SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

Tecnológico Nacional de México

Instituto Tecnológico de Apizaco

División de Estudios de Posgrado e Investigación

Aceleración de Algoritmo de Procesamiento de Movimiento en Vídeo basado en la Fase

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

Maestro en Sistemas Computacionales

PRESENTA:

Lauro Reyes Cocoletzi

Director:
Dr. José Federico Ramírez Cruz

Codirector: Dr. José Crispin Hernández Hernández

Apizaco, Tlaxcala, Agosto 2017

Esta tesis esta dedicada a todos los compañeros que conformamos la generación 2015-2017, sin ellos compartiendo experiencia y habilidades no habriamos podido llevar a buen termino este trabajo.

A los catedraticos que conforman al posgrado de Sistemas Computacionales por compartir conocimientos y experiencia.

A mi familia por el soporte durante estos años mi madre Carmen y mis hermanos Ana Laura, Lilia, Mario y Pilar. Una mención especial a mis amigos Lupita Maya, Laura Reyes, Angeles y Lorena con quienes comparti aula y diversión...GRACIAS TOTALES!!!

Abstract

The execution time of the algorithms is a common problem in any computational application and it is necessary to solve these problems through the use of computational tools to improve the performance of the processing speed.

This work aims to use computational acceleration tools to reduce the execution time of a video processing algorithm. It is proposed to accelerate the Phase-Based Video Motion Processing algorithm by means of the OpenCV computer vision library and its integration with parallel processing through a graphics card (Nvidia) used as a co-processor.

The Phase-Based Video Motion Processing algorithm consists of magnifying small movements difficult to see with the naked eye captured on video.

This algorithm allows to observe slightly movements in a scene due to the variation of the space-time magnitude of the pixels between a frame and the frame that precedes it, processing a considerable amount of convolution masks and digital filters to each individual frame of the video, which implies a high computational cost.

A time analysis is done that takes the algorithm to process the convolution and filtering operations to determine the stages in which the acceleration of the latter must be implemented.

We used OpenCV library functions to efficiently process data and parallel programming in CUDA to simultaneously handle data blocks and functions.The filtering operations were reduced in an 1.43-times ratio compared against the non- accelerated algorithm through the implementation of the accelerated functions of the CUDA- OpenCV tools.

The motion magnification stage was reduced in 2.98-times through the processing of the matrices in parallel through the development of operations on the graphics card.

The functional evaluation of the accelerated algorithm is performed, the evaluation reveals a reduction of a lesser 4-times in the implementation of the video processed in the accelerated algorithm against the non-accelerated algorithm. The overall results show that the integration of OpenCV and parallel programming with CUDA serve to take advantage of the capabilities of Nvidia graphics cards as co-processors and greatly reduce execution times in algorithms programmed sequentially.

Resumen

El tiempo de ejecución de los algoritmos es un problema común en cualquier aplicación computacional y es necesario resolver esta problemática a través del uso de herramientas de cómputo para mejorar el rendimiento de la velocidad de procesamiento.

Este trabajo tiene como objetivo utilizar herramientas computacionales de aceleración para reducir el tiempo de ejecución de un algoritmo de procesamiento de vídeo. Se plantea acelerar el algoritmo de Procesamiento de Movimiento en Vídeo basado en la Fase por medio de la biblioteca de visión por computadora OpenCV y su integración con procesamiento paralelo a través de una tarjeta gráfica (Nvidia) utilizada como co-procesador.

El algoritmo de Procesamiento de Movimiento en Video Basado en la Fase consiste en magnificar movimientos pequeños difíciles de observar a simple vista capturados en video, este algoritmo permite observar movimientos pequeños en una escena debido a la variación en magnitud espacio-temporal de los pixeles entre un cuadro y el cuadro que lo precede, procesando una cantidad considerable de máscaras de convolución y filtros digitales a cada cuadro individual del video, lo que implica un alto costo computacional.

Se realiza un análisis del tiempo que le toma al algoritmo procesar las operaciones de convolución y filtrado para determinar las etapas de mayor costo computacional en donde se debe implementar la aceleración de este.

Se uso la biblioteca de OpenCV para procesar los datos de forma eficiente y la programación paralela en CUDA para manejar bloques de datos y funciones en forma simultánea. Las operaciones de filtrado se reducen en una proporción de 1.43 veces en comparación contra el algoritmo no acelerado a través del uso de las funciones aceleradas de las herramientas CUDA-OpenCV.

La etapa de magnificación de movimiento se reduce en 2.98 veces a través del procesamiento de las matrices en forma paralela por medio del desarrollo de operaciones en la tarjeta gráfica. Se realiza la evaluación funcional del algoritmo acelerado, los resultados arrojan una reducción de un tiempo 4 veces menor en la ejecución del video procesado en el algoritmo acelerado contra el algoritmo no acelerado. Finalmente los resultados globales obtenidos muestraron que la integración de OpenCV y la programación en paralelo de CUDA sirve para aprovechar las capacidades de las tarjetas gráficas de Nvidia como co-procesadores y reducir considerablemente los tiempos de ejecución en los algoritmos programados secuencialmente.

Índice general

Índice de figuras

Índice de tablas

Lista de algoritmos

Lista de Ecuaciones

Capítulo 1

Introducción

En el campo del procesamiento computacional, es necesario realizar un sin número de aplicaciones, manejo de grandes cantidades de datos, ejecución de operaciones, procesamiento de señales, búsqueda de información entre muchas otras. Estas necesidades requieren ser realizadas de forma precisa y en tiempo mínimo [7].

Actualmente las necesidades de manejo de cantidades considerables de datos y los requerimientos en reducción en el tiempo de ejecución para obtener la información procesada han hecho que se tenga que mejorar la implementación de los algoritmos.

En años recientes se logró un avance significativo para resolver las necesidades de aceleración computacional al introducir el uso de tarjetas gráficas como procesadores auxiliares (GPU) para expandir el manejo operaciones y datos. Las tarjetas gráficas inicialmente se implementaron para procesar gráficos para aligerar la carga de trabajo del procesador central, sin embargo, con el rendimiento que han demostrado, se ha optado para que sean utilizadas para el funcionamiento como un multiprocesador en las tareas de cómputo[12].

Las tarjetas gráficas combinan los beneficios del hardware y el software para llevar a cabo la aceleración de los algoritmos.

En este trabajo de tesis, con el objetivo de acelerar el procesamiento de las operaciones del algoritmo Procesamiento de Movimiento en Vídeo basado en la Fase, se propone el uso de la tarjeta gráfica (Nvidia) junto con la funciones de la biblioteca de OpenCV para reducir el tiempo computacional.

El análisis del algoritmo y la propuesta de aceleración en etapas específicas muestran un rendimiento superior en cuanto al desempeño, uso de los recursos del hardware y en los tiempos de procesamiento con respecto al algoritmo no acelerado.

1.1 Descripción del problema

El algoritmo Procesamiento de Movimiento en Vídeo basado en la Fase (PMVF) consiste en la amplificación de movimientos tenues, en vídeo, difíciles de observar a simple vista.

El algoritmo PMVF procesa la información contenida en el vídeo a través del tratamiento temporal y en frecuencia (principios de frecuencia de Fourier) de los píxeles que conforman a las imágenes para magnificar movimiento sin amplificar ruido ambiental [19].

En este algoritmo existen etapas que requieren que las características locales de información sean procesadas en el menor tiempo posible usando vídeos de resolución VGA (640 x 480 píxeles) o alta definición (1280 x 720 píxeles).

En el algoritmo PMVF la etapa de filtrado identifica la variación de frecuencia de fase, en función de la transformada discreta de Fourier, de los datos de la imagen compuesta por píxeles, los cambios de frecuencia (al usar la pirámide orientable) corresponden a los movimientos locales en sub-bandas espaciales de la imágen.

La etapa de magnificación corresponde al incremento de variación de frecuencia por un factor de multiplicación para amplificar cambios sutiles.

Las desventajas de este algoritmo radican en cuanto al costo computacional, resultado de procesos complejos repetitivos para obtener movimientos pequeños magnificados en vídeo, siendo una de las más costosas la etapa de filtrado.

Los algoritmos que se procesan en forma secuencial les toma un gran numero de ciclos de instrucción para arojar el resultado a la salida, lo que implica un problema en el momento de requerir la información de forma rápida y libre de interferencias o alteraciones, para este caso, obtener un vídeo libre de ruido y mostrar movimientos tenues magnificados que no se podian observar.

Por ejemplo, para procesar un vídeo de resolución 860*960 píxeles a una tasa de 24 cuadros por segundo con una duración de 19 segundos en una computadora con un procesador core i7 con 8 gb de memoria ram le toma al algoritmo PMVF obtener el vídeo procesado en un tiempo de 700 segundos. Reducir a una tasa de 10 veces implicaría un tiempo de procesamiento de 70 segundos, que representaría un cambio significativo en la velocidad de trabajo del algoritmo PMVF.

Obtener una tasa superior a 10 veces sería lo deseable para acercarse al procesamiento en tiempo real y poder dar un uso práctico en el que se desee aplicar el algoritmo de magnificación de movimiento.

1.2 Pregunta de investigación

¿En qué magnitud puede la librería OpenCV-Cuda reducir el costo computacional del algoritmo de Procesamiento de Movimiento en Video basado en la fase, se podrá acercar al procesamiento en tiempo real?

1.3 Justificación

En la actualidad, pocas herramientas logran mejorar la tasa de procesamiento y rendimiento de los tiempos de ejecución computacional que puedan ser aplicados sobre el algoritmo PMVF. Así que, es necesario implementar dicho algoritmo de tal forma

que logré trabajar con mejor rendimiento en la velocidad de procesamiento, para que se exploté este algoritmo en las técnicas de visión por computadora tanto como en aplicaciones domésticas, industriales o de seguridad.

Aunado a esto, es muy importante el continuo desarrollo de la aplicación de algoritmos que aprovechen las capacidades y herramientas computacionales actuales (tarjeta gráfica como coprocesador compartido) para reducir el tiempo de respuesta de salida y trabajar con los algoritmos en tiempo real de procesamiento o cercano a ello.

1.4 Objetivos

Proyectando posibles aplicaciones del algoritmo de PMVF en el campo de visión por computadora donde se necesita procesamiento en tiempo real se debe mejorar la limitante actual del alto tiempo de cómputo. Por lo tanto, dada la necesidad de tener variantes o implementaciones más rápidas de este algoritmo, el trabajo se centra en desarrollar un algoritmo acelerado usando hardware (GPU-Nvidia) y software (OpenCV) que minimice el costo computacional de la magnificación de movimiento procesado en vídeo.

1.4.1 Objetivo general

Teniendo en cuenta las ventajas de las tarjetas gráficas GPU en la aceleración de tareas de cómputo, el objetivo de este trabajo es utilizar una integración eficiente para el procesamiento del algoritmo PMVF, que permita reducir en al menos el 25 % de la magnitud los tiempos de procesamiento respecto a la implementación del algoritmo original.

1.4.2 Objetivos específicos

Como objetivos específicos de este trabajo se encuentran los siguientes:

1. Identificar procesos de mayor potencial para la paralelización en la etapa de filtrado y amplificación, que lleven a una aceleración considerable de esta etapa usando la tarjeta gráfica como coprocesador.

2. Usar la biblioteca OpenCV para el manejo eficiente del procesamiento de los píxeles en cada cuadro que conforman al vídeo.

3. Obtener un uso eficiente de dicha biblioteca integrada con el procesamiento acelerado en tarjetas gráficas de Nvidia.

1.5 Contribuciones

En este trabajo se propone usar la biblioteca OpenCV junto con la programación en CUDA en el algoritmo PMVF. Se replantean etapas del algoritmo en función de explotar el rendimiento de las tarjetas gráficas y de esta manera sacar el mayor provecho de una implementación paralela para acercarse al procesamiento en tiempo real.

La principal contribución de la aceleración del algoritmo PMVF, radica en que a medida que decrementa el tiempo de operación el algoritmo se podrá utilizar en aplicaciones del mundo real. Este elemento resulta de gran importancia ya que la tendencia en el rendimiento computacional es reducir el tiempo de ejecución sin descuidar la cantidad de información a procesar. Por lo tanto, para algoritmos que requieran cantidades considerables de funciones para procesar vídeo, el uso de las herramientas aquí aprovechadas puede alentar la aceleración de otras necesidades de procesamiento en diversos algoritmos de Visión por computadora.

Se llegó a la contribución antes mencionada por medio del análisis, pruebas y experimentos realizados, se mostró cuantitativamente la mejora con el paralelizado de procesamientos en las etapas de filtrado y magnificación, obteniéndose reducciones en el ciclo de trabajo del 20 % de la velocidad y con tendencia al decremento si se utiliza hardware más potente. Además, se reportan pequeñas diferencias en la generación de la salida por etapas, por parte de la implementación del algoritmo con la biblioteca OpenCV en comparación con el algoritmo no acelerado y resultados de tiempo menores en el procesamiento global. También se presentan variaciones en etapas del algoritmo donde se incrementó el tiempo de operación debido a la reformulación de operaciones de filtrado frecuencial no disponibles en la biblioteca de OpenCV o CUDA, estos incrementos no son considerables pero si merecen ser mencionados.

Finalmente, el trabajo propuesto está al nivel de los indicadores de tiempo y eficiencia en el uso de procesamiento paralelo establecido en los trabajos relacionados. El trabajo realizado en este proyecto logra procesar el algoritmo PMVF a una aceleración aproximada de 4 veces con respecto a la implementación del algoritmo no acelerado.

1.6 Organización del trabajo de tesis

El proyecto de investigación se divide en 5 capítulos para su lectura, a continuación se resume cada capítulo que lo conforma.

- Capítulo 1. Se introduce al lector en la parte inicial del trabajo, describiendo el tema de investigación, la problemática del tema de investigación, los objetivos a los que se pretende llegar al concluir el trabajo de investigación, las contribuciones realizadas y finalmente el estado del arte para conocer propuestas de diferentes trabajos de investigación que se relacionan al abordar la problemática de interés.

- Capítulo 2. En este capítulo se describe el algoritmo de PMVF sus características y aplicaciones. Posteriormente se desarrolla el análisis de las etapas que comprenden

a este algoritmo, para identificar aquellas etapas viables de acelerar, además del análisis del costo computacional también se realiza el análisis minucioso de los operadores y funciones que comprenden el diseño del algoritmo PMVBF para poder replicarlo de forma acelerada.

- Capítulo 3. El capítulo refiere el uso de la librería OpenCV-Cuda para acelerar el algoritmo PMVF, describe la integración de las operaciones para reducir el costo computacional y las distintas tareas que procesan el vídeo para alcanzar la salida deseada.
- Capítulo 4. Se presenta los resultados obtenidos, la comparativa de los tiempos de ejecución entre el rendimiento serial contra el rendimiento acelerado. Se presenta un análisis por etapas (las de mayor interés) y de igual forma el procesamiento global. Finalmente se plantea la discusión de los alcances del trabajo desarrollado en este proyecto de investigación.
- Capítulo 5. Plantea las conclusiones a las que se llega finiquitado el trabajo de investigación, se plantean y visualizan mejoras que se pueden realizar en el trabajo realizado.

1.7 Estado del arte

A continuación se mencionan un conjunto de algoritmos que utilizan tarjetas gráficas como coprocesadores, estos algoritmos trabajan con vídeo usando distintas técnicas de procesamiento digital de imágenes y de visión por computadora. La mención de estos algoritmos es importante pues construye una base para el desarrollo de este trabajo en el uso de cómputo paralelo para acelerar algoritmos de procesamiento de vídeo.

1.7.1 Aceleración del algoritmo Retinex mejorado

Recientemente la arquitectura multinucleo de las tarjetas gráficas ha llegado a ser el centro de muchas investigaciones en cómputo paralelo de alto desempeño por lo que hay gran variedad de implementaciones que han usado esta tecnología (GPU/CUDA) para acelerar tareas computacionales en procesamiento de imágenes y visión por computadora. Por ejemplo, el algoritmo Canny Paralelo en CUDA demostró un rendimiento de 3.8 veces la velocidad de procesamiento sobre imágenes con una resolución de 3936 x 3936 píxeles comparado contra una versión optimizada del algoritmo Canny usando solo OpenCV [4].

Otro caso mencionado es el algoritmo GPU-Retinex mejorado en el que Wang y Huang proponen un método basado en GPU para el algoritmo Retinex donde la velocidad es acelerada 43 veces [20]. El método Retinex es un método eficaz para remover interferencias producidas por la luz ambiental y usada como etapa de pre-procesamiento en muchos algoritmos de visión por computadora. El algoritmo GPU-Retinex utiliza el modelo de programación heterogénea provisto por CUDA, donde los segmentos de

código serial son ejecutados por el equipo (CPU) y solo los segmentos de código paralelo son ejecutados en el dispositivo (GPU). El equipo carga la imagen original y realiza la transferencia de memoria de la imagen del equipo al dispositivo. Las cinco operaciones incluidas el filtro Gaussiano, el procesamiento logarítmico, la reducción, la normalización y extensión del histograma son ejecutadas de forma paralela en la GPU, en la figura 1.1 [20] se muestra el panorama general de las operaciones del algoritmo Retinex.

El paso final es la transferencia de los resultados del dispositivo al equipo.

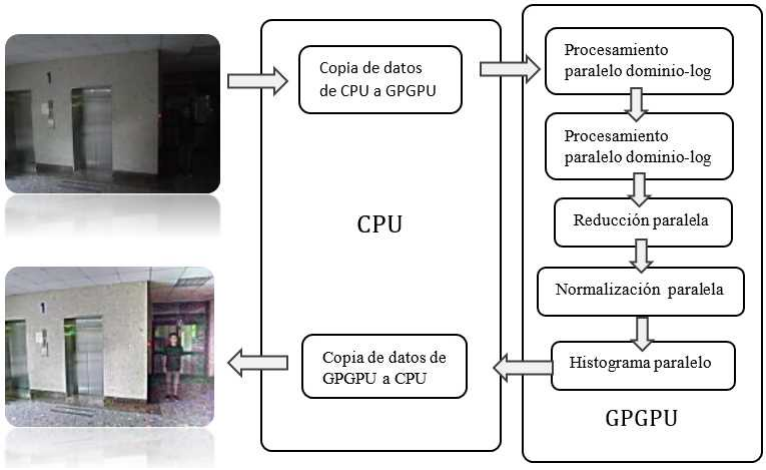


Figura 1.1. Panorama general del algoritmo GPU-Retinex.

Alrededor de 100 imágenes fueron probadas en los experimentos finales para verificar la optimización del algoritmo GPU-Retinex contra el algoritmo Retinex. Los resultados a la salida y el tiempo de procesamiento arrojaron una mejor resolución de la imagen en un menor tiempo. La figura 1.2 muestra el filtro de convolución aplicado en el algoritmo GPU-Retinex [20].

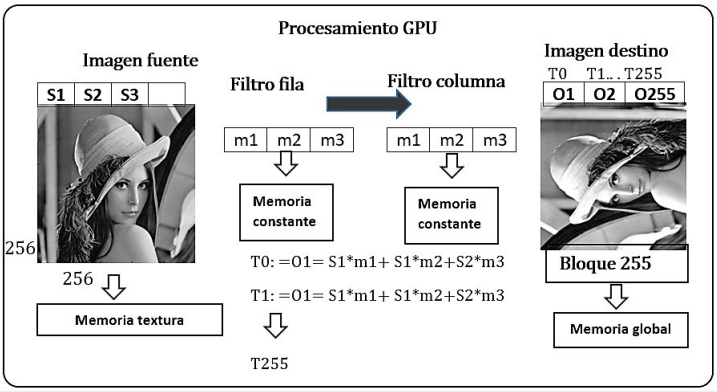


Figura 1.2. Filtro para difuminar, aplicado en las columnas y filas de la imagen Lena.

La figura 1.3 muestra los resultados de los experimentos y la comparación realizada.

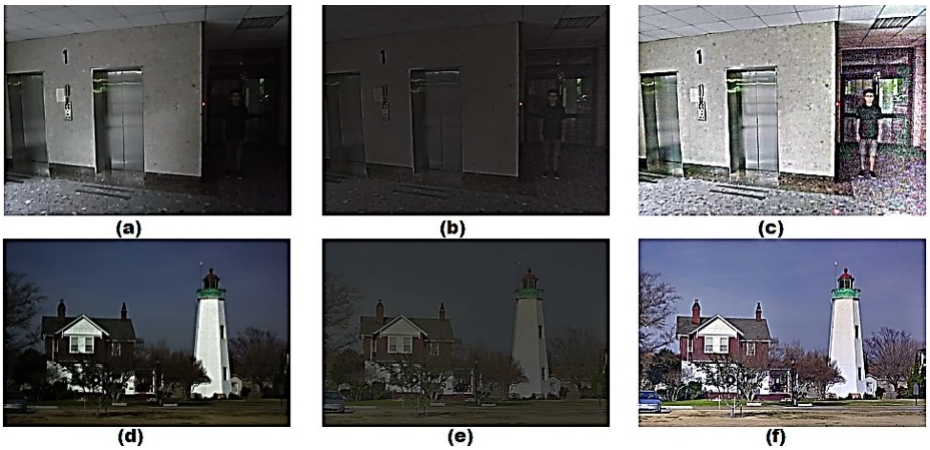


Figura 1.3. (a) (d) Imagen original, (b) (e) imagen de salida procesada en Retinex serial, (c) (f) imagen de salida procesada por GPU-Retinex.

Lo que muestra este trabajo es una implementación funcional de la integración y aceleración de los algoritmos de procesamiento de imágenes y vídeo utilizando tarjetas gráficas Nvidia para mejorar el rendimiento del costo computacional. También se utiliza de mejor forma la distribución de la memoria del dispositivo para realizar los cálculos y operaciones sobre los píxeles. Esta información es útil para tomarla en consideración al desarrollar este trabajo de tesis.

1.7.2 Implememtación eficiente del algoritmo de detección de cambios en vídeo

La demanda de aplicaciones de vídeo digital de alta calidad ha llamado la atención de la industria y de los académicos para el desarrollo de avances técnicos en la codificación de vídeo. El algoritmo de detección de cambios en vídeo (DCV) presenta una solución acelerada desarrollada en una implementación eficiente de un nuevo algoritmo, basado en el acoplamiento de cuadros (frames) de baja dimensionalidad y el algoritmo de registro de imágenes (ECC). El algoritmo DCV alcanza un rendimiento secuencial de un máximo de 10.47 veces en tiempo al procesar un vídeo de prueba de resolución 240x320 píxeles a tres canales, a una tasa de 30 cuadros por segundo (cps). Cuando se descargan partes del cálculo en la unidad GPU, se obtiene un factor máximo de 12.39 veces de aceleración cuando se compara con la versión secuencial ejecutada en el CPU, utilizando una resolución de alta- definición [17].

El algoritmo de alineación de vídeo toma dos vídeos, denominados referencia y consulta, registrados en diferentes momentos en trayectorias similares a velocidades diferentes y encuentra diferencias entre los cuadros de ambos vídeos. Puede detectar los cambios

entre las dos grabaciones que pueden apuntar a eventos importantes, incluso críticos, tales como objetos perdidos. El algoritmo procesa la información en tiempo real, por lo que necesita de herramientas para acelerar el procesamiento de los datos y dar los resultados requeridos. Para obtener los resultados obtenidos se experimenta combinando distintas plataformas de programación, las cuales son Phyton + OpenCV + OpenMP, Matlab + CUDA, OpenCV + C ++ y OpenCV + CUDA + C ++, resultando la combinación de OpenCV + CUDA + C ++ como el conjunto de herramientas mejor integradas entre sí y por lo tanto con mayor eficiencia en la aceleración del algoritmo. De forma específica se utiliza Nvidia NPP que es una biblioteca de funciones para realizar procesamiento acelerado CUDA centrada en el procesamiento de imágenes y vídeo ampliamente aplicable para los desarrolladores en estas áreas. Los resultados de aceleración se muestran en la tabla 1.1 [17].

Tabla 1.1. Tiempos de ejecución en dos etapas del algoritmo DCV en CPU, CPU con varios núcleos (CMP) y tarjeta gráfica Tesla M2070.

Fase	CPU(seg)	CMP (OpenMP)	Tesla M2070
MHS	4.5	1.17	1.25
ECC	11.52	4.43	0.93

Con estos resultados se considera la integración entre OpenCV y CUDA con mejor desempeño en términos de aceleración de algoritmos en procesamiento de vídeo.

1.7.3 Aceleración del algoritmo de Detección de rostros Viola-Jones en GPU

La detección de rostros es un proceso costoso computacionalmente hablando y por lo tanto, la aceleración de este proceso puede influir en el avance de los sistemas de visión por computadora. Por lo tanto, se presenta un sistema de aceleración GPU de detección frontal del rostro utilizando el algoritmo Viola-Jones basado en el Adaptive Boosting (Adaboost) utilizando OpenCV y CUDA. Los resultados de los experimentos muestran que la aceleración de GPU para la detección de rostros es capaz de alcanzar una velocidad de hasta 18 veces y mantener su precisión de detección, en comparación con el algoritmo convencional de la versión no acelerada.

El algoritmo Viola-Jones [3] se divide en dos etapas. La primera parte del algoritmo Viola-Jones consiste en entrenar un conjunto de clasificadores basados en las características débiles de Tipo-Haar y formar un clasificador en cascada con todos los clasificadores débiles prometedores. El clasificador se entrena con miles de imágenes de muestra del rostro (ejemplo positivo) e imágenes arbitrarias (ejemplo negativo) que se escalan al mismo tamaño. La segunda parte es la detección, donde el algoritmo busca en cada ubicación del marco de entrada aplicando el clasificador de cascada entrenado en una ventana de búsqueda dinámica para localizar características de una cara humana.

El algoritmo de Viola-Jones depende de estos valores de característica para juzgar si hay un rostro. El método de búsqueda del rostro en OpenCV consiste en escalar los clasificadores en MxM ventanas de búsqueda con un factor de escala preestablecido. La implementación del kernel CUDA se realiza en el procesamiento de la ventana de búsqueda a escala original aplicando los clasificadores. El kernel usará el resultado de los módulos anteriores para procesar y lanzar hilos de MxM para trabajar dentro de la ventana de búsqueda para el cálculo de las características hasta que se haga la decisión de clasificación [3].

Finalmente el algoritmo da salida a las coordenadas de la cara detectada si el rostro está presente. El framework GPUCV en OpenCV emplea un enfoque denominado procesamiento en paralelo de etapas para hacer la paralelización dentro del procesamiento en cascada de una sola etapa. Así, todo el bloque lineal CUDA está trabajando con una sola ventana de búsqueda y posteriormente se procesará una etapa de clasificación, la figura 1.4 muestra un ejemplo de clasificación.

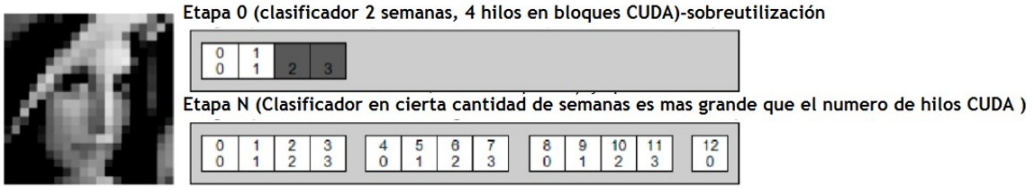


Figura 1.4. Procesamiento etapa paralela de los píxeles en la imagen.

La implementación utiliza la cascada OpenCV de clasificadores para rostros (parte frontal) con aproximadamente 2730 clasificadores de tipo Haar. Se usan los parámetros de detección de 20x20 píxeles para el tamaño mínimo de la ventana de búsqueda y 1.1 para el factor de escala de la ventana. Se realizan experimentos con imágenes estáticas y con vídeos en vivo de resolución VGA (de la cámara) a una tasa de 24 cps. La figura 1.5 muestra el marcaje de los rostros en un cuadro azul [3].

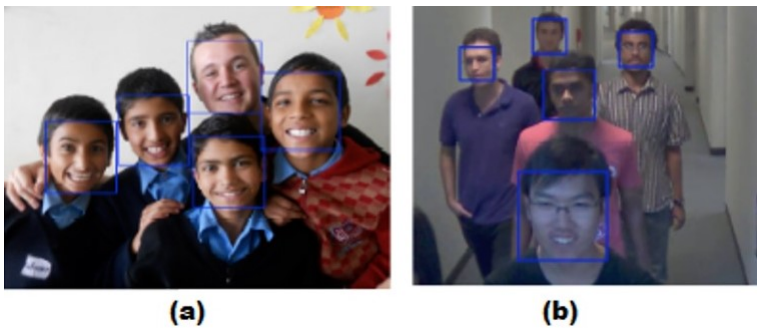


Figura 1.5. Resultados obtenidos de la detección del rostro en (a) imagen (b) y vídeo.

Comparando el procesamiento acelerado a través de la GPU del algoritmo de Viola -Jones se puede observar el rendimiento y las ventajas en comparación con la implementación en CPU. Se muestran los resultados en la tabla 1.2 [3].

Tabla 1.2. Desempeño de la respuesta del tiempo de la aceleración de la detección de rostro en comparación con el mismo algoritmo pero no acelerado.

Respuesta en el tiempo según el Hardware (mseg.)			
CPU	Aceleración propuesta en GPU Nvidia C2075	Aceleración propuesta en GPU Nvidia K20	Aceleración propuesta en GPU Nvidia K40
0.55237	0.0308	0.02718	0.02394

Este trabajo demuestra que la integración de la biblioteca OpenCV y la programación paralela mejoran la velocidad de procesamiento. La distribución y el procesamiento de datos se desarrollan de modo masivo para realizar mayor operaciones con los píxeles y obtener los resultados en un tiempo corto.

1.7.4 Algoritmo de magnificación de movimiento Euleriano

Este algoritmo revela variaciones temporales de movimiento imposibles de ver a simple vista en secuencias de vídeo y es la primera versión del algoritmo PMVF. Este algoritmo aplica una descomposición espacial seguida de un filtro temporal, esto es similar a las operaciones que plantea el algoritmo de flujo óptico. Por ejemplo, considere una región delimitada en un vídeo (objeto y fondo) donde se observan cambios de color, los cambios de coloración en la escena pueden ser atribuidos a dos razones posibles:

1. El objeto en la región delimitada ésta estático y cambio de color.
2. El objeto en la región delimitada se mueve, en cuyo caso la región (fondo) contiene partes del objeto que se movió, es decir, un píxel definido en la región de interés en el transcurso del tiempo puede contener información del objeto (que estuvo allí antes) o del fondo [19].

De esta manera los movimientos tenues se pueden hacer visibles al amplificar localmente los cambios de color *(I (x, t))* y renderizar el video.

El punto de referencia básico para la magnificación de movimiento, radica en considerar los valores del color de cada píxel en una locación espacial y así amplificar las variaciones espaciales en el tiempo de cada píxel en una banda de frecuencia temporal de interés, por ejemplo, la frecuencia de interés puede ser el pulso cardiaco [21].

Se puede amplificar la banda de frecuencia del pulso cardiaco al seleccionar la frecuencia de 0.88 Hz (53 pulsaciones por minuto), la amplificación revelará la variación en la tonalidad de la piel producida por el flujo sanguíneo lo que se reflejará en un movimiento perceptible en el vídeo procesado.

Para este tipo de aplicaciones el filtro temporal debe aplicarse a bajas frecuencias espaciales para evitar de igual forma la amplificación del ruido presente en los cambios de posición de los píxeles procesados en el cambio de cuadro.

El procesamiento planteado puede amplificar movimientos pequeños a pesar de no rastrear el movimiento como en los métodos Lagrangianos. El procesamiento temporal en el algoritmo Euleriano produce aumento de movimiento utilizando un análisis que se basa en las expansiones de la serie de Taylor de primer orden comunes en el análisis de flujo óptico [9].

1.7.4.1 Movimiento en primer orden

Para explicar la relación entre el procesamiento temporal y la ampliación del movimiento, se considera el caso simple de una señal 1D (señal senoidal) sometida a movimiento de traslación que representa el cambio de valor de un píxel en el tiempo, la magnificación de movimiento implica el desplazamiento en la fase en la señal senoidal.

Dado *I(x,t)* que denota la *intensidad de la imagen* en la posición x en el tiempo t, desde que la *imagen f(x)* experimenta movimiento traslacional, se puede expresar la intensidad observada con respecto a la función de *desplazamiento* $\delta(t)$, de tal forma que $I(x,t) = f(x+\delta(t))$ y $I(x,0) = f(x)$. La meta de magnificación de movimiento $\hat{I}$*(x, t)* es sintetizar la señal de modo que:

$$\hat{I}(x,t) = f(x + (1+\alpha)\delta(t)) \tag{1.1}$$

Para algún factor de amplificación α (seleccionado en una escala de 0.1 a 1.0 para evitar la magnificación del ruido). Asumiendo que la imagen puede ser aproximada por la expansión de Taylor, se escribe la imagen en el tiempo t, $f(x+\delta(t))$ en función de la serie de Taylor de primer orden con x, como se muestra a continuación:

$$I(x,t) \approx f(x) + \delta(t)\frac{\partial f(x)}{\partial x} \tag{1.2}$$

El resultado de aplicar un filtro temporal paso-banda a $I(x,t)$ en cada posición de x se designa como $B(x,t)$, se asume que la señal de movimiento $\delta(t)$ está dentro de la banda de paso del filtro de paso de banda temporal [21]. Por lo que tenemos:

$$B(x,t) = \delta(t)\frac{\partial f(x)}{\partial x} \tag{1.3}$$

A continuación se amplifica la señal $B(x,t)$ con respecto al valor α y se agrega a $I(x,t)$, resultando en la señal procesada $\tilde{I}(x,t)$:

$$\tilde{I} = I(x,t) + \alpha B(x,t) \tag{1.4}$$

Combinando las ecuaciones 1.2, 1.3 y 1.4 se tiene que:

$$\tilde{I} \approx f(x) + (1+\alpha)\delta(t)\frac{\partial f(x)}{\partial x} \tag{1.5}$$

Suponiendo que la expansión de Taylor de primer orden se mantiene para la perturbación amplificada mayor $(1+\alpha)\delta(t)$ se puede relacionar la amplificación de la señal pasa banda temporal para la magnificación de movimiento. El proceso de salida se simplifica como:

$$\tilde{I}(x,t) \approx f(x + (1+\alpha)\delta(t)) \tag{1.6}$$

Esta aproximación muestra que el desplazamiento $\delta(t)$ de la imagen local $f(x)$ en el tiempo t se amplifica a una magnitud de $(1+\alpha)$. Finalmente, para la señal procesada $\tilde{I}(x,t)$ se puede aproximar a $\hat{I}(x,t)$ a través de:

$$\tilde{I}(x,t) \approx \hat{I}(x,t) \tag{1.7}$$

$$\Rightarrow f(x) + (1+\alpha)\delta(t)\frac{\partial f(x)}{\partial x} \approx f(x) + (1+\alpha)\delta(t) \tag{1.8}$$

Con lo que se obtiene la magnificación de movimiento .En la figura 1.6 se muestra el resultado de la ecuación de magnificación $\tilde{I}(x,t)$ aplicado en una señal senoidal y la comparación con la señal original *f(x)* [3].

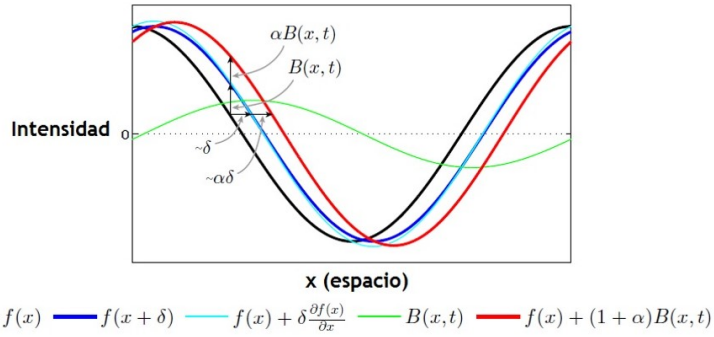


Figura 1.6. Señales unidimensionales que demuestran la traslación espacial.

El análisis se generaliza directamente al movimiento local-translacional en 2D, es decir, el cambio de valor de los píxeles en una posición determinada $(I(x_0, y_0, t_0)$ hasta $I(x_n, y_n, t_n))$ con respecto al tiempo (valor de los pixeles en el conjunto de cuadros que

conforman al video).

Los efectos de altas frecuencias, factores de amplificación altos y movimientos grandes sobre la amplificación de movimiento de una señal sinusoidal provocan ruido ambiental. La figura 1.7 muestra la amplificación de movimiento, para valores $\alpha > 1$ se magnifica el ruido [21].

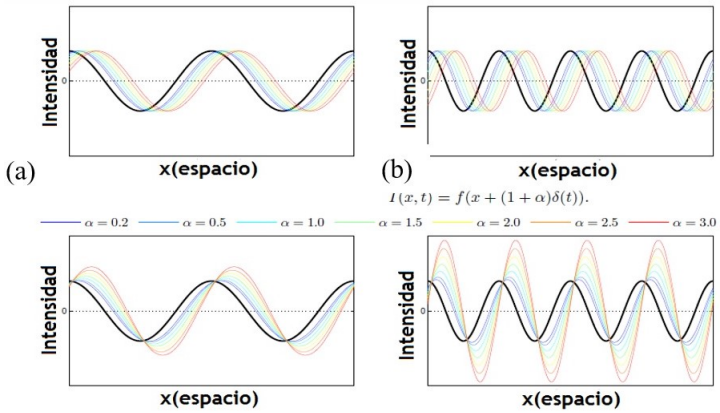


Figura 1.7. Ilustración de amplificación de movimiento sobre una señal unidimensional para diferentes frecuencias espaciales y valores de α. (a) Magnificación de desplazamiento para frecuencias bajas y valores de $\alpha <= 1$,(b).Magnificación de ruido para frecuencias altas y valores de $\alpha > 1$.

Las etapas del algoritmo de Magnificación de Movimiento Euleriano se ilustran en la figura 1.8 [21].

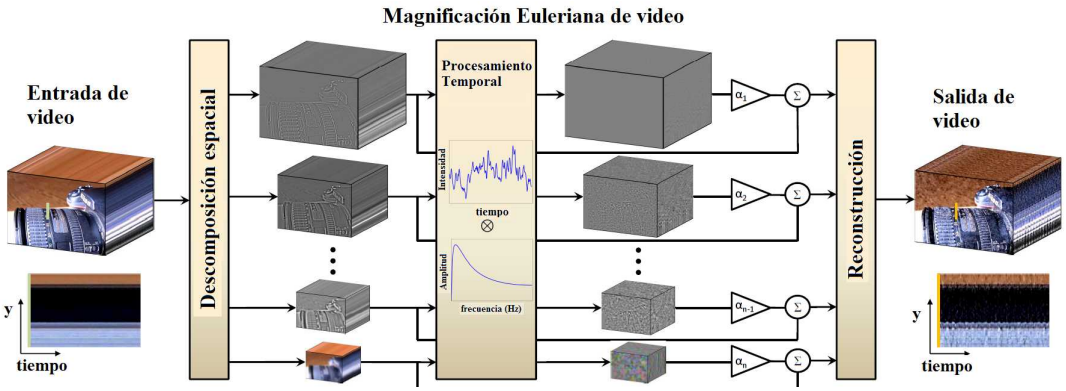


Figura 1.8. Diagrama a bloques del algoritmo Euleriano de magnificación de movimiento.

El primer paso es descomponer la secuencia de vídeo dentro de diferentes bandas espacio temporales, estas bandas pueden ser amplificadas de forma diferente porque pueden presentar diferente relación señal-ruido o pueden contener frecuencias espaciales

para las cuales la aproximación lineal no sostiene la magnificación de movimiento. En el último caso, se reduce la amplificación para estas bandas para suprimir efectos no deseados.

Se considera la serie de tiempo correspondiente al valor del píxel en una frecuencia de banda y se aplica un filtro pasa bajo para extraer la frecuencia de interés. Por ejemplo se puede seleccionar la frecuencia de 0.4-4 Hz que corresponde a 24 – 240 pulsos por minuto si es que se desea magnificar el pulso cardiaco. Si es posible extraer la frecuencia del pulso se puede usar una banda alrededor de ésta y procesar de manera uniforme para los niveles espaciales y para cada píxel dentro de estos niveles. La señal extraída se multiplica por un valor dado, el cual puede ser especificado por el usuario, y se contrae la pirámide espacial para obtener la salida final.

Dado que los vídeos son espaciales y temporalmente continuos, el filtro opera de manera uniforme sobre los píxeles para mantener la coherencia espacio-temporal de los resultados [21]. Sin embargo la amplificación está limitada para evitar la magnificación del ruido producido al cambio de los valores de posición en el tiempo de cada píxel inherente al cambio en el tiempo. La figura 1.9 muestra la superposición de los cuadros del vídeo procesado para magnificar el micro movimiento del pulso cardiaco en la muñeca del brazo.

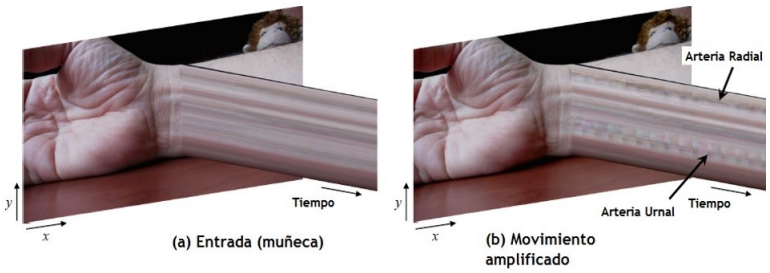


Figura 1.9. Resultados del algoritmo de magnificación de movimiento para amplificar movimientos del flujo sanguíneo en el sistema arterial. (a) Compendio de los cuadros del vídeo de entrada (muñeca) (b) Amplificación de la frecuencia cardiaca. Para este caso se toma la frecuencia con valor de 0.88 Hz. Y un factor de amplificación $\alpha = 10$.

1.7.4.2 Suceptibilidad al ruido

La variación de amplitud de la señal de interés es a menudo mucho menor que el ruido inherente en el vídeo. En tales casos, la mejora directa de los valores de píxeles no revelará la señal deseada. El filtrado espacial se puede utilizar para mejorar estas señales sutiles.

Si el filtro espacial aplicado no es lo suficientemente grande, la señal de interés no se revelará (figura 1.10). Suponiendo que el ruido es cero, significa en estado estacionario con respecto al espacio, puede así demostrarse que el filtrado paso bajo espacial reduce la varianza del ruido de acuerdo con el área del filtro de paso bajo. Con el fin de

aumentar la potencia de una señal específica, por ejemplo la señal de impulsos en la cara, se puede utilizar las características espaciales de la señal para estimar el tamaño del filtro espacial [21].

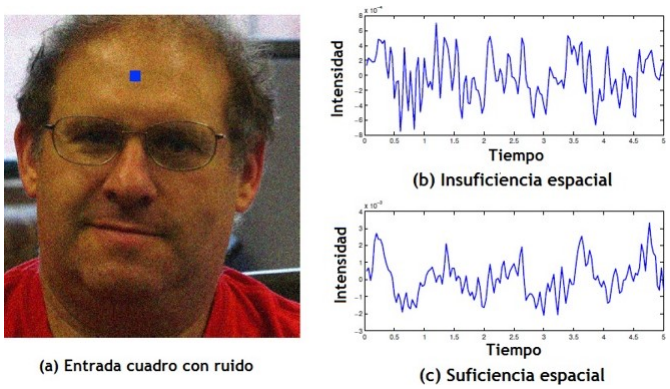


Figura 1.10. Relación de la potencia de la señal en la imagen. (a) Imagen con ruido de entrada (b) insuficiencia espacio-temporal del comportamiento de un píxel, (c) Capacidad espacio-temporal en el comportamiento del píxel.

Capítulo 2

Algoritmo de Procesamiento de Movimiento en Vídeo basado en la Fase

El método Euleriano que se presenta en la sección 1.7, produce una aproximación lineal al movimiento amplificado en vídeo, es simple y rápido, pero sufre de dos inconvenientes principales:

1. Funciona con factores de ampliación relativamente pequeños.
2. Puede amplificar significativamente el ruido cuando aumenta el factor de magnificación.

Para contrarrestar estos problemas, se explora otro enfoque Euleriano del procesamiento de movimiento que se basa en pirámides orientables de valor complejo [19], e inspirado en la fase basada en flujo óptico, así como en las variaciones de fase de Fourier [8]. Las variaciones de fase de la pirámide orientable compleja corresponden a movimientos locales en sub-bandas espaciales de una imagen. Se obtiene el cálculo de las variaciones locales de fase para medir el movimiento sin flujo óptico y realizar el procesamiento temporal para amplificar el movimiento en el tiempo seleccionando bandas de frecuencia (para el filtro pasa bandas) y reconstruir el vídeo modificado.

Las principales ventajas de este algoritmo son:

- Soporta amplificaciones mayores que el algoritmo lineal.
- Presenta resultados substancialmente mejores con respecto al ruido.

Esto debido a que el método lineal amplifica los cambios de brillo temporal y también amplifica el ruido de forma lineal, no así, cuando se modifican las fases, no se amplifican las amplitudes por lo que el ruido no aumenta [7].

Existe una relación entre el movimiento y la fase en pirámides orientables, se demuestra que aumentando las variaciones de fase por un factor multiplicativo se pueden amplificar movimientos sutiles. La pirámide orientable es una transformación que descompone una imágen de acuerdo con una escala espacial, orientación y posición.

2.1 Descripción del algoritmo

El funcionamiento del algoritmo PMVF se describe en el diagrama a bloques de la figura 2.1, como paso inicial se obtiene el vídeo a procesar, el vídeo debe cumplir con mantener dentro del campo de visión el objeto, persona o animal con movimientos tenues a magnificar y preferentemente no debe salir de cuadro de la cámara al momento de grabar la escena.

La lectura de los cuadros que comprenden el vídeo implica la lectura de los píxeles que forman a cada cuadro, posteriormente se realiza la descomposición de los píxeles en el dominio espacio-temporal, el filtrado en el dominio de la frecuencia y finalmente la reconstrucción.

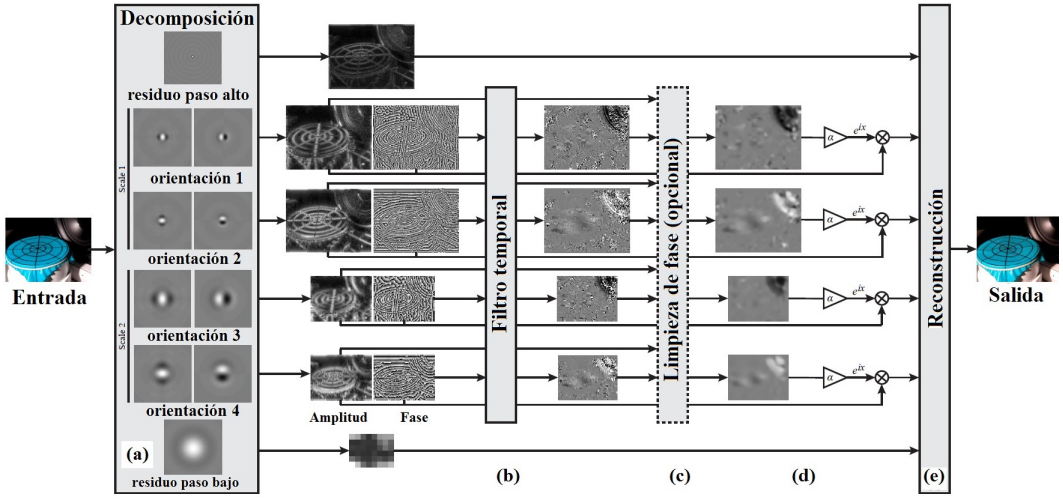


Figura 2.1. Diagrama a bloques del algoritmo PMVF (a) descomposición de la imágen, (b) filtro temporal (c) supresión de fases (d) reconstrucción de imágen.

El enfoque del filtrado en el dominio de la frecuencia implica el uso de pirámides direccionables, es decir, una serie de transformaciones de Fourier de fase en cada cuadro para identificar los movimientos tenues en el vídeo [19]. La reconstrucción procesa los datos obtenidos en las etapas previas por medio de la máscara Gaussiana y la conversión de espacio de color (de los *canales RGB* a *canales YIQ*) para presentar la salida (magnificación de movimiento) obtenida. En síntesis, se calculan las fases locales en el tiempo en cada escala espacial y la orientación de una pirámide direccionable para cada cuadro del vídeo. Se pasan temporalmente las fases locales para aislar las frecuencias temporales específicas relevantes para filtrar y eliminar cualquier componente temporal.

Estas fases temporalmente anilladas corresponden al movimiento en diferentes escalas espaciales y orientadas. Para sintetizar el movimiento ampliado, se multiplican las fases anteriores por un factor de amplificación α . A continuación, se utilizan estas diferencias de fase para amplificar (o atenuar) el movimiento en la secuencia mediante la modificación de las fases de cada coeficiente para cada cuadro.

2.2 Características y aplicaciones

Este enfoque basado en la fase de la magnificación de movimientos sutiles permite revelar con claridad y detalle fenómenos imperceptibles, no visualizados previamente. Se pueden magnificar movimientos sutiles, involuntarios, de baja amplitud (10-400 micras) en el ojo humano para visualizar y hacer hincapié en el movimiento alrededor del iris. Tal sistema de detección puede tener aplicaciones médicas, ya que el contenido de frecuencia de micro-tremor ocular tiene importancia clínica para detectar algunas patologías [5].

Las estructuras hechas por el hombre, como edificios y puentes, están diseñadas para balancearse en el viento, pero su movimiento es a menudo invisible por lo que al utilizar este algoritmo se puede obtener información útil al observar los desplazamientos de estas estructuras y prever daños o comportamiento inadecuado.

En maquinarias de dimensiones grandes (grúas) se puede recabar información útil. Por ejemplo, en la figura 2.2, al tomar un vídeo de una grúa en un día ventoso no parece moverse, sin embargo cuando se amplifican los movimientos de baja frecuencia se observa en el vídeo el balanceo del mástil de la grúa y la ondulación de su gancho, esto puede servir para evitar accidentes en estructuras frágiles cercanas a esta maquinaria.

Éste es un ejemplo en donde se puede aprovechar el usó de este método, sin embargo, existen más aplicaciones y en diversos campos, sólo queda que sean desarrollados e implementados de forma específica adecuando a las necesidades de los procesos [19].

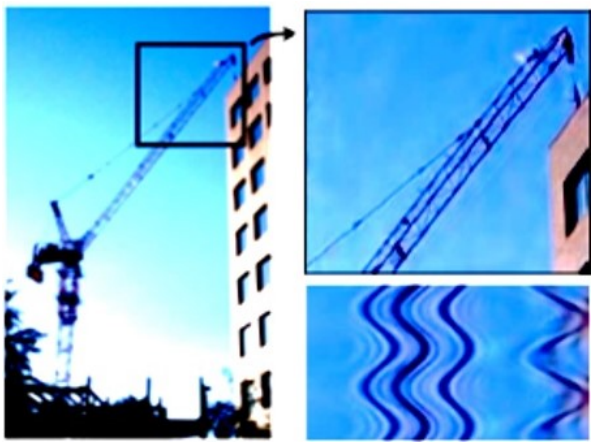


Figura 2.2. Experimentos realizados para mostrar movimientos sutiles en una grúa.

2.3 Análisis de tiempos

Para definir la complejidad del algoritmo se deben analizar las etapas que la conforman e identificar las etapas que sean posibles de acelerar.

La viabilidad de la aceleración depende en gran medida de las operaciones y funciones realizadas para el procesamiento de los datos y la salida obtenida.

2.3.1 Operadores y funciones de magnificación

El enfoque de magnificación de movimientos sutiles basado en la fase se fundamenta en pirámides orientables con valores complejos porque permiten medir y modificar movimientos locales. Para demostrar el procesamiento de movimiento basado en fase se da un ejemplo donde se usa una base de Fourier global y se considera el caso de un perfil de *intensidad de imágen* f en una dimensión (1D) en el tiempo, $f(x + \delta(t))$, para alguna función de desplazamiento $\delta(t)$. Se requiere sintetizar una secuencia con variación de movimiento, $f(x + (1 + \alpha)\delta(t))$, por algún factor de magnificación α [21].

Usando la descomposición de la serie de Fourier, se puede escribir el perfil de imágen desplazada como $f(x + \delta(t))$ como una suma de sinusoides complejas.

$$f(x + \delta(t)) = \sum_{\omega=-\infty}^{\infty} A_\omega e^{i\omega(x+\delta(t))} \tag{2.1}$$

En la cual cada banda corresponde a solo una frecuencia ω.

De la ecuación 2.1, la banda de frecuencia ω es la sinusoide compleja

$$S_\omega(x, t) = A_\omega e^{i\omega(x+\delta(t))} \tag{2.2}$$

S_ω es una sinusoide y la fase $\omega(x + \delta(t))$ contiene la información de movimiento.

2.3.2 Operadores y funciones de filtrado

Se puede modificar el movimiento cambiando la fase con el teorema de Fourier. Para aislar el movimiento en frecuencias temporales específicas, se filtra temporalmente la fase $\omega(x + \delta(t))$ con un filtro balanceado. Para simplificar la derivación se asume que el filtro temporal no tiene otro efecto excepto el de remover la componente de ω_x . La función resultante es:

$$B_\omega(x, t) = \omega\delta(t) \tag{2.3}$$

Se multiplica la fase por la función en pasa-banda $B_\omega(x, t)$ por α y se incrementa la fase en la sub-banda $S_\omega(x, t)$ para obtener la magnificación de movimiento en esta sub-

banda, como se muestra en la ecuación siguiente:

$$\hat{S}\omega(x,t) := S_\omega(x,t)e^{i\alpha B_\omega} = A_\omega e^{i\omega(x+(1+\alpha)\delta(t))} \tag{2.4}$$

El resultado dado $\hat{S}_\omega(x,t)$ es una sinusoide compleja que tiene movimientos exactamente $1+\alpha$ veces la entrada (figura 2.3). Se puede reconstruir el vídeo magnificado el movimiento al colapsar la pirámide. En este análisis, se suman todas las sub-bandas para obtener la secuencia ampliada del movimiento $f(x+(1+\alpha)\delta(t))$ [21].

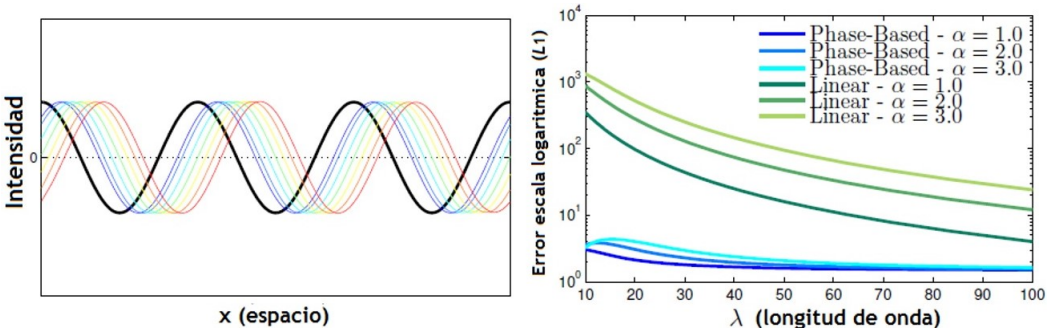


Figura 2.3. Señal sinusoidal magnificada en la fase, comparativo con distintos valores de magnificación α.

En general, los movimientos en un vídeo son locales y $\delta(t)$ es en realidad $\delta(x,t)$. Se utiliza la pirámide orientable compleja para tratar con movimientos locales ya que sus filtros tienen respuestas de impulso con soporte espacial finito (figura 2.4) [19].

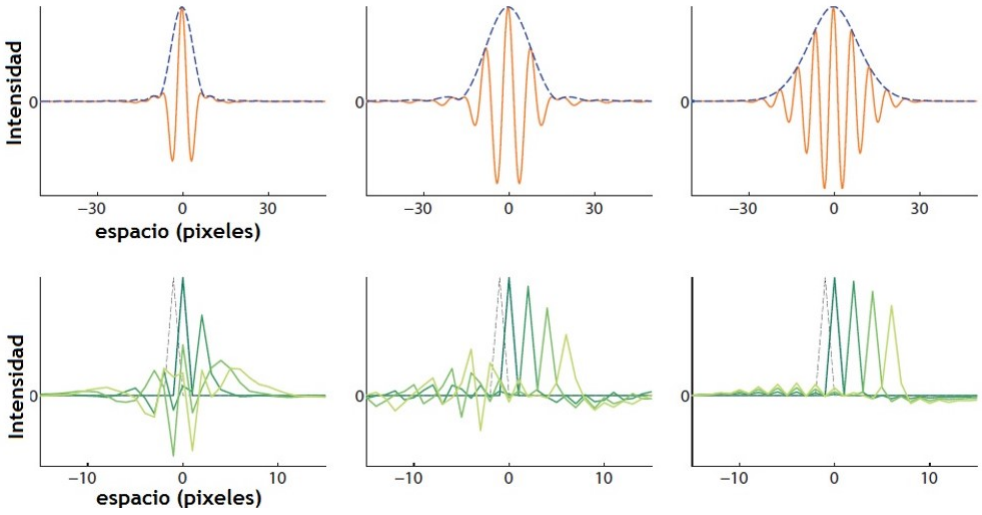


Figura 2.4. Señales del comportamiento de los píxeles en el algoritmo PMVF.

2.3.3 Análisis de tiempo

Las etapas base del algoritmo son el filtrado y la magnificación pero ambas tienen subprocesos para el manejo de la información (fases). El algoritmo completo consta de la adquisición de datos, lectura de información para procesar el vídeo, asignación de la información del vídeo en variables para su procesamiento, conversión de la información del dominio espacio-temporal al dominio de la frecuencia, filtrado, amplificación y finalmente reconstrucción del vídeo.

2.3.4 Filtrado y magnificación

Las etapas y subprocesos tienen un costo computacional global por lo que se debe analizar y cuantificar el costo específico de cada etapa y subproceso para analizar la viabilidad de la aceleración de las operaciones más robustas y por lo tanto con mayor costo de tiempo para generar un algoritmo optimizado en velocidad.

En la figura 2.5 se muestra la gráfica que resume los resultados del análisis porcentual de tiempo realizado al algoritmo de Procesamiento de Vídeo basado en la Fase.

Se observa que el mayor costo computacional radica en el filtrado y la amplificación que es representado por el procesamiento de los 26 niveles (o más) de la pirámide orientable con un costo del 76 % del tiempo total de ejecución, el segundo costo computacional está en la reconstrucción de las imágenes con las fases de interes.

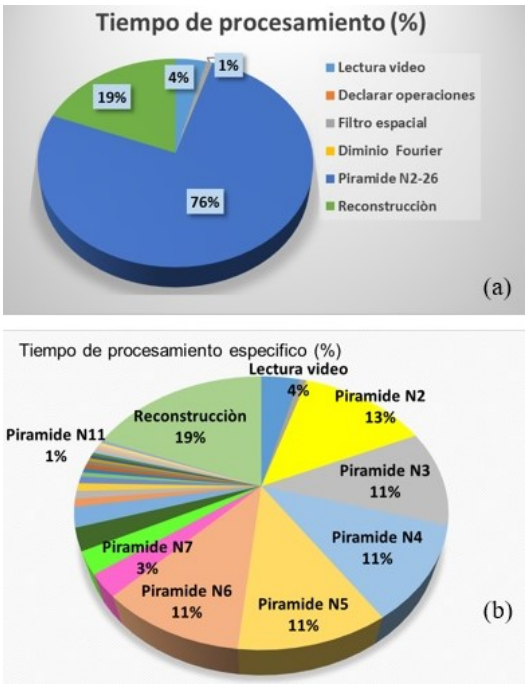


Figura 2.5. Gráfica de los porcentajes (costo temporal) que le corresponden a cada etapa del algoritmo.

En la figura 2.6 se muestra la gráfica de tiempo de las operaciones que lleva a cabo el algoritmo, se observa que el mayor consumo de tiempo se encuentra en el procesamiento de los primeros niveles de la pirámide orientable (del nivel 2 hasta el nivel 6).

La reconstrucción del vídeo también tiene un alto costo pero la viabilidad para acelerar este proceso es menor. Cabe mencionar que la pirámide orientable se procesa en un ciclo iterativo desde el segundo nivel hasta el nivel 26, por lo tanto si se acelera el segundo nivel se aceleran los niveles inferiores, solo la aceleración en los cinco primeros niveles aporta una reducción considerable del costo de tiempo.

Esto es debido al tamaño de las máscaras de transformación, pues entre mayor sea el nivel menor es la matriz, lo que implica menor cantidad de operaciones a realizar.

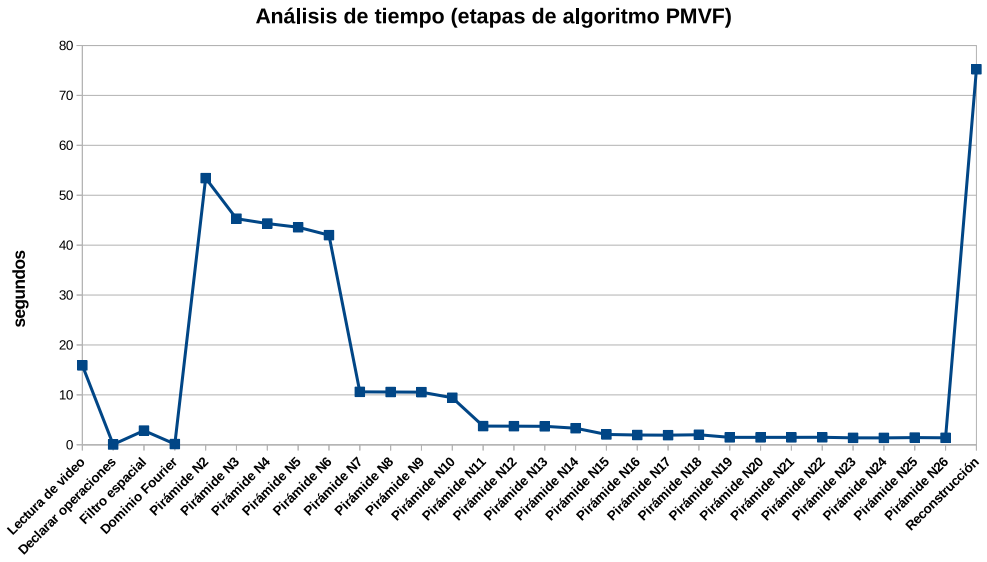


Figura 2.6. Gráfica de tiempos, ejecución de cada proceso realizado por el algoritmo.

En base a la información obtenida se decide acelerar la etapa de la pirámide orientable. La aceleración de esta etapa se baso en utilizar funciones optimizadas de la biblioteca OpenCV integradas con las tarjetas gráficas de Nvidia. Esto se presenta en el capítulo 3.

Capítulo 3

Aceleración del Algoritmo de Procesamiento de Movimiento en Vídeo Basado en la Fase

Como se mencionó en el capítulo 2 las etapas de mayor costo computacional son las que se deben acelerar para tener un mejor rendimiento en cuanto al tiempo.

En la teoría de cómputo un algoritmo paralelo, en contraparte a los algoritmos clásicos o algoritmos secuenciales, es un algoritmo que puede ser ejecutado por partes en el mismo instante de tiempo por varias unidades de procesamiento, para finalmente unir todas las partes y obtener el resultado requerido, todo esto depende de cuan fácil sea dividir el algoritmo en partes. Los algoritmos paralelos son importantes porque es mejor tratar grandes tareas de cómputo de forma paralela que de forma secuencial.

Las opciones para la aceleración del algoritmo son el uso de hardware específico, en este caso, una tarjeta gráfica Nvidia usada como multiprocesador y una librería diseñada para el procesamiento digital de imágenes con opción a paralelizar tareas, como lo es OpenCV. A continuación se plantean detalles en cuanto a cómo utilizar estas herramientas.

3.1 Uso de librería OpenCV

OpenCV es una biblioteca de funciones de programación dirigidas principalmente a la visión por computadora en tiempo real. Originalmente desarrollado por el centro de investigación de Intel en Nizhny Novgorod (Rusia), fue apoyado más adelante por Willow Garage y ahora es mantenido por Itseez. La biblioteca es multiplataforma y gratuita para su uso bajo la licencia BSD de código abierto. Con OpenCV se puede avanzar en aplicaciones intensivas en CPU o GPU que impliquen procesamiento de

imágenes y vídeo en tiempo real [13]. Los objetivos de utilizar OpenCV en este trabajo son:

- Usar código abierto proporcionado para optimizar la estructura básica del algoritmo.
- Aprovechar la infraestructura de la librería OpenCV para que el código del algoritmo sea fácilmente legible y transferible a otras aplicaciones o a un modelo portátil.
- Usar el apartado de paralelización con CUDA e integrarlos para ejecutar el algoritmo en forma acelerada.
- Obtener una reducción del costo computacional al procesar el algoritmo y acercarse a un procesamiento en tiempo real.

OpenCV tiene una estructura modular, lo cual significa que el paquete incluye una gran cantidad de librerías estáticas compartidas [2], algunos de estos módulos son:

- Core: módulo para definir estructuras básicas, incluyendo el arreglo multidimensional *(array Mat)* y funciones básicas usadas por otros módulos.
- Imgproc: módulo de procesamiento de imágenes que incluye filtros lineales y no lineales, transformación geométrica de imágenes, conversión a espacios de color, histogramas y muchos más.
- Vídeo: módulo de análisis de vídeo que incluye estimación de movimiento, substracción de fondo y algoritmos de seguimiento de objetos.
- Calib3d: algoritmos geométricos básicos de múltiples vistas, calibración de cámara, estimación de la posición del objeto, algoritmos de correspondencia estereoscópica y reconstrucción de elementos 3d.
- Objdetect: detección de objetos.
- Highgui: interfase de captura de vídeo, códigos de procesamiento de imágen.
- Gpu: algoritmos de aceleración de diferentes módulos de OpenCV.

De forma específica existen una serie de métodos para trabajar con los píxeles de las imágenes en forma matricial, por ejemplo la clase Mat permite realizar aplicaciones básicas sobre las imágenes (suma, resta, multiplicación, etc.) al mismo tiempo que permite realizar operaciones más detalladas y funciones específicas en las imágenes (filtros espaciales, operaciones con más de un canal en las imágenes, operaciones con números complejos, etcetera) [16].

Las aplicaciones para OpenCV abarcan áreas como la segmentación y el reconocimiento de patrones, herramientas para procesamiento en 2D y 3D, identificación de objetos, reconocimiento facial, seguimiento de movimiento entre otras [13]. Además,

para apoyar algunas de las aplicaciones anteriores se incluye un módulo con funciones estadísticas de aprendizaje automático.

Las ventajas de usar OpenCV es que está escrito en C y su interfaz primaria está en C ++, prácticamente la librería de OpenCV facilita las operaciones de procesamiento de imagen más comunes y algunas operaciones no tan comunes [14]. Para el caso de que se requiera alguna función en el tratamiento de píxeles y no exista en OpenCV de igual forma se pueden utilizar como sub funciones aquellas operaciones existentes en OpenCV para crear la función requerida y optimizar en espacio de procesamiento. OpenCV es útil también porque puede crear enlaces con software de uso común como Python, Java y MATLAB / OCTAVE [18].

3.2 Uso de tarjetas gráficas Nvidia-CUDA

El procesamiento paralelo necesita de hardware específico como lo son las tarjetas gráficas de Nvidia o AMD. Las tarjetas gráficas cuentan con procesadores diseñados para trabajar con datos en tareas específicas en forma masiva. En este caso Nvidia cuenta con CUDA, que es una arquitectura de cálculo paralelo, que aprovecha la gran potencia del GPU (unidad de procesamiento gráfico) para proporcionar un incremento extraordinario del rendimiento del sistema.

Los sistemas computacionales están pasando de realizar el procesamiento central en el CPU a realizar coprocesamiento repartido entre el CPU y el GPU de ahí que las tarjetas gráficas son usadas como procesador paralelo de propósito general accesible para cualquier aplicación [10]. La figura 3.1 muestra el potencial de las tarjetas gráficas (GPU Nvidia) al procesar distintos algoritmos relacionados con visión por computadora [12].

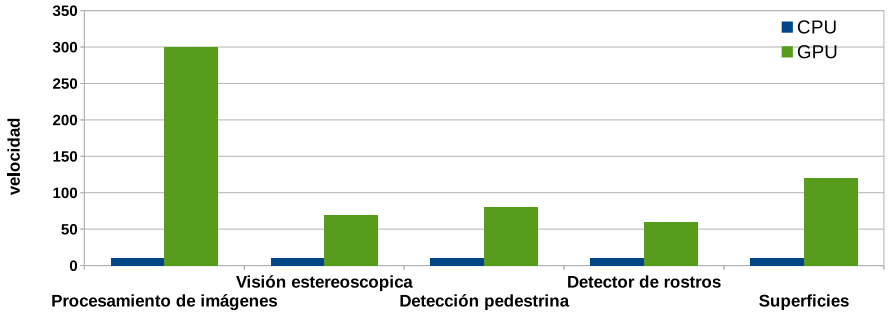


Figura 3.1. Gráfica comparativa entre la velocidad de procesamiento de CPU vs GPU Nvidia.

El lenguaje CUDA utilizado en tarjetas Nvidia permite programar en lenguaje C con extensiones. Parte del código se ejecuta en el CPU y cuando se necesita potencia de cálculo paralelo se deriva el código para que sea ejecutado en el GPU por medio de una llamada, luego la secuencia de ejecución es devuelta al CPU. De esta manera se maximiza el potencial de proceso de la computadora. Las herramientas de desarrollo

están constituidas por varios componentes clave: un controlador de vídeo con soporte para CUDA, el controlador de CUDA propiamente dicho para comenzar a programar en paralelo, el kit de desarrollo de software (SDK), contiene entre otras cosas un compilador C de Nvidia (NVCC), librerías de programación, un controlador de runtime y un debugger para el GPU. La librería de programación de CUDA también incluye programas de álgebra lineal y transformaciones de Fourier [11].

El modelo de programación tiene algunas limitaciones, por ejemplo, las funciones recursivas deben convertirse a lazos (loops) porque no están soportadas, los hilos (threads) deben ser ejecutados en grupos para obtener buen rendimiento y el paso de datos entre el CPU y el GPU puede convertirse en un cuello de botella según la implementación [6]. Aun así, si se utiliza correctamente se obtienen grandes beneficios. Por todo lo mencionado anteriormente se usa esta herramienta para acelerar el algoritmo PMVF.

3.3 Integración OpenCV-CUDA

Los módulos CUDA de Nvidia para OpenCV incluyen la clase para declarar las matrices de las imágenes (cv::cuda Mat), que es un contenedor primario para los datos guardados en la memoria GPU. Su interfaz es muy similar con la declaración de las matrices (cv::Mat) en su contraparte del CPU. Todas las funciones de GPU reciben cuda::Mat como argumentos de entrada y salida. Esto permite invocar varios algoritmos de GPU sin descargar datos. El módulo GPU de interfaz de la API también se mantiene similar con la interfaz del CPU cuando sea posible. Así al desarrollar aplicaciones creadas en OpenCV en el CPU pueden usar GPU sin cambios excesivos [15].

El módulo OpenCV GPU es un conjunto de clases y funciones para utilizar las capacidades de computación GPU. Se implementa utilizando la API NVIDIA CUDA Runtime y sólo admite GPUs NVIDIA. El módulo OpenCV GPU incluye funciones de utilidad primarias de visión de bajo nivel y algoritmos de alto nivel. Las funciones de utilidad y las primitivas de bajo nivel proporcionan una potente infraestructura para desarrollar algoritmos de visión rápida aprovechando el GPU, mientras que la funcionalidad de alto nivel incluye algunos algoritmos de última generación (como correspondencia estéreo, detectores de rostros entre otros).

El módulo de GPU está diseñado como una API de nivel del equipo (host), esto significa que si tiene binarios de GPU OpenCV pre-compilados, no es necesario que tenga instalado el CUDA Toolkit o escribir ningún código extra para utilizar el GPU

El módulo GPU OpenCV está diseñado para facilitar su uso y no requiere ningún conocimiento de CUDA. Sin embargo, tal conocimiento sin duda será útil para manejar casos no triviales o lograr el más alto rendimiento. Es útil entender el costo de varias operaciones, qué hace el GPU, cuáles son los formatos de datos preferidos para aprovechar el mayor rendimiento. El módulo GPU es un instrumento eficaz para la implementación rápida de algoritmos de visión computarizados acelerados por GPU. Sin embargo, si el algoritmo implica muchas operaciones simples, entonces, para el mejor rendimiento posible, se deben crear sus propios *kernels* para evitar operaciones adicionales de escritura y lectura en los resultados intermedios.

3.4 Estrategias de la aceleración

Es difícil incrementar la capacidad de procesamiento con un único procesador por eso es mejor aumentar la capacidad de cómputo mediante la inclusión de más unidades trabajando en paralelo, logrando así la ejecución de varios flujos de instrucciones dentro del proceso. Se debe ser cauto con la excesiva paralelización de los algoritmos ya que cada algoritmo paralelo tiene una parte secuencial y debido a esto los algoritmos paralelos pueden llegar a un punto de saturación, ley de Amdahl [10]. El algoritmo PMVF se basa en el procesamiento de imágenes a nivel de píxeles y el cambio de valores de estos a través del vídeo permite manejar la información para ampliar el espectro de frecuencias y hacer visible los movimientos tenues. La información en un vídeo consta de grandes cantidades de datos, el tipo de datos a trabajar permite utilizar herramientas de paralelización para incrementar la velocidad de cálculo de operaciones y decrementar el tiempo de respuesta.

Las funciones de filtrado se pueden acelerar pues las librerías de OpenCV están diseñadas para optimizar la cantidad de operaciones a procesar a través de la distribuciones de operaciones en los GPU y realizar multiprocesos. La figura 3.2 muestra las etapas del procesamiento en donde se usan las capacidades de cómputo del GPU.

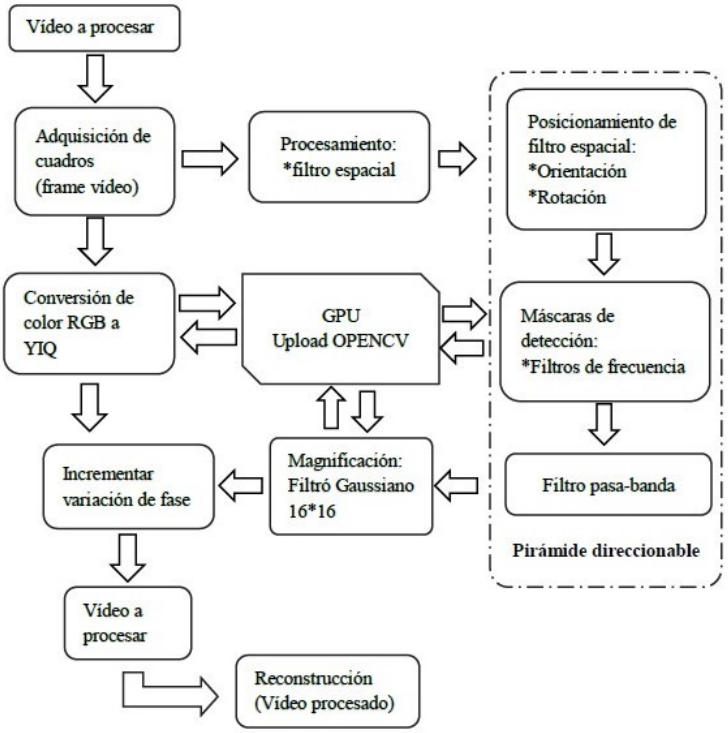


Figura 3.2. Descripción general del algoritmo a bloques en los puntos de envió de datos para procesar en el GPU para cada cuadro que compone al vídeo.

La carga y descarga de datos se realiza de manera eficiente debido a la capacidad de OpenCV para enviar y recibir datos al GPU.

Etapas criticas de procesamiento:

- Máscara de detección que contiene los filtros de frecuencia.
- Filtros gaussianos.
- Conversión de color.

Todas estas etapas se llevan a cabo para cada uno de los niveles de la máscara piramidal orientable para cada uno de los cuadros *(frames)*, en términos de operaciones a realizar se expresa de la siguiente forma.

$operaciones = NPO * (Gauss\_Filter * Frames + M\_Frecuencia * Frames)$
NPO = *niveles de pirámide orientable*
$Gauss\_Filter$ = *Filtro gaussiano dimension de* $16 * 16$
$M\_Frecuencia$ = *Cuadros en función de la frecuencia*
$Frames$ = *Cuadros totales en el vídeo*

Adicionalmente se puede tomar en consideración la resolución de vídeo pues entre mejor resolución tenga mayor cantidad de píxeles se van a procesar. En definitiva la cantidad de información a procesar en el algoritmo PMVF se debe y se puede acelerar con el software y el hardware adecuado.

3.4.1 Aplicación de funciones de procesamiento en OpenCV

Para compilar las estructuras y métodos necesarios para procesar las imágenes de vídeo en el algoritmo, la transferencia de información entre el equipo (CPU) y el dispositivo (GPU) es importante y por ello se cuenta con funciones específicas para facilitar el acceso al dispositivo, el primer paso es incluir las librerías denominadas $\#include < opencv/core2/cuda.hpp >$ e $\#include < opencv/core2/cudaarithm.hpp >$ las cuales permiten el llamado e inicialización de la comunicación para el envió de datos, en versiones anteriores a OpenCV 3.1.0 la librería era nombrada con la etiqueta GPU, en la versión 2.4.3 se manda a traer por medio de la etiqueta CUDA.

Las matrices de datos (imagen con dimensión n-filas, n-columnas) se referencian entre los espacios de memoria del dispositivo y equipo, así, el envió de imágenes en formato matricial se descarga sobre la memoria del dispositivo y se ejecuta la operación en el GPU para devolver el resultado obtenido hacia la memoria del CPU, esto se realiza las veces que sea necesario $(CPU_{memoria} \leftarrow GPU_{memoria}, GPU_{memoria} \leftarrow CPU_{memoria})$.

Las funciones de procesamiento y compilación del código se realizan en el compilador integrado en OpenCV al instalar las librerías de CUDA, así, el código principal hace la transferencia de datos mediante la instrucción “upload” desde el equipo al dispositivo y viceversa todo esto de forma transparente para el usuario.

El uso de las funciones y llamados en el código serial se dan en las etapas con mayores repeticiones y no en todo el código serial, esto debido al costo de tiempo en la transferencia de datos. Cabe mencionar que la librería OpenCV procesa operaciones específicas en el GPU mientras tenga los datos en su memoria de trabajo, el resultado obtenido se envía a la memoria del CPU para continuar con más operaciones, si es que las hay, por ejemplo para procesar una suma entre el GPU y el CPU se lleva a cabo el siguiente proceso.

Paso 1: cargar datos hacia GPU (matriz de datos de la imágen)
Paso 2: procesar suma en GPU (matriz de datos de la imágen)
Paso 3: cargar resultado obtenido en CPU (matriz de datos resultante de la imágen)
Paso 4: continuar con el procesamiento en CPU hasta que se requiera otro llamado a el GPU

Por estó es conveniente procesar un conjunto de operaciones para evitar múltiples transferencias entre la memoria del GPU y el CPU lo que provoca retraso en la operación global del algoritmo, por ejempló se puede realizar la operación suma, mantener el resultado en la memoria del GPU y aplicar la transformada de Fourier seguido de un filtro Gaussiano para enviar finalmente el resultado a la memoria del CPU, esto se describe en los pasos siguientes.

Paso 1: cargar datos GPU ($upload[matriz_{píxeles}$ *, valores de los píxeles*$]$)
Paso 2: procesar suma en GPU ($matriz_{píxeles}$ *+ constante)*
Paso 3: procesar Transformada de Fourier en GPU ($DFT($ $matriz_{píxeles}$ $)$)
Paso 4: procesar filtro Gaussiano en GPU ($GaussBlur($ $matriz_{píxeles}$ $)$)
Paso 5: carga Resultado CPU (upload ($GaussBlur[matriz_{píxeles}$ $]$))

El procesamiento de las operaciones en OpenCV – CUDA se da de tal forma que dependiendo de la operación, datos y complejidad que se esté usando, puede acelerar las operaciones en rangos diferentes. Por ejemplo:

- Función suma *ADD* con un grupo de datos de imagen 1200∗750 píxeles procesa a una velocidad de 10 veces en GPU contra la velocidad en CPU.
- Función suma *ADD* con un grupo de datos de imagen 700∗450 píxeles procesa a una velocidad de 13 veces en GPU contra la velocidad en CPU.
- Función conversión de color *BGR2XYZ* con un grupo de imagen 1200∗750 píxeles procesa a una velocidad de 15 veces en GPU contra la velocidad en CPU.
- Función conversión de color *BGR2XYZ* con un grupo de imagen 700∗450 píxeles procesa a una velocidad de 10 veces en GPU contra la velocidad en CPU.

3.4.2 Aceleración pirámides direccionables

La detección de los cambios pequeños en los píxeles entre cuadro y cuadro en el vídeo se realiza a través de los filtros en frecuencia. La base de filtrado en frecuencia radica en el cálculo de los filtros espaciales en 4 orientaciones en las que se rotan y procesan las imágenes. Se aplica la transformación de Fourier a la matriz de filtros espaciales para obtener las máscaras de transformación en el dominio de la frecuencia, lo que en conjunto forman las pirámides orientables, cabe señalar que estas máscaras de transformación sirven como filtros de frecuencia y escalamiento.

El escalar los filtros espaciales sirve para tomar en consideración el número de frecuencias al momento de filtrar, así, las cinco primeras máscaras contienen mayor cantidad de frecuencias a filtrar por lo que el número de operaciones es más grande, para las ultimas máscaras el número de operaciones de filtrado es menor por lo que la cantidad de datos procesados es considerablemente menor.

3.4.2.1 Coeficientes filtro espacial

La construcción de las máscaras de filtrado espacial se lleva acabo al obtener los coeficientes de las máscaras ($coef\_mascara\_transf[i]$), para esto se obtiene la función de transferencia que se construye a partir de una malla rectangular. La malla rectangular contiene la información del ángulo y de la fase, de cada píxel en el cuadro o imagen, que corresponden a las coordenadas polares de los puntos con origen en el centro de la malla. La raíz cuadrada con rango de 1 hasta la n-orientación es igual a la diferencia π en la función ventana angular con una pendiente (coseno del ángulo en el dominio de la frecuencia) que se aproxima a una función Gaussiana. Esta operación se describe en el algoritmo 1.

Algoritmo 1 Construcción de la malla de coeficientes de filtro espacial

Entrada: orientacion $\leftarrow 4$
$orden \leftarrow (orientacion - 1)$
$coef\_mascara\_transf[i] \leftarrow 0$
nPixel es el valor de píxeles en la n-filas
NF es el número de filas
Salida: Coeficientes de la malla ($coef\_mascara\_transf[i]$)
para $i = 1$ hasta NF **hacer**
$fase \leftarrow \frac{2^{(2*orden)}*factorial(orden^2)}{orientacion*factorial(orden^2)}$
$ángulo \leftarrow \frac{(\pi+nPixel[i]-\pi(b-1))}{orientacion} - \pi$
Obtener máscara de transformación
$coef\_mascara\_transf[i] \leftarrow 2\sqrt{fase} * cos(angulo)^{orden} | angulo < \frac{\pi}{2}|$
fin para
devolver $malla[i,j] \leftarrow coef\_mascara\_transf[i]$
Repetir *hasta completar el número de* ***coeficientes de la malla*** *de acuerdo a la cantidad de columnas del cuadro.*

Una vez obtenidos los coeficientes de las máscaras ,se rota en 3 ángulos para obtener las cuatro orientaciones del filtro (la original, tres rotaciones y la máscara inicial) que son los primeros cinco niveles de la pirámide orientable (la rotación de los filtros permite detectar los cambios de posición y valor de un píxel en el transcurso del vídeo). El procesamiento del número de niveles de la pirámide orientable se realiza como se menciona a continuación.

La suma de los coeficientes con rango mayor igual a 1 hasta la n-columna *(filters)* es igual a la función de ventana angular con un incremento en los valores del coseno (valores de los píxeles en el dominio de la frecuencia), así, el conjunto de índices o coeficientes que corresponden a valores diferentes de cero, en la malla rectangular, construyen los niveles de la pirámide orientable. Se retornan los índices ($FiltIdx[j,i]$) con un desfasamiento de 180^o .

El procedimiento se muestra en el algoritmo 2.

Algoritmo 2 Construcción de las máscaras de filtrado espacial

Entrada: $coef[n]$ son los coeficientes de la máscara por columna
filters[i] son los coeficientes de la máscara por fila
Salida: $FiltIdx$ es la matriz de coeficientes
para $i = 1$ hasta NP **hacer**
$filters[i] \leftarrow \sum_1^n coef[n]$
mientras $filters[i] > 0$ **hacer**
$Dim1[i] \leftarrow$ **evaluar**$(filters[i])$
fin mientras
$dim1 \leftarrow$ **rotar_angulo**$(Dim1)$
$FiltIdx \leftarrow$ **dimension**$(filters)$
para $j = 1$ hasta $min(dim1)$ **hacer**
$FiltIdx[j,i] \leftarrow dim1[j]$
fin para
fin para
devolver $FiltIdx[j,i]$
Repetir hasta completar el número de *niveles[j]* de la pirámide, los niveles se obtienen por $NP = floor([Log_2(minimo(Filas_{imagen}, Columnas_{imagen}))]) - 2$

En la figura 3.3 se observa las primeras cinco máscaras espaciales, la máscara (a) es para el filtrado de un mayor número de frecuencias y las máscaras de (b) hasta (e) son las frecuencias subsiguientes en distintas posiciones en el cuadro del vídeo.

Para obtener los niveles faltantes de la pirámide orientable se repite el procedimiento con los coeficientes que corresponden a cada subsiguiente nivel (se reduce secuencialmente a la mitad el tamaño de la máscara hasta la mínima dimensión), para este punto el número de coeficientes se reduce por lo que el tamaño de las máscaras de transformación disminuye hasta llegar al nivel superior (figura 3.4).Se almacenan en memoria las máscaras de transformación para aplicar sobre los cuadros del vídeo posteriormente.

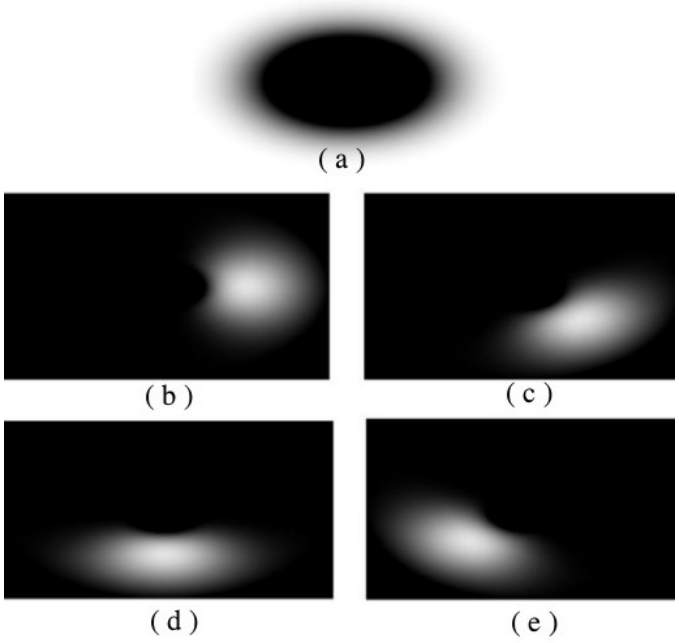


Figura 3.3. Máscara espacial de filtraje (a) filtro paso bajo, (b) filtro banda lateral, (c) rotación filtro lateral 45^o, (d) rotación filtro lateral 90^o, (e) rotación filtro lateral 135^o.

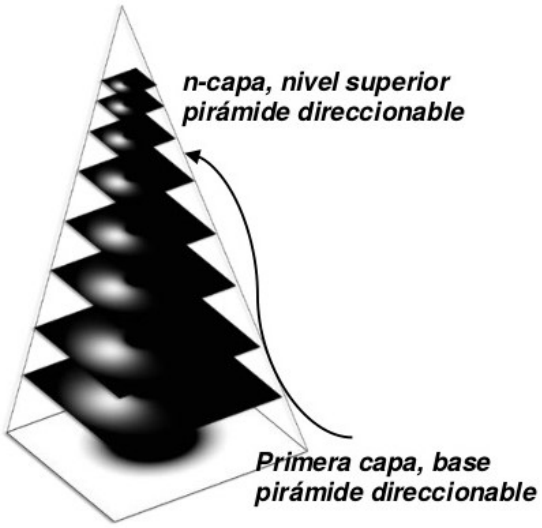


Figura 3.4. Representación de la pirámide direccionable.

3.4.2.2 Conversión en frecuencia

La conversión al dominio de la frecuencia es necesaria para obtener la fase y frecuencia de los cuadros del vídeo, por eso se debe procesar las pirámides orientables (hasta

este punto las máscaras de coeficientes espacial) con las operaciones de Fourier.

$FiltIdx_{frecuencia} \leftarrow conversión[FiltIdx]$

Los pasos a seguir para obtener la matriz $FiltIdx_{frecuencia}$ se presentanen el algoritmo 3, el cual consiste en generar el espacio de memoria temporal para el repositorio de los planos de la transformación de Fourier y la redistribución de los píxeles (en el dominio de la frecuencia), posteriormente se carga la información a la memoria del GPU para procesar los datos con las operaciones de Fourier.

La operación *fft2* en OpenCV consiste en obtener la matriz resultante de la función transformada discreta de Fourier aplicada sobre la matriz de datos, posteriormente aplicar la transposición de las posiciones de la matriz resultante (operación transpuesta) y finalmente aplicar la transformada de Fourier seguida de la transposición de la matriz resultante nuevamente [1], es decir, $fft(fft(x)')'$.

La operación ffshift consiste en un reacomodo de los cuatro cuadrantes de la matriz obtenida, en este caso (para OpenCV) se utiliza la función rotación.

El proceso termina al devolver los datos procesados a la memoria accesible al CPU, se repite el proceso con cada matriz comenzando nuevamente con la limpieza de los repositorios.

Algoritmo 3 Conversión de la malla de coeficientes en el dominio de la frecuencia

Entrada: *Mat planes*[ ]
  $Matq0(repositorio)$
  $nivel \leftarrow NP$
Salida: $FiltIdx_{frecuencia}$
  para $i = 1$ hasta *nivel* **hacer**
    para $i = 1$ hasta NFr **hacer**
      $canal\_GPU \leftarrow$ **cargar_GPU**$(Filtidx_{[i]})$
      $Trans\_Fourier \leftarrow$ **dft_GPU**$(canal\_GPU, nivel, planes)$
      $Trans\_Fourier2 \leftarrow$ **fftshift**$(Trans\_Fourier, planes)$
      $salida \leftarrow$ **fft2**$(Trans\_Fourier, 1hastanPix, planes)$
    fin para
    $q0[j] \leftarrow$ **cargar_CPU**$(salida)$
  fin para
  devolver $FiltIdx\_frecuencia \leftarrow q0[j]$
  Repetir hasta completar el número de *niveles[j]* de la pirámide, los niveles se obtienen por $NP = floor([Log_2(minimo(Filas_{imagen}, Columnas_{imagen}))]) - 2$

En la figura 3.5 se muestra, para un cuadro, el resultado de procesar las frecuencias y obtener la fase de interés.

Una vez obtenidas las máscaras de transformación en frecuencia se requiere pre procesar los cuadros del vídeo (obtener el *canal Y* de cada cuadro) para aplicar los filtros.

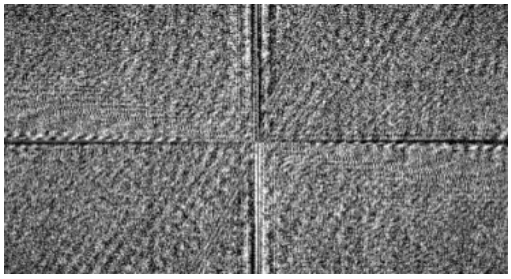


Figura 3.5. Transformación de Fourier para un cuadro del vídeo, información en el domino de la frecuencia.

3.4.2.3 Conversión canales de color

La detección de los cambios de movimiento requiere la primera capa de la transformación *YIQ* y el filtro gaussiano. La transformación de color de canales de *RGB* a *YIQ* se realiza a través de la matriz de operación sobre cada canal de la imagen, el sistema *YIQ* está destinado a aprovechar las características de la respuesta humana al color. El ojo humano es más sensible a los cambios en el rango naranja-azul (equivalente a I) que en el rango púrpura-verde *(Q)*, así se requiere menos ancho de banda para Q que para I. La transmisión en NTSC limita el ancho de banda de la señal I a 1,5 MHz y el de Q en 0,5 MHz. Las señales I y Q son entrelazadas en frecuencia dentro de la señal Y de 4 MHz, lo cual mantiene el ancho de banda total por debajo de 4,2 MHz [8].

La transformación de color se realiza a través de la multiplicación de los parámetros constantes para la conversión del espacio de color (operación matricial 3.1), y la contraparte para regresar a formato *RGB* se hace usando de igual forma un conjunto de parámetros constantes definidos (operación 3.2).

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.5957 & -0.2744 & -0.3212 \\ 0.2114 & -0.5225 & 0.311 \end{bmatrix} = \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{3.1}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0.9563 & 0.6210 \\ 1 & -0.2721 & -0.6474 \\ 1 & -1.1070 & 1.7046 \end{bmatrix} = \begin{bmatrix} Y \\ I \\ Q \end{bmatrix} \tag{3.2}$$

El conjunto de cuadros almacenados se clonan (uno a la vez) para poder procesar la transformación y obtener el canal Y sin afectar a los canales I y Q .La transferencia de la información hacia la memoria del GPU permite realizar las operaciones de forma acelerada con las funciones de OpenCV-CUDA, y devolver la información procesada de los cuadros a la memoria accesible al CPU.

El algoritmo 4 describe las operaciones mencionadas.

Algoritmo 4 Conversión de la imágen al espacio de *color YIQ*

Entrada: $v_{imagen[NF]} \leftarrow tamano(NFr)$
NFr es el número de cuadros
$v_{imagen[nf]}$ es el vector de matrices para almacenar los cuadros procesados
Salida: $FiltIdx_{frecuencia}$
para $i = 1$ hasta NFr **hacer**
$ImC_{GPU} \leftarrow$ **cargar_GPU** (ImC)
$ImConv \leftarrow$ **colorRGB2XYZ** (ImC_{GPU})
$canal_y \leftarrow$ **separar** $(ImConv)$
$canal_yCPU \leftarrow$ **cargar_CPU** $(canal_y)$
$v_{imagen[i]} \leftarrow canal_yCPU$
fin para
devolver $v_{imagen[i]}$

Como se observa en la figura 3.6 el *canal Y* resalta la forma de la imagen, los *canales I* y *Q* son complementarios, así que, las operaciones de transformación en frecuencia, el filtrado, la reconstrucción del vídeo se realiza en este canal para finalmente unirse con los canales I y Q originales para formar el vídeo magnificado resultante.

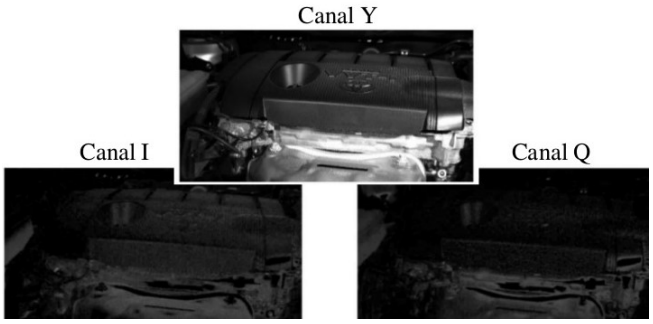


Figura 3.6. Conversión de color de *RGB* a *YIQ*, como se observa el canal *Y* contiene mayor cantidad de información y sobresale de los canales *I* y *Q*.

Para la detección de las fases de interés en cada cuadro se convierte la información al espacio de frecuencias, esto se realiza aplicando la máscara de coeficientes ($FiltIdx_{frecuencia}$) al *canal Y* de cada cuadro.

3.4.2.4 Detección de frecuencia de interés

Las frecuencias de interés denotan el movimiento a magnificar, el filtro apropiado evita el ruido ambiental y mejor amplificación de los datos de interés. Para la detección de la fase y frecuencia a magnificar en los cuadros del video se sigue el procedimiento mencionado a continuación.

Se generan los repositorios para los planos de transformación y la redistribución de los datos (píxeles en el dominio de la frecuencia). Se envían los datos del *canal Y* obtenido en el punto 3.4.2.2 y los coeficientes de las pirámides direccionables hacia la memoria del GPU, se llama a la función $ifft2$, posteriormente se reordenan los datos ($fftshift$). La función $ifft2$ en OpenCV debe agregar el parámetro en escala real ($idft \leftarrow DFT\_SCALE \mid DFT\_REAL\_OUTPUT$) para calcular la distribución de Fourier y reacomodar los cuadrantes. Los coeficientes de la matriz Delta resultante contienen las frecuencias de interés para cada cuadro que son la referencia en la etapa de filtrado. Se envía el resultado de regreso al CPU.

Cabe mencionar que la primera matriz de coeficientes es la separación de las coordenadas polares extremas (valores bajos de repetición de píxel contra valores altos de repetición).

El proceso y conjunto de operaciones mencionados se muestra en el algoritmo 5.

Algoritmo 5 Filtro de frecuencias

Entrada: *Mat planes [ ] , complexI[ ] , canal_x_frecuencia , vector nPix(píxeles)*
Salida: *q0*

```
para i = 1 hasta NFr hacer
  para j = 1 hasta nivel hacer
    canal_GPU ← cargar_GPU (q0[i])
    v_GPUimagen ← cargar_GPU (v_imagen[i])
    Transformacion ← idft_GPU(canal_GPU, v_GPUimagen[i])
    Transformacion2 ← fftshift(Transformacion, NP)
    Delta[j] ← angulo(Transformación2)
  fin para
  Delta_CPU[j] ← cargar_CPU (Delta[j])
fin para
devolver  q0 ← Delta_CPU[j]
```

La conversión en frecuencia de la figura 3.7 muestra el resultado de cada una de las etapas de procesamiento.

3.4.2.5 Filtro respuesta al impulso

Filtro respuesta al impulso (técnica de la ventana) en paso bajo para la selección de frecuencias de interés, describe el paso de los micro movimientos, se utiliza la ecuación FIR fase lineal para realizar la operación (respuesta al impulso *h(x)* y frecuencia de corte f_t) [8]:

$$h(x) = 2f_t sinc(2\pi f_t x) \tag{3.3}$$

$$h(x) = 2f_t \frac{sin(2\pi f_t x)}{2\pi f_t x} \tag{3.4}$$

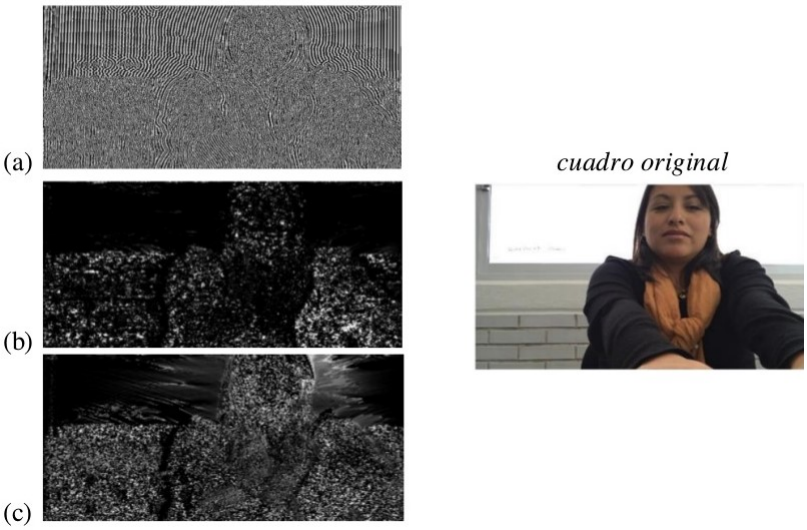


Figura 3.7. Cuadro original en color, los cuadros siguientes muestran las etapas de transformación y detección. (a) cuadro donde se observa el resultado de la transformada *ifft*, (b) cuadro que muestra el reacomodo de píxeles a través de la función *fftshift* (c) cuadro que muestra la fase de cada píxel.

$$h(x) = 2f_t \frac{sin(2\pi f_t x)}{2\pi x} \tag{3.5}$$

Al modificar la frecuencia de corte se puede obtener la selección de las frecuencias a rechazar por el algoritmo y eliminar la magnificación de ruido mediante la técnica de ventanas como se observa en la figura 3.8, donde el número de los coeficientes depende del número de frecuencias a filtrar, el punto central es la frecuencia de corte con las respectivas atenuaciones en las bandas de interés.

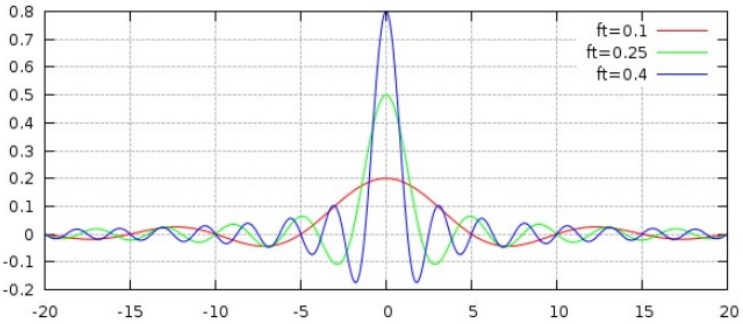


Figura 3.8. Gráfico de la selección de distintos factores de transformación en la frecuencia de corte.

El muestreo y procesamiento de las frecuencias se realiza de acuerdo al número de píxeles en el cuadro del vídeo a procesar, la respuesta al impulso toma los puntos particulares y de mayor peso en el muestreo con base a la ventana seleccionada en este caso es la ventana Hamming [8].

La obtención de la frecuencia de corte se debe seleccionar de acuerdo a la frecuencia a la que se pretende amplificar y la velocidad de captura de los cuadros del vídeo, las ecuaciones para relacionar estos parámetros son:

$$M = longitud\_del\_filtro - 1 \tag{3.6}$$

$$f_t = \frac{frecuencia\_de\_corte}{frecuencia\_de\_muestreo} \tag{3.7}$$

$$\omega_{lpf} = \begin{cases} \frac{sin[2\pi f_t(n-\frac{M}{2})]}{\pi(n-\frac{M}{2})} & n \neq \frac{M}{2} \\ 2f_t & n = \frac{M}{2} \end{cases} \tag{3.8}$$

Los valores de los pesos para la construcción del filtro digital se presentan en la figura 3.9 con entrada a respuesta al impulso, para cada vídeo procesado se obtendrán distintos pesos pero siempre en el rango -0.15 – 0.5.

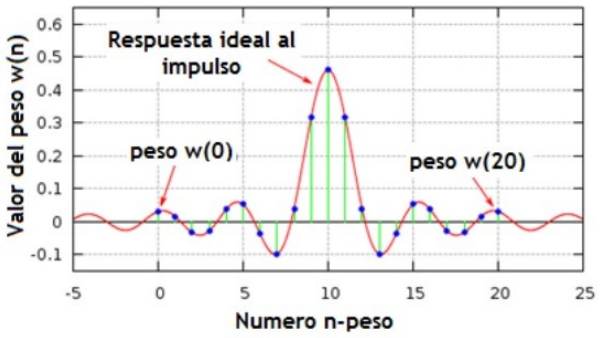


Figura 3.9. Gráfica de los pesos en los coeficientes del filtro, ecuación de la ventana de Hamming.

El diagrama de Bode sirve para interpretar el comportamiento de acuerdo a la banda de paso con respecto al orden del filtro y la frecuencia de corte, esto sirve para seleccionar la frecuencia necesaria para evitar rizos en la señal de salida, los rizos u ondulaciones provocan cambios abruptos en los valores de los píxeles del rango de frecuencias y el filtrado obtenido. Para ejemplificar este funcionamiento en la figura 3.10 se puede observar la respuesta de un filtro de ventana rectangular Hamming al impulso unitario y a la izquierda la gráfica con la ganancia en decibeles.

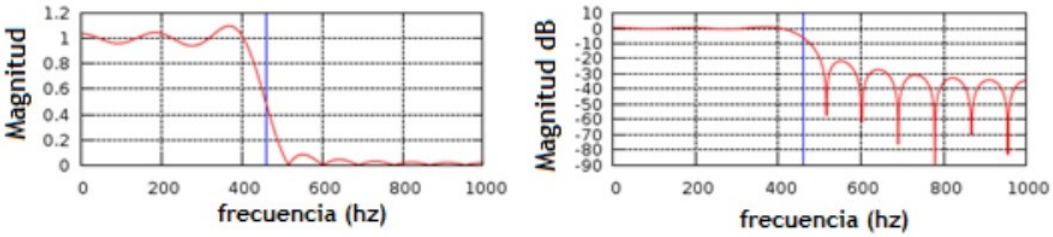


Figura 3.10. Análisis en frecuencia del filtro para distintas frecuencias.

Al conocer la frecuencia de muestreo y la frecuencia de corte en mayor parte se suprime mayor cantidad de ruido, además del ruido se desecha la posibilidad de producir magnificación de frecuencias de los píxeles del fondo contra los píxeles de la forma que provocan distorsión [14].

La figura 3.11 muestra una comparación de un cuadro de vídeo resultante procesado libre de perturbaciones entre el fondo y la forma contra el mismo cuadro procesado con presencia de ruido.

Figura 3.11. Vídeo procesado con coeficientes de filtro calculados de forma errónea contra cálculo realizado hacia la frecuencia de corte mejor aproximada.

La figura 3.12 muestra que el error en la selección de los coeficientes de filtrado para la frecuencia de corte seleccionada lo que siempre provoca ruido blanco, es decir, píxeles con valores altos en los tres canales del cuadro.

Figura 3.12. Cuadro procesado con coeficientes del filtro ventana Hamming erróneos.

El procedimiento para la etapa del filtro paso-bajo, se describe en el algoritmo 6, genera espacios en memoria para el vector de coeficientes del filtro paso bajo, la matriz de coeficientes para reaplicar el vector de coeficientes, el repositorio de los planos de la transformación de Fourier y la matriz de distribución de los datos *(ifftshift)*.

La función Hamming consiste en reaplicar lo establecido en las ecuaciones (3.5) a (3.8), la frecuencia de corte y la frecuencia de muestreo se proporcionan de acuerdo a la frecuencia del micro movimiento a magnificar. La transferencia al espació de memoria al GPU acelera las operaciones.

Las operaciones de Fourier son utilizadas para la redistribución y procesamiento de las frecuencias de interés. Finalmente se envía los datos procesados por el filtro al espacio de memoria accesible al CPU. Es importante resaltar que solo se almacenan los valores reales de los datos de *Delta_CPU*.

Algoritmo 6 Filtro ventana Hamming

Entrada: *Mat Delta_CPU[ ]*, $frec\_corte \leftarrow K_1$,$frec_muestreo \leftarrow K_2$,
$nf \leftarrow filas$, $nc \leftarrow columnas$, K_1 y K_2 son constantes proporcionadas
Salida: $Delta\_CPU[j]$
para $i = 1$ hasta nf **hacer**
Calcular los coeficientes del filtro (ventana Hamming)
$wind[i] \leftarrow 0.54 - 0.46 * cos(\frac{2*\pi*i}{N-1})$;
$wind[i] \leftarrow$ ***Hamming***$(wind[i], K_1, K_2)$
$coef\_filtro \leftarrow$ **cargar_CPU**$(wind)$
$Transforma \leftarrow$ **ifftshift**$(coef\_filtro)$
$Delta[i] \leftarrow$ ***fft*** $(transforma)$
fin para
$planes[i : nc] \leftarrow Delta$
$Delta\_salida \leftarrow$ ***ifft***$(Delta)$
$Delta\_CPU \leftarrow$ ***cargar_CPU*** $(Delta\_salida)$
$Delta\_CPU \leftarrow$ ***Real***$(Delta\_CPU)$
devolver $Delta\_CPU[j]$

La figura 3.13 muestra el resultado obtenido al aplicar el filtro paso bajo a un conjunto de cuadros del vídeo (motor de auto).

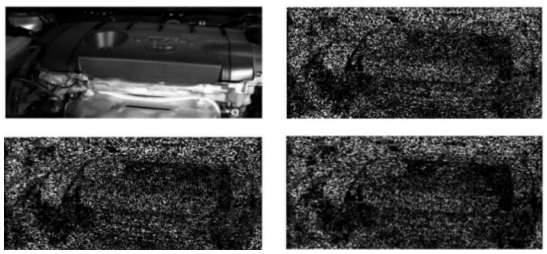


Figura 3.13. Cuadro de vídeo (Motor de auto Toyota) canal Y y tres cuadros consecutivos procesados por el filtro.

3.5 Reconstrucción de vídeo

Hasta este punto se ha procesado las etapas para magnificar los movimientos y filtrar frecuencias, la última etapa es la reconstrucción del vídeo. La reconstrucción inicia con el filtro gaussiano que sirve para evitar la magnificación de perturbaciones. El filtro Gaussiano es un filtro espacial, los filtros y etapas anteriores están en el dominio de la frecuencia y es importante resaltar que se realiza una combinación de un filtro espacial con datos en frecuencia para la reconstrucción de los cuadros (usar Filtro Gaussiano con datos en el dominio de la frecuencia). La función filtro Gaussiano en OpenCV requiere el tamaño de la máscara de convolución (*Dim_Máscara*) y la dispersión de los datos (*sigma_e*). La carga de los datos hacia la memoria del GPU también incluye el resultado del filtro paso bajo (sección 3.4.2.3, *Delta_CPU*). Es importante resaltar que si se coloca una frecuencia, que no esté cercano a la frecuencia de los desplazamientos tenues, y un factor alfa (*MagFase*) de magnificación alto se generaría una dispersión de los píxeles lo que dará como resultado a la salida un movimiento de la imagen en forma de ondulación. Se finaliza con la redistribución de los píxeles y transformación de Fourier. El algoritmo 7 muestra la secuencia de pasos a seguir.

Algoritmo 7 Máscara Gaussiana

Entrada: $Dim\_mascara[\,] \leftarrow Matriz[16,16]$, $tipo \leftarrow$ *Máscara tipo Gaussiana* , $sigma\_e \leftarrow$ *dispersión de convolución*, $MagFase \leftarrow$ factor de magnificación
Salida: $plantilla[i]$

```
para i = 1 hasta nf hacer
    Calcular los coeficientes del filtro (ventana Hamming)
    DeltaG ← cargar_GPU(Delta_CPU)
    Gaussiano ← filtro(DeltaG, tipo, sigma_e, Dim_mascara)
    amp ← MagFase * Gaussiano
    Redistribución de los píxeles (Máscara, desplazamiento circular)
    Luma ← fft2 (amp)
    Luma ← fftshift (amp)
    devolver plantilla[i] ← cargar_CPU(Luma)
fin para
```

El resultado del filtro de frecuencia se muestra en la figura 3.14.

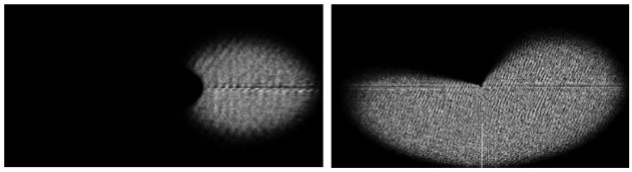


Figura 3.14. Resultado obtenido del filtrado Gaussiano para finalmente reconstruir la imagen.

A continuación tenemos la etapa de reconstrucción, en donde el resultado obtenido después del filtro Gaussiano (plantilla) se procesa retornando la información al dominio espacio-temporal a través de las funciones de la transformación de Fourier (*fftshift, ifft2*) uniendo el resultado con los canales I y Q culminando con la conversión al espacio de color de los canales *RGB*.

El procedimiento que se muestra en el algoritmo 8 (se repite para cada cuadro del vídeo).

Algoritmo 8 Reconstrucción de imágen

Entrada: $MatY_{vid}, Z_{vid} \rightarrow$ inicializar espacio en memoria (canales de color)
Salida: $video\_salida[j]$
  para $i = 1$ hasta NF **hacer**
    $filtrado \leftarrow$ ***cargar_GPU***$(plantilla[i])$
    $OutFrame \leftarrow$ ***fftshift***$(filtrado)$
    $OutFrame2 \leftarrow$ ***ifft2***$(OutFrame)$
    $matriz\_Real \leftarrow$ ***real*** $(OutFrame2)$
    $filtrado\_CPU[i] \leftarrow$ ***cargar_CPU*** $(matriz\_Real)$
  fin para
  para $j = 1$ hasta NF **hacer**
    $Vid[j] \leftarrow$ ***BGR2XYZ***$(video[j])$
    $frame[j] \leftarrow$ ***unir***$(Vid[i], Y_{vid}, Z_{vid})$
    $frame\_GPU[j] \leftarrow$ ***cargar_GPU***$(frame[j])$
    $resultado[j] \leftarrow$ ***XZY2BGR***$(frame\_GPU)$
    $Resultado[j] \leftarrow$ ***cargar_CPU***$(resultado[j])$
    devolver $video\_salida[j] \leftarrow$ ***guardar***$(Resultado[j])$
  fin para

La figura 3.15 muestra las etapas para obtener un cuadro reconstruidó con las fases de interes resaltadas. La superposición de los cuadros procesados mostrara el movimiento magnificado

(a) (b) (c)

Figura 3.15. La reconstrucción hacia el cuadro procesado, (a) datos en frecuencia para pasar hacía la (b) imagen espacio temporal en canales *YIQ* y finalmente realizar la (c) conversión a canales *RGB*.

Algunos puntos importantes a resaltar en el procesamiento OpenCV-Cuda se mencionan a continuación.

- Integrar la librería OpenCV y Cuda causa conflicto si no se elige la versión compatible, generalmente la versión compatible de OpenCV abarca el espacio de tiempo de la liberación de la última versión de CUDA.
- Declarar los espacios de memoria para almacenar el conjunto de píxeles se debe realizar de acuerdo al tipo de dato procesado (Mati, Mat, Matlf, Mat complexI, Mat planes [ ]) para evitar errores de procesamiento.
- Limpiar los espacios de memoria al terminar de usar para reutilizar o evitar saturación de la memoria temporal.
- Almacenar y definir los canales con los que cuenta la imágen a procesar.
- Definir los vectores de matrices para almacenar el conjunto de cuadros que comprenden al vídeo así como a los resultados de procesamiento, máscaras de convolución, máscaras de conversión, filtros, etc.
- Crear operaciones que no están construidas en OpenCV, las operaciones no construidas implican el acceso a los píxeles de forma individual lo que se traduce en procesos lentos.
- Usar las distintas funciones de la Transformada de Fourier. Existen para cada operación especifica por realizar, por ejemplo se tienen la transformada inversa discreta (IDFT), la transformada discreta (DFT), la transformada discreta con segmentación IDFT a la que se le tiene que agregar los parametros a utilizar (fuente, destinó DFT_SCALE | DFT_REAL_OUTPUT) entre otras.
- Para las operaciones de Fourier se deben declarar los 2 planos de almacenamiento (valor real y valor imaginario).
- Seleccionar la codificación del formato de lectura para obtener los datos del vídeo (para este caso, se puede procesar los tipo $*$.mp4, $*$.avi, $*$.mov).
- Procesar videos no mayores a 18 segundos a una tasa de 30 cps. con dimensiones de imagen de hasta 1200$*$950 píxeles, para no saturar la memoria temporal del equipo, en otro caso se puede optar el almacenar cada cuadro en el disco duro.
- Utilizar el identificador de vídeo, el formato de compresión y el formato de color para almacenar el vídeo procesado según el estándar fourCC (cuatro códigos de caracteres).
- Definir el tamaño de acuerdo a las dimensiones de los cuadros del vídeo para almacenar la información.

El módulo de OpenCV para el procesamiento acelerado es una herramienta muy útil cuando las operaciones a realizar, sobre imágen o vídeo, ya están construidas en la librería (OpenCV-Cuda), sin embargo, si el algoritmo presenta muchas operaciones

aun no implementadas la mejor opción es escribir y procesar el kernel de acuerdo a las necesidades, es decir, crear un kernel a la medida.

Los resultados obtenidos al procesar el algoritmo acelerado del Procesamiento de Movimiento en Vídeo basado en la fase se muestran en el capítulo siguiente.

Capítulo 4

Resultados

4.1 Herramientas de desarrollo

La aceleración del algoritmo PMVF permite reducir los tiempos de procesamientos sin excesivo ruido o costo computacional en la magnificación de movimiento de vídeo que puede utilizarse en distintas aplicaciones en la vida cotidiana.

Se desarrolló la aceleración del algoritmo en una tarjeta gráfica Nvidia GForce Gt 630M con 81 núcleos-CUDA con un PC portátil con procesador dual Core i5 con 6 Gb en RAM, los vídeos procesados tienen un tamaño de 800*1020 píxeles aproximadamente a una tasa de 24,26,30 cuadros por segundo con una duración de 15 $\pm$ 4 segundos, el sistema operativo utilizado es Linux en entorno Ubuntu a 64 bits con un compilador GNU compiler colección GCC si se desea procesar en entorno Windows se requiere instalar Microsoft Visual Studio con la versión compatible con OpenCV y CUDA.

4.2 Replicación de módulos

Como se mencionó en el capítulo 3, el algoritmo serial tiene módulos viables de ser acelerados para reducir el costo computacional, la replicación de estos módulos acelerados, el análisis por etapas y el procesamiento en forma global sirve para realizar una comparativa de desempeño.

La replicación de módulos del algoritmo PMVF ejecutado en forma serial contra la forma acelerada se describe en la tabla 4.1. Se muestra el tiempo que le toma procesar un vídeo al algoritmo serial y el tiempo que le toma procesar el mismo vídeo al algoritmo acelerado, se describe el tiempo para cada fase.

Como se puede observar en la figura 4.1 el rendimiento mejoró como se esperaba en el procesamiento de las pirámides N2-N6, las subsiguientes pirámides (N7-N26) también muestran mejora de rendimiento pero tienen menor peso considerando el procesamiento global. La etapa de reconstrucción también incrementó el rendimiento, sin embargo, al ser la etapa donde se procesa el vídeo de salida se debe ser cuidadoso en la restauración de los píxeles de cada cuadro para evitar sobre escritura de datos en las imágenes finales

Tabla 4.1. Comparativa de tiempo por etapas del algoritmo serial contra el acelerado en el procesamiento de vídeo.

Comparativa procesamiento del algoritmo PMVF			
Fase del algoritmo		Procesamiento serial (seg.)	Procesamiento acelerado (seg.)
1	Lectura de vídeo	15.91	15.81
2	Declarar operaciones	0.06	0.04
3	Filtro espacial	2.84	0.98
4	Dominio de Fourier	0.15	0.19
5	Pirámide N2	53.43	25.01
6	Pirámide N3	45.30	35.12
7	Pirámide N4	44.33	22.16
8	Pirámide N5	43.59	21.79
9	Pirámide N6	42.01	21.01
10	Pirámide N7	10.60	5.30
11	Pirámide N8	10.57	5.28
12	Pirámide N9	10.54	5.27
13	Pirámide N10	9.42	4.71
14	Pirámide N11	3.74	1.87
15	Pirámide N12	3.73	1.87
16	Pirámide N13	3.70	1.84
17	Pirámide N14	3.32	1.66
18	Pirámide N15	2.07	1.03
19	Pirámide N16	1.96	0.98
20	Pirámide N17	1.93	0.10
21	Pirámide N18	2.01	0.10
22	Pirámide N19	1.49	0.75
23	Pirámide N20	1.49	0.75
24	Pirámide N21	1.49	0.74
25	Pirámide N22	1.50	0.75
26	Pirámide N23	1.39	0.69
27	Pirámide N24	1.38	0.69
28	Pirámide N25	1.44	0.72
29	Pirámide N26	1.40	0.70
30	Reconstrucción	75.24	56.43

por lo que para asegurar esto la carga y descarga de los datos de memoria del GPU al CPU es fluida, es decir, cada operación en el GPU se descarga en el CPU para cada cuadro de video.

La figura 4.2 muestra la comparativa entre los valores uno contra uno de los valores de los píxeles en un cuadro del vídeo magnificado en forma serial (en este caso se trata del cuadro 100 del vídeo motor Toyota) contra el mismo cuadro del vídeo magnificado en forma acelerada.

Los valores de los píxeles entre uno y otro vídeo procesado idealmente debe ser cero, sin embargo existe una variación de $\pm$ 0.3 puntos en la desviación estándar, esto de forma cuantitativa. De forma cualitativa no se logra percibir la diferencia entre el vídeo procesado serial contra el vídeo procesado de forma acelerada. Sin embargo un factor

importante a tomar en consideración es el ajuste de la frecuencia a magnificar pues puede provocar un incremento en la contaminación de la imágen.

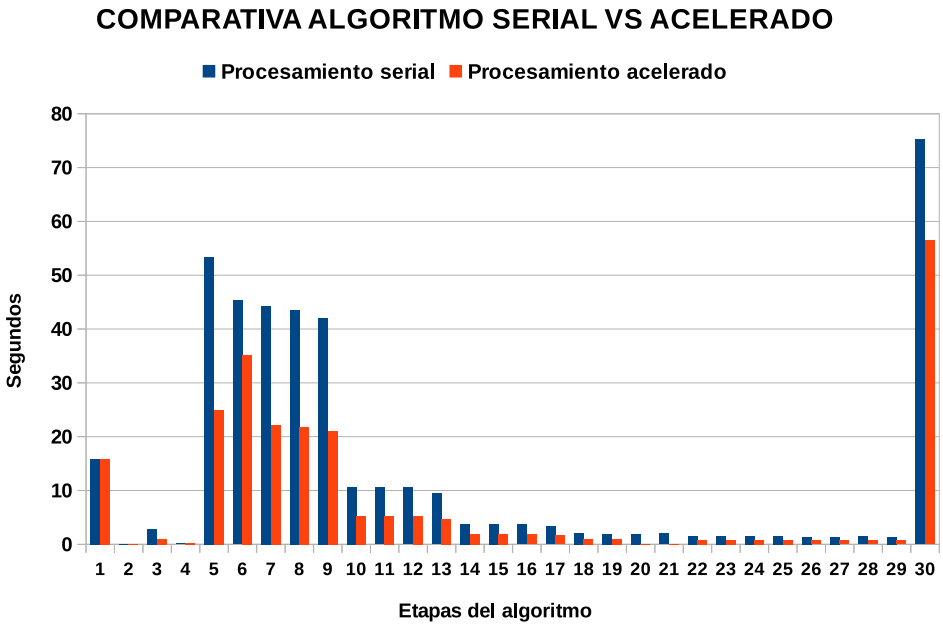


Figura 4.1. La gráfica muestra el decremento en el costo computacional del algoritmo procesando las etapas con mayor costo computacional en el GPU.

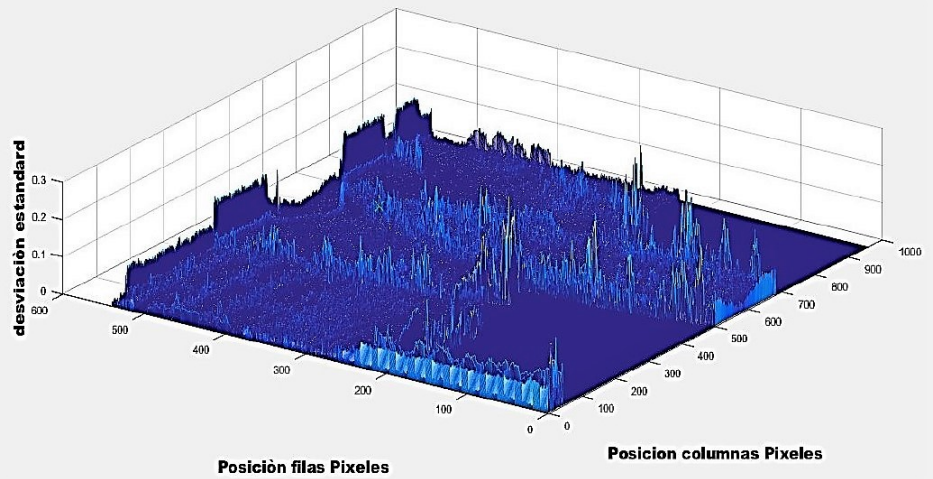


Figura 4.2. Variación entre los píxeles que forman a un cuadro de vídeo procesado serial vs acelerado (diferencia = píxel procesado forma serial - píxel procesado forma acelerada) idealmente debería ser cero.

4.2.1 Banco de pruebas y algoritmo

Se tiene un conjunto de vídeos que sirven de referencia para poder realizar las pruebas del procesamiento de vídeo. Los parámetros de los vídeos pueden ser modificados seleccionando diferentes factores de amplificación y distintos umbrales de prueba (frecuencia de corte e intervalo de frecuencia), el cambio en la salida resultante puede ser medido con los valores en las posiciones de los píxeles y con el ruido presente en el vídeo obtenido, el tiempo de procesamiento se mantiene variable en un umbral delimitado, debido a las operaciones de carga y descarga de la memoria del CPU y GPU. Otra forma de evaluar cualitativamente el desempeño del algoritmo es la cantidad de ruido presente en la salida (de vídeo) como factor muy importante de considerar para establecer la factibilidad del procesamiento acelerado.

La desviación estándar entre los valores de los píxeles y la posición dentro de la imagen muestra de forma cuantitativa la diferencia entre cada resultado procesado, esta diferencia a simple vista en el vídeo no se puede observar. La selección de los vídeos procesados se controla pues el fondo y la forma deben tener un elemento con micro movimientos para evitar magnificación de dos distintas frecuencias.

4.2.2 Rendimiento serial contra rendimiento acelerado

El desempeño logrado con el uso de las librerías de OpenCV junto con la tarjeta gráfica usado como coprocesador redujó el tiempo de procesamiento de los vídeos en un valor de 4 veces. Este desempeño se logró a través de algunos cambios realizados en el procesamiento del algoritmo como la máscara de convolución donde originalmente se tenía un tamaño de 20∗20 píxeles y se modificó a una razón de 16∗16 píxeles, esto porque no se logró observar un cambio significativo en el ruido de la imágen del vídeo magnificado.

Otro cambio radicó en el desplazamiento espacial de los píxeles a través de la operación *iffshift* reemplazada por la operación *fftshift*, esto porque no se observa ningún cambio significativo en la salida de vídeo. Este cambio se debe a que no es necesario crear esta función en `C++` para poder utilizarla y se recicla la función *fftshift*.

Una de las razones del uso de vídeos de 12 segundos es el almacenamiento de imágenes (cuadros totales del vídeo) a procesar pues existe el inconveniente de la saturación de memoria. Para este algoritmo se hace una copia (*clon*) del cuadro actual del vídeo y se almacena en un vector de matrices en espacio de memoria transferible del CPU al GPU.

Los resultados obtenidos se adquieren de un banco de 20 vídeos, en este apartado se presenta la información relevante de 5 de aquellos vídeos.

4.3 Resultados particulares

De forma general el algoritmo PMVF obtiene a la salida global de la magnificación de movimientos tenues presente en vídeo, sin embargo, también es importante

exhibir resultados particulares como son los tiempos de ejecución de la fase de filtrado,transformaciones de Fourier, conversión de color, filtro espacial entre otros. A continuación se describe el análisis realizado. En primera instancia se tiene la transferencia y procesamiento de la máscara gaussiana.

4.3.1 Procesamiento máscara Gaussiana

La máscara gaussiana tiene un tamaño de 16∗16 píxeles con función de filtro paso bajo (en el dominio espacial) para la detección de ruido. El tiempo de procesamiento secuencial es de 1.43 veces en promedio más lento contra el acelerado. En la tabla 4.2 se puede observar los datos de procesamiento para distintas resoluciones de vídeo y en la última columna la relación entre $\frac{Serial\_Optimizado}{Acelerado\_OpenCV-Cuda}$ para obtener la razón de aceleración, en la tabla 4.2 se puede observar esta comparación.

Tabla 4.2. Información y características de procesamiento de la función filtro Gaussiano.

Comparativa procesamiento del algoritmo PMVF						
Características de vídeo		Tiempo de procesamiento filtro Gaussiano (seg)				
Vídeos de prueba	Resolución vídeo (píxeles)	Serial optimizado (Matlab)	Serial paralelizado (Matlab-Cuda)	Serial en OpenCV	Acelerado OpenCV-Cuda	Razón de aceleración
Vídeo 1	780x450	0.044	0.012	0.052	0.034	1.483
Vídeo 2	1200x950	0.053	0.014	0.053	0.031	1.846
Vídeo 3	1500x1020	0.051	0.018	0.071	0.039	1.368
Vídeo 4	800x600	0.041	0.016	0.049	0.29	1.219
Vídeo 5	600x400	0.044	0.012	0.039	0.28	1.261

Es importante hacer notar que la función de filtro Gaussiano tiene un rendimiento de 16 veces al ser procesado con la función disponible en OpenCV-Cuda, sin embargo la transferencia de la memoria del GPU hacia el CPU le lleva un lapso de tiempo considerable por lo que la operación incluyendo el envió del resultado incrementa el costo computacional y disminuye la razón de aceleración.

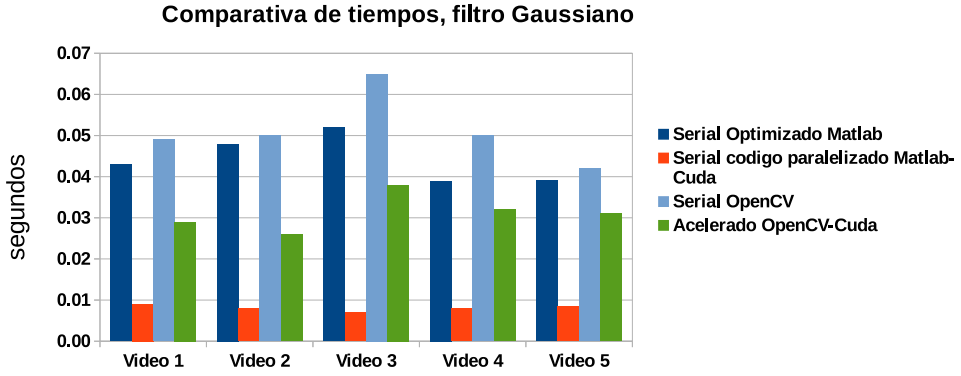


Figura 4.3. Gráfica comparativa para mostrar las diferencias en el procesamiento del filtro Gaussiano de forma acelerada.

La figura 4.3 muestra el desempeño en el tiempo de las distintas formas de procesar el filtro Gaussiano siendo OpenCV-Cuda la segunda que tiene mayor rendimiento en tiempo, sin embargo en el procesamiento global se compensa este desempeño e incrementa el rendimiento.

4.3.2 Procesamiento conversión de color

La relación de tiempo en la conversión de color está dada de igual forma por la razón de cambio en el algoritmo $\frac{Serial\_Optimizado}{Acelerado\_OpenCV-Cuda}$ (tabla 4.3), el cual en promedio arroja un resultado de 1.06 veces el rendimiento del algoritmo acelerado contra el algoritmo procesado en forma serial.

Tabla 4.3. Resultados e información de procesamiento de la función conversión de color.

Análisis de tiempos, evaluación parcial de resultados						
Características de vídeo		Tiempo de procesamiento conversión de color (seg)				
Vídeos de prueba	Resolución vídeo (pixeles)	Serial optimizado (Matlab)	Serial paralelizado (Matlab-Cuda)	Serial en OpenCV	Acelerado OpenCV-Cuda	Razón de aceleración
Vídeo 1	780x450	0.048	0.040	0.052	0.045	1.067
Vídeo 2	1200x950	0.054	0.054	0.055	0.051	1.059
Vídeo 3	1500x1020	0.068	0.062	0.062	0.065	1.046
Vídeo 4	600x400	0.045	0.038	0.041	0.42	1.071
Vídeo 5	600x400	0.042	0.039	0.043	0.039	1.077

En la figura 4.4 se observa que esta función no obtiene una mayor ganancia al ser procesada en forma acelerada, sin embargo en conjunto con las demás fases logra un mejor rendimiento global del algoritmo.

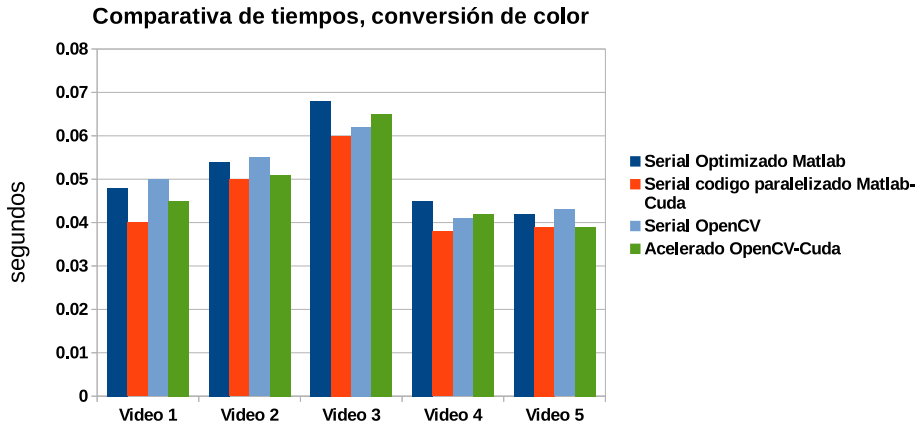


Figura 4.4. Gráfica comparativa para mostrar el bajo rendimiento al acelerar la conversión de color.

4.3.3 Procesamiento pirámide orientable

La transformación de Fourier aplicada a los cuadros del vídeo para generar el filtro frecuencia, magnificación de movimiento y reconstrucción que conforman a la pirámide orientable tiene un rendimiento en etapas descrito a continuación.

La tabla 4.4 muestra la primera fase de la pirámide orientable donde el rendimiento acelerado contra el serial tiene una razón de aceleración en promedio de 1.89 veces el rendimiento del algoritmo acelerado OpenCV-Cuda contra el serial optimizado.

Tabla 4.4. Resultados e información de procesamiento de la primera fase de la pirámide orientable.

Análisis de tiempos, evaluación parcial de resultados					
Características de vídeo		Tiempo de procesamiento $fft2$ y $fftshift$ (seg)			
Vídeos de prueba	Resolución vídeo (píxeles)	Serial optimizado (Matlab)	Serial en OpenCV	Acelerado OpenCV-Cuda	Razón de aceleración
Vídeo 1	780x450	0.14	0.12	0.07	2.00
Vídeo 2	1200x950	0.32	0.25	0.18	1.81
Vídeo 3	1500x1020	0.45	0.41	0.24	1.84
Vídeo 4	1500x1020	0.16	0.11	0.08	1.96
Vídeo 5	600x400	0.12	0.09	0.07	1.83

La figura 4.5 muestra la comparativa de procesamiento, siendo la forma acelerada con OpenCV-Cuda la de mayor rendimiento, como en etapas anteriores el rendimiento se ve afectado por la transferencia de información entre la memoria del GPU al CPU.

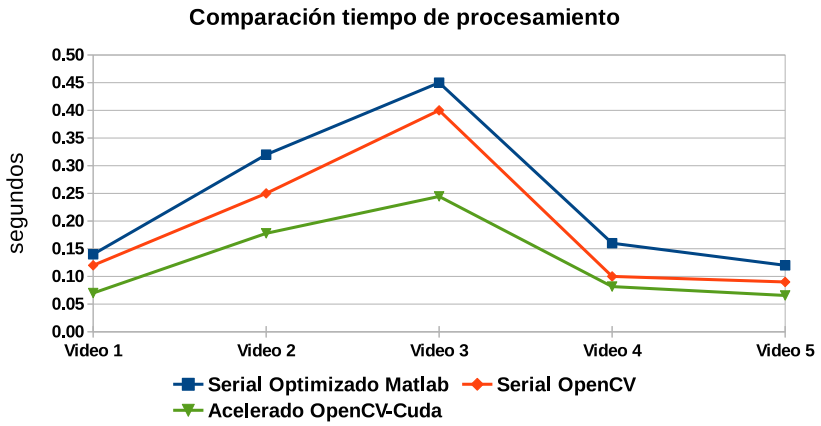


Figura 4.5. Gráfica comparativa para mostrar el rendimiento al acelerar la primera fase de la pirámide orientable.

La tabla 4.5 muestra la segunda etapa de la pirámide y su rendimiento en promedio es de 1.51 veces el algoritmo acelerado contra el algoritmo serial esto debido a los distintos tamaños de los cuadros del vídeo.

Tabla 4.5. Resultados e información de procesamiento de la segunda fase de la pirámide orientable.

Análisis de tiempos, evaluación parcial de resultados					
Características de vídeo		Tiempo de procesamiento $fftshift$ y $ifft2$ (seg)			
Vídeos de prueba	Resolución vídeo (píxeles)	Serial optimizado (Matlab)	Serial en OpenCV	Acelerado OpenCV-Cuda	Razón de aceleración
Vídeo 1	780x450	0.15	0.11	0.11	1.49
Vídeo 2	1200x950	0.42	0.32	0.22	1.95
Vídeo 3	1500x1020	0.52	0.49	0.35	1.48
Vídeo 4	1500x1020	0.25	0.19	0.25	1.01
Vídeo 5	600x400	0.21	0.15	0.12	1.63

La figura 4.6 muestra la comparativa de procesamiento, siendo la forma acelerada (en promedio) con OpenCV-Cuda la de mayor rendimiento. Cabe resaltar que para un cuadro de vídeo con dimensiones de 800∗600 píxeles el mejor rendimiento está en la forma serial en OpenCV, esto debido a la simetría del cuadro siendo un caso particular encontrado.

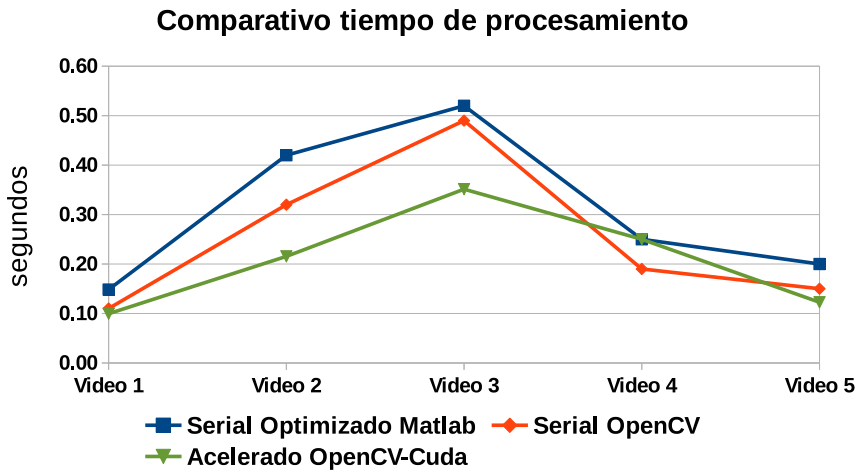


Figura 4.6. Gráfica comparativa para mostrar el rendimiento al acelerar la segunda fase de la pirámide orientable.

La tabla 4.6 que abarca la última etapa de la pirámide y la reconstrucción de vídeo, muestra que tienen un promedio de 2.98 veces de rendimiento el algoritmo acelerado contra el serial optimizado.

En la figura 4.7 se muestra la comparativa en el rendimiento de la última etapa de la pirámide orientable, se denota que es la de mayor eficiencia en aceleración de las tres etapas (de la pirámide orientable) ,lo que es relevante para el procesamiento global del algoritmo PMVF.

Tabla 4.6. Resultados e información de procesamiento de la segunda fase de la pirámide orientable.

Análisis de tiempos, evaluación parcial de resultados					
Características de vídeo		Tiempo de procesamiento $fftshift$ y $ifft2$ (seg)			
Vídeos de prueba	Resolución vídeo (pixeles)	Serial optimizado (Matlab)	Serial en OpenCV	Acelerado OpenCV-Cuda	Razón de aceleración
Vídeo 1	780x450	0.55	0.42	0.14	3.80
Vídeo 2	1200x950	0.75	0.65	0.25	3.01
Vídeo 3	1500x1020	0.85	0.75	0.30	2.81
Vídeo 4	1500x1020	0.45	0.39	0.16	2.90
Vídeo 5	600x400	0.45	0.39	0.16	2.40

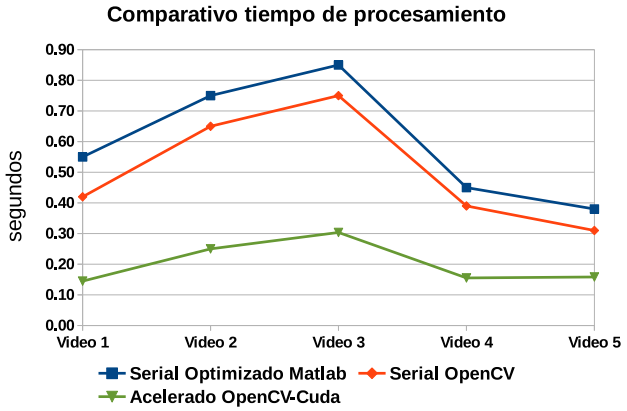


Figura 4.7. Gráfica comparativa para mostrar el rendimiento al acelerar la última fase de la primera orientable y la reconstrucción de vídeo.

En resumen, al obtener el promedio de las 3 etapas del procesamiento de la pirámide orientable, la razón de aceleración en esta etapa es de 1.86 veces.

Para concluir, solo resta conocer el rendimiento global del algoritmo PMVF acelerado, obtener el tiempo de procesamiento total y hacer la comparación contra el algoritmo optimizado serial para definir la reducción del costo computacional.

4.3.4 Prueba global del algoritmo

La generación de la estructura acelerada se basa en el análisis de tiempo en cada una de las partes que comprenden al algoritmo, finalmente al procesar estas etapas aceleradas sobre el número total de cuadros del vídeo con las especificaciones de la constante de magnificación y el umbral de magnificación se obtiene el rendimiento total de las fases aceleradas del algoritmo de PMVF. La tabla 4.7 muestra la comparación entre el tiempo de procesamiento del algoritmo serial optimizado en comparación con

el algoritmo acelerado en un conjunto de vídeos procesados.

Tabla 4.7. Resultados e información de procesamiento de 5 vídeos procesados en el algoritmo en forma secuencial (distintas formas) contra el algoritmo acelerado.

Análisis de tiempo, ejecución global de resultados)												
Características de vídeo			Parámetros de procesamiento					Tiempo de procesamiento algoritmos (*segundos*)				
Vídeos de prueba	Resolución vídeo (píxeles)	cps	α	λ_c	$\omega_{l(Hz)}$	$\omega_{h(Hz)}$	$F_{s(Hz)}$	Duracion video	Serial optimizado	Serial Matlab-Cuda	Serial openCV	Acelerado OpenCV-Cuda
vídeo 1	780x450	24	10	16	10	60	26	12	500	490	480	125
vídeo 2	1200x950	29	5	14	5	14	10	14	796	780	785	199
vídeo 3	1500x1020	30	7	45	7	9	30	11	500	480	490	125
vídeo 4	800x600	24	6	35	6	19	40	10	690	670	660	172
vídeo 5	600x400	26	48	29	48	59	50	12	680	660	430	170

α = factor de magnificación λc = frecuencia de corte

ωl = frecuencia, limite inferior ωh = frecuencia, limite superior

Fs = frecuencia de muestreo (se refiere a la frecuencia del micro-movimiento a magnificar)

La razón de cambio entre el algoritmo $\frac{Serial\_Optimizado}{Acelerado\_OpenCV-Cuda}$ es en promedio de 3.98 veces el tiempo de procesamiento entre el algoritmo desarrollado de forma serial optimizado contra el desarrollado en forma acelerada. La figura 4.8 muestra la gráfica con los tiempos de procesamiento del algoritmo ejecutado globalmente.

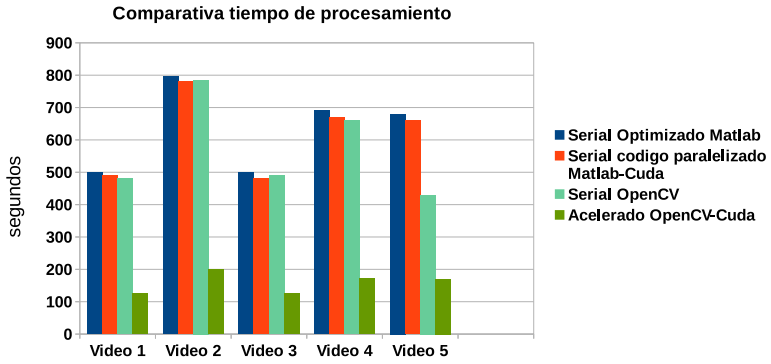


Figura 4.8. Comparativo entre el procesamiento del algoritmo con etapas seriales y aceleradas.

4.4 Discusión

Muchas tareas en procesamiento digital de imágenes con procesadores gráficos se aceleran eficientemente, sin embargo, existen otras tareas que no se paralelizan de igual forma, ya que contienen segmentos en serie donde los resultados de las etapas posteriores dependen de los datos obtenidos en fases previas. Estos algoritmos en serie no funcionan de manera sustentable en el GPU y no son adecuados para programar en esa arquitectura, por lo que a menudo se ejecutan más rápido en el CPU.

Puesto que muchas tareas del algoritmo PMVF consisten en subtareas paralelas y en serie, algunas tareas seleccionadas se pueden acelerar ejecutando partes de sus componentes en el CPU y otras en el GPU. Desafortunadamente, esto introduce dos fuentes de ineficiencia. Una es la sincronización, es decir, cuando una subtarea depende de los resultados de otra, la etapa posterior necesita esperar hasta que se termine la etapa anterior.

Otra ineficiencia es la sobrecarga al mover los datos de un espacio de memoria a otro entre el GPU y el CPU ya que las tareas de visión por computadora necesitan procesar muchos píxeles, puede significar mover grandes trozos de datos hacia adelante y hacia atrás. Estos son los desafíos clave en la aceleración de las tareas de procesamiento de imágenes y visión por computadora en un sistema con CPU y GPU.

Capítulo 5

Conclusiones

En este trabajo se propone usar cómputo acelerado a través de tarjetas gráficas y librería OpenCV para acelerar el procesamiento del algoritmo Procesamiento de Vídeo basado en la Fase.

En primer lugar se ha realizado un análisis del costo computacional del algoritmo serial y se han identificado las etapas de mayor costo en procesamiento y en memoria. El costo en memoria es difícil de reducir pues al procesar vídeo no se puede dejar de almacenar bits de los píxeles procesados o los que falten de procesar, sin embargo, la cantidad de operaciones a realizar si se deben reducir por la identificación de las etapas y operaciones de mayor costo computacional.

El mayor costo radica en la etapa de las pirámides direccionables u orientables que incluyen filtros digitales, transformaciones de las imágenes en el dominio de la frecuencia (filtros paso bajo y paso banda a una frecuencia específica para cada vídeo) y transformaciones en el dominio temporal (máscara gaussiana y conversión de color). Una vez identificadas las etapas de mayor costo se propuso usar la librería OpenCV específica integrada con CUDA para acelerar dichas etapas. Cabe mencionar que la transferencia de los datos del equipo en la primera llamada al dispositivo le toma mayor tiempo pues se inicializa la comunicación entre ellos por lo que se coloca un bloque de transferencia de suma de datos para tener habilitada la comunicación en los bloques posteriores en donde se realizan las operaciones concernientes al algoritmo, esto evita pérdidas de tiempo.

Los bloques paralelizados refieren los datos de la memoria del equipo al dispositivo mediante el módulo de OpenCV integrado con CUDA que está diseñado para su uso fácil y eficiente para lograr el más alto desempeño. La respuesta en cuanto a la aceleración del algoritmo de Procesamiento de Vídeo basado en la Fase tiene una ganancia de 4 veces contra el algoritmo no acelerado, esto se debe a la cantidad de ciclos iterativos que se deben realizar de cada uno de los filtros a procesar en cada uno de los cuadros del vídeo, la resolución de las imágenes influyen también en la tarea de reducir el tiempo. Para poder procesar el algoritmo PMVF en tiempo real aun es necesario un desempeño mayor a 4 veces.

Finalmente la comparación entre los cuadros resultantes al procesar el algoritmo

serial contra el algoritmo acelerado arroja una variación del 10 % en cuanto a la comparación de los valores de los píxeles. Esta variación no se logra percibir en el vídeo final pero está presente al realizar un análisis de la información cualitativa.

5.1 Conclusiones específicas

Con respecto a los objetivos planteados se puede concluir que:

- Las operaciones con mayor costo computacional implican la transformación al dominio de la frecuencia y la pirámide orientable, esto debido al uso de memoria para depositar las matrices procesadas.
- La librería OpenCV permite el procesamiento eficiente de los píxeles que conforman cada cuadro del video, la relación entre el algoritmo $\frac{serialoptimizado}{serialOpenCV}$ para el procesamiento del filtro Gaussiano es de 0.088 veces, para la conversión de color es de 1.015 veces, para la pirámide orientable es de 1.2 veces esto obtenido de los resultados del capítulo 4.
- La librería OpenCV-Cuda, integrada para el procesamiento en la tarjeta gráfica, se usó para procesar la etapa de filtro Gaussiano a un factor de 1.43 veces comparado contra el algoritmo serial. La etapa de amplificación que abarca la conversión de color y la reconstrucción del vídeo tiene un rendimiento de aceleración de 2.02 veces según los resultados obtenidos, la pirámide orientable tiene un factor de 2.98 veces el rendimiento del algoritmo acelerado contra el serial.

El uso de las tarjetas gráficas para acelerar el procesamiento de aplicaciones utilizándolas como coprocesadores es una realidad y está en su etapa inicial, las principales empresas fabricantes de GPU Nvidia y AMD ya tienen el hardware y el software disponible para aprovechar estas herramientas, solo es cuestión de explotar este potencial.

5.2 Trabajos a futuro

Las mejoras que se pueden llevar a cabo en el procesamiento del algoritmo de Magnificación de movimiento basado en la fase son:

- Realizar la descarga del algoritmo en sistemas embebidos (tarjeta Nvidia Jetson JTX, raspberry Pi, arduino, celulares, entre otros) para aplicar en sistemas de visión computacional.
- Incrementar el rendimiento del algoritmo en cuanto a la velocidad de procesamiento utilizando la memoria compartida del GPU, la memoria compartida se puede usar en OpenCV mediante la programación orientada a objetos de C++.
- Explorar la creación de kernel y funciones específicas para las operaciones de carga y descarga de los datos del vídeo.

- Integrar el hardware (vídeo cámara, Kinect o tecnologías similares) necesario para la aplicación del algoritmo mediante imágenes capturadas en tiempo real.

Bibliografía

[1] Bernal, J., Gómez, P., and Bobadilla, J. (1999). Una visión práctica en el uso de la transformada de fourier como herramienta para el análisis espectral de la voz. *Estudios de fonética experimental*, 10:75–105. 33

[2] Bradski, G. and Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library*. OReilly Media, Inc. 24

[3] Dawwd, S. A. and Salim, U. T. (2013). Gpu acceleration of object detection on video stream using cuda. In *Electrical, Communication, Computer, Power, and Control Engineering (ICECCPCE), 2013 International Conference on*, pages 198–203. IEEE. 8, 9, 10, 12

[4] García, G. B., Suarez, O. D., Aranda, J. L. E., Tercero, J. S., Gracia, I. S., and Enano, N. V. (2015). *Learning Image Processing with OpenCV*. Packt Publishing Ltd. 5

[5] Hartati, S., Harjoko, A., and Supardi, T. W. (2011). The digital microscope and its image processing utility. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 9(3):565–574. 18

[6] Lindholm, E., Nickolls, J., Oberman, S., and Montrym, J. (2008). Nvidia tesla: A unified graphics and computing architecture. *IEEE micro*, 28(2). 26

[7] Medel, J., Guevara López, P., and Flores Rueda, A. (2004). Caracterización de filtros digitales en tiempo real para computadoras digitales. *Computación y Sistemas*, 7(3):190–209. 1, 16

[8] Medina, González, D. V. (2010). Transformada de fourier en aplicación en el diseño de filtros digitales para el procesamiento de imágenes. 16, 34, 36, 38

[9] Monteiro, E., Vizzotto, B., Zatt, B., Bampi, S., et al. (2011). Applying cuda architecture to accelerate full search block matching algorithm for high performance motion estimation in video encoding. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2011 23rd International Symposium on*, pages 128–135. IEEE. 11

[10] Nickolls, J., Buck, I., Garland, M., and Skadron, K. (2008). Scalable parallel programming with cuda. *Queue*, 6(2):40–53. 25, 27

[11] Nvidia, C. (2010). Programming guide. 26

[12] Nvidia Corporation (Agosto de 2016). `www.nvidia.com/object/cuda_home.html`. 1, 25

[13] OpenCV (Noviembre de 2016). `www.opencv.org`. 24

[14] Pulli, K., Baksheev, A., Kornyakov, K., and Eruhimov, V. (2012). Real-time computer vision with opencv. *Communications of the ACM*, 55(6):61–69. 25, 39

[15] Sonka, M., Hlavac, V., and Boyle, R. (2014). *Image processing, analysis, and machine vision*. Cengage Learning. 26

[16] Steve, S. and SOYINKA, W. (2007). *Manual de administración de Linux*. MCGRAW-HILL. 24

[17] Şuşu, A. E., Codreanu, V., Evangelidis, G., and Petrică, L. (2016). *Efficient Implementation of a Video Change Detection algorithm*. In *Communications (COMM), 2016 International Conference on*, pages 77–82. IEEE. 7, 8

[18] Szeliski, R. (2010). *Computer vision: algorithms and applications*. Springer Science & Business Media. 25

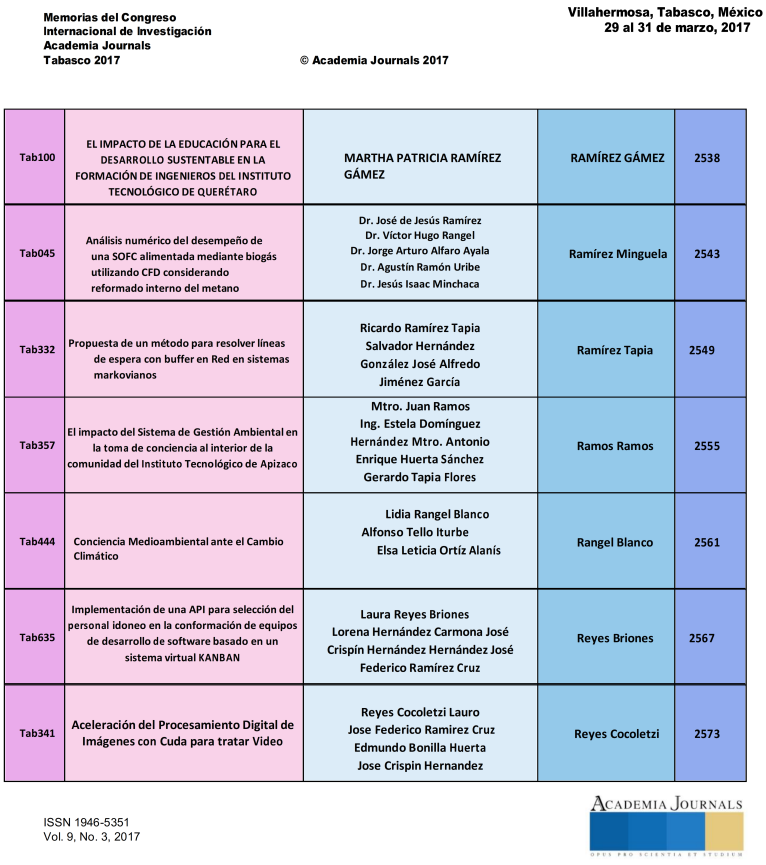
[19] Wadhwa, N., Rubinstein, M., Durand, F., and Freeman, W. T. (2013). Phase-based video motion processing. *ACM Transactions on Graphics (TOG)*, 32(4):80. 2, 10, 16, 17, 18, 20

[20] Wang, Y.-K. and Huang, W.-B. (2011). Acceleration of an improved retinex algorithm. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, pages 72–77. IEEE. 5, 6

[21] Wu, H.-Y., Rubinstein, M., Shih, E., Guttag, J., Durand, F., and Freeman, W. (2012). Eulerian video magnification for revealing subtle changes in the world. 10, 11, 13, 14, 15, 19, 20

Apéndice A

Publicaciones



Memorias del Congreso
Internacional de Investigación
Academia Journals
Tabasco 2017

Villahermosa, Tabasco, México
29 al 31 de marzo, 2017

© Academia Journals 2017

Tab100	EL IMPACTO DE LA EDUCACIÓN PARA EL DESARROLLO SUSTENTABLE EN LA FORMACIÓN DE INGENIEROS DEL INSTITUTO TECNOLÓGICO DE QUERÉTARO	MARTHA PATRICIA RAMÍREZ GÁMEZ	RAMÍREZ GÁMEZ	2538
Tab045	Análisis numérico del desempeño de una SOFC alimentada mediante biogás utilizando CFD considerando reformado interno del metano	Dr. José de Jesús Ramírez Dr. Víctor Hugo Rangel Dr. Jorge Arturo Alfaro Ayala Dr. Agustín Ramón Uribe Dr. Jesús Isaac Minchaca	Ramírez Minguela	2543
Tab332	Propuesta de un método para resolver líneas de espera con buffer en Red en sistemas markovianos	Ricardo Ramírez Tapia Salvador Hernández González José Alfredo Jiménez García	Ramírez Tapia	2549
Tab357	El impacto del Sistema de Gestión Ambiental en la toma de conciencia al interior de la comunidad del Instituto Tecnológico de Apizaco	Mtro. Juan Ramos Ing. Estela Domínguez Hernández Mtro. Antonio Enrique Huerta Sánchez Gerardo Tapia Flores	Ramos Ramos	2555
Tab444	Conciencia Medioambiental ante el Cambio Climático	Lidia Rangel Blanco Alfonso Tello Iturbe Elsa Leticia Ortíz Alanís	Rangel Blanco	2561
Tab635	Implementación de una API para selección del personal idoneo en la conformación de equipos de desarrollo de software basado en un sistema virtual KANBAN	Laura Reyes Briones Lorena Hernández Carmona José Crispín Hernández Hernández José Federico Ramírez Cruz	Reyes Briones	2567
Tab341	Aceleración del Procesamiento Digital de Imágenes con Cuda para tratar Video	Reyes Cocoletzi Lauro Jose Federico Ramirez Cruz Edmundo Bonilla Huerta Jose Crispin Hernandez	Reyes Cocoletzi	2573

ISSN 1946-5351
Vol. 9, No. 3, 2017

ACADEMIA JOURNALS

Figura A.1. Índice de la publicación en el congreso Internacional de investigación Academia Journals Tabasco 2017

ACELERACIÓN DEL PROCESAMIENTO DIGITAL DE IMÁGENES CON CUDA PARA TRATAR VIDEO

Lauro Reyes Cocoletzi[1], José Federico Ramírez Cruz[2],
Miguel Octavio Arias Estrada[3] y Edmundo Bonilla Huerta[4]

Resumen— **Actualmente las necesidades de procesamiento de información en tiempo real, en específico, en el campo de visión por computadora y la capacidad actual de la tecnología para proveer de herramientas para llevar a cabo dicho trabajo nos lleva a usar las tarjetas gráficas multinucleo (GPU Nvidia y su software CUDA) para incrementar la velocidad de procesamiento de los algoritmos.**

El procesamiento paralelo en los algoritmos de visión por computadora es necesario pues se trabaja con miles de imágenes para realiza operaciones que deben realizarse en lapsos de tiempo muy corto. En este trabajo se acelera un filtro gaussiano, y una conversión de color del espacio RGB a YIQ en el algoritmo denominado 'Procesamiento de video basado en la fase' que es usado para magnificar movimientos pequeños e imperceptibles en video.
Se logra acelerar el algoritmo en 2.5 veces al magnificar los movimientos de los videos procesados.

***Palabras clave*— Computo paralelo, kernel, espacio de color YIQ, filtros espaciales, filtros temporales, hilos y CUDA.**

INTRODUCCIÓN.

El algoritmo de Procesamiento de video basado en la fase es un algoritmo desarrollado en el Instituto Tecnológico de Massachusetts (Rubenstein et al. 2013), dicho algoritmo procesa videos donde existen movimientos tenues e imperceptibles a simple vista y los magnifica de tal forma que el video resultante muestra una amplificación de dichos movimientos.

En la figura 1 se muestra de forma general el funcionamiento del algoritmo en cuanto al procesamiento de los cuadros que corresponde a un video. Se logra observar la diferencia entre la entrada y la salida.

En la parte superior izquierda tenemos cuatro cuadros que corresponden al video de entrada, en la parte superior derecha tenemos la superposición de los cuadros que conforman a dicho video, se observa un patrón constante en la superposición de las imágenes, en la parte inferior izquierda tenemos cuatro cuadros procesados por el algoritmo y finalmente en la parte inferior derecha tenemos la superposición de todos los cuadros procesados por el algoritmo y se logra apreciar los cambios sutiles del movimiento magnificado (Wu et al. 2012).

Figura 1.- Procesamiento de magnificación de movimiento.

El funcionamiento del algoritmo de forma general procesa en el espacio-temporal los pixeles que conforman a cada

[1]Lauro Reyes Cocoletzi es Estudiante de la Maestría en Sistemas Computacionales en el Instituto Tecnológico de Apizaco, Tlaxcala lrc07reyes@gmail.com.
[2]Dr. José Federico Ramírez Cruz es Profesor de la Maestría en Sistemas Computacionales en el Instituto Tecnológico de Apizaco, Tlaxcala, México.
[3]El Dr. Miguel Octavio Arias Estrada es investigador en el Instituto Nacional de Astrofísica, Óptica y Electrónica San Andrés Puebla, México.
[4]El Dr. Edmundo Bonilla Huerta es Profesor (SNI) de la Maestría en Sistemas Computacionales en el Instituto Tecnológico de Apizaco, Tlaxcala, México.

Figura A.2. Publicación en el congreso Internacional de investigación Academia Journals Tabasco 2017, ISNN 1946-5351,vol. 9, No.3

Congreso Internacional de Investigación Academia Journals
Tabasco 2017

AcademiaJournals.com

CERTIFICADO

otorgado a

Reyes Cocoletzi Lauro
Jose Federico Ramirez Cruz
Edmundo Bonilla Huerta
Jose Crispin Hernandez Hernandez

por su artículo intitulado

Aceleración del Procesamiento Digital de Imágenes con Cuda para tratar Video
(Artículo No. Tab341)

el cual fue presentado en el Congreso desarrollado del 29 al 31 de marzo 2017 en Villahermosa, Tabasco, México, y publicado en el portal de Internet AcademiaJournals.com, con ISSN 1946-5351, Vol. 9 #3, 2017 y en el libro electrónico online con ISBN intitulado "Aplicación Del Saber: Casos y Experiencias", Vol. 3

Dr. Humberto José Cervera Brito
Director
Instituto Tecnológico de Villahermosa

Dr. Rafael Moras
Editor, Academia Journals
Profesor de Ing. Industrial, St. Mary´s University

Figura A.3. Certificado congreso Internacional de investigación Academia Journals Tabasco 2017

Table of Contents

Figura A.4. Índice de la Publicación en el X Congreso Internacional en Tecnologías Inteligentes y de la Información (CITII) Apizaco, Tlaxcala 2016. ISSN: 1870-4069, vol 128.

Avances en el desarrollo de un clasificador de imágenes termográficas de planta del pie diabético basado en una red neuronal de retropropagación.

Ramírez Cruz José Federico[1], Bonilla Huerta Edmundo[1], Reyes Cocoletzi Lauro[1], Hernández Hernández José Crispín[1]

[1] Instituto Tecnológico de Apizaco
Apizaco Tlaxcala, México.

Resumen. Actualmente el incremento de enfermedades crónicas (diabetes) y el diagnostico preventivo en el campo de la medicina nos llevan a usar la tecnología para prevenir complicaciones a largo plazo o incluso a predecir con un rango de confiabilidad el porcentaje de probabilidad a ser propensos a estas patologías. En la literatura médica se tiene una relación entre la temperatura de algunas zonas de las plantas del pie en relación a el flujo sanguíneo y el grado de azúcar en este, por lo que en este trabajo se presenta una propuesta para la implementación de una red neuronal (tipo retropropagación) que será capaz de realizar la clasificación de patrones de imágenes digitalizadas en base a la variación térmica e identificar patrones para la clasificación de angiosomas planares de individuos sanos y posibles individuos diabéticos no diagnosticados (a través de anormalidades en la lectura de la imagen térmica).La red neuronal de retropropagación implementada tiene la característica que pre-clasifica a través del almacenamiento de los pesos de los patrones de entrenamiento, para evaluar la mayor relación (a través de la correlación de Pearson) existente entre las distintas imágenes evaluadas lo que conlleva a un mayor grado de confiabilidad en la clasificación.

Palabras clave—Correlación de Pearson, red neuronal de retropropagación, angiosomas planares

Figura A.5. Publicación en el X Congreso Internacional en Tecnologías Inteligentes y de la Información (CITII) Apizaco, Tlaxcala 2016

Figura A.6. Certificado X Congreso Internacional en Tecnologías Inteligentes y de la Información (CITII) Apizaco, Tlaxcala 2016

Apéndice B

Estancias

Tonatzintla, Puebla, 20 de junio de 2016

INSTITUTO TECNOLOGICO DE APIZACO
MAESTRIA EN SISTEMAS COMPUTACIONALES

A quien corresponda

Por medio de la presente hacemos constar que el estudiante de maestría de su institución, Lauro Reyes Cocoletzi, fue aceptado para realizar una estancia de investigación en el Laboratorio de Cómputo Reconfigurable y de Alto Desempeño, de la Coordinación de Ciencias Computacionales, en la que el estudiante realizará investigación alrededor de su tema de maestría. La estancia será del 20 de junio al 22 de diciembre de 2016. En particular, se apoyará a la paralelización de sus algoritmos en GPU y la evaluación de las alternativas de paralelización.

Si requiere mayor información, por favor no dude en contactarme.

Atentamente

Dr. Miguel O. Arias Estrada
Investigador Titular B
Laboratorio de Cómputo Reconfigurable y de Alto Desempeño
Departamento de Ciencias Computacionales - INAOE
ariasmo@inaoep.mx
Tel: +52 222 266-3100 ext. 8316

INSTITUTO NACIONAL DE ASTROFISICA OPTICA Y ELECTRONICA
Coordinación de Ciencias Computacionales
Apdo. Postal 51 y 216, Puebla, Pue. 72000.
Tel. (222) 266-3100

Figura B.1. Carta de aceptación otorgada en la estancia de investigación

Mtro. Felipe Pascual Rosario Aguirre
Director
Instituto Tecnológico de Apizaco

Tonantzintla, Puebla, 14 de julio de 2017

ASUNTO: Carta de satisfacción

Estimado Mtro. Rosario Aguirre:

La presente es para informar que el estudiante de maestría **Lauro Reyes Cocoletzi**, del Instituto Tecnológico de Apizaco, realizó una estancia de trabajo en la Coordinación de Ciencias Computacionales del INAOE, con el tema Aceleración de Algoritmo de Procesamiento de Movimiento en Video Basado en la Fase. La estancia inició el 01 de julio de 2016 y terminó el 07 de abril de 2017. La estancia y trabajo del estudiante fue satisfactoria.

Si requiere información adicional, por favor no dude en contactarme.

Atentamente

Dr. Miguel O. Arias Estrada
Investigador Titular B
Laboratorio de Cómputo Reconfigurable y de Alto Desempeño
Departamento de Ciencias Computacionales - INAOE
ariasmo@inaoep.mx
Tel: +52 222 266-3100 ext 8316

INSTITUTO NACIONAL DE ASTROFISICA OPTICA Y ELECTRONICA
Coordinación de Ciencias Computacionales
Apdo. Postal 51 y 216, Puebla, Pue. 72000.
Tel. (222) 266-3100

Figura B.2. Carta de satisfacción otorgada en la estancia de investigación