basada en Golomb-Rice (ver 4.1.5) para codificar el error de predicción, exactamente como el propuesto en el estándar JPEG-LS presentado en [6], donde el error de predicción se halla como:

$$e_{t,0} = \text{piso}(x_{t,0}(0) - \hat{x}_{t,0}(0)) \quad (3.31)$$

Observar que a_0 y $\hat{x}_{t,0}(0)$ pueden ser calculados con información del pasado del píxel actual, por lo que el decodificador puede replicar las operaciones sin necesidad de que los coeficientes del predictor sean enviados como información "de lado".

Determinar los coeficientes óptimos para el predictor requiere el cálculo de y la inversión de una matriz $N \times N$, imponiendo una importante carga operativa.

Para evitar una cantidad excesiva de cuentas, no se calcula un nuevo predictor para cada píxel, sino que se actualiza el predictor sólo en el caso de que el valor absoluto de la predicción de error supere un umbral predefinido S . De este modo, si el valor absoluto del error de predicción para x_t supera S , se calcula un nuevo predictor para x_{t+1} .

A pesar de poseer una importante complejidad computacional, el esquema de codificación presentado por LOPT-3D puede ser aceptable para un uso práctico en la compresión de etapas iniciales de producción de video.

3.2. Comparación de Desempeños

Una vez inspeccionadas las principales características de los algoritmos para la compresión de video sin pérdidas presentes en el marco de trabajo propuesto, es momento de tener en cuenta su desempeño.

Una evaluación comparativa de la performance de los métodos vistos proporciona los argumentos necesarios para decidir sobre la factibilidad de la utilización de I-CALIC, M-CALIC o LOPT-3D para un sistema con las características propuestas en 1.2. Además en caso de que esta decisión sea negativa, quedan establecidos los parámetros de evaluación, así como los procedimientos seguidos, de modo de poder aplicarlos a algún otro método alternativo para justificar su elección por sobre los vistos en los puntos 3.1.2, 3.1.3 y 3.1.4.

Esta evaluación quedará definida a partir de un *conjunto de videos de prueba* y de ciertos *parámetros de evaluación*, los cuales se definen y comentan en las próximas subsecciones.

Es conveniente mencionar en este punto que la evaluación comparativa para I-CALIC, M-CALIC y LOPT-3D se extrajo de [2] ya que los resultados que de esta se desprenden se consideraron suficientes para justificar la decisión adoptada.

3.2.1. Conjunto de Videos de Prueba

Este conjunto está formado por secuencias de videos ampliamente conocidas y utilizadas en el área como instrumento comparativo de la más amplia clase de algoritmos para el procesamiento de imágenes y videos digitales, entre ellos los algoritmos de compresión de video.

El formato en que se distribuyen estas secuencias es *YCbCr 4:2:0*. Esto significa que el espacio de color en que están codificadas es YCbCr, mientras que existe un submuestreo en las componentes Cb y Cr de cada cuadro que compone la secuencia de video. El espacio de color YCbCr se vincula con el RGB a través de:

$$\begin{cases} Y = 0.257R + 0.504G + 0.098B + 16 \\ C_b = -0.148R - 0.291G + 0.439B + 128 \\ C_r = 0.439R - 0.368G - 0.071B + 128 \end{cases} \quad (3.32)$$

De este modo se almacena para cada píxel en la imagen un valor de Y, mientras que para las componentes Cb y Cr se almacena un único valor por cada cuatro píxeles de la imagen. Este procedimiento se comprende claramente a partir de la figura 3.7.

Cuando son utilizados con el fin de comparar diferentes algoritmos de compresión de imágenes o video, es usual expresar estas secuencias en el espacio de color RGB y trabajar con la componente G, es decir el canal verde.

Con respecto al tamaño de cuadro, estas secuencias están disponibles en los formatos CIF y QCIF, siendo el primero el más comúnmente utilizado a la hora de evaluación de algoritmos de compresión, ya que este posee un área cuatro veces mayor al segundo, proporcionando así mayor solidez en los componentes estadísticos del método evaluado.

En [2] el conjunto de videos de prueba está formado por las secuencias Salesman, Mobile and Calendar, Container, Tempete, Kitchgrass, Sean, Silent y Foreman, con tamaño de cuadro CIF. En lo que sigue se analizan brevemente las principales características de cada una de estas secuencias.

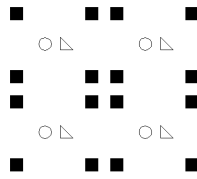


Figura 3.7: YCbCr 4:2:0. El cuadrado representa valores para cada píxel de Y, mientras que el triángulo y el círculo representan valores compartidos de Cb y Cr para cada cuatro píxeles

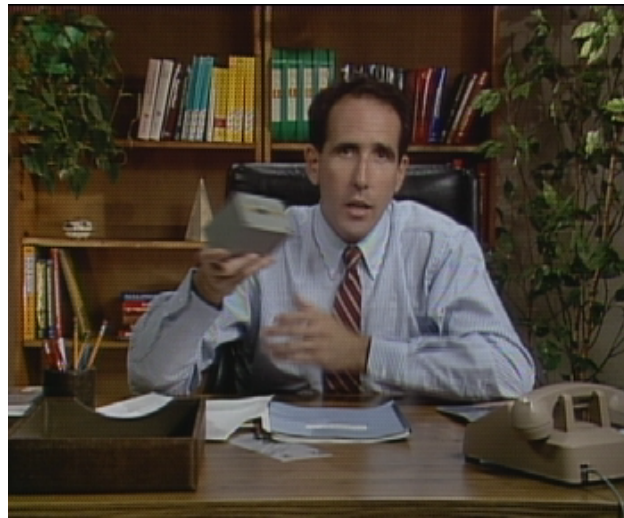


Figura 3.8: Cuadro 10 de la secuencia “Salesman”

SALESMAN

Se muestran los cuadros 10 y 100 de este video de prueba en las figuras 3.8 y 3.9 respectivamente. Esta secuencia está compuesta por 449 cuadros. Durante toda la secuencia la cámara está fija y no existen acercamientos o alejamientos. Los movimientos están dados por los brazos y cabeza de la persona, junto con una pequeña caja que el actor posee en su mano derecha. Por lo tanto, la mayor parte del movimiento está presente en la región central de todos los cuadros. El fondo es poco uniforme, compuesto por una biblioteca con libros de diversos colores, plantas y papeles sobre un escritorio.

MOBILE AND CALENDAR

En las figuras 3.10 y 3.11 se muestran los cuadros 10 y 250 del este video de prueba, el cual posee un total de 300. Esta secuencia es una de las más complejas del conjunto. El movimiento de la cámara es de translación y alejamiento. Al mismo tiempo un “tren de juguete” atraviesa los cuadros de derecha a izquierda empujando una pelota que rueda, aparece por la izquierda un móvil y un calendario sube y baja. Por lo tanto la combinación de todos estos elementos trae consigo movimientos de translación, rotación y alejamiento. El resto del área está compuesta por un mural complejo.

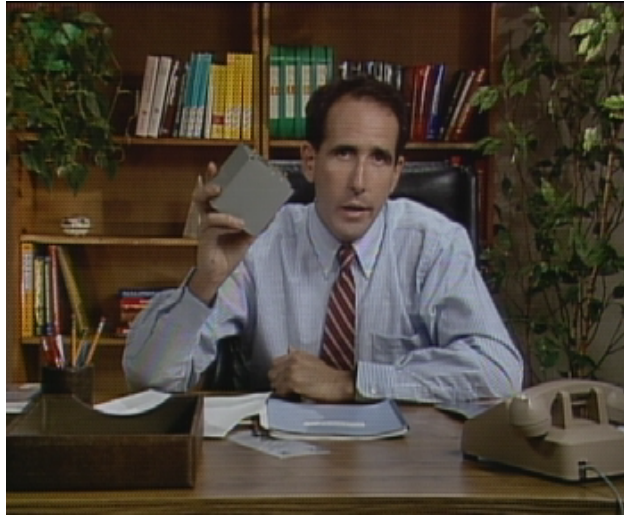


Figura 3.9: Cuadro 100 de la secuencia “Salesman”



Figura 3.10: Cuadro 10 de la secuencia “ Mobile and Calendar”



Figura 3.11: Cuadro 250 de la secuencia “Mobile and Calendar”



Figura 3.12: Cuadro 10 de la secuencia “Container”

CONTAINER

En las figuras 3.12 y 3.13 se muestran los cuadros 10 y 250 de este video de prueba. La cámara permanece estática durante los 300 cuadros que tiene la secuencia. Un barco atraviesa los cuadros de izquierda a derecha, seguido por otro más pequeño que se desplaza más rápido. El resto del movimiento esta constituido por olas en el agua, una bandera flameando y un par de aves que siguen un movimiento errático. Por lo tanto el movimiento del objeto principal consiste en una translación. El fondo está dado por el cielo y el mar.

TEMPETE

Los cuadros 10 y 250 son mostrados en las figuras 3.14 y 3.15 respectivamente. Durante los 260 cuadros que tiene este video de prueba, la cámara efectúa un alejamiento a la vez que presenta leves oscilaciones horizontales. Además, durante toda la secuencia muchas hojas secas se mueven en forma errática y veloz. Otros movimientos están dados por olas en un charco



Figura 3.13: Cuadro 250 de la secuencia “Container”



Figura 3.14: Cuadro 10 de la secuencia “Tempete”

y diversa vegetación que se agita. La mayor parte del área esta compuesta por un paisaje de campiña.

KITCHGRASS

Se muestran en las figuras 3.16 y 3.17 los cuadros 10 y 99 del video de prueba “Kitchgrass”. Esta secuencia tiene 100 cuadros de largo, durante los cuales la cámara efectúa un acercamiento con oscilaciones verticales. El resto del movimiento lo aportan la cabeza de la actriz sus hombros y su sombrero. El fondo va reduciendo paulatinamente su complejidad por el efecto de acercamiento al rostro de la actriz.



Figura 3.15: Cuadro 250 de la secuencia “Tempete”



Figura 3.16: Cuadro 10 de la secuencia “Kitchgrass”



Figura 3.17: Cuadro 99 de la secuencia “Kitchgrass”



Figura 3.18: Cuadro 10 de la secuencia “Sean”

SEAN

Los cuadros 10 y 150 del video de prueba Sean se muestran en las figuras 3.18 y 3.19. La cámara esta quieta durante los 300 cuadros de esta secuencia. El movimiento es aportado en su totalidad por el cuerpo del actor, a través de sus manos y cabeza. El resto del área la ocupa un fondo estable y simple.

SILENT

Este video de prueba está constituido por 300 cuadros, de los cuales se muestran el 10 y el 150 en las figuras 3.20 y 3.21. La cámara está fija durante toda la secuencia y muestra en el centro de los cuadros a una actriz que utiliza lenguaje de señas, por lo que la mayor parte del movimiento es aportado por sus manos y cabeza, en particular los gestos de su rostro. El fondo es complejo pero estático.



Figura 3.19: Cuadro 150 de la secuencia “Sean”



Figura 3.20: Cuadro 10 de la secuencia “Silent”



Figura 3.21: Cuadro 100 de la secuencia “Silent”



Figura 3.22: Cuadro 10 de la secuencia “Foreman”

FOREMAN

Por último, en las figuras 3.22 y 3.23 se muestran los cuadros 10 y 280 del video de prueba Foreman, el cual posee 300 cuadros. Aproximadamente hasta los dos tercios del total de cuadros la cámara muestra el rostro de un actor. En esta etapa la cámara se mueve constantemente en forma tanto horizontal como vertical. La cabeza del actor se mueve de un lado a otro y el mismo realiza variados gestos. En el último tercio la cámara rota hacia la derecha, desapareciendo así el actor de los cuadros y mostrando una obra en construcción. En esta etapa la cámara efectúa pequeños movimientos sobre un fondo complejo y estático.

3.2.2. Parámetros de Evaluación

Si bien existe una amplia variedad de parámetros para caracterizar el desempeño de algoritmos de compresión de video, nos centraremos en los dos que resultan más relevantes para



Figura 3.23: Cuadro 280 de la secuencia “Foreman”

los objetivos que sigue este proyecto. Estos son la cantidad de cuadros que el algoritmo puede procesar por segundo (ya sea para compresión o de-compresión) y el nivel de compresión alcanzado, expresado o bien como la razón entre la cantidad de bits utilizados y la cantidad de símbolos representados por éstos o bien como porcentaje. Dado que los símbolos involucrados son píxeles, el primer caso corresponde a bits por píxel o *bpp*, mientras que en el segundo el porcentaje se calcula en forma relativa a la cantidad de bits utilizados en la representación digital de la imagen sin compresión (lo que equivale a 8 bits para cada píxel, en las secuencias de video utilizadas en este proyecto).

Debe aclararse que una vez implementado un algoritmo de compresión en particular, otros parámetros de evaluación se sumarán a los antes mencionados, como por ejemplo la evolución del nivel de compresión con respecto a los cuadros de una secuencia, o alguna expresión que permita combinar las medidas de nivel de compresión y costo computacional en un único valor.

3.2.3. Desempeños de I-CALIC, M-CALIC y LOPT-3D

En esta subsección se presenta la comparación realizada en [2]. Los resultados fundamentales de la misma se muestran en las tablas 3.2 y 3.3. Antes de analizarlos, es conveniente comentar algunas características importantes de este experimento.

Con respecto a M-CALIC este fue implementado utilizando dos cuadros anteriores para formar la predicción, lo que implica $M = 2$ en la ecuación 3.16. Se seleccionó un tamaño de bloque de 16×16 y el método de compensación de movimiento utilizado fue *búsqueda exhaustiva* (ver sección 5.3.2) con un rango de búsqueda de ± 8 píxeles. Además los componentes del vector de ponderación W en la ecuación 3.17 fueron cuantizados a 5 bits.

Para LOPT-3D los parámetros fueron calibrados de acuerdo a lo especificado en [15], por lo tanto el tamaño de bloque se seleccionó de 16×16 y el rango de búsqueda de ± 8 píxeles. El predictor se fijó en orden 7 (o sea $N = 7$ en la ecuación 3.25) y la matriz C , definida en 3.28, se calculó teniendo en cuenta una región de píxeles correspondiente a 70 vecinos. Dicho predictor se actualizó cada vez que el error de predicción superaba el umbral $S = 3$ (ver sección 3.1.4). Además se utilizó la codificación de Golomb-Rice (ver sección 4.1.5) de acuerdo a lo establecido en [15].

Secuencia de Video	Inicio	Fin	Cantidad de Cuadros
Salesman	100	151	51
Mobile and Calendar	100	251	151
Container	100	149	49
Tempete	100	259	159
Kitchgrass	50	100	50
Sean	100	301	201
Silent	100	433	333
Foreman	110	149	39

Tabla 3.1: Cantidad de cuadros utilizados de cada video de prueba. Se detalla el primer cuadro utilizado, el último y la cantidad total.

Secuencia de Video	LOPT-3D	I-CALIC	M-CALIC
Salesman	4.32	4.28	3.33
Mobile and Calendar	4.53	5.36	4.25
Container	3.51	3.37	3.16
Tempete	4.53	4.50	4.45
Kitchgrass	4.22	4.60	4.11
Sean	3.40	3.05	2.94
Silent	3.55	3.29	3.13
Foreman	4.24	4.89	4.24
Promedio	4.04	4.17	3.70

Tabla 3.2: Nivel de Compresión en bpp alcanzado por los algoritmos LOPT-3D, I-CALIC, M-CALIC para el canal G de las secuencias de prueba especificadas. Fuente: [2]

Un punto importante a tener en cuenta es que la compresión sobre los videos de prueba fue realizada excluyendo los 50 primeros cuadros de cada secuencia. Estos 50 primeros cuadros son destinados al “entrenamiento” de los algoritmos, como se explica en [2].

Posteriormente, en comunicación con *Elias S. G. Carotti* se obtuvieron dos detalles más acerca de estos experimentos. El primero de ellos es que se descartaron los 50 cuadros posteriores a los utilizados para el entrenamiento “*para limitar la dependencia entre cuadros de entrenamiento y el resto del video de prueba*”. Como única excepción, en la secuencia Kitchgrass si se utilizaron los 50 cuadros posteriores a los 50 de entrenamiento debido a que la misma tiene solo 100 cuadros de longitud.

El segundo dato a destacar es la cantidad de cuadros de cada secuencia utilizados en cada caso. Los mismos se muestran en la tabla 3.1 Una observación sobre la tabla 3.1 es que hasta este momento no se halló una secuencia Silent con más de 300 cuadros.

3.3. Comparación de Resultados y Conclusiones

A partir de la tabla 3.3, los detalles del experimento y la cantidad de cuadros utilizados de cada secuencia estipulados en la tabla 3.1 se genera directamente la tabla 3.4.

Secuencia de Video	LOPT-3D	I-CALIC	M-CALIC
Salesman	01:17.34	00:07.67	00:21.58
Mobile and Calendar	03:44.25	00:24.35	01:05.40
Container	00:50.84	00:07.18	00:20.78
Tempete	04:06.34	00:24.55	01:10.19
Kitchgrass	01:12.92	00:07.63	00:21.10
Sean	03:25.84	00:30.49	01:25.22
Silent	06:24.90	00:50.86	02:34.81
Foreman	00:57.15	00:06.20	00:16.93
Promedio	02:44.95	00:19.86	00:57.00

Tabla 3.3: Tiempo de procesamiento consumido por los algoritmos LOPT-3D, I-CALIC, M-CALIC para el canal G de las secuencias de prueba especificadas, en un PC con CPU Intel Pentium IV, 2.6GHz. Fuente: [2]

Secuencia de Video	Cantidad de Cuadros	LOPT-3D	I-CALIC	M-CALIC
Salesman	51	0.66	6.65	2.36
Mobile and Calendar	151	0.67	6.20	2.31
Container	49	0.96	6.82	2.36
Tempete	159	0.65	6.48	2.27
Kitchgrass	50	0.69	6.55	2.37
Sean	201	0.98	6.59	2.36
Silent	333	0.87	6.55	2.15
Foreman	39	0.68	6.29	2.30
Promedio	0	0.78	6.50	2.27

Tabla 3.4: Cuadros procesados por segundo por los algoritmos LOPT-3D, I-CALIC, M-CALIC para el canal G de las secuencias de prueba especificadas, en un PC con CPU Intel Pentium IV, 2.6GHz. Fuente: [2]

En base a las tablas 3.2 y 3.4 pueden extraerse algunas conclusiones. En primer lugar, el algoritmo más “rápido” en el marco de trabajo propuesto, I-CALIC, apenas alcanza los 6.50 fps en media con un tamaño de cuadro CIF. Además alcanza esta rapidez a expensas de una compresión que en media se ubica en los 4.17 bpp, es decir un 47.9 %.

Por otro lado, M-CALIC es quien logra los mejores niveles de compresión en media con 3.70 bpp o 53.8 % pero logrando comprimir en media 2.27 fps.

LOPT-3D por otra parte, se ubica entre los anteriores en lo que hace al nivel de compresión y último en rapidez.

De este modo se contestan las dos cuestiones planteadas en los objetivos del proyecto:

- ¿Cuáles son los niveles de compresión alcanzados por los métodos actuales de codificación de video sin pérdidas?
- ¿Cuál es el costo computacional asociado a cada método?

Sin embargo aún resta por contestar la pregunta planteada al comienzo de esta sección y es si I-CALIC, M-CALIC y LOPT-3D son adecuados para ser implementados en un sistema con las características planteadas en 1.2.

Para contestar esta pregunta con propiedad, debemos recordar cuáles son los dos puntos fuertemente condicionantes en el diseño del sistema. El primero de ellos es que la compresión debe ser llevada a cabo sin pérdidas. Es evidente que esta restricción no es problema.

Sin embargo, la segunda restricción es la rapidez con que puedan ejecutarse los procesos de compresión y de-compresión para un cuadro. Es en este punto donde los algoritmos I-CALIC, M-CALIC y LOPT-3D parecen fallar. Es decir, en lo que hace a la compresión, los tres métodos vistos se ubican en un rango del 5.8 % entre máximo y mínimo. Así, el nivel de compresión alcanzado no es determinante para seleccionar uno de ellos, por lo que se seleccionaría el más rápido, I-CALIC. En ese caso es razonable suponer que la capacidad de procesamiento del sistema, para un cuadro de tamaño CIF sería en media inferior a 6.5 fps.

Suponiendo que se consiguiera una implementación de I-CALIC tan eficiente como la presentada en [2], este parece dejar un margen de tiempo muy estrecho para la actividad de red.

Llegamos entonces a la conclusión de que I-CALIC, M-CALIC y LOPT-3D no son adecuados para el sistema que se busca. La fundamentación de esta respuesta se halla en la baja cantidad de cuadros por segundo que las implementaciones de estos algoritmos logran procesar.

Una vez encontrada la limitante que excluye los métodos de compresión de video vistos, se buscará en el capítulo siguiente comprender el algoritmo de compresión de imágenes sin pérdidas LOCO-I/JPEG-LS [6, 9, 5]. Este es conocido por lograr niveles de compresión casi tan buenos como los obtenidos por el algoritmo CALIC [7, 16] pero con una fracción de la complejidad de este último.

Luego de un análisis en profundidad de LOCO-I/JPEG-LS los esfuerzos se centraran en buscar una forma simple de *extenderlo* a video, o sea, estudiar algunos procedimientos básicos para incluir información de cuadros pasados al momento de efectuar la codificación, de modo de alcanzar mayores niveles compresión respecto a los logrados codificando cada cuadro utilizando información solo del mismo. El conjunto de videos de prueba definido aquí, junto con los parámetros de evaluación, permitirán justificar las decisiones adoptadas en base a resultado comparables.

4 LOCO-I JPEG-LS

LOCO-I (de la sigla en inglés LOw COmplexity LOssless COmpression for Images) como su nombre lo indica es un algoritmo para la compresión sin pérdidas de imágenes de baja complejidad. Fue presentado por Weinberg y Seroussi en [9] y luego de algunos refinamientos fue adoptado como recomendación ITU-T [6, ITU Serie T, recomendación T.87] y estándar internacional ISO/IEC 14495, popularmente conocido como *JPEG-LS*.

A su vez en [5] los autores del algoritmo presentan una descripción más detallada del algoritmo y su evolución hacia la estandarización. En ésta, se observan sólidas bases teóricas que justifican cada paso en combinación con un “espíritu” de baja complejidad presente en cada una de las decisiones adoptadas. Como se verá más adelante, esto concluye en una implementación del método formada por una cadena de pasos extraordinariamente simples, pero derivados a través de algunas aproximaciones de procedimientos con justificación teórica no trivial.

En lo que sigue, dado que consideraremos la versión de LOCO-I que implementa la recomendación ITU T.87 nos referiremos al método por “LOCO-I/JPEG-LS”, LOCO-I o JPEG-LS indistintamente.

Finalmente, tanto en [5] como [6] se presenta un modo de “pérdidas controladas” o “cercano a sin pérdidas”. Dado que el mismo admite cierto nivel de pérdidas, es contrario a la característica más básica planteada en los objetivos del proyecto, por lo que no será considerado en esta documentación ni se hará más referencia al mismo.

4.1. El Codificador

Los bloques principales del algoritmo se muestran en la figura 4.1.

Las etapas de predicción y modelado de contexto están basadas en el molde causal mostrado en la figura 4.2, donde x denota la muestra actual (o muestra a codificar) y a , b , c y d son muestras vecinas. De aquí en más, haciendo un abuso de notación, nos referiremos por a , b , c y d tanto a la posición de las muestras en el molde causal, como a sus valores.

El algoritmo divide la codificación de la muestra actual en dos modos independientes. Estos son, el modo normal y el modo rachas.

El modo normal es seleccionado cuando se infiere, a partir del contexto de la muestra a codificar, que muy probablemente esta muestra no sea idéntica a las muestras de su contexto. Por otro lado, el modo rachas es seleccionado cuando ocurre lo contrario, es decir, cuando se estima probable a partir del contexto de la muestra a codificar, que las sucesivas muestras sean idénticas.

Esta decisión se basa en las diferencias $D1$, $D2$, y $D3$ definidas como:

$$D1 = d - b, D2 = b - c \text{ y } D3 = c - a \quad (4.1)$$

Estas diferencias representan el gradiente local, el cual contiene información acerca de la uniformidad o la presencia de bordes en la imagen, y es además quien determina el comportamiento estadístico del error de predicción.

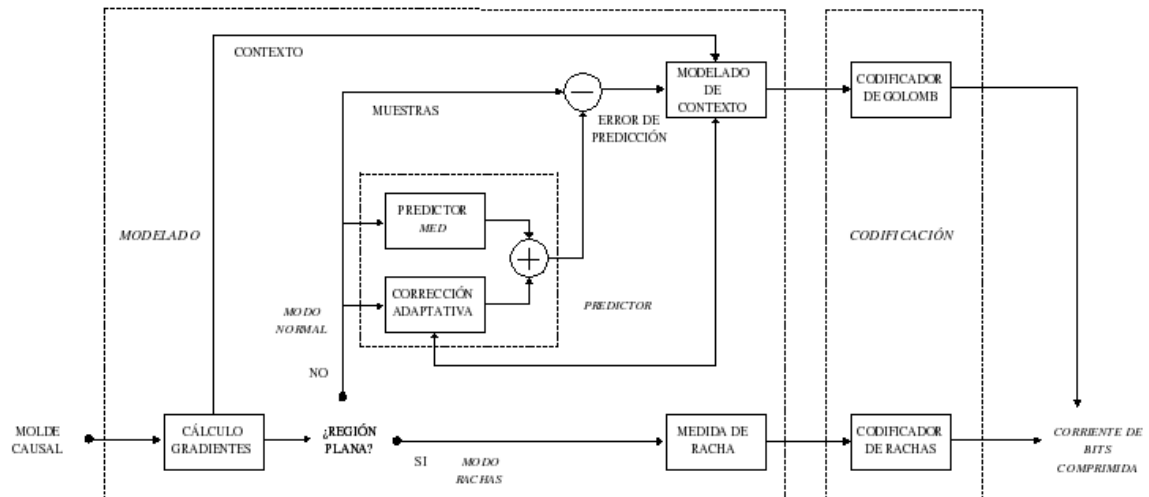


Figura 4.1: Diagrama de bloques de JPEG-LS

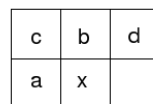


Figura 4.2: Molde causal usado para modelado de contexto y predicción

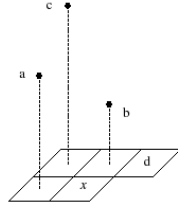


Figura 4.3: El primer caso en que se elige el valor b como predicción

Específicamente, si $a = b = c = d$, o lo que es lo mismo $D1 = D2 = D3 = 0$, el codificador decide ingresar en modo rachas. En caso contrario el modo normal es seleccionado. Intuitivamente en la figura 4.2, si la muestra d es igual a la muestra b , la muestra b es igual a la muestra c y la muestra c es igual a la muestra a , nos encontramos en una región plana de la imagen. Entonces una racha del valor a es esperable. En caso que esto no ocurra, la muestra a codificar podría encontrarse sobre un borde de la imagen (horizontal o vertical) o en una región intermedia. Estas dos últimas posibilidades son contempladas en el modo normal.

4.1.1. El Modo Normal

Una vez que el modo normal es seleccionado, el siguiente paso es elaborar una predicción de la muestra actual x , a partir de las muestras a , b , c y d .

Esta predicción se construye de la siguiente forma:

$$\hat{x}_{MED} \triangleq \begin{cases} \min(a, b) & \text{si } c \geq \max(a, b) \\ \max(a, b) & \text{si } c \leq \min(a, b) \\ a + b - c & \text{en otro caso} \end{cases} \quad (4.2)$$

El predictor 4.2 conmuta entre tres simples predictores:

- El valor b es seleccionado como predicción cuando un borde vertical existe a la izquierda de la muestra actual.
- El valor a es seleccionado como predicción cuando un borde horizontal existe sobre la muestra actual.
- El valor $a + b - c$ es seleccionado como predicción en otro caso.

La relación entre estas afirmaciones y la ecuación 4.2 puede hacerse explícita observando que, el valor b se elige como predicción cuando:

$$b < a \text{ y } c \geq a \quad (4.3)$$

o cuando:

$$b > a \text{ y } c \leq a \quad (4.4)$$

Llevando las ecuaciones 4.3 y 4.4 a las figuras 4.3 y 4.4, se observa que el valor b es seleccionado como predicción cuando un borde vertical existe a la izquierda de la muestra a codificar. Análogamente, el valor a se elige como predicción cuando:

$$a < b \text{ y } c \geq b \quad (4.5)$$

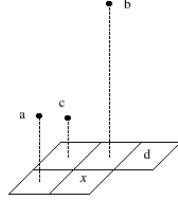


Figura 4.4: El segundo caso en que se elige b como predicción

o cuando:

$$a > b \text{ y } c \leq b \quad (4.6)$$

lo que conduce a figuras análogas a las 4.3 y 4.4.

Finalmente, el valor $a + b - c$ es seleccionado como predicción cuando no se detectan bordes. Este valor, sería el que tendría la muestra x si esta perteneciera al plano definido por a , b , y c , siendo los valores de estos píxeles interpretados como alturas.

Esto se puede demostrar simplemente asumiendo que la posición en la imagen de un píxel define dos coordenadas, X e Y , mientras que el valor de este una tercer coordenada Z . Así, la condición para que la muestra x pertenezca al plano definido por a , b y c puede obtenerse a partir de la ecuación del plano:

$$\alpha X + \beta Y + \gamma Z + \delta = 0 \quad (4.7)$$

Donde:

$$\begin{aligned} \alpha &= \begin{vmatrix} Y_a - Y_b & Y_a - Y_c \\ Z_a - Z_b & Z_a - Z_c \end{vmatrix} \\ \beta &= \begin{vmatrix} X_a - X_b & X_a - X_c \\ Z_a - Z_b & Z_a - Z_c \end{vmatrix} \\ \gamma &= \begin{vmatrix} X_a - X_b & X_a - X_c \\ Y_a - Y_b & Y_a - Y_c \end{vmatrix} \\ \delta &= \begin{vmatrix} X_a - X_b & X_a - X_c & X_x \\ Y_a - Y_b & Y_a - Y_c & Y_x \\ Z_a - Z_b & Z_a - Z_c & Z_x \end{vmatrix} \end{aligned}$$

Lo que conduce a:

$$(b - c)X - (c - a)Y - Z + a = 0$$

Tomando el origen en el píxel a e imponiendo $X = 1$ e $Y = 0$ (lo que corresponde a evaluar la ecuación para el píxel x se llega a:

$$b - c - Z + a = 0$$

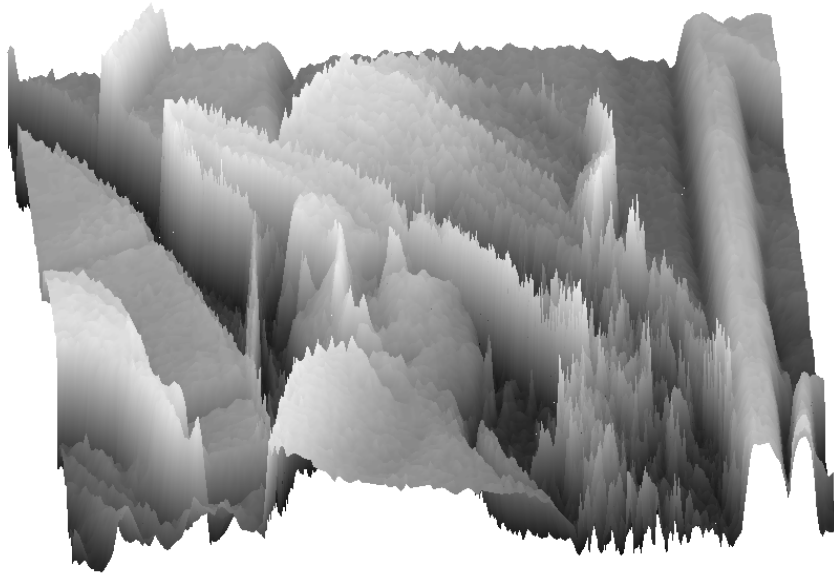


Figura 4.5: Interpretación de nivel de gris como coordenada Z , para la imagen “Lena”

O, lo que es lo mismo:

$$Z = a + b - c$$

Se expresa por medio de este valor la “suavidad” esperada en un punto de la imagen en ausencia de bordes. Esto se puede visualizar en la figura 4.5, en la cual se ha interpretado la posición de un píxel en la imagen como coordenadas X e Y , y el nivel de gris del píxel como la coordenada Z .

Dado que la ecuación 4.2 puede ser interpretada como el cálculo de la mediana entre a , b y $a + b - c$ este predictor ha sido denominado “MED” de la sigla en inglés *Median Edge Detector*. El origen de este nombre es evidente a partir de la explicación de su funcionamiento, y es el que utilizaremos de aquí en más para identificar este predictor.

Pasamos ahora, a la siguiente etapa en el proceso de codificación, el modelado de contexto.

4.1.2. Modelado de Contexto

La codificación del error de predicción está determinada por el contexto en el cual está inmersa la muestra a codificar. Este se identifica a partir de las ecuaciones 4.1. Estas tres diferencias influyen de igual modo en el proceso de modelado, a partir de hipótesis sobre la estructura de

la imagen a codificar (la imagen a codificar es esencialmente simétrica respecto a la dirección horizontal o vertical, sentido izquierdo o derecho, arriba o abajo así como respecto a la transformación de “negación de valor” $x \rightarrow (\alpha - 1 - x)$).

Como parte de las políticas para disminuir la complejidad del método, cada diferencia D_i de las ecuaciones 4.1 es cuantizada en un número pequeño y fijo de regiones conexas aproximadamente equiprobables.

El hecho de que estas diferencias influyan el modelado de igual modo se hace presente observando que el cuantizador $K(D_i)$ es independiente de i , así como en el hecho de que las regiones de cuantización son indexadas de la forma $-T, \dots, -1, 0, 1, \dots, T$, con $K(D) = -K(-D)$.

Se llega entonces a un total de $(2T + 1)^3$ contextos diferentes. Ahora bien, si se tienen en cuenta nuevamente las hipótesis sobre simetría en la estructura de la imagen, es razonable asumir que:

$$Prob\{\epsilon_{t+1} = \Delta \mid C_t = [q_1, q_2, q_3]\} = Prob\{\epsilon_{t+1} = -\Delta \mid C_t = [-q_1, -q_2, -q_3]\}$$

Donde C_t representa el vector de contextos cuantizados $q_i = K(D_i)$. Entonces, si el primer elemento no nulo de C_t resulta negativo, el error de predicción $-\epsilon_{t+1}$ es codificado usando el contexto $-C_t$. Con esta “mezcla” de contextos de signos opuestos, el número total de contextos se reduce a $((2T + 1)^3 + 1)/2$.

En JPEG-LS se selecciono $T = 4$ resultando en 365 contextos diferentes. Esta elección logra un balance entre requerimientos de memoria y nivel de compresión alcanzado. Para imágenes de gran tamaño más contextos pueden agregarse sin elevar significativamente el costo computacional, pero no resultan en una mejora apreciable del nivel de compresión. Para imágenes pequeñas, es posible del mismo modo, reducir el número de contextos establecidos por el estándar.

La especificación de los contextos para JPEG-LS se completa definiendo las fronteras entre las regiones de cuantización como $\{0\}, \pm\{1, 2\}, \pm\{3, 4, 5, 6\}, \pm\{7, 8, \dots, 20\}, \pm\{e \mid e \geq 21\}$, para un alfabeto de 8 bits por muestra. La equiprobabilidad de estas regiones se basa en el número de diferencias que cada una de ellas contempla. Por ejemplo, la diferencia 0 se estima cuatro veces más probable que la diferencia 1, -1, 2 o -2.

Una vez determinado el contexto para el píxel actual, pasamos a la corrección adaptativa de la predicción brindada por el predictor MED.

4.1.3. Corrección Adaptativa

La función de distribución de probabilidad asumida por el algoritmo LOCO-I para el error de predicción es TSGD, de la sigla en inglés *Two Sided Gemoetric Distribution* centrada en cero.

Sin embargo, dado que se observa una desviación sistemática o *DC offset* en la distribución de errores de predicción generados por predictores condicionados por contextos, se incluye el parámetro μ con el objetivo de compensarlo. El origen de este offset, según los autores así lo afirman en [5] se encuentra principalmente en la restricción a valores enteros y en posibles tendencias en la etapa de predicción.

Este parámetro μ es descompuesto en su parte entera R (denominada *bias*) y su parte fraccionaria s (denominada *shift*, con $0 \leq s < 1$ naturalmente) de modo tal que:

$$\mu = R - s \tag{4.8}$$

Así se llega al modelo para la distribución del error de predicción utilizado por LOCO-I/JPEG-LS:

$$P_{\theta,\epsilon} = C(\theta, \epsilon) \theta^{|\epsilon - R + s|} \quad (4.9)$$

Con $\epsilon = 0, \pm 1, \pm 2, \dots$, $C(\theta, \epsilon) = \frac{1-\theta}{\theta^{1-s} + \theta^s}$ un factor de normalización y $\theta \in (0, 1)$ controlando el decaimiento exponencial. Sin embargo como se verá más adelante, la parte entera del offset R es cancelada por una corrección adaptativa entera, trabajando junto con el predictor MED, como se observa en la figura 4.1. Por lo tanto el modelo de la ecuación 4.9 se reduce a:

$$P_{\theta,\epsilon} = C(\theta, \epsilon) \theta^{|\epsilon + s|} \quad (4.10)$$

Así, el aspecto del histograma del error de predicción (luego de la corrección adaptativa) será bimodal, con modos en 0 y -1 . Si bien cualquier intervalo unitario podría ser usado en principio, el intervalo seleccionado ($0 \leq s < 1$) está adecuado al esquema de codificación que se presentara posteriormente.

Es importante notar que los parámetros que definen el offset son dependientes del contexto del píxel a codificar. Aunque eso no se haga explícito en la notación usada, se volverá evidente en el momento de detallar la implementación del algoritmo.

Estimación de R

Como en todas las etapas que componen el algoritmo LOCO-I, el requerimiento de baja complejidad está presente a la hora de estimar R . Esto lleva a implementar dicha estimación en base a la media. Manteniendo un conteo N de la cantidad de ocurrencias de un contexto y la suma acumulativa D de los *errores de predicción* (es decir, la diferencia entre el valor de la muestra a codificar y el de la predicción $\hat{x}_{MED}$) cometidos para dicho contexto, la corrección podría obtenerse como:

$$C' = \lceil D/N \rceil \quad (4.11)$$

Donde los símbolos $\lceil \cdot \rceil$ indican techo. Sumando la corrección a la predicción generada por el predictor MED, $\hat{x}_{MED}$ se obtiene $\hat{x}$ la predicción definitiva del píxel a codificar, o predicción corregida.

Sin embargo esta forma de estimar R presenta dos desventajas:

- Requiere una división, lo que va en contra del enfoque de mínima complejidad de LOCO-I/JPEG-LS
- La estimación es demasiado sensible a “outliers estadísticos”. Esto se refiere al hecho de que un valor en el error particularmente alto podría afectar futuros valores de C' hasta que más ocurrencias del mismo contexto lo devuelvan a su valor típico.

Esto se resuelve notando que la ecuación 4.11 implica:

$$D = NC' + B' \quad (4.12)$$

Con B' un entero que satisface $-N < B' \leq 0$.

La corrección se implementa entonces a partir de los valores almacenados de C' y B' , ajustando recursivamente ambos para cada ocurrencia del contexto. El *error de predicción corregido* ϵ (es decir, la diferencia entre el valor de la muestra a codificar x y el de la predicción corregida $\hat{x}$) se suma a B' y luego se le suma o resta N hasta que B' cae en el rango debido ($-N < B' \leq 0$) en tanto que C' se decrementa o incrementa en 1 respectivamente. Este simple procedimiento

puede expresarse así:

1. $B' = B' + \epsilon$
2. $N = N + 1$
3. Si $B' \leq -N$
 - a) $C' = C' - 1$
 - b) $B' = B' + N$
 - c) Si $B' \leq -N$ volver a paso 3.a, en caso contrario *fin*
4. Si $B' > 0$
 - a) $C' = C' + 1$
 - b) $B' = B' - N$
 - c) Si $B' > 0$ volver a paso 4.a, en caso contrario *fin*

Donde B' y C' son inicializados en cero. Es claro que mientras $B' \leq -N$, C' será decrementado en 1, mientras que B' lo será en N . De este modo se cumple la ecuación 4.12 al mismo tiempo que la restricción para B ($-N < B' \leq 0$). Se procede análogamente cuando $B' > 0$). Si se expresa la ecuación 4.12 como:

$$\frac{D}{N} = C' + \frac{B'}{N} \quad (4.13)$$

y se compara con la ecuación 4.8 es claro que C' es un estimador de R , mientras que $-\frac{B'}{N}$ es un estimador de s , así como $\frac{D}{N}$ de μ .

Nuevamente en base al argumento de mantener la complejidad del método lo más baja posible, LOCO-I/JPEG-LS restringe el número de sumas y restas en N de B' a *una por iteración*, al igual que los decrementos o incrementos en 1 de C' . Luego el valor de B' es forzado al rango debido, de ser necesario. Además, en base al mismo argumento, C no es incrementado por sobre 127 ni decrementado por debajo de -128 . El procedimiento pasa entonces a ser el siguiente:

1. $B = B + \epsilon$
2. $N = N + 1$
3. Si $B \leq -N$
 - a) $B = B + N$
 - b) Si $C > -128 \rightarrow C = C - 1$
 - c) Si $B \leq -N \rightarrow B = -N + 1$
4. Si $B > 0$
 - a) $B = B - N$
 - b) Si $C < 127 \rightarrow C = C + 1$
 - c) Si $B > 0 \rightarrow B = 0$

De esta forma los estimadores C' y B' son remplazados por otros, C y B . Este procedimiento tiende a llevar el promedio de los errores de predicción corregidos al intervalo $(-1, 0]$

4.1.4. Reducción del rango para ϵ

En principio el error de predicción corregido ϵ puede tomar cualquier valor en el rango $-(\alpha - 1) \leq \epsilon \leq (\alpha - 1)$ (donde $\alpha = 2^\beta$ es el tamaño del alfabeto de entrada, igual a 256 para $\beta = 8$, lo que corresponde a 8 bits por píxel).

Sin embargo, conocido $\hat{x}$ dado que $\epsilon = x - \hat{x}$, ϵ puede tomar valores solo en el rango $-\hat{x} \leq \epsilon \leq (\alpha - 1) - \hat{x}$. Teniendo en cuenta además que, obviamente, $\hat{x}$ es conocido por el decodificador esta propiedad puede explotarse para reducir el rango de ϵ módulo α a:

$$\frac{-\alpha}{2} \leq \epsilon \leq \frac{\alpha}{2} - 1 \quad (4.14)$$

Esto es explotado por LOCO-I/JPEG-LS mapeando grandes valores de ϵ en valores pequeños a través de:

1. Si $\epsilon < 0$ entonces $\epsilon = \epsilon + \alpha$
2. Posteriormente, si $\epsilon > \frac{\alpha + 1}{2}$ entonces $\epsilon = \epsilon - \alpha$

Visualmente, se está mezclando las “colas” de la distribución TSGD dada por 4.10 con su “parte central”. Esto no afecta *significativamente* el comportamiento TSGD de $P_{(\theta, \epsilon)}$ ya que los valores grandes de ϵ son notablemente menos probables que los pequeños. Además, dado que α es usualmente un valor grande, LOCO-I/JPEG-LS usa de todos modos el modelo de alfabeto infinito para ϵ dado por 4.10, aún cuando el rango real para éste sea el dado por 4.14.

4.1.5. Codificación

Como se observa en la figura 4.1 una vez que se tiene la predicción corregida se procede a codificar el error de predicción corregido ϵ distribuido de acuerdo a 4.10 utilizando una *sub-familia de mínima complejidad* de la familia de códigos de prefijo óptimo para TSGDs, basados en los *Códigos de Golomb*. Los códigos de Golomb son un tipo de codificación de entropía desarrollado por Solomon W. Golomb y presentado en [18], óptimo para alfabetos que siguen una distribución de probabilidad geométrica.

Por este motivo se dedica el siguiente punto en este documento al estudio de los mismos, y su derivación en el método de codificación utilizado por LOCO-I/JPEG-LS.

Codificación de Golomb

Entendemos por *sucesión geométrica* aquella donde el cociente entre elementos consecutivos es una constante q denominada *geométrica*. Este tipo de sucesiones tienen los elementos de la forma $a, aq, aq^2, \dots, aq^n, \dots$. La suma infinita de estos elementos es una *serie geométrica*.

$$\sum_{i=0}^{\infty} aq^i \quad (4.15)$$

Muchos métodos de compresión se basan en codificación por largo de corridas o *RLE* (de si sigla en inglés Run-Length Coding). A modo de ejemplo, suponemos una cadena binaria donde el cero tiene una probabilidad de ocurrencia p y el uno tiene una probabilidad $1 - p$. Si p es

grande, habrá corridas de ceros, sugiriendo el uso RLE para comprimir la cadena. La probabilidad de una corrida de n ceros es p^n y la probabilidad de una corrida de n ceros seguida por un uno es $p^n(1-p)$. Si denominamos $a = 1-p$, la función de probabilidad de una corrida de n ceros es $P(n) = ap^n$, recordando a una sucesión geométrica, dando origen a la denominación *distribución geométrica*.

Una forma simple de codificar estas cadenas es calcular la probabilidad de cada corrida y aplicar codificación Huffman (ver [17]) para obtener el mejor código prefijo para la corrida. Un código prefijo es un código de largo variable que satisface la propiedad prefijo (ver 2.2). En la práctica, sin embargo, es posible que existan un gran número de corridas y que éstas no sean conocidas previamente. Por lo tanto no podemos aplicar el método de Huffman, ya que para ello es necesario listar todas las corridas comenzando con la menos probable.

Una mejor forma de atacar el problema es construir una *familia infinita de códigos prefijo óptimos*, ya que no importando que tan larga sea la corrida, habrá un código en la familia para codificarla. Los códigos en la familia deben depender de la probabilidad p , de modo que estamos buscando un conjunto infinito de *códigos prefijo parametrizados*. Los códigos de Golomb son estos códigos y son óptimos para la compresión de enteros no negativos que siguen una distribución geométrica, lo que generalmente se denomina *OSGDs* (de la sigla en inglés One Sided Geometric Distributions). Esto quiere decir que para toda fuente de enteros no negativos y con función de probabilidad geométrica, existe un m tal que $G_m(y)$ conduce al código de menor longitud media sobre todos los códigos unívocamente decodificables (donde $G_m(y)$ indica la codificación por Golomb de y para el parámetro m).

Examinemos algunos números para ver la forma en que estos códigos dependen de la probabilidad p . Para $p = 0.99$, la probabilidad de una corrida de dos ceros es $0.99^2 = 0.9801$, y la probabilidad de una corrida de diez ceros es $0.99^{10} = 0.904$, observándose que ambas son altas. Sin embargo, para $p = 0.6$, la probabilidad de una corrida de dos ceros es $0.6^2 = 0.36$ y para una corrida de diez ceros, $0.6^{10} = 0.006$ observándose una baja probabilidad de rachas. Por lo tanto un valor alto de p implica una alta probabilidad de rachas largas, mientras que un bajo valor de p implica escasa probabilidad de las mismas.

El código Golomb para valores enteros no negativos n depende de la elección de un parámetro m . Para RLE este parámetro depende de la probabilidad p .

El primer paso para hallar el código Golomb para valores enteros no negativos n es calcular tres cantidades: q , r y c definidas como:

$$\begin{cases} q = \lfloor \frac{n}{m} \rfloor \\ r = n - q \times m \\ c = \lceil \log_2(m) \rceil \end{cases} \quad (4.16)$$

En el paso siguiente se construye el código en dos partes: un prefijo que representa el valor del cociente q codificado en *unario* y un sufijo.

El código unario del entero no negativo q está definido por q unos seguidos por un cero, como se muestra en la tabla 4.1. Es evidente que así la longitud del código unario para un entero q es de $q + 1$ bits.

El lo que hace al sufijo, si m es potencia de dos (lo que implica $m = 2^c$) entonces el sufijo es la representación binaria del resto r en c bits. El cálculo de los estos prefijos y sufijos para $n \in [0, 15]$ y $m \in \{2, 4\}$ se muestra en la tabla 4.2.

Si m no es potencia de dos, entonces el sufijo es de longitud variable, dividiéndose en dos casos:

- Si $r < 2^c - m$, el sufijo es la representación binaria del resto r , representado en $\lceil \log_2(m) \rceil =$

q	Código Unario	Código Unario Alternativo
0	0	1
1	10	01
2	110	001
3	1110	0001
4	11110	00001
5	111110	000001
6	1111110	0000001
$\vdots$	$\vdots$	$\vdots$

Tabla 4.1: Código Unario de q

n	Prefijo Unario $m = 2$	Sufijo Binario $m = 2$	Prefijo Unario $m = 4$	Sufijo Binario $m = 4$
0	0	0	0	00
1	0	1	0	01
2	10	0	0	10
3	10	1	0	11
4	110	0	10	00
5	110	1	10	01
6	1110	0	10	10
7	1110	1	10	11
8	11110	0	110	00
9	11110	1	110	01
10	111110	0	110	10
11	111110	1	110	11
12	1111110	0	1110	00
13	1111110	1	1110	01
14	11111110	0	1110	10
15	11111110	1	1110	11

Tabla 4.2: Códigos de Golomb cuando $m = 2^c$, o *GPO2*

n	Prefijo Unario $m = 3$	Sufijo Binario $m = 3$	Prefijo Unario $m = 5$	Sufijo Binario $m = 5$
0	0	0	0	00
1	0	10	0	01
2	0	11	0	10
3	10	0	0	110
4	10	10	0	111
5	10	11	10	00
6	110	0	10	01
7	110	10	10	10
8	110	11	10	110
9	1110	0	10	111
10	1110	10	110	00
11	1110	11	110	01
12	11110	0	110	10
13	11110	10	110	110
14	11110	11	110	111
15	111110	0	1110	00

Tabla 4.3: Códigos de Golomb cuando $m \neq 2^c$

$c - 1$ bits.

- Si $r \geq 2^c - m$, el sufijo es la representación binaria de $r + 2^c - m$ en $\lceil \log_2(m) \rceil = c$ bits.

Para fijar estos conceptos, supongamos que $m = 3$, produciendo $c = 2$ y tres restos $r = 0$, $r = 1$, $r = 2$. Calculamos $2^c - m = 2^2 - 3 = 1$, entonces el primer sufijo representa al resto 0 codificado en $c - 1 = 1$ bit, mientras que los sufijos siguientes representan la codificación binaria de $r + 2^c - m$ en $c = 2$ bits, obteniendo 10,11 correspondiente a los restos $r = 1$ y $r = 2$ respectivamente.

Si seleccionamos $m = 5$ resulta en $c = 3$ y produce naturalmente cinco restos 0, 1, 2, 3, 4.

Los primeros tres ($2^c - m = 2^3 - 5 = 3$) son codificado en $c - 1 = 2$ bits, y los dos restantes son codificados en $c = 3$ bits. Por lo tanto la codificación del sufijo es 00, 01, 10, 110 y 111. En la tabla 4.3 se muestran la codificación de Golomb para $n \in [0, 15]$ y $m \in \{3, 5\}$.

Decodificación de Golomb

Los códigos de Golomb fueron diseñados de forma que su decodificación sea sencilla. Para explicar la misma distinguiremos nuevamente los entre el caso en que m es potencia de dos, y cuando no lo es.

Comenzamos por el caso más simple, cuando m es potencia de dos. En este caso el primer paso a seguir es contar el número U de 1 consecutivos precediendo al primer cero a la izquierda. Entonces la longitud del código es $U + 1 + c$, donde el 1 corresponde al cero de la representación unaria y $c = \lceil \log_2(m) \rceil$ como en la etapa de codificación.

Si denotamos por R a los $1 + c$ bits a la derecha, entonces el valor del código es $m \times U + R$. Los simple de la de-codificación refleja la forma robusta en que el código fue construido.

Lo mostraremos con un ejemplo sencillo. Supongamos $m = 16$, y el siguiente código a decodificar 01100. Primero calculamos $c = 4$ obteniendo $R = 12$, luego no hay 1 antes del primer cero, por lo tanto $U = 0$. Así, la longitud del código es de $U + 1 + c = 5$ y el valor correspondiente

es $16 \times 0 + 12 = 12$.

Por otro lado, si m no es potencia de dos, entonces la decodificación es ligeramente diferente a la vista anteriormente. Nuevamente comenzamos determinando la cantidad de U unos consecutivos seguidos de un cero. Llamaremos R a los $(C - 1)$ bits siguientes.

- Si $R < 2^c - m$ entonces el total de la longitud del código es $U + 1 + (c - 1)$ y el valor del código es $m \times U + R$.
- Si $R \geq 2^c - m$ en este caso la longitud del código es $U + 1 + c$ y el valor del código es $m \times U + R - (2^c - m)$, donde R es un número entero de c bits, formado con los bits de R y el bit siguiente.

A modo de ejemplo supongamos $m = 14$ y el código $0001xxx$. No hay unos antes del primer cero, entonces $U = 0$. Eliminamos el cero de la representación unaria y calculamos $c - 1 = 3$, entonces los siguientes tres bits corresponden a R ($R = 001$). Estamos en el primer caso $R < 2^c - m$, por lo tanto el valor codificado es 1.

Cálculo del m óptimo para RLE

Hasta ahora hemos dado por conocido el valor m con que se construye el código de Golomb. Veremos que para utilizar los códigos de Golomb en RLE el valor óptimo para m es el número entero el más cercano a $\frac{-1}{\log_2(p)}$, lo que equivale al valor m que satisface la siguiente ecuación:

$$p^m \approx \frac{1}{2} \quad (4.17)$$

La ecuación 4.17 puede ser refinada y ampliada la en la relación más exacta:

$$p^m + p^{m+1} \leq 1 < p^m + p^{m-1} \quad (4.18)$$

Ellos demostraron que el código de Golomb es el mejor código de prefijo cuando m es seleccionado por su desigualdad. Se muestra a continuación que para un p dado, la desigualdad 4.18 tiene sólo un m de solución:

$$p^m(1 + p) \leq 1 < p^{m-1}(1 + p) \quad (4.19)$$

$$p^m \leq \frac{1}{(1 + p)} < p^{m-1} \quad (4.20)$$

$$m \geq \frac{1}{\log_2(p)} \log_2 \left(\frac{1}{p + 1} \right) > m - 1 \quad (4.21)$$

$$m \geq \frac{-\log_2(1 + p)}{\log_2(p)} > m - 1 \quad (4.22)$$

El único valor de m , que cumple con la inecuación 4.22 es:

$$m = \lceil \frac{-\log_2(p + 1)}{\log_2(p)} \rceil \quad (4.23)$$

Este cálculo de m sirve solo para el caso donde p es conocido (o puede ser estimado). De esta manera *no puede calcularse* m en el caso de que p es desconocido y no puede ser calculado por adelantado, como por ejemplo cuando la cadena de bits a codificar es muy larga y tiene que ser comprimida en tiempo real. En tal caso, es necesario utilizar un algoritmo adaptativo que varíe m según la entrada.

ϵ	$M(\epsilon)$	$M'(\epsilon)$
0	0	1
-1	1	0
1	2	3
-2	3	2
2	4	5
-3	5	4
3	6	7
$\vdots$	$\vdots$	$\vdots$

Tabla 4.4: Mapeo de Rice

Golomb-Rice en LOCO-I/JPEG-LS

Recordemos que en LOCO-I/JPEG-LS el error de predicción corregido ϵ , distribuido de acuerdo a 4.10, es codificado utilizando una sub-familia de mínima complejidad de la familia de códigos de prefijo óptimo para TSGD, basados en los Códigos de Golomb. Las características principales de estos últimos fueron vistas en las sub-secciones anteriores. Entre ellas se observó que los son óptimos para OSGD.

Sin embargo, el error de predicción corregido puede tomar de hecho valores *negativos*. Para alcanzar el requerimiento OSGD a partir de TSGD, LOCO-I/JPEG-LS hace uso del *Mapeo de Rice* definido a partir de:

$$M(\epsilon) = \begin{cases} 2\epsilon & \text{si } \epsilon \geq 0 \\ 2|\epsilon| - 1 & \text{si } \epsilon < 0 \end{cases} \quad (4.24)$$

De este modo el los valores del error de predicción corregido ϵ en el rango:

$$\frac{-\alpha}{2} \leq \epsilon \leq \frac{-\alpha}{2} - 1 \quad (4.25)$$

son mapeados a los valores $M(\epsilon)$ en el rango:

$$0 \leq M(\epsilon) \leq \alpha - 1 \quad (4.26)$$

El mapeo de Rice [19] $M(\epsilon)$, suponiendo $s \leq \frac{1}{2}$ ordena el error de predicción en un orden (al menos) *no creciente de probabilidad* asociando valores de ϵ positivos y negativos a valores de $M(\epsilon)$ pares e impares respectivamente (y asociando a 0 el valor 0). Esto se observa en la tabla 4.4. En el caso en que $s > \frac{1}{2}$ el mapeo correspondiente es $M'(\epsilon) = M(-\epsilon - 1)$ efectuando este el orden de probabilidad no creciente en este caso.

Se obtienen así valores enteros no negativos $M(\epsilon)$ cuya distribución puede ser aproximada por una geométrica, los cuales puede así ser codificados utilizando apropiadamente Golomb.

Como vimos en las sub-secciones anteriores tanto la codificación como la de-codificación de Golomb son muy sencillas para el caso en que m es potencia de dos. En este caso particular el método de codificación es denominado *GOP2* (de la sigla en inglés “Golomb Power of 2”) o *Golomb-Rice*, ya que fue este último quien utilizó por primera vez los códigos GOP2 para codificar TSGDs centradas en cero, mapeadas a través de $M(\epsilon)$.

Con el objetivo de mínima complejidad presente en el compromiso *nivel de compresión-complejidad*, como en todas las decisiones importantes de LOCO-I/JPEG-LS, los códigos de Golomb-Rice son seleccionados como método de codificación. En este caso se tiene $m = 2^k$ con lo que la

notación para la codificación pasa a ser G_{2^k} o R_k .

Como observación, el mapeo $M'(\epsilon)$ es solo relevante para $k = 0$ ya que para todo $k > 0$ se tiene que el largo del código para $R_k(M(\epsilon))$ es igual al largo del código para $R_k(M(-\epsilon - 1))$. De este modo solo resta para realizar la codificación la elección apropiada del parámetro k , y en el caso $k = 0$ la elección de un mapeo, $M(\cdot)$ o $M'(\cdot)$.

Estimación del Parámetro k óptimo

En este punto se repasan brevemente los puntos principales del procedimiento para la selección del k óptimo, es decir aquel que conduce a la mínima longitud media de código para $M(\epsilon)$ bajo las hipótesis establecidas anteriormente para la distribución de ϵ (es decir aproximadamente TSGD centrada en cero). Los esquemas de codificación basados en Golomb-Rice siguen normalmente dos enfoques para esta tarea:

- Estimación orientada a bloque. Se divide la imagen en bloques rectangulares de píxeles de 6×6 píxeles o 8×8 píxeles. Luego se determina el k óptimo por bloque, y este es enviado al decodificador junto con el bloque codificado. Este parámetro puede ser estimado tanto con una búsqueda exhaustiva como con un calculo explícito.
- Estimación secuencial. En este caso cada píxel es codificado por separado, y el codificador determina el parámetro óptimo en base a los valores de los píxeles anteriores. Al igual que la estimación orientada a bloques el parámetro k puede ser estimado tanto con una búsqueda exhaustiva como con un calculo explícito.

En modelos basados en contextos una estimación secuencial es necesaria ya que debido a que los píxeles codificados en un determinado contexto no son necesariamente contiguos, por lo tanto no pueden ser separados en bloques. El método utilizado en LOCO-I/JPEG-LS para la estimación del parámetro k es secuencial, explícito y basado en la esperanza del valor absoluto del error de predicción corregido $E[|\epsilon|]$ de la secuencia pasada.

Como se estableció en 4.10 si se considera una distribución $P_{(\theta, \epsilon)} = C(\theta, \epsilon)\theta^{|\epsilon+s|}$ para el error de predicción (con todos los parámetros como antes) la esperanza del valor absoluto del error de predicción corregido ϵ es:

$$a_{(\theta, \alpha)} \triangleq E[|\epsilon|] = \sum_{\epsilon = \frac{-\alpha}{2}}^{\frac{\alpha}{2}-1} C(\theta, \epsilon)\theta^{|\epsilon+s|}|\epsilon| \quad (4.27)$$

Necesitamos encontrar una relación entre $a(\theta, \alpha)$ y la longitud media del código $L_{\theta, k}$ obtenido aplicando la codificación de Golomb-Rice R_k al error de predicción mapeado por $M(\epsilon)$. Esto se debe a que buscamos la k que produzca la longitud de código menor. Par un alfabeto infinito de ϵ está demostrado que el valor óptimo de k puede ser estimado por la siguiente ecuación:

$$k = \lceil \log_2 a_{(\theta, \alpha)} \rceil \quad (4.28)$$

Para hallar computacionalmente k , tanto el codificador como el decodificador mantienen dos variables por contexto N y A donde N es como antes un contador de la ocurrencia del contexto y A es la suma acumulada del valor absoluto del error de predicción corregido para tal contexto. De esta forma la esperanza del valor absoluto del error de predicción corregido puede ser estimada por la relación $\frac{A}{N}$, y k puede calcularse en forma iterativa a través de:

ϵ	$M(\epsilon)$	$M'(\epsilon)$	$R_k(M(\epsilon))$	$R_k(M'(\epsilon))$
0	0	1	00	01
-1	1	0	01	00
1	2	3	100	101
-2	3	2	101	100
2	4	5	1100	1101
-3	5	4	1101	1100
3	6	7	11100	11101
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Tabla 4.5: $\epsilon, M(\epsilon), M'(\epsilon), R_k(M(\epsilon))$ y $R_k(M'(\epsilon))$ para $k = 1$

$$k = \min \left\{ k' | 2^{k'} N \geq A \right\} \quad (4.29)$$

Esto posibilita el simple cálculo de la palabra código para cualquier muestra sin utilizar tablas de códigos como en el caso de los códigos de Huffman (ver [17]) genéricos no estructurados.

Selección del mapeo $M(\cdot)$ o $M'(\cdot)$

Observando la tabla 4.5 vemos que debido al orden impuesto por el mapeo $M(\cdot)$ el codificador asignará un código más corto a los ϵ negativos que a los positivos.

Teniendo en cuenta que el “centro” de los errores de predicción corregidos TSGD se encuentra en $-s$, este mapeo parece natural y adecuado si $s \leq \frac{1}{2}$, lo que implica el centro de la TSGD más cercano a 0 que -1 .

Por otro lado si $\frac{B}{N} < -0.5$, recordando que $\frac{-B}{N}$ es un estimador de s , el centro de la distribución será más cercano a -1 que a 0 por lo que, como se observa en la tabla 4.4 el mapeo más adecuado es $M'(\cdot)$.

Sin embargo, como se menciono anteriormente que el mapeo $M'(\epsilon)$ es solo relevante para $k = 0$ ya que para todo $k > 0$ se tiene que el largo del código para $R_k(M(\epsilon))$ es igual al largo del código para $R_k(M(-\epsilon - 1))$. Esto se puede visualizar como ejemplo en la tablas 4.5 y 4.6. En la tabla 4.5 se ve que no hay diferencia entre la longitud del código $R_k(M(\epsilon))$ y $R_k(M(-\epsilon - 1))$ cuando $k \neq 0$, mientras que si existe tal diferencia cuando $k = 0$.

Así, en LOCO-I/JPEG-LS el mapeo utilizado para $k \neq 0$ es $M(\epsilon)$ mientras que si $k = 0$ y $\frac{B}{N} \leq -0.5$ el mapeo utilizado es $M'(\epsilon)$. Si $k = 0$ pero $\frac{B}{N} > -0.5$ el mapeo utilizado continúa siendo $M(\epsilon)$.

Golomb-Rice limitado

El método de codificación visto hasta este punto tiene la desventaja de permitir significativas expansiones para algunas situaciones particulares. Por ejemplo si $\alpha = 256$, $k = 0$ y $\epsilon = -128$ la codificación de este valor emplearía 256 bits.

En general esto no es causa problemas con el nivel de compresión alcanzado, ya que las expansiones significativas son raras en imágenes tanto naturales como artificiales y de ocurrir no llegan a afectar las estadísticas globales. No obstante, puede ser un problema a la hora de implementar el algoritmo debido al espacio limitado para almacenar datos comprimidos temporalmente.

Por este motivo LOCO-I/JPEG-LS toma la decisión de limitar el largo de código por muestra

ϵ	$M(\epsilon)$	$M'(\epsilon)$	$R_k(M(\epsilon))$	$R_k(M'(\epsilon))$
0	0	1	0	10
-1	1	0	10	0
1	2	3	110	1110
-2	3	2	1110	110
2	4	5	11110	111110
-3	5	4	111110	11110
3	6	7	1111110	11111110
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Tabla 4.6: $\epsilon, M(\epsilon), M'(\epsilon), R_k(M(\epsilon))$ y $R_k(M'(\epsilon))$ para $k = 0$

codificada a 4β con $\beta \triangleq \lceil \log_2(\alpha) \rceil$, lo que conduce a 32 bits para $\alpha = 256$ (o sea, imágenes de 8 bits por píxel). Este límite es denominado L_{max} .

Así, la codificación de un entero n en el rango $0 \leq n \leq \alpha$ procede como hasta ahora siempre que q (definido por 4.16 con $m = 2^k$, como antes para Golomb-Rice) cumpla:

$$q < L_{max} - \beta - 1 \triangleq q_{max} \quad (4.30)$$

lo cual constituye la situación más común. Para el caso de 8 bits por píxel $q_{max} = 32 - 8 - 1 = 23$. Ahora bien, si resulta $q \geq q_{max}$ se introduce un “código de escape” consistente en la representación unaria de q_{max} . A continuación se inserta la representación binaria de $n - 1$ usando β bits. La longitud de código total llega así a $q_{max} + 1 + \beta = L_{max}$ bits.

Es evidente que $n = 0$ siempre cumple con 4.30, de modo que $n - 1$ en el rango $0 \leq n < \alpha$ siempre podrá ser codificado con β bits.

4.1.6. Reseteo

Para finalizar la descripción del modo normal de LOCO-I/JPEG-LS notaremos que con el fin de lograr cierta adaptabilidad a las estadísticas no estacionarias que presentan las imágenes, para el recorrido que este algoritmo hace de las mismas, esta previsto un reseteo periódico de las variables por contexto N , A y B . Esto se lleva a cabo cuando la ocurrencia de un determinado contexto Q supera cierto umbral N_0 y consiste en dividir entre dos el valor de las variables por contexto antes mencionadas.

En LOCO-I/JPEG-LS el valor por defecto para efectuar el reset es $N_0 = 64$, aunque se ha encontrado una performance aceptable para N_0 entre 32 y 256.

4.1.7. El Modo de Rachas

Recordamos que este modo es elegido cuando $a = b = c = d$, o lo que es lo mismo $D1 = D2 = D3 = 0$, ya que entonces una racha del valor a es esperable (ver figura 4.2). La codificación “símbolo por símbolo” de TSGD que se lleva a cabo en LOCO-I/JPEG-LS en el modo normal adolece de ser ineficiente en regiones de baja entropía. Esto es debido a que en estas regiones de la imagen un error de predicción de cero es extremadamente probable, y dado que con este tipo de codificación se agrega al menos un bit por codificación, el código producido puede poseer una redundancia significativa.

Algunos esquemas de codificación encaran este problema mediante *extensión del alfabeto*. Esto significa codificar bloques de datos como "super símbolos". LOCO-I / JPEG-LS sigue una estrategia análoga al ingresar en modo rachas. Se puede decir que este modo esta compuesto de tres pasos principales:

1. *Medir la Racha*. Esto consiste simplemente en determinar cuantas veces se repite el píxel a en la fila a la cual este pertenece. La racha no se extiende más allá de la fila en proceso de codificación. El largo de la racha puede ser cero.
2. *Codificar la Racha*. El largo de la racha es codificado. No hay necesidad de codificar valores de píxel, ya que todos son iguales al píxel a . Mientras el codificador se encuentra en una racha, el contexto no es controlado, aunque para algunos píxeles de la racha este puede ser distinto del que conduce al modo rachas.
3. *Codificar Muestra de Interrupción*. Si la racha termina antes del fin de la fila, se codifica el error de predicción $\epsilon = x - b$ (siendo en este caso b la predicción del píxel x , al suponer que la racha termina en un borde vertical que pasa por x) y se vuelve al modo normal.
4. *Reseteo*. Este se realiza del mismo modo, sobre las variables N , A , B y N_n (cuya utilidad se describe en los siguientes puntos) cuando se cumplen las mismas condiciones que en el modo normal.

De estos puntos, solo los dos centrales merecen una explicación detallada, las cuales se presentan en las siguientes subsecciones.

Codificar Racha

La codificación del largo de la racha en LOCO-I/JPEG-LS está de nuevo basada en los códigos de Golomb, siguiendo una estrategia adaptativa.

Dado m_r el orden del código, con m_r una potencia kr de dos (o sea $m_r = 2^{kr}$), un 1 es utilizado para codificar segmentos de la racha de longitud m_r así como segmentos de longitud más corta interrumpidos por un fin de fila. Por otra parte si la racha es interrumpida por un píxel $x \neq a$, una palabra código de largo $kr + 1$ es utilizada para para codificar el segmento de racha faltante.

Cuando la longitud del segmento de racha m_r es alcanzada se dice que ocurrió un "acierto" mientras que cuando la racha es interrumpida antes de alcanzar el largo "máximo" de segmento m_r) se asocia con un "fallo".

En caso de fallo por un píxel diferente al píxel a la palabra código que sigue se compone de un *bit prefijo* igual a 0 en este caso, más la representación binaria del segmento de racha faltante utilizando kr bits. Si la racha es interrumpida por un fin del fila solo se agrega el bit prefijo, pero igual a 1 en este caso.

El valor m_r es calculado en forma adaptativa, a partir de la tabla "J" de 32 entradas (tabla 4.7). El procedimiento consiste en incrementar el índice i cada vez que ocurre un acierto y decrementarlo cada vez que ocurre un fallo. Esta tabla brinda el valor $kr = J(i)$ para cada índice i de modo de calcular $m_r = 2^{kr}$.

A partir de la tabla J se llega a que el mínimo valor para m_r es $1 = 2^0$ y el máximo $32768 = 2^{15}$. Un par de ejemplos clarifican el procedimiento. Supongamos que se ha de codificar una racha de largo 8 interrumpida por un fin de fila. Esta puede escribirse como $1 + 1 + 1 + 1 + 2 + 2$, por lo que se codifica como seis 1 consecutivos (seis aciertos). Cada vez que un 1 se escribe, se

i	kr
1	0
2	0
3	0
4	0
5	1
6	1
7	1
8	1
9	2
10	2
11	2
12	2
13	3
14	3
15	3
16	3
17	4
18	4
19	5
20	5
21	6
22	6
23	7
24	7
25	8
26	9
27	10
28	11
29	12
30	13
31	14
32	15

Tabla 4.7: Tabla $kr = J(i)$

incrementa i en una unidad y se resta del largo de la racha la longitud del segmento de racha correspondiente $2^{J(i)}$, de modo que en este ejemplo la misma decrece así:

$$7, 6, 5, 4, 2, 0$$

En este ejemplo no ocurrieron fallos, así que no hay nada que agregar. Sin embargo hay casos en que el largo de la racha no se puede expresar como suma de elementos $2^{J(i)}$ (cuando se presenta un fallo).

Por ejemplo para un largo de racha 7 que termina por un píxel $x \neq a$ la codificación consiste en cinco 1 ($1 + 1 + 1 + 1 + 2 = 6$), más el bit prefijo igual a 0 seguido por lo que resta de la racha, en este caso 1, representado utilizando $kr = 2^{J(5)} = 1$ bits. Si el origen del fallo hubiese sido el fin de fila solo se agregaría (a los cinco 1) el bit prefijo igual a 1.

Codificar Muestra de Interrupción

La codificación de la muestra de interrupción es similar a la codificación de una muestra en el modo normal, pero los pasos de predicción MED y corrección adaptativa son obviados. Además, existen solo dos posibles “contextos especiales”, estos son $a = b$ o $a \neq b$. Se introduce la variable $RIttype$ (de la sigla en inglés *Run Interruption Type* para tener en cuenta estas posibilidades, es decir:

- $RIttype = 0$ si $a \neq b$
- $RIttype = 1$ si $a = b$ ¹

Como se dijo anteriormente el error de predicción a codificar es $\epsilon = x - b$, al cual se le efectúa la reducción de rango del mismo modo que en el modo normal (ver 4.1.4).

Al igual que en el modo normal se procede a determinar la k de Golomb-Rice, pero en este caso a través de:

$$k = \min \left\{ k' \mid 2^{k'} N \geq T \right\} \quad (4.31)$$

donde:

- $T = A$ si $RIttype = 1$
- $T = A + \frac{N}{2}$ si $RIttype = 0$

Ahora bien, dado que en este modo no se realiza corrección adaptativa de la predicción, la variable B no es actualizada, por lo que la selección del mapeo a utilizar está basada en la cantidad de ocurrencias de un error de predicción negativo para cada contexto (como se dijo antes estos son $a = b$ y $a \neq b$), lo que se guarda en la variable N_n (la cual como N , A , B y C depende del contexto). De este modo el mapeo utilizado es:

$$M(\epsilon) = 2|\epsilon| - RIttype - map \quad (4.32)$$

donde la variable auxiliar $map \in \{0, 1\}$ es calculada a partir de la tabla 4.8.

Una vez realizado el mapeo sobre ϵ , $M(\epsilon)$ es codificado utilizando Golomb-Rice del mismo modo que se codifica en el modo normal, excepto que en este caso se utiliza Golomb-Rice limitado

¹En este caso se tiene siempre $\epsilon \neq 0$, ya que por definición la interrupción a una racha ocurre cuando $x \neq a$

$k = 0$	$\epsilon > 0$	$\frac{N_p}{N} < 0.5$	map
f	f	f	0 si $\epsilon = 0$, 1 si $\epsilon < 0$
f	f	v	0 si $\epsilon = 0$, 1 si $\epsilon < 0$
f	v	f	0
f	v	v	0
v	f	f	0 si $\epsilon = 0$, 1 si $\epsilon < 0$
v	f	v	0
v	v	f	0
v	v	v	1

Tabla 4.8: Tabla para determinar map . El símbolo v indica que la condición se cumple (“verdadera”) y el símbolo f indica lo contrario (“falsa”)

largo máximo L'_{max} que se calcula como:

$$L'_{max} = L_{max} - (kr + 1) \quad (4.33)$$

donde $kr + 1$ tiene en cuenta los bits utilizados en la codificación del último segmento de racha.

4.2. Resumen de la Implementación del Algoritmo: JPEG-LS

Antes de describir el funcionamiento del decodificador JPEG-LS es conveniente detallar algunos puntos de la implementación del método. Una descripción detallada de casi todas las etapas del mismo se puede encontrar en [6, ITU-T recomendación T.87]. Cabe mencionar que la implementación desarrollada se obtuvo siguiendo paso a paso lo allí especificado, salvo en el caso de procedimientos no especificados en esta recomendación internacional. En esos puntos excepcionales la implementación es propia de este grupo de proyecto de fin de carrera.

Inicialización Este es un punto fundamental para la implementación el método. En especial, la definición de la “imagen auxiliar” que determina el modo en como se codifica los píxeles de los bordes izquierdo, superior y derecho de la imagen, solo se encontró especificado en [6, ITU-T recomendación T.87]. Es evidente que sin este detalle el método no puede ser implementado correctamente.

La inicialización se compone de los siguientes pasos:

1. Calcular $RANGE$ definido como:

$$RANGE = MAXVAL + 1 \quad (4.34)$$

para una imagen de 8 bits por píxel se tiene $MAXVAL = 255$ y $RANGE = 256$. Además se le da a la constante $RESET$ el valor 64.

2. Calcular los siguientes parámetros:

$$\begin{cases} qbpp = \lceil \log_2 RANGE \rceil \\ bpp = \max(2, \lceil \log_2 RANGE \rceil) \\ LIMIT = 2 \times (bpp + \max(8, bpp)) \end{cases} \quad (4.35)$$

se tiene así $qbp = 8$, $bp = 8$ y $LIMIT = 32$

3. Inicializar los valores de $A[0 \dots 366]$ al valor:

$$\max \left(2, \left\lfloor \frac{RANGE + 2^5}{2^6} \right\rfloor \right) \quad (4.36)$$

mientras que los de $N[0 \dots 366]$ se inicializan a 1. Cabe destacar que los índices del vector 365 y 366 son los utilizados en el modo rachas, mientras que los $0 \dots 364$ son utilizados en el modo normal. Los vectores $B[0 \dots 364]$ y $C[0 \dots 364]$ se inicializan a al valor 0.

Como comentario, esta es una forma explicita de ver el funcionamiento “por contextos” de JPEG-LS, ya que índice a estos vectores será precisamente el contexto Q .

4. Inicializar el índice a la tabla 4.7 $J[0 \dots 31]$ $RUNindex$ a 0 y las variables $Nn[365]$ y $Nn[366]$ a 0. Todas estas variables se utilizan en el modo de rachas.
5. Generar la imagen auxiliar. Si la imagen original tiene dimensiones $(filas, columnas)$ la imagen auxiliar tendrá dimensiones $(filas+1, columnas+2)$. La misma se completa de modo tal que las diferencias $D1$, $D2$ y $D3$ de la ecuación 4.1 puedan calcularse para la primer fila, la primer columna y la última columna, en el orden usual con que JPEG-LS recorre la imagen. A modo de ejemplo, si la imagen a codificar es la siguiente:

0	1	10	20
3	69	120	120
2	75	120	200
4	1	10	20

La imagen auxiliar se crea del siguiente modo:

0	0	0	0	0	0
0	0	1	10	20	20
0	3	69	120	120	120
3	2	75	120	200	200
2	4	1	10	20	X

Si en esta imagen numeramos las filas desde 0 hasta $filas$ y las columnas de 0 a $columnas + 1$, el proceso de compresión obviamente comienza por el píxel $(1, 1)$ y termina por el $(filas, columnas)$. Es decir, involucra a los píxeles con coordenadas (i, j) con $1 \leq i \leq filas$ y con $1 \leq j \leq columnas$ recorriendo así los píxeles de la imagen original. El resto de los píxeles solo son necesarios para el calculo de las diferencias en los bordes de la imagen, como se dijo anteriormente.

La X en el píxel del ángulo inferior derecho de la imagen indica que el valor de esta píxel no interesa, ya que no pertenece a la imagen original y no participa en el cálculo de las diferencias $D1$, $D2$ y $D3$.

Cálculo de “gradientes locales” Se calculan las diferencias $D1$, $D2$ y $D3$ como se definen en 4.1. Si resulta que todas ellas son nulas, ir al modo de rachas. En caso contrario, continuar. Una observación importante es que de acuerdo a como se forma la imagen auxiliar, el primer píxel de cada imagen es codificado en el modo rachas, esperando una racha del valor 0. Este hecho, conocido por codificador y decodificador proporciona la “llave”

con la cual comenzar a decodificar un flujo de bits proveniente de un codificador JPEG-LS.

Cuantización de gradientes Las diferencias son cuantizadas en $Q1$, $Q2$ y $Q3$. La forma más simple de explicar este procedimiento es mediante el siguiente código en lenguaje C:

```
if(Di<=-T3) Qi=-4;
else if(Di<=-T2) Qi=-3;
else if(Di<=-T1) Qi=-2;
else if(Di<=0) Qi=-1;
else if(Di<=0) Qi=0;
else if(Di<T1) Qi=1;
else if(Di<T2) Qi=2;
else if(Di<T3) Qi=3;
else Qi=4;
```

A través del mismo se observa como una diferencia D_i es cuantizada a $-T, \dots, -1, 0, 1, \dots, T$ con $T = 4$ con $T1 = 3$, $T2 = 7$, $T3 = 21$ intervalos de cuantización definidos en 4.1.2.

Cálculo de $SIGN$ y Q Si el primer elemento no nulo de la terna $(Q1, Q2, Q3)$ es negativo, entonces la variable $SIGN$ se hace igual a -1 y los elementos de la terna invierten su signo para obtener $(-Q1, -Q2, -Q3)$. Esto ejecuta la “mezcla de contextos” justificada en 4.1.2 a en base a la suposición:

$$Prob\{\epsilon_{t+1} = \Delta \mid C_t = [q_1, q_2, q_3]\} = Prob\{\epsilon_{t+1} = -\Delta \mid C_t = [-q_1, -q_2, -q_3]\}$$

Si el primer elemento no nulo de la terna mencionada es positivo, entonces la variable $SIGN$ se hace igual a 1 y no se altera la terna.

Posteriormente, es necesario mapear las 365 combinaciones de la terna $(Q1, Q2, Q3)$ a un entero Q para indexar los vectores A , B , C , N y Nn .

En [6, ITU-T recomendación T.87] no se especifica como realizar este procedimiento, solo se afirma que *debe realizarse en una base uno a uno, produciendo un entero en el rango $[0 \dots 364]$ y debe estar definida para todos los posibles valores del vector $(Q1, Q2, Q3)$ incluyendo el $(0, 0, 0)$.*

Teniendo en cuenta que cada uno de los Q_i puede tomar 9 valores diferentes, la forma de hacer esto propuesta por este grupo de proyecto es a través de:

$$Q = 81 \times Q1 + 9 \times Q2 + Q3 \quad (4.37)$$

Predicción y corrección de la misma Se elabora la predicción del píxel a codificar x utilizando el predictor MED definido en , obteniéndose P_x (o $\hat{x}$ en las secciones anteriores). Si la variable $SIGN$ es positiva la corrección se efectúa como:

$$P_x = P_x + C[Q]$$

en caso contrario como:

$$P_x = P_x - C[Q]$$

Estas asignaciones deben interpretarse como instrucciones en un lenguaje de programación como C, y no como ecuaciones matemáticas. De ahora en más el uso de uno u otro significados será claro a partir del contexto.

Luego de la corrección, si $P_x > MAXVAL$ se hace $P_x = MAXVAL$ o si $P_x < 0$ se hace $P_x = 0$.

Error de predicción El error de predicción $Errval$ se calcula como

$$Errval = x - P_x \quad (4.38)$$

Si la variable $SIGN$ es negativa se invierte el signo del mismo haciéndose $Errval = -Errval$.

Reducción del rango Como se especifica en 4.1.4 esto se consigue mediante el siguiente procedimiento:

```
if (Errval<0)
Errval=Errval+RANGE;
if (Errval>=((RANGE+1)/2))
Errval=Errval-RANGE;
```

Es importante notar que la segunda sentencia `if` se ejecuta *siempre* al igual que la primera. De este modo se mezclan las “colas” de la TSGD con su parte central. Nótese que $Errval$ cumple la misma función que la variable ϵ en secciones anteriores.

Cálculo del k de Golomb óptimo Se vió en la sección que el k óptimo es aquel que cumple:

$$k = \min \left\{ k' | 2^{k'} N \geq A \right\}$$

Este valor puede determinarse mediante el simple procedimiento:

`for(K = 0; (N[Q] << K) < A[Q]; K ++);`

lo que ha sido tomado en diversas publicaciones como ícono de la simple implementación del método de compresión JPEG-LS.

Selección del mapeo La elección de un mapeo $M(\cdot)$ o $M'(\cdot)$ para el error de predicción corregido $Errval$ justificada en 4.1.5 se puede implementar junto con el mapeo mismo a través del siguiente bloque de código:

```
if ( (K==0) && (2*BQ<=-NQ) ){
if (Errval>=0) MErrval=(2*Errval+1);
else MErrval=(-2*(Errval+1));
}
else{
if (Errval>=0) MErrval=(2*Errval);
else MErrval=(-2*Errval-1);
}
```

Codificación El valor $MErrorval$ es codificado ahora de acuerdo a lo especificado en 4.1.5 donde se detallan los códigos de Golomb limitados. Para ello se utiliza el valor de k que arroja el procedimiento visto líneas arriba y el valor límite para la longitud del código $LIMIT$.

Actualización y Reseteo Para culminar el modo normal, solo resta por realizar actualizaciones y, de cumplirse la condición necesaria, reseteos. De este modo las variables A y B se actualizan primero como:

$$B[Q] = B[Q] + Errorval$$

y

$$A[Q] = A[Q] + |Errorval|$$

Luego, si se ha alcanzado el número $RESET$ de ocurrencias de un contexto Q el contenido de las variables A , B y N es dividido entre dos mediante:

```

if (N[Q]==RESET){
A[Q]=A[Q]>>1;
if (B[Q]>=0) B[Q]=B[Q]>>1;
else B[Q]=-((1-B[Q])>>1);
N[Q]=N[Q]>>1;
}

```

Por último se actualizan las variables involucradas en la estimación del corrimiento con un procedimiento idéntico al de la sección 4.1.3.

Se pasa entonces a la siguiente muestra a codificar.

El modo rachas Cuando se ingresa al modo rachas, se ejecutan los siguientes pasos:

1. El primer paso es determinar el largo de la misma. Para ello se inicializa el valor $RUNval$ igual al valor del píxel a (ver figura 4.2). Cada vez que la muestra a codificar x resulta igual a $RUNval$ se incrementa la variable $RUNcnt$ en uno.
2. Se codifica el largo de la racha utilizando el procedimiento descrito en 4.1.7, es decir haciendo uso de la tabla J , tabla 4.7 y del índice a la misma $RUNindex$. Una implementación de este procedimiento es la siguiente:

```

while (RUNcnt>=(1<<J[RUNindex])){
Stream->append(1,1);
RUNcnt=RUNcnt-(1<<J[RUNindex]);
if (RUNindex<31) RUNindex++;
}

```

donde, el método “ $append(x, y)$ ” de la clase “Stream”, agrega al flujo de bits producidos por el codificador la representación binaria de x utilizando y bits.

3. Si la racha es interrumpida por un fin de línea:

- a) Si $RUNcnt > 0$ se agrega un 1 al flujo de bits mediante `Stream->append(1,1)`
- b) Ya no hay más que hacer y se pasa a la siguiente muestra a codificar.
- 4. Si por otro lado la racha es interrumpida por un valor de x distinto de $RUNval$:
 - a) Agregar al flujo de bits un 0 mediante `Stream->append(0,1)`
 - b) Agregar al flujo de bits la representación binaria de $RUNcnt$ usando $J[RUNindex]$ bits mediante `Stream->append(RUNcnt, J[RUNindex])`
 - c) Si $RUNindex > 0$ decrementarlo en uno.
- 5. Se codifica la muestra de interrupción siguiendo el procedimiento descrito en 4.1.7.
- 6. Finalmente, en forma análoga al modo normal se actualizan las variables A y Nn , luego se resetean de ser necesario A Nn y N , y para culminar se actualiza N .

4.2.1. Comentarios

Si bien pueden resultar algo tediosos los detalles expuestos sobre la implementación del algoritmo, son útiles ya que permiten visualizar el proceso de codificación como una cadena de procedimientos con implementación sencilla.

Cuando existen restricciones sobre el tiempo que el proceso de codificación puede insumir, esta característica se vuelve muy interesante.

Un hecho a destacar es que *no son necesarias* variables en representación de *punto fijo* o *punto flotante*. Además las operaciones involucradas son la suma, resta, y la traslación o shift de los bits que forman una variable.

Todo el método de principio a fin puede ser implementado utilizando las variables del lenguaje C o C++ “unsigned char” “short” e “int”, siendo la primera la más utilizada requiere solo 1 byte para su almacenamiento.

Esto conduce a implementaciones muy eficientes en el uso de memoria y *veloces*.

También es necesario comentar que el modo rachas es el más rápido y el que logra los mayores niveles de compresión, pero es también en el que menos ingresa el codificador para imágenes naturales. Por el contrario, para imágenes generadas por un computador, el codificador ingresa más a menudo en el modo rachas, ya que se cumplen las características necesarias en la imagen para ello.

4.3. El Decodificador

El proceso de decodificación en LOCO-I/JPEG-LS es totalmente análogo al de codificación. Mas aún, gran parte de los procedimientos que realiza el codificador se repiten de forma idéntica en el decodificador.

En [5] se afirma que el proceso de decodificación es aproximadamente 10 % más lento que el de codificación, por lo que el algoritmo puede ser considerado casi simétrico en ese aspecto.

Describimos a continuación el proceso de decodificación. Al ser tan similar al proceso de codificación, esta descripción será rápida.

Es necesario acotar, que en [6, ITU-T recomendación T.87] se explica lo que deben hacer algunos procedimientos, pero no se detalla como. Si bien la implementación de estos no fue trivial, concluyo en series de rutinas simples en *armonía* con el resto de lo implementado.

Inicialización La inicialización de todas las variables es idéntica a la que realiza el codificador.

La imagen auxiliar se forma del mismo modo que en el decodificador, aunque al comienzo solo se conocen los siguientes elementos (para una imagen original de 4×4):

0	0	0	0	0	0
0	?	?	?	?	?
?	?	?	?	?	?
?	?	?	?	?	?
?	?	?	?	?	X

Cálculo de “gradientes locales” Se computan de igual modo que en el codificador. Como se observa en la imagen auxiliar anterior, análogamente a como procede el codificador, el primer píxel de cada cuadro debe decodificarse en el modo rachas, ya que en este caso siempre se tiene $D1 = D2 = D3 = 0$.

El modo rachas En la decodificación de este modo se procede como sigue:

1. Se lee un bit R del flujo de bits.
2. Si $R = 1$:
 - a) Inicializar $RUNcnt = 2^J[RUNindex]$ y completar la imagen con $RUNcnt$ píxeles de valor a (ver figura 4.2), o hasta que se termine la fila.
 - b) Si se escribieron exactamente $2^J[RUNindex]$ píxeles en el paso anterior (equivalente a $RUNcnt = 0$) y $RUNindex < 31$, entonces incrementar $RUNindex$ en uno.
 - c) Si no se ha terminado la fila, se vuelve al comienzo a leer un nuevo bit R . Si por otro lado, se ha llegado al fin de fila, se pasa a la siguiente muestra a decodificar.
3. Si $R = 0$:
 - a) Se deben leer $J[RUNindex]$ bits del flujo de bits y completar la imagen con tantos píxeles iguales al píxel a como el número formado por estos bits lo indique.
 - b) Si $RUNindex > 0$ debe decrementarse en una unidad.
 - c) Ahora es momento de decodificar la muestra de interrupción. Este es el punto de mayor dificultad en este modo, aunque su implementación no es compleja.
 - 1) El primer paso es calcular RI_{type} del mismo modo que en la codificación y a partir de este la predicción P_x , lo cual se logra con el simple procedimiento:


```
if(a==b) RItype=1;
else RItype=0;
if(RItype==1) Px=a;
else Px=b;
```

 Como se observo en el codificador, P_x en realidad es siempre igual al píxel b , ya que cuando se tiene $P_x = a$ también se tiene $a = b$.
 - 2) En este punto se computan las variables k , $glimit$, Q y T del mismo modo que en el codificador (donde $glimit = L'_{max}$).
 - 3) Los puntos que siguen implican, leer $MErrval$ del flujo de bits e invertir el mapeo para obtener $Errval$. El primero de estos se logra mediante:

```
R=Stream->get(1);
CEROcnt=0;
```

```

while(R==0)
{
  CER0cnt++;
  R=Stream->get(1);
}
if(CER0cnt<(glimit-qbpp-1))
{
  if (K>0)
  { MErrval=CER0cnt<<K;
    MErrval|=Stream->get(K);
  }
  else MErrval=CER0cnt;
}
else
MErrval=1+Stream->get(qbpp);

```

El método “*get(x)*” de la clase “Stream” devuelve el numero formado por la representación binaria de los x bits que toma del flujo de bits. Ahora, se obtiene Errval de acuerdo a como este fue mapeado:

```

Errval=((MErrval+RIttype+1)>>1);
if((MErrval+RIttype+1)%2==0) map=1;
else map=0;
if((K==0)&&(map==1)&&((2*Nn[Q])<N[Q]))
Errval=Errval;
else if((map==1)&&((2*Nn[Q])>=N[Q]))
Errval=-Errval;
else if ((map==1)&&(K!=0))
Errval=-Errval;
else
{
  if((K!=0)||((2*Nn[Q])>=N[Q]))
  Errval=Errval;
  else
  Errval=-Errval;
}

```

Ahora, solo resta actualizar Nn invertir el cambio efectuado por la variable $SIGN$ y reconstruir el valor de la muestra de interrupción Rx invirtiendo la reducción de rango:

```

if(Errval<0)
Nn[Q]=Nn[Q]+1;
if((RIttype==0)&&(Ra>Rb)) SIGN=-1;
else SIGN=1;
if(SIGN==-1) Errval=-Errval;
Rx=Px+Errval;
if(Rx<0) Rx=Rx+RANGE;
else if(Rx>MAXVAL) Rx=Rx-RANGE;
if(Rx<0) Rx=0;

```

```
else if(Rx>MAXVAL) Rx=MAXVAL;
```

- d) Una vez decodificada la muestra de interrupción solo resta por realizar las actualizaciones de variables y el reseteo de ser necesario, en forma totalmente análoga al proceso de codificación.

El modo normal La decodificación del modo normal es aún más similar a la codificación en este modo en comparación al modo rachas. El proceso seguido se muestra en los siguientes puntos:

1. Se calcula Q a partir de las diferencias $D1$, $D2$ y $D3$ del mismo modo que en el codificador.
2. Se calcula la variable $SIGN$ del mismo modo que el codificador.
3. Se calcula $\hat{x}$ o Px la predicción de la muestra a decodificar utilizando el predictor MED del mismo modo que el codificador.
4. Se corrige la predicción y se computa la k de Golomb del mismo modo que en el codificador.
5. Se lee del flujo de bits el error de predicción corregido mapeado $MErrval$ mediante el siguiente procedimiento:

```
while(R==0)
{
  CER0cnt++;
  R=Stream->get(1);
}
if(CER0cnt<(LIMIT-qbpp-1))
{
  if (K>0)
  {
    MErrval=CER0cnt<<K;
    MErrval|=Stream->get(K);
  }
  else MErrval=CER0cnt; }
else MErrval=1+Stream->get(qbpp);
```

6. Ahora, una vez leído $MErrval$ se obtiene $Errval$ invirtiendo el mapeo mediante:

```
if((K==0)&&((2*BQ)<=-NQ))
if(((MErrval-1)%2)==0) Errval=(MErrval-1)/2;
else Errval=(MErrval/(-2))-1;
else if((MErrval%2)==0) Errval=MErrval/2;
else Errval=(MErrval+1)/(-2);
```

7. Se realiza la actualización de las variables y reseteo de ser necesario, del mismo modo que lo hace el codificador.
8. Utilizando la variable $SIGN$ se invierte el signo del error de predicción corregido decodificado (de ser necesario) y se invierte la reducción de rango mediante:

```
Rx=Px+Errval;  
if(Rx<0) Rx=Rx+RANGE;  
else if(Rx>MAXVAL) Rx=Rx-RANGE;  
if(Rx<0) Rx=0;  
else if(Rx>MAXVAL) Rx=MAXVAL;
```

Con lo que concluye la decodificación.

5 ALGORITMOS PROPUESTOS PARA EL SISTEMA

Una vez llevada a cabo la implementación estándar del algoritmo de compresión de imágenes sin pérdidas JPEG-LS, se abre un amplio campo para la experimentación en la compresión de secuencias de video. Es decir, entendiendo cada paso del algoritmo y de su implementación, es posible realizar modificaciones razonables a éste y observar qué ocurre con los parámetros de evaluación mencionados en 3.2.2. Como se verá en las siguientes secciones, estas modificaciones estarán orientadas a tratar de explotar la redundancia temporal presente en las secuencias de video, cosa que obviamente JPEG-LS no contempla, ya que es un método de compresión de imágenes y no de video. En definitiva, se buscará *extender* JPEG-LS a video. Cabe destacar que todos los experimentos que se mencionarán fueron realizados en un PC con procesador Pentium IV de 3.06 GHz.

5.1. JPEG-LS Cuadro a Cuadro

Si bien lo que se busca es extender JPEG-LS a video, el primer paso debe ser considerar JPEG-LS como método para la compresión sin pérdidas de cada cuadro de este video. Esto no solo hará explícitas las fortalezas y debilidades de este procedimiento, sino que será un punto de referencia importante, al tratarse de una implementación estándar de JPEG-LS aplicada a secuencias de pruebas ampliamente conocidas y utilizadas.

Llamaremos al método que consiste en comprimir cada cuadro del video de prueba mediante JPEG-LS *JPEG-LS Cuadro a Cuadro*.

Los resultados de JPEG-LS Cuadro a Cuadro aplicado al conjunto de videos de prueba se muestran en las tablas 5.1 y 5.2.

Entre sus ventajas se encuentran las siguientes:

- Es un método simple, basado en un estándar ITU. Esto hace posible reproducir las medidas anteriores con total precisión.
- Es notablemente más rápido que cualquiera de los métodos vistos en 3.1.2, 3.1.3 y 3.1.4 procesando, en media, aproximadamente cinco veces más cuadros por segundo que I-CALIC.
- La codificación cuadro a cuadro es independiente, es decir, en la codificación de un cuadro sólo interviene información obtenida de este cuadro. Esto es importante cuando por algún motivo la secuencia de bits que genera el codificador es contaminada con ruido (y no se implementan métodos para la detección o corrección de errores en este bitstream). Los cuadros al los que “pertenecen” estos bits serán decodificados con error. Pero, si el

Secuencia	Nivel de Compresión en bpp	Tiempo de Procesamiento
Salesman	4.63	00:13.60
Mobile y Calendar	5.82	00:09.76
Container	4.37	00:09.04
Tempete	4.98	00:08.16
Kitchgrass	4.68	00:03.03
Sean	3.97	00:08.96
Silent	4.50	00:09.01
Foreman	4.29	00:09.08
Promedio	4.66	00:08:63

Tabla 5.1: Nivel de compresión en bpp y tiempo de procesamiento para JPEG-LS Cuadro a Cuadro aplicado al conjunto de videos de prueba

Secuencia	Nivel de compresión en %	Cuadros procesados por segundo
Salesman	42.12	33.01
Mobile y Calendar	27.24	30.74
Container	45.32	33.19
Tempete	37.77	31.86
Kitchgrass	41.55	33.00
Sean	50.35	33.48
Silent	43.79	33.30
Foreman	46.33	33.04
Promedio	41.81	32.69

Tabla 5.2: Nivel de compresión en % y cuadros procesados por segundo para JPEG-LS Cuadro a Cuadro aplicado al conjunto de videos de prueba

Secuencia	Nivel de Compresión en bpp	Tiempo de Procesamiento
Salesman	4.62	00:13.28
Mobile y Calendar	5.79	00:09.75
Container	4.37	00:08.89
Tempete	4.97	00:08.10
Kitchgrass	4.67	00:02.98
Sean	3.96	00:07.27
Silent	4.48	00:08.90
Foreman	4.27	00:08.94
Promedio	4.64	00:08:51

Tabla 5.3: Nivel de compresión en bpp y tiempo de procesamiento para MeJPEG-LS aplicado al conjunto de videos de prueba

bitstream esta adecuadamente delimitado, este error no se propagará indefinidamente, ya que el siguiente cuadro puede decodificarse sin problemas.

Mientras que se pueden considerar desventajas los siguientes puntos:

- El método no explota la redundancia temporal presente en las secuencias de video. Esto conduce a niveles de compresión más bajos a los alcanzados en 3.1.2, 3.1.3 y 3.1.4 con un 6.1 % menos que I-CALIC y un 11.9 % menos que M-CALIC.
- Los vectores (ver 4) C , B , N , A y Nn deben ser re-inicializados a sus valores por defecto antes de codificar un nuevo cuadro, agregando tiempo de computo.

5.2. JPEG-LS con “Memoria” Inter-Cuadro

Como vimos en el punto anterior la principal desventaja de JPEG-LS Cuadro a Cuadro como método de compresión de video es el no explotar la redundancia temporal de las secuencias de video. Para contemplar este punto, una primera medida es mantener una *estadística común* al ejecutar la codificación y decodificación de los cuadros mediante JPEG-LS. Esto se traduce en no reinicializar los vectores C , B , N , A y Nn luego de que se ha codificado o decodificado un cuadro.

Justificamos esta decisión tomando como hipótesis que los cuadros temporalmente próximos de una secuencia de video son imágenes muy similares entre sí. De este modo, con una mayor cantidad de ocurrencias de un determinado contexto es esperable un mejor funcionamiento del algoritmo al codificar píxeles inmersos en dicho contexto. Ya que se mantiene una memoria inter-cuadro de la estadística de contextos llamaremos a este método *JPEG-LS Con Memoria* o en forma abreviada MeJPEG-LS.

Comparando las tablas 5.2 y 5.4 se desprende que:

- La mejora en el nivel de compresión respecto a JPEG-LS es de apenas un 0.18 % en media.
- La mejora en la cantidad de cuadros procesados por segundo con respecto a JPEG-LS es de 0.46 fps en media.

El incremento marginal en la velocidad del método se explica en que los vectores C , B , N , A y Nn no deben ser reinicializados luego de cada codificación o decodificación.

Secuencia	Nivel de compresión en %	Cuadros procesados por segundo
Salesman	42.24	33.81
Mobile y Calendar	27.60	30.77
Container	45.43	33.75
Tempete	37.87	32.10
Kitchgrass	41.57	33.56
Sean	50.61	34.05
Silent	44.02	33.71
Foreman	46.61	33.56
Promedio	41.99	33.15

Tabla 5.4: Nivel de compresión en % y cuadros procesados por segundo para MeJPEG-LS aplicado al conjunto de videos de prueba

Secuencia	Nivel de compresión en %	Cuadros procesados por segundo
Salesman	42.15	34.41
Mobile y Calendar	27.64	31.65
Container	45.32	34.01
Tempete	37.86	33.04
Kitchgrass	41.50	33.56
Sean	50.63	34.56
Silent	43.99	34.09
Foreman	46.59	34.05
Promedio	41.96	33.71

Tabla 5.5: Nivel de compresión en % y cuadros procesados por segundo para MeJPEG-LS aplicado al conjunto de videos de prueba, con $RESET = 128$

Con respecto al primer punto, es esperable que la mejora en el nivel de compresión no sea muy significativa debido al conocido carácter no estacionario de las imágenes que componen un video para el “recorrido” que JPEG-LS hace de ellas.

Recordemos que JPEG-LS toma como medida el reseteo (ver 4.1.6) de los vectores C , B , N , A y Nn cuando la ocurrencia de un contexto es mayor a 64 veces, controlando esto a través del parámetro $RESET$. Si incrementamos este valor, por ejemplo a 128 o 256 es esperable un peor desempeño, ya que el número 64 fue determinado como óptimo en [5] para la media de las imágenes en el compromiso *memoria contextual - no estacionariedad*. Sin embargo, nada se pierde con verificarlo evaluando el nivel de compresión para MeJPEG-LS aplicado al conjunto de los videos de prueba con $RESET = 128$ y $RESET = 256$. Los resultados se muestran en las tablas 5.5 y 5.6.

A partir de las tablas 5.4, 5.5 y 5.6 se obtiene la 5.7.

Se confirma la afirmación anterior, y es que el mejor nivel de compresión medio se obtiene para $RESET = 64$ como fue determinado en [5] para imágenes individuales. Con respecto al tiempo de procesamiento, la mejora marginal que se obtiene para $RESET = 256$ es menor a la obtenida para $RESET = 128$.

Resumimos a continuación las consideraciones sobre MeJPEG-LS. Como aspectos positivos, al no reinicializar los vectores C , B , N , A y Nn :

Secuencia	Nivel de compresión en %	Cuadros procesados por segundo
Salesman	42.08	34.49
Mobile y Calendar	27.66	31.75
Container	45.23	33.90
Tempete	37.84	32.87
Kitchgrass	41.42	33.44
Sean	50.61	34.52
Silent	43.93	34.13
Foreman	46.56	33.82
Promedio	41.92	33.67

Tabla 5.6: Nivel de compresión en % y cuadros procesados por segundo para MeJPEG-LS aplicado al conjunto de videos de prueba, con $RESET = 256$

RESET	Nivel de compresión medio en %	Cuadros procesados por segundo, en media
64	41.99	33.15
128	41.96	33.71
256	41.92	33.67

Tabla 5.7: Nivel de compresión en % y cuadros procesados por segundo para MeJPEG-LS aplicado al conjunto de videos de prueba, con diferentes valores de $RESET$

- Existe una mejora en el nivel de compresión, del 0.18 % en el mejor caso, que corresponde a $RESET = 64$.
- Existe también una mejora en la cantidad de cuadros procesados por segundo, de 0.56 fps en el mejor caso, que corresponde a $RESET = 128$

Como contrapartida:

- Ambas mejoras son claramente marginales.
- Se pierde la independencia en la codificación entre cuadros que brinda JPEG-LS Cuadro a Cuadro. Es decir, si un cuadro es mal decodificado por algún tipo de error en el bitstream, este error contaminará el contenido de los vectores C , B , N , A y Nn causando que se propague hacia adelante indefinidamente.

Esto nos lleva a que deben implementarse otras medidas para explotar algo de la redundancia temporal presente entre los cuadros del video. En el siguiente punto veremos la forma más usual de lograr este objetivo.

5.3. JPEG-LS + Compensación de Movimiento

Como vimos anteriormente, el mantener los vectores C , B , N , A y Nn sin ser re-inicializados entre codificaciones sucesivas no es una medida que explote la fuerte redundancia temporal presente entre cuadros próximos de secuencias de videos. La forma de explotar esta redundancia, tanto en algoritmos de codificación de video *con* pérdidas, como en los métodos vistos en

3.1.3 y 3.1.4) es mediante la *Compensación de Movimiento*. Por lo tanto, si queremos combinar JPEG-LS con técnicas para la compensación de movimiento debemos explorar el estado del arte para éstas; sin dejar de tener en cuenta su aptitud para ser utilizadas en un sistema con las características planteadas en 1.2.

5.3.1. Compensación de Movimiento

La Compensación de Movimiento es una técnica utilizada en codificación de video con la finalidad de explotar la redundancia temporal existente entre cuadros de la secuencia, y así disminuir la cantidad de bits necesarios para la codificación de dicho video. Una completa descripción de métodos para la compensación de movimiento se encuentra en [20].

El proceso de compensación de movimiento se basa en *estimar el movimiento* de objetos con respecto a un fondo, presentes en las imágenes que componen el video, mediante un algoritmo que analiza una secuencia de cuadros consecutivos del mismo. Tomando como hipótesis que los cuadros temporalmente próximos de una secuencia de video son imágenes muy similares entre sí, una buena predicción del cuadro a codificar estará dada por la compensación del movimiento respecto al cuadro anterior.

La idea básica es que si el codificador descubre que una región del cuadro anterior, ya sea un píxel o un conjunto de píxeles, se ha movido a un lugar diferente en el cuadro actual (manteniendo sus propiedades) entonces esa región puede ser codificada con la siguiente información: la ubicación en el cuadro anterior, la ubicación en el cuadro actual e información acerca de la forma de esa región de la imagen. La diferencia entre la posición anterior y la actual de esa región del cuadro se denomina *vector de movimiento*. Una vez estimado el movimiento, ya sea de un píxel o un conjunto, se envía el vector de movimiento junto con la codificación del *error de predicción* al receptor. El error de predicción se calcula como la diferencia entre la región en el cuadro actual y su contraparte dada por la estimación de movimiento en el cuadro anterior.

En general para explotar la redundancia temporal, se divide al cuadro en bloques de píxeles y se estima el movimiento de cada uno de éstos, calculando un único vector de movimiento por bloque.

En síntesis esta técnica se divide en dos etapas:

1. Estimación del movimiento: Consiste en identificar los movimientos entre cuadros consecutivos, es decir, hallar los vectores de movimiento.
2. Compensación del movimiento: Una vez hallados los vectores del movimiento, éstos son utilizados para la reconstrucción del cuadro.

Estimación del movimiento

Para realizar la estimación del movimiento es necesario efectuar un modelado del mismo. Este modelado puede realizarse a la imagen completa, píxel a píxel o a regiones. Cuanto mayor sea el número de parámetros a tener en cuenta mejor será la estimación del movimiento, pero también se incurrirá en un aumento en la carga de procesamiento, lo que implica un compromiso entre mejor estimación y complejidad.

La técnica usual utilizada en compresión de video es dividir los cuadros en bloques rectangulares no solapados, de igual tamaño, suponiendo que los píxeles dentro de un mismo bloque sufren un mismo movimiento. Además de dicha suposición, se asume que los píxeles de una imagen pueden ser modelados como una traslación de los píxeles de alguna imagen anterior.

La realidad indica claramente que no solo de traslaciones están compuestos los movimientos entre imágenes. Rotaciones, deformaciones y *zooms* escapan a la simplificación de un movimiento

exclusivamente compuesto de traslaciones, dificultando el proceso de estimación de movimiento. Sin embargo, empíricamente se observa, que para movimientos pequeños, la hipótesis de movimientos traslacionales funciona bastante bien, reduciendo en gran medida la carga operacional de los algoritmos. Otros factores que dificultan la estimación son el ruido, los cambios en la iluminación y la presencia de objetos no opacos en la escena.

Una vez modelado el movimiento es necesario encontrar un criterio de estimación que permita discernir entre los distintos movimientos posibles. Uno de los principales criterios de estimación es intentar minimizar la distorsión (definida según cierta medida) entre el bloque a codificar y un conjunto de candidatos en el área de búsqueda.

Búsqueda del vector de Movimiento

La búsqueda del vector del movimiento se basa en la estimación del movimiento local utilizando algoritmos de *block matching*. Estos algoritmos predicen el movimiento de bloques buscando en la imagen temporalmente anterior un candidato que minimice alguna medida de la distorsión. alguna de estas medidas puede ser el *MAD* (de la sigla en inglés Mean of Absolute Differences, media de la diferencias absolutas), el *SAD* (de la sigla en inglés Sum of Absolute Differences, suma de los valores absolutos de las diferencias), o el *MSE* (Mean Squared Error, error cuadrático medio).

La más popular de estas medidas es el *SAD*, debido a su simplicidad computacional y a la efectividad de sus resultados. Si dividimos la imagen en bloques de $m \times n$, la expresión para el *SAD* es la siguiente:

$$SAD = \sum_{i=0}^n \sum_{j=0}^m |I_1(x+i, y+j) - I_2(x+i, y+j)| \quad (5.1)$$

donde I_1 e I_2 son los distintos bloques a comparar. Otra medida popular es el *MAD*, que es igual a $SAD/(m \times n)$, aunque esta medida implica el uso de valores no enteros, haciéndolo ineficiente computacionalmente.

La selección del tamaño de los bloques es un compromiso entre la carga computacional y la calidad de la predicción. Experimentalmente se determinó que un bloque de 16×16 es la mejor elección, y así lo han determinado estándares internacionales de codificación de video como H.261, H.263, MPEG-1 y MPEG-2.

Una vez seleccionada una medida para la distorsión, se procede a seleccionar un método de búsqueda del vector de movimiento apropiado para la aplicación en cuestión.

5.3.2. Métodos Considerados

El procedimiento que siguen los algoritmos de búsqueda es calcular la distorsión entre el bloque a codificar y cierto conjunto de candidatos dentro de un área de búsqueda determinada y elegir el candidato que minimice dicha distorsión.

Considerando un bloque centrado en (x, y) , se define un área de búsqueda de dicho bloque como el rectángulo $p \times q$ centrado en (x, y) en el cuadro anterior, donde p y q son dados por el máximo desplazamiento permitido al bloque en las dos direcciones, vertical y horizontal, según el algoritmo de búsqueda seleccionado.

Se supone que este máximo desplazamiento es igual en ambas direcciones y se denomina d . Existen varios algoritmos de búsqueda, los más utilizados son:

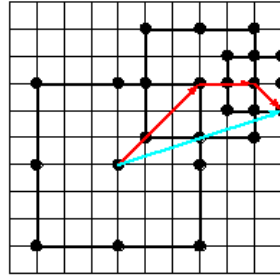


Figura 5.1: Búsqueda en tres pasos

- Búsqueda exhaustiva
- Búsqueda en tres pasos
- Búsqueda en tres pasos rápida
- Búsqueda logarítmica
- Búsqueda en la dirección conjugada
- Búsqueda en capas
- Búsqueda por el vecino más cercano

Búsqueda exhaustiva

Es la técnica más simple y también la más exacta. Realiza una comparación exhaustiva con todos los bloques que se hallan en el interior de un área de búsqueda, encontrando así los vectores de movimiento óptimos. Existen $(2d + 1)(2d + 1)$ posiciones que deben ser examinadas, por lo que, como desventaja este método tiene un alto costo computacional, lo cual hace lo hace prohibitivo en aplicaciones en tiempo real.

A modo de disminuir la carga operacional se deben desarrollar otros métodos que se presentan a continuación.

Búsqueda en tres pasos

El método de búsqueda en tres pasos es uno de los métodos más populares por su simplicidad y su alto rendimiento. En primer lugar, se elige la longitud del paso de búsqueda d igual a tres. Luego se calcula la distorsión en el centro de cada bloque de la imagen de referencia y en ocho posiciones a una distancia de un paso alrededor de su centro en forma de cuadrado, como se muestra en la figura 5.1. En cada paso la posición de menor distorsión se convierte en el centro del cuadrado del próximo paso, y se reduce el tamaño del cuadro en uno hasta el tercer paso donde se realiza a una distancia de un píxel.

Búsqueda en tres pasos rápida

El algoritmo de búsqueda en tres pasos rápida (ver figura 5.2) es una adaptación del algoritmo de búsqueda en tres pasos, donde se busca bajar el número de posiciones a chequear de forma de disminuir el costo computacional. Cada paso es dividido en dos sub-pasos. En cada sub-paso se chequea solamente en tres posiciones y dependiendo del resultado del chequeo se

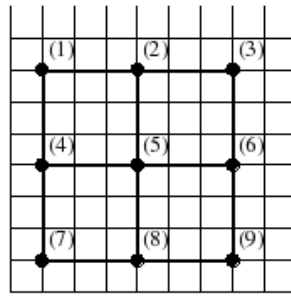


Figura 5.2: Búsqueda en tres pasos rápida

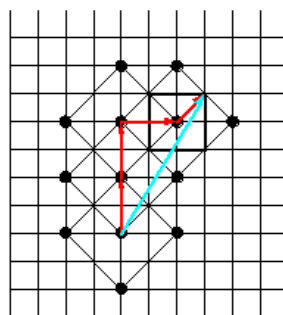


Figura 5.3: Búsqueda logarítmica

elige el sub-paso a seguir.

Los pasos a seguir en la búsqueda se detallan a continuación:

- Primer sub-paso: Calcular SAD(5), SAD(6) Y SAD(8).
- Segundo sub-paso:
 - Si SAD(6) es mínimo calcular SAD(3) y SAD(9) y se elige el mínimo entre 3,6,9.Fin.
 - Si SAD(8) es mínimo calcular SAD(7) y SAD(9) y se elige el mínimo entre 7,8,9.Fin.
 - Si SAD(5) es mínimo calcular SAD(2) y SAD(4).
 - Si SAD(2) es mínimo calcular SAD(1) y SAD(3) y se elige el mínimo entre 1,2,3. Fin.
 - Si SAD(4) es mínimo calcular SAD(1) y SAD(7) y se elige el mínimo entre 1,4,7. Fin.
 - Si SAD(5) es mínimo el mínimo es 5.Fin.

Búsqueda logarítmica

El método de búsqueda logarítmica (ver figura 5.3) es muy parecido al método de búsqueda en tres pasos. Consiste en una búsqueda distribuida en etapas donde se va reduciendo la ventana sucesivamente hasta alcanzar una distancia de un píxel. Este método aunque requiere más pasos que el método de búsqueda en tres pasos, suele proporcionar mayor precisión, especialmente cuando la ventana de búsqueda es grande.

El bloque en el centro de la región de exploración y los cuatro candidatos a un paso de distancia situados en los ejes vertical y horizontal, son comparados con el bloque actual para determinar

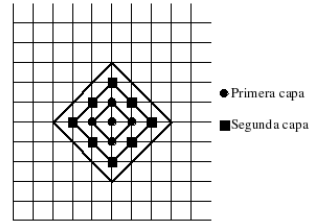


Figura 5.4: Búsqueda en capas

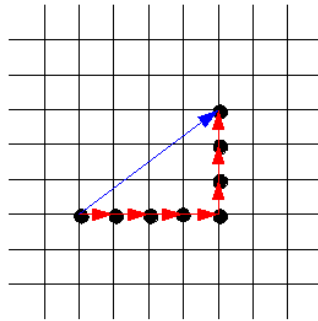


Figura 5.5: Búsqueda de dirección conjugada

la menor distorsión. Si la posición escogida es el punto del centro, la longitud del paso se reduce a la mitad, si no es así, la posición escogida es el nuevo centro y se vuelve a iniciar el proceso. Cuando la longitud del paso llega a ser 1, los nueve bloques alrededor del centro son cotejados para hallar el requerido.

Búsqueda en capas

Este método realiza la búsqueda en capas concéntricas en forma de diamante como se muestra la figura 5.4. Se calcula la distorsión en todas las posiciones de una capa, y se busca el mínimo por capa. En el paso i , se busca el mínimo de la capa i , si el mínimo es mayor que el de la capa $i-1$, se detiene la búsqueda. De no ser así se aumenta el área de la capa hasta que el mínimo de la capa i sea mayor que el mínimo de la capa $i-1$. Es decir la búsqueda se realiza desde adentro hacia afuera.

Búsqueda de dirección conjugada

Este algoritmo (ver figura 5.5) busca para cada bloque de la imagen actual la posición de menor distorsión en la dirección horizontal, luego busca la posición de menor distorsión en la dirección vertical y con estas posiciones halla el vector de movimiento.

5.3.3. Implementación

En [20] se realiza una comparación de los distintos métodos de búsqueda del vector de movimiento según la calidad de la estimación y el tiempo de búsqueda. De allí se desprende que la búsqueda en tres pasos se ajusta a las características deseadas a nuestro sistema, es decir, una buena estimación y un costo computacional relativamente bajo respecto a la búsqueda exhaustiva que es la de mayor costo computacional. El algoritmo elegido para hallar el vector de movimiento fue por este motivo la búsqueda en tres pasos.

La medida de la distorsión se realizó mediante el cálculo del SAD, dado que presenta buenos resultados a un bajo costo operacional.

El procedimiento realizado para la compensación de movimiento consistió en dividir cada cuadro en bloques de $N \times N$, estudiándose el comportamiento del sistema para $N = 16, 32, 64, 128$ en relación al tiempo requerido para el procesamiento y el nivel de compresión alcanzado.

Luego de dividida la imagen en bloques, se buscó el vector de movimiento de modo que minimice el SAD.

Una de las dificultades encontradas al buscar los vectores de movimiento es el problema de los bordes. Es decir, se debe tomar una postura frente a cómo hallar dicho vector para los píxeles que pertenecen al borde de la imagen, dado que el área de búsqueda se ve limitada. Se consideraron tres formas distintas de hallarlo: espejando la imagen, repitiendo los píxeles del borde y no teniendo en cuenta dichos píxeles para la búsqueda. El espejado de la imagen consiste en extender la imagen realizando una simetría axial respecto al borde en cuestión, mientras que la repetición de los píxeles del borde consiste en extenderla mediante la repetición del valor de los píxeles del borde a modo de permitir un área de búsqueda adecuada. Por último no considerar los píxeles del borde significa que se dividirán en bloques las áreas tales que no exista la posibilidad de generar un área de búsqueda completa.

Finalmente al medir el desempeño del sistema frente al conjunto de videos de prueba (ver 3.2.1), optamos por no tomar en cuenta los bloques con problema de borde, ya que el beneficio obtenido al tomar en cuenta dichos bloques, tanto en el espejado como en la repetición del valor del píxel, no justifica la carga computacional que esto implica.

5.3.4. Resultados Obtenidos

Recordando lo mencionado anteriormente, una vez implementado el método de los tres pasos para la compensación de movimiento, se tiene un cuadro $\hat{n}$ formado “reordenando” bloques del cuadro $n - 1$ según lo indicado por un vector de movimiento $\vec{v}_i \in V$ para cada bloque i .

Enviando estos vectores V al decodificador este puede construir el cuadro $\hat{n}$. Por lo tanto codificador y decodificador conocen los cuadros $\hat{n}$ y $n - 1, n - 2, \dots, n - q$.

En base a los requerimientos de mínima complejidad necesarios para cumplir con la restricción sobre la rapidez con que puedan ejecutarse los procesos de compresión y decompresión para un cuadro, se optó por tomar $q = 1$. Esto significa que en todo momento como máximo sólo se mantendrán tres cuadros en memoria, a saber, el cuadro actual n (a codificar), el cuadro anterior $n - 1$ y el cuadro compensado $\hat{n}$.

Ahora bien, resta aún determinar el modo de utilizar la información aportada por el cuadro $\hat{n}$ para explotar la redundancia temporal.

Un enfoque posible, siguiendo estrategias empleadas por algoritmos de compresión de video con pérdidas es utilizar el cuadro $\hat{n}$ para ser restado píxel a píxel del cuadro n . Con esto se obtiene un nuevo cuadro $n_r = n - \hat{n}$ del cual es razonable esperar píxeles caracterizados por

una distribución más angosta y alta que la de los cuadros n o $\hat{n}$.

Esto puede fundamentarse en que el cuadro n_r estará esencialmente compuesto por restas de bloques de 16×16 o 32×32 píxeles de lado muy similares entre sí; asumiendo un método para la compensación de movimiento eficiente como el implementado, y una tasa de adquisición de cuadros alta (lo que implica un muestreo temporal “fino”) como lo es 30 fps en las secuencias de prueba utilizadas. El resultado será un cuadro n_r con la mayoría de sus píxeles nulos, excepto en algunos de los bordes de los objetos que cambian de posición entre un cuadro y otro.

Sin embargo, el rango para los píxeles del cuadro n_r pasa a ser:

$$-(\alpha - 1) \leq p_r \leq (\alpha - 1) \quad (5.2)$$

donde p_r es un píxel del cuadro n_r .

Esto supone que debe realizarse algún tipo de reducción de rango y mapeo, análogamente a los realizados a nivel de píxel en LOCO-I/JPEG-LS (ver 4.1.4 y 4.1.5) con el error de predicción. Realizamos entonces la misma reducción de rango que en 4.1.4 y el mapeo $M(\cdot)$ de Rice. Llamamos al cuadro así obtenido $M(n_r)$.

Esta enfoque no resultó exitoso, ya que luego de estas etapas el cuadro $M(n_r)$ ingresa a un codificador JPEG-LS donde sus píxeles serán predichos utilizando el predictor espacial MED, cuyo desempeño no será necesariamente mejor al logrado en el cuadro n en modo JPEG-LS Cuadro a Cuadro. Además los cómputos extra requeridos se suman a la costosa compensación de movimiento teniendo como consecuencia un método prohibitivamente lento.

Alternativamente se pueden utilizar los píxeles del cuadro $\hat{n}$ junto con los del cuadro n (anteriores al píxel a codificar, de acuerdo a la causalidad necesaria) para elaborar una predicción del píxel actual siguiendo un algoritmo determinado. Este enfoque seguido en M-CALIC [14] y LOPT-3D [15] es descartado en base a la tabla 3.4. En esta se muestra que, en media, estos métodos alcanzan tan solo 5.06 y 1.74 fps respectivamente, debido al alto costo computacional de dichos algoritmos.

Surge entonces la alternativa, algo rudimentaria, de utilizar como predicción del píxel a codificar, el píxel de iguales coordenadas en el cuadro $\hat{n}$, remplazando al predictor MED. De este modo no es necesario realizar reducciones de rango ni mapeos. Llamaremos a este método JPEG-LS con Compensación de Movimiento, o MoJPEG-LS de forma abreviada.

El resultado de MoJPEG-LS aplicado al conjunto de videos de prueba utilizando el método de compensación de movimiento de los tres pasos se muestra en las tablas 5.8, 5.9 para tamaño de bloque de $L \times L$ con $L = 16$ y distancia de paso $D = 3$. En las tablas 5.10, 5.11, 5.12, 5.13, 5.14, 5.15 se repite el mismo experimento variando solo los tamaños de bloque a 32×32 , 64×64 y 128×128 respectivamente, como se indica al pie de las mismas.

Para facilitar la interpretación de las tablas anteriores resumimos el nivel de compresión medio en % y la cantidad media de cuadros procesados por segundo en la tabla 5.16.

De ella se desprenden las siguientes observaciones:

- El nivel de compresión más alto se obtiene para el tamaño de bloque menor y es decreciente para tamaños de bloque mayores. Esto es completamente previsible, ya que con un tamaño de bloque menor se pueden reproducir más fielmente los movimientos de los objetos, obteniendo así cuadros $\hat{n}$ más similares (por ejemplo, en términos de SAD) a los n , codificando así errores de predicción con distribuciones de probabilidad más delgadas y altas.
- La mayor cantidad de cuadros procesados por segundo ocurre para el tamaño de bloque

Secuencia	Nivel de Compresión en bpp	Tiempo de Procesamiento en segundos
Salesman	4.2357	232.55
Mobile y Calendar	5.4011	154.86
Container	3.3134	153.22
Tempete	4.9788	133.62
Kitchgrass	4.4883	51.87
Sean	3.2234	154.52
Silent	3.6019	154.96
Foreman	4.5837	154.41
Promedio	4.2283	148.75

Tabla 5.8: Nivel de compresión en bpp y tiempo de procesamiento para MoJPEG-LS con $L = 16$

Secuencia	Nivel de compresión en %	Cuadros procesados por segundo
Salesman	47.05	1.93
Mobile y Calendar	32.49	1.94
Container	58.58	1.96
Tempete	37.76	1.95
Kitchgrass	43.90	1.93
Sean	59.71	1.94
Silent	54.98	1.94
Foreman	42.70	1.94
Promedio	47.15	1.94

Tabla 5.9: Nivel de compresión en % y cuadros procesados por segundo para MoJPEG-LS con $L = 16$

Secuencia	Nivel de Compresión en bpp	Tiempo de Procesamiento en segundos
Salesman	4.3330	71.40
Mobile y Calendar	5.8670	48.72
Container	3.6309	47.95
Tempete	5.0330	41.61
Kitchgrass	4.6172	15.98
Sean	3.5235	47.76
Silent	3.7257	47.50
Foreman	4.7041	48.05
Promedio	4.4293	46.12

Tabla 5.10: Nivel de compresión en bpp y tiempo de procesamiento para MoJPEG-LS con $L = 32$

Secuencia	Nivel de compresión en %	Cuadros procesados por segundo
Salesman	45.84	6.29
Mobile y Calendar	26.66	6.16
Container	54.61	6.26
Tempete	37.09	6.25
Kitchgrass	42.28	6.26
Sean	55.96	6.28
Silent	53.43	6.32
Foreman	41.20	6.24
Promedio	44.63	6.26

Tabla 5.11: Nivel de compresión en % y cuadros procesados por segundo para MoJPEG-LS con $L = 32$

Secuencia	Nivel de Compresión en bpp	Tiempo de Procesamiento en segundos
Salesman	4.3651	37.70
Mobile y Calendar	5.7947	26.18
Container	3.3204	24.89
Tempete	5.1071	22.23
Kitchgrass	4.8970	8.480
Sean	3.2823	24.92
Silent	3.8702	25.22
Foreman	5.1040	25.60
Promedio	4.4676	24.40

Tabla 5.12: Nivel de compresión en bpp y tiempo de procesamiento para MoJPEG-LS con $L = 64$

Secuencia	Nivel de compresión en %	Cuadros procesados por segundo
Salesman	45.44	11.91
Mobile y Calendar	27.57	11.46
Container	58.49	12.05
Tempete	36.16	11.70
Kitchgrass	38.79	11.79
Sean	58.97	12.04
Silent	51.62	11.90
Foreman	36.20	11.72
Promedio	44.16	11.83

Tabla 5.13: Nivel de compresión en % y cuadros procesados por segundo para MoJPEG-LS con $L = 64$

Secuencia	Nivel de Compresión en bpp	Tiempo de Procesamiento en segundos
Salesman	4.3631	26.56
Mobile y Calendar	5.7954	18.68
Container	3.3231	17.49
Tempete	5.1064	16.03
Kitchgrass	4.8970	5.98
Sean	3.2805	17.56
Silent	3.8711	17.64
Foreman	5.1049	18.18
Promedio	4.4677	17.27

Tabla 5.14: Nivel de compresión en bpp y tiempo de procesamiento para MoJPEG-LS con $L = 128$

Secuencia	Nivel de compresión en %	Cuadros procesados por segundo
Salesman	45.46	16.91
Mobile y Calendar	27.56	16.06
Container	58.46	17.15
Tempete	36.17	16.22
Kitchgrass	38.79	16.72
Sean	58.99	17.08
Silent	51.61	17.01
Foreman	36.19	16.50
Promedio	44.15	16.72

Tabla 5.15: Nivel de compresión en % y cuadros procesados por segundo para MoJPEG-LS con $L = 128$

Tamaño de bloque	Nivel de compresión en %	Cuadros procesados por segundo
$L = 16$	47.15	1.94
$L = 32$	44.63	6.26
$L = 64$	44.16	11.83
$L = 128$	44.15	16.72

Tabla 5.16: Resumen de performance para $L = 16$, $L = 32$, $L = 64$ y $L = 128$. Tanto el nivel de compresión como la cantidad de cuadros procesados por segundo corresponden a valores medios

Secuencia	Nivel de Compresión en bpp	Tiempo de Procesamiento en segundos
Salesman	4.3631	12.14
Mobile y Calendar	5.7933	9.14
Container	3.3183	7.94
Tempete	5.1064	7.60
Kitchgrass	4.8970	2.90
Sean	3.2805	7.92
Silent	3.8694	8.05
Foreman	5.1040	8.78
Promedio	4.4665	8.06

Tabla 5.17: Nivel de compresión en bpp y tiempo de procesamiento para AnJPEG-LS

mayor y es decreciente para tamaños de bloques menores. Nuevamente, esto no es una sorpresa, ya que si los bloques son de tamaño menor más veces por cuadro deberán calcularse vectores de movimiento, con el costo computacional asociado a esta tarea.

- Aún el mejor valor de cantidad de cuadros procesados por segundo es peor al obtenido en media utilizando JPEG-LS cuadro a cuadro con 32.69 fps.
- Aún el peor valor de compresión alcanzado es mejor al obtenido en media utilizando JPEG-LS cuadro a cuadro con un 41.81 % de nivel de compresión.

Las conclusiones que se obtienen no son muy positivas. Siempre en comparación con JPEG-LS cuadro a cuadro, una mejora de un 5.97 % en el nivel de compresión se obtiene a costas de una reducción de cuadros procesados por segundo del 94 % aproximadamente. Procesando un 50 % menos de cuadros por segundo, se obtiene un pobre 2.34 % de mejora en el nivel de compresión. Ninguna opción resulta tentadora.

Si bien la baja cantidad de cuadros procesados por segundo puede deberse a una *mala* implementación del método de compensación de movimiento de los tres pasos iterativa y no recursiva, creemos que de todos modos la mejora en los niveles de compresión no justifica el gasto en este caso.

A pesar de esto, es notable que con una predicción tan simple se logren resultados mejores que JPEG-LS cuadro a cuadro, teniendo en cuenta que el contexto del píxel a codificar sigue siendo calculado a partir de las diferencias $D1 = d - b$, $D2 = b - c$ y $D3 = c - a$ las cuales tienen en cuenta características solo espaciales. Es evidente que si en la etapa de predicción es modificada en forma definitiva para tener en cuenta otra información además de la espacial, el cálculo de contexto debe reflejar este cambio.

Antes de abordar este punto, y a modo de referencia o comparación aplicaremos el método anterior al conjunto de los videos de prueba con una variante. Se tomará como predicción del píxel a codificar el píxel de iguales coordenadas pero del cuadro $n - 1$ y no del cuadro $\hat{n}$. O sea, el predictor MED es substituido por el píxel de iguales coordenadas del cuadro anterior y la compensación de movimiento no se utiliza. Llamaremos a este método JPEG-LS con Cuadro Anterior, o AnJPEG-LS en forma abreviada. Los resultados se presentan en las tablas 5.17 y 5.18.

En ellas se puede observar que si bien el nivel medio de compresión es de 44.17 % la tasa media de cuadros procesados por segundo es de 35.82 fps. Esto significa un nivel de compresión apenas mayor al alcanzado por MoJPEG-LS con $L = 64$ pero procesando aproximadamente el triple de cuadros por segundo que en este caso.

Secuencia	Nivel de compresión en %	Cuadros procesados por segundo
Salesman	45.46	36.99
Mobile y Calendar	27.58	32.82
Container	58.52	37.78
Tempete	36.17	34.21
Kitchgrass	38.79	34.48
Sean	58.99	37.88
Silent	51.63	37.27
Foreman	36.20	34.17
Promedio	44.17	35.82

Tabla 5.18: Nivel de compresión en % y cuadros procesados por segundo para AnJPEG-LS

La comparación con JPEG-LS cuadro a cuadro (usando la tabla 5.2) y MeJPEG-LS (usando la tabla 5.4) arroja los siguientes resultados:

- Comprime en media un 2.36 % más que JPEG-LS Cuadro a Cuadro procesando en media 3.13 fps más por segundo.
- Comprime en media un 2.18 % más que MeJPEG-LS procesando en media 2.67 fps más por segundo.

Vuelve a llamar la atención el alto nivel de compresión alcanzado, para lo simple de la predicción y el incompleto cálculo del contexto. Este resultado indica que hay mucho “por explotar” de la redundancia temporal presente en las secuencias de video.

Si bien los resultados de las evaluaciones para AnJPEG-LS parecen ser buenos, hay que recordar que prácticamente nada de la redundancia espacial se está explotando. Es de esperar que estos resultados estén “inflados” por la alta resolución temporal de las secuencias de videos de prueba. Es decir, si la tasa de adquisición baja de 30 fps por algún motivo, el píxel de iguales coordenadas del cuadro anterior ya no constituye una buena predicción del píxel a codificar en las regiones de la imagen donde hay objetos cambiando de posición entre cuadros. En definitiva esto es equivalente a decir que ante secuencias de video con menor redundancia temporal, el método no será adecuado.

Podemos “simular” esta situación submuestreando temporalmente las secuencias de video de pruebas. A modo de ejemplo tomaremos para cada secuencia un cuadro de cada dos, por ejemplo solo los impares, formando así secuencias de video de longitud mitad a la original. Nos referiremos a esta situación como $SALTO = 2$. Repetiremos el mismo procedimiento pero esta vez tomando uno de cada tres cuadros, por ejemplo solo los múltiplos de tres, formando una nueva secuencia de video con un tercio del largo a la original. Analogamente identificaremos esta situación por $SALTO = 3$.

Los resultados de aplicar AnJPEG-LS para las secuencias de videos de pruebas con $SALTO = 2$ y $SALTO = 3$ se muestran en las tablas 5.19 y 5.20 respectivamente.

Como era de esperarse el nivel de compresión alcanzado por AnJPEG-LS cae para $SALTO = 2$ y vuelve a caer para $SALTO = 3$. Específicamente:

- Para $SALTO = 2$ el nivel de compresión cae un 4.58 %, respecto al nivel alcanzado por AnJPEG-LS sin submuestreo temporal.

Secuencia	Nivel de Compresión en bpp	Tiempo de Procesamiento en segundos
Salesman	3.6256	6.11
Mobile y Calendar	6.4644	4.65
Container	3.7463	4.05
Tempete	5.6717	3.88
Kitchgrass	5.5739	1.46
Sean	3.5126	4.02
Silent	4.3433	4.13
Foreman	5.7261	4.56
Promedio	4.8330	4.11

Tabla 5.19: Nivel de compresión en bpp y tiempo de procesamiento para AnJPEG-LS con secuencias de video submuestreadas temporalmente con $SALTO = 2$

Secuencia	Nivel de compresión en %	Cuadros procesados por segundo
Salesman	54.68	36.66
Mobile y Calendar	19.20	32.26
Container	53.17	37.04
Tempete	29.10	33.51
Kitchgrass	30.33	34.25
Sean	56.09	37.31
Silent	45.71	36.32
Foreman	28.42	32.89
Promedio	39.59	35.12

Tabla 5.20: Nivel de compresión en % y cuadros procesados por segundo para AnJPEG-LS con secuencias de video submuestreadas temporalmente con $SALTO = 2$

Secuencia	Nivel de Compresión en bpp	Tiempo de Procesamiento en segundos
Salesman	4.6714	4.12
Mobile y Calendar	6.7840	3.14
Container	4.0304	2.75
Tempete	6.0543	2.61
Kitchgrass	5.9419	0.98
Sean	3.6671	2.67
Silent	4.6074	2.77
Foreman	6.0698	2.97
Promedio	5.2283	2.75

Tabla 5.21: Nivel de compresión en bpp y tiempo de procesamiento para AnJPEG-LS con secuencias de video submuestreadas temporalmente con $SALTO = 3$

Secuencia	Nivel de compresión en %	Cuadros procesados por segundo
Salesman	41.61	36.17
Mobile y Calendar	15.20	31.85
Container	49.62	36.36
Tempete	24.32	32.95
Kitchgrass	25.73	33.67
Sean	54.16	37.45
Silent	42.41	36.10
Foreman	24.13	33.67
Promedio	34.65	34.89

Tabla 5.22: Nivel de compresión en % y cuadros procesados por segundo para AnJPEG-LS con secuencias de video submuestreadas temporalmente con $SALTO = 3$

Algoritmo	Nivel de compresión en %	Cuadros procesados por segundo
JPEG-LS Cuadro a Cuadro	41.81	32.69
MeJPEG-LS	41.99	33.15
MoJPEG-LS con $L = 16$	47.15	1.94
AnJPEG-LS con $S = 0$	44.17	35.82
AnJPEG-LS con $S = 3$	34.65	34.89

Tabla 5.23: Resumen de performance para los algoritmos propuestos. Tanto el nivel de compresión como la cantidad de cuadros procesados por segundo corresponden a valores medios

- Para $SALTO = 3$ el nivel de compresión cae un 9.52 %, respecto al nivel alcanzado por AnJPEG-LS sin submuestreo temporal.

En conclusión, AnJPEG-LS pone de manifiesto por una lado, la *posibilidad* de alcanzar buenos niveles de compresión utilizando píxeles del cuadro anterior en la etapa de predicción *sin* utilizar la costosa compensación de movimiento y por otro la *necesidad* de que esta predicción contemple la redundancia espacial para hacer frente a secuencias de video con baja redundancia temporal.

Además aún resta por contemplar la información introducida del cuadro anterior en el cálculo del contexto del píxel a codificar.

Con base en estas tres premisas, se presenta en la subsección siguiente un método para la compresión de video sin pérdidas que busca cumplir con las restricciones planteadas para el algoritmo del sistema a implementar.

5.4. LOVICO

Se resumen en la tabla 5.23 los resultados medios obtenidos hasta ahora para nivel de compresión en % y cuadros procesados por segundo en fps por los algoritmos propuestos.

Del mismo modo las conclusiones más importantes a las que hemos llegado hasta este punto pueden resumirse así:

- Es necesario modificar JPEG-LS para que utilice información de cuadros anteriores, para

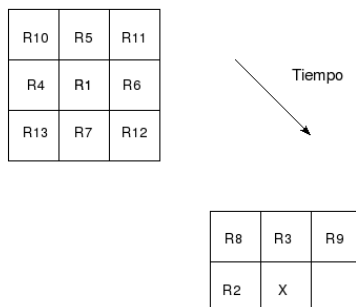


Figura 5.6: Molde causal “espacio-temporal”

explotar de ese modo la redundancia temporal. Sin embargo el mantener vectores C , B , N , A y Nn comunes a todos los cuadros no es suficiente.

- Si bien la compensación de movimiento es la forma usual de encarar el ítem anterior, la implementación del método a la que se llegó conduce a un procedimiento prohibitivamente lento.
- La redundancia temporal puede ser explotada si se utilizan adecuadamente los píxeles del cuadro anterior en la etapa de predicción, pero de tal modo que el método no falle para secuencias de video de baja resolución temporal.
- En forma paralela al punto anterior, de introducir píxeles del cuadro anterior en la etapa de predicción, el contexto del píxel a codificar debe pasar a tener componentes temporales.

Luego de considerar los puntos anteriores, muchas ideas surgieron y muchos métodos fueron probados, todos ellos orientados a buscar un buen predictor *espacio-temporal* en base a moldes causales análogos al de la figura 5.6, es decir, que contemplan píxeles anteriores del cuadro actual y píxeles del cuadro anterior.

Las propuestas fueron desde filtros lineales cuyos coeficientes eran entrenados adaptativamente (utilizando píxeles de moldes causales y algoritmos con buena performance en ambientes no estacionarios como LMS [21, sección 9.2.2]) hasta largas tablas lógicas que buscaban elegir la predicción más razonable en función de que se cumpliesen o no ciertas relaciones entre diferencias espaciales y temporales.

Como resultado de estos experimentos y del estudio detallado del algoritmo LOCO-I/JPEG-LS, llegamos a la conclusión de que no es trivial introducir elementos *ajenos* al mismo para explotar la redundancia temporal. Existe en LOCO-I/JPEG-LS una delicada relación entre la etapa de predicción, la de codificación y la de cálculo del contexto que se vincula a las dos primeras en forma estrecha. Cada paso en LOCO-I/JPEG-LS está cuidadosamente fundamentado con un espíritu de baja complejidad que afecta prácticamente todas las decisiones tomadas.

Una forma simple y razonable de encarar esta situación es intentar utilizar *elementos propios* de LOCO-I/JPEG-LS organizados con ligeras variantes para tratar de explotar la redundancia temporal presente en las secuencias de video.

Como veremos en lo que sigue esta premisa condujo al algoritmo que denominamos LOVICO, por la sigla en inglés *LOssless Video COding*. El mismo aplica modificaciones en las etapas de predicción y de cálculo del contexto a LOCO-I/JPEG-LS, las cuales se muestran en los puntos

siguientes.

5.4.1. Modificación del Predictor

Expresamos aquí nuevamente las opciones entre las cual conmuta el predictor MED de JPEG-LS:

- El valor b es seleccionado como predicción cuando un borde vertical existe a la izquierda de la muestra actual.
- El valor a es seleccionado como predicción cuando un borde horizontal existe sobre la muestra actual.
- El valor $a + b - c$ es seleccionado como predicción en otro caso.

A partir de este momento comenzaremos a notar la posición de los píxeles (o el valor de los mismos, haciendo nuevamente un abuso de notación) como se muestra en la figura 5.6. Esta es la forma utilizada en [15], reproducida en la sección 3.1.4 en base a las distancias 2D y 3D que allí se presentan. De este modo los píxeles a , b , c y d del cuadro actual (cuadro a codificar, o cuadro n) pasan a ser denominados píxeles R_2 , R_3 , R_8 y R_9 respectivamente.

Una observación trivial es que el predictor MED utiliza solo tres píxeles para elaborar la predicción (la cual posteriormente es corregida, en función del contexto). Otra, es que trabaja sobre imágenes, es decir sobre los cuadros que componen el video detectando bordes o regiones uniformes. Además, en un cuadro son necesarias dos coordenadas para individualizar a un píxel, filas y columnas mientras que en un video se agrega una tercera coordenada, el tiempo.

Ahora bien, un cuadro esta compuesto por los píxeles del video que tienen un valor temporal común, mientras que filas y columnas pueden tomar cualquier valor dentro de un rango determinado por el tamaño del cuadro. La agrupación de estos píxeles presenta las características de bordes horizontales o verticales y regiones “planas” que el predictor MED explota para elaborar una buena predicción.

Surge entonces la pregunta: ¿Qué ocurre con la agrupación de píxeles del video que tienen un valor común de columnas (respectivamente filas) y cualquier valor de tiempo y filas (respectivamente tiempo y columnas) dentro del rango dado por el tamaño del cuadro y el tiempo de adquisición?.

Las “imágenes” que se muestran en las figuras 5.7 a 5.18 fueron elaboradas de tal modo, manteniendo constantes las filas (respectivamente columnas) durante todo el video y almacenando los valores de píxeles según su tiempo y columnas (respectivamente filas).

Observando las figuras 5.7 a 5.18 vemos que de hecho estas “imágenes espacio-temporales” poseen muchas de las características que se presentan en imágenes puramente espaciales y que el predictor MED explota para elaborar su predicción, más aún considerando una escala de tres píxeles.

Entonces, el predictor MED puede “lanzarse” sobre las coordenadas *tiempo-filas* y *tiempo-columnas* además de las coordenadas *filas-columnas* utilizadas por LOCO-I/JPEG-LS. Esto se presenta en la figura 5.19. En ella se muestran los tres píxeles con que se elabora la predicción en cada caso y el píxel a codificar x . Además se marcan en color gris los píxeles utilizados en el predictor MED que pertenecen al cuadro anterior.

Denotemos por $MED(\alpha, \beta, \gamma)$ a:

$$MED(\alpha, \beta, \gamma) = \text{mediana } [\alpha, \beta, \alpha + \beta - \gamma] \quad (5.3)$$



Figura 5.7: Píxeles de la secuencia Mobile y Calendar con columna fija igual a un cuarto del total



Figura 5.8: Píxeles de la secuencia Mobile y Calendar con columna fija igual a un medio del total



Figura 5.9: Píxeles de la secuencia Mobile y Calendar con fila fija igual a un cuarto del total

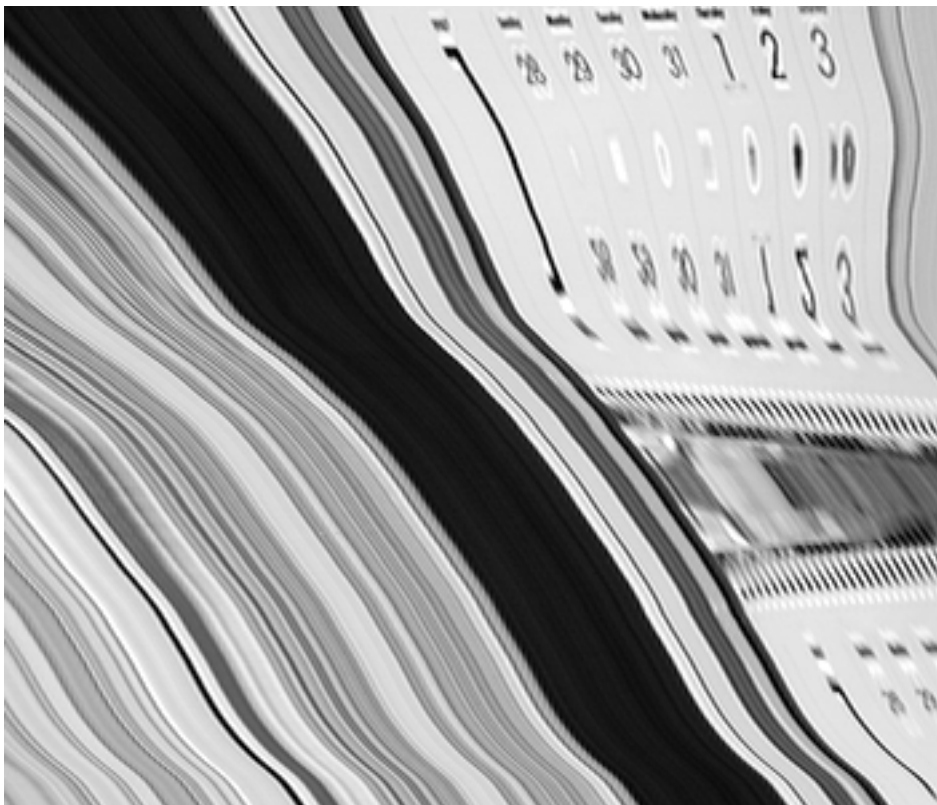


Figura 5.10: Píxeles de la secuencia Mobile y Calendar con fila fija igual a un medio del total



Figura 5.11: Píxeles de la secuencia Container con columna fija igual a un cuarto del total

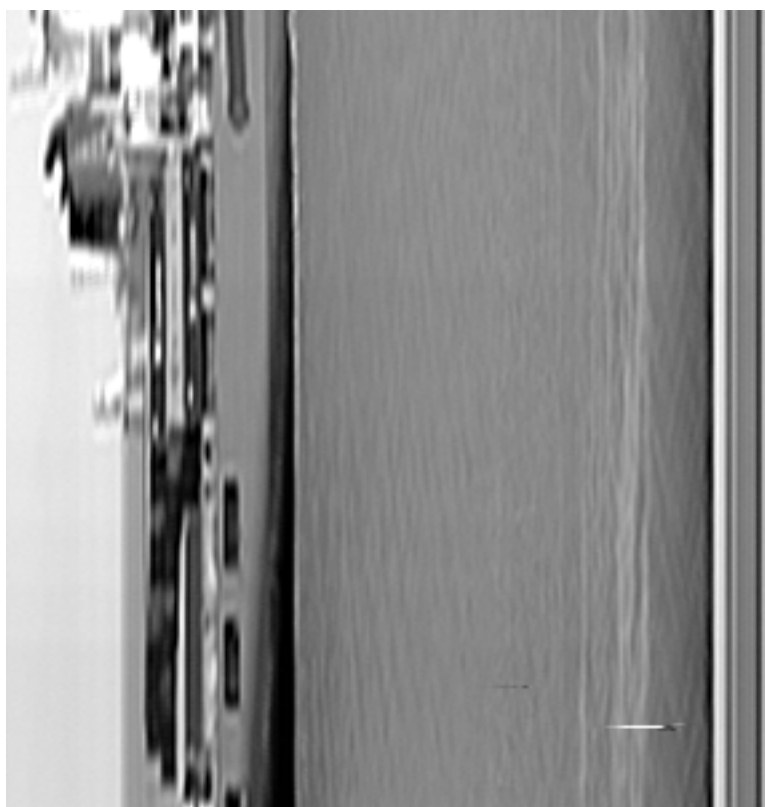


Figura 5.12: Píxeles de la secuencia Container con columna fija igual a un medio del total

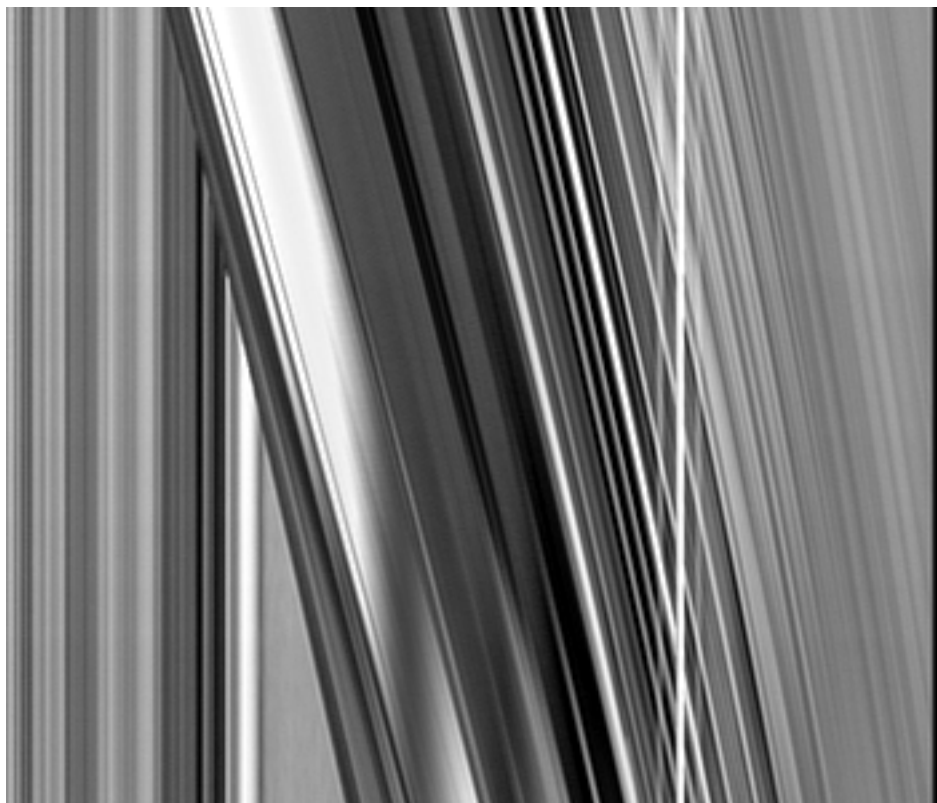


Figura 5.13: Píxeles de la secuencia Container con fila fija igual a un cuarto del total

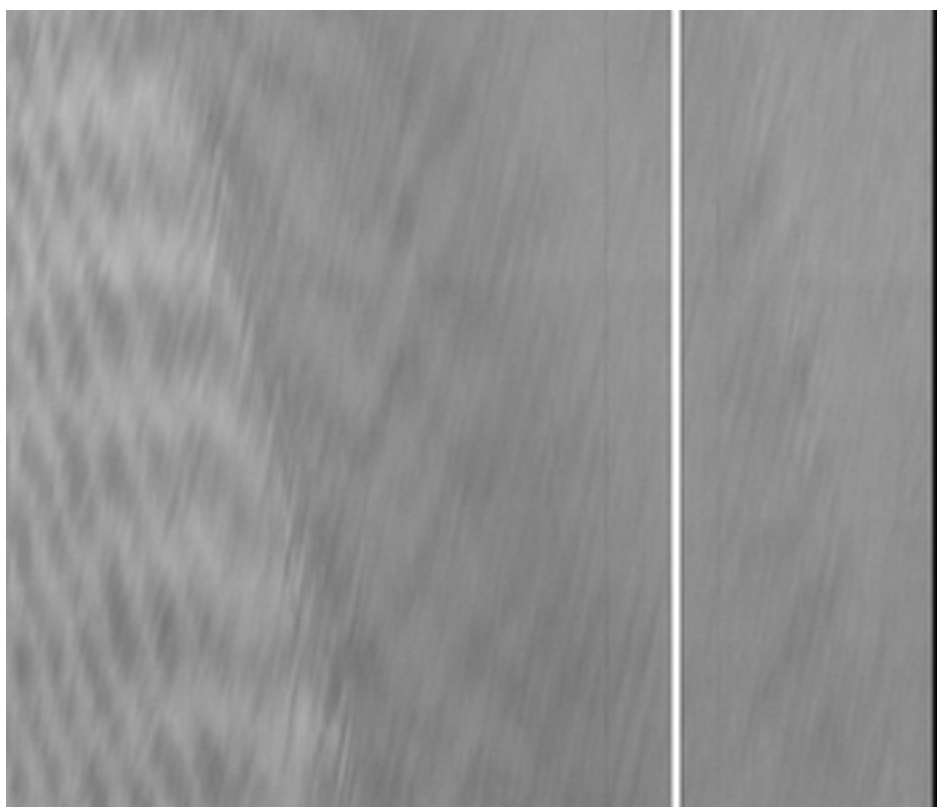


Figura 5.14: Píxeles de la secuencia Container con fila fija igual a un medio del total

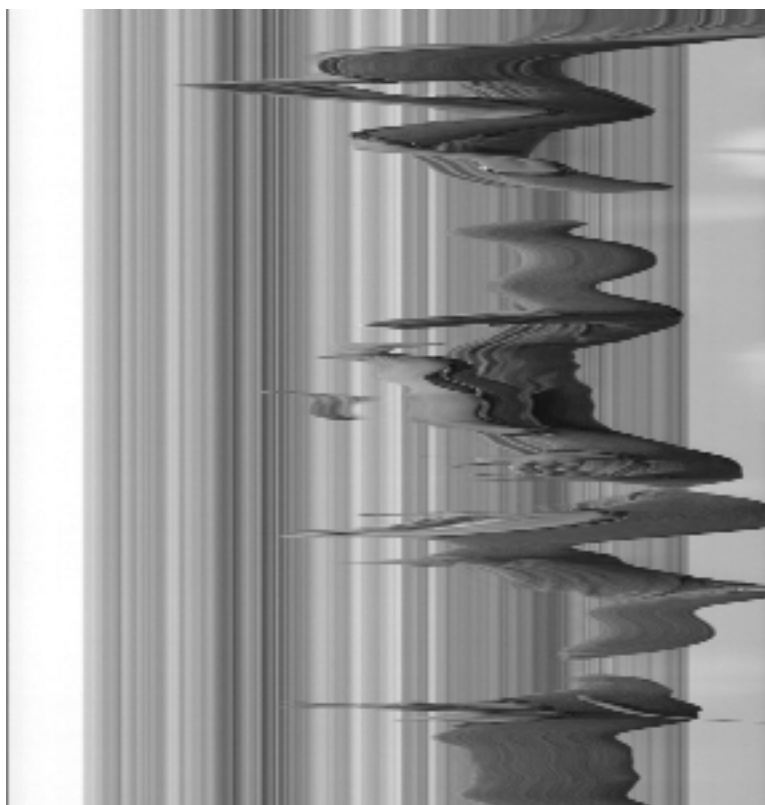


Figura 5.15: Píxeles de la secuencia Silent con columna fija igual a un cuarto del total



Figura 5.16: Píxeles de la secuencia Silent con columna fija igual a un medio del total

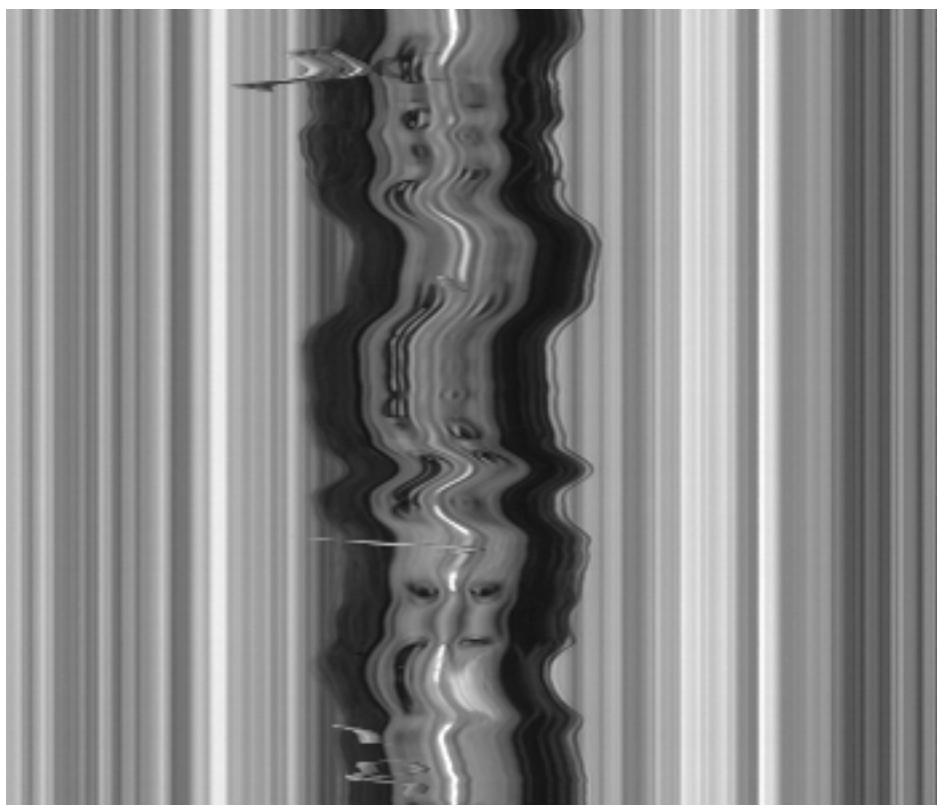


Figura 5.17: Píxeles de la secuencia Silent con fila fija igual a un cuarto del total

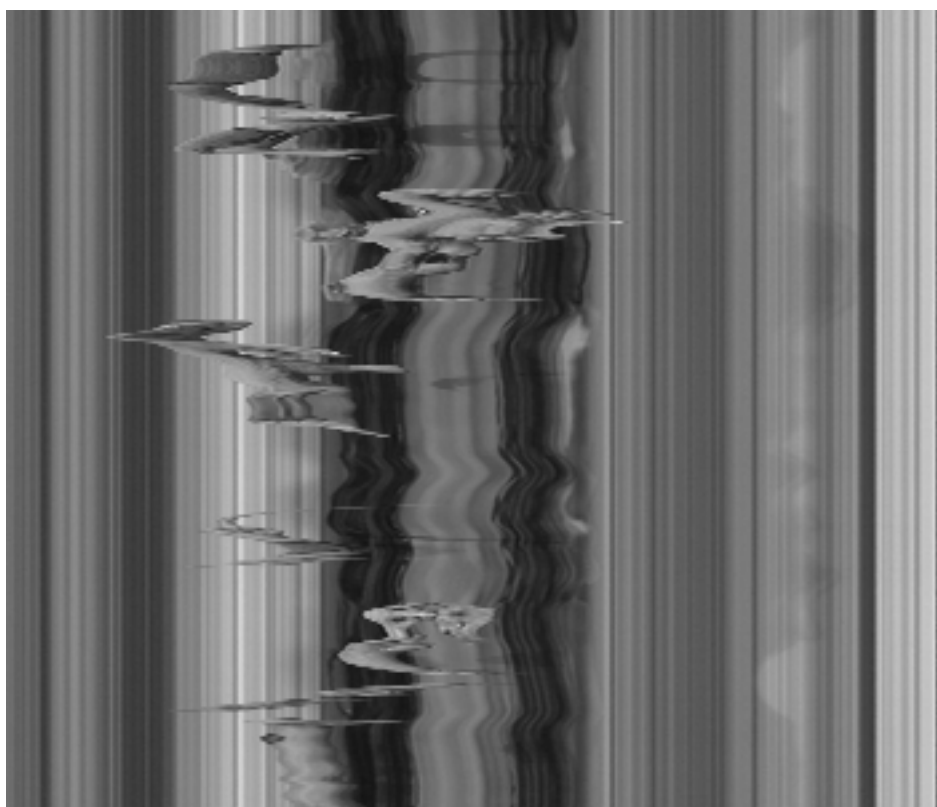


Figura 5.18: Píxeles de la secuencia Silent con fila fija igual a un medio del total

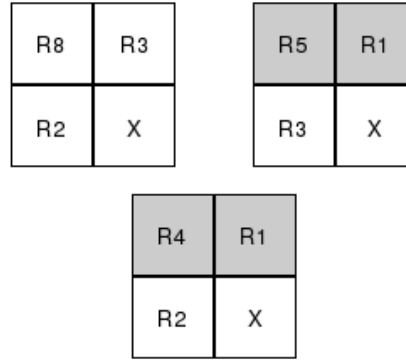


Figura 5.19: Predictores MED utilizado por LOVICO

Es evidente que si $\alpha = R_2$, $\beta = R_3$, y $\gamma = R_8$ $MED(\alpha, \beta, \gamma)$ representa el valor que arroja el predictor MED utilizado por LOCO-I/JPEG-LS. Así, las predicciones MED sobre las coordenadas *tiempo-filas*, *tiempo-columnas* y *filas-columnas* se elaboran del siguiente modo:

- La predicción sobre coordenadas *filas-columnas* corresponde a $MED(R_2, R_3, R_8)$ (superior izquierdo en figura 5.19).
- La predicción sobre coordenadas *tiempo-columnas* corresponde a $MED(R_2, R_1, R_4)$ (inferior en figura 5.19).
- La predicción sobre coordenadas *tiempo-filas* corresponde a $MED(R_3, R_1, R_5)$ (superior derecho en figura 5.19).

Obtenemos de este modo tres predicciones del píxel a codificar, por lo que resta un procedimiento para seleccionar una de ellas. Naturalmente, lo que se busca es elegir la predicción más acertada. Una forma de aproximarse a la misma es utilizar la predicción que involucra a las dos coordenadas sobre las cuales el video sufre menos cambios locales entre cuadros adyacentes. LOCO-I/JPEG-LS determina si es esperable una racha de píxeles iguales o por otro lado un borde (horizontal o vertical) o una región intermedia en base a las diferencias 4.1 con la notación actual para los píxeles, $D1 = R_9 - R_3$, $D2 = R_3 - R_8$ y $D3 = R_8 - R_2$. Podemos adoptar una estrategia análoga utilizando el valor absoluto de dos de estas diferencias locales para asociarlas a la magnitud del cambio sufrido localmente por el video en cada coordenada.

En tal sentido, la diferencia $D2$ aportará información sobre el cambio en la dirección horizontal o entre columnas, mientras que la diferencia $D3$ lo hará en lo que respecta a cambios en la dirección vertical o entre filas. Resta aún incorporar información sobre el cambio en la dirección temporal. Seleccionamos para este fin la diferencia entre los píxeles R_2 y R_4 . De este modo la selección de la predicción a utilizar se basa en las magnitudes g_z , g_x y g_y definidas como:

$$g_z = |R_2 - R_4|, \quad g_x = |R_3 - R_8|, \quad g_y = |R_8 - R_2| \quad (5.4)$$

Una vez que se tienen estas cantidades puede elaborarse el esquema de decisiones. Por ejemplo, si la diferencia local entre cuadros (dada por g_z) es mayor a la presente entre filas y columnas del mismo cuadro (dadas por g_y y g_x respectivamente) la predicción seleccionada ha de ser $MED(R_2, R_3, R_8)$, es decir la que utiliza LOCO-I/JPEG-LS. Si pasamos la condición anterior de “mayor” a “mayor o igual”, podría esperarse la interesante propiedad de obtener una predicción al menos tan buena como la que utiliza JPEG-LS cuadro a cuadro, ya que en caso de $g_z = \max(g_x, g_y)$ se seleccionará el predictor puramente espacial. La misma puede volverse

sumamente útil en el caso videos con baja redundancia temporal.

En caso de que la diferencia local entre cuadros no sea la mayor de las tres consideradas, aún podemos elegir la predicción sobre las dos coordenadas restantes con más regularidad local (*tiempo-columnas* o *tiempo-filas*).

Estas decisiones pueden implementarse en forma sencilla como sigue:

- Si $g_z \geq \max(g_x, g_y)$ utilizar $MED(R_2, R_3, R_8)$
- En caso contrario:
 - Si $g_x \geq \max(g_y, g_z)$ utilizar $MED(R_3, R_1, R_5)$
 - En caso contrario utilizar $MED(R_2, R_1, R_4)$

5.4.2. Modificación del Cálculo del Contexto

Hasta ahora en este documento los cambios propuestos para JPEG-LS en la codificación del cuadro n solo se han orientado a modificar la etapa de predicción incluyendo información aportada por píxeles del cuadro anterior $n - 1$ o del cuadro obtenido mediante compensación de movimiento $\hat{n}$. Como se mencionó anteriormente estas modificaciones son incompletas, ya que la codificación del error de predicción corregido depende del contexto en el cual se encuentra el píxel a codificar.

Repasando lo visto en 4.2 LOCO-I/JPEG-LS calcula el contexto del siguiente modo:

- Se cuantizan las diferencias $D1$, $D2$ y $D3$ obteniendo $Q1$, $Q2$ y $Q3$.
- Si el primer elemento no nulo de la terna $(Q1, Q2, Q3)$ es negativo se invierten los signos de los elementos de la misma, obteniéndose $(-Q1, -Q2, -Q3)$ y se da a la variable $SIGN$ en valor -1 . En caso contrario no se afectan los elementos de la terna y la variable $SIGN$ se hace igual a 1 .
- Se mapea la terna $(Q1, Q2, Q3)$ en uno de los $((2T + 1)^3 + 1)/2$ valores posibles de Q , en JPEG-LS 365 para $T = 4$ (ver sección 4.1.2). Este mapeo no es especificado por [6], como se vió en 4.2 una forma de realizarlo es mediante $Q = 81 \times Q1 + 9 \times Q2 + Q3$.

El hecho de que LOCO-I/JPEG-LS realice la cuantización a:

$$\{0\}, \pm\{1, 2\}, \pm\{3, 4, 5, 6\}, \pm\{7, 8, \dots, 20\}, \pm\{e | e \geq 21\}$$

con $T = 4$ (para un alfabeto de 8 bits por muestra), y utilice tres diferencias ($D1$, $D2$ y $D3$) para el cálculo del contexto se explica en que estos valores fueron determinados como óptimos para imágenes en el compromiso *requerimientos de memoria - nivel de compresión*. Se afirma en [5] que de hecho más contextos pueden ser utilizados para imágenes de grandes tamaños, pero la mejora en el nivel de compresión alcanzada es marginal.

Entonces, una forma de modificar este cálculo del contexto para incluir información temporal es simplemente agregar las diferencias temporales dadas por 5.5.

$$D4 = R_2 - R_4, D5 = R_3 - R_5; \quad (5.5)$$

Si utilizamos $T = 4$ obtenemos un total de $((2T + 1)^3 + 1)/2$ contextos diferentes, o sea 29525. El procedimiento del cálculo del contexto se realiza en forma análoga, esta vez para el quinteto $(Q1, Q2, Q3, Q4, Q5)$ utilizando el mapeo dado por 5.6.

$$Q = 6561 \times Q1 + 729 \times Q2 + 81 \times Q3 + 9 \times Q4 + Q5 \quad (5.6)$$

Secuencia	Nivel de Compresión en bpp	Tiempo de Procesamiento en segundos
Salesman	3.98	20.25
Mobile y Calendar	5.38	14.48
Container	3.21	13.40
Tempete	4.52	12.27
Kitchgrass	4.43	4.63
Sean	2.99	13.46
Silent	3.14	13.08
Foreman	4.09	13.67
Promedio	3.97	13.16

Tabla 5.24: Nivel de compresión en bpp y tiempo de procesamiento para LOVICO aplicado al conjunto de videos de prueba

Secuencia	Nivel de compresión en %	Cuadros procesados por segundo
Salesman	50.30	22.17
Mobile y Calendar	32.71	20.72
Container	59.84	22.39
Tempete	43.52	21.19
Kitchgrass	44.66	21.60
Sean	62.64	22.29
Silent	60.73	22.94
Foreman	48.87	21.95
Promedio	50.41	21.94

Tabla 5.25: Nivel de compresión en % y cuadros procesados por segundo para LOVICO

El resultado de aplicar LOVICO al conjunto de los videos de prueba con $T = 4$ y cinco diferencias se muestra en las tablas 5.24 y 5.25.

Resta aún por verificar que LOVICO, a diferencia de AnJPEG-LS puede tener una buena performance en secuencias de video con menor redundancia temporal a la presente en los videos del conjunto de prueba. Una forma de simple de estudiar su comportamiento en estas situaciones es aplicar LOVICO al conjunto de videos de prueba submuestreados temporalmente para $SALTO = 3$ (ver sección 5.3.4).

El hecho notable aquí es que, si bien el nivel de compresión cae en media un 4.11 % respecto a LOVICO aplicado al conjunto de videos de prueba sin submuestreo temporal (lo que es totalmente razonable, ya que ha desaparecido parte de la redundancia temporal), el desempeño es aún mejor por un 4.49 % que el de JPEG-LS cuadro a cuadro, en media.

Esto parece indicar que la conmutación implementada entre una predicción MED puramente espacial y predictores MED espacio-temporales, es adecuada para explotar la redundancia temporal si esta se presenta o “reducirse” a JPEG-LS cuadro a cuadro en caso contrario. Al submuestrear el conjunto de videos de prueba con $SALTO = 3$ no toda la redundancia temporal fue eliminada, ya que hay regiones pertenecientes al “fondo” en varias secuencias que no cambian demasiado a lo largo de la misma. En estas regiones, un predictor MED con componentes temporales puede ser aún seleccionado, lo que explica la mejor performance por sobre JPEG-LS cuadro a cuadro.

Secuencia	Nivel de Compresión en bpp	Tiempo de Procesamiento en segundos
Salesman	4.13	7.38
Mobile y Calendar	5.90	5.32
Container	3.71	4.93
Tempete	5.06	4.50
Kitchgrass	4.78	1.65
Sean	3.14	4.87
Silent	3.34	4.82
Foreman	4.31	5.00
Promedio	4.30	4.81

Tabla 5.26: Nivel de compresión en bpp y tiempo de procesamiento para LOVICO con secuencias de video submuestreadas temporalmente con $SALTO = 3$

Secuencia	Nivel de compresión en %	Cuadros procesados por segundo
Salesman	48.42	20.19
Mobile y Calendar	26.28	18.80
Container	53.64	20.28
Tempete	36.71	19.11
Kitchgrass	40.21	20.00
Sean	60.69	20.53
Silent	58.28	20.75
Foreman	46.18	20.00
Promedio	46.30	19.96

Tabla 5.27: Nivel de compresión en % y cuadros procesados por segundo para LOVICO con secuencias de video submuestreadas temporalmente con $SALTO = 3$

Algoritmo	Compresión en bpp	Compresión en %
M-CALIC	3.70	53.73
LOVICO	3.97	50.41
LOPT-3D	4.04	49.53
I-CALIC	4.17	47.91
MoJPEG-LS (L=16)	4.23	47.15
MeJPEG-LS	4.64	41.99
JPEG-LS CaC ¹	4.66	41.81

Tabla 5.28: Nivel de compresión medio en bpp y % alcanzado por los algoritmos vistos.

5.5. Comparación de Resultados y Conclusiones

En esta sección se comparan los desempeños de los diversos métodos propuestos, aplicados al conjunto de videos de prueba. En base a estas comparaciones, que se anticipan de resultados presentados en los puntos anteriores de este documento, se extraen las conclusiones que justifican la selección de los métodos de compresión de video sin pérdidas para el sistema a implementar.

En primera instancia se examinarán los niveles de compresión alcanzados, para luego pasar a la cantidad de cuadros procesados por segundo. Finalmente en las conclusiones, se tienen en cuenta ambos parámetros a la vez.

5.5.1. Comparación: Nivel de Compresión

La tabla 5.28 resume los niveles de compresión medios alcanzados por los algoritmos considerados, tanto los presentes en el marco de trabajo propuesto como los considerados a lo largo de este capítulo. La misma está ordenada de forma decreciente en el nivel de compresión, es decir, el que logra la compresión más alta se encuentra en primer lugar, luego el segundo y así sucesivamente.

Observamos que el método propuesto LOVICO alcanza niveles de compresión medios que son solo superados por el algoritmo M-CALIC. Este hecho de por sí es un muy buen resultado ya que como resultado de los ensayos realizados sobre LOCO-I/JPEG-LS se ha obtenido un método competitivo en lo que respecta al nivel de compresión alcanzado. Como hecho simbólico, solo M-CALIC y LOVICO logran superar un 50 % en el nivel de compresión medio.

En la figura 5.20 se expresa la tabla 5.28 como gráfica de barras. En ella la escala ha sido ajustada para que las diferencias entre los distintos algoritmos puedan ser observadas.

Otra cantidad que resulta interesante considerar es la mejora en el nivel de compresión alcanzada por los diferentes algoritmos respecto de JPEG-LS Cuadro a Cuadro, ya que este es nuestro método “base” el cual, como hemos visto, implementa una compresión basada únicamente en explotar redundancia espacial. Dado que los videos de prueba poseen una buena resolución temporal, esta comparación será interesante para tener una noción cuantitativa de que tan eficazmente es utilizada esta la información de extraída de cuadros anteriores para lograr una mayor compresión. Esto se observa en la figura 5.21.

Así, puede verse que es M-CALIC quien logra la más alta mejora con un 20,49 % seguido por LOVICO con un 14,78 %, LOPT-3D con un 13,27 %, I-CALIC con un 10,48 %, MoJPEG-LS con un 9,17 % y por último MeJPEG-LS con un 0,32 %.

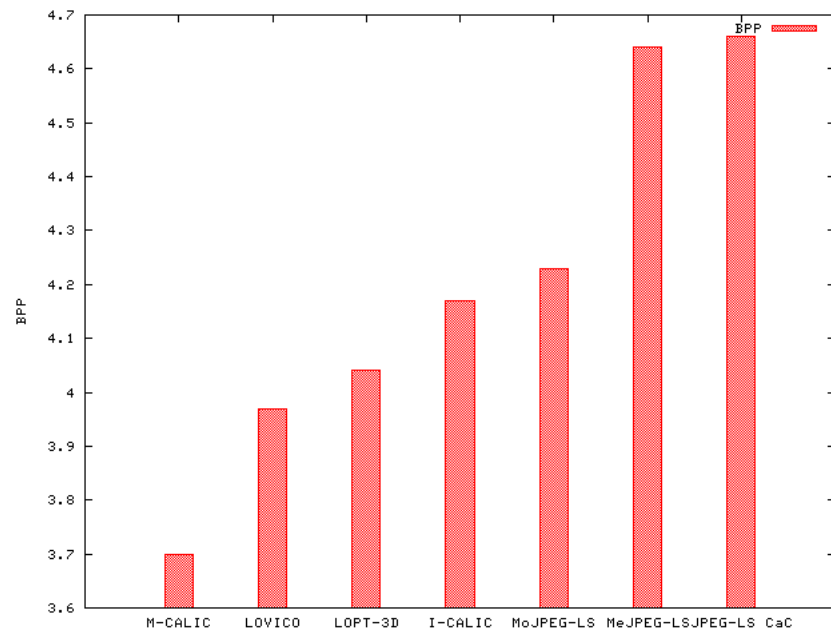


Figura 5.20: Nivel de compresión absoluto medio alcanzado por los diferentes algoritmos aplicados al conjunto de videos de prueba, en bpp.

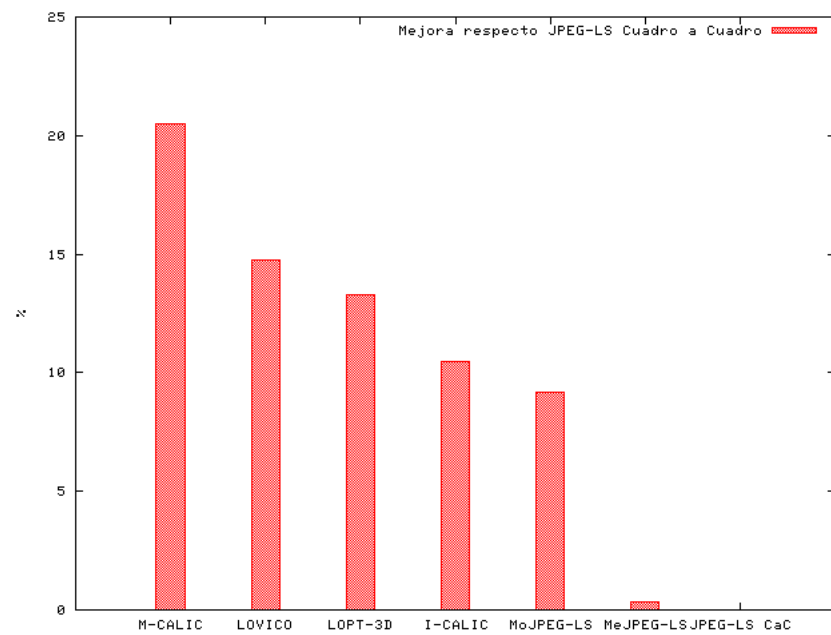


Figura 5.21: Mejora en el nivel de compresión medio respecto a JPEG-LS cuadro a cuadro alcanzada por los diferentes algoritmos

Algoritmo	FPS
MeJPEG-LS	33.15
JPEG-LS CaC	32.69
LOVICO	21.94
I-CALIC	6.50
M-CALIC	2.27
MoJPEG-LS	1.94
LOPT-3D	0.78

Tabla 5.29: Cuadros procesados por segundo.

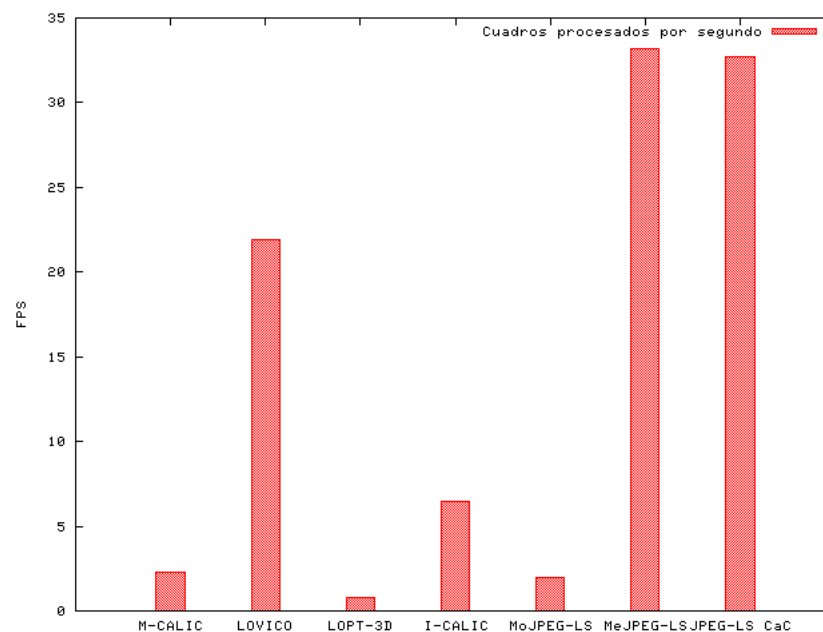


Figura 5.22: Cuadros procesados por segundo en media para los algoritmos considerados

5.5.2. Comparación: Cuadros Procesados por Segundo

La tabla 8.6 muestra la cantidad de cuadros procesados por segundo en media para los algoritmos considerados aplicados al conjunto de videos de prueba. En ella, los métodos están ordenados según capacidades de procesamiento de cuadros por segundo decrecientes.

Como se dijo en la sección 3.2.3 la evaluación para I-CALIC, M-CALIC y LOPT-3D se extrajo de [2]. Allí se afirma que la evaluación de esos tres algoritmos fue efectuada en un PC con procesador Intel Pentium IV de 2.6 GHz. Recordamos que la evaluación de MeJPEG-LS, JPEG-LS CaC, LOVICO y MoJPEG-LS fueron realizadas en un PC con procesador Intel Pentium IV de 3.06 GHz.

Sin embargo, aún considerando una diferencia a favor de MeJPEG-LS, JPEG-LS CaC, LOVICO y MoJPEG-LS proporcional a la relación entre las frecuencias de operación de los procesadores, existe una clara ventaja de los métodos MeJPEG-LS, JPEG-LS CaC y LOVICO.

La tabla 8.6 se expresa gráficamente en la figura 5.22. Se ha mantenido intencionalmente el orden de los algoritmos respecto a su nivel de compresión como en la figura 5.20.

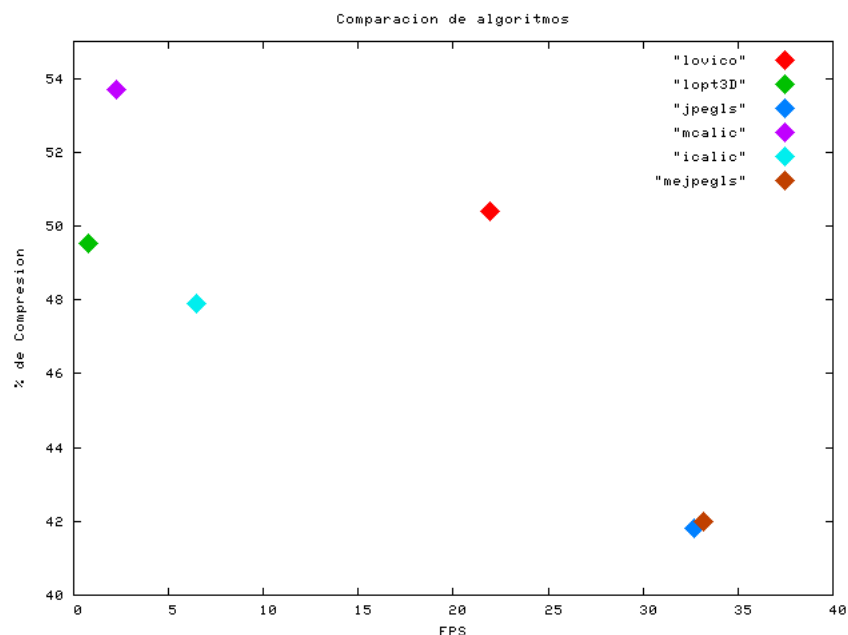


Figura 5.23: Cuadros procesados por segundo vs nivel de compresión en %, ambos en media

5.5.3. Conclusiones

Como primer conclusión observamos que en el mejor caso, la ganancia en el nivel de compresión medio respecto de JPEG-LS cuadro a cuadro es de un 20.49 %, correspondiente a M-CALIC. Sin embargo esta mejora moderada se logra a costas de severas reducciones en la cantidad de cuadros procesados por segundo. Para M-CALIC esta reducción es de 93.07 %, para LOPT-3D 97.61 %, para I-CALIC 80.12 % y para MoJPEG-LS 94.06 %.

Ahora bien, en el caso de LOVICO, la reducción de la cantidad de cuadros procesados por segundo respecto de JPEG-LS cuadro a cuadro es de 32.88 %. Aún volviendo a tener en cuenta la diferencia entre los procesadores de los PC en que se evaluaron los algoritmos, las diferencias son notorias. Por otro lado se tiene además que LOVICO logra niveles de compresión competitivos con los mayores alcanzados por los algoritmos existentes en el marco de trabajo propuesto.

La característica antes mencionada es muy valiosa y deseable en un algoritmo de compresión de video (ya sea con o sin pérdidas) y se traduce en nuestra segunda conclusión. Una forma de expresar claramente el éxito de LOVICO en el compromiso *Nivel de Compresión - Cuadros Procesados por Segundo* es a través de un gráfico que muestre en el eje x la cantidad de cuadros procesados por segundo y en el eje y el nivel de compresión alcanzado, ambos en media. Este gráfico es el de la figura 5.23.

En la figura 5.23 se observa que LOVICO está en el sector más “deseable” del diagrama, es decir en el ángulo superior derecho, logrando al mismo tiempo tener el segundo más alto nivel de compresión y procesar más cuadros por segundo que cuatro de los siete algoritmos considerados. Otra forma de observar esta misma característica es comparando los gráficos 5.20 y 5.22. LOVICO se opone en términos relativos al compromiso “alto nivel de compresión con pocos cuadros procesados por segundo” o “bajo nivel de compresion con muchos cuadros procesados por segundo”.

En base a las observaciones hechas en esta y otras secciones de este documento, se decidió incluir como algoritmos de codificación y decodificación de video sin pérdidas a *JPEG-LS cuadro*

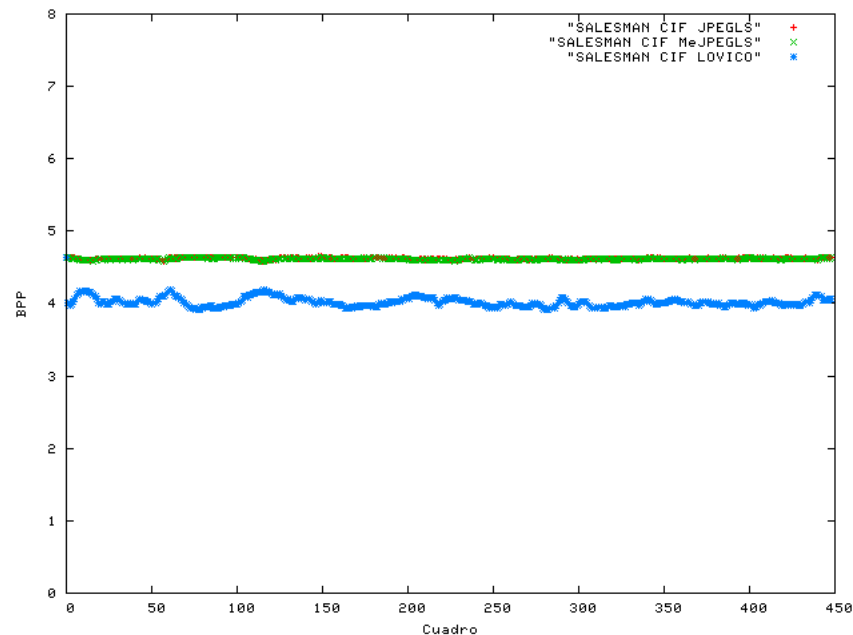


Figura 5.24: Nivel de compresión en bpp “cuadro a cuadro” para SALESMAN

a *cuadro* como método rápido, estándar y con independencia de codificación entre cuadros, a *MeJPEG-LS* por ser el más rápido de los métodos analizados y de implementación trivial a partir de JPEG-LS cuadro a cuadro y a *LOVICO* por presentar una muy buena relación *costo-beneficio* logrando altos niveles de compresión con una fracción de la complejidad de otros métodos.

Ahora bien, esta decisión ha sido tomada en base a valores medios. En lo que hace a los cuadros procesados por segundo, los valores para cada método seleccionado aplicado a videos de prueba notoriamente diferentes, son estables. El nivel de compresión por otra parte presenta variaciones mayores, por lo que para observar con más detalle este punto se muestran en las figuras 5.24 a 5.31 el nivel de compresión “cuadro a cuadro” para cada algoritmo seleccionado (rojo para JPEG-LS cuadro a cuadro, verde para MeJPEG-LS y azul para LOVICO).

En las figuras 5.24 a 5.31 se observa que en las secuencias más “simples” (como CONTAINER o SILENT) el nivel de compresión es prácticamente constante con los cuadros, mientras que en secuencias más complejas (como FOREMAN o MOBILE and CALENDAR) este varía de acuerdo a la complejidad de la misma. Sin embargo, un hecho destacable es que estas variaciones son más bien suaves y están acotadas para todos los videos de prueba por 6 bpp. Esta propiedad es muy deseable cuando el algoritmo de codificación debe interactuar con otros componentes de un sistema de comunicación.

Si bien en secuencias como SEAN, SILENT o CONTAINER se aprecia una mayor diferencia en el nivel de compresión de LOVICO respecto a JPEG-LS cuadro a cuadro o MeJPEG-LS, esto no puede sorprender ya que se tratan de secuencias con alta redundancia temporal, que LOVICO explota a diferencia de JPEG-LS cuadro a cuadro o MeJPEG-LS. Es más interesante el comportamiento que se observa en secuencias complejas como FOREMAN, TEMPETE, o KITCHGRASS, en donde LOVICO “sigue” muy de cerca el nivel de compresión alcanzado por JPEG-LS cuadro a cuadro y MeJPEG-LS, pero en la enorme mayoría de los cuadros, por debajo de estos últimos.

Llegamos así a la tercera conclusión en este punto, y es que LOVICO se comporta aproxima-

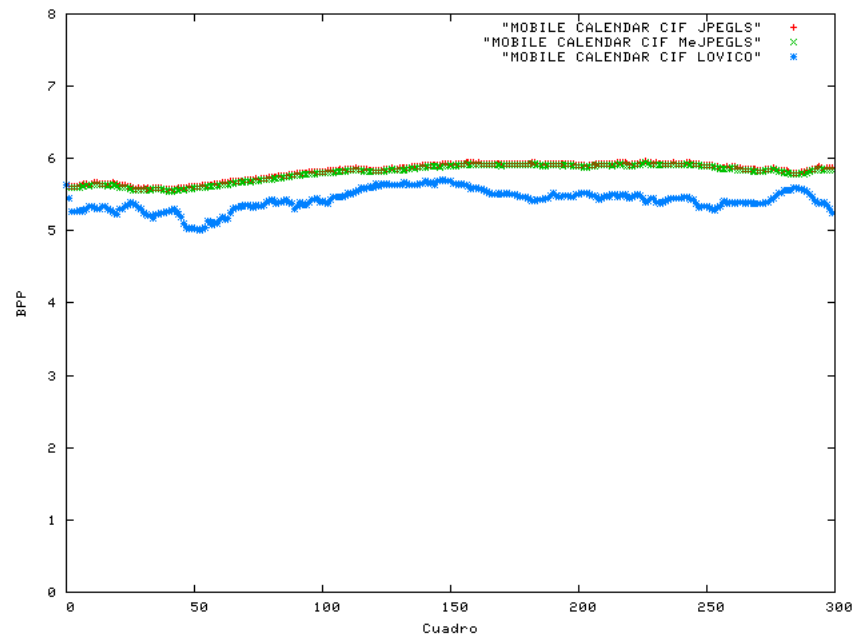


Figura 5.25: Nivel de compresión en bpp “cuadro a cuadro” para MOBILE and CALENDAR

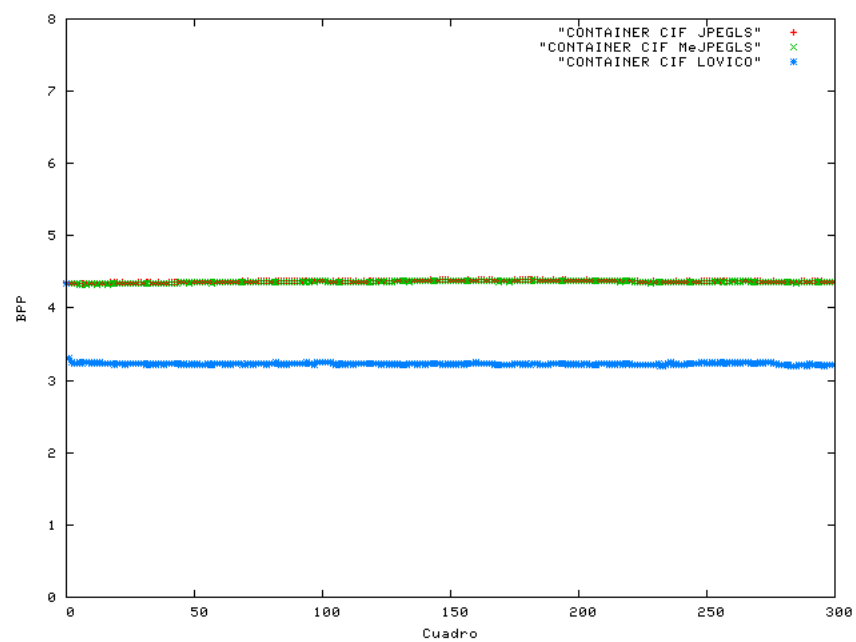


Figura 5.26: Nivel de compresión en bpp “cuadro a cuadro” para CONTAINER

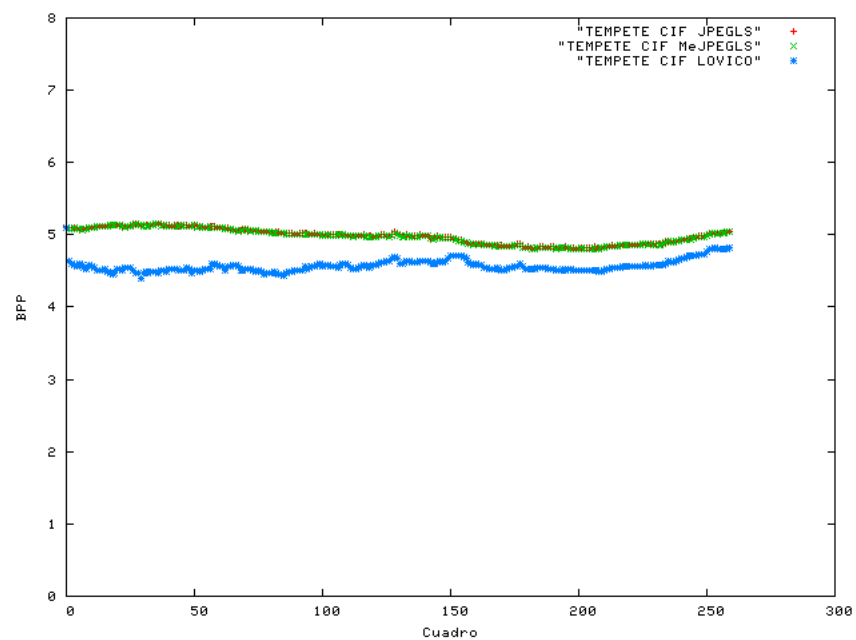


Figura 5.27: Nivel de compresión en bpp “cuadro a cuadro” para TEMPETE

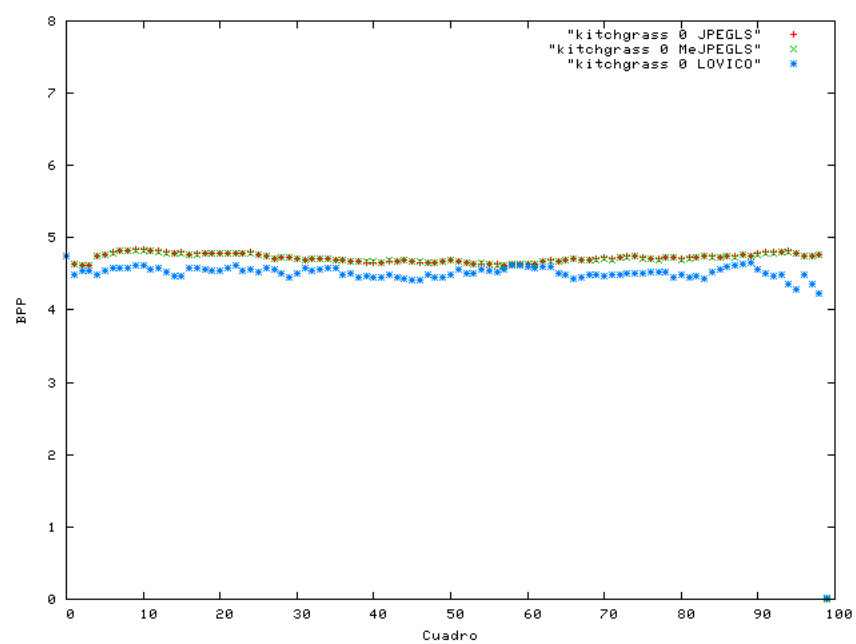


Figura 5.28: Nivel de compresión en bpp “cuadro a cuadro” para KITCHGRASS

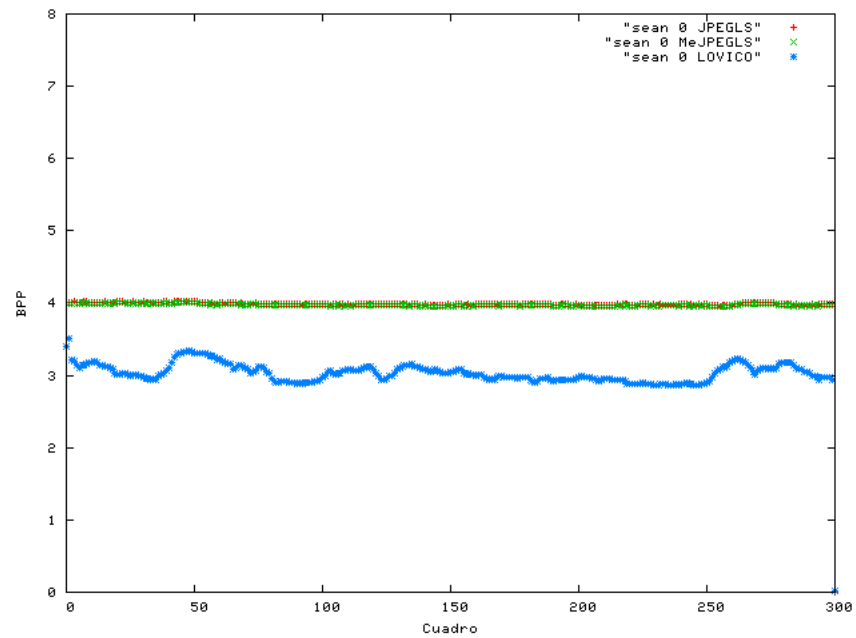


Figura 5.29: Nivel de compresión en bpp “cuadro a cuadro” para SEAN

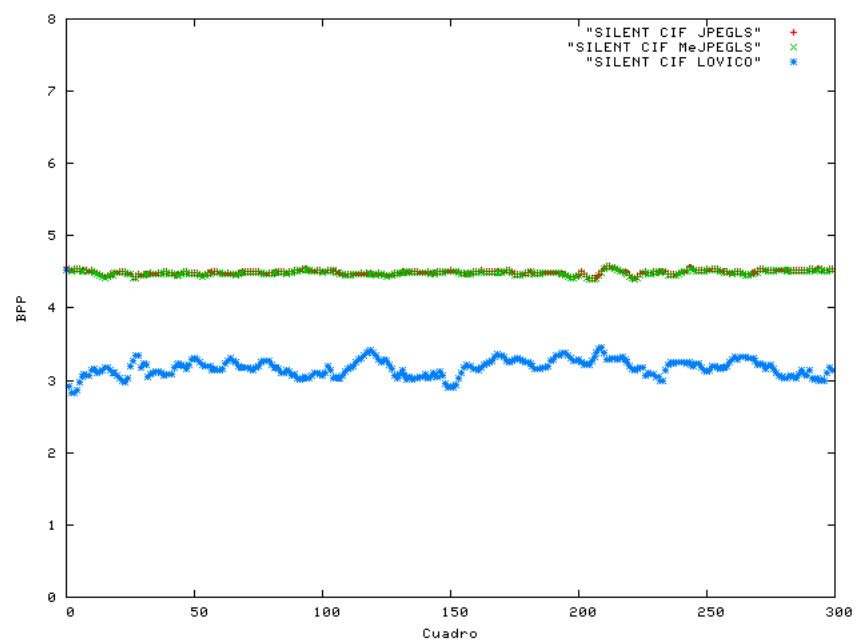


Figura 5.30: Nivel de compresión en bpp “cuadro a cuadro” para SILENT

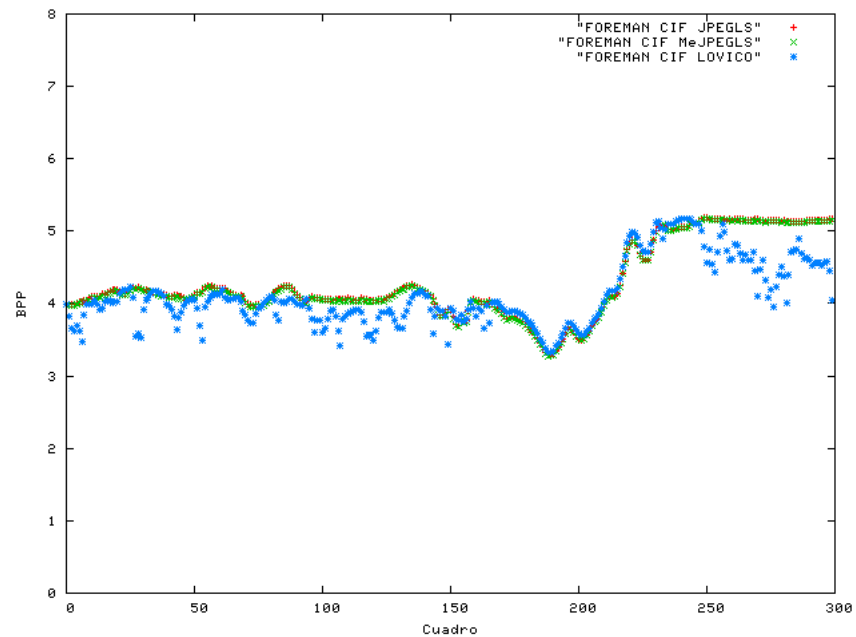


Figura 5.31: Nivel de compresión en bpp “cuadro a cuadro” para FOREMAN

damente *en el peor caso* tan mal como JPEG-LS cuadro a cuadro. Esto permite no solo la comparación con un método estándar sino que también puede ser de suma utilidad a la hora de seleccionar un método de compresión en una situación particular. Además los tres métodos en cuestión presentan cierta regularidad en el nivel de compresión alcanzado para cada cuadro, lo que facilitará su integración al resto del sistema a implementar.

6 STREAMING DE VIDEO

6.1. Introducción

En [22] se realiza un análisis de los diferentes conceptos y características de un *streaming de video*. Se entiende por *streaming de video* a la acción de transmisión de un flujo de datos de video en forma codificada por un servidor a través de una red y decodificada por un cliente remoto a medida que recibe este flujo. Este *stream* o flujo de datos, puede ser “en demanda” donde el stream es almacenado en un servidor por un largo periodo de tiempo o “en vivo” donde tiene un tiempo de vida acotado.

El desarrollo en la capacidad de cómputo, la infraestructura de redes y la accesibilidad del hardware al usuario promedio, ha fomentado el desarrollo de la comunicación de video sobre redes de paquetes.

Con el desarrollo de Internet se ha popularizado desde hace tiempo la transferencia de archivos de audio y video, pero estas transferencias de traducen generalmente en tiempos de espera largos y la imposibilidad de comenzar la reproducción del contenido antes de finalizar la descarga. Un método alternativo sería generar un flujo de video o audio (o ambos) desde un servidor a pedido de un cliente. El cliente reproduce ese flujo entrante en tiempo real a medida que va recibiendo los datos. Este es el caso de un streaming de video. Brinda además la posibilidad de finalizar la conexión con el servidor sin necesidad de descargar todo el contenido antes de poder visualizar una parte de él, haciéndolo atractivo para muchas aplicaciones, por ejemplo transmisión de eventos en vivo y la videoconferencia.

A continuación se analizan las diversas características de un streaming de video.

6.1.1. Diferentes Tipos de Transmisión de Video

Las propiedades específicas de la aplicación de video tienen una fuerte influencia en el diseño del sistema. Debido a esto, se presentan a continuación algunas características a tener en cuenta en los diferentes tipos de comunicación de video en una red de datos.

Un punto clave es reconocer cuántos receptores tendrá el emisor del flujo. Se clasifican de este modo según el tipo de comunicación como *Punto a Punto*, *Multipunto* o *Difusión*.

El caso más típico de comunicación de video es desde un punto a muchos, llamada comunicación multipunto o *multicast*, o de un punto a todos, llamada comunicación por difusión o *broadcast*. Como ejemplo clásico de comunicación por difusión podría presentarse la transmisión de una señal televisiva. Un aspecto a tener en cuenta en la difusión es que el sistema debe diseñarse para otorgar a cada potencial cliente una señal con una calidad determinada.

Por otro lado, a veces es deseable establecer una comunicación únicamente entre dos puntos, llamada punto a punto, como por ejemplo en una videoconferencia. Una característica importante en el caso punto a punto es la existencia o no de un canal de retorno posibilitando una retroalimentación entre emisor y receptor.

Una forma intermedia entre el punto a punto y la difusión es el multipunto. Esta es una forma de comunicación entre uno y muchos puntos, pero no entre uno y todos los puntos como la di-

fusión. El método multipunto es más eficiente para poder comunicarse con múltiples receptores que el punto a punto y tiene pros y contras similares a la difusión.

Otra característica a tener en cuenta es el tipo de codificación realizada a la datos. El video puede ser capturado y codificado en tiempo real o puede ser codificado y almacenado en algún medio para su posterior transmisión. Las aplicaciones interactivas como la video conferencia son ejemplo de aplicaciones que requieren captura y codificación en tiempo real. En otras aplicaciones el video es codificado y luego almacenado para posterior utilización, hecho que se puede realizar en forma local o remota. Un ejemplo de almacenamiento local es el DVD y de almacenamiento remoto es el video bajo demanda. El video previamente codificado no tiene las restricciones del tiempo real, lo que permite mejores tasas de compresión debido a que es posible un mayor número de operaciones sobre los datos, por ejemplo, habilita a realizar una compensación de movimiento (ver sección 5.3.1) exhaustiva. Esta característica está muy vinculada con el tipo de interacción existente con el cliente. Esto es clave para determinar los requerimientos de latencia entre puntas. Se distinguen aplicaciones interactivas y no interactivas. Las aplicaciones interactivas son sometidas a las restricciones del tiempo real, donde la información tiene un tiempo acotado de validez. Si la información llega, pero lo hace fuera de tiempo, es inútil. Las aplicaciones no interactivas permiten latencias mucho más holgadas.

El tipo de canal en el cual se realiza la transmisión influye claramente en el diseño. Una clasificación de las características el canal puede ser: *Canales Estáticos* o *Dinámicos*.

El diseño del sistema depende en gran medida de la existencia o no de variaciones significativas en las características del canal como ser ancho de banda, retardo y tasa de error. Un canal estático nos brindará de antemano mayor información acerca de las posibilidades de nuestro streaming.

Algunos canales soportan bit rate constante o CBR (de la sigla en inglés de Constant Bit Rate) mientras que otros se caracterizan por bit rate variable o VBR (de la sigla en inglés de Variable Bit Rate). Las secuencias de video tienen complejidad variable en el tiempo, por lo que codificar una secuencia con una calidad fija requiere un bit rate variable y codificar con un bit rate fijo requiere una calidad variable.

6.1.2. Arquitectura de un sistema de streaming de video

Un sistema de streaming de video está compuesto por varios bloques. El video sin compresión proveniente de una cámara de video u otro dispositivo de captura requiere una ancho de banda inaceptable para la gran mayoría de las redes. Esta razón hace del bloque de compresión una etapa fundamental. Los esquemas de compresión de video pueden clasificarse en dos categorías *escalables* o *no escalables*. La escalabilidad se refiere a la capacidad de recuperar información desde imágenes o video que tengan un significado físico mediante la decodificación de secuencias de bits con información parcial. Los *codificadores escalables* o *codificadores en capas*, pueden tener un granulado grueso de 2 o 3 capas por imagen, o fino. En este último caso el flujo de datos puede ser cortado en cualquier punto obteniendo, cuanto mayor la cantidad de bits recibidos, una mayor calidad de la imagen. En [23] se detallan distintos modos de codificación escalable de video.

Para contrarrestar las dificultades presentes en la red, tales como congestión y las pérdidas de datos, es deseable implementar algún tipo de calidad de servicio en la capa de aplicación. Se analizarán a continuación el control de la congestión y las pérdidas en la red.

6.1.3. Control de congestión

La congestión en la red lleva a pérdidas de paquetes y retardos en la transmisión. Una forma de evitar estos efectos es aplicando un control de congestión al emisor o al receptor del stream. Si el control es tarea del emisor este debe adaptar la transmisión de los datos de video al ancho de banda disponible. Este control se puede realizar de dos formas: realizando sondeos o siguiendo un modelo. La forma de control realizando sondeos se adapta al ancho de banda disponible ajustando la tasa de envío de modo de tener una tasa de pérdida de paquetes p debajo de cierto umbral P_{th} . La tasa de envío se controla de dos formas: incrementando aditivamente y decreciendo multiplicativamente, o incrementando y decreciendo multiplicativamente. Otra forma de encarar el control por parte del emisor es siguiendo un modelo, basado en el modelo TCP del control de congestión. La tasa de datos de una conexión TCP se puede caracterizar por [24]:

$$\lambda = \frac{1.22 \times MTU}{RTT \times \sqrt{p}} \quad (6.1)$$

donde λ es la tasa de datos de una conexión TCP, MTU (de la sigla en inglés Maximun Transfer Unit) es el tamaño de paquete, RTT (de la sigla en inglés Round Trip Time) es el tiempo de ida y vuelta de la conexión y p es la tasa de pérdida de paquetes.

De esta forma se ajusta la tasa de envío de datos del stream a este modelo.

Por otro lado, es posible que el control de la congestión se realice por parte del receptor del stream de video, descartando o agregando una mayor cantidad de datos según el ancho de banda disponible. Este tipo de control está estrechamente vinculado con la codificación y decodificación escalable. De una forma similar al control realizado en el emisor, el receptor puede actuar de formas: siguiendo un modelo o realizando sondeos del ancho de banda.

En el control por sondeos si no detecta congestión agrega una capa más al video, si esto no es exitoso, la descarta.

La forma de control por modelado del ancho de banda disponible, sigue las pautas de lo analizado anteriormente con el emisor.

Además del control realizado exclusivamente por el emisor o el receptor, existe una solución híbrida donde tanto el emisor como el receptor toman parte en el control de la congestión.

6.1.4. Acciones frente a las pérdidas en la red

Un problema fundamental que afecta la transmisión de video son las pérdidas. Una pequeña pérdida puede tener un efecto destructivo en el stream completo si no se toman medidas al respecto. Ocurren diferentes tipos de pérdidas dependiendo de la red en consideración. Por ejemplo en Internet, la causa principal de las pérdidas de paquetes es la congestión, mientras que en canales inalámbricos las pérdidas ocurren debido a errores en los bits a nivel de capa física.

De este modo los sistema de streaming de video son dotados con algún tipo de control de errores para disminuir el efecto de la pérdidas.

Retransmisiones: Este tipo de acción es posible en sistemas que poseen un canal de retorno. El receptor confirma o no la llegada correcta de paquetes, permitiendo un uso eficiente del ancho de banda ya que son reenviados solamente los paquetes dañados o perdidos. Esta retransmisión tiene como consecuencia mayores demoras en el tiempo de envío punta a punta (RTT), pudiendo ser inadmisibles en aplicaciones estrictas de tiempo real.

Sin embargo si el tiempo es corto respecto a la demora máxima admisible la retransmisión con

demora acotada es una opción posible. Cuando el receptor detecta la pérdida del paquete N se chequea si es válida la condición 6.2 para solicitar el reenvío del paquete al emisor.

$$(T_c + RTT + D_s < T_d(N)) \quad (6.2)$$

donde T_c es el tiempo actual, D_s es un término de ajuste que contempla el tiempo de decodificación del receptor, y T_d es el tiempo en el que se espera desplegar el cuadro que contiene datos en el paquete N .

Cuando el control se encuentra del lado del emisor, si este recibe un pedido de retransmisión del paquete N , chequea si es válida la condición 6.3 para reenviar el paquete al receptor.

$$(T_c + RTT/2 + D_s < T'_d(N)) \quad (6.3)$$

donde $T'_d(N)$ es una estimación de T_d .

Corrección de errores hacia adelante (o FEC): El objetivo de esta acción es agregar redundancia para recuperarse frente a errores. Esto posee ventajas y desventajas. A diferencia de la retransmisión no se necesita un canal de retorno posibilitando menores retardos comparado a la retransmisión de los paquetes, ya que no depende del RTT. Por otro lado dentro de las desventajas tenemos una cantidad de sobreprocesamiento, algunas veces innecesario si no hay pérdidas. Este método es eficiente si la tasa de errores o pérdidas en el canal es conocida o estimable, siendo ineficiente si la redundancia utilizada es menor a la necesaria para el caso en cuestión. En redes de paquetes no se posee el conocimiento previo de dicha información.

Ocultamiento del error: Esta acción tiene como objetivo estimar la información perdida explotando la correlación espacial y temporal que se presentan en las secuencias de video. Hay tres opciones para el ocultamiento del error: interpolación espacial, interpolación temporal e interpolación temporal con compensación de movimiento. El objetivo de la interpolación espacial es estimar los píxeles faltantes mediante una extrapolación de los píxeles vecinos que se hayan recibido correctamente. La extrapolación temporal se basa en estimar los píxeles faltantes mediante la copia de los píxeles correctos del cuadro anterior en la misma ubicación espacial. Por otro lado la extrapolación temporal con compensación de movimiento estima el bloque perdido a partir del bloque en la misma posición del cuadro anterior compensándolo en movimiento. Cabe destacar que el proceso de ocultamiento se realiza enteramente en el decodificador y es un proceso con pérdida de información.

Codificación de video resistente al error: La codificación de video resistente al error busca generar tramas de bits resistentes a ciertos tipos específicos de errores. Los dos problemas básicos que afectan al stream son la pérdida de sincronización y la propagación del error en caso de sistemas con codificación predictiva.

La sincronización de la trama puede solucionarse utilizando códigos de largo fijo dado que en estos se conoce la posición de comienzo y finalización de la palabra. Sin embargo estos códigos no permiten una buena buena compresión comparado a los códigos de longitud variable.

La forma más simple de lograr una resincronización de la trama es agregar marcadores de resincronismo. La idea es dotar al flujo de datos de una secuencia de bytes única y fácil de ubicar, tal que si el decodificador pierde sincronismo, pueda resincronizarse ubicando estas secuencias.

6.2. Protocolos para streaming de video

Protocolos de transporte

Los protocolos de transporte para streaming incluyen UDP, TCP, RTP (de la sigla en inglés Real-Time Transport Protocol) y RTCP (de la sigla en inglés Real-Time Control Protocol). Tanto TCP como UDP aportan las funciones básicas de transporte mientras que RTP y RTCP corren encima de estos.

Dado que las retransmisiones de TCP provocan demoras, UDP es el protocolo más usado para streaming de video. Se debe tener en cuenta que UDP no asegura la entrega de los paquetes, por lo tanto el receptor deberá confiar en una capa superior (RTP) para detectar las pérdidas. RTP es un protocolo estándar para Internet que provee transporte de punta a punta soportando aplicaciones de tiempo real. RTCP es el protocolo que lo acompaña a modo de proveer calidad de servicio a los participantes de la sesión.

Protocolos de control de sesión RTSP

El RTSP (de la sigla en inglés Real Time Streaming Protocol) establece y controla uno o muchos flujos sincronizados de datos, ya sean de audio o de video. Actúa como un mando a distancia mediante la red para servidores multimedia. En la mayoría de los casos RTSP usa TCP para datos de control del reproductor y UDP para los datos de audio y video aunque también puede usar TCP en caso de que sea necesario.

6.3. Streaming de Video con TCP

En [25] , [26] y [27] se analiza la viabilidad de TCP como protocolo de streaming de video. TCP presenta ciertas características que lo hacen apto para un streaming, como el control de congestión y la entrega confiable de datos. A pesar de que su uso provee una entrega confiable del streaming de video, es difícil alcanzar una buena calidad del streaming debido a su comportamiento en la tasa de transferencia con incrementos aditivos y decrementos multiplicativos. El uso de retransmisiones puede introducir demoras inaceptables haciendo que los datos a desplegar pierdan validez.

Estos problemas pueden ser solucionados mediante el uso de un buffer en el receptor según [27]. El buffer debe ser lo suficientemente grande como para asegurar la tasa de cuadros y el tamaño de los mismos.

En [27] se expresa una cota inferior para el tamaño del buffer de recepción como:

$$q_0 \geq \frac{0.16}{pP_u} \left[1 + \frac{9.4}{b} \left(\frac{T_0}{RTT} \right)^2 \min \left(1, 3\sqrt{\frac{3bp}{8}} \right) p (1 + 32p^2) \right] \quad (6.4)$$

Donde: q_0 es el tamaño del buffer de recepción, p es la tasa de paquetes perdidos, P_u es la probabilidad deseada de agotamiento del buffer, b es el número de paquetes que son reconocidos por un ACK, T_0 es el timeout en la retransmisión y RTT es el tiempo de ida y vuelta.

6.3.1. Implementación

El sistema de streaming consta de un servidor encargado de la captura, codificación del video y transmisión del mismo, y de un cliente receptor del streaming que determina el tamaño y cantidad de cuadros a capturar, además del canal (R,G,B o Y) que será codificado. El cliente se encarga además de la decodificación y desplegado en pantalla del video contenido en el stream

recibido.

El servidor de streaming posee una dirección IP conocida por el cliente y se inicia escuchando un puerto TCP determinado, 5005, en espera de una conexión. Al recibir una petición de conexión, se inicia el conocido “three way handshake” de TCP, tras lo cual el cliente envía los parámetros del video a capturar por parte del servidor. Estos parámetros son: cantidad de cuadros, formato de los cuadros (CIF, QCIF) y canal a codificar.

Tras recibir estos parámetros el servidor inicia el bloque de captura. El video generado por este bloque de captura es codificado y almacenado en un buffer de transmisión. Al mismo tiempo, otro proceso se encarga de enviar dichos datos almacenados a través de la conexión establecida con el cliente receptor del stream. Del lado del cliente, un buffer almacena los datos obtenidos a través de la conexión TCP y paralelamente inicia la etapa de decodificación y presentación del mismo.

Al tratarse de una conexión TCP se tiene garantizado la llegada correcta de los datos al receptor. Es decir no es necesario realizar un control de errores en la aplicación ni tampoco tomar acciones sobre la entrega ordenada de los datos.

A modo de controlar el flujo de bits hacia el receptor se implementan dos acciones: se asignan buffers de transmisión y recepción, y se intercambia entre tres modos de codificación distintos. Al comenzar la captura y codificación de cuadros este stream es almacenado en un buffer de transmisión, no permitiéndose el inicio de la transferencia por la red hasta que dicho buffer no contiene una cantidad de datos determinada. Esta cantidad inicial de llenado parcial del buffer aún no ha sido optimizada y se encuentra en estudio.

Del lado del receptor ocurre la situación inversa, el decodificador no se inicia hasta que los datos contenidos en el buffer de recepción no sean suficientes.

El efecto que producen estos buffers en un retardo inicial en la transferencia del video permitiendo, si el tamaño asignado al llenado parcial es correcto, una tasa de cuadros por segundos estable durante el proceso.

Para evitar el desborde o vaciado de los buffers se debe implementar alguna acción frente a la tasa de datos emitida, y es así que surge intercambiar entre los diferentes modos de codificación implementados.

Los tres modos de codificación implementados son JPEG-LS cuadro a cuadro, MeJPEG-LS y LOVICO. JPEG-LS presenta en promedio (ver tabla 5.2) para los videos de prueba un nivel de compresión de 41.8 % y una tasa de cuadros por segundo de 32.69. El algoritmo MeJPEG-LS presenta para ese mismo conjunto de videos un promedio (ver tabla 5.4) un nivel de compresión de 41.99 % y una cantidad de 33.15 fps. Por su parte LOVICO presenta en promedio (ver tabla 5.25) un nivel de compresión de 50,41 % y 21.94 fps.

Cabe destacar que los datos de cuadros por segundo para cada algoritmo son obtenidos a través de una aplicación que realiza exclusivamente la codificación de un video. Ningún proceso de red está ejecutándose durante esas mediciones.

Estos datos representan para JPEG-LS cuadro a cuadro, MeJPEG-LS y LOVICO un flujo de datos por parte del codificador de 15.43 Mbps, 15.60 Mbps y 8.82 Mbps respectivamente para un tamaño de cuadro CIF. Estos valores surgen de la ecuación 6.5.

$$\lambda = \frac{F_c \times C_c \times bpp \times fps \times (100 - N_{comp})}{100} \quad (6.5)$$

Donde λ es la tasa de bits por segundo del codificador, F_c es la cantidad de filas que componen cada cuadro, C_c es la cantidad de columnas que componen cada cuadro, bpp es la cantidad de bits usados para representar un píxel sin compresión (en este caso 8), fps la cantidad de cuadros por segundo y N_{comp} es el nivel de compresión en porcentaje alcanzado por el algoritmo.

Suponiendo que el proceso de red es capaz de igualar o aproximar la tasa de transferencia de datos a la red con el flujo de bits generado por el codificador, entonces es posible que intercambiando entre estos tres modos de compresión se pueda controlar el nivel de ambos buffers y mantener la tasa de cuadros por segundo estable.

Una descripción del software de red desarrollado puede encontrarse en el apéndice A.

7 ADQUISICIÓN DE VIDEO

Una etapa que compone el sistema a implementar aún no descripta es la de adquisición del video. La misma consiste en lograr manipular una tarjeta de adquisición de video conectada a una cámara de modo de entregar cuadros al procedimiento de compresión cuando este lo requiera.

Por lo tanto en este capítulo se exponen las características principales del software y hardware involucrado en la captura de video.

El primero de ellos concierne principalmente a las características de la interfaz de programación de aplicaciones o *API* (de la sigla en inglés Application Programming Interface) utilizada.

En lo que hace al hardware, se listan las características principales del mismo, el cual se limitó al que poseen los integrantes de este grupo de proyecto.

7.1. API V4L2

Una API es una biblioteca que le permite al programador desarrollar aplicaciones en una capa intermedia entre la aplicación y el hardware. En este proyecto el API para captura de video utilizada fue *Video for Linux Two*, comúnmente conocido por V4L2. La misma soporta una gran variedad de hardware, como dispositivos del tipo bttv (bt848, bt878), saa713x, saa7146, cx88 , etc.

En el caso de los dispositivos multimedia, la API necesita interactuar con el núcleo del sistema operativo. En Linux esto se hace mediante la carga de módulos o *LKM* (de la sigla en inglés Loadable Kernel Modules). En forma muy simplificada, estos son archivos objeto utilizados para extender la funcionalidad del núcleo. En este caso la funcionalidad es el manejo de dispositivos de adquisición de video. Cuando la funcionalidad ya no es requerida esta puede ser descargada, liberando así memoria.

Mas allá de estos detalles, lo relevante en este punto es poder utilizar V4L2 para interactuar con el dispositivo de captura para que este nos entregue cuadros cuando lo solicitemos. Para lograr este objetivo la programación de un dispositivo de V4L2 requiere los siguientes pasos:

1. Apertura del dispositivo
2. Configuración del dispositivo
3. Negociar un tipo de datos
4. Negociar un método de entrada-salida
5. Bucle de entrada-salida
6. Cerrado del dispositivo

Nombre del dispositivo	Descripción
/dev/video0..63	Capturadoras de video
/dev/radio0..63	Dispositivos de radio AM/FM
/dev/vtx0..318	Dispositivos de teletexto
/dev/vbi0..15	Dispositivos VBI

Tabla 7.1: Dispositivos reconocidos por V4L2

Los mismos se describen en las siguientes subsecciones.

7.1.1. Apertura del dispositivo

Para abrir un dispositivo V4L2 (es decir, habilitarlo para su uso) las aplicaciones llaman a la función `open()` con el nombre del dispositivo deseado como parámetro de entrada junto una bandera para el modo de acceso.

```
#include <fcntl.h>
int open(const char *nombre_dispositivo, int flags);
```

En la tabla 7.1 se muestran los nombres de dispositivos reconocidos por V4L2. Por otro lado el modo de acceso debe ser seteado a `O_RDWR`. Los dispositivos de entrada solo soportarán lecturas y los de salida escritura.

Esta función retornará un descriptor de archivo al dispositivo que desea ser abierto. En caso de error la función retornará `-1` y se configurará la variable de error en un valor adecuado.

Un detalle importante es que cuando la función `open()` es llamada, el dispositivo se inicializa en sus valores por defecto, nunca en un estado indefinido.

7.1.2. Configuración del dispositivo

Para realizar la configuración del dispositivo previamente abierto, la aplicación debe primero consultar a la API para saber que opciones de configuración soporta. Las propiedades más importantes para setear en un dispositivo de captura de video son:

- Entrada de video
- Salida de video
- Estándar de video
- Modulador de entrada
- Opciones de imagen
- Norma de video
- Canal de entrada (para el caso de entradas auxiliares y sintonizadores)
- Resolución de la ventana de captura

Los dispositivos V4L2 utilizan la función `ioctl()` para su configuración:

```
#include <sys/ioctl.h>
int ioctl(int fd, int petición, void *argp);
```

Donde `fd` es el descriptor de archivo retornado por la función `open()`, `petición` es el código de la petición V4L2 definido en `videodev.h` (por ejemplo: `VIDIOC_QUERYCAP`) y `argp` es un puntero a una estructura que contendrá los parámetros de entrada-salida según la petición.

Una salida exitosa de la función retorna 0, ante una falla retorna `-1` y setea la variable de error a un valor apropiado.

7.1.3. Negociar un tipo de datos

Los diferentes dispositivos intercambian diferentes tipos de datos con las aplicaciones. Aún dentro de un mismo tipo de dispositivo existen diferentes formatos posibles, por ejemplo en el caso de captura de imágenes.

A pesar de que los controladores deben proveer un tipo por defecto, las aplicaciones deben negociar un formato particular antes de proceder al intercambio de datos. Negociar implica que la aplicación en cuestión seleccione el tipo de formato adecuado según el reporte dado por el controlador.

El mecanismo para negociar el formato de los datos es utilizar la función `ioctl()` con el estructurado `v4l2_format` y las peticiones `VIDIOC_G_FMT` o `VIDIOC_S_FMT` según se desee obtener o setear el formato.

En lo que respecta a imágenes, existen dos grandes familias de formatos de captura: YUV y RGB.

Se debe tener en cuenta que no se supone que los controladores conviertan el formato de la imagen en el espacio del kernel, ellos solamente deben enumerar los formatos soportados directamente por el hardware. Si es necesaria una conversión, ésta debe ser integrada en la aplicación.

7.1.4. Negociar un método de entrada-salida

Un dispositivo V4L2 posee tres tipos de captura-escritura posibles:

- Usando funciones `Read()`-`Write()`. Es el método clásico de entrada-salida, automáticamente seleccionado cuando un dispositivo es abierto. Otros métodos deben ser negociados en la etapa correspondiente. En este método los drivers pueden necesitar el uso de la CPU para copiar datos, pero también pueden soportar el uso de DMA de o desde la memoria del usuario, lo que haría que este método no sea necesariamente menos eficiente que otros métodos que solo intercambian punteros a un buffer.
- Usando mapeo a memoria. Este método es uno de los más rápidos, no es necesario usar funciones `read()` o `write()`, si no que se usa la función `mmap()`, que devuelve un puntero al buffer que contiene los datos. Un dispositivo soporta este método cuando tiene activada la bandera `V4L2_CAP_STREAMIN`, en el campo “*capabilities*” de la estructura

V4l2_capability.

- Usando “*user pointer*”. Este método combina las ventajas de los dos métodos anteriores. Los buffers son colocados en memoria por la aplicación misma, y pueden, por ejemplo, residir en memoria virtual o compartida. Entre la aplicación y la API, sólo son intercambiados punteros a memoria.

7.1.5. Bucle de entrada-salida

Si se está usando el método *user pointer* o mapeo a memoria, primero se debe ejecutar un paso intermedio, que consiste en iniciar la transmisión del flujo de datos. Estos métodos poseen un buffer en forma de cola, de manera que los dispositivos de captura puedan encolar los datos y la aplicación sacar estos en orden FIFO (de la sigla en inglés First In First Out). En el caso de los dispositivos de salida, la aplicación coloca datos en la cola y la API saca datos desde la cola. El buffer previamente tiene que haberse configurado, para que pueda almacenar cuadros con las mismas propiedades seteadas para la captura.

En el caso de usar las funciones `read/write`, en el ciclo de captura, sólo se cuenta con un buffer que permite almacenar un cuadro, es decir no hay una cola de cuadros, y en cada ciclo se debe llamar a la función `read()`.

```
#include <unistd.h>
size_t read(int fd, void *buf, size_t cantidad);
```

Con `fd` el descriptor de archivo retornado por la función `open()`, `buf` es el buffer donde se almacenará la captura y `cantidad` es la cantidad de bytes que se intentarán leer desde el dispositivo en cuestión.

Por defecto `read()` se bloquea hasta que hayan datos disponibles. Ante una salida exitosa de la función esta retorna el número de bytes leídos. No se considera un error si el número es menor que el número de bytes requeridos, o la cantidad de datos requeridos para un cuadro. Esto puede ocurrir si la función fue interrumpida por alguna señal. En caso de ocurrir un error la función retorna `-1` y setea la variable de error a un valor adecuado según el caso. En el próximo llamado a la función, comenzará al principio de un nuevo cuadro.

7.1.6. Cerrado del dispositivo

Consiste en cerrar el dispositivo de captura abierto. Esta acción no depende del método de entrada-salida utilizado. Para el cierre se ocupa la función `close()`.

```
#include <unistd.h>
int close(int fd);
```

El único argumento es `fd`, de nuevo, el descriptor de archivo retornado por la función `open()`. Cualquier entrada-salida en progreso es finalizada y los recursos asociados al descriptor de archivo son liberados. Sin embargo los formatos de datos y otras propiedades permanecen invariables. La función retorna un `0` ante una salida exitosa y `-1` en caso de falla configurando la variable de error correspondiente.

7.2. Implementación

A continuación se listan algunas de las funciones implementadas para el manejo del dispositivo de adquisición de video. El interés de este punto radica en la simplicidad con que se puede a partir del hardware instalado, utilizando V4L2, obtener cuadros en el objeto que utiliza la biblioteca de procesamiento de imágenes CImg (ver [28]).

- `int abre_dispositivo(char* DISPOSITIVO);`
Crea un descriptor de archivo para realizar operaciones de entrada salida con el `DISPOSITIVO` (ej.: `/dev/video0`). Devuelve `-1` en caso de error, de lo contrario devuelve un descriptor de archivo válido.
- `int captura_size_cuadro(int descriptor, int fil, int col);`
Setea parámetros de la captura como ser entrada de video, tamaño de cuadro, colores. Permite seleccionar un tamaño de cuadro de cuadro. Devuelve `-1` en caso de error.
- `CImg<unsigned char> leer_cuadro(int descriptor, int fil, int col);`
Devuelve un objeto imagen de CImg con tamaño $fil \times col$ de tipo de píxel `unsigned char`, es decir un valor entre 0 y 255 correspondiente a 8 bits por píxel.
- `int cierra_captura(int descriptor);`
Cierra el descriptor de archivo `descriptor` y finaliza la captura.

7.3. Hardware Utilizado

Se emplearon para la captura los siguientes dispositivos:

- Capturadora FlyVideo 2000. Utiliza chipset SAA1730 de Philips. Interfase con el computador: PCI 2.2. Entrada de video: S-Video (conector mini-Din 4-pin) y Video compuesto (conector RCA).
- Cámara de video Panasonic NV-G303B. Salida de video RCA norma PAL B. $SNR_{\text{dB}} 40$.

8 SISTEMA IMPLEMENTADO

En este capítulo se detallan las características del software implementado y hardware utilizado para el funcionamiento del *sistema de punta a punta*. El sistema de punta a punta está compuesto por seis bloques: *captura*, *codificación*, *transmisión*, *recepción*, *decodificación* y por último el bloque de *presentación*. Un diagrama de este sistema se puede apreciar en la figura 8.1.

El bloque de captura es el encargado de adquirir cuadros de video procedentes del video analógico generado en la cámara de video. De este modo, en esta etapa ingresa un video analógico y a su salida se obtiene una secuencia de imágenes digitales correspondientes a los cuadros del video.

Por otro lado, el bloque de codificación utiliza en un principio el algoritmo JPEG-LS para codificar un cuadro y luego alguno de los tres algoritmos decodificación implementados: JPEG-LS cuadro a cuadro, MeJPEG-LS ó LOVICO. Es así que este bloque tendrá en su entrada una imagen digital y a su salida un flujo de bits correspondiente a la codificación de los distintos cuadros que integran al video. En forma simétrica el bloque de decodificación recibe a su entrada una cadena de bits y devuelve una imagen digital a través de los correspondientes decodificadores de video.

Los bloques de recepción y transmisión son los encargados del transporte del stream de video del servidor al cliente. El bloque de transmisión envía los datos entregados por el codificador al receptor, y este último los entrega al decodificador. En todo este proceso se garantiza la entrega y el orden de los datos transmitidos, por lo que el codificador y el decodificador no deben ocuparse de dicho control. Por último el bloque de presentación es el encargado de desplegar las secuencias de imágenes obtenidas a la salida del decodificador.

Luego de realizada la descripción del hardware y software en cada uno de los bloques se procederá a la evaluación del sistema completo y con los datos relevados se tendrán las bases para elaborar las conclusiones al final del capítulo.

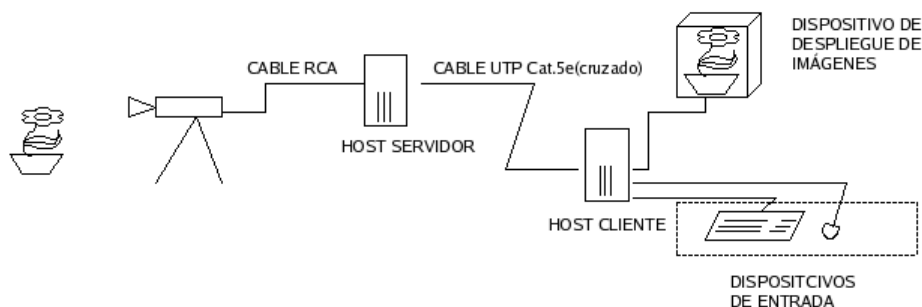


Figura 8.1: Esquema del sistema completo

Secuencia de Video	JPEG-LS	MeJPEG-LS	LOVICO
Salesman	16.91	17.10	12.90
Mobile and Calendar	16.09	16.35	12.31
Container	16.94	17.28	13.15
Tempete	16.55	16.71	12.49
Kitchgrass	17.09	12.42	12.90
Sean	17.40	17.75	13.39
Silent	17.35	17.63	13.42
Foreman	17.24	17.66	12.94
PROMEDIO	16.95	16.61	12.94

Tabla 8.1: Cuadros por segundo generados en codificación en el host servidor por JPEG-LS cuadro a cuadro, MeJPEG-LS y LOVICO para los distintos videos de prueba utilizando el canal verde.

8.1. Hardware Utilizado

Se describe en esta sección el hardware empleado para la puesta en funcionamiento del sistema desde la etapa de captura a la etapa de presentación final del video capturado.

El bloque de captura está compuesto por una cámara de video analógica y una capturadora de video. La capturadora es una FlyVideo 2000 con chipset SAA1730 de Philips. Tiene una interfase con el computador PCI 2.2. Posee entradas de video S-Video (conector mini-Din 4-pin) y video compuesto (conector RCA). La cámara de video es una Panasonic NV-G303B que posee salida de video RCA norma PAL B. El conector RCA de salida de video se une a la entrada correspondiente en la capturadora por medio de un cable RCA a RCA simple. La tarjeta capturadora se conecta al bus PCI del host servidor.

Los bloques de compresión y transmisión están constituidos por un *host servidor*. Dicho host posee un procesador *AMD Sempron 2500* a 1750 MHz con 384 MB de memoria RAM, tarjeta de red Ethernet de 100 Mbps y 4 slots PCI. Este host se conecta al *host cliente* a través de un cable cruzado *UTP Cat.5e* uniendo físicamente las tarjetas de red incluidas en cada host.

Dado que este host difiere del utilizado para la evaluación de desempeño de los algoritmos implementados en el proyecto, es prudente realizar un relevamiento conteniendo los cuadros por segundo que es capaz de generar en codificación para los algoritmos JPEG-LS cuadro a cuadro, MeJPEG-LS y LOVICO.

Se observa comparando la tabla 8.1 con los resultados obtenidos a partir de las tablas 5.2, 5.4 y 5.25 que el host servidor tiene un desempeño en promedio un 46 % inferior al del equipo empleado para la evaluación de los algoritmos el cual actuará como host cliente.

El host cliente integra a los bloques de recepción, decodificación y presentación. Posee un procesador *Intel Pentium 4* funcionando a 3.06 MHz, 512 MB de memoria RAM y una tarjeta de red Ethernet de 100 Mbps. Además posee un dispositivo de salida de video, ya sea monitor o proyector, y dispositivos de entrada (teclado y mouse) que permiten el ingreso de datos al usuario del sistema.

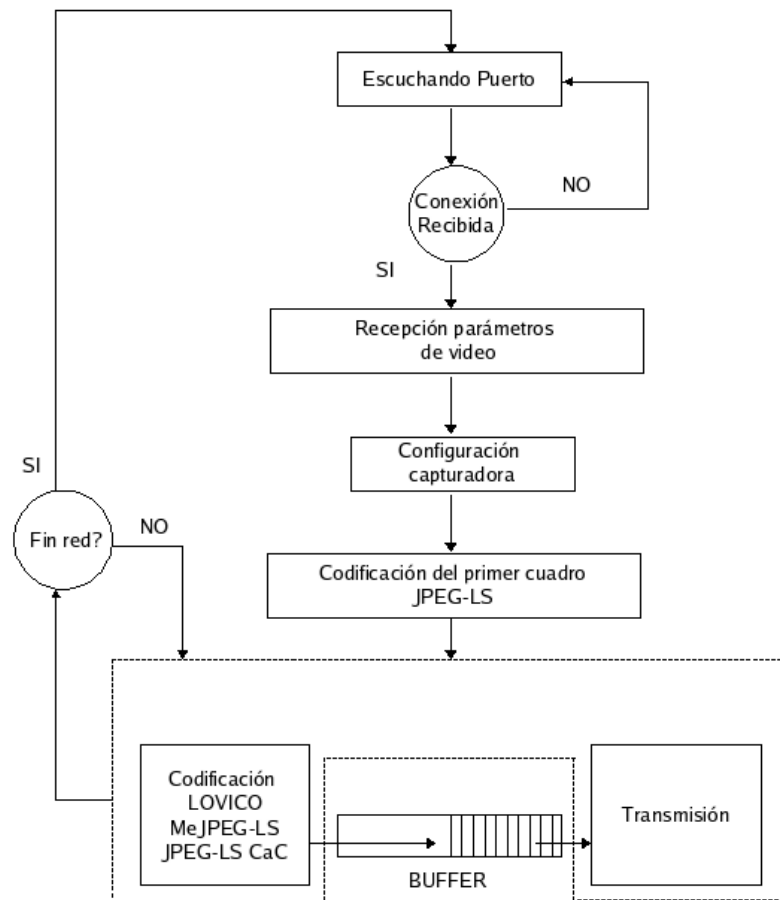


Figura 8.2: Diagrama de flujo del servidor

8.2. Software Implementado

El software desarrollado se compone de dos aplicaciones, una para cada extremo del sistema. Es así que se tiene un *software servidor* encargado de la captura, codificación y transmisión del video, y otro *software cliente* cumpliendo las funciones de recepción, decodificación y presentación del video.

En las siguientes subsecciones se detallan cada uno de ellos.

8.2.1. Software servidor

El software servidor es el encargado de la captura, compresión y transmisión del video a solicitud del cliente remoto. Dicho software fue desarrollado íntegramente en C++ y se emplearon bibliotecas estándar o de lo contrario ampliamente conocidas en el área. Un diagrama de flujo de este servidor se puede apreciar en la figura 8.2.

En una primera instancia se crea un descriptor de socket que especifica el tipo de protocolo de comunicación (TCP sobre IPv4) mediante la función implementada “StreamIPv4” y se le asigna el puerto 5005 mediante la función “mi_bind”. Luego de realizada la asignación del puerto, mediante la función “mi_Listen” se convierte al socket sin conexión a uno pasivo, indicando

al kernel que puede aceptar llamadas de conexión direccionadas a este socket. De este modo se establecen las condiciones necesarias para esperar una conexión lo que se realiza efectivamente con la función “`aceptar_conex`”. Esta función se bloquea en espera de una conexión TCP por parte del cliente.

Para una descripción del software de red utilizado se recomienda ver el capítulo A.

Al recibir dicha conexión se sale del bloqueo, se configura el socket en modo de *no bloqueo* y se reciben los parámetros del video a capturar. Estos parámetros son tamaño del cuadro (CIF o QCIF), canal a codificar (R,G,B o Y) y cantidad de cuadros a ser capturados.

Con los datos de tamaño de cuadro y canal se configura el bloque de captura. El software desarrollado en este bloque se basa en el API de Video for Linux 2 (ver sección 7.1). Utilizando la función “`leer_cuadro`” ó “`leer_cuadro_gris`” se captura un cuadro adquirido por la cámara de video y se codifica JPEG-LS (ver sección 4.2 para tener una descripción detallada del proceso). Cabe destacar que la librería de imágenes empleada en todo el software desarrollado es *CImg*. Dicha librería es la encargada de cargar, guardar y procesar las imágenes utilizadas en todo el proceso.

La cadena de bits producida por el codificador JPEG-LS es almacenada en un buffer a medida que es generada. Este buffer es miembro de una clase creada especialmente para el manejo y control a nivel de bits en la aplicación, y es el encargado de almacenar cada bit generado por el proceso de codificación del sistema.

Luego de codificado el primer cuadro se procede a la codificación del resto del video y paralelamente se inicia el proceso de transmisión en la red.

El proceso paralelo se ejecuta utilizando *hilos POSIX* de forma tal de poder compartir memoria entre los procesos de codificación y transmisión en la red. Es fundamental para la aplicación que el contenido del buffer sea conocido por el codificador y el transmisor. El codificador llena el buffer, mientras que el transmisor los envía en el mismo orden que van ingresando.

Finalizada la codificación del primer cuadro, el resto del video puede codificarse mediante 3 algoritmos: JPEG-LS cuadro a cuadro, MeJPEG-LS y LOVICO ya analizados en capítulos anteriores.

El proceso de red transmite, de existir bytes disponibles, la cadena de bits contenida en el buffer. Esta transmisión es sin bloqueo y culmina con el envío del último byte contenido en el buffer perteneciente al último cuadro codificado del video, tras lo cual se cierra la conexión y se termina el proceso de transmisión. Al terminarse este proceso, lo cual implica el fin del proceso de codificación, el software servidor vacía el buffer, reinicia las variables y vuelve al estado de espera de una nueva conexión.

8.2.2. Software cliente

El software cliente es el encargado de enviar los parámetros de captura al servidor, decodificar el stream de video recibido y presentarlo en algún dispositivo de salida de video.

La figura 8.3 presenta el diagrama de flujo del software cliente. Este software cliente permite al usuario ingresar los parámetros deseados para la captura. Para ello dispone de una interfaz gráfica que permite el ingreso de la cantidad de cuadros a capturar, su formato (CIF o QCIF) y el canal a recibir, además de la dirección IP y puerto del servidor de video.

La librería empleada para la realización de la interfaz gráfica fue *Qt* y su elección se debió principalmente a la amplia información disponible sobre sus funciones, además tener un buen compromiso entre facilidad de uso y performance.

Luego de seleccionados los parámetros, los cuales son ingresados en modo texto o mediante un

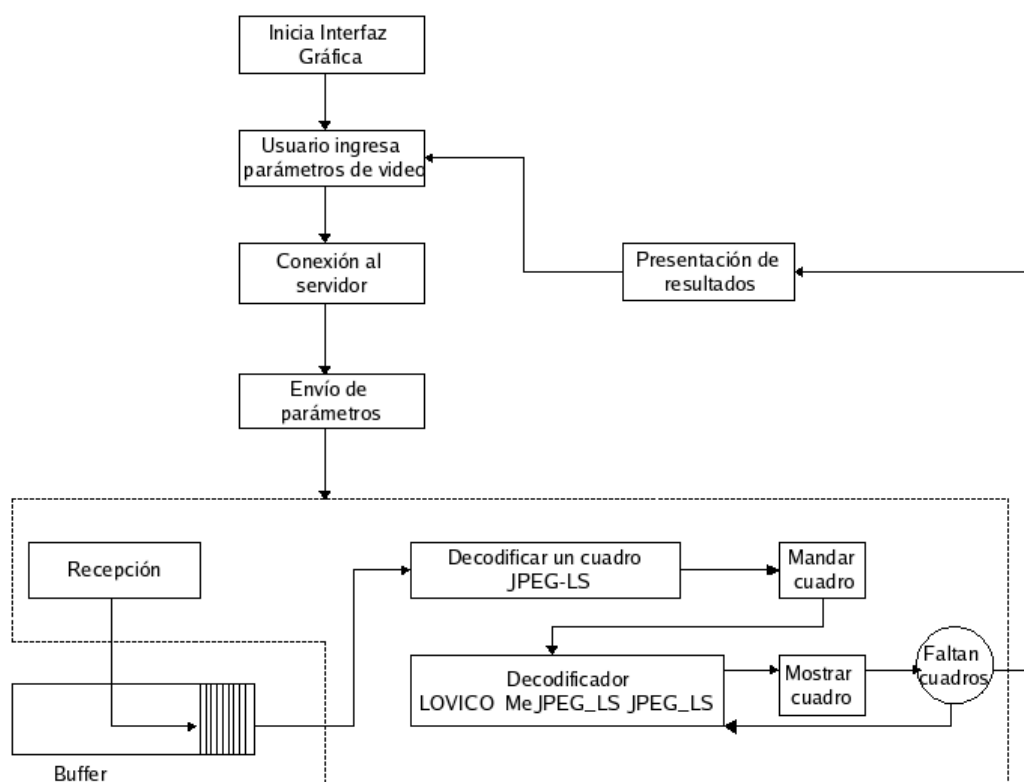


Figura 8.3: Diagrama de flujo del cliente.

menú desplegable, el usuario dispone de un botón para iniciar la conexión al servidor. Se crea tras este paso un socket TCP a través de la función “StreamIPv4” y se procede a conectar al servidor con la IP y puerto ingresados.

Tras este paso se configura el socket para no bloquear y de este modo mejorar el desempeño del sistema. Un socket que no bloquea permite retornar inmediatamente tras hacer lecturas o escrituras al buffer TCP. Por ejemplo, si en el buffer hay disponibles 5 bytes e intento una lectura de 7 bytes un socket configurado para no bloquear me devolverá inmediatamente esos 5 bytes, mientras que si está configurado para bloquear, esperará a tener 7 bytes disponibles para retornar el control.

Tras configurar el socket al estado de no bloqueo se envían los parámetros al servidor y se comienza con el proceso de decodificación previo al lanzamiento en paralelo, mediante la utili-

zación de hilos POSIX, de un proceso de recepción de datos de la red y almacenamiento en un buffer. Este buffer será parte, al igual que en el caso del software servidor, de una clase dedicada al manejo y control a nivel de bits en la aplicación.

El proceso de decodificación, de existir datos disponibles en el buffer, decodifica un primer cuadro utilizando el algoritmo JPEG-LS. Existen datos disponibles si el proceso de red ingresó al buffer una cantidad de bits mayor o igual a lo que el codificador solicita. Tras decodificar despliega el cuadro en pantalla y sigue con el proceso de decodificación optando entre tres decodificadores posibles (JPEG-LS cuadro a cuadro, MeJPEG-LS ó LOVICO). Luego de la decodificación de cada cuadro se realiza el despliegue en pantalla del mismo. Esto se realiza utilizando la biblioteca *CImg*.

Este proceso finaliza cuando se decodifican todos los cuadros requeridos, tras lo cual se presentan los resultados numéricos de la transferencia como ser cuadros por segundo, nivel de compresión en porcentaje y en bpp, y ancho de banda utilizado en promedio. Tras este paso se retorna el control al usuario.

El proceso paralelo de recepción de datos se encuentra en un bucle capturando los datos enviados en la red por el servidor y almacenando los mismos en el buffer mencionado anteriormente.

8.3. Relevamiento del Sistema

En esta sección se presentan una serie de relevamientos sobre el funcionamiento del sistema implementado. Los mismos tienen como objetivo caracterizar al sistema de acuerdo a lo especificado en los objetivos del proyecto.

El primer relevamiento consiste en excluir la etapa de captura de video, utilizando el conjunto de videos de prueba (ver sección 3.2.1) como videos a transmitir por el servidor. De este modo se tiene la ventaja de que los tres algoritmos utilizados (JPEG-LS cuadro a cuadro, MeJPEG-LS y LOVICO) codificarán exactamente la misma secuencia, independizándose de las variaciones que introducen la capturadora de video, la cámara y el ambiente que está siendo “grabado”. Además, dado que los algoritmos implementados en el sistema han sido aplicados anteriormente al conjunto de videos de prueba, esto permitirá tener una visión general del desempeño del módulo de streaming exclusivamente.

El segundo, consiste en relevar el sistema completo, incluyendo la etapa de captura. Con tal fin se utilizaron tres secuencias de videos caseros almacenadas en una cinta de video analógica, las cuales fueron digitalizadas por la capturadora de video que emplea el sistema.

8.3.1. Relevamiento Sobre el Conjunto de Videos de Prueba

Como se dijo anteriormente, este consiste en poner en funcionamiento el sistema haciendo que el cliente solicite el tamaño de cuadro y la cantidad de cuadros del video de prueba que el servidor ha cargado en memoria. Esta prueba se realizó para el canal verde, y los algoritmos de compresión empleados fueron los tres implementados dentro del sistema, es decir JPEG-LS cuadro a cuadro, MeJPEG-LS y LOVICO.

Es importante destacar que para efectuar la evaluación anterior los videos de prueba completos deben ser cargados en memoria, lo que afectará el desempeño del sistema, teniendo en cuenta su tamaño (por ejemplo, para un video de prueba de 300 cuadros tamaño CIF se tiene que este ocupa sin compresión $300 \text{ cuadros} \times 288 \text{ filas} \times 325 \text{ columnas} \times 8 \text{ bpp} = 243302400 \text{ bits}$).

En la tabla 8.2 se presentan los resultados obtenidos. Allí se muestra la cantidad de cuadros

Secuencia de Video	Compresión en %	fps	Ancho de banda medio (Mbps)
Salesman	42.12	10.22	4.80
Mobile and Calendar	27.24	9.76	5.76
Container	45.32	10.39	4.61
Tempete	37.77	10.15	5.12
Kitchgrass	41.55	11.06	5.25
Sean	50.35	10.55	4.26
Silent	43.79	10.52	4.79
Foreman	46.33	10.42	4.53
Promedio	41.81	10.38	4.89

Tabla 8.2: Nivel de compresión, cantidad de cuadros por segundo que el sistema puede procesar y ancho de banda consumido, utilizando JPEG-LS cuadro a cuadro para los distintos videos de prueba.

Secuencia de Video	Compresión %	fps	Ancho de banda promedio (Mbps)
Salesman	42.24	10.49	4.91
Mobile and Calendar	27.60	10.04	5.89
Container	45.43	10.71	4.74
Tempete	37.87	10.39	5.23
Kitchgrass	41.57	11.42	5.47
Sean	50.61	10.86	4.35
Silent	44.02	10.88	4.94
Foreman	46.61	10.76	4.65
Promedio	41.99	10.69	5.02

Tabla 8.3: Nivel de compresión, cantidad de cuadros por segundo que el sistema puede procesar y ancho de banda consumido, utilizando MeJPEG-LS para los distintos videos de prueba.

por segundo que el sistema puede procesar y el ancho de banda medio empleado, utilizando JPEG-LS cuadro a cuadro como algoritmo de compresión.

Este procedimiento se repite en las tablas 8.3 y 8.4 utilizando MeJPEG-LS y LOVICO como métodos para la compresión y decompresión.

La performance del sistema para el conjunto de los videos de prueba y los algoritmos JPEG-LS cuadro a cuadro, MeJPEG-LS y LOVICO se muestra en la tabla 8.5.

En la tabla 8.5 puede observarse que mientras que la reducción de la cantidad de cuadros procesados por segundo al pasar de MeJPEG-LS a LOVICO es de un 38.73 %, la reducción en el ancho de banda requerido es de un 48.01 %. Es decir, la reducción en el ancho de banda requerido no solo se explica por procesar menos cuadros por segundo, sino por utilizar un codificador que logra un mayor nivel de compresión, del orden un 10 %.

Ahora bien, en la tabla 8.6 se muestran los cuadros procesados por segundo para los algoritmos JPEG-LS, MeJPEG-LS y LOVICO aplicados al conjunto de videos de prueba *fuera del sistema*.

Se observa que al utilizarlos dentro del sistema, la cantidad de cuadros procesados por segundo cae un 70, 14 % para LOVICO, un 67.29 % para MeJPEG-LS y un 68.68 % para JPEG-LS cuadro a cuadro.

Secuencia de Video	Compresión %	fps	Ancho de banda promedio (Mbps)
Salesman	50.30	6.36	2.56
Mobile and Calendar	32.71	6.01	3.28
Container	59.84	6.86	2.23
Tempete	43.52	6.17	2.83
Kitchgrass	44.66	6.66	2.99
Sean	62.64	6.89	2.09
Silent	60.73	6.83	2.18
Foreman	48.87	6.60	2.74
Promedio	50.41	6.55	2.61

Tabla 8.4: Nivel de compresión, cantidad de cuadros por segundo que el sistema puede procesar y ancho de banda consumido, utilizando LOVICO para los distintos videos de prueba.

Algoritmo	Compresión %	fps	Ancho de banda promedio (Mbps)
JPEG-LS CaC ¹	41.81	10.38	4.89
MeJPEG-LS	41.99	10.69	5.02
LOVICO	50.41	6.55	2.61

Tabla 8.5: Valores medios de nivel de compresión, cantidad de cuadros por segundo que el sistema puede procesar y ancho de banda consumido par los métodos de codificación implementados en el sistema

Algoritmo	fps
MeJPEG-LS	33.15
JPEG-LS CaC	32.69
LOVICO	21.94

Tabla 8.6: Cuadros procesados por segundo.

Secuencia de Video	Cantidad de Cuadros	fps	Ancho de banda medio (Mbps)
BOOM (CIF)	300	12.60	4.76
BOOM (QCIF)	301	24.28	1.97
EIFFEL (CIF)	80	12.91	4.63
VENTILADOR (QCIF)	301	25.28	2.06

Tabla 8.7: Cantidad de cuadros por segundo y ancho de banda consumido por JPEG-LS cuadro a cuadro para los videos mencionados utilizando el canal Y.

Secuencia de Video	Cantidad de Cuadros	fps	Ancho de banda medio (Mbps)
BOOM (CIF)	300	12.14	5.19
BOOM (QCIF)	301	25.39	2.69
EIFFEL (CIF)	80	12.40	5.28
VENTILADOR (QCIF)	301	25.32	2.61

Tabla 8.8: Cantidad de cuadros por segundo y ancho de banda consumido por JPEG-LS cuadro a cuadro para los videos mencionados utilizando el canal verde.

8.3.2. Relevamiento Sobre Videos Caseros

Este relevamiento consiste en utilizar el sistema incluyendo la etapa de captura, sobre un pequeño conjunto de videos generados por los integrantes de este grupo proyecto de fin de carrera. Los mismos llevan los nombres VENTILADOR, EIFFEL y BOOM.

Se relevó el funcionamiento del sistema utilizando las secuencias BOOM y EIFFEL en formato CIF y en formato QCIF la misma BOOM y VENTILADOR.

En todos los casos se emplearon los algoritmos de codificación JPEG-LS cuadro a cuadro, MeJPEG-LS y LOVICO, codificando el canal verde y la luminancia (o canal Y).

En las tablas 8.7 y 8.8 se detallan los resultados numéricos para JPEG-LS cuadro a cuadro aplicado a la luminancia y al canal verde de los videos caseros, respectivamente. En las tablas 8.9 y 8.10 se hace lo mismo para MeJPEG-LS, y en las tablas 8.11 y 8.12 para LOVICO.

En las tablas 8.13 y 8.14 se presenta un resumen del desempeño promedio del sistema empleando los diferentes algoritmos de codificación.

Secuencia de Video	Cantidad de Cuadros	fps	Ancho de banda medio (Mbps)
BOOM (CIF)	300	12.61	4.80
BOOM (QCIF)	301	25.31	2.03
EIFFEL (CIF)	80	12.92	4.68
VENTILADOR (QCIF)	301	25.32	2.02

Tabla 8.9: Cantidad de cuadros por segundo y ancho de banda consumido por MeJPEG-LS para los videos mencionados utilizando el canal Y.

Secuencia de Video	Cantidad de Cuadros	FPS	Ancho de banda promedio (Mbps)
BOOM (CIF)	300	12.53	5.36
BOOM (QCIF)	301	25.22	2.66
EIFFEL (CIF)	80	12.49	5.23
VENTILADOR (QCIF)	301	25.27	2.57

Tabla 8.10: Cantidad de cuadros por segundo y ancho de banda consumido por MeJPEG-LS para los videos mencionados utilizando el canal verde.

Secuencia de Video	Cantidad de Cuadros	FPS	Ancho de banda promedio (Mbps)
BOOM (CIF)	300	8.39	3.16
BOOM (QCIF)	301	25.36	1.96
EIFFEL (CIF)	80	8.54	3.18
VENTILADOR (QCIF)	201	25.51	1.98

Tabla 8.11: Cantidad de cuadros por segundo y ancho de banda consumido por LOVICO para los videos mencionados utilizando el canal Y.

Secuencia de Video	Cantidad de Cuadros	FPS	Ancho de banda promedio (Mbps)
BOOM (CIF)	300	8.39	4.21
BOOM (QCIF)	301	13.71	1.38
EIFFEL (CIF)	80	8.58	3.77
VENTILADOR (QCIF)	201	12.67	1.30

Tabla 8.12: Cantidad de cuadros por segundo y ancho de banda consumido por LOVICO para los videos mencionados utilizando el canal verde.

Algoritmo	Canal Verde	Luminancia
JPEG-LS CaC	12.27	12.76
MeJPEG-LS	12.51	12.77
LOVICO	8.49	8.47

Tabla 8.13: Cantidad de cuadros por segundo promedio generados por los distintos algoritmos en los videos caseros formato CIF para el canal verde y la luminancia.

Algoritmo	Canal Verde	Luminancia
JPEG-LS CaC	25.36	24.78
MeJPEG-LS	25.25	25.32
LOVICO	13.19	25.44

Tabla 8.14: Cantidad de cuadros por segundo promedio generados por los distintos algoritmos en los videos caseros formato QCIF para el canal verde y la luminancia.

8.4. Conclusiones

En esta sección se presentan las conclusiones del sistema completo. Es importante destacar que se logró una implementación funcional del sistema en su conjunto como una unión de tres grandes bloques: codificación, captura y streaming.

El relevamiento realizado sobre el sistema de streaming permitió evaluar el desempeño del mismo. Si comparamos la tasa de cuadros por segundo procesada en el sistema con la procesada en cada host individualmente fuera del mismo, se observa una caída en la capacidad de cuadros por segundo para los videos de prueba del 68,7 % promedio entre JPEG-LS CaC, MeJPEG-LS y LOVICO respecto al host cliente, mientras que esta cifra se reduce al 41.26 % respecto al host servidor. De aquí puede deducirse que en cierta medida es el hardware y en particular el host servidor el que está limitando la máxima tasa de cuadros. Por lo tanto, de haber contado con dos host con la misma capacidad de procesamiento que el cliente, el desempeño se hubiera incrementado.

Estas caídas en el desempeño pueden deberse al extensivo uso de memoria que requiere el almacenado y codificación de las secuencias de video. Al evaluar el sistema con captura este requerimiento extensivo de memoria deja de ser tal ya que no es necesario mantener toda la secuencia en memoria, cada cuadro es codificado a medida que se captura.

Para el sistema con captura, si bien no son comparables las secuencias de video estándar con las generadas por los integrantes del proyecto, se obtiene que la tasa de cuadros por segundo aumenta para los videos en formato CIF procesando el canal verde. Este aumento es mayor en el caso de LOVICO donde alcanza el 29.6 %, mientras que para JPEG-LS CaC y MeJPEG-LS es del 18,21 % y el 17.03 % respectivamente. Dado que los requerimientos de memoria de LOVICO son mayores al de los otros dos, esto explicaría el mayor aumento de desempeño para este algoritmo cuando se posee una mejor administración del recurso memoria.

Si bien el host servidor limita la capacidad máxima de cuadros por segundo a desarrollar, no explica completamente la baja tasa desarrollada. La diferencia de desempeño puede ser asociada a la implementación no optimizada del streaming realizado, que es donde deberán realizarse mayores esfuerzos a futuro.

Un hecho importante a destacar es que para el sistema completo empleando un formato de video QCIF se alcanzaron tasas de cuadros por segundo en el orden de 25fps, cantidad limitada por la capacidad de la capturadora de video. De este modo se alcanzó la meta de producir un stream de video sin pérdidas en una red LAN en tiempo real.

También es relevante destacar que la capacidad de procesamiento del sistema en su conjunto (medida en cuadros por segundo) resultó en el peor de los casos, mayor a la obtenida por algoritmos como I-CALIC, M-CALIC y LOPT-3D.

9 CONCLUSIONES GLOBALES

En este capítulo se exponen las conclusiones del proyecto en su globalidad, más allá del software implementado o el hardware utilizado para el sistema de adquisición compresión sin pérdidas y transmisión de video.

Como primer punto queremos resaltar el haber conseguido implementar el sistema en sí, y que este funcione como tal. El mismo involucra el control de un periférico de captura de video, algoritmos de compresión y decompresión de video, establecimiento de diferentes conexiones TCP y otros componentes igual de heterogéneos. El haber logrado que estos trabajen juntos en forma eficaz requirió el estudio de temas tan diversos como algoritmos de compresión sin pérdidas de imágenes y video, hasta los sockets de Berkeley, pasando por los módulos del núcleo del sistema operativo Linux. Este proceso fue naturalmente provechoso para los integrantes de este grupo de proyecto de fin de carrera.

Como se refleja en esta documentación la mayor parte del tiempo empleado para el desarrollo del proyecto fue dedicado a las etapas de estudio del estado del arte en métodos de compresión de video sin pérdidas, el desarrollo de métodos alternativos y la implementación de los mismos. Se trata de un campo relativamente nuevo y sin una referencia clara, estandarizada y conocida. Sin embargo se consiguió información suficiente como para responder las preguntas planteadas en los objetivos del proyecto, mostradas de nuevo aquí:

- ¿Cuáles son los niveles de compresión alcanzados por los métodos actuales de codificación de video sin pérdidas?
- ¿Cuál es el costo computacional asociado a cada método? (Cuantificado a través de los retardos introducidos en los procesos de compresión y decompresión)

En este punto fue de especial importancia la comunicación establecida con *Elias S. G. Carotti*, coautor de [2] y [14], quien proporcionó detalles sobre la evaluación de los métodos I-CALIC, M-CALIC y LOPT-3D sin los cuales la correcta comparación con los métodos implementados no hubiera sido posible.

Sin lugar a dudas el principal aporte de este grupo fue en esta área. Además del estudio del estado del arte en métodos para la compresión de video sin pérdidas se obtuvo una implementación propia de un método de compresión de imágenes sin pérdidas (LOCO-I/JPEG-LS) en base al estándar que lo define, lo que constituye un primer logro. Requirió mucho tiempo comprenderlo como para implementarlo e implementarlo correctamente. Creemos que fue una inversión justificada, ya que abrió la posibilidad de realizar modificaciones al mismo orientadas a tratar de explotar de algún modo la redundancia temporal presente en las secuencias de video. Esto condujo a estudiar una variedad de opciones para lograr tal fin, destacándose la compensación de movimiento, que constituye un campo fundamental en la compresión de video con pérdidas.

Estas modificaciones concluyeron con LOVICO, un método para la compresión de video sin pérdidas original, competitivo con los máximos niveles de compresión alcanzados por los

algoritmos del marco de trabajo propuesto, pero bastante menos complejo, o lo que es lo mismo más rápido. En el resto de los algoritmos implementados o encontrados en esta área de estudio se observa un compromiso muy rígido entre la cantidad de cuadros por segundo que el método puede procesar y el nivel de compresión alcanzado. Esto significa que se obtienen altos niveles de compresión procesando muy pocos cuadros por segundo, o que se procesan muchos cuadros por segundo logrando bajos niveles de compresión. LOVICO rompe en buena medida con estas restricciones, logrando un buen compromiso entre cantidad de cuadros procesados por segundo y nivel de compresión.

LOVICO posee además otras propiedades interesante como el hecho de estar compuesto por elementos que se encuentran en LOCO-I/JPEG-LS. De esta forma busca “heredar” la solidez del método LOCO-I, la cual se basa en un vínculo muy preciso entre una implementación simple y una justificación teórica clara. Dicho de otro modo en varios puntos de LOCO-I la implementación se “mezcla” con la bases teóricas a través de largas cadenas de simplificaciones y suposiciones. Además, este hecho podría facilitar su implementación en sistemas ya instalados que utilizan LOCO-I/JPEG-LS.

Otra característica interesante encontrada fue que en el peor caso, LOVICO se comporta (respecto al nivel de compresión) como JPEG-LS cuadro a cuadro. Saber que es lo que se puede esperar de un método nuevo para la compresión sin pérdidas de video en función de un estándar ampliamente conocido, es una ventaja a la hora de su integración en diversos sistemas.

Un resultado evidente de esta etapa es que queda mucho por hacer en el campo de la compresión de video sin pérdidas. La cantidad creciente de aplicaciones que requieren este tipo de procedimientos seguramente orientará los esfuerzos de investigación en esta dirección.

Una etapa corta en este proyecto, pero también exitosa, fue la del manejo de la capturadora de video utilizada desde un código C++ empleando la API “Video para Linux Dos (V4L2)”. Requerí nociones sobre la configuración de hardware en computadores con sistema operativo Linux, las cuales se adquirieron. Posteriormente en base a la documentación de V4L2 se logró sin mayores inconvenientes manipular el dispositivo de captura según nuestras necesidades.

El lo que hace a la comunicación entre los extremos del sistema, se utilizó el protocolo TCP directamente a través de los sockets de Berkeley y sus primitivas. No se utilizaron bibliotecas de más alto nivel. Este punto en particular creemos que ha sido favorable en algunos aspectos y desfavorable en otros, ya que si bien la comunicación se logró establecer en forma correcta, esta no está optimizada al día de hoy.

Además, si bien son muy provechosos los conocimientos adquiridos en esta área, necesarios para lograr la implementación, también fueron costosos en términos de tiempo requerido para asimilar los mismos.

Es en esta área donde pensamos que queda más por hacer. Creemos que el siguiente paso a realizar debería ser utilizar UDP como protocolo de transporte y restringir TCP al control.

Otro pendiente algo más refinado debería ser la capacidad de cliente o servidor de cambiar el método de codificación de acuerdo al estado del enlace, enviando más cuadros por segundo codificados con métodos de menor compresión si este está holgado en su capacidad, o en caso contrario enviando menos cuadros por segundo con métodos de mayor compresión si la capacidad disminuye. Esto requiere desde luego de una aplicación más definida para el sistema.

Pero no todos son puntos pendientes en este bloque del sistema, existen varios puntos muy positivos. Uno de ellos, algo básico, es que se logró la comunicación por streaming entre servidor y cliente de videos comprimidos sin pérdidas utilizando algoritmos de compresión (originales y estándar) implementados por los integrantes de este grupo. Además, como dato simbólico, la

capacidad de procesamiento de cuadros por segundo del sistema implementado completo para el algoritmo más lento es aún mayor que la capacidad de procesamiento del más rápido de los algoritmos del marco de trabajo propuesto, considerandolo a este en forma aislada. Esto habla claramente de una implementación quizás no optimizada, pero si razonable.

Para lograr este resultado fue de suma importancia la implementación de un procesamiento paralelo de las tareas entre los bloques de codificación y streaming.

Tal vez sea prudente destacar que al inicio de este proyecto de fin de carrera ninguno de los integrantes del mismo tenía experiencia alguna en programación con C++ o desarrollo de software sobre Linux. Al finalizar el proyecto *todo* el software implementado correspondiente al sistema fue programado por los integrantes del mismo. Las excepciones son la excelente biblioteca para el procesamiento de imágenes CImg y la API V4L2. Sobre CImg debemos resaltar su funcionalidad, su fácil utilización y lo completa de la misma. Además debemos destacar la disponibilidad de su creador David Tschumperlé, para responder todo tipo de consultas y brindar soporte a los usuarios de la misma.

Debe tenerse en cuenta además que el programar métodos para la compresión de video (donde el tamaño de los datos involucrados es enorme) con el objetivo de evaluar su performance en un sistema completo, de punta a punta impone en cada decisión severas restricciones sobre el manejo de recursos que no facilitan la tarea. Aún así se lograron implementar algoritmos de compresión de video que logran interactuar con los otros componentes del sistema sin problema alguno.

Otro pendiente dado lo acotado del tiempo de desarrollo para un sistema de semejante complejidad, es una evaluación más detallada del sistema como se deseó en un principio y como si se realizó para los algoritmos de compresión de video propuestos o LOCO-I/JPEG-LS. Sin embargo, pudo ser relevada la tasa de cuadros por segundo y ancho de banda medio requerido para distintos tamaños de cuadro y canales utilizando secuencias de video estándar, permitiendo de este modo una comparación con otros sistemas similares. Creemos que el detalle que falte en el relevamiento del sistema completo puede ser fácilmente obtenido.

Por último recordamos un objetivo cumplido no mencionado en las conclusiones de ningún capítulo hasta ahora, y es que todo el software utilizado en este proyecto de fin de carrera es *software libre*, desde el entorno de programación utilizado hasta el editor con que se generó esta documentación, pasando por bibliotecas de procesamiento de imágenes, compiladores, debuggers, etc.

Nuestra experiencia con el mismo ha sido variada, hemos encontrado software de nuestro agrado y útil a nuestros intereses así como algunas complicaciones con determinados programas o bibliotecas. Sin embargo es un hecho que dichas complicaciones pudieron ser solucionadas utilizando otras alternativas libres.

En resumen, se cumplieron todos los objetivos planteados en el plan del proyecto, salvo un relevamiento más detallado de las características del sistema implementado. Se consiguió en el camino un algoritmo original de compresión de video sin pérdidas (LOVICO) que presenta, en comparación con los demás algoritmos considerados, una excelente relación *nivel de compresión vs cuadros procesados por segundo*.

Bibliografía

- [1] David Salomon. Data compression, the complete reference, 3rd edition. *Department of Computer Science, California State University, Northridge, USA*, 2004.
- [2] Elias S. G. Carotti and Juan Carlos De Martin. Motion-compensated lossless video coding in the calic framework. *DAUIN / IEIIT-CNR Politecnico di Torino, Torino, Italy*, 2005.
- [3] ISO/IEC-13818. ISO/IEC-11172. Mpeg - moving picture experts group. *MPEG-1, MPEG-2*, 1988.
- [4] IEEE. Carrier sense multiple access with collision detection (csma/cd) access method and physical layer specification. *IEEE 802.3u*, 1995.
- [5] Marcelo J. Weinberger Gadiel Seroussi and Guillermo Sapiro. The loco-i lossless image compression algorithm: Principles and standardization into jpeg-ls. *Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis and Hewlett-Packard Laboratories, Palo Alto, California*, 1996.
- [6] International Telecommunication Union. Lossless and near-lossless compression of continuous-tone still images-baseline. *Series T: Terminals for Telematic Services, Recommendation T.87*, 06/1998.
- [7] Xiaolin Wu. Lossless compression of continuous-tone images via context selection, quantization, and modeling. *Conditionally accepted by IEEE Trans. on Image Processing, and was presented in parts at the 1995 and 1996 IEEE Data Compression Conferences*, 5/1995.
- [8] Andreas E. Savakis. Evaluation of lossless compression methods for gray scale document images. *Department of Computer Engineering, Rochester Institute of Technology, Rochester, New York, USA*, 6/2000.
- [9] Marcelo J. Weinberger Gadiel Seroussi and Guillermo Sapiro. Loco-i: A low complexity, context-based, lossless image compression algorithm. *Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis and Hewlett-Packard Laboratories*, 1996.
- [10] International Telecommunication Union. Sistemas de television convencionales. *ITU-R: Radiocommunication Sector Recomendacion ITU-R BT.470-6*, 1998.
- [11] International Telecommunication Union. Conventional analog television systems. *ITU-R: Radiocommunication Sector Recomendacion ITU-R BT.470-7*, 1998.
- [12] International Telecommunication Union. Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios. *ITU-R: Radiocommunication Sector Recomendacion ITU-R BT.601-5*, 1995.

- [13] Claude E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, vol.27,pp.379-423,623-656, 1948.
- [14] Elias S. G. Carotti and Juan Carlos De Martin. Lossless video coding using multi-frame motion compensation. *DAUIN / IEIIT-CNR Politecnico di Torino, Torino, Italy*, 2005.
- [15] G.A.Mian D.Brunello, G.Calvagno and R.Rinaldo. Lossless video coding using optimal 3d prediction. *Dipartimento di Elettronica e Informatica, Padova, Italy*, 2005.
- [16] X. Wu and N. Memon. Context-based, adaptive, lossless image coding. *IEEE Transactions on Communications*, vol 45, no 4, pp. 437-444, 04/1997.
- [17] Mohammed Ghanbari. Standard codecs: Image compression to advanced video coding. *IEE Telecommunication Series*, 2003.
- [18] Solomon Wolf Golomb. Run-length encodings. *IEEE Transactions on Information Theory*, 1966.
- [19] Solomon Wolf Golomb. *IEEE Transactions on Information Theory*, 1966.
- [20] Federico Lecumberry Adriana Piazza, Flavio Caldeiro. Larunbat codificación de video utilizando técnicas de cuantificación vectorial. *Proyecto de fin de carrera, Instituto de Ingeniería Eléctrica, Facultad de Ingeniería, Universidad de la República, Montevideo Uruguay*, 12/2000.
- [21] Monson H. Hayes. Statistical digital signal processing and modeling. *John Wiley and Sons*, 3/1996.
- [22] Susie J. Wee John G. Apostolopoulos, Wai-tian Tan. Video streaming: Concepts, algorithms and systems. *Hewlett-Packard Laboratories, Palo Alto, California*, 9/2002.
- [23] Daniel Rijo Sciara. Fundamentos de video streaming. *Monografía para el curso de postgrado: Codificación de Imágenes y Video, Instituto de Ingeniería Eléctrica-Facultad de Ingeniería-Montevideo, Uruguay*, 12/2004.
- [24] S.Floyd and K.Fall. Promoting the use of end-to-end congestion control in the internet. *IEEE/ACM Trans. on Networking*, vol. 7, no. 4, pp.458-472, 8/1999.
- [25] Kang Li Charles Krasic and Jonathan Walpole. The case for streaming multimedia with tcp. *Oregon Graduate Institute, Beaverton, Oregon, USA*.
- [26] Jonathan Walpole Molly H. Shor Kang Li, Charles Krasic and Calton Pu. The minimal buffering requirements of congestion controlled interactive multimedia applications. *Proceedings of the 8th International Workshop on Interactive Distributed Multimedia Systems*, pp.181-192, 2001.
- [27] Taehyun Kim and Mostafa H. Ammar. Reciver buffer requirement for video streaming over tcp. *Wireless and Mobile Systems Group, Freescale Semiconductor, Austin TX 78735, USA*, 2005.
- [28] David Tschumperlé. The cimg library reference manual. 2006.
- [29] Andrew M. Rudoffens Bill Fenner Andrew M. Rudoff Richard Stevens, Bill Fenner. Unix network programming volume 1, 3rd edition: The sockets networking. 2003.

A SOFTWARE DE RED

A.1. Los sockets de Berkeley

Se utilizó como interfaz de programación en lenguaje C++ los sockets de Berkeley. Estos conforman el estándar de facto en sockets de red, y por ello se seleccionó este tipo de interfaz. La referencia fundamental en este punto fue [29].

Los sockets de Berkeley permiten la comunicación entre hosts a través del concepto de socket. Un socket representa un extremo de la comunicación. En sistemas UNIX estos son descriptores de archivo, permitiendo llamadas al sistema como lectura y escritura para recibir o enviar datos en la red.

En los orígenes de Internet, las primeras computadoras en implementar sus protocolos fueron aquellas de la universidad de Berkeley. Dicha implementación tuvo lugar en una variante del sistema operativo Unix conocida como BSD Unix, dando origen a la primera especificación e implementación de sockets, también en Unix.

Un socket es un punto terminal, por el cual dos programas pueden intercambiar datos. Esta determinado por un número de puerto, un protocolo y una dirección IP. Las propiedades de un socket dependen de las características del protocolo en el que se implementan ya sea TCP o UDP. Los sockets implementados en TCP, son orientados a conexión, garantizan la transmisión de todos los datos sin errores ni omisiones y garantizan que todo segmento llegará a su destino en el mismo orden en que se ha transmitido. Por otro lado, el protocolo UDP es un protocolo no orientado a la conexión. Sólo se garantiza que si un mensaje llega, llegue bien. En ningún caso se garantiza que llegue o que lleguen todos los mensajes en el mismo orden que se mandaron.

A.1.1. Cliente-Servidor

El funcionamiento de la red es el de cliente-servidor, es decir, para poder poner en funcionamiento el sistema en forma correcta necesitamos que nuestro servidor se encuentre accesible.

Para el establecimiento de una estructura cliente-servidor mediante TCP, utilizando llamadas al sistema suministradas por los sockets de Berkeley, se siguen los siguientes pasos (entre paréntesis se nombran las funciones Berkeley que implementan dicho paso):

- SERVIDOR**
- Se crea el socket (**socket**)
 - Se asocia un puerto TCP al socket (**bind**)
 - Se prepara el socket para escuchar conexiones (**listen**)
 - Se bloquea el servidor en espera de conexiones (**accept**)
 - Una vez recibida una conexión se procede a la transferencia de datos (**read** y **write**)
 - Se cierra la conexión (**close**)

CLIENTE ■ Se crea el socket (**socket**)

- Se realiza la conexión con el servidor (**connect**)
- Una vez realiza la conexión se procede a la transferencia de datos (**read** y **write**)
- Se cierra la conexión (**close**)

Todos los sockets proporcionan una serie de primitivas, las cuales están orientadas a ser utilizadas por sus usuarios, de forma que se pueda establecer una comunicación determinada, y determinar las características de la misma.

La primitiva **socket** crea un punto terminal y le asigna espacio de tablas de la entidad de transporte. Los parámetros con que es llamada la primitiva especifican el formato de direccionamiento, el tipo de servicio deseado, y el protocolo. Una llamada a **socket** con éxito devuelve un descriptor de archivo, para ser utilizado en las siguientes llamadas.

Los sockets cuando son creados no tienen direcciones. Las direcciones se les asignan mediante la primitiva **bind**. Una vez que un servidor le asignado una dirección a un sockets los clientes remotos pueden conectarse a él.

Mediante una llamada a **listen**, se asigna espacio para poner en cola las llamadas entrantes en caso de que varios clientes intentan conectarse al mismo tiempo.

A continuación tenemos la primitiva **accept** la cual se utiliza para bloquear en espera de una conexión entrante.

Estas cuatro primeras primitivas son ejecutadas en el orden en que se nombraron por el servidor. Cuando llega un mensaje al servidor solicitando una conexión, el servidor crea un socket nuevo con las mismas propiedades que el original y devuelve un descriptor de archivo. En este punto el servidor se puede ramificar mediante un proceso (**fork**) o mediante un hilo para manejar la conexión en el socket nuevo y regresar a esperar la siguiente conexión en el socket original.

El cliente también debe ejecutar una llamada a **socket** para crear un socket, luego utiliza la primitiva **connect** que bloquea al invocador y comienza activamente la conexión. Al recibir la respuesta adecuada del servidor el cliente se desbloquea y se establece la conexión.

Las primitivas **send** y **receive** se utilizan para transmitir y recibir datos. Siendo utilizadas tanto por el cliente como por el servidor ya que la conexión es dúplex integral.

Por último tenemos la primitiva **close** utilizada para liberar la conexión. La liberación de los sockets es simétrica, esto quiere decir que se libera la conexión cuando ambos lados han ejecutado la primitiva **close**.

A.1.2. Código Implementado

A continuación se presentan las funciones de red implementadas en este proyecto.

CONFIGURA DIRECCION void configura_direccion(struct sockaddr_in* DIRECCION, short PUERTO, const char* IP)

Asigna a una dirección vacía su familia, puerto y dirección IP. Realiza dicha acción llenando los campos dentro de un estructurado tipo **sockaddr_in** que es una estructura de direcciones tipo IPv4.

La definición de la misma es la siguiente incluida en `<netinet/in.h>`:

```
struct in_addr {
in_addr_t s_addr; /* 32-bit dirección IPv4 */
};
```

```

struct sockaddr_in {
uint8_t sin_len; /* largo en bytes de la estructura (16) */
sa_family_t sin_family; /* AF_INET */
in_port_t sin_port; /* 16-bit TCP */
struct in_addr sin_addr;
char sin_zero[8]; /*sin uso */
};

```

Siendo:

- `in_addr` los 32 bits correspondientes a la dirección IPv4 con la ordenación de bytes de la red
- `uint8_t` entero sin signo de 8 bits
- `sa_family` la familia del protocolo. Puede tomar los siguientes valores (entre otros):
- `AF_INET` : para protocolo IPv4
- `AF_INET6`: para protocolo IPv6
- `AF_LOCAL`: para dominio UNIX
- `in_port` puerto TCP (para `AF_INET`), entero sin signo de 16 bits

STREAMIPV4 `int StreamIPv4();`

Retorna un descriptor de socket especificando el tipo de protocolo de comunicación TCP sobre IPv4. Devuelve `-1` en caso de error.

SETNONBLOCKING `int setNonblocking(int SOCKFD);`

Configura el socket en modo de no bloqueo.

MI_BIND `int mi_Bind(int SOCKFD,const struct sockaddr_in MI_DIRECCION);`

Asigna al socket `SOCKFD` (devuelto por la función `StreamIPv4()`) la dirección de protocolo. Útil para establecer un servidor en un puerto determinado, de otra forma este escucharía en un puerto previamente desconocido y posiblemente diferente en cada reinicio.

Si se especifica el valor 0 al puerto, el kernel seleccionará un puerto efímero para dicho socket, si se especifica el valor de 0 para la dirección el kernel seleccionará la dirección del host.

La selección del puerto deberá cumplir ciertos requisitos: debe ser mayor a 1023 para no quedar dentro del rango de puertos bien conocidos (ejemplo ftp, http, etc), debe ser mayor a 5000 para evitar conflicto con puertos efímeros de implementaciones derivadas de sockets Berkeley, menor que 49152 para evitar conflicto con el rango “correcto” de puertos efímeros, y debe evitar conflictos con puertos ya registrados. Retorna `-1` en caso de error.

MI_LISTEN `int mi_Listen(int SOCKFD,int BACKLOG);`

Convierte un socket `SOCKFD` sin conexión (devuelto por la función `StreamIPv4()`) a uno pasivo indicando que el kernel puede aceptar llamadas de conexión direccionadas a este socket. También establece el máximo de conexiones que este socket puede tener en cola. Retorna `-1` en caso de error.

ACEPTAR_CONEX `int aceptar_conex(int SOCKFD, struct sockaddr *DIR_CLIEN-TE);`

Es llamada por un servidor TCP para que devuelva un descriptor de socket de una conexión nueva que se encontraba en cola. Si la cola está vacía el proceso queda en modo "sleep".

SOCKFD es el descriptor de socket que devuelve `StreamIPv4()`.

La función retorna hasta tres valores: un entero que representa un nuevo descriptor de socket para esta nueva conexión (-1 en caso de error), la dirección de protocolo del cliente a través del puntero a `DIR_CLIEN-TE` y el tamaño de dicha dirección.

CONECTARSE `int conectarse(int SOCKFD, const struct sockaddr *DIR_SERVIDOR);`

Usada por un cliente TCP para establecer una conexión con un servidor TCP.

Inicia el "three way handshake" de TCP.

SOCKFD es el descriptor de socket que devuelve `StreamIPv4()`.

MI_WRITE `int mi_write(int SOCKFD, const void *buffer, int BYTES);`

Escribe la cantidad de BYTES indicada a un descriptor de socket (SOCKFD). Devuelve la cantidad de bytes escritos o -1 en caso de error.

MI_READ `int mi_read(int SOCKFD, const void *buff, int BYTES);`

Lee una cantidad de BYTES indicada a un descriptor de socket (SOCKFD). Devuelve la cantidad de bytes leídos o -1 en caso de error.

MI_WRITE_NB `int mi_write_nb(int SOCKFD, const void *buffer, int BYTES);`

Escribe la cantidad de bytes indicada BYTES a un descriptor de socket (SOCKFD), devuelve la cantidad de bytes escritos o -1 en caso de error. Esta función es utilizada en caso se transmisiones sin bloqueo.

MI_READ_NB `int mi_read_nb(int SOCKFD, const void *buff)`

Lee de la red la mayor cantidad de bytes posibles que permitan no bloquear el procesamiento y los asigna a *buff.

CERRAR_SOCKET `int cerrar_socket(int SOCKFD);`

Cierra el socket (SOCKFD) y termina la conexión TCP. Este descriptor no podrá ser utilizado más por el proceso. Si quedan datos en la cola de envío TCP intentará enviarlos y luego de enviados TCP iniciará el fin de conexión.

B CONTENIDO DEL CD

Se adjunta un CD ROM con esta versión de la documentación, corregida por el tribunal, en formato PDF. Los requerimientos mínimos para visualizar la misma son KPDF 0.4.1, XPDF 3.0, o similar. Se incluyen además el conjunto de videos de prueba.