0424-4219896　0412-3465251

UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA DE TELECOMUNICACIONES
DEPARTAMENTO DE SEÑALES Y SISTEMAS

DESARROLLO DE UN SISTEMA DE AFORO VEHICULAR MEDIANTE PROCESAMIENTO DIGITAL DE VIDEO

BLANCO BRITO OSCAR L.
HERNANDEZ ZAFRA EDWING J.

Bárbula, 2 de diciembre del 2016

UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA DE TELECOMUNICACIONES
DEPARTAMENTO DE SEÑALES Y SISTEMAS

DESARROLLO DE UN SISTEMA DE AFORO VEHICULAR MEDIANTE PROCESAMIENTO DIGITAL DE VIDEO

TRABAJO ESPECIAL DE GRADO PRESENTADO ANTE LA ILUSTRE UNIVERSIDAD DE CARABOBO PARA OPTAR AL TÍTULO DE INGENIERO DE TELECOMUNICACIONES

BLANCO BRITO OSCAR L.
HERNANDEZ ZAFRA EDWING J.

Bárbula, 2 de diciembre del 2016

Dedicatoria

A mis padres por haberme apoyado en todo
momento, por sus consejos, sus valores
y la motivación constante.
A mis familiares y seres queridos,
por su apoyo y afecto.
A Dios por haberme dado salud y haberme
permitido lograr mis objetivos.

BLANCO BRITO OSCAR L.

A mi madre por el apoyo incondicional y siempre
estar ahí para apoyarme y alentarme
a terminar lo que empiezo.
A mis dos padres y hermanos por siempre haber
estado de una u otra forma durante
las etapas de mi carrera.

HERNANDEZ ZAFRA EDWING J.

Agradecimientos

Durante la elaboración de este trabajo de investigación contamos con la colaboración de muchas personas. En primer lugar le agradecemos a Dios, a nuestros padres y familiares por siempre apoyarnos y estar presentes en todo momento y a nuestro tutor, el Ing. Carlos Aponte por guiarnos durante la realización de esta investigación, dandonos todo su apoyo.

Tambien hacemos extensivo este agradecimiento a:

Ing. Bettys Farias por brindarnos sus conocimientos y tiempo a lo largo de toda esta investigación.

Los profesores Lic. Pedro Linares, Ing. Dimas Véliz, Ing. Angel Villegas e Ing. Willmer Sanz por disponer de su tiempo en tantas ocaciones, compartiendo sus conocimientos y opiniones.

Los Ing. Carlos Nuñez y Daniel Lelli de la alcaldia de los municipios San Diego y Valencia, respectivamente.

Los Ing. Daniel Marquez y Luis Rámirez por brindarnos apoyo y consejos durante la realización de este arduo trabajo.

Daniela Marcano y su familia por apoyarnos en todo momento.

David Blanco y Luz Dary Camargo por su apoyo, tiempo, espacio físico para desarrollar la tesis.

Todos nuestros amigos y seres queridos que de una u otra manera estuvieron presentes a lo largo de este trabajo.

Índice general

Índice de Figuras XI

Índice de Tablas XIII

Índice de Códigos XV

Resumen XVII

I. Introducción 1
- 1.1. Motivación . . . 1
- 1.2. Objetivos . . . 4
  - 1.2.1. Objetivo General . . . 4
  - 1.2.2. Objetivos Específicos . . . 4
- 1.3. Alcance . . . 4

II. Marco conceptual 7
- 2.1. Ingeniería de Tráfico . . . 7
  - 2.1.1. Volumen e Intensidad de Circulación . . . 7
  - 2.1.2. Medición . . . 8
- 2.2. Aforo Vehicular . . . 8
  - 2.2.1. Conteo Manual . . . 9
  - 2.2.2. Conteo Mecánico . . . 9
- 2.3. Procesamiento Digital de Imágenes . . . 9
  - 2.3.1. Binarización de una imágen . . . 10
  - 2.3.2. Modificación del contraste . . . 10
  - 2.3.3. Modificación del histograma . . . 11
  - 2.3.4. Filtrado de una imágen . . . 11
    - 2.3.4.1. Realce de bordes . . . 11
    - 2.3.4.2. Detección de contornos . . . 11

III. Procedimientos de la investigación 13
- 3.1. Fase 1: Revisión teórica . . . 13

3.1.1. Actividad A. Recopilar documentos e información referente al conteo manual vehicular del Departamento de Vialidad de la Escuela de Ingeniería Civil . . . 13
3.1.2. Actividad B. Seleccionar el algoritmo de procesamiento de imágenes orientado al aforo vehicular . . . 14
3.1.3. Actividad C. Comparar los diferentes métodos de detección de objetos en escenas de tránsito . . . 15
3.2. Fase 2: Desarrollo . . . 18
3.2.1. Actividad D. Obtener los videos de tráfico vehicular para la ejecución del sistema . . . 18
3.2.2. Actividad E. Diseñar el software de aforo vehicular . . . 18
3.2.2.1. Soporte para el desarrollo del sistema . . . 18
3.2.2.2. Módulo de inicialización . . . 19
3.2.2.3. Módulo de procesamiento . . . 21
3.2.2.4. Módulo de generación de resultados . . . 30
3.2.2.5. Interfaz gráfica . . . 31
3.2.3. Actividad F. Evaluar el desempeño del sistema de aforo vehicular diseñado con base en el tiempo de procesamiento y exactitud de conteo según la clasificación establecida . . . 32

IV. Análisis, interpretación y presentación de los resultados **33**
4.1. Aplicación del sistema de aforo vehicular a video de transito . . . 33
4.1.1. Análisis del tiempo de procesamiento . . . 34
4.1.2. Análisis de la exactitud de conteo . . . 36

V. Conclusiones y recomendaciones **45**
5.1. Conclusiones . . . 45
5.2. Recomendaciones . . . 46

A. Sistema aforo vehicular mediante procesamiento digital de video **49**
1.1. Página inicial . . . 49
1.2. Página de selección de bordes . . . 51
1.3. Página de identificación de bordes . . . 52
1.4. Página de configuración del aforo vehicular . . . 54
1.5. Página de aforo . . . 54
1.6. Página de resultados . . . 56
1.7. Archivos resultantes . . . 57

B. Códigos del sistema de aforo vehicular desarrollado **59**
2.1. Código Principal . . . 59
2.2. Módulo Principal . . . 72
2.3. Módulo de Rastreo . . . 80

2.4. Módulo Básico 117

Referencias Bibliográficas **133**

Índice de figuras

3.1. Diagrama general del algoritmo de aforo vehicular mediante procesamiento digital de video 16
3.2. Esquema general del diseño el software de aforo vehicular. 19
3.3. Generación de fondo promedio. 20
3.4. Diagrama general del módulo de detección de aforo vehicular mediante procesamiento digital de video 21
3.5. Proceso de detección vehicular. (a) Fondo. (b) Imagen original. (c) Imagen convertida al espacio de color gris. (d) Resta entre el fondo y la imagen en escala de gris. (e) Imagen binarizada con el método de detección de bordes Canny. (f) Imagen 'e' dilatada. (g) Imagen 'f' tras realizar el proceso de llenado de bordes. (h) Imagen 'g' erosionada. (i) Detección de un vehículo, el rectángulo que lo enmarca y el centro del rectángulo 22
3.6. Proceso de descarte de vehículos en contacto con el borde. (a) Imagen de la etapa anterior. (b) Imagen 'a' con los bordes pintados en blanco. (c) Imagen 'b' pintada de blanco fuera de la zona de estudio. (d) Imagen 'c' pintada de negro fuera de la zona de estudio. 24
3.7. Convergencia entre bloques. 26
3.8. Divergencia entre bloques. 26
3.9. Combinación compleja entre bloques. 27

4.1. Gráfica de error porcentual respecto al porcentaje de segundos con 5 cuadros recibidos o menos de los 4 intervalos analizados. 42

1.1. Página Inicial de la interfaz gráfica. 50
1.2. Página Inicial al ser valida la extensión del video. 50
1.3. Página Inicial al ser invalida la extensión del video. 51
1.4. Página de selección de bordes de la zona de estudio. 51
1.5. Página de selección de bordes luego de haber sido seleccionados. . . 52
1.6. Página de identificación de bordes de intersección. 53
1.7. Página de identificación de bordes una vez identificados los bordes de la intersección. 53
1.8. Página de configuración de la interfaz gráfica. 54
1.9. Página de aforo de la interfaz gráfica. 55

1.10. Página de culminación del proceso de aforo de la interfaz gráfica. . . 56
1.11. Carpeta del sistema con los archivos generados con los resultados. . 57
1.12. Página de culminación del proceso de aforo de la interfaz gráfica si se presiona el botón Detener. 57
1.13. Histograma resultante de nombre Histograma_Aforo_Vehicular.html. 58
1.14. Tabla resumen resultante de nombre Resultado_Aforo.xlsx. 58

Indice de tablas

4.1. Resultados obtenidos en base a los tiempo de procesamiento. 34
4.2. Resultados obtenidos de los diferentes tipos de aforo. 35
4.3. Resultados obtenidos en la ejecución del sistema de aforo vehicular utilizando el video en estudio. 36
4.4. Resultados obtenidos en la ejecución del conteo vehicular de forma manual. 36
4.5. Comparación de los resultados obtenidos de aforo vehicular expresado en error absoluto. 37
4.6. Resultados de exactitud obtenidos para los vehículos que cruzan hacia el frente en los intervalos donde tienen permitido el paso los vehículos. 38
4.7. Resultados de cuadros recibidos por segundo del intervalo 1. 39
4.8. Resultados de cuadros recibidos por segundo del intervalo 7. 39
4.9. Resultados de cuadros recibidos por segundo del intervalo 13. 39
4.10. Resultados de cuadros recibidos por segundo del intervalo 17. 40
4.11. Resultados de análisis del intervalo 1. 40
4.12. Resultados de análisis del intervalo 7. 41
4.13. Resultados de análisis del intervalo 13. 41
4.14. Resultados de análisis del intervalo 17. 41

Índice de códigos

aforo_gui.py 59
modulo_aforo.py 72
modulo_tracking.py 80
modulo1.py 117

DESARROLLO DE UN SISTEMA DE AFORO VEHICULAR MEDIANTE PROCESAMIENTO DIGITAL DE VIDEO

por

BLANCO BRITO OSCAR L. y HERNANDEZ ZAFRA EDWING J.

Presentado en el Departamento de Señales y Sistemas
de la Escuela de Ingeniería en Telecomunicaciones
el 2 de diciembre del 2016 para optar al Título de
Ingeniero de Telecomunicaciones

RESUMEN

En la Universidad de Carabobo - Facultad de Ingeniería - Escuela de Ingeniería Civil - Departamento de Vialidad, los alumnos realizan semestralmente un aforo vehicular manual con la finalidad de contar con datos de escenarios reales para analizar el nivel de servicio de intersecciones semaforizadas. Para aliviar la problemática y el factor error por causa de conteo manual, se plantea el desarrollo de un sistema de aforo vehicular mediante procesamiento digital de video, lo que supone el conteo vehicular acorde a determinado hora y duración. Para lograr este objetivo, se analizó el tráfico vehicular en las intersecciones, se realizó un estudio en el área de procesamiento de imágenes orientado a la detección vehicular, se diseñó el sistema de aforo vehicular mediante procesamiento de imágenes, utilizando librería

de visión por computador OpenCV en lenguaje Python. Se obtuvo que el sistema es capaz de analizar el video en menor tiempo en comparación al requerido por los método de aforo manual, también se observó como un video con segundos con pocos cuadros afecta significativamente los resultados de aforo.

Palabras Claves: Aforo Vehicular, Procesamiento Digital de Señales, Procesamiento Digital de Imágenes, Detección de Objetos

Tutor: APONTE DEZZEO CARLOS D.
Profesor del Departamento de Señales y Sistemas
Escuela de Telecomunicaciones. Facultad de Ingeniería

Capítulo I

Introducción

1.1. Motivación

El tránsito vehicular es un tema que precisa de estudio debido a la gran cantidad de automóviles que circulan en las vías [1], generando congestión y con ella, sus típicas consecuencias [2]. Por tanto, al momento de diseñar una nueva infraestructura (ya sea una avenida, centro comercial, hotel, etc), reestructurar una vialidad existente, programar el mantenimiento asfáltico, estimar la emisión de gases o reprogramar un semáforo, es imprescindible realizar un Estudio de Impacto Vial (EIV), cuyo principal objetivo es proveer la circulación vehicular de manera segura, rápida y eficiente [3].

Al realizar un EIV es necesario efectuar un aforo vehicular, el cual también se conoce como conteo vehicular y representa el principal método de recolección de datos a ser analizados. Así pues, una diversa gama de usuarios requieren de dicha herramienta, entre ellos se encuentran las direcciones de tránsito y vialidad, compañías de ingeniería, constructoras, consultorías, empresas de mercadotecnia y cualquier otra empresa que requiera un aforo vehicular para un EIV, previo o posterior a algún proyecto [4].

Ahora bien, el aforo vehicular es el proceso de separar vehículos de acuerdo a diferentes clases predefinidas e intervalos de tiempo, obteniendo como resultado

un histograma que muestra la distribución de automóviles que circulan un cierto trayecto por lapso de tiempo, para cada una de las clases definidas [3].

Con respecto a la Universidad de Carabobo, en la Facultad de Ingeniería, en la Escuela de Ingeniería Civil, en el Departamento de Vialidad semestralmente se realiza el proceso de aforo vehicular de forma manual a través de censos visuales. Esto con la finalidad de contar con datos de escenarios reales para analizar el nivel de servicio de intersecciones semaforizadas, es parte de los objetivos a cumplir en la asignatura Vías de Comunicación I. Esta tarea de conteo la realizan los estudiantes de dicha asignatura en un tiempo aproximado de dos a tres horas diarias (horas picos), durante una semana de estudio, según consultas realizadas a profesores del departamento.

No obstante, el mecanismo de conteo manual presenta varias desventajas, como lo es su alto costo por la necesidad de disponer de personal cualificado en una cantidad acorde al tipo de clasificación necesaria durante varias horas al día, propenso al error humano por descuidos momentáneos u otras causas y la limitación temporal por factor humano que genera incertidumbre en los datos a analizar, siendo este último el mayor inconveniente [5][6].

En cuanto a las herramientas que suelen ser utilizadas para solventar los inconvenientes generados del conteo manual, la video detección ofrece la mayor cantidad de ventajas, obteniendo mejor resultado en el reconocimiento vehicular [7]; siendo las cámaras una de las herramientas más simples de instalar, ubicadas en la categoría de sensores no intrusivos [8], es decir, no afecta al tránsito durante su instalación o medición, permitiendo una detección óptima y una solución económica. En otras palabras, esta es una herramienta versátil que le permite al usuario simplificar el trabajo y disminuir el tiempo de procesamiento, impulsada por el avance tecnológico en el área de la visión por computador, además de contar vehículos, proporcionar datos como velocidad, tipo de vehículo, densidad, reconocimiento de placa, entre otros [5][9].

El trabajo de Quesada (2015) [10] sobre un algoritmo de estimación del número de elementos móviles en videos digitales orientado a la gestión del tráfico vehicu-

lar, muestra un estudio sobre conteo vehicular utilizando videos con fondo estático de la base de datos Lankershin Boulevard, detectando a los vehículos y contándolos al pasar por una línea marcada por el programador. Los resultados fueron satisfactorios, disminuyendo el costo computacional, arrojando un error porcentual muy próximo a otros métodos actuales, generando el aporte de la implementación del algoritmo Principal Component Pursuit (PCP) y recomendaciones de gran importancia para futuros estudios.

La clasificación en estudios previos de detección y conteo vehicular se han realizado en tramos de vía recta, cuyo criterio principal de clasificación ha sido el tamaño de los vehículos, por tal motivo se decide explorar el criterio de sentido de giro como clasificador vehicular, visto que es un clasificador de gran importancia en intersecciones viales, que proporciona información para el diseño de intersecciones, análisis de maniobras de tránsito, evaluación de impacto regional, programación de semáforos, estudios de velocidad, capacidad vial y nivel de servicio, reasfaltado, restauración y rehabilitación de vialidad y la construcción, rehabilitación y modificación de elevados [11][12][13][14].

El método manual de aforo vehicular utilizado por los estudiantes de la Escuela de Ingeniería Civil y muchos otros profesionales involucrados en esta área, se sustituirá por el sistema automático de detección y conteo vehicular basado en video que ofrece muchas ventajas sobre el proceso manual tales como: la posibilidad de extender la duración de los estudios a más de dos horas diarias, minimizar la asistencia humana en la etapa de conteo y una mayor consistencia y confiabilidad en la tasa de error. Con respecto a la ejecución semestral de aforo vehicular por los estudiantes de la facultad, una herramienta automática permitirá disminuir el grado de incertidumbre presente por el aforo manual y así realizar el análisis con datos más reales.

El principal aporte de este proyecto es realizar un sistema de aforo vehicular sobre un criterio de clasificación diferente a los definidos en otros estudios de conteo vehicular. Así mismo, distinguir el sentido que toman los automóviles al entrar a una intersección semaforizada es una característica esencial en los análisis de tránsito ubicados en intersecciones.

1.2. Objetivos

1.2.1. Objetivo General

Desarrollar un sistema de aforo vehicular mediante procesamiento digital de video.

1.2.2. Objetivos Específicos

- Analizar el proceso manual de aforo vehicular.
- Seleccionar el algoritmo de procesamiento de imágenes orientado al aforo vehicular.
- Diseñar el software para el aforo vehicular.
- Validar el desempeño del sistema de aforo vehicular diseñado.

1.3. Alcance

El presente trabajo se orienta al estudio de capturas de video a color (luz visible) de escenas de tránsito vehicular que presenten las siguientes características:

- Video con fondo estático.
- La toma debe ser realizada desde una altura superior a 12 m, esto con el fin de evitar oclusión entre vehículos, ya que esta situación queda fuera del estudio a realizar.
- Las grabaciones deben ser en horario diurno con buena iluminación, por el motivo de la perdida en información que se produce en escenas de poca iluminación.
- La nitidez del video debe permitir identificar los contornos de los vehículos.

- Se busca que el conteo sea lo más exacto posible, por ello el sistema recibirá videos pre-grabados, dado que resulta de más utilidad este enfoque para los estudiantes e ingenieros civiles.
- Se utilizará una grabación de la intersección Av. Julio Centeno, del Municipio San Diego, Estado Carabobo, proporcionada por el Instituto Autónomo Municipal Policía de San Diego, debido a que dicho instituto cuenta con cámaras ya instaladas en torres de iluminación lo suficientemente altas y cerca de la intersección.

Una vez realizado el procesamiento del video, los resultados obtenidos del aforo vehicular serán mostrados a través de un histograma y un archivo de texto, organizado de manera similar al cuadro utilizado por el Departamento de Vialidad en el proceso manual de aforo vehicular.

Capítulo II

Marco conceptual

2.1. Ingeniería de Tráfico

La ingeniería del tráfico es una rama de la ingeniería civil, que se encarga de planear, diseñar y organizar la operación del tráfico en calles y autopistas, con el fin de obtener una movilidad segura y eficiente de vehículos. Como herramienta en el diseño de vías, se utilizan modelos que son diseñados basándose en muestras de varios parámetros de tráfico:

2.1.1. Volumen e Intensidad de Circulación

Son dos medidas que indican el número de vehículos que pasan por un segmento de vía durante un intervalo de tiempo determinado.

El volumen se define como el número total de vehículos que pasan por una determinada sección de la carretera en un intervalo dado. Ya sean años, días, horas o menos. Este es un valor real medido directamente en la vía en el tiempo total de medición.

La intensidad horaria se define como el número de vehículos que pasan por un segmento de vía durante un intervalo de tiempo inferior a una hora, pero expresado como una intensidad horaria equivalente. Se obtiene dividiendo el volumen

registrado en un periodo entre la duración del mismo expresado en horas "vehículos/hora" [15].

2.1.2. Medición

Para llevar un registro de los parámetros de la vía en estudio, se usan diversos sistemas de monitoreo como detectores magnéticos, tubos de precisión, pistolas radar, sensores de microondas, conteos manuales y visión artificial. De lejos, los detectores magnéticos son los más utilizados para el monitoreo de sistemas de tráfico. Son instalados en el suelo, sobre la superficie de la vía para contar el número de carros [15].

Estos sistemas solo permiten recopilar una cantidad limitada de información relacionada con el tráfico, mientras que la visión artificial podría reemplazar simultáneamente a varios de estos sistemas. Además de contar vehículos y medir la velocidad, se puede proporcionar ruta del vehículo, tipo de vehículo, tasa de flujo de vehículos, congestiones de tráfico e identificación de vehículos por número de placas. Otro tipo de información a largo plazo que se podría tomar sería tiempo de desplazamiento en ciertas rutas, la longitud y tiempo de espera en una intersección, el número de cambios de carril y la rápida aceleración o desaceleración de vehículos

2.2. Aforo Vehicular

El conteo de tráfico vehicular es realizado con el propósito de obtener información relacionada con el movimiento de vehículos sobre puntos o secciones específicas dentro de un sistema vial. Estos datos son expresados con respecto al tiempo y de su conocimiento se hace posible el desarrollo de estimaciones razonables de la calidad de servicio prestado a los usuarios [16].

Generalmente se realiza en el punto o tramo de carretera de interés, la cual se realizará durante todo el día, fijando mayor énfasis a la hora pico. Como sabemos,

son dos picos de afluencia vehicular, una es cuando los usuarios de la vía se dirigen a su lugar de trabajo o estudio y la otra cuando retornan a sus hogares[16].

2.2.1. Conteo Manual

En su forma más simple requiere de una persona que anote el número de autos que circulan por el punto o tramo de estudio, en intervalos de tiempo de 15 minutos, manejando los movimientos por dirección y por tipo de vehículo, el cual se registra en una hoja de campo. En el registro se realiza un croquis del movimiento con respecto a la dirección del norte. La clasificación de los vehículos puede ser tan simple como la distinción entre automóvil y camión. Se puede utilizar una descripción más detallada de los vehículos comerciales (camiones), por número de ejes y peso. Cabe destacar que a mayor distinción entre vehículos y mayor afluencia, es necesario disponer de aproximadamente 15 personas [16].

2.2.2. Conteo Mecánico

Son contadores que funcionan de forma automática sobre la vía, los cuales transmiten impulsos o señales por los vehículos que pasan. Este mecanismo debe ser considerado en la mayoría de los aforos en que se requieren más de 12 horas de datos continuos del mismo lugar. Sirve además para determinar la variación horaria en particular y selecciona la hora de máxima demanda vehicular.

De este tipo de aforo existen varios tipos como lo son detectores magnéticos, tubos de precisión, pistola radar, sensores de microondas y detección por computadora mediante procesamiento digital de señales [16].

2.3. Procesamiento Digital de Imágenes

El procesamiento digital de imágenes es el conjunto de técnicas que se aplican a las imágenes digitales con el objetivo de mejorar la calidad o facilitar la búsqueda

de información dentro de la misma. Antes de extraer la información directamente de la imágen, se acostumbra a ejecutar un procesamiento previo para obtener otra que nos permita realizar la extracción de datos, más sencilla y eficientemente [17].

2.3.1. Binarización de una imágen

Consiste en comparar los niveles de gris presentes en la imágen con un valor (umbral) predeterminado. Si el nivel de gris de la imágen es menor que el umbral predeterminado, se le asigna al píxel de la imágen binarizada el valor 0 (negro), y si es mayor, se le asigna un 1 (blanco); de esta forma se obtiene una imágen en blanco y negro. Generalmente se utiliza un umbral de 128 si se trabaja con 256 niveles de gris, sin embargo, en algunas aplicaciones se requiere de otro umbral [17].

2.3.2. Modificación del contraste

La modificación del contraste consiste en aplicar una función a cada uno de los píxeles de la imágen, de la forma: $p = (m)^a$ donde:

- m es el valor de gris de la imágen original.
- p es el nuevo valor de gris en la imágen resultante.
- a es la potencia a la que se eleva.

El valor 255 se utiliza para normalizar los valores entre 0 y 255 si se trabaja con imágenes con niveles de gris de 8 bits, de lo contrario se debe remplazar este valor por el valor máximo representable con el número de bits utilizados. Con la función cuadrada y cúbica se oscurece la imágen resultante. Con las funciones raíz cuadrada, raíz cúbica y logarítmica sucede lo inverso [17].

2.3.3. Modificación del histograma

Si se desea adquirir información global de la imágen, la forma más fácil de hacerlo es analizar y modificar el histograma. Esto se hace con la idea de que éste se ajuste a una forma predeterminada; la forma más usual se conoce como ecualización del histograma, en la que se pretende que éste sea horizontal, es decir, que para todos los valores de gris se tenga el mismo número de píxeles [17].

2.3.4. Filtrado de una imágen

El filtrado es una técnica para modificar o mejorar a una imágen. Por ejemplo, un filtro puede resaltar o atenuar algunas características. El filtrado es una operación de vecindario, en la cual el valor de un píxel dado en la imágen procesada se calcula mediante algún algoritmo que toma en cuenta los valores de los píxeles de la vecindad de la imágen original [17].

2.3.4.1. Realce de bordes

El realce de bordes en una imágen tiene un efecto opuesto a la eliminación de ruido; consiste en enfatizar o resaltar aquellos píxeles que tienen un valor de gris diferente al de sus vecinos. Cabe resaltar que si la imágen contiene ruido, su efecto se multiplicará, por lo que ser recomienda primero eliminar el ruido [17].

2.3.4.2. Detección de contornos

La detección de contornos es un paso intermedio en el reconocimiento de patrones en imágenes digitales. En una imágen, los contornos corresponden a los limites de los objetos presentes en la imágen. Para hallar los contornos se buscan los lugares en la imágen en los que la intensidad del píxel cambia rápidamente, generalmente usando alguno de los siguientes criterios:

1. Lugares donde la primera derivada (gradiente) de la intensidad es de magnitud mayor que la de un umbral predefinido.

2. Lugares donde la segunda derivada (laplaciano) de la intensidad tiene un cruce por cero.

En el primer caso se buscarán grandes picos y en el segundo cambios de signo [17].

Capítulo III

Procedimientos de la investigación

En este capítulo se establecen los métodos y las técnicas para desarrollar un sistema de aforo vehicular mediante procesamiento digital de video, llevado a cabo en dos faces:

3.1. Fase 1: Revisión teórica

Durante esta fase se realizaron las siguientes actividades con la finalidad de recolectar toda la información posible para conocer diferentes esquemas y tipos de detección vehicular mediante procesamiento digital de video. De esta forma crear el soporte de contenidos, recursos y algoritmos, así como la selección de las herramientas de programación a utilizar.

3.1.1. Actividad A. Recopilar documentos e información referente al conteo manual vehicular del Departamento de Vialidad de la Escuela de Ingeniería Civil

Se procedió a realizar una revisión bibliográfica sobre el aforo vehicular para conocer acerca de su metodología. Se consultaron libros especializados en el tema, así

como también diversas tesis de pre-grado, post-grado y artículos con la finalidad de reforzar los conceptos básicos y necesarios relacionados con el aforo vehicular.

También se realizaron visitas al Departamento de Vialidad, en la Escuela de Ingeniería Civil, en la Facultad de Ingeniería, de la Universidad de Carabobo (donde semestralmente se realiza el proceso de aforo vehicular a través de censos manuales) con el Profesor Dimas Véliz, quien recomendó el manual del departamento y planillas de resultado utilizadas en el conteo manual, explicó la metodología del aforo realizado en su cátedra, limitaciones y desventajas del mismo.

3.1.2. Actividad B. Seleccionar el algoritmo de procesamiento de imágenes orientado al aforo vehicular

La selección del algoritmo de procesamiento de imágenes orientado al aforo vehicular consistió en la realización de cursos online de detección de objetos en dos sitios web: coursera.com (Deteccion de Objetos) y edx.org (Introducción a la visión por computador: OpenCV) donde se conocieron los diferentes tipos de métodos y sistemas utilizados para el procesamiento de imágenes y como ser orientados a la detección de objetos por computador.

Posteriormente se realizaron lecturas de artículos y libros sobre los temas referentes a detección vehicular mediante procesamiento digital de imágenes, con el fin de adquirir los conocimientos necesarios para el desarrollo del sistema. Del mismo modo, se realizaron visitas al Departamento de Computación, Facultad Experimental de Ciencias y Tecnologías (FACYT), de la Universidad de Carabobo con Profesor Pedro Linares, quien guió en la búsqueda de información pertinente al tema.

Se realizó un estudio de investigación de los diferentes tipos de lenguaje de programación (Java, C++, Python, MATLAB) y librerías de detección de objetos (OpenCV, MATLAB, PIL, SimpleCV). De los cuales se seleccionó del lenguaje Python y librería OpenCV como lo más idóneo para la realización del algoritmo de procesamiento de imágenes, debido a:

Python:

- Software libre
- Multi-plataforma
- Orientado a objetos
- Alto nivel

OpenCV:

- Librería libre de visión por computador
- Robusta en procesamiento de imágenes
- Mas de 500 funciones que abarcan diversas áreas de visión por computador
- Gran cantidad de documentación en la web

Una vez definido el lenguaje de programación y librería a utilizar, se planteó el esquema general, mostrado en la Figura 3.1, el cual separa al sistema en 2 etapas:

- *Inicialización:* Extrae información previa del video necesaria para su procesamiento.
- *Procesamiento:* Analiza cada imagen del video.

3.1.3. Actividad C. Comparar los diferentes métodos de detección de objetos en escenas de tránsito

Dado el esquema general definido en la Figura 3.1, en esta actividad se procedió a comparar los diferentes métodos de detección de objetos en escenas de tránsito, realizando la revisión y selección de los métodos de detección a ser incluidos en la etapa de procesamiento, seleccionando los métodos más significativos en detección vehicular, para luego evaluar el rendimiento mediante simulaciones de pruebas y codificarlos.

De esta revisión se obtuvieron los siguientes métodos de detección de objetos:

Figura 3.1: Diagrama general del algoritmo de aforo vehicular mediante procesamiento digital de video

- Descriptor simple
- Template Matching
- HAAR
- Local Binary Patterns (LBP)
- Regresión Logística
- Métodos de detección de bordes
- Umbralización
- Resta de fondos

Luego de esto, se realizó un estudio de las posibilidades existentes para implementar los métodos en escenas de tránsito. Quedando como viables HAAR, métodos de detección de bordes y resta de fondos.

Luego de esta selección preliminar, se realizaron simulaciones de prueba para cada uno de los métodos seleccionados anteriormente.

El método de HAAR fue descartado dado que debe ser previamente entrenado con una base de datos que consta de imágenes separadas en dos grupos: un grupo de imágenes con los objetos a detectar y otro grupo con imágenes donde estos objetos no aparecen; la base de datos debe contener al menos 3000 imágenes para un entrenamiento óptimo, las imágenes del primer grupo sólo deben contener al objeto a identificar (el objeto ocupa toda la imagen). Es decir, seria necesario buscar mas de 1500 imágenes de vehículos con la perspectiva que requiere el programa y acondicionarlas. Al ser tan trabajoso generar una base de datos propia, se buscó una base de datos en la web sin tener éxito, pues ninguna compartía la misma perspectiva del video a utilizar. Debido a esta limitante, se probó con varios archivos .xml generados por cascadas ya entrenadas, obteniendo resultados nada satisfactorios.

Dentro de los métodos basados en características, se encuentra la detección de bordes, el cual permite identificar el contorno de los vehículos. Entre los posibles métodos de detección de bordes se encuentra Canny, Laplaciano, Sobel, entre otros, los cuales se pusieron a prueba y se obtuvieron mejores resultados con Canny, pues se caracteriza por ser más estricto y preciso a la hora de dibujar el contorno, gracias a la implementación de dos umbrales para seleccionar los píxeles que pertenecen a un borde en la imagen, comparado a otros métodos que detectaban bordes erróneamente en zonas donde no había vehículos. Hasta este punto se logró detectar los vehículos relativamente bien, presentando errores a lo largo del video.

Finalmente se decidió probar la robustez de implementar en conjunto los métodos basados en características y resta de fondo. Este último al restar cada imagen del video con una imagen de fondo (imagen sin vehículos dentro la intersección) resulta una imagen negra con zonas en gris donde se presentaron diferencias con el fondo, que al aplicarle Canny permite binarizar la imagen de forma robusta a lo largo del video.

En resumen, los métodos seleccionados para realizar la detección vehicular fue la resta de fondo y la detección de bordes con Canny. Lo siguiente fue acondicionar la imagen resultante con operaciones morfológicas, obteniendo una imagen negra con las zonas en blanco donde se encuentre un vehículo.

3.2. Fase 2: Desarrollo

Una vez realizada la fase de análisis, se procede al desarrollo del sistema y su evaluación. Lo que se realizó de la siguiente manera:

3.2.1. Actividad D. Obtener los videos de tráfico vehicular para la ejecución del sistema

La obtención de los videos para la ejecución del sistema se llevó acabo mediante el estudio inherente al reconocimiento de cámaras de video instaladas en los semáforos de la ciudad, ubicación con respecto a la intersección, ángulo de la cámara y altura de la misma; una vez seleccionada la intersección con la cámara que mejor se adaptaba a los requerimientos necesarios para la detección vehicular, se realizó una carta donde se especificó el motivo y video requerido, la cual se introdujo ante el Instituto Autónomo Municipal Policía de San Diego, quien es el encargado del registro de las cámaras de vigilancia publica instaladas en el Municipio de San Diego. De este modo se obtuvo el video de estudio.

3.2.2. Actividad E. Diseñar el software de aforo vehicular

El diseño del sistema de aforo vehicular se llevó acabo en cinco etapas: instalación del lenguaje de programación y librería a utilizarse, desarrollo del módulo de inicilización y módulo de procesamiento, generación de resultados y la interfaz gráfica, siguiendo el siguiente esquema 3.2

3.2.2.1. Soporte para el desarrollo del sistema

Se llevó acabo la instalación de los programas Python 2.7, OpenCV 2.4.13, ffmpeg y PyQt4 mediante instaladores descargados de sus páginas oficiales y de los paquetes: numpy, scipy, dateutil, pytz, pyparsing, cycler, setuptools, matplotlib,

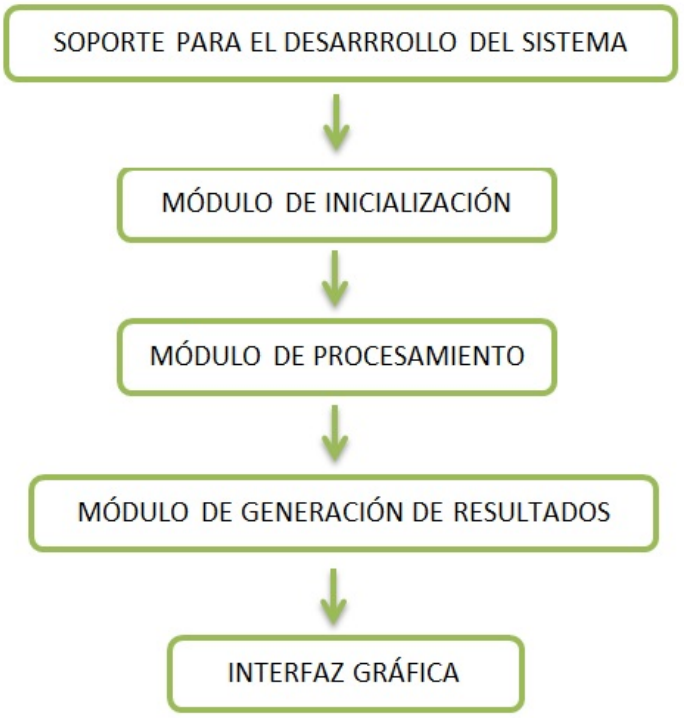


Figura 3.2: Esquema general del diseño el software de aforo vehicular.

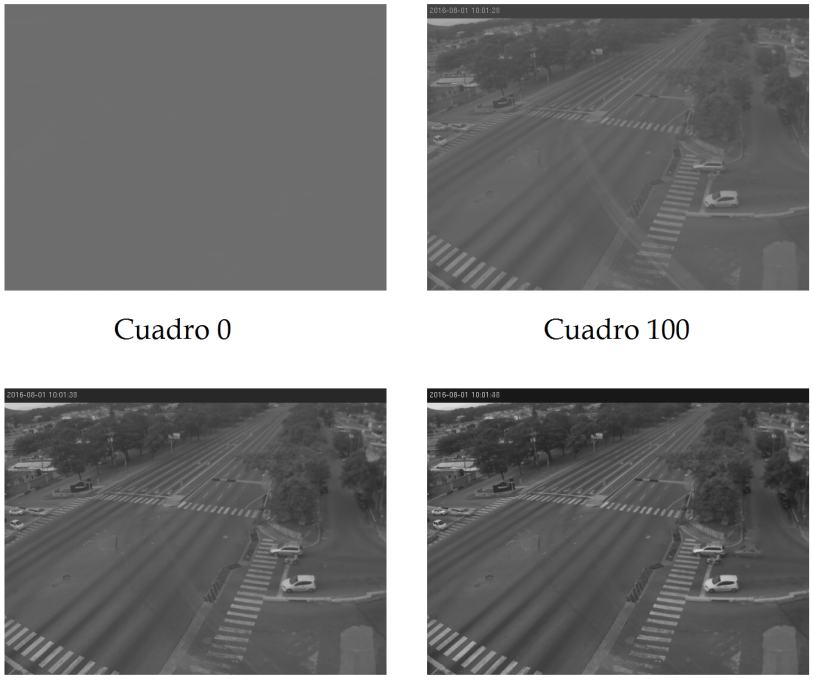
scikit-learn, plotly y xlsxwriter requeridos por OpenCV y necesarios para las operaciones realizadas. La instalación de los paquetes mencionadas se realizó utilizando PIP, la herramienta nativa de gestión de paquetes de Python.

3.2.2.2. Módulo de inicialización

Una vez acondicionado el entorno de programación, se procedió al desarrollo del módulo de inicialización, donde se realizan los procesos previos al módulo de procesamiento. En este módulo se recibe el video de la intersección que será utilizado para aforar y el usuario introduce la duración del video e indica la duración de los intervalos que se tomarán para el histograma.

Seguidamente se procede a inicializar las variables a utilizar posteriormente en el módulo de procesamiento y a realizar el cálculo de imagen de fondo de la intersección, cuyo proceso se basa en promediar los primeros cuadros del video para

obtener una imagen sin vehículos dentro de la intersección, utilizando la función de OpenCV *cv2.accumulateWeighted*, partiendo de una imagen gris con píxeles de valor 110. Tras realizar diferentes pruebas, se obtuvo que al promediar los primeros 300 cuadros del video se obtiene el mejor resultado. El procedimiento se muestra en la Figura 3.3.



Cuadro 0 Cuadro 100

Cuadro 200 Cuadro 300

Figura 3.3: Generación de fondo promedio.

Tras haber calculado la imagen de fondo, se realiza un proceso de binarización para diferenciar el interior de la intersección con el resto de la imagen, generando una imagen negra que identifica gráficamente la zona de estudio pintada de blanco, con el fin de reducir el costo computacional únicamente a la zona de estudio (máscara de intersección). Por ello, el usuario debe delimitar la intersección e identificar los bordes, el procedimiento consiste en recibir dos clicks por cada borde (ya que no siempre las esquinas de la intersección serán visibles en el video); cada pareja de coordenadas permite calcular la recta que describe un borde, las cuales serán interceptadas convenientemente resultando en las esquinas de la intersección.

Los cuatro puntos ubicados en las esquinas de la zona de estudio serán unidos para formar el borde total, así el usuario pasa a identificar cada lado de la intersección como borde de entrada (sentido por donde entran los vehículos a ser aforados), borde de derecha (sentido de salida cuando los vehículos cruzan a la derecha), borde de izquierda y borde de frente.

De la identificación de los bordes, se calculan los umbrales; estas son zonas que permiten identificar cuando un vehículo se encuentra próximo al límite de la zona de estudio, utilizados en el módulo de procesamiento.

3.2.2.3. Módulo de procesamiento

El módulo de procesamiento esta formado por dos sub-módulos: el módulo de detección vehicular y el módulo de rastreo.

1. Módulo de detección vehicular

   En este módulo se procede a realizar la detección vehicular, representado por el flujograma de la Figura 3.4, donde se detectan y ubican los vehículos dentro de la zona de estudio, de modo que recibe un cuadro del video por vez, lo analiza y genera una lista con la ubicación de los vehículos detectados. El proceso se lleva a cabo en las siguientes etapas y puede verse representado en la Figura 3.5:

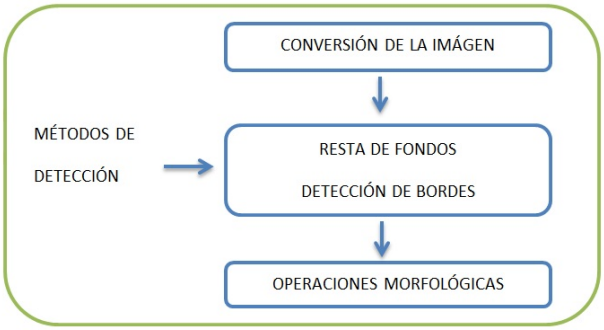


Figura 3.4: Diagrama general del módulo de detección de aforo vehicular mediante procesamiento digital de video

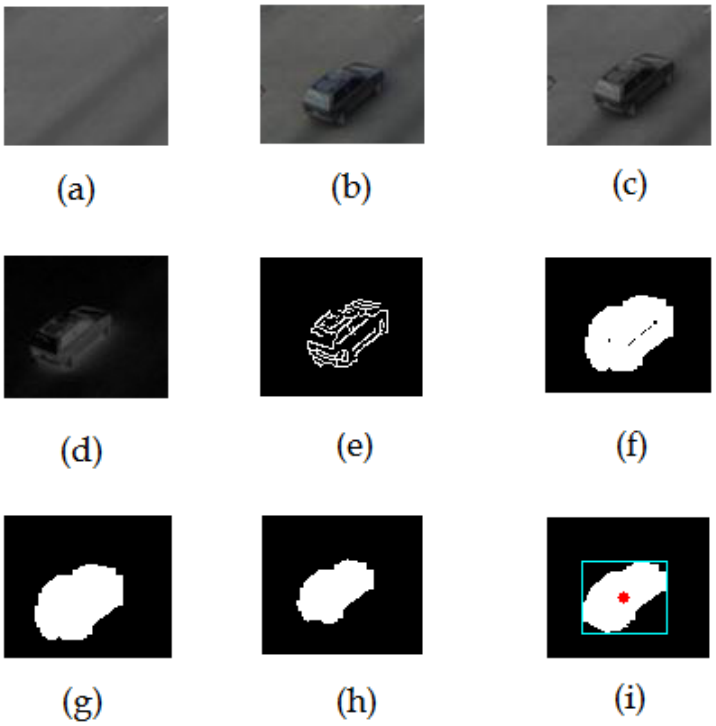


Figura 3.5: Proceso de detección vehicular. (a) Fondo. (b) Imagen original. (c) Imagen convertida al espacio de color gris. (d) Resta entre el fondo y la imagen en escala de gris. (e) Imagen binarizada con el método de detección de bordes Canny. (f) Imagen 'e' dilatada. (g) Imagen 'f' tras realizar el proceso de llenado de bordes. (h) Imagen 'g' erosionada. (i) Detección de un vehículo, el rectángulo que lo enmarca y el centro del rectángulo

- *Conversión de espacio de color:* La primera operación realizada es convertir las imágenes del video del espacio de color BGR al gris, con la finalidad de simplificar los cálculos. Un ejemplo se presenta en la Figura 3.5 (c).
- *Resta de fondo:* Luego se realiza el método de detección de resta de fondo. El método consiste en restar el valor de cada píxel de la imagen en escala gris con la imagen de fondo, obteniendo una imagen en escala de gris mayormente negra a excepción de las zonas donde existe variación. Un ejemplo se presenta en la Figura 3.5 (d).
- *Detección de bordes:* El siguiente paso consiste en binarizar la imagen (asignar valor blanco o negro a cada píxel) mediante el método de detección de bordes Canny. Un ejemplo se presenta en la Figura 3.5 (e).
- *Operaciones morfológicas:* Se busca obtener una imagen donde todas las

zonas de los vehículos detectados sea blanca por ello se realizan las operaciones morfológicas de dilatación, llenado de bordes y erosión. Con la dilatación se asegura la conexión entre los bordes obtenidos de un mismo vehículo, el llenado de bordes pinta de blanco el interior de los contornos obtenidos y elimina los objetos detectados fuera de la zona de estudio (multiplicando con el operador lógico AND la imagen binarizada y la máscara de intersección) y la erosión ayuda en muchos ocasiones a separar las zonas detectadas de diferentes vehículos que están en contacto, pues disminuye el área que ocupa. Un ejemplo se presenta en la Figura 3.5 (f)(g)(h).

- *Descarte de vehículos:* Se descartan los vehículos que están en contacto con el borde de la zona de estudio, siguiendo el siguiente procedimiento: se pintan los bordes de la zona de estudio de blanco, se pinta lo que esta fuera de la zona de estudio de blanco y por último se pintan nuevamente fuera de la zona de estudio de negro. Un ejemplo se presenta en la Figura 3.6.

- *Descarte de objetos estáticos:* Con la finalidad de eliminar objetos estáticos presentes en la zona de estudio (como es el caso de sombras y posibles errores en la detección), se promedia la imagen binaria del paso anterior con las obtenidas previamente, se umbraliza de manera inversa y se multiplica, con el operador lógico AND, con una imagen inicialmente blanca, que acumula todos los píxeles en negro detectados. Dicho en otras palabras, se crea una imagen blanca que toma el color negro en las zonas donde se ha detectado un objeto por varios cuadros seguidos. Para dejar de tomar en cuenta objetos estáticos que ya no están presente en la zona de estudio, se utilizan dos acumuladores que se reinician de forma alterna cada 200 cuadros y se utiliza como máscara el que lleva acumulado más de 100 cuadros.

  La máscara obtenida se multiplica, con el operador lógico AND, con la imagen binaria del paso anterior (imagen negra con zonas en blanco donde se detectaron objetos), de esta forma sólo quedan detectados los objetos en movimiento dentro de la intersección.

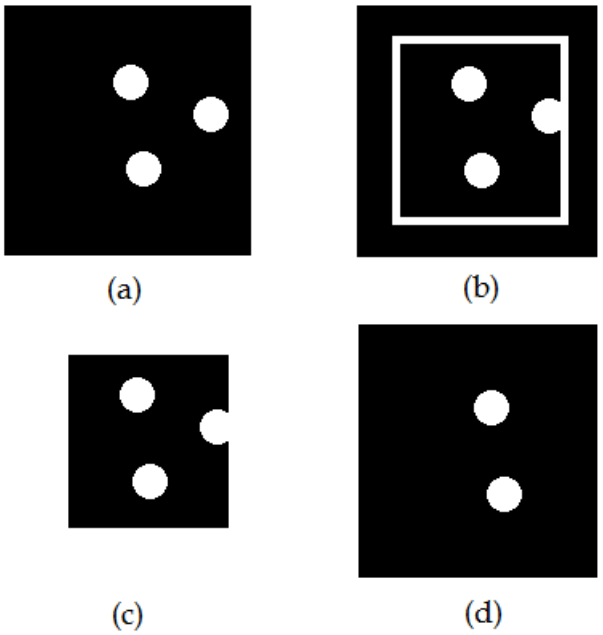


Figura 3.6: Proceso de descarte de vehículos en contacto con el borde. (a) Imagen de la etapa anterior. (b) Imagen 'a' con los bordes pintados en blanco. (c) Imagen 'b' pintada de blanco fuera de la zona de estudio. (d) Imagen 'c' pintada de negro fuera de la zona de estudio.

- *Generación de la lista de vehículos detectados:* Finalmente, a partir de la última imagen resultante se obtiene una lista con los rectángulos más pequeños que enmarcan cada una de las zonas detectadas, de esta manera se conoce la ubicación y dimensión de los objetos, ejemplificado en la Figura 3.5 (i). Este último paso se realiza mediante las funciones de OpenCV *cv2.updateMotionHistory* y *cv2.segmentMotion*.

2. Módulo de rastreo

   En ocasiones suele detectarse dos o más vehículos como uno sólo debido a la perspectiva de la cámara o por encontrarse cerca uno del otro. Cabe destacar que en la mayoría de los casos estos vehículos se separan y juntan a lo largo de su paso por la zona de estudio; la función de este módulo consiste en asociar los vehículos detectados en la imagen actual con los detectados en el cuadro anterior permitiendo actualizar la cantidad de vehículos contenidos en cada bloque (conjunto de vehículos detectados como uno sólo).

El módulo sigue un proceso en cascada donde cada etapa verifica el cumplimiento de condiciones y asigna a un registro los vehículos que las cumplan. Cada etapa está representada por un caso en particular que puede suceder a causa de la recombinación de los vehículos detectados y el orden de verificación de los casos es de acuerdo a la prioridad de detección. A su vez, los vehículos que cumplan las condiciones de una etapa, serán omitidos por las siguientes.

Los registros que se generan son listas, donde en cada posición se tiene el registro de cada bloque en los cuadros donde fue detectado. Dentro de esta segunda lista, cada elemento contiene el cuadro donde fue detectado, el rectángulo que enmarca al bloque, cantidad de vehículos dentro del bloque y una bandera que se utiliza como condición en los casos que lo necesiten.

Siguiendo el proceso de cascada y prioridad de detección, las etapas vienen representadas de la siguiente forma:

- *Identifica adyacentes:* Se encarga de asociar a los bloques detectados en el cuadro actual que tienen una coordenada del cuadro anterior cercana, verificando si la distancia entre los centros es menor a un umbral definido. De ser así, no hay duda que es su posición anterior y se asigna al registro correspondiente.
- *Identifica convergentes:* Se encarga de asociar dos o más bloques del cuadro anterior que ahora son uno sólo. Esto sucede con los bloques del cuadro actual cuyo rectángulo contiene dos o más centros de bloques del cuadro anterior. Un ejemplo se presenta en la Figura 3.7.
- *Identifica divergentes:* Se encarga de asociar bloques del cuadro anterior que ahora son dos o más bloques. Esto sucede con los bloques del cuadro anterior cuyo rectángulo contiene dos o más centros de bloques del cuadro actual. Al cumplir esta condición, la cantidad de vehículos del bloque del cuadro anterior debe repartirse en los nuevos bloques. Un ejemplo se presenta en la Figura 3.8.

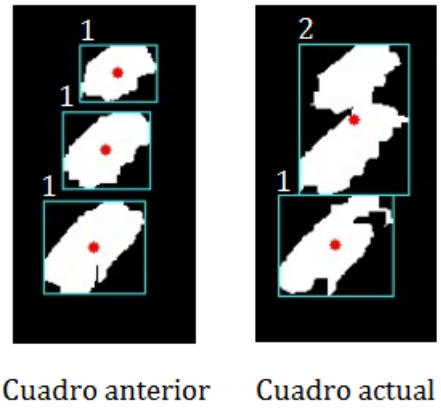


Figura 3.7: Convergencia entre bloques.

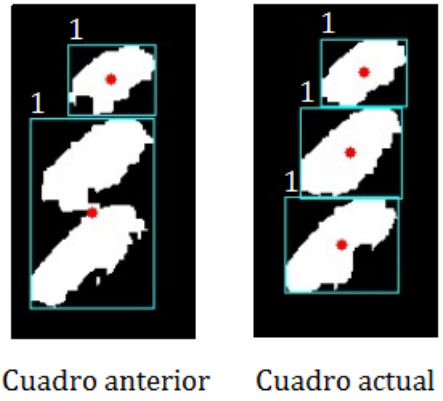


Figura 3.8: Divergencia entre bloques.

- *Identifica combinaciones complejas:* Se encarga de identificar las combinaciones más complejas, donde del cuadro anterior al cuadro actual ocurre convergencia y divergencia entre un mismo grupo de vehículos. Un ejemplo se presenta en la Figura 3.9. Para satisfacer este caso, se buscan bloques del cuadro anterior (c,d) cuyo centro esté dentro de un rectángulo (a) ampliado del cuadro actual (ampliado por un factor igual a 1.2), donde debe encontrarse al menos dos bloques del cuadro anterior que satisfagan la condición y de los cuales se busca si uno de sus rectángulos (d) contiene un centro del cuadro actual (b), diferente al que se mencionó inicialmente (a). En total se identifican al menos dos bloques del cuadro anterior (c,d) y dos del cuadro actual (a,b).

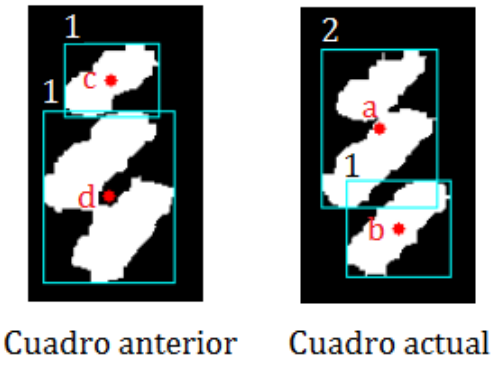


Figura 3.9: Combinación compleja entre bloques.

- *Identifica convergentes cerca de borde:* Se encarga de asociar bloques del cuadro anterior que ahora son uno sólo, realizando el mismo procedimiento de *Identifica convergentes*, con la característica de sólo detectar un bloque del cuadro anterior cuyo centro está dentro de un rectángulo del cuadro actual, siempre que el bloque del cuadro actual esté cerca del borde de entrada. Se dice que un bloque esta cerca de un borde si alguna esquina del rectángulo que lo enmarca está a una distancia menor a la definida por el umbral de dicho borde.

  El motivo de este caso es que los bloques que tocan el borde de la zona de estudio son descartados, por ello, cuando entra un nuevo vehículo e inmediatamente converge con un bloque ya registrado anteriormente, se debe interpretar como una convergencia donde sólo se debe añadir un vehículo al bloque ya registrado. Por otra parte, los bloques que entran en este caso y se encuentran alejados del borde de entrada, se asocian sin añadir un nuevo vehículo al registro.

- *Identifica divergentes cerca de borde:* Se encarga de asociar bloques del cuadro anterior que ahora son varios bloques, realizando el mismo procedimiento de *Identifica divergentes* con la característica de ocurrir cerca de un borde de salida y aceptar como mínimo un sólo bloque del cuadro actual que cumpla la condición. Este caso ocurre cuando un bloque toca un borde de salida e inmediatamente diverge, los cuales deben asociarse al bloque que desapareció y la cantidad de vehículos del bloque

del cuadro anterior debe repartirse en los nuevos bloques restando un vehículo al bloque. Por otra parte, los bloques que entran en este caso y se encuentran alejados de un borde de salida, se asocian sin añadir un nuevo vehículo al registro, esto se interpreta como una divergencia entre un bloque y una moto.

- *Generación de coordenadas aproximadas:* Como se ha mencionado anteriormente, los bloques que están en contacto con el borde son descartados en el módulo de detección para evitar errores. Sin embargo, los vehículos no desaparecen y es por ello que se continua rastreando su ubicación de forma aproximada en el cuadro actual, a partir de las ubicaciones que se han registrado a lo largo de su paso por la zona de estudio. La utilidad de esta etapa es lograr actualizar y corregir la cantidad de vehículos que contiene cada bloque, pues en muchas ocasiones ocurren convergencias y divergencias cerca de los bordes.

  Para que un bloque entre en este caso no se debe tener registro de ubicación en el cuadro actual y se debe verificar que al menos una de las esquinas de su rectángulo esté cerca de algún borde de la intersección.

  De ser así, se toman las últimas diez posiciones (como mínimo se requieren tres) del registro del bloque; donde se calcula la pendiente de la recta que contiene los centros en cuadros adyacentes, además se calcula la variación de posición de dichos centros en x y y. Finalmente se promedian los valores obtenidos y con la pendiente, el centro registrado más recientemente y el mayor valor de variación (x o y) se determina la ubicación aproximada en el cuadro actual. En caso de contar con menos de tres posiciones registradas para el bloque, se aproxima a la ubicación mas cerca fuera de la intersección (a menos que esté cerca del borde de entrada).

- *Identifica adyacentes ampliados:* Se encarga de asociar los bloques que tienen una coordenada anterior relativamente cercana. El procedimiento consiste aplicar nuevamente la etapa de *Identifica adyacentes*, esta vez buscando en una zona mayor a la del rectángulo del cuadro anterior (ampliado por un factor igual a 1.5). Con esto se busca disminuir los

errores ocasionados por la posible perdida de cuadros entre el cuadro actual y el cuadro anterior.

- *Transfiere bloques:* Se encarga de mover los registros de cada bloque a otra lista, siempre que cumplan con las condiciones requeridas. En primer lugar, el bloque comienza siendo registrado en una lista llamada *vchequeo* con la finalidad de evitar tomar en cuenta objetos que no son de interés, donde permanecerá hasta contar con tres ubicaciones que no sean aproximadas, al ocurrir esto, se transfiere a una lista llamada *vactuales*. En *vactuales* se mantiene el registro de cada bloque mientras se encuentre dentro de la zona de estudio; una vez se verifique que el centro del bloque en su última ubicación está fuera de la intersección (utilizando la máscara de intersección), el registro de dicho bloque es transferido a *vlistos*. La utilidad de transferir elementos a otra lista (*vlistos*) es limpiar las listas utilizadas para realizar los cálculos para cada imagen del video. Si se mantuvieran todos los elementos en *vactuales*, cada vez habría que iterar más posiciones para realizar los cálculos de cada cuadro del video.
- *Descarta bloques:* Su función es descartar bloques que no provienen del borde de entrada. Esto se logra verificando que los centros registrados de un bloque estén más alejados del borde de entrada a medida que éstas son más recientes. En *vchequeo* se verifican los últimos 3 registros y los bloques que tengan 2 avances negativos serán eliminados. Se dice que realiza un avance negativo cuando se aproxima al borde de entrada en vez de alejarse. En *vactuales* se verifica la presencia de avance negativo para bloques que tengan entre 3 y 7 ubicaciones registradas; además se toma en cuenta el avance entre la primera y la última ubicación.
- *Elimina bloques:* Se encarga de eliminar los registros de los bloques que no han sido actualizados en los últimos dos cuadros.
- *Agrega bloques:* En esta etapa se agregan los nuevos bloques que no han cumplido las condiciones de las etapas anteriores, siempre y cuando al menos una esquina del rectángulo que enmarca al bloque se encuentre cerca del borde de entrada.

- *Actualiza conteo:* Por último se actualiza la cantidad de vehículos aforados, agregando la cantidad de vehículos contenidos en cada bloque de *vlistos* que fue transferida en el último cuadro del video. Se diferencia entre los vehículos que finalizaron su recorrido por la intersección según el borde que tengan más próximo a su última ubicación registrada.

3.2.2.4. Módulo de generación de resultados

Una vez analizado el video, este módulo se encarga de recibir la información recolectada, procesarla y generar un archivo .xlsx y .html para mostrar los resultados del mismo modo que en el proceso de aforo de forma manual.

Del módulo de procesamiento se recibe la cantidad de cuadros que tiene el video y las listas con los valores acumulados de vehículos contados en cada cuadro, diferenciados por borde de salida. Según sea el valor de duración de intervalos (seleccionado por el usuario) y la cantidad de cuadros por segundo (fps) promedio (determinada dividendo la cantidad total de cuadros entre la duración del video en segundos), se calcula la cantidad de cuadros que corresponde a cada intervalo.

Los valores de aforo correspondiente por intervalo serán la resta entre el valor acumulado del cuadro inicial y el cuadro final del intervalo. Finalmente se gráfica el histograma que representa visualmente los resultados, mediante el módulo plotly, que genera un archivo llamado Histograma_Aforo_Vehicular.html y se muestran los valores tabulados en el archivo llamado Resultado_Aforo.xlsx, utilizando el módulo xlsxwriter.

El histograma permite visualizar rápidamente los resultados y dado que su formato es html, es interactivo al usuario al pasar el cursor sobre la gráfica, proporcionando información referente al resultado por cada intervalo de tiempo de aforo; mientras que con la tabla de datos, se podrá analizar los valores detalladamente.

3.2.2.5. Interfaz gráfica

En esta fase se realizó el diseño y codificación de la interfaz gráfica, bajo el lenguaje de programación Python. Para cada una de las secciones que se planteó fueron generados los elementos respectivos (botones, listas desplegables, etiquetas, radio botones, etc.).

La herramienta seleccionada para crear la interfaz gráfica de usuario en Python fue PyQt (librería de desarrollo de la interfaz).

El primer paso fue crear un objeto del tipo QWidget que cumple la función de ventana principal, sobre este objeto se fueron creando cada una de las etapas de cálculo, por tanto se agregaron los elementos (botones, listas desplegables, etiquetas, radio botones, etc) y se definieron los eventos asociados. Las páginas que conforman la interfaz son:

- *Página inicial:* En la primera página se selecciona el archivo de video que se desea abrir para realizar el aforo vehicular y se muestra de fondo del video.
- *Página de selección de bordes:* En esta segunda página el usuario delimita los bordes de la zona de estudio. En el lado izquierdo se presentan las instrucciones y, en una ventana emergente, la imagen de fondo del video donde se realiza la selección de forma gráfica.
- *Página de identificación de bordes:* En esta página el usuario realiza la identificación de los bordes de la zona de estudio. Las instrucciones se muestran del lado izquierdo y, en una ventana emergente, la imagen de fondo del video donde se realiza la identificación de forma gráfica.
- *Página de configuración:* Su función es configura las características del aforo vehicular, donde se solicita la duración del video y la duración de los intervalos de tiempo del aforo vehicular.
- *Página de aforo y resultados:* Finalmente se ejecuta la etapa de procesamiento y generación resultado, mientras se muestra un mensaje que indica que se esta realizando el proceso de aforo y se visualiza el video.

Una vez culminado el aforo vehicular se generan dos archivos con los resultados obtenidos, un archivo Excel: Resultado_Aforo.xlsx y un histograma: Histograma_Aforo_Vehicular.html.

3.2.3. Actividad F. Evaluar el desempeño del sistema de aforo vehicular diseñado con base en el tiempo de procesamiento y exactitud de conteo según la clasificación establecida

Con la finalidad de evaluar el sistema de aforo vehicular, se tomó en cuenta el desempeño con base en el tiempo de procesamiento y con base a la exactitud de aforo.

Para estudiar el tiempo de procesamiento, se tomó el instante antes y después de entrar al módulo de procesamiento (pues el resto del algoritmo sólo se ejecuta una vez) y se comparó con la duración del video.

Para el estudio de exactitud, se realizó el aforo vehicular de forma manual y utilizando el sistema desarrollado; se compararon los resultados correspondientes para cada sentido de giro, tanto los totales contados durante todo el video como los valores correspondientes de cada intervalo. Dichos intervalos corresponden al formato que se utiliza para resumir los resultados obtenidos, agrupando la cantidad de vehículos que pasa por la zona de estudio según la hora, con la intención de generar un histograma.

Capítulo IV

Análisis, interpretación y presentación de los resultados

4.1. Aplicación del sistema de aforo vehicular a video de transito

Se procedió a realizar pruebas con el video pre-grabado de extensión .asf y duración 35 minutos con 52 segundos, proporcionado por el Instituto Autónomo Municipal Policía de San Diego del Municipio de San Diego, basado en dos criterios: el tiempo de procesamiento y la exactitud de conteo según la clasificación establecida. En el video se observa la zona de estudio con punto ciego en las dos esquinas del borde de entrada.

El proceso de transmisión desde la cámara hasta el instituto policial antes mencionado comienza capturando la escena con una cámara IP instalada en la torre de iluminación ubicada en la esquina sur oeste de la intersección, el video es transmitido mediante enlace de microondas a una estación repetidora que recibe videos de otras cámaras y luego se transmite de la estación repetidora al instituto policial, donde se es almacenado para su posterior uso (un aproximado de una semana).

4.1.1. Análisis del tiempo de procesamiento

La evaluación del tiempo de procesamiento consistió en comparar la duración del video pre-grabado con el tiempo de procesamiento del mismo en el sistema de aforo vehicular desarrollado.

Para obtener el tiempo de procesamiento se procedió a ejecutar el sistema de aforo vehicular desarrollado y medir el tiempo utilizando la librería de Python datetime, tomando el instante antes y después de entrar al módulo de procesamiento, ya que los procesos previos a este módulo se realizan una sola vez por video y el tiempo estaría influenciado por la duración del usuario al cargar la información requerida.

Las pruebas realizadas fueron ejecutando el sistema de aforo vehicular en dos computadores con características mostradas a continuación:

- Computador 1: Intel Core i3-3110M CPU @ 2.40 GHz, Memoria RAM 4 GB, sistema operativo de 64 bits.
- Computador 2: Intel Core i5-2410M CPU @ 2.30 GHz, Memoria RAM 6 GB, sistema operativo de 64 bits.

A continuación se muestra en la Tabla 4.1 el tiempo de procesamiento obtenido al procesar el video en el sistema de aforo vehicular:

	Computador 1	Computador 2
Tiempo de procesamiento [minutos]	18,28	16,96

Tabla 4.1: Resultados obtenidos en base a los tiempo de procesamiento.

Se observa que en los dos casos, el tiempo de procesamiento fue menor al tiempo de duración del video (35,87 minutos), tardando entre un 50,9 % y 47,3 % de la duración total del video, respectivamente. Es decir, el sistema es capaz de procesar el video y realizar el aforo vehicular en un tiempo de aproximadamente la mitad del tiempo real. Ahora, comparando los resultados por los diferentes computadores, se observa una variación de 3,6 % debido a sus características, cabe destacar

que la diferencia de costo entre ambos procesadores es de aproximadamente $20. Este costo adicional no es necesario pues la diferencia de desempeño es despreciable; suponiendo un video de 8 horas, al computador 1 le tomaría solo 17,28 minutos adicionales para realizar el análisis del video.

Para evaluar el proceso de aforo manual realizado con varias personas toma como mínimo la duración del video; analizando un caso basado en datos proporcionados por el departamento de vialidad, tomando la cantidad mínima de 6 personas (2 personas por sentido) para realizar el conteo visual y alternarse la tarea entre ellos (3 aforadores por vez) para poder ser precisos, ya que al mantenerse mucho tiempo se presenta cansancio y aumenta el grado de error; luego se procede a totalizar los resultados y obtener la tabla resumen, como la que genera el sistema desarrollado, tomando 5 minutos. Totalizando, este caso dura 40 minutos con 52 segundos, representando un 113,94 % de la duración total del video.

En el caso del proceso de aforo visual por una persona, observando el video pregrabado, se analiza basado en la experiencia de conteo por parte de los autores. El proceso de aforo tomó cerca de 2 horas, debido a es que es necesario pausar el video constantemente para registrar la cuenta de vehículos por cada sentido. Siendo así, el proceso de aforo manual representa un 334,54 % de la duración total del video.

En la Tabla 4.2 se muestran los tres tipos de aforo tomados como prueba y sus respectivos tiempos de aforo.

Tipo de aforo utilizado	Tiempo de aforo [min]	Tiempo respecto al video [%]
Sistema de aforo vehicular	16,96	47,3
Aforo manual con 6 personas	40,52	113,94
Aforo manual con una persona	120,0	334,54

Tabla 4.2: Resultados obtenidos de los diferentes tipos de aforo.

Observando la Tabla 4.2 es evidente que el sistema de aforo vehicular desarrollado minimiza a un 47,3 % el tiempo de procesamiento. Teniendo el aforo visual por 6 personas un tiempo de aproximadamente más del doble de la duración del usado por el sistema, representado por un 113,94 % de duración respecto al video

y en cuanto al aforo visual realizado por una persona representaría 334,54 % de la duración del video.

4.1.2. Análisis de la exactitud de conteo

La evaluación de exactitud de conteo vehicular consistió en la ejecución del sistema desarrollado con el video de estudio, la realización del conteo vehicular de forma manual con el mismo video y la comparación de los resultados obtenidos en los dos conteos. Los resultados obtenidos se muestran en las Tablas 4.3, 4.4 y 4.5 respectivamente.

Intervalo	Izquierda	Frente	Derecha
00:00 - 04:59	4	127	0
05:00 - 09:59	10	144	1
10:00 - 14:59	10	103	2
15:00 - 19:59	2	137	5
20:00 - 24:59	3	146	3
25:00 - 29:59	5	128	3
30:00 - 34:59	5	120	2
35:00 - 35:52	1	38	0
Total	40	943	16

Tabla 4.3: Resultados obtenidos en la ejecución del sistema de aforo vehicular utilizando el video en estudio.

Intervalo	Izquierda	Frente	Derecha
00:00 - 04:59	4	182	0
05:00 - 09:59	10	147	1
10:00 - 14:59	13	187	2
15:00 - 19:59	1	125	2
20:00 - 24:59	4	238	5
25:00 - 29:59	6	234	5
30:00 - 34:59	6	175	2
35:00 - 35:52	0	0	0
Total	44	1288	17

Tabla 4.4: Resultados obtenidos en la ejecución del conteo vehicular de forma manual.

De la Tabla 4.5 se observa que el error absoluto total para los vehículos que cruzan a la derecha es de -1 (equivalente a un 5,88 % de error porcentual), similar al

Intervalo	Izquierda	Frente	Derecha
00:00 - 04:59	0	-55	0
05:00 - 09:59	0	-3	0
10:00 - 14:59	-3	-84	0
15:00 - 19:59	1	12	3
20:00 - 24:59	-1	-92	-2
25:00 - 29:59	-1	-106	-2
30:00 - 34:59	-1	-55	0
35:00 - 35:52	1	38	0
Total	-4	-345	-1

Tabla 4.5: Comparación de los resultados obtenidos de aforo vehicular expresado en error absoluto.

de los vehículos que cruzan a la izquierda, con -4 (equivalente a un 9,09 % de error porcentual). Por otra parte, se observa que el error absoluto para los vehículos que cruzan hacia el frente varía mucho mas, desde -106 hasta 12, con un error absoluto total de -345 (equivalente a un 26,79 % de error porcentual).

Los resultados de la Tabla 4.5 muestran errores absolutos elevados en diferentes periodos. Motivado a que durante la ejecución del video se observó pérdida de cuadros a causa de la transmisión inalámbrica del video, se procedió a evaluar sí la pérdida de cuadros influye en el resultado del sistema de aforo. Para ello se determinó como escenario de prueba: aforar en todos los periodos donde transita una gran cantidad de vehículos que cruzan hacia el frente, lo cual ocurre desde que la luz del semáforo cambia a verde hasta que dejan de circular a causa del cambio a luz roja.

Habiendo definido los intervalos como muestra a analizar, se procedió a efectuar el aforo vehicular en los mismos. Los resultados obtenidos se presentan en la Tabla 4.6, donde se pueden observar los intervalos, la cantidad de vehículos que realmente cruzó, la cantidad de vehículos contada por el sistema y el error porcentual correspondiente.

Se observa que los menores valores de error porcentual se encuentran en el intervalo 1, con 4,29 %, seguido por el intervalo 13 con 16,07 %, de igual forma, los valores con mayor error porcentual se observaron en el intervalo 7, con 55,88 %,

	Intervalo [cuadro]	Cant. Real	Cant. Aforada	Error [%]
1	380 - 820	70	67	4,29
2	1600 - 1920	66	53	19,70
3	2530 - 2990	59	49	16,95
4	3640 - 3925	67	50	25,37
5	4430 - 4820	67	42	37,31
6	5200 - 5670	58	39	32,76
7	6560 - 6760	68	30	55,88
8	7420 - 7985	64	43	32,81
9	8844 - 9340	57	43	24,56
10	9715 - 9967	69	49	28,98
11	10340 - 10730	68	51	25,00
12	11290 - 11754	84	44	47,62
13	12390 - 13152	112	94	16,07
14	13880 - 14595	107	82	23,36
15	15580 - 16333	106	85	19,81
16	17290 - 17945	96	80	16,67
17	18700 - 19024	86	38	54,65

Tabla 4.6: Resultados de exactitud obtenidos para los vehículos que cruzan hacia el frente en los intervalos donde tienen permitido el paso los vehículos.

seguido por el intervalo 17 con 54,65 %.

Conocidos los intervalos y su respectivo error porcentual, se procede a realizar un estudio minucioso de los cuadros recibidos por segundo de los 4 intervalos nombrados anteriormente, para analizar lo sucedido en dichos intervalos de tiempo.

En primer lugar se le realizó un conteo de cuadros recibidos por segundo de forma manual a cada intervalo. Estos resultados se muestran en las Tablas 4.7, 4.8, 4.9 y 4.10.

Segundo	1	2	3	4	5	6	7	8	9	10
Cuadros	10	8	12	13	13	12	8	12	10	9
Segundo	11	12	13	14	15	16	17	18	19	20
Cuadros	12	8	10	11	8	5	6	11	10	7
Segundo	21	22	23	24	25	26	27	28	29	30
Cuadros	9	9	11	8	11	10	10	10	13	13
Segundo	31	32	33	34	35	36	37	38	39	40
Cuadros	14	11	11	10	11	14	12	10	14	15

Tabla 4.7: Resultados de cuadros recibidos por segundo del intervalo 1.

Segundo	1	2	3	4	5	6	7	8	9	10
Cuadros	11	5	7	6	4	6	8	4	4	3
Segundo	11	12	13	14	15	16	17	18	19	20
Cuadros	4	4	1	1	1	3	1	1	2	2
Segundo	21	22	23	24	25	26	27	28	29	30
Cuadros	2	5	5	7	4	5	5	2	3	2
Segundo	31	32	33	34	35	36	37	38	39	40
Cuadros	4	7	3	4	6	7	7	7	5	4
Segundo	41	42	43	44						
Cuadros	8	9	3	3						

Tabla 4.8: Resultados de cuadros recibidos por segundo del intervalo 7.

Segundo	1	2	3	4	5	6	7	8	9	10
Cuadros	15	14	9	14	13	13	15	12	14	16
Segundo	11	12	13	14	15	16	17	18	19	20
Cuadros	10	11	7	5	4	4	7	9	8	8
Segundo	21	22	23	24	25	26	27	28	29	30
Cuadros	9	10	13	9	12	11	11	13	15	12
Segundo	31	32	33	34	35	36	37	38	39	40
Cuadros	14	15	13	9	7	11	7	8	5	4
Segundo	41	42	43	44	45	46	47	48	49	50
Cuadros	3	7	5	7	7	7	9	8	9	11
Segundo	51	52	53	54	55	56	57	58	59	60
Cuadros	11	8	14	14	12	14	14	11	12	15
Segundo	61	62	63	64	65	66	67	68	69	70
Cuadros	14	13	11	10	14	12	9	13	12	12
Segundo	71									
Cuadros	13									

Tabla 4.9: Resultados de cuadros recibidos por segundo del intervalo 13.

Segundo	1	2	3	4	5	6	7	8	9	10
Cuadros	2	2	2	2	2	1	1	3	3	5
Segundo	11	12	13	14	15	16	17	18	19	20
Cuadros	9	9	4	8	13	10	14	13	13	12
Segundo	21	22	23	24	25	26	27	28	29	30
Cuadros	15	12	13	7	11	13	13	12	15	12
Segundo	31	32	33	34	35	36	37	38	39	40
Cuadros	12	9	2	2	0	1	1	0	1	1
Segundo	41	42	43	44	45	46	47	48	49	50
Cuadros	0	1	1	1	1	3	0	1	3	3
Segundo	51	52	53	54	55	56	57	58	59	60
Cuadros	1	2	4	4	4	1	1	0	1	1
Segundo	61	62	63	64						
Cuadros	2	1	1	0						

Tabla 4.10: Resultados de cuadros recibidos por segundo del intervalo 17.

Luego, se totalizó la cantidad de segundos que conforman al intervalo, se determinó la cantidad máxima y mínima de cuadros recibidos por segundo y se calculó el promedio de cuadros recibidos por segundo. Debido a que en el intervalo con menor error porcentual (intervalo 1) la cantidad mínima de cuadros recibidos por segundo fue 5, se seleccionó este valor como referencia para expresar la pérdida de cuadros y comparar dicho valor entre los 4 intervalos, por lo que se determinó la cantidad de segundos con 5 cuadros recibidos o menos y el porcentaje que esta representa.

Los resultados se presentan en las Tablas 4.11, 4.12 4.13 y 4.14 respectivamente.

Cantidad de segundos:	40
Máximo valor de cuadros recibidos por segundo:	15
Mínimo valor de cuadros recibidos por segundo:	5
Promedio de cuadros recibidos por segundo:	10,52
Cantidad de segundos con 5 cuadros o menos:	1
Porcentaje de segundos con 5 cuadros o menos:	2,5

Tabla 4.11: Resultados de análisis del intervalo 1.

De los resultados obtenidos en el conteo de cuadros recibidos por segundo de cada intervalo podemos observar lo siguiente:

Cantidad de segundos:	44
Máximo valor de cuadros recibidos por segundo:	11
Mínimo valor de cuadros recibidos por segundo:	1
Promedio de cuadros recibidos por segundo:	4,48
Cantidad de segundos con 5 cuadros o menos:	31
Porcentaje de segundos con 5 cuadros o menos:	70,45

Tabla 4.12: Resultados de análisis del intervalo 7.

Cantidad de segundos:	71
Máximo valor de cuadros recibidos por segundo:	16
Mínimo valor de cuadros recibidos por segundo:	3
Promedio de cuadros recibidos por segundo:	10,52
Cantidad de segundos con 5 cuadros o menos:	6
Porcentaje de segundos con 5 cuadros o menos:	8,45

Tabla 4.13: Resultados de análisis del intervalo 13.

Cantidad de segundos:	64
Máximo valor de cuadros recibidos por segundo:	15
Mínimo valor de cuadros recibidos por segundo:	0
Promedio de cuadros recibidos por segundo:	4,95
Cantidad de segundos con 5 cuadros o menos:	43
Porcentaje de segundos con 5 cuadros o menos:	67,19

Tabla 4.14: Resultados de análisis del intervalo 17.

- Intervalo 1: El promedio de cuadros recibidos por segundo es de 10,52, con valores entre 5 y 15 cuadros, tiene la presencia de un solo segundo con un valor igual a 5 cuadros, representando el 2,5 % de los segundos del intervalo.
- Intervalo 13: Este intervalo tiene en promedio 10,52 cuadros recibidos por segundo, con valores entre 3 y 16 cuadros, tiene la presencia de 6 segundos con valores inferiores o iguales a 5 cuadros recibidos por segundo, lo que representa el 8,45 % de los segundos del intervalo.
- Intervalo 17: En este intervalo, el promedio de cuadros recibidos por segundo es de 4,95, con valores entre 0 y 15 cuadros, presentando una alta pérdida de cuadros; tiene la presencia de 43 segundos con valores de cantidad de cuadros recibidos inferiores o iguales a 5, lo que representa el 67,19 % de los segundos del intervalo.

- Intervalo 7: En este último intervalo, el promedio de cuadros recibidos por segundo es de 4,48, con valores entre 1 y 11 cuadros, presentando una alta pérdida de cuadros; tiene la presencia de 31 segundos con valores de cantidad de cuadros recibidos inferiores o iguales a 5, representando el 70,45 % de los segundos del intervalo.

Observando el promedio de cuadros recibidos por segundo de los intervalos 1 y 13, se aprecia que son iguales, sin embargo el error porcentual obtenido durante el proceso de aforo es de 4,29 % para el intervalo 1 y 16,07 % para el 13; lo cual significa que el promedio de cuadros recibidos por segundo no está relacionado con la exactitud del sistema.

Por el contrario, se puede observar que los intervalos con menor error porcentual presentan un menor porcentaje de segundos con 5 cuadros recibidos o menos. Basado en ello, se puede afirmar que la perdida de cuadros influye directamente en el error ocasionado en el proceso de aforo, pues contar con menos cuadros del video significa contar con menos información al momento de realizar el proceso de aforo vehicular, específicamente en el módulo de rastreo. Esta relación se puede observar gráficamente en la Figura 4.1.

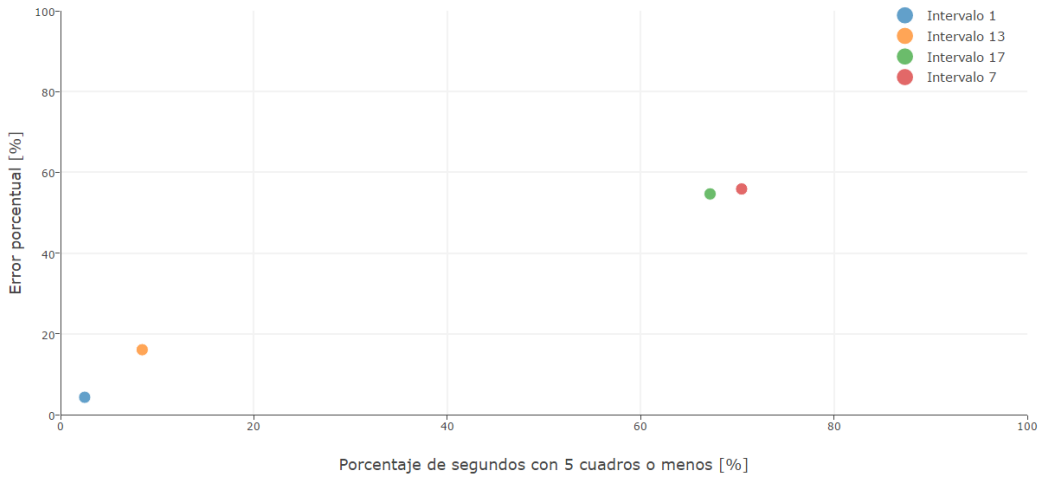


Figura 4.1: Gráfica de error porcentual respecto al porcentaje de segundos con 5 cuadros recibidos o menos de los 4 intervalos analizados.

Cabe destacar que el valor máximo de cuadros recibidos por segundo no condiciona la exactitud, puesto que la información perdida corresponde a un instante de tiempo dado y ésta no puede ser recuperada en otro instante. Esto se aprecia en las Tablas 4.6, 4.7 y 4.10, donde se muestra que el intervalo 1 que posee un error porcentual de 4,29 % y solo 1 segundo con 15 cuadros, y el intervalo 17 presenta un error de 54,65 % y 2 segundos con 15 cuadros; es decir, el intervalo 17 tiene el doble de segundos con 15 cuadros recibidos respecto al intervalo 1 y por el contrario, su error es mucho mas elevado (mas de 50 %).

En resumen, la perdida de cuadros o los segundos con pocos cuadros recibidos afectan drásticamente la exactitud de los resultados obtenidos por el sistema desarrollado, por lo que el video no debe tener mas del 2,5 % de sus segundos con 5 cuadros recibidos o menos para poder lograr una exactitud de 95 % y poder cumplir con la exigencia de exactitud del Departamento de Vialidad de la Escuela de Ingeniería Civil.

Capítulo V

Conclusiones y recomendaciones

5.1. Conclusiones

El aporte principal de esta investigación fue el desarrollo de un sistema de aforo vehicular mediante procesamiento digital de video, bajo el lenguaje Python, a través de la librería OpenCV y PyQt; el mismo permite realizar el conteo y rastreo de vehículos en una intersección mediante videos pre-grabados, diseñado para disminuir el tiempo de aforo manual realizado por los estudiantes del Departamento de Vialidad, en la Escuela de Ingeniería Civil, en la Facultad de Ingeniería, de la Universidad de Carabobo.

Con base en la experiencia obtenida en el desarrollo de cada una de las etapas de la investigación se concluye que:

1. El sistema desarrollado es capaz de procesar el video en tiempo real, invirtiendo solo la mitad de la duración total del mismo, dado que el tiempo de procesamiento fue de 16,96 minutos, con respecto a los 35,87 minutos de duración del video.
2. El método de aforo manual que se utiliza en la actualidad requiere la asistencia de al menos 6 personas y más del doble del tiempo que toma el sistema desarrollado. Gracias a esta notoria mejora, los estudiantes de Ingeniería

Civil podrán profundizar más sobre el tema de aforo, además de obtener experiencia sobre el manejo de un sistema de aforo automatizado; del mismo modo, en el área laboral de los ingenieros civiles se podrá reducir el personal y tiempo empleados en realizar el estudio de aforo vehicular, obteniendo una reducción en los costos de trabajo.

3. Es posible desarrollar un sistema de aforo vehicular mediante procesamiento digital de video que permita arrojar valores de exactitud mayores a 95 %. La cantidad de cuadros por segundo del video afecta directamente a la exactitud del sistema de aforo vehicular desarrollado; se obtuvo como resultado que es necesario un video con valores mayores a 5 cuadros por segundo para asegurar una exactitud mayor a 95 %.

5.2. Recomendaciones

Con base al estudio realizado y a los resultados obtenidos, se recomienda:

1. Utilizar un video que tenga un valores de cuadros por segundo mayores a 5, de esta forma se obtendrán resultados de aforo con menor error.
2. Implementar el sistema de aforo vehicular desarrollado en modelos prácticos, donde pueda ser utilizado como una herramienta piloto en diferentes tipos de estudio, como: estudios de vías de escape en caso de emergencia, estudios de seguridad vial, impacto medio-ambiental, etc.

Así mismo, como mejoras a la presente investigación, se recomienda:

1. Modificar el sistema desarrollado para ser utilizado en videos en vivo.
2. Buscar otra forma de delimitar los intervalos de aforo y así distribuir en el tiempo correctamente los vehículos aforados, de manera que no se requiera de un video con valores de cuadros por segundo constantes.

3. Añadir al sistema un módulo de reconocimiento que sea capaz de monitorear, detectar y clasificar los resultados del módulo de detección con la finalidad de determinar si un vehículo corresponde a la clasificación de liviano, bus, pesado, entre otros.

4. Adaptar el sistema para que se desempeñe en horario nocturno.

5. Modificar el sistema desarrollado para disminuir el tiempo de aforo, implementando procesamiento en paralelo. De esta forma se podría analizar videos de larga duración en fracciones de su tiempo.

Apéndice A

Sistema aforo vehicular mediante procesamiento digital de video

A continuación se muestra detalladamente la interfaz del sistema de aforo vehicular desarrollado.

1.1. Página inicial

La aplicación inicia con una pantalla de 1000x600 píxeles, en la cual se encuentra un mensaje para seleccionar el video que se desea aforar seguido de un botón habilitado para abrir el archivo de video extensión .asf, icono de cerrar y minimizar la ventana. La página inicial se muestra en la Figura 1.1.

Al presionar el botón Abrir Archivo, se abre una ventana de dialogo que permite ubicar el archivo en memoria e importarlo a la interfaz, una vez seleccionado el video con extensión .asf se muestra en la interfaz el nombre del video y extensión, aparece el botón de Cargar Video al lado del botón Abrir Archivo y la imágen de fondo en una ventana emergente (imágen sin vehículos dentro de la zona de estudio) a modo de vista previa del video, como se muestra en la Figura 1.2

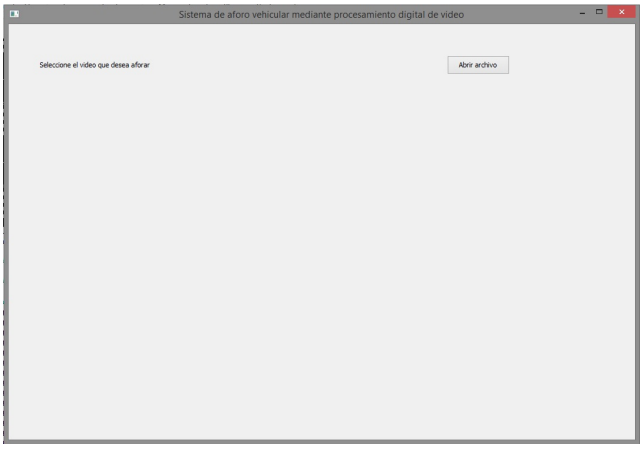


Figura 1.1: Página Inicial de la interfaz gráfica.

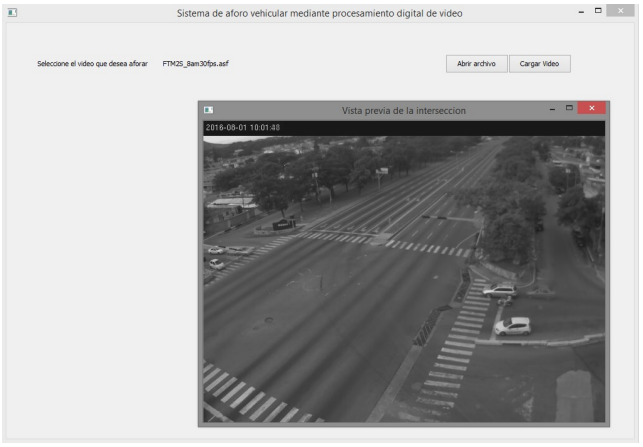


Figura 1.2: Página Inicial al ser valida la extensión del video.

Si el archivo no es de extensión .asf, no se importa el video a la interfaz, aparece un mensaje indicando ”EL ARCHIVO SELECCIONADO ES INVALIDO” y no aparecerá el botón de Cargar Video, como se muestra en la Figura 1.3

Una vez importado el video con extensión .asf, se presiona el botón Cargar Video el cual esta habilitado para cargar el video a la interfaz y proyectar la siguiente página.

Si se presiona el ícono Cerrar, se cierra la interfaz y se minimiza si se presiona el ícono de Minimizar.

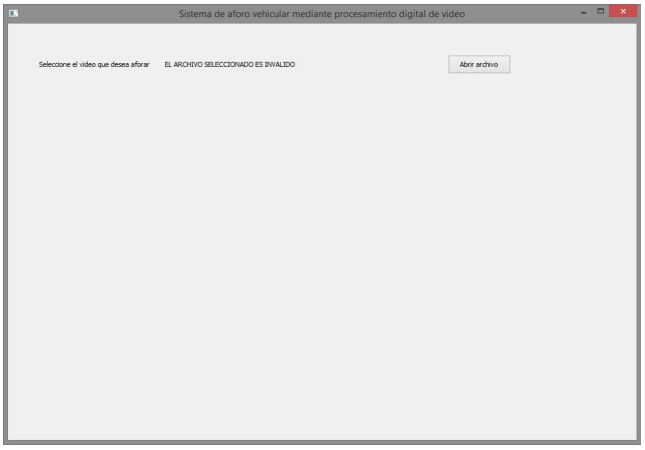


Figura 1.3: Página Inicial al ser invalida la extensión del video.

1.2. Página de selección de bordes

Al cargar el archivo de video, se presenta la página de Selección de Bordes, la cual tiene el aspecto mostrado en la Figura 1.4, donde se observa: las instrucciones para realizar la selección de bordes y la ventana emergente mostrando imágen de fondo donde se realiza la selección de bordes.

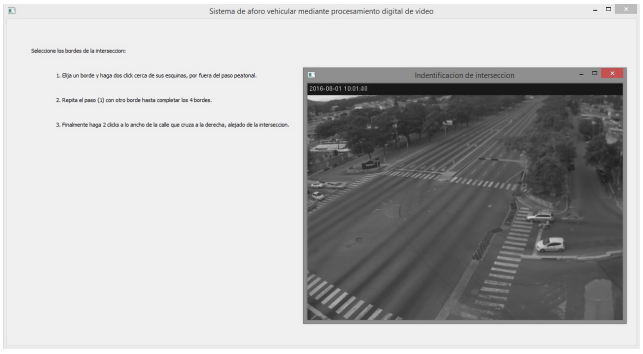


Figura 1.4: Página de selección de bordes de la zona de estudio.

Las instrucciones son las siguientes:

Seleccione los bordes de la intersección:

1. Elija un borde y haga dos clicks cerca de sus esquinas, por fuera del paso peatonal.

2. Repita el paso (1) con otro borde hasta completar los 4 bordes.

3. Finalmente haga 2 clicks a lo ancho de la calle que cruza a la derecha, alejado de la intersección.

El programa toma los primeros 10 clicks que se realicen sobre la imágen mostrada, esta información es guardada en una lista hasta pasar a la siguiente ventana. Luego de haber realizado este paso, se muestran dos botones: el botón Cargar Bordes, que permite pasar a la siguiente página y el botón Volver a Seleccionar Bordes, que permite realizar nuevamente la selección de bordes. Un ejemplo se presenta en la Figura 1.5.

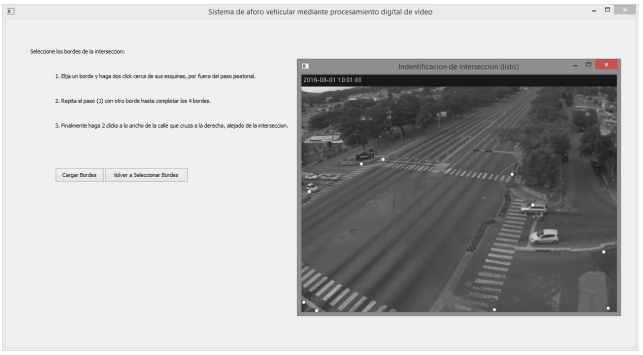


Figura 1.5: Página de selección de bordes luego de haber sido seleccionados.

1.3. Página de identificación de bordes

Esta página permite realizar la identificación de los bordes de la zona de estudio seleccionados previamente. La página muestra las instrucciones y una ventana emergente muestra la imágen de fondo con los bordes marcados con una linea blanca. Un ejemplo se presenta en la Figura 1.6.

Las instrucciones son las siguientes:

Identifique cada borde que delimita la intersección, en el siguiente orden:

1. Borde de entrada (por donde entran los vehículos de interes).

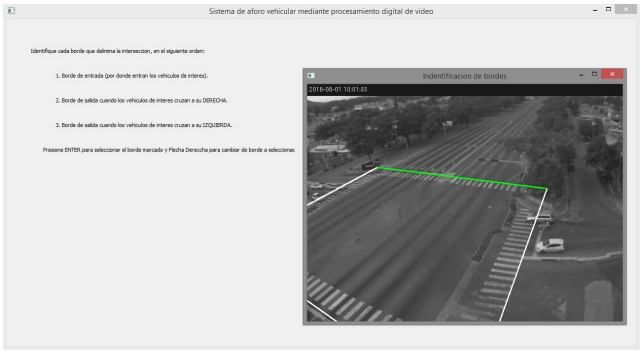


Figura 1.6: Página de identificación de bordes de intersección.

2. Borde de salida cuando los vehículos de interes cruzan a su DERECHA.

3. Borde de salida cuando los vehículos de interes cruzan a su IZQUIERDA.

Presione ENTER para seleccionar el borde marcado y Flecha Derecha para cambiar de borde a seleccionar.

Al iniciar esta página se continua con la ejecución del módulo de inicialización, calculando el resto de valores iniciales y permitiendo al usuario identificar de manera gráfica cada borde. Luego de culminar la identificación de bordes, se muestran los botones: Determinar, que permite pasar a la siguiente página y el botón Volver a Cargar Bordes, que permite realizar el proceso de identificación de bordes nuevamente. Un ejemplo se presenta en la Figura 1.7.

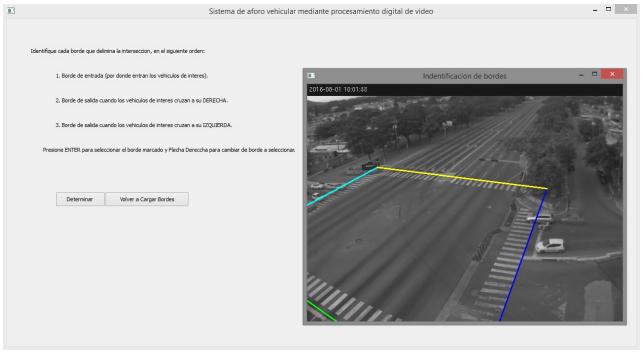


Figura 1.7: Página de identificación de bordes una vez identificados los bordes de la intersección.

1.4. Página de configuración del aforo vehicular

Esta página permite configurar las características del aforo vehicular a realizar, mostrada en la Figura 1.8, recibiendo los siguientes parámetros en el respectivo orden:

- *Duración del video:* Solicita que se indique la duración del video, esta se introduce por una línea de edición separada para horas, minutos y segundos.
- *Duración de los intervalos de tiempo:* Solicita la duración de los intervalos de tiempo del Aforo Vehicular, introducido mediante radio botones predeterminado para 5, 10 y 15 minutos.

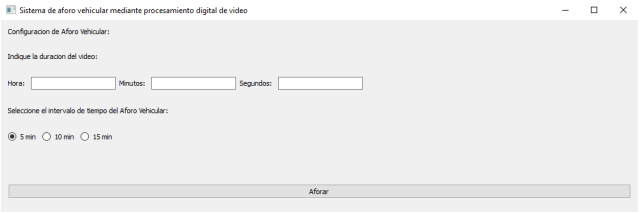


Figura 1.8: Página de configuración de la interfaz gráfica.

Luego de haber introducido los datos, se debe presionar el botón Aforar, que se muestra abajo de los parámetros a configurar y permite pasar a la siguiente página.

1.5. Página de aforo

En esta página se da inicio al aforo vehicular, notificando al usuario con el texto Realizando aforo vehicular ..."dentro de la ventana, presentando el botón Detener y mostrando el video. Un ejemplo se presenta en la Figura 1.9.

Al presionar el botón Aforar, además de cambiar de página, comienza el proceso de aforo vehicular, donde el módulo de procesamiento recibe cada uno de los cuadros del video.

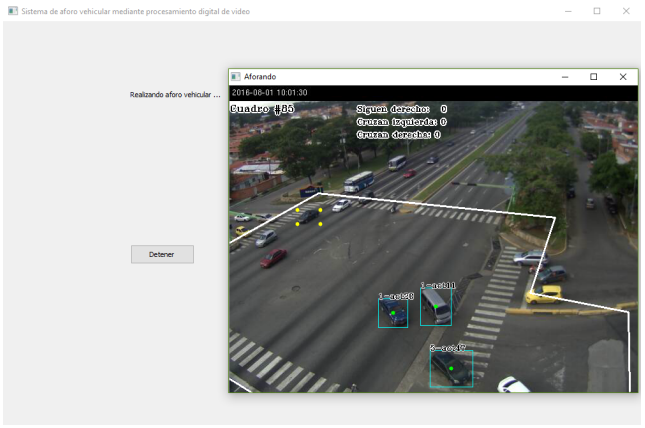


Figura 1.9: Página de aforo de la interfaz gráfica.

Siendo el módulo de procesamiento el núcleo del sistema, fue desarrollado para detectar objetos de forma robusta y descartar aquellos que no se traten de vehículos, discriminando el tamaño de los mismos; así como omitir objetos en contacto con el borde de la zona de estudio y aquellos que se presenten estáticos, para reducir errores.

Del mismo modo, fue necesario estudiar los diferentes casos que se pueden presentar por la recombinación entre vehículos, pues a pesar de utilizar un video tomado desde una altura considerable, no fue posible eliminar completamente el efecto de la oclusión entre los vehículos; sin embargo, si se logró reducir significativamente su consecuencia sobre la exactitud de conteo. Luego de comprender las diferentes maneras en que se pueden recombinar los vehículos (como lo son la convergencia y divergencia entre vehículos), se codificaron los algoritmos que describen cada uno de los casos (con sus respectivas condiciones) para que el módulo sea capaz de interpretar que sucede con los bloques, actualizar la cantidad de vehículos que contiene durante su paso por la zona de estudio y así sumar al conteo con la cantidad correcta, en la gran mayoría de los casos.

La ventaja de haber realizado el algoritmo de rastreo se ve reflejada en los resultados obtenidos, ya que se adapta el sistema de aforo a la necesidad y perspectiva que será común en la mayoría de los caso donde será implementado.

Finalmente, culminado el análisis del video, se ejecuta automáticamente el mó-

dulo de generación de resultados y se cierra la ventana emergente donde se visualiza el video.

Si el usuario presiona el botón Detener durante la ejecución del proceso de aforo, se interrumpe el módulo de procesamiento, se cambia a la siguiente página y los archivos de resultados no son generados.

1.6. Página de resultados

Esta página aparece inmediatamente luego de haber culminado el módulo de procesamiento y generación de resultados o tras haber presionado el botón Detener. Su función es informar al usuario de la culminación del proceso de aforo vehicular e indicar la ubicación y nombre de los archivos generados con los resultados obtenidos. Un ejemplo se presenta en la Figura 1.10. Los resultados obtenidos se guardan automáticamente en la misma carpeta donde se encuentra el programa, como se presenta en la Figura 1.11 , de dos forma:

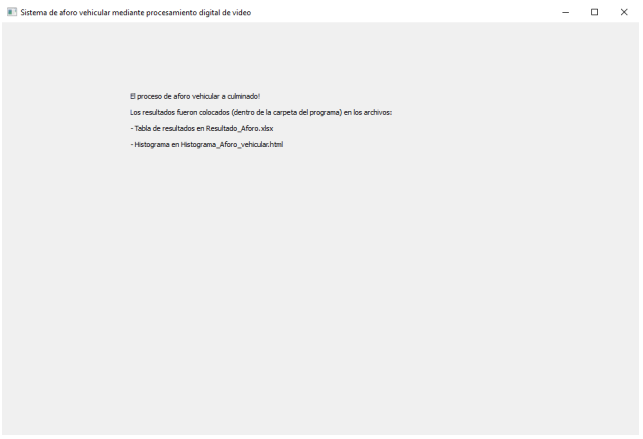


Figura 1.10: Página de culminación del proceso de aforo de la interfaz gráfica.

- Generación de un archivo Excel de nombre Resultado_Aforo.xlsx.
- Generación de un histograma de nombre Histograma_Aforo_Vehicular.html.

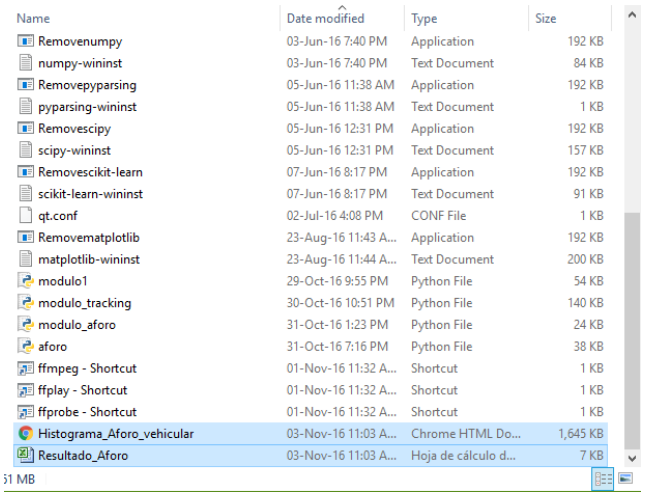


Figura 1.11: Carpeta del sistema con los archivos generados con los resultados.

En caso de haber presionado el botón Detener en la página anterior, el mensaje mostrado es distinto, informando de la interrupción del proceso, como se muestra en la Figura 1.12.

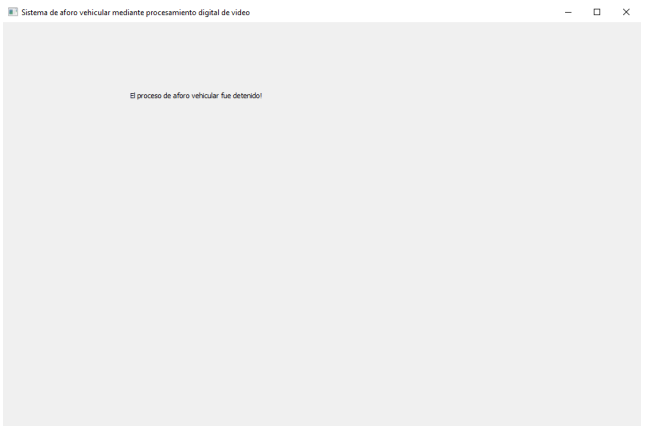


Figura 1.12: Página de culminación del proceso de aforo de la interfaz gráfica si se presiona el botón Detener.

1.7. Archivos resultantes

Finalmente el histograma y la tabla resumen, generados por el sistema, tienen la apariencia de las Figuras 1.13 y 1.14.

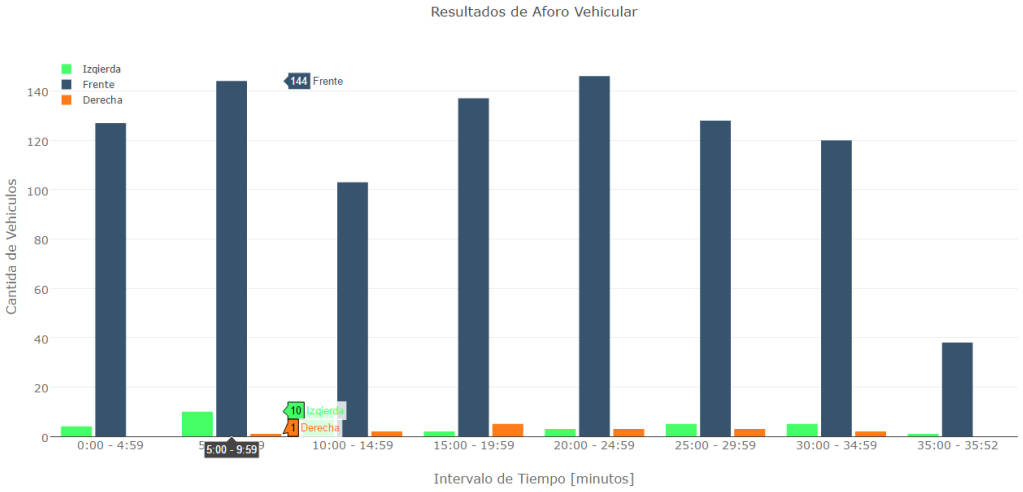


Figura 1.13: Histograma resultante de nombre Histograma_Aforo_Vehicular.html.

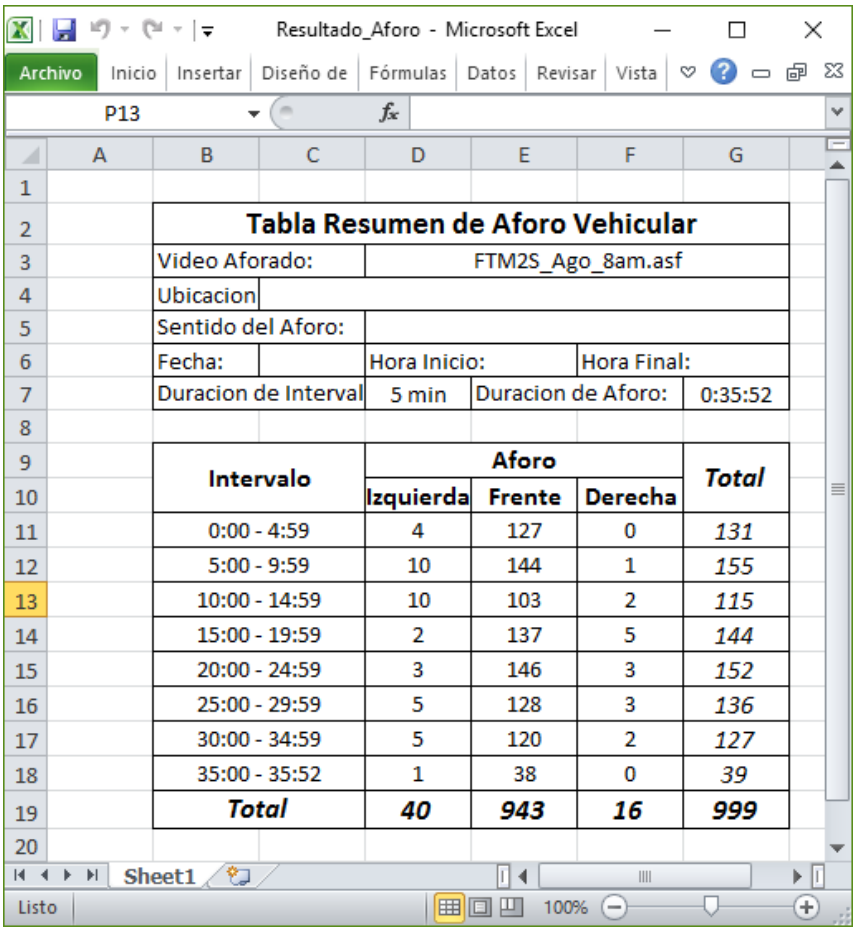


Tabla Resumen de Aforo Vehicular				
Video Aforado:	FTM2S_Ago_8am.asf			
Ubicacion				
Sentido del Aforo:				
Fecha:		Hora Inicio:	Hora Final:	
Duracion de Interval	5 min	Duracion de Aforo:	0:35:52	

Intervalo	Aforo			*Total*
	Izquierda	Frente	Derecha	
0:00 - 4:59	4	127	0	*131*
5:00 - 9:59	10	144	1	*155*
10:00 - 14:59	10	103	2	*115*
15:00 - 19:59	2	137	5	*144*
20:00 - 24:59	3	146	3	*152*
25:00 - 29:59	5	128	3	*136*
30:00 - 34:59	5	120	2	*127*
35:00 - 35:52	1	38	0	*39*
Total	***40***	***943***	***16***	***999***

Figura 1.14: Tabla resumen resultante de nombre Resultado_Aforo.xlsx.

Apéndice B

Códigos del sistema de aforo vehicular desarrollado

2.1. Código Principal

El código a continuación se encarga de mostrar la interfaz y realizar todas las operaciones requeridas, utilizando los módulo desarrollados y las librerías.

```
import sys
import os
from PyQt4.QtCore import *
from PyQt4.QtGui import *
from PyQt4.QtGui import QSizePolicy
import  cv2
import  numpy
import  math
import  re
import  datetime
import  modulo1
import  modulo_tracking
import  modulo_aforo
numpy.seterr(divide='ignore', invalid='ignore', over='ignore')
class App(QWidget):
    def __init__(self, parent=None):
```

```
        QWidget.__init__(self, parent)
        self.setWindowTitle("Sistema de aforo vehicular mediante proces\
amiento digital de video")
        self.resize(1000, 600)
        self.etiqueta_1 = QLabel("Seleccione el video que desea aforar"\
, self)
        self.etiqueta_1.move(50,50)
        self.etiqueta_1.resize(400,30)
        self.etiqueta_2 = QLabel("                                      \
            ", self)
        self.etiqueta_2.move(300,50)
        self.etiqueta_2.resize(600,30)
        self.boton_abrir = QPushButton("Abrir archivo", self)
        self.boton_abrir.move(700, 50)
        self.boton_cargar_video = QPushButton("Cargar Video", self)
        self.boton_cargar_video.move(800, 50)
        self.boton_cargar_video.hide()
        self.connect(self.boton_abrir, SIGNAL("clicked()"), self.Abrir_\
Video)
        self.connect(self.boton_cargar_video, SIGNAL("clicked()"), self\
.Cargar_Video)
        self.etiqueta_3 = QLabel("Seleccione los bordes de la intersecc\
ion: ", self)
        self.etiqueta_3.move(50,50)
        self.etiqueta_3.resize(400,30)
        self.etiqueta_3.hide()
        self.etiqueta_4 = QLabel("1. Elija un borde y haga dos click ce\
rca de sus esquinas, por fuera del paso peatonal.", self)
        self.etiqueta_4.move(100,100)
        self.etiqueta_4.resize(800,30)
        self.etiqueta_4.hide()
        self.etiqueta_5 = QLabel("2. Repita el paso (1) con otro borde \
hasta completar los 4 bordes.", self)
        self.etiqueta_5.move(100,150)
        self.etiqueta_5.resize(800,30)
        self.etiqueta_5.hide()
        self.etiqueta_6 = QLabel("3. Finalmente haga 2 clicks a lo anch\
o de la calle que cruza a la derecha, alejado de la interseccion.", sel\
f)
        self.etiqueta_6.move(100,200)
```

```
        self.etiqueta_6.resize(800,30)
        self.etiqueta_6.hide()
        self.boton_cargar_bordes = QPushButton("Cargar Bordes", self)
        self.boton_cargar_bordes.move(100, 300)
        self.boton_cargar_bordes.hide()
        self.boton_cargar_video_dnuevo = QPushButton("Volver a Seleccio\
nar Bordes", self)
        self.boton_cargar_video_dnuevo.setFixedWidth(170)
        self.boton_cargar_video_dnuevo.move(200, 300)
        self.boton_cargar_video_dnuevo.hide()
        self.connect(self.boton_cargar_bordes, SIGNAL("clicked()"), sel\
f.Cargar_Bordes)
        self.connect(self.boton_cargar_video_dnuevo, SIGNAL("clicked()"\
), self.Cargar_Video)
        self.etiqueta_7 = QLabel("Identifique cada borde que delimina l\
a interseccion, en el siguiente orden: ", self)
        self.etiqueta_7.move(50,50)
        self.etiqueta_7.resize(400,30)
        self.etiqueta_7.hide()
        self.etiqueta_8 = QLabel("1. Borde de entrada (por donde entran\
 los vehiculos de interes).", self)
        self.etiqueta_8.move(100,100)
        self.etiqueta_8.resize(800,30)
        self.etiqueta_8.hide()
        self.etiqueta_9 = QLabel("2. Borde de salida cuando los vehicul\
os de interes cruzan a su DERECHA. ", self)
        self.etiqueta_9.move(100,150)
        self.etiqueta_9.resize(800,30)
        self.etiqueta_9.hide()
        self.etiqueta_10 = QLabel("3. Borde de salida cuando los vehicu\
los de interes cruzan a su IZQUIERDA. ", self)
        self.etiqueta_10.move(100,200)
        self.etiqueta_10.resize(800,30)
        self.etiqueta_10.hide()
        self.etiqueta_11 = QLabel("Presione ENTER para seleccionar el b\
orde marcado y Flecha Dereccha para cambiar de borde a seleccionar. ", \
self)
        self.etiqueta_11.move(75,250)
        self.etiqueta_11.resize(800,30)
        self.etiqueta_11.hide()
```

```
        self.boton_determinar = QPushButton("Determinar", self)
        self.boton_determinar.move(100, 350)
        self.boton_determinar.hide()
        self.boton_determinar_dnuevo = QPushButton(" Volver a Cargar Bo\
rdes", self)
        self.boton_determinar_dnuevo.setFixedWidth(170)
        self.boton_determinar_dnuevo.move(200, 350)
        self.boton_determinar_dnuevo.hide()
        self.connect(self.boton_determinar, SIGNAL("clicked()"), self.I\
dentificar_Bordes)
        self.connect(self.boton_determinar_dnuevo, SIGNAL("clicked()"),\
 self.Cargar_Bordes)
        self.etiqueta_12 = QLabel("Configuracion de Aforo Vehicular: ",\
 self)
        self.etiqueta_12.move(20,50)
        self.etiqueta_12.hide()
        self.etiqueta_13 = QLabel("Indique la duracion del video:", sel\
f)
        self.etiqueta_13.move(50,80)
        self.etiqueta_13.hide()
        self.etiqueta_14 = QLabel("Seleccione el intervalo de tiempo de\
l Aforo Vehicular:", self)
        self.etiqueta_14.move(50,160)
        self.etiqueta_14.hide()
        self.boton_aforar = QPushButton("Aforar", self)
        self.boton_aforar.move(200, 350)
        self.boton_aforar.hide()
        self.boton_detener = QPushButton("Detener", self)
        self.boton_detener.move(200, 350)
        self.boton_detener.hide()
        self.hora = QLineEdit(self)
        self.hora.hide()
        self.min = QLineEdit(self)
        self.min.hide()
        self.seg = QLineEdit(self)
        self.seg.hide()
        self.hora1 = QLabel("Hora: ")
        self.hora1.hide()
        self.min1 = QLabel("Minutos: ")
        self.min1.hide()
```

```
self.seg1 = QLabel("Segundos: ")
self.seg1.hide()
self.radiobutton1 = QRadioButton('5 min', self)
self.radiobutton1.hide()
self.radiobutton2 = QRadioButton('10 min', self)
self.radiobutton2.hide()
self.radiobutton3 = QRadioButton('15 min', self)
self.radiobutton3.hide()
self.tiempo_aforo = QButtonGroup(self)
self.tiempo_aforo.addButton(self.radiobutton1)
self.tiempo_aforo.addButton(self.radiobutton2)
self.tiempo_aforo.addButton(self.radiobutton3)
self.ver_aforo = QButtonGroup(self)
self.radiobutton1.setChecked(True)
self.grid = QVBoxLayout(self)
self.grid1 = QHBoxLayout(self)
self.grid2 = QHBoxLayout(self)
self.grid3 = QHBoxLayout(self)
self.grid4 = QHBoxLayout(self)
self.grid1.addWidget(self.hora1)
self.grid1.addWidget(self.hora)
self.grid1.addWidget(self.min1)
self.grid1.addWidget(self.min)
self.grid1.addWidget(self.seg1)
self.grid1.addWidget(self.seg)
self.grid1.addStretch(1)
self.grid2.addWidget(self.radiobutton1)
self.grid2.addWidget(self.radiobutton2)
self.grid2.addWidget(self.radiobutton3)
self.grid2.addStretch(1)
self.grid4.addStretch(1)
self.grid.addWidget(self.etiqueta_12)
self.grid.addSpacing(20)
self.grid.addWidget(self.etiqueta_13)
self.grid.addSpacing(20)
self.grid.addLayout(self.grid1)
self.grid.addSpacing(20)
self.grid.addWidget(self.etiqueta_14)
self.grid.addSpacing(20)
self.grid.addLayout(self.grid2)
```

```
        self.grid.addSpacing(20)
        self.grid.addSpacing(20)
        self.grid.addLayout(self.grid4)
        self.grid.addSpacing(20)
        self.grid.addWidget(self.boton_aforar)
        self.grid.addSpacing(500)
        self.connect(self.boton_aforar, SIGNAL("clicked()"), self.afora\
r)
        self.radiobutton1.toggled.connect(lambda:self.btnstate(self.rad\
iobutton1))
        self.radiobutton2.toggled.connect(lambda:self.btnstate(self.rad\
iobutton2))
        self.radiobutton3.toggled.connect(lambda:self.btnstate(self.rad\
iobutton3))
        self.connect(self.boton_detener, SIGNAL("clicked()"), self.dete\
ner)
        self.time_aforo     = 5
        self.bool_ver_aforo = True
        self.bool_detener   = False
        self.etiqueta_18 = QLabel("Realizando aforo vehicular ... ", se\
lf)
        self.etiqueta_18.move(200, 100)
        self.etiqueta_18.resize(400,30)
        self.etiqueta_18.hide()
        self.etiqueta_19 = QLabel("El proceso de aforo vehicular a culm\
inado!", self)
        self.etiqueta_19.move(200, 100)
        self.etiqueta_19.resize(400,30)
        self.etiqueta_19.hide()
        self.etiqueta_20 = QLabel("Los resultados fueron colocados (den\
tro de la carpeta del programa) en los archivos:", self)
        self.etiqueta_20.move(200, 125)
        self.etiqueta_20.resize(500,30)
        self.etiqueta_20.hide()
        self.etiqueta_21 = QLabel("- Tabla de resultados en Resultado_A\
foro.xlsx", self)
        self.etiqueta_21.move(200, 150)
        self.etiqueta_21.resize(400,30)
        self.etiqueta_21.hide()
        self.etiqueta_22 = QLabel("- Histograma en Histograma_Aforo_veh\
```

```
icular.html", self)
        self.etiqueta_22.move(200, 175)
        self.etiqueta_22.resize(400,30)
        self.etiqueta_22.hide()
        self.etiqueta_23 = QLabel("El proceso de aforo vehicular fue de\
tenido!", self)
        self.etiqueta_23.move(200, 100)
        self.etiqueta_23.resize(400,30)
        self.etiqueta_23.hide()
        self.hora.returnPressed.connect(self.boton_aforar.click)
        self.min.returnPressed.connect(self.boton_aforar.click)
        self.hora.returnPressed.connect(self.boton_aforar.click)
    def btnstate(self,b):
        if b.text() == '5 min':
            if b.isChecked() == True:
                self.time_aforo = 5
        if b.text() == "10 min":
            if b.isChecked() == True:
                self.time_aforo = 10
        if b.text() == "15 min":
            if b.isChecked() == True:
                self.time_aforo = 15
    def btnstate3(self,d):
        if d.text() == "Si":
            if d.isChecked() == True:
                self.bool_ver_aforo = True
        if d.text() == "No":
            if d.isChecked() == True:
                self.bool_ver_aforo = False
    def Abrir_Video(self):
        global fps, width, height, dis_fueradeimagen, mhi, largo_rastro\
, kernel_dilate7, img_foreg_prom, mask_backg_total_1, mask_backg_total_\
2, img_backg
        cv2.destroyAllWindows()
        self.boton_cargar_video.hide()
        self.archivo = str(QFileDialog.getOpenFileName(self, "Abrir vid\
eo"))
        bool_video_correcto = True
        videoCapture = cv2.VideoCapture(self.archivo)
        try:
```

```
            fps = videoCapture.get(cv2.cv.CV_CAP_PROP_FPS)
            success, frame = videoCapture.read()
            success, frame = videoCapture.read()
            success, frame = videoCapture.read()
            size = frame.shape[2]
        except:
            bool_video_correcto = False
        if (not videoCapture.isOpened()) or (not bool_video_correcto):
            print '\n
            self.etiqueta_2.setText('EL ARCHIVO SELECCIONADO ES INVALID\
O')
        else:
            videoCapture, fps, width, height = modulo1.abrir_video(self\
.archivo)
            img_backg = modulo1.fondo_promedio(self.archivo)
            cv2.imshow('Vista previa de la interseccion', img_backg )
            dis_fueradeimagen   = (width ** 2 + height ** 2 + 1)
            mhi                 = numpy.zeros((height,width), numpy.flo\
at32)
            largo_rastro        = 10
            kernel_dilate7      = numpy.ones((7,7), numpy.uint8)
            img_foreg_prom      = numpy.zeros((height,width), numpy.flo\
at32)
            mask_backg_total_1  = 255 * numpy.ones((height,width), nump\
y.float32)
            mask_backg_total_2  = 255 * numpy.ones((height,width), nump\
y.float32)
            self.etiqueta_2.setText(self.archivo.split("/")[-1])
            self.boton_cargar_video.show()
    def Cargar_Video(self):
        global coord
        self.boton_cargar_bordes.hide()
        self.boton_cargar_video_dnuevo.hide()
        del coord[:]
        self.etiqueta_1.hide()
        self.etiqueta_2.hide()
        self.boton_abrir.hide()
        self.boton_cargar_video.hide()
        self.etiqueta_3.show()
        self.etiqueta_4.show()
```

```
        self.etiqueta_5.show()
        self.etiqueta_6.show()
        cv2.destroyAllWindows()
        coord = modulo1.guardar_coord_clicks(img_backg)
        self.boton_cargar_bordes.show()
        self.boton_cargar_video_dnuevo.show()
    def Cargar_Bordes(self):
        global p_umbral, inters_mask, img_backg_int, coord
        global umbral_entra, umbral_frente, umbral_izq, umbral_der, tol\
_der, coord_borde_entrada, coord_borde_derecha, p_der, coord_der, p_ent\
ra_izq
        global coord_inters, B_entra, C_entra, B_entra_amp, C_entra_amp\
, B_frente, C_frente, B_izq, C_izq, B_der, C_der, B_der_amp, C_der_amp
        self.boton_determinar.hide()
        self.boton_determinar_dnuevo.hide()
        self.etiqueta_3.hide()
        self.etiqueta_4.hide()
        self.etiqueta_5.hide()
        self.etiqueta_6.hide()
        self.boton_cargar_bordes.hide()
        self.boton_cargar_video_dnuevo.hide()
        self.etiqueta_7.show()
        self.etiqueta_8.show()
        self.etiqueta_9.show()
        self.etiqueta_10.show()
        self.etiqueta_11.show()
        cv2.destroyAllWindows()
        coord_borde, limites_inters, umbrales, p_umbral, inters_mask, i\
mg_backg_int = modulo1.calc_limites_interseccion(width, height, coord, \
img_backg)
        umbral_entra, umbral_frente, umbral_izq, umbral_der, tol_der   \
            = umbrales
        coord_borde_entrada, coord_borde_derecha, p_der, coord_der, p_e\
ntra_izq      = coord_borde
        coord, coord_inters, B_entra, C_entra, B_entra_amp, C_entra_amp\
, B_frente, C_frente, B_izq, C_izq, B_der, C_der, B_der_amp, C_der_amp \
= limites_inters
        if not coord_der == []:
            B_der = B_der_amp
            C_der = C_der_amp
```

```
        self.boton_determinar.show()
        self.boton_determinar_dnuevo.show()
    def Identificar_Bordes(self):
        self.etiqueta_7.hide()
        self.etiqueta_8.hide()
        self.etiqueta_9.hide()
        self.etiqueta_10.hide()
        self.etiqueta_11.hide()
        self.boton_determinar.hide()
        self.boton_determinar_dnuevo.hide()
        self.etiqueta_12.show()
        self.etiqueta_13.show()
        self.hora.show()
        self.min.show()
        self.seg.show()
        self.hora1.show()
        self.min1.show()
        self.seg1.show()
        self.etiqueta_14.show()
        self.radiobutton1.show()
        self.radiobutton2.show()
        self.radiobutton3.show()
        self.boton_aforar.show()
        self.setLayout(self.grid)
        cv2.destroyAllWindows()
    def aforar(self):
        global fps, width, height, dis_fueradeimagen, mhi, largo_rastro\
, kernel_dilate7, img_foreg_prom, mask_backg_total_1, mask_backg_total_\
2, img_backg
        global p_umbral, inters_mask, img_backg_int, coord
        global umbral_entra, umbral_frente, umbral_izq, umbral_der, tol\
_der, coord_borde_entrada, coord_borde_derecha, p_der, coord_der, p_ent\
ra_izq
        global coord_inters, B_entra, C_entra, B_entra_amp, C_entra_amp\
, B_frente, C_frente, B_izq, C_izq, B_der, C_der, B_der_amp, C_der_amp
        global cont_frame, bool_mask_backg, kernel_dilate, kernel_erode\
, coord_random, tol_area, vchequeo, vactuales, vlistos, n_vehiculo
        global umbral_adyacente, umbral_relacion_area, umbral_blob_apro\
x, wh_adya_ampl, index_listos
        global cont_vehic_frente, cont_vehic_izq, cont_vehic_der, frame\
```

```
_ultimo, cont_vehic_histo, interv_histo_min
        self.etiqueta_12.hide()
        self.etiqueta_13.hide()
        self.hora.hide()
        self.min.hide()
        self.seg.hide()
        self.hora1.hide()
        self.min1.hide()
        self.seg1.hide()
        self.etiqueta_14.hide()
        self.radiobutton1.hide()
        self.radiobutton2.hide()
        self.radiobutton3.hide()
        self.boton_aforar.hide()
        self.etiqueta_18.show()
        self.boton_detener.show()
        interv_histo_min    = int( self.time_aforo  )
        duracion_horas      = int( self.hora.text() ) if not self.hora.\
text() == '' else 0
        duracion_minutos    = int( self.min.text()  ) if not self.min.t\
ext()  == '' else 0
        duracion_segundos   = int( self.seg.text()  ) if not self.seg.t\
ext()  == '' else 0
        print 'intervalos de:', interv_histo_min, 'duracion:', duracion\
_horas, duracion_minutos, duracion_segundos
        duracion_total_seg  = duracion_segundos + ( duracion_minutos + \
duracion_horas * 60 ) * 60
        interv_histo_seg    = ( interv_histo_min ) * 60
        videoCapture, fps, width, height = modulo1.abrir_video(self.arc\
hivo)
        success = True
        t_frame = datetime.datetime.now()
        print 'realizando aforo vehicular ( finaliza en aproximadamente\
', str( int( 0.7 * duracion_total_seg / 60 ) ) ,'minutos )...'
        if not self.bool_ver_aforo:
            cv2.namedWindow('Aforando (NO CERRAR ESTA VENTANA)')
            cv2.resizeWindow('Aforando (NO CERRAR ESTA VENTANA)', 400, \
1)
        while success and (not self.bool_detener):
            cv2.waitKey(1)
```

```
            success, frame = videoCapture.read()
            if not success: continue
            res_calc_per_frame = modulo_aforo.calc_per_frame( cont_fram\
e, frame, img_backg, img_foreg_prom, mask_backg_total_1, mask_backg_tot\
al_2,
                                                              bool_ma\
sk_backg, mhi, coord_inters, p_der, coord_der, kernel_dilate, kernel_er\
ode,
                                                              coord_b\
orde_entrada, coord_borde_derecha, inters_mask, width, height, coord_ra\
ndom, tol_area,
                                                              p_umbra\
l, largo_rastro, vchequeo, vactuales, vlistos, n_vehiculo, B_entra, C_e\
ntra,
                                                              B_entra\
_amp, C_entra_amp, B_frente, C_frente, B_izq, C_izq,  B_der, C_der, p_e\
ntra_izq,
                                                              umbral_\
entra, umbral_frente, umbral_izq, umbral_der, umbral_adyacente, umbral_\
relacion_area,
                                                              umbral_\
blob_aprox, tol_der, wh_adya_ampl, dis_fueradeimagen, index_listos, con\
t_vehic_frente,
                                                              cont_ve\
hic_izq, cont_vehic_der, frame_ultimo, cont_vehic_histo )
            cont_frame, img_foreg_prom, mask_backg_total_1, mask_backg_\
total_2, bool_mask_backg, mhi, \
            coord_random, vchequeo, vactuales, vlistos, n_vehiculo, ind\
ex_listos, cont_vehic_frente, \
            cont_vehic_izq, cont_vehic_der, frame_ultimo, cont_vehic_hi\
sto, img_fast, segmask1              = res_calc_per_frame
            if self.bool_ver_aforo:
                cv2.imshow('Aforando', img_fast)
        cv2.destroyAllWindows()
        self.boton_detener.hide()
        t_final = datetime.datetime.now()
        if not self.bool_detener:
            modulo_aforo.calc_pos_video(cont_frame, duracion_total_seg,\
 interv_histo_seg, cont_vehic_histo, self.archivo )
            cont_frame += 1
```

```
            t_frame_final = ((t_final - t_frame)).total_seconds()
            fps_promedio = float(cont_frame) / t_frame_final
            if t_frame_final == 0:
                print "\n[INFO] No se proceso el video"
            else:
                print '\n[INFO] *Se detuvo en el cuadro: ', cont_frame
                print '[INFO] *Resultados del aforo:'
                print '[INFO]\t\tVehiculos que siguieron derecho:      \
 {0}'.format(cont_vehic_frente)
                print '[INFO]\t\tVehiculos que cruzaron a la izquierda:\
 {0}'.format(cont_vehic_izq)
                print '[INFO]\t\tVehiculos que cruzaron a la derecha:  \
 {0}'.format(cont_vehic_der)
                print '[INFO] *Tiempo de procesamiento:'
                print "[INFO]\t\tproceso:\t\t{:.4} minutos".format(t_fr\
ame_final/60.0)
                print "[INFO]\t\tcalculo por cuadro:\t{:.4} s".format( \
1 / fps_promedio )
                print "[INFO]\t\tfps:\t\t\t{:.4}".format( fps_promedio \
)
        else:
            print 'Se detuvo el proceso de aforo'
        self.etiqueta_18.hide()
        if not self.bool_detener:
            self.etiqueta_19.show()
            self.etiqueta_20.show()
            self.etiqueta_21.show()
            self.etiqueta_22.show()
        else:
            self.etiqueta_23.show()
    def detener(self):
        self.bool_detener = True
umbral_adyacente        = 30
umbral_blob_aprox       = 10
umbral_relacion_area    = 0.6
tol_area                = 0.5
wh_adya_ampl            = 125
cont_vehic_frente   = 0
cont_vehic_izq      = 0
cont_vehic_der      = 0
```

```
index_listos          = -1
cont_frame            = -1
n_vehiculo            = 0
coord         = []
coord_blob    = []
coord_obj     = []
vchequeo      = []
vactuales     = []
vlistos       = []
cont_vehic_histo = []
kernel_dilate         = numpy.ones((5,5), numpy.uint8)
kernel_erode          = numpy.ones((7,7), numpy.uint8)
largo_rastro          = 10
kernel_dilate7        = numpy.ones((7,7), numpy.uint8)
bool_mask_backg       = 1
coord_random          = []
frame_ultimo          = 0
if __name__ == "__main__":
    app = QApplication(sys.argv)
    qb = App()
    qb.show()
    sys.exit(app.exec_())
```

2.2. Módulo Principal

El código a continuación contiene las funciones que engloban los procesos utilizados posteriores y durante el análisis del video.

```
import cv2
import numpy
import math
import plotly
import xlsxwriter
import os
import re
import datetime
import modulo1
```

```
import modulo_tracking
def calc_per_frame(cont_frame, frame, img_backg, img_foreg_prom, mask_b\
ackg_total_1, mask_backg_total_2, bool_mask_backg, mhi, coord_inters, p\
_der, \
                   coord_der, kernel_dilate, kernel_erode, coord_borde\
_entrada, coord_borde_derecha, inters_mask, width, height, coord_random\
, tol_area, \
                   p_umbral, largo_rastro, vchequeo, vactuales, vlisto\
s, n_vehiculo, B_entra, C_entra, B_entra_amp, C_entra_amp, B_frente, C_\
frente, \
                   B_izq, C_izq, B_der, C_der, p_entra_izq, umbral_en\
tra, umbral_frente, umbral_izq, umbral_der, umbral_adyacente, umbral_re\
lacion_area, \
                   umbral_blob_aprox, tol_der, wh_adya_ampl, dis_fuera\
deimagen, index_listos, cont_vehic_frente, cont_vehic_izq, cont_vehic_d\
er, \
                   frame_ultimo, cont_vehic_histo ):
    cont_frame          += 1
    coord_blob_frame     = []
    coord_obj_frame      = []
    resultado_foreground = modulo1.foreground(cont_frame, frame, img_ba\
ckg, img_foreg_prom, mask_backg_total_1, mask_backg_total_2, bool_mask_\
backg,
                                              mhi, coord_inters, p_de\
r, coord_der,
                                              kernel_dilate, kernel_e\
rode, coord_borde_entrada, coord_borde_derecha, inters_mask,
                                              width, height, coord_ra\
ndom)
    img_foreg_conborde, img_foreg, img_foreg_nsombra, segmask, bounding\
rects, mhi, img_foreg_prom, \
                                  mask_backg_total_1, mask_backg_total_2,\
 bool_mask_backg, coord_random = resultado_foreground
    _,segmask1 = cv2.threshold(segmask,0.1,255,cv2.THRESH_BINARY)
    segmask1 = cv2.cvtColor(segmask1, cv2.COLOR_GRAY2BGR)
    detec_obj = []
    detec_obj = [rect for rect in boundingrects if ( math.sqrt( rect[2]\
*rect[3] ) + tol_area ) > ( -16 + 0.166 * rect[1] ) ]
    img_fast = frame.copy()
    for i in range(4):
```

```
        if not coord_der == []:
            if ( tuple(coord_inters[i]) in coord_borde_derecha ) and ( \
tuple(coord_inters[i + 1]) in coord_borde_derecha ):
                cv2.line(img_fast, p_der[0], p_der[1], [255,255,255], 2\
)
                cv2.line(img_fast, p_der[1], coord_der[0], [255,255,255\
], 2)
                cv2.line(img_fast, coord_der[0], coord_der[1], [255,255\
,255], 2)
            else:
                cv2.line(img_fast, tuple(coord_inters[i]), tuple(coord_\
inters[i + 1]), [255,255,255], 2)
        else:
            cv2.line(img_fast, tuple(coord_inters[i]), tuple(coord_inte\
rs[i + 1]), [255,255,255], 2)
    for i, rect in enumerate(detec_obj):
        x, y, w, h = rect
        cv2.rectangle(segmask1, (x, y), (x + w, y + h), (255,255,0), 1)\

        cv2.circle(segmask1, (x + w/2, y + h/2), 3, (0,0,255), -1)
        coord_blob_frame.append([i, rect ])
    list_cood_v, img_fast, cont_vehiculos = modulo_tracking.main_tracki\
ng( cont_frame, coord_blob_frame, vchequeo, vactuales, vlistos, n_vehic\
ulo,
                                                                    \
    width, height,
                                                                    \
    B_entra, C_entra, B_entra_amp, C_entra_amp, B_frente, C_frente, B_\
izq, C_izq,
                                                                    \
    B_der, C_der, p_entra_izq, umbral_entra, umbral_frente, umbral_izq\
, umbral_der,
                                                                    \
    umbral_adyacente, umbral_relacion_area, umbral_blob_aprox, tol_der\
, wh_adya_ampl,
                                                                    \
    dis_fueradeimagen, inters_mask, img_fast, index_listos,
                                                                    \
    cont_vehic_frente, cont_vehic_izq, cont_vehic_der, frame_ultimo)
    coord_blob_frame, vchequeo, vactuales, vlistos, n_vehiculo      \
```

```
                = list_cood_v
    cont_vehic_frente, cont_vehic_izq, cont_vehic_der, frame_ultimo, in\
dex_listos      = cont_vehiculos
    if cont_frame %2 == 0:
        cont_vehic_histo.append( (cont_vehic_izq, cont_vehic_frente, co\
nt_vehic_der) )
    return cont_frame, img_foreg_prom, mask_backg_total_1, mask_backg_t\
otal_2, bool_mask_backg, mhi, coord_random, vchequeo, vactuales, vlisto\
s, n_vehiculo, \
           index_listos, cont_vehic_frente, cont_vehic_izq, cont_vehic\
_der, frame_ultimo, cont_vehic_histo, img_fast, segmask1
def draw_histograma( cont_frame, duracion_total_seg, interv_histo_seg, \
cont_vehic_histo ):
    fps_promedio        = float( cont_frame + 1 ) / float(duracion_tota\
l_seg)
    print 'fps_promedio', fps_promedio
    interv_histo_frame  = float(interv_histo_seg) * fps_promedio
    print 'interv_histo_frame', interv_histo_frame
    ihm                 = ( interv_histo_seg / 60 ) - 1
    list_histo = []
    acum_interv_histo_frame = interv_histo_frame
    j = 0
    len_cont_vehic_histo = len( cont_vehic_histo )
    while True:
        i = int( acum_interv_histo_frame / 2 )
        if i >= len_cont_vehic_histo:
            i = len_cont_vehic_histo - 1
            valor_local = ( cont_vehic_histo[i][0] - cont_vehic_histo[j\
][0],
                            cont_vehic_histo[i][1] - cont_vehic_histo[j\
][1],
                            cont_vehic_histo[i][2] - cont_vehic_histo[j\
][2] )
            list_histo.append( valor_local )
            print 'Ultimo intervalo', j
            break
        valor_local = ( cont_vehic_histo[i][0] - cont_vehic_histo[j][0]\
,
                        cont_vehic_histo[i][1] - cont_vehic_histo[j][1]\
,
```

```
                    cont_vehic_histo[i][2] - cont_vehic_histo[j][2]\
 )
        list_histo.append( valor_local )
        j = i
        acum_interv_histo_frame += interv_histo_frame
    print 'Histograma (en lista):', list_histo
    x = range(0, len(list_histo) * ( interv_histo_seg / 60 ) , ( interv\
_histo_seg / 60 ) )
    if x[-1] < 60:
        x = [ '{}:00 - {}:59'.format( t, t + ihm )  for t in x[:-1] ] +\
 [ '{}:00 - {}:{}'.format( x[-1], duracion_total_seg / 60, duracion_tot\
al_seg %60 ) ]
    else:
        x = [ ( t, 00, t + ihm, 59 ) for t in x[:-1] ] +  [ ( x[-1], 00\
, duracion_total_seg / 60, duracion_total_seg %60 ) ]
        x = [ '{}:{:02d}:{:02d} - {}:{:02d}:{:02d}'.format( (tm1 // 60)\
, (tm1%60), ts1, (tm2 // 60), (tm2&60), ts2 ) for tm1, ts1, tm2, ts2 in\
 x ]
    y_izq       = [ y for y,_,_ in list_histo ]
    y_frente    = [ y for _,y,_ in list_histo ]
    y_der       = [ y for _,_,y in list_histo ]
    trace1      = plotly.graph_objs.Bar( x = x, y = y_izq,      name = \
'Izqierda',  marker = dict( color = 'rgb(70, 255, 102)' ) )
    trace2      = plotly.graph_objs.Bar( x = x, y = y_frente,   name = \
'Frente',    marker = dict( color = 'rgb(55, 83, 109)' ) )
    trace3      = plotly.graph_objs.Bar( x = x, y = y_der,      name = \
'Derecha',   marker = dict( color = 'rgb(255, 123, 26)' ) )
    data        = [trace1, trace2, trace3 ]
    layout      = plotly.graph_objs.Layout( title = 'Resultados de Afor\
o Vehicular',
                                            xaxis = dict( title='Interv\
alo de Tiempo [minutos]',
                                                          titlefont   = d\
ict( size = 16, color = 'rgb(107, 107, 107)' ),
                                                          tickfont    = d\
ict( size = 14, color = 'rgb(107, 107, 107)' ) ),
                                            yaxis=dict( title       = '\
Cantida de Vehiculos',
                                                          titlefont   = d\
ict( size = 16, color = 'rgb(107, 107, 107)' ),
```

```
                                                    tickfont    = d\
ict( size = 14, color = 'rgb(107, 107, 107)' ) ),
                                                legend = dict( x = 0, y = 1\
.0,
                                                    bgcolor = 'rgba\
(255, 255, 255, 0)',
                                                    bordercolor = '\
rgba(255, 255, 255, 0)' ),
                                                barmode     = 'group',
                                                bargap      = 0.15,
                                                bargroupgap = 0.1 )
    fig_hist    = plotly.graph_objs.Figure( data = data, layout = layou\
t )
    plotly.offline.plot( fig_hist, auto_open = False )
    if 'Histograma_Aforo_vehicular.html' in os.listdir(os.getcwd()):
        os.remove('Histograma_Aforo_vehicular.html')
    os.rename('temp-plot.html','Histograma_Aforo_vehicular.html')
    return list_histo
def genera_excel( list_histo, duracion_total_seg, interv_histo_seg, vid\
eo_name ):
    video_name = str(video_name)
    ihm = ( interv_histo_seg / 60 ) - 1
    x = range(0, len(list_histo) * ( interv_histo_seg / 60 ) , ( interv\
_histo_seg / 60 ) )
    if x[-1] < 60:
        x = [ '{}:00 - {}:59'.format( t, t + ihm ) for t in x[:-1] ] +\
 [ '{}:00 - {}:{}'.format( x[-1], duracion_total_seg / 60, duracion_tot\
al_seg %60 ) ]
    else:
        x = [ ( t, 00, t + ihm, 59 ) for t in x[:-1] ] + [ ( x[-1], 00\
, duracion_total_seg / 60, duracion_total_seg %60 ) ]
        x = [ '{}:{:02d}:{:02d} - {}:{:02d}:{:02d}'.format( (tm1 // 60)\
, (tm1%60), ts1, (tm2 // 60), (tm2&60), ts2 ) for tm1, ts1, tm2, ts2 in\
 x ]
    y_izq       = [ y for y,_,_ in list_histo ]
    y_frente    = [ y for _,y,_ in list_histo ]
    y_der       = [ y for _,_,y in list_histo ]
    workbook    = xlsxwriter.Workbook('Resultado_Aforo.xlsx')
    worksheet   = workbook.add_worksheet()
    format_14           = workbook.add_format({ 'bold'      : True,
```

```
                                        'border'    : 1,
                                        'align'     : 'center',\

                                        'valign'    : 'vcenter'\
,
                                        'font_size' : 14})
    format_izq          = workbook.add_format({ 'border'    : 1,
                                        'align'     : 'left',
                                        'valign'    : 'vcenter'\
,
                                        'font_size' : 11})
    format_cent         = workbook.add_format({ 'border'    : 1,
                                        'align'     : 'center',\

                                        'valign'    : 'vcenter'\
,
                                        'font_size' : 11})
    format_cent_bold    = workbook.add_format({ 'bold'      : True,
                                        'border'    : 1,
                                        'align'     : 'center',\

                                        'valign'    : 'vcenter'\
,
                                        'font_size' : 12})
    format_total        = workbook.add_format({ 'italic'    : True,
                                        'border'    : 1,
                                        'align'     : 'center',\

                                        'valign'    : 'vcenter'\
,
                                        'font_size' : 12})
    format_total_bold   = workbook.add_format({ 'bold'      : True,
                                        'italic'    : True,
                                        'border'    : 1,
                                        'align'     : 'center',\

                                        'valign'    : 'vcenter'\
,
                                        'font_size' : 13})
    worksheet.merge_range( 'B2:G2', 'Tabla Resumen de Aforo Vehicular',\
```

```
 format_14)
    worksheet.merge_range( 'B3:C3', 'Video Aforado:', format_izq)
    worksheet.merge_range( 'D3:G3', video_name, format_cent)
    worksheet.write( 'B4',          'Ubicacion:', format_izq)
    worksheet.merge_range( 'C4:G4', '', format_cent)
    worksheet.merge_range( 'B5:C5', 'Sentido del Aforo:', format_izq)
    worksheet.merge_range( 'D5:G5', '', format_cent)
    worksheet.write( 'B6',          'Fecha:', format_izq)
    worksheet.write( 'C6',          '', format_cent)
    worksheet.write( 'D6',          'Hora Inicio:', format_izq)
    worksheet.write( 'E6',          '', format_cent)
    worksheet.write( 'F6',          'Hora Final:', format_izq)
    worksheet.write( 'G6',          '', format_cent)
    worksheet.merge_range( 'B7:C7', 'Duracion de Intervalos:', format_i\
zq)
    worksheet.write( 'D7',          '{} min'.format( interv_histo_seg /\
 60 ), format_cent)
    worksheet.merge_range( 'E7:F7', 'Duracion de Aforo:', format_izq)
    duracion_aforo = '{}:{:02d}:{:02d}'.format( (duracion_total_seg/60/\
/60), (duracion_total_seg/60%60), duracion_total_seg%60%60)
    worksheet.write('G7',           duracion_aforo, format_cent)
    worksheet.merge_range( 'B9:C10','Intervalo', format_cent_bold)
    worksheet.merge_range( 'D9:F9', 'Aforo', format_cent_bold)
    worksheet.write( 'D10',         'Izquierda', format_cent_bold)
    worksheet.write( 'E10',         'Frente', format_cent_bold)
    worksheet.write( 'F10',         'Derecha', format_cent_bold)
    worksheet.merge_range( 'G9:G10','Total', format_total_bold)
    for i in range(10, 10 + len(list_histo)):
        worksheet.merge_range( i,1, i,2, x[i - 10], format_cent)
        worksheet.write(i, 3, y_izq[i - 10],    format_cent)
        worksheet.write(i, 4, y_frente[i - 10], format_cent)
        worksheet.write(i, 5, y_der[i - 10],    format_cent)
        cell1 = xlsxwriter.utility.xl_rowcol_to_cell(i, 3)
        cell2 = xlsxwriter.utility.xl_rowcol_to_cell(i, 5)
        worksheet.write_formula(i, 6, '=SUM(' + cell1 + ':' + cell2 +')\
', format_total )
    worksheet.merge_range( (i + 1),1, (i + 1),2, 'Total', format_total_\
bold)
    cell1 = xlsxwriter.utility.xl_rowcol_to_cell(10, 3)
    cell2 = xlsxwriter.utility.xl_rowcol_to_cell(i, 3)
```

```
    worksheet.write_formula((i + 1), 3, '=SUM(' + cell1 + ':' + cell2 +\
')', format_total_bold )
    cell1 = xlsxwriter.utility.xl_rowcol_to_cell(10, 4)
    cell2 = xlsxwriter.utility.xl_rowcol_to_cell(i, 4)
    worksheet.write_formula((i + 1), 4, '=SUM(' + cell1 + ':' + cell2 +\
')', format_total_bold )
    cell1 = xlsxwriter.utility.xl_rowcol_to_cell(10, 5)
    cell2 = xlsxwriter.utility.xl_rowcol_to_cell(i, 5)
    worksheet.write_formula((i + 1), 5, '=SUM(' + cell1 + ':' + cell2 +\
')', format_total_bold )
    cell1 = xlsxwriter.utility.xl_rowcol_to_cell(10, 6)
    cell2 = xlsxwriter.utility.xl_rowcol_to_cell(i, 6)
    worksheet.write_formula((i + 1), 6, '=SUM(' + cell1 + ':' + cell2 +\
')', format_total_bold )
    workbook.close()
def calc_pos_video(cont_frame, duracion_total_seg, interv_histo_seg, co\
nt_vehic_histo, video_name):
    print 'graficando el histograma...'
    list_histo = draw_histograma( cont_frame, duracion_total_seg, inter\
v_histo_seg, cont_vehic_histo )
    print 'copiando resultados en excel...'
    genera_excel( list_histo, duracion_total_seg, interv_histo_seg, vid\
eo_name )
```

2.3. Módulo de Rastreo

El código a continuación contiene las funciones que forman el módulo de rastreo.

```
import cv2
import numpy
import math
import re
import modulo1
numpy.seterr(divide='ignore', invalid='ignore', over='ignore')
def iden_vehiculos_adyacentes( cont_frame, coord_blob_frame, vchequeo, \
vactuales, B_entra, C_entra, umbral_entra, umbral_adyacente, umbral_rel\
```

```
acion_area ):
    copiar_borrar_chequeo = []
    copiar_borrar_actuales = []
    for i, rect in coord_blob_frame:
        bool_skip = False
        x, y, w, h = rect
        dis_coord_blob = modulo1.dis_punto_borde((x + w/2), (y + h/2), \
B_entra, C_entra)
        for i_chequeo, blob_chequeo in enumerate(vchequeo):
            bool_cerca_entra = False
            bool_skip = False
            f_chequeo, rect_chequeo, cant_v, flag = blob_chequeo[-1]
            if (not f_chequeo == (cont_frame - 1)) or rect_chequeo is N\
one: continue
            x_chequeo, y_chequeo, w_chequeo, h_chequeo = rect_chequeo
            if not ((x + w/2) - (x_chequeo + w_chequeo/2))**2 + ((y + h\
/2) - (y_chequeo + h_chequeo/2))**2 <= umbral_adyacente**2: continue
            dis_chequeo = modulo1.dis_punto_borde((x_chequeo + w_cheque\
o/2), (y_chequeo + h_chequeo/2), B_entra, C_entra)
            if not (dis_coord_blob - dis_chequeo) >= - 0.3: continue
            bool_skip = True
            copiar_borrar_chequeo.append([i,i_chequeo])
            break
        if bool_skip: continue
        for i_actuales, blob_actuales in enumerate(vactuales):
            bool_cerca_entra = False
            f_actuales, n_actuales, rect_actuales, cant_v_actuales, fla\
g = blob_actuales[-1]
            if (not f_actuales == (cont_frame - 1)) or rect_actuales is\
 None: continue
            x_actuales, y_actuales, w_actuales, h_actuales = rect_actua\
les
            if not ((x + w/2) - (x_actuales + w_actuales/2))**2 + ((y +\
 h/2) - (y_actuales + h_actuales/2))**2 < umbral_adyacente**2: continue\


            dis_actuales = modulo1.dis_punto_borde((x_actuales + w_actu\
ales/2), (y_actuales + h_actuales/2), B_entra, C_entra)
            if not (dis_coord_blob - dis_actuales) >= - 1: continue
            relacion_area = (w * h) / ( float(w_actuales) * h_actuales)\
```

```
            if relacion_area > 1:
                relacion_area = 1 / relacion_area
            esquinas = [ (x_actuales, y_actuales),
                         (x_actuales + w_actuales, y_actuales),
                         (x_actuales, y_actuales + h_actuales),
                         (x_actuales + w_actuales, y_actuales + h_actua\
les) ]
            for px, py in esquinas:
                dis_entra   = modulo1.dis_punto_borde(px, py, B_entra, \
C_entra)
                if dis_entra < umbral_entra:
                    bool_cerca_entra = True
                    break
            if bool_cerca_entra:
                if relacion_area < umbral_relacion_area:
                    if len(blob_actuales) > 1:
                        x_act, y_act, w_act, h_act = blob_actuales[-2][\
2]
                        relacion_area = (w * h) / ( float(w_act) * h_ac\
t)
                        if relacion_area > 1:
                            relacion_area = 1 / relacion_area
                        if relacion_area < umbral_relacion_area: contin\
ue
            else:
                if relacion_area < umbral_relacion_area * 1.17:
                    if len(blob_actuales) > 1:
                        x_act, y_act, w_act, h_act = blob_actuales[-2][\
2]
                        relacion_area = (w * h) / ( float(w_act) * h_ac\
t)
                        if relacion_area > 1:
                            relacion_area = 1 / relacion_area
                        if relacion_area < umbral_relacion_area * 1.17:\
 continue
            copiar_borrar_actuales.append([i,i_actuales])
            break
    borrar_de_coord = []
    for i, i_chequeo in copiar_borrar_chequeo:
        flag_chequeo = vchequeo[i_chequeo][-1][3].replace(',coord_aprox\
```

```
', '').replace(',coord_exac', '')
        vchequeo[i_chequeo].append( [cont_frame, coord_blob_frame[i][1]\
, vchequeo[i_chequeo][-1][2], flag_chequeo + ',coord_exac' ] )
        borrar_de_coord.append(i)
    for i, i_actuales in copiar_borrar_actuales:
        flag_actuales = vactuales[i_actuales][-1][4].replace(',coord_ap\
rox', '').replace(',coord_exac', '')
        vactuales[i_actuales].append( [cont_frame, vactuales[i_actuales\
][-1][1], coord_blob_frame[i][1], vactuales[i_actuales][-1][3],
                                       flag_actuales + ',coord_exac' ]\
 )
        borrar_de_coord.append(i)
    for i_coord in sorted(list(set(borrar_de_coord)), reverse=True):
        coord_blob_frame.pop(i_coord)
    return coord_blob_frame, vchequeo, vactuales
def iden_vehiculos_convergen( cont_frame, coord_blob_frame, vchequeo, v\
actuales, n_vehiculo ):
    coincidencias_chequeo = []
    coincidencias_actuales = []
    borrar_de_chequeo = []
    borrar_de_coord_blob_frame = []
    for i_coord, i_rect in enumerate(coord_blob_frame):
        i, rect = i_rect
        x, y, w, h = rect
        for i_chequeo, blob_chequeo in enumerate(vchequeo):
            f_chequeo, rect_chequeo, cant_v, flag = blob_chequeo[-1]
            if (not f_chequeo == (cont_frame - 1)) or rect_chequeo is N\
one: continue
            x_chequeo, y_chequeo, w_chequeo, h_chequeo = rect_chequeo
            cx = (x_chequeo + w_chequeo/2) - x
            cy = (y_chequeo + h_chequeo/2) - y
            if cx > w or cx < 0 or cy > h or cy < 0: continue
            coincidencias_chequeo.append([i_coord, i_chequeo])
        for i_actuales, blob_actuales in enumerate(vactuales):
            f_actuales, n_actuales, rect_actuales, cant_v_actuales, fla\
g = blob_actuales[-1]
            if (not f_actuales == (cont_frame - 1)) or rect_actuales is\
 None: continue
            x_actuales, y_actuales, w_actuales, h_actuales = rect_actua\
les
```

```
            cx = (x_actuales + w_actuales/2) - x
            cy = (y_actuales + h_actuales/2) - y
            if cx > w or cx < 0 or cy > h or cy < 0: continue
            coincidencias_actuales.append([i_coord, i_actuales])
    list_rect_convergen = [[k] for k in range(len(coord_blob_frame))]
    for i_coord, i_chequeo in coincidencias_chequeo:
       list_rect_convergen[i_coord].append([i_chequeo, 'chequeo'])
    for i_coord, i_actuales in coincidencias_actuales:
       list_rect_convergen[i_coord].append([i_actuales, 'actuales'])
    list_rect_convergen = [ i_list for i_list in list_rect_convergen if\
 len(i_list) >= 3]
    for i_list in list_rect_convergen:
        i_coord = i_list[0]
        borrar_de_coord_blob_frame.append(i_coord)
        i_conv = i_list[1:]
        if not str(i_conv).find('chequeo') == -1:
            for i_chequeo in i_conv:
                if i_chequeo[1] == 'chequeo':
                    borrar_de_chequeo.append(i_chequeo[0])
                    vchequeo_i_copia = vchequeo[i_chequeo[0]]
                    len_vchequeo_i_copia = len(vchequeo_i_copia)
                    for i_cheq in range( len_vchequeo_i_copia ):
                        cant_v_chequeo = vchequeo_i_copia[i_cheq][2]
                        vchequeo_i_copia[i_cheq].insert(1, range( n_veh\
iculo + 1 , n_vehiculo + cant_v_chequeo + 1 ))
                    n_vehiculo += cant_v_chequeo
                    vactuales.append( vchequeo_i_copia )
                    flag_chequeo = vchequeo[i_chequeo[0]][-1][4]
                    vactuales[-1][-1][4] = flag_chequeo + ',chequeo_con\
v'
        i_conv = [m[0] for m in i_conv if m[1] == 'actuales'] + range( \
len(vactuales) - str(i_conv).count('chequeo') , len(vactuales) )
        cant_v_sum = 0
        n_conv_min = [n_vehiculo + 1, 0]
        n_conv     = []
        for i_actuales in i_conv:
            cant_v_sum  += vactuales[i_actuales][-1][3]
            n_conv      += vactuales[i_actuales][-1][1]
            if min(vactuales[i_actuales][-1][1]) < n_conv_min[0]:
                n_conv_min[0] = min(vactuales[i_actuales][-1][1])
```

```
                n_conv_min[1] = i_actuales
        flag_conv_de = ',conv_de=' + str( i_conv ).replace(',' , '+').r\
eplace('[' , '(').replace(']' , ')')
        flag_conv_a  = ',conv_a=(' + str(n_conv_min[1]) + ')'
        for i_actuales in i_conv:
            if n_conv_min[1] == i_actuales:
                flag_actuales = vactuales[i_actuales][-1][4].replace(',\
coord_aprox', '')
                vactuales.append([ [cont_frame , n_conv , coord_blob_fr\
ame[i_coord][1] , cant_v_sum , flag_actuales + flag_conv_de] ])
                vactuales[i_actuales].append( [cont_frame , vactuales[i\
_actuales][-1][1] , None , 0 , flag_actuales + flag_conv_a] )
                continue
            flag_actuales = vactuales[i_actuales][-1][4].replace(',coor\
d_aprox', '')
            vactuales[i_actuales].append( [cont_frame , vactuales[i_act\
uales][-1][1] , None , 0 , flag_actuales + flag_conv_a] )
    for i_chequeo in sorted(list(set(borrar_de_chequeo)), reverse=True)\
:
        vchequeo.pop(i_chequeo)
    for i_coord in sorted(list(set(borrar_de_coord_blob_frame)), revers\
e=True):
        coord_blob_frame.pop(i_coord)
    return coord_blob_frame, vchequeo, vactuales, n_vehiculo
def iden_vehiculos_divergen( cont_frame, coord_blob_frame, vchequeo, va\
ctuales, n_vehiculo ):
    list_rect_divergen = [[k] for k in range(len(vchequeo))]
    for i_chequeo, blob_chequeo in enumerate(vchequeo):
        f_chequeo, rect_chequeo, cant_v_chequeo, flag_chequeo = blob_ch\
equeo[-1]
        if (not f_chequeo == (cont_frame - 1)) or rect_chequeo is None:\
 continue
        x_chequeo, y_chequeo, w_chequeo, h_chequeo = rect_chequeo
        for i_coord, i_rect in enumerate(coord_blob_frame):
            rect = i_rect[1]
            x, y, w, h = rect
            cx = (x + w/2) - x_chequeo
            cy = (y + h/2) - y_chequeo
            if cx > w_chequeo or cx < 0 or cy > h_chequeo or cy < 0: co\
ntinue
```

```
            list_rect_divergen[i_chequeo].append(i_coord)
    list_rect_divergen = [ i_list for i_list in list_rect_divergen if l\
en(i_list) >= 3]
    for i_list in list_rect_divergen:
        i_chequeo = i_list[0]
        i_div = i_list[1:]
        cant_v_chequeo = vchequeo[i_chequeo][-1][2]
        if cant_v_chequeo == len(i_div):
            for i_coord in i_div:
                vchequeo.append([ [ cont_frame, coord_blob_frame[i_coor\
d][1], 1, 'div_de' ] ])
        elif cant_v_chequeo > len(i_div):
            cant_v_areas = dict()
            for i_coord in i_div:
                rect = coord_blob_frame[i_coord][1]
                x, y, w, h = rect
                cant_v_areas[ str(w*h) + '-' + str(i_coord) ] = cant_v_\
areas.get( str(w*h) + '-' + str(i_coord) , 0 )
            cont_v_area = cant_v_chequeo
            bool_area = True
            while bool_area:
                for area in sorted(cant_v_areas, reverse=True):
                    cant_v_areas[ area ] = cant_v_areas.get( area , 0 )\
 + 1
                    cont_v_area -= 1
                    if cont_v_area == 0:
                        bool_area = False
                        break
            for i_coord in i_div:
                rect = coord_blob_frame[i_coord][1]
                x, y, w, h = rect
                vchequeo.append([ [ cont_frame, coord_blob_frame[i_coor\
d][1], cant_v_areas[ str(w*h) + '-' + str(i_coord) ] , 'div_de' ] ])
        else:
            for i_coord in i_div:
                vchequeo.append([ [ cont_frame, coord_blob_frame[i_coor\
d][1], 1, 'div_de' ] ])
    borrar_de_coord = []
    for i_list in list_rect_divergen:
        i_chequeo = i_list[0]
```

```
        i_div = i_list[1:]
        flag_chequeo = vchequeo[i_chequeo][-1][3].replace(',coord_aprox\
', '').replace(',div_a', '')
        vchequeo[i_chequeo].append([ cont_frame, None, 0, flag_chequeo \
+ ',div_a' ])
        for i_coord in i_div:
            borrar_de_coord.append(i_coord)
    for i_coord in sorted(list(set(borrar_de_coord)), reverse=True):
        coord_blob_frame.pop(i_coord)
    list_rect_divergen = [[k] for k in range(len(vactuales))]
    for i_actuales, blob_actuales in enumerate(vactuales):
        f_actuales, n_actuales, rect_actuales, cant_v_actuales, flag_ac\
tuales = blob_actuales[-1]
        if (not f_actuales == (cont_frame - 1)) or rect_actuales is Non\
e: continue
        x_actuales, y_actuales, w_actuales, h_actuales = rect_actuales
        for i_coord, i_rect in enumerate(coord_blob_frame):
            rect = i_rect[1]
            x, y, w, h = rect
            cx = (x + w/2) - x_actuales
            cy = (y + h/2) - y_actuales
            if cx > w_actuales or cx < 0 or cy > h_actuales or cy < 0: \
continue
            list_rect_divergen[i_actuales].append(i_coord)
    list_rect_divergen = [ i_list for i_list in list_rect_divergen if l\
en(i_list) >= 3]
    for i_list in list_rect_divergen:
        i_actuales = i_list[0]
        i_div = i_list[1:]
        cant_v_actuales = vactuales[i_actuales][-1][3]
        n_actuales = vactuales[i_actuales][-1][1]
        n_actuales_cont = 0
        if cant_v_actuales == len(i_div):
            for i_coord in i_div:
                vactuales.append([ [ cont_frame, n_actuales[n_actuales_\
cont : n_actuales_cont + 1 ]  , coord_blob_frame[i_coord][1], 1, 'div_d\
e' ] ])
                n_actuales_cont += 1
        elif cant_v_actuales > len(i_div):
            cant_v_areas = dict()
```

```
            for i_coord in i_div:
                rect = coord_blob_frame[i_coord][1]
                x, y, w, h = rect
                cant_v_areas[ str(w*h) + '-' + str(i_coord) ] = cant_v_\
areas.get( str(w*h) + '-' + str(i_coord) , 0 )
            cont_v_area = cant_v_actuales
            bool_area = True
            while bool_area:
                for area in sorted(cant_v_areas, reverse=True):
                    cant_v_areas[ area ] = cant_v_areas.get( area , 0 )\
 + 1
                    cont_v_area -= 1
                    if cont_v_area == 0:
                        bool_area = False
                        break
            for i_coord in i_div:
                rect = coord_blob_frame[i_coord][1]
                x, y, w, h = rect
                cant_v_area = cant_v_areas[ str(w*h) + '-' + str(i_coor\
d) ]
                vactuales.append([ [ cont_frame, n_actuales[n_actuales_\
cont : n_actuales_cont + cant_v_area ] , coord_blob_frame[i_coord][1],
                                    cant_v_area, 'div_de' ] ])
                n_actuales_cont += cant_v_area
        else:
            n_actuales = n_actuales + range( n_vehiculo, n_vehiculo + l\
en(i_div) )
            n_vehiculo += len(i_div)
            for i_coord in i_div:
                vactuales.append([ [ cont_frame, n_actuales[n_actuales_\
cont : n_actuales_cont + 1 ] , coord_blob_frame[i_coord][1], 1, 'div_de\
' ] ])
                n_actuales_cont += 1
    borrar_de_coord = []
    for i_list in list_rect_divergen:
        i_actuales = i_list[0]
        i_div = i_list[1:]
        flag_actuales = vactuales[i_actuales][-1][4].replace(',coord_ap\
rox', '').replace(',div_a', '')
        vactuales[i_actuales].append( [ cont_frame, vactuales[i_actuale\
```

```
s][-1][1], None, 0, flag_actuales + ',div_a' ] )
        for i_coord in i_div:
            borrar_de_coord.append(i_coord)
    for i_coord in sorted(list(set(borrar_de_coord)), reverse=True):
        coord_blob_frame.pop(i_coord)
    return coord_blob_frame, vchequeo, vactuales, n_vehiculo
def iden_comb_complejas( cont_frame, coord_blob_frame, vchequeo, vactua\
les, n_vehiculo ):
    list_rect_coinciden = [[k] for k in range(len(coord_blob_frame))]
    m = .2
    for i_coord, i_rect in enumerate(coord_blob_frame):
        rect = i_rect[1]
        x, y, w, h = rect
        for i_chequeo, blob_chequeo in enumerate(vchequeo):
            f_chequeo, rect_chequeo, cant_v_chequeo, flag_chequeo = blo\
b_chequeo[-1]
            if (not f_chequeo == (cont_frame - 1)) or rect_chequeo is N\
one: continue
            x_chequeo, y_chequeo, w_chequeo, h_chequeo = rect_chequeo
            cx = (x_chequeo + w_chequeo/2) - (x - w * m)
            cy = (y_chequeo + h_chequeo/2) - (y - h * m)
            if cx > w * (1 + 2 * m) or cx < 0 or cy > h * (1 + 2 * m) o\
r cy < 0: continue
            list_rect_coinciden[i_coord].append( [i_chequeo, 'chequeo']\
 )
        for i_actuales, blob_actuales in enumerate(vactuales):
            f_actuales, n_actuales, rect_actuales, cant_v_actuales, fla\
g_actuales = blob_actuales[-1]
            if (not f_actuales == (cont_frame - 1)) or rect_actuales is\
 None: continue
            x_actuales, y_actuales, w_actuales, h_actuales = rect_actua\
les
            cx = (x_actuales + w_actuales/2) - (x - w * m)
            cy = (y_actuales + h_actuales/2) - (y - h * m)
            if cx > w * (1 + 2 * m) or cx < 0 or cy > h * (1 + 2 * m) o\
r cy < 0: continue
            list_rect_coinciden[i_coord].append([i_actuales, 'actuales'\
] )
    list_rect_coinciden = [ i_list for i_list in list_rect_coinciden if\
 len(i_list) >= 3]
```

```
    list_rect2_coinciden = [[k] for k in range(len(coord_blob_frame))]
    for i_list in list_rect_coinciden:
        i_coord = i_list[0]
        i_comp = i_list[1:]
        for i_cheact in i_comp:
            if i_cheact[1] == 'chequeo':
                f_chequeo, rect_cheact, cant_v_chequeo, flag_chequeo = \
vchequeo[i_cheact[0]][-1]
            if i_cheact[1] == 'actuales':
                f_actuales, n_actuales, rect_cheact, cant_v_actuales, f\
lag_actuales = vactuales[i_cheact[0]][-1]
            x_cheact, y_cheact, w_cheact, h_cheact = rect_cheact
            for i_coord2, i_rect2 in enumerate(coord_blob_frame):
                if i_coord == i_coord2: continue
                rect2 = i_rect2[1]
                x2, y2, w2, h2 = rect2
                cx =(x2 + w2/2) - x_cheact
                cy =(y2 + h2/2) - y_cheact
                if cx > w_cheact or cx < 0 or cy > h_cheact or cy < 0: \
continue
                list_rect2_coinciden[i_coord].append( [i_cheact[0], i_c\
heact[1], i_coord2 ] )
        if len(list_rect2_coinciden[i_coord]) < 2: continue
        i_cheact = list_rect2_coinciden[i_coord][1][:2]
        list_rect2_coinciden[i_coord].append( [i_cheact[0], i_cheact[1]\
, i_coord ] )
        i_div = list_rect2_coinciden[i_coord][1:]
        if i_cheact[1] == 'chequeo':
            cant_v_cheact = vchequeo[ i_cheact[0] ][-1][2]
            n_actuales    = range( n_vehiculo , n_vehiculo + cant_v_che\
act )
            n_vehiculo    += cant_v_cheact
        if i_cheact[1] == 'actuales':
            cant_v_cheact = vactuales[ i_cheact[0] ][-1][3]
            n_actuales    = vactuales[ i_cheact[0] ][-1][1]
        n_actuales_cont = 0
        if cant_v_cheact == len(i_div):
            for i_coord2 in i_div:
                vactuales.append([ [ cont_frame, n_actuales[ n_actuales\
_cont : n_actuales_cont + 1 ] , coord_blob_frame[ i_coord2[2] ][1], 1, \
```

```
'div_de' ] ])
                n_actuales_cont += 1
        elif cant_v_cheact > len(i_div):
            cant_v_areas = dict()
            for i_coord2 in i_div:
                rect = coord_blob_frame[ i_coord2[2] ][1]
                x, y, w, h = rect
                cant_v_areas[ str(w*h) + '-' + str(i_coord2) ] = cant_v\
_areas.get( str(w*h) + '-' + str(i_coord2) , 0 )
            cont_v_area = cant_v_cheact
            bool_area = True
            while bool_area:
                for area in sorted(cant_v_areas, reverse=True):
                    cant_v_areas[ area ] = cant_v_areas.get( area , 0 )\
 + 1
                    cont_v_area -= 1
                    if cont_v_area == 0:
                        bool_area = False
                        break
            for i_coord2 in i_div:
                rect = coord_blob_frame[ i_coord2[2] ][1]
                x, y, w, h = rect
                cant_v_area = cant_v_areas[ str(w*h) + '-' + str(i_coor\
d2) ]
                flag_actuales = vactuales[-1][-1][4].replace(',coord_ap\
rox', '').replace(',div_de_complx', '')
                vactuales.append([ [ cont_frame, n_actuales[n_actuales_\
cont : n_actuales_cont + cant_v_area ] , coord_blob_frame[ i_coord2[2] \
][1],
                                   cant_v_areas[ str(w*h) + '-' + str(i_co\
ord2) ] , flag_actuales + ',div_de_complx' ] ])
                n_actuales_cont += cant_v_area
        else:
            n_actuales = n_actuales + range( n_vehiculo, n_vehiculo + l\
en(i_div) - cant_v_cheact )
            n_vehiculo += len(i_div) - cant_v_cheact
            for i_coord2 in i_div:
                vactuales.append([ [ cont_frame, n_actuales[n_actuales_\
cont : n_actuales_cont + 1 ] , coord_blob_frame[ i_coord2[2] ][1], 1, '\
div_de' ] ])
```

```
                n_actuales_cont += 1
        i_coord = i_list[0]
        i_comp = i_list[1:]
        for i_cheact in i_comp:
            if list_rect2_coinciden[i_coord][-1][0] == i_cheact[0]: con\
tinue
            if i_cheact[1] == 'chequeo':
                cant_v_cheact = vchequeo[ i_cheact[0] ][-1][2]
                n_actuales    = range( n_vehiculo , n_vehiculo + cant_v\
_cheact )
                n_vehiculo    += cant_v_cheact
            if i_cheact[1] == 'actuales':
                cant_v_cheact = vactuales[ i_cheact[0] ][-1][3]
                n_actuales    = vactuales[ i_cheact[0] ][-1][1]
            flag_actuales = vactuales[-1][-1][4].replace(',coord_aprox'\
, '').replace(',conv_de_complx', '')
            vactuales[-1][-1] = [ vactuales[-1][-1][0], vactuales[-1][-\
1][1] + n_actuales , vactuales[-1][-1][2],
                                  vactuales[-1][-1][3] + cant_v_cheact , \
flag_actuales + ',conv_de_complx' ]
    borrar_de_coord = []
    borrar_de_chequeo = []
    actualizar_vactuales = []
    for i_list in list_rect_coinciden:
        for i_cheact in i_list[1:]:
            if i_cheact[1] == 'actuales':
                actualizar_vactuales.append(i_cheact[0] )
    for i_actuales in actualizar_vactuales:
        vactuales[i_actuales].append( [cont_frame , vactuales[i_actuale\
s][-1][1] , None , 0 , vactuales[i_actuales][-1][4] + ',comb_complx'] )\

    for i_list in list_rect2_coinciden:
        for i_borrar in i_list[1:]:
            if not i_borrar[2] in borrar_de_coord:
                borrar_de_coord.append(i_borrar[2])
            if i_borrar[1] == 'chequeo':
                borrar_de_chequeo.append(i_borrar[0])
    for i_coord in sorted(list(set(borrar_de_coord)), reverse=True):
        coord_blob_frame.pop(i_coord)
    for i_chequeo in sorted(list(set(borrar_de_chequeo)), reverse=True)\
```

```
:
            vchequeo.pop(i_chequeo)
        return coord_blob_frame, vchequeo, vactuales, n_vehiculo
def iden_vehiculos_convergen_borde( cont_frame, coord_blob_frame, vcheq\
ueo, vactuales, n_vehiculo, B_entra, C_entra, umbral_entra, width, heig\
ht ):
    borrar_de_coord_blob_frame = []
    for i_coord, i_rect in enumerate(coord_blob_frame):
        i, rect = i_rect
        bool_skip = False
        x, y, w, h = rect
        for i_chequeo, blob_chequeo in enumerate(vchequeo):
            bool_cerca_entra = False
            f_chequeo, rect_chequeo, cant_v, flag = blob_chequeo[-1]
            if (not f_chequeo == (cont_frame - 1)) or rect_chequeo is N\
one: continue
            x_chequeo, y_chequeo, w_chequeo, h_chequeo = rect_chequeo
            cx = (x_chequeo + w_chequeo/2) - x
            cy = (y_chequeo + h_chequeo/2) - y
            if cx > w or cx < 0 or cy > h or cy < 0: continue
            esquinas = [ (x_chequeo, y_chequeo),
                         (x_chequeo + w_chequeo, y_chequeo),
                         (x_chequeo, y_chequeo + h_chequeo),
                         (x_chequeo + w_chequeo, y_chequeo + h_chequeo)\
 ]
            for px, py in esquinas:
                dis_entra   = modulo1.dis_punto_borde(px, py, B_entra, \
C_entra)
                if dis_entra < umbral_entra or px < umbral_entra/3 or p\
y < umbral_entra/3 or px > width - umbral_entra/3 or py > height - umbr\
al_entra/3:
                    bool_cerca_entra = True
                    break
            borrar_de_coord_blob_frame.append(i_coord)
            bool_skip = True
            relacion_m = 2
            if len(blob_chequeo) > 1:
                for cont_blob_che, blob_che in enumerate(blob_chequeo[:\
:-1]):
                    f_che, rect_che, cant_v_che, flag_che = blob_che
```

```
                    if (flag_che.find('conv') == -1) or (flag_che.find(\
'div')): None
                    else: break
                    if cont_blob_che == 5: break
                x_che, y_che, w_che, h_che = rect_che
                try:
                    m0 = float(((y_chequeo + h_chequeo/2)) - (float(y +\
 h/2))) / ((x_chequeo + w_chequeo/2) - (x + w/2))
                except:
                    m0 = float(((y_chequeo + h_chequeo/2)) - (float(y +\
 h/2))) / ((x_chequeo + w_chequeo/2) - (x + w/2) + 1)
                try:
                    m1 = float(((y_che + h_che/2)) - ((y + h/2))) / (fl\
oat(x_che + w_che/2) - (x + w/2))
                except:
                    m1 = float(((y_che + h_che/2)) - ((y + h/2))) / (fl\
oat(x_che + w_che/2) - (x + w/2) + 1)
                try:
                    relacion_m = abs(m0/m1)
                    if relacion_m > 1: relacion_m = 1 / relacion_m
                except:
                    None
            if (bool_cerca_entra) and (relacion_m < 0.8):
                flag_chequeo = vchequeo[i_chequeo][-1][3].replace(',coo\
rd_aprox', '').replace(',conv_de_borde', '')
                vchequeo[i_chequeo].append( [cont_frame , rect , vchequ\
eo[i_chequeo][-1][2] + 1 , flag_chequeo + ',conv_de_borde'] )
            else:
                flag_chequeo = vchequeo[i_chequeo][-1][3].replace(',coo\
rd_aprox', '').replace(',conv_de_borde', '')
                vchequeo[i_chequeo].append( [cont_frame , rect , vchequ\
eo[i_chequeo][-1][2] , flag_chequeo + ',conv_de_borde'] )
            break
        if bool_skip: continue
        for i_actuales, blob_actuales in enumerate(vactuales):
            bool_cerca_entra = False
            f_actuales, n_actuales, rect_actuales, cant_v_actuales, fla\
g = blob_actuales[-1]
            if (not f_actuales == (cont_frame - 1)) or rect_actuales is\
 None: continue
```

```
            x_actuales, y_actuales, w_actuales, h_actuales = rect_actua\
les
            cx = (x_actuales + w_actuales/2) - x
            cy = (y_actuales + h_actuales/2) - y
            if cx > w or cx < 0 or cy > h or cy < 0: continue
            borrar_de_coord_blob_frame.append(i_coord)
            esquinas = [ (x_actuales, y_actuales),
                         (x_actuales + w_actuales, y_actuales),
                         (x_actuales, y_actuales + h_actuales),
                         (x_actuales + w_actuales, y_actuales + h_actua\
les) ]
            for px, py in esquinas:
                dis_entra   = modulo1.dis_punto_borde(px, py, B_entra, \
C_entra)
                if dis_entra < umbral_entra or px < umbral_entra/3 or p\
y < umbral_entra/3 or px > width - umbral_entra/3 or py > height - umbr\
al_entra/3:
                    bool_cerca_entra = True
                    break
            relacion_m = 2
            if len(blob_actuales) > 1:
                for cont_blob_act, blob_act in enumerate(blob_actuales[\
::-1]):
                    f_act, n_act, rect_act, cant_v_act, flag_act = blob\
_act
                    if (flag_act.find('conv') == -1) or (flag_act.find(\
'div')): None
                    else: break
                    if cont_blob_act == 5: break
                x_act, y_act, w_act, h_act = rect_act
                try:
                    m0 = float(((y_actuales + h_actuales/2)) - (float(y\
 + h/2))) / ((x_actuales + w_actuales/2) - (x + w/2))
                    m1 = float(((y_act + h_act/2)) - ((y + h/2))) / (fl\
oat(x_act + w_act/2) - (x + w/2))
                except:
                    try:
                        m0 = ((x_actuales + w_actuales/2) - (x + w/2) /\
 float(((y_actuales + h_actuales/2)) - (float(y + h/2))))
                        m1 = (float(x_act + w_act/2) - (x + w/2)) / flo\
```

```
at(((y_act + h_act/2)) - ((y + h/2)))
                    except:
                        break
                try:
                    relacion_m = abs(m0/m1)
                    if relacion_m > 1: relacion_m = 1 / relacion_m
                except:
                    break
            if (bool_cerca_entra) and (relacion_m < 0.8):
                n_vehiculo += 1
                flag_actuales = vactuales[i_actuales][-1][4].replace(',\
coord_aprox', '').replace(',conv_de_borde', '')
                vactuales[i_actuales].append( [cont_frame , vactuales[i\
_actuales][-1][1] + [n_vehiculo] , rect , vactuales[i_actuales][-1][3] \
+ 1 ,
                                               flag_actuales + ',conv_de_b\
orde'] )
            else:
                flag_actuales = vactuales[i_actuales][-1][4].replace(',\
coord_aprox', '').replace(',conv_de_borde', '')
                vactuales[i_actuales].append( [cont_frame , vactuales[i\
_actuales][-1][1] , rect , vactuales[i_actuales][-1][3] ,
                                               flag_actuales + ',conv_de_b\
orde'] )
            break
    for i_coord in sorted(list(set(borrar_de_coord_blob_frame)), revers\
e=True):
        coord_blob_frame.pop(i_coord)
    return coord_blob_frame, vchequeo, vactuales, n_vehiculo
def iden_vehiculos_divergen_borde( cont_frame, coord_blob_frame, vchequ\
eo, vactuales, n_vehiculo,\
                                   B_frente, C_frente, B_izq, C_izq, u\
mbral_frente, umbral_izq):
    list_rect_divergen = [[k] for k in range(len(vactuales))]
    borrar_de_coord = []
    for i_actuales, blob_actuales in enumerate(vactuales):
        f_actuales, n_actuales, rect_actuales, cant_v_actuales, flag_ac\
tuales = blob_actuales[-1]
        if (not f_actuales == (cont_frame - 1)) or rect_actuales is Non\
e: continue
```

```
        x_actuales, y_actuales, w_actuales, h_actuales = rect_actuales
        for i_coord, i_rect in enumerate(coord_blob_frame):
            rect = i_rect[1]
            x, y, w, h = rect
            cx = (x + w/2) - x_actuales
            cy = (y + h/2) - y_actuales
            if cx > w_actuales or cx < 0 or cy > h_actuales or cy < 0: \
continue
            dis_frente = modulo1.dis_punto_borde((x_actuales + w_actual\
es/2), (y_actuales + h_actuales/2), B_frente, C_frente)
            dis_izq = modulo1.dis_punto_borde((x_actuales + w_actuales/\
2), (y_actuales + h_actuales/2), B_izq, C_izq)
            borrar_de_coord.append(i_coord)
            if dis_frente < umbral_frente*2 or dis_izq < umbral_izq*2:
                list_rect_divergen[i_actuales].append(i_coord)
            else:
                flag_actuales = vactuales[i_actuales][-1][4].replace(',\
coord_aprox', '').replace(',div_de_borde', '')
                vactuales[i_actuales].append( [cont_frame , vactuales[i\
_actuales][-1][1] , rect , vactuales[i_actuales][-1][3] ,
                                               flag_actuales + ',div_de_bo\
rde'] )
            break
    list_rect_divergen = [ i_list for i_list in list_rect_divergen if l\
en(i_list) >= 2]
    for i_list in list_rect_divergen:
        i_actuales = i_list[0]
        i_coord = i_list[1]
        cant_v_actuales = vactuales[i_actuales][-1][3]
        cant_v_actuales1 = int(cant_v_actuales/2.0)
        cant_v_actuales2 = math.ceil(cant_v_actuales/2.0)
        n_actuales = vactuales[i_actuales][-1][1]
        rect = coord_blob_frame[i_coord][1]
        if cant_v_actuales == 2:
            vactuales.append([ [ cont_frame, [n_actuales[0]] , rect, 1,\
 'div_de' ] ])
            flag_actuales = vactuales[i_actuales][-1][4].replace(',coor\
d_aprox', '').replace(',div_de', '')
            vactuales[i_actuales].append([ cont_frame-1, [n_actuales[1]\
] , vactuales[i_actuales][-1][2], 1, flag_actuales + ',div_de' ])
```

```
        elif cant_v_actuales > 2:
            vactuales.append([ [ cont_frame, n_actuales[0: cant_v_actua\
les1] , rect, cant_v_actuales1, 'div_de' ] ])
            flag_actuales = vactuales[i_actuales][-1][4].replace(',coor\
d_aprox', '').replace(',div_de', '')
            vactuales[i_actuales].append([ cont_frame-1, n_actuales[can\
t_v_actuales1:] , vactuales[i_actuales][-1][2], cant_v_actuales2,
                                           flag_actuales + ',div_de' ]\
)
        else:
            n_vehiculo += 1
            vactuales.append([ [ cont_frame, [n_vehiculo] , rect, 1, 'd\
iv_de' ] ])
            flag_actuales = vactuales[i_actuales][-1][4].replace(',coor\
d_aprox', '').replace(',div_de', '')
            vactuales[i_actuales].append([ cont_frame-1, n_actuales , v\
actuales[i_actuales][-1][2], 1, flag_actuales + ',div_de' ])
    for i_coord in sorted(list(set(borrar_de_coord)), reverse=True):
        coord_blob_frame.pop(i_coord)
    return coord_blob_frame, vchequeo, vactuales, n_vehiculo
def agrega_coord_aprox( cont_frame, coord_blob_frame, vchequeo, vactual\
es, B_entra, C_entra, B_frente, C_frente, B_izq, C_izq, \
                        B_der, C_der, umbral_entra, umbral_frente, umbr\
al_izq, umbral_der, umbral_blob_aprox, \
                        width, height, dis_fueradeimagen, inters_mask, \
img_fast):
    list_rect_aprox = []
    for i_chequeo, blob_chequeo in enumerate(vchequeo):
        f_chequeo, rect_chequeo, cant_v_chequeo, flag_chequeo = blob_ch\
equeo[-1]
        if (not f_chequeo == (cont_frame - 1)) or rect_chequeo is None:\
 continue
        x_chequeo, y_chequeo, w_chequeo, h_chequeo = rect_chequeo
        esquinas = [ (x_chequeo, y_chequeo), (x_chequeo + w_chequeo, y_\
chequeo),
                     (x_chequeo, y_chequeo + h_chequeo), (x_chequeo + w\
_chequeo, y_chequeo + h_chequeo) ]
        for px, py in esquinas:
            cv2.circle(img_fast, (px, py), 3, (255, 0, 255), -1)
            dis_entra   = modulo1.dis_punto_borde(px, py, B_entra, C_en\
```

```
tra)
            dis_frente  = modulo1.dis_punto_borde(px, py, B_frente, C_f\
rente)
            dis_izq     = modulo1.dis_punto_borde(px, py, B_izq, C_izq)\

            dis_der     = modulo1.dis_punto_borde(px, py, B_der, C_der)\

            if dis_entra < umbral_entra or dis_frente < umbral_frente o\
r dis_izq < umbral_izq or dis_der < umbral_der:
                list_rect_aprox.append(i_chequeo)
                break
    for i_chequeo in list_rect_aprox:
        cant_pos = len(vchequeo[i_chequeo])
        cant_v_chequeo = vchequeo[i_chequeo][-1][2]
        x_chequeo, y_chequeo, w_chequeo, h_chequeo = vchequeo[i_chequeo\
][-1][1]
        flag_chequeo = vchequeo[i_chequeo][-1][3]
        mx      = 0
        my      = 0
        b       = 0
        delta_x = 0
        delta_y = 0
        if cant_pos <= 2:
            None
        else:
            j = min(10, cant_pos)
            bool_skip = False
            bool_mx = True
            bool_my = True
            for i_cheq in range(2, j + 1):
                x_chequeo1, y_chequeo1, w_chequeo1, h_chequeo1 = vchequ\
eo[i_chequeo][-i_cheq + 1][1]
                x_chequeo2, y_chequeo2, w_chequeo2, h_chequeo2 = vchequ\
eo[i_chequeo][-i_cheq][1]
                p1 = ( (x_chequeo1 + w_chequeo1/2.0) , (y_chequeo1 + h_\
chequeo1/2.0) )
                p2 = ( (x_chequeo2 + w_chequeo2/2.0) , (y_chequeo2 + h_\
chequeo2/2.0) )
                if i_cheq == 2:
                    p_ultima = p1
```

```
                try:
                    mx += float(p1[1] - p2[1]) / (p1[0] - p2[0])
                except:
                    bool_mx = False
                try:
                    my += float(p1[0] - p2[0]) / (p1[1] - p2[1])
                except:
                    bool_my = False
                delta_x += p1[0] - p2[0]
                delta_y += p1[1] - p2[1]
            mx       = mx / (j - 1)
            my       = my / (j - 1)
            delta_x = delta_x / (j - 1)
            delta_y = delta_y / (j - 1)
            if bool_mx:
                b = p_ultima[1] - mx * p_ultima[0]
                if abs(delta_x) >= abs(delta_y):
                    x_aprox = int(p_ultima[0] + delta_x)
                    y_aprox = numpy.int16(mx * x_aprox + b)
                else:
                    y_aprox = int(p_ultima[1] + delta_y)
                    x_aprox = numpy.int16((y_aprox - b) / mx)
            elif bool_my:
                b = p_ultima[0] - my * p_ultima[1]
                if abs(delta_x) >= abs(delta_y):
                    x_aprox = int(p_ultima[0] + delta_x)
                    y_aprox = numpy.int16((x_aprox - b) / my)
                else:
                    y_aprox = int(p_ultima[1] + delta_y)
                    x_aprox = numpy.int16(my * y_aprox + b)
            else:
                continue
            if x_aprox < 0 or x_aprox > width or y_aprox < 0 or y_aprox\
 > height:
                x_aprox = int(p_ultima[0] + delta_x)
                y_aprox = int(p_ultima[1] + delta_y)
                bool_aux = False
            if math.sqrt( ( x_aprox - p_ultima[0] ) ** 2 + ( y_aprox - \
p_ultima[1] ) ** 2 ) < 2: continue
            flag_chequeo = flag_chequeo.replace(',coord_aprox', '')
```

```
            vchequeo[i_chequeo].append([ cont_frame, [ x_aprox - w_cheq\
ueo/2 , y_aprox - h_chequeo/2 , w_chequeo, h_chequeo ], cant_v_chequeo,\

                                        flag_chequeo + ',coord_aprox' \
])
    list_rect_aprox = []
    list_bloque_borde = []
    for i_actuales, blob_actuales in enumerate(vactuales):
        f_actuales, n_actuales, rect_actuales, cant_v_actuales, flag_ac\
tuales = blob_actuales[-1]
        if (not f_actuales == (cont_frame - 1)) or rect_actuales is Non\
e: continue
        x_actuales, y_actuales, w_actuales, h_actuales = rect_actuales
        esquinas = [ (x_actuales, y_actuales), (x_actuales + w_actuales\
, y_actuales),
                     (x_actuales, y_actuales + h_actuales), (x_actuales\
 + w_actuales, y_actuales + h_actuales),
                     (x_actuales + w_actuales/2, y_actuales + h_actuale\
s/2) ]
        for px, py in esquinas:
            cv2.circle(img_fast, (px, py), 3, (255, 0, 255), -1)
            dis_entra   = modulo1.dis_punto_borde(px, py, B_entra, C_en\
tra)
            dis_frente  = modulo1.dis_punto_borde(px, py, B_frente, C_f\
rente)
            dis_izq     = modulo1.dis_punto_borde(px, py, B_izq, C_izq)\

            dis_der     = modulo1.dis_punto_borde(px, py, B_der, C_der)\

            if dis_entra < umbral_entra or dis_frente < umbral_frente o\
r dis_izq < umbral_izq or dis_der < umbral_der:
                list_rect_aprox.append(i_actuales)
                break
    for i_actuales in range(len(vactuales)):
        if i_actuales in list_rect_aprox: continue
        f_actuales, n_actuales, rect_actuales, cant_v_actuales, flag_ac\
tuales = vactuales[i_actuales][-1]
        if (not f_actuales == (cont_frame - 1)) or rect_actuales is Non\
e: continue
        x_actuales, y_actuales, w_actuales, h_actuales = rect_actuales
```

```
        esquinas = [ (x_actuales, y_actuales), (x_actuales + w_actuales\
, y_actuales + h_actuales) ]
        bool_skip = False
        for i_act in range(len(vactuales)):
            if bool_skip: break
            if i_actuales == i_act: continue
            f_act, n_act, rect_act, cant_v_act, flag_act = vactuales[i_\
act][-1]
            if (not f_act == (cont_frame - 1)) or rect_act is None: con\
tinue
            if flag_act.find('coord_aprox') == -1: continue
            x_act, y_act, w_act, h_act = rect_act
            lados_act = [ ((x_act           , y_act + h_act ), (x_act +\
 w_act   , y_act + h_act )),\
                          ((x_act           , y_act         ), (x_act +\
 w_act   , y_act         )),\
                          ((x_act + w_act   , y_act         ), (x_act +\
 w_act   , y_act + h_act )),\
                          ((x_act           , y_act         ), (x_act  \
        , y_act + h_act )) ]
            for p_lado1, p_lado2 in lados_act:
                if bool_skip: break
                B_act = - float( p_lado1[0] - p_lado2[0] ) / \
                             ( 1 + p_lado1[1] - p_lado2[1] )
                C_act = - (p_lado1[0] + p_lado1[1] * B_act)
                for px, py in esquinas:
                    cv2.circle(img_fast, (px, py), 3, (255, 0,0), -1)
                    dis_blob = modulo1.dis_punto_borde(px, py, B_act, C\
_act)
                    if dis_blob < umbral_blob_aprox:
                        list_rect_aprox.append(i_actuales)
                        list_bloque_borde.append(i_actuales)
                        bool_skip = True
                        break
        if (not i_actuales in list_rect_aprox) and flag_actuales.find('\
aprox_bloque_borde') > -1:
            list_rect_aprox.append(i_actuales)
    for i_actuales in list_rect_aprox:
        cant_pos = len(vactuales[i_actuales])
        f_actuales, n_actuales, rect_actuales, cant_v_actuales, flag_ac\
```

```
tuales = vactuales[i_actuales][-1]
        x_actuales, y_actuales, w_actuales, h_actuales = rect_actuales
        mx       = 0
        my       = 0
        b        = 0
        delta_x = 0
        delta_y = 0
        list_dmin_borde = [ dis_fueradeimagen , dis_fueradeimagen , dis\
_fueradeimagen ]
        if cant_pos <=2:
            None
            esquinas = [ (x_actuales                , y_actuales       \
        ),
                         (x_actuales + w_actuales   , y_actuales       \
        ),
                         (x_actuales                , y_actuales + h_ac\
tuales   ),
                         (x_actuales + w_actuales   , y_actuales + h_ac\
tuales   ) ]
            for px, py in esquinas:
                cv2.circle(img_fast, (px, py), 3, (255, 0, 255), -1)
                dis_frente  = modulo1.dis_punto_borde(px, py, B_frente,\
 C_frente)
                if dis_frente < umbral_frente:
                    if dis_frente < list_dmin_borde[0]:
                        list_dmin_borde[0] = dis_frente
                dis_izq     = modulo1.dis_punto_borde(px, py, B_izq, C_\
izq)
                if dis_izq < umbral_izq:
                    if dis_izq < list_dmin_borde[1]:
                        list_dmin_borde[1] = dis_izq
                dis_der     = modulo1.dis_punto_borde(px, py, B_der, C_\
der)
                if dis_der < umbral_der:
                    if dis_der < list_dmin_borde[2]:
                        list_dmin_borde[1] = dis_der
            dis_min = min(list_dmin_borde)
            if dis_min == dis_fueradeimagen: continue
            pos_min = [index for index,value in enumerate(list_dmin_bor\
de) if value == dis_min][0]
```

```
            if pos_min == 0:
                B_min = B_frente
                C_min = C_frente
            if pos_min == 1:
                B_min = B_izq
                C_min = C_izq
            if pos_min == 1:
                B_min = B_der
                C_min = C_der
            cx, cy = (x_actuales + w_actuales/2, y_actuales + h_actuale\
s/2)
            dis_min_bloque = modulo1.dis_punto_borde(x_actuales + w_act\
uales/2, y_actuales + h_actuales/2, B_frente, C_frente)
            a = B_min ** 2 + 1
            b = 2 * B_min * ( C_min + cx ) - 2 * cy
            y_inter = int(( - b ) / ( 2 * a ))
            x_inter = - int( B_min * y_inter + C_min )
            list_interacion = [ (x_inter, y_inter), (x_inter + 3, y_int\
er + 3), (x_inter + 3, y_inter - 3),
                                        (x_inter - 3, y_inter + 3), (x_in\
ter - 3, y_inter - 3) ]
            for x_aprox, y_aprox in list_interacion:
                if x_aprox >= width or x_aprox < 0 or y_aprox >= height\
 or y_aprox < 0: break
                if inters_mask[ y_aprox ][ x_aprox ] == 0:
                    break
            flag_actuales = flag_actuales.replace(',coord_aprox', '') +\
 ',coord_aprox'
            if i_actuales in list_bloque_borde: flag_actuales = flag_ac\
tuales.replace(',aprox_bloque_borde', '') + ',aprox_bloque_borde'
            vactuales[i_actuales].append([ cont_frame, n_actuales , [ x\
_aprox - w_actuales/2 , y_aprox - h_actuales/2 , w_actuales, h_actuales\
 ],
                                            cant_v_actuales, flag_actual\
es ])
        else:
            j = min(20, cant_pos)
            bool_mx = True
            bool_my = True
            for i_act in range(2, j + 1):
```

```
                x_actuales1, y_actuales1, w_actuales1, h_actuales1 = va\
ctuales[i_actuales][ -i_act + 1 ][2]
                x_actuales2, y_actuales2, w_actuales2, h_actuales2 = va\
ctuales[i_actuales][ -i_act ][2]
                p1 = ( (x_actuales1 + w_actuales1/2.0) , (y_actuales1 +\
 h_actuales1/2.0) )
                p2 = ( (x_actuales2 + w_actuales2/2.0) , (y_actuales2 +\
 h_actuales2/2.0) )
                if i_act == 2:
                    p_ultima = p1
                try:
                    mx += float(p1[1] - p2[1]) / (p1[0] - p2[0])
                except:
                    bool_mx = False
                try:
                    my += float(p1[0] - p2[0]) / (p1[1] - p2[1])
                except:
                    bool_my = False
                delta_x += p1[0] - p2[0]
                delta_y += p1[1] - p2[1]
            mx       = mx / (j - 1)
            my       = my / (j - 1)
            delta_x = delta_x / (j - 1)
            delta_y = delta_y / (j - 1)
            if bool_mx:
                b = p_ultima[1] - mx * p_ultima[0]
                if abs(delta_x) >= abs(delta_y):
                    x_aprox = int(p_ultima[0] + delta_x)
                    y_aprox = numpy.int16(mx * x_aprox + b)
                else:
                    y_aprox = int(p_ultima[1] + delta_y)
                    x_aprox = numpy.int16((y_aprox - b) / mx)
            elif bool_my:
                b = p_ultima[0] - my * p_ultima[1]
                if abs(delta_x) >= abs(delta_y):
                    x_aprox = int(p_ultima[0] + delta_x)
                    y_aprox = numpy.int16((x_aprox - b) / my)
                else:
                    y_aprox = int(p_ultima[1] + delta_y)
                    x_aprox = numpy.int16(my * y_aprox + b)
```

```
            else:
                continue
            if x_aprox < 0 or x_aprox > width or y_aprox < 0 or y_aprox\
 > height:
                x_aprox = int(p_ultima[0] + delta_x)
                y_aprox = int(p_ultima[1] + delta_y)
                bool_aux = False
            if math.sqrt( ( x_aprox - p_ultima[0] ) ** 2 + ( y_aprox - \
p_ultima[1] ) ** 2 ) < 2: continue
            flag_actuales = flag_actuales.replace(',coord_aprox', '') +\
 ',coord_aprox'
            if i_actuales in list_bloque_borde: flag_actuales = flag_ac\
tuales.replace(',aprox_bloque_borde', '') + ',aprox_bloque_borde'
            vactuales[i_actuales].append([ cont_frame, n_actuales , [ x\
_aprox - w_actuales/2 , y_aprox - h_actuales/2 , w_actuales, h_actuales\
 ],
                                           cant_v_actuales, flag_actual\
es ])
    return coord_blob_frame, vchequeo, vactuales, img_fast
def iden_vehiculos_adyacentes_ampliado( cont_frame, coord_blob_frame, v\
chequeo, vactuales, B_entra, C_entra, wh_adya_ampl ):
    copiar_borrar_chequeo = []
    copiar_borrar_actuales = []
    m = .5
    for i, i_rect in enumerate(coord_blob_frame):
        _, rect = i_rect
        bool_skip = False
        x, y, w, h = rect
        dis_coord_blob = modulo1.dis_punto_borde((x + w/2), (y + h/2), \
B_entra, C_entra)
        for i_chequeo, blob_chequeo in enumerate(vchequeo):
            bool_skip = False
            f_chequeo, rect_chequeo, cant_v, flag = blob_chequeo[-1]
            if (not f_chequeo == (cont_frame - 1)) or rect_chequeo is N\
one: continue
            x_chequeo, y_chequeo, w_chequeo, h_chequeo = rect_chequeo
            cx = (x + w/2) - (x_chequeo + w_chequeo/2 - wh_adya_ampl/2)\

            cy = (y + h/2) - (y_chequeo + h_chequeo/2 - wh_adya_ampl/2)\
```

```
            if cx > wh_adya_ampl or cx < 0 or cy > wh_adya_ampl or cy <\
 0: continue
            dis_chequeo = modulo1.dis_punto_borde((x_chequeo + w_cheque\
o/2), (y_chequeo + h_chequeo/2), B_entra, C_entra)
            if not (dis_coord_blob - dis_chequeo) >= - 0.5: continue
            bool_skip = True
            copiar_borrar_chequeo.append([i,i_chequeo])
            break
        if bool_skip: continue
        for i_actuales, blob_actuales in enumerate(vactuales):
            f_actuales, n_actuales, rect_actuales, cant_v_actuales, fla\
g = blob_actuales[-1]
            if (not f_actuales == (cont_frame - 1)) or rect_actuales is\
 None: continue
            x_actuales, y_actuales, w_actuales, h_actuales = rect_actua\
les
            cx = (x + w/2) - (x_actuales + w_actuales/2 - wh_adya_ampl/\
2)
            cy = (y + h/2) - (y_actuales + h_actuales/2 - wh_adya_ampl/\
2)
            if cx > wh_adya_ampl or cx < 0 or cy > wh_adya_ampl or cy <\
 0: continue
            dis_actuales = modulo1.dis_punto_borde((x_actuales + w_actu\
ales/2), (y_actuales + h_actuales/2), B_entra, C_entra)
            if not (dis_coord_blob - dis_actuales) >= - 0.5: continue
            copiar_borrar_actuales.append([i,i_actuales])
            break
    borrar_de_coord = []
    for i, i_chequeo in copiar_borrar_chequeo:
        flag_chequeo = vchequeo[i_chequeo][-1][3].replace(',coord_aprox\
', '').replace(',coord_exac', '')
        vchequeo[i_chequeo].append( [cont_frame, coord_blob_frame[i][1]\
, vchequeo[i_chequeo][-1][2], flag_chequeo + ',coord_exac' ] )
        borrar_de_coord.append(i)
    for i, i_actuales in copiar_borrar_actuales:
        flag_actuales = vactuales[i_actuales][-1][4].replace(',coord_ap\
rox', '').replace(',coord_exac', '')
        vactuales[i_actuales].append( [cont_frame, vactuales[i_actuales\
][-1][1], coord_blob_frame[i][1], vactuales[i_actuales][-1][3],
                                       flag_actuales + ',coord_exac' ]\
```

```
 )
        borrar_de_coord.append(i)
    for i_coord in sorted(list(set(borrar_de_coord)), reverse=True):
        coord_blob_frame.pop(i_coord)
    return coord_blob_frame, vchequeo, vactuales
def transferir_bloques( cont_frame, vchequeo, vactuales, vlistos, n_veh\
iculo, width, height, inters_mask ):
    for i_actuales in range(len(vactuales))[::-1] :
        blob_actuales = vactuales[i_actuales][-1]
        f_actuales, n_actuales, rect_actuales, cant_v_actuales, flag_ac\
tuales = blob_actuales
        if rect_actuales is None: continue
        x_actuales, y_actuales, w_actuales, h_actuales = rect_actuales
        cx = x_actuales + w_actuales/2
        cy = y_actuales + h_actuales/2
        if cx >= width or cx < 0 or cy >= height or cy < 0: continue
        if inters_mask[ cy ][ cx ] == 0:
            vlistos.append(vactuales[i_actuales])
            vactuales.pop(i_actuales)
    for i_chequeo in range(len(vchequeo))[::-1] :
        if len(vchequeo[i_chequeo]) < 3: continue
        cant_coord_aprox = str(vchequeo[i_chequeo]).find('coord_aprox')\

        if cant_coord_aprox == -1:
            vchequeo_i_copia = vchequeo[i_chequeo]
            for i_cheq in range( len(vchequeo_i_copia) ):
                cant_v_chequeo = vchequeo_i_copia[i_cheq][2]
                vchequeo_i_copia[i_cheq].insert(1, range( n_vehiculo + \
1 , n_vehiculo + cant_v_chequeo + 1 ))
            n_vehiculo += cant_v_chequeo
            vactuales.append( vchequeo_i_copia )
            vchequeo.pop(i_chequeo)
            continue
        else:
            cont_coord_no_aprox = 0
            for f_chequeo, rect_chequeo, cant_v, flag in vchequeo[i_che\
queo]:
                if flag.find('coord_aprox') == -1:
                    cont_coord_no_aprox += 1
                elif cont_coord_no_aprox >=3:
```

```
                    continue
                else:
                    cont_coord_no_aprox = 0
            if cont_coord_no_aprox >=3:
                vchequeo_i_copia = vchequeo[i_chequeo]
                for i_cheq in range( len(vchequeo_i_copia) ):
                    cant_v_chequeo = vchequeo_i_copia[i_cheq][2]
                    vchequeo_i_copia[i_cheq].insert(1, range( n_vehicul\
o + 1 , n_vehiculo + cant_v_chequeo + 1 ))
                n_vehiculo += cant_v_chequeo
                vactuales.append( vchequeo_i_copia )
                vchequeo.pop(i_chequeo)
    return vchequeo, vactuales, vlistos, n_vehiculo
def eliminar_blobs( cont_frame, vchequeo, vactuales ):
    for i_actuales in range(len(vactuales))[::-1] :
        blob_actuales = vactuales[i_actuales][-1]
        f_actuales, n_actuales, rect_actuales, cant_v_actuales, flag_ac\
tuales = blob_actuales
        if f_actuales <= cont_frame - 2:
            vactuales.pop(i_actuales)
    for i_chequeo in range(len(vchequeo))[::-1] :
        blob_chequeo = vchequeo[i_chequeo][-1]
        f_chequeo, rect_chequeo, cant_v_chequeo, flag_chequeo = blob_ch\
equeo
        if f_chequeo <= cont_frame - 2:
            vchequeo.pop(i_chequeo)
    return vchequeo, vactuales
def agregar_nuevo_blob( cont_frame, coord_blob_frame, vchequeo, B_entra\
, C_entra, B_entra_amp, C_entra_amp, umbral_entra, width, height, img_f\
ast ):
    for i_coord, rect in coord_blob_frame:
        x, y, w, h = rect
        if (x <= 1) or (y <= 1) or (x + w >= width - 1) or (y + h >= he\
ight - 1): continue
        esquinas = [ (x     , y     ),
                     (x + w , y     ),
                     (x     , y + h ),
                     (x + w , y + h ) ]
        for px, py in esquinas:
            cv2.circle(img_fast, (px, py), 3, [0,255,255], -1)
```

```
            dis_entra       = modulo1.dis_punto_borde(px, py, B_entra, \
C_entra)
            dis_entra_amp   = modulo1.dis_punto_borde(px, py, B_entra_a\
mp, C_entra_amp)
            if dis_entra < umbral_entra or dis_entra_amp < umbral_entra\
:
                vchequeo.append([ [cont_frame, rect, 1, ''] ])
                break
    return vchequeo, img_fast
def marca_bloques_img( cont_frame, vchequeo, vactuales, vlistos, B_fren\
te, C_frente, B_izq, C_izq, B_der, C_der, p_entra_izq, \
                       umbral_frente, umbral_izq, umbral_der, tol_der,\
 dis_fueradeimagen, index_listos, cont_vehic_frente, cont_vehic_izq, co\
nt_vehic_der, \
                       frame_ultimo, img_fast ):
    cv2.putText( img_fast, 'Cuadro
    cv2.putText( img_fast, 'Cuadro
    for blob_chequeo in vchequeo:
        f_chequeo, rect_chequeo, cant_v_chequeo, flag_chequeo = blob_ch\
equeo[-1]
        if f_chequeo <> cont_frame or rect_chequeo is None: continue
        x, y, w, h = rect_chequeo
        if flag_chequeo.find('coord_aprox') == -1:
            cv2.rectangle(img_fast, (x, y), (x + w, y + h), (0, 255, 25\
5), 1)
            cv2.circle(img_fast, (x + w/2, y + h/2), 3, [0,0,255], -1)
        else:
            cv2.rectangle(img_fast, (x, y), (x + w, y + h), (255, 0, 25\
5), 1)
            cv2.circle(img_fast, (x + w/2, y + h/2), 3, [255,0,255], -1\
)
        cv2.putText( img_fast, str(cant_v_chequeo), (x,y), cv2.FONT_HER\
SHEY_TRIPLEX, 0.4, (255,255,255), 2 )
        cv2.putText( img_fast, str(cant_v_chequeo), (x,y), cv2.FONT_HER\
SHEY_TRIPLEX, 0.4, (0,0,0) )
    for blob_actuales in vactuales:
        f_actuales, n_actuales, rect_actuales, cant_v_actuales, flag_ac\
tuales = blob_actuales[-1]
        if f_actuales <> cont_frame or rect_actuales is None: continue
        x, y, w, h = rect_actuales
```

```
        if flag_actuales.find('coord_aprox') == -1:
            cv2.rectangle(img_fast, (x, y), (x + w, y + h), (255, 255, \
0), 1)
            cv2.circle(img_fast, (x + w/2, y + h/2), 3, [0,255,0], -1)
        else:
            cv2.rectangle(img_fast, (x, y), (x + w, y + h), (255, 0, 25\
5), 1)
            cv2.circle(img_fast, (x + w/2, y + h/2), 3, [0,255,0], -1)
        cv2.putText( img_fast, str(cant_v_actuales), (x,y), cv2.FONT_HE\
RSHEY_TRIPLEX, 0.4, (255,255,255), 2 )
        cv2.putText( img_fast, str(cant_v_actuales), (x,y), cv2.FONT_HE\
RSHEY_TRIPLEX, 0.4, (0,0,0) )
    for i_listos, blob_listos in enumerate(vlistos):
        if i_listos <= index_listos: continue
        index_listos += 1
        f_listos, n_listos, rect_listos, cant_v_listos, flag_listos = b\
lob_listos[-1]
        if f_listos <> cont_frame or rect_listos is None: continue
        x, y, w, h = rect_listos
        dis_frente      = modulo1.dis_punto_borde( (x + w/2), (y + h/2)\
, B_frente, C_frente)
        dis_izq         = modulo1.dis_punto_borde( (x + w/2), (y + h/2)\
, B_izq, C_izq)
        dis_der         = modulo1.dis_punto_borde( (x + w/2), (y + h/2)\
, B_der, C_der)
        list_dis_borde  = [ dis_frente, dis_izq, dis_der ]
        dis_min = min(list_dis_borde)
        if dis_min == dis_fueradeimagen: continue
        pos_min = [index for index,value in enumerate(list_dis_borde) i\
f value == dis_min][0]
        if pos_min == 1 or dis_izq <= umbral_izq:
            cont_vehic_izq += cant_v_listos
            frame_ultimo = f_listos
        elif pos_min == 2 or dis_der <= umbral_der:
            p1 = [(x + w/2), (y + h/2)]
            _, _, rect_listos2, _, _ = blob_listos[0]
            x2, y2, w2, h2 = rect_listos2
            p2  = [(x2 + w2/2), (y2 + h2/2)]
            m   = float(p1[1] - p2[1]) / (p1[0] - p2[0])
            b   = p1[1] - m * p1[0]
```

```
            y_res   = b - m * ( ( B_izq * b + C_izq ) / ( 1 + B_izq * m\
 ) )
            y_int   = p_entra_izq[1]
            if y_res > y_int + tol_der:
                cont_vehic_der += cant_v_listos
        elif pos_min == 0:
            cont_vehic_frente  += cant_v_listos
            frame_ultimo        = f_listos
        cv2.rectangle(img_fast, (x, y), (x + w, y + h), (0, 255, 0), 1)\

        cv2.circle(img_fast, (x + w/2, y + h/2), 3, [0,255,0], -1)
        cv2.putText( img_fast, str(cant_v_listos), (x,y), cv2.FONT_HERS\
HEY_TRIPLEX, 0.4, (255,255,255), 2 )
        cv2.putText( img_fast, str(cant_v_listos), (x,y), cv2.FONT_HERS\
HEY_TRIPLEX, 0.4, (0,0,0) )
    cv2.putText( img_fast, 'Siguen derecho:    ' + str(cont_vehic_frente\
), (200, 40), cv2.FONT_HERSHEY_TRIPLEX, 0.4, (255,255,255), 3 )
    cv2.putText( img_fast, 'Siguen derecho:    ' + str(cont_vehic_frente\
), (200, 40), cv2.FONT_HERSHEY_TRIPLEX, 0.4, (0,0,0) )
    cv2.putText( img_fast, 'Cruzan izquierda: ' + str(cont_vehic_izq), \
(200, 60), cv2.FONT_HERSHEY_TRIPLEX, 0.4, (255,255,255), 3 )
    cv2.putText( img_fast, 'Cruzan izquierda: ' + str(cont_vehic_izq), \
(200, 60), cv2.FONT_HERSHEY_TRIPLEX, 0.4, (0,0,0) )
    cv2.putText( img_fast, 'Cruzan derecha: ' + str(cont_vehic_der), (2\
00, 80), cv2.FONT_HERSHEY_TRIPLEX, 0.4, (255,255,255), 3 )
    cv2.putText( img_fast, 'Cruzan derecha: ' + str(cont_vehic_der), (2\
00, 80), cv2.FONT_HERSHEY_TRIPLEX, 0.4, (0,0,0) )
    return int(cont_vehic_frente), int(cont_vehic_izq), int(cont_vehic_\
der), frame_ultimo, index_listos, img_fast
def descarta_bloques( cont_frame, vchequeo, vactuales, B_entra, C_entra\
, umbral_entra ):
    borrar_chequeo = []
    for i_chequeo in range(len(vchequeo)):
        blob_chequeo = vchequeo[i_chequeo]
        if len(blob_chequeo) < 3: continue
        bool_primero = True
        cont_avance_neg = 0
        f_cheq, rect_cheq, cant_v_cheq, flag_cheq = blob_chequeo[-1]
        if rect_cheq is None: continue
        x_cheq, y_cheq, w_cheq, h_cheq = rect_cheq
```

```
        p0 = ( x_cheq + w_cheq/2, y_cheq + h_cheq/2 )
        f_cheq, rect_cheq, cant_v_cheq, flag_cheq = blob_chequeo[0]
        if rect_cheq is None: continue
        x_cheq, y_cheq, w_cheq, h_cheq = rect_cheq
        p1 = ( x_cheq + w_cheq/2, y_cheq + h_cheq/2 )
        dis0 = modulo1.dis_punto_borde(p0[0], p0[0], B_entra, C_entra)
        dis1 = modulo1.dis_punto_borde(p1[0], p1[0], B_entra, C_entra)
        if dis0 > dis1:
                borrar_chequeo.append(i_chequeo)
                continue
        for f_cheq, rect_cheq, cant_v_cheq, flag_cheq in blob_chequeo[:\
:-1]:
            if rect_cheq is None: break
            x_cheq, y_cheq, w_cheq, h_cheq = rect_cheq
            if bool_primero:
                bool_primero = False
                p0 = ( x_cheq + w_cheq/2, y_cheq + h_cheq/2 )
                continue
            p1 = ( x_cheq + w_cheq/2, y_cheq + h_cheq/2 )
            dis0 = modulo1.dis_punto_borde(p0[0], p0[0], B_entra, C_ent\
ra)
            dis1 = modulo1.dis_punto_borde(p1[0], p1[0], B_entra, C_ent\
ra)
            if dis0 > dis1:
                cont_avance_neg += 1
            if cont_avance_neg == 2:
                borrar_chequeo.append(i_chequeo)
                break
            p0 = p1
    for i_chequeo in sorted(list(set(borrar_chequeo)), reverse=True):
        vchequeo.pop(i_chequeo)
    borrar_actuales = []
    for i_actuales in range(len(vactuales)):
        blob_actuales = vactuales[i_actuales]
        if len(blob_actuales) < 3: continue
        if len(blob_actuales) > 7: continue
        bool_primero = True
        cont_avance_neg = 0
        f_act, n_act, rect_act, cant_v_act, flag_act = blob_actuales[-1\
]
```

```
        if rect_act is None: continue
        x_act, y_act, w_act, h_act = rect_act
        p0 = ( x_act + w_act/2, y_act + h_act/2 )
        f_act, n_act, rect_act, cant_v_act, flag_act = blob_actuales[0]\

        if rect_act is None: continue
        x_act, y_act, w_act, h_act = rect_act
        p1 = ( x_act + w_act/2, y_act + h_act/2 )
        dis0 = modulo1.dis_punto_borde(p0[0], p0[0], B_entra, C_entra)
        dis1 = modulo1.dis_punto_borde(p1[0], p1[0], B_entra, C_entra)
        if dis0 > dis1:
                borrar_actuales.append(i_actuales)
                continue
        for f_act, n_act, rect_act, cant_v_act, flag_act in blob_actual\
es[::-1]:
            if rect_act is None: break
            x_act, y_act, w_act, h_act = rect_act
            if bool_primero:
                bool_primero = False
                p0 = ( x_act + w_act/2, y_act + h_act/2 )
                continue
            p1 = ( x_act + w_act/2, y_act + h_act/2 )
            dis0 = modulo1.dis_punto_borde(p0[0], p0[0], B_entra, C_ent\
ra)
            dis1 = modulo1.dis_punto_borde(p1[0], p1[0], B_entra, C_ent\
ra)
            if dis0 > dis1:
                cont_avance_neg += 1
            if cont_avance_neg == 2:
                borrar_actuales.append(i_actuales)
                break
            p0 = p1
    for i_actuales in sorted(list(set(borrar_actuales)), reverse=True):\

        vactuales.pop(i_actuales)
    return vchequeo, vactuales
def main_tracking( cont_frame, coord_blob_frame, vchequeo, vactuales, v\
listos, n_vehiculo, width, height, \
                    B_entra, C_entra, B_entra_amp, C_entra_amp, B_frent\
e, C_frente, B_izq, C_izq, B_der, C_der, p_entra_izq, \
```

```
                    umbral_entra, umbral_frente, umbral_izq, umbral_der\
, umbral_adyacente, umbral_relacion_area, umbral_blob_aprox, tol_der, \\

                    wh_adya_ampl, dis_fueradeimagen, inters_mask, img_f\
ast, index_listos, cont_vehic_frente, cont_vehic_izq, cont_vehic_der, f\
rame_ultimo):
    coord_blob_frame, vchequeo, vactuales                = iden_vehiculo\
s_adyacentes( cont_frame, coord_blob_frame, vchequeo, vactuales,
                                                                        \
                B_entra, C_entra, umbral_entra,
                                                                        \
                umbral_adyacente, umbral_relacion_area )
    coord_blob_frame, vchequeo, vactuales, n_vehiculo   = iden_vehiculo\
s_convergen( cont_frame, coord_blob_frame, vchequeo, vactuales, n_vehic\
ulo )
    coord_blob_frame, vchequeo, vactuales, n_vehiculo   = iden_vehiculo\
s_divergen( cont_frame, coord_blob_frame, vchequeo, vactuales, n_vehicu\
lo )
    coord_blob_frame, vchequeo, vactuales, n_vehiculo   = iden_comb_com\
plejas( cont_frame, coord_blob_frame, vchequeo, vactuales, n_vehiculo )\

    coord_blob_frame, vchequeo, vactuales, n_vehiculo   = iden_vehiculo\
s_convergen_borde( cont_frame, coord_blob_frame, vchequeo, vactuales, n\
_vehiculo,
                                                                        \
                B_entra, C_entra, umbral_entra, width, height)
    coord_blob_frame, vchequeo, vactuales, n_vehiculo   = iden_vehiculo\
s_divergen_borde( cont_frame, coord_blob_frame, vchequeo, vactuales, n_\
vehiculo,
                                                                        \
                B_frente, C_frente, B_izq, C_izq, umbral_frente, u\
mbral_izq)
    coord_blob_frame, vchequeo, vactuales, img_fast     = agrega_coord_a\
prox( cont_frame, coord_blob_frame, vchequeo, vactuales,
                                                                        \
        B_entra, C_entra, B_frente, C_frente, B_izq, C_izq, B_der, C_d\
er,
                                                                        \
        umbral_entra, umbral_frente, umbral_izq, umbral_der, umbral_bl\
ob_aprox,
```

```
                                                                    \
        width, height, dis_fueradeimagen, inters_mask, img_fast)
    coord_blob_frame, vchequeo, vactuales                = iden_vehiculo\
s_adyacentes_ampliado( cont_frame, coord_blob_frame, vchequeo, vactuale\
s,
                                                                    \
                        B_entra, C_entra, wh_adya_ampl )
    vchequeo, vactuales                                  = descarta_bloq\
ues( cont_frame, vchequeo, vactuales, B_entra, C_entra, umbral_entra )
    vchequeo, vactuales, vlistos, n_vehiculo             = transferir_bl\
oques( cont_frame, vchequeo, vactuales, vlistos, n_vehiculo,
                                                                    \
        width, height, inters_mask )
    vchequeo, vactuales                                  = eliminar_blob\
s( cont_frame, vchequeo, vactuales )
    vchequeo, img_fast                                   = agregar_nuevo\
_blob( cont_frame, coord_blob_frame, vchequeo, B_entra, C_entra,
                                                                    \
        B_entra_amp, C_entra_amp, umbral_entra, width, height, img_fas\
t )
    resultado_marca_bloques_img                          = marca_bloques\
_img( cont_frame, vchequeo, vactuales, vlistos,
                                                                    \
     B_frente, C_frente, B_izq, C_izq, B_der, C_der, p_entra_izq,
                                                                    \
     umbral_frente, umbral_izq, umbral_der, tol_der,
                                                                    \
     dis_fueradeimagen, index_listos,
                                                                    \
     cont_vehic_frente, cont_vehic_izq, cont_vehic_der,
                                                                    \
     frame_ultimo, img_fast )
    cont_vehic_frente, cont_vehic_izq, cont_vehic_der, frame_ultimo, in\
dex_listos, img_fast    = resultado_marca_bloques_img
    list_cood_v = coord_blob_frame, vchequeo, vactuales, vlistos, n_veh\
iculo
    cont_vehiculos = cont_vehic_frente, cont_vehic_izq, cont_vehic_der,\
 frame_ultimo, index_listos
    return list_cood_v, img_fast, cont_vehiculos
```

2.4. Módulo Básico

El código a continuación contiene las funciones que forman el módulo de detección, módulo de inicialización y operaciones básicas.

```
import cv2
import numpy
import math
def abrir_video(video_name):
    videoCapture = cv2.VideoCapture(video_name)
    fps = videoCapture.get(cv2.cv.CV_CAP_PROP_FPS)
    side = (int(videoCapture.get(cv2.cv.CV_CAP_PROP_FRAME_WIDTH)), int(\
videoCapture.get(cv2.cv.CV_CAP_PROP_FRAME_HEIGHT)))
    width, height= side[:2]
    return videoCapture, fps, width, height
coord = []
def detec_click(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:
        coord.append((x,y))
    if event == cv2.EVENT_RBUTTONDOWN:
        del coord[:]
def guardar_coord_clicks(img_g):
    win_name = 'Indentificacion de interseccion'
    img_ejem = cv2.imread('img_ejem.png')
    cv2.imshow(win_name + ' (EJEMPLO)', img_ejem)
    cv2.imshow(win_name, img_g)
    cv2.setMouseCallback(win_name, detec_click)
    while True:
        img_g0 = img_g.copy()
        for p in coord:
            cv2.circle(img_g0, p, 3, 255, -1)
        cv2.imshow(win_name, img_g0)
        letra = cv2.waitKey(10)
        if len(coord)==10 or letra == 13:
            break
    for p in coord:
        cv2.circle(img_g0, p, 3, 255, -1)
    cv2.imshow(win_name + ' (listo)', img_g0)
    cv2.destroyWindow(win_name)
    return coord
```

```
def calculo_vertices_inters(coord):
    coord_recta = []
    bool = True
    aux_coord = []
    for x,y in coord:
        if bool:
            bool = not bool
            aux_coord.append(x)
            aux_coord.append(y)
        else:
            bool = not bool
            aux_coord.append(x)
            aux_coord.append(y)
            coord_recta.append(aux_coord)
            aux_coord = []
    recta = []
    for x1,y1,x2,y2 in coord_recta:
        aux_recta = []
        aux_recta.append(float(y2 - y1) / (x2 - x1))
        aux_recta.append(y1 - aux_recta[0] * x1 )
        recta.append(aux_recta)
    coord_inters = []
    bool = True
    for m,b in recta:
        if bool:
            bool = not bool
            m2,b2 = [m,b]
            continue
        aux_inters = [0,0]
        aux_inters[0] = -float(b2 - b)/(m2 - m)
        aux_inters[1] = m * aux_inters[0] + b
        coord_inters.append(aux_inters)
        m2,b2 = [m,b]
    [m,b] = recta[0]
    aux_inters = [0,0]
    aux_inters[0] = -float(b2 - b)/(m2 - m)
    aux_inters[1] = m * aux_inters[0] + b
    coord_inters.append(aux_inters)
    coord_inters = list( map(int,x) for x in coord_inters )
    coord_inters.append(coord_inters[0])
```

```
    return coord_inters
def mask_interseccion(width, height, coord_inters, img_g):
    inters_mask = 255 * numpy.ones((height, width), numpy.uint8)
    img_g_aux = img_g.copy()
    for i in range(4):
        cv2.line(inters_mask, (coord_inters[i][0], coord_inters[i][1]),\
 (coord_inters[i + 1][0], coord_inters[i + 1][1]), 0, 2)
        cv2.line(img_g_aux, (coord_inters[i][0], coord_inters[i][1]), (\
coord_inters[i + 1][0], coord_inters[i + 1][1]), 255, 2)
    mask = numpy.zeros((height+2, width+2), numpy.uint8)
    for x, y in [ [x,y] for x in [0, width-1] for y in [0, height-1] ]:\

        cv2.floodFill(inters_mask, mask, (x,y), 0)
    return inters_mask, img_g_aux
def identifica_bordes(coord_inters, img_backg_int):
    win_name = 'Indentificacion de bordes'
    img_backg_int0 = cv2.cvtColor(img_backg_int, cv2.COLOR_GRAY2BGR)
    print '\nINSTRUCCIONES: A continuacion es necesario que identifique\
 cada borde que delimina la interseccion EN ORDEN.\n\
Primero:\tBorde de entrada (por donde entran los vehiculos de interes),\
\n\
Segundo:\tBorde de salida cuando los vehiculos de interes cruzan a su D\
ERECHA\n\
Tercera:\tBorde de salida cuando los vehiculos de interes cruzan a su I\
ZQUIERDA.\n\n\
Presione:\n\
\t"q"\t\tpara salir del proceso.\n\
\t"ENTER"\t\tpara seleccionar el borde marcado.\n\
\tFLECHA DERECHA\tpara cambiar de borde a seleccionar.\n'
    i_borde_interes = 0
    coord_borde = []
    i_borde_select = []
    while True:
        img_backg_int1 = img_backg_int0.copy()
        letra = cv2.waitKey(1)
        if letra == 113: break
        if len(coord_borde) == 0:
            cv2.line(img_backg_int1, (coord_inters[i_borde_interes][0],\
 coord_inters[i_borde_interes][1]),
                                    (coord_inters[i_borde_interes + 1]\
```

```
[0], coord_inters[i_borde_interes + 1][1]), [0,255,0], 2)
        elif len(coord_borde) == 1:
            cv2.line(img_backg_int1, (coord_inters[i_borde_interes][0],\
 coord_inters[i_borde_interes][1]),
                                     (coord_inters[i_borde_interes + 1]\
[0], coord_inters[i_borde_interes + 1][1]), [255,0,0] , 2)
        elif len(coord_borde) == 2:
            cv2.line(img_backg_int1, (coord_inters[i_borde_interes][0],\
 coord_inters[i_borde_interes][1]),
                                     (coord_inters[i_borde_interes + 1]\
[0], coord_inters[i_borde_interes + 1][1]), [255,255,0] , 2)
        else:
            cv2.line(img_backg_int1, (coord_inters[i_borde_interes][0],\
 coord_inters[i_borde_interes][1]),
                                     (coord_inters[i_borde_interes + 1]\
[0], coord_inters[i_borde_interes + 1][1]), [0,255,255], 2)
            coord_borde.append([(coord_inters[i_borde_interes][0], coor\
d_inters[i_borde_interes][1]),
                                (coord_inters[i_borde_interes + 1][0], \
coord_inters[i_borde_interes + 1][1])])
            cv2.imshow(win_name, img_backg_int1)
            break
        if letra == 13:
            coord_borde.append([(coord_inters[i_borde_interes][0], coor\
d_inters[i_borde_interes][1]),
                                (coord_inters[i_borde_interes + 1][0], \
coord_inters[i_borde_interes + 1][1])])
            img_backg_int0 = img_backg_int1.copy()
            i_borde_select.append(i_borde_interes)
        elif letra == 2555904:
            i_borde_interes += 1
            i_borde_interes = i_borde_interes %4
        while (i_borde_interes in i_borde_select):
            i_borde_interes += 1
            i_borde_interes = i_borde_interes %4
        cv2.imshow(win_name, img_backg_int1)
    return coord_borde
def llenado_bordes(img_canny_d, inters_mask, width, height, coord_rando\
m):
    img_canny_d = img_canny_d & inters_mask
```

```
    img_floodfill = img_canny_d.copy()
    height, width = img_canny_d.shape[:2]
    list_coord = [ (x,y) for x in [0, width-1] for y in [0, height-1] ]\
 + coord_random
    for p in list_coord:
        mask = numpy.zeros((height+2, width+2), numpy.uint8)
        cv2.floodFill(img_floodfill, mask, p,255)
    img_floodfill_inv = cv2.bitwise_not(img_floodfill)
    img_foreg = img_canny_d | img_floodfill_inv
    return img_foreg
def calc_fondo(video_name,coord_inters):
    videoCapture, fps, width, height = abrir_video(video_name)
    success, frame = videoCapture.read()
    img_g = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    inters_mask, img_g = mask_interseccion(width, height,coord_inters,i\
mg_g)
    kernel_dilate_5 = numpy.ones((5,5), numpy.uint8)
    img_canny = cv2.Canny(img_g,30,240)
    img_canny_d = cv2.dilate(img_canny,kernel_dilate_5,iterations=1)
    img_foreg = llenado_bordes(img_canny_d, inters_mask, width, height)\

    img_backg = numpy.float32(img_foreg)
    cont_frame = -1
    while success and cont_frame < 1000:
        cont_frame += 1
        img_g = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        img_canny = cv2.Canny(img_g,30,240)
        img_canny_d = cv2.dilate(img_canny,kernel_dilate_5,iterations=1\
)
        img_foreg = llenado_bordes(img_canny_d, inters_mask, width, hei\
ght)
        cv2.accumulateWeighted(img_foreg, img_backg, 0.9)
        success, frame = videoCapture.read()
    _,img_backg = cv2.threshold(img_backg,10,255,cv2.THRESH_BINARY)
    return img_backg
def tester_color_pixel(img_g):
    name = 'Imagen a analizar'
    print '\n_____Modo de prueba:_____\
_____\n\
***Presione click izquierdo en "Imagen a analizar" para conocer\n\
```

```
   el color y coordenadas de un pixel.\n***Presione ENTER para salir.\n\
'
    cv2.imshow(name, img_g)
    cv2.setMouseCallback(name, detec_click)
    coord.append((0,0))
    p = coord[-1]
    while True:
        if coord == []:
            cv2.waitKey(10)
            continue
        if not p == coord[-1]:
            p = coord[-1]
            print 'P', p, ' = ', img_g[p[1],p[0]]
        letra = cv2.waitKey(10)
        if letra == 13:
            print '_____________________________________________________\
__________\n'
            break
def fondo_promedio(video_name):
    videoCapture, fps, width, height = abrir_video(video_name)
    img_backg = 110 * numpy.ones((height, width) , numpy.float32)
    success = True
    cont_frame = -1
    while success and cont_frame < 300:
        success, frame = videoCapture.read()
        cont_frame += 1
        if frame.shape[2] == 3:
            img_g = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        else:
            img_g = frame.copy()
        cv2.accumulateWeighted(img_g, img_backg, 0.005)
        if cont_frame %100 == 0:
            None
    img_backg = numpy.uint8(img_backg)
    return img_backg
def foreground(cont_frame, frame, img_backg, img_foreg_prom, mask_backg\
_total_1, mask_backg_total_2, bool_mask_backg, \
               mhi, coord_inters, p_der, coord_der, kernel_dilate, ker\
nel_erode, coord_borde_entrada, \
               coord_borde_derecha, inters_mask, width, height, coord_\
```

```
random):
    img_g = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    img_g_backg = cv2.absdiff(img_g, img_backg)
    img_canny = cv2.Canny(img_g_backg,40,100)
    img_canny_d = cv2.dilate(img_canny,kernel_dilate,iterations=1)
    img_canny_fill = llenado_bordes(img_canny_d, inters_mask, width, he\
ight, coord_random)
    img_foreg = cv2.erode(img_canny_fill, kernel_erode)
    img_foreg_conborde = img_foreg.copy()
    for i in range(4):
        if not coord_der == []:
            if ( tuple(coord_inters[i]) in coord_borde_derecha ) and ( \
tuple(coord_inters[i + 1]) in coord_borde_derecha ):
                cv2.line(img_foreg, p_der[0], p_der[1], 255, 15)
                cv2.line(img_foreg, p_der[1], coord_der[0], 255, 15)
                cv2.line(img_foreg, coord_der[0], coord_der[1], 255, 15\
)
            else:
                cv2.line(img_foreg, (coord_inters[i][0], coord_inters[i\
][1]), (coord_inters[i + 1][0], coord_inters[i + 1][1]), 255, 15)
    mask_ffill = numpy.zeros((height+2, width+2), numpy.uint8)
    for x, y in [ [x,y] for x in [0, width-1] for y in [0, height-1] ]:\

        cv2.floodFill(img_foreg, mask_ffill, (x,y), 255)
    mask_ffill = numpy.zeros((height+2, width+2), numpy.uint8)
    for x, y in [ [x,y] for x in [0, width-1] for y in [0, height-1] ]:\

        cv2.floodFill(img_foreg, mask_ffill, (x,y), 0)
    cv2.accumulateWeighted(img_foreg, img_foreg_prom, 0.1)
    _,mask_backg = cv2.threshold(img_foreg_prom,245,255,cv2.THRESH_BINA\
RY_INV)
    mask_backg_total_2 = cv2.bitwise_and(mask_backg_total_2, mask_backg\
)
    mask_backg_total_1 = cv2.bitwise_and(mask_backg_total_1, mask_backg\
)
    if cont_frame %200 == 0:
        bool_mask_backg     = 2
        mask_backg_total_1  = 255 * numpy.ones((height,width), numpy.fl\
oat32)
    elif cont_frame %100 == 0:
```

```
        bool_mask_backg      = 1
        mask_backg_total_2   = 255 * numpy.ones((height,width), numpy.fl\
oat32)
    if bool_mask_backg == 1:
        mask_backg_total = mask_backg_total_1
    else:
        mask_backg_total = mask_backg_total_2
    img_foreg_float = numpy.float32(img_foreg)
    img_foreg_nsombra = cv2.bitwise_and(img_foreg_float, mask_backg_tot\
al)
    img_foreg_nsombra = numpy.uint8(img_foreg_nsombra)
    cv2.updateMotionHistory(img_foreg_nsombra, mhi, cont_frame, 1)
    segmask, boundingrects  = cv2.segmentMotion(mhi, cont_frame, 1)
    inters_mask_inv         = cv2.bitwise_not(inters_mask)
    segmask_coord_random    = img_foreg | inters_mask_inv
    coord_random            = []
    cont_coord_random       = 0
    bool_coord_random       = False
    while not bool_coord_random:
        coord_random_x = numpy.random.randint(0, width)
        coord_random_y = numpy.random.randint(0, height)
        if segmask_coord_random[coord_random_y][coord_random_x] == 0:
            cont_coord_random += 1
            coord_random.append((coord_random_x, coord_random_y))
            if cont_coord_random == 10:
                bool_coord_random = True
    return img_foreg_conborde, img_foreg, img_foreg_nsombra, segmask, b\
oundingrects, mhi, img_foreg_prom, \
            mask_backg_total_1, mask_backg_total_2, bool_mask_backg, co\
ord_random
def dis_punto_borde(x, y, B, C):
    dis = abs( float( (x) + B * float(y) + C ) / math.sqrt( 1 + float(B\
) ** 2 ) )
    return dis
def dibuja_umbrales(coord_inters, coord_der, coord_borde_entrada, coord\
_borde_izquierda, img_backg_int, B_entra, C_entra, B_frente, C_frente, \
\
                    B_izq, C_izq, B_der_amp, C_der_amp, umbral_entra, u\
mbral_frente, umbral_izq, umbral_der ):
    pareja_esquinas = [ [e1,e2] for e1 in coord_inters[:-1] for e2 in c\
```

```
oord_inters[:-1] if not e1 is e2 and e1 > e2  ]
    d_diagonal = 0
    esquinas_opuestas = 0
    for e1,e2 in pareja_esquinas:
        d_e1e2 = math.sqrt( (e1[0] - e2[0]) ** 2 + (e1[1] - e2[1]) ** 2\
 )
        if d_e1e2 > d_diagonal:
            d_diagonal = d_e1e2
            esquinas_opuestas_max = [e1,e2]
    esquinas_opuestas = [ esq for esq in pareja_esquinas if (not sorted\
(esq)[0] == sorted(esquinas_opuestas_max)[0]) and\
                                                        (not sorted\
(esq)[1] == sorted(esquinas_opuestas_max)[1]) ]
    esquinas_opuestas.append(esquinas_opuestas_max)
    recta = []
    for e1,e2 in esquinas_opuestas:
        x1,y1 = e1
        x2,y2 = e2
        aux_recta = []
        aux_recta.append(float(y2 - y1) / (x2 - x1))
        aux_recta.append(y1 - aux_recta[0] * x1 )
        recta.append(aux_recta)
    coord_centro_inters = []
    bool = True
    for m,b in recta:
        if bool:
            bool = not bool
            m2,b2 = [m,b]
            continue
        aux_inters = [0,0]
        aux_inters[0] = -float(b2 - b)/(m2 - m)
        aux_inters[1] = m * aux_inters[0] + b
        coord_centro_inters = aux_inters
        m2,b2 = [m,b]
    coord_centro_inters = tuple(map(int,coord_centro_inters))
    cv2.circle(img_backg_int, coord_centro_inters, 3, 255, -1)
    p_umbral_entra  = []
    p_umbral_frente = []
    p_umbral_izq    = []
    p_umbral_der    = []
```

```
    p_umbral        = []
    for p_inters in coord_inters[:-1]:
        if coord_centro_inters[0] < p_inters[0]:
            rango_x = range( coord_centro_inters[0], p_inters[0] + 1 )
        else:
            rango_x = range( coord_centro_inters[0], p_inters[0] - 1, -\
1 )
        m = float(p_inters[1] - coord_centro_inters[1]) / (p_inters[0] \
- coord_centro_inters[0])
        b = coord_centro_inters[1] - m * coord_centro_inters[0]
        bool_entra  = True
        bool_frente = True
        bool_izq    = True
        cont_puntos = 0
        for x in rango_x:
            y = m * x + b
            dis_entra = dis_punto_borde(x, y, B_entra, C_entra)
            if abs( abs(dis_entra) - umbral_entra) < 3 and bool_entra:
                bool_entra = False
                p_umbral_entra.append( (x, int(y)) )
                cont_puntos += 1
            dis_frente = dis_punto_borde(x, y, B_frente, C_frente)
            if abs( abs(dis_frente) - umbral_frente) < 3 and bool_frent\
e:
                bool_frente = False
                p_umbral_frente.append( (x, int(y)) )
                cont_puntos += 1
            dis_izq = dis_punto_borde(x, y, B_izq, C_izq)
            if abs( abs(dis_izq) - umbral_izq) < 3 and bool_izq:
                bool_izq = False
                p_umbral_izq.append( (x, int(y)) )
                cont_puntos += 1
            if cont_puntos == 2 and False:
                continue
    p_umbral_der    = []
    for p_inters in coord_der:
        if coord_centro_inters[0] < p_inters[0]:
            rango_x = range( coord_centro_inters[0], p_inters[0] + 1 )
        else:
            rango_x = range( coord_centro_inters[0], p_inters[0] - 1, -\
```

```
1 )
        m = float(p_inters[1] - coord_centro_inters[1]) / (p_inters[0] \
- coord_centro_inters[0])
        b = coord_centro_inters[1] - m * coord_centro_inters[0]
        for x in rango_x:
            y = m * x + b
            dis_der = dis_punto_borde(x, y, B_der_amp, C_der_amp)
            if abs( abs(dis_der) - umbral_der) < 3:
                p_umbral_der.append( (x, int(y)) )
                break
    if len(p_umbral_entra) == 2:
        cv2.line(img_backg_int, p_umbral_entra[0], p_umbral_entra[1], 2\
55 , 1)
        p_umbral.append(p_umbral_entra)
    if len(p_umbral_frente) == 2:
        cv2.line(img_backg_int, p_umbral_frente[0], p_umbral_frente[1],\
 255 , 1)
        p_umbral.append(p_umbral_frente)
    if len(p_umbral_izq) == 2:
        cv2.line(img_backg_int, p_umbral_izq[0], p_umbral_izq[1], 255 ,\
 1)
        p_umbral.append(p_umbral_izq)
    if len(p_umbral_der) == 2:
        cv2.line(img_backg_int, p_umbral_der[0], p_umbral_der[1], 255 ,\
 1)
        p_umbral.append(p_umbral_der)
    B_entra_amp, C_entra_amp = [1,1]
    if not coord_der == []:
        p1 = ( (coord_der[0][0] + coord_der[1][0]) / 2, (coord_der[0][1\
] + coord_der[1][1]) / 2 )
        p1 = ( (p1[0] + coord_der[1][0]) / 2, (p1[1] + coord_der[1][1])\
 / 2 )
        for e1 in coord_borde_entrada:
            for e2 in coord_borde_izquierda:
                if e1 == e2:
                    cx, cy = e1
        a = B_izq ** 2 + 1
        b = 2 * B_izq * ( C_izq + cx ) - 2 * cy
        c = ( C_izq + cx ) ** 2 + cy ** 2 - (umbral_entra ** 2)
        y_inter1    = int( ( - b - math.sqrt(b ** 2 - 4 * a * c) ) / ( \
```

```
2 * a ) )
        x_inter1    = - int( B_izq * y_inter1 + C_izq )
        dis_frente1 = dis_punto_borde(x_inter1, y_inter1, B_frente, C_f\
rente)
        y_inter2    = int( ( - b + math.sqrt(b ** 2 - 4 * a * c) ) / ( \
2 * a ) )
        x_inter2    = - int( B_izq * y_inter2 + C_izq )
        dis_frente2 = dis_punto_borde(x_inter2, y_inter2, B_frente, C_f\
rente)
        if dis_frente1 < dis_frente2:
            p2 = ( x_inter1, y_inter1 )
        else:
            p2 = ( x_inter2, y_inter2 )
        p_umbral.append([p1,p2])
        cv2.line(img_backg_int, p2 , p1, 255 , 1)
        B_entra_tem = - float( p1[0] - p2[0] ) / ( p1[1] - p2[1] )
        C_entra_tem = - (p1[0] + p1[1] * B_entra_tem)
        p_umbral_entra_amp = []
        for p_inters in coord_borde_entrada:
            if coord_centro_inters[0] < p_inters[0]:
                rango_x = range( coord_centro_inters[0], p_inters[0] + \
1 )
            else:
                rango_x = range( coord_centro_inters[0], p_inters[0] - \
1, -1 )
            m = float(p_inters[1] - coord_centro_inters[1]) / (p_inters\
[0] - coord_centro_inters[0])
            b = coord_centro_inters[1] - m * coord_centro_inters[0]
            i_final = len(rango_x) - 1
            for i, x in enumerate(rango_x):
                y = m * x + b
                dis_ent = dis_punto_borde(x, y, B_entra_tem, C_entra_te\
m)
                if abs( dis_ent - umbral_entra) < 3:
                    p_umbral_entra_amp.append( (x, int(y)) )
                    break
                if i == i_final:
                    p_umbral_entra_amp.append( (x, int(y)) )
        B_entra_amp = - float( p_umbral_entra_amp[0][0] - p_umbral_entr\
a_amp[1][0] ) / ( p_umbral_entra_amp[0][1] - p_umbral_entra_amp[1][1] )\
```

```
        C_entra_amp = - (p_umbral_entra_amp[0][0] + p_umbral_entra_amp[\
0][1] * B_entra_amp)
    return p_umbral, p_umbral_entra, p_umbral_frente, p_umbral_izq, p_u\
mbral_der, B_entra_amp, C_entra_amp, img_backg_int
def calc_limites_interseccion(width, height, coord, img_backg):
    coord_der   = []
    p_der       = 0
    umbral_der  = 1
    if len(coord) == 10:
        coord_der               = coord[-2:]
        coord                   = coord[:8]
    if (not len(coord) == 8):
        print '\n
        raise StandardError('Hubo un error al seleccionar las coordenad\
as. Es necesario seleccionar 8 puntos.')
    coord_inters = calculo_vertices_inters(coord)
    inters_mask, img_backg_int = mask_interseccion(width, height, coord\
_inters, img_backg)
    coord_borde_entrada, coord_borde_derecha, coord_borde_izquierda, co\
ord_borde_frente = [[(375, 718), (-164, 344)], [(499, 208), (375, 718)]\
,
                                                                      \
                    [(-164, 344), (137, 168)], [(137, 168), (499, 208)]\
]
    coord_borde_entrada, coord_borde_derecha, coord_borde_izquierda, co\
ord_borde_frente = identifica_bordes(coord_inters, img_backg_int)
    p_entra_izq = [ p for p in coord_borde_entrada if p in coord_borde_\
izquierda ][0]
    B_entra = - float( coord_borde_entrada[0][0] - coord_borde_entrada[\
1][0] ) / ( coord_borde_entrada[0][1] - coord_borde_entrada[1][1] )
    C_entra = - (coord_borde_entrada[0][0] + coord_borde_entrada[0][1] \
* B_entra)
    B_frente = - float( coord_borde_frente[0][0] - coord_borde_frente[1\
][0] ) / ( coord_borde_frente[0][1] - coord_borde_frente[1][1] )
    C_frente = - (coord_borde_frente[0][0] + coord_borde_frente[0][1] *\
 B_frente)
    B_izq = - float( coord_borde_izquierda[0][0] - coord_borde_izquierd\
a[1][0] ) / ( coord_borde_izquierda[0][1] - coord_borde_izquierda[1][1]\
 )
```

```
    C_izq = - (coord_borde_izquierda[0][0] + coord_borde_izquierda[0][1\
] * B_izq)
    B_der = - float( coord_borde_derecha[0][0] - coord_borde_derecha[1]\
[0] ) / ( coord_borde_derecha[0][1] - coord_borde_derecha[1][1] )
    C_der = - (coord_borde_derecha[0][0] + coord_borde_derecha[0][1] * \
B_der)
    B_der_amp = 0
    C_der_amp = 0
    if not coord_der == []:
        B_der_amp = - float( coord_der[0][0] - coord_der[1][0] ) / ( co\
ord_der[0][1] - coord_der[1][1] )
        C_der_amp = - (coord_der[0][0] + coord_der[0][1] * B_der_amp)
    p0 = coord_borde_entrada[0]
    p1 = coord_borde_entrada[1]
    centro_borde_entra = ( ( p0[0] + p1[0])/ 2.0, ( p0[1] + p1[1] )/2.0\
 )
    p0 = coord_borde_frente[0]
    p1 = coord_borde_frente[1]
    centro_borde_frente = ( ( p0[0] + p1[0])/ 2.0, ( p0[1] + p1[1] )/2.\
0 )
    dis_entra_frente = math.sqrt( ( centro_borde_entra[0] - centro_bord\
e_frente[0] ) ** 2 + ( centro_borde_entra[1] - centro_borde_frente[1] )\
 ** 2 )
    p0 = coord_borde_izquierda[0]
    p1 = coord_borde_izquierda[1]
    centro_borde_izq = ( ( p0[0] + p1[0])/ 2.0, ( p0[1] + p1[1] )/2.0 )\

    p0 = coord_borde_derecha[0]
    p1 = coord_borde_derecha[1]
    centro_borde_der = ( ( p0[0] + p1[0])/ 2.0, ( p0[1] + p1[1] )/2.0 )\

    dis_izq_der = math.sqrt( ( centro_borde_izq[0] - centro_borde_der[0\
] ) ** 2 + ( centro_borde_izq[1] - centro_borde_der[1] ) ** 2 )
    if not coord_der == []:
        if coord_borde_entrada[0] in coord_borde_derecha:
            p_entra_der = coord_borde_entrada[0]
        else:
            p_entra_der = coord_borde_entrada[1]
        p_der1 = [ p for p in coord_borde_derecha if not p == p_entra_d\
er ][0]
```

```
        cx, cy = (p_entra_der[0] , p_entra_der[1] )
        dis_entra_der = math.sqrt( ( p_entra_der[0] - p_der1[0] ) ** 2 \
+ ( p_entra_der[1] - p_der1[1] ) ** 2  )
        dis_entra_der = dis_entra_der * (6.0 / 8 )
        a = B_der ** 2 + 1
        b = 2 * B_der * ( C_der + cx ) - 2 * cy
        c = ( C_der + cx ) ** 2 + cy ** 2 - (dis_entra_der ** 2)
        y_inter = int( ( - b - math.sqrt(b ** 2 - 4 * a * c) ) / ( 2 * \
a ) )
        x_inter = - int( B_der * y_inter + C_der )
        if dis_entra_der < math.sqrt( ( x_inter - p_der1[0] ) ** 2 + ( \
y_inter - p_der1[1] ) ** 2 ):
            y_inter = int( ( - b + math.sqrt(b ** 2 - 4 * a * c) ) / ( \
2 * a ) )
            x_inter = - int( B_der * y_inter + C_der )
        p_der = (x_inter, y_inter)
        dis0 = math.sqrt( ( coord_der[0][0] - p_der[0] ) ** 2 + ( coord\
_der[0][1] - p_der[1] ) ** 2  )
        dis1 = math.sqrt( ( coord_der[1][0] - p_der[0] ) ** 2 + ( coord\
_der[1][1] - p_der[1] ) ** 2  )
        if dis0 <= dis1:
            cv2.line(inters_mask, coord_der[0], p_der, 255, 2)
            umbral_der = dis0
        else:
            cv2.line(inters_mask, coord_der[1], p_der, 255, 2)
            umbral_der = dis1
        cv2.line(inters_mask, coord_der[0], coord_der[1], 255, 10)
        img_floodfill = inters_mask.copy()
        height, width = inters_mask.shape[:2]
        mask = numpy.zeros((height+2, width+2), numpy.uint8)
        cv2.floodFill(img_floodfill, mask, (width-1,height-1),255)
        img_floodfill_inv = cv2.bitwise_not(img_floodfill)
        inters_mask = inters_mask | img_floodfill_inv
        p_der = [ p_der1, p_der]
    umbral_entra    = dis_entra_frente * 0.4
    umbral_frente   = dis_entra_frente * 0.1
    umbral_izq      = dis_izq_der * 0.08
    if not coord_der == []:
        umbral_der  = umbral_der * 0.5
    else:
```

```
        umbral_der      = dis_izq_der * 0.08
    tol_der = math.sqrt( ( coord_borde_izquierda[0][0] - coord_borde_iz\
quierda[1][0] ) ** 2 + \
                         ( coord_borde_izquierda[0][1] - coord_borde_iz\
quierda[1][1] ) ** 2 )
    tol_der = tol_der * 0.73
    resultado_dibuja_umbrales = dibuja_umbrales(coord_inters, coord_der\
, coord_borde_entrada, coord_borde_izquierda, img_backg_int,
                                                B_entra, C_entra, B_fre\
nte, C_frente, B_izq, C_izq, B_der_amp, C_der_amp,
                                                umbral_entra, umbral_fr\
ente, umbral_izq, umbral_der )
    p_umbral, p_umbral_entra, p_umbral_frente, p_umbral_izq, p_umbral_d\
er, B_entra_amp, C_entra_amp, img_backg_int = resultado_dibuja_umbrales\


    limites_inters = [coord, coord_inters, B_entra, C_entra, B_entra_am\
p, C_entra_amp, B_frente, C_frente, B_izq, C_izq, B_der, C_der, B_der_a\
mp, C_der_amp]
    umbrales = [ umbral_entra, umbral_frente, umbral_izq, umbral_der, t\
ol_der ]
    coord_borde = coord_borde_entrada, coord_borde_derecha, p_der, coor\
d_der, p_entra_izq
    return coord_borde, limites_inters, umbrales, p_umbral, inters_mask\
, img_backg_int
```

Referencias Bibliográficas

[1] Lílido N Ramírez. «El parque automotor en la República Bolivariana de Venezuela 1990-2011, estratos medios de la población y elecciones 2012». En: *Mundo Universitario* 10.1 (2012), págs. 38-48.

[2] Ian Thomson y Alberto Bull. *La congestión del tránsito urbano: causas y consecuencias económicas y sociales*. CEPAL, 2001.

[3] Fernando Torres Dugarte. *Manual de vías de comunicación I*. U.C. Valencia: U.C. Publicaciones, 2009.

[4] Tecnoligente. *Aforos Vehiculares*. 2016. URL: http://www.tecnoligente.com/ingenieria-vial/aforos-vehiculares/.

[5] Luis Bañón Blázquez. «Manual de carreteras». En: (2000).

[6] Mauro Maldonado Chan, Rafael Gallegos López, MC Federico López Vázquez, Juan Antonio Sandoval Cortina y Mauricio Cabrera Ríos. «Hacia un sistema automático de aforo vehicular basado en secuencias de video y redes neuronales artificiales». En: *CIENCIA FIC* ().

[7] Windmill Software. *Vehicle Sensing: Ten Technologies to Measure Traffic*. 2016. URL: http://www.windmill.co.uk/monitor/monitor211.pdf.

[8] Rosalía Pérez. «Desarrollo de un sistema de conteo y monitoreo para tráfico vehicular». En: (2012).

[9] Mario Alejandro Prieto Valdés. «Algoritmo para conteo vehicular en tiempo real con base en franjas de interés». En: (2010).

[10] Jorge Gerardo Quesada Pacora. «Algoritmo de estimación del número de elementos móviles en videos digitales orientado a la gestión del tráfico vehicular». En: (2015).

[11] Jairo Pedro Palomeque Vilela. «Análisis del Tráfico vehicular en la Av. La Ferroviaria desde el distribuidor de Tráfico (TREN) hasta la Parroquia El Cambio.» En: (2015).

[12] Department of Transportation. «Project Traffic Forecasting Handbook». En: *State of Florida* (2002).

[13] Effective Date y Superseded Issuances. «HIGHWAY DESIGN MANUAL REVISION NO. 62». En: ().

[14] U.S. Department of Transportation. «Traffic Monitoring Guide». En: *Federal Highway Administration* (2013).

[15] Francisco Carlos Calderón Bocanegra y Germán Enrique Urrego Niño. «[15].» En: *Pontificia Universidad Javeriana, Facultad de Ingenieria, Departamento de Electrónica* (2008).

[16] Angamarca S Silvia R y Ibadango L Fausto R. «Estudio de tráfico y soluciones en las intersecciones: Avenida Universitaria - Eustorgio Salgado y Eustorgio Salgado - Bolivia de la ciudad de Quito». En: *Universidad Central de Ecuador, Facultad de Ingenieria, Ciencias Físicas y Matemática, Carrera* (2014).

[17] M. C. José Jaime Esqueda Elizondo. «Fundamentos de Procesamiento de Imágenes"». En: *Universidad Autónoma de Baja California, Unidad Tijuana, CONATEC 2002* (NOVIEMBRE 2002).