Universidad Rey Juan Carlos

Máster en Visión Artificial

Escuela de Másteres Oficiales

Detección de Líneas en Vídeo
Comparación CPU vs GPU

Autores:

TAREF BILEL SEIFEDDINE
DANIEL CÁMARA NÚÑEZ

Asignatura: Herramientas Software para Tratamiento de Imágenes
Curso académico: 2025–2026

17 de diciembre de 2025

Índice

1. Introducción y objetivos **2**

2. Descripción general del software **2**
2.1. Arquitectura del proyecto 2
2.2. Flujo de ejecución 3

3. Métodos implementados **4**
3.1. Pipeline común de detección 4
3.1.1. Detección basada en líneas largas 4
3.1.2. Agrupamiento de líneas mediante clustering 4
3.2. Detección en la noche 4

4. Resultados **5**
4.1. Método 1: Detección basada en líneas largas 5
4.2. Método 3: Agrupamiento de líneas mediante clustering 5
4.3. Método 6: Detección en la noche 6

5. Conclusion **6**

6. Futuras líneas de trabajo **6**

1. Introducción y objetivos

La detección de líneas en secuencias de vídeo es una tarea fundamental en sistemas de visión por computador, especialmente en aplicaciones relacionadas con la conducción asistida y los sistemas inteligentes de transporte. En este contexto, resulta esencial disponer de soluciones que no solo sean precisas, sino también eficientes desde el punto de vista computacional, permitiendo su ejecución en tiempo real.

El objetivo de este trabajo es desarrollar un software modular para la detección de líneas en vídeo y analizar su rendimiento comparando distintas implementaciones ejecutadas en CPU y GPU. Para ello, se han implementado varios métodos basados en técnicas clásicas de procesamiento de imagen, como la detección de bordes, el filtrado por regiones de interés y la transformada de Hough, incorporando además estrategias de mejora visual y estabilidad temporal.

La memoria se centra en describir la estructura del software, las partes más relevantes de su implementación y los resultados obtenidos en términos de tiempo de procesamiento y fotogramas por segundo (FPS). De este modo, se evalúa el impacto de la aceleración por GPU y se justifica su uso en aplicaciones de visión por computador orientadas a tiempo real.

2. Descripción general del software

Esta sección describe la estructura general del software desarrollado, haciendo especial énfasis en su organización modular. El objetivo principal del diseño es facilitar la comparación entre diferentes métodos de detección de líneas y su ejecución tanto en CPU como en GPU, permitiendo además una fácil extensión y reutilización del código.

2.1. Arquitectura del proyecto

El software ha sido diseñado siguiendo una arquitectura modular, donde cada componente cumple una función específica dentro del flujo de procesamiento. Esta organización permite separar claramente la lógica de control, las funciones comunes, la aceleración por GPU y los distintos métodos de detección implementados.

A continuación se describen los principales módulos del proyecto:

El software ha sido diseñado siguiendo una arquitectura modular, donde cada componente cumple una función específica dentro del flujo de procesamiento. Esta organización permite separar claramente la lógica de control, las funciones comunes, la aceleración por GPU y los distintos métodos de detección implementados.

A continuación se describen los principales módulos del proyecto:

- **main.py**: módulo principal encargado de la ejecución del programa. Gestiona la carga del vídeo, la selección del método de detección, la medición del tiempo de procesamiento y la comparación del rendimiento entre CPU y GPU, así como la visualización de los resultados.
- **common.py**: contiene funciones comunes utilizadas por todos los métodos, como el redimensionado de imágenes, la definición de la región de interés (ROI), el filtrado de líneas detectadas y las funciones de medición de tiempo por fotograma.
- **common_gpu.py**: incluye las implementaciones de los kernels CUDA desarrollados mediante Numba. En este módulo se encuentran las operaciones aceleradas por GPU, como la conversión a escala de grises y el suavizado de imágenes, que representan las etapas más costosas del preprocesado.
- **metodoX.py**: conjunto de módulos que implementan los distintos métodos de detección de líneas. Cada archivo corresponde a un enfoque diferente y contiene dos versiones del procesamiento: una ejecutada en CPU y otra que combina CPU y GPU.

2.2. Flujo de ejecución

El diagrama 1 muestra el flujo de ejecución:

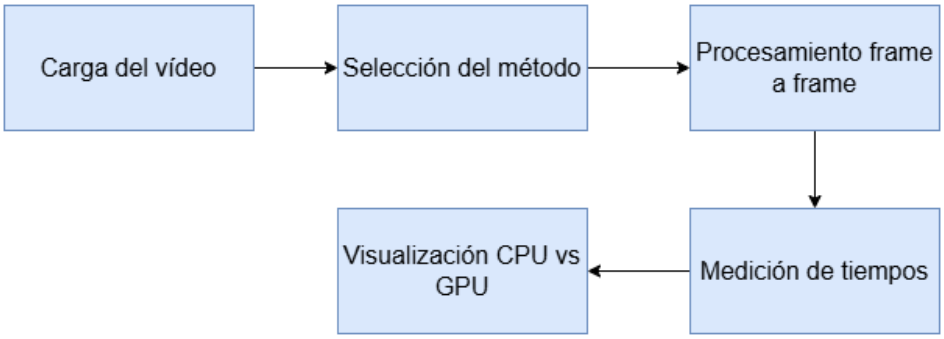


Figura 1: el flujo de ejecución del *Software*

3. Métodos implementados

En este trabajo se han implementado distintos métodos de detección de líneas basados en técnicas clásicas de procesamiento de imagen. Todos los métodos comparten un pipeline común, diferenciándose principalmente en las estrategias empleadas para mejorar la robustez de la detección, la estabilidad temporal y la adaptación a distintas condiciones de iluminación.

Cada método ha sido desarrollado en dos versiones: una ejecutada íntegramente en CPU y otra que combina CPU y GPU, con el objetivo de analizar el impacto de la aceleración por hardware en las etapas más costosas del procesamiento.

3.1. Pipeline común de detección

Independientemente del método utilizado, el procesamiento de cada fotograma sigue una secuencia común de etapas. En primer lugar, la imagen se redimensiona y se convierte a escala de grises, aplicándose posteriormente un suavizado para reducir el ruido. A continuación, se realiza la detección de bordes mediante el algoritmo de Canny y se limita el área de análisis mediante una región de interés (ROI), centrada en la parte inferior del fotograma.

Sobre esta región se aplica la transformada de Hough probabilística para la detección de segmentos de línea. Finalmente, las líneas detectadas se filtran en función de su pendiente, descartando aquellas que no corresponden a marcas viales relevantes. Este pipeline base permite mantener una estructura homogénea y facilitar la comparación entre métodos.
Lo que sigue a este parrafo se centrara en los **tres** metodos mas importantes utlizados en este trabajo

3.1.1. Detección basada en líneas largas

Este método constituye el enfoque base del sistema y se apoya directamente en la transformada de Hough para la detección de líneas. Tras el filtrado por pendiente, las líneas detectadas se ordenan según su longitud, seleccionando únicamente las más largas para su visualización.
El objetivo de este enfoque es reducir el ruido producido por detecciones espurias y centrarse en aquellas líneas que representan con mayor probabilidad las marcas viales principales. Este método ofrece buenos resultados en condiciones de iluminación normales y sirve como referencia para evaluar el resto de enfoques implementados.

3.1.2. Agrupamiento de líneas mediante clustering

Este método introduce una etapa adicional de agrupamiento sobre las líneas detectadas. A partir de la pendiente de cada segmento, se aplica un algoritmo de clustering para separar las líneas en dos grupos, correspondientes a los carriles izquierdo y derecho.

Esta estrategia permite una representación más clara y estructurada de la escena, facilitando la identificación de ambos carriles de forma diferenciada. Aunque el coste computacional es ligeramente superior, el impacto visual y la estabilidad de la detección mejoran notablemente.

3.2. Detección en la noche

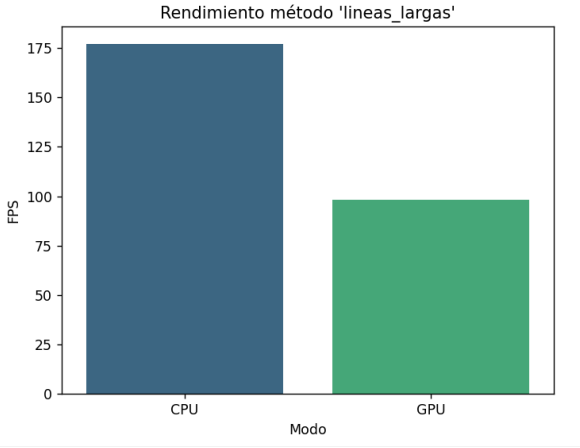
Este método está orientado a la detección de líneas en escenarios nocturnos o de baja iluminación, donde los métodos convencionales presentan un rendimiento reducido. Para ello, se emplea un filtrado por color en el espacio HSV, centrado en las marcas viales blancas y amarillas, junto con una región de interés que limita el área de análisis.

La detección de bordes se realiza utilizando umbrales más bajos, permitiendo identificar líneas con menor contraste. Posteriormente, las líneas se extraen mediante la transformada de Hough y se filtran según su pendiente.

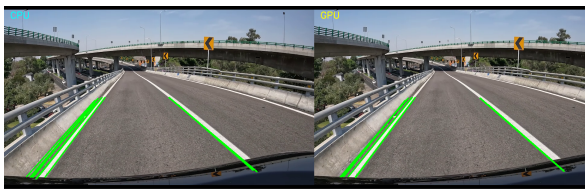
4. Resultados

En esta sección se presentan los resultados obtenidos en el trabajo, organizados según los métodos aplicados, y se muestran algunos de los resultados más representativos correspondientes a los métodos considerados más relevantes.

4.1. Método 1: Detección basada en líneas largas



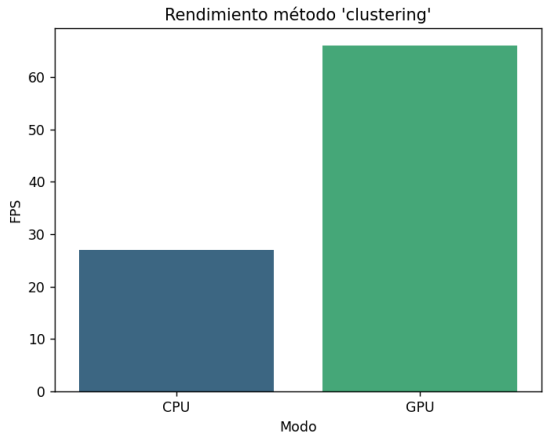
(a)



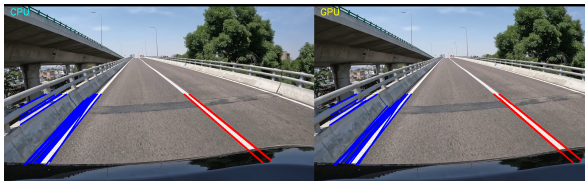
(b)

Figura 2: Comparación visual del método `lineas_largas` ejecutado en CPU y GPU

4.2. Método 3: Agrupamiento de líneas mediante clustering



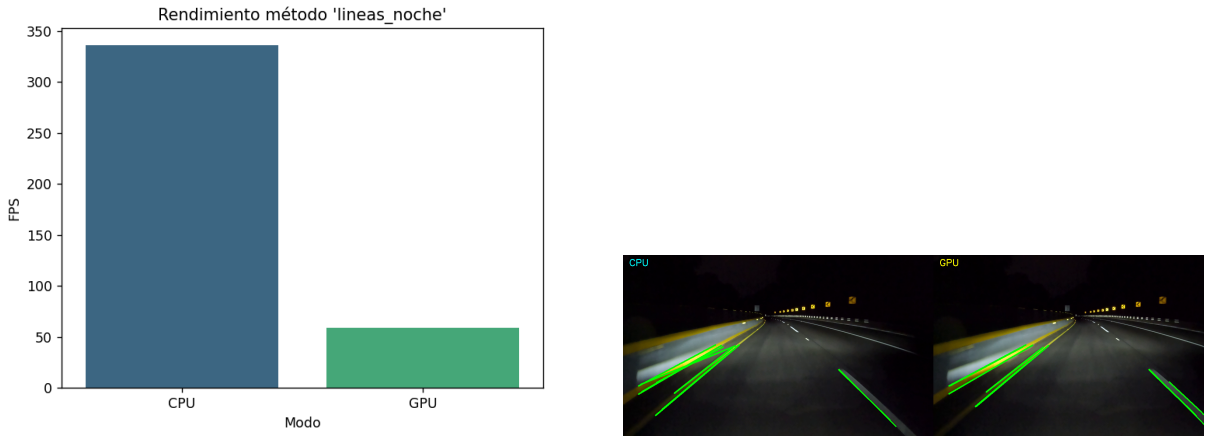
(a)



(b)

Figura 3: Comparación visual del método `Clustering` ejecutado en CPU y GPU

4.3. Método 6: Detección en la noche



(a) (b)

Figura 4: Comparación visual del método `en la noche` ejecutado en CPU y GPU

5. Conclusion

En este trabajo se ha desarrollado un sistema modular de detección de líneas en vídeo, comparando su ejecución en CPU y GPU. Los resultados obtenidos muestran que la implementación en CPU ofrece un mejor rendimiento en términos de FPS que la versión acelerada por GPU. Esto se debe principalmente a que solo una parte del pipeline ha sido paralelizada, mientras que las operaciones más costosas se ejecutan en CPU, además del sobrecoste asociado a la transferencia de datos entre CPU y GPU. Estos resultados evidencian que la aceleración por GPU debe aplicarse de forma selectiva para resultar efectiva en sistemas de visión por computador en tiempo real.

6. Futuras líneas de trabajo

Como trabajo futuro, se propone ampliar el sistema mediante la implementación de un pipeline completamente acelerado en GPU, incorporando versiones paralelas de las etapas más costosas del procesamiento, como la detección de bordes y la transformada de Hough. Asimismo, se podría mejorar la eficiencia del sistema reduciendo el coste de transferencia de datos entre CPU y GPU.
Por otro lado, resulta de interés explorar la integración de técnicas basadas en aprendizaje profundo para la detección de carriles, con el objetivo de aumentar la robustez del sistema frente a variaciones de iluminación, ruido y condiciones ambientales adversas. Finalmente, la adaptación del software a plataformas embebidas permitiría evaluar su viabilidad en entornos reales de tiempo real.