Universidad Nacional Autónoma de México

Facultad de Ingeniería

Monitoreo de flujo vehicular en tiempo real

Tesis

Que para obtener el título de

Ingeniero en Computación

Presentan:

Ruiz Santiago Sócrates

Trejo Madrigal Josué Abenamar

Director de tesis:

Dr. Escalante Ramírez Boris

2014

Agradecimientos

En primer lugar quiero agradecer a mi familia por haberme apoyado en todo momento durante el transcurso de mis estudios. A mi padre Néstor por haberme dado los ejemplos a seguir para ser una persona responsable y consciente de su entorno en la familia y sociedad. A mi madre por su motivación, sus valores y su amor que siempre me ha manifestado. A mi hermana Xóchitl por sus consejos y cuidados. A mi hermana Natzul por sus ejemplos de valentía y perseverancia. A mi hermano Appa por haber sido mi amigo y escucharme en los momentos difíciles.

A mis profesores por sus enseñanzas y por compartir sus experiencias en el trabajo y en la vida. En especial al Dr. Boris Escalante, por su paciencia, su conocimiento y sus consejos.

A mis compañeros de generación por haber sido un equipo con el cual contar en toda la carrera. A Juan Carlos, Leopoldo, Noé y Edwin por sus ánimos y su amistad. A Josué por su espíritu emprendedor y su colaboración durante este tiempo en el trabajo.

Al proyecto PAPIIT IG100814 por su apoyo en este trabajo.

Por último quiero agradecer a la Universidad Nacional Autónoma de México, en especial al Centro de Enseñanza de Lenguas Extranjeras y a la Facultad de Ingeniería, por haberme otorgado las herramientas necesarias para continuar desarrollándome personal y profesionalmente.

Sócrates Ruiz Santiago

Agradecimientos

A mi familia, por su incondicional apoyo en esta etapa de mi vida. A mis padres, Graciela y Fernando, que con su ejemplo y dedicación me han permitido lograr mis objetivos y perseguir nuevos retos. A mi hermano, Fernando, que ha estado presente en los momentos importantes. A mi novia, Pris, por acompañarme y escucharme en todo momento.

A mis amigos, Noé, Miguel, Marcos, Adrián, Jesús, Juan José, Yazmín, Andreś, Gustavo, Raúl, Javier, Josué, Carlos, y muchos más, por todas las experiencias vividas, y los recuerdos agradables. A Sócrates, por todo el tiempo compartido, y por su colaboración en este proyecto.

A la Universidad Nacional Autónoma de México, por darme la oportunidad de pertenecer a la comunidad universitaria, y poner a mi disposición todos sus recursos.

A mis profesores, por brindarme sus conocimientos y experiencias.

Al Dr. Boris Escalante, por su apoyo, sus consejos y su paciencia.

Al Laboratorio de Procesamiento Digital de Imágenes, por la oportunidad de realizar este proyecto.

Al proyecto PAPIIT IG100814 por el apoyo obtenido.

Josué Abenamar Trejo Madrigal

Índice General

- **Introducción. 1**
  - Descripción del problema. 1
  - Objetivos. 2
  - Justificación. 2
  - Alcances. 3
  - Propuesta. 4

1. Antecedentes y actualidad. 6

1.1. Geolocalización. 6

1.2. Monitoreo de flujo vehicular. 7

2. Arquitecturas de cómputo. 12

2.1. Arquitectura secuencial. 13

2.2. Arquitectura en paralelo. 16

2.3. Elementos de procesamiento. 18

3. Tecnologías de cómputo gráfico y concurrente. 20

3.1. Compute Unified Device Architecture (CUDA). 20

3.2. OpenCV. 22

3.3. Intel IPP. 23

3.4. Integración de tecnologías. 24

4. Detección de objetos. 26

4.1. Estado del arte. 26

4.2. Algoritmos de detección de objetos. 28

4.2.1. Haar Feature-based Cascade Classifier. 31

4.2.2. Histogram of Oriented Gradients (HOG). 33

5. Estimación de movimiento. 35

5.1. Estado del arte. 35

5.2. Algoritmos de estimación de movimiento. 36

5.3. Block Matching. 38

5.3.1. Métodos de Block Matching. 39

5.3.2. Comparativa entre métodos. 45

6. Monitoreo de flujo vehicular. 48

6.1. Detección de vehículos. 48

6.2. Estimación de movimiento. 51

6.3. Monitoreo de flujo vehicular. 56

6.4. Resultados. 68

7. Conclusiones y proyección a futuro. 89

- **Referencias. 94**

Introducción

La Ciudad de México es considerada una de las urbes más grandes del mundo, además de representar uno de los lugares de mayor afluencia en México, tanto de sus mismos habitantes como de las personas que diariamente visitan sus calles.

Existen muchas opciones para trasladarse dentro de la ciudad: a pie, bicicleta, transporte público, automóvil particular, entre otras. Esta diversidad de opciones puede generar la impresión de que circular en la urbe es algo sencillo, sin embargo, debe considerarse la enorme cantidad de personas que se mueven dentro de la ciudad día con día, la mayor parte de ellas utilizando su propio automóvil, dificultando la posibilidad de transitar libre y velozmente.

La circulación de vehículos representa una fuente importante de problemas de la Ciudad de México, generando situaciones como embotellamientos, choques, así como daño a personas o a la misma ciudad. Este tipo de conflictos son muy frecuentes, además de inesperados, lo cual impide evitarlos adecuadamente, ya que pueden presentarse en cualquier punto; sin embargo, es posible tomar medidas a fin de reducir la cantidad de incidencias y disminuir los problemas derivados de alguna de ellas.

Descripción del problema

Este documento propone realizar el monitoreo del flujo vehicular en las calles de la Ciudad de México como medida para analizar y recibir alertas ante cualquier tipo de conflicto vehicular, a fin de contrarrestar las problemáticas que puedan surgir a partir del mismo. Dicho monitoreo debe ser realizado en tiempo real.

Objetivos

La realización de este proyecto conlleva los siguientes objetivos:

- Identificar los elementos y factores que componen el flujo vehicular.

- Analizar los algoritmos de detección de objetos y estimación de movimiento a fin de determinar cuáles son los más adecuados para ser aplicados a la problemática existente.

- Realizar un sistema de monitoreo de flujo vehicular que implemente los algoritmos seleccionados y posea un rendimiento en tiempo real.

Justificación

Actualmente existen soluciones que permiten conocer el estado del flujo vehicular mediante el uso de diversas herramientas, sin embargo, tales soluciones dependen en gran medida de la retroalimentación de sus usuarios, ya sea a través de reportes de tráfico o mediante datos que pueden ser obtenidos al utilizar un dispositivo de geolocalización; por lo que requieren de una participación activa de sus usuarios en todo momento.

La solución contenida en este documento busca no solamente robustecer la cantidad de opciones disponibles, sino proponer como una alternativa a la información proporcionada por los usuarios, el utilizar como fuente de datos el flujo de vídeo obtenido de una videocámara, lo cual brinda la posibilidad de automatizar el proceso de monitoreo.

Alcances

Realizar un monitoreo de flujo vehicular en la ciudad implica considerar una amplia gama de factores, tanto de la misma estructura de las calles como de las condiciones ambientales presentes en el momento del monitoreo. Es por ello que resulta importante delimitar estos factores a fin de establecer un marco de trabajo que será utilizado a lo largo de este documento.

Se proponen las siguientes condiciones:

- El monitoreo se realizará para la vialidad denominada Circuito Interior, en la Ciudad de México.

- El óptimo funcionamiento del monitoreo será en condiciones de luz con buena iluminación, sea un día soleado o nublado ligeramente, así como condiciones climáticas sin lluvia, cubriendo hasta vientos ligeros.

La selección de Circuito Interior como vialidad monitoreada es debido a la gran cantidad de vehículos que circulan diariamente y de forma contante en esta vía. Por otro lado, el ruido que puede generar la presencia de lluvia o poca iluminación en las imágenes obtenidas por la videocámara, no permite asegurar el adecuado funcionamiento del monitoreo bajo estas condiciones, por lo que se han descartado estos escenarios.

Propuesta

Este documento contiene una propuesta de solución, esquematizada con la siguiente estructura:

1. Antecedentes y actualidad.

Este capítulo contiene información relativa a los sistemas e iniciativas existentes con respecto a monitoreo de flujo vehicular, considerando cualquier tipo de tecnología o plataforma.

2. Arquitecturas de cómputo.

Este capítulo realiza un planteamiento respecto a la arquitectura secuencial y en paralelo, a fin de establecer las ventajas o desventajas de cada una, así como los mejores usos para cada enfoque.

3. Tecnologías de cómputo gráfico y concurrente.

Este capítulo muestra el panorama general de algunas tecnologías de cómputo para el manejo de datos orientado a gráficos y programación concurrente. Contiene información sobre tecnologías como CUDA, OpenCV e Intel IPP.

4. Detección de objetos.

Este capítulo contiene la teoría respectiva a la detección de objetos, además de la descripción y funcionamiento de los principales algoritmos, tales como *Haar Feature-based Cascade Classifier* y *Histogram of Oriented Gradients*.

5. Estimación de movimiento.

Este capítulo contiene la teoría respectiva a la estimación de movimiento, así como los algoritmos relacionados, destacando el algoritmo de *Block Matching*. Asimismo, se indican los principales métodos asociados a *Block Matching*, y se realiza una comparativa entre éstos a fin de seleccionar aquellos que serán analizados más a detalle durante el desarrollo del sistema.

6. Monitoreo de flujo vehicular.

Este capítulo contiene el desarrollo del sistema, destacando la toma de decisiones en cada aspecto del sistema, desde la selección de los algoritmos a utilizar, hasta la integración de las componentes para obtener información de entrada, realizar el análisis correspondiente y generar la salida que indique el estado del flujo vehicular.

7. Conclusiones y proyección a futuro.

Este capítulo incluye las conclusiones respecto a las tecnologías y algoritmos utilizados, así como de los resultados obtenidos. Asimismo, contiene una propuesta de crecimiento de la aplicación, a un sistema completo de monitoreo de flujo vehicular.

Antecedentes y actualidad

El continuo avance del conocimiento ha permitido la creación y surgimiento de nuevas tecnologías, además de facilitar el acceso a las mismas, abaratando costos y dando paso a la producción de diversos productos de calidad y a precios cada vez más bajos. Esta tendencia ha provocado un aumento considerable en la cantidad de personas que disponen de bienes que eran considerados de lujo y sólo al alcance de muy pocos, tales como vehículos, computadoras o dispositivos móviles; generando con ello no solamente nuevas formas de bienestar, sino también nuevas necesidades y problemáticas.

Geolocalización

La necesidad de las personas de saber en dónde se encuentran o cómo llegar a un lugar en específico no es algo nuevo, sin embargo, con el auge de los teléfonos móviles con tecnologías de geolocalización integradas (GPS o servicios de localización asistida vía Internet) surgieron muchas soluciones enfocadas al uso de mapas que contaban con la capacidad de mostrar la ubicación de la persona en tiempo real. Posteriormente, estos sistemas añadieron una serie de funcionalidades más avanzadas, tales como indicar la forma de llegar a un lugar del cual se conoce su dirección o su ubicación expresada en latitud y longitud.

La información manejada por sistemas de geolocalización no es solamente de lectura para los usuarios, ya que además puede obtener retroalimentación de los mismos, lo cual ha dado pie a generar información adicional y de gran valor; por ejemplo, la generación de reportes de tráfico en tiempo real.

Monitoreo de flujo vehicular

Actualmente, existe una diversidad de soluciones enfocadas a dar a conocer a las personas el estado del flujo vehicular en las vialidades de las principales ciudades. Cada solución cuenta con un enfoque propio, aunque todas se caracterizan por la necesidad indispensable de intervención humana para su funcionamiento, ya sea para llevar a cabo el monitoreo, o para generar la información necesaria para ello.

Algunos de los principales exponentes de estos sistemas son:

Google Maps

Es un servicio que forma parte de la plataforma de Google. Consiste en un servidor de aplicaciones de mapas, que ofrece información a una aplicación web y a diversas aplicaciones móviles. [1]

Ofrece diversos servicios al público en general: mapas, búsqueda de lugares por coordenadas y direcciones, generación de rutas, imágenes satelitales, estado del clima y estado del tráfico.

El servicio que ofrece Google Maps para conocer el estado del tráfico funciona en diversas ciudades, e indica mediante un código de colores el estado de las principales vialidades de cada ciudad. Este código utiliza el color rojo para representar retrasos significativos en la vialidad, amarillo para problemas menores, y verde para indicar que no existe ningún tipo de congestión. El color gris indica que no existen datos disponibles. [2]

El funcionamiento del servicio de estado del tráfico es en tiempo real, aunque también puede realizar predicciones si cuenta con datos históricos suficientes, y su principal fuente de información son los mismos usuarios, en particular aquellos que utilizan la aplicación de Google Maps para dispositivos móviles, ya que la información de su ubicación y velocidad (con el consentimiento explícito del usuario) es retransmitida a los servidores de Google, lo cual permite generar esquemas de tráfico para cada vialidad. [3]

El servicio de estado del tráfico funciona actualmente en México desde 2012.

Nokia Here Maps

Es un servicio que ofrece Nokia para dispositivos móviles, aunque también existe una aplicación web. Ofrece servicios de mapas, búsqueda de ubicaciones, generación de rutas, mapas satelitales, así como estado del tráfico. [4]

El servicio de estado del tráfico es similar al de Google Maps, aunque posee una base de información incluso más amplia, ya que los datos que recibe para generar el estado del tráfico no son solamente por parte de sus usuarios, sino que en algunos países cuenta incluso con el apoyo de servicios de entrega de paquetería, tales como UPS y Fedex. Este servicio funciona en México desde 2012. [5]

Actualmente, la división de móviles de Nokia fue adquirida por Microsoft, pero productos como Here Maps no forman parte de dicha compra. [6]

Waze

Es una aplicación móvil de tráfico y navegación desarrollada por Waze Mobile. Cuenta con una gran comunidad de usuarios que continuamente aportan información al servidor, reportando estados de tráfico, congestionamiento, accidentes, y cualquier tipo de situación que afecte el estado de las vialidades; todo ello reportado y mostrado a otros usuarios en tiempo real. [7]

Esta aplicación permite conocer el estado del tráfico, pero además posee información de lugares relevantes para los automovilistas, tales como estacionamientos o gasolineras. La aplicación se encuentra disponible para plataformas como Android, iOS y Windows Phone 8.

Actualmente, Waze fue adquirido por Google, que posee Google Maps. Google ha manifestado que ambos proyectos continuarán como aplicaciones independientes, aunque poco a poco se ha ido incorporando información de Waze en Maps, tal como el estado de algunas vialidades, o determinados reportes de incidencias en las calles: accidentes, bloqueos, desastres, etc. [8]

072 Atención Ciudadana

Es un servicio que ofrece la Secretaría de Obras del Gobierno del Distrito Federal a los ciudadanos de la Ciudad de México, a través del cual pueden realizar solicitudes de información o de atención a quejas respecto a problemas de la ciudad. Este servicio puede ser solicitado a través de vía telefónica, Facebook, Twitter, o incluso mediante una aplicación móvil. [9]

A través de este servicio es posible solicitar información relativa al estado del tráfico de algunas vialidades de la ciudad. Esta información se visualiza a través de un mapa en un sitio web, el cual es generado por los reportes de los ciudadanos, o de las autoridades. Este mapa contiene todas las incidencias ocurridas en la ciudad, siendo un servicio disponible únicamente para el Distrito Federal.

Mediante este servicio es posible identificar problemas que afectan al flujo vehicular, tales como accidentes, manifestaciones, embotellamientos, entre otras afectaciones. [10]

Actualmente este servicio es gestionado por la Agencia de Gestión Urbana (AGU) del Distrito Federal.

Sistema Inteligente de Movilidad

Es un sistema inteligente de control y monitoreo vehicular implantado en la ciudad de Medellín, en Colombia. El objetivo de este sistema es potenciar el control del tráfico en la ciudad y ofrecer mejores alternativas de movilidad a los ciudadanos, utilizando para ello los recursos que ofrecen las tecnologías de la información. [11]

Este sistema se compone de elementos tales como: fotomultas, circuito cerrado de televisión, paneles informativos con el estado del tráfico, así como el uso de semáforos inteligentes.

El monitoreo de flujo vehicular se basa en el uso de cámaras de vídeo, conectadas a un circuito cerrado de televisión, las cuales son monitoreadas por personal capacitado perteneciente a una entidad denominada Centro de Control de Tránsito.

El equipo de trabajo del Centro de Control de Tránsito genera reportes del estado de las vialidades utilizando software especializado. Entre la información generada en los reportes se encuentra: conteo y clasificación de vehículos, medición de velocidades por carriles, ocupación, tiempo entre vehículos, congestión vehicular, entre otros.

Arquitecturas de cómputo

La arquitectura de cómputo representa la estructura, funcionamiento y diseño con que es construida una computadora, e involucra aspectos físicos (*hardware*) y lógicos (*software*).

La tecnología en general, y de forma muy particular la computación, son áreas que se han desarrollado y evolucionado de forma muy rápida, resolviendo diversidad de problemas, y a su vez, dando surgimiento a otros. Esto ha derivado en la creación de una serie de arquitecturas de cómputo para la resolución de dichos problemas.

Un elemento importante para comprender la forma de trabajar de las arquitecturas de cómputo es la denominada Taxonomía de Flynn [1].

La Taxonomía de Flynn es un esquema de clasificación de arquitecturas propuesto por Michael Flynn en 1966. Esta clasificación considera especialmente características como el número de flujos o secuencias de instrucciones y de datos.

Dentro de esta clasificación podemos encontrar los siguientes esquemas:

- *Single Instruction, Single Data* (SISD).
  Consiste en un flujo único de instrucciones y uno de datos. Solamente se procesa una operación a la vez.

- *Single Instruction, Multiple Data* (SIMD).
  Consiste en un flujo de instrucciones para varios flujos de datos. Esto representa realizar la misma operación sobre diversos conjuntos de datos de forma simultánea.

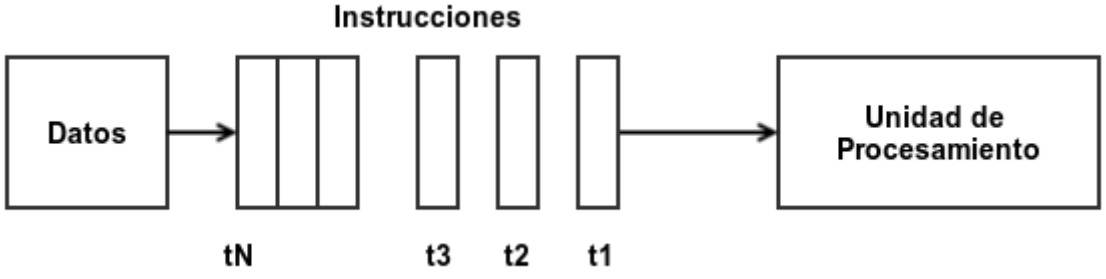
- *Multiple Instruction, Single Data* (MISD).
  Consiste en varios flujos de instrucciones que operan sobre un mismo flujo de datos. Es un modelo conceptual muy poco utilizado en la realidad.

- *Multiple Instruction, Multiple Data* (MIMD).
  Consiste en varios flujos de instrucciones operando sobre múltiples flujos de datos. Es el esquema más completo, y permite la ejecución concurrente de diversos procesos.

Arquitectura secuencial

La arquitectura del tipo secuencial se basa en el esquema SISD de la Taxonomía de Flynn. Es la arquitectura más relevante en la historia de la computación, ya que desde las primeras generaciones de computadoras se trabaja utilizando este modelo. [2]

Esta arquitectura consiste en un conjunto de instrucciones que son ejecutadas una a una por un elemento denominado unidad central de procesamiento; hasta que una instrucción termina es que puede ejecutarse la siguiente. Es debido a este esquema que la arquitectura es llamada secuencial, ya que consiste en una secuencia de instrucciones.



Funcionamiento de la arquitectura secuencial

Siendo esta arquitectura la predominante, los lenguajes de programación que fueron surgiendo también adoptaron este modelo de ejecución, condicionando el rendimiento de las aplicaciones realizadas al hecho de aprovechar al máximo la memoria y la capacidad de la unidad central de procesamiento.

La arquitectura secuencial de forma general define el modo de procesar datos e instrucciones, siendo realmente la especificación de esta arquitectura, planteada por John Von Neumann, la que ha dominado en las generaciones del cómputo [3].

La arquitectura de Von Neumann plantea un modelo consistente en los siguientes elementos:

- Memoria.
  Consiste en un conjunto de bloques de almacenamiento, que pueden guardar tanto datos como instrucciones. Cada bloque puede ser direccionado unívocamente.

- Unidad central de procesamiento.
  Es el elemento encargado del procesamiento, y se compone a su vez de los siguientes subelementos:

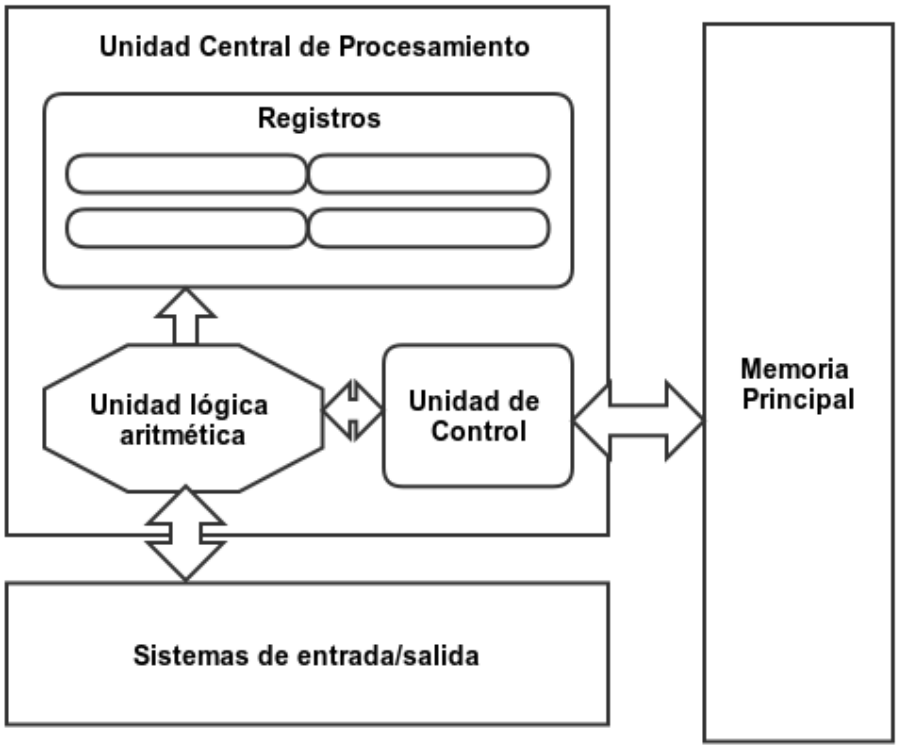
  - Unidad de control.
    Se encarga de llevar el control sobre las instrucciones a ser ejecutadas.

  - Unidad lógica aritmética.
    Realiza el procesamiento de instrucciones a través de operaciones aritméticas.

- Registros.
  Consiste en una serie de espacios de memoria de rápido acceso.

- Buses de comunicación
  Consiste en los elementos de comunicación entre la memoria y la unidad de procesamiento. Se trata de un conjunto de cables y un controlador que regula el flujo de transmisión de información.

Esta arquitectura queda estructurada de la siguiente forma:



Esquema de la arquitectura de Von Neumann

Con la evolución de la tecnología, los componentes de procesamiento han adquirido mejoras en lo que respecta a frecuencia de reloj, memoria caché, tamaño de palabra y las memorias de más capacidad. Con todos estos avances, las aplicaciones han llegado a poseer un mejor rendimiento, sin embargo esto es debido solamente a mejoras en el hardware, ya que la implementación de los programas sigue siendo secuencial, lo cual no permite aprovechar al máximo las capacidades de los componentes.

Arquitectura en paralelo

La arquitectura secuencial es fundamental en el cómputo, pero el surgimiento de nuevas necesidades y problemáticas ha convertido a esta arquitectura en una limitante, lo cual ha dado pie a la búsqueda de nuevas y mejores soluciones.

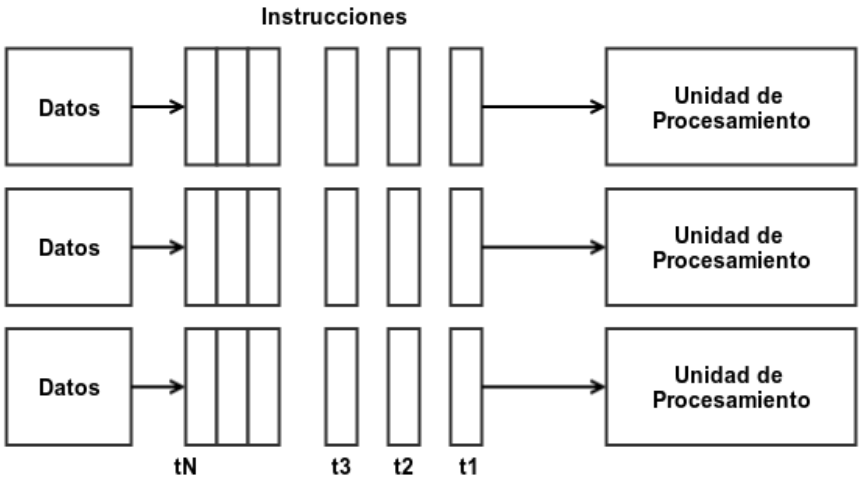
Un aspecto que ha sido relevante en este proceso ha sido la evolución de los sistemas operativos, principalmente el gran salto que representó pasar de sistemas monotarea a las implementaciones multitarea; aunque la realidad es que en sus inicios, a pesar de ser denominados multitarea, se trataba únicamente de sistemas secuenciales, diferenciados por el hecho de que trabajaban otorgando ráfagas de tiempo de procesamiento a cada tarea, lo cual daba la sensación de tener distintos procesos trabajando simultáneamente, siendo que se ejecutaba un solo proceso a la vez, pero alternando rápidamente con otros.

Dentro de la Taxonomía de Flynn encontramos el modelo *Single Instruction, Multiple Data* (SIMD), el cual refiere un esquema de un flujo de instrucciones que son aplicadas a un conjunto de flujos de datos de forma simultánea. Este esquema en la práctica ha resultado en unidades de procesamiento con una unidad de control y múltiples unidades lógicas aritméticas. [4]

Debido a estos nuevos planteamientos, surgieron conceptos como el paralelismo a nivel de hilo (*Thread Level Parallelism* o TLP), que propone incrementar el número de hilos (subprocesos) que una unidad de procesamiento pueda ejecutar simultáneamente, y el paralelismo a nivel instrucción (*Instruction Level Parallelism* o ILP), que propone incrementar la utilización de los recursos en ejecución. Sin embargo, estas implementaciones no resultaron ser por completo soluciones de procesamiento en paralelo, pues las instrucciones y procesos siguen corriendo bajo una misma unidad de procesamiento.

El esquema más complejo perteneciente a la Taxonomía de Flynn es *Multiple Instruction, Multiple Data* (MIMD). Este modelo especifica una serie de flujos de instrucciones, aplicadas a múltiples flujos de datos. Esto implica un conjunto de procesos independientes siendo ejecutados de forma simultánea, es decir, verdaderamente un procesamiento en paralelo.

La aplicación real del modelo MIMD ha derivado en sistemas con múltiples unidades de procesamiento independientes, las cuales ejecutan sus procesos de forma asíncrona.



Funcionamiento de la arquitectura en paralelo

El modelo MIMD especifica dos variantes en lo que respecta al uso de memoria:

- Memoria compartida.
  Esta variante especifica un componente de memoria único, el cual es compartido por todas las unidades de procesamiento. En este esquema es fundamental controlar los accesos a la memoria de forma simultánea, ya que si los mismos datos son accedidos por distintos procesadores al mismo tiempo, esto puede generar la corrupción en los mismos.

- Memoria distribuida.
  Esta variante propone componentes de memoria independientes para cada unidad de procesamiento. Este esquema no presenta los problemas de acceso múltiple, ya que cada unidad de procesamiento trabaja solamente con su memoria, pero requiere un fuerte control sobre la sincronización de los datos.

Cada variante de este modelo presenta ventajas y desventajas, por lo que no puede considerarse que una sea mejor a la otra. El uso dependerá del propósito del sistema o de las aplicaciones a ejecutar.

Elementos de procesamiento

Las arquitecturas de cómputo planteadas a lo largo de la historia siempre han considerado como elemento principal a la unidad de procesamiento. Esto es debido a que esta componente es la encargada de realizar la ejecución de las instrucciones, lo cual es el objetivo primario de un sistema de cómputo.

La principal componente utilizada como unidad de procesamiento es la Unidad Central de Procesamiento (*Central Processing Unit* o CPU).

La CPU es una componente fundamental en cualquier arquitectura de cómputo. Se trata de la componente central de procesamiento, y se encarga de ejecutar los procesos e instrucciones de propósito general.

Otra componente de procesamiento que ha cobrado relevancia mayormente en las arquitecturas en paralelo es la Unidad de Procesamiento Gráfico (*Graphics Processing Unit* o GPU). La GPU es una componente destinada de origen al procesamiento de gráficos y operaciones de punto flotante.

El fundamento de la creación de la GPU es aligerar la carga de procesamiento de la CPU; sin embargo, los avances en la creación de GPUs, y en particular el surgimiento de tecnologías para programación de estas componentes, han permitido aprovechar sus capacidades y utilizarlas de forma natural en el procesamiento de operaciones de ámbito general.

Las componentes CPU y GPU son utilizadas para procesamiento; sin embargo, la CPU suele ser utilizada de forma natural como la componente principal, mientras que la GPU es requerida como componente auxiliar, y en determinadas arquitecturas como un núcleo dentro de un conjunto de procesamiento en paralelo.

Tecnologías de cómputo gráfico y concurrente

La tecnología ha avanzado progresivamente en la dirección del cómputo dirigido a gráficos y a programación concurrente; y es de resaltar que, por un lado, muchas soluciones de ámbito general se han adaptado para resolver problemas en estos temas, pero además, han surgido nuevas soluciones tecnológicas orientadas de origen a problemas de esta naturaleza. Algunos de los ejemplos más destacados son CUDA, OpenCV e Intel IPP.

Compute Unified Device Architecture (CUDA)

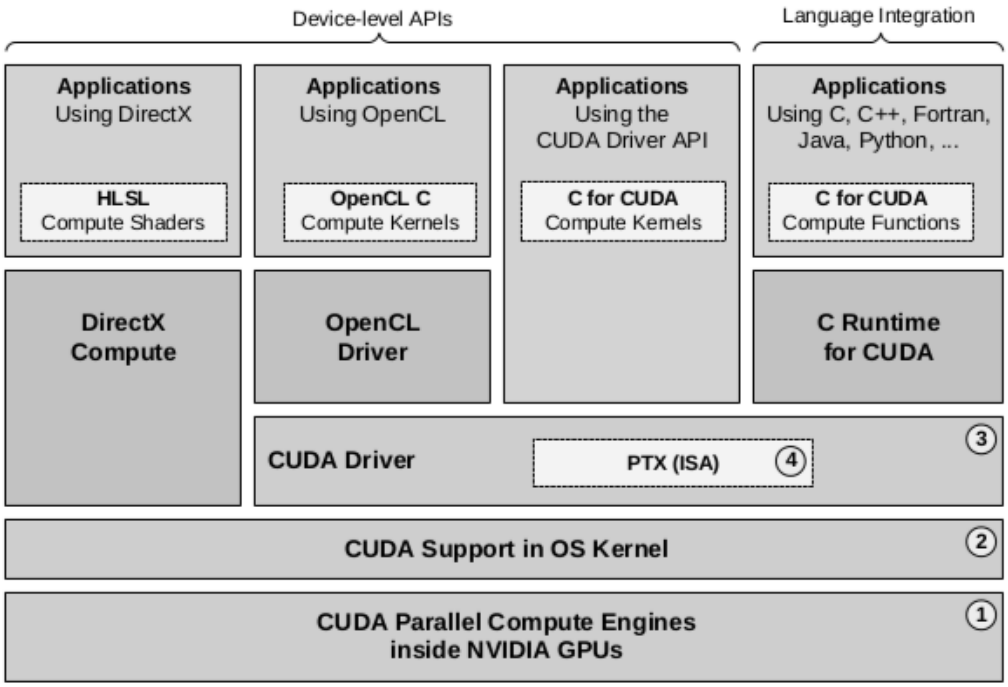
CUDA es una plataforma creada por la empresa de tecnología NVIDIA. Es un producto enfocado al desarrollo y operación de cómputo en paralelo. Una de sus características principales es haber sido desarrollado pensando en aprovechar el potencial que ofrece la GPU. [1]

La evolución de la computación, desde la perspectiva de requerimientos del usuario, implicó centrar los esfuerzos no solamente en mejorar los tiempos de procesamiento y las capacidades de los equipos, sino además en generar interfaces de usuario cada vez más visuales y sencillas. Esta tendencia dio origen a la necesidad de tener un componente dedicado específicamente al procesamiento de gráficos, la GPU.

Desde finales de los noventas, el *hardware* cada vez se volvió más programable, lo que culminó con la primera GPU de NVIDIA en 1999, dispositivo del cual los investigadores empezaron a aprovechar su excelente rendimiento en operaciones de coma flotante, lo que derivó en el uso de la GPU para operaciones de propósito general.

El proceso de programar para la GPU resultaba muy complejo, por lo que NVIDIA se dedicó a la tarea de desarrollar una solución que permitiera acelerar los tiempos de desarrollo y facilitara el acceso a los recursos de la GPU. Estos esfuerzos culminaron finalmente en 2006 con la creación de la API conocida como CUDA.

Usando lenguajes de alto nivel, las aplicaciones aceleradas por la GPU ejecutan la parte secuencial de su carga de trabajo en la CPU mientras se acelera el procesamiento paralelo en la GPU.



Arquitectura de CUDA

La arquitectura de CUDA está formada de tal manera que pueda ser usada en diferentes sistemas operativos y teniendo como base unos motores de cómputo paralelo. Sin embargo, CUDA presenta la limitante de estar disponible únicamente para dispositivos GPU NVIDIA.

Open Source Compute Vision (OpenCV)

OpenCV es una biblioteca de código abierto orientada a aplicaciones de visión artificial y aprendizaje automático.

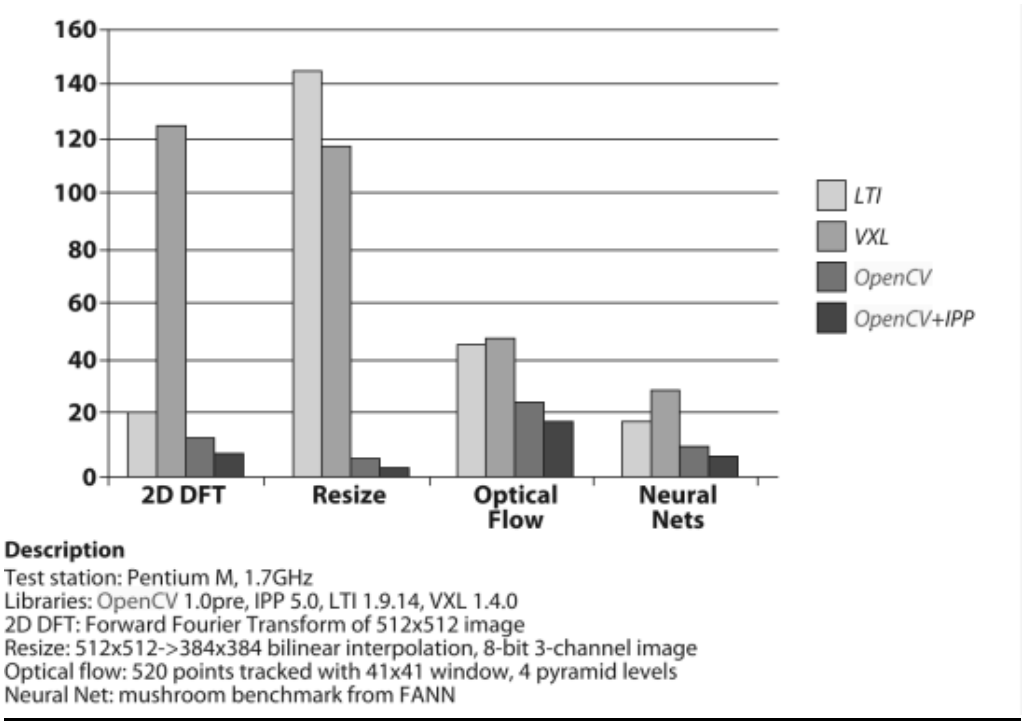
OpenCV fue creada por Intel en junio del 2000, con la finalidad de proveer una infraestructura común para las aplicaciones de visión artificial. La biblioteca está desarrollada en C y C++ y se encuentra disponible en sistemas Linux, Windows y Mac OS X. [2]

El proyecto fue diseñado para tener una gran eficiencia en los programas y tiene un fuerte enfoque en aplicaciones de tiempo real. Una de sus ventajas es que aprovecha las capacidades de los procesadores multinúcleo.

OpenCV es muy utilizada en proyectos científicos, de investigación y docencia, así como en diversos desarrollos que implican procesamiento de imágenes o video. Cuenta con una gran variedad de funciones, que pueden agruparse en los siguientes bloques: [3]

- Operaciones básicas con estructuras de datos (matrices, vectores, grafos, árboles, etc.).
- Procesamiento y análisis de imágenes.
- Análisis de estructuras.
- Análisis de movimiento y seguimiento de objetos.
- Detección de objetos.
- Calibración de cámara.
- Reconstrucción tridimensional de objetos.
- Interfaces gráficas de usuario, para control de aplicaciones y adquisición de video.

En la siguiente imagen podemos ver el rendimiento de diferentes bibliotecas de visión artificial, LTI y VXL con respecto a OpenCV. [4]



Rendimiento de OpenCV

Intel Integrated Performance Primitives (IPP)

IPP es un desarrollo de Intel con la finalidad de proporcionar una biblioteca de algoritmos altamente optimizados para aplicaciones de procesamiento de señales, datos y multimedia. [5]

La portabilidad de esta biblioteca nos permite usarla en múltiples sistemas, y nos brinda la posibilidad de maximizar el aprovechamiento de la capacidad de la CPU, así como optimizar las operaciones SIMD (*Single Instruction, Multiple Data*).

Dentro de la colección que nos ofrece podemos encontrar algoritmos para diversos problemas, tales como: [6]

- Procesamiento de señales.
- Procesamiento de imágenes.
- Visión artificial.
- Reconocimiento de voz.
- Compresión de datos.
- Criptografía.
- Procesamiento de audio.
- Codificación de video.
- Operaciones de matrices.
- Operaciones de procesamiento y presentación en 3D.

Intel IPP presenta la limitante de funcionar únicamente en arquitecturas Intel.

Integración de tecnologías

Una de las principales características de las tecnologías orientadas a gráficos y programación concurrente es que están desarrolladas buscando su fácil integración entre las mismas. Esto permite aprovechar las capacidades que ofrece cada una, e incluso en algunos casos, potenciar el rendimiento de alguna característica en particular.

Plataformas como CUDA, OpenCV e Intel IPP fueron creadas como componentes que pueden interactuar entre sí, realizando tareas que se complementan y permiten aprovechar al máximo las capacidades de las tecnologías disponibles.

En particular, la combinación de estas plataformas es especialmente útil para el desarrollo de aplicaciones que requieren manejo de gráficos y procesamiento en paralelo y en tiempo real.

La componente OpenCV permite utilizar de forma ágil y efectiva los recursos gráficos del sistema, tales como captura de video y obtención de imágenes.

La componente Intel IPP es importante para aumentar el rendimiento de las aplicaciones en plataformas Intel. En particular, puede ser aprovechada junto a OpenCV como componentes independientes o incluso integrarse al mismo OpenCV. El resultado de esta combinación puede aumentar el rendimiento de una aplicación hasta 2.21 veces. [7]

La componente CUDA permite la integración de una aplicación al GPU en plataformas NVIDIA, además de facilitar el uso de un modelo de procesamiento en paralelo, mediante el uso de matrices y vectores.

Detección de objetos

La detección es el proceso a través del cual se identifican objetos en una imagen o secuencia de video. Este proceso es realizado de forma natural por un ser humano, pero en el caso de un sistema de cómputo es una tarea que conlleva una gran complejidad.

Estado del arte

Existen diversas aplicaciones que utilizan el concepto de detección de objetos, destacando los siguientes ejemplos:

Detección de rostros

Este es un caso particular de la detección de objetos, orientado a la identificación de características faciales. Se encuentra implementado en el software de la mayoría de las cámaras digitales modernas. Esto permite ajustar las imágenes, a fin de hacer resaltar los rostros presentes en las fotografías o videos. [1]

Esta implementación también se encuentra presente en la red social Facebook, que incluye un software de reconocimiento de rostros, el cual permite identificar a las personas que aparecen en las fotografías almacenadas en esta red. [2]

Otro uso destacado de la detección de rostros es la identificación biométrica que utilizan algunos dispositivos como medio de autenticación. Particularmente, dispositivos como los de la línea Galaxy de Samsung con sistema Android permiten el desbloqueo del sistema mediante la identificación del rostro del usuario principal a través de la cámara frontal. [3]

Una nueva funcionalidad presente en los dispositivos Galaxy de Samsung denominada *Smart Stay*, permite el ahorro de energía del dispositivo, apagando la pantalla en el momento en que detecta que el usuario no está observándola. Esta función utiliza de nueva cuenta el reconocimiento de rostros para llevar a cabo su tarea. [4]

Detección de cuerpos y movimiento

La detección de cuerpos humanos, y particularmente su movimiento, son funcionalidades que se han vuelto muy populares, ya que han permitido explorar nuevas formas de interacción de los usuarios con sistemas computacionales.

Un ejemplo particular de la detección de cuerpos es el que han presentado las nuevas generaciones de consolas de videojuegos, que utilizan dispositivos como cámaras para obtener una representación del usuario de la consola en tiempo real, y así definir su interacción tanto con el sistema como con los videojuegos creados para ello. [5]

De forma adicional al mundo de los videojuegos, han surgido proyectos que buscan revolucionar la interacción entre usuarios y sistemas de cómputo, basando su premisa en la detección de gestos y movimiento.

Un ejemplo importante de detección de gestos es el proyecto Leap Motion. El dispositivo creado por esta propuesta consiste en un sensor que detecta los gestos y el movimiento de las manos, obteniendo una representación virtual de las mismas, lo cual permite la interacción con una computadora en un nuevo nivel de profundidad. [6]

Algoritmos de Detección de Objetos

La detección de objetos es realizada mediante distintos algoritmos, cada uno de los cuales posee características específicas; sin embargo, hay conceptos generales que son utilizados por todos ellos, y que es conveniente considerar:

Categorización

Este concepto se refiere a la clasificación de imágenes en diversas categorías predefinidas. Los procesos de categorización son importantes para la identificación de un objeto particular.

Falso positivo

En el ámbito de la detección, un falso positivo refiere a la detección errónea de un objeto en una imagen en donde no lo hay; o si lo que se busca es un tipo de objeto específico, a la identificación de un objeto de un tipo distinto, en vez del requerido.

Característica o *feature*

En el área de procesamiento de imágenes, una característica es un conjunto de datos que es relevante para la identificación de un objeto. Puede tratarse de una estructura particular de una imagen, como un punto, un tipo de arista, una región, entre otros. Su principal cualidad es que son invariantes.

Borde

Los bordes en una imagen se definen como las transiciones entre dos regiones con niveles de grises, cuya diferencia es significativa.

Histograma

Es la representación gráfica de una variable en forma de barras verticales, donde la altura de cada barra es proporcional a la frecuencia de cada valor que toma la variable. En el caso de una imagen, su histograma representa la distribución existente de las distintas tonalidades de grises, con relación al número de pixeles. [7][8]

Transformada *Wavelet*

Es una herramienta matemática que facilita el análisis de datos, funciones u operadores en distintas componentes de frecuencia, permitiendo el estudio de cada componente en una escala distinta. [9]

Los algoritmos de detección de objetos pueden ser clasificados en las siguientes categorías: [9]

- Algoritmos basados en conocimiento.

Este tipo de algoritmos trabajan a partir de ciertas reglas definidas de antemano sobre como identificar un objeto. Suelen utilizar atributos geométricos codificados en forma de reglas.

- Algoritmos basados en plantillas.

Los algoritmos que utilizan esta técnica requieren de un modelado geométrico de la forma del objeto a identificar, utilizando elementos geométricos básicos, que constituyen las plantillas.

- Algoritmos basados en la apariencia del objeto.

Este tipo de algoritmos no requieren ningún tipo de conocimiento previo del objeto a detectar, en lugar de ello, utilizan un conjunto de imágenes a partir de las cuales aprenden y codifican lo necesario para detectar los objetos.

- Algoritmos basados en características o *features*.

Los algoritmos que utilizan este tipo de técnicas requieren el uso de características, que les permiten identificar un objeto a partir de sus componentes. Este tipo de algoritmos son muy eficaces, pero suelen afectarse mucho por componentes como la iluminación o el ruido de la imagen.

Actualmente existen diversas investigaciones que buscan encontrar métodos más eficientes y confiables para detectar objetos, ya que es un gran reto el hacer la identificación de objetos y su categorización correspondiente en tiempos de respuesta cortos, y con la certeza de identificar a todos los objetos teniendo el menor número posible de errores o falsos positivos.

La complejidad de la detección de objetos no va solamente en el sentido del rendimiento, sino también en la forma en que los algoritmos manejan los problemas en las imágenes de entrada, principalmente al tratarse de imágenes que provienen de videos. Pueden existir muchas variaciones entre el modelo de un objeto que se desea identificar, y el objeto presente en una imagen, principalmente en cuanto a posición del objeto, escala, forma, apariencia y orientación. Por otro lado, también existen factores adicionales como la iluminación o el ruido en la escena, generados por condiciones climáticas o por fallas en el dispositivo que captura la imagen. [10]

El método moderno más común de detección de objetos consiste en escanear la imagen por objetos candidatos y evaluar a cada uno respecto a un modelo predefinido del objeto que se quiere detectar. Este modelo puede ser generado a partir de la búsqueda de bordes del objeto (cambios repentinos de iluminación en la imagen) o esquinas (intersección de dos bordes), o incluso mediante el análisis de ciertas características del objeto como su iluminación, su histograma, su transformada *Wavelet*, entre otras. [11]

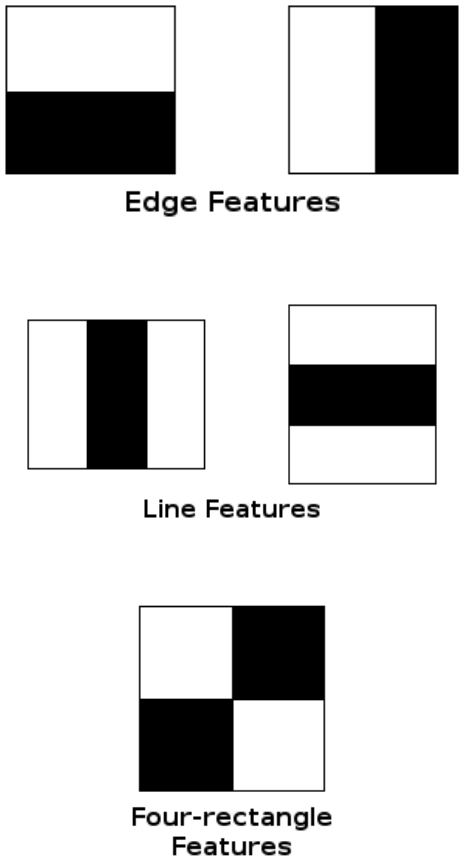
Existen dos algoritmos basados en características de particular relevancia que parten de esta vertiente, y que se caracterizan por sus excelentes resultados: *Haar Feature-base Cascade Classifier* y *Histogram of Oriented Gradients*.

Haar Feature-based Cascade Classifier

Es un método basado en una máquina de aprendizaje, donde una función cascada (el clasificador) es entrenada con varias imágenes positivas, es decir, imágenes que contienen el objeto a detectar; y negativas, que son aquellas que pueden consistir en cualquier cosa distinta al objeto deseado. Esta función cascada es utilizada posteriormente para realizar la detección del objeto en otras imágenes. [12]

El término cascada se refiere a que el clasificador resultante consiste en diversos clasificadores simples (etapas) que son aplicados a la región de interés hasta que el candidato es rechazado o aceptado en caso de pasar todas las etapas.

Los clasificadores simples consisten en un árbol de decisiones con al menos dos hojas. La entrada de estos clasificadores son las características del objeto. [13]



Ejemplos de características o *features*

Cada característica es un solo valor obtenido al sustraer la suma de los pixeles de la región blanca a la suma de los pixeles de la región negra. [14]

Para descartar rápidamente aquellas imágenes que no contienen al objeto, se ponen los clasificadores más sencillos al inicio de la cascada y se aumenta la complejidad de los subsecuentes, de modo que si la imagen no cumple con el criterio del clasificador se detiene el proceso, evitando cálculos innecesarios. También se puede declarar un umbral para que al llegar a una determinada hoja del clasificador se acepte a la imagen, evitando el recorrer todo el árbol de decisiones.

Con esto si se requiere tener un buen nivel de detección, normalmente los clasificadores tendrán un mayor número de características, causando que se tenga a la vez un mayor tiempo de cálculo.

Así podemos ver que los puntos a considerar para hacer el balance entre rapidez y nivel de detección son:

- Número de etapas (clasificadores simples).
- Número de características en cada etapa.
- Umbral en cada etapa.

Histogram of Oriented Gradients (HOG)

En un inicio este algoritmo estuvo enfocado a la detección de peatones en imágenes estáticas, teniendo un mejor desempeño que al usar trasformadas *Wavelets* tipo *Haar,* logrando reducir al número de falsos positivos. [15]

Los descriptores que utiliza este método tienen como base los histogramas de orientación de bordes, descriptores de escala invariante y los contextos de la forma (*shape context*), donde la idea es que la apariencia y forma del objeto a detectar en una imagen puede ser descrito por medio de la distribución de los gradientes.

Así, la detección del objeto se logra mediante la extracción de características de la imagen, tomando en consideración los bordes, calculando los gradientes y las orientaciones de los pixeles.

El histograma de gradientes orientados tiene como rango de valores posibles las distintas orientaciones que pueden tomar los gradientes de los pixeles, los cuales indican la dirección en la cual se produce un mayor cambio en la intensidad o color de la imagen. En el caso de imágenes en escala de grises, el gradiente se dirige de pixeles blancos a pixeles más negros.

Este rango de valores del histograma se divide en clases del mismo tamaño y se almacena en cada una de ellas la suma de las magnitudes de gradiente de los pixeles cuyo ángulo de gradiente se encuentra dentro de la clase.

Así como en el método basado en características tipo *Haar*, se debe entrenar un detector usando una colección de imágenes positivas y otra de imágenes negativas, y la técnica de aprendizaje que se utiliza es la Máquina de Vectores de Soporte (*Support Vector Machine* o SVM).

Flujo de detección utilizando el método HOG

Para obtener este detector, la imagen se divide en pequeñas regiones, llamadas celdas, y de cada una de ellas se calcula su histograma de gradientes orientados. Después se procede a normalizar cada una de las celdas, con el fin de que las condiciones de iluminación no afecten mucho en la detección.

Estimación de Movimiento

La estimación de movimiento es un proceso aplicado en el campo del procesamiento de imágenes y video para determinar los vectores de movimiento que describen la transformación de una imagen en otra.

Los procesos de estimación de movimiento son utilizados principalmente en aplicaciones como: [1]

- Procesamiento de video.
- Visión por computadora.

Estado del arte

Con el surgimiento de dispositivos capaces de grabar y reproducir videos con gran calidad de definición y resolución, los archivos resultantes resultaron en muchos casos muy grandes para su almacenamiento o distribución vía *streaming*.

Para atender la necesidad de facilitar el manejo y distribución de archivos de imagen y video, surgieron métodos para reducir su tamaño, minimizando o evitando la pérdida de calidad. Estos métodos derivaron en la creación de componentes de software denominados códecs.

Un códec (abreviatura de codificador – decodificador) es un componente capaz de transformar un archivo a un flujo de datos o a una señal. Un códec codifica y decodifica la información para su uso en transmisiones o almacenamiento. [2]

Un elemento clave en los códecs es la estimación de movimiento, ya que permite disminuir la redundancia en el dominio del tiempo, además de estar presente en otras aplicaciones como la reducción de ruido, estabilización de la imagen, conversión de estándares y *frame-rate*. [3]

Ejemplos destacados de códecs son JPEG para imágenes, así como MPEG, Theora, WMV y H.264 para videos.

Algoritmos de Estimación de Movimiento

En general los algoritmos de estimación de movimiento, consisten en dividir la imagen en bloques de tamaño de unos cuantos pixeles, y a su vez, agruparlos en otro de mayor tamaño llamado macrobloque (MB), a partir del cual se calcula el vector de movimiento (VM) correspondiente.

Los vectores de movimiento permiten crear una nueva imagen en base a la imagen de referencia de la que fueron obtenidos. Este nuevo proceso se conoce como compensación de movimiento.

Retomando el caso de los códecs, éstos aprovechan los conceptos de macrobloque (MB) y vectores de movimiento (VM), pues en muchos casos dentro de un video, la única diferencia entre un cuadro o *frame* y el siguiente es el resultado del movimiento de la cámara o de un solo objeto aislado dentro de la escena, resultando en que gran parte de los MB no cambien, y consecuentemente, teniendo valores nulos en sus VM (de manera teórica, ya que existen factores que alteran este comportamiento, como iluminación y ruido).

Un concepto fundamental en la estimación de movimiento es el de función de costo. Esta función es utilizada para realizar la comparativa entre los distintos macrobloques de una imagen.

Existen diversas funciones de costo que pueden ser utilizadas, sin embargo, las más habituales son:

- *Mean Absolute Difference* (MAD).

$$MAD = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |C_{ij} - R_{ij}| \quad (1)$$

- *Mean Squared Error* (MSE).

$$MSE = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (C_{ij} - R_{ij})^2 \quad (2)$$

En estas expresiones, N representa el tamaño del macrobloque, mientras que Cij y Rij representan el valor del *pixel* en una determinada posición dentro de cada macrobloque. Mientras más bajo sea el valor resultante de la función de costo, más equivalentes serán los macrobloques comparados.

Otro concepto utilizado es el *Peak-Signal-to-Noise-Ratio* (PSNR), el cual define el ruido que afecta a la imagen creada a partir de los vectores de movimiento y los macrobloques de referencia y la original, lo cual implica que entre mayor sea el valor del PSNR, mejor se realizó la estimación de movimiento.

Este parámetro toma en consideración a la función MSE y se expresa en escala logarítmica, tomando como unidad el decibelio.

$$PSNR = 10 \log_{10} \left[\frac{(PeakValueOfOriginalData)^2}{MSE} \right] \quad (3)$$

Los métodos para realizar la estimación de movimiento en imágenes o videos se pueden clasificar de la siguiente forma:

- Directos.

  Son aquellos que tienen como base los pixeles de la imagen. En este grupo encontramos el algoritmo de *Block Matching*, los métodos de dominio de frecuencia y correlación de fase, flujo óptico y algoritmos de pixel recursivo. [4]

- Indirectos.

  Son aquellos que tienen como base alguna característica o *feature* de la imagen. [5]

Block Matching

Este algoritmo es ampliamente usado en diversos sistemas de codificación de video, como los recomendados por los estándares H.261 y MPEG, para remover la redundancia que hay entre los cuadros o *frames* de un video. [6][7]

De forma general, el algoritmo consiste en dividir el cuadro actual en bloques cuadrados, llamados macrobloques (MB). Luego por cada macrobloque del cuadro actual, se busca el bloque más parecido dentro de una ventana de búsqueda en el cuadro previo, el cual actúa como referencia. A esta ventana se le establece un desplazamiento máximo de búsqueda, produciendo que entre mayor sea el desplazamiento, más costoso será el proceso de estimación.

La posición relativa entre el macrobloque de referencia y el que más se le parece, se conoce como vector de movimiento. [8]

Métodos de Block Matching

Dado que el uso del algoritmo de *Block Matching* en la codificación de video ha sido de gran ayuda, han surgido gran cantidad de variaciones o métodos de este algoritmo, buscando diferentes objetivos, como pueden ser la velocidad o la calidad.

En los métodos enfocados a la velocidad encontramos que generalmente poseen un patrón rectangular o en rombo, mientras que en aquellos enfocados a la calidad, se encuentran los del tipo jerárquico, adaptativos y bloques internos. [9]

A continuación se mencionan algunos de estos métodos.

Búsqueda Exhaustiva (Full Search o FS)

Es el algoritmo más costoso de su tipo, pues calcula la función de costo por cada posible locación en la ventana de búsqueda, por lo que es el algoritmo con mayor PSNR al recorrer toda la ventana.

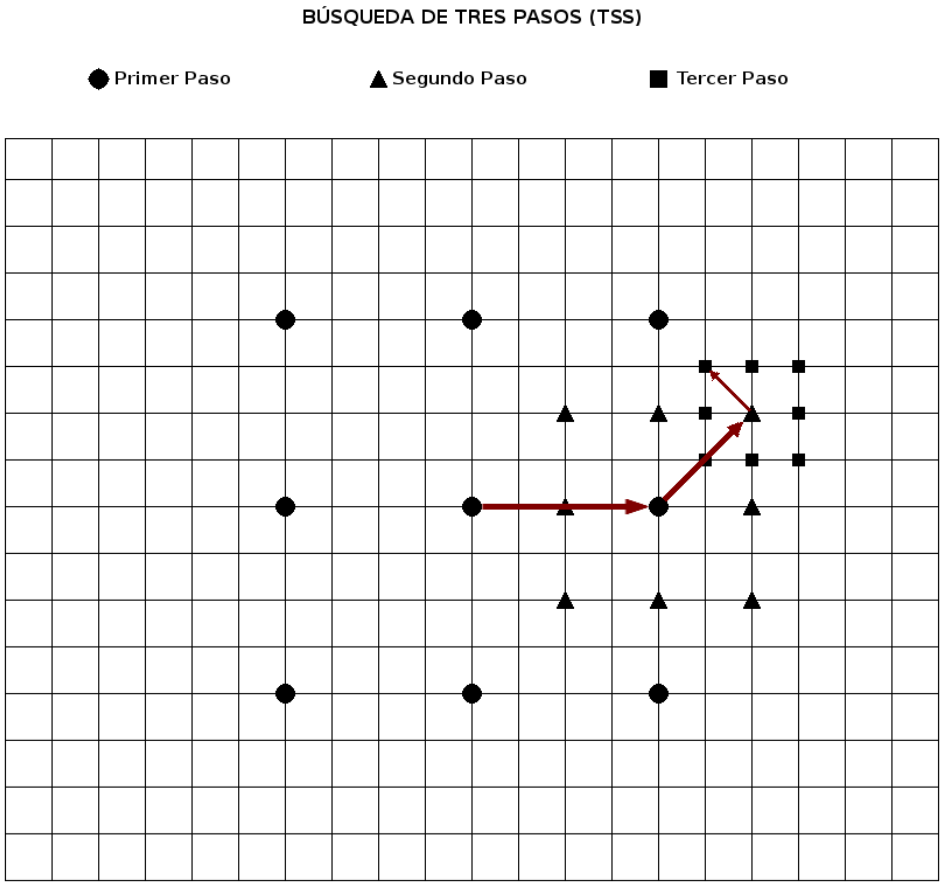
Dado que entre más grande sea la ventana de búsqueda, mayor será el número de cálculos a realizar, este algoritmo no se usa para aplicaciones en tiempo real. Sin embargo los algoritmos rápidos de Block Matching lo usan de referencia al buscar obtener el mismo PSNR, pero con menor número de cálculos.

Búsqueda de Tres Pasos (Tree Step Search o 3SS)

Es uno de los primeros algoritmos rápidos de *Block Matching*, consiste en delimitar una ventana de búsqueda de 8 x 8 pixeles y un desplazamiento inicial (paso) de 4 pixeles.

Se calculan las funciones de costo por cada una de las ocho locaciones alrededor de la posición de origen (0,0) y a sí misma, en el orden de izquierda a derecha, de arriba hacia abajo.

La posición con la menor función de costo se escoge como el nuevo origen, realizando otra búsqueda pero ahora con un paso de 2 pixeles, para luego hacer otra iteración con un paso de 1 pixel. Una vez terminada dicha iteración, se dice que el bloque con la posición resultante es el mejor candidato al macrobloque de referencia.



Proceso de *Block Matching* con 3SS

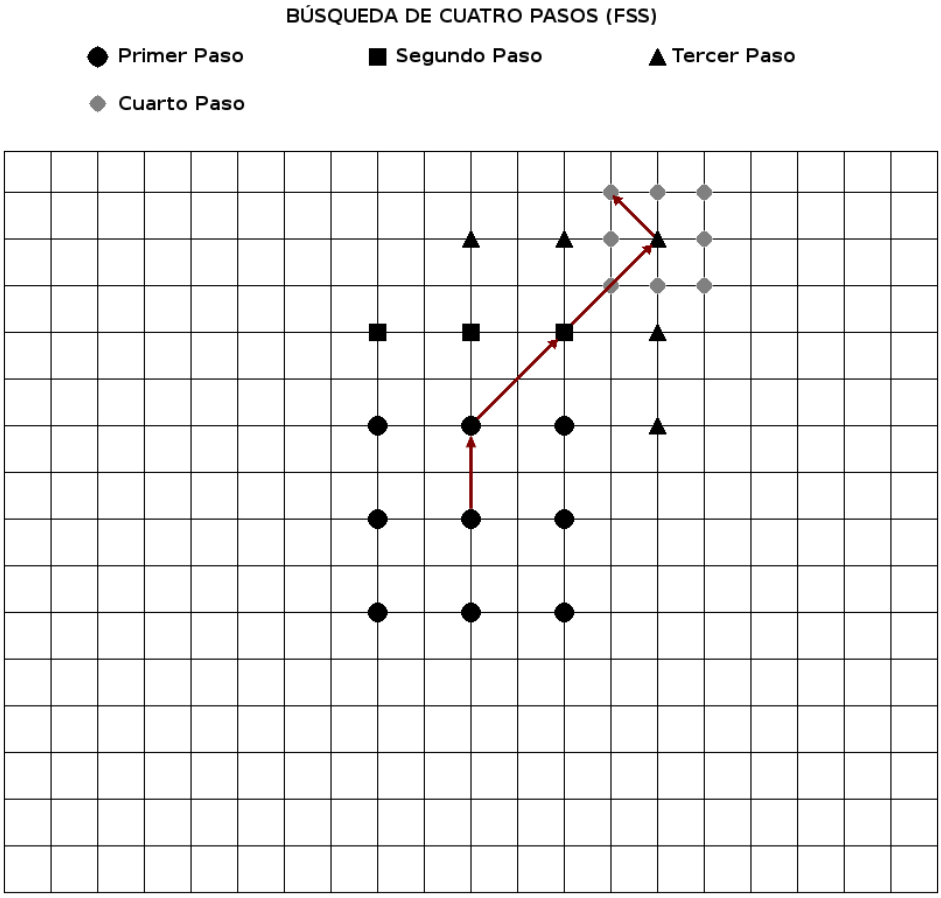
Búsqueda de cuatro pasos (Four Step Search o 4SS)

En este método se establece un desplazamiento inicial de 2 pixeles, no importando el tamaño de búsqueda.

El primer paso es similar al método anterior, al calcular las 9 funciones de costo respecto a las locaciones ya mencionadas, con las diferencias que la distancia de las locaciones ahora es de 2 pixeles y que si la posición (0,0) resulta tener la menor función de costo, entonces se salta al cuarto paso.

En el segundo paso se tiene nuevamente un desplazamiento de 2 pixeles, por lo que ahora se tienen 3 o 5 nuevas locaciones donde calcular las funciones de costo, dependiendo si la posición origen es un borde o una esquina, de las locaciones del paso anterior, respectivamente. Una vez más se elige la posición con la menor función de costo y si la posición (0,0) es la resultante se salta al cuarto paso.

De no ser así, se realiza un tercer paso idéntico al segundo, terminando en un cuarto paso con desplazamiento de un pixel.

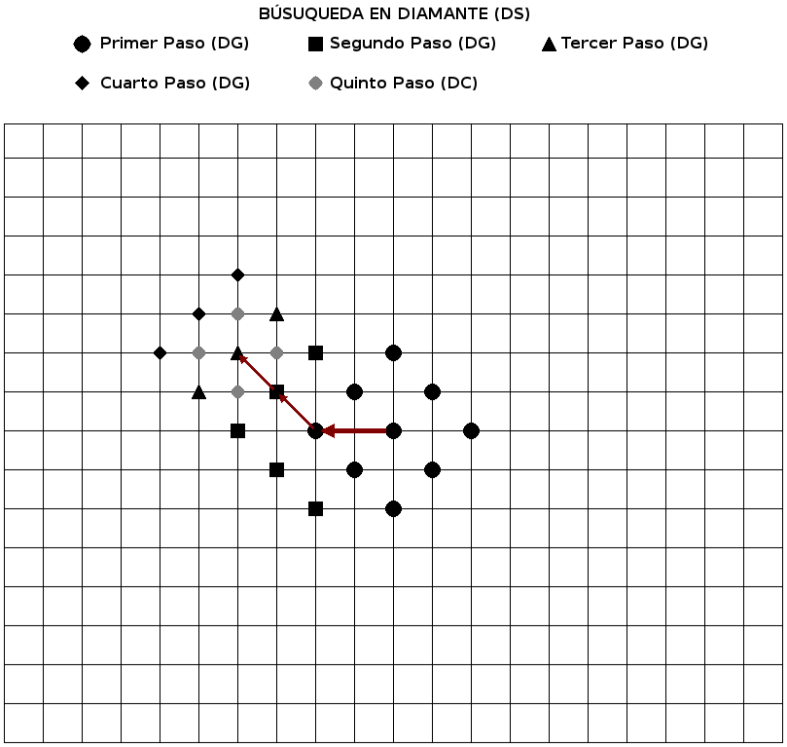


Proceso de *Block Matching* con 4SS

Búsqueda en diamante (Diamond Search o DS)

Este método es similar al de cuatro pasos, con la diferencia que el patrón de búsqueda es un diamante en lugar de un cuadrado y que no tiene un número fijo de pasos.

Para realizar la búsqueda se tienen 2 patrones, un diamante grande y un diamante chico. El primer patrón se usará en la primera iteración y en las subsecuentes, hasta resulte que la posición con la menor función de costo sea la central, entonces se realiza una última iteración con el segundo patrón, siendo la posición final de esta última iteración, la posición del bloque mejor candidato.



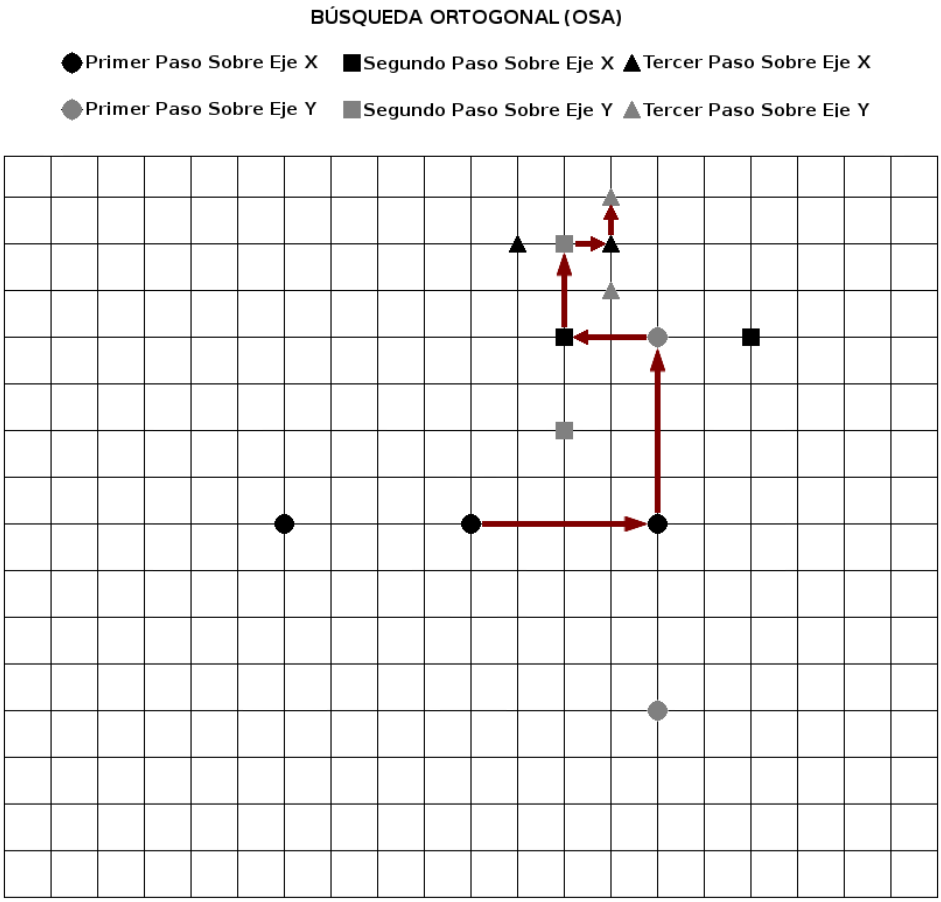
Proceso de *Block Matching* con DS

Búsqueda ortogonal (Ortogonal Search Algorithm u OSA)

En este método el patrón de búsqueda consiste en realizar una iteración de manera horizontal y otra de manera vertical. En él, la longitud del desplazamiento inicial corresponde (S) a la mitad del tamaño de la ventana de búsqueda.

En la primera iteración se calcula la función de costo al centro y a ambos lados sobre el eje x con desplazamiento S. Después se escoge la que tenga el menor valor de las 3 y se usa como origen del siguiente paso. En este segundo paso ahora se calcula las funciones del centro y de ambos lados pero del eje y, teniendo el mismo desplazamiento S que el paso anterior, y nuevamente se selecciona la posición con el menor costo para la siguiente iteración.

En la siguiente iteración se disminuye el desplazamiento a la mitad y se repiten los 2 pasos que se mencionaron previamente. Y se continúa con este ciclo hasta que el desplazamiento sea 1.



Proceso de *Block Matching* con OSA

Comparativa entre métodos

De forma general, el algoritmo *Block Matching* es considerado la alternativa más útil y confiable para realizar la estimación de movimiento; sin embargo, el uso de cada una de sus variantes depende totalmente del objetivo a buscar, considerando los factores de velocidad, calidad, o incluso un punto medio entre ambas necesidades.

Existen algunos estudios que se han realizado con el objetivo de comparar las alternativas de *Block Matching*. Son de particular interés los realizados por el equipo de Chhotray, Kannoujia y Jha, así como del equipo de King, Lo y Tang. Estos estudios realizan la comparación entre diversos métodos a partir de ciertas imágenes reconocidas, tomando en consideración elementos como una función de costo MSE, y el factor PSNR correspondiente. [10][11]

Para este proyecto se han seleccionado las variantes del algoritmo anteriormente descritas (FS, 3SS, 4SS, DS y OSA). En particular, los métodos seleccionados pueden ser clasificados conforme a dos parámetros: la calidad de los vectores de movimiento obtenidos, y el rendimiento, determinado por la velocidad de los cálculos realizados.

A continuación, se presenta la clasificación de los métodos del algoritmo conforme a los resultados de los estudios anteriormente planteados. La numeración representa al 1 como el mejor exponente en cada categoría, y el 5 el peor. Se muestra la diferencia de los algoritmos tomando como referencia a aquel que mejor se comporta.

Algoritmo	**Posición**	**% Calidad**
FS	1	100
3SS	2	99.5
4SS	3	99.0
DS	4	98.0
OSA	5	95.8

En esta tabla el porcentaje que se muestra es respecto al algoritmo de búsqueda exhaustiva (FS) que posee la mejor calidad en los resultados de los estudios, por lo que se le otorga el 100%.

Algoritmo	Posición	Factor de Rendimiento
OSA	1	1
4SS	2	1.51
3SS	3	1.72
DS	4	1.92
FS	5	17.30

En esta segunda tabla se muestra la relación de los algoritmos respecto al de mejor rendimiento, el cual fue el algoritmo de búsqueda ortogonal (OSA). El factor de rendimiento refleja qué tanto más tardado es un algoritmo, considerando el valor de 1 como el mejor rendimiento.

A partir de las tablas anteriores, podemos notar que el algoritmo de mayor calidad es también el de menor rendimiento, ya que sus tiempos de cálculos son de 17.3 veces el tiempo que los del algoritmo más rápido (OSA).

Monitoreo de flujo vehicular

La propuesta para realizar el monitoreo de flujo vehicular en tiempo real conlleva la combinación de dos elementos de gran importancia, la detección de objetos (vehículos) y la estimación de movimiento.

Detección de vehículos

La detección de vehículos juega un rol importante en la realización del monitoreo, ya que esto permitirá conocer la cantidad de vehículos presentes en la zona monitoreada.

Para realizar la detección se ha seleccionado el método *Haar Feature-based Cascade Classifier*, ya que posee una gran flexibilidad para indicar el descriptor del objeto a detectar, en este caso el tipo de vehículos deseados. Por otro lado, otro de los métodos planteados, *Histogram of Oriented Gradients*, suele enfocarse en este tipo de aplicaciones a la detección de peatones.

Para la implementación de la detección se contará con las facilidades que brindan las librerías de OpenCV, que cuentan con funciones orientadas a la realización del método deseado. Por otro lado, estas funciones permiten configurar la detección para hacer más fiable el descriptor del objeto. [1]

La implementación de la detección de vehículos requiere obtener un clasificador, que se encargará de considerar todos los casos de detección del objeto deseado. El clasificador que se utilizará, será entrenado con la finalidad de detectar vehículos totalmente de frente o en diagonal desde la cámara hacia el vehículo.

Casos deseados para realizar la detección de vehículos

Para realizar el entrenamiento del clasificador, se requiere de un conjunto de muestras positivas, es decir, imágenes de vehículos con el perfil deseado. Además, se requiere de otro conjunto de muestras negativas, consistente en imágenes de cualquier objeto u objetos, tomando en consideración que no debe aparecer ningún vehículo en la imagen. En total se obtuvieron 376 imágenes positivas y 2803 imágenes negativas. [2][3][4]

El siguiente paso es normalizar las imágenes positivas para el entrenamiento, redimensionándolas a imágenes de 30 x 25 pixeles, para luego crear las muestras a usar en el entrenamiento. Estas muestras se generan a partir de aplicar ligeras distorsiones a las imágenes positivas, obteniendo así el primer conjunto de imágenes para el entrenamiento.

El segundo conjunto se compone de muestras para probar el clasificador en las diferentes etapas del entrenamiento y ver la tasa de acierto y los falsos positivos que tiene el clasificador en esa etapa. Este conjunto es creado incrustando las diferentes imágenes del primer conjunto en las imágenes negativas.

Por último se comienza el entrenamiento con estos 2 conjuntos, indicando el número de muestras que se usarán, la tasa de acierto y falsas alarmas que se desea y el número de etapas. [5]

Cabe señalar que el entrenamiento con OpenCV puede terminar previo al número de etapas que se le indica, lo cual implica que se alcanzaron las tasas deseadas.

Para este caso, se realizaron 4 entrenamientos con diferente número de muestras. Cada entrenamiento fue especificado indicando parámetros como el número de etapas con un máximo 20, una tasa de aceptación de 0.999 (por cada 1000 muestras se desea detecte 999), y un número de falsas alarmas de 0.5 (por cada 1000 muestras puede haber 500 falsas alarmas).

Dado que estas tasas son por etapa, teóricamente la tasa final del clasificador será de $0.999^{20} \approx 0.98$ de aceptación y de $0.5^{20} \approx 9.54 \times 10^{-7}$ de falsas alarmas.

A partir de los entrenamientos realizados se obtuvieron 4 clasificadores, los cuáles fueron probados a fin de verificar su eficacia. Para realizar la prueba, se creó un programa elaborado especialmente para mostrar los objetos detectados por cada clasificador. La prueba se efectuó utilizando el video en el cual pasan 30 vehículos. Se obtuvo el porcentaje de detecciones de los vehículos, y el promedio de falsos positivos detectados.

Clasificador	% Detectados correctos	# Falsos Positivos
1 (500 muestras)	16	0
2 (1000 muestras)	80	1
3 (2000 muestras)	73	0
4 (3000 muestras)	100	3

Con estos resultados se optó por utilizar el clasificador 4, ya que es el que detectó todos los vehículos, a pesar de devolver más falsos positivos.

Estimación de movimiento

La estimación de movimiento es esencial para realizar el monitoreo de flujo vehicular, ya que nos permite saber el estado de la vialidad determinada, en términos de qué tanto se encuentran avanzando los objetos de la zona monitoreada.

La implementación de la estimación de movimiento será realizada mediante el algoritmo de *Block Matching*. Para determinar la variante del algoritmo a utilizar, se han elegido los dos métodos, de los anteriormente mencionados, que conforme a la clasificación planteada en el capítulo 5, han presentado los mejores tiempos de respuesta: *Ortogonal Search Algorithm* (OSA) y *Four Step Search* (4SS).

La selección de los métodos OSA y 4SS se ha realizado buscando privilegiar el factor de la velocidad, ya que debe cumplirse el objetivo de realizar el monitoreo en tiempo real. El método OSA está considerado como el más rápido de las variantes seleccionadas, mientras que el 4SS, a pesar de no ser tan rápido, presenta una calidad mucho más aceptable.

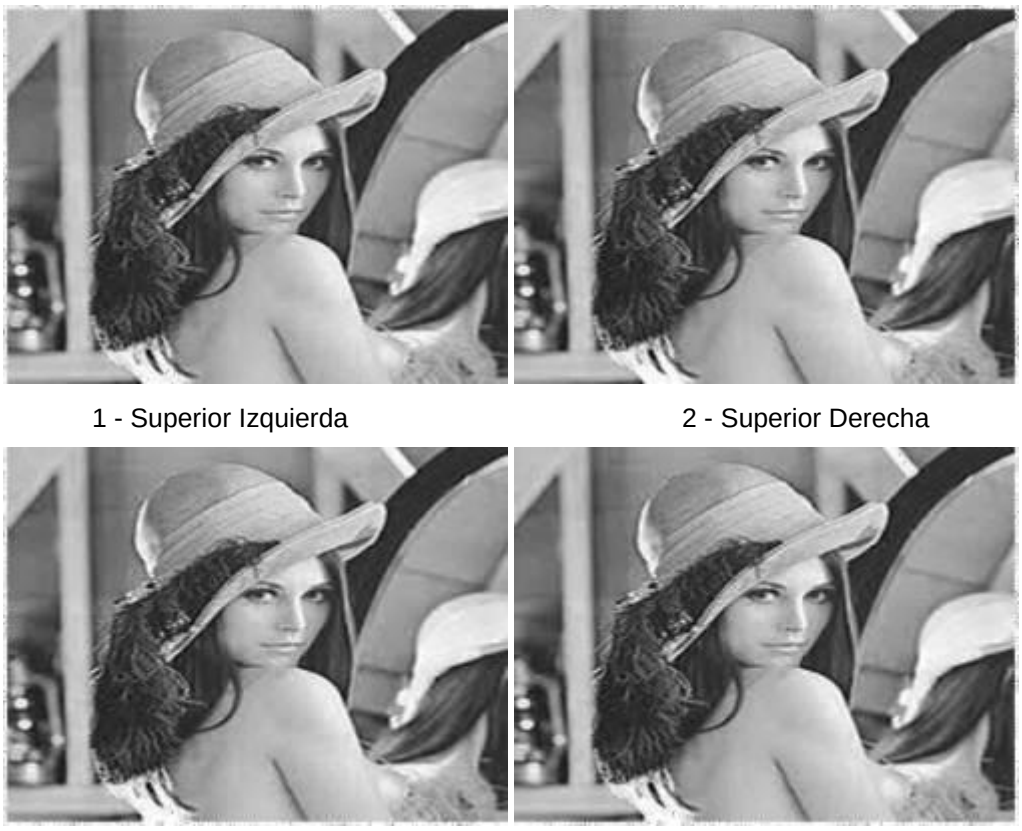
Los algoritmos de *Block Matching* proporcionan los vectores de movimiento asociados a la diferencia entre la imagen actual con respecto a la de referencia, es decir, de un cuadro del video de monitoreo, con respecto al cuadro anterior.

El número de vectores obtenidos depende tanto del tamaño de la imagen, como del tamaño seleccionado para cada macrobloque. Este valor se obtiene a partir de la siguiente expresión:

$$Número\ de\ Vectores = \frac{(Ancho\ de\ imagen)(Altura\ de\ imagen)}{(Tamaño\ del\ macrobloque)^2} \quad (4)$$

A fin de determinar el método más adecuado para utilizar en el monitoreo, se realizó una serie de pruebas con los algoritmos seleccionados.

Como entrada del algoritmo, se utilizó una matriz de 600 x 800 pixeles, a la cual se asignaron los valores en escala de grises (0 a 255) de una imagen ejemplo; esta fue la primera muestra. Para simular el movimiento, se obtuvieron más muestras tomando como base la primera imagen y aplicando un desplazamiento de 7 pixeles verticalmente, y 7 pixeles horizontalmente en distintas direcciones. Las posiciones faltantes de la matriz, debido al desplazamiento, fueron cubiertas con ceros en una primera instancia, y con valores aleatorios posteriormente.



1 - Superior Izquierda 2 - Superior Derecha

3 - Inferior Izquierda 4 - Inferior Derecha

Muestras utilizadas en la prueba

Se usaron dos tamaños de macrobloque: de 8 x 8 pixeles, dando como resultado 7500 vectores de movimiento; y de 16 x 16 pixeles, resultando en 1875 vectores de movimiento, conforme a la ecuación 4.

Considerando los cambios aplicados a las diferentes muestras, se espera obtener el porcentaje de los vectores calculados correctamente, tanto en magnitud como en dirección; así como el porcentaje de aquellos vectores a los que solamente se logró calcular correctamente la dirección.

Para determinar si un vector es correcto, o si solamente está en la dirección correcta, se compara con los siguientes resultados esperados:

Muestra	Esquina	Vector correcto	Dirección Correcta
1	Superior Izquierda	(-7,-7)	(- , -)
2	Superior Derecha	(7,-7)	(+ , -)
3	Inferior Izquierda	(-7,7)	(- , +)
4	Inferior Derecha	(7,7)	(+ , +)

Por otro lado, se busca verificar el rendimiento, obteniendo el número de iteraciones promedio en que el algoritmo encontró el vector de movimiento por cada macrobloque.

Finalmente, es de interés conocer los márgenes de error del resultado obtenido con respecto al esperado, por lo que se aplicará la función del error cuadrático medio sobre los vectores de movimiento resultantes de todos los macrobloques, y los esperados de cada muestra. En este caso, se aplicará el proceso en las direcciones horizontal y vertical.

A continuación se muestran los resultados de las pruebas realizadas.

Para cada prueba se indican los parámetros utilizados: tamaño de macrobloque (cuadrado) en pixeles y valores utilizados para rellenar los espacios vacíos después del desplazamiento, ya sea ceros o aleatorios.

De igual forma, se indica el algoritmo y las muestras utilizadas para la prueba, y en cada caso los cálculos efectuados en promedio para cada MB, el porcentaje de vectores correctos (VC), el porcentaje de los vectores con dirección correcta solamente (DC), y el error cuadrático medio (MSE), dado en pixeles cuadráticos, para la dirección vertical y horizontal de la imagen.

Tamaño de macrobloque: 8
Valor de relleno: ceros

Muestra	OSA					4SS				
	Cálculos por MB	%VC	%DC	MSE HOR	MSE VER	Cálculos por MB	%VC	%DC	MSE HOR	MSE VER
1	12.9021	4.3	36.6	56.48	53.17	22.0573	4.5	36.0	49.54	50.61
2	12.8989	2.2	34.7	62.80	51.70	22.0368	1.7	32.6	56.73	49.80
3	12.9021	1.2	32.6	57.01	60.45	22.1160	1.6	30.9	50.39	59.06
4	12.9072	0.7	31.3	62.65	61.35	22.0672	1.0	30.4	56.60	59.16

Tamaño de macrobloque: 8
Valor de relleno: aleatorios

Muestra	OSA					4SS				
	Cálculos por MB	%VC	%DC	MSE HOR	MSE VER	Cálculos por MB	%VC	%DC	MSE HOR	MSE VER
1	12.8980	4.3	36.2	56.90	53.18	22.0255	4.4	35.8	49.96	50.52
2	12.8956	2.2	34.5	62.73	51.98	21.9972	1.7	32.5	56.54	49.86
3	12.8969	1.1	32.1	57.56	60.34	22.0697	1.5	30.4	50.96	59.02
4	12.8999	0.6	31.1	62.67	61.37	22.0135	1.0	30.5	56.26	58.86

Tamaño de macrobloque: 16
Valor de relleno: ceros

Muestra	OSA					4SS				
	Cálculos por MB	%VC	%DC	MSE HOR	MSE VER	Cálculos por MB	%VC	%DC	MSE HOR	MSE VER
1	16.8259	0.2	33.0	94.14	80.42	21.5484	0.2	32.2	54.57	53.45
2	16.8097	0.4	33.4	98.25	76.97	21.3259	0.3	33.4	53.85	52.74
3	16.8167	0.5	35.6	91.23	78.94	21.4449	0.3	30.2	53.89	55.12
4	16.8243	0.2	31.8	98.50	80.79	21.4643	0.0	31.9	53.63	55.85

Tamaño de macrobloque: 16
Valor de relleno: aleatorios

Muestra	OSA					4SS				
	Cálculos por MB	%VC	%DC	MSE HOR	MSE VER	Cálculos por MB	%VC	%DC	MSE HOR	MSE VER
1	16.8158	0.2	32.7	94.15	80.45	21.5524	0.2	32.7	54.62	53.13
2	16.8065	0.5	33.5	98.09	77.30	21.3140	0.3	33.0	53.99	53.01
3	16.8189	0.4	35.6	91.18	78.52	21.4173	0.2	30.9	53.53	54.97
4	16.8097	0.2	31.5	98.66	81.03	21.3946	0.0	31.7	53.47	55.61

A partir de estos resultados se puede observar que de forma general, el algoritmo 4SS presenta un menor error con respecto a OSA. Por otro lado, el algoritmo OSA incrementa su error considerablemente al aumentar el tamaño del macrobloque.

Otro aspecto interesante es que se observa que OSA calcula una mayor cantidad de vectores correctos que 4SS en muchos de los casos; sin embargo, al ser mayor su error en gran parte de los resultados, esto indica que los vectores incorrectos que obtiene OSA tienen un margen de error mucho mayor con respecto a los obtenidos por 4SS.

En relación al número de cálculos, OSA tiene un menor número, sin embargo, también se nota que aumentando el tamaño del macrobloque, aumentan las iteraciones, ya que la ventana de búsqueda se definió igual al tamaño del macrobloque. Esto resulta en que el desplazamiento máximo es este tamaño menos uno, mientras que con 4SS permanece en el mismo rango, pues el desplazamiento máximo es 7.

Tomando en consideración estos puntos, se optó por usar 4SS pues se tendrá una mejor precisión con respecto a la obtenida con OSA, y la diferencia del número de cálculos es poco significativa.

Finalmente, una mejora práctica al algoritmo 4SS que se encontró, fue la de evaluar primero si la posición (0,0) tiene la menor función de costo, y de no ser así, evaluar qué otra posición tiene el menor valor. Esto es debido a que se puede dar el caso de que las imágenes presenten una superficie plana estática y si se calcula en primera instancia la posición (-1,-1), que es lo que indica formalmente el algoritmo, el vector resultante tendría un margen de error debido a que indicaría un desplazamiento del macrobloque, cuando la realidad indica que no se movió.

Monitoreo de flujo vehicular

El monitoreo de flujo vehicular implica la convergencia de los dos tipos de algoritmos planteados, detección de objetos y estimación de movimiento.

Como se mencionó anteriormente, la detección de objetos permitirá saber la cantidad de vehículos presente en la zona monitoreada. Por otro lado, la estimación de movimiento permitirá conocer los vectores de movimiento de todos los objetos presentes en dicha zona.

Es importante la combinación de ambos factores, a fin de tener una visión completa de la escena, y evitar obtener conclusiones equivocadas si no se cuenta con toda la información.

Un posible caso erróneo es obtener una lectura de estimación de movimiento que nos indique que existe muy poco o nulo movimiento en la zona monitoreada. Si no se cuenta con una adecuada lectura de la cantidad de vehículos presente, no se puede saber con exactitud si la ausencia de movimiento es debido a que no hay vehículos, o a que existe algún tipo de congestión vial o embotellamiento.

Monitoreo optimizado utilizando programación en paralelo

Como se mencionó anteriormente, se utilizarán los algoritmos de *Haar Feature based Cascade Classifier* para realizar la detección de objetos, y *Block Matching* en su método *Four Step Search* o 4SS para obtener la estimación de movimiento.

Los algoritmos utilizados son bastante potentes y eficientes; sin embargo, para llegar al objetivo de realizar el monitoreo en tiempo real, se requiere optimizar al máximo el proceso, para lo cual se ha optado por el uso del paradigma de programación en paralelo.

A fin de maximizar el uso de los componentes disponibles, aprovechando el potencial que ofrece la GPU para realizar operaciones de manera paralela, de forma adicional al procesamiento que puede hacer la CPU en el mismo tiempo, se planteó el dividir las tareas en dos: Por una parte la GPU realizará la estimación de movimiento mediante CUDA, mientras que la CPU llevará a cabo la detección de objetos, mediante OpenCV.

Los procesos llevados a cabo por la CPU no requieren algún tipo de condición especial de hardware, sin embargo, en el caso de la GPU se requiere de una tarjeta gráfica creada por NVIDIA. Este requerimiento es debido al uso de CUDA.

Para este proyecto se utilizó una tarjeta gráfica NVIDIA con las siguientes características:

Modelo	GeForce GT 520
Versión de Driver CUDA	4.2
Multiprocesadores	1
Memoria Global	2 GB
Núcleos de CUDA / MP	48
Frecuencia de reloj del GPU	1.62 Ghz
Frecuencia de reloj de la memoria	533 Mhz
Memoria compartida por bloque	48 Kb
Máximo de hilos por multiprocesador	1536
Máximo de hilos por bloque	1024

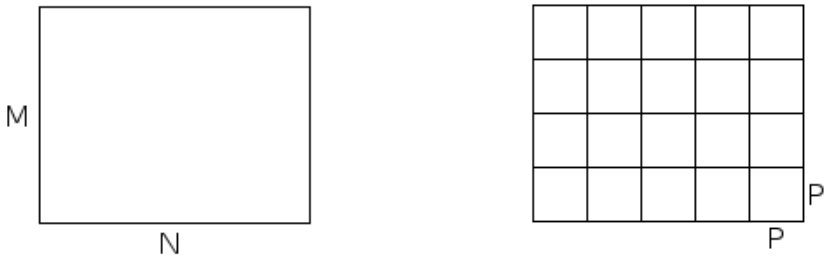
En primera instancia se planteó la implementación del algoritmo de 4SS considerando un hilo de ejecución para cada vector de movimiento. Este hilo sería el único en cada bloque, a fin de aprovechar al máximo la memoria local que presenta este último, la cual es de 48KB.

Sin embargo debido a que el video puede tener una resolución máxima de 800x600, una imagen de esta resolución ocuparía en un arreglo de tipo *char* 480KB, por lo que la memoria local de cada hilo es insuficiente para alojar una copia de cada imagen que se necesita para realizar los cálculos, lo que forzaría a usar la memoria global, perdiendo las ventajas de la paralelización que se busca, y volviendo más lenta la aplicación.

Tomando en consideración la limitante de la memoria local se busco otro enfoque, consistente en dividir la imagen en subimágenes y realizar la estimación de manera individual para cada subimagen en un hilo, para posteriormente juntar los vectores de movimiento en una sola matriz, representando la estimación de la imagen completa.

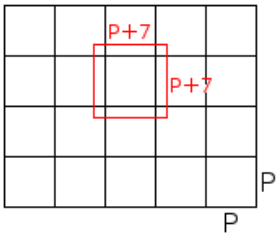
Dicho proceso se realiza de la siguiente forma:

A partir de una imagen de MxN se divide a la imagen en subimágenes de tamaño PxP, donde P es un múltiplo de 8, debido a que el tamaño del macrobloque es también un múltiplo de este número.



División inicial de la imagen completa en subimágenes

Después, considerando que el desplazamiento máximo que puede obtener un macrobloque en 4SS es de 7 pixeles, esta imagen se complementa con los valores de los 7 primeros pixeles vecinos a su alrededor



Se complementa la subimagen con un borde adicional

De esta manera, podemos delimitar el tamaño de la subimagen para aprovechar al máximo los 48KB de memoria local del hilo, y junto con el tamaño del macrobloque, encontrar los valores con los que se obtiene el mejor rendimiento en tiempo.

Para ello se tomo en consideración que el valor de los pixeles va de 0 a 255, por lo que se usaran variables de tipo *char* con la finalidad de tener manejar una subimagen de gran tamaño y dejar la suficiente memoria para las variables auxiliares que se usan para el método. Así, podemos aspirar a un tamaño máximo de 256 x 64 pixeles por una subimagen.

Considerando los posibles tamaños para las subimágenes, se realizó una prueba para evaluar los tiempos de respuesta, dando los siguientes resultados:

Tiempos de respuesta [segundos]					
Tamaño subimagen [pixeles]	**Tamaño macrobloque [pixeles]**				
	8	**16**	**32**	**64**	**128**
39 x 39	0.3890	0.5010	0.3850	NA	NA
39 x 71	0.4570	0.4630	0.3870	NA	NA
71 x 71	0.5000	0.4540	0.4660	0.4860	NA
39 x 135	0.5460	0.5360	0.4980	NA	NA
71 x 135	0.5430	0.5210	0.4610	0.4990	NA
135 x 135	0.4310	0.4330	0.3520	0.4180	0.3640
39 x 263	0.6250	0.5010	0.5270	NA	NA
71 x 263	0.3700	0.3900	0.3940	0.3860	NA

Considerando los mejores tiempos de respuesta, y partiendo de la premisa de utilizar subimágenes cuadradas, además de macrobloques de menor tamaño con respecto al tamaño de la subimagen; se seleccionó la combinación de subimagen de 39 x 39 pixeles (32 + 7), así como el macrobloque de 8 pixeles.

Con el tamaño de macrobloque seleccionado, tenemos 7500 macrobloques en total para toda la imagen, y dado que por cada subimagen existen 16 macrobloques, esto nos da un factor de alrededor de 469, lo que implica el uso de 469 bloques en la tarjeta grafica, y a su vez de los 469 hilos correspondientes, lo cual se mantiene dentro del rango de valores soportados por la tarjeta.

Considerando esto, al inicializar el proceso, se le asigna un ID al macrobloque para saber que el vector N le corresponde, para que al finalizar el cálculo se almacene el valor del vector en la posición N de una matriz que está ubicada en la memoria global, la cuál es la que se le regresa al proceso en la CPU al final.

Proceso de monitoreo de flujo vehicular

El proceso de monitoreo conlleva la supervisión de diversos elementos presentes en la vialidad supervisada, ya sea la cantidad de vehículos que circulan, si su movimiento es constante o si se encuentran detenidos, etc. La combinación de estos elementos será representada por un solo valor numérico, el cual será denominado flujo vehicular.

El valor del flujo vehicular será definido como la suma de dos valores, obtenidos de los algoritmos anteriormente planteados, es decir:

$$Flujo\ vehicular = Valor_{Detección\ de\ objetos} + Valor_{Estimación\ de\ movimiento} \quad (5)$$

Para calcular el valor del flujo vehicular, se utilizará el clasificador de objetos que se obtuvo con el entrenamiento realizado anteriormente, además de la implementación del algoritmo de estimación de movimiento.

La aplicación de monitoreo requiere como entrada el video capturado de la vialidad deseada, en este caso Circuito Interior; esto nos permite obtener el flujo de cuadros o *frames* asociado, donde cada uno de los cuáles es transformado a su equivalente en escala de grises.

Posteriormente, se selecciona una región de interés en el video para delimitar el área monitoreada, lo cual optimiza el funcionamiento de la aplicación, y permite seleccionar únicamente la zona que sea relevante para el monitoreo, por ejemplo, el sentido de la circulación que nos interesa, o alguna sección de la vialidad en particular.

Una vez que se tiene listo el flujo de cuadros de entrada, y se ha indicado una región de interés sobre la cual será realizada el monitoreo, se procede a la aplicación de los algoritmos.

Como se indicó anteriormente, se aprovechan las posibilidades que brinda la programación en paralelo, a fin de realizar la ejecución de la estimación de movimiento (CPU) y de la detección de objetos (GPU) al mismo tiempo.

Para obtener el valor de la estimación de movimiento, se procede a aplicar el algoritmo sobre cada uno de los cuadros del flujo, convertidos a escala de grises. Es importante indicar que el valor obtenido cambiará de un cuadro a otro, es decir, el valor de la estimación será válido solamente para un determinado momento.

Para mejorar los resultados obtenidos, es conveniente establecer valores de referencia que permitan ajustar los valores obtenidos, a fin de reducir el impacto negativo de factores como el ruido de las imágenes, o las variaciones generadas por condiciones de luz o clima.

Los valores de referencia a utilizar serán calculados a partir de los vectores de movimiento obtenidos de los primeros cuadros del video. Los valores que nos interesan son la longitud promedio y la longitud máxima de los vectores.

Considerando que el desplazamiento máximo posible de un vector en un solo eje de referencia es de 7 pixeles, entonces tenemos que la longitud máxima que puede alcanzar equivale a:

$$\sqrt{7^2+7^2} = \sqrt{98} = 9.89$$

En el caso de que todos los vectores de la imagen presenten este movimiento, su longitud tendrá un valor de 9.89, mientras que en el caso de una imagen estática el valor será de 0. Por tanto, el rango para los valores de referencia queda definido como:

$$Longitud\ (Promedio\ y\ Máxima) \in [0, 9.89]$$

El valor de la estimación de movimiento dependerá del valor de referencia utilizado para el cálculo.

Si se utiliza como valor de referencia la longitud promedio, el valor del movimiento para un determinado cuadro será equivalente a la siguiente expresión:

$$Valor_{Estimación\ de\ movimiento} = \frac{Longitud_{Promedio} - Longitud_{Cuadro}}{Longitud_{Promedio}} \quad (6)$$

Donde la longitud del cuadro es la longitud promedio de los vectores del cuadro actual.

Si se utiliza como valor de referencia la longitud máxima, el valor del movimiento para un determinado cuadro será equivalente a la siguiente expresión:

$$Valor_{Estimación\ de\ movimiento} = 1 - \left(\frac{Longitud_{Cuadro}}{Longitud_{Máxima}} * 2 \right) \quad (7)$$

Donde la longitud del cuadro es la longitud promedio de los vectores del cuadro actual.

Para obtener el valor dado por la detección de objetos, se utiliza el clasificador entrenado para detectar vehículos. Al aplicar el clasificador a los cuadros, nos devolverá la cantidad de objetos detectados dentro del área de interés.

El valor propuesto para la detección de objetos será equivalente a la parte entera de la división entre la cantidad de vehículos detectados y el número de carriles de la vialidad, lo cual representa la cantidad de filas completas de autos circulando, es decir, la cantidad de veces que hay un vehículo para cada carril.

Este valor se expresa mediante la siguiente expresión:

$$Valor_{Detección\ de\ objetos} = \frac{Número\ de\ vehículos\ detectados}{Número\ de\ carriles\ de\ la\ vialidad} \quad (8)$$

$$Valor_{Detección\ de\ objetos} \in \mathbb{N}$$

El ajuste a números enteros se realiza a fin de reducir el impacto que un solo vehículo puede tener en el flujo completo.

De la misma forma que el valor obtenido para la estimación de movimiento, el valor que se tiene para la detección de objetos está sujeto a variar en cada cuadro, ya que la cantidad de objetos presente cambiará.

Para el caso propuesto de Circuito Interior, se tomará un valor de 3 carriles, debido principalmente a la situación de las zonas analizadas. Esto nos da la siguiente expresión:

$$Valor_{Detección\ de\ objetos} = \frac{Vehículos_{Cuadro}}{3} \tag{9}$$

$$Valor_{Detección\ de\ objetos} \in \mathbb{N}$$

Finalmente, se obtiene el valor del flujo vehicular. Dependiendo del valor de referencia que se desee considerar para la estimación de movimiento, se plantean las siguientes expresiones:

$$Flujo\ Vehicular = \frac{Vehículos_{Cuadro}}{3} + \frac{Longitud_{Promedio} - Longitud_{Cuadro}}{Longitud_{Promedio}} \tag{10}$$

$$Flujo\ Vehicular = \frac{Vehículos_{Cuadro}}{3} + \left(1 - \left(\frac{Longitud_{Cuadro}}{Longitud_{Máxima}} * 2\right)\right) \tag{11}$$

Donde cada expresión es válida únicamente para el cuadro analizado.

En el caso de que no se detecten vehículos, las expresiones indicadas darán como resultado un número negativo. En tal caso se considerará como resultado del flujo el valor de cero. Esto se puede representar de la siguiente forma:

$$Flujo\ Vehicular_{Final} = \begin{cases} Flujo\ Vehicular & Flujo\ Vehicular >= 0 \\ 0 & Flujo\ Vehicular < 0 \end{cases} \quad (12)$$

Es importante recordar que el valor del flujo vehicular obtenido es válido para cada cuadro de forma independiente, por lo que este valor se calcula continuamente.

De forma teórica, se muestran dos casos posibles para calcular el flujo vehicular, dependiendo de las condiciones de la zona monitoreada.

Para el primer caso, se utilizará como valor de referencia la longitud promedio de los primeros cuadros. Se considera:

$$Longitud_{Promedio} = 5$$

Valores del flujo vehicular										
Vehículos detectados	**Longitud promedio de los vectores por cuadro**									
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0 – 2	1	0.8	0.6	0.4	0.2	0	0	0	0	0
3 – 5	2	1.8	1.6	1.4	1.2	1	0.8	0.6	0.4	0.2
6 – 8	3	2.8	2.6	2.4	2.2	2	1.8	1.6	1.4	1.2
9 – 11	4	3.8	3.6	3.4	3.2	3	2.8	2.6	2.4	2.2
12 – 14	5	4.8	4.6	4.4	4.2	4	3.8	3.6	3.4	3.2

Para el segundo caso, se usará como valor de referencia la longitud máxima de los vectores de movimiento iniciales. Se considera:

$$Longitud_{Máxima} = 9.89$$

Valores del flujo vehicular										
Vehículos detectados	**Longitud promedio de los vectores por cuadro**									
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0 - 2	1	0.79	0.59	0.39	0.19	0	0	0	0	0
3 - 5	2	1.79	1.59	1.39	1.19	0.98	0.78	0.58	0.38	0.17
6 - 8	3	2.79	2.59	2.39	2.19	1.98	1.78	1.58	1.38	1.17
9 - 11	4	3.79	3.59	3.39	3.19	2.98	2.78	2.58	2.38	2.17
12 - 14	5	4.79	4.59	4.39	4.19	3.98	3.78	3.58	3.38	3.17

Los valores obtenidos en estos dos casos son bastante similares, lo cual muestra el uso válido de ambas expresiones. Por otro lado, se puede observar cierto rango entre los valores calculados, lo cual nos permitirá establecer patrones para definir la situación de la vialidad con base en el valor calculado del flujo vehicular.

Los casos anteriores son teóricos y permiten tener una referencia sobre los valores esperados en un caso real.

Resultados

A fin de aplicar la implementación realizada sobre un caso real, se realizó una serie de pruebas utilizando como fuente un conjunto de videos grabados en la vialidad a monitorear. Estos videos muestran la situación de Circuito Interior en diferente horario y lugar.

Las pruebas se realizaron utilizando la aplicación de monitoreo, la cual muestra el video de entrada, además de los indicadores necesarios para conocer el estado de la vialidad. Los indicadores mostrados son:

- Flujo vehicular 1.
  Representa el valor del flujo vehicular tomando como valor de referencia la longitud promedio de los vectores de inicio (ecuación 10).

- Flujo vehicular 2.
  Representa el valor del flujo vehicular tomando como valor de referencia la longitud máxima de los vectores de inicio (ecuación 11).

- Cuadros por segundo (*Frames per second* o FPS).
  Representa la cantidad de cuadros que son procesados por la aplicación cada segundo.

- Color.
  Este indicador muestra de forma gráfica el estado de la vialidad. Su tonalidad depende del valor del flujo vehicular, y puede adoptar tonos desde el verde (flujo rápido), pasando por el amarillo (flujo intermedio), y terminando en rojo (flujo lento).

Los resultados de las pruebas fueron comparados con un valor de referencia denominado patrón dorado, el cual es calculado mediante la observación de lo que sucede en el video, a fin de establecer una comparativa entre lo que puede detectar la aplicación, y lo que es detectado por un ser humano.

El cálculo del patrón dorado del flujo vehicular se realiza considerando dos factores, el equivalente al valor de la detección de objetos y el equivalente al valor de la estimación de movimiento. Así, se propone la siguiente expresión:

$$Flujo\ Vehicular_{Patrón\ Dorado} = Valor_{Objetos} + Valor_{Movimiento} \tag{13}$$

El primer valor es obtenido de forma similar a lo planteado por la ecuación 9, reemplazando la cantidad de vehículos detectados por el valor promedio de vehículos presentes en cada cuadro del video, esto es:

$$Valor_{Objetos} = \frac{Vehículos_{Promedio}}{3} \tag{14}$$

$$Valor_{Objetos} \in \mathbb{N}$$

El valor del movimiento se propone considerando como factor principal la relación entre la cantidad de vehículos presentes en la escena en una ventana de tiempo, dividida entre la duración de esa ventana de tiempo, dada en segundos. Esto da origen a la expresión:

$$Valor_{Movimiento} = 1 - \frac{\Delta_{Vehículos}}{\Delta_{Tiempo}} \tag{15}$$

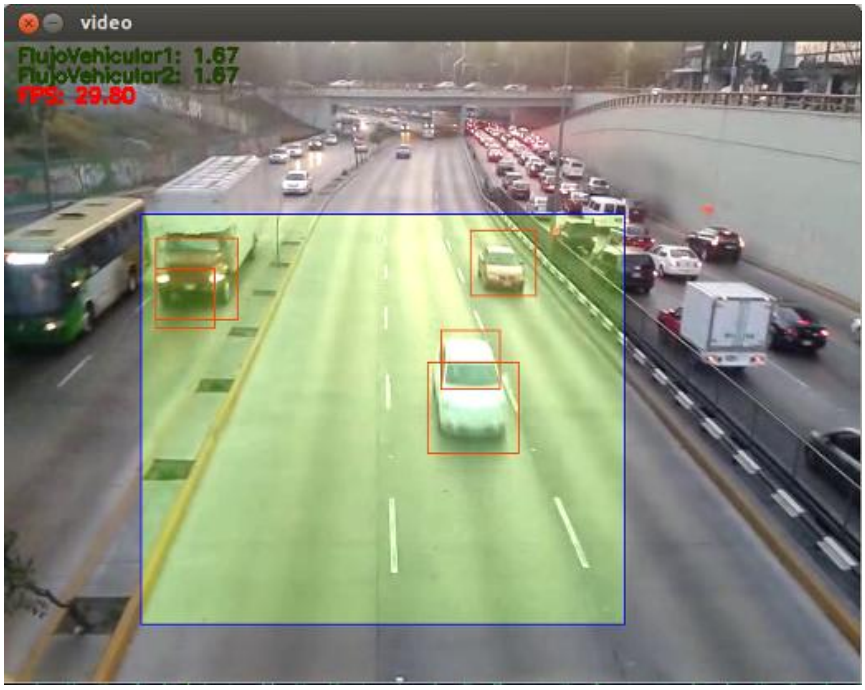
Finalmente, obtenemos:

$$Flujo\ Vehicular_{Patrón\ Dorado} = \frac{Vehículos_{Promedio}}{3} + \left(1 - \frac{\Delta_{Vehículos}}{\Delta_{Tiempo}}\right) \quad (16)$$

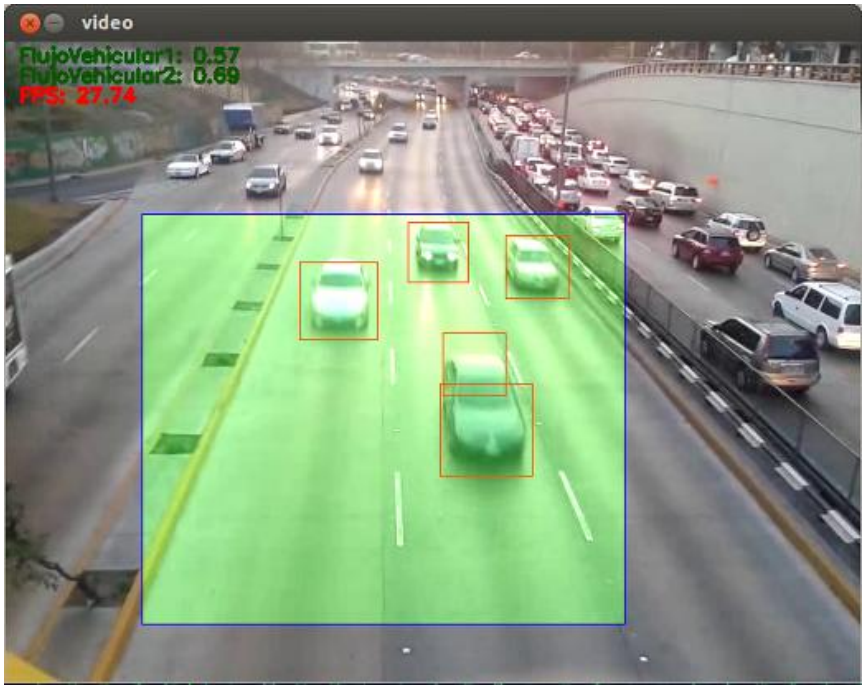
Prueba 1

La primera prueba fue realizada a partir de un video grabado enfocando el cruce de Circuito Interior y Leibnitz, en la colonia Anzures. El video fue tomado alrededor de las 6:00 pm. Las condiciones de iluminación son claras, aunque comienza el atardecer y el cielo está nublado.

La resolución del video es de 640 x 480 pixeles, con 24 cuadros por segundo y una duración de un minuto. Se utilizaron 300 cuadros para obtener los valores de referencia.

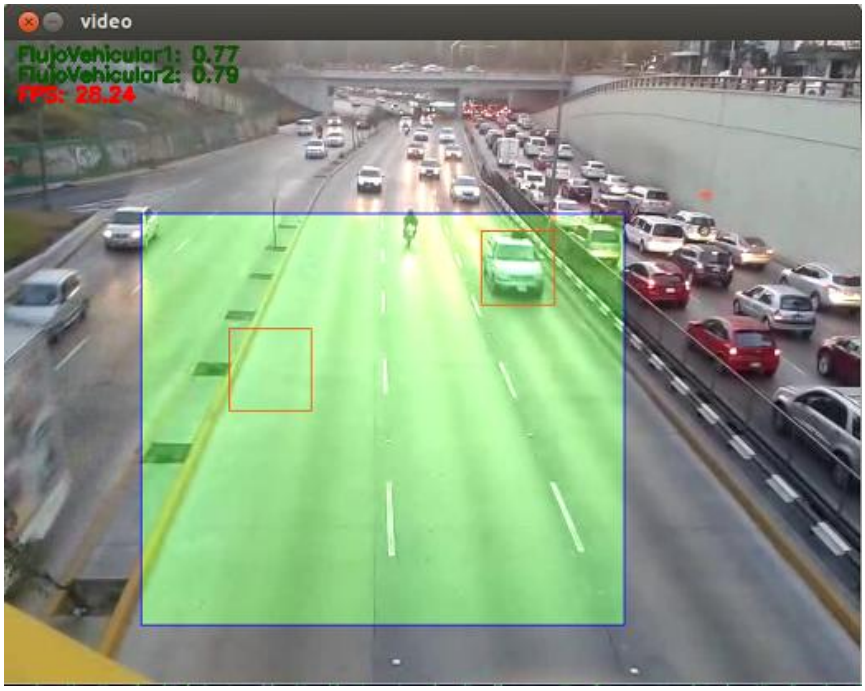


Muestra tomada en el minuto 0:10



Muestra tomada en el minuto 0:20

Muestra tomada en el minuto 0:30



Muestra tomada en el minuto 0:40

Muestra tomada en el minuto 0:50

A continuación se muestra el valor de los indicadores para cada muestra:

Minuto	Flujo Vehicular 1	Flujo Vehicular 2	Cuadros por segundo
0:10	1.67	1.67	29.80
0:20	0.57	0.69	27.74
0:30	0.10	0.17	28.29
0:40	0.77	0.79	28.27
0:50	1.02	1.10	28.02
Promedio	0.88	1.04	28.18

Los valores indicados como promedio corresponden a todo el video, no solamente a las muestras referenciadas.

Mediante la observación del video se determinó un total de vehículos de 52, con un promedio de 2.2 vehículos por cuadro en la región de interés. Con estos datos, obtenemos un valor para el patrón dorado de 0.14.

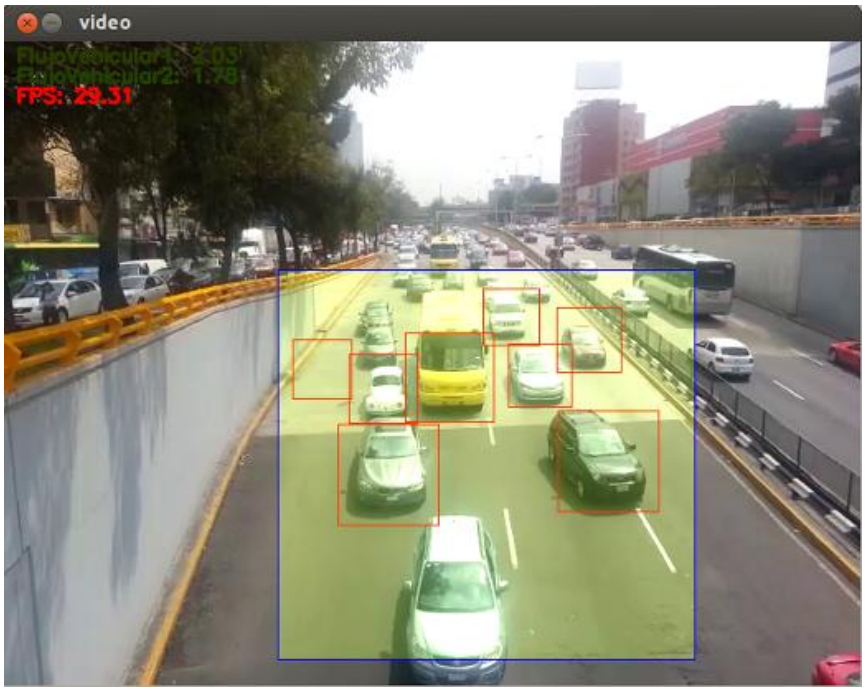
Considerando los valores obtenidos de la prueba, se calculó el error cuadrático medio (MSE) de cada flujo con respecto al valor del patrón dorado.

MSE	
Flujo Vehicular 1	**Flujo Vehicular 2**
0.5476	0.81

Prueba 2

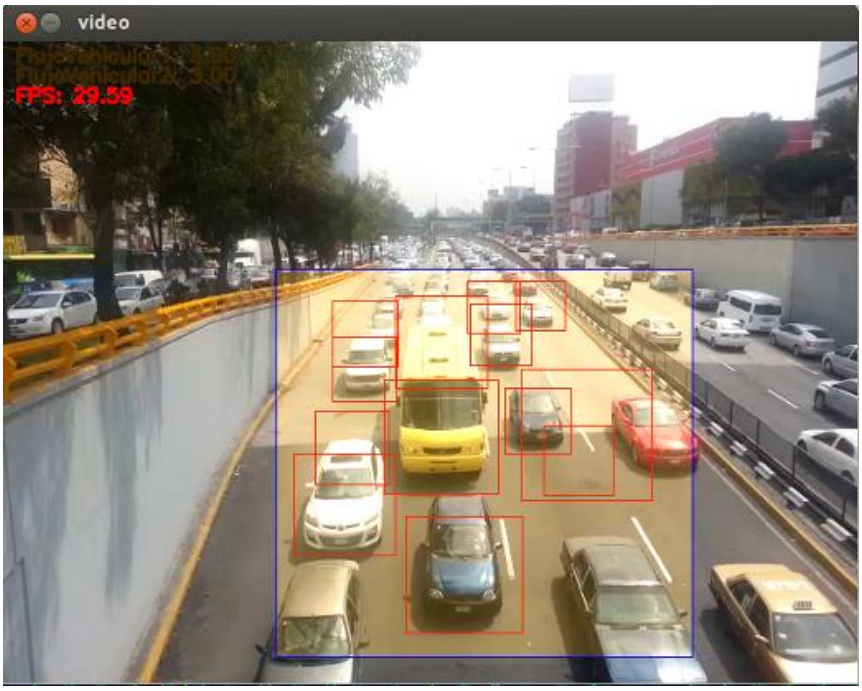
Para la segunda prueba se utilizó un video grabado en el cruce Circuito Interior y Marina Nacional, en la colonia Verónica Anzures. Este video se grabó alrededor de las 2:00 pm, y cuenta con buenas condiciones de iluminación y cielo despejado.

La resolución del video es de 640 x 480 pixeles, con 30 cuadros por segundo y una duración de un minuto. Se utilizaron 300 cuadros para obtener los valores de referencia.

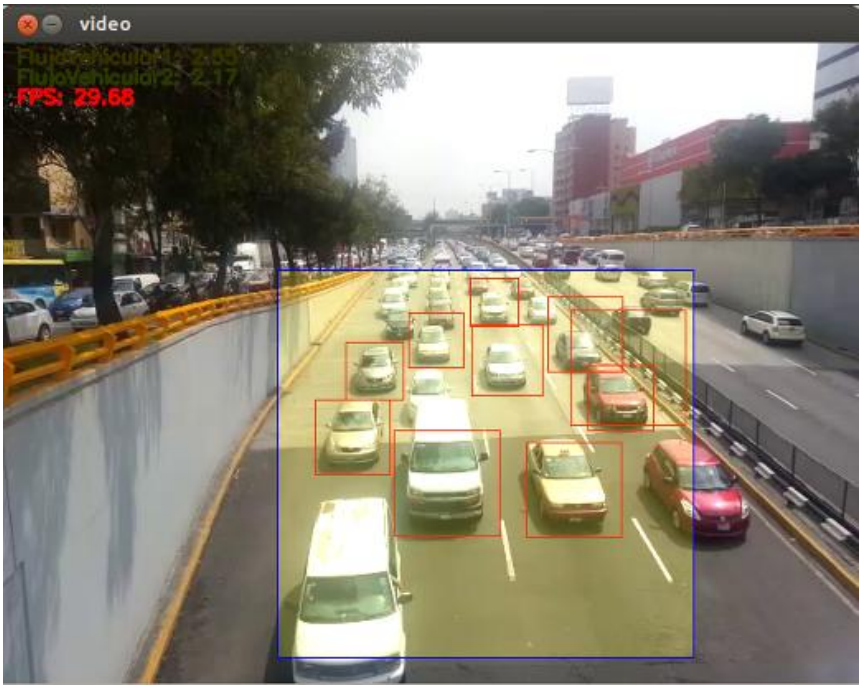


Muestra tomada en el minuto 0:10

Muestra tomada en el minuto 0:20



Muestra tomada en el minuto 0:30



Muestra tomada en el minuto 0:40

Muestra tomada en el minuto 0:50

A continuación se muestra el valor de los indicadores para cada muestra:

Minuto	Flujo Vehicular 1	Flujo Vehicular 2	Cuadros por segundo
0:10	2.03	1.78	29.31
0:20	3.96	3.94	30.48
0:30	3.00	3.00	29.59
0:40	2.55	2.17.	29.68
0:50	4.38	4.22	29.52
Promedio	3.57	3.47	29.62

Los valores indicados como promedio corresponden a todo el video, no solamente a las muestras referenciadas.

El total de vehículos observados es de 93, con un promedio de 12.15 vehículos por cuadro en la región de interés.

El valor obtenido para el patrón dorado en esta prueba es de 3.45.

La comparativa de los valores obtenidos con respecto al patrón dorado nos da los siguientes valores de MSE.

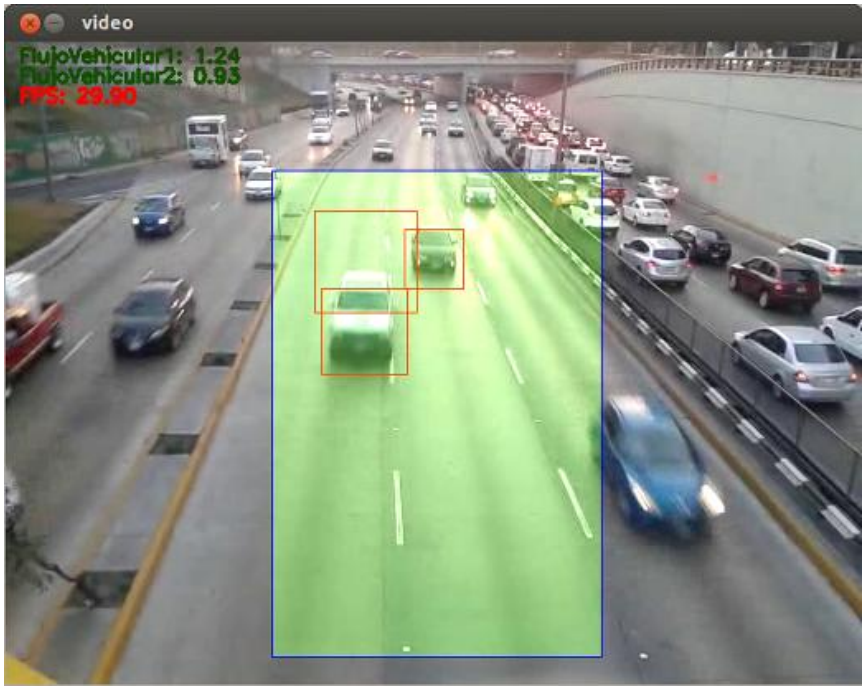
MSE	
Flujo Vehicular 1	**Flujo Vehicular 2**
0.0144	0.0004

Prueba 3

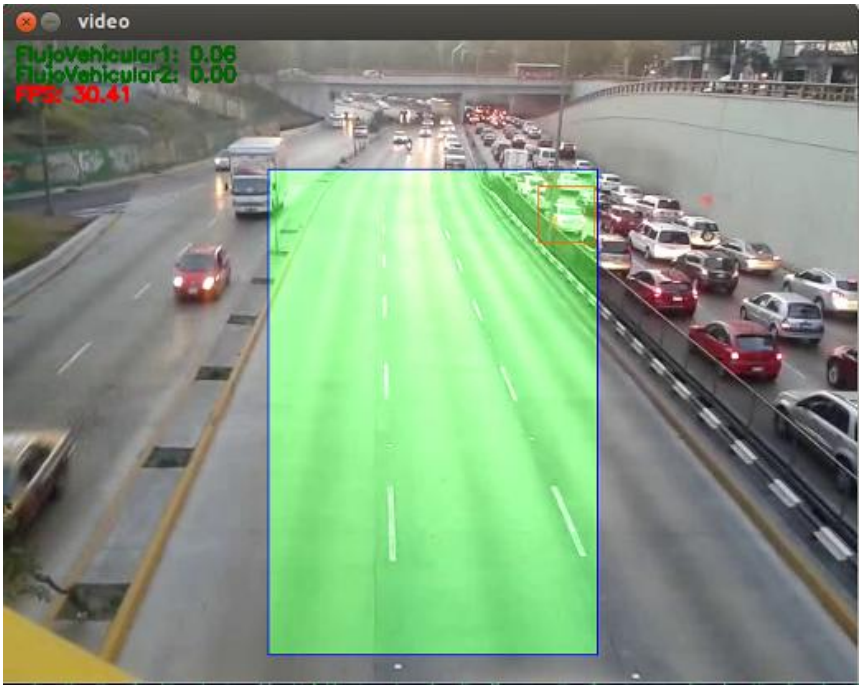
La tercera prueba fue llevada a cabo utilizando un video editado a partir de los videos de las pruebas anteriores. El nuevo video fue formado comenzando por el primer video, posteriormente el segundo, de nueva cuenta el primero, y finalmente el segundo.

La resolución del video es de 640 x 480 pixeles, con 30 cuadros por segundo y una duración de 4 minutos. Se utilizaron 300 cuadros para obtener los valores de referencia.

Este video se hizo con la finalidad de verificar el comportamiento de la aplicación ante un cambio repentino, pasando de una escena con tránsito fluido a otra con tránsito lento, y viceversa.

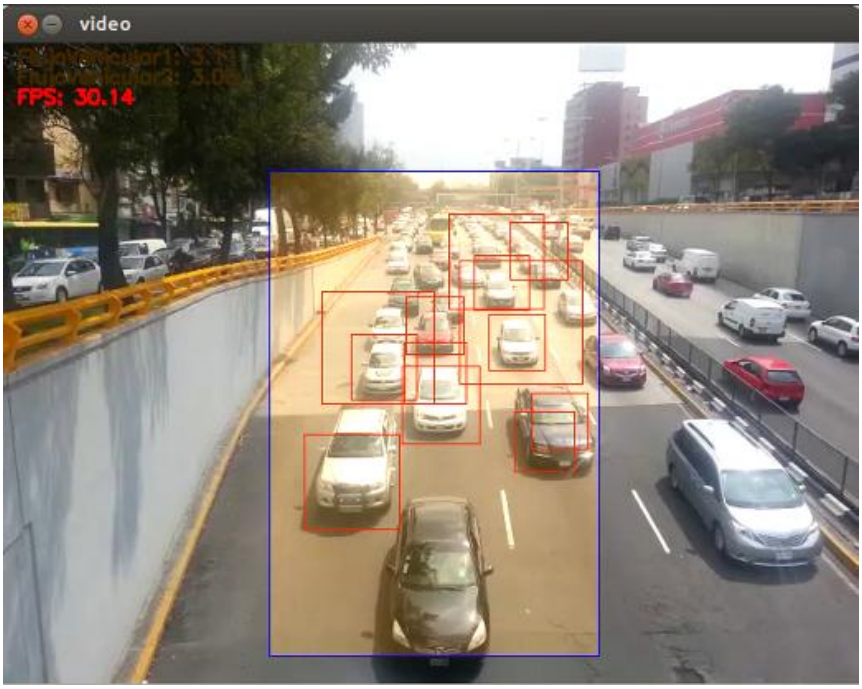


Muestra tomada en el minuto 0:20

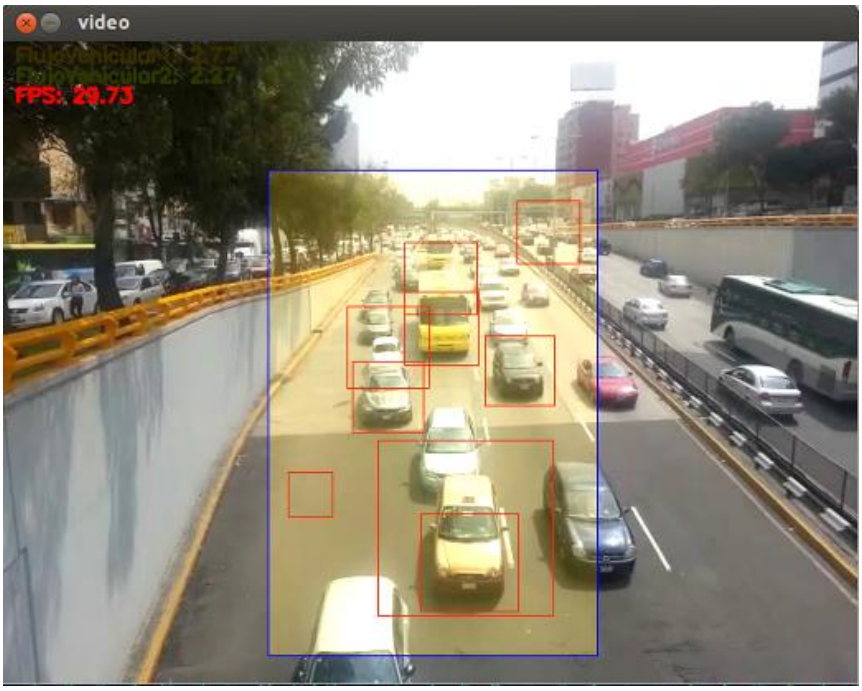


Muestra tomada en el minuto 0:40

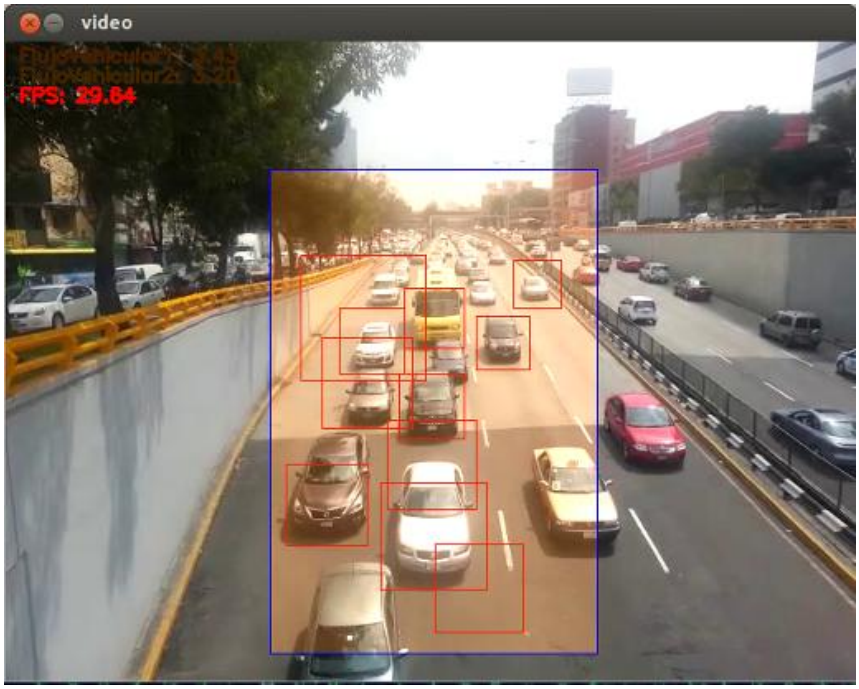
Muestra tomada en el minuto 1:00



Muestra tomada en el minuto 1:20



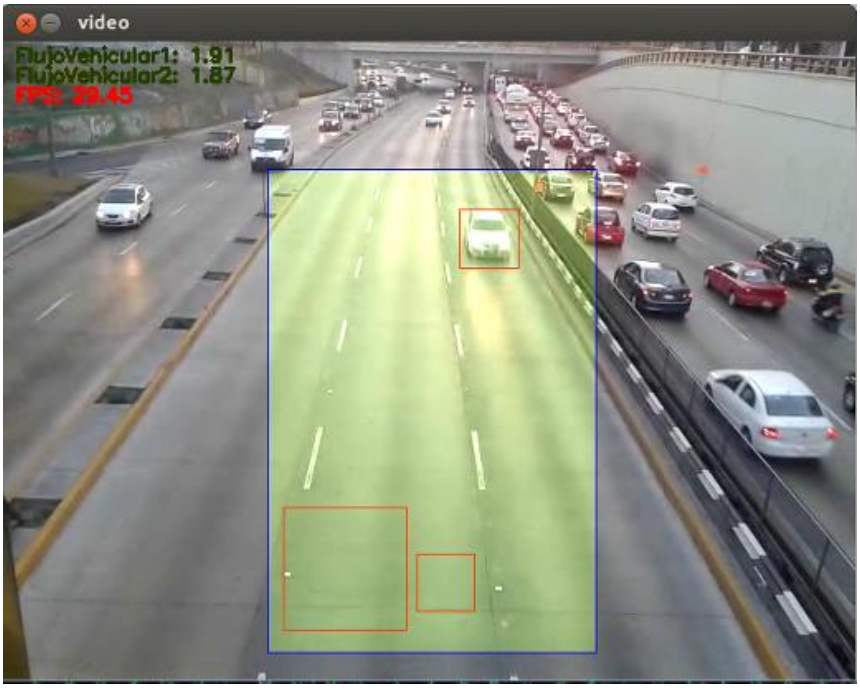
Muestra tomada en el minuto 1:40



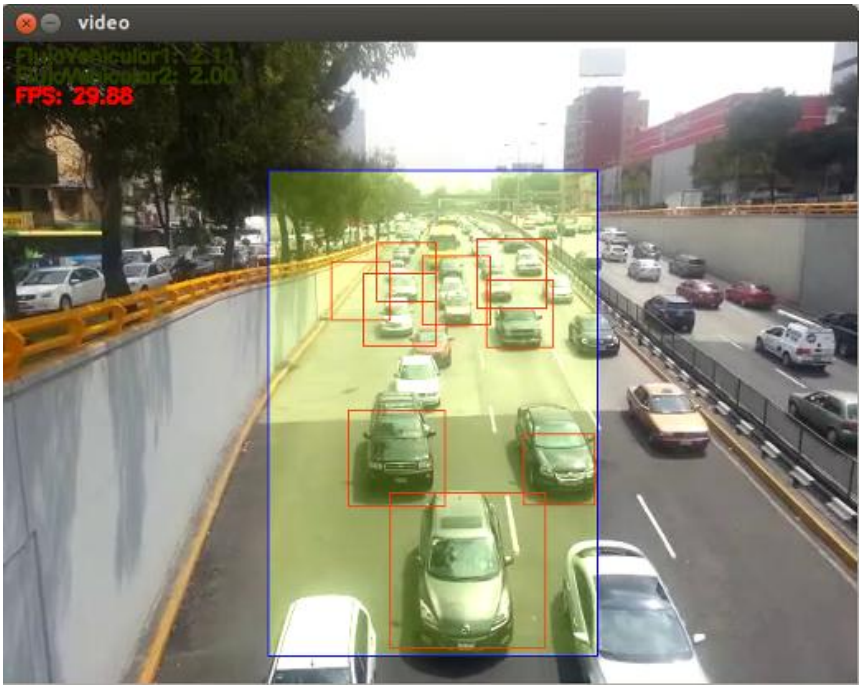
Muestra tomada en el minuto 2:00

Muestra tomada en el minuto 2:20

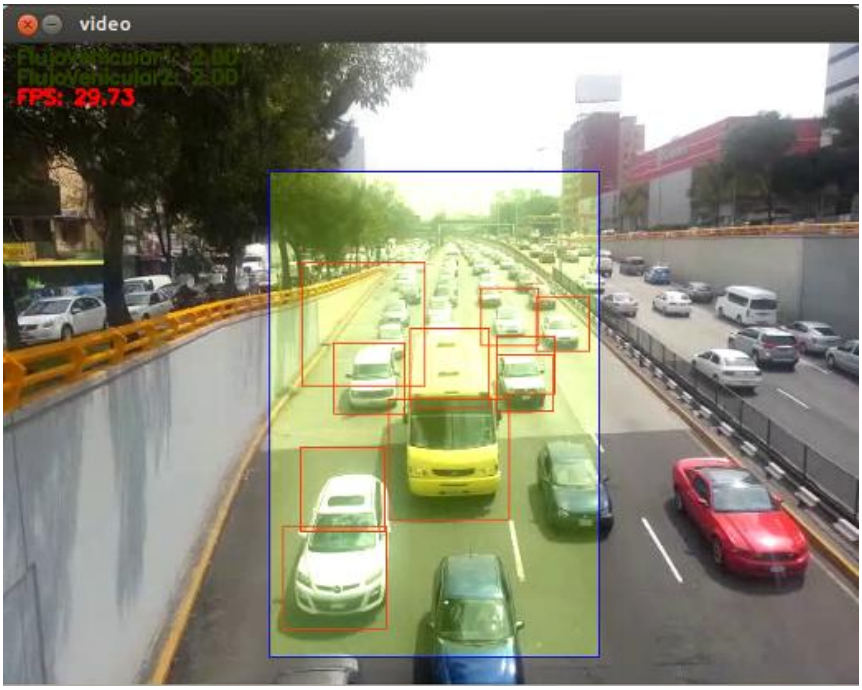
Muestra tomada en el minuto 2:40



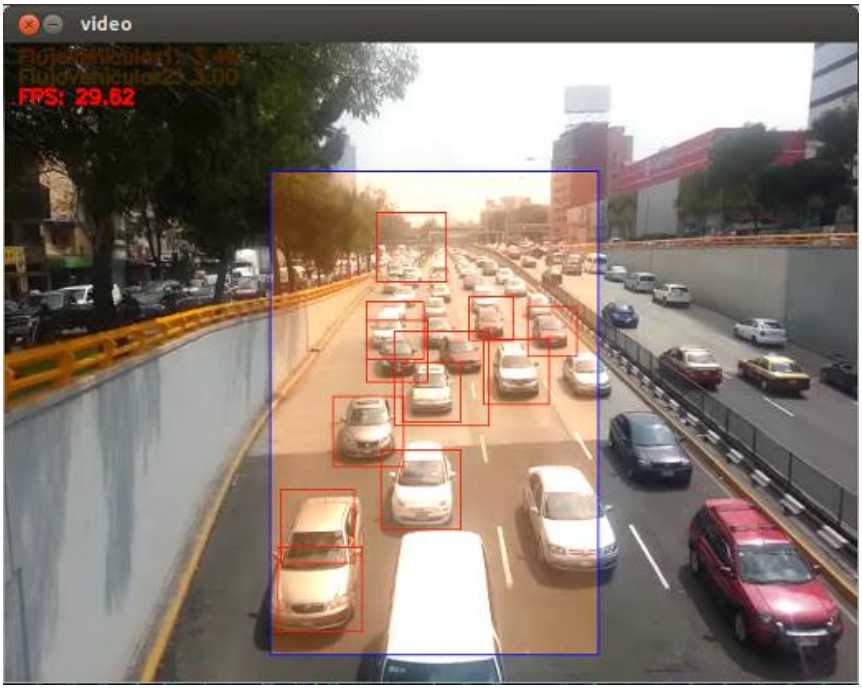
Muestra tomada en el minuto 3:00



Muestra tomada en el minuto 3:20



Muestra tomada en el minuto 3:40



Muestra tomada en el minuto 4:00

A continuación se muestra el valor de los indicadores para cada muestra:

Minuto	Flujo Vehicular 1	Flujo Vehicular 2	Cuadros por segundo
0:20	1.24	0.93	29.90
0:40	0.06	0.00	30.41
1:00	1.00	1.00	30.18
1:20	3.11	3.08	30.14
1:40	2.77	2.27	29.73
2:00	3.43	3.20	29.64
2:20	0.33	0.33	28.22
2:40	0.81	0.73	30.55
3:00	1.91	1.87	29.45

Minuto	Flujo Vehicular 1	Flujo Vehicular 2	Cuadros por segundo
3:20	2.11	2.00	29.88
3:40	2.00	2.00	29.73
4:00	3.48	3.00	29.62
Promedio	1.89	2.25	29.13

Los valores indicados como promedio corresponden a todo el video, no solamente a las muestras referenciadas.

El total de vehículos en todo el video es de 290, con un promedio de 7.17 vehículos por cuadro en la región de interés.

Con estos datos, obtenemos un valor para el patrón dorado de 1.79.

Los valores de MSE obtenidos de la comparación de los flujos obtenidos con respecto al valor del patrón dorado son los siguientes.

MSE	
Flujo Vehicular 1	**Flujo Vehicular 2**
0.01	0.2116

Prueba 4

Se realizaron pruebas complementarias utilizando una serie de videos adicionales de menor duración, cada uno identificado mediante una letra.

A continuación se describen las condiciones de cada video:

- **A, B**. Videos grabados en el cruce de Circuito Interior y Leibnitz con un enfoque frontal.

- **C, D**. Videos grabados en el cruce de Circuito Interior y Leibnitz con un enfoque frontal.

- **E**. Video grabado en el cruce de Circuito Interior y Marina Nacional con un enfoque en diagonal

A través de estas pruebas se obtuvieron los siguientes resultados promedio:

Prueba	Flujo Vehicular 1	Flujo Vehicular 2	Cuadros por segundo
A	0.45	0.37	31.32
B	0.98	0.94	29.01
C	3.51	3.69	29.38
D	3.89	3.99	23.15
E	3.18	2.89	29.78

Prueba	Flujo Vehicular 1	Flujo Vehicular 2	Cuadros por segundo
A + C + E	1.32	1.24	32.05
B + D + E	1.87	2.05	29.44
A + B + C + D + E	2.22	2.28	28.13
A + C + B + D + E	1.67	1.47	31.17
E + A + D	1.44	1.17	30.57

Los resultados de las 5 primeras pruebas complementarias corresponden a cada video por separado. Las pruebas posteriores fueron realizadas con videos generados como combinaciones de los videos indicados para cada prueba.

A partir de las condiciones de cada video, se obtuvieron los siguientes parámetros, a fin de obtener el valor del patrón dorado en cada prueba.

Prueba	Total de vehículos	Promedio de vehículos por cuadro	Duración [segundos]	Flujo Vehicular Patrón Dorado
A	47	2.73	60	0.216
B	59	2.03	60	0.016
C	75	12.30	50	3.50
D	67	12.22	50	3.66
E	84	12.20	60	3.60
A + C + E	206	8.88	170	1.78
B + D + E	210	8.61	170	1.76
A + B + C + D + E	332	8.01	280	1.81
A + C + B + D + E	332	8.01	280	1.81
E + A + D	198	8.86	170	1.83

Finalmente, se obtuvo el MSE en cada prueba.

MSE		
Prueba	**Flujo Vehicular 1**	**Flujo Vehicular 2**
A	0.0547	0.0237
B	0.9292	0.8537
C	0.0001	0.0361
D	0.0529	0.1089
E	0.1764	0.5041
A + C + E	0.2116	0.2916
B + D + E	0.0121	0.0841
A + B + C + D + E	0.1681	0.2209
A + C + B + D + E	0.0196	0.1156
E + A + D	0.1521	0.4356
Promedio	0.17768	0.26743

El promedio de MSE en todas las pruebas es de 0.18 para el flujo vehicular 1, mientras que para el flujo vehicular 2 es de 0.28.

Conclusiones y proyección a futuro

La aplicación desarrollada cumplió el objetivo de realizar el cálculo del flujo vehicular. A fin de determinar la validez de los resultados generados por la aplicación se realizó una serie de pruebas aplicadas en escenarios reales.

Las pruebas dieron resultados acordes a lo observado en cada escenario. En los videos de entrada donde se presentaba un tránsito fluido, la aplicación indicó un valor promedio bajo, de alrededor de 1, lo cual representa un flujo vehicular rápido. En aquellos escenarios con tránsito moderado, los valores del flujo tendían entre 2 y 3, representando un caso intermedio. Finalmente, aquellos videos que mostraron un tránsito lento, generaban valores para el flujo mayores a 3, lo cual implica un flujo vehicular lento.

Los valores obtenidos del flujo vehicular y su interpretación son coherentes con respecto a los valores de referencia teóricos obtenidos de forma previa a la realización de las pruebas.

El planteamiento del patrón dorado permitió realizar otro tipo de comparativa de los valores del flujo vehicular, mostrando también ciertas tendencias en el comportamiento del monitoreo. Por ejemplo, los valores obtenidos mediante la ecuación 10 presentan en la mayoría de las pruebas un MSE menor con respecto a los de la ecuación 11 al ser comparados con el patrón dorado.

La comparativa realizada mostró resultados muy prometedores, ya que en promedio, el MSE obtenido fue menor a 0.19, por lo que se puede establecer un rango de precisión del valor del flujo vehicular de ±0.38 con respecto al patrón dorado.

De forma general, se observaron cambios en el valor del flujo vehicular conforme acontecían eventos como el aumento o la disminución de los vehículos detectados, o de la cantidad de movimiento de cada uno de ellos. Por tanto, la aplicación cumplió el objetivo de identificar los componentes del flujo vehicular, y responder adecuadamente ante los cambios en cada uno de ellos.

Por otro lado, uno de los objetivos más importantes a lograr era crear una aplicación que realizará el monitoreo del flujo vehicular en tiempo real, es decir, que pudiera obtener el valor del flujo en cada momento, con el mejor rendimiento posible, involucrando la detección de los objetos y la estimación de su movimiento.

Uno de los indicadores generados por la aplicación es el de los cuadros por segundo. Este indicador representa la cantidad de cuadros que son procesados y visualizados cada segundo por la aplicación, es decir, aquellos a los que se les aplican los algoritmos de detección y estimación.

En todas las pruebas realizadas, se obtuvo un promedio de cuadros por segundo mayor a 23 en el caso más lento, y de alrededor de 30 en la mayoría de los casos, lo cual representa un valor adecuado para ser denominado como tiempo real. En particular, el valor de 24 cuadros por segundo es considerado el mínimo aceptable para observar movimiento fluido en un video. Además, la mayoría de las cámaras de vigilancia posee una velocidad de captura de 30 cuadros por segundo. [1]

Para la implementación de la aplicación, se determinó la mejor opción para cada algoritmo a utilizar, considerando la mejor relación rendimiento-calidad de estimación, logrando un resultado aceptable en tiempo real, y confiable en el aspecto de la detección de objetos y estimación de movimiento.

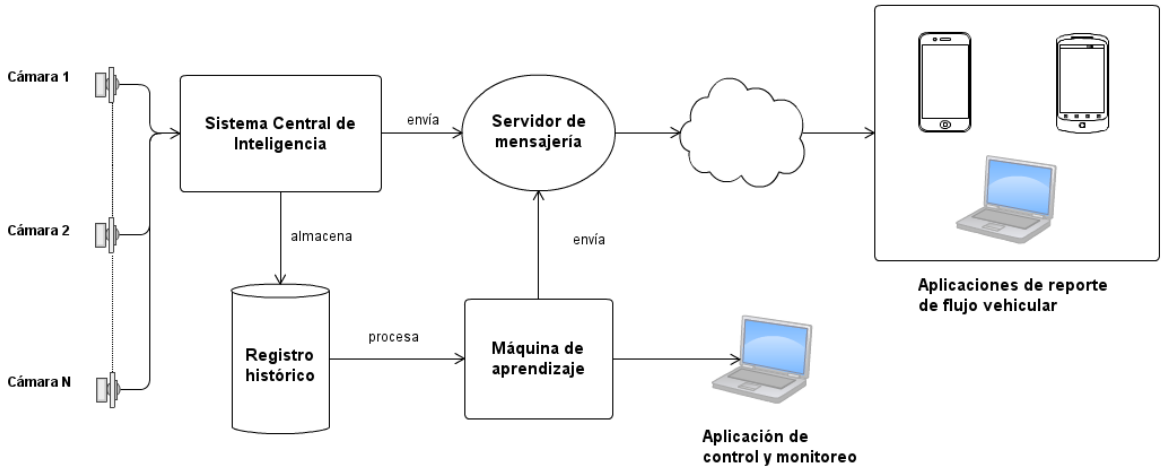
Proyección a futuro

Proyectando el alcance de este aplicativo a futuro, se plantea la creación de un sistema completo de monitoreo de flujo vehicular, que no se limite únicamente a detectar los cambios en el flujo dentro de la ciudad, sino que constituya una plataforma de control y planeación urbana, enfocada al tema vial.

Actualmente, el alcance del proyecto queda delimitado de la siguiente forma:

Arquitectura actual

Como perspectiva a futuro, se propone la siguiente arquitectura integral:



Arquitectura propuesta

Para lograr integrar dicha plataforma, se propone la centralización de la información recibida por múltiples cámaras, ubicadas en diversos puntos de la ciudad. Los datos obtenidos a través de estos dispositivos deben ser reunidos por un sistema, que será denominado Sistema Central de Inteligencia. Este sistema se encargará de la distribución de los datos a diversos componentes de apoyo.

Los datos capturados por las cámaras son almacenados para su posterior análisis, a fin de determinar patrones de comportamiento de los vehículos, y a partir de ello elaborar estrategias que ayuden a mejorar las vialidades. A fin de colaborar en la generación de esta planificación, se propone la inclusión de un componente denominado Máquina de Aprendizaje, que se encargará de la revisión de los datos guardados, y constantemente elaborará estadísticas y buscará patrones; los cuales a su vez, permitirán afinar las estrategias obtenidas, con mayor conocimiento de cada vialidad en particular, y por ende, de lo que es más recomendable realizar en cada caso por separado.

La información procesada por la Máquina de Aprendizaje será observada por los encargados de la plataforma a través de una aplicación de control y monitoreo, en la cual podrán conocer los reportes oportunos de las estrategias de planificación vial, así como obtener el estado actual de las vialidades asociadas a cada una de las cámaras que forman parte del sistema.

Por otro lado, se contará con un sistema servidor de mensajería, que se encargará de proveer información a diversas aplicaciones enfocadas a los ciudadanos, a través de las cuales podrán obtener el reporte del flujo vehicular, y recibir alertas en tiempo real ante cualquier incidencia en la ciudad, como accidentes, bloqueos, o tránsito lento.

La realización de un sistema con estas características permitiría adecuar elementos como semáforos, sentido de las vialidades, e incluso disposición de policías de tránsito, conforme a una estrategia bien definida para mejorar el flujo vehicular de la ciudad.

Referencias

Capítulo 1. Antecedentes y actualidad.

[1] Google Maps. (2014). ***Google Maps***. Recuperado de https://maps.google.com

[2] Boswell, Patrick. (2010). ***How Google Traffic Works***. Recuperado el 10 de agosto de 2013, de http://ezinearticles.com/?How-Google-Traffic-Works&id=5119108

[3] Barth, Dave. (2009). ***The bright side of sitting in traffic: Crowdsourcing road congestion data***. Recuperado el 10 de agosto de 2013, de http://googleblog.blogspot.in/2009/08/bright-side-of-sitting-in-traffic.html

[4] Nokia. (2014). ***HERE Maps – City and Country Maps – Driving Directions – Satellite Views – Routes***. Recuperado de http://here.com

[5] Madrigal, Alexis. (2012). ***The Forgotten Mapmaker: Nokia Has Better Maps Than Apple and Maybe Even Google***. Recuperado el 11 de agosto de 2013, de http://www.theatlantic.com/technology/archive/2012/10/the-forgotten-mapmaker-nokia-has-better-maps-than-apple-and-maybe-even-google/263150

[6] Microsoft. (2013). ***Microsoft to acquire Nokia's devices & services business, license Nokia's patents and mapping services***. Recuperado el 14 de septiembre de 2013, de http://news.microsoft.com/2013/09/03/microsoft-to-acquire-nokias-devices-services-business-license-nokias-patents-and-mapping-services

[7] Waze. (2014). ***Waze***. Recuperado de https://www.waze.com

[8] Lunden, Ingrid. (2013). ***Google Bought Waze For $1.1B, Giving A Social Data Boost To Its Mapping Business***. Recuperado el 21 de septiembre de 2013, de http://techcrunch.com/2013/06/11/its-official-google-buys-waze-giving-a-social-data-boost-to-its-location-and-mapping-business

[9] Secretaría de Obras y Servicios del GDF. (2013). ***072 Atención Ciudadana***. Recuperado de http://www.agu.df.gob.mx/072-atencion-ciudadana

[10] Secretaría de Obras y Servicios del GDF. (2013). ***072 Mapa General***. Recuperado de http://072gdf.force.com/apoyovial

[11] Alcaldía de Medellín. (2013). ***Sistema Inteligente de Movilidad SIMM***. Recuperado de http://www.medellin.gov.co/transito/sistema_inteligente_movilidad.html

Capítulo 2. Arquitectura secuencial y en paralelo.

[1] Simon Cuellar, Yesenia. (2012). ***Taxonomía de Flynn***. Recuperado el 5 de octubre de 2013, de http://prezi.com/vwljawyntz_j/taxonomia-de-flynn

[2] Stallings, William. (2000). ***Organización y arquitectura de computadores***. (Quinta edición). Madrid, España. Prentice Hall.

[3] Pacheco, Peter. (2011). ***An introduction to parallel programming***. (Primera edición). San Francisco, California, Estados Unidos. Morgan Kaufmann.

[4] Culler, David. Pal, Jaswinder. Gupta, Anoop. (1997). ***Parallel Computer Architecture***. (Primera edición). San Francisco, California, Estados Unidos. Morgan Kaufmann.

Capítulo 3. Tecnologías de cómputo gráfico y concurrente.

[1] NVIDIA. (2014). ***Plataforma de Computación Paralela CUDA NVIDIA***. Recuperado de http://www.nvidia.com.mx/object/cuda_home_new_la.html

[2] OpenCV Team. (2014). ***OpenCV***. Recuperado de http://opencv.org

[3] Arévalo, V. González, J. Ambrosio, G. (2004). ***La librería de visión artificial OpenCV. Aplicación a la docencia e investigación***. Recuperado el 7 de diciembre de 2013, de http://mapir.isa.uma.es/varevalo/drafts/arevalo2004lva1.pdf

[4] Bradski, Gary. Kaehler, Adrian. (2008). ***Learning OpenCV: Computer Vision with the OpenCV Library***. (Primera edición). Estados Unidos. O'Reilly.

[5] Intel. (2014). ***Intel Integrated Performance Primitives***. Recuperado de http://software.intel.com/en-us/intel-ipp

[6] Intel. (2009). ***Intel Integrated Performance Primitives for Linux OS on IA-32 Architecture. User's Guide***. Recuperado el 18 de enero de 2014, de http://www.cms.hu-berlin.de/dl/systemservice/computeservice/docs/intel_ipp_ug.pdf

[7] Intel. (2012). ***Using Intel IPP with OpenCV***. Recuperado el 18 de enero de 2014, de http://software.intel.com/sites/default/files/m/d/4/1/d/8/OpenCV_ApplicationNotes.pdf

Capítulo 4. Detección de objetos.

[1] Vilchez, Angel. (2009). ***¿Qué es la detección de rostros?***. Recuperado el 30 de marzo de 2014, de http://www.configurarequipos.com/doc1157.html

[2] F., A. (2014). ***El reconocimiento facial de Facebook, tan preciso que da miedo***. Recuperado el 30 de marzo de 2014, de http://www.abc.es/tecnologia/redes/20140321/abci-facebook-reconocimiento-facial-preciso-201403211737.html

[3] Pino, Daniel. (2012). ***Cómo desbloquear el Samsung Galaxy S3 con el reconocimiento facial***. Recuperado el 30 de marzo de 2014, de http://www.tuexpertomovil.com/2012/08/22/como-desbloquear-el-samsung-galaxy-s3-con-el-reconocimiento-facial

[4] Mazo, Gary. (2012). ***How to use the Smart Stay on the Galaxy S3***. Recuperado el 30 de marzo de 2014, de http://www.androidcentral.com/how-use-smart-stay-galaxy-s3

[5] Ramis, Alberto. (2011). ***Interfaces humanas en los videojuegos***. Recuperado el 30 de marzo de 2014, de http://personales.alumno.upv.es/alrafua/asignaturas/SES/Perifericos/Interfaces_humanas/index.html

[6] Oleaga, Jon. (2014). ***Probamos Leap Motion, el controlador por gestos en 3D para tu ordenador***. Recuperado el 30 de marzo de 2014, de http://www.abc.es/tecnologia/informatica/20140204/abci-probamos-leap-motion-201402041411.html

[7] Atienza, Vicente. (2010). ***El histograma de una imagen digital***. Recuperado el 13 de abril de 2014, de http://riunet.upv.es/bitstream/handle/10251/12711/El%20histograma%20una%20 imagen%20digital.pdf

[8] Escalante Ramírez, Boris. (2006). ***Realce de la imagen***. Recuperado el 13 de abril de 2014, de http://verona.fi-p.unam.mx/boris/teachingnotes/Capitulo4.pdf

[9] Guevara-Díaz, Jorge. (2006). ***Detección de Rostros por medio de las Wavelets de Morlet***. Recuperado el 18 de abril de 2014, de http://www.vision.ime.usp.br/~jorjasso/files/WaveletMorlet.pdf

[10] Pinto, N. Barhomi, Y. Cox, D. DiCarlo, J. (2010). ***Comparing State-of-the-Art Visual Features on Invariant Object Recognition Tasks***. Recuperado el 20 de abril de 2014, de http://pinto.scripts.mit.edu/uploads/Research/soainv.pdf

[11] Murphy, K. Torralba, A. Eaton, D. Freeman, W. (2005). ***Object detection and localization using local and global features***. Recuperado el 20 de abril de 2014, de http://people.csail.mit.edu/torralba/publications/localAndGlobal.pdf

[12] OpenCV Team. (2014). ***Haar Feature-based Cascade Classifier for Object Detection***. Recuperado el 26 de abril de 2014, de http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html

[13] Papageorgiou, C. Oren, M. Poggio, T. (1998). ***A General Framework for Object Detection***. Recuperado el 27 de abril de 2014, de http://pdf.aminer.org/000/293/432/a_general_framework_for_object_detection.pdf

[14] Viola, Paul. Jones, Michael. (2001). ***Rapid Object Detection using a Boosted Cascade of Simple Features***. Recuperado el 3 de mayo de 2014, de https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf

[15] Dalal, Navneet. Triggs, Bill. (2005). ***Histograms of Oriented Gradients for Human Detection***. Recuperado el 4 de mayo de 2014, de http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf

Capítulo 5. Estimación de movimiento.

[1] López, Iván. Prados, Jonathan. (2012). ***Estimación de Movimiento***. Recuperado el 23 de mayo de 2014, de http://www.ilopez.es/proyectos/teoriadelasenal/EstimMov.pdf

[2] Seguridad Informática. (2007). ***¿Qué son los códecs?***. Recuperado el 24 de mayo de 2014, de http://seguinfo.wordpress.com/2007/07/22/%C2%BFque-son-los-codecs-2

[3] Argyriou, V. Vlachos, T. (2006). ***A study of sub-pixel motion estimation using phase correlation***. Recuperado el 24 de mayo de 2014, de http://www.macs.hw.ac.uk/bmvc2006/papers/328.pdf

[4] Torr, Phillip. Zisserman, Andrew. (1999). ***Feature Based Methods for Structure and Motion Estimation***. Recuperado el 25 de mayo de 2014, de http://link.springer.com/chapter/10.1007%2F3-540-44480-7_19

[5] Irani, Michal. Anandan, P. (1997). ***All About Direct Methods***. Recuperado el 25 de mayo de 2014, de http://link.springer.com/chapter/10.1007%2F3-540-44480-7_18

[6] ITU-T. (1994). ***Video Codec for Audiovisual Services at p x 64 kbits***. Recuperado el 31 de mayo de 2014, de http://www.itu.int/rec/T-REC-H.261-199303-I/en

[7] MPEG. (1991). ***ISO CD 11172-3. Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to About 1.5 MBits/s***. Recuperado el 31 de mayo de 2014, de http://cutebugs.net/files/mpeg-drafts/11172-3.pdf

[8] Lin, Yuh-Chuan. Tai, Shen-Chuan. (1997). ***Fast Full-Search Block-Matching Algorithm for Motion-Compensated Video Compression***. Recuperado el 31 de mayo de 2014, de http://web.stanford.edu/class/ee398a/handouts/papers/Lin%20-%20Fast%20Motion%20Compensation.pdf

[9] Patel, Bhavina. Kshirsagar, R. Nitnawate, Vilas. (2013). ***Review and comparative study of motion estimation techniques to reduce complexity in video compression***. Recuperado el 1 de junio de 2014, de http://www.ijareeie.com/upload/2013/august/11_REVIEW.pdf

[10] Chhotray, Santosh. Kannoujia, Dheeraj. Jha, Samir. (2012). ***An efficient block matching algorithm for fast motion estimation using combined three step search and diamond search algorithm***. Recuperado el 14 de junio de 2014, de http://interscience.in/IJCCT_Vol3Iss6-7-8/113-116.pdf

[11] Ning, C. Lo, K. Tang, H. (1996). ***New block motion estimation algorithm for video compression***. Recuperado el 14 de junio de 2014, de http://www.ee.cityu.edu.hk/~ISCE2000/026.doc

Capítulo 6. Monitoreo de flujo vehicular

[1] OpenCV Team. (2014). ***Cascade Classifier Training***. Recuperado el 28 de junio de 2014, de http://docs.opencv.org/doc/user_guide/ug_traincascade.html

[2] Computer Vision Laboratory. (2013). ***Multi-View Car Dataset***. Recuperado el 3 de julio de 2014, de http://cvlab.epfl.ch/data/pose

[3] Center for Biological and Computational Learning at MIT and MIT. (2000). ***CBCL Car Database #1***. Recuperado el 3 de julio de 2014, de http://cbcl.mit.edu/software-datasets/CarData.html

[4] Fisher, Robert. (2011). ***CVonline: Image Databases***. Recuperado el 3 de julio de 2014, de http://homepages.inf.ed.ac.uk/rbf/CVonline/Imagedbase.htm

[5] Rhonda Software. (2009). ***FAQ: OpenCV Haartraining***. Recuperado el 12 de julio de 2014, de http://www.computer-vision-software.com/blog/2009/11/faq-opencv-haartraining

Capítulo 7. Conclusiones y proyección a futuro

[1] 100fps. (2004). ***How many frames per second can the eye see?***. Recuperado el 30 de agosto de 2014, de http://www.100fps.com/how_many_frames_can_humans_see.htm