Sistema de contabilización del tráfico vehicular en áreas urbanas de la ciudad de Quito por el grado de circulación

Almeida Salas, Jonathan Santiago y Guamán Figueroa, Steven Andrés

Departamento de Ciencias de la Computación

Carrera de Ingeniería de Sistemas e Informática

Trabajo de titulación, previo a la obtención del título de Ingeniero en Sistemas e Informática

Sang Gunn Yoo, Ph.D.

17 de enero del 2020

URKUND

Urkund Analysis Result

Analysed Document: TESIS FINAL FINAL.docx (D64788997)
Submitted: 3/3/2020 6:11:00 PM
Submitted By: jbolanos@difusion.com.mx
Significance: 2 %

Sources included in the report:

TesisJosueAlba2020.pdf (D63107650)
https://es.wikipedia.org/wiki/Morfolog%C3%ADa_matem%C3%A1tica
https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_video/py_bg_subtraction/py_bg_subtraction.html#background-subtractionBlippar.
https://www.universidadviu.com/reconocimiento-de-imagenes-software-y-ejemplos

Instances where selected sources appear:

8

Sang Guun Yoo, Ph.D.
Director

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

CERTIFICACIÓN

Certifico que el trabajo de titulación, "**Sistema de contabilización del tráfico vehicular en áreas urbanas de la ciudad de Quito por el grado de circulación**" fue realizado por los señores **Almeida Salas, Jonathan Santiago** y **Guamán Figueroa, Steven Andrés** el cual ha sido revisado y analizado en su totalidad por la herramienta de verificación de similitud de contenido; por lo tanto cumple con los requisitos legales, teóricos, científicos, técnicos y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, razón por la cual me permito acreditar y autorizar para que lo sustente públicamente.

Sangolquí, 17 de enero del 2020

Firma:

SANG GUUN YOO, Ph.D

C.C: 1306853720

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

RESPONSABILIDAD DE AUTORÍA

Nosotros, **Almeida Salas Jonathan Santiago** y **Guamán Figueroa Steven Andrés** con cédulas de ciudadanía n° **1717410672** y **1720282589**, declaramos que el contenido, ideas y criterios del trabajo de titulación**: Sistema de contabilización del tráfico vehicular en áreas urbanas de la ciudad de Quito por el grado de circulación** es de nuestra autoría y responsabilidad, cumpliendo con los requisitos teóricos, científicos, técnicos y metodológicos establecidos por la Universidad de Fuerzas Armadas ESPE, respetando los derechos intelectuales de terceros y referenciando las citas bibliográficas.

Sangolquí, 17 de Enero del 2020

Firmas

Almeida Salas, Jonathan Santiago

C.C: 1717410672

Guamán Figueroa, Steven Andrés

C.C: 1720282589

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

AUTORIZACIÓN DE PUBLICACIÓN

Nosotros **Almeida Salas Jonathan Santiago** y **Guamán Figueroa Steven Andrés**, con cédulas de ciudadanía n° **1717410672** y **1720282589**, autorizamos a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de titulación: **Sistema de contabilización del tráfico vehicular en áreas urbanas de la ciudad de Quito por el grado de circulación** en el Repositorio Institucional, cuyo contenido, ideas y criterios son de mi/nuestra responsabilidad.

Sangolquí, 17 de Enero del 2020

Firmas

Almeida Salas, Jonathan Santiago

C.C: 1717410672

Guamán Figueroa, Steven Andrés

C.C: 1720282589

Dedicatoria

Este trabajo está dedicado quienes han estado a mi lado durante toda mi vida, quienes son mi apoyo incondicional y me brindan la fuerza para seguir adelante cada día, por quienes hoy puedo ser el hombre que he llegado a ser; este trabajo está dedicado a mi madre, a mi padre y mi hermano.

A mi madre demostrarme lo fuerte que puedo ser con su ejemplo de valor y perseverancia frente a la vida. A mi padre, por enseñarme que, aunque la vida te golpee, hay que seguir adelante y no rendirse jamás; además de brindarme las bases que me han servido para desempeñarme en esta hermosa carrera. A mi hermano Alexis, por siempre cuidar de mí, por enseñarme los valores que hacen a un hombre, por guiarme en el camino de la vida y porque sé que siempre podré contar con él como amigo y como hermano

Este logro de mi vida se los dedico a ustedes mi querida familia.

Steven Guamán

El presente trabajo lo dedico a esas personas que Dios y la vida pusieron a mi lado para enseñarme, acompañarme y apoyarme en todo momento: mi familia. Sin ellos no sería el profesional ni mucho menos el hombre que soy.

A mi madre, Sylvia, por ser mi más grande ejemplo de responsabilidad y por guiarme con sus sabios consejos a lo largo de toda mi vida. A mi padre, Santiago, por mostrarme cómo comportarme a la altura de un profesional y enseñarme a afrontar cada reto que se me presente, con valentía y por sobre todo, a ser un hombre de bien. A mi hermano, Daniel, a quien considero una gran persona y mi mejor amigo, y, de seguro a quien podré acudir cuando lo necesite.

Con mucho cariño, Jonathan Santiago.

Jonathan Almeida

Agradecimiento

Esta tesis merece el agradecimiento a quienes sirvieron de apoyo para su realización, y por quienes Jonathan y yo hemos conseguido finalizar esta etapa de nuestras vidas.

Agradezco a mi madre, a mi padre y mi hermano por siempre brindarme su apoyo y guiarme en la vida, porque, aunque hemos pasado por momentos muy difíciles, siempre nos hemos levantado juntos.

Agradezco a mis amigos, quienes pude conocer durante mi etapa en la Universidad, a mi compañera Panela por compartir grandes momentos inolvidables juntos y brindarme su apoyo incondicional, a los PejeLagartos con quienes sin duda hicimos divertida la Universidad. Gracias amigos míos, sin ustedes la carrera no hubiese sido la misma.

Agradezco a quienes aportaron con su granito de arena para poder finalizar esta tesis, a Gaby por darnos un poquito de tu valioso tiempo para ayudarnos a mejorar la redacción, a Nicole por conseguirnos los videos necesarios para las pruebas de funcionamiento.

Finalmente agradezco el apoyo brindado por mis antiguos compañeros de AlterBios de quienes puedo contar hoy como grandes amigos.

Steven Guamán

Palabras me van a faltar para agradecer a todos quienes nos apoyaron no solo en la realización de esta tesis, sino en todo nuestro paso por esta bella pero demandante carrera, y por quienes tanto Steven como yo hemos culminado con éxito esta etapa.

Primeramente, agradezco a Dios por darme la fortaleza, la paciencia y el convencimiento de cumplir una meta más de mi vida; a mis padres y hermano, por su apoyo incondicional, a mis maestros, a los cuales no los menciono por temor a olvidarme de alguno de ellos, pero fueron quienes, con sus sabios conocimientos, supieron guiarme en la culminación de la carrera y sobre todo, en la realización de esta tesis. A la majestuosa Universidad de las Fuerzas Armadas ESPE, por el ser el templo del saber en el cual cimenté los conocimientos necesarios para ahora, con orgullo llamarme Ingeniero.

Quiero agradecer a mis amigos, quienes sin importar la carrera, hicieron inolvidable mi paso por la Universidad. A los “Pejelagartos”, a Solange, a Soledad, a mis amigos de los diferentes recorridos en los que estuve, con quienes sin duda pase los momentos más gratificantes e imperecederos.

No quiero dejar pasar por alto mi profundo y especial agradecimiento a Gabriela, Nicole, Aldo, David, Alisson, quienes fueron parte fundamental en este proceso de titulación.

Y por último, y no menos importante, agradezco el apoyo brindado por las jefaturas y compañeros de ALTERBIOS, empresa a la cual con orgullo pertenezco.

Jonathan Almeida

índice

Índice de tablas

Índice de figuras

Resumen

En la actualidad, controlar los problemas de tráfico vehicular se ha vuelto una prioridad para los gobiernos y municipalidades de distintas ciudades del mundo. Con el pasar de los años, el número de vehículos en circulación en las calles y vías en la ciudad de Quito han aumentado considerablemente, y para que las autoridades correspondientes tomen decisiones al respecto, es necesario que se basen en información confiable y precisa. Por esta razón, en este documento se describe el proceso de desarrollo del prototipo de un sistema de contabilización del tráfico vehicular, basándose en una disciplina denominada visión por computadoras, la cual permite analizar el contenido de videos grabados del tráfico vehicular y procesarlos, utilizando para ello la librería basada en c++ llamada OpenCV aplicada en Pyhton versión 3.7. La necesidad de contar con esta aplicación, nace de la dificultad con la que actualmente se obtiene la información sobre circulación vehicular, donde utilizan procedimientos que consumen tiempo y dinero. El automatizar con una tecnología de este tipo a dichos procedimientos en las áreas urbanas del Distrito Metropolitano de Quito, permite obtener resultados más rápidamente y con mayor precisión, facilitando la toma de decisiones por parte de las autoridades en temas de movilidad vehicular.

Palabras clave:

- **VISIÓN POR COMPUTADORAS**
- **TRÁFICO VEHICULAR**
- **OPENCV**
- **DETECCIÓN DE IMÁGENES**
- **CONTEO VEHICULAR**

Abstract

Actually, controlling vehicular traffic problems has become a priority for governments and municipalities in different cities and countries around the world. Over the years, the number of cars in circulation on the roads and avenues in the city of Quito has increased considerably, and for the authorities corresponding to the decisions in this regard, they need to be based on reliable and accurate information. For this reason, this document describes the development process of the prototype of a vehicle traffic accounting system, identified in a given computer vision discipline, which allows the content of recorded videos of vehicular traffic and processed it to be analyzed. For this purpose, the use of a library based on c++ called OpenCV applied in Pyhton version 3.7 is essential. The need to have this application arises from the difficulty with which the information on the movement of different vehicles is currently obtained, where procedures are used that consume a lot of time and money. The automation of this procedures with such technology in the urban areas of the Metropolitan District of Quito, allows to obtain efficient results, more quickly and with greater precision, facilitating the decision-making by the authorities in matters of vehicular mobility.

Key words:

- **COMPUTER VISION**
- **VEHICULAR TRAFFIC**
- **OPENCV**
- **IMAGE DETECTION**
- **CAR COUNTING**

Capítulo I

Introducción

Antecedentes

En los últimos años, la congestión vehicular se ha vuelto un problema para la mayoría de las ciudades modernas del mundo. Los distintos gobiernos, para intentar dar solución a estos problemas de tráfico, necesitan analizar y procesar información concerniente a la circulación vehicular para realizar una buena planificación. Para ello, es importante que la recolección de esta información sea efectiva y eficiente.

En la actualidad, el procedimiento que se lleva a cabo para contabilizar el número de vehículos que transitan por una calle o avenida, es poco automático, y estos procesos van desde personas que contabilizan los vehículos, hasta sensores que con el tiempo llegan a deteriorarse debido al clima y cambios meteorológicos, necesitando la presencia de al menos una persona que supervise el proceso.

El artículo realizado por Felipe Torres Espinoza, Gabriel Barros, y María José Barros, “Computer Vision classifier and platform for automatic counting: more than cars”, (Torres Espinoza, Barros G., & Barros, 2017) identifica problemas como los descritos anteriormente, así como el alto costo de los sensores y la gran cantidad de información tomada y no procesada.

En un estudio realizado por Fabio Nelli plasmado en el libro Python Data Analytics en el capítulo “Image Analysis and Computer Vision with OpenCV”, explica que actualmente el análisis de imágenes ha experimentado un gran desarrollo debido al Deep learning, pudiendo resolver problemas que antes eran imposibles, creando una nueva disciplina llamada “Visión por computadora” (Nelli, Image Analysis and Computer Vision with OpenCV, 2018).

En el artículo publicado en el sitio Web de la Universidad de Valencia, por su Equipo de expertos, se explica que, aunque la visión de un ordenador permite, por ejemplo, a un robot adquirir el sentido de la

vista, no le garantiza la comprensión del entorno físico que rodea al objeto que se visualiza. Para ello, se plantea el uso de la visión por computadoras, la cual utiliza la inteligencia artificial para reproducir la forma en que el cerebro percibe las imágenes (Valencia, 2019).

Desde hace varios años, el reconocimiento de imágenes utilizando visión por computadora, se ha hecho presente, desde reconocer pequeños objetos, hasta el reconocimiento facial que ha dado lugar a diferentes usos ya sea para controlar el ingreso biométrico a un lugar, o inclusive comprar en tiendas sin necesidad de utilizar efectivo.

Por otro lado, en cuanto al uso de la visión por computadora, en otros sectores o áreas como la circulación vehicular, se ha visualizado a través del reconocimiento de vehículos en estacionamientos o grandes avenidas, pudiendo así, mostrar información del modelo y año de un vehículo, como lo hace la aplicación Blippar (Blippar, s.f.).

Una de las más reconocidas librerías utilizada para visión por computadoras, se denomina OpenCV u Open Source Computer Vision, la cual admite diferentes algoritmos de visión artificial y de aprendizaje automático que el libro de Finis describe en una serie de pasos y ejemplos para su utilización. Así, a través de este documento, se hará uso del mismo, aplicado a un área específica que corresponde al Distrito metropolitano de Quito.

Planteamiento del problema

En la actualidad la congestión vehicular es un inconveniente que continúa agravándose en todas partes del mundo; es por ello, que los gobiernos y municipalidades utilizan como uno de sus principales ejes de política, a la movilidad.

Pavón 2016, mencionará que los problemas de movilidad requieren de soluciones inmediatas, y para ello, se requiere de una buena planificación vehicular, la cual demanda información veraz y precisa. Esta información se refiere principalmente, al número de vehículos que transita por una vía en determinados horarios.

Actualmente estos datos son tomados mediante varios métodos de recolección, siendo algunos de ellos: La utilización de sensores de pavimento, los cuales requieren que sean instalados en un carril, significando el cierre temporal del carril durante la instalación.

El siguiente método es mediante sensores infrarrojos, los cuales funcionan mediante la comparación de la radiación térmica que generan los vehículos que circulan por el carril donde están instalados. Sin embargo, este método representa el más costoso.

Al presente, el método más utilizado, es mediante mangueras o tubos, el cual funciona mediante la presión que genera el paso de los vehículos sobre las mangueras, generando pulsos de aire. Esta forma de conteo de tráfico necesita que al menos una persona supervise el procedimiento para evitar la pérdida de los equipos.

Finalmente, tenemos el procedimiento manual, en el que una o varias personas contabilizan los autos que circulan por una vía y los registran en base a lo que observan (Loza Herrera, 2015).

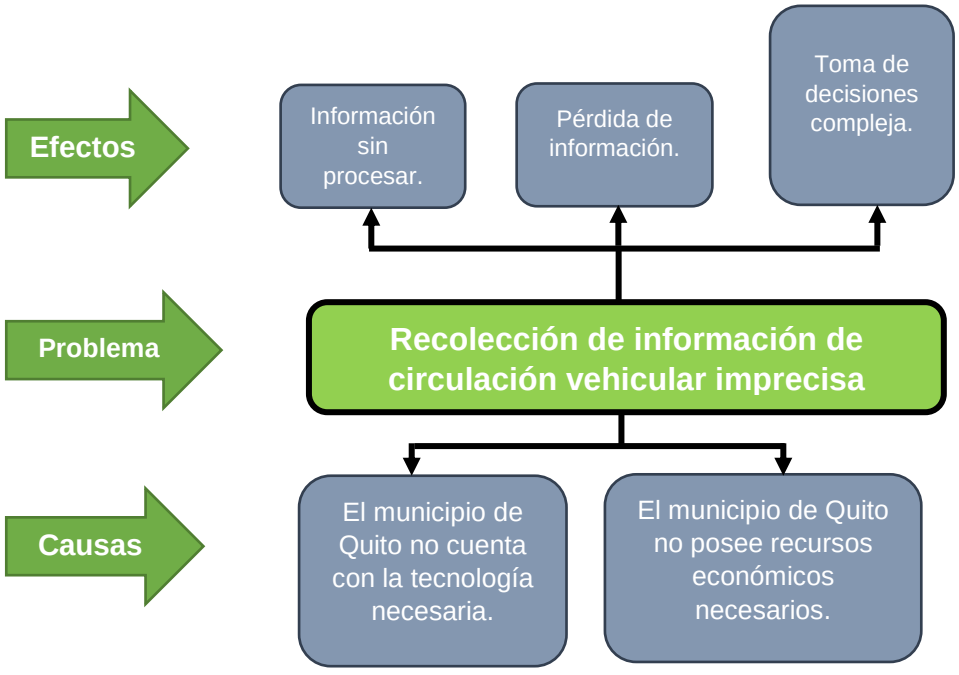
No obstante, los inconvenientes o problemas que traen consigo, se refieren a la pérdida de información, imprecisión de los datos obtenidos por estos procesos, así como los recursos y tiempo humano empleados, generando así, pérdida de recursos humanos y económicos.

De esta manera, para el municipio de Quito, la toma de decisiones sobre movilidad vehicular se torna compleja al no contar con datos confiables. Al mismo tiempo, esta entidad, no cuenta con la tecnología y el presupuesto para realizar el tratamiento de toma y recolección de información, de manera automatizada.

A continuación, en la **Figura 1** se representa el árbol de problemas sobre el tema planteado, con sus respectivas causas y efectos:

Figura 1

Árbol de problemas.



Como se observa en la figura antes descrita, las causas principales por los que la recolección de información de movilidad vehicular no es eficaz, se refieren principalmente, a las falencias del Municipio de Quito al no contar la tecnología ni el presupuesto necesario.

Es así que estas causas provocan gastos en personal humano y tecnológico, además de pérdida de información e información sin procesar, con lo cual los datos obtenidos, provocan resultados alejados a

la realidad.

Justificación

De acuerdo con el informe de INRIX 2018 Global Traffic Scorecard, el cual presenta un análisis de congestión y tendencias de movilidad, Quito (área urbana) es considerada como la segunda ciudad con mayor tráfico del Ecuador; ocupando el puesto 26 a nivel mundial.

La congestión vehicular en la ciudad de Quito tiene repercusiones sociales que afectan a la ciudadanía en general, siendo así que los conductores pierden alrededor de 28 horas al año atascados en el tráfico (El Comercio, 2018), aumentando los niveles de estrés y frustración.

Estos impactos negativos sobre la comunidad social van a requerir de diferentes esfuerzos multidisciplinarios, a cargo del Gobierno y Municipio de Quito, para así, lograr su mitigación.

Durante varios años, el Municipio del Distrito Metropolitano de Quito ha venido desarrollado de manera continua una planificación, a través de guías que permitirían el control y manejo adecuado de la movilidad vehicular; varias de estas guías, se refieren al Plan Maestro de Transporte y Vialidad del 2002, o el Plan Maestro de Movilidad del 2009, entre otros. Sin embargo, y como se ha descrito en el planteamiento del problema, es necesario que la información y datos recolectados, sean precisos y confiables.

De esta manera, la tecnología actual y en base a la cual se desarrolla este documento, permite constatar la realización de todos estos procesos de manera automática; evitando así, el gasto de recursos, presupuesto, ahorrando tiempo, y garantizando que la información obtenida sea confiable y exacta.

La disciplina conocida como Visión por Computadoras se encarga de que una computadora interprete la información que captura visualmente de manera similar a la forma en que lo hace el cerebro humano. Por esta razón, ha sido utilizada en diferentes áreas como el reconocimiento facial y el reconocimiento de objetos.

Así, el presente documento, se centra en el uso de la visión por computadoras en el área de la movilidad

vehicular. La misma que ha sido utilizada en estacionamientos, y en carreteras para reconocer la placa, modelo, marca y otras características de vehículos en circulación o estacionados.

Objetivos

General

Desarrollar un sistema que brinde solución a la problemática de la información inexacta relacionada al conteo vehicular en un área urbana de la ciudad de Quito a través la identificación del tráfico vehicular mediante el uso de la visión por computadoras.

Específicos

i. Entender la situación actual para determinar los métodos, técnicas y tecnologías que se utilizan al presente para medir el tráfico de vehículos.
ii. Generar un software prototipo de conteo vehicular en base a visión por computadoras.
iii. Evaluar el funcionamiento del prototipo identificando la cantidad de vehículos que circulan en un área urbana determinada.

Alcance

Este trabajo comprende el desarrollo del prototipo de una aplicación que contabilice la cantidad de vehículos que transitan en una calle o vía en el área urbana de la ciudad de Quito. El entregable será un archivo ejecutable del prototipo.

El sistema será desarrollado en el lenguaje de programación Python en su versión 3.7, y utilizará como base la librería OpenCV, la cual trabaja con la disciplina denominada Visión por Computadoras, permitiendo que el reconocimiento de vehículos sea eficiente. El prototipo de la aplicación permitirá contabilizar el número de vehículos que circulan en una vía del área urbana de Quito, utilizando como entrada un video grabado.

Capítulo II

Marco teórico

En este capítulo, se analiza detalladamente los fundamentos teóricos utilizados durante el desarrollo del sistema de contabilización del tráfico vehicular.

Inteligencia artificial

Inteligencia artificial se entiende como la ciencia que interrelaciona a la informática con sistemas inteligentes, los cuales son capaces de simular el pensamiento y el comportamiento humano; de modo que, son aptos para razonar, aprender y solucionar problemas mediante el análisis y entendimiento de la información proporcionada. En el capítulo denominado "Artificial Intelligence" del libro Pictorial Data Analys, se mencionan los conceptos básicos de la inteligencia artificial y cómo su uso impacta en la creación de sistemas autónomos (Degano, 1983). Ejemplos de los usos de sistemas inteligentes son los siguientes: Moses (1971), Mathlab (1977) en la matemática, Dendral (1971) en la química;n cada uno de los cuales manejan una gran cantidad de datos para facilitar procesos manuales y poder automatizarlos. No obstante, se especifica que estas herramientas nacidas de la investigación en el área de inteligencia artificial se separan de la misma cada vez más, para formar parte de las ciencias aplicadas en las que se las utilizan, llegando a ser consideradas como herramientas estándar. "Así, la Inteligencia Artificial crece al cambiar sus objetivos inmediatos tan pronto como contribuye exitosamente a la solución de problemas en algún área específica" (Degano, 1983).

Los objetivos con los que se refieren a las capacidades de la inteligencia artificial son los siguientes:

- Razonar la información obtenida del mundo exterior transformándola en conocimiento;
- Manejar dicho conocimiento para poder figurarlo en un modelo interno del mundo;

- Interpretar el modelo interno para que el conocimiento se pueda manipular para interactuar con el mundo exterior.

En el artículo “The Physics and Metaphysics of Computation and Cognition" del libro "Philosophy and Theory of Artificial Intelligence" (C. Müller, 2019), se explica las diferentes comparaciones que se tienen entre el procesamiento de información en un humano (cerebros y mentes) con diferentes sistemas inteligentes. Sin embargo, por más modificaciones de estos sistemas para alcanzar un modelo computacional que simule la mente humana, siempre existe una pequeña dificultad al momento de categorizar ciertas variables. Críticos como John Searle y Hilary Putnam, defienden este punto, indicando que, para una visión del mundo correcta y totalmente real, ciertos objetos intrascendentes, pueden ser interpretados como datos a ser calculados. Dichos datos e información a analizar se encuentran dentro de varios campos de la ciencia y de la vida real. Algunos presentan connotaciones dentro de los campos de la informática, la biología, la psicología, la lingüística, las matemáticas y la ingeniería.

Visión por computadora

En un artículo de DataScience, el Prof. Fei-Fei Li define a la visión por computadoras como "un subconjunto de la inteligencia artificial principal, que se ocupa de hacer que las computadoras o las máquinas estén habilitadas visualmente, es decir, pueden analizar y comprender una imagen". En relación a la visión humana, los ojos de una persona toman aproximadamente una imagen cada 200 milisegundos, mientras que la visión por computadoras comienza con la entrada de datos a la máquina (Mishra, 2019).

La visión por computadoras se puede utilizar de tres formas:

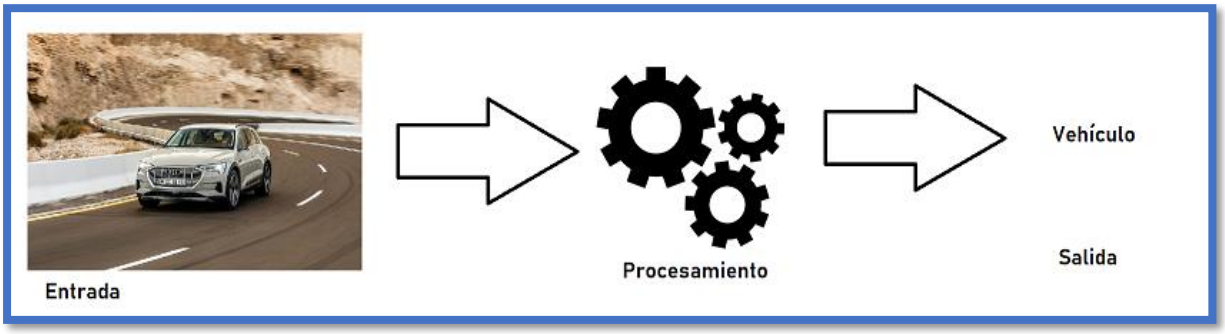
- Sin inteligencia artificial, analizando formas y colores.
- Con aprendizaje automático, aprendiendo de las características.

- Con aprendizaje profundo, aprendiendo solo.

Para la visión por computadoras, la entrada es una imagen que va a ser analizada por el sistema, mientras que la salida es la identificación del objeto en la imagen a analizar; el objetivo es que el computador analice e infiera sobre la imagen de entrada y presente su respectiva interpretación de ella como salida, como la secuencia mostrada en la **Figura 2***:*

Figura 2

Flujo de proceso de visión por computadoras.



OpenCV

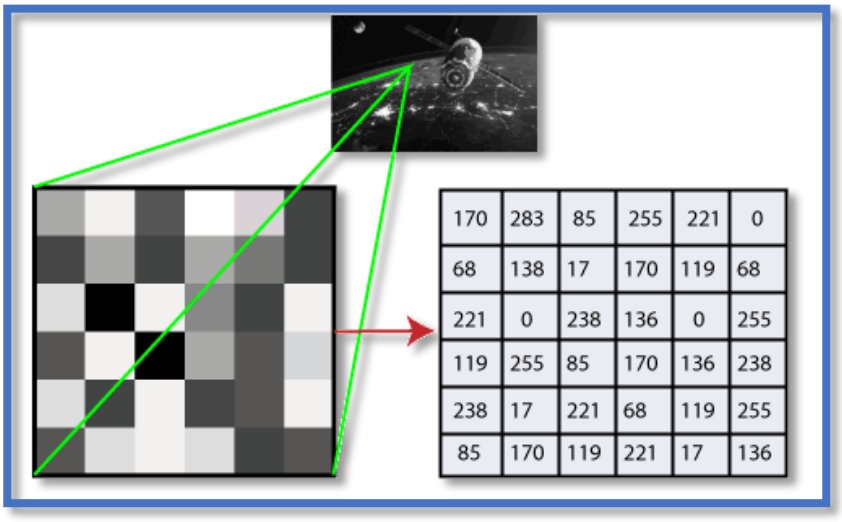
Como se mencionó anteriormente, la visión por computadoras tiene como propósito fundamental comprender el contenido dentro de una imagen identificando objetos de interés de ellas. Intel, vio la necesidad de desarrollar un entorno maneje la visión por computadoras y que sea de sencillo entendimiento, por lo cual, bajo la mano de Gary Bradsky se desarrolló OpenCV en 1999 y fue publicada por primera vez en el año 2000 (K & Mordvintev, 2013).

OpenCV es una librería de código abierto desarrollada en Python, utilizada comúnmente para solucionar problemas que requieren el análisis y procesamiento de imágenes; para lo cual, posee más de 2500 algoritmos. De esta manera, OpenCV funciona como un intérprete entre la imagen o video y el computador. Esto lo hace mediante el análisis individual de cada pixel de la imagen o video. Un píxel es la

unidad más pequeña de una imagen digital que se pueden mostrar y representar en un dispositivo. Así, se revisa una ubicación específica dentro de la imagen, la cual es analizada pixel por pixel. Un ejemplo del uso de OpenCV es el análisis por medio de escala de grises, la cual se encuentra ejemplificado en la **Figura 3.**

Figura 3

Transformación de pixeles de la imagen en números (JavaTpoint, s.f.)



Una de las librerías necesarias para el manejo correcto de OpenCV en Python es Numpy, la cual es una librería optimizada para operaciones numéricas dentro del entorno de Python, aportando una sintaxis similar a la que se encuentra en MATLAB.

Operaciones de opencv

Kernel

Para el procesamiento de imágenes, la visión por computadoras se basa en un concepto llamado filtrado de imágenes. Un filtro se conoce como un algoritmo cuya entrada es una imagen I(x,y), y calcula una nueva imagen I'(x,y), en donde se genera para cada pixel (x,y) en I' una función de los píxeles en I, que se localizan en una pequeña área alrededor de la misma ubicación (x,y). La plantilla que define la forma

de esta pequeña área, así como la forma en que se combinan los elementos de ella, se denomina kernel (núcleo).

Cambio de espacios de color

En el procesamiento de una imagen, un paso importante es convertir el espacio de color de una imagen, para posteriormente utilizar los algoritmos de detección de objetos y contabilización, pues es más óptimo trabajar con imágenes en blanco y negro. Esto permite identificar de mejor manera el fondo de la imagen y los objetos presentes en ella.

Existen varios métodos de cambio de espacios de color que OpenCV nos permite utilizar. El proceso que se sigue es separar un video en fotogramas y sobre cada uno cambiar el espacio de color, de modo que, al aplicar el cambio de espacios de color sobre un video, se obtiene el mismo video cambiado los espacios de color.

Sustracción de fondo

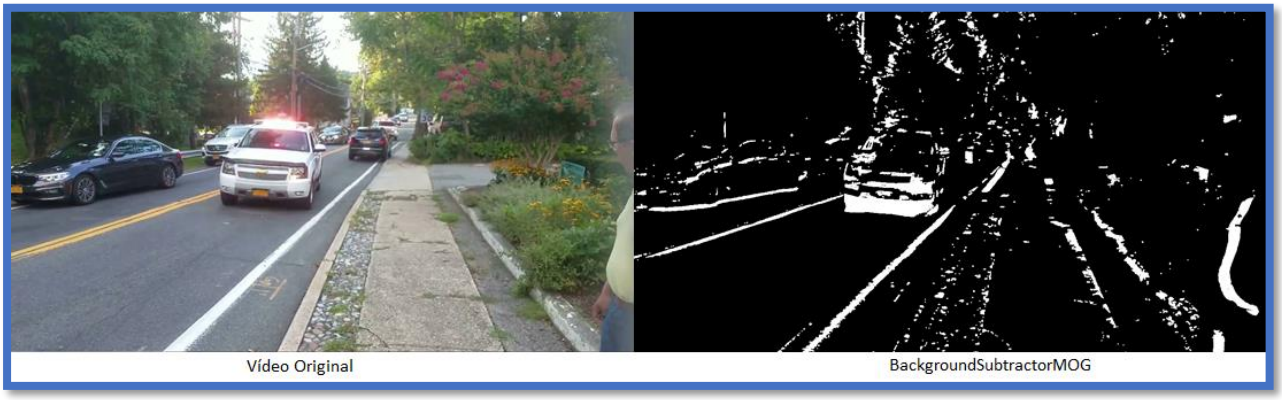
Otro de los principales pasos aplicados en diferentes aplicaciones de visión por computadora es el Background Subtraction o Sustracción de Fondo en español, el cual se basa en el tratamiento del video separado en fotogramas. El algoritmo se encarga de procesar el fondo del fotograma, manteniendo los objetos de interés en un primer plano. Esta técnica, se encarga de generar una imagen binaria que contiene los píxeles pertenecientes a los objetos en movimiento del video. Para hacer este procedimiento, el algoritmo resta el fotograma actual con todo lo que considere como fondo, en base a las características de la escena del video. Esta sustracción se podría considerar la base fundamental para ciertos procesos de preprocesamiento que se manejan en diversas aplicaciones que utilizan visión por computadoras (Background Subtraction, 2013).

Sin embargo, la principal complicación que se puede encontrar, es cuando en el video aparecen ciertos factores como el caso de un objeto moviéndose bruscamente, o la presencia de sombras; para solventar dicha problemática, OpenCV cuenta con diversas variaciones del algoritmo BackgroundSubstractor, a continuación, se describen algunos de ellos:

BackgroundSubtractorMOG: Es un algoritmo basado en la mezcla gaussiana, con el que se modela cada pixel del fondo de la imagen mediante distribuciones k Gaussian. Con este método los colores del fondo se mantienen estáticos. A continuación, se utiliza el siguiente fragmente de código para utilizar la función *cv2.createBackgroundSubtractorMOG(),* la cual tiene parámetros predeterminados, pero son modificables para perfeccionar la sustracción del fondo, como lo son la longitud del historial y el número de k (mezclas gaussianas). Para finalizar, se aplica el método backgroundsubtractor.apply() dentro de la sección de video a analizar, y se obtiene la máscara del primer plano, con el fondo eliminado. En la **Figura 4** mostrada a continuación, se presenta un ejemplo de la aplicación del BackgroundSubstractorMOG.

Figura 4

Sustracción del fondo mediante MOG



BackgroundSubtractorKNN: Este algoritmo parte de BackgroundSubtractorMOG y mantiene la particularidad de ser un algoritmo de segmentación de fondo basado en la mezcla gaussiana. La característica principal es la selección del número apropiado de distribución gaussiana para cada píxel, a diferencia del anterior algoritmo que toma un valor k a lo largo del algoritmo. Esta diferencia brinda la ventaja a adaptarse a diferentes escenarios, tomando para ello los cambios de iluminación. Este algoritmo utiliza el método KNN (k-nearest neighbors, k vecinos más cercanos en español) para mejorar la estimación. En la **Figura 5** mostrada a continuación, se presenta un ejemplo de la aplicación del BackgroundSubstractoKNN.

Figura 5

Sustracción del fondo mediante KNN

BackgroundSubtractorGMG: En este caso, se basa en un algoritmo probabilístico de segmentación para separar los objetos ubicados en el primer plano, utilizando inferencia bayesiana. Al igual que BackgroundSubtractorKNN, este algoritmo se adapta a la iluminación del entorno para tener mayor ponderación; además, utiliza un concepto denominado filtrado morfológico o morfología matemática, el cual se encarga de la eliminación del ruido no deseado en el video.

BackgroundSubtractorGMG realiza el mismo proceso de los dos algoritmos anteriormente mencionados; es decir, elimina el fondo y realza el objeto a analizar. Sin embargo, la diferencia radica en que toma al video y elimina el ruido utilizando la mencionada apertura morfológica. En la **Figura 6** mostrada a continuación, se presenta un ejemplo de la aplicación del BackgroundSubstractorGMG.

Figura 6

Sustracción del fondo mediante GMG

BackgroundSubtractorMOG2: Este algoritmo es una evolución del BackgroundSubtractorMOG, y también está basado en mezcla gaussiana. La particularidad de este algoritmo viene dada porque selecciona el número apropiado de distribución gaussiana para cada pixel del fondo. Básicamente, se basa en la cantidad de tiempo que los colores permanecen en escena. Lo que intenta el algoritmo es identificar el fondo de manera más precisa, basándose en la idea que mientras más tiempo permanezca el color, existe mayor probabilidad de que sea parte del fondo. Este algoritmo se adapta mejor que otros a la variación en la iluminación. En la **Figura 7** mostrada a continuación, se presenta un ejemplo de la aplicación del BackgroundSubstractoMOG2.

Figura 7

Sustracción del fondo mediante MOG2

Transformaciones morfológicas

Las operaciones morfológicas son simples operaciones que se basan en la forma de las imágenes para procesarlas. El flujo del procedimiento de estas transformaciones es, ingresar una imagen como entrada, a continuación, se procesa esta imagen y se genera una nueva imagen de salida. Además, se debe agregar en la entrada el kernel, que decide la naturaleza de la operación a realizar con la imagen de entrada.

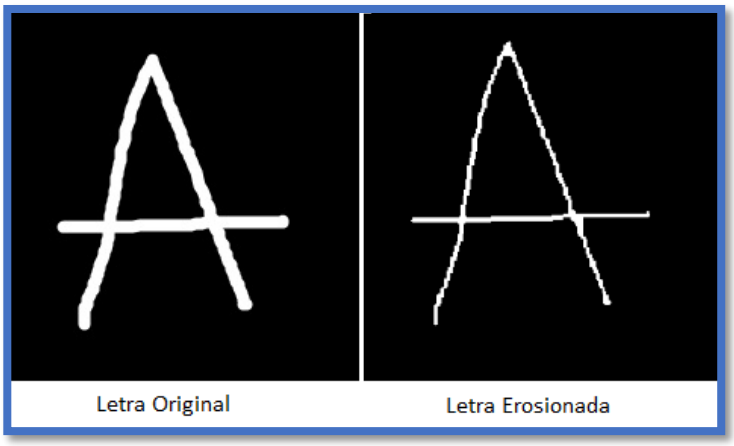
Dos de las transformaciones morfológicas, utilizadas principalmente para limpiar el contenido de una imagen son erosión y dilatación, las cuales se explican a continuación.

Erosión: Esta transformación morfológica parte de la idea de la erosión del suelo, de manera que en una imagen erosiona los límites del objeto en primer plano. El kernel, debe ser una matriz impar (3,5,7) que se desliza a través de la imagen, y actúa de forma que, un pixel de la imagen original, cuyo valor puede ser ‘1’ o ‘0’, se considerará ‘1’ si y solo si, todos los píxeles debajo del núcleo son ‘1’; de no ser así, se considerará ‘0’; es decir, se erosiona. De esta manera, los tamaños de los objetos en primer plano se reducen. Con una imagen convertida en blanco y negro, se puede decir que la región blanca disminuye.

La erosión es útil para eliminar ruido en la imagen, como objetos que no nos interesan detectar (OpenCV, s.f.). En la **Figura 8** mostrada a continuación, se presenta un ejemplo de la aplicación del Erosión a una imagen.

Figura 8

Ejemplo de Erosión en imágenes

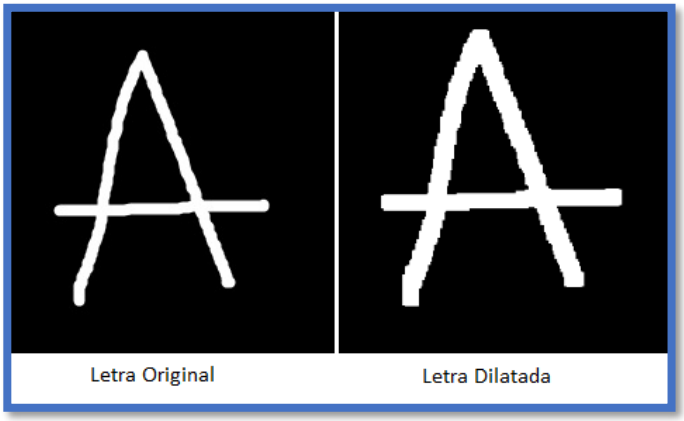


Dilatación: La dilatación es lo contrario de la erosión. Funciona de forma que, si existe un pixel de valor '1', y al menos un pixel debajo de núcleo es '1', aumentará el tamaño del objeto en primer plano; es decir, de la región blanca en la imagen.

Es común que, cuando se intenta limpiar el ruido de una imagen, se aplique erosión, pero esta reduce el tamaño del objeto, para lo cual, se recomienda dilatar; una vez eliminado el ruido, se puede aumentar el tamaño del objeto mediante la dilatación. Además, la dilatación es útil para unir partes rotas de un objeto (OpenCV, s.f.). En la **Figura 9** mostrada a continuación, se presenta un ejemplo de la aplicación del Dilatación a una imagen.

Figura 9

Ejemplo de Dilatación en imágenes

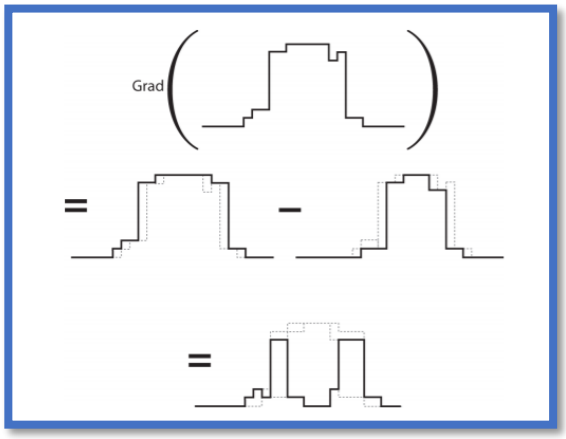


Gradiente morfológico

El gradiente morfológico se utiliza para aislar los perímetros de regiones brillantes de una imagen, con la finalidad de tratarlos como objetos completos. En la **Figura 10** se muestra cómo funciona la aplicación del gradiente morfológico. Además, es necesario recalcar que el gradiente es el resultado de restar la dilatación de la erosión.

Figura 10

Gradiente Morfológico (Kaehler & Bradski, 2017).



Construcción del kernel

En las operaciones morfológicas, lo más común es utilizar kernels cuadrados de 3x3. Pero, se puede crear una matriz del tamaño que se desee, para ser utilizada como elemento estructurante; aunque esto representa a menudo más trabajo del necesario. OpenCV permite el manejo de matrices de diferentes tamaños, sin importar si son cuadradas o no, mediante la función *getStructuringElement()*; este método recibe dos parámetros, el primero, especifica la forma básica que utilizará el elemento, el cual puede ser rectangular, elíptico o en cruz. El segundo parámetro es un objeto que contiene ksize y anchor, los cuales especifican el tamaño del elemento y la ubicación del punto de anclaje respectivamente (Kaehler & Bradski, 2017).

Capítulo III

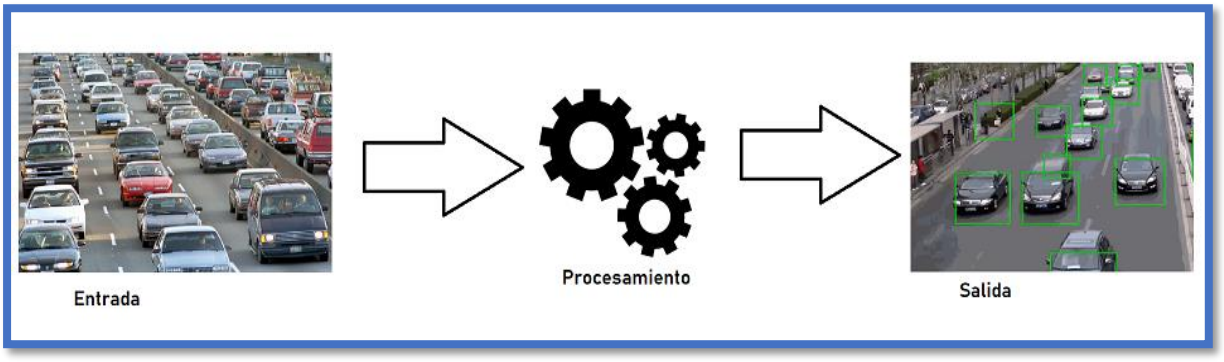
Implementación y desarrollo

El sistema desarrollado durante esta tesis, fue desarrollado en el lenguaje de programación Python 3.7, manejando como principal herramienta la librería OpenCV, para el uso de visión por computadoras.

El flujo de proceso del sistema es el mostrado en la **Figura 11**, donde se especifica que, a partir de un video de entrada, se obtiene como salida un video presentando los objetos detectados.

Figura 11

Flujo de proceso del prototipo.



Para el desarrollo de esta tesis, se seleccionó la metodología de desarrollo ágil conocida como Desarrollo Iterativo, la cual se caracteriza por capturar de mejor manera los requisitos cambiantes, y divide un proyecto en varias iteraciones, cada una produciendo un producto completo (Tinoco Gómez, Rosales López, & Salas Bacalla, 2010). De esta manera, esta tesis atravesó por seis iteraciones; las cuales se explican en este capítulo.

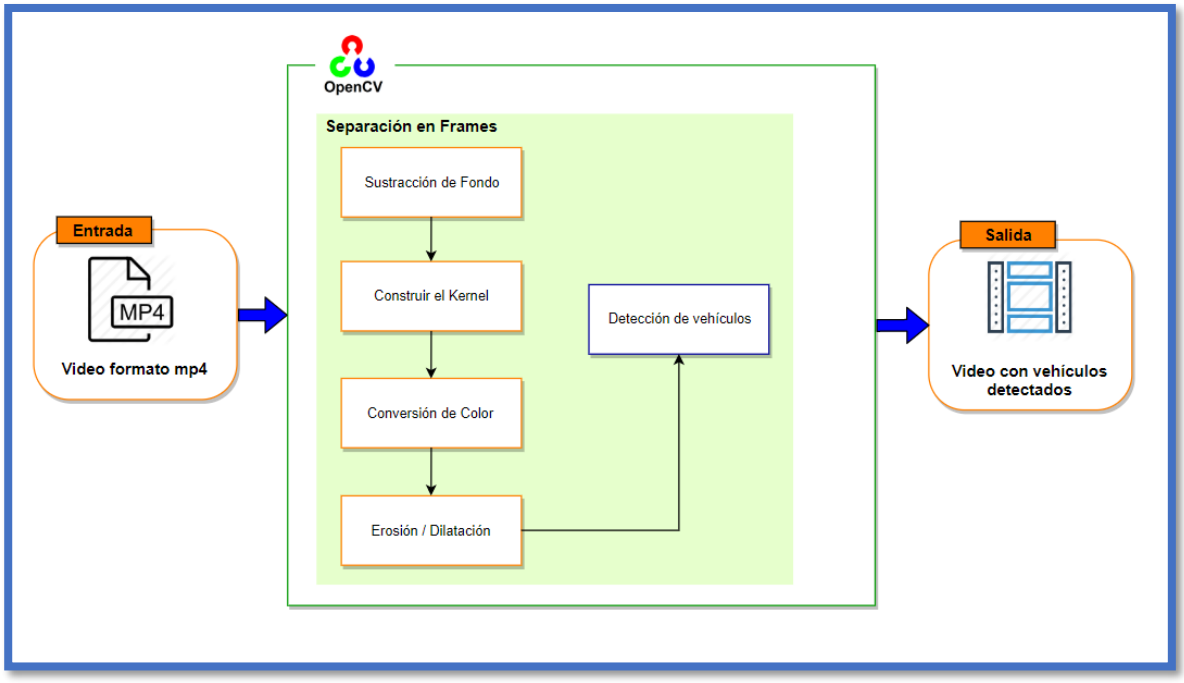
Primera iteración

Arquitectura

En primera instancia, en la primera iteración, se consideró necesario se planteó la siguiente arquitectura, detallada en la **Figura 12**.

Figura 12

Arquitectura del Proyecto (Primera Iteración).



Como se puede observar en la **Figura 12**, la solución planteada, propone identificar vehículos a partir de videos utilizados como entrada, con un proceso interno que separa el video en frames, tratando a cada frame como una imagen. A continuación, se sustrae el fondo de cada imagen, se convierte el color en blanco y negro, permitiendo a un computador identificar objetos de manera más fácil, en este caso vehículos. Posteriormente, se aplican métodos de dilatación y erosión, para que la imagen quede lo más limpia posible y libre de ruido, para que la detección de vehículos sea lo más precisa posible. Finalmente,

una vez tratado el frame, se procede a detectar los objetos dentro y se grafica un cuadrado alrededor del objeto detectado.

En la primera iteración, lo más importante es elegir el algoritmo adecuado que sustraiga el fondo de cada frame de manera más óptima. Pero antes, es necesario identificar el tiempo de duración de los videos con los cuales se harán las pruebas necesarias durante el desarrollo del proyecto.

Tiempo de duración de los videos de prueba

Para establecer el tiempo promedio de duración de los videos utilizados para realizar las pruebas, se tomó como base la duración de las muestras utilizadas en (Ferreira Junior, 2013), (Tang, Do, Ba Dinh, & Ba Dinh, 2012), (Lan, Jiang, Fan, Zhang, & Yu, 2016) , los cuales utilizan videos de 15 a 60 segundos en el primer documento; 2 minutos en el segundo documento; 2, 3 y 6 minutos separados en fragmentos de 8.5 segundos aproximadamente en el tercer documento. Por lo cual realizamos la media entre estos tiempos para determinar el tiempo adecuado a usar dentro de las pruebas de funcionamiento.

- Media de tiempo mostrado en el primer documento (MT1): 37,5 segundos.
- Media de tiempo mostrado en el segundo documento (MT2): 2 minutos (120 segundos).
- Media de tiempo mostrado en el tercer documento (MT3): En este caso específico, se muestran 6 intervalos de tiempo, en los que se tomó cada muestra dividida por la cantidad de frames procesados, esta media de tiempo da como resultado 8.18 segundos.

Ecuación 1. Fórmula de la media de tiempo a utilizar en pruebas de rendimiento

$$Media\ a\ utilizar\ \ (en\ segundos): \frac{MT1 + MT2 + MT3}{3}$$

El resultado de la aplicación de la **Ecuación 1**, es un tiempo de 55.22 segundos. Por lo tanto, se decidió tomar un rango de tiempo que contemplara la media propuesta por los artículos anteriormente

mencionados, dando así un rango de entre 30 a 60 segundos de duración de cada video para realizar las pruebas de funcionamiento de los métodos de análisis de videos proporcionados por las librerías de OpenCV.x

Construcción del kernel

En esta iteración, se utilizó un kernel cuadrado de tamaño 3x3, con forma de elipse, con un ksize y anchor de tamaño 3 cada uno. Se utilizó la línea de código mostrada en la **Figura 13**:

Figura 13

Construcción del kernel.

```
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3))
```

Selección del algoritmo de sustracción de fondo

Para el análisis de la selección del algoritmo de sustracción de fondo, se presenta la **Figura 14**, una captura de video en su formato original, para después ser tratado con los algoritmos.

Figura 14

Captura de video.

BackgroundSubtractorMOG. Para la implementación de este algoritmo, es necesario presentar el código en Python que permite sustraer el fondo del video de entrada. En la **Figura 15** se muestra la codificación del algoritmo.

Figura 15

Codificación del algoritmo BackgroundSubtractorMOG

```
import numpy as np
import cv2
cap = cv2.VideoCapture( 'video_name.avi' )
fgbg = cv2.bgsegm.createBackgroundSubtractorMOG ()
while(1):
   ret , frame = cap . read ()
   fgmask = fgbg.apply(frame)
   CV2.imshow( 'frame' , fgmask )
   k = cv2.waitKey(30) & 0xff
   if k == 27 :
      break
cap.release ()
cv2.destroyAllWindows()
```

Resultado de la aplicación del algoritmo, se obtuvo el video con las siluetas de los objetos a analizar en un primer plano, mientras que el fondo fue totalmente eliminado. A continuación, en la **Figura 16**, se presenta la captura del fragmento de video aplicado el algoritmo.

Figura 16

Captura de video aplicando BackgroundSubtractorMOG.

BackgroundSubtractorKNN. De la misma manera que el anterior algoritmo, en la **Figura 17**, se presenta un fragmento de la codificación del BackgroundSubtractorKNN.

Figura 17

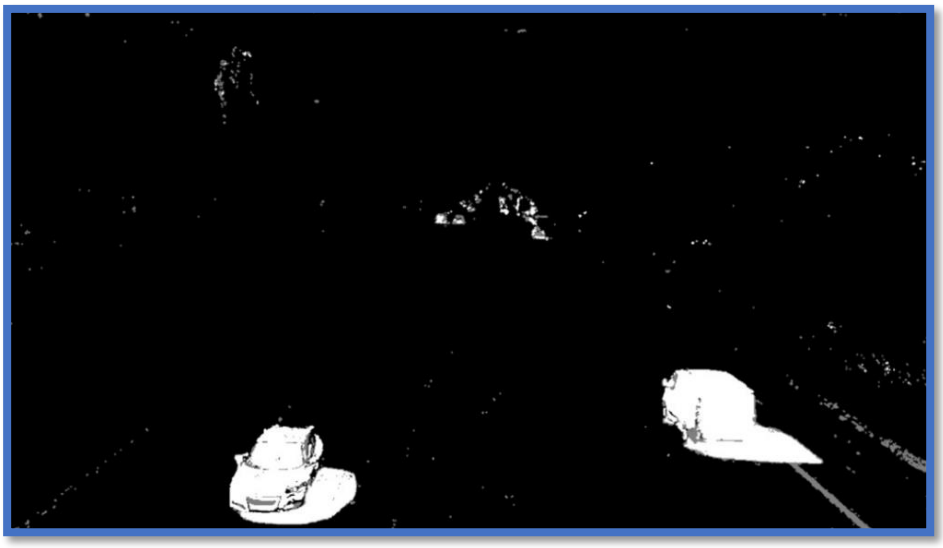
Codificación del algoritmo BackgroundSubtractorKNN.

```
import numpy as np
import cv2
cap = cv2.videoCapture('video_name.avi')
fgbg = cv2.createBackgroundSubtractorKNN()
while(1):
   ret , frame = cap.read()
   fgmask = fgbg.apply(frame)
   CV2.imshow ('frame' , fgmask)
   k = cv2.waitKey(30)&0xff
   if k == 27 :
      break
cap.release()
cv2.destroyAllWindows()
```

Al igual que con el anterior método, se obtiene los objetos de análisis en el frente con un fondo blanco, mientras desecha todo el fondo. La diferencia es que, con este método se muestra también las sombras proyectadas por los objetos. En la **Figura 18** se presenta la captura del fragmento de video aplicado el algoritmo.

Figura 18

Captura de video aplicando BackgroundSubtractorKNN.



BackgroundSubtractorGMG. Como se presentó en los métodos anteriores, en la **Figura 19**, se muestra un fragmento de la codificación básica del algoritmo.

Figura 19

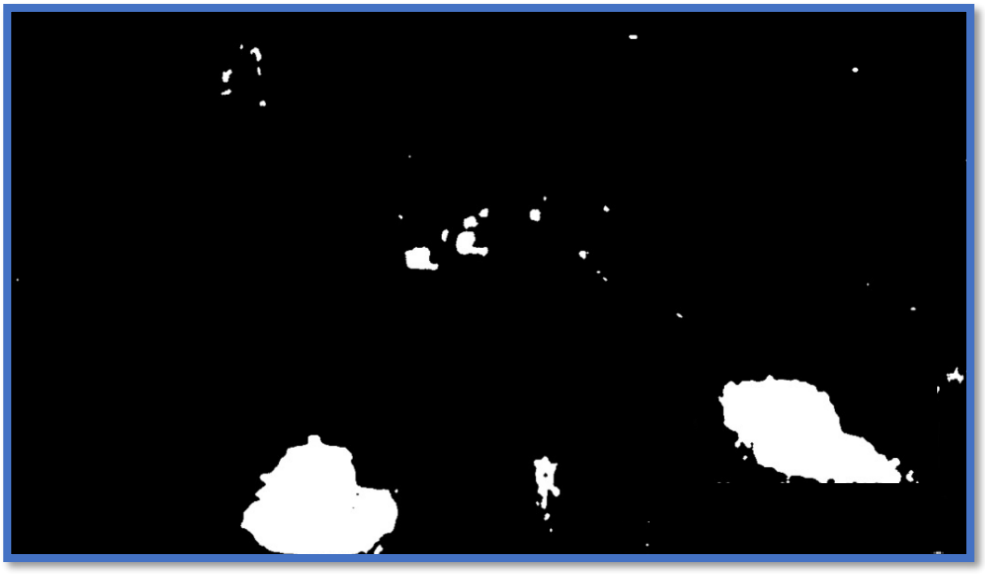
Codificación del algoritmo BackgroundSubtractorGMG.

```
import numpy as np
import cv2
cap = cv2.videoCapture( 'vtest.avi' )
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE , ( 3 , 3 ))
fgbg = cv2.bgsegm.createBackgroundSubtractorGMG ()
while ( 1 ):
   ret , frame = cap.read()
   fgmask = fgbg . apply(frame)
   fgmask = cv2.morphologyEx(fgmask , cv2 . MORPH_OPEN , kernel)
   CV2.imshow (frame , fgmask)
   k = cv2.waitKey(30) & 0xff
   if k == 27 :
      break
cap.release()
cv2.destroyAllWindows()
```

En la **Figura 20** se presenta la captura del fragmento de video aplicado el algoritmo BackgroundSubtractorGMG.

Figura 20

Captura de video aplicando BackgroundSubtractorGMG



Para determinar el algoritmo más adecuado para el desarrollo del sistema final, se ha tomado en consideración realizar pruebas de detección de vehículos aplicando los tres diferentes algoritmos (MOG, KNN, GMG) a tres diferentes videos de duración de entre 30 y 60 segundos con diferentes perspectivas.

En el **Anexo 1**, se compara los vehículos detectados por los tres diferentes algoritmos, frente al valor real de vehículos que circulan en el video. Después de analizar los resultados, podemos determinar que el algoritmo MOG presenta un menor error relativo de vehículos detectados, presentando un menor porcentaje de error relativo frente a los otros dos algoritmos con un 12.01% frente a 26,70% y 18,97% de KNN y GMG respectivamente. Además, se pudo identificar errores de detección como falsos positivos al detectar objetos que no son vehículos, falsos negativos al no detectar algunos vehículos en circulación, y múltiples detecciones al detectar varias veces al mismo vehículo durante la reproducción del video.

En base al análisis realizado en el **Anexo 1**, se determina que el algoritmo MOG es el más adecuado para realizar el sistema de detección de vehículos.

Cambio de espacios de color

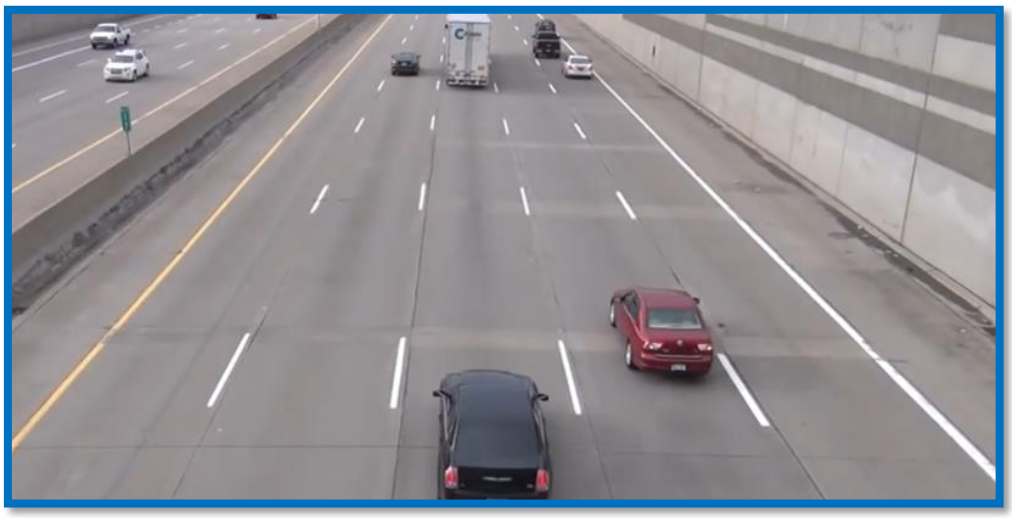
OpenCV ofrece una extensa cantidad de métodos para convertir los colores de las imágenes. Entre estos métodos, dos de los más conocidos son: BGR □ Gray y BGR □ HSV.

En Python, se utiliza la función cv2.cvtColor para convertir los colores del fotograma a analizar. Para el caso de BGR □ Gray, se utiliza la bandera cv2.COLOR_BGR2GRAY, y para BGR □ HSV, se utiliza la bandera cv2.COLOR_BGR2HSV.

A continuación, en la **Figura 21** se presenta el fotograma utilizado para realizar las pruebas de funcionamiento de los métodos de conversión de espacios de color:

Figura 21

Fotograma utilizado para pruebas de conversión de color.



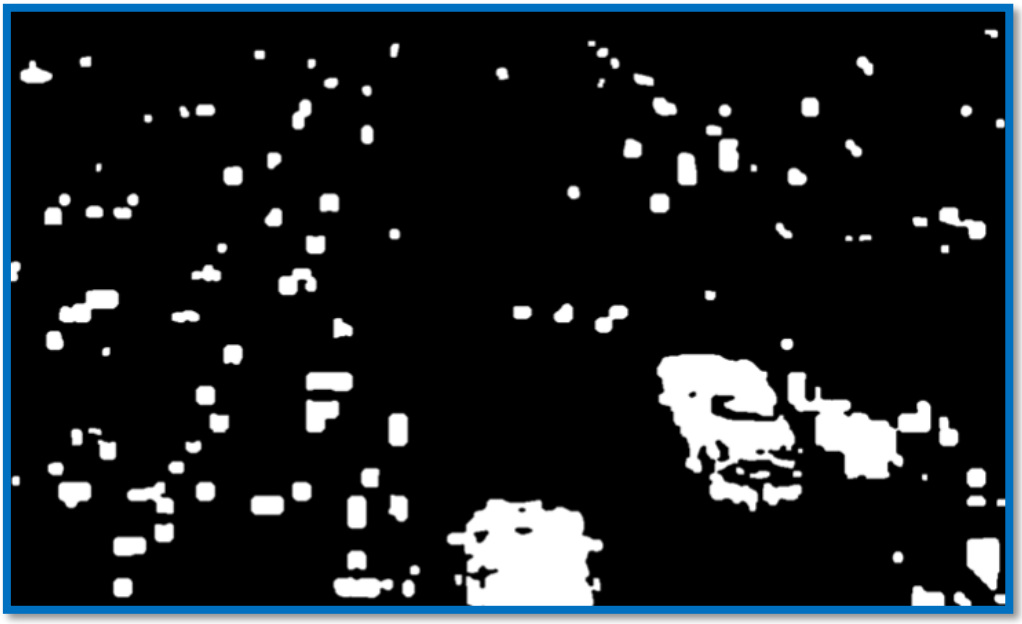
En la **Figura 22** y **Figura 23**, se muestra el funcionamiento de los dos métodos aplicados sobre el fotograma utilizado para pruebas.

Figura 22

Aplicación del método BGR Gray

Figura 23

Aplicación del método BGR HSV

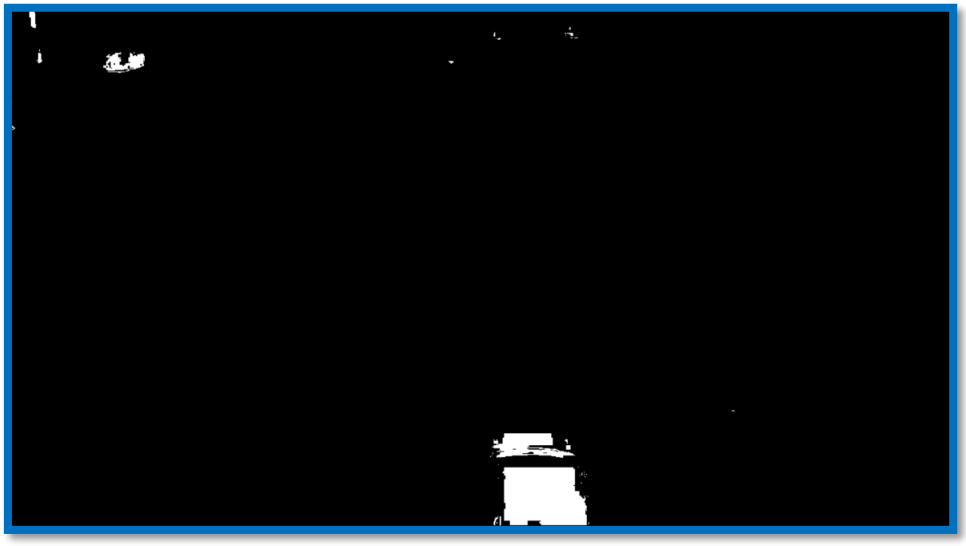


Después de la aplicación de los dos métodos mencionados para cambio de espacios de color, se determinó, que BGR Gray es el método adecuado para utilizar en el prototipo, puesto que presenta una menor cantidad de ruido en la imagen; es decir, para nuestra aplicación de detección de vehículos, detecta de mejor manera los objetos que necesitamos reconocer en pantalla.

Adicionalmente, el método BGR HSV permite la recepción de parámetros para mejorar la detección de objetos y reducir el ruido. La aplicación del método utilizando estos parámetros, se puede visualizar en la **Figura 24**, donde podemos observar que se reduce considerablemente el ruido en la imagen, pero también se ve afectada la detección de los demás objetos. Por esta razón, se ratifica la decisión de seleccionar el método BGR Gray para cambio de espacios de color.

Figura 24

Aplicación del método BRG HSV con parámetros.



Erosión y dilatación

Para el tratamiento de los frames, es necesario dilatar o erosionar la imagen para mejorar la identificación de los objetos en ella, puesto que permiten limpiar el ruido encontrado la imagen o perfeccionar los objetos encontrados.

Con la erosión, se permite eliminar el ruido, pero a su vez disminuye el tamaño del objeto; además, es factible para la separación de objetos. De forma contraria, la dilatación, permite completar objetos incompletos, y aumentar el área de los elementos encontrados.

Para el caso de prueba se utiliza la **Figura 21**, debido a que el proceso de dilatación y erosión viene de la mano del cambio de espacios de color por ser un procedimiento directamente previo a este.

En primer lugar, se procede a probar la aplicación de dilatación sobre la imagen, presentando el resultado mostrado en la **Figura 25**:

Figura 25

Aplicación de dilatación.

A continuación, se prueba la aplicación de erosión sobre la imagen, presentando el resultado mostrado en la **Figura 26**:

Figura 26

Aplicación de Erosión.

Como se puede apreciar en las **Figuras 25** y **26**, la dilatación representa una mejor interpretación de los vehículos en la imagen. Para intentar mejorar el resultado, se procede a aplicar la erosión, para eliminar ruido, seguido de dilatación para completar el área afectada por la erosión aplicada. El resultado se aprecia en la **Figura 27**.

Figura 27

Aplicación de Dilatación y Erosión.

Como se puede observar en la **Figura 27**, el resultado distorsiona la presentación de los vehículos en la imagen. Por esta razón, se ha seleccionado utilizar únicamente la dilatación de la imagen, para obtener mejores resultados.

Detección del objeto

Para la detección de los objetos de interés, se planteó unas medidas mínimas que debe tener el objeto en cuestión para ser considerado como vehículos. Se debe tener en cuenta que la detección se hace luego de realizar las operaciones de erosión o dilatación, así que la detección se hace sobre la transformada del video original. Para efectos prácticos, se optó por inicializar la altura y anchura mínima que un objeto debe tener para ser considerado como vehículo en 80 pixeles para cada una. Por lo que, si

el objeto tiene sus dimensiones menores a las determinadas, se pasara automáticamente al siguiente frame sin graficar visualmente el rectángulo, caso contrario, se grafica sobre el frame con el código mostrado a continuación en la **Figura 28**:

Figura 28

Creación del rectángulo alrededor del objeto

```
cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)
```

Siendo x, y las dimensiones que posee el objeto detectado que previamente pasaron una validación. Este proceso se realizó en cada uno de los frames que cumplan con las características anteriormente explicadas.

Pruebas de funcionamiento

En esta iteración, las pruebas de funcionamiento se realizaron sobre cinco videos. Tres de estos videos son: **Video, Video1** y **Video2**, los cuales son videos grabados en el día, sin presencia de lluvia. Además, los otros dos videos son: **Video3** y **Video4**, los cuales son videos grabados en la noche, estos últimos, presentaron varios inconvenientes debido a que al existir presencia de luz de los vehículos y de los postes de luz, se detecta como objetos a la luz en el video. Las capturas del funcionamiento de esta iteración, se puede apreciar en la **Figura 29**, **Figura 30**, **Figura 31**, **Figura 32** y **Figura 33.**

Figura 29

Prueba de Funcionamiento (Video.mp4)

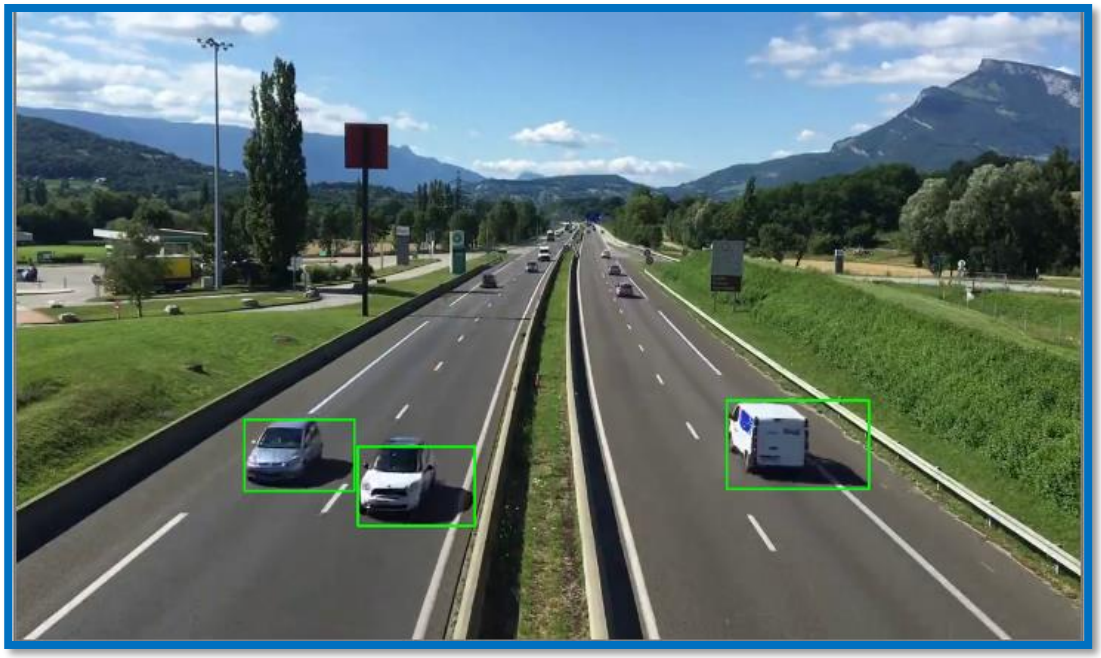


Figura 30

Prueba de Funcionamiento (Video1.mp4)

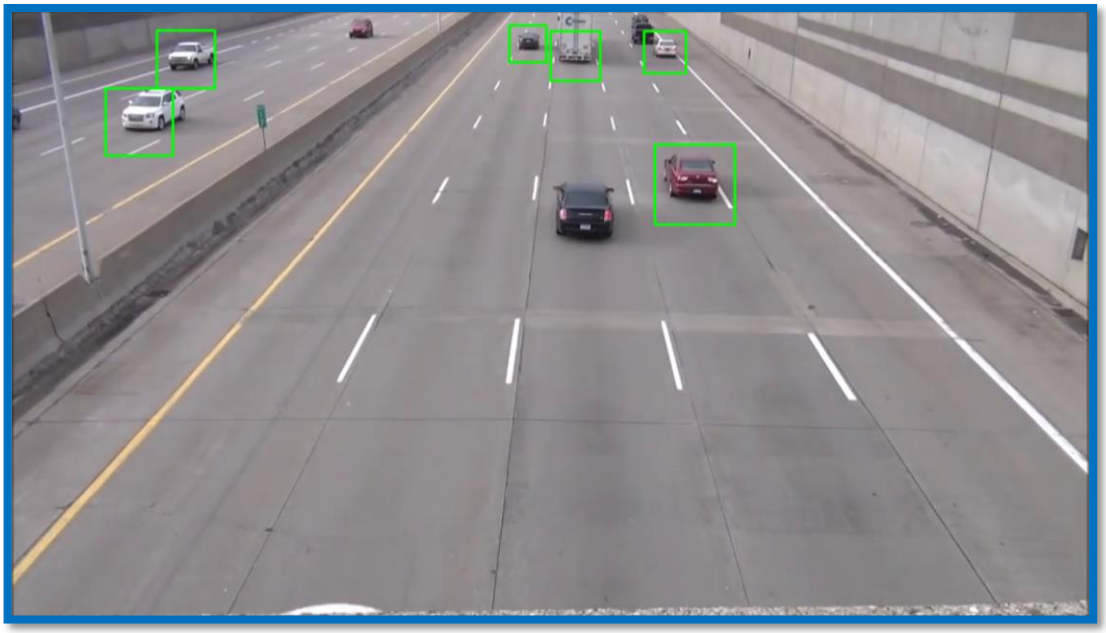


Figura 31

Prueba de Funcionamiento (Video2.mp4)

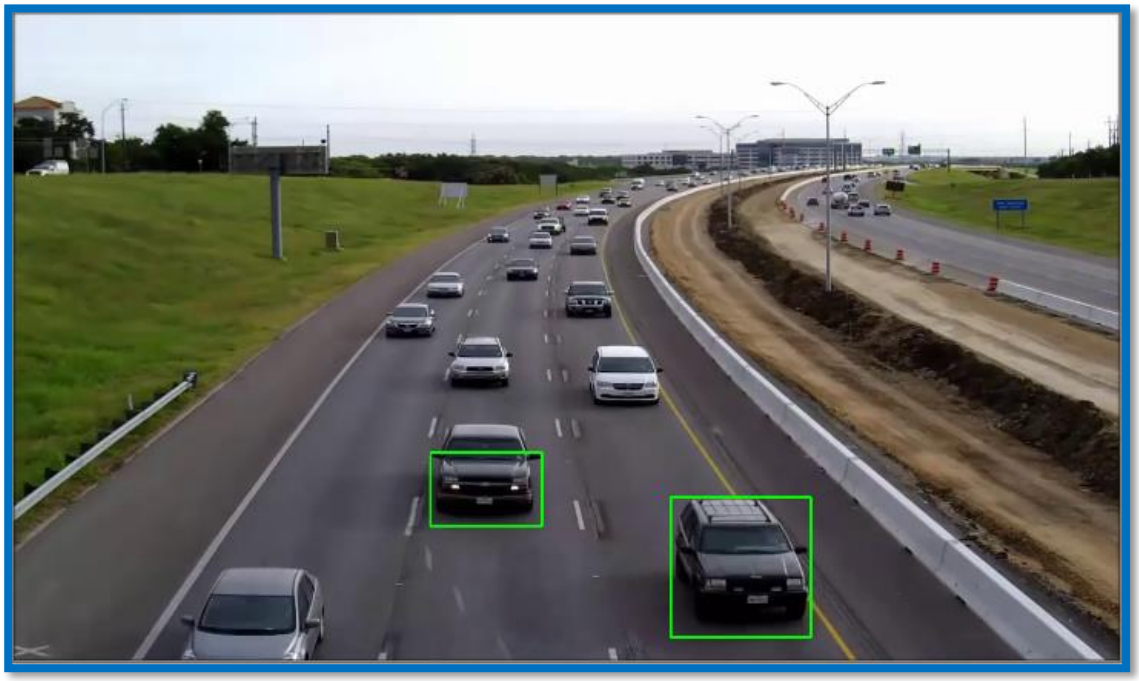


Figura 32

Prueba de Funcionamiento (Video3.mp4)

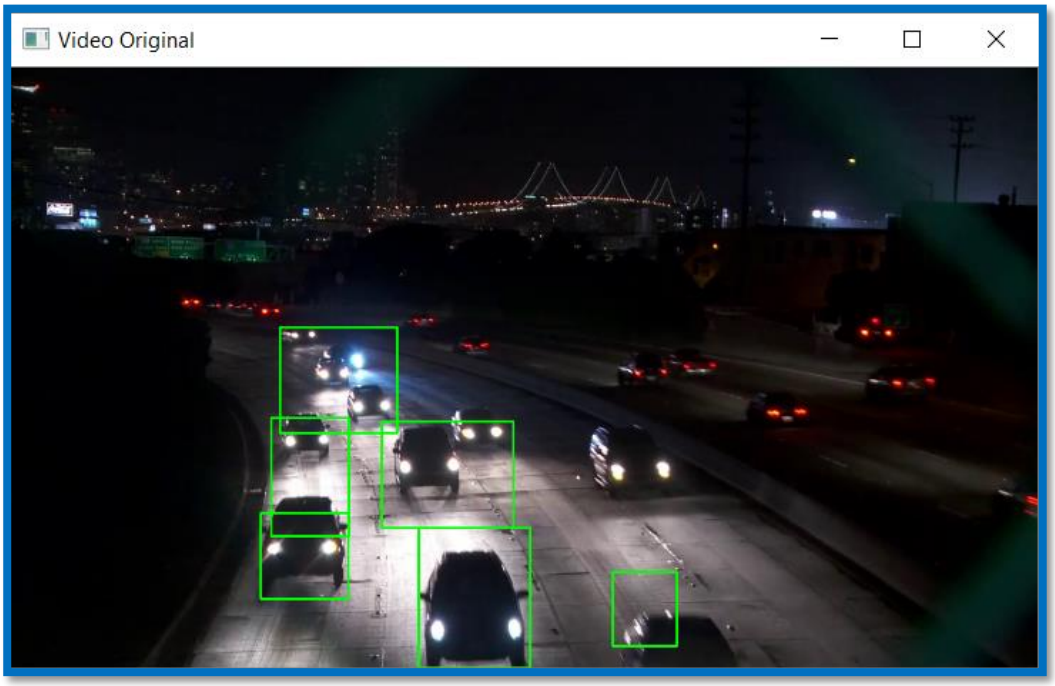
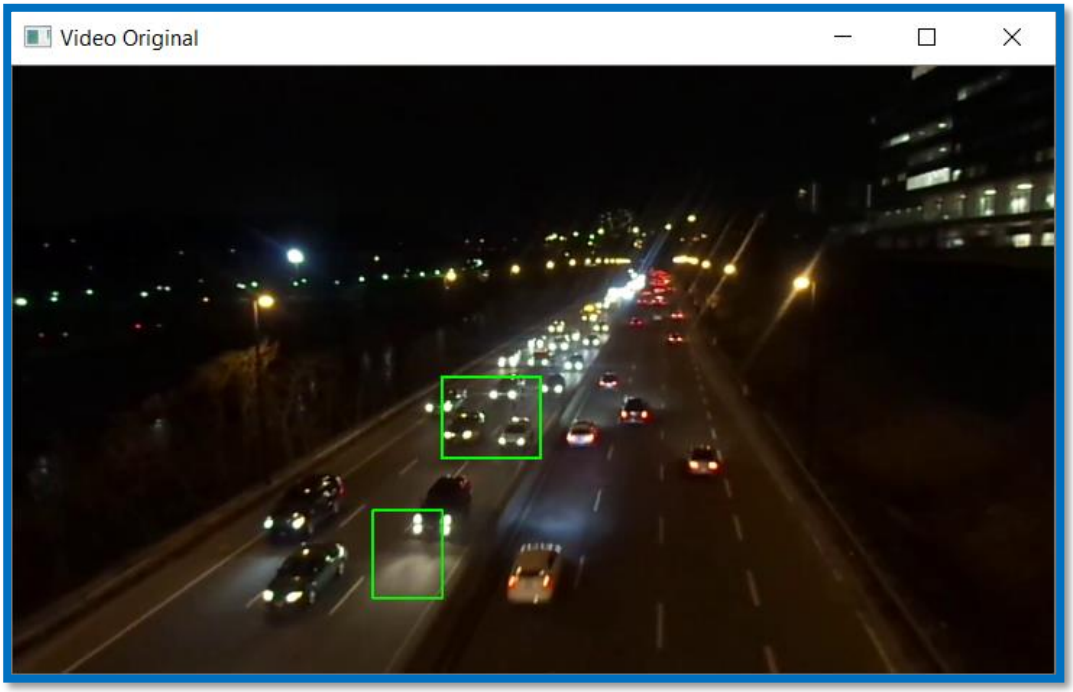


Figura 33

Prueba de Funcionamiento (Video4.mp4)



Resultados

A partir de las pruebas de funcionamiento de la primera iteración, cuyo contenido se puede observar en el **Anexo 1,** se pudo obtener los siguientes resultados:

- El tiempo de duración de los videos utilizados para las pruebas de funcionamiento de este proyecto, son de entre 30 y 60 segundos.
- Se determinó que el algoritmo adecuado para este proyecto, utilizado para la sustracción de fondo es MOG.
- Los videos utilizados para realizar las pruebas de funcionamiento presentan resultados óptimos si son grabados durante el día.

- El prototipo desarrollado en esta primera iteración, logra detectar vehículos. Pero, aunque el porcentaje de detección es muy grande con un 70% a 88% de efectividad, es necesario aumentar este valor.

Limitaciones y restricciones

Después de un primer análisis a una serie de videos, logrando identificar vehículos, se pudieron identificar varias limitaciones y restricciones, que se intentan solucionar en las siguientes iteraciones:

- Los videos utilizados deben ser grabados de forma frontal o trasera a los vehículos, y no por los lados.
- La resolución de los videos debe ser de al menos 720p.
- Los videos utilizados deben ser grabados durante el día.
- La versión que se debe utilizar de la librería de OpenCV dentro de Python debe ser menor a la versión 4.0.0 (en este caso se usó la versión 3.4.2.16). Como ejemplo se realizó pruebas con la versión 4.1.1 dando como resultado un “cv2.error”.

Solución planteada

Para reducir la presencia de errores en base a detección de falsos positivos, falsos negativos y detección múltiple, se propone trazar una línea en la pantalla de reproducción, y graficar un punto en los vehículos detectados; de manera que cuando el punto cruce por la línea graficada, se contabilice los vehículos.

Segunda iteración

En la segunda Iteración, se aplica las soluciones propuestas en la Primera Iteración, la cual se basa en que una vez identificado el vehículo, se ocupe una variable interna que se encargue de contabilizar los

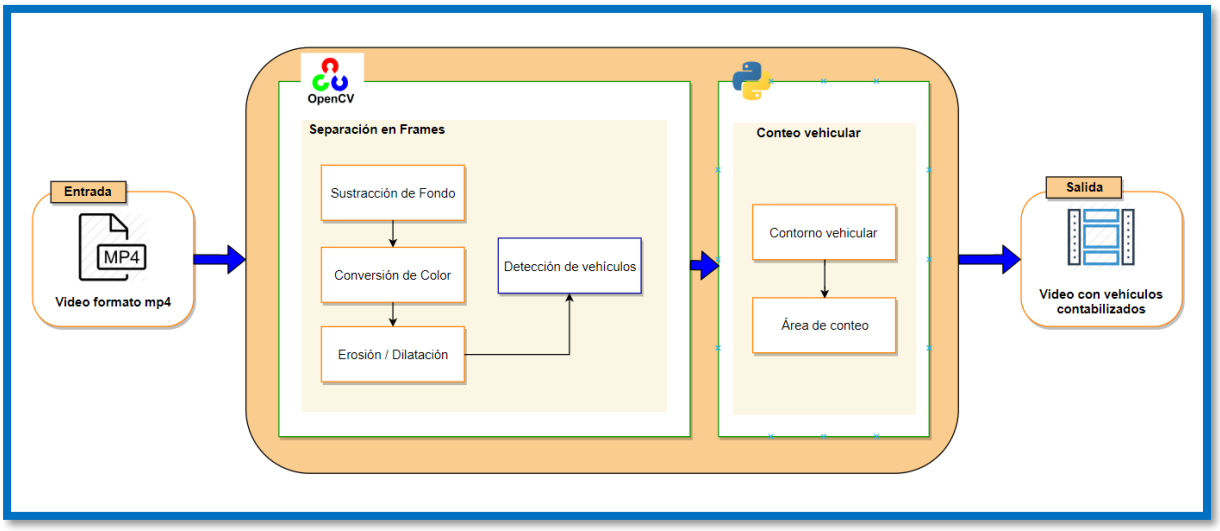
vehículos que han circulado atravesando una línea graficada en pantalla que representa el área de colisión.

Arquitectura

Para esta nueva Iteración, se ha modificado la arquitectura propuesta en la primera Iteración, debido a que se agrega el módulo de contabilización de vehículos, y en la se obtiene un video con los vehículos contabilizados. En la **Figura 34** se puede observar a detalle la arquitectura utilizada en esta iteración.

Figura 34

Arquitectura (Segunda Iteración).



Contabilización

Para la contabilización de los vehículos, se modificó el código del prototipo utilizado en la primera iteración. La contabilización de vehículos actúa sobre cada frame separado del video, de manera que una vez que el programa detecta un vehículo, se procede a graficar un cuadro alrededor del objeto, además de un punto en el centro del objeto. Al mismo tiempo, mediante una línea graficada en la pantalla, se

verifica que en el momento que el punto graficado en el centro del objeto cruce la línea graficada en la pantalla, se contabilice sumando en 1 a una variable encargada de almacenar la cuenta. Con este método, además de detectar los vehículos, se reduce considerablemente el error de falsos negativos, debido a que los objetos detectados que no son vehículos, comúnmente no van a cruzar por la línea de colisión graficada en pantalla.

Para graficar la línea de colisión en pantalla, se utiliza el comando que se muestra en la **Figura 35**, en donde se indica que se graficará un objeto de tipo línea y se especifica el lugar y tamaño de la línea. La variable pos_linea, se declara al inicio del programa con valor de 550.

Figura 35

Creación visual de la línea en el área de colisión

```
cv2.line(frame1, (15, pos_linea), (1500, pos_linea), (255,127,0), 3)
```

En la **Figura 36**, se puede ver el fragmento de código que se encarga de la contabilización de los vehículos. La contabilización se asigna en la variable carros.

Figura 36

Codificación de la contabilización

```
for (x,y) in detec:
        if y<(pos_linea+offset) and y>(pos_linea-offset):
            carros+=1
            cv2.line(frame1, (25, pos_linea), (1200, pos_linea), (0,127,255), 3)
            detec.remove((x,y))
```

Pruebas de funcionamiento

A diferencia de la iteración anterior, para esta iteración se decidió utilizar los videos **Video**, **Video1, y Video2** para realizar las pruebas de funcionamiento. En la **Figura 37**, **Figura 38** y **Figura 39** se

observa la ejecución de las pruebas de funcionamiento en el **Video1**, donde se grafica la línea en la pantalla y el punto en los objetos detectados.

Figura 37

Prueba de funcionamiento (A) Iteración 2.

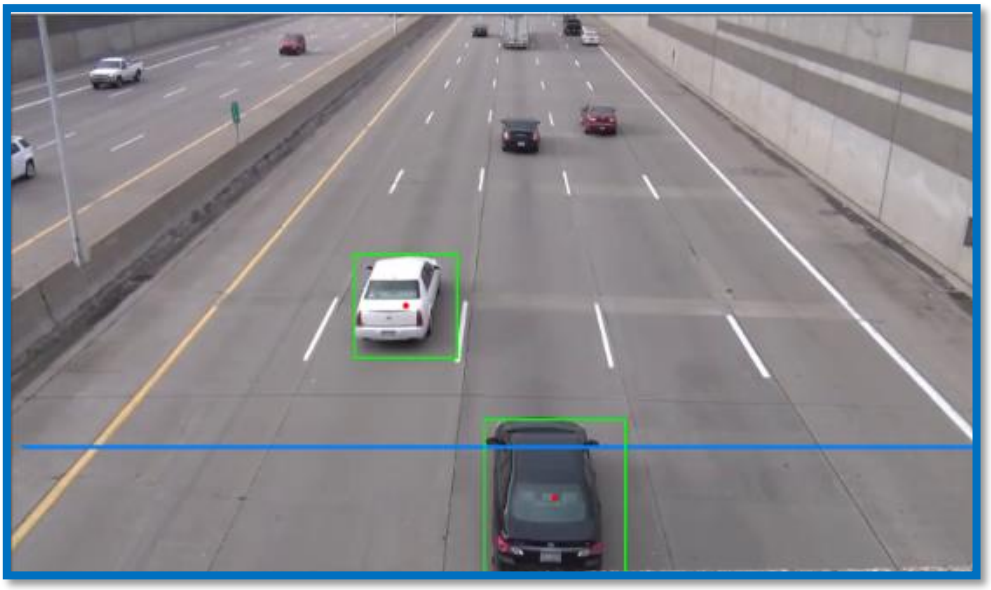


Figura 38

Prueba de funcionamiento (B) Iteración 2.

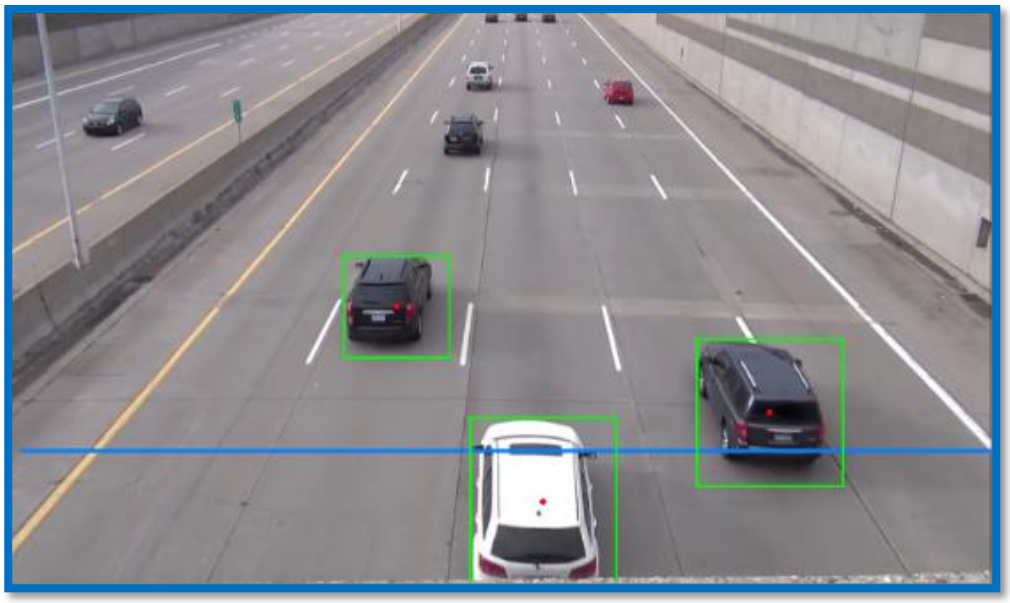
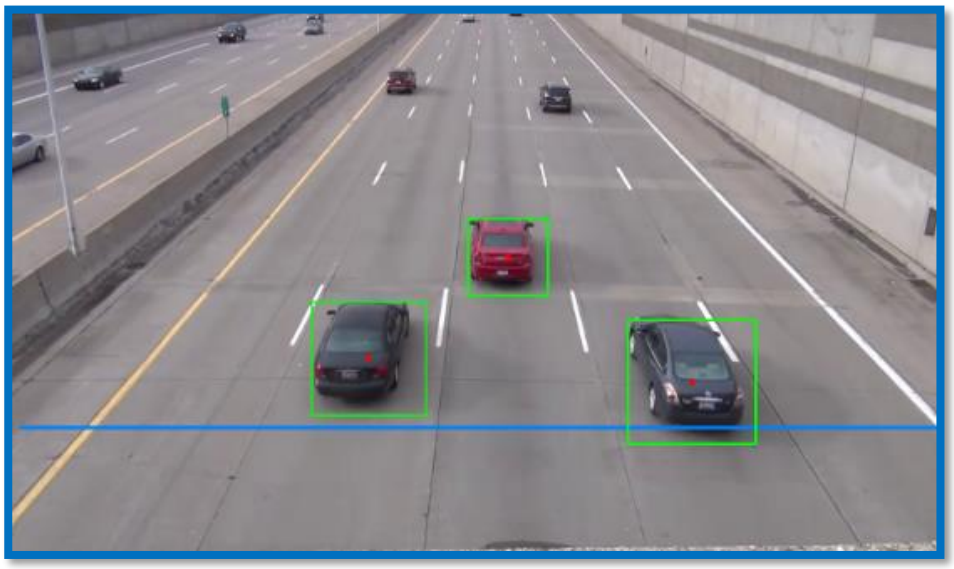


Figura 39

Prueba de funcionamiento (C) Iteración 2.



Pruebas de rendimiento

Un punto adicional en esta iteración, es analizar el funcionamiento del sistema al trabajar con videos de larga duración, verificando si afecta el resultado de las detecciones y contabilizaciones realizadas.

Para ejecutar una evaluación sobre el rendimiento del computador sobre el cual se ejecuta la aplicación, se realizaron diferentes pruebas, las cuales se detallan en la **Tabla 1**.

Tabla 1

Pruebas de rendimiento.

No.	Duración del video (mm:ss)	Verificación
1	01:00	Estable
2	01:00	Estable
3	00:40	Estable
4	00:40	Estable
5	00:32	Estable
6	00:32	Estable
7	70:00	Estable
8	70:00	Estable

Nota: Esta tabla describe las pruebas de rendimiento realizadas en la segunda iteración.

Con el análisis de las pruebas de rendimiento, concluimos que el prototipo soporta videos de larga duración, sin afectar su rendimiento; pero, durante las pruebas de funcionamiento se continúan utilizando videos con una duración de 30 a 60 segundos.

Resultados

Los resultados de esta iteración, se basan en el contenido mostrado en el **Anexo 2**, donde podemos determinar lo siguiente:

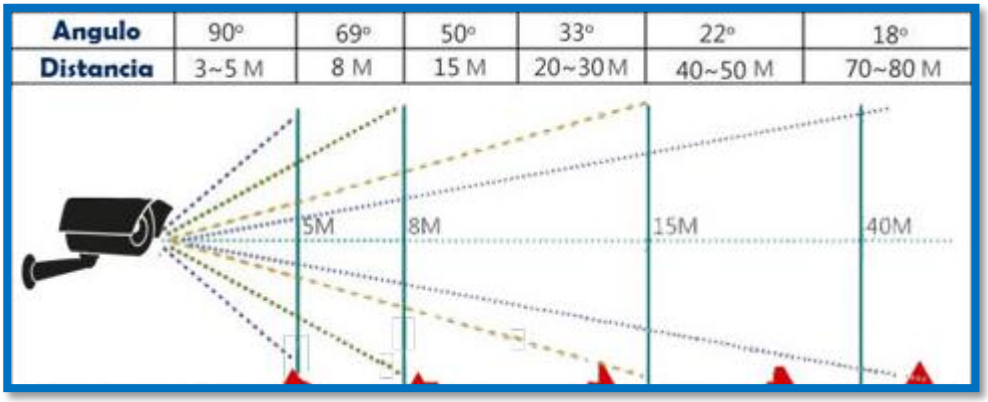
- En la primera iteración, se obtuvo un alto porcentaje de acierto en cuanto a detección de los vehículos, pero esto se debía a que mientras varios autos no eran detectados, se detectaban objetos que no eran autos o se detectaban varias veces un auto. Esto generaba un falso valor en

cuanto a efectividad. Estos inconvenientes se solventaron con la solución propuesta para esta iteración.

- En esta iteración se obtuvo porcentajes de efectividad de detección de 94% para el video **Video**, y de entre 35% y 40 % para los videos **Video1** y **Video2**.
- Los videos utilizados deben ser grabados de frente o de espalda a los vehículos, y se enfatiza que es deben ser grabados durante el día para mejorar la efectividad de detección de vehículos.
- Se logra reducir considerablemente el error de falsos positivos, que ocurre cuando se detectan objetos que no son vehículos o cuando se detectan varias veces un mismo objeto.
- Se detectó la presencia de errores que denominamos falsos negativos, que suceden cuando el punto centro dibujado en el objeto detectado no atraviesa la línea trazada en pantalla y no registra la contabilización.
- Los videos deben ser grabados en base a la **Figura 40**, donde el área de interés es entre 3 y 15 metros. Estos videos deben estar de frente a la circulación de los vehículos, y la calidad debe ser de mínimo 720p.

Figura 40

Posición de Cámara para grabar videos.



Limitaciones y restricciones

Después de analizar los resultados y en base al análisis realizado, es necesario señalar las siguientes limitaciones y restricciones:

- Se recalca que los videos deben ser grabados con la cámara de frente o de espalda a los vehículos, sin importar el sentido de circulación de los vehículos.

Solución planteada

En el **Anexo 2**, se compara el conteo de los vehículos utilizando la primera propuesta de solución, que consiste en el trazo de una línea en pantalla y el gráfico de un punto en los vehículos detectados. En esta primera solución propuesta se puede identificar que, para el primer video, existe un error relativo de 7,14%, para el segundo video un 64,14%, y para el tercero un 66,67%.

Se pretende reducir la presencia de un alto porcentaje de error relativo en el **Anexo 2** con el planteamiento de una nueva solución, la cual consiste en ampliar el tamaño de la línea graficada en pantalla, esto permite que se minimice la contabilización de falsos positivos.

Tercera iteración

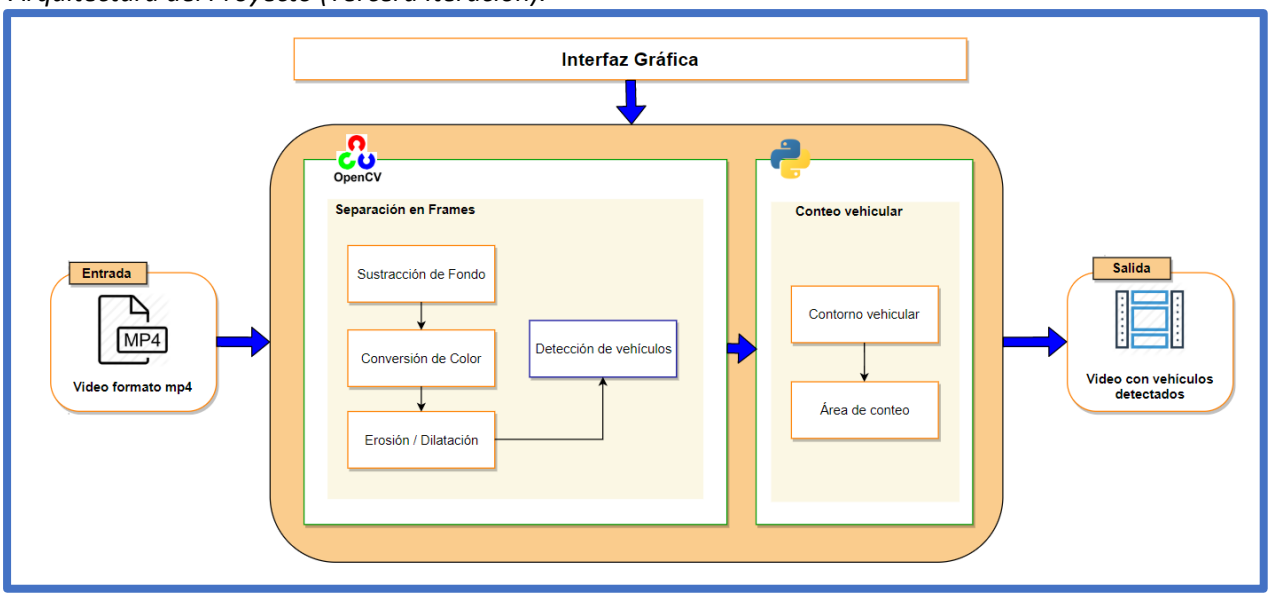
Para esta iteración, se realiza la primera versión de la interfaz gráfica con la que se propone trabajar para la presentación del conteo de vehículos hacia el usuario mediante el uso de la librería PyQt5 y el IDE QT Designer.

Arquitectura

Para la tercera iteración, la arquitectura del sistema es la detalla en la **Figura 41.**

Figura 41

Arquitectura del Proyecto (Tercera Iteración).



Como se puede observar en la arquitectura de esta iteración, se incluyó el módulo de conteo vehicular con su segunda aproximación a la solución planteada. Además, en esta iteración se ha disminuido el área de contabilización de objetos y se aumentó el tamaño de la línea graficada en pantalla, para mejorar la efectividad del conteo.

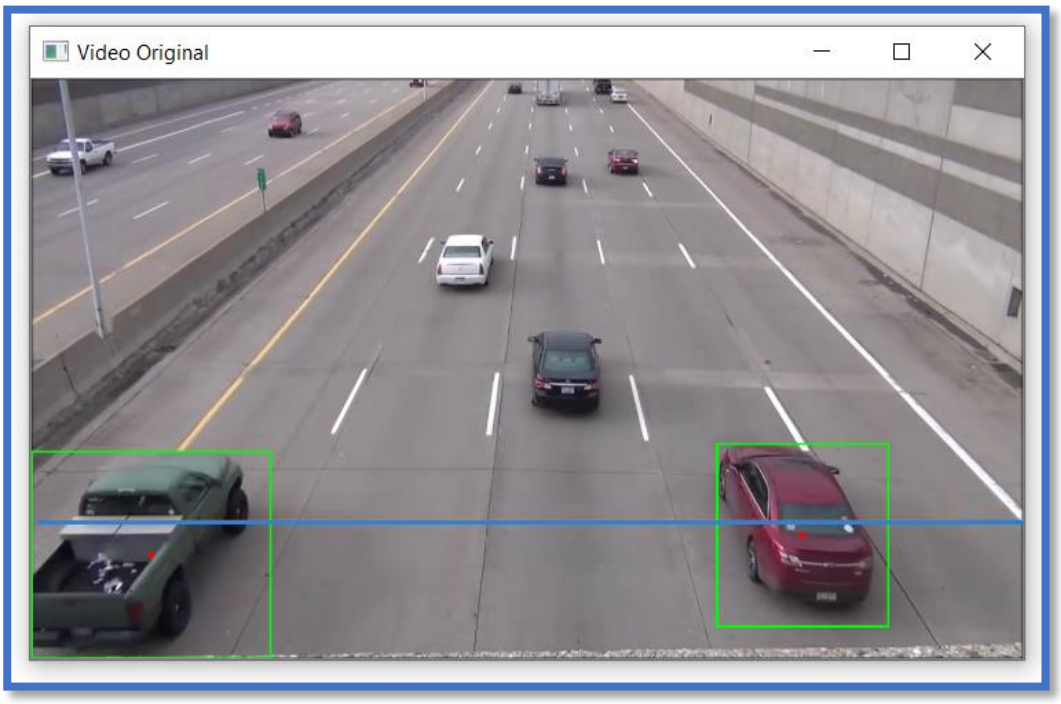
En esta iteración, lo más importante es optimizar la solución al conteo vehicular realizada en la segunda iteración, modificando el algoritmo planteado.

Mejora del algoritmo para el conteo vehicular

Para mejorar la efectividad en la contabilización de vehículos de la iteración anterior, se revisó el comportamiento del algoritmo codificado, en el momento que un vehículo atraviesa la línea graficada en pantalla. Para este análisis se modificó la condición que determina si existe un frame siguiente en la lectura del video, esto mediante el método *waitKey()* perteneciente a la librería cv2; con este método, se puede determinar la velocidad de lectura de frames por segundo.

Figura 42

Frame con vehículo no contado



Con el análisis realizado, se determinó que la línea graficada en pantalla es muy delgada y en ocasiones el punto graficado en el centro del objeto detectado pasa por la línea sin toparla, por lo que no se realiza el conteo. En la **Figura 42** se puede observar el momento previo a la colisión entre el punto y la línea, dando como resultado un falso negativo. Para solucionar este inconveniente, se modificaron los valores especificados para el área de colisión, denominada *offset*.

Una vez que se modificaron los valores del área de colisión, se observó que el sistema cuenta los vehículos anteriormente clasificados como falsos negativos; sin embargo, a los vehículos correctamente contados, ahora los cuenta duplicados. Para evitar esto, se optó por poner una bandera a cada uno de los objetos detectados de interés, evitando que se contabilicen varias veces.

Pruebas de funcionamiento

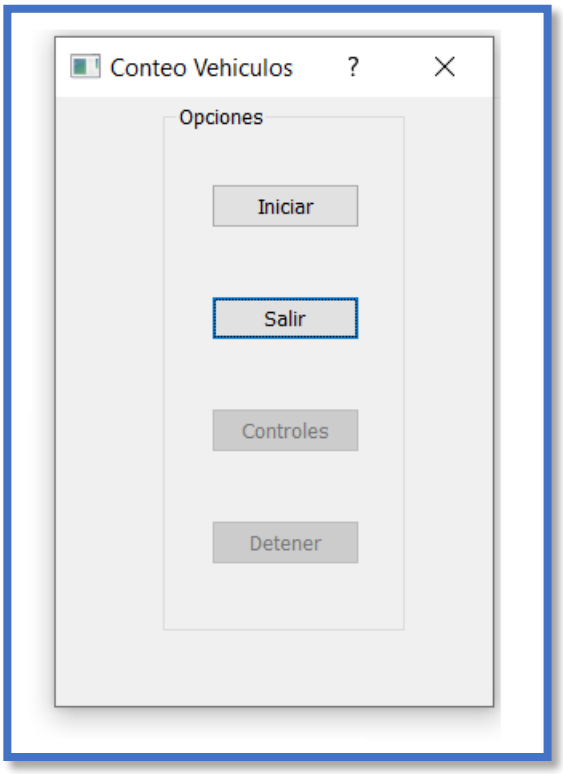
Como se mencionó en las limitaciones detalladas en la segunda iteración, para esta iteración se realizan las pruebas de funcionamiento con el video **Video1**, el cual posee dichas características. Siendo así, los resultados se los detalla en el **Anexo 3**, en donde se puede ver que en el video de prueba con una duración de 00:46:01, se detectaron 53 vehículos, teniendo un 99% de eficiencia, dando así un error absoluto de 1% y un error relativo de 1,96%. De esta manera, dando una solución casi exacta a los vehículos que transitaron por dicha zona.

Interfaz de usuario

Para la realización de la interfaz de usuario, se planteó trabajar con la librería PyQt5 y el IDE de desarrollo QT Designer, el cual daba la facilidad de desarrollo de la interfaz gracias a la característica de "Drag and Drop" (Arrastrar y soltar), con la cual se puede organizar de manera sencilla los componentes que conformaran la interfaz de usuario del sistema mostrada en la **Figura 43.**

Figura 43

Interfaz de usuario (Tercera Iteración)



Al guardar la interfaz, se guarda como un archivo *.ui.* Sin embargo, para poder editarlo, agregar funciones, se lo debe transformar al archivo a un formato que el lenguaje de programación usado pueda usar, en este caso, transformarlo en un archivo *.py*. Para ello se usó la siguiente línea de código:

Figura 44

Generar archivos .py a partir de .ui

```
python -m PyQt5.uic.pyuic -x [nombre del archivo].ui -o [nombre del archivo].py
```

Cuando se ha realizado la conversión del archivo, se debe hacer el llamado del conteo vehicular; esto se puede hace de dos maneras, siendo la primera poniendo todo el algoritmo de la detección y conteo vehicular dentro del mismo archivo de la interfaz, y la segunda (y por la que se optó), mantener dos clases

separadas, e instanciar un objeto que referencie a la clase conteo_vehicular.py dentro de la interfaz. Para ello, en la clase *conteo_vehicular.py* todo el proceso debe estar definido dentro de un método, y esto a su vez dentro de una clase, la cual será llamada desde la interfaz. Adicionalmente, también se puede mandar las declaraciones de las variables a un método general, denominado *init(),* siempre y cuando antes de la declaración misma de la variable se escriba *self* y como parámetro en la declaración del método que hace el detección y conteo, de esta manera, al momento de instanciar el objeto de la clase *conteo_vehicular.py*, se ejecutará el método *init().*

Dentro de la clase de la interfaz únicamente se importó la clase *conteo_vehicular*, y dentro del botón "Iniciar" se llama similar a como se ejemplifica en la **Figura 45.**

Figura 45

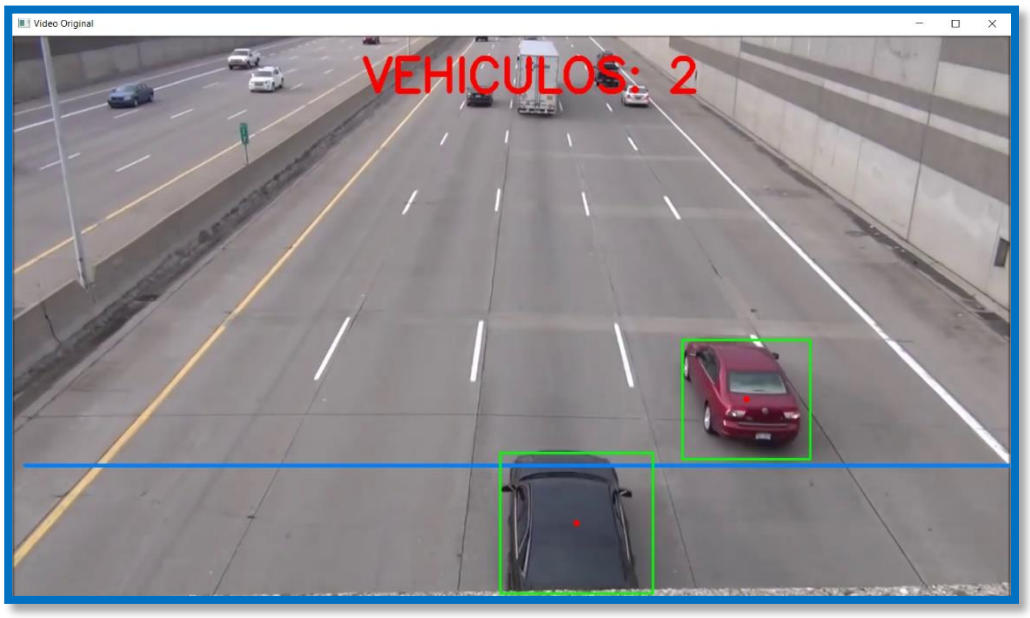
Declaración de los métodos dentro de la interfaz.

```
metodoIniciar = conteo_metodos.main()
metodoIniciar.iniciar()
```

Para la ventana emergente donde se encuentra la detección y conteo vehicular, adicionalmente de la línea que representa el área de colisión en donde se contaran los vehículos y los cuadros que representan los bordes del objeto detectado, se planteó escribir en el video las palabras "Vehículos contados", seguido del número final de vehículos que transitaron por el sector. De esta manera la ventana emergente que contiene el conteo vehicular se muestra en la **Figura 46.**

Figura 46

Ventana emergente final (Tercera Iteración)



Resultados

En esta iteración se contempla el análisis del algoritmo codificado en la iteración anterior y la mejora del algoritmo. Los resultados de esta iteración, se basan en el contenido mostrado en el **Anexo 3**, donde podemos determinar lo siguiente:

- El tamaño que tenía la línea graficada en pantalla durante la segunda iteración, permitía la generación de falsos negativos.
- Con la ampliación de la línea graficada en pantalla se redujeron los falsos negativos.
- La contabilización múltiple de un mismo objeto detectado se controló con el uso de una bandera sobre el objeto detectado de interés.
- Se consiguió una efectividad del algoritmo de entre 98% y 99% utilizando únicamente el video **Video1** para las pruebas de funcionamiento.

Solución planteada

Para la próxima iteración, se propone el uso de varias líneas de colisión dibujadas en pantalla para controlar y hacer más eficiente la contabilización de vehículos.

Cuarta iteración

En la cuarta iteración, se planteó abordar la solución detallada en la segunda iteración, la cual propone la utilización de varias líneas para corregir el problema de los falsos negativos. Conjuntamente a esto se cambió la librería encargada de la interfaz, para permitir que se pueda realizar la búsqueda del video a analizar de manera más dinámica. Funcionalidad que la anterior librería limitaba.

Arquitectura

Para la cuarta iteración, la arquitectura del sistema se es la misma que se mostró en la tercera iteración. Sin embargo, los métodos dentro del módulo de conteo vehicular fueron modificados para dar solución desde otra perspectiva a la problemática planteada.

Mejora del algoritmo para el conteo vehicular

Como se mencionó anteriormente, esta solución nació a partir del análisis de los resultados mostrados en el **Anexo 3**, por lo que se planteó añadir una segunda línea graficada en pantalla, la cual serviría para detectar a los vehículos que no fueron detectados al cruzar por la primera línea. Sin embargo, luego de realizar la tercera iteración, se determinó que también se puede modificar el área de colisión y detección de vehículos. Por lo que para esta iteración se planteó determinar un área de detección vehicular (umbral), para que únicamente dentro del área determinada, se presente el cuadro alrededor del objeto detectado y al salir del área se elimine dicho cuadro. Las líneas de conteo se colocaron en medio de este umbral; de esta manera, si un objeto detectado atraviesa la primera línea y es contabilizado, al atravesar por la segunda línea ya no se será contabilizado.

Para lograr esta solución, se separó nuevamente la clase de *conteo_vehicular.py*, la cual contiene todos los algoritmos de detección y conteo, en una nueva clase llamada *vehicles.py.* Dentro de esta nueva clase se determinan las medidas mínimas y máximas que puede tener un objeto detectado para poder ser considerado un vehículo (variables que anteriormente estaban detalladas dentro del método *init()*). Además, se declaran variables para la descomposición RGB por separado, siendo R, G, B variables individuales, mediante la implementación del método *randint()*, el cual da como resultado un entero aleatorio dentro del rango dado mediante parámetros del método. También, se añaden dos variables denominadas *age* y *max_age*, que determinan el tiempo de paso de un vehículo detectado. Adicionalmente, dentro de la clase *vehicles.py* se crearon tres métodos auxiliares para la detección y conteo correcto; el primero de ellos es *updateCoords,* el cual actualiza los puntos x, y correspondientes al objeto detectado por cada frame; el segundo es *age_one*, que determina si vehículo detectado se mantiene más tiempo en la misma posición, esto con el objetivo de no volver a detectarlo como un objeto nuevo; y por último, *goin_up* y *going_down* los cuales detectan si el vehículo está yendo en manera ascendente o descendente con respecto al video. Cabe mencionar que, en la segunda iteración, en la sección de “Limitaciones y Restricciones Encontradas”, se mencionó que para precisión se tomaría en cuenta únicamente un sentido. No obstante, con las modificaciones realizadas dentro de la tercera y de esta iteración, relacionada al algoritmo de conteo vehicular, se puede determinar que se puede realizar en ambos sentidos a la vez.

Dentro de la clase *conteo_vehiculos.py* se determina el umbral en la que se dibujaran las líneas que representan el área de colisión y los cuadrados alrededor de los objetos de interés detectados. Además, se declaran dos variables, las cuales proporcionan los cuadros por segundo de acuerdo al video seleccionado. En el caso de que se usen videos en tiempo real se debe cambiar este dato manualmente,

caso contrario se usa *get(cv2.CAP_PROP_FPS).* Como resultante estos valores permiten detectar objetos sin ruido, ni falsos positivos.

Como se mostró en la primera iteración, el método de sustracción de fondo que se uso fue el MOG, pero para esta iteración se optó por escojer el MOG2, el cual es la evolución del MOG y que permitió enviar como parámetro la característica de que se detectaran las sobras de los objetos, para poder dar más precisión en la detección.

Para la creación del kernel, se optó por usar el método *np.ones(),* el cual devuelve un conjunto nuevo en forma de un array, rellenos con unos, conjuntamente con el método *np.uint8()*, el cual devuelve un entero entre 0 a 255, lo cual dará forma al área del kernel.

Se creó una máscara para el frame, el cual se manejó mediante la aplicación del método *MORPH_OPEN*, el cual tiene la funcionalidad de erosionar el frame para luego dilatarlo. Es principalmente útil para quitar el ruido que se puede producir en la imagen al momento de sustraer el fondo. Luego, se aplicó el método inverso, denominado *MORPH_CLOSE,* el cual dilata la imagen mostrada en el frame para luego erosionarla. Es útil para cerrar los pequeños agujeros que se pueden presentar dentro de los objetos en primer plano. Posteriormente, para la detección del mejor contorno posible del vehículo a contar se utilizó el método *cv2.CHAIN_APPROX_NONE*.

En la parte del algoritmo que se encarga específicamente del conteo vehicular se creó una validación, para que, en el caso de que el objeto detectado se encuentre fuera del umbral definido, no lo tome en cuenta y pase al siguiente frame, hasta que el objeto se encuentre dentro. Cuando esto se da, primero se determina la posición del vehículo en relación al frame; para ello, se usó el método *cv2.moments(),* el cual indica las posiciones x, y del objeto detectado en ese instante.

Finalmente se realiza el conteo vehicular como se lo hizo en las iteraciones anteriores, con la diferencia que, esta vez se revisa la dirección en la que el vehículo se mueve, para luego compararlo con las áreas de colisión presentadas visualmente como dos líneas. En caso de que un vehículo se lo detecte por primera vez dentro del umbral, se lo guarda dentro de un array de la clase *vehicles.py* y se definen sus valores iniciales.

Pruebas de funcionamiento

Con esta solución realizada, se realizó una prueba con el video **Video1**, para poder contrarrestar con los resultados mostrados dentro de la tercera iteración. Esta prueba se la hizo en base a los criterios manejados durante todas las pruebas de rendimiento anteriores.

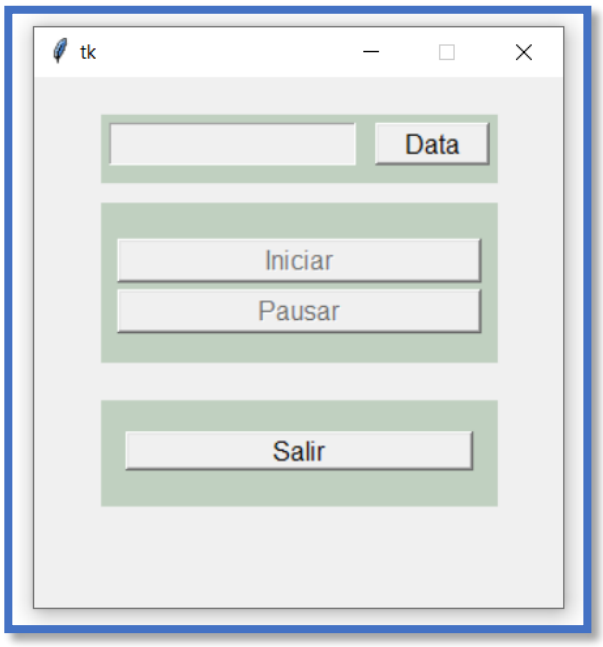
Siendo así, los resultados se los detalla en el **Anexo 4**, en donde se puede ver que en el video de prueba con una duración de 00:46:01, tiene un 100% de efectividad. Se mantiene la restricción mencionada en la segunda iteración, la cual determina que el conteo se centrara en los vehículos que estén de frente a la cámara.

Interfaz de usuario

Para la realización de la interfaz de usuario, en esta iteración se planteó cambiar de librería, debido a que se determinó que el ingreso del video a ser analizado debía ser de manera dinámica mediante una búsqueda dentro de los archivos del sistema. La librería Tkinter permitió realizar esta búsqueda de manera gráfica. Esta interfaz tiene la característica de desplegar una ventana emergente, basándose en el sistema operativo, para la realizar la búsqueda del video. Además, bloquea cualquier funcionalidad que presentan los botones, excepto el de salir. De esta manera, la interfaz desarrollada está mostrada en la **Figura 47**.

Figura 47

Interfaz de usuario (Cuarta Iteración)

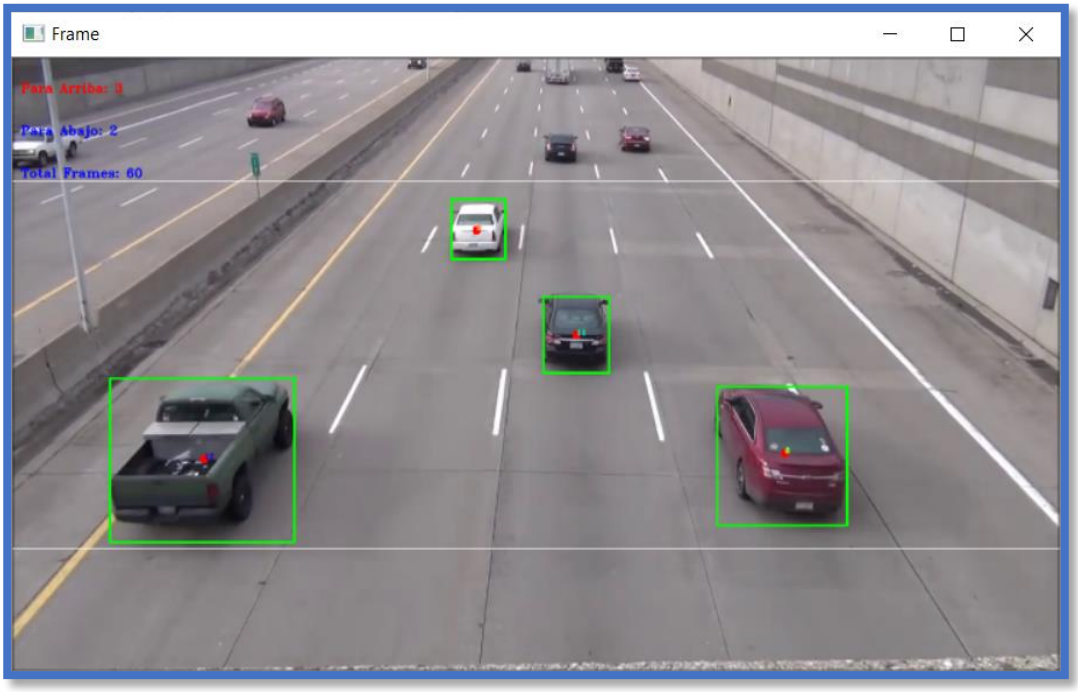


Como se trabaja con interfaz, la división de métodos dentro de la clase *conteo_vehicular.py* se mantiene de la misma manera que la anterior iteración, con el detalle que dentro de esta clase se debe hacer la importación de la clase *vehicles.py*, para que se puedan ver las instancias de los vehículos anteriormente explicados.

La ventana emergente también se vio modificada por los cambios realizados para la mejora de su rendimiento, explicada en esta misma sección en el literal de "Mejora del algoritmo para el conteo vehicular". Lo primero que se hizo, fue modificar el texto que se mostraba en la parte superior del video; ahora se presenta en la esquina superior izquierda dos textos, uno para cada sentido en el que los vehículos se trasladan. Así mismo, para conveniencia de análisis, se imprimió en pantalla el número de frame en el que se encuentra en la ejecución, ubicado debajo de los dos textos anteriores. De esta manera, la ventana emergente que contiene el conteo vehicular se muestra en la **Figura 48.**

Figura 48

Ventana emergente final (Cuarta Iteración)



Pruebas de funcionamiento

Realizada la interfaz con la que se podrá acceder a la detección y conteo vehicular, y viendo que los resultados en la prueba de rendimiento en el video **Video1**, se procedió a evaluar su funcionamiento dentro de varios videos grabados de manera casera. Para ello se tuvo en cuenta las diferentes limitaciones de video que se mencionaron en la primera iteración, en el literal de “Limitaciones y Restricciones Encontradas”, por lo que los videos de prueba están grabados tomando en cuenta el ángulo de inclinación y la hora del día principalmente. Las pruebas obtenidas de los videos seleccionados se muestran a continuación en la **Figura 49, 50, 51** y **52.**

Figura 49

Prueba de Funcionamiento (Video Casero 1)

Figura 50

Prueba de Funcionamiento (Video Casero 2)

Figura 51

Prueba de Funcionamiento (Video Casero 3)

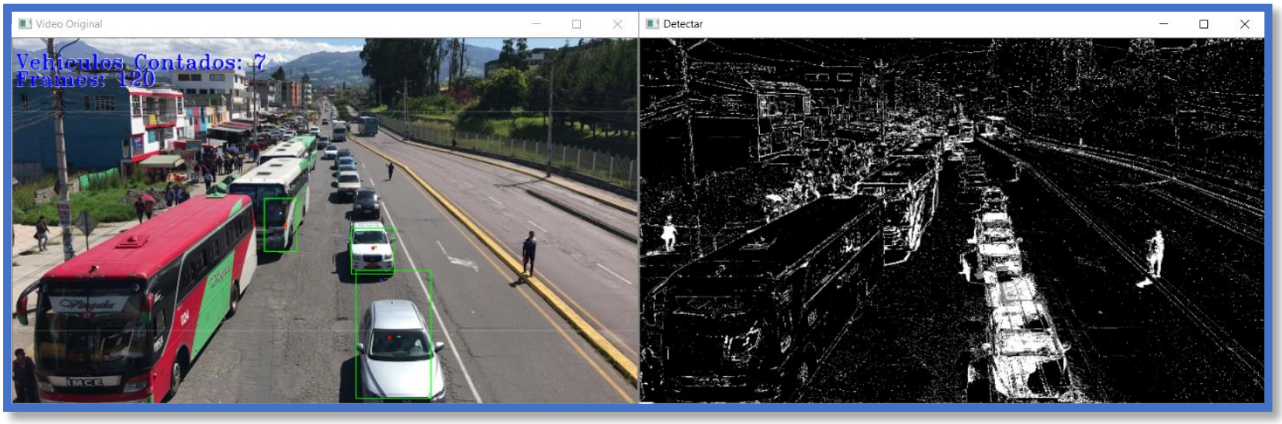
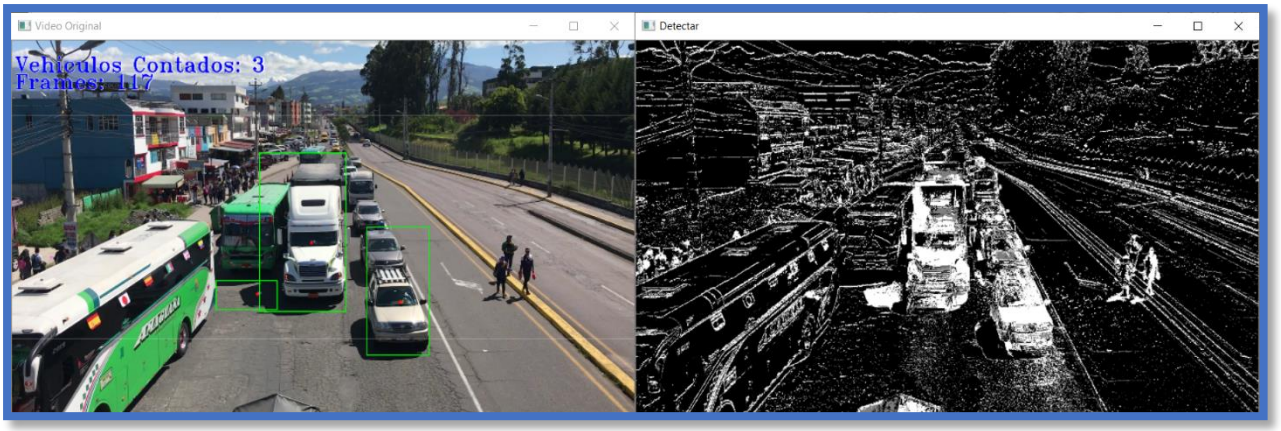


Figura 52

Prueba de Funcionamiento (Video Casero 4)



Resultados

Los resultados de esta iteración, se basan en el contenido mostrado en el **Anexo 4** donde podemos determinar lo siguiente:

- Las grietas en las calles o vías que se ven en los videos, afectan a la eficiencia del sistema.
- Se comprobó la limitante que específica que los videos deben estar grabados de frente con respecto a la cámara se cumple.

- Los videos utilizados deben ser grabados sin movimiento de la cámara; los movimientos de la cámara generan ruido en la grabación del video y afecta a la efectividad de detección y conteo de vehículos.
- Mientras mejor sea la calidad de los videos utilizados (mayor a 720p), el algoritmo presentará mejores resultados con mayor exactitud.
- Los videos deben ser grabados entre las 07h00 y las 18h00 con la presencia de sol, o sin la presencia de lluvia.

Solución planteada

Para la próxima iteración, se propone generar un área de interés, en la cual se detalla el área sobre la cual se aplica la sustracción del fondo y se trabaja sobre ella, de manera que se reduzca el ruido en los videos.

Quinta iteración

Mejoras del algoritmo para el conteo vehicular

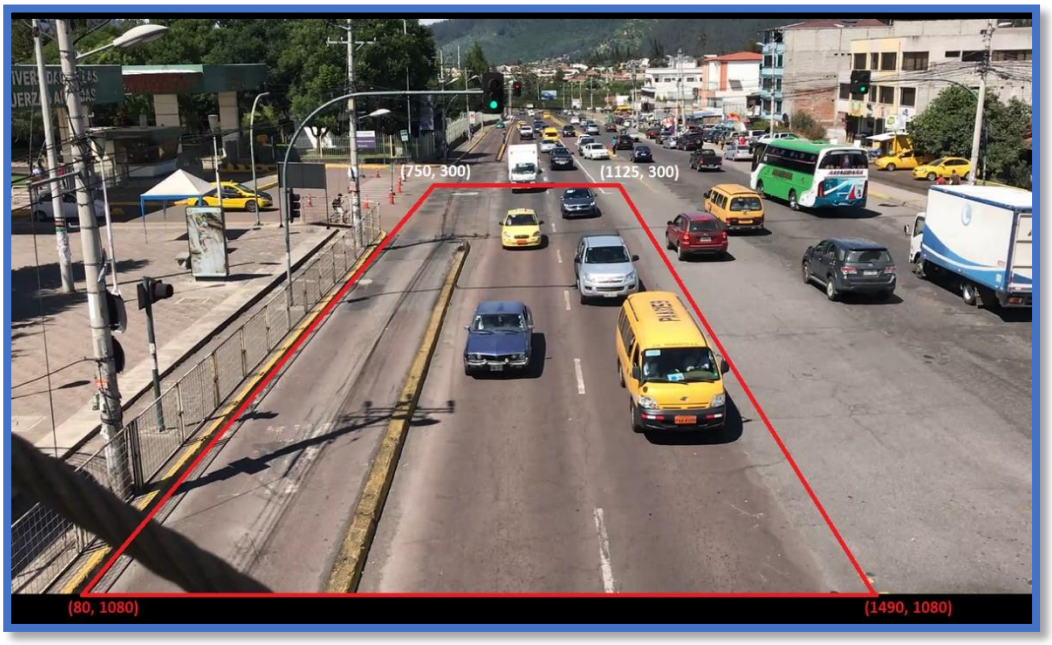
Se empezó creando un área de interés en el que se detectaran los vehículos (por sus siglas en inglés: Region Of Interest: ROI), para determinar únicamente la vía que se encuentra al frente de la cámara. El ROI se obtuvo a partir de la transformada del video, para limitar la detección fuera de esta área sin afectar el video final que se muestra en la pantalla emergente. Para ello, se aplicó el método *numpy.zeros_like(),* el cual devuelve una matriz llena de ceros con la misma forma y tipo de la matriz que se le indique al momento de instanciarla. El parámetro que necesita es la imagen a la cual se debe modificar (esta imagen la detecta como una matriz de datos, de la cual el método sacará el tipo y forma para devolver el dato deseado). Adicionalmente se puede asignar tres parámetros de tipo condicionante

para definir la forma de la matriz resultante; sin embargo, no fueron necesarios para la aplicación que se buscaba de este método.

Además, se utilizó *cv2.fillConvexPoly()*, el cual usa diferentes tipos de datos para sus matrices de puntos, como lo es la matriz resultante de *numpy.zeros_like(),* un array con los puntos en x, y que delimitan el área de interés y el color que mostrarán los objetos detectados. Este método da como resultante un polígono sobre el cual se aplicará la detección y el conteo. En primera instancia se seleccionó al video 1, con el cual se sacaron los puntos a tomar en cuenta para la formación del polígono; esto se muestra en la **Figura 53.**

Figura 53

Puntos delimitantes en (x,y) para el ROI en video 1



Finalmente, se utilizó el método *cv2.bitwise_and()*. Este método obtiene como parámetros al frame original y al frame transformado (mask), para dar como resultado la unión de las matrices que conforman ambas imágenes. Para ello, concatena bit a bit cada posición de la matriz para buscar

coincidencias, en el caso de encontrarlas, las almacena en una tercera matriz y las devuelve como matriz resultante. En ejecución, la aplicación del ROI se ve reflejada en la transformación del video, como se comprueba en la **Figura 54.**

Figura 54

Imagen resultante del método cv2.bitwise_and()

Cabe destacar que esta modificación debe ser manual para cada tipo de video, dado de que las medidas de la sección de relevancia en pixeles difieren de video a video. Solo en el caso específico de videos grabados con las mismas características (ángulo, posición de la cámara) funcionaran de manera similar. En la **Figura 55,** se puede ver un código de muestra para la correcta creación del ROI.

Figura 55

Código general de ROI

```
mask = np.zeros((h, w), np.uint8)
vectores = np.array([p1, p2, p4, p3])
cv2.fillConvexPoly(mask, vectores, 255)
masked = cv2.bitwise_and(img, mask)
```

Pruebas de funcionamiento

Como se mencionó previamente, las configuraciones para la creación del ROI están internamente en el código, por lo que para realizar las pruebas de funcionamiento se usaron videos con las mismas características para determinar si el ROI afecta de manera positiva al conteo final. Cada prueba realizada en los videos caseros de prueba tiene una especificación de vértices que forman el polígono deseado; esto se ve reflejado en la **Figura 56** y **57,** como en la **Figura 58** y **59.**

Figura 56

Vértices usados para el Video Casero 1

```
vertices = np.array(([80,1080],[750,300],[1125,300],[1490,1080]))
```

Figura 57

Prueba de Funcionamiento aplicando ROI (Video Casero 1)

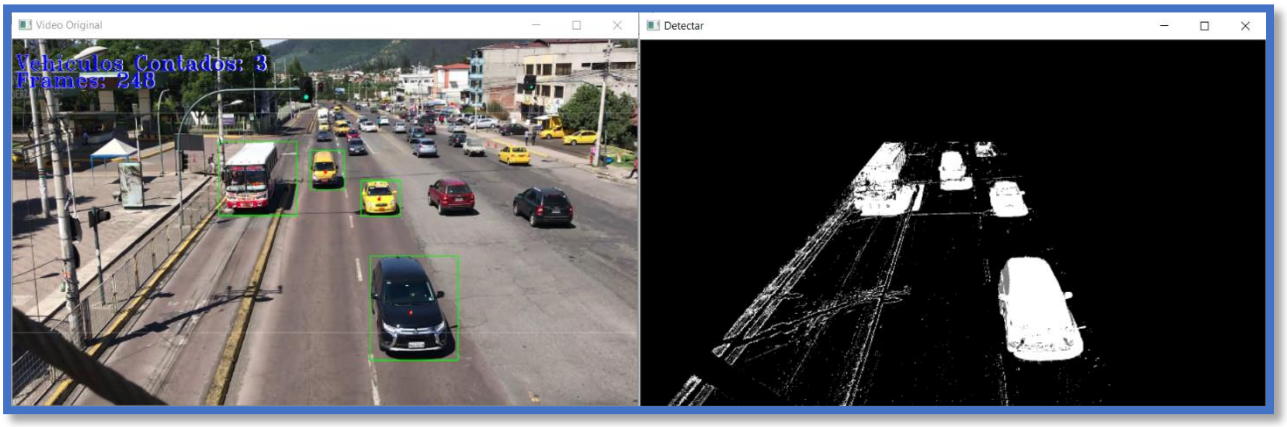


Figura 58

Vértices usados para el Video Casero 2

```
vertices = np.array(([690,1080],[900,300],[1200,300],[1600,1080]))
```

Figura 59

Prueba de Funcionamiento aplicando ROI (Video Casero 2)

Resultados

Los resultados de esta iteración, se basan en el contenido mostrado en el **Anexo 5**, donde podemos determinar lo siguiente:

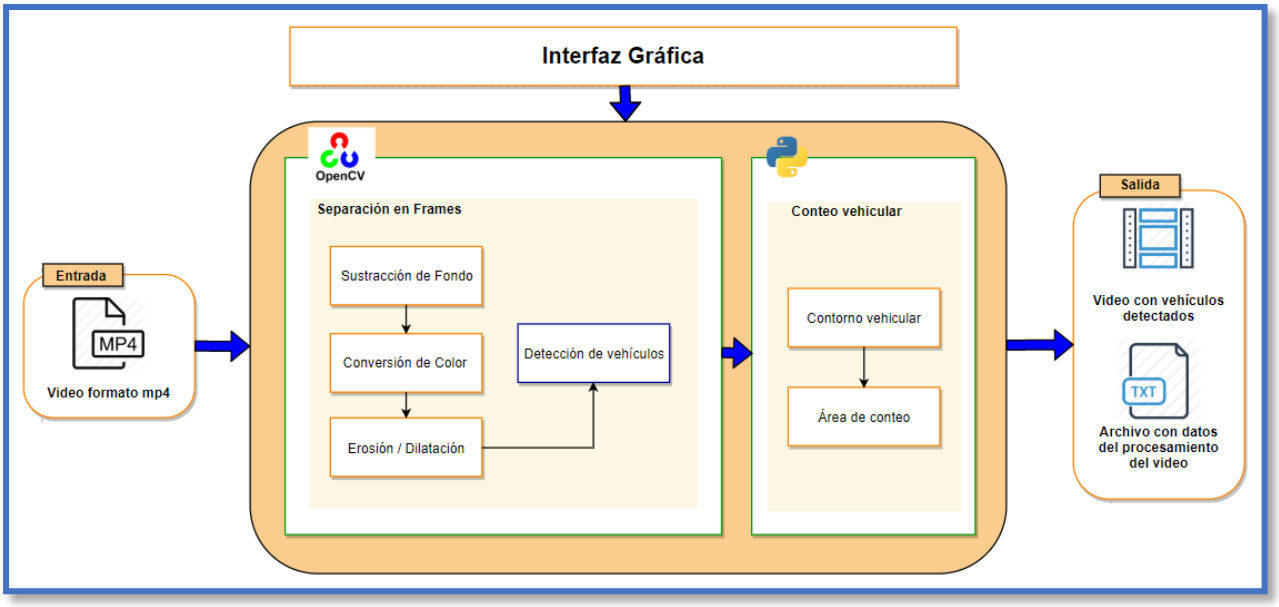
- En el primer video se obtuvo una exactitud de 100% con 2 errores detectados. Sin embargo, estas detecciones se obtuvieron afuera del área de conteo, por lo cual no se vio reflejado este dato en el conteo final.
- Para el segundo video de prueba, el resultado fue satisfactorio, obteniendo un 96% de precisión, con la presencia de 5 falsos positivos. Estos errores fueron producto de la grabación del video con movimiento de cámara y presencia de ruido.
- Con estos resultados se puede constatar el funcionamiento correcto del sistema para la aplicación en videos en las áreas urbanas de quito.

Sexta iteración

Arquitectura

Figura 60

Arquitectura Final del sistema



Para esta iteración, se necesitó respaldar la información referente al procesamiento de los videos para su futuro análisis; es por ello que la muestra gráfica del video modificado con su detección y conteo no es la única salida, sino que se creó conjuntamente un archivo plano, formato .txt, en el cual se detalla el comportamiento del sistema, indicando la fecha exacta en la que se realizó cada conteo.

Adicionalmente, se modificó la interfaz gráfica, reorganizando el contenido de los paneles y presentando cuadros de texto que indican las características del video y el resultado en tiempo real de la ejecución de la aplicación, información extraída directamente de los archivos guardados; además, se almacena esta información en archivos planos para su posterior análisis.

Interfaz Gráfica

En primer lugar, se modificó la posición de los paneles de la interfaz gráfica, presentando la búsqueda del video en la parte superior de la ventana, debajo, a la izquierda el menú de opciones y la descripción del video a analizar. Y finalmente, a la derecha un cuadro de texto que presenta la información en tiempo real del procesamiento del video, y debajo del cuadro de texto, el botón de salir. Adicionalmente, en la ventana principal del sistema, se modificó el ícono y el título mostrado en la cabecera de la ventana. En la **Figura 61**, se puede observar el diseño final del prototipo, y en la **Figura 62**, se presenta el diseño final del prototipo en ejecución.

Figura 61

Presentación final de la interfaz gráfica.

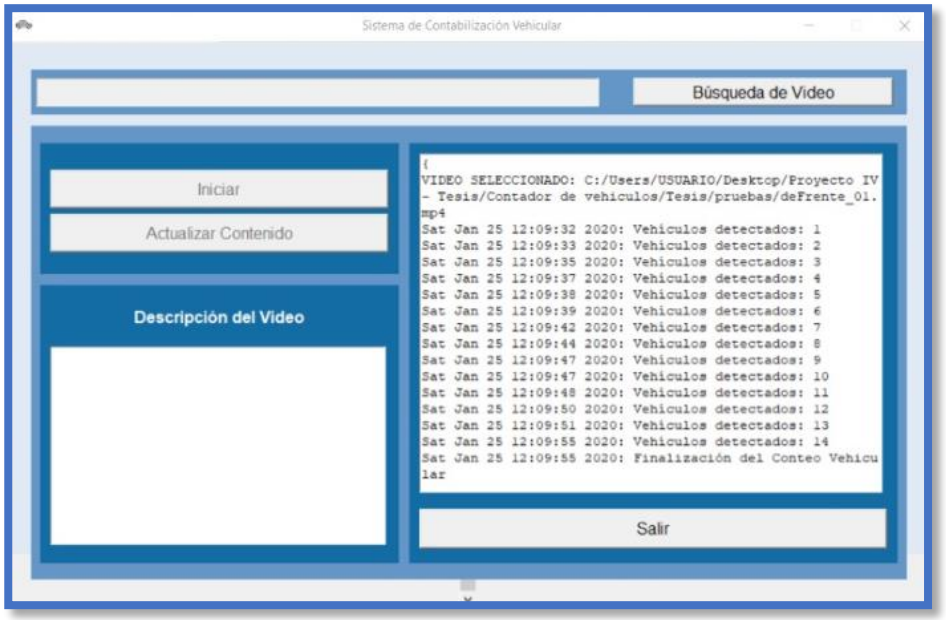
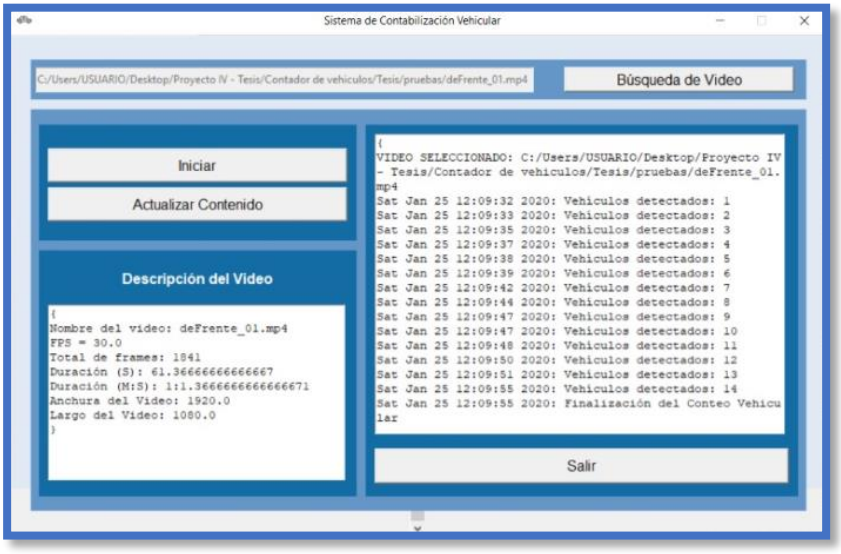


Figura 62

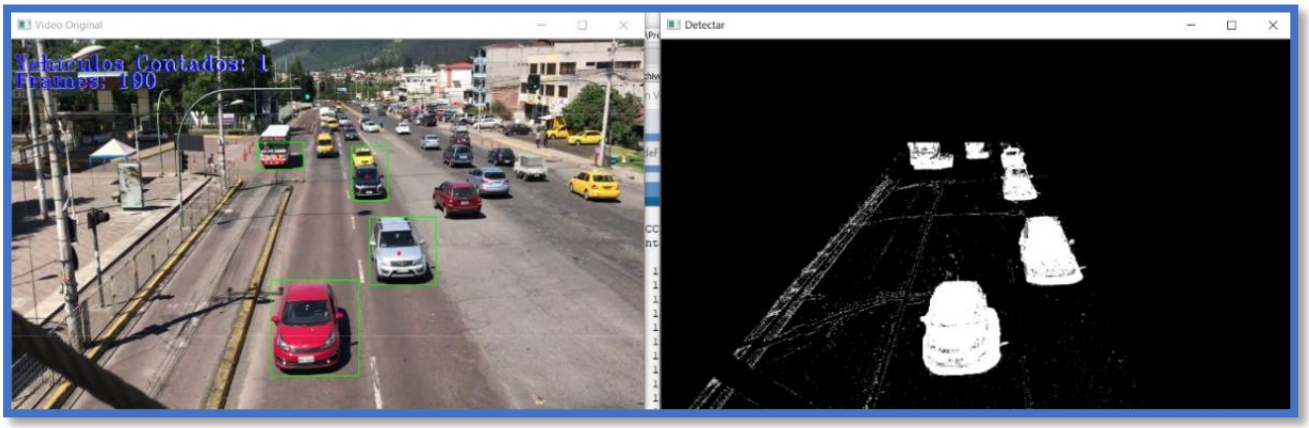
Presentación final de la interfaz gráfica en ejecución.



El funcionamiento es igual al de la iteración anterior; al momento de seleccionar el botón “Iniciar”, se despliegan dos ventanas, la primera con el video mostrando los objetos detectados y la contabilización, y la segunda mostrando el video aplicado la sustracción del fondo y el cambio de espacios de color, como se muestra en la **Figura 63**.

Figura 63

Funcionamiento del sistema.



Almacenamiento de información

En esta iteración, se ha detallado que se agregan dos cuadros de texto en la interfaz gráfica, los cuales presentan la información de las características del video y la información del procesamiento del video en tiempo real. Estos datos, además de ser mostrados en la interfaz gráfica, se almacenan en dos archivos planos; el primero se llama “atributos_video.txt”, el cual registra los siguientes atributos del video:

- Nombre del video,
- Frames por segundo (FPS),
- Total de frames,
- Duración (segundos),
- Duración (minutos:segundos),
- Dimensiones del video.

En el segundo archivo plano, se registran los datos del procesamiento en tiempo real del video, los cuales indican el video seleccionado, el momento en que se registró un vehículo contabilizado, y un registro que indica cuando se cerró el sistema. Este archivo sirve para un posterior análisis de la información registrada por el sistema.

Resultados

Para esta iteración, los siguientes resultados se basan en la usabilidad presentada por la interfaz final y la información almacenada en los archivos planos:

- La nueva interfaz permite tener presente la información que procesa el sistema sobre el video en tiempo real.

- Se puede dividir un video de larga duración en varios videos cortos y analizarlos por separado. Con el registro de la información procesada en los archivos planos, se puede revisar y analizar su contenido.

Capítulo IV

Conclusiones y recomendaciones

Conclusiones

El desarrollo de esta tesis, basado en la metodología de desarrollo ágil conocida como Desarrollo Iterativo, atravesó por cinco iteraciones; tres de ellas planteadas inicialmente, las cuales consistían en solventar la problemática planteada, el desarrollo y prueba del sistema. Sin embargo, al realizar las pruebas correspondientes se verificó que se necesitaba mejorar el prototipo, alterando el código y el procesamiento interno del video, para tener un conteo vehicular más exacto, por lo cual se generaron dos iteraciones adicionales.

Los videos utilizados en este sistema deben ser en una resolución de al menos 720p; utilizando videos de mejor calidad, el sistema presentará resultados más precisos.

Para que el sistema presente un funcionamiento óptimo, los videos utilizados en el sistema deben ser grabados entre las 07h00 y las 18h00, sin la presencia de lluvia; además, deben ser grabados de frente o de espalda a los vehículos con un ángulo de inclinación de aproximadamente 30° a una altura de 2 metros.

El estado de las calles o vías, afectan al rendimiento del sistema, grietas en la vía causan que se necesite aplicar más filtros para limpiar las imperfecciones del video.

El prototipo final de esta tesis, necesita que los videos sean tratados independientemente, para que la contabilización sea más efectiva. Esto se debe a que cada video puede ser grabado de forma diferente; y la selección del área de detección, y la presentación de las líneas de colisión en pantalla se realizan dependiendo de cada video.

El tiempo de duración de los videos que se pueden utilizar en el prototipo puede alcanzar la hora de duración, sin presentar inconvenientes. Pues se realizaron pruebas de rendimiento para verificar el funcionamiento del prototipo.

Recomendaciones

Para asegurar el correcto funcionamiento del sistema, se recomienda que la cámara con la cual se esté grabando los videos se encuentre en una posición fija y no presente movimiento durante la grabación.

En un escenario en el que los vehículos no circulen con fluidez, el conteo se puede ver afectado debido a que varios vehículos juntos, pueden ser detectados como uno solo; para cual se recomienda que, en estos casos, la cámara apunte de forma perpendicular a la circulación de los vehículos, de modo que se visualice la separación entre ellos.

Para analizar videos sumamente largos (mayor a 1 hora), se recomienda fragmentar el video completo en varios videos de menor tamaño y analizarlos por separado.

Para el análisis de múltiples videos, se recomienda revisar los archivos planos generados por el sistema, los cuales registran la información analizada de cada video ingresado al sistema, de manera similar a un log del sistema.

Para la detección de la región de interés (ROI), se recomienda que el manejo de inteligencia artificial podría facilitar su construcción, detectando el área de interés de manera automática. Así, se podría ampliar el campo de aplicación de este sistema, poniéndolo en funcionamiento en carreteras con distintas configuraciones.

Para futuros trabajos, se puede añadir al desarrollo un sistema externo denominado YOLO (You Only Look Once), el cual es una pequeña red neuronal destinada para la detección de los objetos y su posterior categorización.

Como aplicativo, este sistema puede ser utilizado como un módulo de un sistema de parqueaderos inteligentes, pues se adapta a cualquier tipo de área rural o urbana.

Referencias

Shokrolah Shirazi, M., & Morris, B. (2015). Vision-Based Vehicle Counting with High Accuracy for Highways with Perspective View. In *Lecture Notes in Computer Science* (Vol. 9475). doi:https://doi.org/10.1007/978-3-319-27863-6_76

Allamehzadeh, A., S. Aminian, M., Mostaed, M., & Olaverri-Monreal, C. (2018). Automatic Vehicle Counting Approach Through Computer Vision for Traffic Management. In *Lecture Notes in Computer Science* (Vol. 10672). doi:https://doi.org/10.1007/978-3-319-74727-9_48

Almache, M. (2013). *GUIA-IA.*

Anónimo. (2019). *OpenCV: How to Use Background Subtraction Methods.* Retrieved from https://docs.opencv.org/master/d1/dc5/tutorial_background_subtraction.html

Background Subtraction. (2013). Retrieved from Opencv-python-tutroals.readthedocs.io.: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_video/py_bg_subtraction/py_bg_subtraction.html#background-subtraction

Blippar. (n.d.). *The Blippar App*. Retrieved from https://www.blippar.com/the-blippar-app

C. Müller, V. &. (2019). *Philosophy and Theory of Artificial Intelligence*. Retrieved from SpringerLink: https://link.springer.com/book/10.1007/978-3-642-31674-6

Carvajal, A. M. (2019, 02 21). *El Comercio*. Retrieved from El Comercio: https://www.elcomercio.com/actualidad/congestion-vehicular-ranking-movilidad-amt.html

Cohen, J. (2018, 01 24). *Towards Data Science.* Retrieved from https://towardsdatascience.com/ai-and-the-vehicle-went-autonomous-e176c73239c6

Degano, P. (1983). *Artificial Intelligence*. Retrieved from Pictorial Data Analysis, 239-264.: doi:10.1007/978-3-642-82017-5_12

El Comercio. (2018). *¿Cuántas Horas Al Año Pasan Los Quiteños Atascados En El Tráfico?* Retrieved from https://www.elcomercio.com/actualidad/medida-picoyplaca-efectividad-quito-parqueautomotor.html.

Ferreira Junior, J. R. (2013). Urban Traffic Management System by Videomonitoring. *Advances in Computational Science, 225*. doi:https://doi.org/10.1007/978-3-319-00951-3_1

Geometric Transformations of Images. (2013). Retrieved from Opencv-python-tutroals.readthedocs.io.: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_geometric_transformations/py_geometric_transformations.html

INRIX. (2018). *INRIX Global Traffic Scorecard*. Retrieved from http://inrix.com/scorecard-city/?city=Quito&index=26.

JavaTpoint. (n.d.). *JavaTpoint*. Retrieved from JavaTpoint: https://www.javatpoint.com/opencv

K, A., & Mordvintev, A. (2013). *OpenCV-Python Tutorials*. Retrieved from OpenCV-Python Tutorials: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_setup/py_intro/py_intro.html

Kadiķis, R., & Freivalds, K. (2013). Efficient Video Processing Method for Traffic Monitoring Combining Motion Detection and Background Subtraction. In *Lecture Notes in Electrical Engineering* (Vol. 221). doi:https://doi.org/10.1007/978-81-322-0997-3_12

Kaehler, A., & Bradski, G. (2017). *Learning OpenCV 3.* O'Reilly Media, Inc.

Lan, J., Jiang, Y., Fan, G., Zhang, Q., & Yu, D. (2016). Real-Time Automatic Obstacle Detection method for TrafficSurveillance in Urban Traffic. *Journal of Signal Processing Systems*, 357-371.

Loza Herrera, W. E. (2015). *INFORME FINAL CASO DE ESTUDIO PARA UNIDAD DE TITULACIÓN ESPECIAL.* Quito.

Mishra, M. (2019, 01 10). Retrieved from https://www.datascience.com/blog/computer-vision-in-artificial-intelligence

Nelli, F. (2018). Image Analysis and Computer Vision with OpenCV. In F. Nelli, *Python Data Analytics* (pp. 507-535). Apress, Berkeley, CA. doi:https://doi.org/10.1007/978-1-4842-3913-1_14

Nelli, F. (2018). Python Data Analytics. In F. Nelli.

OpenCV. (n.d.). Retrieved from OpenCV: Morphological Transformations: https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html

Pavón, A. (2016). ¿Qué pasa con nuestro terrible tránsito vehicular? *El Telégrafo*. Retrieved from https://www.eltelegrafo.com.ec/noticias/columnistas/1/que-pasa-con-nuestro-terrible-transito-vehicular

Platero, C. (2019). *Procesamiento morfológico*. Retrieved from Apuntes de Visión Artificial: http://www.elai.upm.es/webantigua/spain/Asignaturas/Robotica/ApuntesVA/cap6VAProcMorf.pdf

Quito, M. d. (2015). *Diagnóstico Estratégico - Eje de la Movilidad.* Quito. Retrieved from http://www7.quito.gob.ec/mdmq_ordenanzas/Sesiones%20del%20Concejo/2015/Sesi%C3%B3n%20Extraordinaria%202015-02-13/PMDOT%202015-2025/Volumen%20I/8.%20Diagn%C3%B3stico%20Movilidad.pdf

Tang, N., Do, C., Ba Dinh, T., & Ba Dinh, T. (2012). Urban Traffic Monitoring System. doi:https://doi.org/10.1007/978-3-642-25944-9_74

Torres Espinoza, F., Barros G., G., & Barros, M. (2017). Computer Vision classifier and platform for automatic counting: more than cars.

Valencia, U. d. (2019). *Reconocimiento de imágenes: software y ejemplos* . Retrieved from https://www.universidadviu.com/reconocimiento-de-imagenes-software-y-ejemplos

Anexos