UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ACATLÁN

DESARROLLO DE PREDICTOR DE PIXEL ADAPTATIVO, PARA LA OBTENCIÓN DE VERSIONES TRANSFORMADAS DE MAPAS DE BITS DE MAYOR CAPACIDAD DE COMPRESIÓN ENTRÓPICA

T E S I S

QUE PARA OBTENER EL TÍTULO DE

LICENCIADO EN MATEMÁTICAS APLICADAS Y COMPUTACIÓN

P R E S E N T A

JAVIER GARDUÑO CIMENTAL

ASESORA
M. EN C. SARA CAMACHO CANCINO

EDO. DE MEX. **SEPTIEMBRE 2011**

Universidad Nacional
Autónoma de México

Biblioteca Central

Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Agradecimientos

No daré las gracias a Dios como suele ser costumbre aquí. Mi pensamiento está más con David Hume cuando dice:

> *Si procediéramos a revisar las bibliotecas convencidos de estos principios, ¡qué estragos no haríamos! Si tomamos cualquier volumen de Teología o metafísica escolástica, por ejemplo, preguntemos: ¿Contiene algún razonamiento abstracto sobre la cantidad y el número? No. ¿Contiene algún razonamiento experimental acerca de cuestiones de hecho o existencia? No. Tírese entonces a las llamas, pues no puede contener más que sofistería e ilusión.*

Como dice una versión del antiguo principio africano: "*una persona es tal a causa de los demás*", somos el resultado de nuestras circunstancias y cuanta influencia ha llegado a nosotros. Somos hijos de nuestro tiempo. Podemos deducir, sin temor a equivocarnos, que nuestro entorno nos hace y modela. De forma que, siendo estricto, debería dar gracias a todo cuanto conozco y más. Pero no haré tal cosa. Solo daré las gracias a aquellos, que de forma consciente y en un acto deliberado, han favorecido la creación de este documento.

Mi agracedicimiento a Araceli Ramírez García, quién más tiempo ha permanecido a mi lado en el proceso de creación de este documento, y más apoyo ha puesto en él. A mis padres, J. Janet Cimental Velasco y Javier Garduño Hernández, por su fe en mi, además de su apoyo e insistencia para la culminación de este trabajo. A mis amigos y compañeros que, de buena o mala fe, hayan realizado cualquier acto que empujara a mi mente a continuar, escribir, buscar y pensar.

Índice general

Introducción **1**

1. Tratamiento digital de imágenes **3**
1.1. La luz . . . 3
1.2. El color . . . 5
1.2.1. El espectro cromático . . . 5
1.2.2. Teoría del color . . . 5
1.2.3. Espacios de color . . . 8
1.3. Tipos se imágenes . . . 9
1.4. Rangos y modos de color en los mapas de bits . . . 10
1.5. Formatos de almacenamiento . . . 11
1.6. Teoría de la información . . . 12
1.6.1. Entropía . . . 12
1.6.2. Compresión . . . 13
1.6.3. Técnicas de compresión aplicadas en este trabajo . . . 15

2. Obtención del predictor de pixel **22**
2.1. Antecedentes . . . 22
2.2. Propósito del predictor de pixel . . . 23
2.3. Supuestos . . . 23
2.3.1. Imagen ideal . . . 23
2.3.2. Imágenes posibles e imágenes con sentido . . . 24
2.3.3. Pixeles vecinos . . . 25
2.4. Plano simple . . . 28
2.4.1. Estimación del gradiente . . . 29
2.5. Evaluación del método de plano simple . . . 30
2.5.1. Condiciones de prueba . . . 32
2.5.2. Resultados . . . 33
2.6. Predicción de errores en c_s . . . 36
2.6.1. Distribución de pixeles y errores por tipo de inclinación 37
2.6.2. Análisis de propuestas . . . 41
2.7. Plano simple secundario . . . 43
2.7.1. Predicciones con el plano simple secundario . . . 45

2.7.2. Combinación de los métodos de plano simple y plano simple secundario 47
2.8. Elección de los predictores 47

3. Implementación y pruebas **50**
3.1. La necesidad de una especificación de mapa de bits 50
3.2. Especificación SPB 50
3.2.1. Convenciones 50
3.2.2. El formato SPB 52
3.2.3. Estructura del flujo de datos 53
3.2.4. Imagen de referencia a imagen SPB 55
3.2.5. Transformación de pixel 56
3.2.6. Asunciones para la transformación de pixel 58
3.2.7. Tipos de plano 58
3.3. Implementación 59
3.3.1. Elección del lenguaje 59
3.4. Pruebas 60

Conclusiones **63**

Apéndice **64**
Implementación de SPB en Python 64
Estructura de directorios 64
Código fuente 64

Bibliografía **83**

Resumen

Partiendo de supuestos sobre el comportamiento y características que una imagen posee a nivel de pixel, se hace un análisis heurístico que lleva a la creación de un novedoso predictor de pixel, un algoritmo reversible, destinado a transformar imágenes como paso necesario para facilitar la compresión sin pérdida de las mismas. Entendiendo como *compresión* el procedimiento que, tomando un conjunto de datos, lo transforma en otro diferente que utiliza menor espacio de almacenamiento, de forma que sea posible "reconstruir" la información original.

Se incluye un software que implementa las técnicas desarrolladas en el presente trabajo.

Introducción

Mi interés por la infografía digital me hizo descubrir las transformadas integrales, en particular, una conocida como *wavelet*. Su aplicación a la compresión de imágenes me intrigó, al grado de sugerirme a mi mismo la investigación de dicho tema para la elaboración de mi tesis. Desafortunadamente, el tema es demasiado basto, y depende de multitud de datos experimentales difíciles de obtener, como la respuesta del ojo humano a diferentes longitudes de onda electromagnética, entre otros. Todos ellos necesarios en el proceso de elaboración de un método de *compresión con pérdida* de imágenes digitales. Es decir, un método que implica la eliminación de datos poco importantes a fin de mejorar la compresión. Ante esto, decidí centrarme en el tema de la *compresión sin pérdida*, dejando a un lado el tópico de las imágenes. Ahí no habría que concentrarse en información experimental inaccesible.

Gasté mucho tiempo intentando descubrir el hilo negro de un territorio prácticamente explorado en su totalidad. Me encontré entrando a muchos callejones sin salida, o razonamientos claramente equivocados. Ya existían métodos que demostraban matemáticamente ser óptimos, o que mejoraban su rendimiento indefinidamente, según la capacidad de procesamiento y memoria destinados al mismo. Contra la entropía poco se podía hacer.

Opté por dar un paso atrás al tema de la compresión de imágenes, esta vez sin pérdida, explorando y buscando algo que pudiera realizarse para obtener una mejora a lo ya hecho. Consideré entonces el desarrollo de un mecanismo que coadyuvara a los sistemas de compresión ya existentes, esperando que dicho proceso fuese una mejora, incluso pequeña, de lo que existía hasta ese momento. Este trabajo es el resultado de dicha labor.

¿En qué consiste esta mejora? Para que ello se exprese correctamente, es preciso definir claramente el problema. Los mapas de bits, que usualmente representan imágenes, consumen gran cantidad de espacio de almacenamiento debido a la gran cantidad de elementos que las componen (pixeles). Para su transporte y almacen, es ideal la compresión de la información que constituye dicho *mapa de bits* (imagen). Entendiendo como *compresión* el procedimiento que, tomando un conjunto de datos, lo transforma en otro diferente que utiliza menor espacio de almacenamiento, de forma que sea posible "reconstruir" la información original utilizando ese nuevo conjunto de datos.

Si bien existen ya muchos y muy sofisticados métodos de compresión, la gran mayoría de ellos están diseñados para realizar una reducción de tamaño

con pérdida de información, quedando relegados los métodos que realizan la misma tarea sin pérdida de la misma.

Normalmente, estos últimos incluyen un proceso encargado de transformar la información a fin de facilitar el alcance de altas tasas de compresión. Sin embargo, dichos procesos son en extremo simples y su avance no se corresponde con el llevado a cabo por los sistemas de compresión con pérdida. El aumento en la potencia típica de un sistema de cómputo, así como el crecimiento de la misma en el futuro, permiten la creación de métodos más sofisticados que arrojen mejores resultados, sin la penalización de un tiempo excesivo de cálculo.

Con la creación de este algoritmo se pretende llenar el nicho creado por las técnicas usadas tradicionalmente. La aportación es, simplemente, la propuesta de un predictor de pixel diferente a los ya existentes. Si bien puede ser computacionalmente más costoso que otros, intenta ser también más eficaz. Sobre todo en imágenes fotográficas, que son el tipo de imágenes utilizadas en las pruebas y desarrollo del trabajo.

El problema abordado en este documento es encontrar una forma de transformar un *mapa de bits* (imagen) de forma que resulte más fácil su compresión utilizando un algoritmo tradicional de compresión entrópica (Huffman), o similar.

La obtención de este tipo de transformaciones es importante, pues coadyuva a la obtención de archivos más pequeños, en este caso imágenes, con el consiguiente ahorro de espacio de almacenamiento y consumo de ancho de banda.

¿Es acaso ésta una razón relevante en un mundo donde la capacidad de almacenamiento es cada vez más barata y el ancho de banda cada vez mayor? Lo es desde el momento en que los datos ocupan un espacio y consumen ancho de banda. Siempre implicará un costo, por muy pequeño que este sea. La compresión y una mejora de la misma siempre estarán justificadas.

De forma que en el **Capítulo 1** se aborda el tratamiento digital de las imágenes. Los formatos de almacenamiento destinados a ese propósito. Los tipos de imágenes digitales, y su naturaleza intrínseca. También se hará una breve reseña de los tópicos de la teoría de la información que están implicados en la compresión de datos, y que sean reelevantes para este trabajo.

En el **Capítulo 2** se expone la búsqueda y desarrollo del predictor de pixel propósito de esta tesis. Se explican y justifican las bases y supuestos sobre los cuales se sustenta su desarrollo. El algoritmo resultante es expuesto y explicado.

En el **Capítulo 3** se lleva a cabo la implementación del algoritmo de predictor de pixel en una especificación para el almacenamiento de mapas de bits, creada expresamente para dicho propósito. La especificación es llevada a la práctica, en un programa capaz de codificar y decodificar en un formato comprimido imágenes tratadas con el predictor desarrollado en el capítulo precedente. De igual manera, se lleva a cabo un análisis comparativo en prestaciones y razones de compresión de diversos formatos y tipos de imagen, contra el desarrollado en este trabajo.

Capítulo 1

Tratamiento digital de imágenes

A las imágenes paganas suceden las imágenes cristianas; pero, siempre las ficciones abstractas sostienen las civilizaciones.

André Maurois

1.1. La luz

Toda sensación asociada al sentido de la vista tiene su razón de ser en la estimulación y procesos que tienen lugar en la corteza visual, que se encuentra en la parte posterior de nuestro cerebro.

También es verdad que dichos estímulos y procesos son motivados casi siempre por la luz. Aquella que entra por los ojos con la intensidad suficiente para generar un estímulo en el nervio óptico.

Resulta evidente que todo estudio asociado con la imagen está íntimamente relacionado con la luz, con todas sus propiedades y características. Pero, ¿qué es la luz? ¿En qué consiste esa casi intangible "sustancia" que parece manar de toda cosa brillante, capaz de iluminar todo lo demás? Esta pregunta ha inquietado al hombre durante mucho tiempo. Los griegos, que se caracterizaron por sus indagaciones acerca de la naturaleza del mundo, llegaron a varias conclusiones.

La escuela pitagórica supuso que todo objeto capaz de ser visto emitía una constante corriente de partículas que, al entrar en el ojo, otorgaban la visión. No parece que fuera un problema para ellos el hecho de que los objetos no pudieran ser vistos en la oscuridad. Por otro lado, Aristóteles concluyó que la luz viajaba en algo parecido a ondas.

Con el paso de los siglos, dichos conceptos sufrieron de graduales modificaciones. Sin embargo, a través del tiempo siguió vigente parte del debate original: ¿era la luz un conjunto de partículas, o por el contrario, era algo que

viajaba en ondas? Según el momento de la historia en que se tenga puesta la atención, prevalecerá uno u otro punto de vista.

Hasta hace muy poco tiempo, en la primera mitad del siglo XX, se encontró una respuesta más satisfactoria, concluyendo que la luz, sin ser partículas u ondas, posee características de ambas entidades.

Siguiendo el precepto de estudiar los fenómenos de la naturaleza en base a sus propiedades, los griegos encontraron que la luz se mueve en línea recta. Un segundo descubrimiento de importancia se debe a Herón de Alejandría. Él observó que el ángulo de incidencia de un rayo de luz es igual al ángulo de reflexión en una superficie reflejante adecuada. Por ejemplo, un espejo.

Durante los siglos siguientes no hubo avances significativos en la comprensión de la naturaleza de la luz, hasta ya entrado el siglo XVII.

En 1621, Willebrord Snell, un matemático holandés, elaboró un principio que explicaba uno de los fenómenos más comunes, pero también más desconcertantes de la época: la refracción. Que no es otra cosa que el cambio de dirección que la luz sufre al pasar de un medio transparente a otro distinto. Es aquello que hace parecer a un trozo de madera quebrarse cuando se introduce en el agua. Aunque es cierto que gracias a dicho principio, la refracción y su forma de actuar ya no podían tomar a nadie por sorpresa, también es verdad que Snell jamás descubrió la razón por la cual se desvía la luz.

Le tocó a Christian Huyegens, sugerir una respuesta. Su planteamiento es simple: el índice de refracción de un material es proporcional a la velocidad con que la luz atraviesa dicho medio. De la misma forma, planteó un modelo matemático que describía el fenómeno de la refracción.

Con dicho modelo, la comprensión de la refracción se vio acrecentada enormemente, y el hombre estuvo en posición de comprender y mejorar los instrumentos ópticos, que en ese entonces eran bastante rudimentarios.

Todo instrumento óptico que opere a base de un lente (o un conjunto de ellos), se aprovecha del fenómeno de la refracción. Los lentes poseen una superficie curva, de tal forma que la luz incide con un ángulo diferente según sea el punto de la lente considerado. Siguiendo los principios de la refracción, la desviación de la luz será mayor o menor, según el ángulo de incidencia que tenga esta con la superficie del elemento refractor. Así, los lentes están constituidos para que la luz procedente de un punto que llega a su superficie sea, o bien refractada a otro punto del otro lado, o bien difundida de forma uniforme en todas direcciones.

La refracción posee una extraña característica: si luz va de una sustancia con alto índice de refracción a otro que lo tenga bajo, y si el ángulo con el que la luz incide en la superficie es lo suficientemente oblicuo, la desviación es tal, que ningún rayo escapa, y de hecho, jamás penetra en el segundo medio. Uno de los más conocidos exponentes de este fenómeno lo podemos encontrar en la fibra óptica. En ella, la reflexión interna se repite una y otra vez a lo largo del pequeño tubo, hasta que la luz sale por fin en el otro extremo.

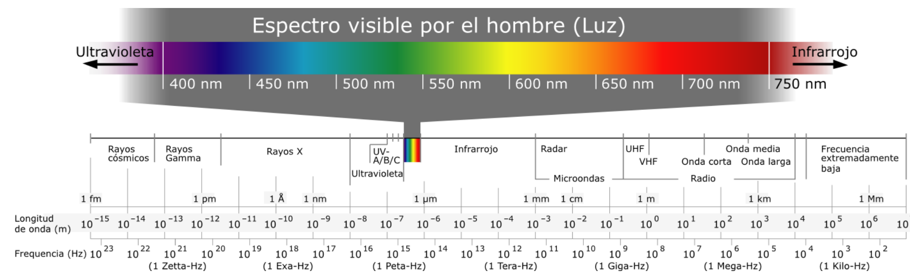

Figura 1.1: El espectro electromagnético y las frecuencias asociadas.

1.2. El color

El color no es una cualidad de los objetos o estímulos luminosos. El color es una impresión meramente subjetiva, que solo existe en la mente de quien lo percibe. Es el resultado de una interpretación por parte de nuestro cerebro de las diferentes longitudes de onda que componen la luz visible.

La luz visible es solo una pequeña parte del espectro electromagnético, que además comprende todos los demás tipos de emisión radial, como pueden ser las ondas de radio, los rayos X, las microondas, entre otras.

1.2.1. El espectro cromático

Con el descubrimiento hecho por Sir Isaac Newton en el siglo XVII de la naturaleza espectral de la luz, surgió la noción moderna de la misma que poseemos hoy día.

En el modelo de Newton, la luz está formada por partículas. Los diversos experimentos que realizó con prismas mostraron que la luz blanca se descomponía en los colores del arcoíris. Este hecho hizo pensar a Newton que las partículas asociadas a cada uno de los tonos del espectro poseían diferentes índices de refracción entre sí.

Ahora sabemos que los cambios en el índice de refracción que Newton observó se deben a las diferentes longitudes de onda, de lo que hoy conocemos como el espectro electromagnético.

1.2.2. Teoría del color

La teoría tricromática del color afirma que, siguiendo unas reglas básicas, es posible obtener los colores del espectro a partir de tres colores primarios preestablecidos. El fundamento de este principio se basa en el hecho de que los humanos, y algunas otras criaturas sensibles al color, poseemos en los ojos unas células llamadas *conos*. En os humanos existen tres clases de conos, cada uno con un pigmento fotosensible diferente. Cada tipo es más sensible a ciertas

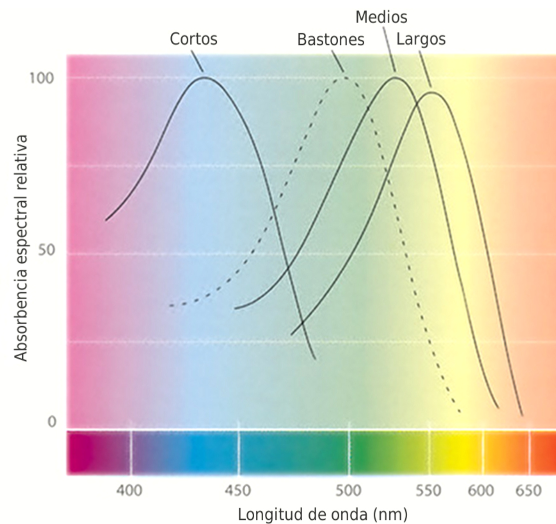

Figura 1.2: Sensibilidad relativa por longitud de onda para los bastones y conos *cortos*, *medios* y *largos*.

longitudes de onda que a otras. La estimulación aplicada a los diferentes conos se combina en nuestro cerebro, produciendo la sensación de color.

Tradicionalmente a los conos se les nombra como *azules*, *verdes* y *rojos*, aunque en realidad la sensibilidad máxima asociada a cada tipo de cono no se corresponde exactamente a dichos colores. Una forma más precisa de llamarlos es *cortos*, *medios* y *largos*.

El observador estándar CIE

Las siglas CIE se refieren al francés *Commission Internationale de l'Eclairage*, es decir: *Comisión Internacional de la Luz*. En el año de 1931, la CIE creó un sistema en el cual se utilizan los valores de tres primarios imaginarios para especificar cualquier estímulo cromático, basándose en la teoría tricrómica del color. Este sistema es conocido como el *observador estándar CIE 1931*.

Con el *observador estándar CIE 1931* se introdujeron métodos para definir las fuentes de luz (o iluminantes), las superficies, y en general, el funcionamiento del sistema visual humano, cuyo comportamiento se midió mediante funciones de correspondencia de color [15].

El *observador estándar CIE 1931* es el resultado de experimentos realizados con sujetos que buscaban encontrar la cantidad de necesaria de cada color primario para igualar un color de frecuencia monocromática[1].

[1]También se le conoce como *observador estándar de 2º*, ya que el área de color donde los sujetos del experimento debían realizar la comparación de estímulos cromáticos, tenía precisamente un tamaño angular de 2 grados sexagesimales.

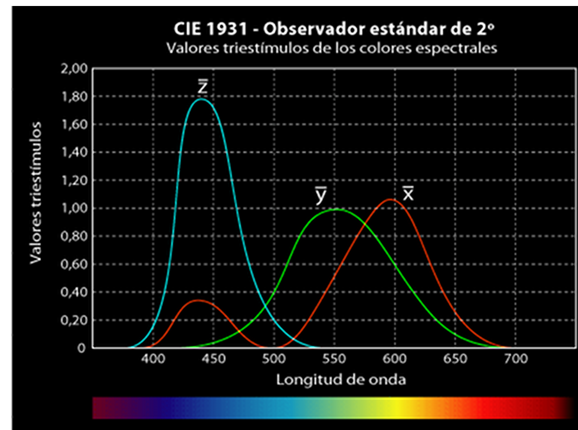

Figura 1.3: Observador CIE de 2 grados, y la cantidad relativa necesaria de cada color primario ideal para la creación de un color específico del espectro.

En la figura 1.3 se pueden observar las funciones de correspondencia para cada color primario especificado por el observador estándar, llamados: $\bar{x}$, $\bar{y}$ y $\bar{z}$ respectivamente.

Debe hacerse hincapié en que los estímulos $\bar{x}$, $\bar{y}$ y $\bar{z}$ del observador estándar son colores primarios idealizados e imaginarios, y que por tanto, no existen en la realidad. Su elección obedece a la necesidad de contar con estímulos que, variando su intensidad, dieran como resultado todos los colores posibles, además de otros factores de índole práctico.

La CIE ha creado otro observador estándar, conocido como el *observador estándar de 10º*, con objeto de subsanar las carencias del anterior. Ambos sistemas se utilizan, y sirven como punto de partida para la definición de espacios de color "base", utilizados como puntos de referencia absolutos en la industria e investigaciones posteriores sobre el color.

Tipos de mezclas

Existen dos formas en que colores primarios pueden mezclarse: la forma *aditiva* y la *sustractiva*.

Aditiva: En la forma aditiva, los colores son resultado de la suma de las señales luminosas de los colores primarios que los componen. Se puede equiparar a la mezcla de diferentes luces coloreadas. Cada una de un color primario distinto. Los colores son resultado de la mezcla de dichas señales luminosas. A la luz emitida por una luz coloreada se le agrega la señal de las demás luces coloreadas. Si la distribución espectral y las intensidades de cada luz son escogidas con cuidado, obtendremos el blanco como la combinación de ellas.

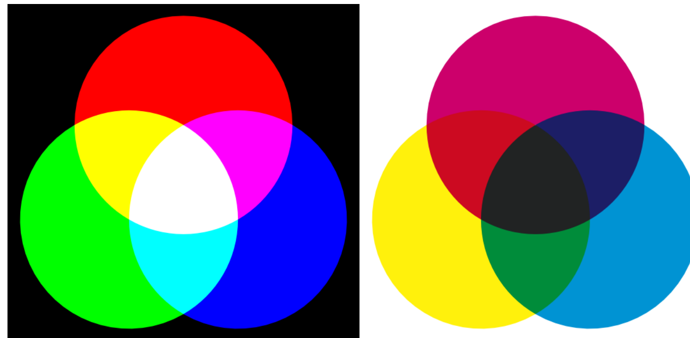

Figura 1.4: A la izquierda, un modelo de color aditivo: el RGB. A la derecha un modelo de color sustractivo: el CMY. Ambos muestran el resultado teórico de mezclar sus colores primarios a la máxima intensidad.

Sustractiva: En la forma sustractiva, los colores son resultado de una resta. Cada color primario elimina frecuencias indeseadas, hasta que solo prevalecen las asociadas al color final obtenido. Se puede equiparar al funcionamiento de las tintas en los medios impresos. En ausencia de tintas, tenemos el blanco del papel. La mezcla de tintas se encargará de eliminar las frecuencias no deseadas, para que la luz reflejada en el papel solo contenga frecuencias asociadas al color buscado.

No existe una triada de colores aditivos en particular que sea la única con la capacidad de generar los demás colores del espectro. Los colores que se pueden conseguir o igualar con un conjunto particular de colores primarios, es a lo que se llama el *gamut* de ese conjunto colores primarios. No es verdad que se puedan crear todos y cada uno de los colores que es capaz de percibir el ojo humano en base a una combinación de colores primarios particular.

Sin embargo, al elegir los colores rojo, verde y azul, podremos conseguir un número bastante grande de colores. De ahí que sean los predilectos para utilizarse como los generadores de todos los demás en la aplicación práctica de este principio.

1.2.3. Espacios de color

Un espacio de color es un modelo matemático que parametriza los colores, de forma que pueden ser expresados como el resultado de la combinación lineal de un conjunto generador de vectores, en un espacio vectorial acotado.

Por lo general, un espacio de color se refiere a un sistema de composición de color. Está basado en el concepto de colores primarios que nace en la teoría del color. En este caso, el espacio de color es un espacio vectorial acotado, que contiene todas las tonalidades resultantes de la combinación lineal de los colores primarios definidos en el modelo de color que representa dicho espacio de

color. Pero esta no es la única forma en que puede ser construido un espacio de color. En lugar de colores primarios, pueden utilizarse dimensiones psicofísicas de la percepción del color como parámetros. El *brillo*, *tono* y *coloración*, son ejemplos de estas dimensiones psicofísicas. Por ejemplo, el espacio de color *CIE L*a*b**, utiliza las dimensiones mencionadas para parametrizar los colores.

Los modelos de color más comunes utilizados en la práctica son los siguientes:

RGB (Red, Green, Blue): Este modelo se basa en la teoría tricromática del color. Es un sistema de síntesis aditiva y utiliza los colores rojo, verde y azul como colores primarios. En la industria existen dos variantes de este modelo: el sRGB y el Adobe RGB. Este último tiene un gamut superior al primero.

CMYK (Cyan, Magenta, Yellow, Black): Al igual que el modelo RGB, se basa en la teoría tricromática del color y es utilizado en los medios impresos. Es un sistema de síntesis sustractiva y utiliza como colores primarios el cian, magenta y amarillo. Además, se agrega un componente extra de color negro, que se hace necesario dada la imperfección de los sistemas de impresión.

HSL (Hue, Saturation, Luma): Este modelo define los colores en términos de variables psicofísicas como son: la tonalidad, la saturación y la luminancia. La tonalidad indica el color del espectro cromático al que se hace referencia. La saturación mide la pureza del color y su distancia al gris. La luminosidad indica la intensidad o brillo del color, que va desde el negro hasta el blanco, pasando por el color especificado.

HSB (Hue, Saturation, Brightness): Al igual que el modelo HSL indica un color en base a variables psicofísicas, Sin embargo el significado de las variables de saturación y brillo varían en su definición.

1.3. Tipos se imágenes

Para ser tratadas y manipuladas por los sistemas informáticos, las imágenes deben seguir alguna especificación de almacenamiento digital. Si bien actualmente existen multitud de variantes y sistemas mixtos para tal propósito, los tipos de representación digital de imágenes pueden ser clasificadas básicamente en:

Vectoriales: Son gráficos construidos en base a funciones matemáticas representadas en un espacio de dos dimensiones. En el argot correspondiente, dichas funciones suelen denominarse *curvas*.

Casi siempre son curvas Bézier, ya sea abiertas o cerradas, que tienen asociados atributos como: color de borde, ancho de borde, estilo de borde, estilo de relleno, color o colores de relleno, translucencia, etc., que

aplicados a un conjunto lo suficientemente grande de curvas, permite la generación de gráficos complejos.

Un rasgo que caracteriza a los gráficos vectoriales es su independencia a la resolución. Es decir, al estar definidos por funciones matemáticas, pueden ser ampliados tanto como se desee, y la representación podrá ser dibujada a la máxima resolución del dispositivo de salida empleado.

Mapas de bits: Tal como su nombre lo indica, los mapas de bits son imágenes generadas por una gran matriz de puntos que, vistos en conjunto, conforman una imagen.

Los puntos que constituyen a un mapa de bits son llamados *pixeles*. Para que ellos puedan crear una imagen, varían en su intensidad luminosa y color de forma independiente. Así, pueden crear los detalles y formas que constituirán la imagen final.

Se caracterizan por ser dependientes de la resolución. Al estar constituidos por una matriz de puntos, amplificar la imagen implica la ampliación de la representación de los puntos que la constituyen, de forma que los detalles más pequeños posibles se vuelvan cada vez más grandes. Lo que tarde o temprano da una apariencia que se conoce como *pixelada*, en honor a lo evidente de los pixeles que la constituyen.

El propósito de este trabajo es el desarrollo de un predictor de pixel que facilite la compresión de un mapa de bits, así que será ese el tipo de imagen tratado en este documento.

1.4. Rangos y modos de color en los mapas de bits

Para almacenar valores de brillo o color en los mapas de bits, es necesario definir primero las convenciones, modos de color, rangos, y valores que un pixel utiliza. A este conjunto de características lo conocemos por *formato de pixel*.

Existen ya formatos de pixel preestablecidos y acordados, con el fin de facilitar la interoperabilidad de los programas de tratamiento de imágenes, programas y dispositivos de visualización. Casi siempre su definición está basada en las características físicas del hardware que las vio nacer y los estándares en el almacenamiento de la información reinantes en ese momento. Así, es frecuente encontrar pixeles almacenados en una cantidad de bits múltiplo de dos u ocho.

A continuación un listado de los formatos de pixel más comunes:

Blanco y negro (1 bit): Cada pixel es almacenado utilizando un bit, lo que significan $2^1 = 2$ estados distintos que los pixeles pueden tomar. Usualmente, a estos valores se asocian los colores blanco y negro, de ahí el nombre de esta modalidad. Sin embargo, esa asociación no es obligada y puede variar a cualquier par arbitrario de colores.

Indexado (8 bits): Los colores asociados a cada pixel son tomados de una paleta que, por norma, puede contener hasta $2^8 = 256$ valores diferentes. El valor de cada pixel indica el color usado. La paleta es arbitraria.

Escala de grises (8 bits): Es una versión especial del modo de color indexado, existiendo también $2^8 = 256$ valores diferentes por pixel. La diferencia radica en la paleta de colores, que no es arbitraria, conteniendo el blanco, el negro, y 254 tonalidades de gris intermedias y equidistantes. Esta cualidad permite la aplicación de transformaciones y procesos a la imagen que resultan imposibles de hacer en una imagen indexada tradicional, debido a la impredictibilidad de su paleta asociada.

Color RGB (24 bits): Este modo toma casi siempre como base el espacio de color sRGB, aunque también puede que utilice el espacio Adobe RGB, asignando 8 bits a cada canal de color (Rojo, Verde y Azul). Para obtener un color cualquiera, basta con especificar la combinación lineal de valores RGB que lo generan. Existen $2^{24} = 16,777,216$ valores posibles para cada pixel.

Color CMYK (32 bits): Este modo toma como base el espacio de color CMYK. Este modelo es usado en los sistemas de impresión de cuatricomía. Se asignan 8 bits a cada canal de color (Cian, Magenta, Amarillo y Negro). De esta forma quedan descritas de manera explícita las proporciones de tinta asignadas a cada pixel.

Existen además variantes de los modelos *Color RGB* y *Escala de grises*, que se obtienen agregando un canal extra, llamado *alpha*. Este canal tiene como propósito indicar el grado de transparencia que dicho pixel posee. Debe hacerse notar que la transparencia solo tiene sentido mientras la imagen es procesada en un sistema informático. La transparencia permite la composición de dos o más imágenes para formar una nueva, producto de las anteriores. El resultado final, por supuesto, es una imagen con valores de color tradicionales.

El canal *alpha* no está presente en el modo de color CMYK, o cualquier otro basado en tintas. El propósito de los modelos de ese tipo es servir como formato final a utilizar en la industria de la impresión. La composición por canal *alpha* carece de sentido ahí.

1.5. Formatos de almacenamiento

Existen ya multitud de formatos y especificaciones destinadas a definir la forma de almacenar un mapa de bits. Algunas son propietarias y exclusivas de un software particular, otras no.

A continuación, la mención de los más comúnes:

JPEG (Joint Photographic Experts Group): Es un formato para el almacenamiento de mapas de bits utilizado ampliamente en el Internet, cámaras

fotográficas no profesionales, y muchos otros dispositivos. Es también el nombre del algoritmo de compresión utilizado por dicho formato. Está diseñado para utilizarse en mapas de bits en modos de color RGB y escala de grises, sin posibilidad de usar un canal *alpha*.

Aunque el estándar JPEG es muy amplio, buena parte de sus especificaciones no son de amplia utilización. El estándar soporta compresión con pérdida y sin pérdida de información, aunque esta última usualmente no es utilizada.

GIF (Graphic Interchangeable Format): El formato GIF fue creado por CompuServe en 1987 y es, aún hoy día, uno de los formatos más utilizados en la web. Su popularidad proviene de su capacidad para mostrar imágenes en sucesión, a fin de producir una animación sencilla.

Utiliza compresión sin pérdida, aunque está limitado a utilizar un máximo de 256 colores a la vez. Soporta transparencia a 1 bit de profundidad. Durante algunos años estuvo sujeto a patentes por el algoritmo LZW en el que se basa su compresión.

PNG (Portable Network Graphics): Fue desarrollado para solventar las principales deficiencias del formato GIF. A diferencia de este, soporta profundidades de color de hasta 24 bits y un canal alpha de 8 bits. Asimismo, utiliza algoritmos de compresión libres de patentes, lo que posibilita su utilización sin la necesidad de pagar regalías a terceros.

Existe una variante que soporta animaciones sencillas a la manera del formato GIF. Sin embargo, poco software está preparado para interpretar esa variante del formato PNG.

TIFF (Tagged Image File Format): Un formato que soporta diferentes modos de color. Es utilizado con relativa frecuencia en el diseño gráfico.

1.6. Teoría de la información

La teoría de la información es una teoría matemática, rama de la probabilidad y la estadística, dedicada al estudio de la información y todo lo concerniente a ella.

Fue Claude E. Shannon quién, en un artículo titulado *Una teoría matemática de la comunicación* publicado en el *Bell System Technical Journal* en 1948, dio inicio a este campo de estudio [11].

En esta teoría, la información es tratada como una magnitud. Es cuantificable, y para caracterizar la información de una secuencia de símbolos se utiliza la **entropía**.

1.6.1. Entropía

La entropía es un concepto no exclusivo de la teoría de la información. Existe en campos aparentemente tan dispares como la termodinámica (el estudio de

los efectos de los cambios de temperatura, presión y volumen de los sistemas físicos), o la mecánica estadística (la parte de la física que trata de determinar el comportamiento agregado termodinámico de sistemas macroscópicos). La entropía, para todos estos casos, se define esencialmente de la misma forma, utilizando la misma ecuación. En términos coloquiales, representa la cantidad de "desorden" que posee o libera un sistema.

En el caso particular de la teoría de la información, refleja la aleatoriedad que existe en la aparición de un carácter dado en una secuencia de caracteres que conforman un mensaje. Para una secuencia aleatoria de caracteres, la entropía es máxima. Cualquier carácter es igualmente probable. Pero si la secuencia no sigue los preceptos anteriores, la frecuencia de aparición puede verse alterada, cambiando la probabilidad asociada a cada símbolo.

Shannon ofrece una definición de entropía que satisface las siguientes afirmaciones:

- Es proporcional. Es decir, un cambio pequeño en las probabilidades de aparición de alguno de los elementos de la secuencia debe cambiar poco la entropía.
- Si los elementos de la secuencia son igualmente probables a la hora de aparecer, entonces la entropía será máxima.

Definición formal de entropía

La entropía medida en *bits* se define formalmente como:

$$H = -\sum_{i=1}^{n} p\left(x_i\right) \log_2 p\left(x_i\right) \tag{1.1}$$

Donde:

$p\left(X\right)$ es la distribución de probabilidad de variable discreta X.

La base del logaritmo involucrado determina la unidad en que la entropía es medida. De forma que la expresión anterior define a la entropía medida en *bits* (2^1).

Se resalta este aspecto de la teoría de la información, pues la entropía de un conjunto de datos se encuentra en relación directa con la cantidad mínima de bits que son necesarios para representar dicho conjunto. Es decir, está poderosamente relacionado con la compresión de la información.

1.6.2. Compresión

La compresión de datos se refiere a la habilidad de reducir la cantidad de espacio de almacenamiento necesario para representar un conjunto de datos. Esta reducción no hace alusión a un espacio físico, sino al número de bits utilizados en el almacenamiento de tales datos.

Tipos de compresión

Existen dos tipos de compresión. Se describen a continuación:

Compresión *con pérdida*: Tal y como su nombre lo indica, este tipo de compresión implica la pérdida o modificación de los datos originales, a fin de obtener una mayor tasa de compactación.

Es útil cuando se utiliza con datos que son la versión digitalizada de información analógica. Por ejemplo: imágenes, sonidos, y en general, cualquier tipo de señal. Por el contrario, resulta completamente inútil donde es vital la conservación de todos y cada uno de los datos originales. Estos casos pueden ser: un programa, una base de datos, un texto, entre otros.

Compresión *sin pérdida*: En este tipo de compresión la información original permanece inalterada y se conserva en su totalidad. Es posible reconstruir íntegramente los datos originales previos a la compresión. Esto casi siempre implica menores razones de compresión con respecto a un sistema de compresión *con pérdida*.

La compresión *sin pérdida* resulta necesaria en situaciones donde la conservación de los datos originales es indispensable. Por ejemplo: archivos ejecutables, documentos de texto, y en general, cualquier dato que se desea conservar sin alteraciones.

El propósito de este trabajo es desarrollar una técnica que mejore la eficiencia de un sistema de compresión sin pérdida. Por tal motivo, se indagará brevemente en la naturaleza de dicho tipo de compresión.

Una propiedad exclusiva de la compresión sin pérdida, que será relevante en el posterior desarrollo de este trabajo, es la que se enuncia a continuación:

No inyectividad de la compresión

Sea M el conjunto de números base n de m dígitos. Sea K el conjunto de números base n de k dígitos.

Donde:

$k < m$ y $k, m, n \in \mathbb{N}$.

M tiene exactamente $m - 1$ subconjuntos K. Llamese T a un conjunto definido como:

$$T = \{K_1, K_2, \ldots, K_{m-1}\}$$

Al ser disjuntos los conjuntos K_i, el conjunto T posee una cardinalidad:

$$|T| = n + n^2 + n^3 + \ldots + n^{m-1}$$

La cardinalidad de T no es otra cosa que la suma de la cardinalidad de cada uno de los subconjuntos K que lo constituyen.

Llamemos *regla de compresión* a la función C que, recibiendo como entrada un elemento de M, de como resultado un elemento de T, tal que para cada elemento de T generado por C, exista uno y solo un elemento de M que lo haya generado.

Dicho lo anterior, resulta evidente que la función C no puede generar un elemento de T para todos y cada uno de los elementos de M. De lo que se concluye que no todo elemento de M puede ser asociado a un elemento de cualquier conjunto K de forma inyectiva. Además, nos muestra una relación inversa entre la razón de compresión y la fracción de datos que se pueden comprimir: la gran mayoría de los datos que se pueden comprimir se comprimen poco, y solo pocos conjuntos de datos tienen altas tasas de compresión.

En otras palabras, podemos decir lo siguiente:

Lema 1.6.1. No es posible que todos los datos de tamaño m se compriman a un tamaño menor k, siempre que sea utilizada la misma regla de compresión.

Lema 1.6.2. A mayor tasa de compresión, menor la cantidad de conjuntos de datos que pueden alcanzarla.

1.6.3. Técnicas de compresión aplicadas en este trabajo

Es importante describir brevemente técnicas de compresión que serán parte importante de este trabajo, pues serán aplicadas al evaluar la efectividad del predictor de pixel contra otros predictores de naturaleza similar.

Codificación de Huffman

La codificación Huffman es una técnica ampliamente usada en la compresión de datos. Fue desarrollada por David A. Huffman como proyecto en sus estudios de doctorado en el MIT, siendo publicado en "*A method for the construction of minimum-redundancy codes*" [6].

Esta técnica re-codifica los símbolos suministrados generando un sistema de codificación variable para el mismo. Un sistema de codificación variable permite asignar una secuencia de bits diferente a cada símbolo, que no necesariamente son del mismo tamaño. Si a los valores que aparecen con frecuencia se les asigna una secuencia corta, y a los valores de poca frecuencia una secuencia larga, la cantidad de bits asignados al total de la información puede verse reducida, al menos, con respecto a la empleada en una codificación de longitud fija.

La entropía medida en bits refleja la mínima cantidad de bits que en promedio es posible asignar a cada símbolo de un conjunto de datos en particular, siguiendo las premisas anteriores. Es demostrable que la codificación Huffman genera una codificación óptima, que reduce al mínimo posible la cantidad de bits utilizados. Por esta razón, se denomina como un método de compresión entrópica.

Se asume que las secuencias de longitud variable asignadas a cada símbolo, deben estar construidas de tal forma que no exista ambigüedad sobre que bits pertenecen a cada valor. Dicho de otra forma: una secuencia particular de bits no puede ser el prefijo de otra más grande, de esta manera es posible leer una secuencia de bits y conocer sin ambigüedad donde inicia la secuencia correspondiente a cada símbolo.

Algoritmo de Huffman

El algoritmo funciona de la siguiente manera:

1. Obtener o especificar la frecuencia de aparición asociada a cada carácter utilizado en los datos originales.
2. Se debe crear un árbol por cada carácter con frecuencia diferente de cero. El árbol consiste de un nodo sin hijos.
3. A cada nodo se le asigna con su carácter asociado, y se le etiqueta con la frecuencia que le corresponde.
4. Se toman los dos árboles de frecuencia menor, uniéndolos para crear un nuevo árbol. Al árbol resultante se le etiqueta con la suma de las frecuencias de las raíces que se unen para formarlo. Las ramas que se obtienen también se etiquetan. A la rama izquierda se le asigna un 0 y a la derecha un 1. Puede ser a la inversa, siempre que se mantenga dicho orden durante todo el algoritmo.
5. Se debe repetir el último paso hasta que solo quede un solo árbol.

Para obtener el código asociado a un carácter se debe proseguir como sigue:

1. Iniciar con un código vacío.
2. Recorrer el árbol hacia arriba, iniciando por la hoja asociada a dicho carácter.
3. Se toma la etiqueta de cada rama recorrida, añadiéndose al código.
4. Una vez recorrido el árbol completo, el código resultante se invierte, obteniendo así el código definitivo.

Repitiendo este proceso en cada hoja, obtendremos el código de todos los símbolos. Para codificar el mensaje solo basta usar los nuevos códigos.

Para obtener un símbolo a partir del código se puede realizar lo siguiente:

1. Comenzar el recorrido del árbol por su raíz, eligiendo cada rama según los valores del código.
2. Al llegar a la hoja, basta con leer el símbolo asociado a la hoja.

En la práctica, resulta más conveniente leer el árbol solo una vez, y almacenar los símbolos y sus códigos asociados en otra estructura de datos más fácil de leer.

A continuación podemos ver un ejemplo de la aplicación del algoritmo.

Símbolo	**Probabilidad**
A	0.15
B	0.30
C	0.20
D	0.05
E	0.15
F	0.05
G	0.10

Tabla 1.1: Frecuencias asociadas a los símbolos A, B, C, D, E , F y G utilizados en el ejemplo.

Conociendo las frecuencias es posible construir el árbol Huffman asociado. En este caso los pasos a seguir son los siguientes:

1. Los nodos de valor más pequeño son los nodos *D* y *F*, con valor de 0.05 cada uno. Se unen en un nuevo nodo árbol con valor 0.10.
2. Se unen el árbol con las hojas *(D, F)* al nodo *G*, con valor de 0.10 cada uno.
3. Los nodos *A* y *E* son los más pequeños con 0.15 cada uno, así que se unen en un nuevo árbol con valor de 0.30.
4. El árbol formado por las hojas *(G, D, F)* se une al nodo *C*, con 0.20 de frecuencia cada uno. Esto forma un nuevo árbol de valor 0.40.
5. El árbol que contiene las hojas *(A, E)* se une al nodo *B*, con 0.30 cada uno. Se forma un nuevo árbol de valor 0.60.
6. Los dos árboles restantes *(B, A, E)* y *(C, G, D, F)* con 0.60 y 0.40 respectivamente, se unen en un nuevo árbol con valor de 1.0. Al ser este el único árbol restante, el proceso termina.

Como resultado, se obtiene el árbol que se presenta en la figura 1.5. Notar que las hojas representan a cada uno de los símbolos de la cadena original.

Siguiendo las instrucciones especificadas arriba podemos descubrir, por ejemplo, que el código asignado al carácter *B* es *00*, el de *F* es *1111* y el de *G* es *110*, donde cada bit corresponde al valor asignado a cada una de las ramas recorridas, para llegar de la raíz al nodo correspondiente.

Debe hacerse notar que la codificación Huffman, solo toma en cuenta la frecuencia de aparición de cada carácter para su elaboración. El orden de los caracteres es irrelevante. De forma que cualquier regularidad contenida de ese modo pasará desapercibida, y no podrá ser aprovechada por este tipo de codificación.

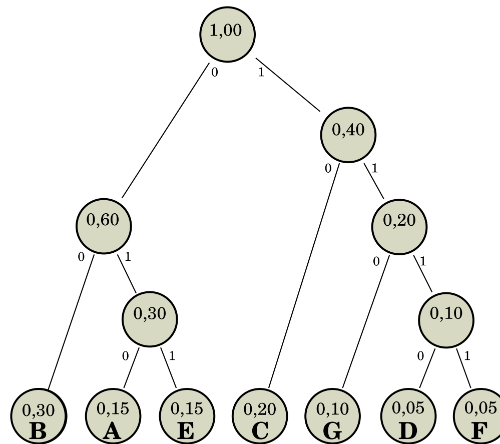

Figura 1.5: Árbol Huffman que resulta de aplicar el algoritmo a los datos de la tabla ejemplo.

Codificación por diccionario

A diferencia de la codificación Huffman, la compresión por diccionario funciona buscando secuencias de caracteres que se repitan a lo largo del conjunto de datos. De forma que no importa tanto la frecuencia de los caracteres individuales, como el orden que estos presenten en el flujo de datos.

El sistema fue propuesto en 1977 por Abraham Lempel y Jacob Ziv, modelando el primer sistema de compresión basado en diccionario. Aunque la idea original se mantiene en la actualidad, el algoritmo se puede implementar de diversas formas aplicando diversos parámetros, por lo cual existen muchas variantes de este tipo de compresión. Una de sus más famosas y utilizadas implementaciones es la conocida con el nombre de LZ77, que funciona de la siguiente manera:

Algoritmo LZ77

Antes de inciar, es necesario aclarar los términos utilizados en este algoritmo:

Flujo de datos: La secuencia de caracteres que se desea comprimir.

Carácter: El elemento de datos básico del flujo de datos.

Posición de codificación: La posición del carácter del flujo de datos que está siendo codificado.

Buffer de datos posteriores: La secuencia de caracteres que va de la posición de codificación hasta el final del flujo de datos.

Ventana: Una ventana de tamaño W contiene W caracteres, de la posición de codificación hacia atrás. Representa los últimos W carácteres procesados.

Puntero: Apunta a una coincidencia en la ventana, indicando su longitud.

Según se decribe en [12], el proceso de codificación es el siguente:

Se busca la ventana que poseea la coincidencia más grande con el buffer de datos posteriores, devolviendo un puntero a dicha coincidencia. Como es posible que no exista una sola coincidencia, la salida no puede contener solo punteros. El algoritmo resuelve este problema colocando, despúes de cada puntero, el primer carácter en el buffer posterior que se encuentre después de la última coincidencia. Si no hay coincidencia, devuelve un puntero nulo y el carácter en la posición de codificación.

El algoritmo sigue los siguientes pasos:

1. Ajustar la posición de codificación al inicio del flujo de datos.
2. Buscar la coincidencia mayor en la ventana del buffer posterior.
3. Devolver el par (P, C), que significa lo siguiente:
   - P es el puntero a la coincidencia en la ventana.
   - C es el primer carácter del buffer posterior que no coincidió.
4. Si el buffer posterior no está vacío, mover la pocisión de codificación y la ventana $L+1$ caracteres hacia adelante, donde L es la longitud de la coincidencia.
5. Regresar al paso 2.

A continuación podemos ver un ejemplo[12] de la aplicación del algoritmo:

Posición	1	2	3	4	5	6	7	8	9
Carácter	A	A	B	C	B	B	A	B	C

Tabla 1.2: Cadena de entrada utilizada en el ejemplo.

El proceso de codificación se presenta en la tabla 1.3.

- La columna *iteración* indica su número corespondiente, se completa cada vez que algoritmo genera una salida.
- La columna *posición* indica la posición de codificación. El primer carácter en el flujo de datos tiene la posición 1.
- La columna *coincidencia* muestra la coincidencia más larga encontrada en la ventana.

Iteración	Posición	Coincidencia	Carácter	Salida
1	1	-	A	(0, 0) A
2	2	A	B	(1, 1) B
3	4	-	C	(0, 0) C
4	5	B	B	(2, 1) B
5	7	A B	C	(5, 2) C

Tabla 1.3: El proceso de codificación.

- La columna *carácter* muestra el primer carácter en el buffer posterior, que se encuentra después de la coincidencia.
- La columna *salida* presenta la salida en el formato $(N, T)C$:
    - (N, T) es el puntero P a la coincidencia. El puntero da la siguiente instrucción al decodificador: Regresa N caracteres en la ventana y copia T caracteres a la salida";
    - C es un carácter explícito.

El proceso de decodificación es más simple que la codificación. Se mantiene una ventana de datos de la misma forma que en la codificación. En cada iteracion es leído el par (P, C) de la entrada, de forma que se obtiene como salida la secuencia especificada por P y el carácter C.

Este algoritmo se caracteríza por tener un sistema de decodificación más simple que el sistema de codificación. La codificación puede ser lenta debido a la constante comparacion de secuencias de caracteres que implica. El consumo de memoria de este algoritmo está en estrecha relación con el tamaño de la ventana utilizada.

Algoritmo Deflate

El algoritmo deflate es una combinación de los métodos descritos con anterioridad: la codificación Huffman y el sistema LZ77. La compresión utilizando ésta especificación es muy flexible.

El sistema ofrece tres posibilidades para tratar los datos:

1. No comprimir los datos en lo absoluto. Es buena idea cuando el flujo de datos ya está comprimido, o que por sus características, resulte en un conjunto de datos mucho más grande que el original al aplicarle algún tipo de compresión.

2. Comprimir. Primero con LZ77 y después aplicar Huffman. Los árboles Huffman utilizados se encuentran definidos en la especificación, y por tanto, no es necesario almacenarlos junto al conjunto de datos.

3. Comprimir. Primero con LZ77 y después aplicar Huffman. A diferencia del método anterior, el árbol Huffman es calculado en base al flujo de datos, de forma que debe ser almacenado junto con los datos codificados.

La forma en que el algoritmo deflate aplica estas diversas formas de tratar la información, es dividiendo el flujo de datos en bloques. Cada bloque es tratado con uno de los tres métodos descritos arriba.

Los bloques están precedidos por una cabecera de 3 bits que indican lo siguiente:

- El primer bit indica si es el último bloque del flujo de datos (`1`: si, `0`: no).
- Los siguientes dos bits indican el tipo de compresión aplicada al bloque:
  - `00`: Se almacena de forma literal.
  - `01`: Se utiliza Huffman con árboles preestablecidos.
  - `10`: Se utiliza Huffman y se proporciona un árbol para la decodificación.
  - `11`: Reservado. No se utiliza.

El conjunto de símbolos que la codificación Huffman utiliza en esta especificación incluye los caracteres literales, las bandera de fin de bloque, así como los elementos de los punteros resultado de la codificación LZ77.

Capítulo 2

Obtención del predictor de pixel

El Demonio le dice a Jesús: -"Me llaman impostor, falsario, embustero, mentiroso y padre de la mentira... Si negara ser el Diablo, tú deberías deducir que lo soy, puesto que siempre miento. Pero como yo sé que tú sabes que miento, diría que sí, que soy el Diablo, a fin de que creyeras que no lo soy. Pero como me imagino que ya sabes que yo sé que tú sabes... La pregunta, pues, sigue en pie: ¿soy yo, realmente, el Demonio?"

José María Cabodevilla

2.1. Antecedentes

En muchos casos, la información que posee un flujo de datos ha sido generado por un proceso o procedimiento no aleatorio, siguiendo un patrón u orden de naturaleza determinista. Esto convierte a dicho flujo de datos en algo potencialmente predecible. La predictibilidad de la información abre la posibilidad de reducir la entropía del conjunto de datos, pues la información realmente significativa se vería reducida.

Por otra parte, recordando el lema 1.6.1, es evidente la imposibilidad de comprimir todos los datos posibles siguiendo el mismo algoritmo de compresión. En el caso de las imágenes, significa que no todas las *imágenes posibles*, de cierto tamaño de almacenamiento, podrán ser comprimidas a un tamaño menor.

Con estas dos ideas presentes, se pretende crear un predictor de pixel que cumpla las siguientes funciones:

1. Explotar el orden y características intrínsecas que poseen las imágenes, con el fin de predecir su comportamiento y reducir su entropía, facilitando la compresión.

2. Servir como método de discriminación, alentando la compresión del subconjunto de imágenes posibles con significado, sobre las imágenss carentes de él.

2.2. Propósito del predictor de pixel

El predictor de pixel pretende ser un auxiliar en la compresión entrópica de los datos asociados a un mapa de bits. Su objetivo es transformar ese conjunto de datos, de forma que la aplicación de un método de compresión resulte en un conjunto de datos más pequeño que de no haber realizado dicha transformación. La forma de conseguirlo es prediciendo el valor de los pixeles, almacenando solamente las diferencias que posea con respecto al valor real.

Se desea además, que la compresión resultante sea "sin pérdida". Para que todo ello resulte posible deben cumplirse las siguientes condiciones:

Condición 2.2.1. El predictor de pixel debe realizar una transformación sin pérdida de información. Es decir, no debe perderse nada de la información original.

Condición 2.2.2. El predictor de pixel debe poseer inversa. Es decir, tomando como punto de partida los datos transformados, debe ser posible realizar una operación que permita la obtención de los datos originales previos a la aplicación del predictor de pixel.

2.3. Supuestos

De ahora en adelante, al hablarse de *imagen*, se hará referencia a un gráfico constituido por un solo canal de color. Para imágenes que contengan un número mayor de canales, se utilizará el término *imagen multicanal*.

Esto tiene el propósito de asegurar que un pixel solo contendrá uno y solo un valor de color asignado, y no una cantidad arbitraria de ellos.

2.3.1. Imagen ideal

Una *imagen ideal* puede ser concebida como un campo escalar, donde el valor asociado a cada punto es su brillo correspondiente. Es una función $\Phi(\vec{r})$ donde su valor depende de la posición (x, y) considerada. Se puede expresar como:

$$\Phi = \Phi(\vec{r}) = \Phi(x, y) \tag{2.1}$$

Donde $\vec{r}$ es un vector de coordenadas cartesianas (x, y) que representa cada punto de la *imagen ideal*.

Un mapa de bits puede ser concebido como un muestreo de la función Φ. Es una versión discreta del campo escalar, donde cada muestra corresponde a un pixel.

2.3.2. Imágenes posibles e imágenes con sentido

Estrictamente hablando, un mapa de bits puede ser cualquier arreglo de valores. El valor asociado a cada pixel puede ser completamente arbitrario, y acepta todas las combinaciones posibles. Afortunadamente, no resulta interesante ni necesario tener la capacidad de comprimir todas las *imágenes posibles*. La principal razón consiste en que no todas las *imágenes posibles* tienen sentido para el observador. Una *imagen posible* puede ser el arreglo de valores de pixel que retrate un paisaje, o la silueta de algún animal. Pero también puede ser un ruido caótico y sin sentido. El propósito será, por tanto, encontrar un método que permita discriminar entre ambos tipos de imágenes: las que tienen "sentido" de las que no. De forma que el pequeño conjunto de *imágenes con sentido* sea el subconjunto de *imágenes posibles* de tamaño n que sea representada por un conjunto de datos posibles de tamaño m. Dónde $n > m$.

La definición de *imagen con sentido* es vaga hasta ahora. Y es de esperar, ya que es el resultado de una apreciación puramente subjetiva. La pregunta entonces es: **¿qué significa exactamente *imagen con sentido*?** Es necesario definirla de manera formal, ya que es la única manera en que podrá ser abordado el problema por medio de un algoritmo, que es lo que se desea obtener.

Para definir a una *imagen con sentido*, necesitamos conocer las características que debe tener una impresión visual, para que nuestro cerebro la interprete como información con significado relevante. No cualquier configuración de pixeles vale para tal propósito. En la realidad el ser humano se confronta con impresiones visuales que se constituyen de imágenes de objetos, patrones, personas, y en general, de entidades claramente identificables entre sí.

Siguiendo principios muy elementales de la psicología del Gestalt, se nos dice que la percepción funciona a base de "recortes" en los cuales se posa la atención y llamamos "figuras". Las zonas que circundan a los "recortes" o "figuras" son llamadas "fondo". A esto se le conoce como *Ley de figura-fondo* [7] Para que nuestro cerebro tome alguna sección de la imagen como figura es necesario que cumpla ciertas características, por ejemplo, que dicha sección posea un color similar, o que tenga un fuerte contraste con los elementos que se encuentran a su alrededor, o bien, poseer alguna clase de patrón que no posee el resto de la imagen. Existen otras leyes de la Gestalt, como la *Ley de cierre*, la *Ley de proximidad* o la *Ley de similaridad*, que nos hablan de como el cerebro tiende a completar la información que recibe, agrupa elementos por distancia y por similitud para considerar parte de una impresión visual como una sola figura.

En este caso particular, se aprovecharán los principios descritos por la *Ley de figura-fondo* como punto de partida para definir de manera más formal las características que cumple una *imagen con sentido*. Estas descripciones serán la piedra angular del algoritmo predictor. En base a ellas y sus implicaciones realizará la predicción.

Las características subjetivas mencionadas anteriormente, que definen a una imagen con significado, es necesario expresarlas en principios o supuestos que nos permitan trabajar con ellas. Podemos decir lo siguiente:

Definición 2.3.1. Una *imagen con sentido* está conformada por grandes áreas de valor (*color*) similar.

Aquí se intenta hacer cumplir la *Ley de figura-fondo* mencionada arriba, y en particular, uno de los rasgos característicos que hacen a la mente humana separar una figura de su fondo circundante: el contraste entre ellos. Asumimos que grandes áreas de color similar serán entendidas como la misma figura.

Hasta aquí, el término *grandes áreas* es ambiguo. Pero veremos que no es necesario definirlo explícitamente para el propósito que se persigue. Los pixeles, como expresiones del componente más pequeño e indivisible de un mapa de bits, son por naturaleza de poco tamaño. Su fin no es representar ni ser percibidos individualmente como una estructura en el gráfico que conforman. Son de hecho, el detalle más insignificante que una imagen puede contener.

En vista de lo anterior, podemos decir que:

Definición 2.3.2. Un *área grande* consiste en un conjunto arbitrario de pixeles adyacentes entre si.

Siguiendo lo anterior, podemos deducir que dentro de un área grande, se cumple que:

Definición 2.3.3. El valor de un píxel *tiende a ser* igual al valor de un pixel adyacente.

De no ser el caso no se podría cumplir la definición 2.3.1, puesto que no se formarían grandes áreas de valor (color) similar.

Grandes áreas de color no implican, necesariamente, que todos los pixeles que las conforman posean *exactamente* el mismo valor. Formulemos aquí, un supuesto un tanto aventurado, y fruto del sentido común:

Definición 2.3.4. Para áreas grandes del mismo color, los valores de los pixeles intermedios a otros dos colocados en posiciones arbitrarias, *tienden a contener* los valores intermedios a los de dichos pixeles arbitrarios.

Si los pixeles intermedios a otros dos no poseen sus valores también intermedios, se fomentarán diferencias más grandes entre pixeles adyacentes en el trayecto que va de un pixel arbitrario a otro. Lo cual, basándonos en la definición 2.3.3 es una situación más improbable, y por tanto, tendrá menos frecuencia de aparición. El comportamiento que fomenta la definición 2.3.3 es una transición suave en los valores que conforman el camino de un pixel a otro, pues minimiza las diferencias de valor entre pixeles adyacentes.

Este comportamiento, la transición suave de color de un pixel a otro, no es lo único con alta probabilidad de aparición en imágenes con sentido, pero sin duda es más probable que el simple ruido, y para conjuntos de pocos pixeles, que es lo que se utilizará el predictor, bastará dicha noción.

2.3.3. Pixeles vecinos

Gracias a 2.3.3 y 2.3.4 se sabe que, mientras más cerca se encuentre un pixel de otro, más probable será que su valor sea el mismo. Con esto en mente, se

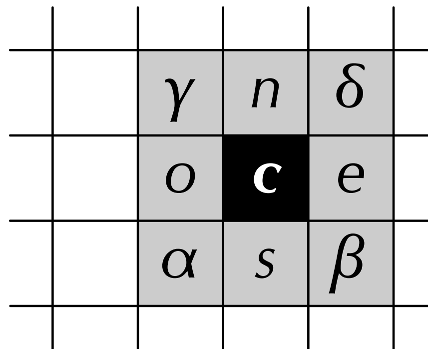

Figura 2.1: Pixeles vecinos. Donde los pixeles en gris son vecinos de c.

utilizarán los pixeles más próximos que sea posible para estimar el valor del pixel que se desea predecir.

Para propósitos de este trabajo, los pixeles vecinos a otro llamado c se han denominado de la siguiente manera: γ (gamma), n (norte), δ (delta), o (oeste), e (este), α(alfa), s (sur) y β (beta). Su configuración se encuentra esquematizada en la figura 2.1.

Disponer de todos los pixeles vecinos de c significa que conocemos el valor de cada uno de ellos, lo cual sería ideal al momento de realizar la predicción. Sin embargo, resulta absurda la idea de conocer todos los pixeles vecinos de c, y al mismo tiempo no conocer c. Si bien es posible calcular una predicción de c basándose en todos y cada uno sus vecinos, resulta inviable como método de predicción reversible, pues resulta necesario conocer el valor de c para predecir a sus vecinos y viceversa.

Por tal motivo es necesario que el algoritmo de predicción se base en datos que no necesiten al pixel por predecir para ser conocidos. Esto se consigue analizando de forma secuencial la imagen, utilizando solamente pixeles previamente analizados. La forma tradicional para almacenar un mapa de bits es secuencial, de forma que se aprovecha una característica intrínseca a su naturaleza de esta manera.

El recorrido usual para ordenar los pixeles de un mapa de bits, que se ha elegido para la definición de este predictor, es el siguiente:

- El primer pixel es el situado en el extremo derecho de la línea superior de la imagen.
- El consecutivo de cualquier pixel será siempre el situado a su derecha, si es que existe. De no existir se optará por el situado en el extremo izquierdo de la línea inmediatamente inferior. Si tampoco existe un pixel con esas características, se dice entonces que se ha alcanzado el último pixel.

Aplicando la definición, los pixeles vecinos a c que pueden ser utilizados en la predicción de forma que sea inversible son: γ, n, δ y o. (Ver figura: 2.3).

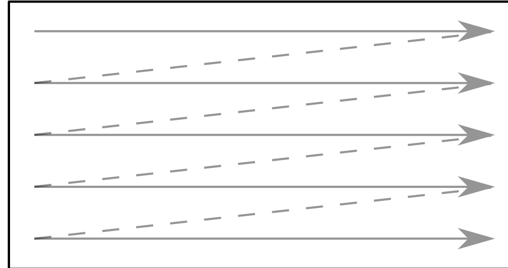

Figura 2.2: Orden de lectura de los pixeles de un mapa de bits, tal y como se expresa en este trabajo. El marco representa las fronteras de la imagen, mientras que el recorrido zigzagueante muestra el orden de lectura de los pixeles.

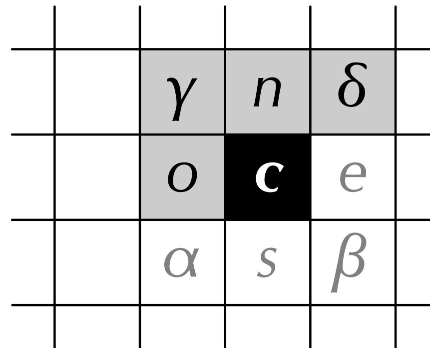

Figura 2.3: Pixeles vecinos a c, en fondo gris, que pueden ser utilizados en una predicción inversible según las pautas señaladas en el texto.

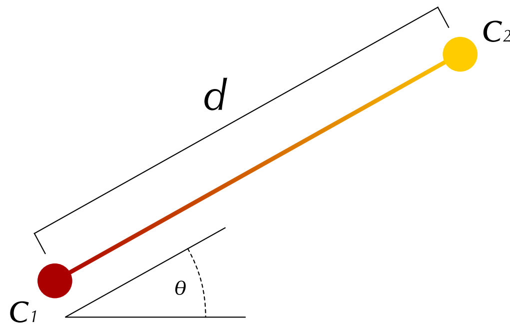

Figura 2.4: Los pixeles c_1 y c_2, se encuentran a una distancia arbitraria d en una dirección arbitraria θ. Sus valores asociados se representan aquí por la tonalidad asignada a cada uno, y la transición de valores en los pixeles intermedios como la línea de color degradado que los une.

2.4. Plano simple

El concepto de *plano simple* tiene como base la transición suave de valores entre pixeles expresada con anterioridad, y constituirá una primera aproximación al valor del pixel desconocido c.

Tal como se describió, tomando dos pixeles arbitrarios de la imagen que pertenezcan a un área grande del mismo color, se tenderá a obtener una transición suave de valores en los pixeles intermedios. Si la posición de los pixeles extremos es arbitraria, el ángulo de una línea que los una es también arbitrario.

Se deduce que:

Definición 2.4.1. La transición de valores tiene una dirección definida (la dirección de la recta que une a los pixeles), y una pendiente promedio, representada por:

$$m = \frac{c_1 - c_2}{d}$$

Donde d es la distancia entre los pixeles c_1 y c_2.

A cada pixel le atraviesan multitud de rectas y transiciones suaves. A cada dirección corresponde una pendiente diferente. De forma que es posible obtener una pendiente considerable en un sentido, y una prácticamente nula en otra.

Si el mapa de bits es considerado un campo escalar muestrado, donde el valor del campo en un punto se exprese como el valor del pixel correspondiente, entonces, se puede hablar de un gradiente en el campo escalar que el mapa de bits representa.

Esto es más fácil de observar si parametrizamos cada pixel con la triada ordenada (x, y, v), donde x e y se corresponden con las coordenadas cartesianas

del pixel en el mapa de bits, y v con el valor asociado a él. Esto convierte la imagen en una función $v = f(x,y)$. Es decir, en una superficie que vive en un espacio tridimensional. Cada pixel se concibe como una muestra de dicha superficie.

La razón de cambio en el valor de un pixel respecto a sus pixeles adyacentes en una dirección dada, puede ser visualizada como la derivada direccional asociada a dicho punto.

2.4.1. Estimación del gradiente

La distancia mínima entre dos puntos cualesquiera de una imagen es la distancia entre dos pixeles vecinos. De forma que esta distancia constituye el incremento más pequeño en el plano (x,y) sobre el cual es posible medir una razón de cambio. En este caso, una cambio en v, que es el valor (color) asociado a la imagen en un punto cualquiera.

De forma que los pixeles vecinos a un pixel c, se convierten en el recurso ideal para realizar una aproximación del gradiente de la imagen en ese punto, y por tanto, del valor que dicho pixel c posee.

Como es fácil intuir, son necesarios al menos tres pixeles que no se encuentren sobre la misma recta, para calcular un gradiente en la superficie bidimensional que constituye a una imagen.

Elegir tres pixeles resulta por demás práctico, pues también es cierto que tres puntos cualesquiera se encuentran siempre sobre un mismo plano. Basta entonces calcular el plano correspondiente y calcular su inclinación para así extrapolar el valor de c.

Mientras más cerca estén los tres pixeles entre si y el pixel a predecir, tanto mayor será la probabilidad de que su inclinación se corresponda con el gradiente asociado al punto de interés. Consultado la figura 2.3, resulta evidente que los pixeles ideales para calcular dicho plano serán γ, n y o. Son los más cercanos entre si y el pixel c, de entre los cuatro pixeles vecinos disponibles para usarse en una predicción invertible.

Haciendo uso de los razonamientos anteriores, se crea la siguiente definición:

Definición 2.4.2. El plano común al que pertenecen los pixeles γ, n y o de un pixel c cualquiera, es el **plano simple** de c.

Al igual que la imagen, el plano simple es una función $v' = g(x,y)$.

Los pixeles mencionados se encuentran posicionados de forma tal, que son los vértices de un cuadrado. Esto resulta conveniente, pues convierte al cálculo del gradiente, y por tanto de una primer estimación del valor de c, en una tarea sumamente sencilla. Es ideal, pues conviene que la predicción sea un procedimiento sencillo y sumamente rápido, dada la enorme cantidad de veces que debe ser realizado para evaluar una imagen completa.

El centro de este cuadrado imaginario, que denominaremos p, es también el punto medio entre los pixeles o y n. Por la definición 2.3.4, sabemos que el

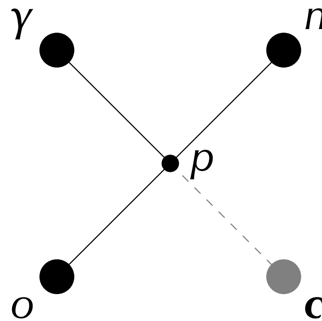

Figura 2.5: Posición relativa en el plano (x, y) de los pixeles vecinos de c, el punto p, y las rectas que los unen.

valor estimado de la imagen en p no es otra cosa que el punto medio de los valores de o y n. Pero no solo eso, es también el punto medio entre los valores de γ y c. De forma que la estimación de c resulta muy simple. Basta estimar el valor de la imagen en p con ayuda de o y n, y extrapolar el valor de c utilizando γ y el valor de p previamente calculado, de forma que el resultado sea el valor de v' en las coordenadas de c.

Así, el valor estimado de c según el cálculo de plano simple es:

$$c_s = n + o - \gamma \tag{2.2}$$

Donde c_s es la estimación de c por el método de plano simple.

En la práctica, no es posible asignar todos los posibles valores de c_s a c definidos en 2.2 a un pixel. Esto se debe a que los pixeles c solo son capaces de almacenar un valor que se encuentre dentro de un rango previamente establecido, usualmente definido por el *formato de pixel*. En los casos donde c_s resulte un valor inválido para c, se seguirán las siguientes directrices:

- Si el valor de c_s es estrictamente mayor al límite superior del rango de c, es decir, del valor máximo que es posible asignar a un pixel, entonces, c_s tomará el valor de dicho límite superior.
- Si el valor de c_s es estrictamente menor al límite inferior del rango de c, es decir, del valor mínimo que es posible asignar a un pixel, entonces, c_s tomará el valor de dicho límite inferior.

2.5. Evaluación del método de plano simple

Antes de analizar el sistema de plano simple para reducir su margen de error, es necesario confirmar el supuesto de que cumple su cometido como un

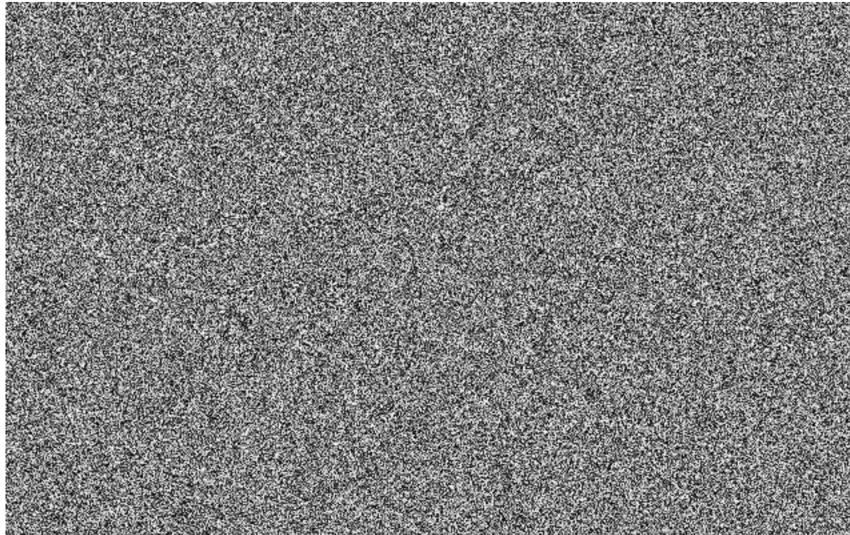

Figura 2.6: Imagen de ruido uniformemente distribuido.

sistema de aproximación al valor de c. Es necesario comparar sus resultados con un procedimiento equivalente que se sepa de antemano resultará fallido.

Si bien la predicción por plano simple resulta intuitivamente buena, es necesario confirmar esta apreciación numéricamente.

Si los supuestos en los que se basa la predicción de plano simple son correctos, su aplicación a un conjunto de datos que no cumpla las características esperadas para una *imagen con sentido* resultará menos eficiente, es decir, cometerá más errores que en una imagen que si cumpla dichos requisitos. En este caso, su aplicación en una *imagen con sentido* será más eficiente que en otra imagen que no cumpla la definición 2.3.1 y todas sus implicaciones.

La imagen más alejada de la ideal *imagen con sentido* es un ruido que cumple las siguientes características:

- Debe existir la misma probabilidad de aparición para cualquier valor de pixel en cualquier momento, es decir, la entropía es máxima.
- Los valores de pixel deben ser independientes entre si. Es decir, el valor de un pixel particular es completamente independiente de su posición en la imagen.

Un ejemplo de este tipo de imagen se puede apreciar en la figura 2.6.

Las dos caraterísticas antes mencionadas vuelven teóricamente inútil al predictor de pixel, que se basa precisamente en la dependencia que tiene el valor de un pixel del valor de sus pixeles adyacentes.

Figura 2.7: Un par de las 132 imágenes utilizadas en la prueba del predictor de pixel.

2.5.1. Condiciones de prueba

Imágenes

El predictor de plano simple se pondrá a prueba en un conjunto de 132 imágenes debidamente seleccionadas. El propósito de la selección es tener un conjunto de gráficos representativos de *imagenes con sentido*, o de forma más precisa, de los planos simples que constituyen a *imagenes con sentido*, según se decribieron con anterioridad.

El conjunto está formado enteramente por imágenes multicanal, siguiendo el modelo sRGB, con profundidad de 8 bits por canal. Sin embargo, solo es necesario por ahora analizar un solo canal de color. Con este propósito, se ha decidido convertir las imágenes a un modelo de escala de grises, siguiendo la definición establecida por la Unión Internacional de Telecominicaciones en su especificación ITU-R 709 [13].

La información de corrección de gamma, de existir, ha sido descartada, utilizando los valores directamente inscritos en los archivos de imagen.

El tema de las imágenes es variado, en su mayoría son fotografías, aunque existen también los gráficos sintéticos. Se ha procurado, de forma subjetiva, abarcar el máximo posible de patrones y circunstancias asociados a las imágenes con sentido. Así, pueden estar presentas o no, patrones repetitivos, grano de película, gradientes suaves de color, bordes duros, etc. El conjunto total de imágenes utilizadas se presenta como anexo a este trabajo escrito.

Como grupo de control, se han utilizado 31 imágenes de ruido en escala de grises.

Pixeles empleados

No todos los pixeles pertenecientes a una imagen se han utilizado en la prueba. La razón es que no todo pixel de una imagen posee vecinos γ, n, δ y o a la vez. Es razonable utilizar solo aquellos que los posean, para cada una de las imagenes involucradas. De forma que no se han tomado en cuenta los

pixeles pertenecientes a la primer fila de la imagen, así como los de la primera y última columna.

Será el procedimiento a seguir en esta prueba y posteriores.

2.5.2. Resultados

Como fue dicho con anterioridad, los resultados de la aplicación del predictor por plano simple deben ser diferentes entre el ruido e imágenes con sentido, además de favorecer este último la reducción de entropía y error.

Distribución del error

En el ruido, los errores se distribuyen en forma triangular. El número de pixeles posibles para cometer un error (por exceso o escases, según el caso) se reduce de forma lineal al aumentar el valor absoluto de dicho error. Solo pixeles de valores extremos (muy pequeños o muy grandes), permiten un error en su estimación de gran magnitud. Existen menos formas de cometer errores grandes que errores pequeños. La distribución típica se puede observar en la figura 2.8.

Es el comportamiento esperado en el caso de un predictor que de resultados sin relación alguna con los valores a predecir. Esta distribución se repite para cada imagen de ruido analizada.

Según la síntesis descrita en la tabla 2.1, se demuestra que la distribución es simétrica, según las pruebas aplicadas con una significancia del 0.05 a 31 diferentes imágenes de ruido.

Ruido (errores)		
	Coeficiente de sesgo	**Prueba de sesgo (valores-p)**
Promedio	0.0001409	0.6856337
Desviación estándar	0.0025134	0.1970427

Tabla 2.1: El promedio para los coeficientes del sesgo y desviación estándar del error obtenido al analizar 31 imágenes de ruido. Asimismo, el promedio de los valores-p para la prueba de sesgo, y la desviación estándar del mismo conjunto.

Es posible observar que el valor-p se encuentra por encima del nivel de significancia establecido, repitiéndose esto en cada imagen particular, demostrandose así que las distribuciones son simétricas de forma estadísticamente significativa.

En el caso de imágenes con sentido, el ruido se distribuye de forma diferente. Tal y como puede apreciarse en la figura 2.9, que es representativa de este comportamiento, se observa un patrón distinto, que concentra la mayoría del error en los números de menor valor absoluto.

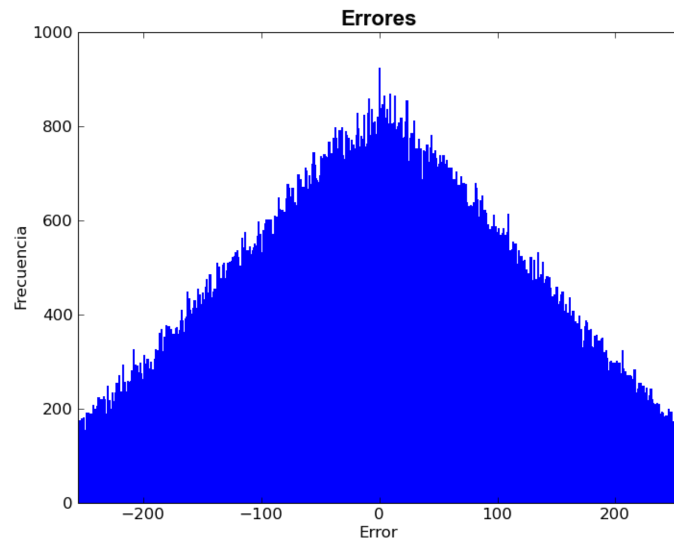

Figura 2.8: Distribución típica de los errores de predicción por plano simple, cuando éste es aplicado al ruido.

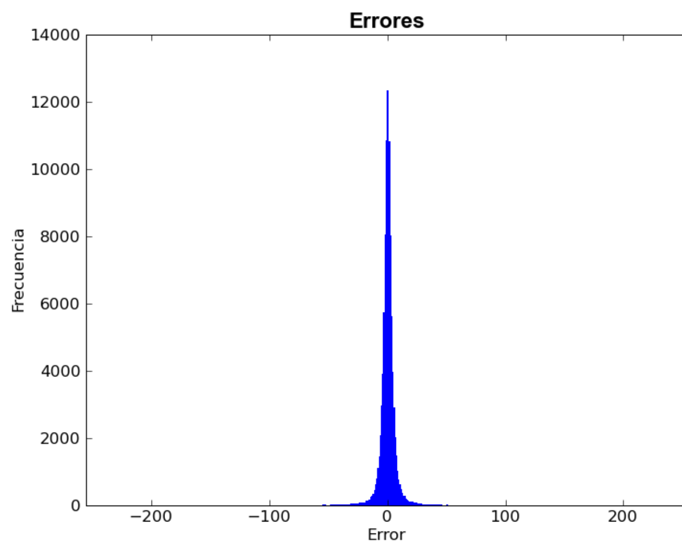

Figura 2.9: Distribución típica de los errores de predicción por plano simple, cuando éste es aplicado a una imagen con sentido.

Salvo contadas excepciones, las distribuciones del error para la pedicción aplicada a imagenes con sentido, no pasan la prueba de simetría. A pesar de su apariencia, son esencialmente no-simétricas. El sesgo es arbitrario, y se deduce que es poderosamente dependiente de las características intrínsecas de la imagen analizada.

Imágenes con sentido (errores)		
	Coeficiente de sesgo	**Prueba de sesgo (valores-p)**
Promedio	-0.0005462	0.0388129
Desviación estándar	0.4401407	0.1502509

Tabla 2.2: El promedio para los coeficientes de sesgo y desviación estándar del error obtenido al analizar imágenes con sentido. De igual manera, el promedio de los valores-p para la prueba de sesgo, y la desviación estándar del mismo conjunto.

Al no ser simétricas, no podemos asumir la distribucion normal de los errores, pero es posible notar una diferencia estadísticamente significativa entre la varianza que presentan estas y la que presenta el predictor por plano simple aplicado al ruido.

Desviación estándar promedio del error	
Ruido	**Imágenes con sentido**
118.9920367	8.6577798

Tabla 2.3: La dispersión por tipo de imagen (ruido e imagen con sentido) es claramente discernible, según los datos arrojados por el análisis.

También es posible verificar que la moda, media y mediana del error, para ambos casos, se aproxima a 0.

	Promedio		
	Moda	**Media**	**Mediana**
Error en ruido	0.0	0.0084720	0.0
Error en imágenes con sentido	0.0	0.0132616	0.0

Tabla 2.4: Resumen de modas, medias y medianas para ruido e imágenes con sentido

El análisis de las varianzas y la diferencia en la simetría de ambas poblaciones, son las pruebas más contundentes sobre la diferencia de comportamiento del predictor entre ellas, así como de su eficacia al momento de intentar predecir el valor de los pixeles.

Entropía calculada

Se presenta el resumen del cálculo de la entropía por tipo de imagen:

Entropía del error		
	Ruido	**Imágenes con sentido**
Promedio	7.9992612	4.2026105
Desviación estándar	0.0000655	0.8729370

Tabla 2.5: Resumen de la entropía del error por tipo de imagen.

Los datos revelan la entropía máxima que posee el error de predicción aplicado al ruido, acercandose al máximo teórico de 8.0. Los datos se muestran muy compactos, característica que se revela en la pequeña cantidad de desviación estándar que posee dicho conjunto de datos.

Por el contrario, el error proveniente de la predicción a imágenes con sentido, si bien es compacto, no lo es tanto como en el caso del ruido, además de presentar una entropía mucho menor.

Se demuestra así, que el predictor de pixel por plano simple realiza una transformación de los datos que es distinguible de una transformación arbitraria, y además, que lo hace de la forma deseada y esperada.

2.6. Predicción de errores en c_s

Aplicado a imágenes con sentido, resulta indudable que la predicción por plano simple contendrá errores, y es deseable averiguar si dichos errores son también predecibles. De ser el caso, es posible incorporar una técnica al predictor por plano simple que mejore su eficiencia.

La forma en que se intentará predecir dicho error, será buscando correlaciones entre él y otras variables involucradas en el proceso.

Las variables elegidas para este análisis son las siguientes:

Error de predicción por plano simple $(c_s - c)$ El error de predicción por el método de plano simple, expresado como la diferencia entre la prediccion y el valor real del pixel por predecir. Es el valor que se busca minimizar.

Diferencia con delta $(\delta - c_s)$ La diferencia entre el valor del pixel δ y el valor de la predicción por plano simple. Se elige con la intención de encontrar alguna correlación entre el error y el pixel δ vecino de c.

Inclinación izquierda $(p - o)$ Se llama de esta forma a la cantidad que representa $p - o$, cumpliendose la siguiente igualdad:

$$p - o = \frac{m\sqrt{2}}{2}$$

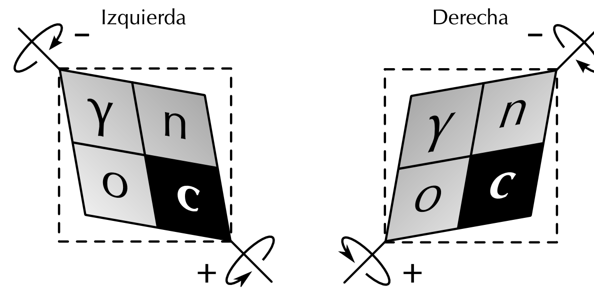

Figura 2.10: Inclinación del plano simple. Esta puede ser izquierda o derecha, según se muestra en el presente esquema. La inclinación del plano simple no es una rotación. Las flechas circulares solo existen para facilitar la comprensión del esquema.

Donde m es la pendiente del segmento de línea dirigido $\overrightarrow{op}$.

La inclinacion izquierda es un indicador de la inclinación del plano simple en la dirección de $\overrightarrow{op}$. Es ortogonal a la inclinación derecha.

Inclinación derecha $(p - \gamma)$ Se llama de esta forma a la cantidad que representa $p - \gamma$, cumpliendose la siguiente igualdad:

$$p - \gamma = \frac{m\sqrt{2}}{2}$$

Donde m es la pendiente del segmento de línea dirigido $\overrightarrow{\gamma p}$.

La inclinacion derecha es un indicador de la inclinación del plano simple en la dirección de $\overrightarrow{\gamma p}$. Es ortogonal a la inclinación izquierda.

Las inclinaciones izquierda y derecha son una herramienta para categorizar al plano simple en función de su gradiente. Este gradiente siempre podrá ser expresado como una combinación lineal de la inclinación izquierda y la inclinación derecha. Esto puede verse con más claridad en la figura 2.10

En este y posteriores análisis se han seguido las mismas directrices que las descritas en la sección 2.5.1.

2.6.1. Distribución de pixeles y errores por tipo de inclinación

Al graficar la inclinación izquierda contra la inclinación derecha, asignando un color diferente según la magnitud del error, se obtiene una gráfica con características típicas, como la presentada en la figura 2.11.

En esta familia de gráficas se descubren las siguientes características:

- Parece existir una correlación entre la inclinación derecha y la magnitud del error, denotada por la prevalencia de errores negativos en la parte inferior de la gráfica, y positivos en la parte superior.

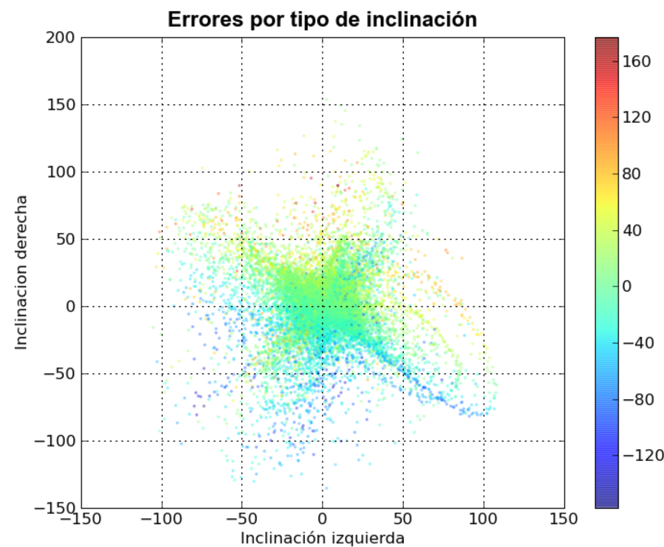

Figura 2.11: Inclinación izquierda vs. inclinación derecha. La magnitud del error por pixel se denota de acuerdo al color mostrado en la escala.

- La distribución de los pixeles no es uniforme, se concentran en el centro, y en aquellos puntos donde el valor absoluto de las inclinaciones es muy similar, formando un patrón en forma de "X". Es posible corroborarlo analizando la densidad de pixeles por grado de inclinación, tal y como se muestra en la figura 2.12

La presencia del patrón en forma de "X", se interpreta como una prueba indirecta de la validez de la definición 2.3.3. Valores absolutos similares en las inclinaciones izquierda y derecha, son los que favorecen la similitud de los pixeles más cercanos sobre los más lejanos en el plano simple.

Si se observan las gráficas de la inclinación derecha vs. error cuando el predictor es aplicado al ruido, es posible apreciar la misma tendencia, como se puede apreciar en la figura 2.13. Los artificios creados al nivel del eje de las abcisas son consecuencia de los valores fuera de rango en la predicción, que han sido ajustados a números válidos.

En el ruido, existe una correlación entre la inclinación derecha y el error, porque a mayor inclinación menor la probabilidad de provocar un error de signo opuesto. Para inclinaciones positivas, se reduce la probabilidad de encontrar un error por escasez (una error negativo), ya que el valor de c_s se aproxima al número máximo que puede asignarse a un pixel, reduciendo la cantidad de pixeles con un valor mayor a él, volviendo más probables los errores

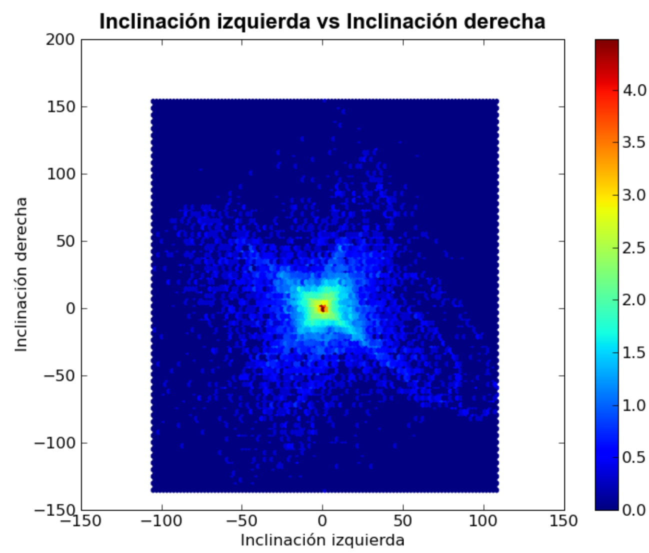

Figura 2.12: Densidad de pixeles por tipo de inclinación correspondiente a la figura 2.11. La densidad se denota de acuerdo al color mostrado en la escala logarítmica.

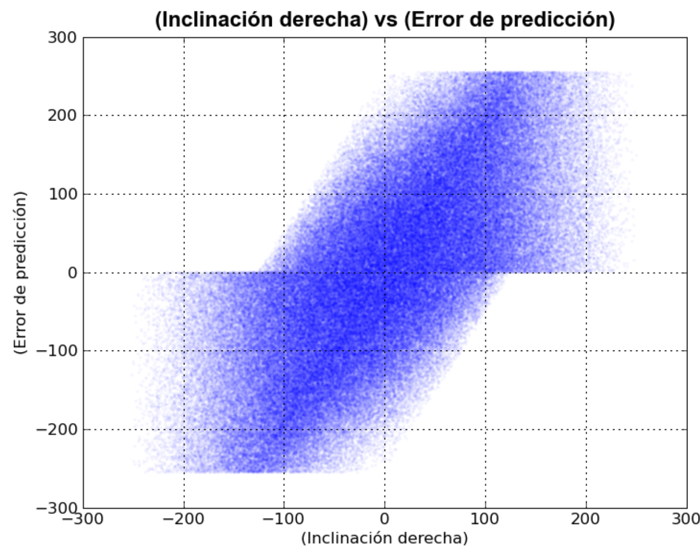

Figura 2.13: Distribución típica de error vs. inclinación derecha utilizando predicción por plano simple aplicado al ruido.

positivos. El caso contrario se produce con inclinaciones derechas negativas, donde predominan los errores negativos.

Este mismo fenómeno se presenta en las imágenes con sentido, a las que se ha aplicado el mismo predictor, tal y como puede verse en la figura 2.14.

Analizando la correlación entre la inclinación derecha y el error de predicción para imágenes con sentido, se obtienen los siguientes datos:

Coeficiente de correlación promedio	**Desviación estándar**
0.4799453	0.1155479

Tabla 2.6: Resumen de la correlación entre la inclinación derecha y el error de predicción por plano simple para imágenes con sentido.

En el total de las imágenes analizadas existe un coeficiente de correlación positiva, que se revela cercano a 0.5, demostrándose así una relación entre dicha inclinación y los errores introducidos por el método de predicción.

Resulta evidente que una corrección del valor predicho, tomando en cuenta lo anterior, debe pasar por utilizar una razón positiva de la inclinación derecha. Sin embargo, se ha considerado irrelevante tomar a consideración la pendiente de la recta de regresión para establecer dicha razón por dos motivos: (1) resulta muy variable entre imágenes y, (2) es recomendable elegir un valor sencillo que

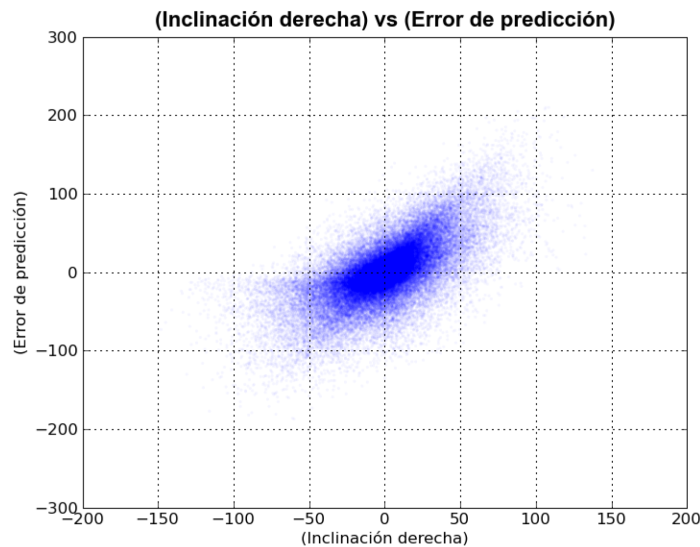

Figura 2.14: Distribución típica de error vs. inclinación derecha utilizando predicción por plano simple aplicado a una imagen con sentido.

favorezca la resolución rápida del problema en la computadora, beneficiando así el rendimiento global del algoritmo.

2.6.2. Análisis de propuestas

Condiciones de pruebas

Se ha elegido probar con tres diferentes variantes del predictor que satisfacen la sencillez mencionada con anterioridad:

Corrección 1 a 1: Por cada unidad de incremento en la inclinación derecha se efectúa un decremento unitario en el valor de la predicción por plano simple. Esto equivale a tomar el valor de p como predicción. Cuando $p \notin \mathbb{N}$, se tomará uno de los dos enteros más próximos a p, eligiendo el más cercano al valor de c_s.

Corrección 1 a 1/2: Por cada unidad de incremento en la inclinación derecha se efectúa un decremento de $1/2$ en el valor de la predicción por plano simple. Cuando esta predicción no sea un valor entero, se tomará uno de los dos enteros más próximos a él, eligiendo el más cercano al valor de p.

Corrección 1 a 3/2: Por cada unidad de incremento en la inclinación derecha se efectúa un decremento de $3/2$ en el valor de la predicción por plano

simple. Cuando esta predicción no sea un valor entero, se tomará uno de los dos enteros más próximos a él, eligiendo el más cercano al valor de c_s.

En los tres casos se aplicará la corrección final, mencionada con anterioridad en este trabajo, para evitar los valores fuera del dominio de c.

Por otra parte, se clasificará a los pixeles según el gradiente del plano simple asociado a ellos, en una forma que resulta plausible una diferencia sustancial en el comportamiento del predictor, según los resultados obtenidos hasta ahora. Las clasificaciones establecidas son las siguientes:

Pixeles tipo X: Son aquellos que poseen un plano simple en el cual el valor absoluto de la inclinación derecha e izquierda es el mismo, o bien, difieren en una unidad. Aquí se concentran la gran mayoría de los pixeles de una imagen típica.

Pixeles tipo U: Son los pixeles que tienen un plano simple donde el valor absoluto de la inclinación derecha es mayor al de la izquierda en más de una unidad.

Pixeles tipo L: Son los pixeles que tienen un plano simple donde cual el valor absoluto de la inclinación izquierda es mayor al de la derecha en más de una unidad.

Resultados

En primera instancia se ha calculado la entropía promedio de cada método propuesto por cada uno de los tipos de pixel establecidos, así como la desviación estándar de los mismos, obteniendose los resultados que se encuentran en las tablas 2.7 y 2.8 respectivamente.

Predictor	**Entropía promedio**			
	X	**U**	**L**	**Todos**
Plano simple	3.4137	5.1118	4.8439	4.2026
Corrección 1 a 1	3.4093	5.0333	4.9954	4.1703
Corrección 1 a 1/2	3.3399	4.7924	4.8901	4.0481
Corrección 1 a 3/2	3.5735	5.5286	5.1672	4.4476

Tabla 2.7: Entropía promedio por variante de método de predicción.

Como es posible apreciar, para todo tipo de pixel, la corrección de 1 a 1/2 funciona mejor que cualquier otro de los predictores listados, excepto en los pixeles de tipo L, donde el mejor predictor resulta ser el predictor por plano simple tradicional.

Predictor	**Desviación estándar de la entropía**			
	X	**U**	**L**	**Todos**
Plano simple	0.7193	0.8698	0.9035	0.8729
Corrección 1 a 1	0.7405	0.8130	0.8741	0.8431
Corrección 1 a 1/2	0.7190	0.8393	0.8744	0.8355
Corrección 1 a 3/2	0.7757	0.8173	0.8785	0.8820

Tabla 2.8: Desviación estándar de la entropía por variante de método de predicción.

Buscando correlación con δ

Utilizando el mejor predictor de los listados según el tipo de pixel, se realizó una prueba similar a la anterior. En este caso, analizando una posible correlación entre el error y la diferencia de la predicción con el valor del pixel δ. Los resultados se presentan el la tabla 2.9

Tipo de pixel	**Correlación promedio (error vs. predicción - δ)**	**Desviación estándar de la correlación**
X	0.1507	0.1516
U	0.2821	0.2166
L	0.0873	0.1168

Tabla 2.9: Correlaciones y sus promedios de error vs. diferencia con δ.

Se descubre que existe poca correlación entre el error de predicción y la diferencia de la predicción con el pixel δ, aunque se destaca que la correlación en todos los promedios es siempre positiva, existiendo un pequeño despunte en los pixeles de tipo U, donde la correlación alcanza 0.28.

Considerando lo anterior, se realiza una prueba donde se aplique el predictor por plano simple con corrección 1 a $1/2$ a los pixeles tipo U, y además, una corrección similar que considere la correlación del error con el valor de δ mencionada anteriormente. Especificamente, se altera el valor de la predicción con la mitad de la diferencia de la predicción con δ.

El resultado no se revela satisfactorio, obteniendo una entropía promedio de 4.8690, resultando mayor a las ya alcanzadas anteriormente.

2.7. Plano simple secundario

En el intento de involucrar al pixel δ en el cálculo de la predicción, se intenta una nueva estrategia creando un segundo plano simple que involucre el valor de dicho pixel. Este segundo plano simple, que será llamado **plano simple secundario**, sigue las mismas premisas utilizadas para la definición del plano simple. La diferencia radica en los pixeles utilizados en su definición.

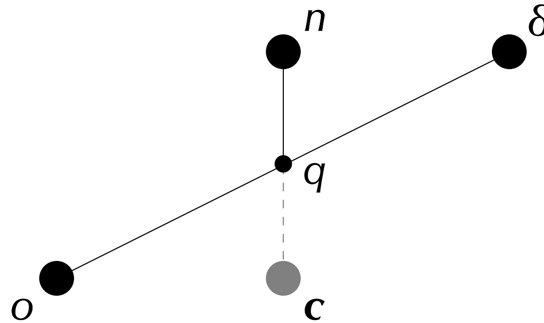

Figura 2.15: Posición relativa de los pixeles vecinos en el plano simple secundario.

La definición 2.4.2 utilizada para el plano simple se sustituye por la siguiente:

Definición 2.7.1. El plano común al que pertenecen los pixeles n, δ y o de un pixel c cualquiera, es el **plano simple secundario** de c.

Resulta fácil descubrir que, si bien este plano no tiene forma cuadrada, se cumplen todas y cada una de las relaciones entre los pixeles involucrados, si se considera la siguente tabla de equivalencias:

Puntos equivalentes entre planos	
Plano simple	**Plano simple secundario**
γ	n
n	δ
o	o
c	c
p	q

Tabla 2.10: Puntos equivalentes entre plano simple y plano simple secundario.

Establecida la equivalencia precedente, resulta inmediato descubrir que el punto equivalente a p, que en este caso se ha llamado q, es el punto medio entre el pixel n y el c que intentamos predecir. La distribución de los pixeles involucrados se puede contemplar en la figura 2.15.

Gracias a 2.3.3 y 2.3.4 se sabe que el nuevo plano simple secundario es menos efectivo que el anterior, pues existe una mayor distancia entre los pixeles involucrados. Su aporte al problema está en involucrar información no contemplada en el método de plano simple. Es decir, el valor del pixel δ.

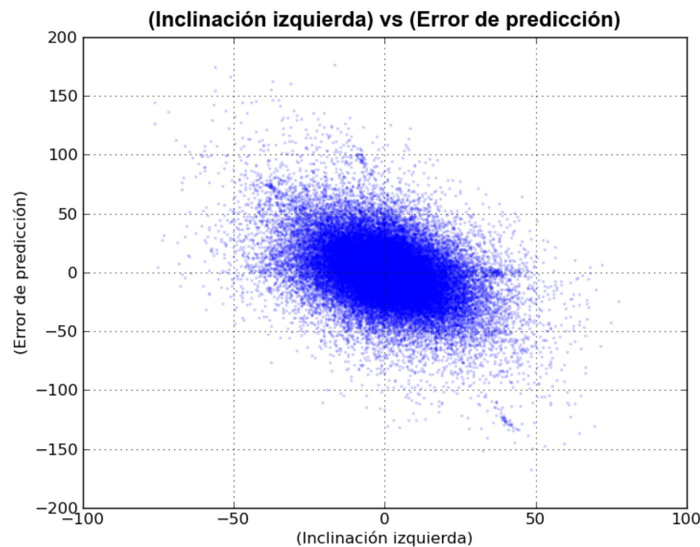

Figura 2.16: Distribución típica de error vs. inclinación izquierda utilizando predicción por plano simple secundario aplicado a una imagen con sentido.

2.7.1. Predicciones con el plano simple secundario

La aplicación inmediata del plano simple secundario, radica en utilizar el mismo algoritmo de predicción usado en el método de plano simple, con la salvedad de aplicarlo a diferentes pixeles adyacentes a c.

Como es de esperar, los resultados son más pobres que en el caso del plano simple tradicional, aunque es de resaltar el comportamiento de los errores respecto a las inclinaciones izquierda y derecha del plano simple.

Esto se muestra evidente analizando las figura 2.17 y 2.16.

De forma análoga al método por plano simple, se han probado dos grados de corrección, en base a la correlación existente entre la inclinación izquierda y el error en la predicción. Las varientes son:

Corrección 1 a 1: Por cada unidad de incremento en la inclinación izquierda se efectúa el incremento en una unidad en el valor de la predicción por plano secundario.

Corrección 1 a 1/2: Por cada unidad de incremento en la inclinación izquierda se efectúa el incremento de un medio en el valor de la predicción por plano secundario.

Para ambos casos, se siguen las mismas medidas para valores fuera de rango y se redondea al entero más próximo a p.

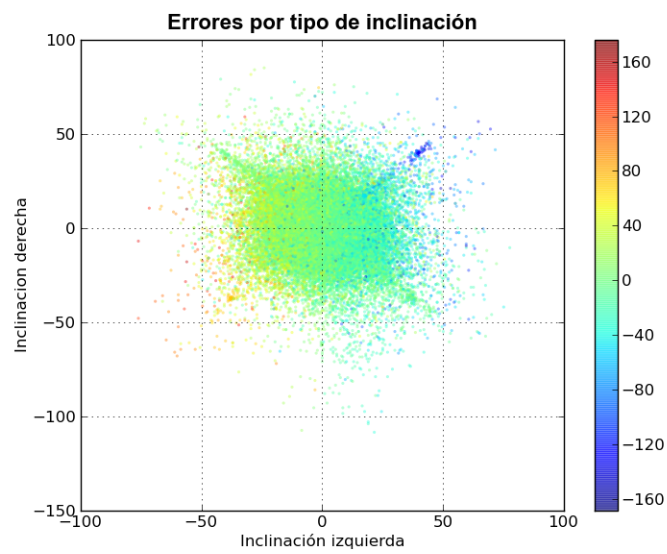

Figura 2.17: Inclinación izquierda vs. inclinación derecha para predicción por plano secundario. La magnitud del error por pixel se denota de acuerdo al color mostrado en la escala.

Promedios para predictores por plano simple secundario		
	Entropía	**Desviación estándar**
Plano simple secundario	4.6348	0.9280
Con corrección 1 a 1	4.5864	0.8695
Con corrección 1 a 1/2	4.5398	0.9014

Tabla 2.11: Promedios para predictores por plano simple secundario aplicado a todo tipo de pixel.

Las entropías promedio y las desviaciones estándar promedio se muestran en la tabla 2.11.

De forma general, el método con corrección 1 a 1/2 ha dado mejores resultados, sin llegar a superar los métodos anteriores basados en el plano simple.

2.7.2. Combinación de los métodos de plano simple y plano simple secundario

Utilizando de forma combinada los métodos de plano simple y plano simple secundario, se intenta involucrar al valor del pixel δ en el cálculo de la predicción. Se ha elegido para este propósito el promedio de ambos predictores de la siguiente manera:

- Promedio del método se plano simple con corrección 1 a 1/2 y plano simple secundario con corrección 1 a 1.
- Promedio del método se plano simple con corrección 1 a 1/2 y plano simple secundario con corrección 1 a 1/2.

El método que ha dado mejores resultados ha sido con la utilización de la corrección 1 a 1/2 por parte del método de plano simple secundario.

2.8. Elección de los predictores

El método de elección consistirá en elegir el mejor predictor de los ya analizados, según el tipo de pixel involucrado (tipo X, U o L).

En las tablas 2.12 y 2.13 se listan las entropias promedio por predictor y tipo de pixel, asi como la desviación estándar de las mismas.

Los predictores asociados a los mejores resultados serán elegidos para el tipo de pixel correspondiente en la implementación final. Estos son:

Pixel tipo X Predictor por plano simple con corrección 1 a 1/2

La predicción es c_e, donde:

$$c_e = \frac{p + n + o - \gamma}{2}$$

Entropías promedio			
	X	**U**	**L**
Plano simple	3.4137	5.1118	4.8439
Pixel gamma	3.8995	6.0318	5.3975
Plano simple (1 a 1)	3.4093	5.0333	4.9954
Plano simple (1 a 3/2)	3.5735	5.5286	5.1672
Plano simple (1 a 1/2)	3.3399	4.7924	4.8901
Plano secundario	3.9641	5.1931	5.6941
Plano simple y plano secundario	3.5515	4.8677	5.1746
Plano secundario (1 a 1/2)	3.8835	5.1272	5.5584
Psimple(1 a 1/2) Psecundario (1 a 1)	3.5758	4.7834	5.2005
Psimple(1 a 1/2) Psecundario (1 a 1/2)	3.5460	4.7737	5.1439

Tabla 2.12: Entropias promedio por predictor y tipo de pixel.

Desviaciones estándar promedio			
	X	**U**	**L**
Plano simple	0.7193	0.8698	0.9035
Pixel gamma	0.7938	0.7761	0.8654
Plano simple (1 a 1)	0.7405	0.8130	0.8741
Plano simple (1 a 3/2)	0.7757	0.8173	0.8785
Plano simple (1 a 1/2)	0.7190	0.8393	0.8744
Plano secundario	0.8501	0.9089	0.9064
Plano simple y plano secundario	0.7765	0.8977	0.9080
Plano secundario (1 a 1/2)	0.8371	0.8813	0.8687
Psimple(1 a 1/2) Psecundario (1 a 1)	0.7425	0.8531	0.8400
Psimple(1 a 1/2) Psecundario (1 a 1/2)	0.7564	0.8683	0.8758

Tabla 2.13: Desviaciones estándar promedio por predictor y tipo de pixel.

$$p = \frac{n + o}{2}$$

Si el valor de c_e no es entero, se redondeara al entero más próximo a p.

Pixel tipo U Predictor por promedio de plano simple con corrección 1 a 1/2 y plano secundario con corrección 1 a 1/2

La predicción es c_e donde:

$$c_e = \frac{\gamma + o}{4} + \frac{\delta - n}{2}$$

Si el valor de c_e no es entero, se redondea al entero más próximo a c_s. Donde:

$$c_s = n + o - \gamma$$

Para los casos donde el pixel δ no está disponible, se aplica el mismo predictor que el usado para los pixeles tipo X.

Pixel tipo L Predictor por plano simple

La predicción es c_e, donde:

$$c_e = n + o - \gamma$$

Se debe anotar que para todos los casos se ha establecido que:

- Si el valor de c_e es estrictamente mayor al límite superior del rango que puede almacenar un pixel, entonces, c_e tomará el valor de dicho límite superior.
- Si el valor de c_e es estrictamente menor al límite inferior del rango que puede almacenar un pixel, entonces, c_e tomará el valor de dicho límite inferior.

Capítulo 3

Implementación y pruebas

Toda obra es deleznable, sólo su ejecución no lo es.

Thomas Carlyle

3.1. La necesidad de una especificación de mapa de bits

Con el propósito de poner a prueba el predictor de pixel desarrollado en el capítulo precedente, es necesaria su implementación en un sistema que permita su aplicación en un entorno de producción. Una especificación de mapa de bits es imperativa, considerando que no existe especificación alguna que use el algoritmo propósito de este trabajo. La presentada a continuación pretende ser independiente de los capitulos precedentes, de forma que no sea necesaria su consulta para la comprensión total del mismo, si bien es posible que existan referencias a información mencionada en ellos.

La especificación propuesta no pretende cubrir todos y cada uno de los aspectos que abarca una especificación típica. Es solo un mécanismo a través del cual es posible el empleo del predictor de pixel.

3.2. Especificación SPB

3.2.1. Convenciones

Definiciones

Se aplican las siguientes definiciones en la siguiente especificación, a fin de evitar la ambiguedad en los términos:

Alpha: Un valor que representa el grado de opacidad de un pixel. A mayor opacidad, mayor capacidad para ocultar el color de fondo sobre el cual

se presenta el pixel. Un alpha con valor cero representa un pixel completamente transparente, mientras que un alpha con el valor máximo posible representa un pixel completamente opaco.

Canal: Un arreglo con toda la información de un tipo particular que pertenece a la imagen. Hay cinco tipos de información: *rojo*, *verde*, *azul*, *gris* y *alpha*.

Flujo de datos: Una sequencia de bytes.

Flujo de datos SPB: La sequencia de bytes que representa la totalidad de una imagen SPB.

Imagen de referencia: Una arreglo rectangular de pixeles, donde todos poseen el mismo número y tipo de muestras. Cada imagen de referencia puede ser representada como un flujo de datos que contiene la imagen en formato SPB, y cada flujo de datos de este tipo puede ser convertido en una imagen de referencia.

Muestra: Intersección de un canal y un pixel.

Pixel: Consiste una sequencia de muestras de todos los canales. La imagen completa es un arreglo rectangular de pixeles.

Trozo: Una sección de un flujo de datos. La mayoría de los trozos incluyen información. El significado y tipo de información almacenada en él depende del tipo de trozo.

Orden de lectura

El orden de lectura y escritura de los pixeles de un mapa de bits que se establece es el siguiente:

- El primer pixel es el situado en el extremo derecho de la línea superior de la imagen.
- El consecutivo de cualquier pixel será siempre el situado a su derecha, si es que existe. De no existir se optará por el situado en el extremo izquierdo de la línea inmediatamente inferior. Si tampoco existe un pixel con esas características, se dice entonces que se ha alcanzado el último pixel.

Pixeles adyacentes

Las muestras adyacentes a una muestra c estarán especificadas de acuerdo a la figura 3.1.

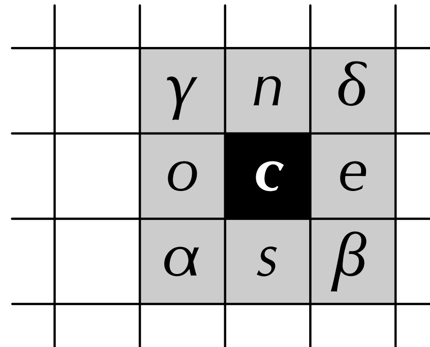

Figura 3.1: Muestras adjacentes. Donde los cuadros en gris son vecinos de la muestra *c*.

3.2.2. El formato SPB

Se propone la especificación del formato SPB (*Simple Purpose Bitmap*). Dicho formato consiste en los siguientes tipos de imagen:

Escala de grises: Cada pixel consiste en una muestra: *gris*.

Escala de grises con canal alpha: Cada pixel consiste en dos muestras: *gris* y *alpha*.

Color verdadero: Cada pixel consiste en tres muestras: *rojo*, *verde* y *azul*.

Color verdadero con canal alpha: Cada pixel consiste en cuatro muestras: *rojo*, *verde*, *azul* y *alpha*.

Descripción del proceso de codificación de una imagen SPB

Los pasos que se siguen para la codificación de una imagen SPB son los siguientes:

1. Se recibe una imagen de referencia que representa a la imagen original.
2. Se generan los trozos que serán el inicio del flujo de datos correspondiente a la imagen SPB. Dichos trozos contienen información sobre el tamaño, profundidad de color y metadatos de lo que será la imagen codificada.
3. Por cada pixel de la imagen de referencia se genera una predicción para cada una de sus muestras, generandose un flujo de datos de dichas predicciones.
4. El flujo de datos generado en el paso anterior es comprimido.

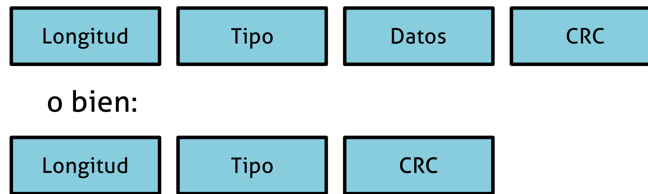

Figura 3.2: Esquema de los componentes de un trozo.

5. El nuevo flujo de datos obtenido de la compresión se anexa al generado en el paso 2. A este nuevo flujo, producto de los dos anteriores, se anexa un nuevo trozo de terminación. Este nuevo flujo, que es resultado de todas las operaciones previas, es la imagen SPB.

3.2.3. Estructura del flujo de datos

Introducción

Aquí se describen de manera detallada los componentes que constituyen un flujo de datos SPB, así como las propiedades de los trozos que lo conforman.

Firma SPB

Al inicio de todo flujo de datos SPB se encuentran la siguiente secuencia de bytes (en hexadecimal):

```
1 53 50 42 49 4D 41 47 45
```

Esta firma indica que el resto del flujo de datos contiene una imagen SPB, flujo que inicia con un trozo tipo `STRT` y termina con un trozo tipo `ENDC`.

Esquema de trozo

Cada trozo consiste en tres o cuatro campos, tal como se describe en la figura 3.2. El significado de cada campo viene dado en la tabla 3.1.

Tipos de trozo

Los tipos de trozo disponibles son los siguientes:

STRT Trozo de inicio, es el primer trozo en una imagen SPB (después de la firma SPB).

TYPE Especifica el tipo de imagen del cual se trata.

SIZE Indica el tamaño de la imagen en pixeles.

Campos de trozo	
Longitud	Un entero sin signo de 4 bytes (little-endian) que indica el tamaño en bytes del campo de datos del trozo. Este campo solo toma en cuenta los bytes asignados al campo de datos y los de ningún otro. 0 es una longitud válida.
Tipo	Una secuencia de 4 bytes que indica el tipo de trozo del que se trata. Los valores de estos bytes están restringidos a los valores decimales 65 a 90 y 97 a 122, los cuales se corresponden a las letras a-z y A-Z en el código ASCII tradicional.
Datos	Contiene los datos apropiados al tipo de trozo del que se trate. Puede ser de longitud 0.
CRC	Un CRC32 (Cyclic Redundancy Code) númerico de 4 bytes (little-endian) calculados a partir de todos los campos predecentes. Los campos CRC son calculados utilizando los métodos estandarizados y definidos en [14].

Tabla 3.1: Descripción de los campos que constituyen a un trozo.

META Contiene información textual asociada a la imagen.

DATA Contiene los datos de la imagen propiamente dicha.

ENDC Trozo final, es el último trozo de una imagen SPB, con el cual debe concluir el flujo de datos.

El orden de aparición de los trozos en el flujo de datos es el mismo que se muestra en la lista precedente. Todos los trozos son obligatorios. El nombre del trozo mencionado en la lista precedente se corresponde con los bytes que forman parte del campo **tipo** de dicho trozo.

Especificaciones para Trozos

STRT El campo longitud de este trozo siempre indicará cero, y su campo de datos será inexistente. Tan solo tiene el propósito de indicar el inicio del flujo de datos SPB.

TYPE Existen 4 tipos de imagenes SPB, a cada uno de los cuales se le asignará un valor según está descrito en la tabla 3.2.

El valor será almacendado como el carácter ASCII correspondiente en el campo **datos** del trozo.

SIZE Este trozo tiene el propósito de indicar el tamaño en pixeles de la imagen. Indica dos valores: el número de columnas y el número de filas que conforman la matriz de pixeles asociada a la imagen de referencia.

Tipo de imagen SPB	Valor	ASCII (Decimal)
Escala de grises	1	49
Escala de grises con alpha	2	50
Color verdadero	3	51
Color verdadero con alpha	4	52

Tabla 3.2: Valor asociado a cada tipo de imagen SPB en el trozo TYPE correspondiente.

Los valores son almacenados como enteros sin signo de 32 bits (little-endian), concatenando los valores en el campo **datos**, lo cual implica una longitud de 8 bytes para el mismo. Los primeros 4 bytes corresponden al número de columnas y los restantes 4 bytes corresponden al número de filas.

Son válidos valores de cero, si bien esto implica la inexistencia de una imagen, y por tanto, los demás trozos deberán ser consistentes con dicha información.

Considerando que son utilizados 32 bits por cada parámetro, se deduce que el tamaño máximo teórico permitido para una imagen es de 4,294,967,296 x 4,294,967,296 pixeles.

META El trozo `META` almacena información textual asociada a la imagen. El contenido del campo **datos** es abierto y no existen restricciones para el mismo, siempre que no supere el tamaño máximo permitido por las características del campo **longitud**.

DATA El campo **datos** de este trozo contiene toda la información asociada al mapa de bits salvo sus medidas, que se encuentran en el trozo `SIZE`. Los datos almacenados aquí han pasado previamente por un proceso de transformación y compresión. Dichos procesos se detallan más adelante.

ENDC Este trozo solo tiene la finalidad de indicar el final del flujo de datos asociado a la imagen SPB. Como el trozo `STRT`, el campo longitud de este trozo siempre indicará cero, y su campo de datos será inexistente.

3.2.4. Imagen de referencia a imagen SPB

Aquí se listan los pasos necesarios para convertir una imagen de referencia a una imagen SPB.

Tipo de imagen

Es necesario conocer el tipo de imagen de la cual se trata, y este debe ser alguno de los mencionados en la tabla 3.2. De esta forma se conocerá el número y tipo de muestras por pixel del mapa de bits.

Esta información se utilizará para elegir correctamente el método de transformación de la imagen, así como el llenado correcto del trozo `TYPE` de la imagen SPB, tal cual se ha especificado con anterioridad.

Tamaño de imagen

Resulta indispensable conocer el tamaño en pixeles de la matriz asociada a la imagen referencia, pues solo así podrá aplicarse correctamente la transformación y posterior representación de la imagen SPB.

El tamaño consta de dos valores: número de columnas y número de filas. En ese orden y siguiendo las especificaciones marcadas para el trozo `SIZE`.

Extracción y transformación de datos

Se analizarán las muestras pixel por pixel, siguiendo el orden de lectura estipulado previamente. Las muestras de cada pixel serán transformadas en el siguiente orden, siempre que existan y se encuentren presentes:

1. *Gris*
2. *Rojo*
3. *Verde*
4. *Azul*
5. *Alpha*

El resultado de la transformación de las muestras será almacenada en el mismo orden en que es generada, de forma que todas las muestras del mismo pixel se encuentren siempre adyacentes. Asi, por ejemplo, un pixel transformado de una imagen tipo 3 *(Color verdadero)*, contendrá las muestras transfromadas del canal *rojo*, *verde* y *azul*, en ese orden.

Compresión

Al flujo datos resultado de la transformación calculada previamente, se le aplicará una compresión gzip de nivel 9.

3.2.5. Transformación de pixel

Cada muestra será transformada siguiendo uno de cinco posibles métodos de predicción de pixel:

Valor por defecto Aplica a la primer muestra de cada canal, cuando no existen muestras previas que puedan ser analizadas para obtener una predicción.

Pixel anterior Se utiliza para todas las muestras que forman parte de la primer fila del mapa de bits, excepto la primer muestra.

Pixel superior Se utiliza para todas las muestras que forman parte de la primer columna del mapa de bits, excepto la primer muestra.

Garduño 3 pixeles Predicción utilizada para todas las muestras de la última columna del mapa de bits, siempre que no pertenezcan a la primera fila.

Garduño 4 pixeles Aplica para todas las muestras restantes.

Valor por defecto

Se asume que el valor de la predicción es igual al valor entero de $a/2$. Donde a es igual al valor máximo que puede tomar una muestra. En el caso que nos ocupa, donde el valor de a es igual a 255, el valor de la predicción siempre será 128.

El propósito es minimizar el máximo valor absoluto del error en la predicción.

Pixel anterior

La predicción es el valor del pixel adyacente izquierdo o.

Pixel superior

La predicción es el valor del pixel adyacente superior n.

Garduño 3 pixeles

El algoritmo para calcular la predicción es el siguiente:

- Si el plano simple asociado al pixel es de tipo L se usara la siguiente fórmula para calcular el valor estimado del pixel, llamado c_e:

$$c_e = n + o - \gamma$$

- En caso contrario se seguirán las siguientes definiciones:

$$p = \frac{n + o}{2}$$

$$c_e = \frac{p + n + o - \gamma}{2}$$

Si el valor de c_e no es entero, se redondeara al entero más próximo a p.

Garduño 4 pixeles

El algoritmo para calcular la predicción es el siguiente:

- Si el plano simple asociado al pixel es de tipo L se usara la siguiente fórmula para calcular el valor estimado del pixel, llamado c_e:

$$c_e = n + o - \gamma$$

- Si el plano simple asociado al pixel es de tipo X se seguirán las siguientes definiciones:

$$p = \frac{n + o}{2}$$

$$c_e = \frac{p + n + o - \gamma}{2}$$

  Si el valor de c_e no es entero, se redondeara al entero más próximo a p.

- Si el plano simple asociado al pixel es de tipo U se seguirá la siguiente fórmula:

$$c_e = \frac{\gamma + o}{4} + \frac{\delta - n}{2}$$

  Si el valor de c_e no es entero, se redondeara al entero más próximo a c_s. Donde:

$$c_s = n + o - \gamma$$

3.2.6. Asunciones para la transformación de pixel

A fin de mantener un comportamiento idéntico para cualquier posible implementación de esta especificación, se asume que el sistema tiene la capacidad para asegurar el manejo de números de punto flotante con, al menos, tres dígitos en la mantisa asignados a los números binarios significativos menores a la unidad. Todo esto, con el fin de asegurar que en todas las operaciones no existen truncamientos ni redondeos no especificados explícitamente.

3.2.7. Tipos de plano

Pixeles tipo X: Son aquellos que poseen un plano simple en el cual el valor absoluto de la inclinación derecha e izquierda es el mismo, o bien, difieren en una unidad.

Pixeles tipo U: Son los pixeles que tienen un plano simple donde el valor absoluto de la inclinación derecha es mayor al de la izquierda en más de una unidad.

Pixeles tipo L: Son los pixeles que tienen un plano simple donde cual el valor absoluto de la inclinación izquierda es mayor al de la derecha en más de una unidad.

Inclinaciones derecha e izquierda

Inclinación izquierda Se llama de esta forma a la cantidad que representa:

$$\frac{n-o}{2}$$

Inclinación derecha Se llama de esta forma a la cantidad que representa:

$$\frac{n+o}{2}-\gamma$$

3.3. Implementación

Con el propósito de llevar a la práctica y poner a prueba el desarrollo de los capítulos precedentes, se ha implementado la especificación SPB descrita arriba. Al mismo tiempo, se ha implementado una especificación similar que utiliza un predictor similar, previamente existente, que es de uso general en la compresión de mapas de bits.

3.3.1. Elección del lenguaje

Para llevar a cabo tal empresa, se ha elegido el lenguaje interpretado Python. Se advierte que dicho lenguaje no es el recomendado para una implementación seria de algo como un programa que almacena y lee mapas de bits. El escenario ideal está compuesto por un lenguaje de bajo nivel que permita la creación de un binario que ejecute las operaciones lo más rápidamente posible, al mínimo coste computacional. Sin embargo, las pruebas se han concebido para mostrar la eficiencia del predictor con respecto a otros algorítmos, sin importar el tiempo de ejecución. De forma que se vuelve irrelevante. Se sabe de antemano que es más costoso computacionalmente, tal y como se expresa en la introducción de este trabajo.

Las principales razones para la elección del lenguaje Python son:

- Es multiplataforma, de manera que es posible ejecutar y desarrollar con amplias libertades de sistema operativo y hardware.
- Es software libre, lo que permite su uso sin el pago de regalías o licencias.

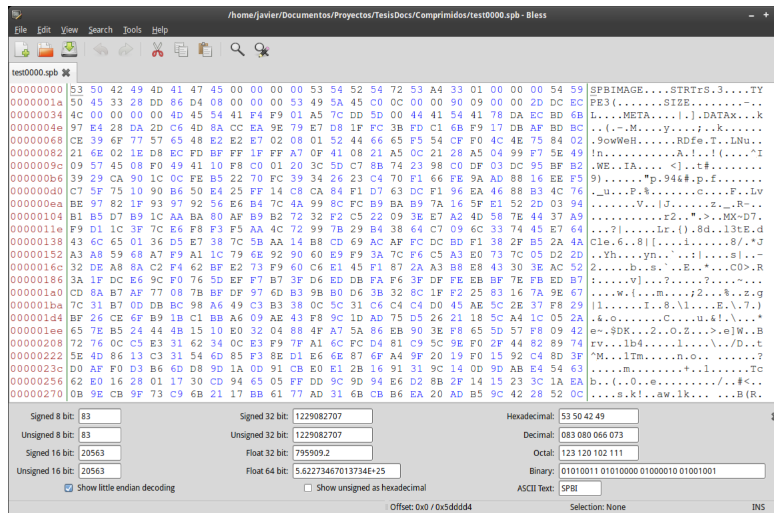

Figura 3.3: Los primeros bytes de una imagen SPB vistos a través de un editor hexadecimal.

- Posee una poderosa librería para la manipulación de mapas de bits, lo que permite fácil acceso, así como lectura y escritura de formatos de imagen conocidos. Esta librería también es software libre.
- Es un lenguaje de alto nivel de fácil lectura y rápido desarrollo. Sus programas parecen pseudocódigo, lo que facilita su comprensión para terceras personas.

3.4. Pruebas

Se ha tomado una muestra de 33 imágenes diferentes a las utilizadas en el desarrollo del predictor, a las que se ha sometido a dos procesos de codificación:

- Codificación como imagen SPB (SPB), tal cual se describe en la especificación precedente.
- Codificación como variante de imagen SPB (SPBtest), que es idéntica a la codificación SPB tradicional, excepto en el proceso de filtrado de pixeles, donde se utiliza el predictor de Paeth siempre que es posible. Para el resto de pixeles se utilizan los mismos predictores que en la especificación SPB.

Los resultados se muestran en la tabla 3.3.

Imagen	**SPBtest**	**SPB**	**SPB/SPBtest**
0	6,399,819	6,151,637	0.961
1	16,793,362	15,787,152	0.940
2	2,518,255	2,399,356	0.953
3	23,805,772	22,536,898	0.947
4	16,791,644	15,556,930	0.926
5	6,344,894	6,187,642	0.975
6	8,023,638	7,715,144	0.962
7	44,676,151	43,563,402	0.975
8	16,984,991	15,886,171	0.935
9	11,760,360	10,667,948	0.907
10	6,540,183	6,157,838	0.942
11	27,749,431	27,534,558	0.992
12	14,210,413	13,256,127	0.933
13	9,912,544	9,422,133	0.951
14	19,399,412	18,176,219	0.937
15	15,793,593	15,341,207	0.971
16	6,202,777	6,219,708	1.003
17	6,148,379	5,766,102	0.938
18	7,128,750	6,845,181	0.960
19	5,446,897	5,116,772	0.939
20	6,469,015	6,130,908	0.948
21	9,730,167	9,092,717	0.934
22	5,540,647	5,295,561	0.956
23	6,397,765	5,892,107	0.921
24	11,977,202	10,663,550	0.890
25	7,024,793	7,013,593	0.998
26	4,751,957	4,431,724	0.933
27	8,244,812	7,812,771	0.948
28	27,094,664	25,811,650	0.953
29	11,923,112	10,854,181	0.910
30	22,616,283	20,997,765	0.928
31	22,887,959	21,642,637	0.946
32	928,870	1,040,294	1.120

Tabla 3.3: Tamaños en bytes de imágenes por tipo de compresión.

Como es posible comprobar, a excepción de la imagen número 16 y 32, todas y cada una de ellas ha resultado en un tamaño de archivo menor al conseguido utilizando el predictor de Paeth. Considerando que los mapas de bits son una muestra aleatoria de imágens con sentido, está más allá de toda duda razonable que el predictor desarrollado aquí supera en desempeño al conocido como Paeth. El resultado de esta simple prueba corrobora los resultados previamente obtenidos en el proceso de creación del predictor de mapa de bits.

Conclusiones

Los resultados obtenidos indican que el predictor de pixel desarrollado en este documento, si bien es computacionalmente más caro que sus competidores típicos, resulta en un algorítmo más eficiente en la mayoría de los casos estudiados. Al menos, en lo que concierne a reducir el nivel de entropía de los mapas de bits. El progresivo aumento en la velocidad de ejecución que presentan las computadoras hoy día, diluye el impacto aparente en el tiempo de ejecución del mismo. El crítico podría sugerir que el progresivo aumento en la capacidad de almacenamiento, diluye cualquier ventaja que una mejora en la compresión podría presentar. Sin embargo, pronunciarse sobre este tópico no es tan simple, ni resulta ser un tema trivial. El costo en el transporte, sea computacional o energético, asi como otros, deben ser tomados en cuenta en un sin fín de situaciones diferentes, en un análisis del balance total de costos y beneficios del sistema de compresión. Tema que, por sí mismo, resulta un interesante trabajo de investigación.

La parte nuclear de esta tesis, más allá de la implementación o las pruebas desarrolladas a los algoritmos candidatos, reside en la abstracción formulada para comprender el concepto de *imagen*, así como de *imagen con sentido*. Resultan no solo elementales, sino también, parte importante del éxito obtenido en este trabajo. Sin una definición precisa de estos conceptos, hubiera resultado imposible todo el desarrollo posterior, volviéndolo impracticable.

La capacidad de discriminar las *imagenes con sentido* del resto permite explotar rasgos elementales, características que hacen de una imagen un patrón reconocible, en beneficio del predictor. Sin perder generalidad, este mecanismo, o una extensión de eĺ, es proclive a ser empleado en una multitud de aplicaciones, que van más allá de este algoritmo de transformación en particular.

Que el algoritmo predictor incluya en sí mismo un mecanismo que permita discriminar los pasos a seguir, según si se está analizando la parte interior de la imagen o sus orillas, resulta poco común y es parte del éxito de su aplicación. El uso de cuatro pixeles adyacentes de manera simultánea, siempre que es posible hacerlo, resulta novedoso, ya que lo usual es la utilización de sólo tres. Esto aumenta la cantidad de información disponible para el predictor, acrecentando las probabilidades de realizar una predicción correcta. El árbol de decisiones que forma parte del algorítmo le da gran versatilidad, quie si bien aumenta su complejidad, no implica un aumento significativo del costo computacional involucrado en cada cálculo.

Apéndice

Implementación de SPB en Python

Estructura de directorios

- SPB Encoder
    - src
        - SPB
            - codec.py
            - const.py
            - error.py
            - filters.py
            - __init__.py
            - utils.py

Código fuente

codec.py

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

import struct
import zlib
import Image
import SPB.error as error
import SPB.const as const
import SPB.filters as filters

# ----- UTILS ----------

def modalDiff(predict, real):
    """modalDiff(predic, real) -> integer

    Devuelve la diferencia (módulo MAX_PIXEL_VALUE)
    entre el valor real y la predicción.
    """

    if predict <= real:
        return real - predict
```

```
22      else:
23          return (const.MAX_PIXEL_VALUE + 1) - (predict - real)
24
25  def modalSum(predict, value):
26      """modalSum(predict, value) - > integer
27
28      Devuelve la suma (módulo MAX_PIXEL_VALUE)
29      de la predicción con el valor real."""
30
31      real = predict + value
32
33      if real > const.MAX_PIXEL_VALUE:
34          real = real - (const.MAX_PIXEL_VALUE + 1)
35
36      return real
37
38  # ----- ENCODER --------
39
40  def writeChunk(file_pointer, chunk_type, chunk_data):
41      """writeChunk(file_pointer, chunk_type, chunk_data) -> none
42
43      Escribe un trozo de imagen SPB"""
44
45      chunk_len = len(chunk_data)
46      chunk = struct.pack("<I", chunk_len) + chunk_type + chunk_data
47
48      file_pointer.write(chunk + struct.pack("<I", zlib.crc32(chunk) & 0xffffffff))
49
50  def getPILAttrib(PIL_image):
51      """getPILAttrib(PIL_image) -> [modo, x_col, y_fil]
52
53      Regresa una lista con el modo de imagen y el número de
54      filas y columnas de la imagen. Todo en el formato utilizado
55      por el archivo SPB"""
56
57      if PIL_image.mode == "L":
58          modo = "1"
59      elif PIL_image.mode == "LA":
60          modo = "2"
61      elif PIL_image.mode == "RGB":
62          modo = "3"
63      elif PIL_image.mode == "RGBA":
64          modo = "4"
65      else:
66          modo = "0"
67
68      x_col, y_fil = PIL_image.size
69
70      image_attrib = (modo, struct.pack("<I", x_col), struct.pack("<I", y_fil))
71
72      return image_attrib
73
74  def saveSPB(file_path, PIL_image, image_comment):
75      """saveSPB(file_path, PIL_image, image_comment) -> integer
76
77      Guarda en disco una imagen en formato SPB"""
78
79      file_pointer = open(file_path, "wb")
80
81      image_attrib = getPILAttrib(PIL_image)
82
83      if ord(image_attrib[0]) > 48 and ord(image_attrib[0]) < 53:
84          file_pointer.write("SPBIMAGE")
85          makeSTRT(file_pointer)
86          makeTYPE(file_pointer, image_attrib)
87          makeSIZE(file_pointer, image_attrib)
88          makeMETA(file_pointer, image_comment)
89          makeDATA(file_pointer, PIL_image, image_attrib)
```

```
90            makeENDC(file_pointer)
91            file_pointer.close()
92            return const.SUCCESS
93        else:
94            raise error.Invalidimage
95            file_pointer.close()
96            return None
97
98    def makeSTRT(file_pointer):
99        """makeSTRT(file_pointer) -> none
100
101       Escribe el trozo STRT"""
102
103       writeChunk(file_pointer, "STRT", "")
104
105   def makeTYPE(file_pointer, image_attrib):
106       """makeTYPE(file_pointer, image_attrib) -> none
107
108       Escribe el trozo TYPE"""
109
110       writeChunk(file_pointer, "TYPE", image_attrib[0])
111
112   def makeSIZE(file_pointer, image_attrib):
113       """makeSIZE(file_pointer, image_attrib) -> none
114
115       Escribe el trozo SIZE"""
116
117       writeChunk(file_pointer, "SIZE", image_attrib[1] + image_attrib[2])
118
119   def makeMETA(file_pointer, image_comment):
120       """makeMETA(file_pointer, image_comment) -> none
121
122       Escribe el trozo META"""
123
124       writeChunk(file_pointer, "META", image_comment)
125
126   def makeDATA(file_pointer, PIL_image, image_attrib):
127       """makeDATA(file_pointer, image_comment) -> none
128
129       Escribe el trozo DATA"""
130
131       mapb = PIL_image.load()
132       flujo = ""
133
134       col_x = struct.unpack("<I", image_attrib[1])[0]
135       fil_y = struct.unpack("<I", image_attrib[2])[0]
136       bands = len(PIL_image.getbands())
137
138       for y in range(fil_y):
139           for x in range(col_x):
140               for b in range(bands):
141                   if x == 0 and y == 0:
142                       predict = filters.initFilter()
143                       if bands == 1:
144                           real = mapb[x, y]
145                       else:
146                           real = mapb[x, y][b]
147                   elif x != 0 and y == 0:
148                       if bands == 1:
149                           predict = filters.firstRowFilter(mapb[x - 1, y])
150                           real = mapb[x, y]
151                       else:
152                           predict = filters.firstRowFilter(mapb[x - 1, y][b])
153                           real = mapb[x, y][b]
154                   elif x == 0 and y != 0:
155                       if bands == 1:
156                           predict = filters.firstColFilter(mapb[x, y - 1])
157                           real = mapb[x, y]
```

```
158                     else:
159                         predict = filters.firstColFilter(mapb[x, y - 1][b])
160                         real = mapb[x, y][b]
161                 elif x == col_x - 1 and y != 0:
162                     if bands == 1:
163                         predict = filters.garduno3Filter(mapb[x - 1, y - 1],
164                                                          mapb[x, y - 1],
165                                                          mapb[x - 1, y])
166                         real = mapb[x, y]
167                     else:
168                         predict = filters.garduno3Filter(mapb[x - 1, y - 1][b],
169                                                          mapb[x, y - 1][b],
170                                                          mapb[x - 1, y][b])
171                         real = mapb[x, y][b]
172                 else:
173                     if bands == 1:
174                         predict = filters.garduno4Filter(mapb[x - 1, y - 1],
175                                                          mapb[x, y - 1],
176                                                          mapb[x - 1, y],
177                                                          mapb[x + 1, y - 1])
178                         real = mapb[x, y]
179                     else:
180                         predict = filters.garduno4Filter(mapb[x - 1, y - 1][b],
181                                                          mapb[x, y - 1][b],
182                                                          mapb[x - 1, y][b],
183                                                          mapb[x + 1, y - 1][b])
184                         real = mapb[x, y][b]
185 
186                 diff = modalDiff(predict, real)
187                 flujo = flujo + struct.pack("<B", diff)
188 
189     flujo_final = zlib.compress(flujo, 9)
190 
191     writeChunk(file_pointer, "DATA", flujo_final)
192 
193 def makeENDC(file_pointer):
194     """makeENDC(file_pointer) -> none
195 
196     Escribe el trozo ENDC"""
197 
198     writeChunk(file_pointer, "ENDC", "")
199 
200 # ----- DECODER --------
201 
202 def readChunk(file_pointer, chunk_type):
203     """readChunk(file_pointer, chunk_type, chunk_data) -> none
204 
205     Lee un trozo de imagen SPB"""
206 
207     data_size_raw = file_pointer.read(4)
208     if len(data_size_raw) != 4:
209         return None
210 
211     data_size = struct.unpack("<I", data_size_raw)[0]
212     chunk_file_type = file_pointer.read(4)
213     if len(chunk_file_type) != 4 or chunk_file_type != chunk_type:
214         return None
215 
216     if data_size == 0:
217         data_raw = ""
218     else:
219         data_raw = file_pointer.read(data_size)
220     if len(data_raw) != data_size:
221         return None
222 
223     chunk_file_crc = file_pointer.read(4)
224     if len(chunk_file_crc) != 4:
225         return None
```

```
226
227     chunk_crc = struct.pack(
228                             "<I",
229                             zlib.crc32(data_size_raw + chunk_file_type + data_raw)
230                            )
231     if chunk_crc != chunk_file_crc:
232         return None
233
234     return data_raw
235
236 def loadSPB(file_path):
237     """loadSPB(file_path) -> (PIL_image, image_comment)
238
239     Lee una imagen en formato SPB"""
240
241     file_pointer = open(file_path, "r")
242
243     signature = file_pointer.read(8)
244
245     if signature != "SPBIMAGE":
246         file_pointer.close()
247         return None
248
249     if readSTRT(file_pointer) == None:
250         file_pointer.close()
251         return None
252
253     image_type = readTYPE(file_pointer)
254
255     if image_type == None:
256         file_pointer.close()
257         return None
258
259     image_size = readSIZE(file_pointer)
260
261     if image_size == None:
262         file_pointer.close()
263         return None
264
265     image_comment = readMETA(file_pointer)
266
267     if image_comment == None:
268         file_pointer.close()
269         return None
270
271     PIL_image = readDATA(file_pointer, image_type, image_size)
272
273     if PIL_image == None:
274         file_pointer.close()
275         return None
276
277     if readENDC(file_pointer) == None:
278         file_pointer.close()
279         return None
280
281     file_pointer.close()
282     return (PIL_image, image_comment)
283
284 def readSTRT(file_pointer):
285     """readSTRT(file_pointer) -> bandera
286
287     Lee el trozo STRT"""
288
289     check = readChunk(file_pointer, "STRT")
290
291     if check == "":
292         return const.SUCCESS
293     else:
```

```
294             return None
295 
296     def readTYPE(file_pointer):
297         """readTYPE(file_pointer) -> image_type
298 
299         Lee el trozo TYPE"""
300 
301         image_type = readChunk(file_pointer, "TYPE")
302 
303         if image_type == "1":
304             return "L"
305         elif image_type == "2":
306             return "LA"
307         elif image_type == "3":
308             return "RGB"
309         elif image_type == "4":
310             return "RGBA"
311         else:
312             return None
313 
314     def readSIZE(file_pointer):
315         """readSIZE(file_pointer) -> (x_col, y_fil)
316 
317         Lee el trozo SIZE"""
318 
319         size_data_raw = readChunk(file_pointer, "SIZE")
320 
321         if len(size_data_raw) != 8:
322             return None
323         else:
324             x_col = struct.unpack("<I", size_data_raw[0:4])[0]
325             y_fil = struct.unpack("<I", size_data_raw[4:8])[0]
326             return (x_col, y_fil)
327 
328     def readMETA(file_pointer):
329         """readMETA(file_pointer) -> string
330 
331         Lee el trozo META"""
332 
333         image_comment = readChunk(file_pointer, "META")
334 
335         if image_comment == None:
336             return None
337         else:
338             return image_comment
339 
340     def readDATA(file_pointer, image_type, image_size):
341         """readDATA(file_pointer, image_type, image_size) -> PIL_image
342 
343         Lee el trozo DATA y devuelve una imagen PIL"""
344 
345         diff_raw = readChunk(file_pointer, "DATA")
346         if diff_raw == None:
347             return None
348 
349         diff = zlib.decompress(diff_raw)
350 
351         PIL_image = Image.new(image_type, image_size, None)
352         mapb = PIL_image.load()
353 
354         col_x, fil_y = image_size
355         bands = len(PIL_image.getbands())
356 
357         for y in range(fil_y):
358             for x in range(col_x):
359                 if bands > 1:
360                     pixel = []
361                 else:
```

```
362                     pixel = 0
363                 for b in range(bands):
364                     value = struct.unpack(
365                                           "<B",
366                                           diff[(y * col_x * bands) + (x * bands) + b]
367                                          )[0]
368 
369                     if x == 0 and y == 0:
370                         predict = filters.initFilter()
371                     elif x != 0 and y == 0:
372                         if bands == 1:
373                             predict = filters.firstRowFilter(mapb[x - 1, y])
374                         else:
375                             predict = filters.firstRowFilter(mapb[x - 1, y][b])
376                     elif x == 0 and y != 0:
377                         if bands == 1:
378                             predict = filters.firstColFilter(mapb[x, y - 1])
379                         else:
380                             predict = filters.firstColFilter(mapb[x, y - 1][b])
381                     elif x == col_x - 1 and y != 0:
382                         if bands == 1:
383                             predict = filters.garduno3Filter(mapb[x - 1, y - 1],
384                                                              mapb[x, y - 1],
385                                                              mapb[x - 1, y])
386                         else:
387                             predict = filters.garduno3Filter(mapb[x - 1, y - 1][b],
388                                                              mapb[x, y - 1][b],
389                                                              mapb[x - 1, y][b])
390                     else:
391                         if bands == 1:
392                             predict = filters.garduno4Filter(mapb[x - 1, y - 1],
393                                                              mapb[x, y - 1],
394                                                              mapb[x - 1, y],
395                                                              mapb[x + 1, y - 1])
396                         else:
397                             predict = filters.garduno4Filter(mapb[x - 1, y - 1][b],
398                                                              mapb[x, y - 1][b],
399                                                              mapb[x - 1, y][b],
400                                                              mapb[x + 1, y - 1][b])
401 
402                     real = modalSum(predict, value)
403 
404                     if bands > 1:
405                         pixel.append(real)
406                     else:
407                         pixel = real
408 
409                 if bands > 1:
410                     pixel = tuple(pixel)
411                 mapb[x, y] = pixel
412 
413         return PIL_image
414 
415     def readENDC(file_pointer):
416         """readENDC(file_pointer) -> none
417 
418         Lee el trozo ENDC"""
419 
420         check = readChunk(file_pointer, "ENDC")
421 
422         if check == "":
423             return const.SUCCESS
424         else:
425             return None
```

Para la implementación SPBtest, este archivo cambia a la versión siguiente:

```python
#! /usr/bin/env python
# -*- coding: utf-8 -*-

import struct
import zlib
import Image
import SPB.error as error
import SPB.const as const
import SPB.filters as filters

# ----- UTILS ----------

def modalDiff(predict, real):
    """modalDiff(predic, real) -> integer

    Devuelve la diferencia (módulo MAX_PIXEL_VALUE)
    entre el valor real y la predicción.
    """

    if predict <= real:
        return real - predict
    else:
        return (const.MAX_PIXEL_VALUE + 1) - (predict - real)

def modalSum(predict, value):
    """modalSum(predict, value) - > integer

    Devuelve la suma (módulo MAX_PIXEL_VALUE)
    de la predicción con el valor real."""

    real = predict + value

    if real > const.MAX_PIXEL_VALUE:
        real = real - (const.MAX_PIXEL_VALUE + 1)

    return real

# ----- ENCODER --------

def writeChunk(file_pointer, chunk_type, chunk_data):
    """writeChunk(file_pointer, chunk_type, chunk_data) -> none

    Escribe un trozo de imagen SPBtest"""

    chunk_len = len(chunk_data)
    chunk = struct.pack("<I", chunk_len) + chunk_type + chunk_data

    file_pointer.write(chunk + struct.pack("<I", zlib.crc32(chunk) & 0xffffffff))

def getPILAttrib(PIL_image):
    """getPILAttrib(PIL_image) -> [modo, x_col, y_fil]

    Regresa una lista con el modo de imagen y el número de
    filas y columnas de la imagen. Todo en el formato utilizado
    por el archivo SPB"""

    if PIL_image.mode == "L":
        modo = "1"
    elif PIL_image.mode == "LA":
        modo = "2"
    elif PIL_image.mode == "RGB":
        modo = "3"
    elif PIL_image.mode == "RGBA":
        modo = "4"
    else:
        modo = "0"

```

```
68        x_col, y_fil = PIL_image.size
69
70        image_attrib = (modo, struct.pack("<I", x_col), struct.pack("<I", y_fil))
71
72        return image_attrib
73
74    def saveSPB(file_path, PIL_image, image_comment):
75        """saveSPB(file_path, PIL_image, image_comment) -> integer
76
77        Guarda en disco una imagen en formato SPBtest"""
78
79        file_pointer = open(file_path, "wb")
80
81        image_attrib = getPILAttrib(PIL_image)
82
83        if ord(image_attrib[0]) > 48 and ord(image_attrib[0]) < 53:
84            file_pointer.write("PAETHIMG")
85            makeSTRT(file_pointer)
86            makeTYPE(file_pointer, image_attrib)
87            makeSIZE(file_pointer, image_attrib)
88            makeMETA(file_pointer, image_comment)
89            makeDATA(file_pointer, PIL_image, image_attrib)
90            makeENDC(file_pointer)
91            file_pointer.close()
92            return const.SUCCESS
93        else:
94            raise error.Invalidimage
95            file_pointer.close()
96            return None
97
98    def makeSTRT(file_pointer):
99        """makeSTRT(file_pointer) -> none
100
101       Escribe el trozo STRT"""
102
103       writeChunk(file_pointer, "STRT", "")
104
105   def makeTYPE(file_pointer, image_attrib):
106       """makeTYPE(file_pointer, image_attrib) -> none
107
108       Escribe el trozo TYPE"""
109
110       writeChunk(file_pointer, "TYPE", image_attrib[0])
111
112   def makeSIZE(file_pointer, image_attrib):
113       """makeSIZE(file_pointer, image_attrib) -> none
114
115       Escribe el trozo SIZE"""
116
117       writeChunk(file_pointer, "SIZE", image_attrib[1] + image_attrib[2])
118
119   def makeMETA(file_pointer, image_comment):
120       """makeMETA(file_pointer, image_comment) -> none
121
122       Escribe el trozo META"""
123
124       writeChunk(file_pointer, "META", image_comment)
125
126   def makeDATA(file_pointer, PIL_image, image_attrib):
127       """makeDATA(file_pointer, image_comment) -> none
128
129       Escribe el trozo DATA"""
130
131       mapb = PIL_image.load()
132       flujo = ""
133
134       col_x = struct.unpack("<I", image_attrib[1])[0]
135       fil_y = struct.unpack("<I", image_attrib[2])[0]
```

```
136         bands = len(PIL_image.getbands())
137 
138         for y in range(fil_y):
139             for x in range(col_x):
140                 for b in range(bands):
141                     if x == 0 and y == 0:
142                         predict = filters.initFilter()
143                         if bands == 1:
144                             real = mapb[x, y]
145                         else:
146                             real = mapb[x, y][b]
147                     elif x != 0 and y == 0:
148                         if bands == 1:
149                             predict = filters.firstRowFilter(mapb[x - 1, y])
150                             real = mapb[x, y]
151                         else:
152                             predict = filters.firstRowFilter(mapb[x - 1, y][b])
153                             real = mapb[x, y][b]
154                     elif x == 0 and y != 0:
155                         if bands == 1:
156                             predict = filters.firstColFilter(mapb[x, y - 1])
157                             real = mapb[x, y]
158                         else:
159                             predict = filters.firstColFilter(mapb[x, y - 1][b])
160                             real = mapb[x, y][b]
161                     elif x == col_x - 1 and y != 0:
162                         if bands == 1:
163                             predict = filters.paethFilter(mapb[x - 1, y - 1],
164                                                           mapb[x, y - 1],
165                                                           mapb[x - 1, y])
166                             real = mapb[x, y]
167                         else:
168                             predict = filters.paethFilter(mapb[x - 1, y - 1][b],
169                                                           mapb[x, y - 1][b],
170                                                           mapb[x - 1, y][b])
171                             real = mapb[x, y][b]
172                     else:
173                         if bands == 1:
174                             predict = filters.paethFilter(mapb[x - 1, y - 1],
175                                                           mapb[x, y - 1],
176                                                           mapb[x - 1, y])
177                             real = mapb[x, y]
178                         else:
179                             predict = filters.paethFilter(mapb[x - 1, y - 1][b],
180                                                           mapb[x, y - 1][b],
181                                                           mapb[x - 1, y][b])
182                             real = mapb[x, y][b]
183 
184                     diff = modalDiff(predict, real)
185                     flujo = flujo + struct.pack("<B", diff)
186 
187         flujo_final = zlib.compress(flujo, 9)
188 
189         writeChunk(file_pointer, "DATA", flujo_final)
190 
191     def makeENDC(file_pointer):
192         """makeENDC(file_pointer) -> none
193 
194         Escribe el trozo ENDC"""
195 
196         writeChunk(file_pointer, "ENDC", "")
197 
198     # ----- DECODER --------
199 
200     def readChunk(file_pointer, chunk_type):
201         """readChunk(file_pointer, chunk_type, chunk_data) -> none
202 
203         Lee un trozo de imagen SPBtest"""
```

```
204
205         data_size_raw = file_pointer.read(4)
206         if len(data_size_raw) != 4:
207             return None
208
209         data_size = struct.unpack("<I", data_size_raw)[0]
210         chunk_file_type = file_pointer.read(4)
211         if len(chunk_file_type) != 4 or chunk_file_type != chunk_type:
212             return None
213
214         if data_size == 0:
215             data_raw = ""
216         else:
217             data_raw = file_pointer.read(data_size)
218         if len(data_raw) != data_size:
219             return None
220
221         chunk_file_crc = file_pointer.read(4)
222         if len(chunk_file_crc) != 4:
223             return None
224
225         chunk_crc = struct.pack(
226                                 "<I",
227                                 zlib.crc32(data_size_raw + chunk_file_type + data_raw)
228                                )
229         if chunk_crc != chunk_file_crc:
230             return None
231
232         return data_raw
233
234     def loadSPB(file_path):
235         """loadSPB(file_path) -> (PIL_image, image_comment)
236
237         Lee una imagen en formato SPBtest"""
238
239         file_pointer = open(file_path, "r")
240
241         signature = file_pointer.read(8)
242
243         if signature != "PAETHIMG":
244             file_pointer.close()
245             return None
246
247         if readSTRT(file_pointer) == None:
248             file_pointer.close()
249             return None
250
251         image_type = readTYPE(file_pointer)
252
253         if image_type == None:
254             file_pointer.close()
255             return None
256
257         image_size = readSIZE(file_pointer)
258
259         if image_size == None:
260             file_pointer.close()
261             return None
262
263         image_comment = readMETA(file_pointer)
264
265         if image_comment == None:
266             file_pointer.close()
267             return None
268
269         PIL_image = readDATA(file_pointer, image_type, image_size)
270
271         if PIL_image == None:
```

```
272            file_pointer.close()
273            return None
274
275        if readENDC(file_pointer) == None:
276            file_pointer.close()
277            return None
278
279        file_pointer.close()
280        return (PIL_image, image_comment)
281
282    def readSTRT(file_pointer):
283        """readSTRT(file_pointer) -> bandera
284
285        Lee el trozo STRT"""
286
287        check = readChunk(file_pointer, "STRT")
288
289        if check == "":
290            return const.SUCCESS
291        else:
292            return None
293
294    def readTYPE(file_pointer):
295        """readTYPE(file_pointer) -> image_type
296
297        Lee el trozo TYPE"""
298
299        image_type = readChunk(file_pointer, "TYPE")
300
301        if image_type == "1":
302            return "L"
303        elif image_type == "2":
304            return "LA"
305        elif image_type == "3":
306            return "RGB"
307        elif image_type == "4":
308            return "RGBA"
309        else:
310            return None
311
312    def readSIZE(file_pointer):
313        """readSIZE(file_pointer) -> (x_col, y_fil)
314
315        Lee el trozo SIZE"""
316
317        size_data_raw = readChunk(file_pointer, "SIZE")
318
319        if len(size_data_raw) != 8:
320            return None
321        else:
322            x_col = struct.unpack("<I", size_data_raw[0:4])[0]
323            y_fil = struct.unpack("<I", size_data_raw[4:8])[0]
324            return (x_col, y_fil)
325
326    def readMETA(file_pointer):
327        """readMETA(file_pointer) -> string
328
329        Lee el trozo META"""
330
331        image_comment = readChunk(file_pointer, "META")
332
333        if image_comment == None:
334            return None
335        else:
336            return image_comment
337
338    def readDATA(file_pointer, image_type, image_size):
339        """readDATA(file_pointer, image_type, image_size) -> PIL_image
```

```
340
341     Lee el trozo DATA y devuelve una imagen PIL"""
342
343     diff_raw = readChunk(file_pointer, "DATA")
344     if diff_raw == None:
345         return None
346
347     diff = zlib.decompress(diff_raw)
348
349     PIL_image = Image.new(image_type, image_size, None)
350     mapb = PIL_image.load()
351
352     col_x, fil_y = image_size
353     bands = len(PIL_image.getbands())
354
355     for y in range(fil_y):
356         for x in range(col_x):
357             if bands > 1:
358                 pixel = []
359             else:
360                 pixel = 0
361             for b in range(bands):
362                 value = struct.unpack(
363                                       "<B",
364                                       diff[(y * col_x * bands) + (x * bands) + b]
365                                      )[0]
366
367                 if x == 0 and y == 0:
368                     predict = filters.initFilter()
369                 elif x != 0 and y == 0:
370                     if bands == 1:
371                         predict = filters.firstRowFilter(mapb[x - 1, y])
372                     else:
373                         predict = filters.firstRowFilter(mapb[x - 1, y][b])
374                 elif x == 0 and y != 0:
375                     if bands == 1:
376                         predict = filters.firstColFilter(mapb[x, y - 1])
377                     else:
378                         predict = filters.firstColFilter(mapb[x, y - 1][b])
379                 elif x == col_x - 1 and y != 0:
380                     if bands == 1:
381                         predict = filters.paethFilter(mapb[x - 1, y - 1],
382                                                       mapb[x, y - 1],
383                                                       mapb[x - 1, y])
384                     else:
385                         predict = filters.paethFilter(mapb[x - 1, y - 1][b],
386                                                       mapb[x, y - 1][b],
387                                                       mapb[x - 1, y][b])
388                 else:
389                     if bands == 1:
390                         predict = filters.paethFilter(mapb[x - 1, y - 1],
391                                                       mapb[x, y - 1],
392                                                       mapb[x - 1, y])
393                     else:
394                         predict = filters.paethFilter(mapb[x - 1, y - 1][b],
395                                                       mapb[x, y - 1][b],
396                                                       mapb[x - 1, y][b])
397
398                 real = modalSum(predict, value)
399
400                 if bands > 1:
401                     pixel.append(real)
402                 else:
403                     pixel = real
404
405             if bands > 1:
406                 pixel = tuple(pixel)
407             mapb[x, y] = pixel
```

```
408
409     return PIL_image
410
411 def readENDC(file_pointer):
412     """readENDC(file_pointer) -> none
413
414     Lee el trozo ENDC"""
415
416     check = readChunk(file_pointer, "ENDC")
417
418     if check == "":
419         return const.SUCCESS
420     else:
421         return None
```

const.py

```
1 #! /usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 """Constantes:"""
5
6 MAX_PIXEL_VALUE = 255
7 PLANE_TOLERANCE = 1
8 SUCCESS = 1
```

error.py

```
1 #! /usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 """Clases de errores"""
5
6 class Invalidinput(Exception):
7     """Error de entrada propio.
8
9     Es lanzado cuando el valor introducido no es válido según el
10     contexto del programa."""
11     def __str__(self):
12         return repr(self.value)
13
14 class Invalidimage(Exception):
15     """Error de entrada propio.
16
17     Es lanzado cuando una imagen no es válida."""
18     def __str__(self):
19         return repr(self.value)
```

filters.py

```
1 #! /usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 import math
```

```
5   import SPB.const as const
6   import SPB.utils as utils
7
8   MAX_PIXEL_VALUE = 255
9
10  def initFilter():
11      """initFilter() => Predicción inicial para el primer pixel.
12
13      Devuelve un valor predeterminado para el valor predicho del
14      primer pixel de la imagen"""
15
16      return int(math.ceil(const.MAX_PIXEL_VALUE / 2.0))
17
18  def firstRowFilter(pix_o):
19      """firstRowFilter(pix_o) => Predicción para pixeles de la primera fila,
20      excepto el primero."""
21
22      return pix_o
23
24  def firstColFilter(pix_n):
25      """firstColFilter(pix_n) => Predicción para la primer
26      columna de pixeles, excepto el primero."""
27
28      return pix_n
29
30  def garduno3Filter(pix_gamma, pix_n, pix_o):
31      """garduno3Filter(pix_gamma, pix_n, pix_o) => Predictor para última
32      columna de la imagen."""
33
34      plane_type = utils.planeType(pix_gamma, pix_n, pix_o, const.PLANE_TOLERANCE)
35
36      c_s = pix_n + pix_o - pix_gamma
37
38      if plane_type == "L": # Predictor por plano simple
39              if c_s > const.MAX_PIXEL_VALUE:
40                  return const.MAX_PIXEL_VALUE
41              elif c_s < 0:
42                  return 0
43              else:
44                  return c_s
45      else: # Predictor por plano simple inclinación 1 a 1/2
46          p_point = (pix_n + pix_o) / 2.0
47          extrapolation = (p_point + c_s) / 2.0
48
49          if extrapolation > const.MAX_PIXEL_VALUE:
50              return const.MAX_PIXEL_VALUE
51          elif extrapolation < 0:
52              return 0
53
54          if extrapolation % 1 != 0:
55              if c_s < extrapolation:
56                  return int(math.ceil(extrapolation))
57              else:
58                  return int(math.floor(extrapolation))
59          else:
60              return int(extrapolation)
61
62  def garduno4Filter(pix_gamma, pix_n, pix_o, pix_delta):
63      """garduno4Filter(pix_gamma, pix_n, pix_o, pix_delta) => Predictor
64      general."""
65
66      plane_type = utils.planeType(pix_gamma, pix_n, pix_o, const.PLANE_TOLERANCE)
67
68      c_s = pix_n + pix_o - pix_gamma
69
70      if plane_type == "L": # Predictor por plano simple
71              if c_s > const.MAX_PIXEL_VALUE:
72                  return const.MAX_PIXEL_VALUE
```

```
73              elif c_s < 0:
74                  return 0
75              else:
76                  return c_s
77      elif plane_type == "X": # Predictor por plano simple inclinación 1 a 1/2
78          p_point = (pix_n + pix_o) / 2.0
79          extrapolation = (p_point + c_s) / 2.0
80
81          if extrapolation > const.MAX_PIXEL_VALUE:
82              return const.MAX_PIXEL_VALUE
83          elif extrapolation < 0:
84              return 0
85
86          if extrapolation % 1 != 0:
87              if c_s < extrapolation:
88                  return int(math.ceil(extrapolation))
89              else:
90                  return int(math.floor(extrapolation))
91          else:
92              return int(extrapolation)
93      elif plane_type == "U": # PS a 1 a 1/2 y PSec a 1 a 1/2
94          c_sec = pix_delta + pix_o - pix_n
95          p_point = (pix_n + pix_o) / 2.0
96          inclinacion_izq = p_point - pix_o
97          extrapolation_s = (p_point + c_s) / 2.0
98          extrapolation_sec = c_sec + (inclinacion_izq / 2.0)
99          extrapolation = (extrapolation_s + extrapolation_sec) / 2.0
100
101         if extrapolation > const.MAX_PIXEL_VALUE:
102             return const.MAX_PIXEL_VALUE
103         elif extrapolation < 0:
104             return 0
105
106         if extrapolation % 1 != 0:
107             if c_s > extrapolation:
108                 return int(math.ceil(extrapolation))
109             else:
110                 return int(math.floor(extrapolation))
111         else:
112             return int(extrapolation)
```

Para la implementación SPBtest este archivo cambia a la versión siguiente:

```
1   #! /usr/bin/env python
2   # -*- coding: utf-8 -*-
3
4   import math
5   import SPB.const as const
6   import SPB.utils as utils
7
8   MAX_PIXEL_VALUE = 255
9
10  def initFilter():
11      """initFilter() => Predicción inicial para el primer pixel.
12
13      Devuelve un valor predeterminado para el valor predicho del
14      primer pixel de la imagen"""
15
16      return int(math.ceil(const.MAX_PIXEL_VALUE / 2.0))
17
18  def firstRowFilter(pix_o):
19      """firstRowFilter(pix_o) => Predicción para pixeles de la primera fila,
20      excepto el primero."""
21
22      return pix_o
23
```

```
24  def firstColFilter(pix_n):
25      """firstColFilter(pix_n) => Predicción para la primer
26      columna de pixeles, excepto el primero."""
27  
28      return pix_n
29  
30  def paethFilter(pix_gamma, pix_n, pix_o):
31      """paethFilter(pix_gamma, pix_n, pix_o) => Predictor por
32      método Paeth.
33  
34      Obtiene la extrapolación por medio del método Paeth de un
35      solo pixel."""
36  
37      est = pix_n + pix_o - pix_gamma # Estimación inicial
38  
39      # Distancias a: o, n, gamma
40      est_o = abs(est - pix_o)
41      est_n = abs(est - pix_n)
42      est_gamma = abs(est - pix_gamma)
43  
44      # Regresar el más cercano de: o, n, gamma en orden: o, n, gamma
45      if est_o <= est_n and est_o <= est_gamma:
46          return pix_o
47      elif est <= est_gamma:
48          return pix_n
49      else:
50          return pix_gamma
```

__init__.py

Es un archivo vacío.

utils.py

```
1   #! /usr/bin/env python
2   # -*- coding: utf-8 -*-
3   
4   def inclination(pix_gamma, pix_n, pix_o):
5       """inclination(pix_gamma, pix_n, pix_o) => Inclinación del plano simple.
6   
7       Devuelve una tupla (inc_izq, inc_der) con la inclinación del plano
8       simple.
9   
10      La inclinación inc_izq es la diferencia entre el centro geométrico del
11      plano simple (p) y el valor del pixel oeste (o).
12  
13      La inclinación inc_der es al diferencia entre el centro geométrico del
14      plano simple (p) y el valor del pixel noroeste (gamma).
15  
16      Los sentidos de inclinación son ortogonales."""
17  
18      p_point = (pix_n + pix_o) / 2.0
19  
20      return (p_point - pix_o, p_point - pix_gamma)
21  
22  def planeType(pix_gamma, pix_n, pix_o, tol):
23      """planeType(pix_gamma, pix_n, pix_o, tol) => Tipo de plano simple
24  
25      Devuelve el tipo de plano simple que gobierna al pixel:
26      X (Tipo X) - La inclinación izquerda (inc_izq)
27      y la inclinación derecha (inc_der) son casi la misma
28  
```

```
29        L (Lateral) - La inclinación izquerda (inc_izq) es superior
30        a la inclinación derecha (inc_der)
31
32        U (Arriba-abajo) - La inclinación derecha (inc_der) es superior
33        a la inclinación izquierda (inc_izq)
34
35        tol: (entero positivo o cero) toleracia para clasificar el plano
36        como tipo X """
37
38        inc = inclination(pix_gamma, pix_n, pix_o)
39
40        if abs(inc[0]) - abs(inc[1]) < -tol: # Cuadro Superior/inferior
41            return "U"
42        elif abs(inc[0]) - abs(inc[1]) > tol: # Cuadro derecho/izquierdo
43            return "L"
44        else: # En caso contrario pertenece a la "X" de inclinaciones similares
45            return "X"
```

Bibliografía

[1] Varios autores autores: *PNG (Portable Network Graphics) Specification, Versión 1.2*, 1999. http://www.libpng.org/pub/png/spec/1.2/, visitado: 2008-04-29.

[2] Sara Camacho Cancino: *Análisis de Algoritmos*. Ediciones Acatlán, 1998.

[3] Peter Deutsch: *DEFLATE Compressed Data Format Specification version 1.3*, 1996. http://www.gzip.org/zlib/rfc-deflate.html.

[4] Antaeus Feldspar: *An Explanation of the Deflate Algorithm*, Agosto 1997. http://www.zlib.net/feldspar.html.

[5] R. C. Gonzalez y R. Wood: *Tratamiento digital de imágenes*. Ediciones Diaz De Santos S.A., 1996.

[6] D.A. Huffman: *A method for the construction of minimum-redundancy codes*. Resonance, 11(2):91–99, 2006.

[7] Wolfgang Köhler: *Psicología de la configuración*. Ediciones Morata, 1967.

[8] Margaret S. Livingstone: *Vision and Art: The Biology of Seeing*. Harry N Abrams, Mayo 2002.

[9] Conrad G. Mueller, Mae Rudolph *y cols.*: *Luz y visón*. Colección científica de Time Life. Time Life, Lito Offset Latina S.A., 1974.

[10] A. W. Paeth: *Image File Compression Made Easy*. En James Arvo (editor): *Graphics Gems II*. Academic Press, 1991.

[11] C. E. Shannon: *A Mathematical Theory of Comunication*. The Bell System Technical Journal, 27:379 – 423, 623 – 653, Julio - Octubre 1948.

[12] Compression Team: *The LZ77 algorithm*, 1997. http://oldwww.rasip.fer.hr/, consultar 'Research' ->'Compress', visitado: 2009-06-22.

[13] International Telecommunications Union: *Basic Parameter Values for the HDTV Standard for the Studio and for International Programme Exchange*, 1990.

[14] International Telecommunications Union: *Error-correcting Procedures for DCEs Using Asynchronous-to-Synchronous Conversion*, 1994.

[15] Stephen Westland: *Frequently asked questions about Colour Physics*, 2000. http://www.colourware.co.uk/cpfaq.htm, visitado: 2008-12-23.

[16] Diethelm Wuertz *y cols.*: *Rmetrics - Markets and Basic Statistics*, Abril 2009. http://cran.r-project.org/, visitado: 2009-07-31.

Índice de figuras

1.1. Espectro electromagnético 5
1.2. Sensibilidad relativa de bastones y conos humanos 6
1.3. Cantidad de colores primarios ideales para el observador CIE de 2 grados 7
1.4. Modelo RGB y CMYK 8
1.5. Ejemplo de árbol Huffman 18

2.1. Diagrama de pixeles vecinos 26
2.2. Secuencia de almacenamiento de los pixeles de un mapa de bits 27
2.3. Pixeles vecinos utilizados en la predicción 27
2.4. Pixeles arbitrarios 28
2.5. Posición relativa de los pixeles vecinos 30
2.6. Ruido uniformemente distribuido 31
2.7. Ejemplo de imagenes utilizadas 32
2.8. Distribución de errores por plano simple aplicado al ruido . . . 34
2.9. Distribución de errores por plano simple aplicado a imagen con sentido 34
2.10. Inclinaciones izquierda y derecha del plano simple 37
2.11. Inclinación izquierda vs. inclinación derecha 38
2.12. Inclinación izquierda vs. inclinación derecha (densidad de pixeles) 39
2.13. Error vs. inclinación derecha en predicción al ruido 40
2.14. Error vs. inclinación derecha en predicción a una imagen con sentido 41
2.15. Posición relativa de los pixeles vecinos en el plano simple secundario 44
2.16. Error vs. inclinación izquierda en una imagen con sentido aplicando predicción por plano simple secundario 45
2.17. Inclinación izquierda vs. inclinación derecha para predicción por plano simple secundario 46

3.1. Diagrama de muestras adjacentes 52
3.2. Estructura de un trozo 53
3.3. Volcado hexadecimal de imagen SPB 60

Índice de tablas

1.1. Frecuencias de símbolos para el ejemplo de codificación Huffman 17
1.2. Cadena de entrada para el ejemplo de codificación LZ77 19
1.3. Proceso de codificación LZ77 20

2.1. Resumen para sesgo del análisis de ruido 33
2.2. Resumen para sesgo del análisis de imágenes con sentido 35
2.3. Diferencia en la dispersión del error por tipo de imagen 35
2.4. Resumen de modas, medias y medianas para ruido e imágenes con sentido 35
2.5. Resumen de la entropía del error por tipo de imagen 36
2.6. Correlación inclinación derecha vs. error de predicción por plano simple 40
2.7. Entropía promedio por variante de método de predicción 42
2.8. Desviación estándar de la entropía por variante de método de predicción 43
2.9. Correlaciones y sus promedios de error vs. diferencia con δ . . . 43
2.10. Puntos equivalentes entre plano simple y plano simple secundario 44
2.11. Promedios para predictores por plano simple secundario 47
2.12. Entropias promedio por predictor y tipo de pixel 48
2.13. Desviaciones estándar promedio por predictor y tipo de pixel . 48

3.1. Campos de un trozo 54
3.2. Valor por tipo de imagen SPB 55
3.3. Tamaños en bytes de imágenes por tipo de compresión 61