CENTRO DE INGENIERÍA Y DESARROLLO INDUSTRIAL

REPORTE DE TESIS

“Sistema de Aprendizaje Automático para Detección y Análisis de Tráfico Vehicular”

ESPECIALIDAD DE TECNÓLOGO EN MECATRÓNICA

PRESENTA
Ing. Oscar Ivvan Capistran Olvera

Tutor Académico
Dr. Leonardo Barriga Rodríguez

Abril 2020, Querétaro, Querétaro

Contenido

1. Introducción 1

2. Planteamiento del Problema 3

3. Objetivos 4

3.1 Objetivo General 4

3.2 Objetivos Específicos 4

4. Justificación 5

5. Antecedentes 8

6. Marco Teórico 9

6.1 Visión por Computadora 9

6.1.1 Detección de Objetos por Eliminación de Fondo. 9

6.1.2 Detección de Objetos por medio de clasificadores Haar. 11

6.1.3 Detección de Objetos utilizando YOLO (You Only Look Once) 12

6.2 Seguimiento de Objetos 13

6.2.1 Seguimiento de Objetos por medio de SORT. 14

6.3 Aprendizaje Automático 15

6.3.1 Aprendizaje Supervisado 15

6.3.2 Aprendizaje No Supervisado 16

7. Metodología 17

7.1 Detección de Vehículos mediante Eliminación de Fondo. 19

7.1.1 Algoritmo de Eliminación de Fondo 20

7.1.2 Threshold. 22

7.1.3 Eliminación de Ruido 25

7.1.4 Detección de Contornos..... 27
7.2 Detección de Objetos por medio de Características Haar. 28
7.3 Detección de Objetos por medio de YOLO. 30
7.4 Seguimiento de Objetos 32
7.5 Aprendizaje Automático 32
8. Resultados..... 33
8.1 Resultados obtenidos al utilizar el método de Eliminación de Fondo: 33
8.2 Resultados obtenidos al utilizar el método de Características Haar..... 34
8.3 Resultados obtenidos al utilizar el método de YOLO. 36
8.4 Resultados obtenidos al utilizar YOLO y SORT. 37
8.5 Mapeo de puntos de los objetos detectados. 38
9. Conclusiones..... 41
10. Bibliografía 42
11. Anexos 46
11.1 Código de Detección de Objetos por Eliminación de Fondo: 46
11.2 Código de Detección de Objetos por Medio de Características Haar: 48
11.3 Código de Detección de Objetos por medio de YOLO y SORT: 49

Índice de Ilustraciones

Ilustración 1. Ejemplo de detección de objetos mediante la eliminación de fondo. 10
Ilustración 2. Ejemplos de características Haar usadas [16] 11
Ilustración 3. Reconocimiento de características Haar en una imagen. Fuente [9] 12
Ilustración 4. Comparación de MOG (imagen izquierda) vs MOG2 (Imagen derecha). 20
Ilustración 5. Eliminación de fondo usando algoritmo MOG2 sin entrenamiento. 21
Ilustración 6. Eliminación de fondo usando algoritmo MOG2 con entrenamiento. 22
Ilustración 7. Simple Thresholding. 23
Ilustración 8. Adaptative Thresholding.– 24
Ilustración 9. Gaussian Threshold. 24
Ilustración 10. Otsu Threshold. 25
Ilustración 11. Forma: Elipse (1,2), Closing (3), Opening (2), Dilatation (2) 27
Ilustración 12. Forma: Elipse (1,2), Opening (1), Closing (2), Dilatation (2) 27
Ilustración 13. Detección de vehículos por medio de Eliminación de Fondo. 34
Ilustración 14. Detección de vehículos por medio de Características Haar. 35
Ilustración 15. Detección de vehículos por medio de YOLO. 36
Ilustración 16. Detección y seguimiento de objetos. 37
Ilustración 17. Mapeo de puntos detectados con una muestra de 100 vehículos sin expandir. 38
Ilustración 18. Mapeo de puntos detectados con una muestra de 100 vehículos expandido. 38
Ilustración 19. Mapeo de puntos detectados con una muestra de 500 vehículos sin expandir. 39
Ilustración 20. Mapeo de puntos detectados con una muestra de 500 vehículos expandido. 39
Ilustración 21. Mapeo de puntos detectados con una muestra de 1000 vehículos sin expandir. .. 40

Índice de Graficas

Grafica 1. Grafica de vehículos registrados en México de 1980-2017..... 5
Grafica 2. Número de Accidentes Vehiculares Registrados basado en datos del INEGI..... 7

Índice de Tablas

Tabla 1. Demanda de Petrolíferos 2011-2018 6

Introducción

El monitoreo del tráfico vehicular ha sido un tema de interés muy frecuente, esto se debe a que hoy en día es cada vez más común para las personas el tener un modo de transportarse de un lugar a otro de manera rápida y eficiente, si a esto se le añade también el incremento acelerado de la población, resulta en un aumento descontrolado del tráfico de vehículos en las calles de las ciudades y carreteras del país, el cual ha ido empeorando día con día.

Debido a la gran cantidad de vehículos que circulan las diferentes calles y carreteras en México se han propuesto diversas leyes y reformas para intentar disminuir y controlar de una mejor manera el flujo vehicular en diversos estados, como ejemplo, solo en el estado de Querétaro se pueden encontrar cámaras de vigilancia las cuales se encargan de vigilar diversos puntos del estado, sin embargo no es una tarea fácil el monitorear todas estas cámaras en tiempo real y alertar cuando sucede algún accidente o se comete una infracción al reglamento vehicular.

Sin embargo, la importancia de la vigilancia vehicular no es exclusivamente para reducir la cantidad de infracciones o violaciones que cometen los vehículos al circular por las calles. Como menciona, Querétaro se ha convertido en un punto importante de movimiento de carga gracias a que se encuentra cerca del cruce de los ejes carreteros que cruzan el centro-norte y centro-occidente del país, aunado a esto se encuentra la rápida expansión del municipio de Querétaro en las últimas décadas lo cual ha ocasionado grandes cambios en la infraestructura vial del municipio. Con esto podemos ver que también es importante la vigilancia vehicular para poder mejorar el manejo del tráfico y al mismo tiempo el impulsar la economía del estado al reducir la cantidad de tiempo que se desperdicia por culpa de accidentes, embotellamientos y otros tipos de percances vehiculares los cuales pueden impedir o ralentizar el flujo vehicular. [1][2]

Es debido a esto por lo cual este proyecto se basa en la creación de un software el cual sea capaz de detectar los diferentes tipos de vehículos que circulan con ayuda de librerías ya existentes de

detección de objetos. Además de esto se planea incluir en el software un sistema de aprendizaje automático el cual se encargará de analizar los patrones de movimiento de los vehículos detectados y de esta manera determinar un patrón de conducta en base a los datos recolectados el cual será utilizado para detectar todo aquel vehículo el cual actúe fuera de los estándares aprendidos por el sistema y mandar una señal de alerta avisando de esta manera cual fue el vehículo en cuestión el que realizó esta acción.

2. Planteamiento del Problema

En la actualidad para diferentes autoridades como las Estatales y Municipales, es muy necesario el monitoreo de sistemas de vigilancia de tránsito para identificar diferentes situaciones referentes a la falta de cumplimiento de las normas y leyes vehiculares por parte de los conductores, lo cual puede llegar a ocasionar tráfico o accidentes vehiculares. Sin embargo, debido al limitado personal que se encuentra monitoreando la gran cantidad de cámaras de seguridad en las ciudades, no es posible vigilar cada una de las calles monitoreadas.

Otra problemática es que cada una de las cámaras de vigilancia presenta una perspectiva y calidad de imagen diferentes, por lo cual es necesario crear un algoritmo el cual no se vea limitado por estas situaciones y pueda aprender en relación con la escena que ve la cámara.

3. Objetivos

3.1 Objetivo General

Desarrollar un software el cual integre algoritmos para detección de objetos, seguimiento de objetos y análisis de datos, los cuales al trabajar en conjunto sean capaces de detectar y seguir diferentes clases de vehículos y, además, mediante el uso de Machine Learning o Aprendizaje Automático, se analicen los datos recolectados para crear un modelo de comportamiento del flujo de circulación de los vehículos detectados.

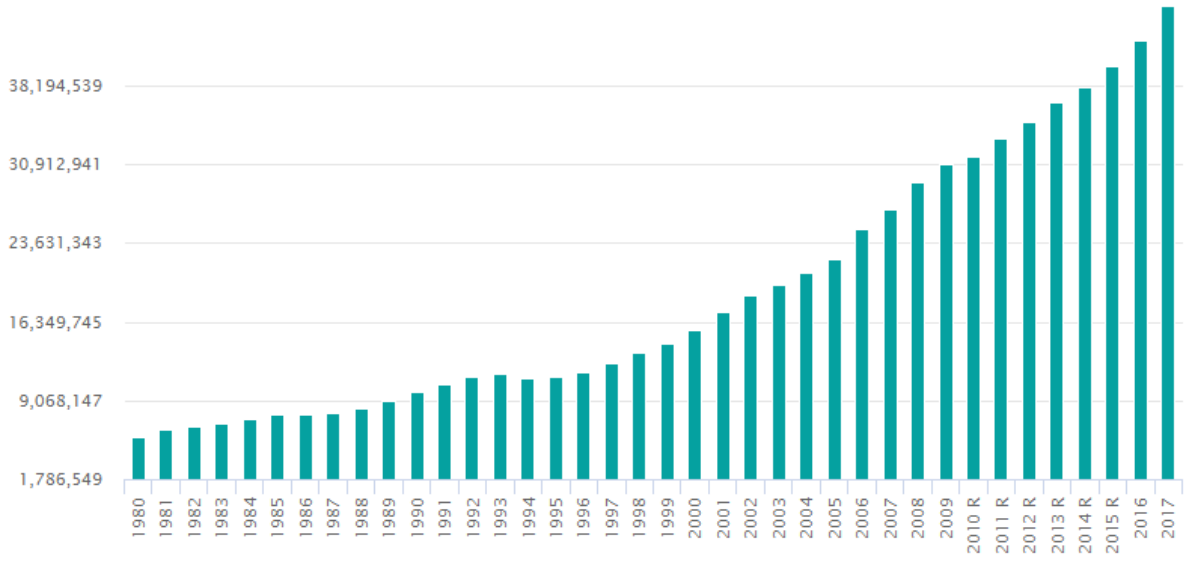
3.2 Objetivos Específicos

1 Analizar diferentes modelos de detección de objetos y seleccionar aquel que mejor se ajuste a los requerimientos del proyecto.
2 Implementar al software a desarrollar la funcionabilidad de diferenciar entre diversos tipos de vehículos.
3 Seleccionar e implementar un algoritmo de detección de objetos el cual rastree los objetos detectados y guarde la posición en la cual fue detectado.
4 Seleccionar e implementar un modelo de Aprendizaje Automático el cual analice los datos recolectados y los use para crear un modelo de comportamiento de flujo.

4. Justificación

Este proyecto fue pensado como una solución a una de las problemáticas que se han vuelto más comunes en la reciente década, la automatización de la vigilancia vehicular. Debido a la cantidad de vehículos circulando por las numerosas calles y avenidas y a la gran cantidad de cámaras de vigilancia vehicular se ha vuelto más complicado el vigilar cada una de las cámaras debidamente, por la cual muchos conductores aprovechan para violar las leyes vehiculares y de conducción en busca de llegar a su lugar de destino de manera más rápida lo cual puede llegar a terminar en accidentes o crear problemas en el flujo de los vehículos que transitan sabiendo que a pesar de que estén siendo vigilados no habrá repercusiones.

Según estudios del INEGI el número de vehículos registrados en el 2017 fue de más de 45,000,000 incluyendo Automóviles, Camiones y camionetas para pasajeros, Camiones para carga y Motocicletas, así como el tipo de servicio Oficial, Público y Particular, mientras que a finales del 2018 se registraron poco más de 43,000,000 sin incluir motocicletas ni vehículos de servicio Oficial. Esto significa que a partir del 2010 hubo un incremento en el uso de vehículos del 43.75%. (Grafica 1) [3]



Grafica 1. Grafica de vehículos registrados en México de 1980-2017

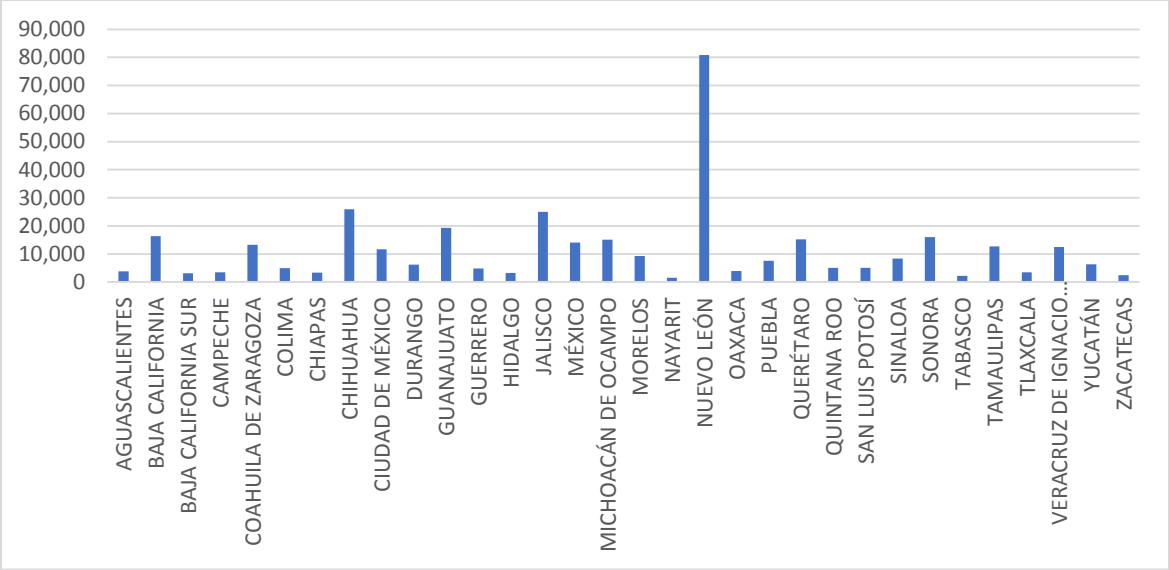
Con el aumento de vehículos en circulación también se puede observar un aumento en el consumo de combustible necesario, un estudio de la Secretaria de Energía (SENER) nos dice que para el mes de Septiembre del 2018 se registró un consumo de gasolina total de 789 Mbd mientras que el consumo de Diesel fue de solo 389 Mbd como se puede observar en la Tabla 1.

Aun cuando estas cantidades son menores a las registradas en años anteriores debido a la mejora en las tecnologías usadas para la creación de vehículos, el uso de vehículos híbridos o eléctricos, entre otras razones, se podrían reducir aún más al mejorar de manera eficiente el flujo vehicular en las ciudades y carreteras.

Año	2011	2012	2013	2014	2015	2016	2017	2018
Demanda (Mbd)	**1,439**	**1,477**	**1,430**	**1,353**	**1,361**	**1,405**	**1,405**	**1,406**
Gasolinas	799	803	787	776	794	823	799	789
Diésel	384	400	392	389	385	387	385	389
Turbosina	56	59	62	67	71	76	82	87
Combustóleo	201	214	190	122	111	118	140	141

Tabla 1. Demanda de Petrolíferos 2011-2018

Además, otro estudio del INEGI nos muestra que en el 2018 se registraron más de 365,000 accidentes de tránsito a nivel nacional de los cuales 15,185 ocurrieron en el estado de Querétaro con lo que se posiciona en el séptimo lugar de más accidentes de tráfico registrados a nivel nacional, que a comparación de años anteriores este número se ha ido reduciendo considerablemente. [4]



Grafica 2. Número de Accidentes Vehiculares Registrados basado en datos del INEGI.

CIDESI actualmente ha incursionado en la realización de proyectos en el área de ciudades inteligentes tanto para la ciudad de Querétaro como para la CDMX, por lo que las aportaciones en algoritmos y software en este sentido contribuyen a contar con una mejor capacidad para enfrentar estos retos.

5. Antecedentes

La vigilancia vehicular ha sido un tema de interés en las décadas pasadas surgiendo así diversos métodos a implementar como por ejemplo el uso de sensores, rastreo de vehículos por GPS u otro tipo de tecnologías celulares, el uso de cámaras de vigilancia en carreteras y caminos, etc., sin embargo, en este documento solo nos enfocaremos en aquellos los cuales hagan uso de Sistemas de Visión Computarizada. Sin embargo, este proyecto no se basa simplemente en detección de objetos, también es importante el mantener detectado un objeto lo mejor posible hasta el momento en el que este deje de ser detectado por lo que también es necesario el Seguimiento de Objetos, el cual, a pesar de ser parecido a la Detección de Objetos, es capaz de etiquetar un objeto y mantener dicha etiqueta en él hasta que la detección falle o el objeto salga fuera del margen de detección (fuera de la vista de la cámara).

Existen diferentes tipos de sistemas de detección de vehículos, por ejemplo, existen aquellos sistemas los cuales incluyen cámaras de vigilancia en los vehículos para para poder detectar la distancia entre los vehículos, detectar peatones u obstáculos que se atraviesen en el camino, etc. [5][6]

Otro claro ejemplo de detección de vehículos son aquellos programas los cuales hacen uso de las cámaras de vigilancia en las ciudades ya sea para contar los vehículos que circulan a través de las calles o para detectar si hay atascos de tráfico o predecir el camino que un vehículo seguirá. [7]

Sin embargo, ninguno de los programas anteriores es capaz de utilizar sistemas de aprendizaje automático para aprender de los vehículos en circulación para establecer modelos de comportamiento y de esta manera detectar cuando algún vehículo realice una acción fuera de lo común.

6. Marco Teórico

6.1 Visión por Computadora

La Visión Computarizada es una disciplina de la ingeniería la cual se utiliza para obtener información que sea de utilidad a partir de imágenes capturadas. La forma de realizarlo es por medio del uso de diferentes técnicas para adquirir, procesar, analizar y comprender la información que se tiene en una imagen. [8] [9]

Uno de los atributos más usados en los sistemas de visión por computadora es la detección de objetos. Existen diferentes métodos, algoritmos y programas disponibles para lograr este fin, entre los cuales destacan los métodos de eliminación de fondo, la detección por medio de características de los objetos y, más recientemente, el uso de redes neuronales o sistemas de aprendizaje. De los tres métodos mencionados anteriormente, el método de eliminación de fondo es el único que no requiere de un entrenamiento previo para determinar cuáles objetos debe detectar.

A continuación, veremos una descripción general del funcionamiento de cada uno de los métodos, así como las ventajas y desventajas que cada uno de estos presentan.

6.1.1 Detección de Objetos por Eliminación de Fondo.

Este método de detección de objetos es uno de los más famosos debido a la simplicidad en la que trabaja y debido a que este método es capaz de detectar los objetos moviéndose con precisión. El funcionamiento de este método es el proporcionar o generar cual es el fondo de la imagen a detectar, por ejemplo, si la cámara a utilizar es estática es posible el obtener una imagen de la escena capturada que no contenga ninguno de los objetos a detectar, en el caso contrario el programa debe de ser capaz de generar e identificar cual es el fondo de la imagen de los objetos que se mueven a través de esta. Ambos métodos realizan un escaneo de cada pixel en una imagen y lo comparan con el modelo de fondo generado o proveído, de esta manera el programa diferencia los objetos que se mueven en la imagen de aquellos que son fijos. Sin embargo, este método presenta diversos problemas, como por ejemplo que el programa no es capaz de

diferenciar entre los objetos detectados, además de que los cambios de luz y el más leve cambio en la posición de la imagen puede afectar la manera de detección. [10] [11] [12]

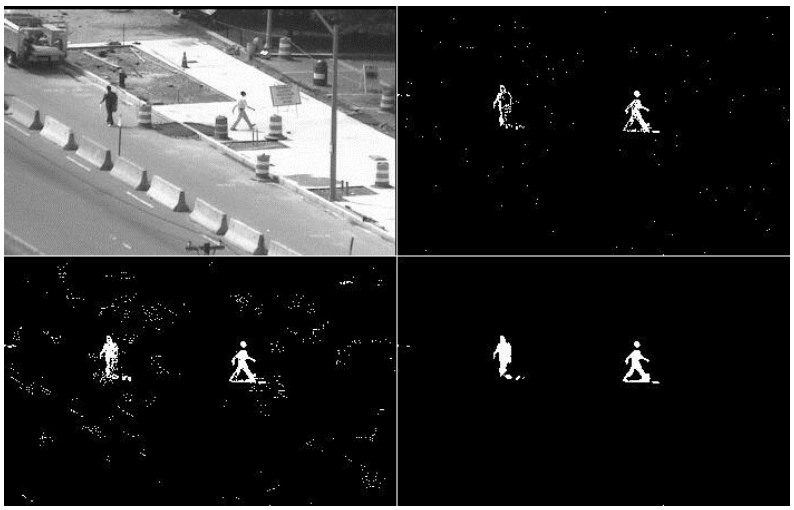


Ilustración 1. Ejemplo de detección de objetos mediante la eliminación de fondo.

En la Ilustración 1 se observa en la parte superior izquierda la imagen original antes de pasar por el algoritmo de detección de objetos. En la imagen superior derecha se muestra el resultado tras comparar la imagen original con la imagen previamente definida como el fondo de la imagen, en esta se puede observar que el detector ha encontrado varios sujetos sin embargo uno de ellos no se encuentra completamente definido. En la imagen inferior izquierda se muestra el resultado obtenido después de haber pasado la imagen detectada por medio de diversos filtros utilizados con el fin de definir más la imagen, sin embargo, esto causa también la creación de ruido blanco que puede interferir con la detección por lo cual es necesario implementar un filtro de eliminación de ruido, el resultado de esto es la imagen inferior derecha en la cual se puede apreciar una imagen más definida de los objetos detectados y con una cantidad de ruido significativamente menor que en las imágenes anteriores.

6.1.2 Detección de Objetos por medio de clasificadores Haar.

La detección de objetos por medio de clasificadores Haar consiste principalmente en el uso de cascadas de reconocimiento de las características Haar para aprender a detectar los bordes y esquinas de un tipo de objeto en específico. Las características Haar son obtenidas al utilizar un par de formas geométricas rectangulares adyacentes del mismo tamaño para obtener la diferencia en el contraste entre ambos, esta diferencia es obtenida sumando el contraste de cada uno de los pixeles contenidos en cada uno de los rectángulos y luego simplemente se obtiene la diferencia entre ambos. Este proceso es realizado en toda la imagen moviendo los rectángulos usados píxel por píxel y volviendo a analizar la imagen cambiando el tamaño y la forma de los rectángulos usados, de esta forma se puede lograr detectar objetos de diferentes tamaños y formas tamaño. Finalmente se aplica un filtro el cual limita el valor mínimo para eliminar la posibilidad de obtener ruido. [13] [14] [15]

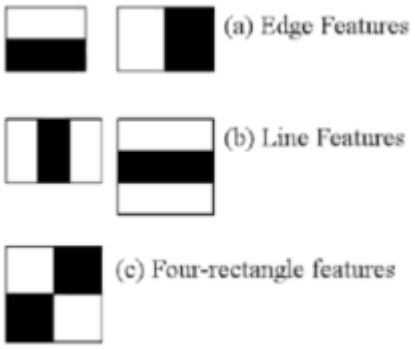


Ilustración 2. Ejemplos de características Haar. [16]

Este tipo de detector de objetos tiene la ventaja de que la detección llega a ser más precisa, en comparación con método de eliminación de fondo, debido a que el método Haar solo identificará aquellos objetos que se quieran detectar. La razón de esto es que se necesita realizar un entrenamiento previo utilizando cientos o miles de imágenes que contengan solo los objetos que sean necesarios, lo cual reduce significativamente la cantidad de objetos que son detectados a la hora de ejecutar el algoritmo. Sin embargo, una de las desventajas de utilizar este método reside en el hecho de que el uso de este algoritmo es susceptible a los cambios en la iluminación de la

imagen a analizar, ya que esta puede llegar a ocasionar falsas detecciones o que sea incapaz de detectar los objetos. Esto se debe principalmente a diversos factores relacionados con las imágenes utilizadas durante el entrenamiento (calidad de la imagen utilizada, contraste, luminosidad, posición del objeto, entre otras) establecen cuales son los patrones que definen las características más prominentes de cada una de las clases de objetos y, una vez que se obtengan estos patrones de clase, utilizar los patrones obtenidos durante el entrenamiento para poder identificar en cualquier otra imagen para ver si existe una coincidencia entre estos y la imagen. [17] [18]

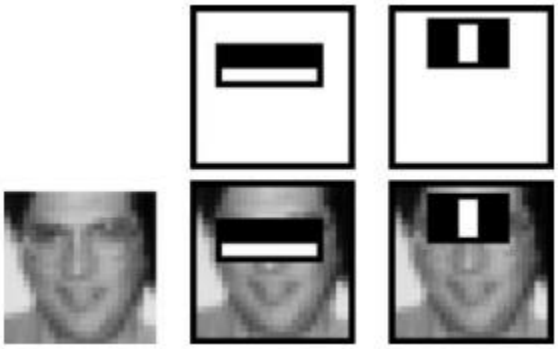


Ilustración 3. Reconocimiento de características Haar en una imagen. Fuente [19]

6.1.3 Detección de Objetos utilizando YOLO (You Only Look Once)

YOLO es un método de detección y clasificación de objetos el cual involucra el uso de redes neuronales convolucionales, o CNN por sus siglas en inglés (Convolutional Neuronal Network), para realizar la detección. A comparación de otros tipos de métodos los cuales utilizan redes neuronales convolucionales, YOLO solo requiere una red neuronal de una sola capa, en lugar de utilizar diversas capas como en el caso del sistema RCNN (Region with Convolutional Neural Network) y sus derivados, lo cual hace que el funcionamiento de este sea mucho más rápido. El funcionamiento de YOLO es el siguiente, cada cuadro del video es dividido en una cuadricula de tamaño SxS, donde cada una de las regiones divididas es sometida a un análisis a la red neuronal, en donde cada neurona de la única capa corresponde a cada miembro de las clases en las cuales haya sido entrenado previamente el sistema, esto con el fin de saber a qué clase pertenece cada

objeto detectado en cada una de las celdas, que tan exacto es la clasificación detectada y al mismo tiempo determina el recuadro de detección de cada uno de los objetos. [19] [20] [21] [22] [23] [24]

Las ventajas que presenta este método de detección es la velocidad y la precisión en la detección y clasificación de los objetos a detectar siempre y cuando estos se encuentren dentro de las clases en las cuales haya sido entrenado el programa, esto se logra ya que el sistema le otorga un valor de exactitud a cada objeto detectado el cual corresponde a las clases con las cuales encuentre similitudes, por lo cual es posible reducir el número de detecciones incorrectas al ajustarse el valor límite de la exactitud que se desea.

La desventajas de este método pueden ser varias, una de las cuales involucra la parte de entrenamiento, ya que de realizarse un entrenamiento con un juego de imágenes pobres, es probable que la detección falle, además de esto, si la cantidad de clases utilizadas en el entrenamiento es muy grande, esto puede ocasionar que el sistema se tarde más en realizar la detección, por lo cual se recomienda que se realice un entrenamiento solo con la cantidad necesaria de clases a detectar, esto con el fin de agilizar el proceso. Por otra parte, como el sistema utiliza cada una de las regiones en las cuales es dividido inicialmente para realizar la detección y la creación del recuadro de detección, si el tamaño del objeto es demasiado pequeño o solo se logra ver una parte del objeto, entonces el sistema no es capaz de crear un recuadro para el mismo, por lo cual no será capaz de detectarlo.

6.2 Seguimiento de Objetos

El seguimiento de objetos es una herramienta bastante útil ya que esta, a diferencia de la detección de objetos, es capaz de detectar y mantener detectado un objeto en específico aun cuando el objeto no sea detectado por alguna razón, por ejemplo, que este se oculte detrás de otro objeto, esto se logra al utilizar los datos recolectados previamente a que el objeto deje de ser detectado para estimar la posición en la cual este se encontrara a continuación. Básicamente, el seguimiento de objetos es la estimación de la posición futura en la cual se encontrara un objeto en el cuadro siguiente en base a los datos previos del mismo, es por esto que cada uno de los seguidores de objetos se basan en el hecho de que el objeto no realizará cambios drásticos en la

dirección o la velocidad, por este motivo, si el objeto realiza un cambio brusco alguno en su dirección o acelera súbitamente el seguidor de objetos fallara en la predicción realizada y al momento de detectar nuevamente el objeto lo reconocerá como otro objeto distinto. Otro caso en el cual el seguidor puede fallar es si el objeto deja de ser detectado después de un cierto número de cuadros.

Existen diferentes métodos de seguimiento de objetos en imágenes, sin embargo, la mayoría son basados utilizando la lógica recursiva Bayesiana, filtros de Kalman y sus derivados, filtros de partículas o combinando uno o más de estos métodos.

6.2.1 Seguimiento de Objetos por medio de SORT.

SORT (Simple, Online, and Realtime Tracker) es un seguidor de múltiples objetos el cual se caracteriza por su rapidez y su simplicidad a comparación de otros seguidores de objetos. Fue diseñado para trabajar en conjunto con una gran diversidad de detectores de objetos, esto significa que este seguidor no es capaz de detectar por su cuenta los objetos en una imagen o cuadro lo cual hace que el procesamiento de los datos sea más rápido, la única información que necesita el programa es saber la ubicación de los objetos detectados en la imagen y la región que ocupan en la imagen, en otras palabras las coordenadas y el tamaño del objeto, también conocido como recuadro de detección. Estos datos que recibe son procesados para crear un modelo de estimación utilizando el filtro de Kalman para estimar la velocidad y la dirección en la que se mueve el objeto mientras avanza, estas estimaciones se logran al recopilar la información de cada uno de los objetos detectados y hacer suposiciones de cuál es el siguiente punto en base a los datos anteriores. En caso de que la predicción falle el filtro de Kalman utiliza los datos actuales recopilados para adaptar el patrón y mejorar la nueva estimación. Cada estimación de la futura posición del objeto detectado también cuenta con su propio recuadro de detección, la manera de realizarlo es utilizando el Algoritmo Húngaro, en el que cada uno de los valores de la matriz de costos es dado por la distancia mínima de la intersección sobre unión, o IOU por sus siglas en ingles Intersection-Over-Union, de cada uno de los recuadros de detección de los puntos detectados con cada uno de los recuadros de detección estimados, esto se realiza para evitar que el algoritmo evite relacionar dos objetos los cuales se encuentren muy cerca el uno del otro sean confundidos entre ellos o en el caso de que alguno de los objetos detectados sean

oculto por otro objeto de manera momentánea por lo que el algoritmo es capaz de realizar las correcciones para cada uno de ellos sin problema. La identificación para cada uno de los objetos detectados es creada si la distancia mínima entre IOU está por debajo del mínimo establecido, sin embargo, también es necesario eliminar las etiquetas que no son necesarias por lo que, si alguno de los objetos no es detectado por cierta cantidad de cuadros, entonces la etiqueta de ese objeto será eliminada. [25]

6.3 Aprendizaje Automático

En la actualidad las maquinas autónomas se han vuelto más comunes, sin embargo, es imposible el crear un programa el cual tome en cuenta cada una de las eventualidades que puedan suceder o modificar la programación cada vez que ocurra una eventualidad fuera de aquellas que fueron programadas anteriormente. Debido a lo anterior, el análisis de datos y el aprendizaje automático o Machine Learning han ido aumentando en popularidad en las décadas recientes. El aprendizaje automático es una herramienta la cual nos permite analizar grandes cantidades de información de manera rápida y eficiente, además de esto, el aprendizaje automático, como su nombre implica, permite que un programa o sistema aprenda en base a los análisis de la información recopilada por el sistema anteriormente y crear respuestas para futuras ocasiones en base a la información recopilada, lo cual aumenta su velocidad de reacción y también mejora de manera significativa la forma en la que realiza la tarea a seguir. [26] [27]

Debido a que no todos los problemas pueden ser abordados de la misma manera existen una extensa cantidad de algoritmos y programas basados en el análisis de datos para el aprendizaje automático, sin embargo, estas pueden ser divididas en 2 clasificaciones dependiendo del tipo de datos y la manera de tratarlos, estas son: Aprendizaje Supervisado y No Supervisado. [28]

6.3.1 Aprendizaje Supervisado

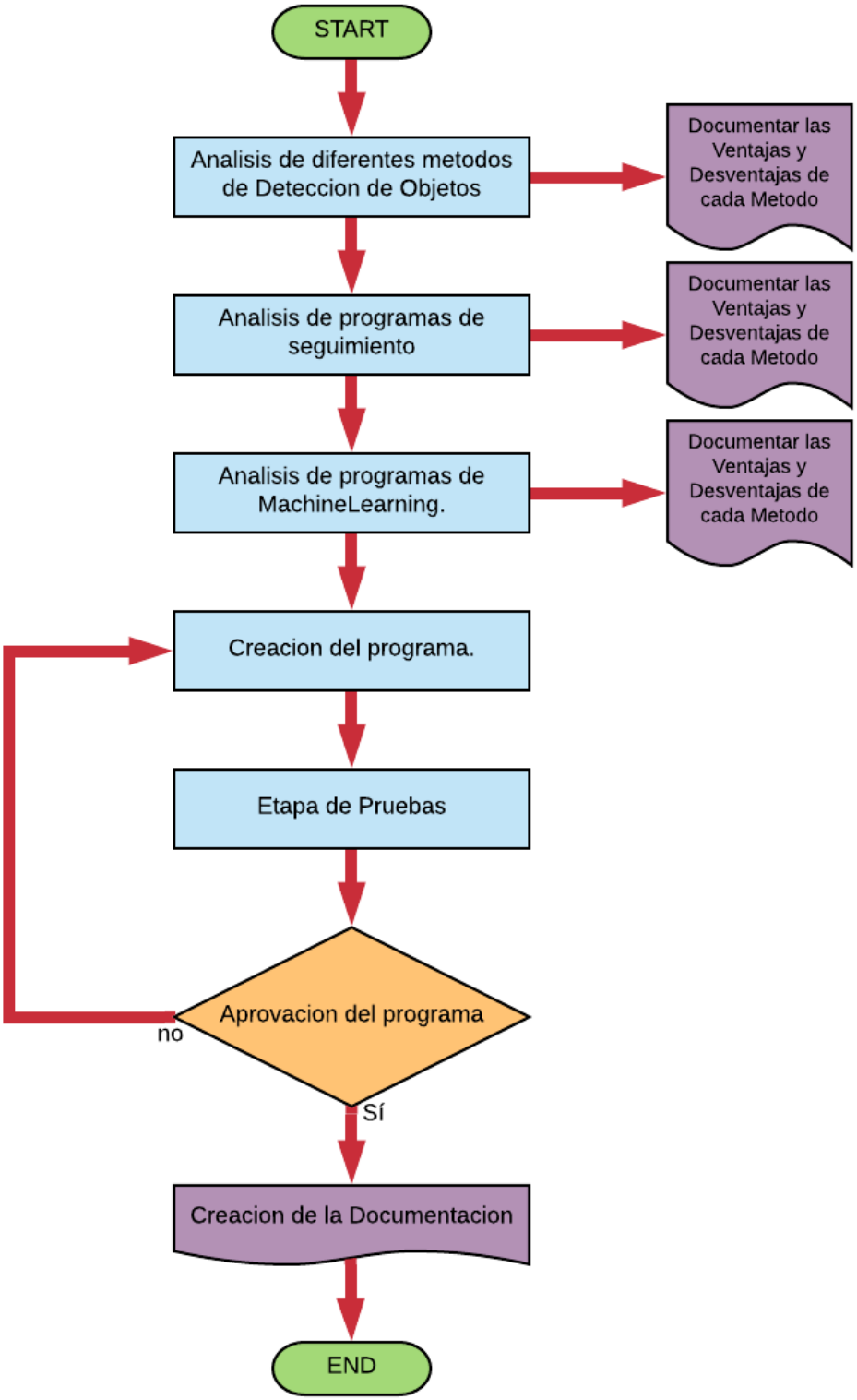
Se conoce como Aprendizaje Supervisado a aquellos algoritmos o métodos en los cuales el programa aprende a distinguir la relación de las diferentes variables de entrada con aquellas de salida, por lo cual, en este caso de aprendizaje, es necesario contar con cierta información previa sobre la relación entre esos, por lo que el sistema aprende de los ejemplos anteriores y en base a

esto es capaz de prevenir suponer como se comportara el sistema en relación con las variables que reciba. Unos ejemplos comunes de los tipos de problemas a solucionar utilizando este tipo de aprendizaje serian: Problemas de Clasificación, Problemas Regresivos. [29] [30] [31]

6.3.2 Aprendizaje No Supervisado

A diferencia del Aprendizaje Supervisado, los algoritmos o métodos solo tienen a disposición datos de entrada por lo que el sistema debe de encontrar cual es la relación entre los datos proveídos o de entrada al realizar hipótesis, inferencias, etc. En este tipo de modelo de Aprendizaje la salida del modelo será cual es la relación que tienen estos datos, por ejemplo, la cantidad de veces que se repiten, cuales son aquellos datos que aparecen más frecuentemente, etc. Los problemas más comúnmente solucionados por este tipo de aprendizaje son: Reducción de Dimensionalidad y Problemas de Agrupamiento [32] [33] [34]

7. Metodología



Para el desarrollo del proyecto usaremos el programa de Anaconda, este programa es una plataforma diseñada para programación mediante el lenguaje de Python, el cual incluye una consola en la cual podemos escribir el programa a desear, además, nos permite el instalar

diversos componentes para poder trabajar con diferentes programas como por ejemplo OPENCV, Visual Studio, entre otros.

Python cuenta con una extensa librería de programas entre las cuales se encuentran librerías de OPENCV las cuales incluyen programas como para detección de objetos mediante eliminación de fondo y por medio de características Haar, de las cuales haremos uso para probar la eficiencia de estos métodos.

La primera parte del proyecto consiste en analizar los diferentes métodos de visión computarizada que existen y comparar su desempeño. Como se mencionó anteriormente, los métodos a analizar serán:

- Detección de Objetos por medio de Eliminación de Fondo.
- Detección de Objetos por medio de Características HAAR.
- Detección de Objetos por medio de YOLO (You Only Look Once)

Cada uno de estos utiliza un método diferente para detectar objetos, siendo el método de Eliminación de Fondo, el más sencillo de ejecutar y el único que no requiere de un entrenamiento previo, sin embargo, la dificultad de este método radica en el ajustar los parámetros necesarios hasta que la detección sea precisa.

Para poder hacer una comparación más acertada todos los métodos serán probados utilizando el mismo video de vigilancia, el video original fue obtenido de YouTube.com mediante el siguiente enlace: https://www.youtube.com/watch?v=wqctLW0Hb_0.

7.1 Detección de Vehículos mediante Eliminación de Fondo.

La detección de objetos por medio de eliminación de fondo consiste en diversos pasos a seguir, primero el video se tiene que separar por cuadros, cada uno de los cuadros siendo una imagen fija. Posteriormente utilizaremos el algoritmo de eliminación de fondo, este programa analizara cada cuadro pixel por pixel con el cuadro anterior para determinar cuáles son los objetos que no se mueven o en otras palabras cual es el fondo. El siguiente paso reducir la cantidad de información que obtenemos de la imagen al aplicar un Threshold a la imagen. A continuación, será necesario el aplicar filtros para reducir la cantidad de ruido que queda de la imagen para

obtener una imagen más limpia y detallada de los objetos a detectar. Y para finalizar será necesario detectar cuales son los contornos que nos interesan de los que no y crear un recuadro sobre estos. Cada uno de estos pasos se explicarán a continuación.

7.1.1 Algoritmo de Eliminación de Fondo

Existen diversos algoritmos de eliminación de fondo, sin embargo, para este proyecto se utilizará el Algoritmo de segmentación de fondo/primer plano basado en Mezcla Gaussiana o MOG. OPENCV nos ofrece dos diferentes códigos los cuales están basados en este tipo de algoritmo, los cuales son llamados MOG (cv2.createBackgroundSubtractorMOG) y MOG2 (cv2.createBackgroundSubtractorMOG2), cada uno de ellos toma un valor diferente de la constante k utilizada en el algoritmo además de cambiar la distribución gaussiana realizada en la imagen por lo que se puede obtener resultados muy diferentes.

Ilustración 4. Comparación de MOG (imagen izquierda) vs MOG2 (Imagen derecha).

Como se puede observar en la imagen de arriba, al utilizar el algoritmo MOG se obtiene una imagen con poco ruido, sin embargo, la forma de los objetos no se encuentra muy bien definida y, en algunos casos, los objetos más alejados no son completamente detectados. En comparación, con el algoritmo MOG2 se puede apreciar de una mejor manera la forma de los objetos a pesar de que la imagen produce más ruido, pero esto puede ser solucionado mediante el uso de filtros y

otros algoritmos. Debido a lo anterior se decidió que para el proyecto se utilizaría el algoritmo MOG2.

Uno de los problemas que tiene el algoritmo de eliminación de fondo es que, en el caso de no contar con una imagen de fondo con la cual comparar la imagen, el algoritmo necesitará determinar el fondo de la imagen al comparar un cuadro con el anterior, sin embargo esto ocasionará que el programa determine el primer cuadro como la imagen de fondo al no tener otro cuadro con la cual compararlo por lo que es necesario que esto se realice dentro de un bucle al inicio a modo de entrenamiento ya que de lo contrario el programa detectara el cuadro inicial como fondo al no tener otro cuadro anterior con la cual comparar por lo que ocasionara falsas detecciones, como se puede observar en la Ilustración 5, hasta que el mismo programa se autocorrija después de un número indeterminado de cuadros.

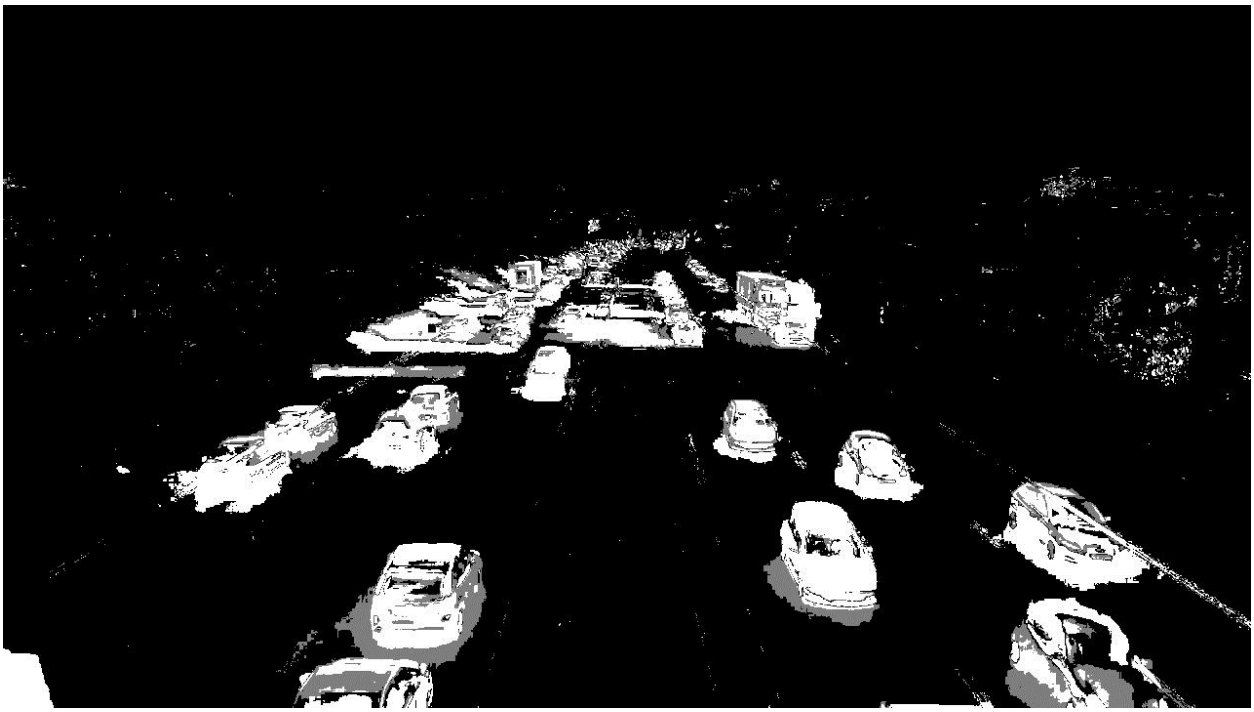


Ilustración 5. Eliminación de fondo usando algoritmo MOG2 sin entrenamiento.

Después de ejecutar el entrenamiento y de obtener la imagen de fondo es necesario volver a aplicar el algoritmo de eliminación de fondo ya que esto nos dará la imagen que buscamos obtener solo con aquellos objetos que se muevan en la imagen con respecto a la imagen de fondo que se obtuvo previamente. Existe un beneficio de poner en un bucle previamente y es que al

realizar esto, podemos reducir la cantidad de ruido que se obtiene como se puede observar en la Ilustración 6.

Ilustración 6. Eliminación de fondo usando algoritmo MOG2 con entrenamiento.

El siguiente paso es intentar corregir la imagen. Como se puede observar en la Ilustración 6, al utilizar el algoritmo de eliminación de fondo, obtenemos cierta cantidad de ruido el cual debe de ser eliminado o reducido lo más que sea posible, otro factor que se tiene que eliminar o reducir son las sombras ya que, a pesar de que el algoritmo de eliminación de fondo incluye una opción para eliminarlo, este no es perfecto. Para lograr esto utilizaremos diferentes algoritmos como lo son el Threshold y Eliminación de Ruido.

7.1.2 Threshold.

El Threshold se trata simplemente de filtrar si el valor de los pixeles detectados excede un valor determinado el algoritmo lo interpretara como blanco, en el caso contrario será pasado a ser negro, esto se hace para convertir el valor de cada pixel en solo 2, 255 o 0 según sea el caso. Para lograr esto existen diferentes tipos de algoritmos, y OPENCV nos muestra 3, estos son:

1. **Simple Thresholding “cv2.threshold(cv2.THRESH_BINARY)”.** Este método consiste simplemente en establecer un valor base para comparar todos y cada uno de los pixeles de la imagen, si estos son iguales o sobrepasan el valor base establecido entonces el valor del pixel analizado pasara a ser 255 o color blanco, en caso contrario el valor del pixel pasara a ser 0 o color negro como se puede observar a continuación.

Ilustración 7. Simple Thresholding.

2. **Adaptative Thresholding.** En este tipo de thresholding lo que se hace es dividir la imagen en áreas para obtener cual es el valor de threshold que debe de tener cada una, esto se hace ya que dependiendo de la iluminación que tenga la imagen ciertas áreas pueden tener una iluminación diferente a las otras. Existen dos algoritmos para logar esto, el primero:
    a. **Mean C Threshold “cv2.adaptiveThreshold(cv2.ADAPTIVE_THRESH_MEAN_C)”.** Es el método mencionado anteriormente de valor medio de áreas (Ilustración 8)
    b. **Gaussian Threshold “cv2.adaptiveThreshold(cv2.ADAPTIVE_THRESH_GAUSSIAN_C)”.** El

valor de threshold del área se obtiene de la suma ponderada del valor medio de las áreas cercanas (Ilustración 9).

Ilustración 8. Adaptative Thresholding.

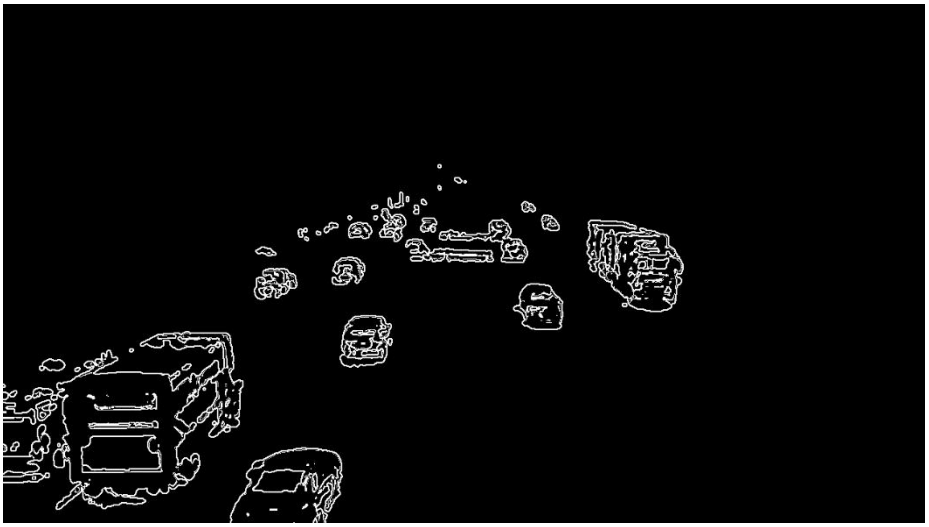


Ilustración 9. Gaussian Threshold.

3. **Binarizacion de Otsu "cv2.threshold(cv2.THRESH_OTSU)"**. En este algoritmo el valor base de threshold se obtiene al encontrar el valor umbral que minimice la variación ponderada dada por el histograma de pixeles.

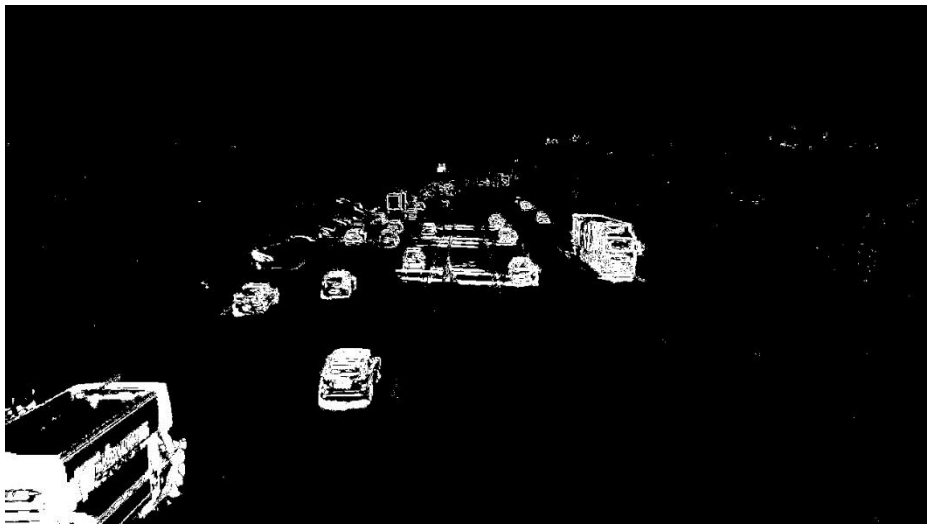


Ilustración 10. Otsu Threshold.

Como se puede observar en las Ilustraciones 7 a la 10, dependiendo del tipo de threshold que se use se obtiene un resultado diferente, a excepción del Mean Adaptative Threshold y del Gaussian Threshold, en los cuales el resultado es muy similar. Sin embargo, para los fines del proyecto en cuestión aquellos que arrojaron un mejor resultado fueron el Normal Threshold y el Otsu Threshold, sin embargo, a pesar de que el Normal Threshold nos ofrece una imagen con poco ruido, también reduce parcialmente la detección de ciertos objetos, a comparación del algoritmo de Otsu el cual, a pesar de que reduce de manera no muy significativa la cantidad de ruido de la imagen, no afecta de manera drástica los objetos detectados. Es por esto que el Método de Threshold a utilizar en el proyecto será el de Otsu.

7.1.3 Eliminación de Ruido

El método de eliminación de ruido es bastante simple, consiste en cambiar la forma en la cual los pixeles están distribuidos, como por ejemplo cambiar su forma, tamaño, grosor, etc.

En este caso utilizaremos los algoritmos de

A. **Cerrado “cv2.morphologyEx(cv2.MORPH_CLOSE)”.**

B. Abierto “cv2.morphologyEx(cv2.MORPH_OPEN)”.

C. Dilatación “cv2.dilate()”.

D. Cambio de morfología “cv2.getStructuringElement(cv2.MORPH_ELLIPSE)”.

El algoritmo de Cambio de Morfología lo que hace es cambiar la forma en la cual se presentan los conjuntos de pixeles, por ejemplo, cambiarlo para que sean de forma rectangular, elipsoidal o en cruz, al mismo tiempo en el que se determina un tamaño para las formas.

Para los algoritmos de Cerrado y Abierto es una combinación de los algoritmos de Dilatación y Erosión, donde cada algoritmo reduce el área de los pixeles o la aumenta, dependiendo del algoritmo usado. Para el caso de Cerrado, la combinación usada es Dilatación (aumentar el tamaño del área de los pixeles en cierta cantidad) seguido de Erosión (reducción del tamaño del área de los pixeles en cierta cantidad). En el caso de Abierto es el inverso, se utiliza Erosión seguido de Dilatación.

Con esto se puede reducir la cantidad de ruido que se obtiene de la imagen dada por el threshold obteniendo una imagen con objetos más definidos y con mucho menos ruido, sin embargo, para encontrar el tamaño que se tiene que utilizar y el orden en el cual se tienen que utilizar cada uno de estos algoritmos para obtener una imagen más clara es mediante prueba y error, como se puede ver a continuación.

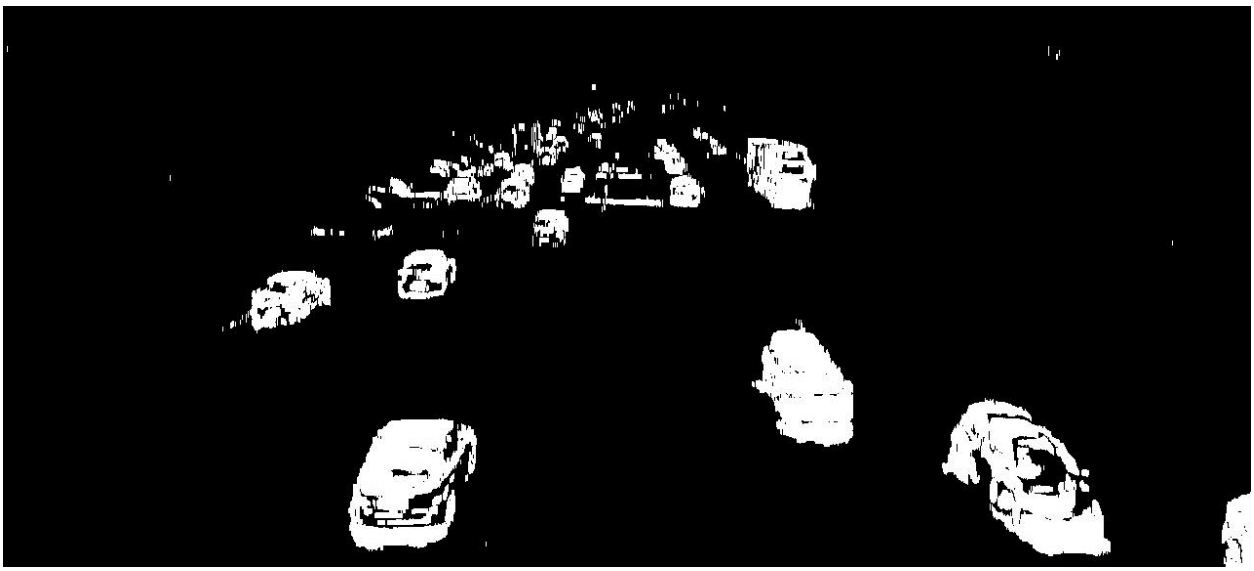


Ilustración 11. Forma: Elipse (1,2), Closing (3), Opening (2), Dilatation (2)

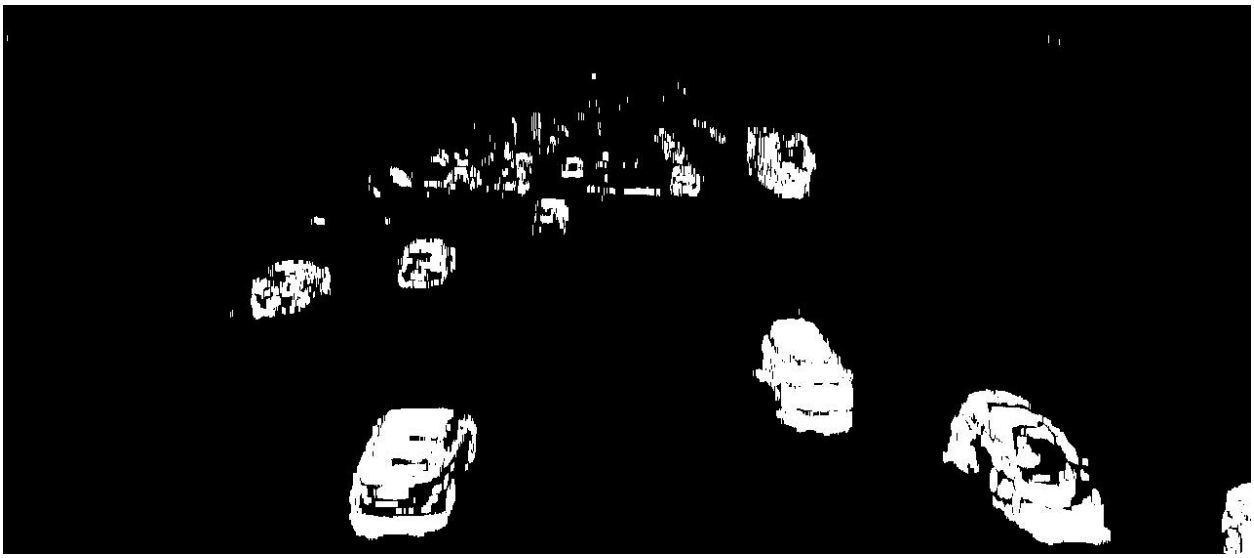


Ilustración 12. Forma: Elipse (1,2), Opening (1), Closing (2), Dilatation (2)

Como se puede observar en las Ilustraciones 11 y 12, al cambiar el orden en el que se aplican los algoritmos de Cerrado y Abierto nos ayuda a reducir la cantidad de ruido que se obtiene en la imagen, siendo la segunda forma la que nos da una imagen con menos ruido, sin embargo, las imágenes más alejadas son más difíciles de apreciar en su totalidad. Debido a la cantidad de ruido eliminado de la imagen optaremos por la segunda secuencia para hacer las pruebas para este método.

7.1.4 Detección de Contornos

Para terminar lo que se debe de realizar es implementar el algoritmo de Detección de Contornos "cv2.findContours(cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_TC89_L1)", un contorno es un conjunto de puntos continuos los cuales poseen el mismo color y la misma intensidad, es por esta razón por la cual al inicio del programa se aplicó un Threshold a la imagen, para que de esta manera todos los puntos de interés tuvieran el mismo color y la misma intensidad, ya que de lo contrario al utilizar el algoritmo de Detección de Contornos obtendríamos diversos contornos dentro de un mismo objeto.

Al utilizar el algoritmo de detección de contornos es necesario introducir cual será el modo de recuperación de los contornos y que método se utilizará para obtener los puntos de referencia de este. Para este caso el método que nos interesa es el saber cuáles son los puntos externos () que conforman el contorno (cv2.RETR_EXTERNAL), en lugar de los puntos interiores, y el método para obtener los puntos de referencia es mediante uno de los algoritmos de aproximación creados por Teh-Chin (cv2.CHAIN_APPROX_TC89_L1).

Este algoritmo nos da como resultado lo siguiente: un conjunto de vectores correspondientes a las coordenadas (x, y) del objeto detectado, así como que tan largo y ancho es el objeto (l, w).

Con estos datos nos es posible el crear el recuadro que encapsulara el objeto en cuestión en la imagen original además de permitirnos crear un filtro extra al código, este filtro funcionará de manera que solo aquellos objetos que tengan un ancho o longitud mayor a un valor dado se les asigne un recuadro, esto a manera de evitar que se detecten diversos objetos dentro de uno solo.

Además de esto, este filtro nos permitiría el determinar si los objetos detectados pertenecen a una clase en particular, como por ejemplo, un carro no tiene el mismo ancho que una motocicleta, ni tampoco la misma altura que un camión, por lo que en teoría, si se conoce cuál es la medida especifica de cada uno de estos, es posible el crear etiquetas para cada uno de los objetos que se quieran detectar, sin embargo esto solo es posible si no se considera que los objetos al alejarse o acercarse de la cámara cambian de tamaño.

7.2 Detección de Objetos por medio de Características Haar.

El programa para la Detección de Objetos por medio de Características Haar es relativamente sencillo como se puede observar en la sección de Anexos. Lo único que nos requiere el programa es saber cuál es el archivo que contiene los datos del entrenamiento de Cascada de Clasificación "cv2.CascadeClassifier()" que se quiere detectar el cual debe de ser realizado previamente y el archivo de video en el cual se quiere buscar los objetos. Sin embargo, al igual que con el Método de Eliminación de Fondo, para que el algoritmo pueda buscar los objetos a detectar se tiene que transformar la imagen a blanco y negro, este paso era realizado automáticamente por el algoritmo de Eliminación de Fondo por lo cual no era necesario hacerlo manualmente, por lo tanto, el archivo de video que se tiene que usar en el algoritmo Haar tiene que ser pasado de su

formato original a un formato en blanco y negro utilizando el algoritmo "cv2.cvtColor(cv2.COLOR_BGR2GRAY)".

Una vez que se tienen los dos archivos necesarios, entonces se puede utilizar el algoritmo de Deteccion de Objetos por Características Haar el cual es complemento del algoritmo de Cascada de Clasificación ".detectMultiScale()".

La parte más importante de un Detector por Características Haar es el archivo de entrenamiento, este archivo contiene todos los datos que fueron recopilados utilizando cientos o miles de imágenes tanto positivas (imágenes que contienen el objeto con el que se quiere entrenar el clasificador) como negativas (imágenes que no contienen el objeto que se desea entrenar, ni siquiera de manera parcial). El algoritmo de Entrenamiento de Clasificador Haar "opencv_createsamples()" requiere que las imágenes positivas y negativas estén guardadas en carpetas diferentes y especificar cuál es la dirección en la cual están guardadas. Otro requisito es un archivo de texto el cual contenga las coordenadas (x, y), el ancho y la altura (l , w) de la ubicación de los objetos por cada una de las imágenes positivas, puede que una imagen positiva contenga más de un objeto con el que se quiere entrenar el programa por lo que es necesario especificar cuantos objetos hay en la imagen y los datos pertinentes para cada uno de ellos en la misma línea.

El funcionamiento del algoritmo de entrenamiento es el siguiente: por cada imagen positiva que se obtenga, el algoritmo ubica el objeto deseado y lo copia, esta copia será "pegada" en cada una de las imágenes negativas colocándolo en una posición y ángulos diferentes. Una vez que se haya realizado lo anterior, se procederá a ejecutar el algoritmo de Clasificación Haar, cuyo funcionamiento se explicó anteriormente, de manera recursiva guardando los resultados obtenidos en archivos de memorias. Estos archivos contienen los parámetros específicos del objeto que debe de buscar el Clasificador en toda la imagen, y también son utilizados como datos iniciales para la siguiente recursión del programa, de esta manera el Clasificador ira aprendiendo cada vez más hasta que se alcance un resultado satisfactorio. Este entrenamiento puede llegar a durar entre minutos, horas, incluso semanas, dependiendo de la cantidad de imágenes positivas y negativas que se utilizaran. Uno podría pensar que mientras más se entrene el Clasificador más certero será la detección, sin embargo, esto es completamente incorrecto, ya que de haber un sobre entrenamiento el Clasificador puede que sea incapaz de detectar los objetos a desear, lo

mismo pasaría si no se agregaran la cantidad suficiente de imágenes positivas y negativas, o si la calidad de estas es muy pobre. En nuestro caso, para evitar el problema de tener que entrenar un Clasificador propio, utilizaremos un archivo de Entrenamiento ya realizado.

7.3 Detección de Objetos por medio de YOLO.

El método de Detección de Objetos por medio de YOLO a diferencia de los métodos anteriores se trata de un método de clasificación múltiple, esto quiere decir, que con un solo entrenamiento que se realice se puede detectar diferentes objetos sin importar si estos son similares en cuanto aspecto o colores. Además de esto, a diferencia del método de detección por Características Haar, este no presenta tantos problemas con respecto a las diferencias del cambio de iluminación que tengan las imágenes utilizadas durante el entrenamiento del clasificador, esto logra que la detección sea más precisa.

Otra de las diferencias que presenta este método corresponde a la parte de entrenamiento, ya que actualmente Python no cuenta con ninguna herramienta para realizar un entrenamiento el cual este enfocado para este tipo de método por lo cual es necesario utilizar las herramientas proporcionadas por el creador de YOLO, estas se pueden encontrar y ser descargadas a través del sitio web oficial de YOLO (el enlace es el siguiente: https://pjreddie.com/darknet/yolo/) en el cual se puede descargar el programa desarrollado por ellos mismos llamado Darknet el cual sirve para ejecutar, entrenar y verificar las detecciones realizadas por el mismo además de contar con un detector previamente entrenado para detectar hasta 80 clases de objetos (este entrenamiento proporcionado por Darknet es el que será utilizado para realizar las pruebas de detección de objetos). Darknet al igual que Python, están basados en el lenguaje de programación de Linux, por lo cual, para utilizar Darknet y sus diversas herramientas es necesario contar con este sistema operativo.

Existen diferentes formas de realizar un entrenamiento mediante YOLO, esto se debe a que, a diferencia de otros algoritmos de detección de objetos basados en redes neuronales, YOLO es capaz de soportar diferentes tipos de formatos de bases de datos, cada uno de los cuales utiliza un archivo de texto con un formato específico para especificar la ubicación, tamaño y cantidad de objetos que se encuentran en cada una de las imágenes que se escogieron para realizar el

entrenamiento, tal es el caso de bases de datos como Pascal VOC (Visual Object Classes) o COCO (Common Objects in Context), los cuales son los más utilizados para realizar los entrenamientos utilizando Darknet, aunque además de estos existen muchas bases de datos de clasificación de objetos, cada una con un formato de texto diferente, aunque todas incluyen la misma información, la cantidad de objetos a detectar en la imagen, las coordenadas en las cuales se encuentra el primer punto margen de cada uno de los objetos, su anchura, longitud, algunas incluyen además el tamaño de la imagen a analizar.

Para realizar un entrenamiento por medio de Darknet, es necesario descargar las imágenes correspondientes a la clase o clases para las cuales deseamos entrenar el clasificador, en este caso nosotros utilizamos la base de datos de COCO del 2017, además de esto es necesario contar con los datos de un clasificador "básico" o inicial, este clasificador solamente es utilizado para reducir el tiempo y mejorar el rendimiento del entrenamiento ya que este cuenta con los pesos iniciales con los que debe de contar la red neuronal, los cuales también pueden ser descargados de manera gratuita y que además ya vienen incluidos al momento de descargar Darknet. Una vez que tenemos las imágenes de clasificación junto con sus respectivos archivos de texto, es necesario modificar un archivo llamado "coco.data", el cual se encuentra en la carpeta donde se instaló Darknet, en este archivo se especificara la cantidad de clases que se pretende entrenar, la dirección en la cual se encuentra la carpeta con las imágenes descargadas de COCO, los archivos de texto correspondientes, y además la dirección en donde se desea que se obtenga el archivo conteniendo los pesos resultantes del entrenamiento. Ya que esto se ha realizado es posible el ejecutar el programa de entrenamiento de Darknet, el cual comenzara a analizar cada una de las imágenes con cada una de las correspondientes clases que se desea entrenar.

Una vez realizado el entrenamiento de clases se puede utilizar el archivo generado con extensión ".weights" para realizar la detección de objetos utilizando el código "Darknet()" el cual nos permite utilizar los pesos generados por YOLO, además de este código existen otros los cuales también nos permiten realizar detección de objetos utilizando RNC como por ejemplo el módulo de "opencv_dnn()" que nos ofrece OPENCV.

7.4 Seguimiento de Objetos

Como se mencionó anteriormente, para este proyecto se eligió específicamente el programa de SORT, el cual solo requiere de descargar el programa de la página oficial utilizando el siguiente link "https://github.com/abewley/sort" y simplemente mandar llamar el programa mediante Python el cual asignara un valor a cada una de las detecciones hechas por el programa y mantendrá dicho valor asignado a ese objeto hasta que este se encuentre fuera de la pantalla o deje de ser detectado.

7.5 Aprendizaje Automático

De los diferentes tipos de algoritmos que existen de aprendizaje automático se escogió el método de K-Nearest Point, este método consiste en asignar como positivos los puntos por los cuales cada uno de los objetos fue detectado, en este caso, las rutas que siguen los vehículos, mientras que como valores negativos son asignados cada uno de los pixeles que conforman la imagen. La tarea del programa es el saber a qué clasificación pertenece un punto determinado, la manera de realizarlo es comparando un cierto número de puntos cercanos y comparando a cuál mayoría estos puntos pertenecen. Un ejemplo sencillo sería un grupo de seis puntos de los cuales 3 son de valor positivo mientras que los otros 2 de valor negativo, por lo tanto, el programa determinara que el punto restante pertenece a la clasificación positiva.

Debido a que no se contaba con un video de prueba lo suficientemente largo y con diferentes rutas a seguir para crear una muestra lo bastante buena se optó por incrementar el radio de cada uno de los puntos detectados, de esta manera por cada punto que se detectara se obtendrían 5 puntos adicionales a la izquierda y derecha del punto original, con esto se facilitaría el realizar un entrenamiento más confiable y rápido.

8. Resultados

8.1 Resultados obtenidos al utilizar el método de Eliminación de Fondo:

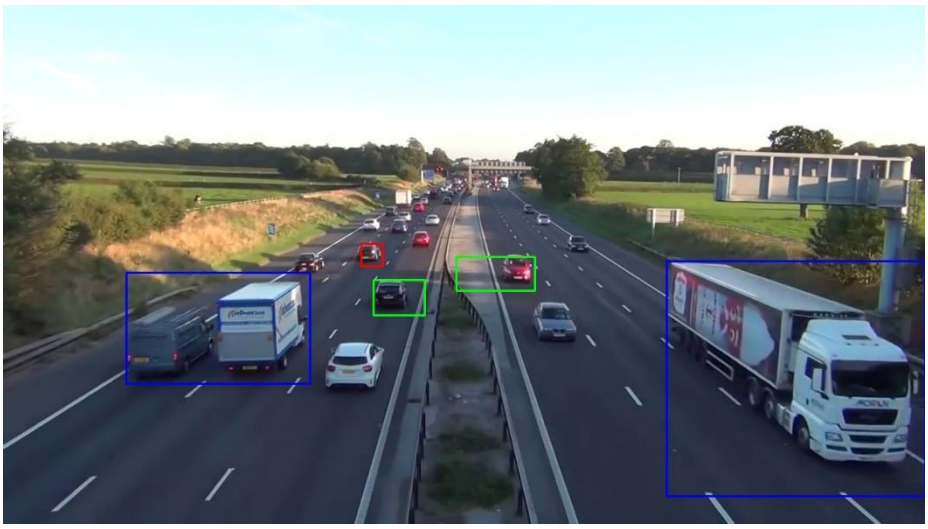


Ilustración 13. Detección de vehículos por medio de Eliminación de Fondo.

En esta imagen se logra apreciar que, al aplicar todos los filtros discutidos, el programa logra diferenciar entre diversos objetos detectados, siendo camiones aquellos con recuadro azul, automóviles aquellos con recuadro verde, y motocicletas o bicicletas aquellos con recuadro rojo. Debido a los parámetros introducidos para diferenciar entre diversos vehículos, el programa ignora varios objetos que detecta y que no cumplen con las características que se introdujeron, además de esto, a medida que los objetos se alejan o se acercan, el programa tendrá dificultades para determinar si son de una clase o de otra. Otro problema con este detector de objetos es que al haber dos objetos muy cerca el uno del otro, o en el caso de las sombras, el detector no es capaz de diferenciar si son varios objetos o si solo se trata de uno solo.

El código completo utilizado para este método de detección de objetos se puede encontrar en la sección de anexos.

8.2 Resultados obtenidos al utilizar el método de Características Haar.

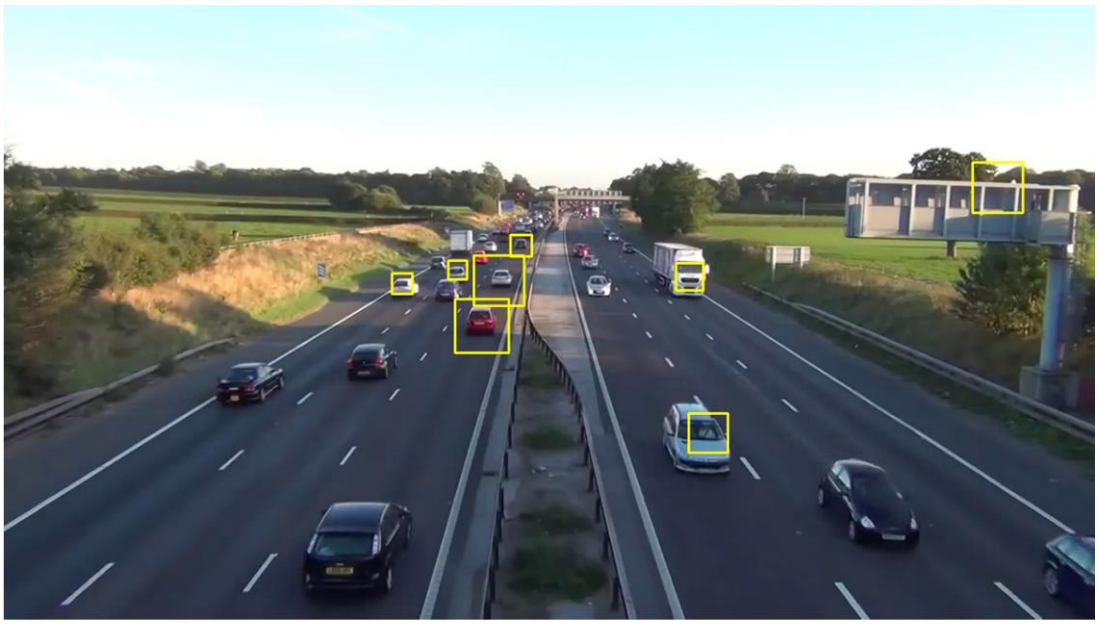


Ilustración 114. Detección de vehículos por medio de Características Haar.

Como se puede observar el la Ilustración 14 el programa logra reconocer los vehículos con una mayor precisión sin embargo no es completamente preciso ya que los vehículos de color obscuro no los reconoce, ni tampoco es capaz de reconocer si el vehículo es un automóvil, un camión o algún otro. Además de esto, si se observa en la parte superior derecha de la imagen se puede observar que el detector reconoció una parte del signo colgante como un vehículo.

8.3 Resultados obtenidos al utilizar el método de YOLO.

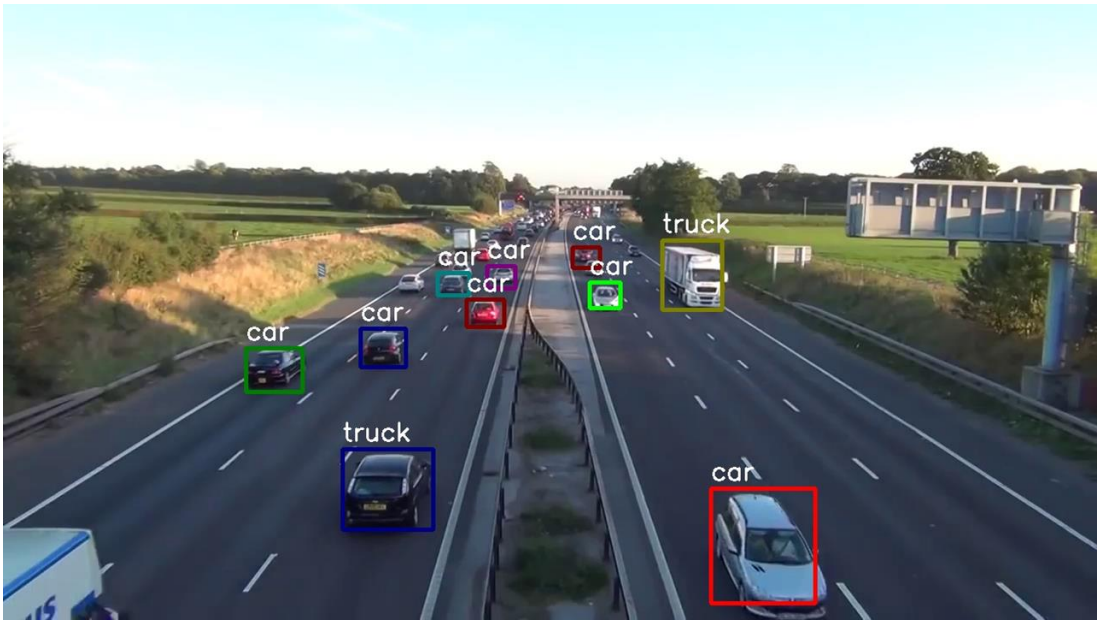


Ilustración 115. Detección de vehículos por medio de YOLO.

A diferencia de los detectores anteriores, al utilizar el método de detección por medio de YOLO se puede observar que las detecciones son mas precisas con solo unos cuantos errores en la detección. Como se observa en la Ilustración 15 de los vehículos detectados solamente un vehículo fue detectado como camión en lugar de ser detectado como automóvil, sin embargo, aun es capaz de detectar que hay un vehículo.

8.4 Resultados obtenidos al utilizar YOLO y SORT.

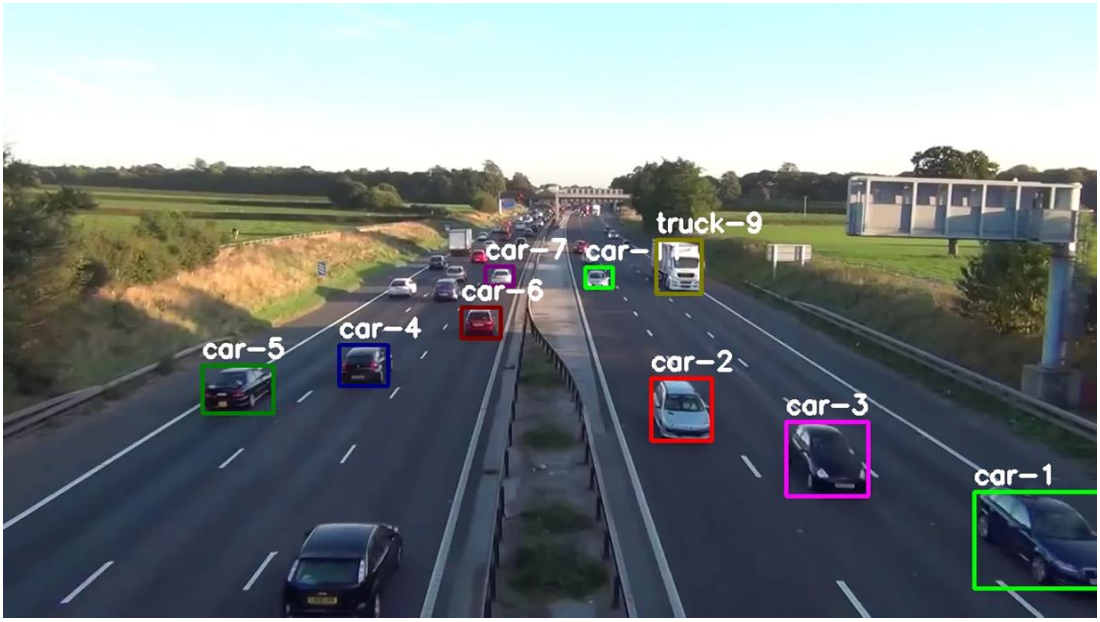


Ilustración 116. Detección y seguimiento de objetos.

La Ilustración 16 nos muestra como es que el programa final es capaz de detectar los vehículos transitando, identificando que tipo de vehículo es cada uno además de asignar un numero de identificación a cada uno de ellos, este numero sigue a cada vehículo hasta que este deja de ser detectado o sale de la pantalla.

8.5 Mapeo de puntos de los objetos detectados.

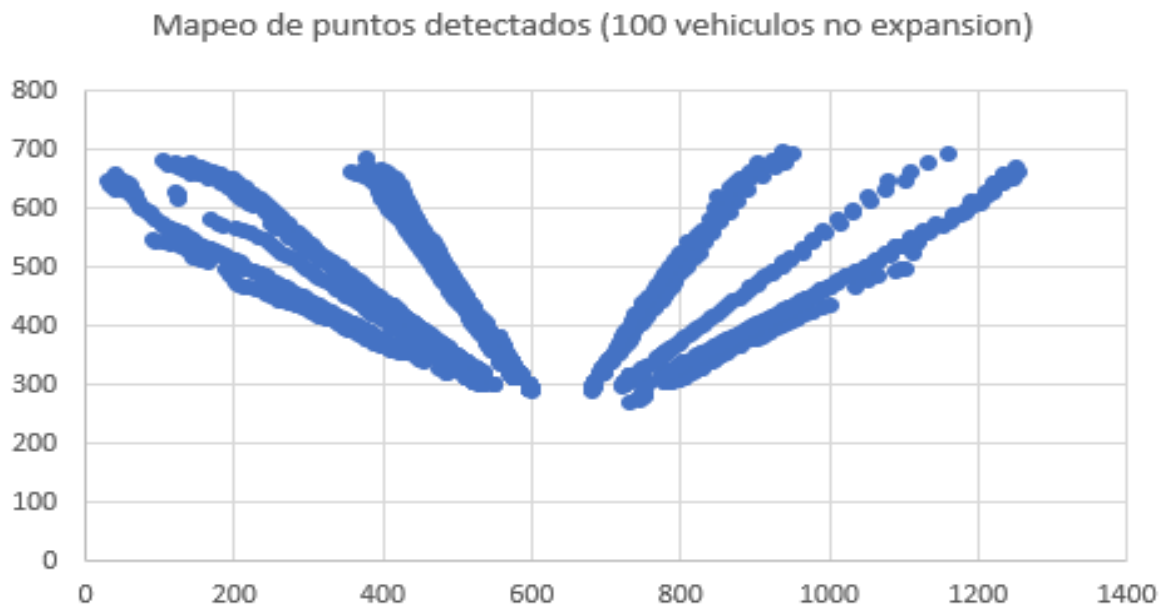


Ilustración 117. Mapeo de puntos detectados con una muestra de 100 vehículos sin expandir.

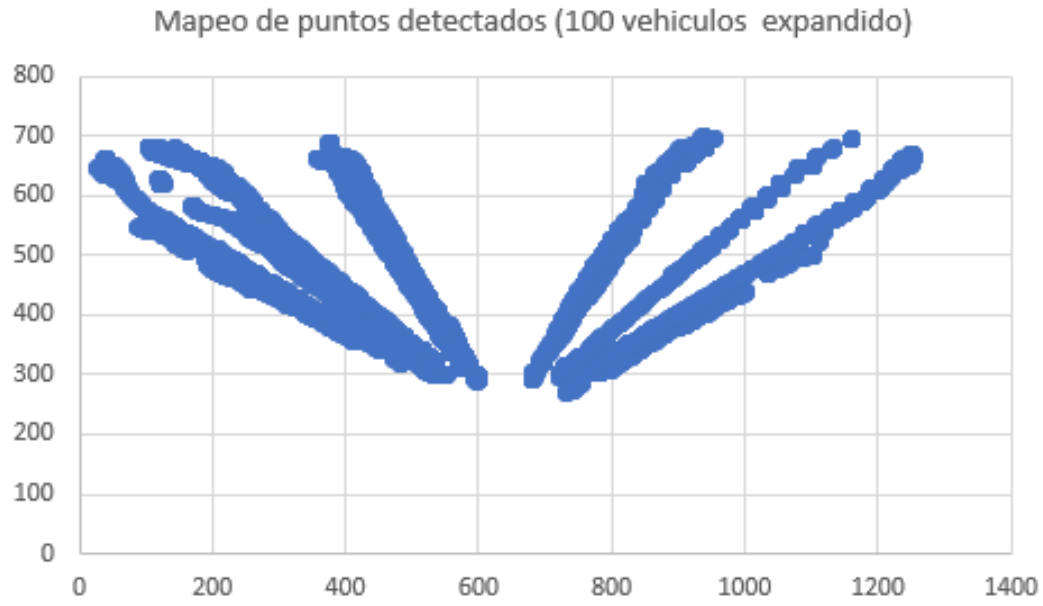


Ilustración 118. Mapeo de puntos detectados con una muestra de 100 vehículos expandido.

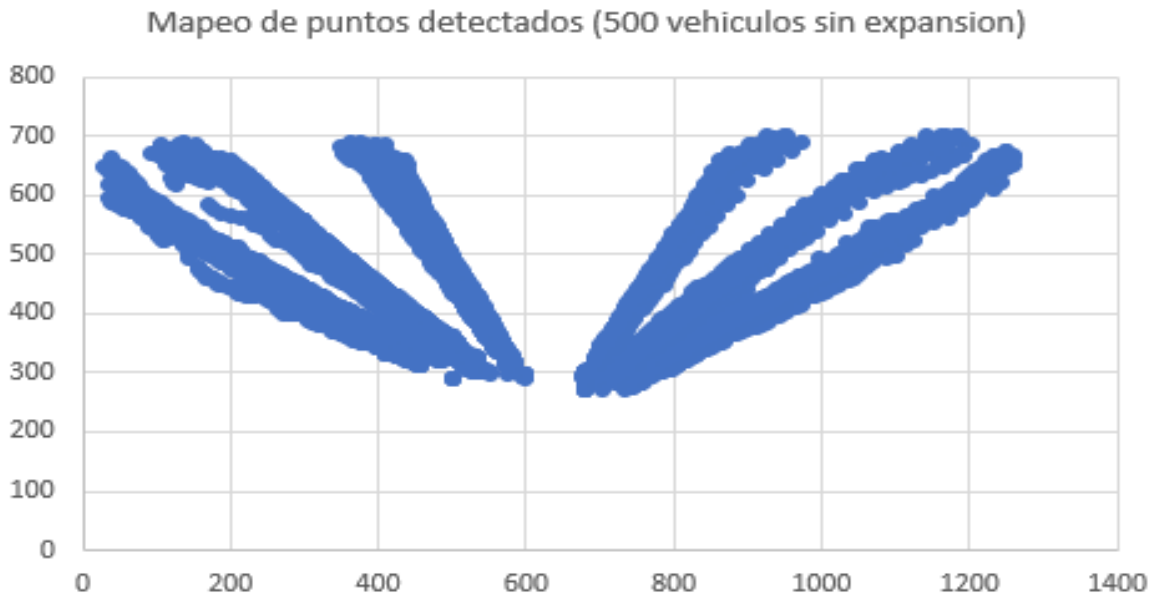


Ilustración 19. Mapeo de puntos detectados con una muestra de 500 vehículos sin expandir.

Ilustración 20. Mapeo de puntos detectados con una muestra de 500 vehículos expandido.

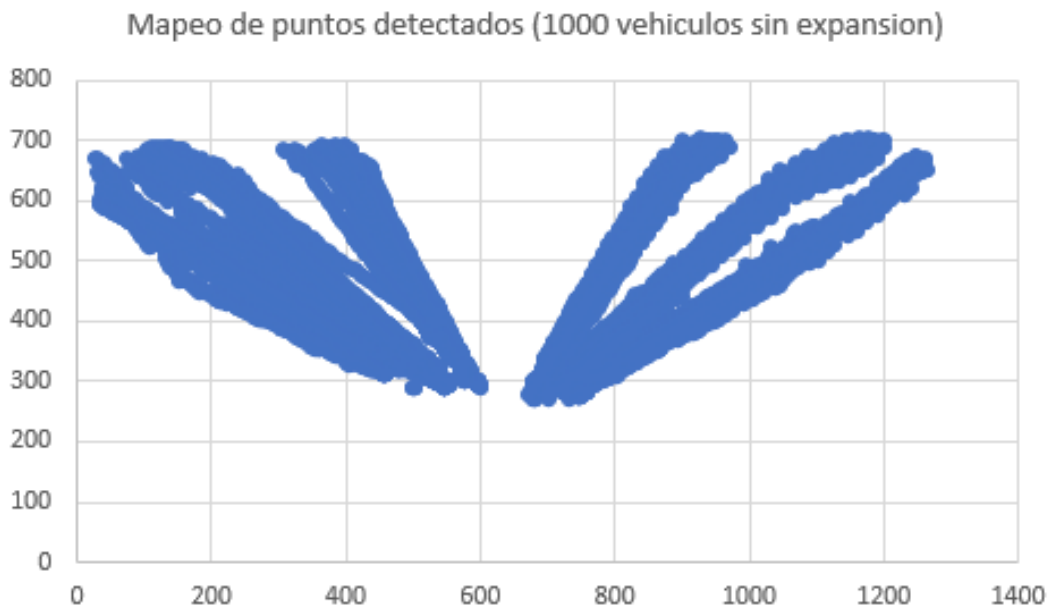


Ilustración 21. Mapeo de puntos detectados con una muestra de 1000 vehículos sin expandir.

9. Conclusiones

Como se puede observar en los resultados obtenidos la detección por medio de eliminación de ruido es eficaz sin embargo es necesario aplicar un sin número de veces para poder obtener una imagen sin ningún tipo de objeto indeseado, sin embargo, al realizar esto también se afectan aquellos objetos que queremos detectar. Por otro lado, la detección por medio de características Haar dejo mucho que desear ya que debido a las condiciones en las cuales fue realizado el entrenamiento y las condiciones del video no eran favorables por lo cual ocurrieron diversas fallas en la detección. Debido a su inestabilidad no se recomienda su uso en situaciones en las cuales las condiciones luminosas no sean las apropiadas, además de que, en el caso de Haar, es necesario realizar un entrenamiento y clasificar los vehículos bajo las condiciones en las cuales se pretende trabajar para mejorar la eficiencia del programa.

La detección de objetos por medio de YOLO es una manera precisa y segura con la cual trabajar, el único problema es la capacidad de hardware necesario para eso, a comparación con las anteriores, consume una gran cantidad de recursos para trabajar.

10. Bibliografía

[1] Betanzo-Quezada, Eduardo. 2011. *Una aproximación metodológica al estudio integrado del transporte urbano de carga: el caso de la Zona Metropolitana de Querétaro en México*. EURE (Santiago), 37(112), 63-87.

[2] Obregón-Biosca, Saúl Antonio, & Betanzo-Quezada, Eduardo. 2015. *Análisis de la movilidad urbana de una ciudad media mexicana, caso de estudio: Santiago de Querétaro*. Economía, sociedad y territorio, 15(47), 61-98.

[3] Instituto Nacional de Estadística y Geografía [INEGI] (2020). Investigación: Estadísticas de Vehículos de Motor Registrados. Recuperado de https://www.inegi.org.mx/programas/vehiculosmotor/

[4] Instituto Nacional de Estadística y Geografía [INEGI] (2020). Investigación: Estadística de accidentes de tránsito terrestre en zonas urbanas y suburbanas. Recuperado de https://www.inegi.org.mx/temas/accidentes/

[5] Zehang Sun, George Bebis and Ronald Miller. *On-Road Vehicle Detection Using Gabor Filters And Support Vector Machines*. Computer Vision Laboratory, Department of Computer Science, University of Nevada, Reno; e-Technology Department, Ford Motor Company, Dearborn, MI.

[6] S. Sivaraman and M. M. Trivedi. 2013. *Looking at Vehicles on the Road: A Survey of Vision-Based Vehicle Detection, Tracking, and Behavior Analysis*. IEEE Transactions on Intelligent Transportation Systems, vol. 14, no. 4, pp. 1773-1795, Dec. 2013.

[7] Andrey Nikishaev. 2017. *Making Road Traffic Counting App based on Computer Vision and OpenCV*. Recuperado de https://medium.com/machine-learning-world/tutorial-making-road-traffic-counting-app-based-on-computer-vision-and-opencv-166937911660

[8] Richard Szeliski. 2010. *Computer Vision: Algorithms and Applications*. Ed. Springer Science & Business Media.

[9] Klevis Ramo. 2019. *Hands-On Java Deep Learning for Computer Vision: Implement machine learning and neural network methodologies to perform computer vision-related tasks*. Ed. Packt Publishing Ltd.

[10] Ahmed Elgammal. 2014. *Background Subtraction: Theory and Practice*. Ed. Morgan & Claypool Publishers.

[11] Soharab Hossain Shaikh, Khalid Saeed, Nabendu Chaki. 2014. *Moving Object Detection Using Background Subtraction*. Ed. Springer.

[12] Gary Bradski, Adrian Kaehler. 2008. *Learning OpenCV: Computer Vision with the OpenCV Library*, Ed. O'Reily.

[13] Robert Burduk, Marek Kurzynski, Michał Wozniak. 2019. *Progress in Computer Recognition Systems*. Ed. Springer.

[14] Luis Alvarez, Marta Mejail, Luis Gomez, Julio Jacobo. 2012. *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications: 17th Iberoamerican Congress, CIARP 2012, Buenos Aires, Argentina, September 3-6, 2012, Proceedings*. Ed. Springer.

[15] Dinesh Tavasalkar. 2019. *Hands-On Robotics Programming with C++: Leverage Raspberry Pi 3 and C++ libraries to build intelligent robotics applications*. Ed. Packt Publishing Ltd.

[16] Jorge Santiago Nolasco Valenzuela. 2018. *Python Aplicaciones prácticas*. Ed. RA-MA.

[17] Wilson, P.I., & Fernandez, J.D. 2006. *Facial feature detection using Haar classifiers.*

[18] Simon J. D. Prince. 2012. *Computer Vision: Models, Learning, and Inference*. Ed. Cambridge University Press

[19] Giuseppe Bonaccorso, Armando Fandango, Rajalingappaa Shanmugamani. 2018. *Python: Advanced Guide to Artificial Intelligence: Expert machine learning systems and intelligent agents using Python*. Ed. Packt Publishing Ltd.

[20] Viola, P. & Jones, M.J. 2004. *International Journal of Computer Vision*.

[21] Jason Brownlee. 2019. *Deep Learning for Computer Vision: Image Classification, Object Detection, and Face Recognition in Python*. Ed. Machine Learning Mastery.

[22] Durai Pandian, Xavier Fernando, Zubair Baig, Fuqian Shi. 2019. *Proceedings of the International Conference on ISMAC in Computational Vision and Bio-Engineering 2018 (ISMAC-CVB*). Ed. Springer.

[23] Shuai Han, Liang Ye, Weixiao Meng. 2019. *Artificial Intelligence for Communications and Networks: First EAI International Conference, AICON 2019, Harbin, China, May 25–26, 2019, Proceedings, Parte 2*. Ed. Springer.

[24] Joseph Redmon, Ali Farhadi, University of Washington. *YOLOv3: An Incremental Improvement*.

[25] A. Bewley, Z. Ge, L. Ott, F. Ramos and B. Upcroft. 2016. "*Simple online and realtime tracking*,". IEEE International Conference on Image Processing (ICIP), Phoenix, AZ, 2016, pp. 3464-3468.

[26] Ruth Rubio, Julio Bernal. 2016. *Introducción a la lingüística computacional*. Ediciones de la U.

[27] Wei-Ming Lee. 2019. *Python Machine Learning*. Ed. John Wiley & Sons.

[28] Rafael Lahoz-Beltrá. 2004. *Bioinformática: simulación, vida artificial e inteligencia artificial.* Ediciones Díaz de Santos.

[29] Teresa Pintado Blanco. 2008. *Desarrollo de un Sistema Predictivo Para Productos de Alta Implicación, Basado en Variables Comportamentales. El Mercado de las Consolas de Videojuegos.* ESIC Editorial.

[30] Pratap Dangeti. 2017. *Statistics for Machine Learning*. Ed. Packt Publishing Ltd.

[31] Ryszard S. Michalski, Jaime G. Carbonell, Tom M. Mitchell. 2014. *Machine Learning: An Artificial Intelligence Approach (Volume I), Volumen 1*. Ed. Elsevier.

[32] Raúl Pino Díez, Alberto Gómez Gómez, Nicolás de Abajo Martínez. 2001. *Introducción a la inteligencia artificial: sistemas expertos, redes neuronales artificiales y computación evolutiva.* Ed. Universidad de Oviedo

[33] Lasse Rouhiainen. 2018. *Inteligencia artificial: 101 cosas que debes saber hoy sobre nuestro futuro*. Ed. Grupo Planeta.

[34] Raquel Flórez López, José Miguel Fernández Fernández. 2008. *Las Redes Neuronales Artificiales*. Ed. Netbiblo.

11. Anexos

11.1 Código de Detección de Objetos por Eliminación de Fondo:

```
import os
import cv2 as cv
frames2train = 500  #Cantidad de cuadros a utilizar para entrenamiento
IMAGE_DIR = '/cars/'
j = 1532
count = 0
current_path = os.getcwd()

cap = cv.VideoCapture('test_video.mp4')
fgbg = cv.createBackgroundSubtractorMOG2(history= frames2train,
                                         detectShadows=True)

def train_bg_subtractor(inst, cap, num = frames2train):

    i = 0
    for frame in cap.read():
        inst.apply(frame, None, 0.001)
        i += 1
        if i >= num:
            return cap

if not os.path.exists(IMAGE_DIR):
    os.makedirs(IMAGE_DIR)

while(1):

    ret, frame = cap.read()
    train_bg_subtractor(fgbg, cap, num=frames2train)
    cv.imwrite("Original-image.jpg", frame)
    fgmask = fgbg.apply(frame, None, 0.001)
    cv.imwrite("MOG2_BGRemover.jpg", fgmask)
```

```
################################################################################
#                    GAUSSIAN BLURR + OTSU THRESHOLD
################################################################################

    blur = cv.GaussianBlur(fgmask,(1,3),1)
    ret3,th = cv.threshold(fgmask,100,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
    cv.imwrite("Otsu-gauss-thres.jpg", th)


################################################################################
#                         SELECT KERNEL TO USE
################################################################################

    kernel = cv.getStructuringElement(cv.MORPH_ELLIPSE,(1,2))

################################################################################
#                              NOISE REMOVING
################################################################################

    opening2 = cv.morphologyEx(th, cv.MORPH_OPEN, kernel, iterations=1)
    closing2 = cv.morphologyEx(opening2, cv.MORPH_CLOSE, kernel, iterations=2)
    dilation2 = cv.dilate(closing2, kernel, iterations=2)
    cv.imwrite("Noise-remover.jpg", dilation2)
```

```
#############################################################################
#                        CONTOUR RETRIVING
#############################################################################

    contours2, hierarchy2 = cv.findContours(dilation2, cv.RETR_EXTERNAL,
                                                cv.CHAIN_APPROX_TC89_L1)

    for (i, contour) in enumerate(contours2):
        x,y,w,h = cv.boundingRect(contour)
        cX = x + (w/2)
        cY = y + (h/2)
        if h>=100:
            Trailer = cv.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)
        elif ((w>= 30) and (h<=40) and (w<=30)):
            bycicle = cv.rectangle(frame,(x,y),(x+w,y+h),(0,0,255),2)
        elif ((w>=50) and (h<=60) and (h>= 30) ) :
            car = cv.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)
        if w>50 and h>50:
            if count >= 150:
                if j <= 3000:
                    croped = frame[y:y+h,x:x+w]
                    resized = cv.resize(croped, (50, 50))
                    cv.imwrite(current_path+IMAGE_DIR+'%03d.jpg' %j, resized)
                    j += 1
                    count = 0
                    print (j)
                if j > 3000:
                    break
            count += 1
#############################################################################
#############################################################################
    cv.imshow('frame_car_detected', frame)
    cv.imwrite("Car_detected.jpg", frame)
    k = cv.waitKey(30) & 0xff
    if k == 27: #esc key
        break
cap.release()
cv.destroyAllWindows()
```

11.2 Código de Detección de Objetos por Medio de Características Haar:

```
import cv2
cascade_src = 'cars.xml'
video_src = 'test_video.mp4'
cap = cv2.VideoCapture(video_src)
car_cascade = cv2.CascadeClassifier(cascade_src)
while True:
    ret, img = cap.read()
    if (type(img) == type(None)):
        break
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    cars = car_cascade.detectMultiScale(gray, 1.1, 2)
    for (x,y,w,h) in cars:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,255),2)
    cv2.imshow('video', img)
    cv2.imwrite("Haar.jpg", img)
    if cv2.waitKey(33) == 27:
        break

cv2.destroyAllWindows()
```

11.3 Código de Detección de Objetos por medio de YOLO y SORT:

```
from models import *
from utils import *
import os, sys, time, datetime, random
import torch
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
from torch.autograd import Variable
from PIL import Image
import cv2
from sort import *
import numpy as np
from sklearn import preprocessing, neighbors
from sklearn.model_selection import train_test_split
import pandas as pd


def detect_image(img):
    # scale and pad image
    ratio = min(img_size/img.size[0], img_size/img.size[1])
    imw = round(img.size[0] * ratio)
    imh = round(img.size[1] * ratio)
    img_transforms = transforms.Compose([ transforms.Resize((imh, imw)),
         transforms.Pad((max(int((imh-imw)/2),0), max(int((imw-imh)/2),0), max(int((imh-imw)/2),0), max(int((imw-imh)/2),0)),
                        (128,128,128)), transforms.ToTensor()])
    # convert image to Tensor
    image_tensor = img_transforms(img).float()
    image_tensor = image_tensor.unsqueeze_(0)
    input_img = Variable(image_tensor.type(Tensor))

    # run inference on the model and get detections
    with torch.no_grad():
        detections = model(input_img)
        detections = utils.non_max_suppression(detections, 80, conf_thres, nms_thres)
    return detections[0]

def k_neighbors(position):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    clf = neighbors.KNeighborsClassifier(n_neighbors=5)
    clf.fit(X_train, y_train)
    #accuracy = clf.score(X_test, y_test)
    #print(accuracy)

    example_measures = np.array([position[0],position[1]])
    example_measures = example_measures.reshape(1, -1)
    prediction = clf.predict(example_measures)
    #print(prediction)
    return prediction[0]
```

```
# Load weights and set defaults
config_path='config/yolov3.cfg'
weights_path='config/yolov3.weights'
class_path='config/coco.names'
img_size=416
conf_thres=0.8
nms_thres=0.4
data_size = 300
Expand_X = 4
Expand_Y = 3

# Load model and put into eval mode
model = Darknet(config_path, img_size=img_size)
model.load_weights(weights_path)
model.eval()

classes = utils.load_classes(class_path)
Tensor = torch.FloatTensor

#Load Video and colors for detected vehicles
videopath = 'test_video_2.mp4'
#colors=[(255,0,0),(0,255,0),(0,0,255),(255,0,255),(128,0,0),(0,128,0),(0,0,128),(128,0,128),(128,128,0),(0,128,128)]
vid = cv2.VideoCapture(videopath)
mot_tracker = Sort()

#Create window to display and size
cv2.namedWindow('Stream',cv2.WINDOW_NORMAL)
cv2.resizeWindow('Stream', (800,600))
fourcc = cv2.VideoWriter_fourcc(*'XVID')

#Read video and check the original size
ret,frame=vid.read()
vw = frame.shape[1]
vh = frame.shape[0]
print ("Video size", vw,vh)

#Create Output video.
outvideo = cv2.VideoWriter(videopath.replace(".mp4", "-det(" + str(data_size) + ").mp4"),fourcc,20.0,(vw,vh))
frames = 0
starttime = time.time()

FOUND_PATHS = {}
past_save_pos = (0,0)
past_save_dim = (0,0)
past_save_center = (0,0)
count = 0

#Create files to save coordenates of detected objects
list_saved = open('list_acquired(' +str(data_size)+ ').txt' , 'w')
k_classifier_data = open('k_classifier_data.txt', 'w')
```

```
##create length of lists
detected = []
expand = []
result_matrix = []

#Create document with original matrix and put headers
k_classifier_data.write('X_position' + ',' + 'Y_position' + ',' + 'Class' + '\n')

while(True):
    ret, frame = vid.read()

    if not ret:
        break
    frames += 1
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    pilimg = Image.fromarray(frame)
    detections = detect_image(pilimg)

    frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
    img = np.array(pilimg)
    pad_x = max(img.shape[0] - img.shape[1], 0) * (img_size / max(img.shape))
    pad_y = max(img.shape[1] - img.shape[0], 0) * (img_size / max(img.shape))
    unpad_h = img_size - pad_y
    unpad_w = img_size - pad_x

    if detections is not None:
        tracked_objects = mot_tracker.update(detections.cpu())
        unique_labels = detections[:, -1].cpu().unique()
        n_cls_preds = len(unique_labels)

        for x1, y1, x2, y2, obj_id, cls_pred in tracked_objects:
            box_h = int(((y2 - y1) / unpad_h) * img.shape[0])
            box_w = int(((x2 - x1) / unpad_w) * img.shape[1])
            y1 = int(((y1 - pad_y // 2) / unpad_h) * img.shape[0])
            x1 = int(((x1 - pad_x // 2) / unpad_w) * img.shape[1])

#########################################################################################################################################
#################################### SAVE DATA###########################################################################################
#########################################################################################################################################

            (saveX, saveY) = (x1, y1)
            (saveW, saveH) = (box_w, box_h)
            centerX = int((x1+(box_w/2)))
            centerY = int((y1+(box_h/2)))
            centroid = (int(centerX), int(centerY))

            #Save as list
            car_ID = obj_id
            if int(obj_id) < data_size:
                if past_save_center != (centroid):
                    aux = []
                    detected.append(centroid)
                past_save_pos = (saveX, saveY)
                past_save_dim = (saveW, saveH)
                past_save_center = centroid
```

```
            if int(obj_id) > data_size:
                if count == 0:
                    for detect in detected:
                        detectX = detect[0]
                        detectY = detect[1]
                        for ExpX in range(Expand_X):
                            for ExpY in range(Expand_Y):
                                newX = detectX + ExpX
                                newY = detectY + ExpY
                                k_classifier_data.write(str(newX) + ',' + str(newY) + ',1' + '\n')
                                expand.append((newX,newY))
                        for ExpX in range(Expand_X):
                            for ExpY in range(Expand_Y):
                                newX = detectX - ExpX
                                newY = detectY - ExpY
                                k_classifier_data.write(str(newX) + ',' + str(newY) + ',1' + '\n')
                                expand.append((newX,newY))
                    for X in range((int(vw)+1)):
                        for Y in range((int(vh)+1)):
                            position = (X, Y)
                            if position in expand:
                                pass
                            else:
                                result_matrix.append(position)
                                k_classifier_data.write(str(X) + ',' + str(Y) + ',2' + '\n')
                    df = pd.read_csv('k_classifier_data.txt')
                    X = np.array(df.drop(['Class'],1))
                    y = np.array(df['Class'])
                    count =+ 1

                prediction = k_neighbors(centroid)

                if prediction == 2:
#                    croped2 = frame[y1:y2+box_h,x1:x1+box_w]
#                    cv.imwrite('%03d.jpg' %j, croped2)
                    cv2.rectangle(frame, (x1, y1), (x1+box_w, y1+box_h), (0,0,255), 4)
                    #cv2.putText(frame, str(int(obj_id)), (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 2)
                    j =+ 1


#Create Box color and ID tag
            #color = colors[int(obj_id) % len(colors)]
            cls = classes[int(cls_pred)]
            #cv2.rectangle(frame, (x1, y1), (x1+box_w, y1+box_h), (255,255,255), 4)
            cv2.putText(frame, cls + '-' + str(int(obj_id)), (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 2)

#If 'Esc' key is pressed stop process
    cv2.imshow('Stream', frame)
    outvideo.write(frame)
    ch = 0xFF & cv2.waitKey(1)
    if (ch == 27):
        break

#Save documents created.
list_saved.write(str(detected) + '\n')
```

```
#Print values obtained.
print('count = ', count)
print('objects detected = ', int(obj_id))


totaltime = time.time()-starttime
print(frames, "frames", totaltime/frames, "s/frame")
cv2.destroyAllWindows()
outvideo.release()
```