

Información Importante

La Universidad Santo Tomás, informa que el(los) autor(es) ha(n) autorizado a usuarios internos y externos de la institución a consultar el contenido de este documento a través del Catálogo en línea del CRAI-Biblioteca y el Repositorio Institucional en la página Web de la CRAI-Biblioteca, así como en las redes de información del país y del exterior con las cuales tenga convenio la Universidad.

Se permite la consulta a los usuarios interesados en el contenido de este documento, para todos los usos que tengan **finalidad académica**, nunca para usos comerciales, siempre y cuando mediante la correspondiente cita bibliográfica se le dé crédito al trabajo de grado y a su autor.

De conformidad con lo establecido en el Artículo 30 de la Ley 23 de 1982 y el artículo 11 de la Decisión Andina 351 de 1993, la Universidad Santo Tomás informa que “los derechos morales sobre documento son propiedad de los autores, los cuales son irrenunciables, imprescriptibles, inembargables e inalienables.”

**Centro de Recursos para el Aprendizaje y la Investigación, CRAI-Biblioteca
Universidad Santo Tomás, Bucaramanga**

Implementación de un algoritmo para el proceso de estimación de movimiento de un codificador basado en el estándar H.264 sobre un dispositivo hardware programable

Rafael Álvarez Bosa

**Trabajo de grado para optar el título de Magíster en Redes y Sistemas
de Comunicaciones**

Director
Sergio Andrés Zabala

Codirector
Holguer Andrés Becerra

Universidad Santo Tomás, Bucaramanga
División de Ingenierías y Arquitectura
Facultad de Ingeniería de Telecomunicaciones

2017

Dedicatoria

Este trabajo de grado es un logro más en mi desarrollo profesional como Ingeniero de Telecomunicaciones y es indispensable mencionar a las personas que hicieron que este objetivo se hiciera hoy realidad.

Dedico este trabajo de grado a:

Jehová el responsable de guiarme en todos los retos que afronto.

Mis padres Marta Bosa Castañeda y Rafael Alvarez Delgado coautores de este proceso y que pese a las dificultades que vivimos, fueron pilar fundamental del presente logro profesional. A mi abuela Sara Castañeda por compartir toda su experiencia de la vida conmigo, por ayudarme a crecer en todo aspecto. A mi hermano Jhon Alvarez por el apoyo para lograr mis objetivos. Y por último a mi pareja Karina Gutiérrez por ser un apoyo incondicional en mi vida.

Agradecimientos

Agradezco al Ingeniero Sergio Andrés Zabala por haber aceptado ser mi director de trabajo de grado para esta etapa, ofreciéndome su guía, experiencia, conocimientos, compromiso y todas las herramientas necesarias para llevar a cabo con éxito mi proyecto final de Maestría. Muchas gracias por sus excelentes recomendaciones y aportes.

Agradezco al Ingeniero Holguer Becerra por haberme ofrecido todos sus conocimientos, su interés por mi trabajo y su amistad.

Agradezco al Ingeniero Tito Raúl Vargas por el interés en nuestra formación como Magíster sin duda contribuyo inmensamente a mi crecimiento profesional.

Agradezco al cuerpo de docentes por entregarnos todo el conocimiento para nuestro crecimiento profesional.

A la Fundación Cardiovascular por ofrecer sus recursos tecnológicos para el desarrollo del presente proyecto.

Por ultimo agradezco a la Maestría en Redes y Sistemas de Comunicaciones, la facultad de Ingeniería de Telecomunicaciones de la Universidad Santo Tomas por su responsabilidad y la calidad de sus programas académicos.

Contenido

Introducción.....	13
1 Implementación de un algoritmo para el proceso de estimación de movimiento de un codificador basado en el estándar H.264 sobre un dispositivo hardware programable.....	15
1.1 Objetivos.....	15
1.1.1 Objetivo General.	15
1.1.2 Objetivo Específicos.	15
2 Marco Referencial.....	16
2.1 Reseña histórica de estándares de codificación de video.....	17
2.2 Estándar de codificación H.264.....	19
2.3 Redundancia de video.....	24
2.3.1 Redundancia estadística.....	24
2.3.2 Redundancia Psicovisual.....	26
2.4 Algoritmos de Block Matching.....	26
2.5 Criterios de búsqueda.....	28
2.6 Tamaño del Macrobloque.....	29
2.7 Rango de búsqueda.....	30
2.8 Algoritmos de estimación de movimiento basados en Block Matching.....	31
2.8.1 Búsqueda Exhaustiva (ES).	32
2.8.2 Algoritmo en tres pasos (TSS).....	33
2.8.3 Nueva búsqueda de tres pasos (NTSS).....	34
2.8.4 Búsqueda simple y eficiente (SES).....	35
2.8.5 Búsqueda en cuatro pasos (4SS).....	37
2.8.6 Método búsqueda en diamante.....	38
2.9 Elección del Algoritmo.....	38
3 Método.....	39
3.1 Requerimientos.....	39
3.2 Tarjeta de desarrollo DE0-Nano.....	39
3.3 Desarrollo.....	41
3.4 Algoritmo.....	44
4 Resultados.....	48
4.1 Simulación.....	48
4.2 Versiones del Algoritmo.....	56

5	Conclusiones	60
	Apendices	61
	Referencias Bibliograficas.....	62

Lista de Tablas

Tabla 1. resultados Simulación.....	55
Tabla 2. Resultados versión 1.2.....	58

Lista de figuras

	pág.
Figura 1 Codificador de video H.264. Adaptado de “Video compression from concepts to the H.264/AVC standard” por Gary J Sullivan, 2005.....	20
Figura 2 Algoritmos de estimación de movimiento. Adaptado de "Estudio de Arquitecturas VLSI de la etapa de predicción de la compensación de movimiento, para compresión de imágenes y videos." Por Armando Mora Campos, 2008.	22
Figura 3 “Clases de redundancia en el video” Adaptado de "Estudio de Arquitecturas VLSI de la etapa de predicción de la compensación de movimiento, para compresión de imágenes y videos." Por Armando Mora Campos, 2008.	23
Figura 4.Redundancia espacial	25
Figura 5.Redundancia Temporal.....	25
Figura 6.Concepto de Block Matching	27
Figura 7. Macrobloques	30
Figura 8. Rango de búsqueda y vector de movimiento. Adaptado de "Estudio de Arquitecturas VLSI de la etapa de predicción de la compensación de movimiento, para compresión de imágenes y videos." Por Armando Mora Campos, 2008.	31
Figura 9. Algoritmo en tres pasos, Adaptado de “Estimación de Movimiento” por I. L. Espejo, 2012.....	33
Figura 10. Nueva búsqueda en tres pasos (NTSS), Adaptado de “Estimación de Movimiento” por I. L. Espejo, 2012.	34
Figura 11. Patrones de búsqueda SES, Adaptado de “Estimación de Movimiento” por I. L. Espejo, 2012.....	35
Figura 12. Búsqueda simple y eficiente (SES), Adaptado de “Estimación de Movimiento” por I. L. Espejo, 2012.	36
Figura 13. Búsqueda en 4 pasos (4SS), Adaptado de “Estimación de Movimiento” por I. L. Espejo, 2012.....	37

Figura 14. Búsqueda en Diamante, Adaptado de “Estimación de Movimiento” por I. L. Espejo, 2012.....	38
Figura 15. DE0-nano FPGA Altera, Adaptado de”User Manual for DE0 Nano Board” por Altera, 2016.....	40
Figura 16. Diagrama de bloques DE0 nano, Adaptado de”User Manual for DE0 Nano Board” por Altera, 2016.....	41
Figura 17. Imagen residuo	42
Figura 18. Código resta pixeles entre frame referencia y frame actual	43
Figura 19. Macrobloques	44
Figura 20. Máquina de estados Algoritmo Estimación de Movimiento	45
Figura 21. Estado Idle	45
Figura 22. Estado READ_MEM.....	46
Figura 23. Estado BUSCAR SIMILAR.....	47
Figura 24. Imagen de Referencia	48
Figura 25. Imagen Actual	49
Figura 26. Imagen con cálculo del SAD, (a) Frame Referencia, (b) Frame Actual	50
Figura 27. Archivos verilog para la simulación.....	51
Figura 28. Simulación Algoritmo estimación de movimiento en Simulink	51
Figura 29. Resultado vector de movimiento encontrados.....	52
Figura 30. Ubicación Macrobloque Referencia	52
Figura 31. Valor del Macrobloque de referencia de ejemplo	53
Figura 32. Ubicación Macrobloque Actual.....	53
Figura 33. Valor del Macrobloque actual de ejemplo.....	53
Figura 34. Grafica Vector de Movimiento Matlab	54
Figura 35. Resultado Versión 1.0 del Algoritmo.....	56
Figura 36. Ventanas para Algoritmo Versión 1.2	57
Figura 37. Simulación Algoritmo Versión 1.2.....	58

Lista de apéndices**pág.**

Apéndices A. Código estimación de movimiento Versión 1.1.....	61
Apéndices B. Control del Algoritmo.....	61
Apéndices C. Fichero vectores de movimiento.....	61

Glosario

Estimación de movimiento. Uno de los procesos de codificación de video empleado en varios estándares.

Block Matching. Técnica empleada para la estimación de movimiento.

H.264. Estándar de codificación de video.

FPGA. Dispositivo de hardware programable.

SAD. Suma de errores absolutos, hace referencia a criterios empleados en la búsqueda de macrobloques.

Macrobloque. Agrupación de píxeles en un bloque con dimensiones estándar.

VGA. Formato de video.

Resumen

Este trabajo presenta el diseño de un algoritmo de estimación de movimiento que puede ser implementado en una plataforma hardware programable FPGA, con el fin de apoyar el desarrollo de conversores de video universales que puedan ser fácilmente implementados en aplicaciones de telemedicina aportando así en la alianza USTA-FCV. Para realizar este trabajo se analizaron las necesidades del proyecto, se identificaron las oportunidades de contribuir con la estimación de movimiento para plataformas hardware programable. Se profundizo en el estado del arte y el marco referencia de las técnicas de estimación de movimiento, generando a partir de este conocimiento adquirido prototipos que fueron evolucionando hasta lograr un algoritmo que puede determinar vectores de movimiento.

Palabras Clave: Igualación de bloques, H.264, Macrobloque, Codificación de Video, Estimación de movimiento, SAD

Abstract

This work presents the design of a motion estimation algorithm that can be implemented in a programmable FPGA hardware platform, in order to support the development of universal video converters that can be easily implemented in telemedicine applications that support USTA-FCV alliance. In order to carry out this work, we analyzed needs of the project, the opportunities to contribute with the estimation of movement for programmable hardware platforms were identified as well. We studied the state of the art and the frame of reference of motion estimation techniques, generating from this acquired knowledge prototypes that were evolving until reaching an algorithm that can determine vectors of movement.

Key Word- Block Matching, H.264, Macroblock, Video Coding, Motion Estimation, SAD

Introducción

Uno de los aspectos fundamentales en la creación de nuevos servicios multimedia es la apropiación de los estándares de codificación, estos permiten reducir considerablemente la cantidad de información además de mantener una calidad de video sobresaliente sin olvidar interoperabilidad entre usuarios de los servicios y desarrolladores de tecnología. Poder generar buenas tasas de comprensión sin sacrificar calidad de video es el objetivo de los estándares de codificación como H.264. Estos estándares son la aplicación de diversas técnicas que se enfocan en reducir la redundancia en la información que existe en los videos.

En ese sentido, existe el interés del grupo de investigación UNITEL de la Facultad de Ingeniería de Telecomunicaciones de la Universidad Santo Tomás, en explorar nuevas y más eficientes estrategias para la transmisión de video; asociado esto a la necesidad de aplicaciones en diferentes campos, como es el caso de la telemedicina. En esta temática en particular, y en alianza con el grupo de investigación de Bioingeniería de la Fundación Cardiovascular de Colombia, se ha determinado la importancia de contar con estrategias eficientes para la transmisión de datos de telemedicina, que faciliten el diagnóstico, monitoreo y tratamiento de enfermedades en sectores alejados; requiriendo la transmisión de señales de video (VGA y/o HDMI) a puntos remotos de análisis.

Para este desarrollo en particular se cuenta con hardware de propósito específico la cual cuenta con capacidades de ejecución de tareas secuenciales (a través de un procesador) y de funciones combinatorias (a través de una FPGA). El reto entonces se centra en contar con algoritmos que permita utilizar esta herramienta para comprimir y facilitar la transmisión del video, detectándose en la revisión con las fuentes primarias, la importancia de contar con un estimador de movimiento eficiente.

Por ello es importante preguntarse si ¿Es posible diseñar e implementar un algoritmo de estimación de movimiento en una plataforma FPGA que se pueda adaptar al estándar H.264, optimizando la compresión de video? En base a la pregunta La estimación de movimiento es un proceso clave en la codificación de video, esta se encarga de reducir de forma significativa la redundancia temporal que se encuentra entre los diferentes frames. En ese sentido se espera que el desarrollo de esta propuesta aporte a la comunidad científica regional y nacional con una alternativa viable en algoritmos que respondan a la necesidad descrita y contribuyan a la transmisión adecuada de señales, en este caso video, para análisis remoto.

1. Implementación de un algoritmo para el proceso de estimación de movimiento de un codificador basado en el estándar H.264 sobre un dispositivo hardware programable

1.1 Objetivos

1.1.1 Objetivo General.

Implementar un algoritmo de estimación de movimiento, sobre una plataforma FPGA que permita cumplir con la etapa de estimación de movimiento de un codificador de video basado en el estándar H.264.

1.1.2 Objetivo Específicos.

Realizar el estado de arte sobre algoritmos Block Matching en la etapa de estimación de movimiento del estándar H.264, para seleccionar el algoritmo con mejores prestaciones sobre una FPGA de la familia Altera Cyclone V.

Diseñar el algoritmo de estimación de movimiento basado en la técnica de estimación de movimiento seleccionada.

Implementar el algoritmo de estimación de movimiento para ser usado en una plataforma hardware programable FPGA de la familia Altera Cyclone V.

Discutir los resultados obtenidos por el algoritmo propuesto en base a procesamiento por macrobloque comparándolo con referentes principales.

2. Marco Referencial

En la actualidad se pueden encontrar gran variedad de técnicas que permiten la compresión de video, las cuales, pueden ser ejecutadas individualmente o aplicadas en conjunto estratégicamente con el objetivo de remover la información redundante, permitiendo así, que la información pueda ser recreada solamente por los datos que en realidad son útiles[1]. Respecto a las técnicas de compresión de video se pueden encontrar dos grupos generales, el primero; son las técnicas sin pérdidas que se enfocan específicamente en suprimir la información redundante, teniendo como ventaja principal que los datos se pueden reconstruir. Algunos ejemplos de técnicas de compresión sin pérdidas son los Algoritmos de Huffman, Algoritmos de Lempel-Ziv o la compresión aritmética. Respecto al otro grupo general se encuentran las técnicas con pérdidas que proporcionan una buena tasa de compresión, ya que no sólo se centran en suprimir la información redundante, sino que además, al ser reconstruida esta información, no se debería afectar la percepción de la calidad[2].

El aumento constante de tecnologías audiovisuales y la alta demanda de servicios multimedia en la red de internet, han provocado el desarrollo de diferentes métodos y técnicas para el tratamiento de video con el fin de lograr altos niveles de compresión de video que no perjudique su calidad y, por ende, la QoE (Calidad de Experiencia) del usuario final. La compresión de video es el resultado de la aplicación de diferentes técnicas con el objetivo de la pérdida intencional de datos redundantes o irrelevantes del video, sin que afecte posteriormente su reconstrucción con la respectiva información relevante[3]. Estos datos reconstruidos pueden ser almacenados en dispositivos de memoria (CD, DVD, BlueRay, USB, entre otras) o transmitidos a través de una

red empleando diferentes técnicas de distribución de contenido multimedia (descarga progresiva, descarga HTTP, Streaming de video, entre otras)[4].

2.1 Reseña histórica de estándares de codificación de video

La codificación de video hace parte de la cotidianidad; un video de YouTube, una reproducción de video desde cualquier dispositivo, telemedicina, la televisión. Innumerables son las aplicaciones que permiten los estándares de codificación, estándares, que deben ser actualizados constantemente para no ser rebasados por el crecimiento de tecnologías audiovisuales o tecnologías que contribuyan a la distribución de contenido multimedia mediante servicios[4].

A través de la historia, se evidencia que instituciones como la Organización Internacional para la Estandarización (ISO), la Comisión Electro Técnica Internacional (IEC) y la Unión Internacional de Telecomunicaciones (ITU-T), han trabajado en común para realizar estándares que puedan ser implementados por la industria y así, generar interoperabilidad dentro de servicios y proveedores tecnológicos:

JPEG (ISO/IEC 10918-1). Este estándar se publicó en 1992 y el objetivo principal fue la compresión de imágenes estáticas. Esta herramienta emplea como novedad la DCT-1D y un sistema de codificación de entropía basado en Huffman [5].

JPEG-2000 (ISO/IEC 15444-1:2000). Fue publicado en el año 2000. El nuevo concepto de este estándar especifica el proceso para descomprimir y reconstruir una imagen estática empleando la transformada de Wavelet [6]

MPEG-1(ISO/IEC 11172-2). Publicado en 1991. Este estándar fue diseñado para el almacenamiento de imágenes con movimiento y audio en medios digitales, la DCT y estimación de movimiento fueron sus principales características [7].

MPEG-2(ISO/IEC 13818-2). Este estándar fue publicado en 1994 y sus principales características fueron las señales de video progresivas y entrelazadas, nuevos esquemas de muestreo y mayores tasas de bits. Sus dos principales aplicaciones fueron la televisión digital y el DVD [1].

MPEG-4(ISO/IEC 14496). Este estándar que fue publicado en 1999 posee características que aún se siguen desarrollando de acuerdo a sus aplicaciones, todas estas, enfocadas a aplicaciones multimedia desde videos en dispositivos de almacenamiento como CD, DVD, televisión digital e Internet. Actualmente el desarrollo de este estándar más desplegado es MPEG-4 AVC (Advance Video Coding) ya que es el que permite una eficiente codificación para aplicaciones en la Internet[8].

H.120. Es un estándar de codificación de video digital publicado en 1988, su principal característica fue emplear técnicas de cuantización, codificador de entropía, compensación de movimiento y un similar a predicción intra-frames [9].

H.261.Estandar especialmente diseñado para aplicaciones en la transmisión en redes de servicios digitales (ISDN), entre sus características se encuentran la predicción temporal y espacial. Su principal aplicación fue la videoconferencia [10].

H.263. Este estándar fue diseñado para emplearse en redes de par trenzado con la particularidad de permitir un método para la pérdida de datos; entre las características más relevantes se encuentra la estimación de movimiento de bloque variable, permitiendo así realizar la estimación de movimiento sobre diferentes tipos de bloques [11].

2.2 Estándar de codificación H.264

MPEG-4 parte 10 o también conocido como H.264, es un estándar que establece y define un códec de vídeo de alta compresión. Este estándar fue desarrollado por ITU-T Vídeo Coding Experts Groups (VCEG) y ISO/IEC Moving Picture Experts Groups (MPEG).

El desarrollo de H.264 se enfocó en una norma que se adaptara a las características de la transmisión de servicios y aplicaciones de contenido multimedia sobre Internet, teniendo en cuenta técnicas ya implementadas en otros estándares o versiones anteriores como emplear la DCT + compensación de movimiento, implementar codificaciones no empleadas anteriormente por su elevado costo y no establecer compatibilidad con versiones MPEG anteriores ya que generaría restricciones en su diseño [12].

Es importante resaltar que el estándar especifica cómo debe ser el flujo de video codificado y el decodificador de ese flujo de video, por lo tanto; el codificador es un poco más flexible, ya que no está estandarizado y por ende se pueden hacer constantes mejoras. En general el proceso de codificación de video se basa en una serie de etapas como se puede observar en la **Error!**

Reference source not found..

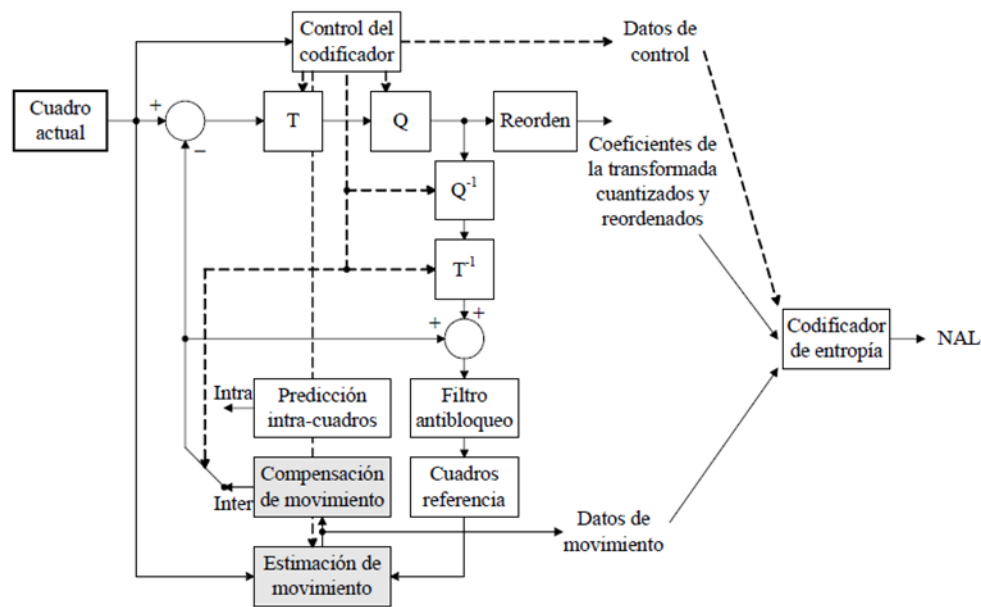


Figura 1 Codificador de video H.264. Adaptado de “Video compression from concepts to the H.264/AVC standard” por Gary J Sullivan, 2005.

Como se mencionó anteriormente el codificador H.264 no está estandarizado, por lo tanto, se pueden suprimir o adicionar etapas. El único precedente es que el flujo de video mantenga las características del estándar para que este pueda ser decodificado. El codificador H.264 básico sintetiza los antecedentes históricos del estándar y el funcionamiento de cada uno de los procesos. En la presente sección se describirá cada una de estas etapas.

Transformada discreta del coseno: es la herramienta matemática adoptada por la mayoría de los codificadores de video, esto se debe a que posee propiedades que se adaptan mejor a las señales de video y audio, logrando, una mayor concentración de energía en pocos coeficientes de la transformada [13]. La transformada discreta del coseno convierte los datos del dominio espacial al dominio de la frecuencia [14]. Es importante resaltar que el estándar emplea la transformada discreta bidireccional del coseno (DCT-2D) con el objetivo de alcanzar mayores niveles de compactación [15]. Para obtener la transformada bidireccional del coseno se pueden desarrollar

dos métodos que conllevan a la solución de la transformada, la primera es desarrollar la ecuación (1) que hace referencia a la transformada discreta del coseno en una dimensión y posteriormente, aplicando una técnica para obtener la DCT-2D [16].

$$C(u) = a(u) \sum_{x=0}^{N-1} f(x) \cos \left[\frac{\pi(2x+1)u}{2N} \right] \quad (1)$$

La segunda estrategia consiste en resolver la ecuación (2) que representa la transformada discreta del coseno en 2 dimensiones [17].

$$C(u) = a(u)a(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos \left[\frac{\pi(2x+1)u}{2N} \right] \cos \left[\frac{\pi(2y+1)v}{2N} \right] \quad (2)$$

Como es más sencillo desarrollar la transformada discreta del coseno en una dimensión, se aplica el desarrollo de la (1) y se implementa una técnica para lograr la DCT-2D. Primero, a la matriz o imagen se le aplica la DCT-1D, luego a la salida de este proceso que es una matriz, se le aplica su transpuesta y se le ejecuta nuevamente el proceso de DCT-1D. El proceso de transformada inversa implica realizar un símil del mismo proceso deshaciendo el procedimiento anteriormente descrito.

Cuantización: este proceso está directamente relacionado con la DCT-2D; cuando una imagen es transformada empleando la DCT se obtiene como resultado un bloque de coeficientes transformados que en teoría ha concentrado gran cantidad de energía en alguno de estos coeficientes, posteriormente, estos coeficientes son transformados en coeficientes por medio de la implementación de un parámetro de cuantización [18]. En H.264 se emplea la cuantización escalar

donde su principio de funcionamiento es la división de los datos de entrada por un parámetro de cuantificación, y realizando un proceso de aproximación o redondeo al entero más cercano [19].

Estimación de movimiento: La estimación de movimiento es el actor principal para obtener buenas tasas de compresión de video, ya que este permite reducir significativamente la cantidad de información que posteriormente deberá ser decodificada, sin embargo, es importante resaltar que el costo computacional de este proceso es considerablemente alto, encontrándose alrededor del 60% al 80% de los codificadores de video [20]. La ventaja para los sistemas de codificación es que el proceso no está estandarizado, esto quiere decir que siempre es posible crear nuevos métodos que permitan mejorar la estimación de movimiento o se adapten a las necesidades de la aplicación, siempre y cuando, se mantenga como se entregan los vectores de movimiento.

En la siguiente figura se pueden observar los diferentes tipos de algoritmos de estimación de movimiento.

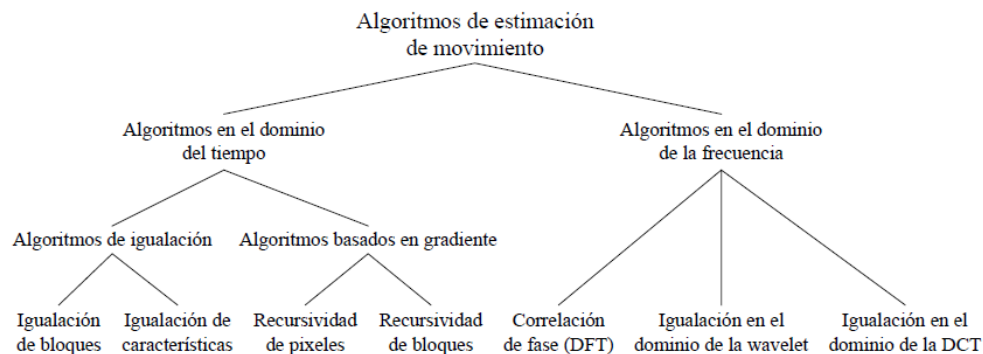


Figura 2 Algoritmos de estimación de movimiento. Adaptado de "Estudio de Arquitecturas VLSI de la etapa de predicción de la compensación de movimiento, para compresión de imágenes y videos." Por Armando Mora Campos, 2008.

Los algoritmos de estimación de movimiento se pueden clasificar en dos tipos: los algoritmos en el dominio de la frecuencia que emplean técnicas de transformación y correlación, y algoritmos

en el dominio del tiempo que se enfocan en emplear técnicas de igualación y recursividad donde su objetivo es la comparación de píxeles, bloques u otras características de los frames. Este trabajo se enfoca en los algoritmos de igualación de bloques o Block Matching, ya que es una técnica empleada comúnmente por los diferentes estándares de codificación de video, especialmente H.264 [20].

Para comprender las diferentes técnicas de compresión de video es importante realizar una revisión un poco más detallada de la redundancia en el video, ya que este es el eje fundamental de actuación en las diferentes técnicas asociadas a dicho concepto[4]. En la **Error! Reference source not found.** se puede observar un esquema general de las diferentes clases y subclases de redundancia de video dividiéndolos en dos categorías principales: la estadística y la psicovisual[3].

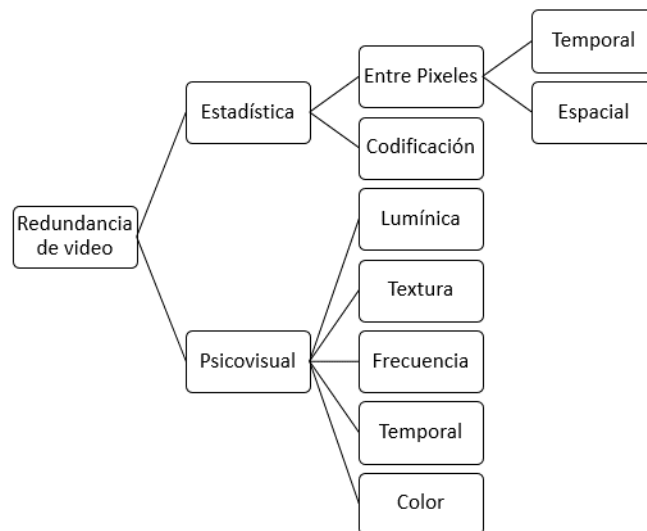


Figura 3 “Clases de redundancia en el video” Adaptado de "Estudio de Arquitecturas VLSI de la etapa de predicción de la compensación de movimiento, para compresión de imágenes y videos." Por Armando Mora Campos, 2008.

2.3 Redundancia de video

A continuación, se profundiza en cada una de las clases asociadas a la redundancia de video.

2.3.1 Redundancia estadística. Se basa en la dependencia de píxeles que existe, ya sea entre diferentes imágenes consecutivas (temporal) y/o la dependencia entre los bloques de píxeles de una misma imagen (espacial), a estos dos tipos se les denomina redundancia entre píxeles. En la estadística existe otro tipo de redundancia de datos que son provocados posterior a la realización del proceso de codificación, el detalle de cada una de estas se ilustra a continuación:

Redundancia espacial: está relacionada con la naturaleza de los objetos, es decir que se pueden encontrar objetos sólidos que poseen características físicas como el color, superficies y textura con uniformidad. La ventaja que ofrece esta redundancia en el video es que los píxeles que representan las características del objeto son similares, además, existe una relación entre un píxel y los píxeles colindantes permitiendo así que durante la transmisión del video no sea necesario enviar toda la información, solo bastaría con enviar un píxel de referencia y la diferencia que existe entre ellos para que pueda ser reconstruido. En la **Error! Reference source not found.** se evidencia la redundancia espacial donde el cuadro rojo representa un bloque de píxeles, allí se puede observar la similitud en los píxeles. Este proceso en el dominio del espacio indica que la compresión se realiza en la misma imagen y no tiene dependencia con una imagen de referencia en una secuencia de video. El resultado de eliminar la redundancia espacial es la representación de una imagen con menos datos sin perjudicar la calidad de la imagen.

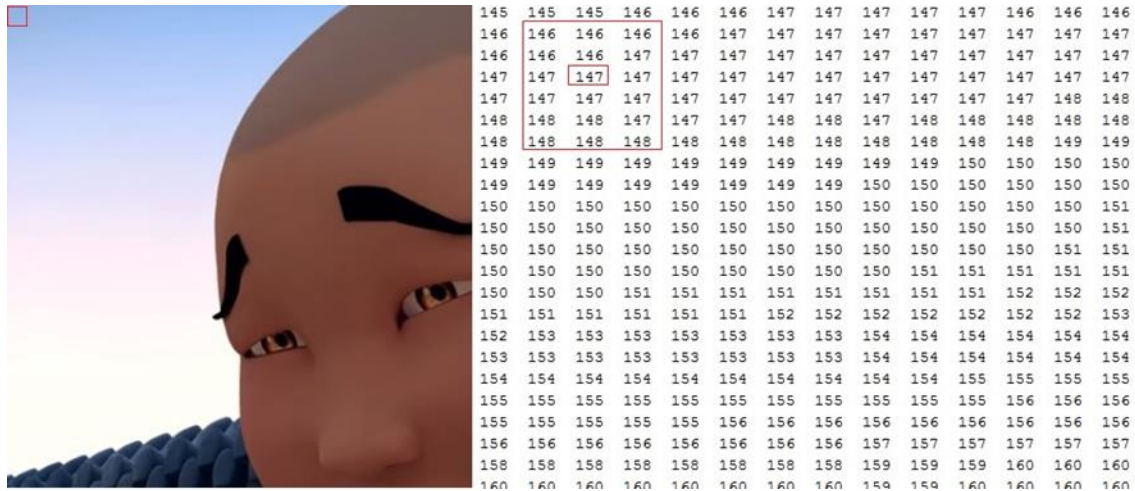


Figura 4.Redundancia espacial

Redundancia temporal: a esta redundancia se le suele denominar redundancia Inter-cuadro porque se fundamenta en la relación que existe entre los pixeles de cuadros consecutivos del video, esto quiere decir que gran cantidad de los elementos en una escena se repiten en el eje temporal[1]. En la **Error! Reference source not found.** se puede apreciar que solo la mariposa se mueve y los demás componentes de la imagen son estáticos entre diferentes imágenes.



Figura 5.Redundancia Temporal

En el desarrollo de técnicas que permiten reducir la redundancia temporal, se han destacado las técnicas que analizan el movimiento de los objetos permitiendo una predicción de las acciones del objeto sin realizar un análisis completo de las diferencias entre imágenes, logrando así, altos niveles de compresión de video con el uso de este tipo de técnicas[21], algunos ejemplos de técnicas son los algoritmos estimación de movimiento y la codificación diferencial de pulsos modulados (DPCM).

Redundancia de codificación: el proceso de codificación genera un tipo de redundancia propia de la cuantización de la señal, esta redundancia está presente en cualquier proceso de codificación[1], como es el caso de las estrategias de codificación Hoffman[22], [23] y/o codificación aritmética [24], [25].

2.3.2 Redundancia Psicovisual. La segunda clase es la redundancia psicovisual que corresponde a las características del video respecto a la experiencia que provoca en el espectador, en ella se pueden encontrar la redundancia de luminancia, textura, frecuencia, temporal y color. Es importante hacer énfasis en que, si bien la redundancia psicovisual tiene un efecto sobre la experiencia del usuario, no es una redundancia netamente subjetiva, ya que lo que se busca eliminar son las características del video que no perjudican la calidad del mismo[1].

2.4 Algoritmos de Block Matching

En el presente capítulo se profundizará en el concepto general de los algoritmos de estimación de movimiento basados en Block Matching, para ellos es fundamental hacer revisión de las

características propias del algoritmo como los criterios de búsqueda, el tamaño del macrobloque, las ventanas de búsqueda o rangos de búsqueda y la diversidad de métodos de estimación de movimiento basados en Block Matching.

El principio de los algoritmos de Block Matching consiste en segmentar un frame en diferentes bloques también llamados macrobloques, y posteriormente compararlos entre ellos, para esto, la división debe realizarse tanto para un frame de referencia, como para los actuales; estas imágenes son temporalmente diferentes, pero contienen mucha información redundante entre sí. Después de realizar esta división cada bloque actual es comparado con los bloques de referencia en un área de búsqueda, para realizar esta comparación se debe considerar que todos los píxeles en un mismo bloque presentan el mismo desplazamiento, por tal razón, no es necesario calcular el vector de movimiento por píxel y solo bastará un vector por bloque. Cuando se encuentra un bloque igual entre los dos frame en la ventana de búsqueda, se calcula el desplazamiento del bloque[26].

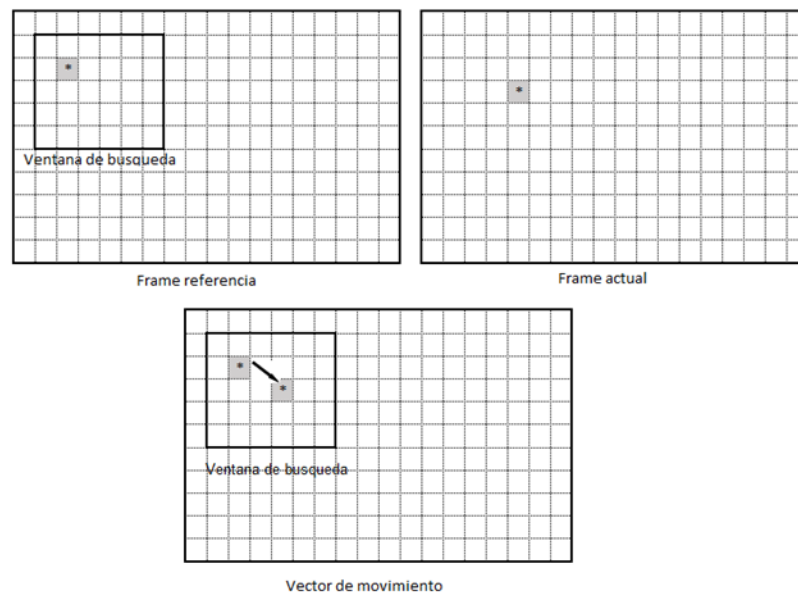


*Figura 6.*Concepto de Block Matching

Poder calcular estos vectores de movimiento permitirá remover los datos redundantes en las imágenes, logrando mayor compresión de los datos ya que solo será necesario para los decodificadores reconstruir la imagen a partir de los vectores de movimiento y la imagen de referencia. Es importante resaltar que la estimación de movimiento basado en Block Matching no es el único método para obtener estos vectores, pero sí es el método más sencillo y común para encontrarlo.

2.5 Criterios de búsqueda

Block Matching puede emplear diferentes técnicas que permiten calcular el error de ajuste entre dos bloques comparados con la imagen actual y la imagen de referencia, esto indica que el criterio de búsqueda permite determinar que tanta similitud existe entre los bloques y así, a partir de este dato poder determinar los vectores de movimiento. Entre los criterios de búsqueda más populares en la estimación de movimiento basada en la técnica Block Matching, se pueden encontrar la suma de errores al cuadrado (SSE), error absoluto medio (MAE), suma de diferencias transformadas absolutas (SATD) y la suma de diferencias absolutas (SAD), técnica que se empleará en el desarrollo de este trabajo[27].

$$SSE = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} (X_{ij} - Y_{ij})^2 \quad (3)$$

Donde X_{ij} es un pixel del macrobloque de la imagen de referencia y Y_{ij} es un macrobloque de la imagen actual.

$$MAE = \frac{1}{M \times N} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} |X_{ij} - Y_{ij}| \quad (4)$$

Donde X_{ij} es un pixel del macrobloque de la imagen de referencia y Y_{ij} es un macrobloque de la imagen actual.

$$SATD = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} |H_{2D}(X_{ij} - R_{ij})| \quad (5)$$

Donde X_{ij} es un pixel del macrobloque de la imagen de referencia, Y_{ij} es un macrobloque de la imagen actual y H_{2D} es la transformada de Hadamard 2-D.

Sum of absolute error (SAD), o sumatoria de diferencias absoluta es un método que permite comparar pixel a pixel, los bloques o macrobloques de una imagen de referencia y una imagen actual empleando el error como herramienta. Este es uno de los algoritmos menos exigentes computacionalmente, ya que su cálculo puede segmentarse fácilmente en tres diferentes pasos, el primero; cálculo de las diferencias entre cada pixel tanto del bloque de referencia como el bloque actual, determinación del valor absoluto de esa diferencia y, por último, la suma de estas diferencias. El SAD se puede representar matemáticamente de la siguiente forma:

$$SAD(X, Y) = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} |X_{ij} - Y_{ij}| \quad (6)$$

Donde X_{ij} es un pixel del macrobloque de la imagen de referencia y Y_{ij} es un macrobloque de la imagen actual.

2.6 Tamaño del Macrobloque

Este es otro de los elementos fundamentales a tener presente al momento de emplear una técnica como Block Matching, el tamaño del macrobloque es un factor importante ya que este influye directamente en la calidad de la predicción y también en el procesamiento del algoritmo. Un ejemplo, es que si se definen macrobloques pequeños se podrá percibir el movimiento de los pixeles con mayor precisión, pero generará un mayor costo computacional ya que los frames se

dividirán en muchos macrobloques y, por lo tanto, el algoritmo deberá recorrer mayor cantidad de macrobloques encontrando mayor cantidad de vectores de movimiento, además de necesitar mayor espacio de almacenamiento.

Actualmente los estándares de codificación acostumbran a emplear macrobloques 16 x 16, pero es posible emplear otra clase de macrobloques como los que se pueden observar en la siguiente imagen.

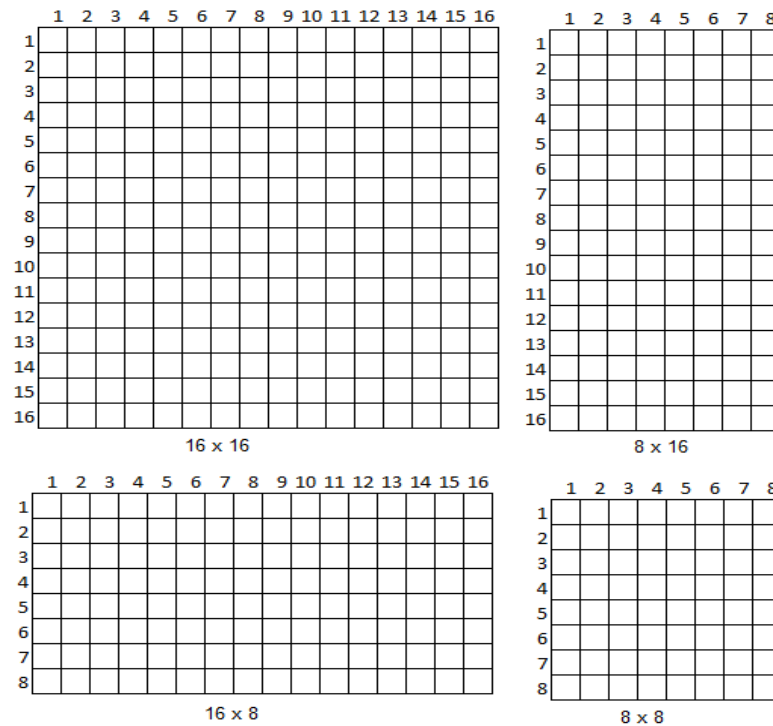


Figura 7. Macrobloques

2.7 Rango de búsqueda

Esta característica es un factor que influye de forma directa a nivel computacional, emplear un rango de búsqueda pequeño podrá reducir significativamente el procesamiento, pero reducirá la efectividad de predecir los vectores de movimiento, en cambio, un rango de búsqueda muy grande permitirá predecir con mayor certeza los vectores de movimiento, pero generará un costo computacional alto al intentar buscar las coincidencias de todos los macrobloques que cambiaron.

En soluciones software de Block Matching la ventana de búsqueda o rango de búsqueda puede ser de $(M+2Ph) \times (N+2Pv)$, es decir el tamaño del macrobloque más dos veces, un posible desplazamiento horizontal y vertical[1].

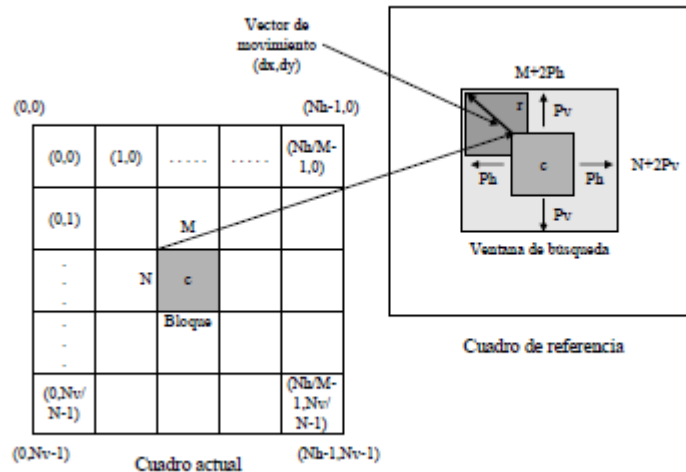


Figura 8. Rango de búsqueda y vector de movimiento. Adaptado de "Estudio de Arquitecturas VLSI de la etapa de predicción de la compensación de movimiento, para compresión de imágenes y videos." Por Armando Mora Campos, 2008.

2.8 Algoritmos de estimación de movimiento basados en Block Matching

Respecto a los algoritmos de estimación de movimiento se han publicado gran variedad que pueden ser implementados en los codificadores de video, estos algoritmos se diferencian en las características mencionadas anteriormente como los criterios de búsqueda, tamaño de macrobloque y rango de búsqueda. A continuación, se profundizará en algunos algoritmos de estimación de movimiento.

2.8.1 Búsqueda Exhaustiva (ES). Este algoritmo se fundamenta en una búsqueda macrobloque a macrobloque a través de todo el frame con el objetivo de encontrar el macrobloque homólogo entre el frame actual y el frame de referencia, la selección del macrobloque se hace por medio de los criterios de búsqueda que permitirán definir cuáles son entre los macrobloques comparados, los que presentan el mínimo error. Las desventajas de este algoritmo es que al ser una búsqueda de posición a posición, el costo computacional es alto pero su ventaja es conceptualmente fácil de desarrollar[28].

Teniendo presente la cantidad de operaciones en la búsqueda exhaustiva, lo que genera una alta carga computacional, se han desarrollado otros algoritmos que permiten aumentar en la velocidad y cantidad de operaciones en las que se realizan las búsquedas, pero reduciendo la precisión para encontrar los vectores de movimiento.

2.8.2 Algoritmo en tres pasos (TSS). Este es un método que mejora considerablemente el coste computacional a comparación de la búsqueda exhaustiva, el procedimiento de este algoritmo se basa en definir un primer paso (S), este primer paso indica que desde un punto se evaluarán macrobloques a (S) de distancia en 8 posiciones diferentes, alrededor del punto central o pixel central, 4 de estas 8 posiciones se encuentran a una distancia raíz del doble del tamaño de S y 4, a la distancia de S, ahora se evalúa con los criterios de búsqueda cuál de estos tiene el menor error; desde el bloque con menor error se posiciona para realizar la segunda búsqueda, se realiza el mismo procedimiento con S reducido a la mitad y se va encontrando un nuevo punto hasta lograr que S llegue a ser 1[29].

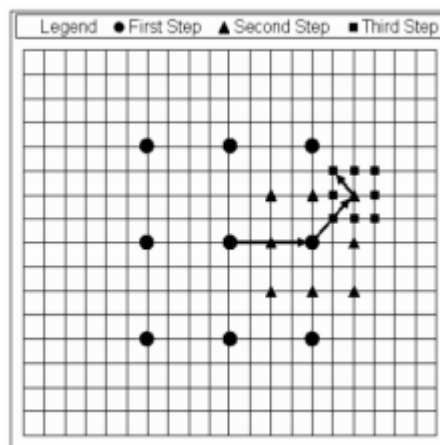


Figura 9. Algoritmo en tres pasos, Adaptado de “Estimación de Movimiento” por I. L. Espejo, 2012.

2.8.3 Nueva búsqueda de tres pasos (NTSS). Este algoritmo fue altamente usado en los estándares de codificación MPEG- y H.261, su principio de funcionamiento se basa en la elección en un punto central escogiendo en total 8 puntos a una distancia $S=1$, y 8 puntos a una distancia $S=4$. El algoritmo realiza una comparación de todos los valores estableciendo dos casos posibles, el primero; si el punto central es el punto de menor valor, el algoritmo finaliza, pero si es alguno de los 8 puntos a distancia $S=1$, se reasigna como punto central el valor encontrado y se realiza el mismo procedimiento hasta encontrar el macrobloque con el valor mínimo. En la siguiente Figura 10 se representa el procedimiento[29].

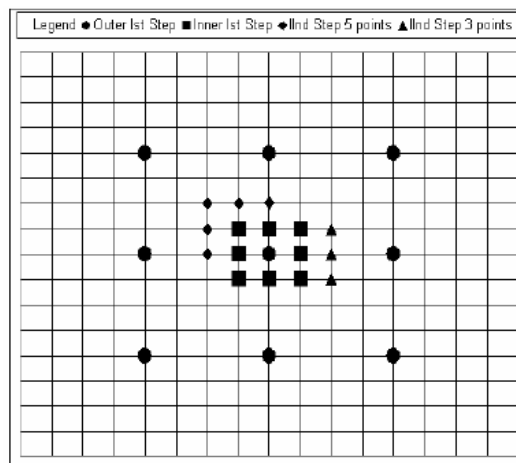


Figura 10. Nueva búsqueda en tres pasos (NTSS), Adaptado de “Estimación de Movimiento” por I. L. Espejo, 2012.

2.8.4 Búsqueda simple y eficiente (SES). El principio de este algoritmo se basa en el funcionamiento del TSS que básicamente se divide en tres pasos, el área es dividida en cuadrantes realizando comprobaciones en tres posiciones; A es el punto central, B y C son puntos a una distancia $S=4$ a partir del punto central A, B entre cuadrante 1 y 4, y C entre el cuadrante 3 y 4 como se evidencia en la siguiente Figura 11.

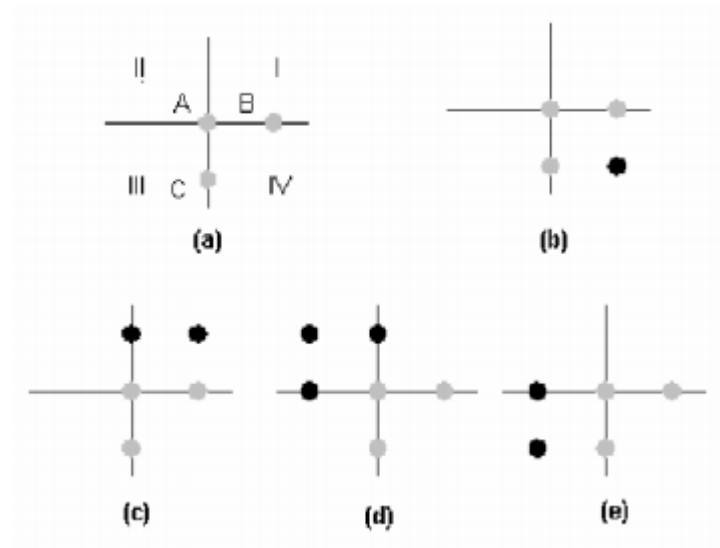


Figura 11. Patrones de búsqueda SES, Adaptado de “Estimación de Movimiento” por I. L. Espejo, 2012.

Se realiza evaluación de cada uno de los puntos y dependiendo el resultado se escoge el patrón que se debe implementar en esta búsqueda.

$SAD(A) \geq SAD(B) \ \& \ SAD(A) \geq SAD(C)$ Seleccionar patrón de búsqueda Figura 11 (b)

$SAD(A) \geq SAD(B) \ \& \ SAD(A) \leq SAD(C)$ Seleccionar patrón de búsqueda Figura 11 (c)

$SAD(A) < SAD(B) \ \& \ SAD(A) < SAD(C)$ Seleccionar patrón de búsqueda Figura 11 (d)

$SAD(A) < SAD(B) \ \& \ SAD(A) \geq SAD(C)$ Seleccionar patrón de búsqueda Figura 11 (e)

Al seleccionar el patrón de búsqueda el algoritmo realiza operaciones para evaluar los puntos seleccionados por el patrón, el algoritmo concluirá cuando encuentre un punto en $S=1$ [29].

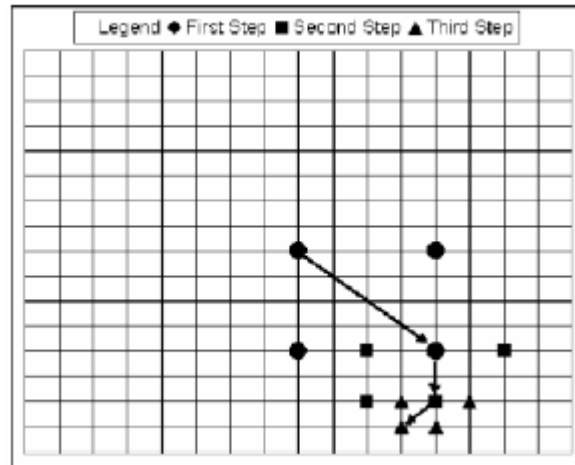


Figura 12. Búsqueda simple y eficiente (SES), Adaptado de “Estimación de Movimiento” por I. L. Espejo, 2012.

2.8.5 Búsqueda en cuatro pasos (4SS). Este algoritmo fija un punto central y 8 puntos para ser evaluados a $S=2$ alrededor del centro, por lo tanto, se realizan comparaciones en los 8 puntos alrededor del centro teniendo una ventana de búsqueda de 5×5 . En este método, se pueden tener algunas condiciones; si se encuentra el error mínimo en el centro, el algoritmo salta al último paso que es una búsqueda entre el punto central y los puntos a la distancia $S=2$. En el caso que el primer paso encuentre el error mínimo en los puntos ubicados a $S=2$, tomará este, como nuevo referente al que posee el error mínimo y empezará a comparar nuevamente con los puntos colindantes, si este encuentra que el error mínimo es el nuevo central pasará automáticamente al paso final, si no, este seguirá buscando el mejor vector[30].

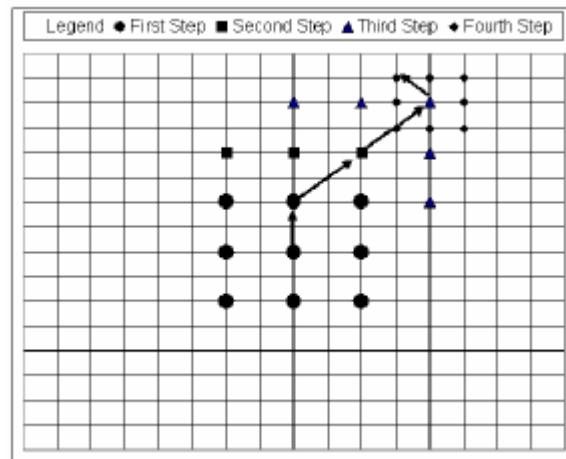


Figura 13. Búsqueda en 4 pasos (4SS), Adaptado de “Estimación de Movimiento” por I. L. Espejo, 2012.

2.8.6 Método búsqueda en diamante. El algoritmo búsqueda en diamante tiene características de operación similares a las expuestas en el método 4SS, entre las diferencias que se pueden encontrar respecto al anterior método es la elección de los puntos a $S=2$ en forma de rombo, otra de las diferencias es la eliminación del límite de búsquedas que hace el algoritmo. El algoritmo culmina cuando encuentra el error mínimo[26].

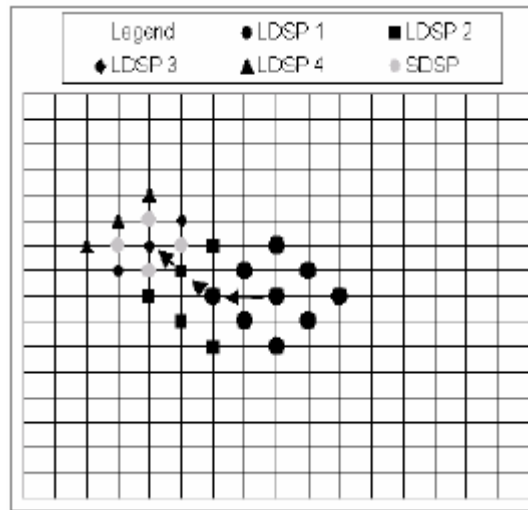


Figura 14. Búsqueda en Diamante, Adaptado de “Estimación de Movimiento” por I. L. Espejo, 2012.

2.9 Elección del Algoritmo

En base a la revisión de los algoritmos de estimación de movimiento se eligió el método de búsqueda exhaustiva debido a que es el método con lógica más sencilla ideal para un desarrollo que pueda implementarse en una FPGA. Las desventajas para este método es que utilizará mayor cantidad de recursos lógicos y sus altos tiempos de procesamiento, que a su vez pueden ser mitigados paralelizando el algoritmo.

3. Método

En el presente capítulo se presentarán los requerimientos que enfocan la propuesta de algoritmo de estima con de movimiento basado en métodos Block Matching, se realiza una revisión general de la tarjeta FPGA empleada, además de una descripción del proceso que se presentó con el desarrollo del algoritmo, las practicas necesarias para comprender el principio del Block Matching y las diferentes versiones del algoritmo desarrollado.

3.1 Requerimientos

Ante la falta de interoperabilidad entre diferentes herramientas de monitoreo en el área de la medicina, y con el objetivo de lograr un prototipo de transmisor universal VGA que sea capaz de codificar video a formatos estándar como h.264, el grupo de investigación de Bioingeniería de la Fundación Cardiovascular de Colombia, optó por que el desarrollo sea realizado en base a una tarjeta electrónica (hardware), con unidad de procesamiento que permita correr algoritmos (software), por tal motivo, la herramienta de trabajo va enfocada a un desarrollo sobre una tarjeta FPGA Altera Cyclone V.

3.2 Tarjeta de desarrollo DE0-Nano

El dispositivo requerido para este desarrollo es la tarjeta DE0-Nano de Altera, esta tarjeta posee las siguientes características:

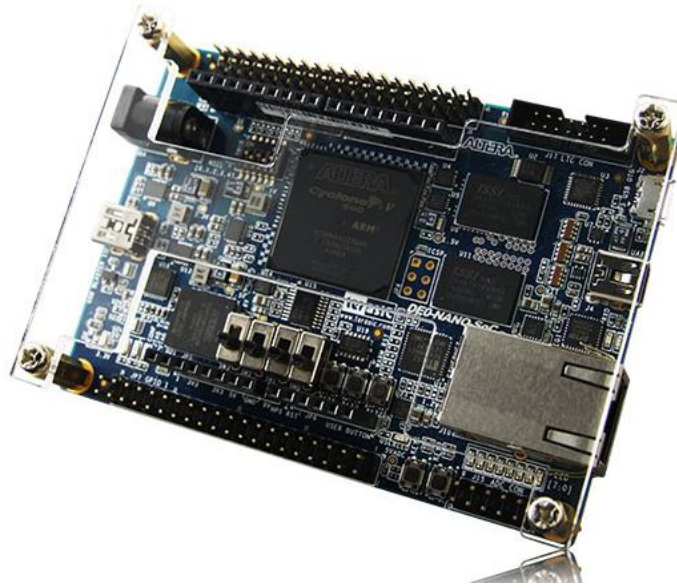


Figura 15. DE0-nano FPGA Altera, Adaptado de "User Manual for DE0 Nano Board" por Altera, 2016

Las siguientes son las características FPGA.

- Altera Cyclone V 5CSEMA4U23C6N.
- USB Blaster II.
- 2 pulsadores.
- Dip switch 4 posiciones.
- 8 leds.
- osciladores de 50 MHz.
- 2 x 20 GPIO.
- Expansión arduino.
- 10 pin análogos.
- Convertidos A/D.
- Las siguientes son las características del procesador HPS
- 925MHz Dual Core ARM Cortex-A9.
- 1GB DDR3 SDRAM.
- Puerto Giga Ethernet RJ45.
- Puerto USB OTG, Micro USB.
- Acelerómetro.
- UART a USB, USB mini.

- Botón reset.

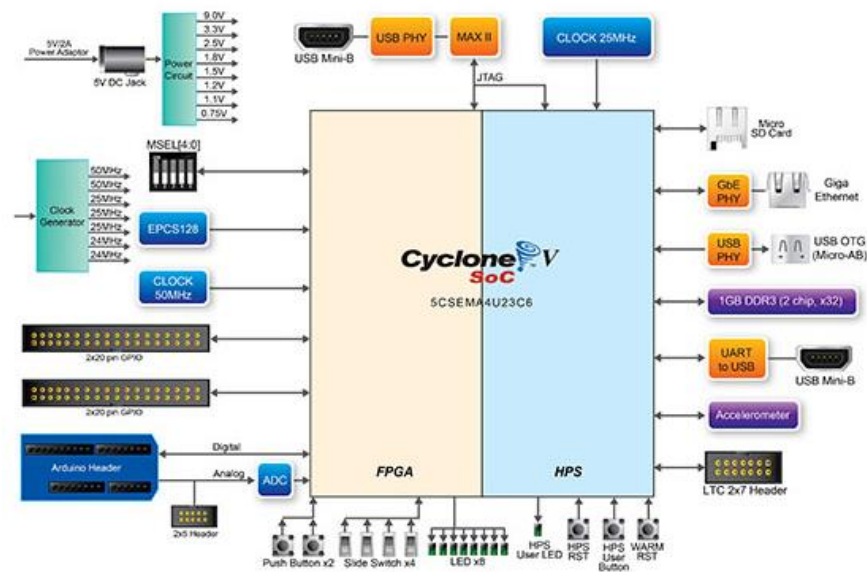


Figura 16. Diagrama de bloques DE0 nano, Adaptado de "User Manual for DE0 Nano Board" por Altera, 2016

3.3 Desarrollo

El desarrollo del proyecto toma como eje fundamental la manipulación a nivel de pixel de imágenes, por lo tanto, el desarrollo del mismo es un avance en cómo poder interactuar dichos pixeles que componen las imágenes por medio de una FPGA con el objetivo de llegar al resultado final, que es un algoritmo de estimación de movimiento. A continuación, se presenta una serie de prácticas realizadas para apropiar conocimientos de codificación específicamente en la estimación de movimiento, y acelerar el aprendizaje con hardware programable específicamente con FPGA De0 nano de altera.

El primer paso fue la generación de imagen de video en formato VGA; en primera instancia se generó el video desde la FPGA, ya que es posible emplearla como recurso multimedia constante

para el desarrollo del algoritmo, en esta etapa se decidió emplear un diseño para videojuegos el cual consta de un sincronizador VGA.

La segunda etapa se basa en la necesidad de manipular la imagen pixel a pixel y, por ende, se decide almacenar la imagen generada en una memoria y restar la siguiente imagen obteniendo como resultado, una imagen residuo entre la imagen de referencia y la imagen actual.

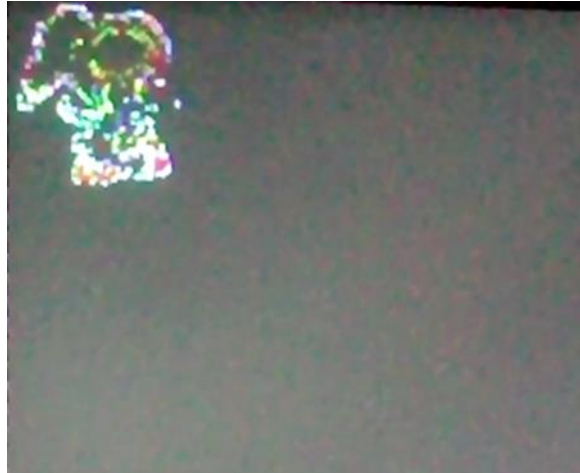


Figura 17. Imagen residuo

Para lograr obtener una imagen residuo fue necesario desarrollar un algoritmo que permitiera realizar una resta pixel a pixel entre dos imágenes, es por ello que se desarrolló una máquina de estados finitos que permite este tipo de exploración logrando obtener una imagen residuo. En la Figura 18, se presenta un fragmento de código del algoritmo que permitió realizar la diferencia entre un frame de referencia y un frame actual a nivel de pixeles, el algoritmo comparaba pixel a pixel los dos frames y determinaba cuál de los dos posee un mayor valor, al definir esto era almacenado en una variable y realizaba a continuación una resta, evitando así que se presentaran valores negativos.

```

wire [3:0]new_pixel;

wire [1:0]data_mayor= (previous_pixel[3:2]>current_pixel[3:2])?previous_pixel[3:2]:current_pixel[3:2];
wire [1:0]data_menor= (previous_pixel[3:2]>current_pixel[3:2])?current_pixel[3:2]:previous_pixel[3:2];
assign new_pixel[3:2]=data_mayor[1:0]-data_menor[1:0];
assign new_pixel[1]  =previous_pixel[1] ^ current_pixel[1];
assign new_pixel[0]  =previous_pixel[0] ^ current_pixel[0];
//-----

```

Figura 18. Código resta pixeles entre frame referencia y frame actual

Posterior a la imagen residuo se planteó como objetivo representar el concepto de Sum Absolute Difference (SAD), el cual consiste en realizar restas de los pixeles posición a posición entre la imagen actual y la imagen de referencia como se realizó en la segunda etapa y sumarlas. Para implementar SAD es importante dividir la imagen en macrobloques, para el caso se implementó macrobloques de 16X16 y 8X8, los cuales se recomiendan en la técnica de estimación de movimiento. Al dividir en macrobloques la imagen, se logra tener un valor de referencia al sumar el SAD en el macrobloque, dato de relevancia para realizar el proceso de estimación de movimiento.

En la Figura 19 se presenta la división de macrobloques de una imagen de prueba, la posición de la imagen y dónde se pueden presentar dos casos. El primero es que el SAD sea diferente a cero, por lo tanto, se evidenciará la suma de diferencias en un macrobloque, y el segundo caso; SAD igual a cero, lo que indicaría que los dos pixeles en la misma posición presentan el mismo valor y, por ende, el resultado es una anulación.



Figura 19. Macrobloques

A partir de los avances anteriormente descritos, se procede a desarrollar como el algoritmo va a barrer la imagen, se plantea el uso de contadores que corresponderán a la posición de macrobloque estos cada posición comparará el SAD en cada posición con el objetivo de encontrar los cambios o movimientos entre las imágenes. En el siguiente numeral se presenta una revisión general del funcionamiento del algoritmo.

3.4 Algoritmo

El algoritmo está desarrollado para realizar la estimación de movimiento empleando el método Block Matching búsqueda exhaustiva, donde dos frames, uno de referencia y uno actual son comparados respectivamente buscando dónde los macrobloques han cambiado definiendo un vector de movimiento. Para ello el algoritmo se basa en una máquina de estados que barre la imagen progresivamente, buscando cuando se han presentado cambios entre las posiciones de los macrobloques y usando como referencia el cálculo del SAD.

La máquina consta de 15 estados como se presenta en la siguiente Figura 20.

	wr_ref	wr_act	img_wr_hps	vector_wr_hps	finish	idle	incre_ref	incre_act	rst_ref	rst_act	state
localparam IDLE	=15'b0	0	0	0	0	1	0	0	1	1	00000;
localparam READ_MEM	=15'b0	0	0	0	0	0	0	0	0	0	00001;
localparam BUSCAR_SIMILAR	=15'b0	0	0	0	0	0	0	0	0	0	00010;
localparam GUARDAR_VECTOR_LOAD	=15'b0	0	0	0	0	0	0	0	0	0	00011;
localparam GUARDAR_VECTOR_WRITE	=15'b0	0	0	1	0	0	0	0	0	0	00100;
localparam SET_REF_BIT_AND_ACT_BIT_1_LOAD	=15'b0	0	0	0	0	0	0	0	0	0	00101;
localparam SET_REF_BIT_AND_ACT_BIT_1_WRITE	=15'b1	1	0	0	0	0	0	0	0	0	00110;
localparam INCREASE_REF_1	=15'b0	0	0	0	0	0	1	0	0	0	00111;
localparam INCREASE_REF_AND_ACT	=15'b0	0	0	0	0	0	1	1	0	0	01000;
localparam INCREASE_ACT	=15'b0	0	0	0	0	0	0	1	0	0	01001;
localparam SET_ACTREF	=15'b0	0	0	0	0	0	0	0	0	0	01010;
localparam SET_REF_BIT_AND_ACT_BIT_2_LOAD	=15'b0	0	0	0	0	0	0	0	0	0	01011;
localparam SET_REF_BIT_AND_ACT_BIT_2_WRITE	=15'b1	1	0	0	0	0	0	0	0	0	01100;
localparam SET_REF_BIT_1_LOAD	=15'b0	0	0	0	0	0	0	0	0	0	01101;
localparam SET_REF_BIT_1_WRITE	=15'b1	0	0	0	0	0	0	0	0	0	01110;
localparam RESET_REF_BEFORE_SAVING_IMG	=15'b0	0	0	0	0	0	0	0	1	0	01111;
localparam LOAD_REF_2_IMG_FX	=15'b0	0	0	0	0	0	0	0	0	0	10000;
localparam WRITE_REF_2_IMG_FX	=15'b0	0	1	0	0	0	0	0	0	0	10001;
localparam INCREASE_REF_2_IMG	=15'b0	0	0	0	0	0	1	0	0	0	10010;
localparam FINISH	=15'b0	0	0	0	1	0	0	0	1	1	10011;

Figura 20. Máquina de estados Algoritmo Estimación de Movimiento

Idle es un estado de espera de señal de inicio, al presentarse dicha señal el algoritmo avanzará de estado. En la Figura 21 se presenta el estado Idle del algoritmo desarrollado.

```

IDLE
begin
    state[14:0] <= IDLE;
    if (start)
    begin
        state[14:0] <= READ_MEM;
    end
end

```

Figura 21. Estado Idle

Read_mem es el primer estado donde se toma el macrobloque analizado y se identifica si su posición es adecuada para seguir la búsqueda, para el caso de un frame VGA 640 X 480 y tomando macrobloques de 8 X 8, se tendría una resolución de 80 macrobloques por 60 macrobloques, se puede obtener un rango límite de 0 a 4800 macrobloques, en caso que el contador referencia sobrepase el límite de referencia, el contador volverá a iniciar con un nuevo frame. En la Figura 22 se presenta el fragmento de código referente al estado.

```

READ_MEM                                     :
begin
    state[14:0] <= BUSCAR_SIMILAR;
    if (ref[`MSBI:0] >= (limit_ref[`MSBI:0] - 1'b1))
    begin
        state[14:0] <= RESET_REF_BEFORE_SAVING_IMG;
    end
end

```

Figura 22. Estado READ_MEM

El estado BUSCAR_SIMILAR es uno de los estados más relevantes del algoritmo, ya que este se encarga de definir si un macrobloque en el frame de referencia cambió en el frame actual. En la Figura 23 se presenta el código para este estado, el cual inicia consultando si el macrobloque de referencia es diferente al macrobloque actual en la posición actual y de referencia, en caso de ser diferentes, el algoritmo valida si el contador que define la posición del macrobloque de la imagen actual está dentro de la ventana límite, si cumple esta condición el contador actual incrementará una posición con el estado INCREASE_ACT, y empezará a realizar nuevamente el proceso hasta cuando no se cumpla la comparación, es decir los dos macrobloques son iguales, o hasta que el contador actual sea mayor a la ventana límite, esto causa que en el estado SET_REF_BIT_LOAD se aumente el contador de referencia y el contador actual tome su valor inicial, volviendo a realizar la comparación nuevamente para el siguiente macrobloque de referencia.

```

BUSCAR_SIMILAR
begin
  if (data_rd_img_ref[23:0] != data_rd_img_Act[23:0] )
  begin
    if (act['MSBI:0] < (window_limit['MSBI:0]-1'b1))
    begin
      state[14:0] <= INCREASE_ACT;
    end
    else
    begin
      state[14:0] <= SET_REF_BIT_1_LOAD;
    end
  end
  end
  else
  begin
    if (act['MSBI:0] == ref['MSBI:0])
    begin
      state[14:0] <= SET_REF_BIT_AND_ACT_BIT_1_LOAD;
    end
    else
    begin
      if (ref['MSBI:0] >= (limit_ref['MSBI:0]-1'b1))
      begin
        state[14:0] <= RESET_REF_BEFORE_SAVING_IMG;
      end
      else
      begin
        state[14:0] <= GUARDAR_VECTOR_LOAD;
      end
    end
  end
end
end

```

Figura 23. Estado BUSCAR SIMILAR

Cuando el algoritmo encuentra que los macrobloques no son diferentes, se consulta la posición en donde fue encontrado, es decir el valor que tiene el contador actual; si estos contadores, tanto el actual como el de referencia son iguales, se puede deducir que entre los frames en la posición del macrobloque no se presentó ninguna variación y el algoritmo cambia al estado SET_REF_BIT_AND_ACT_BIT_1_LOAD. Este estado causa que el contador actual sea ubicado en la posición del contador de referencia y los dos aumentarán una posición.

En el caso que los macrobloques sean iguales en una posición diferente del frame actual, se procederá al estado GUARDAR_VECTOR_LOAD, el cual almacenará el vector encontrado. Si el contador referencia es igual o sobrepasa el límite de la imagen, este automáticamente concluirá el proceso de estimación de movimiento.

4. Resultados

A continuación, se presenta un resumen de los resultados obtenidos con el desarrollo del algoritmo de estimación de movimiento basado en el método de Block Matching.

4.1 Simulación

Para la simulación se escogieron dos frames diferentes y consecutivos temporalmente, además de ser parte de un video proveniente de un monitor de signos vitales. La Figura 24 es la imagen o frame seleccionado como referencia.

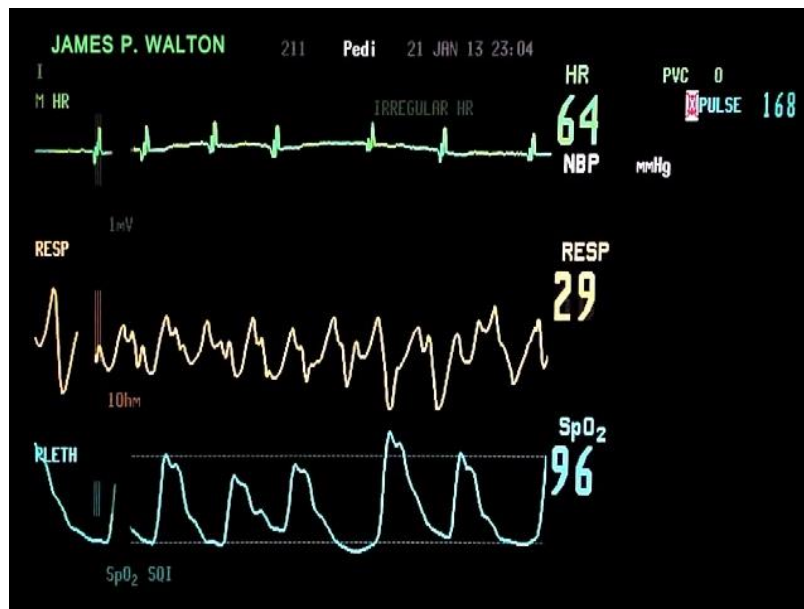


Figura 24. Imagen de Referencia

La Figura 25 es un frame temporalmente posterior a la Figura 24, estos dos frames serán analizados macrobloque a macrobloque con el objetivo de determinar sus vectores de movimiento.

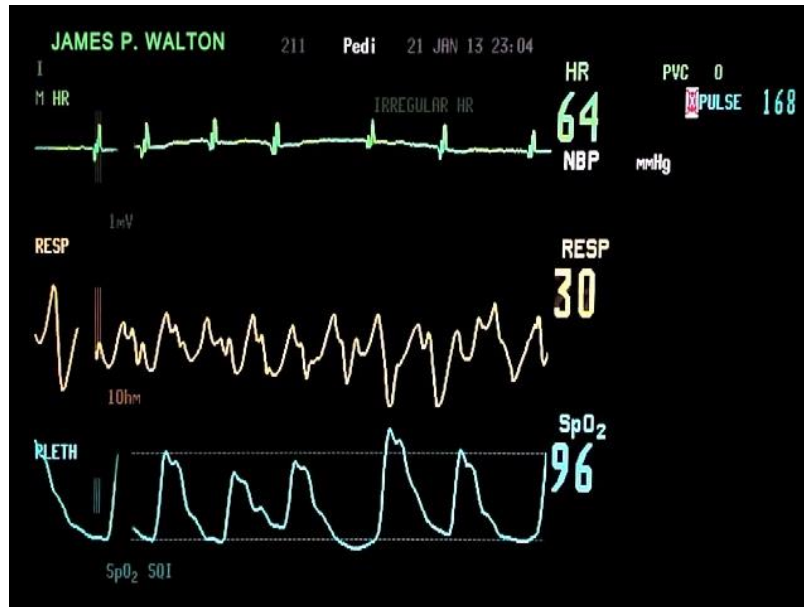


Figura 25. Imagen Actual

Antes de proceder a la simulación es necesario preparar las imágenes con el objetivo que se puedan agregar a las memorias RAM, por lo tanto, las imágenes son previamente convertidas a macrobloques y se les aplica el criterio SAD para posteriormente poder aplicar el algoritmo de estimación de movimiento, permitiendo calcular los vectores de movimiento. Un ejemplo del resultado de estos procesos se puede observar en la Figura 26, una imagen de referencia y una imagen actual convertidas en macrobloque y calculando el SAD para cada una.

243	000000	243	000000
244	000001	244	000001
245	00009C	245	00009C
246	00007C	246	00007C
247	00007C	247	00007C
248	000039	248	00003A
249	00000B	249	00000B
250	000019	250	000019
251	000006	251	000006
252	00000F	252	00000F
253	000003	253	000003
254	0000A3	254	0000A3
255	00007F	255	00007F
256	000077	256	000077
257	000055	257	000055
258	00001C	258	00001C
259	00008C	259	00008C
260	000099	260	000099
261	00008D	261	00008D
262	000006	262	000006
263	000000	263	000000
264	000000	264	000000
265	000000	265	000000
266	000000	266	000000
267	000000	267	000000
268	00000A	268	00000A

(a)
(b)

Figura 26. Imagen con cálculo del SAD, (a) Frame Referencia, (b) Frame Actual

En la Figura 26, se puede observar un ejemplo de macrobloques diferentes en la misma posición , cuando el algoritmo llegue a ese punto empezara a buscar sobre el frame actual hasta encontrar el mismo macrobloque o terminar el barrido de todo el frame.

La simulación del algoritmo consta de tres diferentes archivos verliog que son agregados a la herramienta de simulación simulink.

- Busqueda.v, es el algoritmo de estimación de movimiento diseñado.
- Ram_simulacion.v, contiene el diseño de dos memoria RAM con la capacidad de almacenar 4800 datos. La primera memoria RAM es para la imagen de referencia y la segunda para la imagen actual.
- Simulacion_test_busqueda.v, es el código que permite el control de la memoria RAM y el algoritmo de estimación de movimiento.

busqueda.v	✓	Verilog 1	05/22/2017 10:44:38 ...
ram_simulacion.v	✓	Verilog 2	08/21/2016 07:28:42 ...
simulacion_test_busqueda.v	✓	Verilog 0	05/22/2017 01:03:26 ...

Figura 27. Archivos verilog para la simulación

Al tener los tres archivos verilog cargados en el simulador se procede a compilarlos y posteriormente a simularlos. Para dicha simulación es necesario establecer parámetros iniciales como señal de reloj de 50 Mhz, además de establecer la ventana de búsqueda que en este caso será el tamaño del frame (4800 Macrobloques).

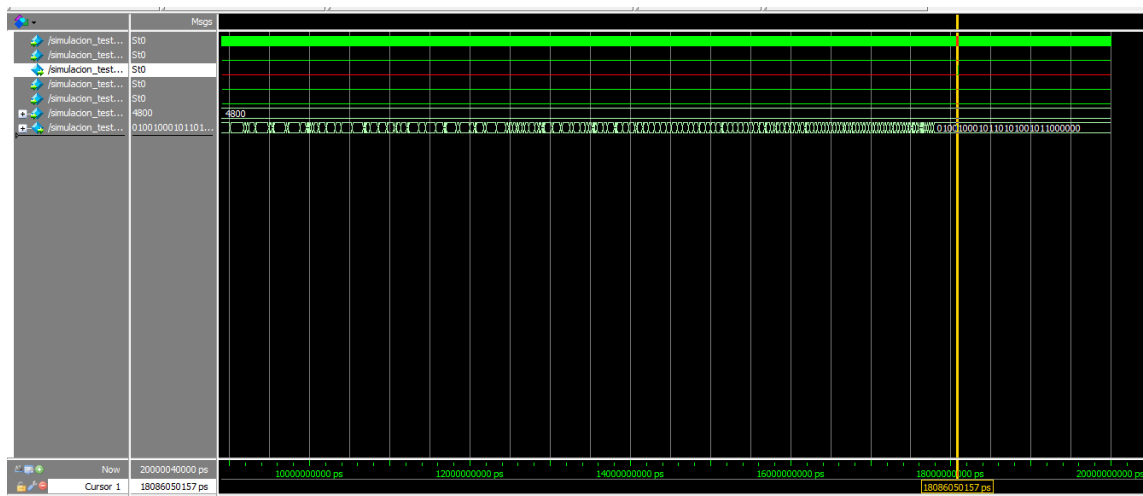


Figura 28. Simulación Algoritmo estimación de movimiento en Simulink

La simulación obtiene como resultado que fueron necesarios 18 ms para encontrar todos los vectores de movimiento de los dos frames seleccionados. En total fueron encontrados 576 vectores de movimiento esto indica que de 4800 macrobloques se logró determinar que 576 macrobloques presentaron un movimiento, ver la Figura 29.

248	00100110001000	00100110001001
249	00100110001001	00100110010001
250	00100110001111	00110010110110
251	00100110010000	00101010111001
252	00100110010011	00100110011011
253	00100110010110	00111010010110
254	00100110010111	00100110111011
255	00100110011000	00100110111111
256	00100110011001	00101001100100
257	00100110110011	01000001011111
258	00100110110100	00111001110110
259	00100110110110	00100110111001
260	00100110111000	01001011000000
261	00100110111010	01001011000000
262	00100111001011	00101001110111
263	00100111001100	00101000100010
264	00100111001110	00100111010110
265	00100111010100	00101010101100
266	00100111011000	00110100001000
267	00100111011111	01000000000101
268	00100111100011	00101011001100
269	00100111100100	01000001010100
270	00100111100110	00101000000111
271	00100111101000	00101000000111
---	-----	-----

Figura 29. Resultado vector de movimiento encontrados

Los datos obtenidos contienen la información del vector de movimiento, cada fila contiene información de la ubicación del macrobloque de referencia y su nueva ubicación de ese mismo macrobloque en el frame actual, en la Figura 30 se tomó un vector al azar y se realizó la conversión a decimal usando el editor Notepad¹, el resultado fue que el macrobloque 2447 del frame de referencia presento un cambio.

247	00100110000100	00101100001101
248	00100110001000	00100110001001
249	00100110001001	00100110010001
250	00100110001111	00110010110110
251	00100110010000	00101010111001
252	00100110010011	00100110011011
253	00100110010110	00111010010110
254	00100110010111	00100110111011
255	00100110011000	00100110111111
256	00100110011001	00101001100100
257	00100110110011	01000001011111
258	00100110110100	00111001110110
259	00100110110110	00100110111001
260	00100110111000	01001011000000
261	00100110111010	01001011000000
262	00100111001011	00101001110111
263	00100111001100	00101000100010
264	00100111001110	00100111010110
265	00100111010100	00101010101100

Conversion

ASCII:

Decimal:

Hexadecimal:

Binary:

Octadecimal:

Figura 30. Ubicación Macrobloque Referencia

¹ Software empleado para edición de código multi-lenguaje

Este macrobloque posee el valor 000090 como se evidencia en la Figura 31.

2445	000000
2446	00001B
2447	000090
2448	000010
2449	000001

Figura 31. Valor del Macrobloque de referencia de ejemplo

A continuación, se debe verificar si el valor en el macrobloque del frame actual coincide por el calculado en el algoritmo. Ver Figura 32.

247	00100110000100	00101100001101
248	00100110001000	00100110001001
249	00100110001001	00100110010001
250	00100110001111	00110010110110
251	00100110010000	00101010111001
252	00100110010011	00100110011011
253	00100110010110	00111010010110
254	00100110010111	00100110111011
255	00100110011000	00100110111111
256	00100110011001	00101001100100
257	00100110110011	01000001011111
258	00100110110100	00111001110110
259	00100110110110	00100110111001
260	00100110111000	01001011000000
261	00100110111010	01001011000000
262	00100111001011	00101001110111
263	00100111001100	00101000100010
264	00100111001110	00100111010110
265	00100111010100	00101010101100

Conversion

ASCII: Copy Insert

Decimal: Copy Insert

Hexadecimal: Copy Insert

Binary: Copy Insert

Octadecimal: Copy Insert

Figura 32. Ubicación Macrobloque Actual

Realizando el mismo procedimiento, se ubica el macrobloque en el frame actual y se evidencia que corresponde al mismo valor que se encuentra en el frame de referencia y ademas fue encontrado en el macrobloque descrito por el resultado del algoritmo. Ver Figura 33.

3252	000000
3253	000000
3254	000090
3255	000031
3256	00002E

Figura 33. Valor del Macrobloque actual de ejemplo

Estos vectores de movimientos obtenidos en el ejemplo descrito en la presente sección fueron graficados empleando la herramienta Matlab. Ver Figura 34.

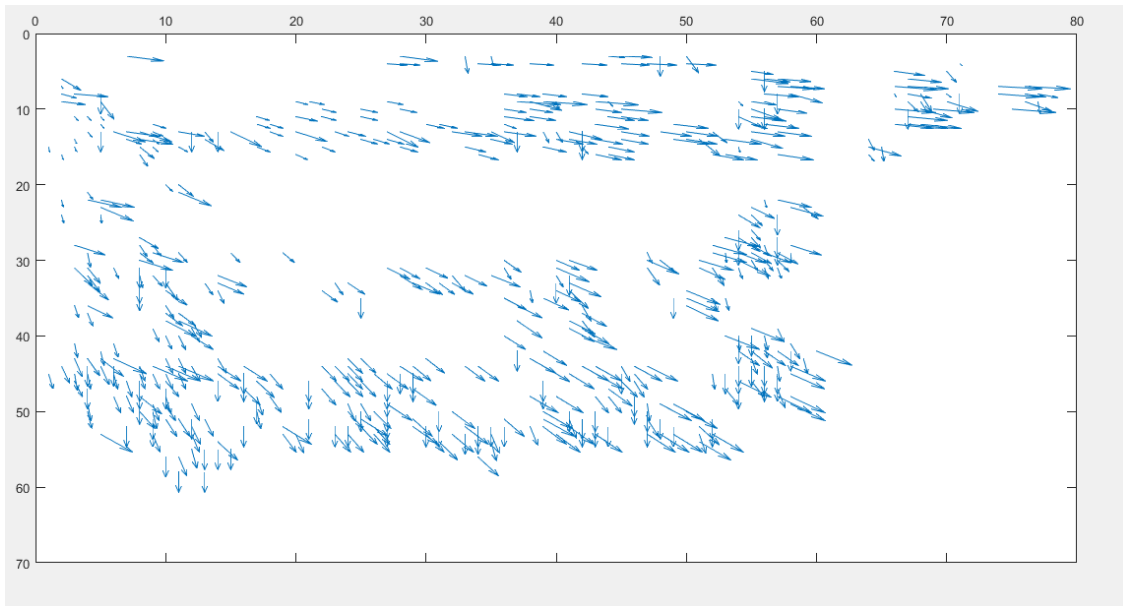


Figura 34. Grafica Vector de Movimiento Matlab

Para el presente caso de simulación entre las dos imágenes propuestas, se obtiene un tiempo total de simulación de 18 milisegundos, esto quiere decir que fue capaz el algoritmo de encontrar 576 vectores de movimiento, lo que implico hacer una búsqueda exhaustiva macrobloque por macrobloque 4800 veces. A continuación, se presenta una tabla que resumen lo descrito en el presente párrafo.

Tabla 1. *Resultados Simulación*

Reloj	50 MHz
Resolución Frame	640x480
Tamaño del Macrobloque	8x8
Tiempo de Calculo	18 milisegundos
Tiempo por frame	540 milisegundos
Tiempo por macrobloque	3.75 microsegundos
Vectores encontrados	576

La Tabla 1. *Resultados Simulación* resume los resultados de la simulación, empleando el algoritmo una frecuencia de 50MHz, frames basados en el formato VGA 640 x 480 resolución que fue convertido a macrobloques de 8x8 generando así una resolución de 80 x 60 que corresponden a 4800 macrobloques. Con las características anteriormente descritas se obtuvo un tiempo de procesamiento total de 18 milisegundos. Tomando como referente el estándar de 30 fps² y los resultados obtenidos en la simulación, una regla de tres determinaría que se necesitan aproximadamente 540 milisegundos un poco más de medio segundo para que el algoritmo pueda procesar 30 fps. Es importante resaltar que la ventana de búsqueda del algoritmo es el frame completo por lo tanto estos resultados están en base al peor de los casos respecto a ventana de búsqueda.

² Frame por Segundo o fotogramas por segundo

4.2 Versiones del Algoritmo

Durante el desarrollo del algoritmo de estimación de movimiento basado en Block Matching, se generaron tres versiones principales del mismo, la primera versión presenta el primer acercamiento al cálculo de los vectores de movimiento, proceso que se cumplía a cabalidad ya que cuando el algoritmo de estimación de movimiento encontraba un macrobloque diferente entre el frame actual y el frame de referencia inmediatamente empezaba su búsqueda hasta encontrar el macrobloque y guardar su posición. Esta primera versión presentaba un problema en la identificación de los macrobloques después de encontrar el primero, ya que, el contador de macrobloque actual el cual tiene como objetivo barrer el frame no restablecía su posición y siempre marcaba una posición adelante antes de empezar a barrer el frame, por lo tanto, siempre el algoritmo empezaba a guardar vectores de movimiento cada posición, aunque los macrobloques fueran iguales, esto causaba que en la misma simulación presentada en el numeral 4 generará 4350 vectores de movimiento, además de tiempos de procesamiento superiores a los 30 ms. Un ejemplo de esto se puede observar en la siguiente Figura 35.

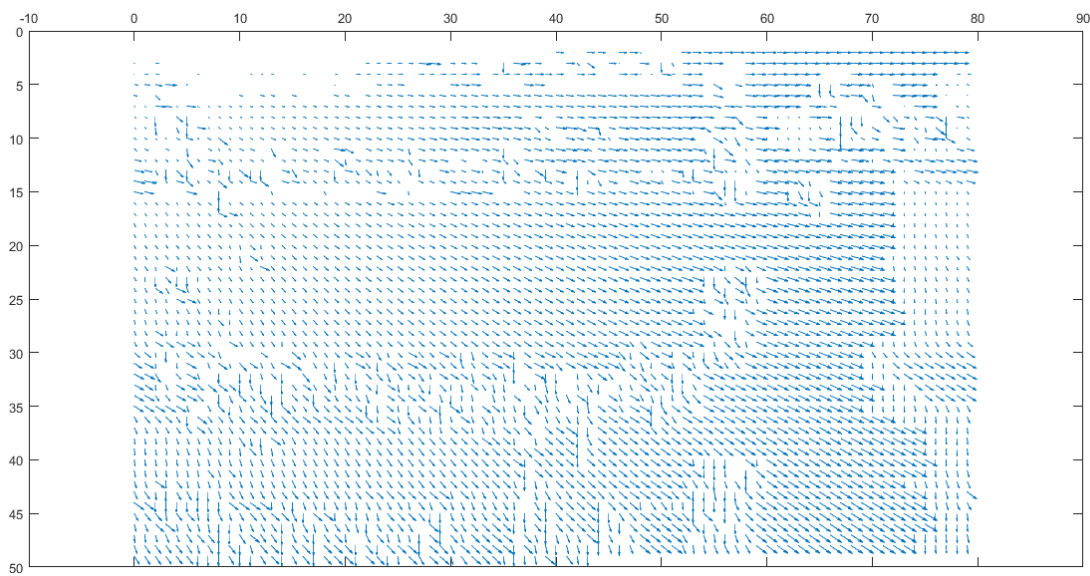


Figura 35. Resultado Versión 1.0 del Algoritmo

Este error se corrigió obligando al contador actual a adquirir la misma posición que el contador de referencia después de almacenar un vector. Realizar esta corrección genero la versión 1.1 del algoritmo donde se obtuvieron resultados óptimos en el algoritmo (para revisar en detalle esta versión leer subtítulo 4.1 Simulación).

A partir de la versión 1.1 se desarrolla una nueva versión del algoritmo que implica mejorar los tiempos de procesamiento del algoritmo, por lo tanto, se decide implementar 3 ventanas de búsqueda que serán procesadas al mismo tiempo, esta ventana de búsqueda hace referencia a la división horizontal de los frames en 3 partes iguales (ver Figura 36), donde se aplicara el algoritmo paralelamente. Los resultados obtenidos con la versión 1.2 se pueden observar en la *Tabla 2*.

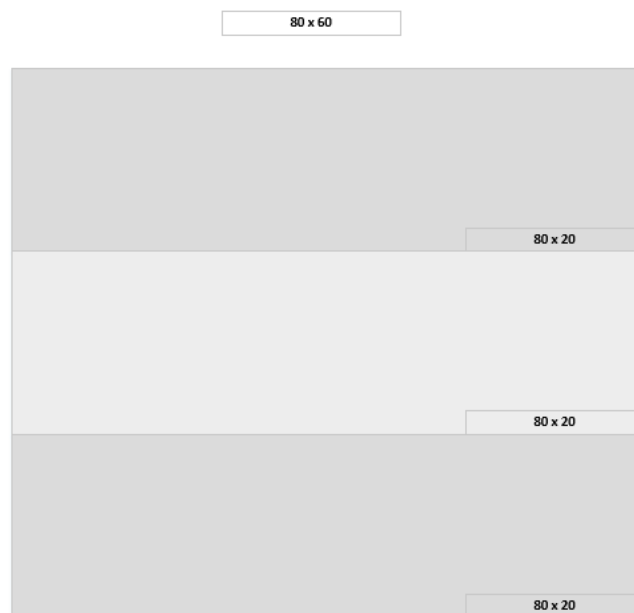


Figura 36. Ventanas para Algoritmo Versión 1.2

Realizando la misma simulación empleando la versión 1.2 del algoritmo se obtuvo que para la ventana superior fueron encontrados 185 vectores de movimiento, para la ventana del medio 124

vectores y para la ventana inferior 170 vectores de movimiento dando como resultado un total de 479 vectores de movimiento en comparación a los 576 obtenidos con el algoritmo Versión 1.1 (Ver Apéndice A.). La Figura 37 presenta la simulación del algoritmo Versión 1.2 allí se evidencia la diferencia en dividir en tres diferentes zonas los frames.

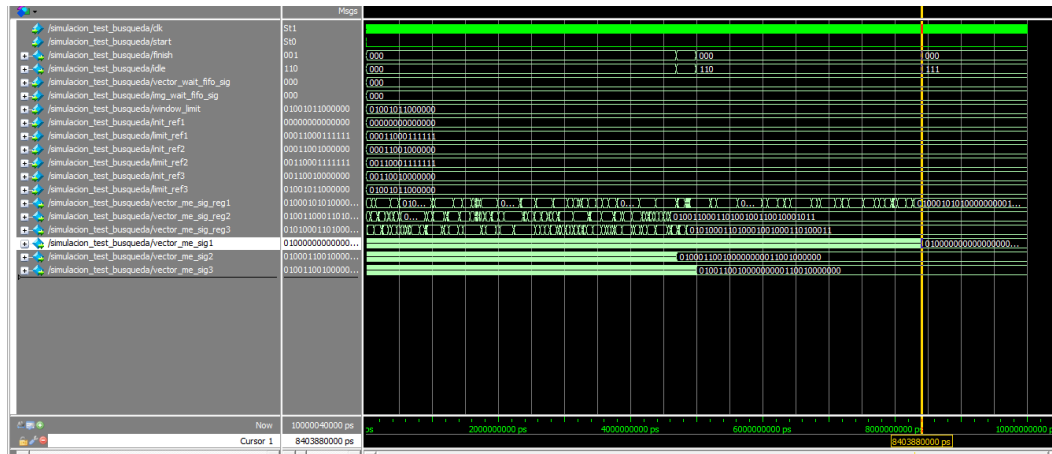


Figura 37. Simulación Algoritmo Versión 1.2

Tabla 2. Resultados versión 1.2

Reloj	50 MHz
Resolución Frame	640x480
Tamaño del Macrobloque	8x8
Tiempo procesamiento ventana 1	8.4 milisegundos
Tiempo procesamiento ventana 2	4.7 milisegundos
Tiempo procesamiento ventana 3	4.9 milisegundos
Tiempo por macrobloque	3.7 microsegundos
(Promedio)	

Empleando a versión 1.2 mejora significativamente el tiempo que el algoritmo tarda en encontrar los vectores de movimiento ya que se pasa de realizar la misma cantidad de cálculos de 18 milisegundos a 8.4 milisegundos, pero es importante resaltar que se sacrifica en precisión debido a que los límites de las ventanas de búsqueda evitan que se busquen los macrobloques en las demás regiones del frame. Otro factor importante es que al paralelizar se van a necesitar mayor cantidad de recursos por lo tanto estas limitantes físicas serán siempre un impedimento al emplear técnicas de paralización.

5. Conclusiones

La estimación de movimiento es uno de los procesos más complejos e importantes en el estándar de codificación h.264, esto se debe a que la compresión será más eficiente evitando que se transmita información redundante, por ende, desarrollar un algoritmo de estimación de movimiento que pueda ser fácilmente implementado en hardware contribuirá en cualquier desarrollo que implique transmisión de video a muy bajas tasas, especialmente en servicios de telemedicina, donde implementar este tipo de soluciones siempre será una ventaja para las limitadas condiciones de las regiones.

Es importante argumentar que la estrategia de paralelizar es una excelente herramienta para mejorar aspectos como el tiempo necesario para procesar frames y que la técnica de Block Matching basada en búsqueda exhaustiva contribuye a que se pueda implementar fácilmente por la lógica del algoritmo. Pero esto no implica que se pierda en otros aspectos relevantes como la precisión y la cantidad de recursos necesarios para realizarlo, siempre se tendrá limitantes en recursos físicos donde el verdadero objetivo del algoritmo es encontrar el mejor equilibrio.

5.1 Trabajo Futuro

Aunque se obtuvieron resultados interesantes en el desarrollo del presente trabajo es importante resaltar que realizar una discusión entre autores referentes implicaría determinar las características del algoritmo de estimación propuesto por ellos y someter a pruebas bajo similares condiciones el algoritmo de estimación de movimiento presentado en este documento, por lo tanto, se realizara una publicación con los resultados de la comparación.

Se está desarrollando un IP CORE con el proceso de estimación de movimiento basado en Block Matching. Este desarrollo será publicado en la página www.fpgalover.com y se evaluará su liberación como contribución a la codificación de video.

Apéndices

Apéndice A: Algoritmo de estimación de movimiento diseñado.

Apéndice B: Controlador del algoritmo de estimación de movimiento.

Apéndice C: Resultado de estimación de movimiento

Referencias Bibliográficas

- [1] A. M. Campos, “Estudio de Arquitecturas VLSI de la Etapa de Predicción de la Compensación de Movimiento, para Compresión de Imágenes y Video con Algoritmos Full-Search. Aplicación al Estándar H. 264/AVC,,” 2008.
- [2] T. C. Chen, Y. W. Huang, C. Y. Tsai, C. T. Huang, and L. G. Chen, “Single reference frame multiple current macroblocks scheme for multi-frame motion estimation in H.264/AVC,” *Proc. - IEEE Int. Symp. Circuits Syst.*, vol. 17, no. 2, pp. 1790–1793, 2005.
- [3] B. P. Statistik, *Image and Video Compression for Multimedia Engineering*, vol. XXXIII, no. 2. 2014.
- [4] R. Alvarez Bosa, “Análisis y evaluación del servicio de video streaming adaptativo bajo el estándar DASH,” Universidad Santo Toams, 2013.
- [5] IEEE, “The JPEG still picture compression standard - Consumer Electronics, IEEE Transactions on,” pp. 1–17, 2004.
- [6] C. Lui, “A Study of the JPEG-2000 Image Compression Standard,” no. May, pp. 9–14, 2001.
- [7] T. Sikora, “MPEG-1 and MPEG-2 Digital Video Coding Standards,” *Digit. Consum. Electron. Handb.*, pp. 1–43.
- [8] The MPEG Industry Forum, “Understanding MPEG-4 : Technologies, Advantage, and Markets,” *Mepg-4*, no. An MPEGIF White Paper, 2005.
- [9] D. R. Tobergte and S. Curtis, “Codecs for videoconferencing using primary digital group transmission,” *J. Chem. Inf. Model.*, vol. 53, no. 9, pp. 1689–1699, 2013.
- [10] ITU, “ITU-T H.261 : Video codec for audiovisual services at p x 64 kbit/s,” *ITU-T SG 16*, vol. 261, 1993.
- [11] S. Sector and O. F. Itu, *ITU-T ITU-T H.263 Video coding for low bit rate communication*,

vol. 263. geneva, 2005.

- [12] A. Gallegos Hernandez, “Televisión sobre IP (IPTV) y Redes de Próxima Generación (NGN),” Universidad Nacional Autónoma de México Facultad de Ingeniería.
- [13] S. A. Khayam, “The Discrete Cosine Transform (DCT): Theory and Application,” 2003.
- [14] A. B. (Nasa A. R. C. Watson, “Image Compression Using the Discrete Cosine Transform,” *Math. J.*, vol. 4, no. 1, pp. 81–88, 1994.
- [15] B. Kaderkulof-Rengifo, J. Velasco-Medina, and M. Vera-Lizcano, “Diseño de Funciones DSP para Compresión de Video Usando FPGAs.”
- [16] C. Wen-Hsiung, C. Smith, and S. Fralick, “A Fast Computational Algorithm for the Discrete Cosine Transform,” *Commun. IEEE Trans.*, vol. 25, no. 9, pp. 1004–1009, 1977.
- [17] V. Britanak, P. C. Yip, and K. R. Rao, *Discrete Cosine and Sine Transforms*, 1st Editio. 23 Oct 2006, 2006.
- [18] M. Grau Garcia, “Estudio de viabilidad técnica e implementación de un sistema de transmisión de video adaptativo H264 / SVC sobre TCP,” Universidad politecnica de Catalunya, 2014.
- [19] J. G. Zambrano Nila, “Cuantificacion vectorial para imagenes con base en los algoritmos LBG y Lamda,” Universidad Tecnologica de la Mixteca, 2012.
- [20] A. Mora Campos and D. D. Francisco José Ballester Merelo Marcos Antonio Martínez Peiró Valencia, “Tesis Doctoral,” 2008.
- [21] G. J. Sullivan, “Rate-Distortion Optimization for Video Compression,” p. 17, 1998.
- [22] R. E. C. Porto and L. V. Agostini, “Project space exploration on the 2-D DCT architecture of a JPEG compressor directed to FPGA implementation,” in *Proceedings -Design, Automation and Test in Europe, DATE*, 2004, vol. 3, pp. 224–229.
- [23] D. W. Trainor, J. P. Heron, and R. F. Woods, “Implementation of the 2D DCT using a Xilinx XC6264 FPGA,” *Proc IEEE Work. Signal Process. Syst. SiPS*, pp. 541–550, 1997.
- [24] G. G. Langdon, “An Introduction to Arithmetic Coding,” *IBM J. Res. Dev.*, vol. 28, no. 2, pp. 135–149, 1984.
- [25] D. Marpe, G. Blattermann, G. Heising, and T. Wiegand, “Video compression using context-based adaptive arithmetic coding,” *Proc. 2001 Int. Conf. Image Process. (Cat. No.01CH37205)*, vol. 3, pp. 558–561, 2001.
- [26] B. Ramirez Magaña, “Análisis de algoritmos utilizados en el metodo de comparacion de bloques para la estimacion de movimiento en video,” Instituto Politecnico Nacional, 2009.

- [27] E. Chan and S. Panchanathan, "Motion estimation architecture for video compression," *IEEE Trans. Consum. Electron.*, vol. 39, no. 3, pp. 292–297, 1993.
- [28] "BlockMatchingAlgorithmsForMotionEstimation."
- [29] I. L. Espejo, "Estimación de Movimiento," no. Marzo, 2012.
- [30] P. C. Benito, "Integración de información de movimiento en la segmentación de secuencias de vídeo basada en el modelado de fondo," Universidad Autonoma de Madrid, 2010.