UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS, ELECTRÓNICA E INDUSTRIAL

DIRECCIÓN DE POSGRADO

MAESTRÍA EN AUTOMATIZACIÓN Y SISTEMAS DE CONTROL

Proyecto de Investigación y Desarrollo

TEMA:

"ANÁLISIS DE TRÁFICO VEHICULAR MEDIANTE VISIÓN ARTIFICIAL"

Trabajo de Investigación, previo a la obtención del Grado Académico de Magister en Automatización y Sistemas de Control.

SUBLÍNEAS DE INVESTIGACIÓN: Visión Artificial

AUTOR: Ing. Jovann Pérez Nasser

Ambato – Ecuador

Enero, 2019

A la Unidad Académica de Titulación de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial.

El Tribunal receptor del Trabajo de Investigación presidido por la Ingeniera Elsa Pilar Urrutia Urrutia, Mg., integrado por los señores: Ingeniero Franklin Wilfrido Salazar Logroño, Mg., Ingeniero Patricio Germán Encalada Ruiz, Mg., Ingeniero Marcelo Vladimir García Sánchez, Dr., designados por la Dirección de Posgrado de la Universidad Técnica de Ambato, para receptar el Trabajo de Investigación con el tema: "Análisis de Tráfico Vehicular Mediante Visión Artificial", elaborado y presentado por el Ingeniero Jovann Pérez Nasser, para optar por el Grado Académico de Magister en Automatización y Sistemas de Control; una vez escuchada la defensa oral del Trabajo de Investigación el Tribunal aprueba y remite el trabajo para uso y custodia en las bibliotecas de la UTA.

Ing. Elsa Pilar Urrutia Urrutia, Mg.

Presidente de Tribunal

Ing. Franklin Wilfrido Salazar Logroño

Miembro de Tribunal

Ing. Patricio Germán Encalada Ruiz

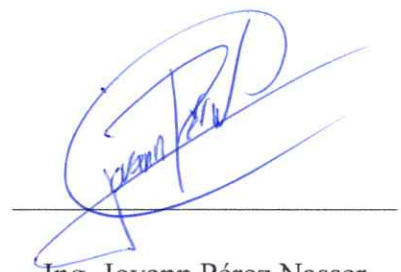
Miembro de Tribunal

Ing. Marcelo Vladimir García Sánchez, Dr.

Miembro de Tribunal

AUTORÍA DEL TRABAJO DE INVESTIGACIÓN

La responsabilidad de las opiniones, comentarios, y críticas emitidas en el trabajo de investigación presentado con el tema: "Análisis de Tráfico Vehicular Mediante Visión Artificial", le corresponde exclusivamente al Ingeniero Jovann Pérez Nasser, autor bajo la dirección del Ing. Darío Javier Mendoza Chipantasi MSc; y el patrimonio intelectual a la Universidad Técnica de Ambato.



Ing. Jovann Pérez Nasser

CC: 180482374-6

AUTOR

Ing. Darío José Mendoza Chipantasi MSc.

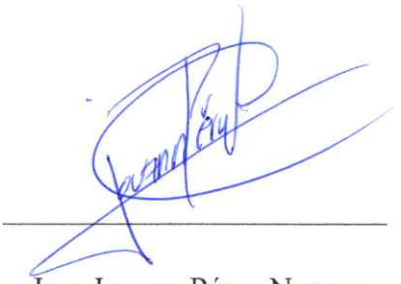
CC: 0603110834

TUTOR

DERECHOS DE AUTOR

Autorizo a la Universidad Técnica de Ambato, para que el trabajo de investigación sirva, como un documento disponible para su lectura, consulta y procesos de investigación, según las normas de la institución.

Cedo los derechos de mi trabajo, con fines de difusión pública, además apruebo la reproducción de éste, dentro de las regulaciones de la Universidad.



Ing. Jovann Pérez Nasser

CC: 180482374-6

AUTOR

DEDICATORIA

A mi madre

A mi hermano

A mis nenes: Balú, Matías y Shiro

AGRADECIMIENTO

A mi madre y mi hermano

A mi adorada Ale

A mi gran amigo Francisco

A mis mentores: Ing. Darío Mendoza, Dr. Ramiro Portero y Psic. Susana Farfán

¡Gracias!

ÍNDICE DE CONTENIDOS

RESUMEN EJECUTIVO XIII

INTRODUCCIÓN XVII

CAPÍTULO I 1

EL PROBLEMA DE INVESTIGACIÓN 1

1.1 Tema de investigación 1

1.2 Planteamiento del problema 1

1.2.1 Contextualización 1

1.2.2 Análisis Crítico 3

1.2.3 Prognosis 4

1.2.4 Formulación del Problema 4

1.2.5 Preguntas Directrices 5

1.2.6 Delimitación 5

1.3 Justificación 5

1.4 Objetivos 6

CAPÍTULO II 7

MARCO TEÓRICO 7

2.1 Antecedentes Investigativos 7

2.2 Categorías Fundamentales 9

2.3 Fundamentación Filosófica 13

2.4 Fundamentación Legal 13

2.5 Visión Artificial 13

2.5.1 Pre Procesamiento de Imágenes 14

2.6 Análisis de tráfico vehicular 15

2.6.1 Seguimiento de Objetos Mediante Visión Artificial 16

2.6.2 Algoritmos de Visión para Detección de Objetos 17
2.7 Algoritmos de Visión para Conteo de Objetos 21
2.7.1 Algoritmos de Aprendizaje Profundo 22
2.8 Teoría de Flujo Vehicular 27
2.8.1 Velocidad 28
2.8.2 Volumen de Tránsito 28
2.8.3 Densidad 28
2.8.4 Tasa de Flujo 29
2.9 Hipótesis 29
2.10 Señalamiento de variables de la hipótesis 29
2.10.1 Variable independiente 29
2.10.2 Variable dependiente 30
CAPÍTULO III 31
METODOLOGÍA 31
3.1 Enfoque 31
3.2 Modalidad básica de la investigación 31
3.3 Nivel o tipo de investigación 31
3.4 Población y muestra 32
3.5 Operacionalización de variables 34
3.6 Recolección de información 36
3.7 Procesamiento y análisis 36
CAPÍTULO IV 37
ANÁLISIS E INTERPRETACIÓN DE RESULTADOS 37
4.1 Algoritmos de Seguimiento de Objetos 37
4.1.1 Algoritmos de Seguimiento 37

4.1.2 Algoritmos de Detección 38
4.2 Tolerancias en Seguimiento de Objetos Mediante Visión Artificial 38
4.3 Calibraciones previas de Sistemas de Seguimiento de Objetos 38
4.3.1 Metodología de Sustracción de Fondo 38
4.3.2 Redes Neuronales 39
4.4 Entorno de Desarrollo para Visión Artificial 39
4.4.1 LabView 40
4.4.2 Matlab 40
4.4.3 OpenCV 40
4.4.4 Simple CV 41
4.4.5 VXL 41
4.5 Licenciamiento 42
CAPÍTULO V 44
CONCLUSIONES Y RECOMENDACIONES 44
5.1 Conclusiones 44
5.2 Recomendaciones 44
CAPÍTULO VI 46
LA PROPUESTA 46
6.1 Tema de la Propuesta 46
6.2 Datos Informativos 46
6.3 Antecedentes a la Propuesta 46
6.4 Justificación 47
6.5 Objetivos 48
6.5.1 Objetivo General 48
6.5.2 Objetivos Específicos 48
6.6 Análisis de Factibilidad 48

6.6.1 Factibilidad Técnica 48
6.6.2 Factibilidad Operativa 48
6.6.3 Factibilidad Económica 49
6.7 Fundamentación Científico – Técnica 49
6.7.1 Clase MultiTracker de OPENCV 49
6.7.2 Algoritmo YOLO (You Only Look Once) 50
6.8 Metodología 53
6.8.1 Configuración del Sistema 54
6.8.2 Seguimiento de Vehículos 55
6.8.3 Conteo de Vehículos 56
6.9 Procesamiento de Datos de Seguimiento 61
6.9.1 Análisis de Resultados de Seguimiento 63
6.10 Procesamiento de Datos de Conteo 66
6.10.1 Análisis de Resultados de Conteo 67
6.11 Administración 75
6.11.1 Recursos Humanos 75
6.11.2 Recursos Materiales 75
6.11.3 Recursos Económicos 75
6.12 Conclusiones y Recomendaciones 76
6.12.1 Conclusiones 76
6.12.2 Recomendaciones 76
REFERENCIAS 78

ÍNDICE DE FIGURAS

Figura 1.1: Árbol del Problema 3
Figura 2.1: Superordinación Conceptual 10
Figura 2.2: Subordinación Conceptual de la Variable Independiente 11
Figura 2.3: Subordinación Conceptual de la Variable Dependiente 12
Figura 2.4: Delimitación de zona de interés en un fotograma 16
Figura 2.5: Esquema de operación de MIL 19
Figura 2.6: Esquema de operación de TLD 20
Figura 2.7: Efectos de la sobre segmentación de objetos 21
Figura 2.8: Resultados de detección de objetos, algoritmo Faster RCNN 23
Figura 2.9: Ilustración del funcionamiento de la búsqueda selectiva 24
Figura 2.10: Esquema Operativo de RCNN 25
Figura 2.11: Comparativa entre SSD y YOLO 27
Figura 3.1: Zona de estudio seleccionada 32
Figura 6.1: Tiempos de ejecución y mAP de YOLOv3 con el juego de datos COCO .. 52
Figura 6.2: Metodología de la Propuesta 53
Figura 6.3: Selección de los vehículos a seguir 55
Figura 6.4: Carga de archivos de pesos y de configuración de la red 60
Figura 6.5: Carga de archivos de pesos y de configuración de la red 61
Figura 6.6: Ingreso de número de vehículos a seguir y sus descripciones 62
Figura 6.7: Procesamiento de la secuencia de entrada - seguimiento 63
Figura 6.8: Ingreso de datos generales del archivo de video 66
Figura 6.9: Procesamiento de la secuencia de entrada - conteo 67

ÍNDICE DE TABLAS

Tabla 3.1 Variable Independiente: Visión Artificial 34

Tabla 3.2 Variable Dependiente: Análisis de Tráfico Vehicular..... 35

Tabla 6.1 Tiempos de inferencia por imagen de algoritmos de aprendizaje profundo .. 57

Tabla 6.2 Resultados de seguimiento con secuencia de prueba 1 – 1080p 64

Tabla 6.3 Resultados de seguimiento con secuencia de prueba 1 – 480p 64

Tabla 6.4 Resultados de seguimiento con secuencia de prueba 2 – 1080p 64

Tabla 6.5 Resultados de seguimiento con secuencia de prueba 2 – 480p 65

Tabla 6.6 Resultados de conteo con secuencia de prueba 1 – 30 FPS. 68

Tabla 6.7 Resultados de conteo con secuencia de prueba 1 – 5 FPS. 68

Tabla 6.8 Resultados de conteo con secuencia de prueba 2 – 30 FPS. 69

Tabla 6.9 Resultados de conteo con secuencia de prueba 2 – 5 FPS. 70

Tabla 6.10 Resultados de conteo con secuencia de prueba 3 – 30 FPS. 70

Tabla 6.11 Resultados de conteo con secuencia de prueba 3 – 5 FPS. 71

Tabla 6.12 Tasa de Flops de Plataformas de Sistemas Embebidos..... 72

Tabla 6.13 Frecuencias observadas…….74

Tabla 6.14 Frecuencias teóricas……...74

UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS, ELECTRÓNICA E INDUSTRIAL – DIRECIÓN DE POSGRADO

MAESTRÍA EN AUTOMATIZACIÓN Y SISTEMAS DE CONTROL

TEMA

ANÁLISIS DE TRÁFICO VEHICULAR MEDIANTE VISIÓN ARTIFICIAL

AUTOR: ING. JOVANN PÉREZ NASSER

TUTOR: ING. DARÍO JOSÉ MENDOZA CHIPANTASI

FECHA: DICIEMBRE, 2018

RESUMEN EJECUTIVO

La inteligencia artificial posee 3 campos de estudio de interés: procesamiento de lenguaje, habla y visión artificial. Esta última comprende un campo de estudio de gran interés. La aplicación de la metodología de la ingeniería inversa, junto a la aplicación de algoritmos especializados, permiten automatizar procesos o solucionar problemas de forma que no podría ser llevada a cabo mediante un esquema operativo tradicional.

El presente trabajo de investigación propone el desarrollo de un programa orientado al análisis de tráfico en modalidad fuera de línea. El análisis de tráfico es el requisito previo a cualquier diseño y/o construcción de obras de vialidad; derivadas de la necesidad de crecimiento sostenible de una urbe. La aplicación de la visión artificial para el estudio de este tipo de procesos es muy poco intrusiva, por lo cual es una de las alternativas más populares, a más de confiables.

El presente trabajo de investigación divide el análisis de tráfico en dos problemáticas principales: el seguimiento de vehículos y el conteo de los mismos para obtener flujos. Para el seguimiento de vehículos se hace un estudio de algoritmos especializados,

eligiendo a los que ofrezcan mayor velocidad de procesamiento y mayor porcentaje de precisión. Se detalla el procedimiento utilizado para obtener estimaciones de distancia recorridas y velocidades promedio registradas, acompañadas de sus reportes respectivos. El conteo de vehículos es implementado mediante la utilización de una red neuronal de flujo directo. Esta alternativa supone un rendimiento significativamente superior a los sistemas basados en sustracción de fondo, debido a su mayor porcentaje de asertividad y tolerancia a perturbaciones.

Se realiza además un estudio de tiempos de ejecución de cada una de las partes del programa, con el fin de garantizar la transparencia de los resultados obtenidos, así como la portabilidad del mismo a entornos diferentes.

Se concluye con pruebas de validación de resultados y verificación de repetividad de resultados, en la cuales se comprueba un rendimiento satisfactorio y equilibrado.

Descriptores: Visión Artificial, OpenCV, Tracking, Redes Neuronales.

UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS, ELECTRÓNICA E INDUSTRIAL – DIRECIÓN DE POSGRADO

MAESTRÍA EN AUTOMATIZACIÓN Y SISTEMAS DE CONTROL

TITLE

VEHICLE TRAFFIC ANALYSIS USING ARTIFICIAL VISION

AUTHOR: ING. JOVANN PÉREZ NASSER

DIRECTED BY: ING. DARÍO JOSÉ MENDOZA CHIPANTASI

DATE: DECEMBER, 2018

EXECUTIVE SUMMARY

Artificial Intelligence possesses 3 major research fields: language processing, speech and computer vision. This last one is the actual tendency in a great deal of research works. Due to its reverse engineering nature and the application of specialized algorithms, [it] allows to automatize processes which has been stuck with old workflow structures.

This research work proposes the development of offline software for traffic analysis. Traffic analysis is the major pre requisite for all design or execution of any kind of road works, which are related to the growth and expansion of cities. The application of computer vision in this king of studies turns out to be almost nothing intrusive. This is the reason why the usage if vision algorithms is one of the most popular as reliable choices to apply.

Traffic analysis is considered in two major processes: vehicle tracking and vehicle counting. For vehicle tracking, a study of vision dedicated algorithms is done. We identify the fastest and most robust ones to perform vehicle tracking. An empirical procedure to determine average distances and speed with their respective report files is

also detailed. Vehicle counting is developed using a single shot neural network. Using this kind of deep learning algorithms offers greater performance compared to background subtraction related systems. This is because their improved precision and disturbances tolerance.

There is also a brief execution time study for both of the developed processes. This, in order to guarantee all the results obtained during the analysis. This also allows using the software in other host systems.

Finally, we show validation and integrity tests for results obtained. We verify a highly balanced and precise performance.

Keywords: Artificial Vision, OpenCV, Tracking, Neural Networks.

INTRODUCCIÓN

El trabajo de titulación presentado a continuación lleva por tema: "ANÁLISIS DE TRÁFICO VEHICULAR MEDIANTE VISIÓN ARTIFICIAL". El mismo se encuentra enfocado al estudio de recursos bibliográficos necesarios para la proposición de una metodología que permita alcanzar los objetivos propuestos en la investigación.

El contenido se organiza en capítulos, de la siguiente manera:

En el Capítulo I, se detalla EL PROBLEMA de la investigación mediante un análisis de la implementación de sistemas basados en visión artificial en un contexto local e internacional. Paralelamente se plantean los objetivos de la investigación.

En el Capítulo II, se desarrolla el MARCO TEÓRICO, con información bibliográfica referente a la investigación, detallando trabajos similares y describiendo contenidos teóricos que respalden la implementación del proyecto.

En el Capítulo III, se detalla la METODOLOGÍA a utilizar para la resolución del problema, así como los componentes que serán sometidos a análisis.

En el Capítulo IV, se realiza el ANÁLISIS E INTERPRETACIÓN DE LA INFORMACIÓN descubierta mediante el método bibliográfico, que permite describir las herramientas existentes para la modelación de la solución al problema de investigación.

En el Capítulo V, se exponen las CONCLUSIONES Y RECOMENDACIONES obtenidas en base al análisis bibliográfico realizado, las cuales permitirán proponer una solución al problema de investigación.

Finalmente, el Capítulo VI describe LA PROPUESTA mediante soluciones a cada uno de los componentes del problema de investigación. Para el efecto se desarrollan los programas de seguimiento y de conteo de vehículos. Adicionalmente se realizan pruebas de validación para las soluciones implementadas.

CAPÍTULO I

EL PROBLEMA DE INVESTIGACIÓN

1.1 Tema de investigación

ANÁLISIS DE TRÁFICO VEHICULAR MEDIANTE VISIÓN ARTIFICIAL

1.2 Planteamiento del problema

1.2.1 Contextualización

En al año de 1963, Lawrence Roberts en su tesis doctoral, analiza la factibilidad de extraer información de 3D a partir de vistas 2D de poliedros (Roberts, 1963). Este trabajo le valdría posteriormente ser reconocido como el primer impulsador de la visión artificial (Huang, 2003) Inmediatamente después, varios investigadores del MIT (*Massachusetts Institute of Technology*, Instituto de Tecnología de Massachusetts), usaron como base el trabajo de Roberts para aplicaciones con vistas simples de bloques geométricos. Fueron necesarios pocos meses para que los investigadores involucrados encontrasen que, en lugar de trabajar con vistas de caras de poliedros, era necesario estudiar imágenes capturadas del mundo real. Esto marcaría el inicio formal de lo que se conoce hoy en día como Visión Artificial, así como su aplicación a la solución de problemas de diversa índole tanto productivos como investigativos.

Las tareas de reconocimiento y detección de objetos han constituido y constituyen un campo de gran interés de estudio para la visión artificial. El primer detector de rostros exitoso fue presentado en el año 2001 por James Viola y Michael Jones (Viola, 2001). Para el 2005, el trabajo de investigación de Navneet y Triggs revolucionó la detección

de peatones mediante la utilización de histogramas orientados mediante gradientes (HOG) (Dalal & Triggs, n.d.). Este trabajo demostró el potencial de la aplicación de la visión artificial para su utilización en temáticas relacionadas al análisis del tráfico. La creciente necesidad de detectar objetos de forma más rápida y precisa (principalmente personas y vehículos) empezaría con el auge de los algoritmos de aprendizaje profundo. En el 2012 surge AlexNET, un algoritmo de reconocimiento de objetos en escenas que logró un 85% de precisión en su análisis (Krizhevsky & Hinton, n.d.). En años posteriores, aparecen algoritmos con distintos enfoques en velocidad de procesamiento y precisión de análisis tales como RCCNN, Fast RCNN, Mask RCNN, SSD, YOLO. Estos comprenden la tendencia actual como herramienta para desarrollo de sistemas de detección y reconocimiento mediante visión artificial, en varios campos de investigación, especialmente en temáticas relacionadas al tráfico vehicular.

La aplicación de algoritmos de aprendizaje profundo ha permitido desarrollar varios sistemas de análisis de tráfico tanto en tiempo real como fuera de línea. Algunos sistemas en tiempo real se valen de dos bloques principales de procesamiento. Un ejemplo consiste en un análisis conjunto entre una red neuronal de alta precisión y la utilización de filtros de Kalman para la detección y conteo de vehículos (Yang & Qu, 2017). Adicionalmente se han presentado también sistemas tolerantes a perturbaciones, oclusiones y cambios en enfoque mediante uso de arquitecturas en red (Kim & Sim, 2017), o sistemas enfocados al estudio de tendencias de flujo de tráfico en zonas conflictivas con el fin de aumentar la seguridad vial (Battiato, Farinella, & Giudice, 2013).

A nivel local se ha realizado una serie de estudios relacionados con el análisis del tráfico vehicular, más no a la utilización de visión artificial y algoritmos de aprendizaje profundo para tratar la temática de estudio. El presente trabajo de investigación propone un sistema de análisis de tráfico basado en visión artificial. La determinación de parámetros tales como el flujo vehicular total constituye, un pre requisito imprescindible al diseño y ejecución de obras de vialidad. Y a su vez, del desarrollo ordenado y sistemático de una urbe.

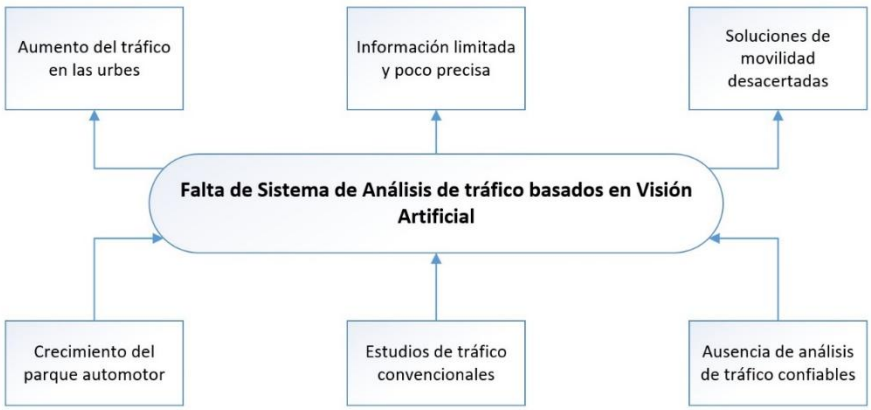


Figura 1.1: *Árbol del Problema*
Fuente: *El Autor*

1.2.2 Análisis Crítico

El tráfico es uno de los problemas inherentes al desarrollo urbano, debido a esto, su estudio es motivo de interés por parte de autoridades locales. Las soluciones y obras que mejoren la movilidad de una urbe, deben estar en concordancia a análisis de tráfico fidedignos. El uso de sistemas basados en visión artificial ofrece soluciones poco intrusivas y de alta confiabilidad relacionados a esta temática.

Los sistemas tradicionales de conteo de vehículos basados en la utilización de zonas de interés y sustracción de fondo, van siendo reemplazados por algoritmos de aprendizaje profundo. La versatilidad de este tipo de algoritmos ha permitido crear sistemas de conteo de vehículos en tiempo real (Yang & Qu, 2017). Otros trabajos se valen de detectores para ser utilizados en conjuntos con UAVs para la identificación y seguimiento de peatones o vehículos (Saribas & Cevikalp, 2018). Sin embargo, se registran algunos casos particulares en los cuales un enfoque tradicional de conteo de vehículos opera conjuntamente con redes neuronales, para reducir los efectos indeseados de los cruces de carril (o ingreso parcial al área de detección) (Zheng & Wang, 2015).

A nivel local, la Dirección de Tránsito, Transporte y Movilidad de la Municipalidad de Ambato cuenta con sistemas de conteo de vehículos localizados en punto de interés repartidos principalmente en el centro de la urbe (Maldonado, 2016). Estos sistemas utilizan la metodología de sustracción de fondo, por lo cual son sensibles a una serie de perturbaciones tales como: oclusión, desenfoque, ingreso parcial al área de detección,

entre otros. Resulta necesaria la aplicación de nuevas tecnologías para el análisis del tráfico a nivel local, con el fin de obtener resultados más confiables a partir de algoritmos más robustos.

El análisis de tráfico mediante visión artificial comprende un área de investigación de importancia significativa para el desarrollo sostenible de la movilidad en zonas urbanas.

1.2.3 Prognosis

El desarrollo de las ramas aplicadas de la ingeniería se logra únicamente a través de la aplicación de pruebas de ensayo, obtención de errores y ajustes del modelo. La visión artificial especialmente, pese a sus amplias capacidades, requiere de pruebas de calibración y optimización, que permitan que el sistema a implementar se siga ajustando de manera progresiva a la obtención completa de su objetivo. La compartición y difusión de estas pruebas y conocimientos dentro la sociedad de investigadores, permitirá reducir tiempos de desarrollo de aplicaciones específicas de visión artificial. De esta manera, se podrá automatizar un gran número de procesos para los cuales aún no se ha establecido un modelo de trabajo definido.

El creciente problema del tráfico se verá notablemente agravado a mediano plazo, de no utilizarse técnicas de análisis basadas visión artificial. Es necesario aplicar las capacidades del procesamiento de imágenes, detección y seguimiento de objetos, para poder realizar un análisis y recopilación de datos para el estudio de soluciones flexibles.

La facilidad de instalación de los elementos del sistema, a más de la amplia variedad de algoritmos y entornos de desarrollo disponibles, hace altamente factible el desarrollo del presente proyecto de investigación. El no investigar y no fomentar la aplicación de la visión artificial para el análisis de tráfico, derivará en un estancamiento en el desarrollo de las urbes; mismo que puede ser evitado mediante la aplicación de esta tecnología flexible.

1.2.4 Formulación del Problema

¿Puede ser la visión artificial utilizada como herramienta para el análisis de tráfico vehicular?

1.2.5 Preguntas Directrices

¿Cuáles algoritmos de visión pueden ser aplicados para realizar el seguimiento de un objeto?

¿Cuál es la tolerancia aceptable de un algoritmo de visión específico al momento de realizar el seguimiento de un objeto?

¿Cuáles métodos y/o calibraciones pueden ser aplicados para conseguir un óptimo encuadre del objeto a seguir?

¿Cuáles entornos de desarrollo enfocados a visión artificial y tratamiento de imágenes se encuentran actualmente disponibles?

¿Resulta factible la utilización de software libre o de pago?

1.2.6 Delimitación

- *Área Académica:* Visión Artificial.
- *Línea de Investigación:* Adquisición y Tratamiento Óptico de Imágenes.
- *Sub Línea de Investigación:* Medición y Adquisición de datos.
- *Delimitación Espacial:* El presente trabajo de investigación será llevada a cabo en ubicaciones seleccionadas por el investigador en la ciudad de Ambato.
- *Delimitación Temporal:* Una vez aprobado por el Honorable Consejo Directivo de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, la presente investigación se llevará a cabo en el transcurso de 6 meses.

1.3 Justificación

Las soluciones de ingeniería basadas en el uso de visión artificial, constituyen una tendencia tecnológica que se encuentra en pleno auge. El análisis de tráfico mediante visión artificial, abre un abanico de oportunidades que pueden ser aprovechadas en beneficio del desarrollo de las urbes a nivel global.

La reducción del tráfico y el aumento de la seguridad vial dependen de la obtención de parámetros tales como el flujo vehicular total, intensidades, entre otras. A nivel local, no se dispone de sistemas de conteo incorporados a vías rápidas y/o urbanas. La implementación de sistemas basados en algoritmos de visión permite obtener este tipo

de datos de forma precisa mediante la combinación de técnicas adecuadas para cada ambiente operativo (Miller, Thomas, Eichel, & Mishra, 2015).

Datos tales como: velocidades de desplazamiento, cantidad de vehículos, rutas descritas; son la base de análisis de diseño de sistemas de control inteligente de tráfico. Los datos recopilados por un sistema de análisis de tráfico mediante visión artificial pueden ser empleados por ejemplo, en la calibración de tiempos y horarios de semáforos de lógica difusa o sistemas inteligentes de control de tráfico. A más del enfoque de la clasificación y la detección, al análisis del tráfico puede ser tratado mediante regresión estadística para la predicción de flujos y diseño de planes de contingencia (Wan, Yuan, & Wang, 2017). Análogamente, trabajos de investigación relacionados a nivel local han planteado metodologías del análisis para la obtención del estado del tráfico en tiempo real, utilizando los datos proporcionados por la Dirección de Transporte, Tránsito y Movilidad de la Municipalidad (DTTM) de Ambato (Maldonado, 2016).

La aplicación de la visión artificial para el estudio del tráfico vehicular influirá positivamente en el desarrollo ordenado de las urbes, así también como para aumentar la seguridad en las vías.

1.4 Objetivos

GENERAL

- Implementar un sistema de visión artificial que permita realizar análisis del tráfico vehicular.

ESPECÍFICOS

- Analizar algoritmos de seguimiento de objetos.
- Determinar la cantidad de vehículos enfocados, así como sus respectivas velocidades de desplazamiento y/o rutas descritas.
- Desarrollar un algoritmo de visión artificial que ofrezca velocidad de identificación y precisión de encuadre para los objetos de estudio seleccionados.

CAPÍTULO II

MARCO TEÓRICO

2.1 Antecedentes Investigativos

Las tareas de detección, rastreo y conteo de vehículos, son elementos clave en el análisis del flujo de tráfico. Los datos de estos análisis permiten mejorar la planeación y el control del tráfico en urbes de considerable flujo vehicular. Las soluciones basadas en visión artificial, no suponen ninguna perturbación al entorno del problema, a más de que su instalación resulta sencilla. Por esta razón, el análisis de tráfico mediante visión artificial supone un campo de gran interés, por lo cual ha sido objeto de estudio por parte de la comunidad científica.

El trabajo de investigación titulado "Detección y Conteo de Vehículos en Tiempo Real en Escenarios Complejos Utilizando Modelos de sustracción de Fondo con Descomposición de Bajo Nivel" (Yang & Qu, 2017) se propone un sistema de detección y conteo de vehículos en escenarios de tráfico pesado. El sistema utiliza un filtro de Kalman para detectar los múltiples vehículos de la escena, evitando incrementar el contador accidentalmente. La intención principal de los autores es proponer una solución relativamente económica para el conteo de vehículos, tarea que se realiza usualmente con sistema de adquisición de datos basados en sensores magnéticos o de ultrasonido.

El trabajo de investigación "Detección y Conteo de Vehículos Basado en Visión Artificial para Análisis de Flujo de Tráfico" (Zhang, Liu, Gao, Li, & Wang, n.d.), propone un sistema de conteo de vehículos en videos de vigilancia captados por cámaras de seguridad. La propuesta difiere significativamente de sistemas similares, en que hace uso de convolución de redes neuronales. Este esquema de trabajo requiere de un entrenamiento previo con videos de tráfico similares. Una de las desventajas de

utilizar redes neuronales, supone el aumento de los tiempos de procesamiento. Sin embargo, el sistema detallado es capaz de responder efectivamente frente a escenarios de oclusión de uno o varios de los vehículos a contar.

El trabajo de investigación "Análisis de Conflictos de Trafico Mediante Visión Artificial" (Battiato et al., 2013) presenta un sistema orientado a mejorar la seguridad vial de zonas con altos índices de accidentes vehiculares. Mediante la captura de imágenes desde estaciones ubicada de forma estratégica, se pretende incrementar significativamente la cantidad de información relativa al incidente, más allá de los datos obtenidos por investigaciones, testimonios y ordenadores a bordo de los vehículos implicados. Este estudio pretende determinar la conducta vehicular de los usuarios, para optimizar el tráfico y la seguridad de las zonas conflictivas.

El trabajo de investigación titulado "Monitoreo de Trafico con Visión Artificial" (Kun & Vámossy, 2009), propone un sistema para el conteo de vehículos en tiempo real. La condición operativa del algoritmo requiere de luz diurna, pudiendo trabajar a una velocidad de procesamiento de cuadros de 15fps o 30fps. El sistema está conformado por tres bloques principales: captura de video, conteo de vehículos e interfaz gráfica. Para optimizar el rendimiento del aplicativo, el sistema ejecuta cada uno de sus bloques en hilos de ejecución independientes.

El trabajo de investigación titulado "Detección en Tiempo Real de Vehículos en Movimiento, Seguimiento y Conteo con Open CV"(Li, Liang, & Zhang, 2014), presenta un sistema de análisis de tráfico fuera de línea: es decir, trabaja consecuencias de video pre grabadas. El sistema utiliza sustracción adaptativa del relleno de la imagen, en conjunto con detectores virtuales y algoritmos dinámicos de detección de formas.

El trabajo de investigación titulado "Detección de Vehículos utilizando Convolución Rápida Simplificada de Área con Redes Neuronales" (S. Hsu & Chuang, 2018), hace uso de redes neuronales para la detección y conteo de vehículos. El esquema de trabajo del sistema presentado permite prescindir del bloque de reconocimiento de objetos, mismo que ha sido calificado como redundante según sus autores. De esta manera se logran reducir tiempos de ejecución del algoritmo. El sistema requiere de un proceso de pre entrenamiento y ajuste fino.

El trabajo de investigación titulado “Un Nuevo Enfoque para el Análisis En Línea y Multivariante de Redes de Tráfico” (Kim & Sim, 2017), se propone un sistema de análisis multivariante de tráfico. Una de las limitaciones de los sistemas de monitoreo de tráfico consiste en la alta cantidad de información a procesar en intervalos de tiempo limitados, a más del surgimiento de perturbaciones indeseadas en el área de captura. La utilización de un modelo de trabajo en red, permite implementar un sistema de análisis de tráfico que responde mejor a oclusiones y cambio de enfoque de los fotogramas a estudiar.

En el trabajo de investigación titulado “Detección y Análisis de Movimiento usando Visión Artificial” (Sanabria JJ ;Archila JF, 2011), propone un modelo de detección de movimiento de personas. Dicho objetivo es conseguido mediante el uso de filtros discriminantes, reducción de ruido y eliminación del fondo de la imagen, para la obtención en tiempo real del objeto de interés. Paralelamente, se ofrecen una visión general sobre los distintos campos para la potencial aplicación de la visión artificial como solución a procesos automatizados.

2.2 Categorías Fundamentales

- *Variable Independiente:* Visión Artificial.
- *Variable Dependiente:* Análisis de Tráfico Vehicular.

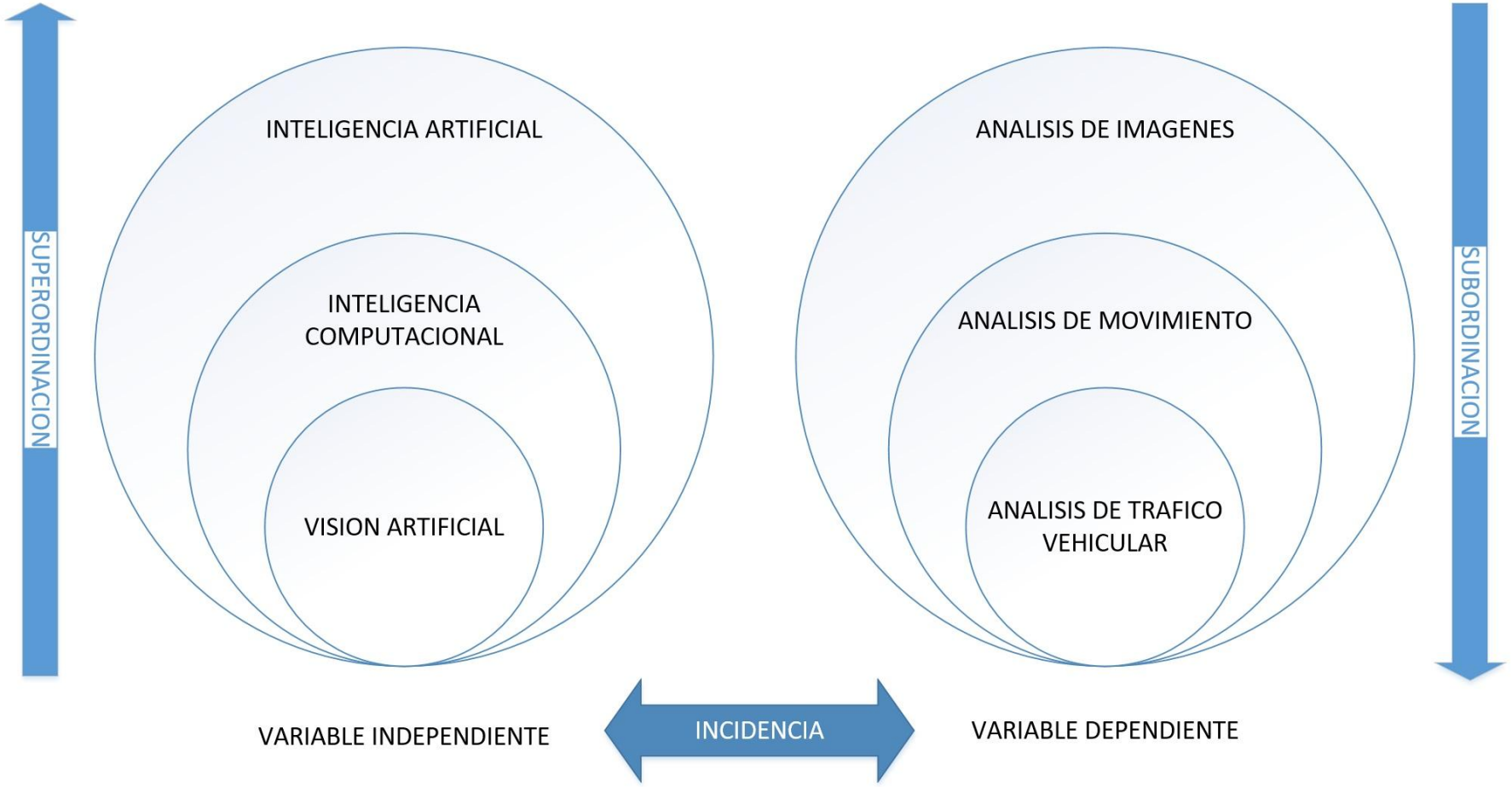


Figura 2.1: *Superordinación Conceptual*
Elaborado por: *El Investigador*

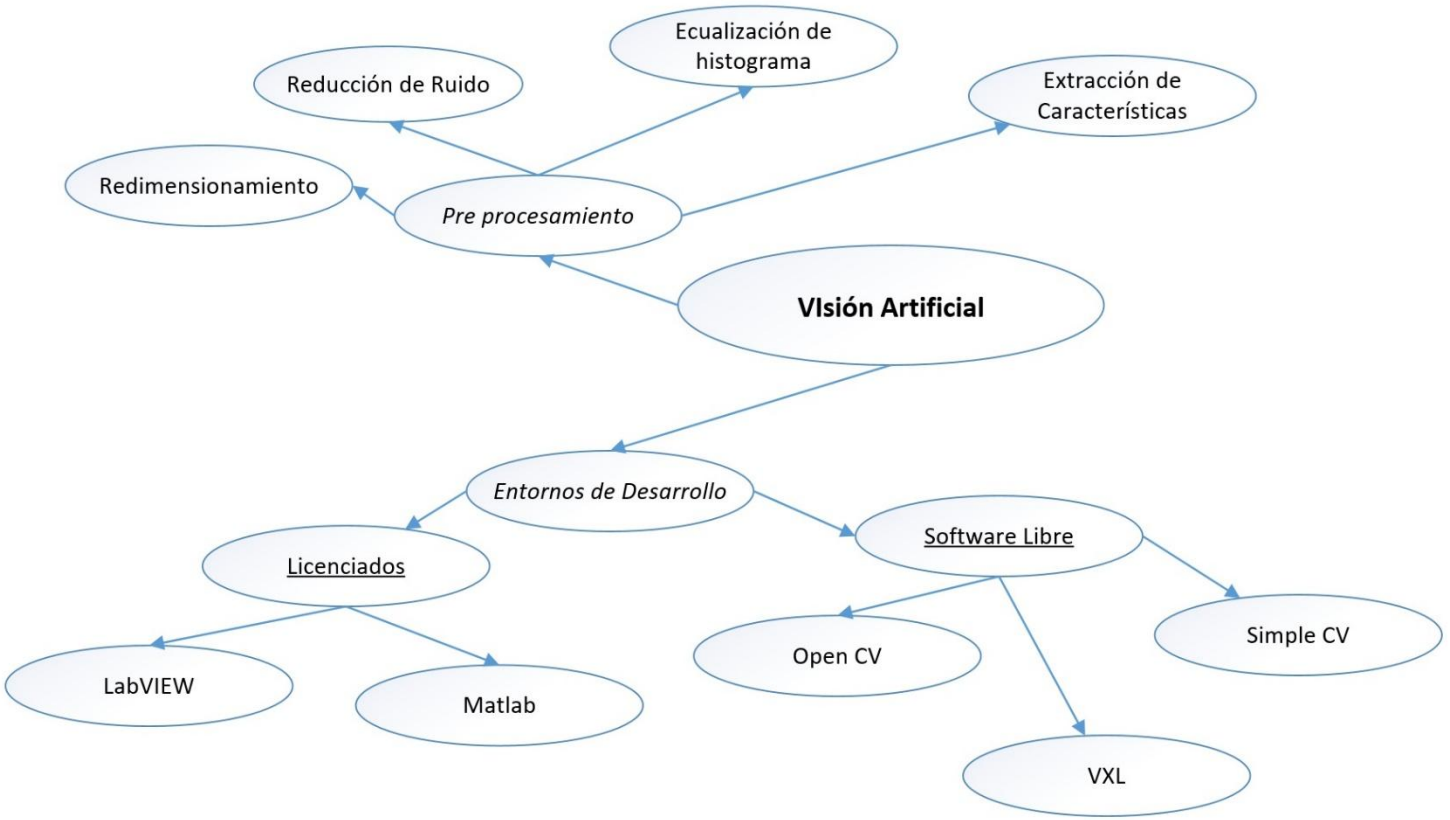


Figura 2.2: *Subordinación Conceptual de la Variable Independiente*
Fuente: *El Investigador*

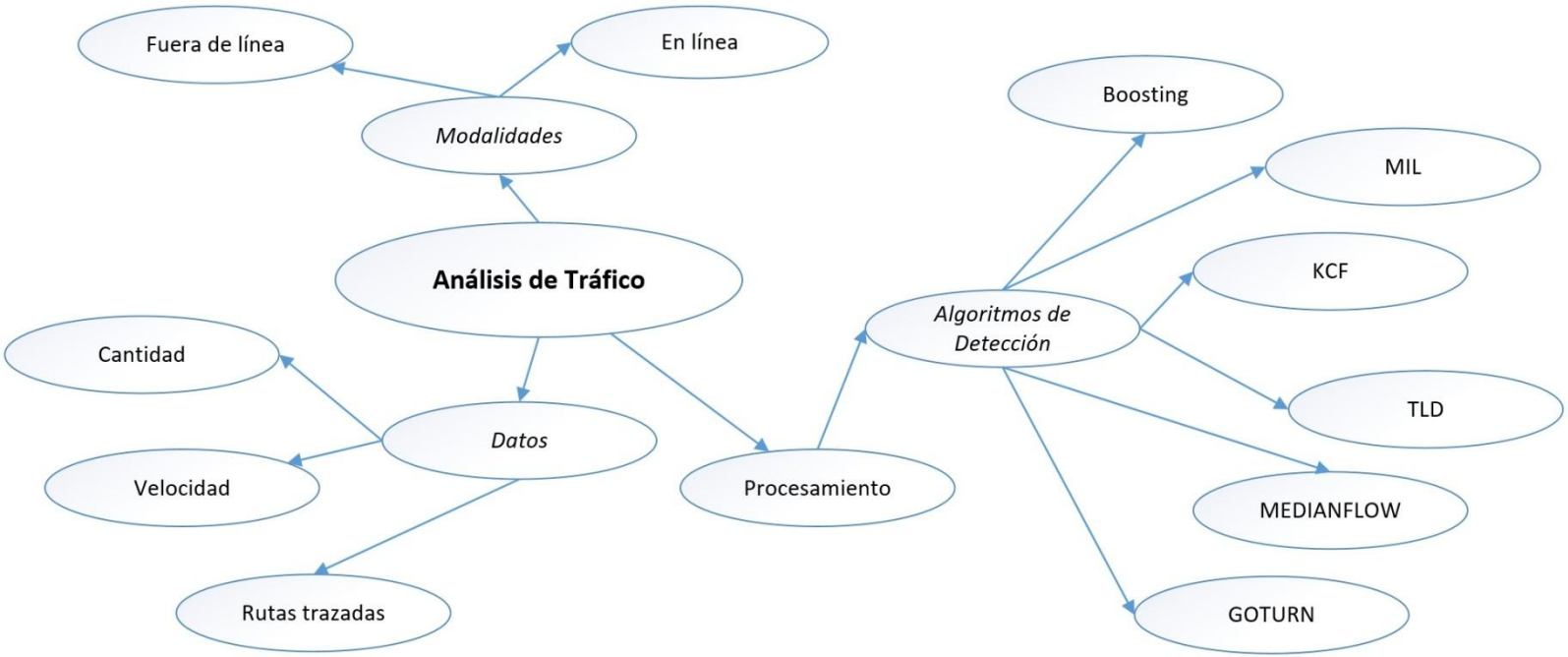


Figura 2.3: *Subordinación Conceptual de la Variable Dependiente*
Fuente: *El Investigador*

2.3 Fundamentación Filosófica

El presente trabajo de investigación se relaciona con el paradigma filosófico crítico propositivo; es crítico ya que se realiza un análisis objetivo del problema de una realidad y propositivo ya que plantea una propuesta de solución al problema investigado basado en la existencia de múltiples realidades técnicamente desarrolladas.

2.4 Fundamentación Legal

El presente proyecto de investigación se fundamentará en la estructura del Código Orgánico de La Producción, Comercio e Inversiones:

Título Preliminar, Del Objetivo y Ámbito de Aplicación. Artículo 4, Inciso t:

"Fomentar y apoyar la investigación industrial y científica, así como la innovación y transferencia tecnológica."

Libro I: Del Desarrollo Productivo, Mecanismos, y Órganos de Competencia. Título I: Del Desarrollo Productivo y su Institucionalidad. Capítulo I: Del Rol del Estado en el Desarrollo Productivo. Artículo 5, Inciso d:

Para la transformación de la matriz productiva, el Estado incentivará la inversión productiva, a través del fomento de:

"La generación de un ecosistema de innovación, emprendimiento y asociatividad mediante la articulación y coordinación de la iniciativas públicas, privadas y populares y solidarias de innovación y transferencia tecnológica productivas, y la vinculación de la investigación a la actividad productiva. Así también fortalecerá los institutos públicos de investigación y la inversión en el mejoramiento del talento humano, a través del programa de becas y financiamiento de estudios de tercer y cuarto nivel;"

2.5 Visión Artificial

Comprende a un conjunto de técnicas y procesos para la adquisición, procesamiento y análisis de imágenes del mundo real mediante un computador. Su alta complejidad se debe a su naturaleza inversa inherente. Es decir, que partiendo de una imagen bidimensional, se pretende reconstruir las características tridimensionales de dicha escena capturada (Szeliski, 2010).

Debida a su estrecha relación con la replicación de una capacidad humana, la visión artificial constituye de uno de las ramas de apoyo de la robótica. Un claro ejemplo viene definido por los sistemas pick & place que utilizan los brazos robóticos para recoger y ubicar / instalar objetos dentro de una línea de producción.

Se lista a continuación, algunas aplicaciones y/o metas de la visión artificial:

- Reconocimiento de rostros, interpretación de expresiones.
- Referencia visual para vehículos autónomos.
- Análisis de imágenes médicas. Adquisición, procesamiento y diagnóstico.
- Reconocimiento de texto.
- Biometría (de rostros o dinámica).
- Seguimiento de objetos.
- Sistemas anti – colisiones.
- Visión estereoscópica.

2.5.1 Pre Procesamiento de Imágenes

Previo a la implementación de cualquier aplicación de visión artificial, es necesario someter a la imagen capturada a un pre procesamiento que facilite la ejecución del o los algoritmos seleccionados. Se lista a continuación algunas de las tareas de pre procesamiento de imágenes as comunes:

- Redimensionamiento: comprende una operación básica de cambio de dimensiones del archivo de imagen a procesar. Trabajar con archivos grandes no implica resultados mejores, a más de elevar los tiempos de procesamiento. Lo cual resulta crítico para aplicaciones de tiempo real.
- Reducción de ruido: consiste en la eliminación de características indeseadas en la imagen capturada, debido a condiciones propias del ambiente de trabajo. Lo mismo se logra a través de substracciones o el “suavizado” de la imagen, mediante filtros gaussianos.
- Ecualización de histograma: generación de una nueva imagen con el mismo valor de intensidad a píxeles que posean valores congruentes entre sí (umbrales). Resulta de particular interés para la identificación de detalles que se ven ocluidos en la imagen original.

- Extracción de características: ciertos algoritmos requieren trabajar con una serie de parámetros propios de los pixeles de la imagen. Dichos parámetros se conocen como características.
    - Características de nivel 1 y 2: consisten en datos de intensidad de color, contraste, saturación, entre otros; de un pixel en específico.
    - Características de nivel 3: las características de nivel 3 son el resultado de aplicar un algoritmo de visión específico a las características de nivel 1 y 2.
    - Vectores de características: contenedores de característica de píxeles. Los mismos son creados con el objeto de reducir tiempo de cómputo, o presentar datos de entrada compatibles con ciertos algoritmos.

2.6 Análisis de tráfico vehicular

Una de los campos potenciales de aplicación de la visión artificial, consiste en el uso de la misma para realizar tanto monitoreo como análisis de tráfico. La creación de sistemas de conteo de vehículos en zona de interés, puede ser nombrada como ejemplo principal. Datos adicionales tales como velocidad de los vehículos, rutas trazadas, e incluso información del tiempo (recogida por sistemas embebidos independientes); son de utilidad a la hora realizar análisis que permitan mejorar la control del flujo del tráfico en zonas de interés.

Existen dos modalidades principales para realizar este tipo de análisis (Kim & Sim, 2017):

- En línea: es decir, en tiempo real. Estos sistemas deben ser calibrados para entornos concretos y se ven limitados por sus condiciones óptimas de funcionamiento (luminosidad ambiental, oclusiones, saturaciones, cambios de ángulo de análisis).
- Fuera de línea: comprenden el análisis de secuencias de video pre grabadas. Los resultados de estos análisis superan a los de tiempo real, ya que no encuentran una limitante en el tiempo de procesamiento de fotograma. Además, estos son la opción a considerar con secuencias de video que presentan perturbaciones no deseadas captadas en su entorno operativo.

Uno de los procedimientos elementales a la hora de realizar análisis de tráfico mediante visión artificial, consiste en la determinación y normalización de la zona de detección o captura de imágenes. Es decir. Delimitar el área efectiva para el análisis. Esto se logra a través del trazado de zonas de interés dentro del fotograma a procesar, como se puede apreciar en la figura 2.4.

Figura 2.4: *Delimitación de zona de interés en un fotograma*
Fuente: (S.-C. Hsu, Huan, & Chuang, 2018)

Posteriormente, el fotograma se encuentra listo para todas las tareas de procesamiento asociadas al análisis. Uno de los procedimientos de especial interés para el presente proyecto de investigación, consiste en la detección (conteo) y rastreo (trazado de ruta) de los vehículos. Para realizar ambas tareas, es necesario hacer uso de algoritmos de rastreo o seguimiento de objetos.

2.6.1 Seguimiento de Objetos Mediante Visión Artificial

Puesto de una manera sencilla, ubicar un objeto específico dentro de una secuencia de imágenes, es a lo que se conoce como "tracking" o (rastreo, seguimiento). Pese a que el concepto suena sencillo y concreto, rastrear un objeto consolida varias ideas diferentes similares en concepto, pero variantes en técnica. Se detalla a continuación, de manera breve algunas de dichas ideas.

- *Densidad de Flujo Óptico:* este tipo de algoritmos permiten determinar el vector de dirección de un píxel específico dentro de una imagen.
- *Caudal de Flujo Óptico:* este tipo de algoritmos, permiten realizar el seguimiento de los puntos característicos de una imagen.
- *Filtros de Kalman:* un algoritmo de visión bastante popular, que permite determinar la posición de un objeto en base a información de movimiento

previa. Una de las primeras aplicaciones de este tipo de algoritmo fue su utilización para el disparo guiado de misiles.

- *Varianza Máxima:* estos algoritmos permiten determinar los picos de una función de densidad, lo cual resulta útil para realizar tracking.
- *Localizadores de Único Objeto:* estos algoritmos encuadran el objeto de interés dentro de un rectángulo. Con esta referencia previa, se puede realizar el seguimiento del objeto en fotogramas subsecuentes.
- *Localizadores de Múltiples Objetos:* cuando se cuenta con un algoritmo de detección de alta velocidad, no se localiza uno, sino varios objetos. La función de estos algoritmos consiste en determinar que rectángulo de identificación pertenece al objeto del fotograma subsecuente.

El seguimiento de objetos posee varias aplicaciones de interés tanto como para procesos de automatización como de investigación. Dos ejemplos exitosos de rastreo con dominios específicos son el seguimiento de rostros y de personas. La detección de objetos también encuentra en escenario potencial de alto interés para su aplicación en su incorporación de sistemas de navegación de vehículos automotores para evitar posible colisiones (Ambata, Lijauco, Nuval, & Vergara, 2016).

Sin embargo, realizar el rastreo de un objeto genérico sigue siendo una tarea desafiante. El seguimiento de objetos en general es implementado mediante el uso de algoritmos de seguimiento en conjunto con el uso de clasificadores o entrenamiento del sistema.

2.6.2 Algoritmos de Visión para Detección de Objetos

A. Boosting Tracker

Este algoritmo está basado en el código que funciona internamente en un detector de rostros de cascada (HAAR). Este clasificador, necesita de entrenamiento previo a su ejecución en tiempo real con negativos y positivos del objeto a seguir. El contenedor *ROI* (*Region Of Interest*, Zona de Interés) inicialmente proporcionado por el usuario (o un algoritmo de detección diferente) es interpretado como los positivos del objeto. El resto de píxeles que se encuentran fuera del *ROI*, son tratados como negativos o el relleno de la imagen.

Para un fotograma determinado en un instante de tiempo ***t***, el algoritmo es ejecutado sobre todos y cada uno de los píxeles de la vecindad correspondiente a la última zona de

interés. La nueva ubicación del objeto está determinada por la zona que ha obtenido el máximo puntaje de aciertos para el clasificador.

Es uno de los algoritmos más antiguos y su desempeño no es particularmente notable, especialmente cuando la detección del objeto de interés ha fallado.

B. MIL Tracker

MIL (*Multiple Instance Learning*, Reconocimiento de Múltiples Instancias) posee un modus operandi bastante similar al del algoritmo descrito previamente. Sin embargo, en lugar de considerar únicamente como un positivo a la ubicación actual del objeto; el algoritmo verifica vecindades próximas para ubicar potenciales positivos. A priori, puede parecer una mala idea, ya que en la mayoría de todos estos potenciales positivos, el objeto de interés no se encontraría enmarcado (centrado) correctamente. El reconocimiento de múltiples instancias no utiliza en sí ejemplo puntuales de positivos y negativos, sino zonas de positivos y negativos. Es así que si la ubicación del objeto no es del todo precisa, existe una alta probabilidad de que la zona de positivos contenga una imagen en la cual el objeto se encuentra adecuadamente centrado.

MIL entrena a un clasificador discriminante que, que permite separar al objeto de interés del relleno de la imagen. Adicionalmente, los algoritmos de seguimiento de objetos basados en detección han demostrado ser altamente eficaces en tiempo real (Babenko, Belongie, & Yang, 2009).

Uno de los problemas conocidos de los algoritmos de tracking que utilizan como referencia o entrenamiento imágenes estáticas precargadas en su modelo, es su mediocre o pobre desempeño en situaciones en las cuales el objeto de interés tiende a cambiar de apariencia. *MIL* propone una solución enfocada a la utilización de un modelo de referencia cambiante en el tiempo, basado en la última detección exitosa del objeto en cuestión; tal y como se puede apreciar en la figura 2.5.

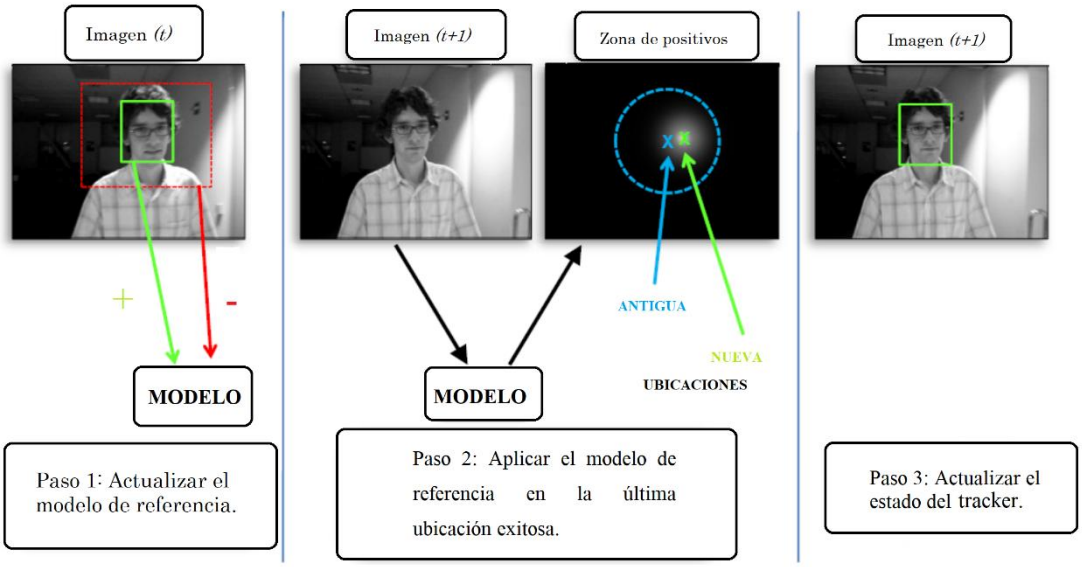


Figura 2.5: *Esquema de operación de MIL*
Fuente: (Babenko, Ming-Hsuan Yang, & Belongie, 2011)

C. KCF Tracker

Los Filtros de Correlación Kernelizados (*Kernelized Correlation Filters*), se basan también en los principios de los dos algoritmos previamente detallados. KCF considera el hecho de que las zonas de positivos de MIL, poseen largas áreas de superposición. Esta superposición permite aplicar ciertas propiedades matemáticas convenientes para mejorar al mismo tiempo tanto la velocidad como la precisión de la detección.

Pese a sus mejoras notables, este algoritmo no puede recuperar la posición del objeto de interés, después de una oclusión total de la imagen.

D. TLD Tracker

Seguimiento, Aprendizaje y Detección (*Tracking, Learning and Detection*). En este algoritmo, el proceso se seguimiento se divide en tres etapas principales: seguimiento del objeto a corto plazo, aprendizaje y detección. La salida tiende a saltar entre objetos de las mismas características. Por ejemplo, al realizar el seguimiento de una persona en un cuadro, si aparece una segunda, se ubicará de forma temporal a la última. Por lo cual, la alta tasa de falsos positivos es una de las desventajas considerables de este algoritmo.

Otro de sus inconvenientes consiste en la determinación manual de la región de seguimiento, lo cual impacta directamente en el desempeño en tiempo real del sistema (Wen, Wu, & Li, 2015). La utilización de filtros de Kalman y modelos de Markov pueden mejorar de forma apreciable la precisión del algoritmo. Un esquema simplificado de la operación del algoritmo es detallado en la figura 2.6.

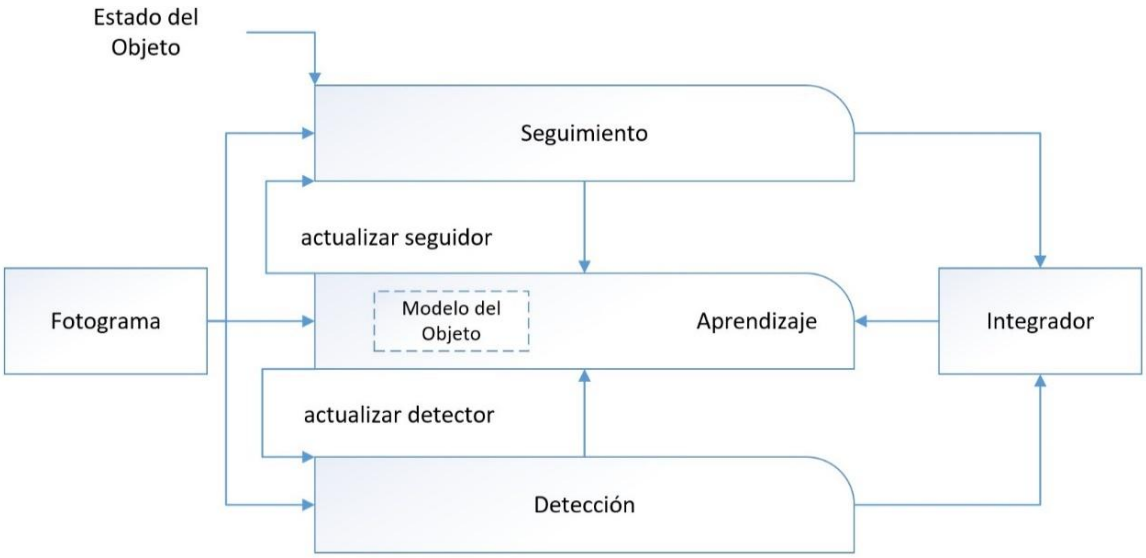


Figura 2.6: *Esquema de operación de TLD*
Fuente: (Wen et al., 2015)

Por el otro lado, este algoritmo responde mejor a la oclusión en varios cuadros de la secuencia.

E. Medianflow Tracker

El modo de funcionamiento de este algoritmo calcula una media de los errores de detección de la ubicación del objeto entre cuadros previos y actuales de la imagen. Minimizar el valor de este error, permite determinar una trayectoria confiable de seguimiento para el objeto en la secuencia de video.

Este algoritmo posee un funcionamiento destacable en el seguimiento de objetos que describen una trayectoria no aleatoria, sin embargo, tiende a fallar durante prolongados períodos de tiempo de ejecución.

F. Goturn Tracker

A diferencia de todos los algoritmos previamente descritos, este se caracteriza por ser el único que basa su funcionamiento en Convolución de Redes Neuronales. Utiliza además un modelo de entrenamiento previamente precargado, por lo cual su velocidad de ejecución es alta. Responde adecuadamente a cambios no predecibles de trayectoria, sin embargo, no maneja bien la oclusión del objeto a seguir.

2.7 Algoritmos de Visión para Conteo de Objetos

Dentro del campo de visión artificial, la metodología de sustracción de fondos ha sido una de las alternativas más populares para realizar conteos de distintos objetos (Li et al., 2014). Destaca su relativa simplicidad de implementación y efectividad de procesamiento. Sin embargo, utilizar esta metodología, conlleva establecer una normalización de la captura de imágenes en el entorno operativo, siendo la oclusión de objetos, uno de los problemas a los cuales no puede hacer frente.

Otro de los problemas asociados comprende la sobre segmentación (Li et al., 2014), producto de la utilización de filtros gaussianos. Uno o varios de los elementos a contar resulta divido en 2 partes distintas, o a su vez posee bordes débiles, como se puede apreciar en la figura 2.7.

Figura 2.7: *Efectos de la sobre segmentación de objetos*
Fuente: (Li et al., 2014)

Esta metodología comprende de una serie de pasos y algoritmos complementarios y subjetivos a cambios y optimizaciones. Utilizarla comprende una solución factible únicamente para conteo de objetos, más no para clasificación de los mismos.

Otra de las alternativas disponibles para realizar conteo de objetos mediante visión artificial comprender la utilización de detectores de aprendizaje profundo o redes neuronales. Los problemas de la metodología previa, oclusión y clasificación, se ven solventados significativamente con este tipo de solución. Por otra parte, el coste computacional asociado a estos algoritmos es relativamente mayor, condicionando su utilización en aplicaciones de tiempo real.

2.7.1 Algoritmos de Aprendizaje Profundo

Uno de los ejemplos más asimilables con los cuales se pueden ilustrar de manera adecuada el concepto de una red neuronal, consiste en compararla con una caja negra. Este componente posee una característica única asociada, que le permite discernir la identidad de un elemento. Para efectos didácticos, la red podrá predecir si el objeto que se le presente es o no es una fruta. Idealmente nuestro sistema tendrá únicamente dos salidas: es fruta o no es fruta. En efecto práctico, se asocia un porcentaje de confianza y uno de duda respecto a la salida de la red. En otras palabras, podría estar un 80% segura de que es una fruta y un 20% segura de que no es una fruta. Una vez que se ha entendido a groso modo la lógica de operación de una red, se puede deducir que la utilización de las mismas genera gran interés dentro de visón artificial.

Estudios realizados sobre la capacidad de la redes neuronales convolucionales en el campo de la visión artificial, permitió a los investigadores desarrollar las RCNNs, o redes neuronales convolucionales mediante regiones. La aplicación de este tipo de redes resulta idónea para tareas de detección, clasificación y etiquetado de objetos en imágenes, frente a las metodologías que venían aplicando con el enfoque tradicional. Por lo general, la salida generada por este tipo de sistemas consiste en una serie de contenedores o *bounding boxes* que enmarcan los objetos detectados, tal y como se puede apreciar en la figura 2.8.

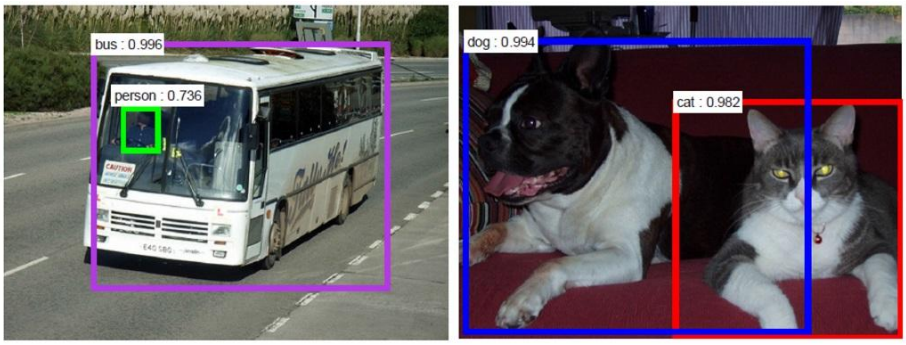


Figura 2.8: *Resultados de detección de objetos, algoritmo Faster RCNN*
Fuente: (Girshick, 2015)

El primero de los algoritmos en hacer su aparición fue RCNN. Han venido desarrollándose variantes de éste, pero en general, las deficiencias de la arquitectura utilizada por este algoritmo pueden ser resumidas en:

- Altos tiempos de entrenamiento de la red.
- El proceso de entrenamiento no se encuentra completamente optimizado.
- El análisis de la red se torna lento con imágenes que no sean de entrenamiento.

Sin embargo, varios trabajos de investigación relacionados han permitido solventar las deficiencias de RCNN mediante cambios tales como optimización del entrenamiento (Girshick, 2015). RCNN puede ser categorizado como el primero de una serie de investigaciones fructíferas en lo que concierne a detectores y/o clasificadores de aprendizaje profundo.

Las redes de aprendizaje profundo están clasificadas en dos grandes categorías: las de análisis por región y las de flujo directo.

El método de la ventana deslizante puede ser considerado como un procedimiento de fuerza bruta para detectar objetos en una imagen. Es necesario recorrerla de izquierda a derecha de arriba hacia abajo para ejecutar todas las tareas de procesamiento y clasificación. Es un enfoque efectivo pero costoso.

A. Algoritmos de Análisis por Región

Las RCNNs utilizan los métodos de proposición de regiones y búsqueda selectiva, cuyo funcionamiento se describe a continuación.

1. Se realiza un análisis de los píxeles de la imagen, se obtiene el valor de su textura.
2. Se procede a juntar grupos pequeños con valores congruentes de textura entre sí.
3. El proceso se repite hasta delimitar regiones bien diferenciadas.

La figura 2.9 ilustra el procedimiento previamente descrito, comparando el número de posibles regiones de interés existentes para cada uno de los pasos.

Figura 2.9: *Ilustración del funcionamiento de la búsqueda selectiva*
Fuente: (Girshick, Donahue, Darrell, & Malik, 2014)

Se lista a continuación los principales algoritmos por análisis de región:

1. RCNN (*Region Convolutional Neural Network*)

Algoritmo de detección de objetos basado en la generación de regiones de propuesta. Una búsqueda selectiva es ejecutada sobre cada una de las regiones, una a la vez haciendo uso de redes neuronales convolucionales (Girshick et al., 2014). La figura 2.10 detalla el esquema operativo de este algoritmo.

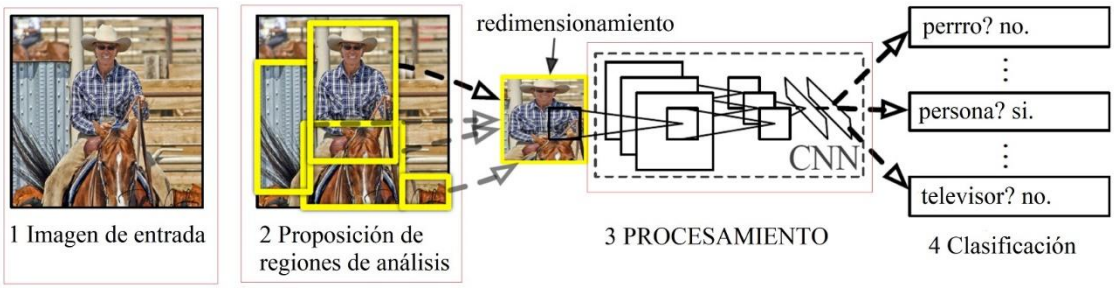


Figura 2.10: *Esquema Operativo de RCNN*
Fuente: (Girshick et al., 2014)

La etapa de proposición de regiones para análisis genera 2000 candidatos para su procesamiento individual. Debido a esto, sus tiempos de ejecución son altos, con 13s por cuadro mediante GPU y 53s por cuadro mediante CPU.

2. Fast RCNN

Consiste de una serie de mejoras aplicadas al RCNN original. Todas las regiones de propuesta son examinadas a la vez haciendo uso de una capa en malla para cada contenedor *roi* (Girshick, 2015). Este algoritmo presente las siguientes ventajas sobre su predecesor:

- Mayor precisión (valores más altos de mAP).
- El entrenamiento de la red se ejecuta en una única etapa.
- El entrenamiento actualiza la información de todas las capas de la red.
- No se requiere espacio en HDD para tareas de cache en procesamiento.

Fast RCNN aumenta considerablemente la velocidad de procesamiento debido a que ejecuta la convolución de la red generando SSPnets (*Spatial Pyramid Pooling Nteworks*, Redes de Agrupación de Espaciamiento Piramidal). Esto le permite reducir su tiempo de entrenamiento cerca de 9 veces contra CNN.

3. Mask RCNN

Sobre la base de FAST RCNN, esta versión incluye una rama paralela encargada de predecir máscaras para cada uno de los objetos a seguir. Al igual que su predecesor, ambos algoritmos son capaces de analizar aproximadamente 5 FPS al ejecutar su procesamiento sobre una GPU.

B. Algoritmos de Flujo Directo

Los detectores de flujo directo tienden a mejorar la velocidad de procesamiento, pero experimentan problemas detectando objetos muy pequeños o que se encuentren bastante cercanos entre sí. Esto se debe principalmente a los redimensionamientos experimentados por la imagen de entrada al pasar por cada una de las capas de la red.

Se lista a continuación los principales algoritmos de flujo directo:

1. YOLO (*You Only Look Once*)

Es un popular algoritmo de detección de objetos, cuyo aliciente principal consiste en su velocidad de ejecución y resultados congruentes a los obtenidos por otros métodos más precisos.

2. SSD (*Single Shot Multibox*)

Con una velocidad similar a la de YOLO, este algoritmo toma su nombre por la característica principal de funcionamiento: entregar en un solo paso la imagen de entrada a la red.

SSD añade varias capas de características al final de la base de la red. Estas capas están encargadas de predecir los valores por defecto para cada clase de objeto detectado. La gráfica compara a SSD con una secuencia de entrada de 300x300 (59 *fps*), con una precisión del 74.3%. En contraparte, YOLO con una entrada de 448x448 (45 *fps*) posee una precisión de 63.4%. Los porcentajes de precisión se basan en la media de predicciones registrada. Esta comparativa es ilustrada en detalle en la figura 2.11.

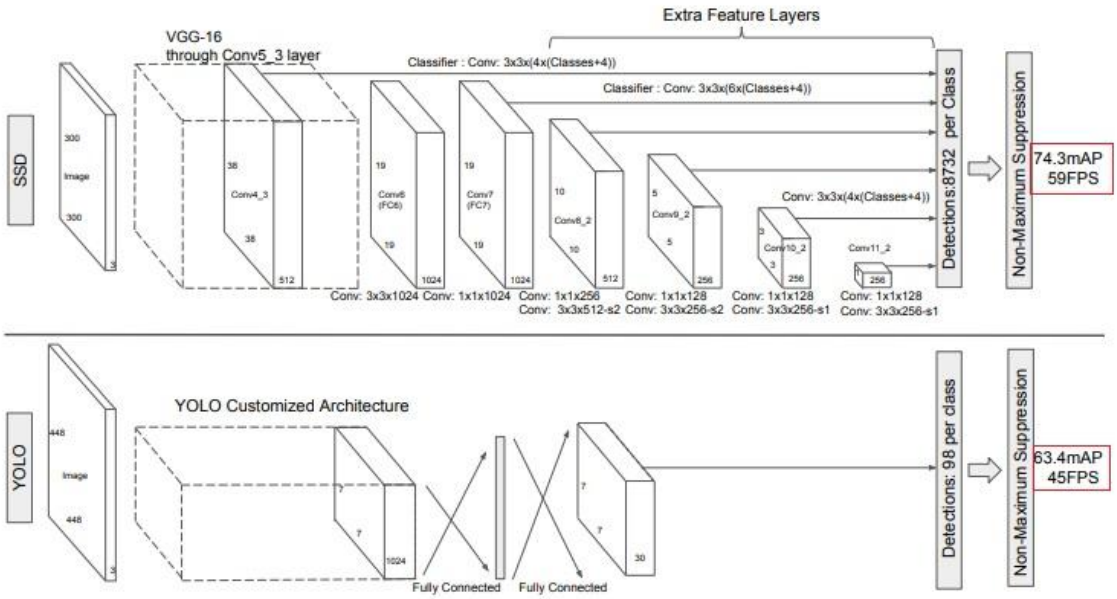


Figura 2.11: *Comparativa entre SSD y YOLO*
Fuente: (Liu et al., n.d.)

El modelo de arquitectura utilizado es el SSD 512, tal y como lo han denominado sus autores. Esta arquitectura supera en rendimiento a Faster RCNN con los juegos de datos COCO y PASCAL VOC. De la misma manera, es 3 veces más rápido en sus tiempos de ejecución (Liu et al., n.d.).

2.8 Teoría de Flujo Vehicular

Se define a tránsito vehicular como la consecuencia directa en flujo de vehículos en una vía, calle, camino o autopista (Mozo Sánchez, 2012). Previo a cualquier diseño y/o construcción de una vía, es imprescindible conocer las características del tránsito vehicular que ocuparan a la misma.

Realizar un análisis de los elementos del flujo vehicular permite comprender las características básicas y el comportamiento del tránsito. Análogamente permite conocer los requisitos para la proyección y operación de calles, carreteras y cualquier tipo de obra complementaria asociada. La aplicación de las formulas de la teoría del flujo vehicular permite determinar los niveles de eficiencia de la operación de una red vial.

Existen tres conceptos fundamentales que pueden ser definidos matemáticamente:

- Velocidad

- Volumen / Intensidad de Tránsito
- Densidad

2.8.1 Velocidad

Corresponde a una relación entre la distancia recorrida por unidad de tiempo. Generalmente suele ser representada en las unidades Km/H.

A. Velocidad Promedio de Viaje

Medida de transito basada en la observación del tiempo de viaje a través de una longitud especificada de carretera. La ecuación, detalle el cálculo de la velocidad promedio de viaje para un vehículo determinado.

$$S = \frac{L}{t_a}$$

En donde:

$S: Velocidad\ Promedio\ de\ Viaje\ [Km/H]$

$L = Longitud\ del\ segemento\ de\ carretera\ [Km]$

$t_a: Tiempo\ promedio\ de\ viaje, paradas\ incluidas\ [H]$

2.8.2 Volumen de Tránsito

Se define como el número de vehículos que pasan a través de determinado punto de interés de una autovía durante un intervalo de tiempo especificado.

Los intervalo de tiempo más comunes consisten en días (vehículos por días), siendo este tipo de volumen los más utilizados para la base de planificación de proyectos viales (Mozo Sánchez, 2012).

2.8.3 Densidad

Número de vehículos que ocupan un determinado tramo de autovía. Se representa mediante las unidades vehículos/Km.

$$D = \frac{v}{S}$$

En donde:

$v = Razon\ de\ flujo\ [vehiculos/Km]$

$S = Velociad\ promedio\ de\ viaje\ [Km/H]$

$D = Densidad\ [vehiculos/Km/carril]$

La densidad es uno de los parámetros más relevantes en lo que concierne al tránsito, ya que es una medida que se encuentra directamente relacionada con la demanda.

2.8.4 Tasa de Flujo

La tasa de flujo es la frecuencia con la cual circulan los vehículos por un punto de un carril o autovía. Esta medida se puede ser expresada en vehículos/hora, manteniendo el concepto de su interpretación. La tasa de flujo se refiere al número de vehículos que han circulado por el punto de interés (paradas incluidas), a diferencia del volumen horario ***Q*** que determina la cantidad de vehículos que han circulado limpiamente por el carril. El cálculo de a tasa de flujo se realiza mediante:

$$q = \frac{N}{T}$$

En donde:

$N = Numero\ de\ vehiculos\ por\ intervalo\ de\ tiempo$

$T = Intervalo\ [vehiculos/min], [vehiculos/s]$

$q = Tasa\ de\ Flujo$

2.9 Hipótesis

La exactitud de los resultados del cálculo de flujo de tráfico de un sistema basado en visión artificial es congruente con respecto a los obtenidos mediante una metodología dedicada no automatizada.

2.10 Señalamiento de variables de la hipótesis

2.10.1 Variable independiente

Visión artificial.

2.10.2 Variable dependiente

Análisis de Tráfico Vehicular.

CAPÍTULO III

METODOLOGÍA

3.1 Enfoque

3.2 Modalidad básica de la investigación

La presente investigación será aplicada. Esto debido a que se utilizaran conocimientos en el área de visión artificial para la creación de un sistema que permita realizar análisis de tráfico vehicular.

La investigación será bibliográfica porque utilizará fuentes como libros, documentos, artículos, revistas, entre otros. De esta manera, se elaborará el marco teórico, así como el diseño del software para el análisis de tendencias del tráfico.

La investigación además, poseerá modalidad de campo. Se obtendrán secuencias de video del tráfico de las zonas seleccionadas para el estudio. Dichas secuencias de imágenes serán utilizadas para la validación y optimización de software.

3.3 Nivel o tipo de investigación

La investigación tendrá los siguientes niveles: exploratoria, descriptiva y correlacional.

La investigación será exploratoria debido a que se buscarán elementos relacionados a la problemática planteada en el primer capítulo de este perfil y así, poder encontrar los procedimientos adecuados para elaborar una investigación con una propuesta de análisis.

La investigación será descriptiva debido a que se detallará la manera de realizar un análisis de tráfico, obteniendo los siguientes datos: cantidad de vehículos, velocidades de desplazamiento y rutas descritas.

El diseño de esta investigación será experimental. Se pretende desarrollar un algoritmo de alta precisión y elevada repetividad que permita determinar eficientemente las tendencias del tráfico en las zonas de estudio seleccionadas. A través de pruebas y métodos de ensayo, se modelará un sistema que cumpla con los objetivos planteados en el presente trabajo de investigación.

3.4 Población y muestra

La población del estudio del presente proyecto de investigación se encuentra conformada por la totalidad del parque automotor de la ciudad de Ambato, provincia de Tungurahua. En el año 2016 fueron matriculados 86469 nuevos vehículos dentro de la provincia (INEC, 2016). Sin embargo, el proyecto se enfoca al análisis del tráfico en una de las zonas y/o intersecciones de mayor confluencia de vehículos dentro de la urbe.

La zona de estudio determinada para la captura de las secuencias de video es el viaducto de la mascota, ubicado en el inicio de la Avda. Oriental, tal y como se puede apreciar en la figura 3.1.

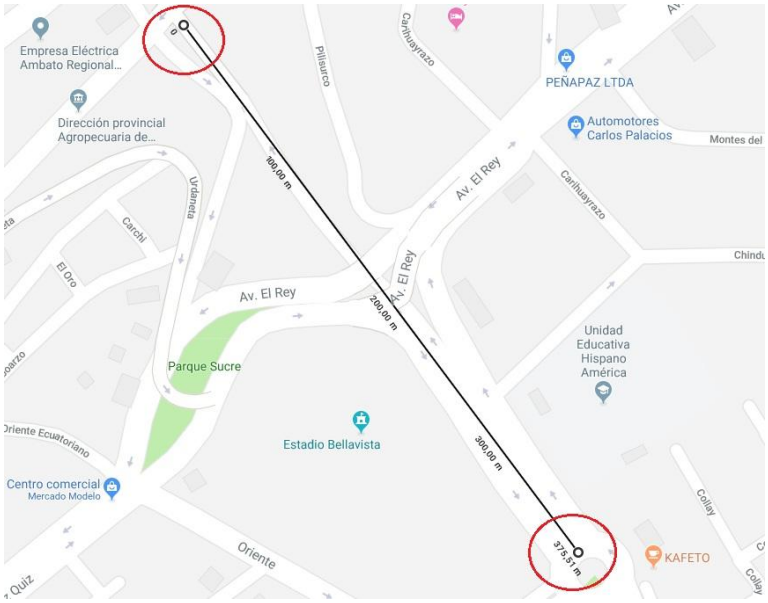


Figura 3.1: *Zona de estudio seleccionada*
Fuente: *Google Maps*

Se ha seleccionado esta autovía por ser una de las rutas principales de acceso al centro de la ciudad, y poseer un tráfico elevado. Posee un largo aproximado de 380m, dos carriles de y un único sentido de circulación.

3.5 Operacionalización de variables

Tabla 3.1
Variable Independiente: Visión Artificial

Conceptualización	Dimensiones	Indicadores	Ítems Básicos	Técnicas e Instrumentos
Comprende a un conjunto de técnicas y procesos para la adquisición, procesamiento y análisis de imágenes del mundo real mediante un computador.	Pre procesamiento. Extracción de características.	Redimensionamiento. Reducción de ruido. Ecualización de histograma. Extracción de características.	¿Cuáles operaciones de pre procesamiento son necesarias previo al análisis de un fotograma?	Investigación – Medición.
			¿Cuáles son los métodos de extracción de características de un archivo de imagen?	Investigación – Medición.
			¿Cuál es la resolución adecuada para el procesamiento de los fotogramas capturados?	Investigación – Medición.
	Entornos de desarrollo	Licenciados. Software Libre.	¿Cuáles entornos de desarrollo permiten implementar aplicaciones de visión artificial?	Investigación – Medición.
			¿Resulta factible la utilización de software libre o licenciado?	Investigación – Medición.

Tabla 3.2

Variable Dependiente: Análisis de Tráfico Vehicular

Conceptualización	**Dimensiones**	**Indicadores**	**Ítems Básicos**	**Técnicas e Instrumentos**
Consiste en la obtención de datos tales como: cantidad de vehículos, velocidades y rutas descritas, con el fin de mejorar la movilidad de zonas urbanas con alto flujo vehicular.	Modalidades	En línea. Fuera de línea.	¿Cuáles son los escenarios de aplicación de un análisis de tráfico en línea? ¿Cuáles son los escenarios de aplicación de un análisis de tráfico fuera de línea? ¿Cuál modalidad de trabajo presenta mayor repetividad de resultados para el análisis de tráfico?	Investigación – Medición. Investigación – Medición. Investigación – Medición.
	Procesamiento	Algoritmos varios. Obtención de datos.	¿Cuáles métodos pueden ser aplicados para obtener rutas y velocidades de un vehículo en una secuencia de video? ¿Cuáles algoritmos de detección de objetos poseen alta estabilidad en muestreos largos? ¿Cuáles algoritmos de detección de objetos responden mejor a oclusión y/o desenfoques de los objetos de interés?	Investigación – Medición. Investigación – Medición. Investigación – Medición.

3.6 Recolección de información

El presenta trabajo de investigación se sustenta en la actualización de trabajos similares previamente desarrollados en el campo de visión artificial. Toda la información recolectada de repositorios especializados será revisas y depurada. Además se creará un repositorio de pruebas con secuencias de video capturadas en las zonas seleccionadas para el estudio.

3.7 Procesamiento y análisis

- Recolección de secuencias de video con imágenes de tráfico vehicular para la creación de la base de pruebas.
- Estudio de algoritmos no basados en detección para realizar rastreo de objetos.
- Estudio de algoritmos basados en detección para realizar rastreo de objetos.
- Pruebas de precisión y repetitividad con secuencias de video de tráfico vehicular de la base.
- Pruebas de diagnóstico con secuencias de video de tráfico obtenidas del área de pruebas designada.
- Optimización del algoritmo.
- Diseño de la interfaz gráfica del aplicativo.
- Elaboración del informe final.

CAPÍTULO IV

ANÁLISIS E INTERPRETACIÓN DE RESULTADOS

4.1 Algoritmos de Seguimiento de Objetos

Pregunta directriz: ¿Cuáles algoritmos de visión pueden ser aplicados para realizar el seguimiento de un objeto?

El seguimiento de vehículos se define como la identificación de los movimientos físicos de los mismos para predecir su ubicación dentro de una escena dinámica (Choudhury, Hazra, & Chattopadhyay, 2017).

4.1.1 Algoritmos de Seguimiento

Tal y como se ha detallado en la sección **2.6.2**, existen varios algoritmos para realizar seguimiento (*tracking*) de objetos por medio de visión artificial.

- MEDIANFLOW
- KCF
- MIL
- MOSSE
- GOTURN

La utilización de cada uno de los mismos se encuentra condicionada mayormente por las características asociadas al objeto a seguir.

4.1.2 Algoritmos de Detección

Existen varias alternativas disponibles para realizar conteo de objetos mediante visón artificial.

- Mediante sustracción de fondo.
- Utilización de detectores de aprendizaje profundo:
    - RCNN
    - Fast RCNN
    - Mask RCNN
    - YOLO
    - SSD

4.2 Tolerancias en Seguimiento de Objetos Mediante Visión Artificial

Pregunta directriz: ¿Cuál es la tolerancia aceptable de un algoritmo de visión específico al momento de realizar el seguimiento de un objeto?

La efectividad de un algoritmo de seguimiento y/o conteo de objetos, depende de diversos factores. Cambios de iluminación, de clima, desenfoques, oclusiones e incluso la introducción de objetos indeseados en ambientes no controlados, influyen directamente en el rendimiento de los mismos (Subaweh & Wibowo, 2016).

Tomando en cuenta las consideraciones previas, es necesario realizar pruebas de repetividad para garantizar la consistencia de los resultados. Por lo general, discrepancias de entre un [5-7] % pueden ser consideradas como aceptables para denominar como estable a un sistema que haga uso de estos algoritmos (Choudhury et al., 2017).

4.3 Calibraciones previas de Sistemas de Seguimiento de Objetos

Pregunta directriz: ¿Cuáles métodos y/o calibraciones pueden ser aplicados para conseguir un óptimo encuadre del objeto a seguir?

Dependiendo de la metodología utilizada, los métodos de calibración para seguimiento y/o conteo de objetos puede generalizarse en 2 grandes categorías.

4.3.1 Metodología de Sustracción de Fondo

Los filtros de Kalman son herramientas ampliamente utilizadas por desarrolladores para la implementación de seguimiento de sistemas lineales dinámicos expuestos a ruido de tipo Gaussiano, el más común efectos indeseados en procesamiento de imágenes (Suryatali & Dharmadikari, 2015). Dentro de las características principales de estos filtros, se pueden nombrar:

- Alta eficiencia a nivel computacional
- Robustez y precisión
- Recursividad

A fines prácticos, los filtros de Kalman pueden ser utilizados para implementar algoritmos de sustracción de fondo, mediante la aplicación de un filtro unidimensional a cada uno de los pixeles de la imagen en cuestión. De esta manera se logra extraer y mantener la referencia del fono de la imagen seleccionada para el análisis.

4.3.2 Redes Neuronales

El grado de efectividad asociado a la utilización de redes neuronales – algoritmos de aprendizaje profundo – para realizar seguimiento o conteo de objetos, depende significativamente de las condiciones de entrenamiento de la red.

La utilización de estos recursos generalmente supone un mayor grado de tolerancia frente a condiciones operativas no deseadas inherentes al entorno de trabajo. Sin embargo, el tiempo de procesamiento requerido para cada imagen de entrada es mayor. Ciertos trabajos de investigación han logrado desarrollar sistemas de conteo de vehículos unificando los conceptos de detección (redes neuronales) y seguimiento (algoritmos varios) (Li et al., 2014). Pese a que se ha logrado trabajar en tiempo real, el tiempo de ejecución de la detección sigue siendo alto, condicionando el alcance de los sistemas propuestos.

4.4 Entorno de Desarrollo para Visión Artificial

Pregunta directriz: ¿Cuáles entornos de desarrollo enfocados a visión artificial y tratamiento de imágenes se encuentran actualmente disponibles?

Se procede a realizar una breve revisión a algunos de los principales entornos de desarrollo disponibles para visión artificial.

4.4.1 LabView

El software de Visión Artificial de LabView es en producto licenciado, ofrecido por su desarrollador National Instruments. LabView consiste en un entorno de programación gráfico orientado a la adquisición de datos en tiempo real. El paquete de Visión Artificial incluye cientos de algoritmos escritos en C++, Visual Basic y .NET.

Un detalle particular de utilizar este IDE, es su alto costo computacional al mantener activos varios snippets o instrumentos virtuales, debido a la cantidad de controladores involucrados en cada uno de los procesos en ejecución.

4.4.2 Matlab

Constituye una herramienta de software matemático con un IDE basado en su propio lenguaje nativo, conocido como Lenguaje M. A más de ofrecer una extensa gama de características y funciones de procesamiento matemático (con Simulink y GUIDE como sus dos herramientas de expansión principales), Matlab ofrece la posibilidad de diseñar y simular sistemas de visión artificial.

Es un software licenciado, en el cual hasta hace no mucho los usuarios estaban sujetos y bloqueados al vendedor. Sin embargo se ha ofrecido la posibilidad de utilizar MATLAB Builder para utilizar funciones propias del software operando en un entorno .NET o Java. Es necesario además que el equipo en cuestión posea el entorno de ejecución MCR (*Matlab Component Runtime*).

4.4.3 OpenCV

OpenCV es el referente en librerías de código abierto para visión artificial, procesamiento de imágenes y redes neuronales. Recientemente ha incluido la funcionalidad de procesamiento por GPU para mejorar el rendimiento de aplicaciones de alto coste computacional. El desarrollador original del proyecto es Intel.

Es distribuido bajo licencia BSD, por lo cual es gratuito tanto como para fines personales como comerciales. Posee las siguientes interfaces: C, C++, Python y Java. Es compatible con Windows, Linux, OS X y Android.

El proyecto fue originalmente desarrollado para optimizar procesamiento, enfocado en aplicaciones de tiempo real. Adicionalmente, cuando se utilizan los entornos de C y C++ se puede hacer uso de procesamiento paralelo.

Se detalla a continuación algunas de las funcionalidades principales de OpenCV:

- Procesamiento, lectura y creación de imágenes y video.
- Detección de objetos y Características.
- Visión artificial estéreo o basada en Geometría Monocular.
- Fotografía.
- Aprendizaje profundo (redes neuronales) y *clustering*.
- Aceleración por GPU.

4.4.4 Simple CV

Es un entorno de desarrollo escrito en Python, que permite desarrollar aplicaciones de visón artificial. Simple CV permite utilizar librerías y funciones de entornos de desarrollo más avanzados (como OpenCV) y presentarlos al usuario en forma de módulos o bloques, que facilitan el desarrollo de aplicaciones de visión.

Al igual que OpenCV, su licencia en BSD y se encuentra disponible para Windows, OS X y Linux (Ubuntu).

4.4.5 VXL

VXL (*Vision Something Libraries*, Compendio de Librerías de Visión), es una colección de recursos escritos en C++ para visión artificial. Se detalla algunas de las funcionalidades principales del entorno:

- Procesamiento de imágenes y video.
- Manipulación de video.
- Geometría estéreo.
- Clasificación, detección y seguimiento de características.

Las librerías principales están diseñadas para ser ligeras e independientes, a diferencia de otros entornos como OpenCV por ejemplo, que requieren de pre compilación y enlace de sus todas sus librerías con el sistema para operar.

4.5 Licenciamiento

Pregunta directriz: ¿Resulta factible la utilización de software libre o de pago?

Esta resulta en una de las mayores condicionantes de para la implementación de proyectos de investigación. Tanto software licenciado, como de licencia libre, presentan una serie de ventajas y desventajas que deben ser analizadas a conciencia para llevar a cabo la elección de la plataforma a utilizar.

Entre las principales ventajas de utilizar software licenciado, es que nos encontramos con soluciones y paquetes especializados desarrollados por los fabricantes. Además se dispone de soporte y acceso preferencial a foros de debate/preguntas/desarrollo. Por su contraparte, los costos (altos en ciertas ocasiones) suponen una inversión económica adicional. Además, el software licenciado tiende a restringir compatibilidad sus productos, a más de en muchas ocasiones, requerir de módulos adicionales e incluso el propio software, para que la aplicación desarrollada se ejecute en el equipo del cliente.

El software libre elimina los costos de licenciamiento, haciendo disponible a la comunidad de desarrollo el acceso a sus programas. Esto se logra (no siempre) mediante la simplificación de entornos y núcleos operativos, reduciendo funcionalidades o aspectos de GUI (*Graphic User Interface*, Interfaz Gráfica de Usuario), comparados con software licenciado. Otra desventaja representa la aparición de bugs o en ciertas ocasiones, comportamientos variantes de los programas en distintos sistemas operativos. Sin embargo, la comunidad del software libre cuenta con muchos foros y colaboradores activos para la solución y revisión de bugs o interrogantes de desarrollo en lo que respecta a la implementación de aplicaciones especializadas.

Se considera para el presente proyecto de investigación los siguientes entornos de desarrollo:

- Matlab
- OpenCV

Se selecciona OpenCV en base a un estudio personal realizado tomado en cuenta tres parámetros principales:

- Costo: el costo de una licencia para Matlab con las herramientas necesarias (toolboxes) necesarios para visión artificial, bordea los USD 7000. OpenCV en contraparte, es gratuito.
- Curva de Aprendizaje: tal como se describió en la sección **4.4.2**, Matlab posee su propio lenguaje, de forma existe una forma única de programar de forma optimizada. Esta forma difiere de métodos de programación generalizados y extendidos en otros lenguajes como C++ y Python. Codificar en Matlab utilizando el esquema de trabajo de otros lenguajes, deriva directamente en un aumento de los tiempos de ejecución de los programas. Por otra parte, tener conocimientos en C y C++, permite modificar incluso las librerías de OpenCV.
- Tiempos de ejecución: las aplicaciones desarrolladas en Matlab corren mucho más lento en comparación a aquellas escritas en C++. Resulta común que se escriban códigos para tareas largas y complejas en C++, para integrarlas a Matlab utilizando *mex*. Las librerías de OpenCV, en cambio, están pensadas para ejecutar un alto número de tareas con un bajo coste computacional. Existen además otras opciones como optimizar el código mediante funciones específicas o utilizar procesamiento en paralelo para elevar la velocidad de procesamiento de la aplicación.

Por los justificantes previamente detallados, se selecciona OpenCV como el entorno de desarrollo idóneo para la implementación del presente trabajo de investigación.

CAPÍTULO V

CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

- Los algoritmos de seguimiento de objetos más adecuados para realizar seguimiento de vehículos son MIL y KCF. MIL ofrece robustez, precisión y cierto grado de tolerancia frente a posibles oclusiones, siendo apto para condiciones de tráfico lento – moderado. KCF por su parte, ofrece la mayor velocidad de seguimiento, siendo la alternativa idónea para seguir a vehículos en condiciones de tráfico rápido.
- La utilización de algoritmos de aprendizaje profundo constituye una sólida opción para la implementación de sistemas de conteo de objetos. Salvando ciertos inconvenientes natos asociados a sus costes computacionales, estos algoritmos ofrecen un desempeño superior a métodos basados en operaciones con imágenes.
- OpenCV constituye el entorno de desarrollo idóneo para la implementación de aplicaciones basadas en visión artificial. Su orientación a aplicaciones en tiempo real y librerías pre optimizadas en rendimiento garantizan robustez, velocidad de procesamiento y repetividad de resultados obtenidos.

5.2 Recomendaciones

- Se recomienda analizar las características inherentes asociadas a algoritmos especializados de visión artificial. Dicha revisión debe ser llevada a cabo de forma bibliográfica y empírica para corroborar el rendimiento de los mismos. De

esta manera se garantizará la selección de los componentes adecuados para la implementación del sistema.

- Se recomienda la utilización de algoritmos de aprendizaje profundo para tareas de detección y/o conteo de objetos. Su rendimiento y capacidades los tornan en objeto de estudio e interés por parte de la comunidad científica, así como su integración garantiza proyectos robustos y eficientes.
- Se recomienda la utilización del entorno de desarrollo OpenCV para la implementación de proyectos de visión artificial. Esto debido a que trabaja con dos lenguajes ampliamente extendidos: C++ y Python. El conocimiento previo de uno o ambos lenguajes permite una rápida comprensión y adhesión al desarrollo de aplicaciones, así también como la modificación de las librerías acorde a las necesidades del desarrollador. Adicionalmente, debido a la naturaleza de su licenciamiento, existe una amplia comunidad de foros de soporte y documentación disponible.

CAPÍTULO VI

LA PROPUESTA

6.1 Tema de la Propuesta

DESARROLLO DE UN PROGRAMA BASADO EN OPENCV PARA EL ANÁLISIS DE TRÁFICO VEHICULAR EN MODALIDAD FUERA DE LÍNEA

6.2 Datos Informativos

***Ejecutada por*:** Ing. Jovann Pérez Nasser, tesista de la Maestría en Automatización y Sistemas de Control de la Universidad Técnica de Ambato.

***Beneficiarios*:**

***Ubicación*:** Facultad de Ingeniería en Sistemas, Electrónica e Industrial de la Universidad Técnica de Ambato.

***Responsables*:**

- Ing. Jovann Pérez (Investigador).
- Ing. Darío Mendoza (Tutor).

Tiempo de ejecución:

Financiamiento: mediante recursos propios del investigador.

6.3 Antecedentes a la Propuesta

Se han encontrado los siguientes trabajos relacionados:

En el trabajo de investigación denominado “Conteo de Vehículos para Administración del Tráfico utilizando YOLO y Filtros Correlacionales” (Asha & Narasimhadhan, 2018) se propone un esquema híbrido para realizar conteo de vehículos. La parte de detección es realizada por la red neuronal de flujo directo, mientras que el seguimiento de los vehículos es llevado a cabo mediante filtros correlacionales. De esta manera se logra reducir el coste computacional y lograr que el aplicativo trabaje en tiempo real. Una de las limitaciones consiste en el análisis de un solo carril, ya que ejecutarlo en ambos elevaría el tiempo requerido por el detector.

En el trabajo de investigación denominado “Detección de Vehículos en Tiempo Real, Seguimiento y Conteo utilizando OpenCV” (Li et al., 2014) se presenta un sistema para análisis de tráfico que no hace uso de algoritmos de aprendizaje profundo. La metodología utilizada consiste en la separación del vehículo con respecto al fondo de imagen combinando las técnicas de umbral de Otsu y separación de la sombra proyectada por el mismo.

En el trabajo de investigación denominado “Detector de Objetos basado en YOLO para Escenas de Tráfico” (Tao, Wang, Zhang, Li, & Yang, 2017) se mejoran los tiempos de ejecución de YOLO para escenas específicas. Dichas escenas están optimizadas para imágenes que contengan personas y vehículos, elementos comunes en capturas de tráfico. Esto se logra mediante una etapa de pre procesamiento basada en la ecualización del histograma de la imagen. Los resultados de la investigación muestran un incremento de velocidad de 1.18 veces la velocidad de procesamiento de YOLO para las clases: persona, auto, motocicleta, camión y autobús.

6.4 Justificación

La inteligencia artificial posee 3 ramas de estudio de alta prioridad: procesamiento de lenguaje, habla y visión artificial. La creciente necesidad de la evolución asociada a la automatización de procesos, requiere de soluciones cada vez más efectivas, innovadoras y con capacidades de replicación. La visión artificial posee particularmente la capacidad de ser aplicada dentro de una amplia gama de procesos, encontrando su máxima utilidad en tareas de detección, identificación y seguimiento de objetos.

La aplicación de soluciones basadas en esta tecnología al análisis del tráfico propone herramientas de procesamiento altamente flexibles y con un grado de intrusión

prácticamente nulo con la naturaleza del proceso. Los procedimientos de ingeniería inversa y la aplicación de algoritmos especializados permiten desarrollar sistemas para el estudio de un problema estrechamente relacionado con el desarrollo de las urbes.

La obtención de valores tales como el flujo de vehículos que circulan por una autovía es un requisito indispensable para la planeación o readecuación de obras de vialidad. Es así que se denota la importancia de sistemas basados en visión artificial que permitan realizar este tipo de análisis.

6.5 Objetivos

6.5.1 Objetivo General

- Implementar una programa basada en OpenCV - C++ para el análisis de tráfico vehicular en modalidad fuera de línea.

6.5.2 Objetivos Específicos

- Desarrollar un algoritmo de seguimiento de vehículos que permita determinar la trayectoria descrita, tiempo de seguimiento y velocidad promedio de los mismos.
- Desarrollar un algoritmo de conteo de vehículos que permita determinar el flujo de distintas clases de automotores capturados en los fotogramas de estudio.
- Establecer procedimientos de validación de operatividad y repetividad de los algoritmos desarrollados.

6.6 Análisis de Factibilidad

La presente propuesta posee factibilidad de ejecución, debido a la existencia y disposición de los recursos técnicos, operativos y económicos para su desarrollo.

6.6.1 Factibilidad Técnica

El investigador dispone de los conocimientos tanto técnicos como científicos para la implementación de la propuesta. Existe además documentación relacionada para la solución de cualquier incidencia durante el desarrollo de la misma.

6.6.2 Factibilidad Operativa

Se dispone del entorno de desarrollo necesario para la implementación de la propuesta, así como la evaluación de sus resultados.

6.6.3 Factibilidad Económica

La propuesta supone un bajo coste para su implementación. La misma será autofinanciada por el investigador.

6.7 Fundamentación Científico – Técnica

6.7.1 Clase MultiTracker de OPENCV

En lo concerniente a las técnicas empleadas por Visión Artificial, el seguimiento de objetos presenta las siguientes ventajas frente a la detección:

- Permite establecer y preservar una *identidad* de los objetos seleccionados.
- El seguimiento de objetos funciona en base a una predicción de la ubicación del objeto a seguir, bajo los parámetros registrados en el cuadro anterior de la imagen. Esto representa una ventaja frente a ciertos casos en los cuales, la detección de objetos falla.
- Algunos algoritmos de seguimiento poseen tiempos de ejecución reducidos, debido a que realizan búsquedas y predicciones locales del objeto a seguir, en lugar de la utilización de ventanas deslizantes o búsquedas globales implementadas por ciertos algoritmos de detección.

Sin embargo, existen escenarios en los cuales el seguimiento de objetos puede fallar, siendo éstos secuencias largas de video. Después de cierto número de ejecuciones asociado a la robustez intrínseca del algoritmo, las predicciones pueden volverse ambiguas. Esto desencadena en una alta probabilidad de desenfoque o pérdida del objeto a seguir. En aplicaciones que operan bajo este tipo de condiciones se combinan las técnicas de seguimiento y detección para contrarrestar estos efectos.

Se ha comprobado experimentalmente que la utilización del seguimiento de objetos por sí solo entrega resultados altamente satisfactorios para el escenario operativo del presente proyecto de investigación.

La clase MultiTracker de OpenCV provee un método de vectorización de múltiples trackers individuales para el seguimiento varios objetos. Esta clase se encuentra disponible desde la versión 3.3.1 (The OpenCV Foundation, 2018).

La utilización de esta clase puede ser resumida en los siguientes pasos principales:

1. Creación de un Tracker Simple: OpenCV en su versión 3.4 dispone de los siguientes algoritmos de seguimiento: BOOSTING, MIL, KCF, TLD, MEDIANFLOW, GOTURN, MOSSE, CSRT.
2. Lectura del primer cuadro de un video: la clase MultiTracker precisa de dos elementos principales para realizar un seguimiento de objetos múltiples:
    - Un cuadro de imagen.
    - La ubicación de los objetos a seguir.
3. Localización de los objetos seguir en el cuadro inicial: una vez que se ha leído el cuadro inicial de la secuencia de video a procesar, es necesario indicar las posiciones de los objetos a seguir. Esto se logra con la delimitación de *rois* mediante *bounding boxes.*
4. Creación del objeto Multitracker: se procede a crear tantos miembros de la clase MultiTracker como *bounding boxes* se hayan especificado en el paso anterior. Los miembros pueden pertenecer al mismo algoritmo (recomendado), pero también se pueden englobar diferentes algoritmos.
5. Actualización del estado del Objeto Multitracker: una vez que se ha inicializado la clase y sus respectivos miembros, es necesario entregar al Multitracker un cuadro de imagen nuevo para que empiece a seguir los objetos detallados en su parametrización inicial.
6. Presentación de resultados.

La clase MultiTracker consolida un contenedor para los Trackers individuales pertenecientes a cada objeto.

6.7.2 Algoritmo YOLO (You Only Look Once)

YOLO consiste en algoritmo de aprendizaje profundo, que inicialmente se encontraba disponible en DARKNET. DARKNET puede ser denominado como un entorno de desarrollo basado en C para la creación y utilización de redes neuronales.

YOLO es un algoritmo de detección de objetos, que unifica los conceptos de localización y reconocimiento. Anteriormente, se utilizaba el método de la ventana deslizante para localizar objetos en distintas ubicaciones y/o escalas. El problema principal inherente a este método era su alto costo computacional. Ciertos algoritmos de aprendizaje profundo, tales como RCNN (*Regional Convolution Neural Network*, Convolución por Regiones de Redes Neuronales) y Fast RCNN (Versión mejorada de RCNN), utilizan un método denominado *búsqueda selectiva,* el cual disminuye considerablemente los tiempos de cómputo frente a la ventana deslizante.

YOLO utiliza un enfoque completamente diferente a los detallados con anterioridad. Ingresa la imagen completa dentro de una red neuronal entrenada previamente. Posteriormente es dividida en una malla de 13x13, donde cada una de estas regiones individuales está encargada de predecir cierto número de *bounding boxes*. Para cada uno de los contenedores que enmarcan los potenciales objetos a ser detectados, la red predice el nivel de confiabilidad de la detección (mediante la utilización de regresión logística) (Redmon & Farhadi, 2017) y su pertenencia a una de las clases establecidas por el algoritmo. Con los niveles de confiabilidad establecidos y asociados a cada uno de los contenedores, se eliminan aquellos con un nivel bajo o a su vez aquellos que encierren al mismo objeto que otro contenedor con un nivel mayor. Esta técnica se denomina Supresión de No Máximos.

El entorno original de YOLO es Darknet, sin embargo, se procede a utilizar la implementación en OpenCV, debido a las siguientes ventajas:

- Integración inmediata: para utilizar YOLO con OpenCV no es necesario compilar ni construir módulos adicionales, a diferencia de lo que ocurre con Darknet.
- Mayor velocidad: el coste computacional de YOLO con OpenCV es mucho menor. Cada imagen puede ser procesada en alrededor de 220ms frente a los 3.2sg que toma en Darknet.
- Calibración sin re entrenamiento: YOLO puede ser ajustado en tres diferentes modos de trabajo: mayor velocidad, balanceado y mayor precisión, sin que sea necesario re entrenar a la red. Esto se logra modificando el tamaño de la imagen de entrada, a una de las tres dimensiones disponibles.

La figura 6.1 ilustra los tiempos de ejecución de YOLO en su versión 3 frente a otros algoritmos de detección similares.

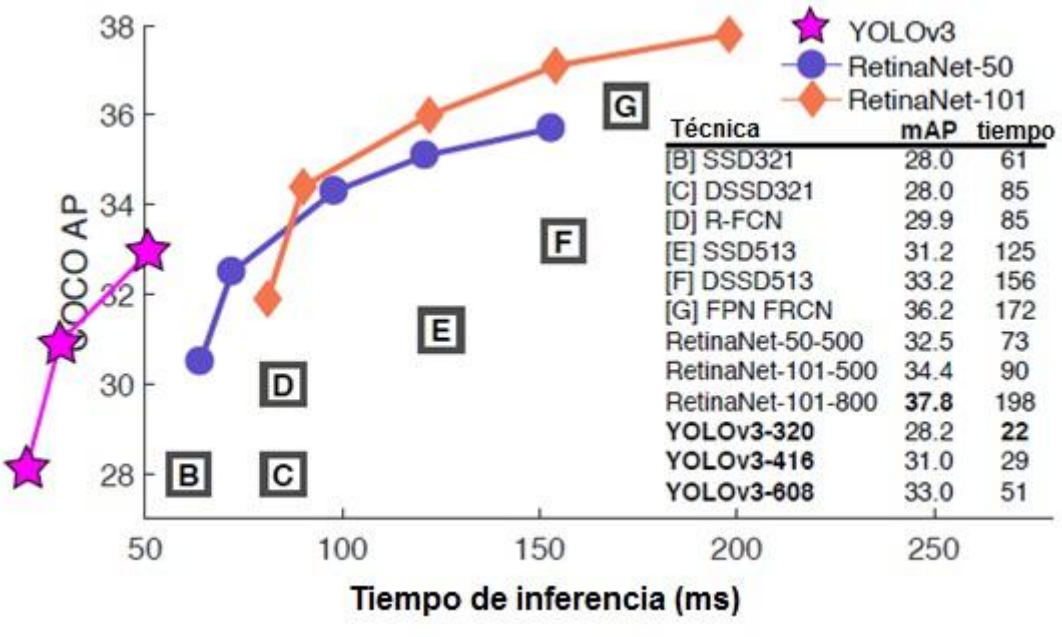


Figura 6.1: *Tiempos de ejecución y mAP de YOLOv3 con el juego de datos COCO*
Fuente: (Redmon & Farhadi, 2017)

La gráfica previa ilustra los tiempos de interferencia de algunos algoritmos de detección correspondientes al estado del arte, incluido YOLO. Se aprecia claramente que sus intervalos de ejecución son significativamente menores. Este algoritmo constituye una de las alternativas más sólidas y rápidas en detección de objetos (Tao et al., 2017), debido a la alta optimización de su algoritmo, así como su precisión y repetividad frente a alternativas similares.

La tendencia actual trabajos de investigación en este campo miden los valores de mAP (*mean average precision*, media de precisión) con sets de datos de COCO (*Common Object Context*, Objetos Comunes en Contexto). COCO comprende una colección de datos comunes para ser utilizados por algoritmos de detección y segmentación. Este juego de datos es ofrecido por Microsoft.

YOLO hace uso de 53 capas convolucionales en su red. Cada una de las capas es entrenada con imágenes de 256x256 y ejecuta sus operaciones con una menor cantidad de cálculos con valores flotantes que otros detectores similares.

6.8 Metodología

Se propone de desarrollar el software para el análisis de tráfico con las siguientes características, tal y como se detalla en la figura 6.2.

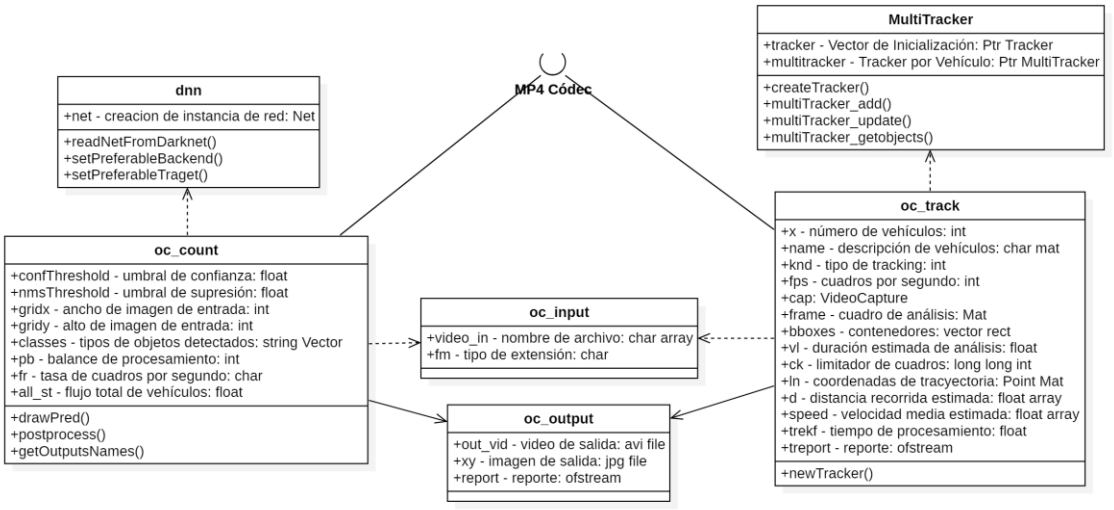


Figura 6.2: *Metodología de la Propuesta*

- Seguimiento de Vehículos (**oc_track**): basado en la implementación de la clase **MultiTracker**. Se emplean 2 algoritmos para análisis variantes en velocidad y precisión. Dichos algoritmos son MIL (mayor precisión) y KCF (mayor velocidad). La selección se ha basado en al análisis de características detallado en la sección **2.6.2** a más de pruebas operativas. El proceso depende a su vez de una interfaz de entrada de códec mp4 y de la clase **oc_input** para la carga de la secuencia. Los resultados del análisis son procesados con los miembros de la clase **oc_output**. Funcionalidades:
    - Trazado de ruta.
    - Cálculo de distancia estimada recorrida.
    - Calculo del tiempo de viaje.
    - Calculo de la velocidad promedio estimada.
    - Reportes: incluyen las variables previamente detalladas y una imagen en formato .jpg con el trazado de la ruta.
- Conteo de Vehículos (**oc_count**): basado en la utilización del algoritmo YOLO. La clase **dnn** provee la creación y la configuración de la instancia del algoritmo para realizar el análisis. Análogamente depende de una interfaz de entrada de

códec mp4 y de la clase **oc_input** para la carga de la secuencia. Los resultados del análisis son procesados con los miembros de la clase **oc_output** Funcionalidades:

- Identificación y obtención de flujo de tráfico para los siguientes tipos de vehículos: autos, camiones, motocicletas, buses.
- Reportes: incluyen las variables previamente detalladas y un video en formato .avi con la identificación de los vehículos (opcional).

6.8.1 Configuración del Sistema

Se ha utilizado un ordenador con las siguientes características físicas:

- Intel Core I7-6700HQ
- 8GB RAM DDR3
- 1TB HDD

Y la siguiente configuración de software:

- Ubuntu 16.04.5 LTS Xenial Xerus
- C++ 11
- OpenCV 3.4.2

Las pruebas de validación, así como tiempos de ejecución, son válidos para trasladables a ordenadores con características similares a las previamente detalladas.

Existen dos modalidades principales para el modelamiento de sistemas para análisis de tráfico vehicular mediante visión artificial:

- Modalidad en Línea: estos sistemas se encuentran optimizados para las condiciones de su entorno operativo para que puedan funcionar en tiempo real.
- Modalidad Fuera de Línea: estos sistemas trabajan con secuencias de video previamente grabadas. Los algoritmos embebidos en este tipo de sistemas pretenden resaltar características específicas de estudio o poseen altos costes computacionales, en pos de realizar análisis más especializados.

6.8.2 Seguimiento de Vehículos

Uno de los inconvenientes principales derivados de utilizar la clase Multitracker representa la pérdida de velocidad de presentación de los cuadros de la secuencia de entrada, en forma proporcional al número de trackers creados. Para mitigar el coste computacional, se ha limitado el número de vehículos a cinco (5) unidades.

La determinación de las velocidades promedio y tiempos de viaje se verían afectadas en un enfoque tradicional enfocado al tiempo de ejecución del programa. Este inconveniente se ve solucionado realizando un análisis en base al número de FPS procesados durante la ejecución del programa. Esto garantiza la confiabilidad de los resultados obtenidos.

A. Configuración de la Clase MultiTracker

Se detalla continuación el procedimiento utilizado para implementar el seguimiento de los vehículos. El código requerido se encuentra detallado en el Anexo A.

1. Se procede a la creación e inicialización de la clase MultiTracker y la carga de la secuencia de entrada.
2. Se implementa la selección de los vehículos a seguir mediante la función **cv::selectROIS**. Este miembro de la clase **Multitracker** permite dibujar rectángulos (*bounding boxes*) para ser almacenados en una variable de tipo **Rect** (bboxes), tal y como detalla la figura 6.3.

Figura 6.3: *Selección de los vehículos a seguir*
Fuente: *El Autor*

3. Inicialización de la clase **MultiTracker** para cada uno los vehículos especificados. Se actualiza el estado de los trackers para cada nuevo cuadro de la secuencia. Paralelamente se cargan las coordenadas de los *bounding boxes* generados durante el seguimiento en un array de tipo **Point**.
4. Se procede a calcular el punto medio de los *bounding boxes* del seguimiento para almacenarlos en una nueva variable **ln** de tipo **Point,** misma que servirá para graficar la trayectoria descrita por el vehículo.
5. Se segmenta la variable **ln** en intervalos equidistantes. De esta manera se procede al cálculo de distancias parciales y obtención de distancia total recorrida aproximada.
6. Se genera la gráfica de la trayectoria de los vehículos utilizando las coordenadas de **ln** mediante la función **line( )** en la variable **xy** de tipo **Mat.** Inmediatamente después se crea un archivo **.jpg** con la información de la variable xy (**Mat**).
7. Generación del reporte de seguimiento, utilizando los siguientes parámetros:
    - Descripción del vehículo.
    - Tiempo de ejecución del programa.
    - Distancia total recorrida por cada vehículo.
    - Velocidad promedio de cada vehículo

B. Tiempo de Ejecución – Seguimiento de Vehículos

Como parámetro adicional anexo a al reporte de seguimiento, se incluye el tiempo de ejecución del programa. Esto para el cálculo del coste computacional del algoritmo en caso de seguir a más de un vehículo y/o analizar secuencias de video con alta resolución.

La utilización de la librería **<chrono>** permite calcular el tiempo de ejecución transcurrido entre un punto de inicio y uno final dentro del código a ejecutar. El código requerido se encuentra detallado en el Anexo A.

El tiempo de ejecución es comparado con la duración estimada de la secuencia de entrada para la obtención del coste computacional.

6.8.3 Conteo de Vehículos

YOLO es uno los algoritmos de detección más rápidos y confiables disponibles en la actualidad. Su tiempo de ejecución de aproximadamente 220ms por cuadro con una

tamaño de malla de 320x320 en un entorno con OpenCV resulta increíblemente superior a su equivalente en Darknet (3.2s por imagen, sin pre visualización de resultados.)

Pese a la velocidad de ejecución de YOLO con su procesamiento por CPU, un análisis en tiempo real para video grabado a 30 FPS, resultaría en aproximadamente 7sg de procesamiento por cada segundo del video de entrada, tal y como se aprecia a continuación:

$$220ms(30\ FPS) = 6600ms * FPS$$

Resulta obvio que este tiempo de ejecución no resulta en absoluto factible para una aplicación en tiempo real, y que puede resultar elevado para una aplicación fuera de línea. Durante el proceso de investigación realizado se ha determinado que existen dos opciones principales para contrarrestar los tiempos de ejecución de YOLO en su procesamiento por CPU.

A. Procesamiento Mediante GPU (OpenCL)

YOLO procesa cada imagen con un tiempo de inferencia aproximado de 22ms utilizando procesamiento con GPU, tal y como se aprecia en la tabla 6.1.

Tabla 6.1
Tiempos de inferencia por imagen de algoritmos de aprendizaje profundo

Metodología	mAP	Tiempo [ms]
SSD321	28.0	61
DSSD321	28.0	85
R-FCN	29.9	85
SSD513	31.2	125
DSSD513	33.2	156
FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	37.8	198
Yolov3-320	28.2	22
Yolov3-416	31.0	29
Yolov3-608	33.0	51

Este modo de procesamiento requiere 2 componentes principales para funcionar:

- GPU compatible: la mayoría de tarjetas NVIDIA con CUDA (Procesamiento Paralelo mediante Unidad de Procesamiento Gráfica).
- Configuración de OPENCL.

Se ha demostrado que el incremento mínimo de velocidad de un procesamiento paralelo mediante GPU (Utilizando CUDA) frente a uno por CPU es del 200% (NVIDIA, 2014). Esto se traduce en una reducción de tiempos de ejecución a la mitad. Estos valores pueden verse incluso reducidos acorde al grado de optimización de la arquitectura de la aplicación desarrollada.

OpenCL (*Open Computing Language*, Lenguaje de Computación Abierto), consiste en un entorno de desarrollo abierto para desarrollar aplicaciones con hilos de ejecución paralelos tanto para CPUs como GPUs. A más de su API nativa CUDA, NVIDIA brinda soporte para OpenCL.

Se han llegado a determinar las siguientes ventajas y desventajas de utilizar este tipo de solución:

- Ventajas: Reducción considerable de tiempos de ejecución. Alta compatibilidad para operación en entornos que requieren aplicaciones en tiempo real.
- Desventajas: Limitación de portabilidad. Sistemas que no cuenten con GPUs compatibles no pueden hacer uso de este procesamiento. También es necesario considerar la inestabilidad de los drivers de las tarjetas gráficas para sistemas operativos basados en Linux. Este punto representa un lastre importante que puede desencadenar en fallas o errores al sistema operativo del host seleccionado.

B. Ajuste de Tasa de FPS

Debido a la naturaleza del proceso del conteo de vehículos, realizar la detección los mismos en todos los cuadros de la secuencia es ineficiente desde un enfoque operativo. Es así que ajustando la tasa de lectura de cuadros por segundo, se puede nivelar de forma empírica los tiempos de análisis para que posean congruencia con los tiempos de duración de la secuencia, con una mínima pérdida de transparencia en los resultados de procesamiento del programa.

Se han llegado a determinar las siguientes ventajas y desventajas de utilizar este tipo de solución:

- Ventajas: Disminución efectiva de coste computacional. Portabilidad garantizada a otros sistemas.
- Desventajas: Ligera robotización de la secuencia de entrada.

En base al análisis previamente detallado, se hace uso de un reajuste en la tasa de FPS del video de entrada.

D. Configuración e Integración de YOLO con OpenCV

Yolo genera *bounding boxes* para cada uno de las posibles salidas (objetos) detectados. Cada uno de estos contenedores se encuentra relacionado con un valor de certeza. En las primeras etapas de procesamiento, se eliminan todos los contenedores de objetos cuyo valor de certeza se encuentre por debajo del valor de umbral establecido.

Se detalla a continuación el procedimiento utilizado para implementar el conteo de vehículos. El código requerido se encuentra detallado en el Anexo A.

1. Para iniciar, es necesario contar con los siguientes archivos:
    - yolov3.weights: archivo que contiene los pesos pre entrenado del algoritmo.
    - yolov3.cgf: archivo de configuración de la red.
    - coco.names: archivo que contiene las clases utilizadas.
2. El siguiente paso consiste en eliminar los contenedores que posean el mismo objeto, es decir, remover *bounding boxes* que se sobrepongan entre sí. Para esto se lleva a cabo una supresión de no máximos (*non maximum supression*), cuyo valor de umbral también puede ser modificado (***nms***).
3. A continuación se establecen las dimensiones de la imagen de entrada con las variables ***gridx*** y ***gridy***. Se pueden adoptar valores distintos: 320, 416 y 608. En donde una imagen de entrada de 608x608 será procesada con la mayor precisión (aumento de coste computacional) y una de 320x320 será procesada con la mayor velocidad disponible (reducción de la precisión).
4. Se procede a realizar la carga de los archivos de pesos y configuración de la red. Se establece además el tipo de procesamiento a utilizar: CPU, como se puede apreciar en al figura 6.4.

```
//load dataset
string klassf="ocelot_cfg/coco.names";
ifstream ifs(klassf.c_str( ));
string line;
while (getline(ifs,line))
     classes.push_back(line);
//load yolo cfg files
String ocelot_cfg="ocelot_cfg/yolov3.cfg";
String ocelot_weights="ocelot_cfg/yolov3.weights";
//yolo setup
Net net=readNetFromDarknet(ocelot_cfg,ocelot_weights);
net.setPreferableBackend(DNN_BACKEND_OPENCV);
     net.setPreferableTarget(DNN_TARGET_CPU);
```

Figura 6.4: *Carga de archivos de pesos y de configuración de la red*

5. Se ejecuta la lectura de la imagen de entrada.
6. Una vez que se ha cargado la imagen de entrada, es necesaria la conversión de la misma antes de que ingrese a la red en un formato especial denominado ***blob***, mediante la función ***blobFromImage***. Esta función escala los valores de pixel de la imagen con un factor de reducción de 1/255 y una salida delimitada en el rango [0,1]. También redimensiona la imagen al tamaño especificado por ***gridx*** y ***gridy***. El número de FPS procesados es almacenado en una variable (***ck2***) que posteriormente servirá para el cálculo del flujo de tráfico.
7. A continuación es necesario identificar las capas de salida mediante la función ***getOutputsNames*** que recorre toda la red.
8. En este punto la red ya entrega los *bounding boxes*, en donde cada uno de estos elementos se encuentra representado mediante un vector de nombres de clases y cinco (5) elementos adicionales. Los cuatro (4) primeros consisten en las coordenadas del centro, largo y ancho del contenedor (***center_x, center_y, width, height***). El último elemento representa el valor de confianza asociado.
9. Todos los contenedores cuyo valor de confianza se encuentren por debajo del valor de umbral son desechados. Todos aquellos cuyo valor se encuentre por encima del mismo, proceden a ser filtrados, para eliminar superposiciones innecesarias. Tal y como de comento previamente, esto se logra mediante el proceso de supresión de no máximos y el ajuste de la variable ***nmsThreshold***. Este valor posee un rango de varianza entre 0 y 1. Si es muy bajo, no se eliminarán los contenedores superpuestos (independientemente de su clase). Si el valor es muy cercano a la unidad, en cambio se obtendrán múltiples

contendores para el mismo objeto detectado. Se recomienda utilizar un valor entre [0.4, 0.5], mismo que ha sido determinado de forma empírica.

10. Finalmente se realiza la presentación de los resultados obtenidos. De manera simplificada, son 3 los principales procedimientos ejecutados por el algoritmo: redimensionamiento de la imagen de entrada, ejecución de la red y supresión de no – máximos, tal y como se resume en la figura 6.5.

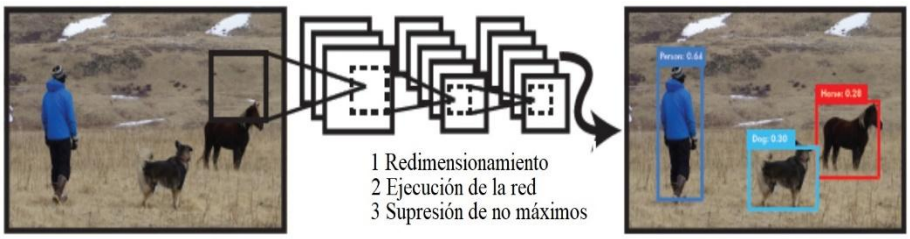


Figura 6.5: *Carga de archivos de pesos y de configuración de la red*
Fuente: (Redmon & Farhadi, 2017)

E. Cálculo del Flujo de Tráfico

Se procede a la clasificación de las detecciones realizadas en las clases de interés para el análisis y a la obtención de la sumatoria de cada una. El código requerido se encuentra detallado en el Anexo A.

Se calcula los flujos acorde a la fórmula detallada en la sección 2.8.

Se procede a la obtención de totales para cada clase de vehículo (auto, bus. camión, motocicleta).

F. Calculo del Tiempo de Ejecución – Conteo de Vehículos

Siguiendo un procedimiento análogo al detallado en la sección **6.8.2**, se incluye también el tiempo de ejecución del programa para el conteo de vehículos. El código requerido se encuentra detallado en el Anexo A.

6.9 Procesamiento de Datos de Seguimiento

Se describe a continuación el procedimiento a seguir para realizar análisis de seguimiento de vehículos con una secuencia de video de entrada determinada.

1. Se ejecuta el programa, se selecciona la opción ***t*** correspondiente a la sección de conteo de vehículos.
2. Se ingresa el número de vehículos a seguir. Adicionalmente se solicita una breve descripción de cada uno de los elementos a procesar, tal y como se detalla en la figura 6.6

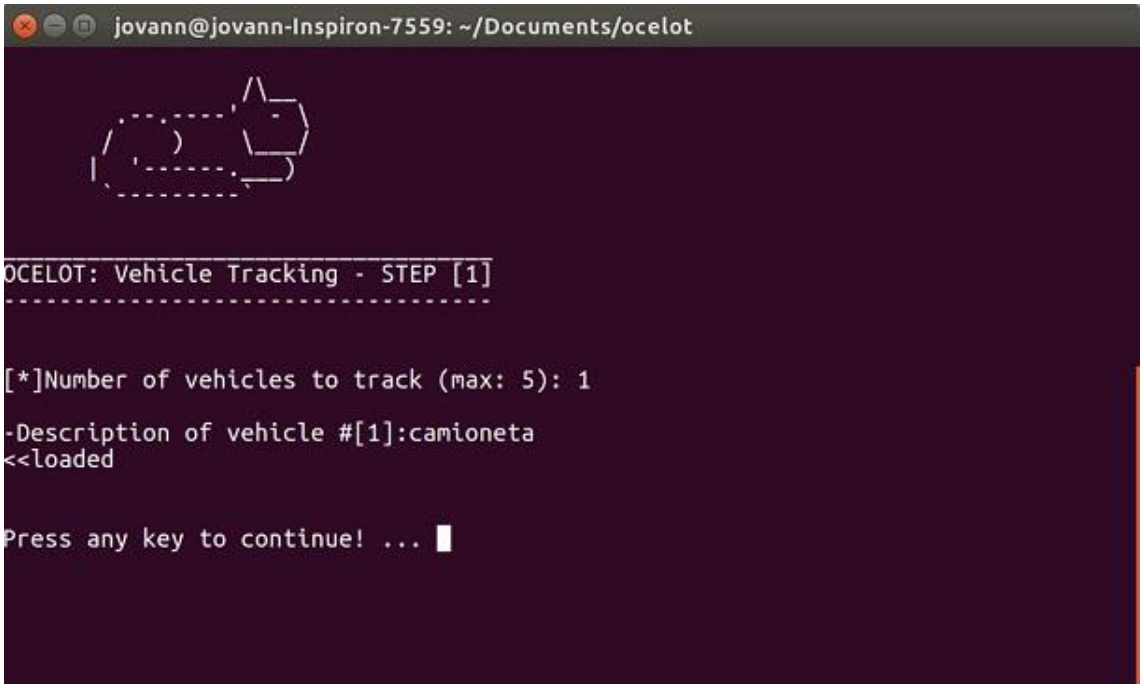


Figura 6.6: *Ingreso de número de vehículos a seguir y sus descripciones*
Fuente: *El Autor*

3. Se ingresa el tipo de procesamiento (1: rápido – algoritmo asociado: KCF, 2: preciso – algoritmo asociado: MIL) a ejecutar para el seguimiento de los vehículos.
4. Se seleccionan los vehículos a seguir.
    - Trazado de un *roi* sobre el vehículo a seguir.
    - Guardar *roi* presionando la tecla ***espacio***. Se prosigue a seleccionar el siguiente objeto. En caso de haber cometido un error al dibujar el roi, el mismo puede ser anulado presionando la tecla ***c***.
5. El programa ejecuta el procesamiento de la secuencia de entrada, como se puede apreciar en la figura 6.7.

Figura 6.7: *Procesamiento de la secuencia de entrada - seguimiento*
Fuente: *El Autor*

6. El programa despliega los resultados. Informe del tiempo de ejecución del análisis, número de cuadros analizados y confirmación de creación de reporte de seguimiento.

6.9.1 Análisis de Resultados de Seguimiento

Unidades a utilizar:

- Tiempo de ejecución del programa: t $[s]$
- Velocidad promedio del vehículo: v $[km/h]$
- Distancia recorrida: $d[m]$

La tabla 6.2, detalla los resultados obtenidos con la secuencia de prueba 1, en un análisis con las siguientes características:

- Resolución del video de entrada: 1920x1080 (1080p)
- Número de vehículos a seguir: 1

Tabla 6.2
Resultados de seguimiento con secuencia de prueba 1 – 1080p

n	$t\ [s]$	$v\ [Km/H]$	$d\ [m]$
1	5.837	25.541	67.256
2	5.603	24.987	66.859
3	5.711	25.526	65.741
4	5.452	25.745	65.258
5	5.324	25.633	66.247

La tabla 6.3, detalla los resultados obtenidos con la secuencia de prueba 1, en un análisis con las siguientes características:

- Resolución del video de entrada: 720x480 (480p)
- Número de vehículos a seguir: 1

Tabla 6.3
Resultados de seguimiento con secuencia de prueba 1 – 480p

n	$t\ [s]$	$v\ [Km/H]$	$d\ [m]$
1	1.796	24.543	65.785
2	1.841	24.987	64.256
3	1.833	25.635	65.113
4	1.795	25.748	65.237
5	1.831	24.411	66.009

La tabla 6.4, detalla los resultados obtenidos con la secuencia de prueba 2, en un análisis con las siguientes características:

- Resolución del video de entrada: 1920x1080 (1080p)
- Número de vehículos a seguir: 3

Tabla 6.4
Resultados de seguimiento con secuencia de prueba 2 – 1080p

n	t	$v\ [Km/H]$			$d[m]$		
		v_1	v_2	v_3	d_1	d_2	d_3
1	6.234	30.578	28.410	29.113	60.151	62.109	52.129
2	6.148	31.114	29.096	29.074	59.826	61.103	52.054
3	6.399	30.421	28.716	29.352	60.254	61.751	51.471
4	6.641	29.997	28.329	28.972	59.533	60.987	51.925
5	5.963	30.584	29.120	28.987	59.874	61.114	51.065

La tabla 6.5, detalla los resultados obtenidos con la secuencia de prueba 2, en un análisis con las siguientes características:

- Resolución del video de entrada: 720x480 (480p)
- Número de vehículos a seguir: 3

Tabla 6.5
Resultados de seguimiento con secuencia de prueba 2 – 480p

n	t	$v\ [Km/H]$			$d\ [m]$		
		v_1	v_2	v_3	d_1	d_2	d_3
1	14.123	30.241	28.541	27.948	59.521	62.582	51.140
2	12.987	31.587	29.541	28.566	60.258	60.541	52.324
3	13.013	29.874	28.745	28.541	59.412	61.258	51.589
4	14.622	30.263	27.448	29.023	61.124	61.488	51.473
5	12.998	31.412	29.541	29.523	59.874	61.244	52.135

La información proporcionada por las tablas, permite denotar las siguientes particularidades:

- Los tiempos de ejecución del sistema varían en función con el número de vehículos a seguir.
- Los tiempos de ejecución del sistema varían en función de la resolución de la secuencia de entrada.

Se establecen además, las siguientes conclusiones:

- La utilización de una variable para el registro de los cuadros procesados garantiza la transparencia de los resultados obtenidos, incluso cuando el coste computacional del sistema aumenta.
- Los resultados de velocidad y distancia recorrida dependen de dos factores: la optimización del algoritmo y la correcta relación entre de unidades métricas y píxeles.

Cabe destacar que los resultados de distancia recorrida y velocidad promedio de cada vehículo son experimentales. La transparencia de los mismos es garantizada a través de un desarrollo a medida.

6.10 Procesamiento de Datos de Conteo

Se describe a continuación el procedimiento a seguir para realizar análisis de conteo de vehículos con una secuencia de video de entrada determinada.

1. Se ejecuta el programa, se selecciona la opción ***c*** correspondiente a la sección de conteo de vehículos.
2. Se ingresan de datos generales de configuración. Nombre del video (sin la extensión), tipo de formato (mp4, avi), confirmación de creación de video con los vehículos detectados (si, no), como se puede apreciar en la figura 6.8. Máxima resolución admitida: FULL HD 1080p.

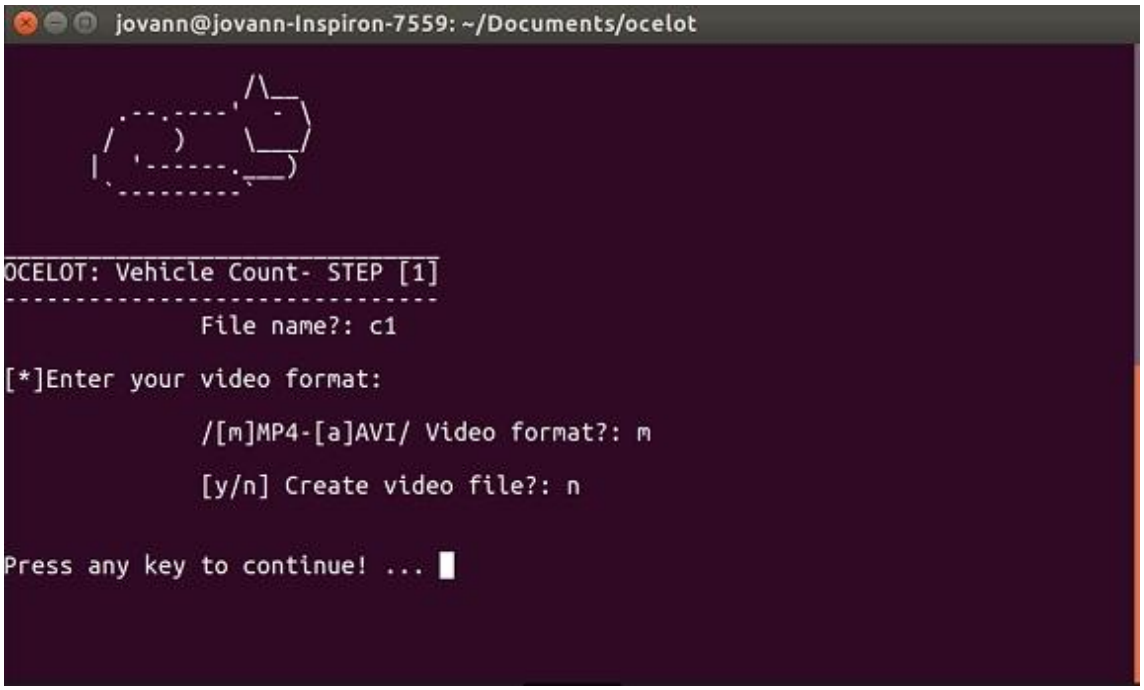


Figura 6.8: *Ingreso de datos generales del archivo de video*
Fuente: *El Autor*

3. Se ingresa el balance del procesamiento (1: mayor velocidad, 2: balanceado, 3: mayor precisión) sobre los vehículos a detectar.
4. Se selecciona la tasa de FPS (f: análisis al video completo, s: análisis a 5FPS).
5. El programa ejecuta el procesamiento de la secuencia de entrada, como se puede apreciar en la figura 6.9.

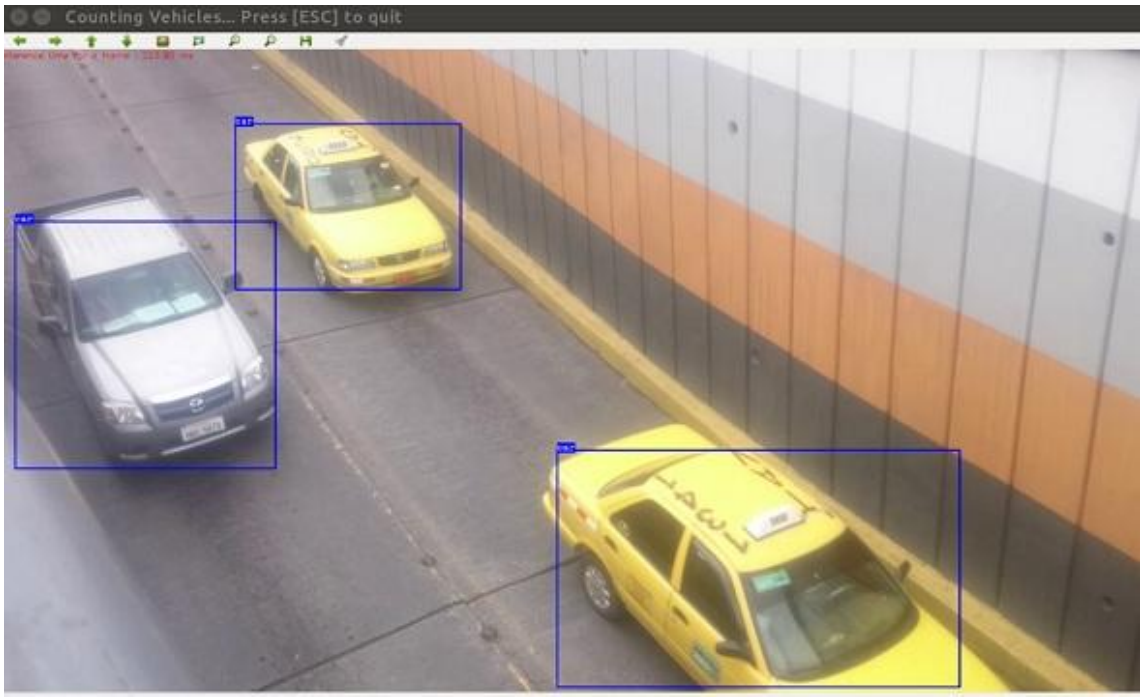


Figura 6.9: *Procesamiento de la secuencia de entrada - conteo*
Fuente: *El Autor*

6. Se despliegan los resultados. Informe del tiempo de ejecución del análisis, número de cuadros analizados y confirmación de creación de reporte de conteo.

6.10.1 Análisis de Resultados de Conteo

Unidades a utilizar:

- Imagen de entrada: pixeles.
- Tiempo de ejecución del programa: $t\ [s]$
- Flujo: Q $[vh/min] - vehiculos\ por\ minuto$

La tabla 6.6, detalla los resultados obtenidos con la secuencia de prueba 1, con un análisis completo (30 FPS) de la entrada.

- *Secuencia 1: 23 vehículos, 30s, 1080p.*

Tabla 6.6

Resultados de conteo con secuencia de prueba 1 – 30 FPS.

Balance	***Imagen de entrada [pix]***	***n***	***t [s]***	***Q [vh/min]***	***Vehículos***
1	320x320	1	232.872	48.641	24 (24.320)
		2	230.344	48.641	
		3	232.110	48.641	
		4	234.016	48.641	
		5	231.452	48.641	
		ȳ	**232.158**	48.641	
2	416x416	1	312.114	47.493	24 (23.746)
		2	309.780	47.493	
		3	310.113	47.493	
		4	311.672	47.493	
		5	311.012	47.493	
		ȳ	**310.938**	**47.493**	
3	608x608	1	586.121	46.495	23 (23.247)
		2	585.843	46.495	
		3	587.320	46.495	
		4	589.211	46.495	
		5	590.244	46.495	
		ȳ	**587.747**	**46.495**	

La tabla 6.7, detalla los resultados obtenidos con la secuencia de prueba 1, con un análisis a 5FPS de la entrada.

- *Secuencia 1: 23 vehículos, 30s, 1080p.*

Tabla 6.7

Resultados de conteo con secuencia de prueba 1 – 5 FPS.

Balance	***Imagen de entrada [pix]***	***n***	***t [s]***	***Q [vh/min]***	***Vehículos***
1	320x320	1	26.447	52.20	26 (26.1)
		2	26.781	52.20	
		3	26.277	52.20	
		4	26.356	52.20	
		5	26.354	52.20	
		ȳ	**26.443**	**52.20**	
2	416x416	1	36.273	51.42	25 (25.71)
		2	36.140	51.42	
		3	36.321	51.42	
		4	35.841	51.42	
		5	36.112	51.42	
		ȳ	**36.137**	**51.42**	
3	416x416	1	69.102	45.97	22 (22.98)
		2	69.309	45.97	
		3	69.743	45.97	
		4	68.726	45.97	
		5	69.014	45.97	
		ȳ	**69.178**	**45.97**	

La tabla 6.8, detalla los resultados obtenidos con la secuencia de prueba 2, con un análisis completo (30 FPS) de la entrada.

- *Secuencia 2: 7 vehículos, 24s, 1080p.*

Tabla 6.8

Resultados de conteo con secuencia de prueba 2 – 30 FPS.

Balance	***Imagen de entrada [pix]***	***n***	***t [s]***	***Q [vh/min]***	***Vehículos***
1	320x320	1	184.259	19.450	8 (7.78)
		2	187.197	19.450	
		3	186.342	19.450	
		4	185.356	19.450	
		5	186.876	19.450	
		ȳ	**186.006**	**19.450**	
2	416x416	1	250.576	19.450	8 (7.78)
		2	252.167	19.450	
		3	253.122	19.450	
		4	251.977	19.450	
		5	251.231	19.450	
		ȳ	**251.814**	**19.450**	
3	416x416	1	476.320	17.527	7 (7.01)
		2	474.221	17.527	
		3	475.101	17.527	
		4	476.989	17.527	
		5	477.239	17.527	
		ȳ	**475.974**	**17.527**	

La tabla 6.9, detalla los resultados obtenidos con la secuencia de prueba 2, con un análisis a 5FPS de la entrada.

- *Secuencia 2: 7 vehículos, 24s, 1080p.*

Tabla 6.9

Resultados de conteo con secuencia de prueba 2 – 5 FPS.

Balance	***Imagen de entrada [pix]***	***n***	***t [s]***	***Q [vh/min]***	***Vehículos***
1	320x320	1	21.298	19.67	7 (7.868)
		2	21.126	19.67	
		3	21.121	19.67	
		4	21.345	19.67	
		5	21.541	19.67	
		ȳ	**21.286**	**19.67**	
2	416x416	1	29.019	19.67	7 (7.868)
		2	28.796	19.67	
		3	30.986	19.67	
		4	29.144	19.67	
		5	30.258	19.67	
		ȳ	**29.640**	**19.67**	
3	416x416	1	55.367	17.70	7 (7.081)
		2	54.769	17.70	
		3	55.012	17.70	
		4	54.971	17.70	
		5	55.298	17.70	
		ȳ	**55.083**	**17.70**	

La tabla 6.10, detalla los resultados obtenidos con la secuencia de prueba 3, con un análisis completo (30 FPS) de la entrada.

- *Secuencia 3: 22 vehículos, 51s, 1080p.*

Tabla 6.10

Resultados de conteo con secuencia de prueba 3 – 30 FPS.

Balance	***Imagen de entrada [pix]***	***n***	***t [s]***	***Q [vh/min]***	***Vehículos***
1	320x320	1	390.120	30.93	26 (26.29)
		2	388.742	30.93	
		3	388.988	30.93	
		4	390.751	30.93	
		5	390.560	30.93	
		ȳ	**389.382**	**30.93**	
2	416x416	1	525.641	29.53	25 (25.10)
		2	526.987	29.53	
		3	523.658	29.53	
		4	526.042	29.53	
		5	525.874	29.53	
		ȳ	**525.64**	**29.53**	
3	416x416	1	995.214	26.71	22 (22.70)
		2	994.854	26.71	
		3	993.241	26.71	
		4	992.481	26.71	
		5	995.442	26.71	
		ȳ	**994.246**	26.71	

La tabla 6.11, detalla los resultados obtenidos con la secuencia de prueba 3, con un análisis a 5FPS de la entrada.

- *Secuencia 3: 22 vehículos, 51s, 1080p.*

Tabla 6.11
Resultados de conteo con secuencia de prueba 3 – 5 FPS.

Balance	***Imagen de entrada [pix]***	***n***	***t [s]***	***Q [vh/min]***	***Vehículos***
1	320x320	1	43.788	27.784	23 (23.616)
		2	46.644	27.784	
		3	43.588	27.784	
		4	43.741	27.784	
		5	44.985	27.784	
		ȳ	**44.549**	27.784	
2	416x416	1	63.417	27.178	23 (23.102)
		2	59.890	27.178	
		3	60.335	27.178	
		4	61.576	27.178	
		5	60.210	27.178	
		ȳ	**61.085**	**27.178**	
3	416x416	1	117.608	26.718	22 (22.710)
		2	114.753	26.718	
		3	114.997	26.718	
		4	115.211	26.718	
		5	116.957	26.718	
		ȳ	**115.905**	**26.718**	

La información proporcionada por las tablas 6.6, 6.7, 6.8, 6.9, 6.10, 6.11; permite denotar las siguientes particularidades:

- Los tiempos de ejecución del detector, son mínimamente variantes entre sí.
- Los resultados del flujo con imágenes de 320x320 y 416x416 son cercanos entre sí. No siendo el caso de la imagen de 608x608, cuyos resultados poseen una varianza ligeramente mayor.
- El valor del flujo es constante, manteniendo los parámetros de *fps* y tamaño de la imagen de entrada (balance de procesamiento).
- La diferencia de flujos entre análisis a 30 *fps* y 5 *fps* es mínima.

Se establecen además las siguientes conclusiones:

- La utilización de un tamaño de imagen de entrada de 320x320 ofrece la mayor velocidad de procesamiento, con resultados altamente transparentes para el usuario.

- Existe un grado de repetividad altamente satisfactorio de los resultados obtenidos.
- El análisis de conteo mediante la reducción de la tasa de *fps* garantiza la integridad de los resultados y una disminución efectiva del coste computacional del sistema.

Los resultados de la tabla permiten denotar además que los mejores resultados pueden ser obtenidos con una imagen de entrada de **608x608**. Utilizar esta configuración permite obtener resultados más precisos, con un coste computacional aproximado del **8%** (tiempo de análisis vs tiempo de duración de la secuencia). Un análisis empírico y estadístico puede ser aplicado para encontrar la relación de corrección para un escenario determinado, utilizando la imagen de entrada de **320x320**, disminuyendo el coste de procesamiento.

El presente software puede ser utilizado en hosts con características similares o superiores a las detalladas en la sección 6.11. Los tiempos de inferencia se ven notablemente incrementados en plataformas de desarrollo de sistemas embebidos tales como Raspberry Pi. Esto se debe a la tasa de flops, misma que se detalla en la tabla 6.12.

Tabla 6.12
Tasa de Flops de Plataformas de Sistemas Embebidos

Plataforma	**Tasa de Flops [M-FLOPS]**
Raspberry Pi 2	157
Raspberry Pi 3	192

Yolo en su versión 3 (la utilizada en el presente proyecto) demanda una tasa de 5.58 GFLOPS (*Floating Operations per Second*, Operaciones de Punto Flotante por Segundo). Para utilizar YOLO en este tipo de sistemas se dispone de Tiny YOLO. Esta versión orientada a sistemas con menor capacidad de procesamiento puede inferir hasta 200 fps a coste de poseer una exactitud de aproximadamente un tercio del algoritmo original (YOLO mAP: 60.6, Tiny YOLO mAP: 23.7) (Redmond & Farhadi, 2017).

Se ha constatado además que la clasificación de vehículos posee una relación de acierto aproximada del **78%**, siendo la clase “auto” la mejor identificada. Existe cierta discrepancia para la clasificación de las clases “camión” y “bus”. En base a las

observaciones y mediciones realizadas se sugieren las siguientes acciones para mejorar la relación de acierto de la clasificación de los vehículos:

- Manipular el ángulo de captura de video para conseguir una toma íntegra de los vehículos. La distancia debe ser la suficiente como para enfocar adecuadamente a cada uno de los mismos, así también como para no ocluir gran parte de la secuencia con el paso de vehículos de tamaño considerable (buses, camiones).

A. Verificación de la Hipótesis

Se procede a realizar la verificación de la hipótesis mediante la metodología del χ^2, cuya ecuación se detalla a continuación:

$$\chi^2 = \sum_{i=1}^{f} \sum_{j=1}^{c} \frac{(fo_{ij} - ft_{ij})^2}{ft_{ij}}$$

(6.1)

En donde:

χ^2: $valor\ calculado\ de\ ji\ cuadrado$

f: $total\ de\ filas$

c: $total\ de\ columnas$

fo_{ij}: $frecuencia\ observada\ en\ la\ casilla\ [i,j]$

ft_{ij}: $frecuencia\ teorica\ de\ la\ casilla\ [i,j]$

Teniendo que:

$$ft_{ij} = \frac{Sumatoria\ de\ la\ fila\ i * Sumatoria\ de\ la\ columna\ j}{Total\ de\ Observaciones}$$

(6.2)

Se describen a continuación las hipótesis para el presente trabajo de investigación:

H_0 - La exactitud de los resultados del cálculo de flujo de tráfico de un sistema basado en visión artificial es congruente con respecto a la obtenida mediante una metodología dedicada no automatizada.

H_1 - La exactitud de los resultados del cálculo de flujo de tráfico de un sistema basado en visión artificial difiere con respecto a la obtenida mediante una metodología dedicada no automatizada.

La tabla detalla 6.13 las frecuencias observadas en base a los datos proporcionados por las tablas **6.7**, **6.9** y **6.11** (análisis a 5 fps) y los valores reales verificados para cada secuencia con sus respectivos marginales.

Tabla 6.13
Frecuencias observadas

Prueba	**S1 320**	**S1 416**	**S1 608**	**S2 320**	**S2 416**	**S2 608**	**S3 320**	**S3 416**	**S3 608**	**TOTAL**
Software	26	25	22	7	7	7	26	25	22	**167**
Manual	23	23	23	7	7	7	22	22	22	**156**
TOTAL	**49**	**48**	**45**	**14**	**14**	**14**	**48**	**47**	**44**	**323**

Por cuanto, se adoptan las siguientes condiciones para la prueba estadística:

$$\chi^2 = (8, N = 323) = 17.5345, p = 0.025$$

Se procede a calcular las frecuencias teóricas correspondientes a cada frecuencia observada, con la ecuación 6.2, teniendo como resultado los valores expuestos en la tabla 6.14.

Tabla 6.14
Frecuencias teóricas

Prueba	**S1 320**	**S1 416**	**S1 608**	**S2 320**	**S2 416**	**S2 608**	**S3 320**	**S3 416**	**S3 608**
Software	25.33	24.82	23.27	7.24	7.24	7.24	24.82	24.30	22.75
Manual	23.67	23.18	21.73	6.76	6.76	6.76	23.18	23.70	21.25

Aplicando la ecuación 6.1 se tiene que:

$$\chi^2{}_{calculado} = 0.0175 + 0.0013 + 0.0689 + 0.0079 + 0.0079 + 0.0079 + 0.0564 + 0.0201 + 0.0247 + 0.0187 + 0.0014 + 0.0738 + 0.0084 + 0.0084 + 0.0084 + 0.0603 + 0.0216 + 0.0264 = \mathbf{0.4399}$$

El valor del χ^2 teórico para un margen de error del 2.5% y 8 grados libertad es de:

$$\chi^2{}_{teorico} = \mathbf{17.5345}$$

Se tiene que:

$$\chi^2_{calculado} < \chi^2_{teorico}$$

Por lo cual se confirma la hipótesis nula:

H_0 - La exactitud de los resultados del cálculo de flujo de tráfico de un sistema basado en visión artificial es congruente con respecto a la obtenida mediante una metodología dedicada no automatizada.

Validando así los resultados del presente proyecto de investigación.

6.11 Administración

6.11.1 Recursos Humanos

El proyecto se implementa con la participación de activa del investigador (Ing. Jovann Pérez) y el tutor (Ing. Darío Mendoza, MSc.), quienes desarrollaron el software de la investigación, de forma que todos los contenidos de este documento permitan entender su funcionamiento y utilización.

6.11.2 Recursos Materiales

El producto del resultado de investigación consiste en un programa escrito en C++ compatible con sistemas operativos basados en Debian (Ubuntu). Es así que se requiere de los siguientes componentes tanto físicos como lógicos:

Características del Host:

- Procesador Intel Core I7-6700HQ o superior
- 8GB RAM DDR3 o superior

Requisitos:

- Sistema Operativo: Ubuntu 16.04.5 LTS – Xenial Xerus
- C++ 11.0
- OpenCV 3.4.2

6.11.3 Recursos Económicos

La tabla, detalla los costes derivados de la implementación del presente proyecto de investigación.

6.12 Conclusiones y Recomendaciones

6.12.1 Conclusiones

- Un programa de análisis de tráfico en modalidad fuera de línea, permite solventar los problemas asociados a los de tiempo real. Cambios de ángulo, cambios de iluminación, oclusiones; los efectos de estas perturbaciones sobre los cálculos del sistema pueden ser mitigados y/o corregidos a través de la modificación de parámetros operativos. Análogamente la selección de muestras de video significativas, permite extrapolar los resultados y validarlos para distintos horarios y/o cadencias de tráfico.
- La clase MultiTracker de OpenCV posee un desempeño satisfactorio para realizar seguimiento de objetos en general. Dos parámetros influyen directamente en la velocidad de procesamiento, el número de objetos a seguir y la resolución del video de entrada. Teniendo en consideración ambos factores, se pueden desarrollar sistemas que aunque aumente su coste computacional, garanticen la integridad de los resultados obtenidos durante el análisis.
- Las redes neuronales de flujo directo constituyen una solución robusta, rápida y eficaz para la ejecución de tareas de detección de objetos. YOLO garantiza repetividad en sus cálculos, mayor de velocidad de procesamiento, y porcentajes de rendimiento comparables a los de algoritmos más precisos basados en RCNNs. Se ha verificado que la utilización de una imagen de entrada de 320x320 resulta adecuada para la mayoría de escenarios de análisis.

6.12.2 Recomendaciones

- La metodología del presente trabajo puede ser extrapolada a casos de estudio similares basados en la utilización de visión artificial. Por lo cual, se recomienda la utilización de OpenCV como entorno desarrollo para este tipo de aplicaciones. Su amplia gama de algoritmos especializados, documentación y comunidad activa, hacen que sea una de las alternativas preferidas por desarrolladores en el campo.
- Se recomienda el estudio métodos de programación paralela para reducir tiempos de ejecución en seguimiento de varios objetos mediante OpenCV.

- Se recomienda la utilización de algoritmos de aprendizaje profundo para la ejecución de tareas de detección y/o conteo de objetos. Este tipo de alternativa ofrece resultados sumamente precisos y constituyen un área de investigación de interés en visión artificial.

REFERENCIAS

Ambata, L. U., Lijauco, M. C. E., Nuval, C. A. C., & Vergara, R. J. V. (2016). Distance monitoring vision system for automobiles using image processing. *8th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management, HNICEM 2015*, (December). https://doi.org/10.1109/HNICEM.2015.7393164

Asha, C. S., & Narasimhadhan, A. V. (2018). Vehicle Counting for Traffic Management System using YOLO and Correlation Filter, 1–6.

Babenko, B., Belongie, S., & Yang, M. (2009). Visual Tracking with Online Multiple Instance Learning, 983–990.

Babenko, B., Ming-Hsuan Yang, & Belongie, S. (2011). Robust Object Tracking with Online Multiple Instance Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *33*(8), 1619–1632. https://doi.org/10.1109/TPAMI.2010.226

Battiato, S., Farinella, G. M., & Giudice, O. (2013). Vision Based Traffic Analysis. *AEIT Annual Conference 2013*, 6.

Choudhury, S., Hazra, T. K., & Chattopadhyay, S. P. (2017). Vehicle Detection and Counting using Haar Feature- Based Classifier, 106–109.

Dalal, N., & Triggs, B. (n.d.). Histograms of Oriented Gradients for Human Detection.

Foundation, T. O. (2018). The OpenCV Tutorials, *2.4.13.7*, 447.

Girshick, R. (2015). Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, *2015 Inter*, 1440–1448. https://doi.org/10.1109/ICCV.2015.169

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 580–587. https://doi.org/10.1109/CVPR.2014.81

Hsu, S.-C., Huan, C.-L., & Chuang, C.-H. (2018). Vehicle Detection using Simplified Fast R-CNN, 3–5.

Hsu, S., & Chuang, C.-H. (2018). Vehicle Detection using Simplified Fast R-CNN, 3–5.

Huang, T. S. (2003). Computer Vision : Evolution and Promise, *1*.

Kim, J., & Sim, A. (2017). A New Approach to Online, Multivariate Network Traffic Analysis. https://doi.org/10.1109/ICCCN.2017.8038520

Krizhevsky, A., & Hinton, G. E. (n.d.). ImageNet Classification with Deep Convolutional Neural Networks, 1–9.

Kun, A. J., & Vámossy, Z. (2009). Traffic Monitoring with Computer Vision, 131–134.

Li, D., Liang, B., & Zhang, W. (2014). Real-time moving vehicle detection, tracking, and counting system implemented with OpenCV. *ICIST 2014 - Proceedings of 2014 4th IEEE International Conference on Information Science and Technology*, 631–634. https://doi.org/10.1109/ICIST.2014.6920557

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C., & Berg, A. C. (n.d.).

SSD : Single Shot MultiBox Detector.

Maldonado, S. (2016). *La Información del Estado del Tráfico en Tiempo Real y su Incidencia en el Traslado Vehicular en el Casco Central de la Ciudad de Ambato*.

Miller, N., Thomas, M. A., Eichel, J. A., & Mishra, A. (2015). A Hidden Markov Model for Vehicle Detection and Counting. *2015 12th Conference on Computer and Robot Vision*. https://doi.org/10.1109/CRV.2015.42

Mozo Sánchez, J. (2012). Capítulo: Teoría de flujo vehicular. *Análisis de Capacidad Y Nivel de Servicio de Segmentos Básicos de Autopistas, Segmentos Trenzados Y Rampas de Acuerdo Al Manual de Capacidad de Carreteras HCM2000 Aplicando MathCad*, 10–29. Retrieved from http://www.ptolomeo.unam.mx:8080/xmlui/bitstream/handle/132.248.52.100/417/A4.pdf

Redmon, J., & Farhadi, A. (2017). YOLOv3 : An Incremental Improvement.

Roberts, L. (1963). *Machine Perception of 3D Solids*. Massachusetss Institute of Technology.

Sanabria JJ ;Archila JF. (2011). Detección y análisis de movimiento usando visión artificial. *Rev. Scientia et Technica*, *3*(49), 180–188.

Saribas, H., & Cevikalp, H. (2018). Car Detection in Images Taken from Unmanned Aerial Vehicles, 2–5.

Subaweh, M., & Wibowo, E. P. (2016). Implementation of Pixel Based Adaptive Segmenter Method for Tracking and Counting Vehicles in Visual Surveillance. *2016 International Conference on Informatics and Computing (ICC)*, (1), 4–8.

Suryatali, A. V, & Dharmadikari, V. B. (2015). Computer Vision Based Vehicle Detection for Toll Collection System Using Embedded Linux. *2015 International Conference on Circuit, Power and Computing Technologies [ICCPCT]*, 7.

Szeliski, R. (2010). *Computer Vision: Algorithms and Applications*. Springer.

Tao, J., Wang, H., Zhang, X., Li, X., & Yang, H. (2017). An Object Detection System Based on YOLO in Traffic Scene. *2017 &th International Conference on Computer Science and Network Technology (ICCSNT)*, 315–319.

Viola, P. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. *Coonference On Computer Vision and Pattern Recognition 2001*.

Wan, J., Yuan, Y., & Wang, Q. (2017). TRAFFIC CONGESTION ANALYSIS : A NEW PERSPECTIVE, 1398–1402.

Wen, H., Wu, G., & Li, J. (2015). An improved tracking-learning-detection method. *Chinese Control Conference, CCC*, *2015–Septe*, 3858–3863. https://doi.org/10.1109/ChiCC.2015.7260234

Yang, H., & Qu, S. (2017). Real-time vehicle detection and counting in complex traffic scenes using background subtraction model with low-rank decomposition. *IET Intelligent Transport Systems*, 75–85. https://doi.org/10.1049/iet-its.2017.0047

Zhang, Z., Liu, K., Gao, F., Li, X., & Wang, G. (n.d.). Vision-based Vehicle Detecting and Counting for Traffic Flow Analysis. *2016*, (13), 2267–2273.

Zheng, J., & Wang, Y. (2015). CNN Based Vehicle Counting with Virtual Coil in Traffic Surveillance Video. *2015 IEEE International Conference on Multimedia Big Data*. https://doi.org/10.1109/BigMM.2015.56

ANEXOS

ANEXO A: CÓDIGO FUENTE (C++)

```
#include <opencv2/opencv.hpp>
#include <opencv2/tracking.hpp>
#include <fstream>
#include <ctime>
#include <iomanip>
#include <iostream>
#include <cstdlib>
#include <chrono>
using namespace cv;
using namespace std;
using namespace dnn;

//count variables
// yolo_parameters
float confThreshold = 0.5; //confidence threshold
float nmsThreshold = 0.4;  // non-maximum suppression threshold
int gridx,gridy;  // 1:320,2:416,3:608
vector<string> classes;
int fg;
long long int kind[50000][100];
long double scp[50000][100];
long double edge[50000],car=0,bike=0,truck=0,bus=0;

//tracking variables
int x=0,knd=0;
float trekf,xi[5],yi[5],xe[5],ye[5],d[5],speed[5],vl;
char video_in[50],ans;
char fm,name[5][30],pname[40];
bool cross=1,expand=0;
Scalar purple(78,45,201),white(255,255,255);
Point cd[5][5000][2];
Mat xy(1000,1000,CV_8UC3,Scalar(255,255,255));
//wide variables
long long int ck=0;
float fps;
char spd[15],choose,fname[40],*now_s;
time_t now_t;
ofstream treport,creport;

//track fxs
vector<string> trackerTypes = {"MIL","KCF"};
// create tracker by name
Ptr<Tracker> createTrackerByName(string trackerType){
  Ptr<Tracker> tracker;
  if (trackerType ==  trackerTypes[0])
    tracker=TrackerKCF:create( );
  else if (trackerType == trackerTypes[1])
    tracker=TrackerMIL:create( );
  return tracker;
}
inline void color_rand(vector<Scalar> &colors, int numColors){
      RNG rng(0);
      for(int j=0; j<numColors;j++)
            colors.push_back(Scalar(rng.uniform(0,255), rng.uniform(0,
255), rng.uniform(0, 255)));
```

```
}
//wide fxs
inline void welcome( ){
      cout<<endl<<endl;
      cout<<"
      cout<<"
      cout<<"
      cout<<"
      cout<<"
}
inline void ocelot ( ){
      cout<<"                  /\\__ "<<endl;
      cout<<"        .--.----'  - \\ "<<endl;
      cout<<"       /    )    \\___/ "<<endl;
      cout<<"      |  '------.___) "<<endl;
      cout<<"       `---------` "<<endl;
}
inline void repeat(char sym, int n){
      for(int k=0;k<n;k++)
                  cout<<sym;
      cout<<endl;
}
inline void bye( ){
      welcome( );
      ocelot( );
      cout<<endl<<endl;
      cout<<"Developed a tested by:"<<endl;
      cout<<setw(34)<<" ";repeat('_',strlen("JVNN"));
      cout<<setw(34)<<" "<<"JVNN"<<endl;
      cout<<setw(34)<<" ";repeat('-',strlen("JVNN"));
}

//count fxs
inline void drawPred(int classId, float conf, int left, int top, int
right, int bottom, Mat& frame){
    rectangle(frame, Point(left, top), Point(right, bottom),
Scalar(230, 215, 30), 2);
    string label = format("%.2f",conf);
    if (!classes.empty()){
        CV_Assert(classId < (int)classes.size( ));
        label=classes[classId];
            //label=classes[classId] + ":" + label;
    }
    int baseLine;
    Size labelSize = getTextSize(label, FONT_HERSHEY_COMPLEX_SMALL,
0.5, 1, &baseLine);
    top = max(top, labelSize.height);
    rectangle(frame, Point(left, top - round(1.5*labelSize.height)),
Point(left + round(1.5*labelSize.width), top + baseLine), Scalar(230,
215, 30), FILLED);
    putText(frame, label, Point(left, top),
FONT_HERSHEY_COMPLEX_SMALL, 0.75, Scalar(255,255,255),1);
}
inline void postprocess(Mat& frame, const vector<Mat>& outs){
      vector<int> classIds;
    vector<float> confidences;
    vector<Rect> boxes;
    for (size_t i = 0; i < outs.size(); ++i){
        float* data = (float*)outs[i].data;
        for (int j = 0; j < outs[i].rows; ++j, data += outs[i].cols){
            Mat scores = outs[i].row(j).colRange(5, outs[i].cols);
```

```
            Point classIdPoint;
            double confidence;
            // Get the value and location of the maximum score
            minMaxLoc(scores, 0, &confidence, 0, &classIdPoint);
            if (confidence>confThreshold){
                int centerX=(int)(data[0]*frame.cols);
                int centerY=(int)(data[1]*frame.rows);
                int width=(int)(data[2]*frame.cols);
                int height=(int)(data[3]*frame.rows);
                int left=centerX-width/2;
                int top=centerY-height/2;
                classIds.push_back(classIdPoint.x);
                confidences.push_back((float)confidence);
                boxes.push_back(Rect(left,top,width,height));
            }
        }
    }
    vector<int> indices;
    NMSBoxes(boxes, confidences, confThreshold, nmsThreshold,
indices);
    edge[ck]=indices.size( );
      cout<<"OCELOT - # of objects detected: "<<indices.size( )<<"- #
of frames: "<<ck<<endl;
      for (size_t j=0;j<indices.size( );j++){
        int idx=indices[j];
        Rect box=boxes[idx];

drawPred(classIds[idx],confidences[idx],box.x,box.y,box.x+box.width,bo
x.y+box.height,frame);
            kind[ck][j]=classIds[idx];
            scp[ck][j]=confidences[idx];
    }
      ck++;
}
inline vector<String> getOutputsNames(const Net& net){
    static vector<String> names;
    if (names.empty( )){
        vector<int> outLayers = net.getUnconnectedOutLayers( );
        //get the names of all the layers in the network
        vector<String> layersNames = net.getLayerNames( );
        // Get the names of the output layers in names
        names.resize(outLayers.size( ));
        for (size_t j=0;j<outLayers.size( );j++)
        names[j]=layersNames[outLayers[j]-1];
    }
    return names;
}

//main function
int main(int argc, char * argv[]){
      do{
            do{
                  system("clear");
                  welcome( );
                  cout<<"[*]Welcome to OCELOT!"<<endl<<endl;
                  cout<<setw(34)<<"[*]What are you up to? Pick your
choice:"<<endl;
                  cout<<endl;
                  cout<<setw(14)<<" ";repeat('=',strlen("(t) - Vehicle
Tracking"));
                  cout<<setw(14)<<" "<<"(t) - Vehicle Tracking"<<endl;
```

```
                  cout<<setw(14)<<" "<<"(c) - Vehicle Count"<<endl;
                  cout<<setw(14)<<" "<<"(q) - Quick reference"<<endl;
                  cout<<setw(14)<< " "<<"(a) - About OCELOT"<<endl;
                  cout<<setw(14)<<" ";repeat('=',strlen("(t) - Vehicle
Tracking"));
                  cout<<endl<<endl;
                  cout<<setw(34)<<"Selection??:";
                  cin.getline(spd,10);
                  choose=spd[0];
            }while(!(choose=='t' || choose=='c' || choose=='q' ||
choose =='a'));
            switch(choose){
                  case 't':{
                        do{
                              system("clear");
                              cout<<endl;
                              ocelot( );
                              cout<<endl;
                              repeat('_',strlen("OCELOT: Vehicle
Tracking - STEP [1]"));
                              cout<<"OCELOT: Vehicle Tracking - STEP
[1]"<<endl;
                              repeat('-',strlen("OCELOT: Vehicle
Tracking - STEP [1]"));
                              cout<<endl<<endl;
                              cout<<setw(34)<<"Number of elements to
track (max 10): ";
                              cin.getline(spd,10);
                              x=atoi(spd);
                              if(!(0<x && x<11))
                                    cout<<setw(34)<<" "<<"Invalid
data!"<<endl;
                              }while(!(0<x && x<11));
                              cout<<endl;
                              for(int j=0;j<x;j++){
                                    cout<<setw(5)<<"Description of
vehicle #["<<j+1<<"]:";
                                    cin.getline(name[j],30);
                                    cout<<"<<loaded"<<endl;
                              }
                              cout<<endl<<endl;
                              cout<<"Press any key to continue! ... ";
                              cin.get( );
                              system("clear");
                              cout<<endl;
                              ocelot( );
                              cout<<endl;
                              repeat('_',strlen("OCELOT: Vehicle
Tracking- STEP [2]"));
                              cout<<"OCELOT: Vehicle Tracking- STEP
[2]"<<endl;
                              repeat('-',strlen("OCELOT: Vehicle
Tracking- STEP [2]"));
                              cout<<endl<<endl;
                              cout<<"[*]Available algorithms
are:"<<endl<<endl;
                              cout<<setw(15)<<"
";repeat('=',strlen("[2] Accurate Traking - KCF Algorithm"));
                              cout<<setw(15)<<" "<<"[1] Fast Tracking -
MIL Algorithm"<<endl;
```

```
                              cout<<setw(15)<<" "<<"[2] Accurate
Traking - KCF Algorithm"<<endl;
                              cout<<setw(15)<<"
";repeat('=',strlen("[2] Accurate Traking - KCF Algorithm"));
                              do{
                                    cout<<endl<<setw(15)<<"Tracking
algorithm to use?: ";
                                    cin.getline(spd,10);
                                    knd=atoi(spd);
                                    if(knd<1 || knd > 2)

      cout<<endl<<setw(15)<<"OCELOT: ERROR - Pick up one of available
options"<<endl;
                                    }while(knd<1 || knd > 2);
                                    cout<<endl<<endl;
                                    cout<<"Press any key to continue!
... ";
                                    cin.get( );
                                    system("clear");
                                    cout<<endl;
                                    ocelot( );
                                    cout<<endl;
                                    repeat('_',strlen("OCELOT: Vehicle
Tracking- STEP [3]"));
                                    cout<<"OCELOT: Vehicle Tracking-
STEP [3]"<<endl;
                                    repeat('-',strlen("OCELOT: Vehicle
Tracking- STEP [3]"));
                                    cout<<endl<<endl;
                                    cout<<setw(15)<<" "<<"File name?:
";
                                    cin.getline(video_in,50);
                                    cout<<endl;
                                    cout<<"[*]Enter your video
format:"<<endl<<endl;
                                    do{
                                          cout<<setw(14)<<"
"<<"/[m]MP4-[a]AVI/ Video format?: ";
                                          cin.getline(spd,10);
                                          fm=spd[0];
                                          if(fm!='m' && fm != 'a')
                                                cout<<setw(30)<<" "<<"-
-Invalid data!!"<<endl;
                                    }while(fm!='m' && fm != 'a');
                                    switch(fm){
                                          case 'm':
                                          strcat(video_in,".mp4");
                                          break;
                                          case 'a':
                                          strcat(video_in,".avi");
                                          break;
                                    }
                                    cout<<endl<<"[*]Enter your video
frames per second rate:"<<endl<<endl;
                                    do{
                                          cout<<setw(14)<<" "<<"[30/60]
Video fps?: ";
                                          cin.getline(spd,15);
                                          fps=atoi(spd);
                                          if(fps!=30 && fps != 60)
```

```
                                                cout<<setw(30)<<" "<<"-
--Invalid data!!"<<endl;
                                    }while(fps!=30 && fps!= 60);
                                    // Set tracker type. Change this to
try different trackers.
                                    string
trackerType=trackerTypes[knd-1];
                                    cv::VideoCapture cap(video_in);
                                    Mat frame;
                                    // quit if unabke to read video
file
                                    if(!cap.isOpened( )) {
                                          cout<<setw(34)<<" "<<"OCELOT:
ERROR - File not found" <<video_in<< endl;
                                          return -1;
                                    }
                                    // read first frame
                                    cap>>frame;
                                    if(frame.rows>1079 || frame.cols >
1919){
                                          cout<<"OCELOT: ERROR - Video
Resolution Exceeded"<<endl;
                                          break;
                                    }
                                    cout<<endl<<endl;
                                    cout<<"Press any key to continue!
... ";
                                    cin.get( );
                                    system("clear");
                                    cout<<endl;
                                    ocelot( );
                                    cout<<endl;
                                    repeat('_',strlen("OCELOT: Vehicle
Tracking- STEP [4]"));
                                    cout<<"OCELOT: Vehicle Tracking-
STEP [4]"<<endl;
                                    repeat('-',strlen("OCELOT: Vehicle
Tracking- STEP [4]"));
                                    repeat('_',60);
                                    cout<<endl;
                                    cout << "Press the [c] key to undo
the selection of the last marked vehicle [X]." << endl;
                                    cout << "Press the [ESC] key to
finish the selection process and start tracking." << endl;
                                     repeat('=',60);
                                     cv::selectROIs("Select
vehicle(s)... Press [ESC] to finish",frame,bboxes,cross, expand);
                                     // quit if there are no objects to
track
                                    if(bboxes.size( )<=0)
                                           return 0;
                                    vector<Scalar> colors;
                                    color_rand(colors, bboxes.size( ));
                                    // Create multitracker
                                    Ptr<MultiTracker> multiTracker =
cv::MultiTracker::create( );
                                    // initialize multitracker
                                    //bboxes.size( ) = total of
selected objects to track
                                    //for(int j=0;j<bboxes.size( );j++)
```

```
                                    destroyWindow("Select vehicle(s)...
Press [ESC] to finish");
                                    for(int j=0;j<x;j++)
                                          multiTracker-
>add(createTrackerByName(trackerType), frame, Rect2d(bboxes[j]));
                                    // process video and track objects
                                    repeat('_',50);
                                    cout<<setw(14)<<" "<<"Trajectory
analysis on-course..." << endl;
                                    cout<<setw(14)<<" "<<"Press [ESC]
key to quit."<<endl;
                                    repeat('-',50);
                                    // get frame from the video
                                    auto
start=chrono::high_resolution_clock::now( );
                                    while(cap.isOpened( )){
                                          cap>>frame;
                                          if(frame.empty( ))
                                                break;
                                          // draw tracked objects
                                          for(unsigned
j=0;j<multiTracker->getObjects( ).size( );j++){
                                                rectangle(frame,
multiTracker->getObjects( )[j], colors[j],2,1);

      cd[j][ck][0]=multiTracker->getObjects( )[j].br( );

      cd[j][ck][1]=multiTracker->getObjects( )[j].tl( );
                                                ck++;
                                          }
                                          // show frame
                                          imshow("Tracking
Vehicle(s)... Press [ESC] to quit.", frame);
                                          // quit on x button
                                          if (waitKey(1) == 27)
                                                break;
                                          if(ck>4998){
                                                cout<<"OCELOT -
Avoiding Stack Oferflow, Exiting..."<<endl;
                                                break;
                                          }
                                    }
                                    vl=float(ck)/(x*fps);
                                    auto
end=chrono::high_resolution_clock::now( );
                                    chrono::duration<double> trek=end-
start;
                                    trekf=float(trek.count( ));
                                    destroyAllWindows( );
                                    system("clear");
                                    cout<<endl;
                                    ocelot( );
                                    cout<<endl;
                                    repeat('_',strlen("OCELOT: Vehicle
Tracking - PROCESSING...."));
                                    cout<<"OCELOT: Vehicle Tracking -
PROCESSING...."<<endl;
                                    repeat('-',strlen("OCELOT: Vehicle
Tracking - PROCESSING...."));
                                    cout<<"Done processing!"<<endl;
                                    cout<<endl<<endl;
```

```
                                    cout<<"Generating
trajectory..."<<endl<<endl;
                                    putText(xy,"OCELOT TRACKING", Point
(20,50), FONT_HERSHEY_SIMPLEX, 1.0, CV_RGB(0,0,0), 1.0);
                                    Point ln[5][ck];
                                    for(int k=0;k<x;k++)
                                          for(int j=0;j<ck/x;j++)

      ln[k][j]=cd[k][(j*x)+k][0]*0.5+cd[k][(j*x)+k][1]*0.5;
                                    for(int j=0;j<x;j++){

      xi[j]=ln[j][0].x;yi[j]=ln[j][0].y;
                                          xe[j]=ln[j][ck-
4].x;ye[j]=ln[j][ck-4].y;
                                          d[j]=sqrt((pow((xe[j]-
xi[j]),2)+pow((ye[j]-yi[j]),2)));
                                          speed[j]=d[j]/10;
                                    }
                                    cout<<"Tracking Results for:
"<<endl<<endl;
                                    for(int j=0;j<x;j++)
                                          cout<<'['<<j+1<<"]:
"<<name[j]<<endl;
                                    cout<<endl<<"Available as TREPORT -
XXXXXX"<<endl;
                                    for(int j=0;j<x;j++)

      putText(xy,name[j],ln[j][0],FONT_HERSHEY_COMPLEX_SMALL,1.0,purpl
e,1.0);
                                    //draw_paths
                                    for(int k=0;k<x;k++)
                                          for(int j=0;j<(ck/x)-1;j++)

      line(xy,ln[k][j],ln[k][j+1],Scalar(50*k,50*k,50*k),2,8);
                                    now_t=time(NULL);
                                    now_s=ctime(&now_t);
                                    strcat(fname,"TREPORT - ");
                                    strcat(pname,"TRACKING - ");
                                    strcat(fname,now_s);
                                    strcat(pname,now_s);
                                    strcat(fname,".txt");
                                    strcat(pname,".jpg");
                                    imwrite(pname,xy);
                                    treport.open(fname);
                                    treport<<'\t'<<'\t'<<'\t'<<"[OCELOT
- TRACKING REPORT]"<<endl<<endl;

      treport<<"_____________________________________________________
_____________________"<<endl;
                                    treport<<"[*]Values from the table
are ready to be exported to MS-EXCEL or LO-CALC"<<endl;

      treport<<"=====================================================
==================="<<endl<<endl;
                                    treport<<"*****"<<endl;

      treport<<"NUMBER"<<'\t'<<"DESCRIPTION"<<'\t'<<"TIME
[s]"<<'\t'<<"DISTANCE [m]"<<'\t'<<"AVG SPEED [Km/H]"<<endl;
                                    for(int j=0;j<x;j++)
```

```
      treport<<j+1<<'\t'<<name[j]<<'\t'<<vl<<'\t'<<d[j]<<'\t'<<speed[j
]<<endl;
                                    treport<<"*****"<<endl;
                                    treport<<endl;
                                    treport<<"Number of tracked
Vehicles: "<<'\t'<<'\t'<<x<<endl;
                                    treport<<"Tracking Algorithm Used:
"<<'\t'<<'\t'<<trackerType<<endl;
                                    treport<<"Total Processing Time:
"<<'\t'<<'\t'<<trekf<<"[s]"<<endl;
                                    treport<<"Operational Cost:
"<<'\t'<<'\t'<<((trekf)/vl)<<endl;
                                    treport<<endl;
                                    //for(int j=0;j<ck;j++)

      //treport<<j+1<<'\t'<<cd[0][j][0].x<<'-
'<<cd[0][j][0].y<<'\t'<<cd[1][j][0].x<<'-
'<<cd[1][j][0].y<<'\t'<<cd[2][j][0].x<<'-'<<cd[2][j][0].y<<endl;
                                    //for(int j=0;j<ck;j++)

      //treport<<j+1<<'\t'<<ln[0][j].x<<'-
'<<ln[0][j].y<<'\t'<<ln[1][j].x<<'-'<<ln[1][j].y<<'\t'<<ln[2][j].x<<'-
'<<ln[2][j].y<<endl;
                                    treport.close( );
                  }
                  break;
                  case 'c':{
                        // Load names of classes
                        string klassf="ocelot_cfg/coco.names";
                        ifstream ifs(klassf.c_str( ));
                        string line;
                        while (getline(ifs,line))
                              classes.push_back(line);
                        String
net=readNetFromDarknet(ocelot_cfg,ocelot_weights);
                        net.setPreferableBackend(DNN_BACKEND_OPENCV);
                        net.setPreferableTarget(DNN_TARGET_CPU);
                        String str,outputFile;

                        VideoWriter video;
                        Mat frame, blob;
                        system("clear");
                        cout<<endl;
                        ocelot( );
                        cout<<endl;
                        repeat('_',strlen("OCELOT: Vehicle Count- STEP
[1]"));
                        cout<<"OCELOT: Vehicle Count- STEP [1]"<<endl;
                        repeat('-',strlen("OCELOT: Vehicle Count- STEP
[1]"));
                        cout<<setw(14)<<" "<<"File name?: ";
                        cin.getline(video_in,30);
                        str+=video_in;
                        cout<<endl;
                        cout<<"[*]Enter your video
format:"<<endl<<endl;
                        do{
                              cout<<setw(14)<<" "<<"/[m]MP4-[a]AVI/
Video format?: ";
                              cin.getline(spd,10);
```

```
                              fm=spd[0];
                              if(fm!='m' && fm != 'a')
                                    cout<<setw(30)<<" "<<"Invalid
parameter"<<endl;
                        }while(fm!='m' && fm != 'a');
                        switch(fm){
                              case 'm':
                              str+=".mp4";
                              break;
                              case 'a':
                              str+=".avi";
                              break;
                        }
                        cout<<endl<<endl;
                        cout<<"Press any key to continue! ... ";
                        cin.get( );
                        system("clear");
                        cout<<endl;
                        ocelot( );
                        cout<<endl;
                        repeat('_',strlen("OCELOT: Vehicle Count- STEP
[2]"));
                        cout<<"OCELOT: Vehicle Count- STEP [2]"<<endl;
                        repeat('-',strlen("OCELOT: Vehicle Count- STEP
[2]"));
                        cout<<endl;
                        cout<<"Enter counting balance"<<endl<<endl;
                        cout<<setw(15)<<" ";repeat('=',strlen("[3]
Improved processing accuracy"));
                        cout<<setw(15)<<" "<<"[1] Improved processing
speed"<<endl;
                        cout<<setw(15)<<" "<<"[2] Balanced
processing"<<endl;
                        cout<<setw(15)<<" "<<"[3] Improved processing
accuracy"<<endl;
                        cout<<setw(15)<<" ";repeat('=',strlen("[3]
Improved processing accuracy"));
                        cout<<endl;
                        do{
                              cout<<setw(14)<<" "<<"Set balance: ";
                              cin.getline(spd,10);
                              fg=atoi(spd);
                              if(!(fg>0 && fg<4))
                                    cout<<setw(30)<<" "<<"Invalid
balance parameter"<<endl;
                        }while(fg<1 || fg >3);
                        switch(fg){
                              case 1:
                              gridx=gridy=320;
                              break;
                              case 2:
                              gridx=gridy=416;
                              break;
                              default:
                              gridx=gridy=608;
                              break;
                        }
                        cout<<endl<<endl;
                        cout<<"Press any key to continue! ... ";
                        cin.get( );
                        system("clear");
```

```
                        cout<<endl;
                        ocelot( );
                        cout<<endl;
                        repeat('_',strlen("OCELOT: Vehicle Counting -
PROCESSING...."));
                        cout<<"OCELOT: Vehicle Counting -
PROCESSING...."<<endl;
                        repeat('-',strlen("OCELOT: Vehicle Counting -
PROCESSING...."));
                        cout<<endl;
                        ifstream ifile(str);
                        if (!ifile)
                              throw("error");
                        cap.open(str);
                        str.replace(str.end( )-4, str.end( ),
"_ocelot.avi");
                        outputFile=str;
                        //video.open(outputFile,
VideoWriter::fourcc('M','J','P','G'), 28,
Size(cap.get(CAP_PROP_FRAME_WIDTH), cap.get(CAP_PROP_FRAME_HEIGHT)));
                        // Create a window
                        static const string kWinName = "Counting
Vehicles... Press [ESC] to quit";
                        //execution time
                        auto start=chrono::high_resolution_clock::now(
);
                        namedWindow(kWinName, WINDOW_AUTOSIZE);
                        // Process frames.
                        while (waitKey(1) < 0){
                              // get frame from the video
                              cap>>frame;
                              // Stop the program if reached end of
video
                              if (frame.empty( )) {
                                    cout << "OCELOT - Vehicle Counting:
Done!"<< endl;
                                    cout << "OCELOT - Results saved as:
" <<outputFile<< endl;
                                    waitKey(3000);
                                    break;
                              }
                              // Create a 4D blob from a frame.
                              blobFromImage(frame, blob, 1/255.0,
cv::Size(gridx,gridy),Scalar(0,0,0), 1, 0);
                              //Sets the input to the network
                              net.setInput(blob);
                              // Runs the forward pass to get output of
the output layers
                              vector<Mat> outs;
                              net.forward(outs, getOutputsNames(net));
                              // Remove the bounding boxes with low
confidence
                              postprocess(frame, outs);
                              // Put efficiency information. The
function getPerfProfile returns the overall time for inference(t) and
the timings for each of the layers(in layersTimes)
                              vector<double> layersTimes;
                              double freq=getTickFrequency( )/1000;
                              double
t=net.getPerfProfile(layersTimes)/freq;
```

```
                                string eff_txt=format("Inference time for
a frame : %.2f ms", t);
                                putText(frame, eff_txt,Point(0,
15),FONT_HERSHEY_SIMPLEX,0.5,Scalar(0, 0, 255));
                                // Write the frame with the detection
boxes
                                Mat detectedFrame;
                                frame.convertTo(detectedFrame,CV_8U);
                                video.write(detectedFrame);
                                imshow(kWinName,frame);
                          }
                          //execution time
                          auto end=chrono::high_resolution_clock::now( );
                          chrono::duration<double> trek=end-start;
                          trekf=float(trek.count( ));
                          cout<<"OCELOT - Execution time: "<<trekf<<"
[s]"<<endl;
                          cout<<"OCELOT - Total FPS processed:
"<<ck<<endl;
                          now_t=time(NULL);
                          now_s=ctime(&now_t);
                          strcat(fname,"CREPORT - ");
                          strcat(fname,now_s);
                          creport.open(fname);
                          creport<<'\t'<<'\t'<<'\t'<<'\t'<<"[OCELOT
REPORT]"<<endl<<endl;

creport<<"_____________________________________________________________
_____________"<<endl;
                          creport<<"[*]Values from the table are ready to
be exported to MS-EXCEL or LO-CALC"<<endl;

creport<<"=============================================================
============"<<endl<<endl;
                          creport<<"***"<<endl;

      creport<<"VEHICLE"<<'\t'<<"TOTAL"<<'\t'<<"AVG"<<endl;
                          creport<<"Car"<<'\t'<<car<<'\t'<<car/ck<<endl;

      creport<<"Bike"<<'\t'<<bike<<'\t'<<bike/ck<<endl;

      creport<<"Truck"<<'\t'<<truck<<'\t'<<truck/ck<<endl;
                          creport<<"Bus"<<'\t'<<bus<<'\t'<<bus/ck<<endl;
                          creport<<"***"<<endl;
                          creport<<endl;
                          creport<<"Execution time:
"<<'\t'<<'\t'<<trekf<<" [s]"<<endl;
                          creport<<"Estimated Video lenght:
"<<'\t'<<'\t'<<ck/30<<"[s]"<<endl<<endl;
                          creport<<"q=N/T [u/sg]"<<endl<<endl;
                          creport<<"Cars Traffic Stream:
"<<'\t'<<'\t'<<car/ck<<" [u/sg]"<<endl;
                          creport<<"Bikes Traffic Stream:
"<<'\t'<<'\t'<<bike/ck<<" [u/sg]"<<endl;
                          creport<<"Trucks Traffic Stream:
"<<'\t'<<'\t'<<truck/ck<<" [u/sg]"<<endl;
                          creport<<"Buses Traffic Stream:
"<<'\t'<<'\t'<<bus/ck<<" [u/sg]"<<endl;
                          creport<<"TRAFFIC STREAM:
"<<'\t'<<'\t'<<(car+bike+bus+truck)/ck<<" [u/sg]"<<endl;
                          creport.close( );
```

```
                        cap.release( );
                        video.release( );
                  }
                  break;
                  case 'a':{
                        cout<<setw(14)<<" "<<"Not available
yet."<<endl;
                  }
                  break;
                  case 'q':{
                        system("clear");
                        cout<<endl;
                        ocelot( );
                        cout<<endl;
                        repeat('_',strlen("OCELOT: Quick reference -
Pg[1]"));
                        cout<<"OCELOT: Quick reference - Pg[1]"<<endl;
                        repeat('-',strlen("OCELOT: Quick reference -
Pg[1]"));
                        cout<<endl;
                        cout<<"OCELOT an app written in C++ for off-
line traffic analysis."<<endl;
                        cout<<"Here is a brief description of its two
main features."<<endl<<endl;
                        cout<<"Press any key to continue! ... ";
                        cin.get( );
                        system("clear");
                        repeat('_',strlen("OCELOT: Quick reference -
Pg[2]"));
                        cout<<"OCELOT: Quick reference - Pg[2]"<<endl;
                        repeat('-',strlen("OCELOT: Quick reference -
Pg[2]"));
                        cout<<endl;
                        cout<<"[*] VEHICLE TRACKING"<<endl<<endl;
                        cout<<"Allows you to track as much as 5
vehicles selected from the inicial frame of"<<endl;
                        cout<<"your sampling video. Trajectory
described by each vehicle will be drawn and"<<endl;
                        cout<<"storaged. "<<endl;
                        cout<<"This a semi - automated traffic analysis
tool, you will need to to tell Ocelot"<<endl;
                        cout<<"when to stop tracking by pressing the
[ESC] key."<<endl;
                        cout<<"Outputs: "<<endl<<endl;
                        cout<<"- Plain text report."<<endl;
                        cout<<"- Jpeg image(s) file(s)."<<endl<<endl;
                        cout<<"Press any key to continue! ... ";
                        cin.get( );
                        system("clear");
                        repeat('_',strlen("OCELOT: Quick reference -
Pg[3]"));
                        cout<<"OCELOT: Quick reference - Pg[3]"<<endl;
                        repeat('-',strlen("OCELOT: Quick reference -
Pg[3]"));
                        cout<<endl;
                        cout<<"[*] VEHICLE COUNT"<<endl<<endl;
                        cout<<"Allows you to obtain the average traffic
stream from your sampling video. Four"<<endl;
                        cout<<"kinds of vehicles are recognized: Cars,
Motorcycles, Buses & Trucks. "<<endl;
```

```
                        cout<<"This is a fully-automated traffic
analysis tool. However, you can tell Ocelot"<<endl;
                        cout<<"when to stop processing by pressing the
[ESC] key."<<endl;
                        cout<<"Outputs: "<<endl<<endl;
                        cout<<"- Plain text report."<<endl;
                        cout<<"- Avi video file."<<endl<<endl;
                        cout<<"Press any key to continue! ... ";
                        cin.get( );
                  }
                  break;
            }
      cout<<endl<<endl;
      do{
            cout<<setw(15)<<" "<<"Exit Ocelot [y/n]?: ";
            cin.getline(spd,10);
            ans=spd[0];
            }while(ans!='n' && ans!='y');
            if(ans!='y'){
            for(int i=0;i<5;i++)
                  for(int j=0;j<30;j++)
                        name[i][j]=0;
            for(int j=0;j<40;j++){
                  fname[j]=0;
                  pname[j]=0;
                  }
            car=bike=truck=bus=trekf=ck=0;
            for(int j=0;j<5;j++)
                  xi[j]=xe[j]=yi[j]=ye[j]=d[j]=speed[j]=0;
            }
      }while(ans!='y');
      system("clear");
      bye( );
      cout<<"Press any key to exit! ... ";
      cin.get( );
      system("clear");
}
```

ANEXO B: INSTALACIÓN DE OPENCV 3.4.2 EN UBUNTU 16.4.05 LTS

Paso 1: Actualización de los paquetes de Ubuntu.

```
sudo apt -y update
sudo apt -y upgrade
```

Paso 2: Eliminación de cualquier paquete previamente instalado de **libx264**.

```
sudo apt -y remove x264 libx264-dev
```

Paso 3: Instalación de librerías requeridas por el sistema.

```
//dependencias principales
sudo apt -y install build-essential checkinstall cmake pkg-config yasm
sudo apt -y install git gfortran
sudo apt -y install libjpeg8-dev libjasper-dev libpng12-dev

sudo apt -y install libtiff5-dev

sudo apt -y install libtiff-dev

sudo apt -y install libavcodec-dev libavformat-dev libswscale-dev libdc1394-22-dev
sudo apt -y install libxine2-dev libv4l-dev
cd /usr/include/linux
sudo ln -s -f ../libv4l1-videodev.h videodev.h
cd $cwd

sudo apt -y install libgstreamer0.10-dev libgstreamer-plugins-base0.10-dev
sudo apt -y install libgtk2.0-dev libtbb-dev qt5-default
sudo apt -y install libatlas-base-dev
sudo apt -y install libfaac-dev libmp3lame-dev libtheora-dev
sudo apt -y install libvorbis-dev libxvidcore-dev
sudo apt -y install libopencore-amrnb-dev libopencore-amrwb-dev
sudo apt -y install libavresample-dev
sudo apt -y install x264 v4l-utils

//dependencias opcionales
sudo apt -y install libprotobuf-dev protobuf-compiler
sudo apt -y install libgoogle-glog-dev libgflags-dev
sudo apt -y install libgphoto2-dev libeigen3-dev libhdf5-dev doxygen
```

Paso 4: Descarga de OpenCV y OpenCV Contrib desde Github.

```
git clone https://github.com/opencv/opencv.git
cd opencv
git checkout 3.4
cd ..

git clone https://github.com/opencv/opencv_contrib.git
cd opencv_contrib
git checkout 3.4
cd ..
```

Paso 5: Creación del directorio build para Cmake.

```
cd opencv
mkdir build
cd build
```

Paso 6: Compilación de Cmake.

```
cmake -D CMAKE_BUILD_TYPE=RELEASE \
            -D CMAKE_INSTALL_PREFIX=/usr/local \
            -D INSTALL_C_EXAMPLES=ON \
            -D WITH_TBB=ON \
            -D WITH_V4L=ON \
        -D WITH_QT=ON \
        -D WITH_OPENGL=ON \
        -D OPENCV_EXTRA_MODULES_PATH=../../opencv_contrib/modules \
        -D BUILD_EXAMPLES=ON ..
```

Paso 7: Construcción de Cmake en base a los parámetros previamente establecidos.

```
make -j4
make install
```

ANEXO C: COMANDOS DE COMPILACIÓN Y EJECUCIÓN

```
//compilación – dentro de la carpeta del archivo

g++ -std=c++11 my_program.cpp `pkg-config --libs --cflags opencv` -o
my_program

//ejecución – dentro de la carpeta del archivo

./my_program
```