UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de
Ingeniería y Sistemas de Telecomunicación

PROYECTO FIN DE GRADO

DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN DIRIGIDA AL CONTROL DEL TRÁNSITO VEHICULAR QUE GENERE REALIDAD AUMENTADA

DANIEL RODRÍGUEZ VENTURA

Grado en Ingeniería de Imagen y Sonido
Junio 2016

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Diseño e implementación de una aplicación dirigida al control del tránsito vehicular que genere realidad aumentada

AUTOR: Daniel Rodríguez Ventura

TUTOR (o Director en su caso): Juan M. Meneses Chaus

TITULACIÓN: Grado en Ingeniería de Sonido e Imagen

DEPARTAMENTO: Departamento de Ingeniería Telemática y Electrónica

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: Martina Eckert

VOCAL: Juan M. Meneses

SECRETARIO: José Fernán Martínez

Fecha de lectura: 13 de Junio de 2016

Calificación:

El Secretario,

A la memoria de mi abuelo Narciso,

Para que siga estando orgulloso de su nieto

Agradecimientos

Este proyecto pone el broche final no sólo a una etapa académica, sino también a una etapa en mi vida. El camino hasta aquí ha sido duro y lleno de momentos de todos los tipos, tanto difíciles como felices. Por eso, quisiera agradecer con estas líneas a todas estas personas que me han brindado todo su apoyo:

A mi madre, por ser la razón y el motivo de mucho de todo lo que soy ahora, es a ella a quién la debo todo. A mi padre, por su apoyo, su visión y sus consejos que tanto me han servido. A mi hermana, por su infinita comprensión y su apoyo incondicional. A mis abuelos por dárnoslo todo cuando más falta nos hacía.

A mi otra familia, mis amigos, los que siempre están ahí: Iván, Mateo, Rebeca, Noemí y Sandra. Por su lealtad, comprensión e infinita paciencia. A Sara, por su cariño, su incansable ánimo, su honestidad y por toda la confianza que ha puesto en mí cuando yo no la he tenido.

A Javier, David y Eduardo, por ser grandes amigos más allá de los muros de la universidad. A Irene, por su paciencia y por todo lo vivido. A los compañeros de la asociación de Kulturales, tanto a los antiguos (Diego, Abdón, Almudena, Christian y Elena) como a los menos antiguos (Stuart, Yannick, David y Leny), por todas las alegrías y buenos momentos vividos a lo largo de mi paso por la carrera.

A mis tutores Henry y Juan, por darme la oportunidad de realizar este proyecto. A todos los profesores que me impartieron clase y que, por una u otra razón, me han dado los conocimientos y los motivos para llegar a donde estoy hoy en día.

Pido perdón por adelantado, porque seguro que alguien me he dejado.

Gracias a todos.

“Tremble, little lion man,

You'll never settle any of your scores"

(Little lion man – Mumford and Sons)

Resumen

Este proyecto realiza el diseño y la implementación de una aplicación que efectúa el control del tránsito vehicular generando realidad aumentada. La aplicación se comunica con el usuario generando y sobreponiendo elementos virtuales a las imágenes del tránsito vehicular aportando la información necesaria para cada caso de control y detección. Aparte de detectar y efectuar el seguimiento de los vehículos en una vía, el usuario puede elegir entre realizar el control de distintas situaciones en ambientes de tráfico como la invasión de algún vehículo en una zona, la monitorización de la velocidad de los vehículos, el control de estacionamiento y la entrada y salida de los mismos en una zona determinada, todo ello en tiempo real, ambiente diurno y desde una cámara estática. Además a través de la interfaz gráfica y el uso del teclado y el ratón el usuario puede configurar zonas de control o elegir la toma de imágenes desde un vídeo de una carpeta o de un dispositivo externo.

El desarrollo y la implementación de la aplicación se han realizado bajo el sistema operativo Windows en lenguaje C++ y con las librerías de OpenCV para visión artificial y realidad aumentada. La interfaz gráfica se ha diseñado e integrado con el software Qt Creator.

Este trabajo fue desarrollado en CITSEM, dentro de la línea de investigación de sistemas de visión por ordenador y realidad aumentada, llevada a cabo por Henry Cruz.

Palabras clave: realidad aumentada, visión por computador, visión artificial, detección de vehículos, control de tráfico, sustracción de background, tracking por modelos de apariencia, C++, OpenCV, Qt Creator.

Abstract

This project is about the design and the implementation of an application which performs the control of the vehicular traffic generating augmented reality. The application communicates with the user generating and overlaying virtual elements to traffic images providing needed information for every case of control and detection. Besides of detect and perform the tracking of the vehicles in a road, the user can choose between control the diverse situations in traffic environment like invasion of a vehicle in an zona, monitoring of the speed of the vehicles, controlling of parking and entry and exit of them in delimited zone, all in real time, diurnal environment and from a static camera. Through the interface user and the use of keyboard and mouse, the user can configure control area or choose shooting from a video in a folder or from an external device.

Development and implementation of the application have been done under the Windows operative system in C++ language and with OpenCV libraries for artificial vision and augmented reality. The user interface has been designed and integrated with Qt Creator software.

realidad aumentada, visión por computador, visión artificial, detección de vehículos, control de tráfico, sustracción de background, tracking por modelos de apariencia, C++, OpenCV, Qt Creator.

This Project was developed in CITSEM, within the line of research of computer vision systems and augmented reality.

Keywords: Augmented reality, computer Vision, artificial vision, vehicle detection, traffic control, background subtraction, tracking with models of appearance, C++, OpenCV, Qt Creator.

Índice de contenido

Índice de figuras

Índice de tablas

1. Introducción

1.1 Introducción

Desde el principio de los tiempos el ser humano se ha preocupado de la seguridad de él mismo y de sus allegados, hasta el punto de hacerla algo indispensable en su día a día. Los avances en seguridad y vigilancia han ido estrechamente ligados a los avances tecnológicos a lo largo del tiempo. Hoy en día, el auge de las tecnologías de Internet en conjunción con dispositivos y cámaras cada vez de menor tamaño y con mayores capacidades ha dado lugar a un estado de vigilancia casi constante.

En el campo del automovilismo y el control de tráfico la vigilancia y la seguridad resulta fundamental a la hora de afianzar la circulación por las vías. En España se dispone de la Ley Orgánica 4/1997 junto a su correspondiente "Reglamento de Desarrollo" [1]. En ambos documentos se estipula que el uso de videocámaras se puede contemplar desde muchos puntos de vista, ya sea el control vehicular en carretera o la monitorización de accidentes o situaciones anómalas en ella. La Dirección General de Tráfico (D.G.T), desde sus centros, monitoriza de forma constante las principales vías de tráfico y las grandes autopistas nacionales. En ambientes más urbanos, la monitorización del flujo vehicular se realiza desde los propios ayuntamientos de cada población siendo los sistemas de circuito cerrado de televisión (CCTV) los más preferidos en estos casos. En CCTV es posible, además, utilizar varios tipos de cámara dependiendo de la necesidad de cada caso. Gracias a los dispositivos de vigilancia tales como cámaras y monitores, se puede obtener información en tiempo real de los estados de las carreteras en muchos puntos clave del país, como por ejemplo saber si hay retenciones o atascos en un tramo.

Una vuelta de tuerca a estas tareas de vigilancia de tráfico sería que los dispositivos fueran capaces de reconocer anomalías e incidencias específicas en una vía como atropellos, accidentes o acceso a vías prohibidas. La visión artificial junto a la realidad aumentada permite alcanzar estos objetivos. Gracias a las áreas de estas tecnologías, un sistema básico como es una cámara junto a un procesador puede recopilar información de un entorno de tráfico (como son los vehículos, los peatones o señales), procesarlas de una determinada forma y finalmente identificar las diferentes anomalías que puede haber, entiendo anomalía como cualquier comportamiento distinto o fuera de la tónica habitual de un entorno establecido.

La ayuda de estos sistemas, sin duda, puede llegar a ser inestimable y de gran utilidad. No obstante, y pese al constante aumento de las tecnologías de realidad aumentada y visión artificial, no pueden sustituir de forma total la interacción humana en la vigilancia. Esto es debido principalmente al gran espacio de la casuística del medio y a la cantidad de variables no controladas que hay que tener en cuenta. Dependiendo del modo de

funcionamiento de cada sistema, los cambios de iluminación, el movimiento de la cámara o incluso los fenómenos atmosféricos pueden llegar a resultar grandes inconvenientes en la detección provocando errores de todo tipo.

1.2 Objetivos

El objetivo principal es el diseño y la implementación de una aplicación autónoma que realice de varias maneras el control del tránsito vehicular desde una cámara instalada o un vídeo. El control del tráfico se realizará desde la monitorización de varias situaciones y anomalías a reconocer siendo estas:

- Control de invasión de vehículos en una zona determinada.
- Estimación y control de la velocidad de un vehículo.
- Control del estacionamiento de los vehículos en una zona determinada.
- Control de entrada y salida de vehículos con conteo de los mismos en una zona determinada.

Como objetivos parciales para aplacar el objetivo principal se encuentran:

- Estudio del estado del arte para el conocimiento de las tecnologías más eficiente para este caso.
- En base al estudio realizado, elegir y proponer soluciones que cumplan los objetivos.
- Estudio del uso y aplicaciones de los métodos HOG, SVM y Flujo Óptico para la detección y seguimiento vehicular.
- Desarrollo de un algoritmo basado conjuntamente en los métodos HOG, SVM y Flujo Óptico que permita el cumplimiento de los objetivos y requisitos.
- Estudio del uso y aplicaciones de los métodos de Sustracción de Background y Tracking por Modelos de Siluetas para la detección y seguimiento vehicular.
- Desarrollo de un algoritmo basado conjuntamente en los métodos de Sustracción de Background y Tracking por Modelos de Siluetas que permita el cumplimiento de los objetivos y requisitos.
- Elección de la solución más eficiente de entre las propuestas.
- Implementación de la solución elegida en código C++ de los algoritmos estudiados.
- Realización de pruebas unitarias para comprobar la eficiencia de la aplicación.

1.3 Organización de los capítulos

El presente documento se encuentra organizado en 6 capítulos principales. El primero de ellos define los objetivos del proyecto junto a una introducción y la justificación del mismo. El segundo capítulo explica el estado del arte, donde se habla de los antecedentes de este proyecto y se detalla las principales técnicas y algoritmos que se han utilizado. En el capítulo tres se presenta las soluciones propuestas y el entorno de desarrollo, donde se detalla el proceso del mismo y la implementación de la solución elegida basada en el estudio del arte del capítulo dos. En el capítulo cuatro se realizan la experimentación y el proceso de pruebas de la solución elegida. En el capítulo cinco se muestran las conclusiones finales y se arrojan también posibles líneas de trabajo futuro basadas en la misma idea. Finalmente, en el último capítulo se realiza una lista de los trabajos e informaciones consultadas para la realización de este proyecto, además de una serie de anexos.

2. Estado del arte

En este capítulo se pretende dar a conocer el estado actual de la temática, es decir, el control del tráfico de vehículos con Realidad Aumentada y visión artificial, así como las bases teóricas sobre las que se ha sustentado hasta ahora. Esto abarca desde los últimos desarrollos tecnológicos hasta los conceptos más fundamentales y necesarios. Por tanto, se trata de una colección crítica de métodos, fuentes y experiencias sobre el tema que justifica la motivación de este proyecto.

2.1 Introducción a la visión artificial y realidad aumentada

Se define **Realidad Aumentada (RA)** como el compendio de tecnologías que permiten añadir un extra de información virtual al mundo real mediante técnicas visuales como la visión artificial y dispositivos o display electrónicos. El objetivo de la Realidad Aumentada es que el usuario no deje de visualizar el mundo real mientras los elementos virtuales se sobreponen de forma interactiva a los reales aportando información diversa y de cualquier tipo. De esa manera el mundo virtual se combina con el real dando lugar a una "realidad mezclada". Por ejemplo, mediante un display o una aplicación de teléfono la Realidad Aumentada puede proveer a una persona de información sobre el precio o las características de un producto mientras está en una tienda. Estos precios y características se mostrarían de forma añadida a la realidad pero nunca fuera de ella.

Para el objetivo perseguido en este proyecto (control de tráfico por vídeo) la Realidad Aumentada supone una herramienta potente y con muchas posibilidades. Esta técnica trabaja conjuntamente con la multitud de métodos y técnicas de visión artificial. Esta última se encarga de recabar todos los datos e informaciones de la imagen mientras que la Realidad Aumentada analiza, adecúa y muestra todos esos datos e informaciones para el usuario. Son muchas las aplicaciones dedicadas al tráfico que utilizan Realidad Aumentada ya sea para su control o para la asistencia del conductor.

Por otro lado, y tomando como referencia las bibliografías [2] y [3], se conoce la **visión artificial (o visión por computador)** como un subcampo de la inteligencia artificial donde se trata de obtener, mediante métodos y técnicas automáticas, información visual de un entorno real con el fin de extraer ciertas características de interés. Dicho de otra manera, el objetivo de la visión artificial es que un sistema sea capaz de reconocer y "entender" la información que está captando.

Estos métodos y técnicas pueden ser numerosos y de gran variedad, y en ello radica en gran medida la funcionalidad y la eficiencia de la extracción de la información del

entorno. De esa forma, no se utilizarán los mismos recursos ni de la misma manera para acometer dos tareas de objetivos distintos, ya que el nivel de complejidad entre ella puede ser muy diferente. Pero aunque haya sistemas de naturalezas muy distintas, todos pueden responder a una estructura general común basada en módulos de alto nivel. Esta estructura modular puede ser:

- **Adquisición**. El sistema puede captar la información visual de muchas maneras posibles. Esto abarca desde fuentes más sencillas como puede ser una imagen estática como una fotografía hasta dispositivos más complejos como cámaras de alta resolución o de cualquier otro tipo de sensor. El resultado puede ser cualquier tipo de imagen en base al tipo de imagen adquirida previamente. Por ser el primer paso, este módulo tendrá cierta importancia ya que de él se obtendrán los datos que se procesaran en posteriores módulos. Interesa, por tanto, que estas imágenes tenga un mínimo de calidad dependiendo de la tarea.

- **Preprocesado**. Una vez obtenidas las imágenes, estas necesitan ser debidamente preparadas a fin de adecuarlas a los métodos de procesados posteriores. Este preprocesamiento se relaciona con tareas como, por ejemplo, mover a otro espacio de color las imágenes captadas (por ejemplo, de RGB a YCbCr), realizar una re-digitalización (*resampling*), normalizar los niveles de señal de acuerdo a normas, adecuar las medidas a unas preestablecidas por los métodos, etc…

- **Extracción de características**. El sistema de visión artificial, mediante múltiples operaciones y procesos, ha de extraer las características más relevantes de todos los datos de imagen anteriormente adquiridos. Estas operaciones abarcan desde la extracción de líneas, aristas, bordes, esquinas, manchas (*blobs*), formas circulares, o características relevantes a nivel de forma o textura como puede ser la extracción de gradientes o de descriptores de imagen tipo HOG (*Histograms of Gradients*) que más adelante se estudiará.

- **Detección de las características**. Una vez extraídas todas las características posibles del flujo de datos, es necesario elegir sólo aquellas que sean válidas para el objetivo del sistema. Aquí se engloban tareas como la segmentación de regiones que cumplan ciertos requisitos o el filtrado y etiquetación de puntos interés. Este módulo también realiza acciones como la estimación de modelos, entrenamiento de sistemas de decisión con muestras positivas/negativas, la clasificación o comparación de las características filtradas anteriormente, y en general, la verificación de los parámetros y resultados del sistema.

- **Toma de decisiones**. Como paso final, el sistema de visión artificial debe tomar una decisión de acuerdo a la información extraída y a la interpretación realizada de la misma. Así, el resultado arrojado por el procesamiento de alto nivel entrará en este módulo para dar lugar a una respuesta final que cumpla con el objetivo para el que se diseñó el sistema. Esta respuesta final puede ser desde una clasificación en varias clases, al posible reconocimiento o no de un objeto en un vídeo o una imagen.

El rango de aplicaciones de la Realidad Aumentada y de la visión artificial es muy extenso y crece a un ritmo cada vez mayor. Además, los usos pueden ser de muy diversos tipos. Se pueden encontrar aplicaciones en el campo de la medicina (reconocimiento para el diagnóstico de enfermedades mediante imágenes médicas), robótica (ayuda a la navegación del sistema autónomo), diseño y montaje industrial (montaje guiado de máquinas y procesos industriales), reconocimiento y clasificación de múltiples objetos (reconocimiento de caracteres escritos, matrículas de vehículos, reconocimiento facial, etc…) o en aplicaciones de calidad (ayuda en los controles de inspección y verificación de un proceso). Estas sólo son algunas de las aplicaciones donde se puede encontrar sistemas de Realidad Aumentada y visión artificial ya implantados. A día de hoy ha aumentado notablemente el uso conjunto en aplicaciones de consumo como videojuegos o aplicaciones en móviles y tablets.

2.1.1 Métodos representativos

En aplicaciones de Realidad Aumentada y visión artificial es muy común encontrar sistemas de detección e identificación de objetos o personas en diversos ambientes o entornos. Dado que un sistema puede tener requerimientos muy distintos de otros dependiendo de su objetivo, las especificaciones y los requisitos variarán. Es posible que sea necesario que la clasificación se realice en tiempo real, lo que implica la optimización de los tiempos de ejecución al mínimo. También puede que sea necesaria cierta eficiencia, lo que implica tener un algoritmo fuerte y robusto con poca tasa de error. O por otro lado que sea fácilmente escalable y adaptable, lo que significa que los métodos y técnicas utilizados han de ser de fácil modificación y actualización. En este caso, para aplicaciones de detección y control de tráfico se requiere poca tasa de error (bajo número de falsos positivos y falsos negativos) y adaptabilidad del sistema (entornos diurnos, con poca visibilidad, con mala toma de imágenes, evitar sensibilidad a la variación de la luminosidad, etc...). Respecto al requisito de detección en tiempo real dependerá de la tipología y la funcionalidad del sistema, pero siempre es un enfoque deseable sin comprometer demasiado al resto de las funcionalidades.

Dado el planteamiento del proyecto, se ha investigado en las técnicas y sistemas de reconocimiento e identificación de vehículos en entornos vehiculares ya instauradas y en uso. Como era de esperar, se han encontrado numerosas alternativas. Aun así, cabe destacar el gran parecido en la estructura general de las aplicaciones encontradas, diferenciándose entre ellas únicamente por los métodos de extracción y detección de características y por los métodos de toma de decisiones en la fase final.

Entre las principales técnicas y algoritmos de extracción, detección y clasificación de características y seguimiento de objetos en imágenes y vídeos se han encontrado las siguientes propuestas válidas para el propósito del proyecto:

- Sustracción de background:
  - Resta de frames.
  - Filtro de mediana (*Median Filter)*.
  - Mezclas de Gaussianas (*Gaussian Mixtures*).
- Detección de puntos característicos:
  - Detección de esquinas: Detector de Harris y Shi. Tomasi.
- Extracción de características del contorno y formas:
  - Extracción de gradientes: Operador Siebel.
  - Extracción de contornos: Descriptores HOG.
- Clasificación de características
  - Modelos de clasificación supervisada: *Support Vector Machine* (SVM)
- Detección de movimiento y tracking:
  - Estimación del movimiento por flujo óptico.
  - Seguimiento de modelos de contorno y silueta.

Entrando ya en aplicaciones funcionales comerciales, se pueden encontrar las siguientes propuestas:

- *BirdWatch Red Light* [4], de la empresa Quercus, es un sistema de visión artificial compuesto de una cámara, un procesador y un módulo de comunicaciones que permite la detección de los vehículos que han violado la prohibición de paso en los semáforos rojos. Funciona mediante algoritmos de sustracción de background con detección de sombras. El sistema se adapta tanto a luz natural como a ambientes nocturnos y es fácilmente instalable en muchos escenarios dado su reducido tamaño.

Figura 1. Sistema *BirdWatch* [4].

- *BusVigía* [5], es un proyecto conjunto de la Universidad Rey Juan Carlos y la empresa de transporte público E.M.T. Trata de un sistema de visión artificial que vigila la invasión del carril bus basado en la observación de varias cámaras instaladas en un bus que lo recorre. El sistema avisa en caso de que el carril bus haya sido invadido por vehículos y si además, dichos vehículos circulan de forma prohibida. Permite también el reconocimiento del vehículo infractor para su posterior denuncia.

- *ASSET (Advanced Safety and Driver Support for Essential Road Transport)* [6], es un proyecto en el que han participado más de 19 organizaciones y 12 países cuyo fin es el control de las incidencias del tráfico, como apunta la referencia [7]. Se trata de un sofisticado sistema que puede realizar numeras detecciones de infracciones realizadas con la circulación indebida en el tráfico. Esto abarca desde el exceso de la velocidad, pasando por la distancia de seguridad entre vehículos hasta llegar a la detección de si el conductor está usando el cinturón de seguridad mientras conduce.

- La empresa española *Ingartek* diseñó un sistema basado en visión artificial que realiza el conteo de vehículos de forma no intrusiva para la Diputación de Vizcaya [8]. El sistema cuenta a su disposición con los datos de una serie de cámaras dispuestas por los puntos clave y vías más importantes. Mediante el procesamiento de dichas imágenes se hallan los flujos de tráfico en cada calle y los estados de las vías.

- La empresa *Invar* ha desarrollado sendos sistemas referidos al control de tráfico y la seguridad vial mediante visión artificial [9]. El primero de ellos es *Invar SAAT* y permite registrar la actividad de una vía obteniendo parámetros de ella tales como el nivel de saturación de vehículos, la clasificación del tipo de vehículos o

el cálculo de la velocidad mediante la estereoscopia. El segundo de ellos es *Invar SV2*, y se trata un de sistema que va montado dentro del propio vehículo para la mejora de la seguridad vial del mismo y que, mediante algoritmos de visión artificial y realidad aumentada, realiza tareas como el reconocimiento de señales tanto verticales como en la calzada o el cálculo de la velocidad y la dirección del vehículo.

Figura 2. Sistema *Invar SV2* [9]

2.2 Extracción y detección de características

En este apartado se explicarán los métodos de extracción y detección de características más relevantes para la realización del proyecto.

2.2.1 Detectores de esquina

Ciertos objetos son fácilmente reconocibles por sus formas y contornos, y pueden ser fácilmente diferenciados del resto por sus esquinas más características. Una esquina es un punto interesante a tener en cuenta, ya que se trata de la intersección entre dos aristas y su valor en el punto al calcular el gradiente de la luminancia de la imagen es moderadamente alto. Teniendo en cuenta esto, en visión artificial suele ser muy corriente el uso de los detectores de esquina para extraer ciertas características de algunos objetos ya segmentados, con el fin de utilizar esta información más tarde en otras tareas, como por ejemplo, el seguimiento del mismo objeto. Uno de los detectores de esquinas más conocidos es el **detector de Harris** [10], y su versión mejorada por **Shi y Tomasi** [11].

El detector de Harris se basa en la búsqueda de lo anteriormente comentado: las esquinas representan un valor alto en el gradiente de la luminancia de la imagen. Por consiguiente, se recorrerá la imagen I mediante una ventana W que se desplaza en vertical u y horizontal v para hallar las diferencias de intensidad, tal y como describe en la ecuación (1), siendo E la diferencia de intensidad hallada.

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u,y+v) - I(x,y)]^2 \qquad (1)$$

Aplicando el desarrollo de Taylor en ambos lados de la ecuación y desarrollando se obtienen las ecuaciones (2) y (3).

$$E(u,v) \approx \sum_{x,y} \left[I(x,y) + uI_x + vI_y - I(x,y)\right]^2 \qquad (2)$$

$$E(u,v) \approx \sum_{x,y} u^2 I_x^2 + 2uvI_xI_y + v^2 I_y^2 \qquad (3)$$

Que en forma matricial quedarían tal y como se muestra en la ecuación (4).

$$E(u,v) \approx [\, u \;\; v \,] \left(\sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix} \qquad (4)$$

Siendo *M* la ecuación (5).

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \qquad (5)$$

Como el cálculo se realiza para cada ventana, para determinar si dentro de cada una puede haber una esquina se recurre al parámetro R, definido por la ecuación (6).

$$R = \det(M) - k\big(trace(M)\big)^2 \qquad (6)$$

Cada componente dela ecuación (6) se desarrolla en las ecuaciones (7).

$$\det(M) = \lambda_1\lambda_2 \qquad \text{trace}(M) = \lambda_1 + \lambda_2 \qquad (7)$$

Finalmente la ecuación (6) se puede reescribir como (8) partiendo de las ecuaciones (7):

$$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2 \qquad (8)$$

Siendo λ_1 y λ_2 los autovalores de la matriz M. Si el valor absoluto de R es pequeño, λ_1 y λ_2 tendrán valores pequeños y la región será plana, sin aristas ni esquinas (zona gris en figura 3 izquierda). Si R es menor que 0, entonces λ_1 será mayor que λ_2 (o viceversa), lo que significará que la ventana está englobando a una arista (zona naranja en figura 3, izquierda). Si R tiene valores altos, entonces λ_1 y λ_2 también serán altos y de valores similares, lo que supone que la ventana engloba una esquina (zona verde en figura 3,

izquierda). Este concepto se puede observar de manera visual en la figura 3, izquierda, donde se ha representado la función λ_2 - λ_1 .

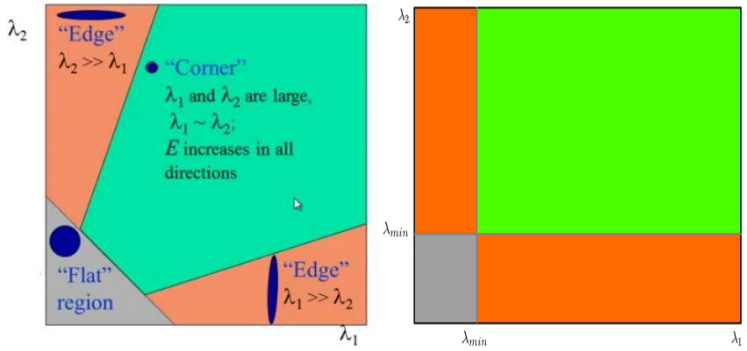


Figura 3. Izquierda, Detector de Harris [12]. Derecha, Detector Shi-Tomasi [12].

Shi y Tomasi propusieron en 1994 en la publicación [11] una modificación en la ecuación de R (ecuación 9) con mejores resultados.

$$R = \min(\lambda_1, \lambda_2) \qquad (9)$$

En este caso, si R es mayor que un umbral previamente definido, se considerará esquina. De manera visual, en la figura 3 derecha se ha representado de nuevo la función λ_2 - λ_1 para este caso, siendo el área en verde la que se corresponde con la detección de una esquina.

2.2.2 HOG

Conceptos previos. Gradiente

Se define gradiente de un campo escalar como el vector que indica la dirección de variación máxima del escalar, siendo el módulo del vector el ritmo de dicha variación [13].

El campo escalar puede ser, por ejemplo, la temperatura de una habitación o la luminancia de una imagen. Por tanto, el concepto de gradiente se puede aplicar, dados estos ejemplos, a encontrar la dirección y el ritmo de máxima variación de temperatura en la habitación en el primer caso, o la dirección y ritmo de máxima variación de la luminancia de una imagen en el segundo.

Matemáticamente, el gradiente se define como el campo vectorial cuyas funciones coordenadas son las derivadas parciales del campo escalar respecto a sus coordenadas,

tal y como se describe en la ecuación (10), donde x son las diferentes coordenadas del campo escalar f.

$$\nabla f(\boldsymbol{r}) = \left(\frac{\partial f(\boldsymbol{r})}{\partial x_1}, \ldots, \frac{\partial f(\boldsymbol{r})}{\partial x_n}\right) \quad (10)$$

Más concretamente, para un sistema de coordenadas cartesiano (x, y, z) la expresión del gradiente, ecuación (11), se definiría como las derivadas parciales del campo escalar respecto a cada una de sus componentes en x, y, z.

$$\nabla f = \frac{\partial f}{\partial x}\,\hat{x} + \frac{\partial f}{\partial y}\,\hat{y} + \frac{\partial f}{\partial z}\,\hat{z} \quad (11)$$

Como detalle, se debe observar que el vector gradiente siempre será perpendicular a las líneas equiescalares (o contornos) del campo escalar en cuestión, ya que en los 'bordes' del campo escalar se encontrarán variaciones abruptas de la magnitud y, por tanto, la dirección del gradiente será perpendicular a dicho contorno o 'borde', tal y como se ilustra en la figura 4.

Figura 4. Ejemplo de la dirección y sentido del vector gradiente. Fuente propia.

Matemáticamente, siendo $I(x, y)$ la luminancia o intensidad de una imagen, donde x es la coordenada horizontal e y la coordenada vertical, el gradiente para cada punto (o píxel) daría como resultado un vector de dos dimensiones con componentes dadas por las primeras derivadas en las direcciones vertical y horizontal. Cada vector apuntaría a la dirección donde la intensidad crece de manera más rápida y su módulo representaría la cuantificación o cantidad de ese incremento.

En lo referente a visión artificial, se utilizará el gradiente para hallar las direcciones y los lugares donde la variación de luminancia de una imagen es rápida. Esta variación es grande sobre todo en bordes y aristas. Por lo tanto, realizar el gradiente sobre una imagen en luminancia, tanto en dirección horizontal y vertical, dará como resultado los bordes y aristas de dicha imagen.

En la figura 5 se ilustra un ejemplo de aplicación del gradiente primero en dirección vertical, después horizontal y finalmente en ambas. Nótese como al calcular el gradiente

en dirección horizontal sólo se remarcan los cambios de luminancia en dicha orientación obviándose los cambios en dirección vertical.

Figura 5. Ejemplos del cálculo del gradiente de una imagen (1ª imagen) en distintas direcciones (imágenes 2ª y 3ª) y en todas a la vez (imagen 4ª).

HOG: Histograma de orientaciones de gradientes

Se trata de una técnica de extracción de características que proporciona un descriptor de la imagen y sus contornos a través del cálculo de los histogramas de las orientaciones de los gradientes [14].

HOG: Cálculo de histograma en una celda

Para hallar el descriptor global de la imagen, previamente se dividirá esta en celdas de tamaño fijo y cuadrado. Para cada una de estas celdas se obtendrá un histograma de las orientaciones de los gradientes calculados dentro de la misma. Estas orientaciones se dispondrán en un histograma que ordenará las orientaciones de los gradientes de la celda por frecuencia de aparición. En el descriptor global, se combinarán todos los histogramas de orientaciones hallados de todas las celdas, como se muestra en la figura 6.

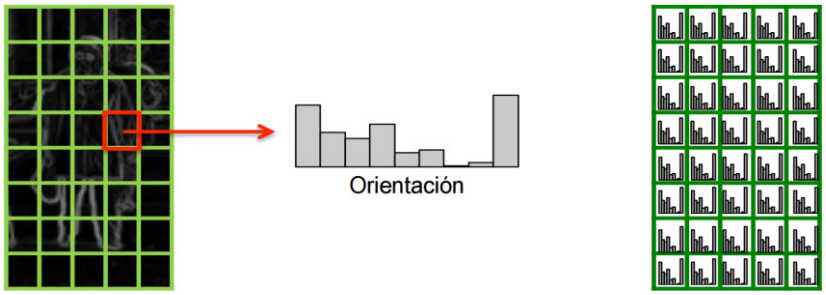


Figura 6. División de una imagen en celdas e histogramas de orientaciones del gradiente calculados en cada celda [15].

En el cálculo del histograma de una sola celda hay que tener en cuenta varios aspectos. El primero de ellos es el tamaño en píxeles de la celda "*Cell Size*". Por otro lado, será importante tener en cuenta como se dividirá el rango de orientaciones en intervalos fijos

("*Number Bins*"). Cada gradiente calculado en la celda tendrá su propia orientación, que tomará valores entre 0 y 360 grados. Este rango de orientación se dividirá en intervalos iguales entre ellos. Por razones de simplicidad, se puede reducir el rango a únicamente 180 grados, y de ahí luego dividir el rango en 9 intervalos de 20 grados como se muestra en la figura 7.

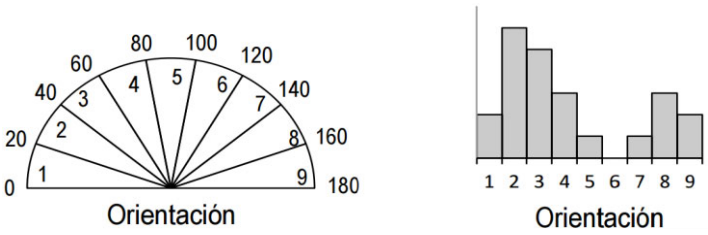


Figura 7. A la derecha el rango de orientaciones dividido en 9 partes. A la izquierda, la composición de un histograma a partir de las orientaciones del gradiente correspondientes a cada partición [15].

Con 180 grados será suficiente para definir todas las orientaciones, ya que dos gradientes en la misma dirección pero en sentido contrarios quedarían asignados al mismo intervalo. En la figura 8 se pueden observar las orientaciones calculadas en una celda de la imagen que se muestra. En azul claro se han recuadrado dos gradientes con la misma dirección pero sentido contrario. Estos dos gradientes se asignarían al mismo intervalo, que a la vista de la anterior figura 7, correspondería el intervalo número 7.

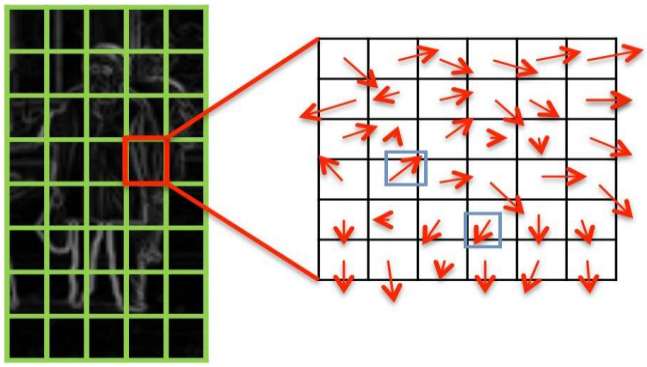


Figura 8. Cómputo de direcciones de los gradientes de una celda. Recuadrados en azul dos gradientes con la misma dirección pero distinto sentido [15].

HOG: Cálculo del descriptor final

Una vez obtenidos los histogramas de las orientaciones de cada celda, se continuará con el cálculo del vector descriptor final y que representará los contornos de la imagen.

Dado que es preferible obtener cierto grado de varianza en las orientaciones de los gradientes calculados para tener en cuenta todas las posibles variaciones en una imagen, es probable encontrarse con ciertas dificultades en los datos de entrada. Estas

adversidades pueden ser desde cambios en la iluminación entre imágenes consecutivas hasta variaciones en el tamaño de la misma. Por lo tanto, surge la necesidad de normalizar el cálculo del descriptor para minimizar estos inconvenientes. La normalización será más eficiente si se realiza de forma local y no de manera global en toda la imagen. Aparece, por ello, la idea de dividir la imagen en bloques.

Un bloque es agrupación de varias celdas unidas. Los bloques deben tomar tamaños cuadrados de *bxb* celdas. Una vez dividida la imagen en bloques, se podrá obtener la representación en vector de un solo bloque concatenando los histogramas de las celdas que lo componen, tal y como aparece en la figura 9.

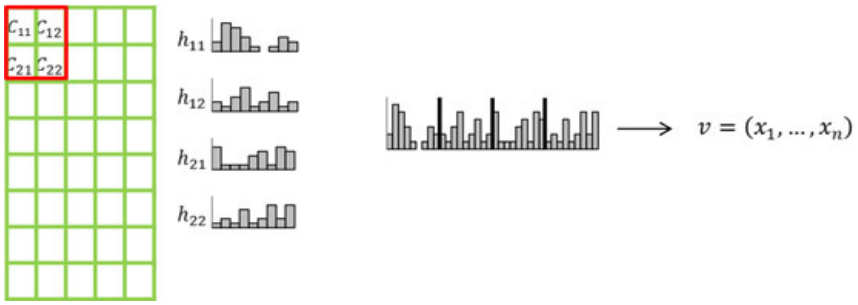


Figura 9. Concatenación de los histogramas de una celda en un vector [15].

Será esta representación en vector lo que se normalizará a nivel de bloque. La normalización se realizará utilizando la norma L2 (o norma Euclídea), por la cual se realiza la raíz cuadrada de suma todas las componentes del vector al cuadrado, tal y como se muestra en la ecuación (13).

$$\|v\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2} \qquad (13)$$

El objetivo de esta operación a nivel de bloque será reducir las diferencias en descriptores finales de imágenes muy similares o parecidas entre ellas a fin de obtener un descriptor más robusto ante ciertos cambios.

El descriptor final HOG se conseguirá concatenando todos los histogramas de cada bloque (figura 10). Estos histogramas de cada bloque serán a su vez los histogramas normalizados de cada celda. Habrá, además, cierto solapamiento entre los bloques, lo que ayuda a la robustez del descriptor final.

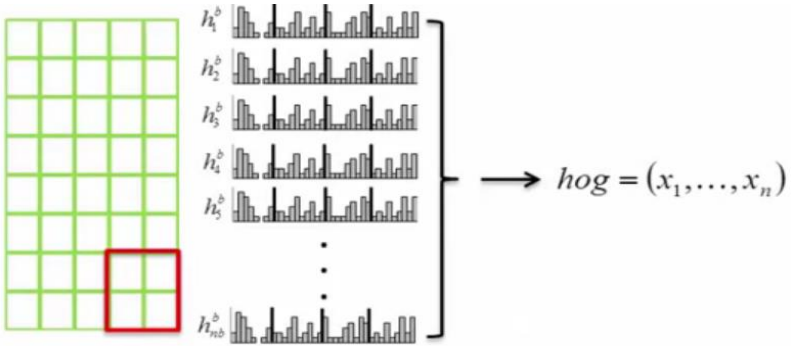


Figura 10. Concatenación de los histogramas de los bloques en un vector descriptor final [15].

Una vez calculado el descriptor, y a modo de resumen, este contará con varios parámetros característicos que lo definen:

- **Tamaño de celda**: Se mide en píxeles y en combinación con el tamaño de la imagen de entrada se obtendrá el número de celdas totales en la imagen.
- **Celdas por bloque**: Indica el número de celdas que compondrá un bloque.
- **Número de intervalos de las orientaciones** y **signo del gradiente**: Si se computan gradientes con signo el rango de orientaciones será de 360º mientras que si no se utiliza sólo será de 180º. El número de intervalos vendrá determinado por la división en partes iguales del rango de orientación.

Por otro lado, el conjunto de todos estos parámetros influirá en el tamaño final del descriptor, es decir, del número de componentes del vector. Este tamaño viene definido por la ecuación (14).

$$n = Nº\ de\ bloques * \frac{Celdas}{Bloque} * Nº\ de\ intervalos \qquad (14)$$

2.2.3 SVM

En *Machine Learning* (en castellano, aprendizaje automático), *Support Vector Machine* (SVM) es un conjunto de modelos discriminativos basados en algoritmos de aprendizaje supervisados utilizados principalmente para tareas de clasificación, detección de casos excepcionales y análisis de regresión [16] [17]. Fue ideado originalmente en 1963 por Vladimir N. Vapnik y Alexey Ya. Chervonenkis [18].

Para un conjunto de muestras o puntos que pertenecen a dos o más tipologías distintas, un modelo de SVM puede definir la pertenencia de cada muestra o punto a cada tipo, utilizando para ello distintos tipos de algoritmos. Para realizar estas operaciones el modelo ha de ser entrenado para que pueda discernir a qué tipología pertenece dado

un punto del conjunto a analizar. De ahí que SVM se trate de un modelo de aprendizaje supervisado.

Partiendo de un espacio dimensional transformado, SVM genera un plano (en caso de dos dimensiones o clases a clasificar) o un hiperplano (en caso de más de dos dimensiones o clases a clasificar) que separa los conjuntos de datos en su tipología dando lugar a la clasificación de cada muestra en su clase. De forma ideal, la clasificación ha alcanzado su éxito cuando se consigue la máxima distancia entre el plano/hiperplano y las muestras de datos más cercanas al mismo, creando un margen entre las muestras de las distintas clases. En otras palabras, la clasificación será eficiente cuando se consiga maximizar el margen que hay entre las muestras de datos más cercanas al plano.

Dado que hay un número de soluciones infinitas a la hora de encontrar un plano, el problema radica en encontrar sólo aquel que cree un margen máximo entre muestras. Las muestras (o vectores) que estén más cercanos al hiperplano son los llamados **Vectores de Soporte** y son las muestras más representativas del modelo.

Como ejemplo sencillo y visual, en la figura 11 se puede ver un modelo SVM para una clasificación de un conjunto de puntos en dos clases: puntos azules y cuadrados rojos. Se observa en la figura los llamados Vectores de Soporte (cuadrados y círculos opacos) y como forman dos líneas paralelas al margen, una para cada clase.

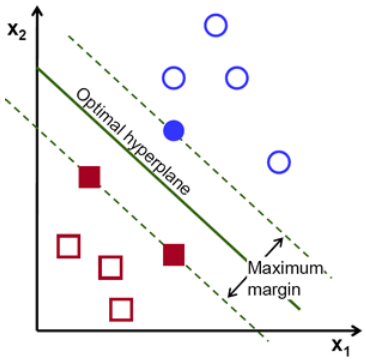


Figura 11. Ejemplo de clasificación por SVM [19].

En un entorno de visión artificial donde se requiere una clasificación de las características previamente extraídas se puede utilizar un modelo SVM binario (o de dos clases). Si hablamos de reconocimiento de personas, las dos clases serán, respectivamente, “persona” y “no-persona. Las muestras tipo “persona” contendrían todas aquellas características que son representativas de una persona, por ejemplo, muestras de gente en diferentes posturas, alturas o contornos. Por tanto, se estaría hablando de muestras positivas para el reconocimiento. Las muestras tipo “no-persona” contendrían lo restante, es decir, todo aquello que no es (ni se parece) a una persona. Esto pueden ser elementos del entorno tales como coches, el pavimento, árboles, etc... Y conformarían lo que se llama muestras negativas.

Si las muestras positivas y negativas son parecidas o mantienen cosas en común, el margen entre el hiperplano y los vectores de soporte será pequeño (ver figura 12) lo que llevará a posibles errores en la clasificación y posteriormente en la detección.

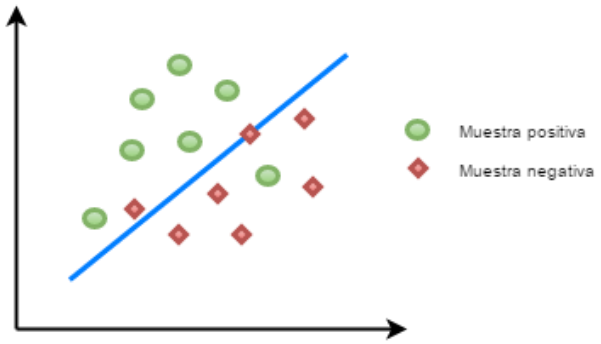


Figura 12. SVM con hiperplano resultado de un mal entrenamiento

Sin embargo, manteniendo la "pureza" de cada tipo de muestra se puede conseguir un hiperplano con un margen maximizado, sin apenas casi posibilidad de error en la clasificación, como se muestra en la figura 13.

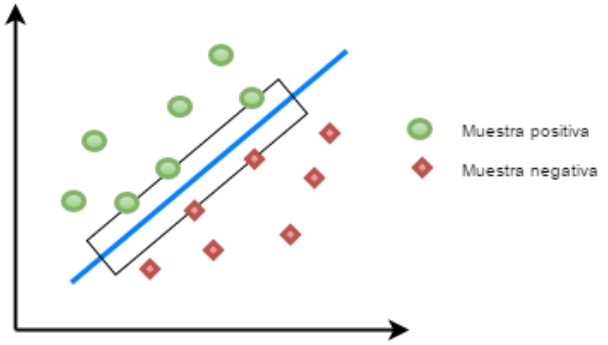


Figura 13. SVM con hiperplano resultado de un buen entrenamiento. En negro se muestra el margen.

2.2.4 Sustracción de Background

Se entiende Sustracción de Background (o sustracción de fondo) como una serie de procesos y algoritmos capaces de estimar y construir una imagen representativa del fondo de la escena que englobe sus elementos más estáticos y recurrentes [20] [21] . Una vez obtenida dicha imagen se realiza una "resta" (o sustracción) entre dicho background y otra nueva imagen con el mismo fondo pero con nuevos elementos añadidos a la escena. Esta diferencia entre imágenes dará como resultado la segmentación de dichos elementos nuevos que se han añadido al fondo previamente sustraído, lo que se conoce como foreground. A esta primera segmentación se la puede

aplicar numerosos procesados (umbralización, supresión de ruido, operaciones morfológicas,…) con el fin de obtener un foreground más definido.

En la figura 14 se puede observar este proceso en forma de diagrama teniendo de ejemplo dos imágenes distintas de una misma carretera. Como se puede apreciar, tras la resta y el procesado, los coches quedan segmentados y diferenciados respecto a la carretera y los árboles (background). El resultado obtenido es lo que se conoce como máscara de foreground y se trata de una imagen binaria donde, normalmente en blanco, suelen aparecer los objetos segmentados del foreground. En el proceso de sustracción de background es muy normal trabajar tanto con imágenes binarias como con imágenes en el espacio colorimétrico RGB. Además, de las imágenes binarias se puede destacar su fácil manejo y su poco coste de computación para realizar las tareas de procesado pertinentes.

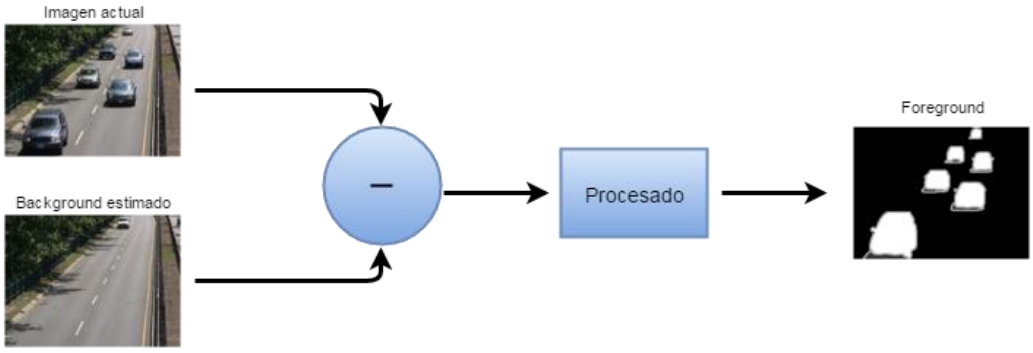


Figura 14. Proceso genérico de sustracción de background.

Esta es la base de cualquier proceso de substracción de background que se precie. La diferencia entre los numerosos y distintos métodos radica en la manera en la cual se extrae el background para la posterior resta. El modo en el que se realice la sustracción definirá en gran parte la calidad de la detección [22] y algunos de ellos se encuentran explicados en los siguientes capítulos de esta.

Por un lado, hay algoritmos que apuestan por estimar un fondo estático y simple teniendo a favor poca la complejidad en el cálculo pero ganando demasiada sensibilidad a pequeños cambios en el background como puede ser el método "*Frame Difference*". Otros se decantan por una actualización constante del mismo cada cierto período del mismo, lo que permite controlar ciertos cambios que varían el background como es el oscurecimiento progresivo de la imagen por la llegada de la noche o la aparición lenta de objetos que se suman al fondo. Estas mejoras suelen traducirse a menudo en altos costes de computación. Las mezclas gaussianas (*Mixtures of Gaussians*) o KDE (*Kernel Density Estimation*) son algoritmos que se encuadran en esta última tipología y son de los más utilizados hoy en día.

Dada la naturaleza de su comportamiento, la substracción de background es utilizada en aplicaciones para el reconocimiento y estimación del movimiento de objetos o personas. Esta característica lo hace muy interesante de cara a tareas de vigilancia por cámara estática de un entorno controlado. Pero también es importante tener en cuenta en qué tipo de aplicaciones es posible implementar esta técnica, y esto depende de su uso final.

Dado que prácticamente en cada frame es necesario realizar la estimación de un background representativo de la escena, esta tiene que ser cuánto menos estática. Dicho de otra manera, los movimientos de la propia cámara que está grabando la escena se traducirán en cambios bruscos del background a estimar. Si constantemente se está estimando un background dinámico que cambia en cada frame esto supondría que cualquier elemento de la escena (incluso toda la propia escena en sí) se convertiría en foreground, al compararlo con un background totalmente diferente. Por tanto, sólo se estimará el movimiento de los objetos respecto a la cámara pero sólo si esta está estática. Además, sucesos como la lluvia, el oscurecimiento de la escena, el leve movimiento de la cámara por el viento o la oclusión entera o parcial del plano pueden ocasionar desmejorías en la estimación del movimiento por un background demasiado dinámico. Es por esto, principalmente, por lo que este algoritmo es únicamente válido para cámaras estáticas que no se van a mover de un sitio fijo. No obstante, hay métodos que hacen posible la sustracción de background para cámara en movimiento teniendo en cuenta el cálculo de la trayectoria de la misma [23], aunque aún no sean algoritmos del todo eficientes.

Diferencia de frames

Se trata de uno de los algoritmos más sencillos para la substracción de background. La idea principal del método *Frame Difference* es básicamente realizar la resta aritmética y la umbralización o *threshold* entre dos imágenes: el frame actual en el instante t y un frame anterior en el instante $t - x$ siendo x un período de tiempo determinado. La imagen anterior en el instante $t - x$ será a la que se considerará background y deberá ser una representación fiel de los elementos estáticos de la escena. Suele ser una imagen al principio de la captura o del vídeo que muestre el escenario totalmente 'vacío' de objetos en movimiento, pero también puede ser cualquier otro frame que represente de manera fiable el background de la escena.

Matemáticamente se puede modelar esta resta de la siguiente manera como se muestra en la ecuación 15, donde la función L se refiere a los valores de la luminancia de los

píxeles de una imagen, F es un frame de la imagen en un instante determinado y D es la imagen resultante de la operación.

$$L[A(t)] - L[A(t-x)] = L[F(t)] \qquad (15)$$

Dado que la operación se hará con imágenes en escala de grises, el resultado es una imagen donde los elementos que hayan coincido en ambos frames se habrán eliminado (valores cercanos al 0 en la luminancia en los píxeles de dichos elementos) y quedando sólo aquellos otros elementos que hay en una imagen pero no en otra (valores cercanos a 255 en la luminancia en los píxeles de dichos elementos). Estos elementos coinciden con aquellos objetos que han entrado a escena y con los que ya estaban pero se han desplazado o cambiado de forma. Visualmente se tendrá una imagen con un fondo negro y con formas y contornos en blanco que delimitan los objetos tal y como se muestra en la figura 15, donde la forma del coche y del árbol a su izquierda aparece remarcada en blanco respecto al fondo negro. Por tanto el coche y el árbol son elementos que se han desplazado entre los frames que se ha realizado la resta.

Figura 15. Ejemplo de sustracción de background por el método de diferencia de frames

El siguiente paso es umbralizar la imagen, es decir, aplicar un **threshold**. Umbralizar la imagen es elegir aquellos píxeles que superen un valor de intensidad de luminancia umbral, tal y como muestra la expresión 16, donde T_0 es el valor umbral y p el valor de la luminancia del píxel en concreto.

$$T(p) = \begin{cases} 0 & si\ p < T_0 \\ 1 & si\ p > T_0 \end{cases} \qquad (16)$$

Estos valores que han superado dicho umbral tomarán el valor máximo (suele ser 255 en escala de grises, es decir, color blanco) mientras que los que no tomarán el valor 0 (color negro en escala de grises). De esta manera, se obtendrá una imagen binaria de sólo dos colores, y en uno de ellos, resaltados se encontrarán los objetos del foreground. Un ejemplo de ello es la imagen que se puede observar en la figura 16.

Figura 16. Ejemplo de aplicación de threshold

Hay varias maneras de realizar el threshold. Pero en cualquier caso el objetivo de este procesado es el de encontrar el valor umbral óptimo y para ello se puede recurrir a varios métodos. Puede haber varios valores umbrales para obtener imágenes más estratificadas. Puede ser un threshold adaptativo donde el valor umbral vaya cambiando de acuerdo a la necesidad que se tenga. O puede obtenerse un threshold a raíz de la varianza de los píxeles como calcula el algoritmo de Otsu [24], que debe su nombre a su inventor japonés a Nobuyuki Otsu.

Llegado a este punto, se realiza un procesado sobre la imagen resultante del threshold de forma que permita borrar el posible ruido que se haya podido añadir y además perfilar mejor las formas del foreground encontradas. Para ello, la primera parte del procesado trata sobre aplicar operaciones morfológicas sobre la imagen. Estas operaciones son la **erosión** y la **dilatación**. La dilatación permite aumentar los contornos más finos aplicando una adición de un elemento estructural mientras que la erosión se encargar de "diezmar" y recortar las formas y contornos mediante otro elemento estructural. Su aplicación puede ser conjunta o no dependiendo del uso y del resultado final de la misma, ya que el objetivo es acabar con el posible ruido que se haya aparecido en la imagen de objetos no deseados en el reconocimiento.

Una de las ventajas de *Frame Difference* como método de sustracción de background es su notable simplicidad. Es fácil de implementar y no realiza un procesado demasiado complejo, lo que le permite funcionar a gran velocidad y sobre una tasa de frames por segundo elevada. Por otro lado, sabiendo que la fortaleza de un algoritmo de sustracción de background reside en el cálculo del modelo de background es importante tener esto en cuenta. Por lo tanto en este método, al elegir un background basado en frames anteriores, habrá problemas debidos sobre todo a los cambios de luz y el ruido. Si el frame actual difiere mucho del frame anterior en condiciones de luz y ruido, pese a no haber cambios en los objetos de la escena, se detectarán muchas formas no deseadas con apenas cambios en sus movimientos. Esta es la gran desventaja de este método.

Otro aspecto negativo de este algoritmo sería su incapacidad para ignorar los objetos estáticos del background que se mueven de forma periódica (por ejemplo, los objetos de la escena mecidos repetidamente por el viento), porque al tener sólo la referencia de un frame anterior el objeto siempre se habrá movido aunque haya sido de forma mínima. Esto se traduce a la hora de realizar la resta en que estos objetos en movimiento siempre se van a detectar.

Filtro de mediana (Mean Filter)

En el método *Mean Filter* el cálculo del background se realiza calculando la media de los valores de cada píxel de una sucesión anterior de frames que se han almacenado en un buffer. Es decir, el background es modelado como una imagen cuyo píxeles son la media de los píxeles de las imágenes que se han ido sucediendo anteriormente. Este método se basa en la suposición de que un solo píxel permanece invariable más de la mitad de los frames anteriores que se han almacenado [21].

Durante la ejecución de *Mean Filter* se almacena un número N de frames anteriores consecutivos en un buffer. Del conjunto de N frames guardados se calcula la media del valor de cada píxel. Esa media de píxeles se utiliza para crear la imagen del background, y se irá actualizando cada frame consiguiendo así una renovación constante del modelo. Después de modelar el background se seguirá el mismo proceso que con el resto de algoritmos de sustracción: se restará el modelo de background al frame actual y se procesará dicha diferencia de la forma más idónea (ya sea por threshold, operaciones morfológicas, etc.). Al final de proceso, se obtendrá la máscara de foreground deseada. Este funcionamiento se puede ver representado en la figura 17.

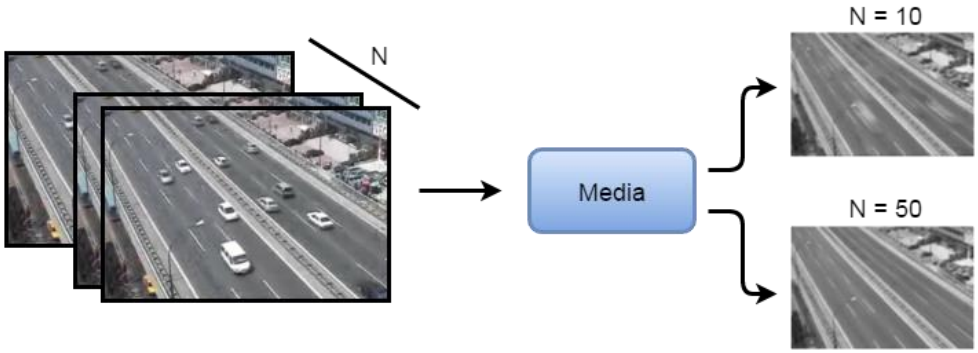


Figura 17. Modelado de background mediante Mean Filter. Fuente [21].

El funcionamiento y la efectividad de este proceso dependen de varios aspectos, pero la elección del parámetro N que almacena las imágenes el buffer es la más relevante. Es lógico que cuánto más alto sea este parámetro, hará falta un buffer más grande y una capacidad de procesamiento mejor; todo ello a cambio de una mejoría en la precisión

del modelaje del background. Si se elige por el contrario almacenar pocos frames anteriores se obtendrá menos complejidad en el procesamiento pero se obtendrá un modelo de background que se actualiza de forma más lenta.

En la figura 18 se puede apreciar las diferencias encontradas en la máscara final de foreground dependiendo de la variación del parámetro N.

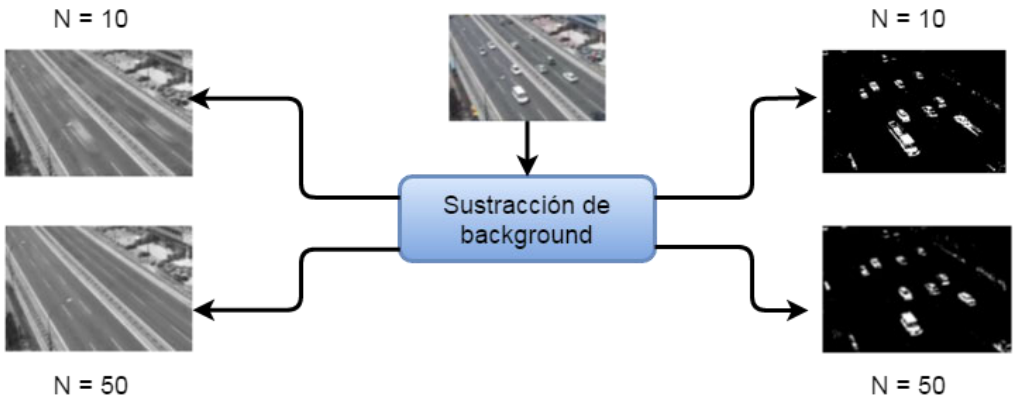


Figura 18. Comparativa Mean Filter. N = 10 y N = 50. Fuente [21].

Matemáticamente, este algoritmo responde a la ecuación 17, donde *B* será la imagen background, *N* el número de frames almacenados en el buffer, *F* el valor en luminancia del frame almacenado en el instante *t*, y la dupla *x* e *y* la posición del píxel en las imágenes.

$$B(x,y,t) = \frac{1}{N}\sum_{i=1}^{N} F\,(x,y,t-i) \qquad (17)$$

Las ventajas de este método se encuentran en su fácil diseño e implementación, así como su escalabilidad. Además permite la renovación constante y automática del modelo de background de manera simple. Por el lado contrario, se puede observar de que se requieren ciertos requisitos de memoria y buffer para almacenar todos aquellos frames de los que se van a calcular la media. Además la calidad de la máscara de foreground final dependerá de la velocidad de los objetos y la tasa de frames por segundo (*fps:* frames por segundo). Esto es por ejemplo, si un objeto atraviesa rápidamente la imagen y ha habido pocos frames que han "capturado" ha dicho objeto, la sustracción no será tan precisa como en el caso en el que hubiera una tasa alta de imágenes por segundo.

Mezclas de gaussianas: Modelado con una sola función gaussiana

En la publicación [25] se introduce el concepto de modelar la probabilidad de que un píxel tome un valor en concreto a través de una función de densidad probabilística gaussiana. Por definición, una función de densidad gaussiana viene definida por sus dos parámetros, a saber, media μ y varianza σ^2. La media se define como el valor medio posible que puede tomar una variable aleatoria, y la varianza como la desviación estándar elevada al cuadrado respecto a la media. Si se aplica una función de densidad gaussiana a cada uno de los píxeles de los frames pasados, cada uno de ellos vendrá definidos por su función de probabilidad: su media (el valor medio probable que podrá tomar el píxel) y su varianza (la capacidad de cambio de valor respecto a la media que tiene el píxel). La función de probabilidad gaussiana se puede observar en la figura 19, donde se muestra también los parámetros nombrados anteriormente.

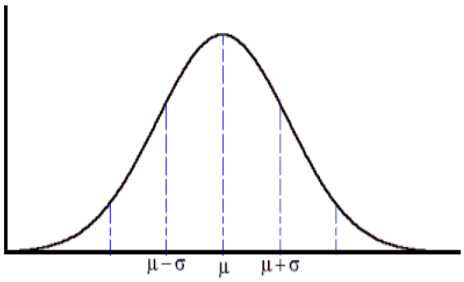


Figura 19. Distribución de Gauss, con media y varianza.

Como planteamiento inicial se puede asumir que primeramente cada píxel es considerado background. Pero, como es lógico, habrá cambios en la escena y otros objetos entrarán y saldrán de ella a distintas velocidades aparte de los ya mencionados posibles cambios de iluminación de la misma. Por ello, es necesarios actualizar la media y la varianza de cada píxel y en cada frame. Esta actualización se hará tal y como muestra en las ecuaciones 18 y 19 de la media y la varianza, donde α será el llamado factor de actualización y la función I la luminancia de un píxel en concreto en el instante actual. El parámetro d es la diferencia absoluta entre el valor del píxel y la media en ese instante t. El factor de memoria α tomará valores entre 0 y 1 teniendo en cuenta la capacidad de actualización del background que se busca para cada situación. A menor valor, mayor retención de la media y la varianza de frames anteriores, y a mayor valor se tendrá más en cuenta el frame actual.

$$\mu_t = \alpha\, I_t + (1-\alpha)\mu_{t-1} \qquad (18)$$

$$\sigma_t^2 = \alpha\, d^2 + (1-\alpha)\sigma_{t-1}^2 \qquad Donde\ d = |I_t - \mu_t| \qquad (19)$$

Una vez definidos estos valores para su renovación constante en cada frame, es necesario precisar un criterio de clasificación para determinar que píxeles serán considerados background o foreground. Para ello, se hará uso de la variable *d* dividida entre la varianza en el instante σ_t. Si este cociente es mayor que un parámetro *k*, definido como el umbral de sensibilidad del algoritmo, se considerará ese píxel como foreground. En caso de que sea menor, se considerará parte del background. Este criterio se explica matemáticamente en las ecuaciones 20 y 21.

$$\frac{|I_t - \mu_t|}{\sigma_t} > k \rightarrow Foreground \qquad (20)$$

$$\frac{|I_t - \mu_t|}{\sigma_t} < k \rightarrow Background \qquad (21)$$

A la vista de dichas expresiones se puede sacar en claro que aquellos píxeles donde la varianza sea muy grande será más fácil y probable que sean clasificados como background. Si la varianza toma valores cercanos al cero es más probable que se clasifique como foreground, todo ello dependiendo del valor elegido para *k*.

Como mejora a este método se puede incluir un factor M que haga que sólo se actualice la media del píxel cuando este se considere background por lo que su funcionamiento es parecido al de un conmutador. En la expresión 22, este factor M valdrá 0 cuando el píxel sea clasificado como foreground y 1 cuando el píxel sea clasificado como background.

$$\mu_t = (1 - M)\mu_{t-1} + M(\alpha\, I_t + (1 - \alpha)\mu_{t-1}) \qquad (22)$$

De esta manera, cuando el píxel sea clasificado como foreground (M = 0), su media será la última que obtuvo antes de dejar de ser background. Este píxel sólo podrá volver a ser considerado background de nuevo cuando tome el mismo valor de intensidad que tenía en el momento de convertirse a foreground.

Modelado con varias funciones gaussianas (Mixtures of Gaussians)

A la vista del modelado de background con una sola función gaussiana, es fácil pensar que su funcionamiento puede mejorar con la introducción de varias gaussianas por píxel [26]. Esto supone un avance en la precisión de la predicción, ya que por lo general las imágenes a veces se pueden volverse complejas con cambios constantes de luminancia u objetos que cambian muy rápido. Por ello, utilizar varias funciones gaussianas en el cálculo del background daría resultados más exactos, de ahí su nombre, *Mixtures of Gaussians* (mezclas gaussianas, en castellano).

Para este método se usarán un número *k* de gaussianas (normalmente entre 3 y 5). En este caso, de nuevo serán importantes los parámetros de la media de todas las distribuciones gaussianas en un momento dado así como la matriz de covarianza de la mezcla de todas las distribuciones.

Por otro lado, existirán pesos entre el conjunto de gaussianas para dar más importancia y prioridad a unas que modelarán mejor el background respecto a otras. Estos pesos harán situar a las gaussianas más propicias a la cabeza de la cola de prioridad para ser elegidas primeras para modelar el background, dejando fuera a aquellas gaussianas que hayan clasificado el píxel como foreground. Se elegirá como modelo de background la distribución que encabeza dicha lista.

Un píxel será considerado background cuando la distancia a la distribución elegida para modelarlo sea mayor que un parámetro previamente elegido y basado en la desviación estándar. En el caso de que el valor del píxel no encaje con ninguna distribución, este se marcará como foreground, y si, en algún momento, coincide con alguna nueva distribución de background dicho píxel se incorporará al fondo. Tal y como ocurre en el caso de modelar con una sola gaussiana, los valores de la media y la matriz de covarianza de las distribuciones se van actualizando en cada momento. No obstante, en este caso sólo se actualizarán los valores de aquellas distribuciones elegidas previamente como background por los pesos asignados a cada una de ellas.

El método de Mezclas de Gaussianas tiene notables ventajas respecto a otros métodos. Su particular clasificación de qué píxeles pertenecen al background y cuáles no hace que funcione en imágenes complejas, con elementos en escena que se mueven de manera repetitiva o muy lenta así como con variaciones de luminancia durante la sucesión de frames. Por otro lado, como desventaja se puede observar que este método conlleva una gran carga computacional ya que cada píxel es procesado por el número de gaussianas elegido. Por ello, a mayor *k*, mayor carga computacional. Esta complejidad de cálculo se añadirá al resto del procesado típico de background, que aunque puede ser ligero tampoco es prescindible. No obstante, el método Mezclas de Gaussianas es un algoritmo fuerte y robusto que funciona bien en la sustracción de background. Además es menos sensible a cambios bruscos de iluminación que otros algoritmos parecidos como puede ser KDE (*Kernel Density Estimation*), en la cual la función de probabilidad para cada píxel se estima mediante la suma de N gaussianas siendo N el número de las muestras más recientes.

2.3 Tracking

Se puede definir *tracking* como el seguimiento de uno o varios elementos a través de una sucesión de imágenes donde aparece. Supone un proceso clave y fundamental en cualquier aplicación de vigilancia o algoritmo de detección de anomalías y análisis de vídeo con visión artificial, como la que se presenta en este proyecto. Por ello, se hace imprescindible realizar un estudio del arte sobre el tracking en vídeo así como de las distintas técnicas que permitirá implementarlo funcionalmente.

En general, se trata de un proceso lento y en ocasiones con gran coste computacional y de memoria debido a la gran cantidad de datos que se manejan. El proceso de seguimiento como tal, tiene sentido como etapa final de una serie de procedimientos necesarios y anteriores. Estos procesos son la detección de características, la extracción de elementos basado en dichas características y por último, la clasificación de los objetos extraídos. Es importante que estas etapas hayan trabajado de manera correcta ya que el resultado final del tracking depende mucho de ello. La etapa final es el tracking en sí, el seguimiento del objeto extraído y clasificado.

Hay multitud de tipos de métodos de tracking cada cual con sus particularidades [27]. Lo común entre todos ellos es que se realiza el seguimiento de una característica precisa del elemento a seguir. Esta característica puede ser un conjunto de puntos, núcleos basados en plantillas de movimiento o siluetas de los propios objetos. En la figura 20 se puede observar un esquema de estos métodos y sus particularidades.

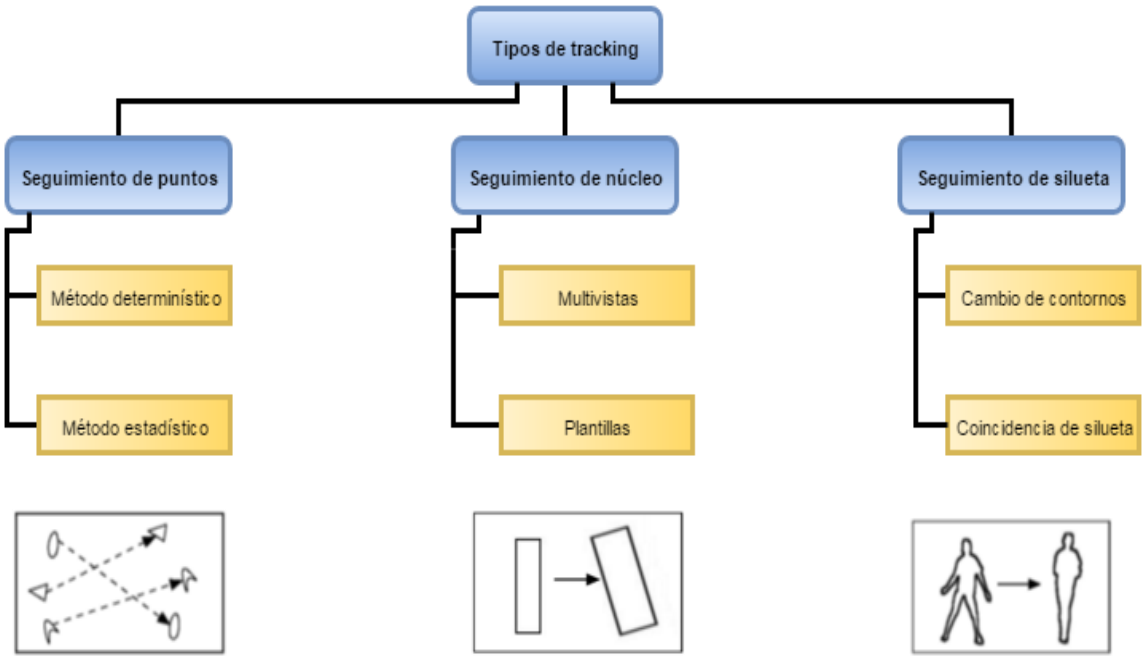


Figura 20: Tipos de tracking según el tipo de seguimiento

2.3.1 Tracking de puntos

Los métodos de seguimiento por puntos se basan en la premisa de que el objeto a seguir está compuesto por una agrupación de puntos (cada píxel que lo forma). Por tanto para realizar el tracking de dicho objeto, se puede efectuar el seguimiento de los puntos que lo forman buscando la equivalencia de dichos puntos en los frames siguientes.

Obviamente un objeto puede estar formado un elevado número de puntos y realizar todas sus correspondencias en los siguientes frames puede ser algo demasiado costoso. Además los objetos suelen cambiar de velocidad, forma y posición, lo que dificulta su seguimiento a través de los frames. Por ello, es más recomendable seguir ciertos puntos característicos que sean fácilmente reconocibles durante su estancia por la escena. Estos puntos pueden ser por ejemplo las esquinas o los bordes de un objeto, dado que son cambios relativamente bruscos en la luminancia de la imagen y definen los contornos y esquinas de los objetos.

Dentro del seguimiento de puntos se pueden encontrar dos tipos de métodos:

- El **método determinístico** utiliza condiciones y un estado inicial medidos anteriormente para dar resultados basadas en dichas condiciones y estados. Estas condiciones pueden ser distancias heurísticas, restricciones de velocidades y proximidad o movimientos del objeto. Un ejemplo de este método sería el flujo óptico [28] o cálculo de gradientes.

- En el **método estadístico** se utiliza la incertidumbre de las medidas del objeto, como puede ser su velocidad, su trayectoria o su movimiento y se realizan observaciones y aproximaciones. Ejemplos de estos métodos serían los filtros de Kalman [29] o filtro de partículas.

Flujo óptico. Algoritmo de Lukas-Kanade [30]

Como algoritmo de tracking, el flujo óptico se engloba dentro de la categoría de seguimiento de puntos o píxeles. El flujo óptico se define como el movimiento de patrones de luminancia entre los píxeles de dos imágenes cercanas o consecutivas. Como normal general, se puede asociar dichos movimientos en los patrones al

movimiento de los objetos entre las escenas, aunque esto no tiene por qué ser así siempre. Hay varias técnicas para hallar el flujo óptico. En este caso se hablará de una técnica que se apoya en el cálculo de gradientes.

Para hallar el flujo óptico se parte siempre de la premisa de que la luminancia de un píxel es constante aunque esté varíe su posición en el tiempo, siendo esto universal para todas las técnicas de flujo óptico. Matemáticamente, esto se muestra en la ecuación 23, donde I representa la luminancia de una imagen en la posición (x, y) siendo Δx, Δy y Δt el incremento en posición y tiempo.

$$I(x + \Delta x, y + \Delta y, t + \Delta t) \approx I(x, y, t) \qquad (23)$$

Desarrollando en series de Taylor en ambos lados, se obtiene el siguiente resultado en la ecuación 24.

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t + O.T \qquad (24)$$

Del anterior resultado se obtienen las ecuaciones 25 y 26.

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0 \qquad (25)$$

$$\frac{\partial I}{\partial x}\frac{\Delta x}{\Delta t} + \frac{\partial I}{\partial y}\frac{\Delta y}{\Delta t} + \frac{\partial I}{\partial t}\frac{\Delta x}{\Delta t} = 0 \qquad (26)$$

Dado que la velocidad es la derivada del espacio respecto del tiempo se obtienen las velocidades de los píxeles en la ecuación 27.

$$\frac{\Delta x}{\Delta t} = V_x\,, \frac{\Delta y}{\Delta t} = V_y\,, \frac{\Delta z}{\Delta t} = V_z \qquad (27)$$

Reescribiendo la ecuación con estos nuevos términos quedaría tal y como se muestran en las ecuaciones 28 y 29. Donde $\widehat{V}$ es el vector de velocidades en las direcciones x e y, y I_x, I_y y I_t son las derivadas parciales de la imagen respecto a las direcciones x e *y* el tiempo.

$$I_x V_x + I_y V_y = -I_t \qquad (28)$$

$$\nabla I^T * \widehat{V} = -I_t \qquad (29)$$

La ecuación 29 no tiene una única solución y se conoce como el **problema de apertura**. De ahí que se propongan distintas maneras y técnicas para resolverlas utilizando restricciones y suposiciones en dicha ecuación. En este caso se propone el método propuesto por Lukas-Kanade.

Bruce D. Lucas y Takeo Kanade propusieron un método [23] donde se parte de la premisa de que el movimiento de los objetos que se desplazan entre dos frames cercanos es pequeño y constante dentro de una región centrada en un punto *P*. Por tanto, los píxeles dentro de dicha región $(q_{1,}, q_2, \dots, q_n)$ cumplirán la ecuación genérica 30.

$$I_x(q_1)V_x + I_y(q_1)V_y = -I_t(q_1)$$

$$I_x(q_2)V_x + I_y(q_2)V_y = -I_t(q_2)$$

…

$$I_x(q_n)V_x + I_y(q_n)V_y = -I_t(q_n) \qquad (30)$$

La ecuación 30 se puede escribir en forma de matriz $Av = b$. Quedando de la forma 31.

$$A = \begin{bmatrix} I_x(q_1) & I_x(q_1) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix} \quad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix} \quad b = -\begin{bmatrix} I_t(q_1) \\ \vdots \\ I_t(q_2) \end{bmatrix} \qquad (31)$$

Este sistema tiene más ecuaciones que incógnitas y para hallar una solución se puede recurrir a la aproximación por mínimos cuadrados como en la ecuación 32.

$$A^T A v = A^T b \qquad (32)$$

Quedando finalmente como en la ecuación 33.

$$v = (A^T A)^{-1} A^T b \qquad (33)$$

La última ecuación muestra la solución del vector de velocidades, pero sólo tendrá solución si la matriz $(A^T A)$ tiene inversa.

Una forma más eficaz de realizar esta técnica es utilizando las imágenes piramidales, procedimiento explicado en apartados siguientes, donde partiendo de una versión reducida de la imagen original, se va hallando el flujo óptico desde la versión más pequeña hasta llegar a la original.

2.3.2 Tracking de núcleo

El tracking de núcleo o de kernel se basa en el seguimiento de un patrón con la forma, geometría o aspecto del objeto a seguir. El tracking de este modelo se hace sucesivamente de un frame a otro en cada uno y su movimiento suele ser representado en pequeñas variaciones de desplazamiento y rotación.

Dentro de este tipo de seguimiento se puede diferenciar entre:

- **Los modelos de apariencia en plantillas y en funciones de densidad** funcionan mediante la búsqueda y coincidencia de una plantilla por fuerza bruta en toda la imagen. Esto es buscar regiones de la imagen que concuerden en gran parte con la plantilla del objeto a seguir. Para hallar dichas coincidencias se suelen utilizar histogramas del espacio de color HSV de la imagen, así como las propiedades de luminancia y color del objeto, y la probabilidad de que un píxel de la imagen valga cierto valor de la plantilla buscada. Dado que esto puede resultar costoso, la mayoría de los algoritmos buscan en regiones cercanas a las coordenadas previamente localizadas.

  Se trata de un método sencillo y bastante extendido debido a su facilidad de implementación. Algoritmos como Mean-Shift [31] o su versión mejorada Cam-Shift [32] tienen esta filosofía de funcionamiento. La técnica de seguimiento de objetos por modelos de apariencia de plantillas y funciones de densidad es utilizada como algoritmo de tracking en el desarrollo de la aplicación que atañe esta memoria y sus aspectos más concretos son desarrollados en apartado siguientes.

- **Los modelos de apariencia de multivista** se encargan de almacenar y "aprender" las múltiples apariencias de un mismo objeto. Estas apariencias puede ser una misma persona en diferentes posiciones y en diferentes puntos de vista ya sea porque se ha girado sobre sí misma o se ha acercado o alejado de la cámara. Estos modelos son muy útiles para el caso en el que el objeto varíe durante la escena y lo haga de forma brusca. Las máquinas de vectores de soporte (SVM) anteriormente explicadas siguen está filosofía, ya que durante el entrenamiento la muestras positivas pueden contener numerosas vistas y apariencias del objeto que se intenta seguir.

2.3.3 Tracking de silueta

En este método se valora la información dentro de la región del objeto. Dicha información se estima en los bordes y la silueta del mismo consiguiendo una descripción. Una vez obtenidas estas descripciones se puede optar por buscar coincidencias por fuerza bruta o bien siguiendo la evolución de la silueta conseguida.

Dentro de este tipo de tracking se puede encontrar las siguientes técnicas:

- En el método de **coincidencia de forma** se busca la silueta del objeto en el frame actual en forma de fuerza bruta, al igual que en el caso de los modelos de apariencia de plantillas y en funciones de densidad. El método buscará la máxima coincidencia de la silueta previamente conformada con la hipotética forma del objeto en el frame actual. Dado que la forma de la silueta del objeto puede haber variado en el frame actual respecto al anterior este método sólo contempla el caso de seguimiento de objetos rígidos.

- En el método de **evolución de contorno** se parte de la suposición de que el contorno o silueta del objeto varía significativamente entre frame y frame consecutivos, por lo que es necesario incluso un solape del objeto entre dichos frames. Este solape se refiere a que la región del objeto en el frame anterior sea en parte la misma con la región del objeto en el frame actual.

3. Soluciones propuestas y desarrollo

3.1 Requisitos de la aplicación

Dada la naturaleza de la aplicación y en base a lo estudiado en el estado del arte, se han definido una lista de requisitos funcionales que se deben de cumplir en la mayoría de su totalidad. Estos requisitos definen el comportamiento y el funcionamiento del sistema a diseñar. Son los siguientes:

- **RF1**. La aplicación debe poder ejecutarse en ordenadores con sistemas operativos Windows 7 y posteriores.
- **RF2**. La aplicación debe realizar la detección y el reconocimiento de los vehículos en un vía de tráfico.
- **RF3**. La aplicación debe realizar el control del tráfico vehicular desde el punto de vista de la opción que el usuario haya elegido previamente.
- **RF4**. Según la opción elegida previamente, la aplicación interactuará con el usuario mediante realidad aumentada.
- **RF5**. La aplicación debe permitir al usuario elegir entre varias opciones de control y supervisión del tráfico.
- **RF6**. La aplicación debe permitir al usuario operar con el ratón y el teclado de forma sencilla a la hora de elegir el tipo de control y supervisión.
- **RF7**. La aplicación debe funcionar en tiempo real. En caso de que la adquisición de la imagen sea por vídeo la tasa de frames por segundo debe mantenerse de manera estable.
- **RF8**. La detección y reconocimiento de los vehículos ha de ser un proceso preciso, con baja tasa de error, tanto de falsos positivos como falsos negativos.
- **RF9**. Siempre que la aplicación acabe debe hacerlo de forma ordenada. Si ocurre un error ha de informarse al usuario de la naturaleza del mismo.
- **RF10**. La aplicación debe contar con una interfaz gráfica sencilla que permita la interacción con el usuario de forma fluida.

3.2 Entorno y especificaciones de desarrollo

Realizar el estudio del estado del arte ha permitido elegir un entorno de desarrollo adecuado y que sea capaz de cumplir los requisitos y funcionalidades marcadas para la aplicación. Por ello, se ha optado por el desarrollo del código de la aplicación en lenguaje **C++** mediante las librerías de **OpenCV** [33] (concretamente la versión 4.2.10 por ser la última versión estable disponible a la hora de empezar a realizar el desarrollo de este proyecto) y **cvBlob** [34]. Como herramienta de desarrollo (o framework) en las primeras

partes del proyecto se ha utilizado el programa **Visual Studio de Microsoft**. El desarrollo final y la implementación de una interfaz gráfica se han realizado con el framework **Qt Creator**.

En el anexo al final de este documento se puede encontrar una guía detallada paso a paso de la instalación de las librerías de OpenCV y su integración tanto en Visual Studio como en Qt Creator.

OpenCV

OpenCV (*Open Source Computer Vision Library*) es una librería de software *open source* de visión artificial y *machine learning* desarrollada por Intel. Fue ideada en 1999 para proveer una estructura común para aplicaciones de visión por computador así como para acelerar el uso de las aplicaciones de máquina en productos comerciales, todo ello englobado en un entorno altamente eficiente dadas las características y requisitos de las mayorías de aplicaciones de visión artificial.

Cuenta con una colección más de 2500 algoritmos capaces de realizar tareas de detección y reconocimiento de objetos, caras, clasificación de humanos, ya sea en tiempo real o no, extracción de modelos en 3D, etc… Estos algoritmos abarcan desde los métodos más clásicos hasta los más modernos del estado del arte. Aunque nativamente esté escrito en C++ (donde dispone de modelos STL implícitos además de estar más optimizado), OpenCV también tiene interfaces para los lenguajes C, Python, MATLAB y Java y funciona para múltiples sistemas operativos como Windows, Linux, iOS, Mac OS y Android.

Figura 21. Símbolo de OpenCV [33]

Las librerías de OpenCV se dividen en módulos dependiendo de cada área de uso. Así, se pueden encontrar desde módulos de funciones para el análisis de vídeo, de fotografía e imágenes o de estructuras básicas en visión por computador como pueden ser filtros o detectores de características. Los más importantes y los utilizados en este proyecto son:

- Módulo *core*: Se trata de un importante módulo con estructuras y tipología de datos básicas que supondrán el "núcleo" del manejo de las operaciones con OpenCV, ya que el resto de los módulos se controlan con las estructuras y tipos de *core*. En este módulo podemos encontrar clases como 'Mat', que modela una imagen como una matriz de tres dimensiones cuyos valores son las tres componentes del espacio colorimétrico RGB, 'Point', que modela puntos en dos o tres dimensiones basadas en las coordenadas del espacio, o 'InputArray/OuputArray', que modela una clase de varios tipos de entrada/salida en funciones siendo estos tipos vectores, matrices o incluso vectores de matrices.

- Módulo de procesamiento de imágenes: A su vez se divide en "Filtrado de imágenes", "Transformaciones Geométricas", "Cálculo de histogramas", "Análisis de estructuras y descriptores de contornos", "Detección de características" y "Análisis de movimiento y tracking". Constituye otro de los módulos principales de OpenCV puesto que contiene un gran número de clases y funciones que realizan diversos procesamientos, filtrados, transformaciones, detecciones y análisis sobre las imágenes. De este módulo se obtendrán la mayoría de las funciones que se utilizarán en el desarrollo de este proyecto.

- Módulo de *machine learning*: Se trata de otro módulo principal que provee de modelos estadísticos de clasificación, regresión y agrupamiento de datos. Cabe destacar especialmente la clase de SVM (Support Vector Machines) que contiene y que se utilizará en la clasificación de objetivos.

OpenCV está abierto para su descarga en su página web donde se pueden encontrar también tutoriales, documentación y otras referencias. Cuenta además con un gran foro de consulta de dudas y también un espacio de contribuciones al código al ser *open source*.

CvBlob

CvBlob [34] es un conjunto de librerías *open source* de visión artificial escritas en C++ basadas en OpenCV. Permiten la detección, el análisis y el etiquetado de regiones (o *blobs*) conectadas entre ellas todo ello basado en el algoritmo de la publicación [35]. Más allá del análisis de componentes, CvBlob también permite realizar el seguimiento (*tracking*) de las regiones analizadas. Este seguimiento está basado en el algoritmo de la publicación [36], explicado en este apartado más adelante.

En general, se trata de unas librerías muy completas ya que contienen múltiples herramientas que realizan el análisis y el seguimiento de las componentes desde varios puntos de vista. Permite mostrar la información de varias maneras, como dibujar cada región analizada de un color distinto, escribir en un archivo de depuración las coordenadas de los centros de cada detección en cada frame, etc…

Estas librerías junto a OpenCV constituirán las herramientas centrales de la programación de la aplicación y se harán uso de ella.

Microsoft Visual Studio

Visual Studio es un IDE (entorno de desarrollo integrado) desarrollado por Microsoft que opera en sistemas operativos Windows. Se trata de una colección de herramientas de desarrollo de aplicaciones de múltiples tipo reunidas bajo un mismo entorno. Visual Studio soporta lenguajes de programación como C++, C#, .NET, Java, Python, PHP, además de dar soporte a otros lenguajes de desarrollo web. La última versión estable a día del comienzo de este proyecto es la Visual Studio 2013.

Figura 22. Logo de Visual Studio 2013

En el caso que atañe a esta memoria, a través de un proyecto (o como es llamado dentro de Visual Studio, “solución”) se desarrollará en las etapas iniciales de desarrollo una aplicación de consola utilizando el lenguaje C++ e integrando las ya citadas librerías de OpenCV y cvBlob.

Visual Studio trabaja de forma nativa con el lenguaje C++ permitiendo el uso de compiladores y correctores mientras se escribe el código, facilitando así la tarea del programador. Por otro lado, y como viene siendo habitual en casi todos los entornos de desarrollo actuales, Visual Studio también ofrece un modo *Debug* por el cual se puede observar los valores de las variables y registros de la aplicación en determinados instantes configurados por los llamados puntos de interrupción.

Qt Creator

Qt es un entorno de desarrollo multiplataforma para programas de escritorio y aplicaciones embebidas y en smartphones. Soporta multitud de plataformas de una gran variedad: desde Linux o Windows hasta llegar a sistemas Android o Blackberry. Se trata de un software libre y de código abierto, apoyado por una amplia comunidad de usuarios así como empresas como Nokia, que fue responsable de parte de su desarrollo en sus orígenes. Qt es comúnmente utilizado para la creación rápida y sencilla de aplicaciones con interfaz gráfica, tanto para escritorio como para sistemas embebidos o teléfonos móviles. También permite la programación de aplicaciones con consolas o líneas de comando para servidores. Su biblioteca cuenta con clases y métodos diversos que son capaces de gestionar servicios como llamadas a bases de datos SQL, gestión de hilos o manejo de redes.

Al contrario de lo que se puede pensar a priori, Qt no es un lenguaje de programación, sino una herramienta de desarrollo programada en lenguaje C++, el cual usa de manera nativa. Por otro lado, utiliza un preprocesador MOC (*Meta-Object Compiler*) para completar con funcionalidades extras al lenguaje C++ tales como el sistema de señales y slots, muy útil para la gestión de eventos en interfaces gráficas (GUI). No obstante, mediante *bindings*, Qt permite la programación en otros lenguajes como Python o Java. Su sistema de *build* (proceso por el cual se realiza la compilación del programa generando sus archivos ejecutables) está basado en uno propio de Qt llamado *qmake*.

Una aspecto importante de Qt es la integración de su propio entorno de desarrollo (IDE: *Integrated Development Environment*), llamado **Qt Creator** del cual se ha utilizado su versión 3.6.1 junto a la versión 5.6 de Qt, por ser las versiones más estables a la hora de realizar este proyecto. Junto a este IDE, Qt utiliza un sistema basado en widgets, un conjunto de herramientas de ayuda para la interfaz y que autogenera código según se realiza el diseño de la gráfica. En el anexo adjunto en la parte final de este documento se explica de forma detallada la instalación y la integración con el resto del software.

Qt y Qt Creator han sido utilizados en las fases finales de desarrollo de la aplicación para realizar la interfaz gráfica. Su elección está fundamentada en la fácil integración que tiene de las librerías OpenCV y el sencillo manejo a la hora de diseñar e implementar interfaces gráficas.

3.3 Soluciones propuestas

A la vista del estudio del arte realizado en el apartado anterior, de las pruebas de diseño y teniendo siempre en cuenta la naturaleza de la aplicación, se ha decidido optar entre dos sistemas que cumplan los requisitos especificados:

- HOG + SVM + Flujo Óptico
- Sustracción de background y tracking por seguimiento de modelos de apariencia por plantillas de color.

Aunque en principio ambas propuestas pueden cumplir con los requisitos funcionales de la aplicación existen grandes diferencias entre ellas, tanto en diseño como en funcionamiento.

3.4 HOG + SVM + Flujo Óptico

En esta solución el algoritmo **HOG** se encarga de la parte de extracción de características de la imagen y **SVM** de la parte de la detección de vehículos en las mismas. Para la parte de seguimiento del objeto se propone un algoritmo de tracking por puntos (en este caso esquinas) basado en gradientes como es el tracking por **Flujo Óptico**.

En la figura 23 se puede ver la estructura de esta solución propuesta.

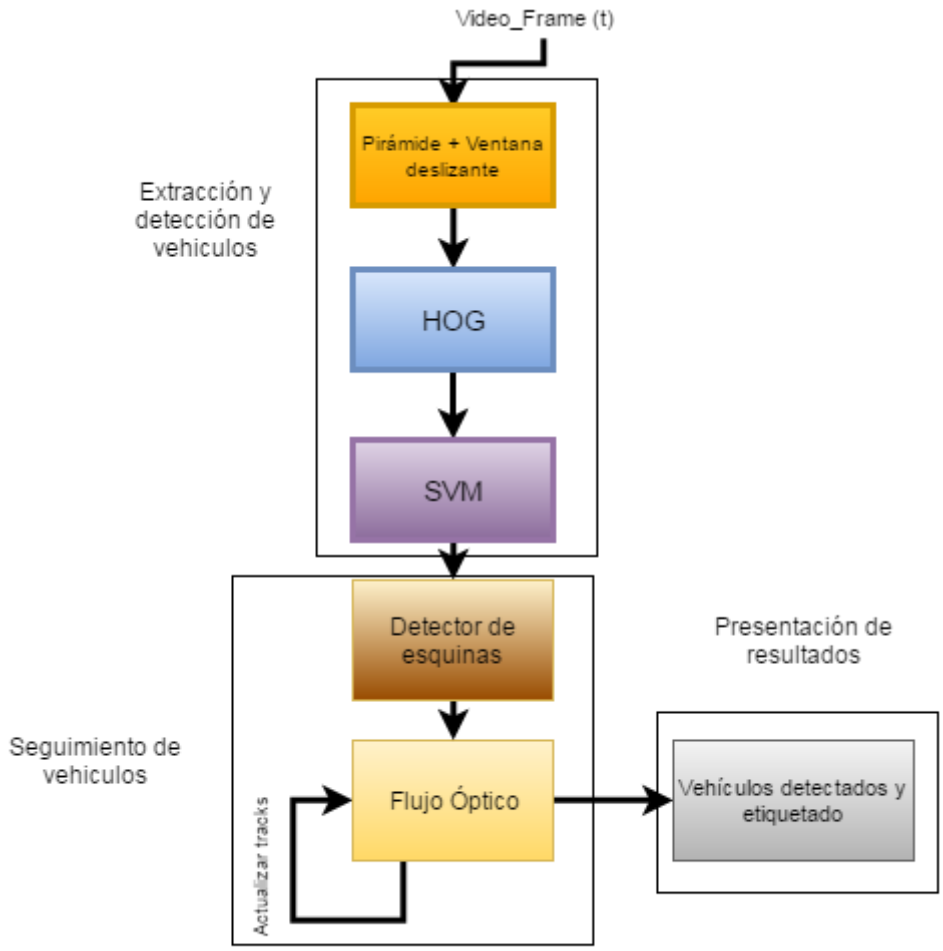


Figura 23. Estructura de la solución propuesta para HOG y SVM

3.4.1 Extracción y detección de vehículos

El primer módulo de la solución data sobre la extracción y detección de los vehículos en la escena. Este módulo a su vez está compuesto por tres submódulos distintos: **Pirámide y Ventana Deslizante** que junto a **HOG** se encargarán de la extracción de candidatos y finalmente **SVM**, que detectará que candidatos son vehículos y cuáles no. A continuación se explica el funcionamiento de cada submódulo.

Ventana deslizante y Pirámide de Imágenes

Para averiguar si una imagen contiene algún objeto de interés primero se necesita analizarla detenidamente en busca de dichos objetos. Para ello, se propone una

segmentación mediante una ventana de una medida fija que se vaya desplazando por toda la imagen, de arriba a abajo, de izquierda a derecha. En cada desplazamiento la ventana segmentará la parte de la imagen que esté dentro de ella. Ese segmento se procesará, y cuando el proceso acabe la ventana se desplazará ciertos píxeles hacia a la derecha hasta el fin de la línea. Cuando esto ocurra, la ventana se desplazará ciertos píxeles verticalmente y volverá al principio de la línea. Así sucesivamente hasta el fin de la imagen, como se muestra en la figura 24, donde la primera posición de la ventana (color rojo) se movería seguidamente un desplazamiento Δx (color azul y verde) hasta llegar al final de la línea (color negro). Terminada la línea horizontal la ventana se desplazaría Δy al principio de una nueva (color amarillo).

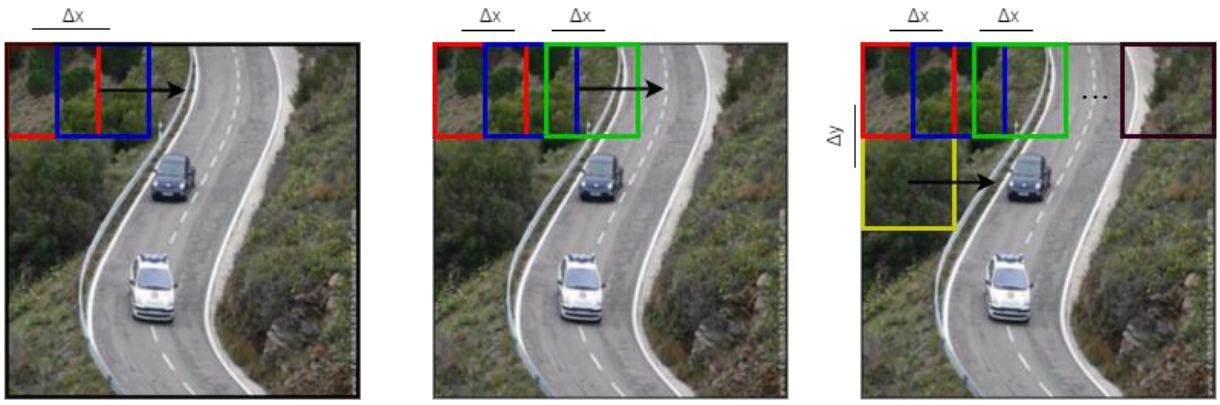


Figura 24. Funcionamiento de la ventana deslizante

El objetivo de esta operación es la generación de candidatos a posibles objetos (figura 25.C). Como es lógico, no todas las segmentaciones contendrán vehículos. Muchas ventanas contendrán candidatos no aptos (cualquier elemento no coche, figura 25.A) y algunas pueden contener parcialmente un coche pero no de forma completa (figura 25.B).

Figura 25. A) Candidatos no aptos. B) Candidatos dudosos C) Candidatos aptos

En la implementación en el código esta operación de recorrer horizontal y verticalmente la imagen se ha realizado tal y como se muestra en el siguiente pseudocódigo:

```
PARA Y IGUAL A 0, MIENTRAS QUEDE IMAGEN POR RECORRER EN VERTICAL ENTONCES
```

```
        PARA X IGUAL A 0, MIENTRAS QUEDE IMAGEN POR RECORRER EN HORIZONTAL ENTONCES
                DEFINIR RECTANGULO CON TAMAÑOS IGUAL A VENTANA DESLIZANTE
                RECORTAR VENTANA DE LA IMAGEN ESCANEADA EN LA COORDENADA X, Y
                REALIZAR CÁLCULOS CON LA IMAGEN ENVENTANADA (HOG + SMV)
                X = X + PASO_HORIZONTAL
        FIN PARA
        Y = Y + PASO_VERTICAL
FIN PARA
```

El recorte de la imagen se ha realizado primero definiendo un rectángulo con la **función *Rect*** con las medidas de la ventana, en este caso de 64x64 píxeles, y después extrayendo un recorte en forma de ese mismo rectángulo de imagen escaneada en las coordenadas X e Y definidas por las **clase *Point***. De esa manera se obtiene un recorte de la imagen con tamaño igual a la ventana deslizante. Con ese recorte se realizan los cálculos pertinentes de HOG y SVM y que se explican más adelante. Tras acabar se aumentan la coordenada horizontal X con un paso horizontal Δx de 20 píxeles hasta el fin horizontal de la imagen. Después se aumentaría la coordenada Y con un paso vertical Δy de 40 píxeles hasta el fin definitivo de la imagen. La elección de que el paso horizontal sea más pequeño que el vertical viene dada para evitar que se reconozca demasiadas veces el mismo coche verticalmente. Cabe destacar que a mayor valor de los pasos de la ventana más precisión a cambio de una ejecución más lenta, y viceversa.

Por otro lado, el tamaño de la ventana define y limita el tamaño de los objetos reconocidos en la imagen. Esto puede traer problemas de dimensiones en caso de que el objeto sea más grande que la ventana, pero se puede resolver de manera más fácil realizando la operación de ventana deslizante en versiones de la misma imagen pero de menor tamaño, lo que se conoce como **pirámide de imágenes.**

En la escena los objetos pueden moverse libremente. Unos objetos aparecerán más lejos que otros, y además esto dependerá de la perspectiva de la cámara. Este problema se traduce en que los objetos que estén más cerca de la cámara aparecerán más grandes que aquellos que se encuentren a una distancia más prudencial. Por ello, cuando la ventana deslizante segmente una parte de la imagen, es posible que el objeto no entre dentro de toda la región de la ventana porque este sea demasiado grande como ocurre en la figura 26.A. Esta situación puede dar lugar a tener problemas de falsos negativos, ya que no se reconocerían los objetos más grandes. Por tanto, se propone realizar una serie de redimensionados de la imagen para que estos objetos quepan y se adecúen más a la ventana deslizante para una mejor clasificación. En la figura 26.B se puede observar como la imagen anterior se acorta en dimensiones con un factor determinado, tanto

horizontalmente como en vertical, y como una ventana del mismo tamaño que antes engloba a la furgoneta.

Figura 26. A) Imagen Original: No se enventana correctamente la furgoneta. B) Imagen reducida en un factor 0.5: La furgoneta se enventana correctamente. Fuente propia.

Así, mediante esta técnica se consigue detectar todos los posibles objetos de la imagen independientemente de su tamaño y posición en la escena. Los objetos pequeños son reconocidos en la imagen original y los objetos más grandes son reconocidos en las imágenes piramidales diezmadas en tamaño.

Si esta operación se realiza un número determinado de veces (hasta que no tenga sentido reducir más el tamaño de la imagen) y se ordena las imágenes verticalmente por tamaño de menor a mayor, dará lugar a una estructura piramidal como se muestra en la figura 27, a modo de ejemplo, donde el factor de reducción es de 0.5.

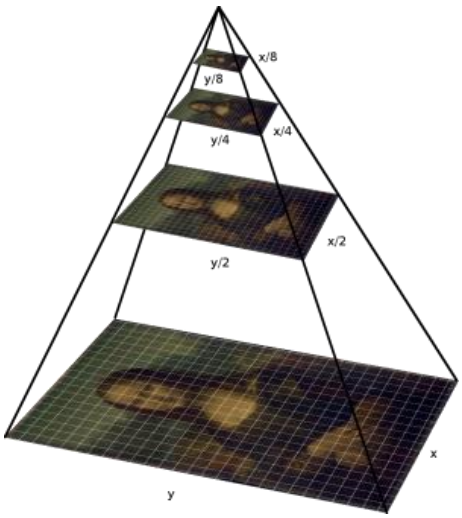


Figura 27. Disposición en pirámide de las imágenes reducidas. Fuente [37]

Esta técnica se ha implementado de forma sencilla en la solución en conjunción con la ventana deslizante, y su funcionamiento se detalla en el siguiente pseudocódigo, donde se ha resaltado en negrita lo relativo a la codificación de la técnica:

```
SE CALCULAN Nº DE VECES QUE SE PUEDE REDUCIR LA IMAGEN -> ITERACIONES

PARA I = 1, MIENTRAS QUE I MENOR ITERACIONES ENTONCES

        /* EN LA PRIMERA ITERACIÓN NO SE REALIZA EL REESCALADO */
        SI I MAYOR QUE 1 ENTONCES
                DIEZMAR DIMENSIONES DE LA IMAGEN SEGÚN UN FACTOR
        FIN SI

        PARA Y = 0, MIENTRAS QUEDE IMAGEN POR RECORRER EN VERTICAL ENTONCES

                PARA X = 0, MIENTRAS QUEDE IMAGEN POR RECORRER EN HORIZONTAL ENTONCES

                        DEFINIR RECTANGULO CON TAMAÑOS IGUAL A VENTANA DESLIZANTE

                        RECORTAR VENTANA DE LA IMAGEN ESCANEADA EN LA COORDENADA X, Y

                        REALIZAR CÁLCULOS CON LA IMAGEN ENVENTANADA (HOG + SMV)

                        X = X + PASO_HORIZONTAL

                FIN PARA

                Y = Y + PASO_VERTICAL

        FIN PARA

FIN PARA
```

El cálculo del número de veces que se puede reducir la imagen se realiza calculando el mínimo entero entre la división del ancho y el alto de la imagen por el factor de escala. Una vez hallado el número de iteraciones, se entrará en un bucle en el que se realizará la ventana deslizante en cada iteración, salvo en la primera que es la imagen original y la basa de la pirámide.

Como forma de mejora, es posible definir previamente una región de interés (en inglés, ROI: *Region Of Interest*) de la imagen, desechando así una parte de ella donde se sabe que no aparecerán objetos de interés. Así, la ventana sólo tendrá que escanear aquella región que se haya definido, aumentando así la rapidez de la operación.

HOG

El funcionamiento detallado del algoritmo HOG se ha explicado previamente en el apartado 2.2.2, aquí se mostrarán sus aspectos relativos a la solución propuesta.

Visualmente, un descriptor HOG utiliza la información del gradiente para describir los contornos y los bordes de los objetos. Esta información es muy útil a la hora de reconocer objetos con formas definidas como puede ser en este caso, los vehículos de

una vía, o en otro caso, personas. En la figura 28 se puede ver una descripción visual de los descriptores HOG de varias imágenes divididas en celdas. En cada celda se muestra en verde la acumulación y la dirección de todos sus gradientes en el centro de la misma. De esta manera, se puede observar en las figuras de la izquierda de la figura 28 cómo en las celdas correspondientes a los bordes tienen las orientaciones del gradiente más marcadas en una dirección, siendo además estas orientaciones perpendiculares a dichos bordes. En las figuras del centro y de la derecha se muestra lo mismo pero con imágenes de vehículos a modo de ejemplo.

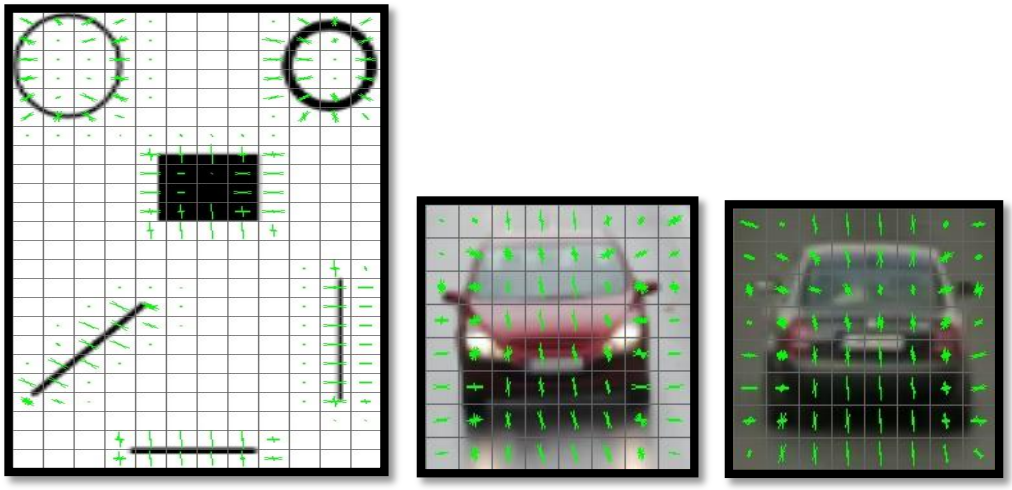


Figura 28. Orientaciones de los gradientes por celdas. DE izquierda a derecha: Varios polígonos, Vehículo de frente y Vehículo de lateral

Las librerías de OpenCV permiten calcular descriptores HOG de una imagen mediante la clase HOGDescriptor contenida en el paquete *objdetect*, relativo a funciones y clases para la detección de objetos. Esta clase también permite la detección de los objetos en una imagen a través un detector SVM interno. No obstante, este detector SVM "interno" necesita ser configurado previamente con unos coeficientes que definan sus vectores de soportes adecuadamente para cada caso. Estos coeficientes se pueden calcular a través de la concatenación de los descriptores de las imágenes de entrenamiento positivas y negativas. Pero este cálculo no está contemplado por ninguna función de la clase ni de OpenCV, lo que dificulta el uso del detector SVM interno. Por esa razón, se decide utilizar una clase SVM externa a OpenCV y prescindir de esta.

Los atributos más importantes de la clase HOGDescriptor se resumen a continuación:

- **winSize**: tamaño de la imagen de entrada, es decir, la imagen segmenta previamente por la ventana deslizante. Será de 64 x 64 píxeles.
- **cellSize**: atributo que controla el tamaño de cada celda. Por defecto vale 8x8 píxeles.

- **blockSize**: atributo que controla el tamaño del bloque en píxeles, normalmente, 16x16 píxeles. Por tanto un bloque contendrá 4 celdas.
- **blockStride**: atributo que controla el desplazamiento de celdas a la hora de agruparlas en bloques. Por defecto vale 8x8 píxeles, es decir, se desplaza dos celdas, lo que significa que siempre va a haber un solape de un 50% del bloque anterior.
- **signedGradient**: si el gradiente lleva signo, las orientaciones sólo se definirán en 180 grados. Por defecto, los gradientes llevarán signo.
- **nBins**: número de particiones en las que se divide las orientaciones del gradiente. Valor fijo a 9, dando lugar a particiones de 180 grados/9 divisiones = 20 grados.

Sus métodos más importantes son:

- **compute** (img, descriptors, winStride, padding, locations): calcula el array de descriptores *descriptors* de la imagen *img*. Aumentado el valor *winStride* se obtendrá un mayor solape entre bloques dando lugar a un descriptor más estable, a cambio de un procesado más lento. El parámetro *padding* es un relleno de píxeles extra que se colocan alrededor de la imagen que mejora el cálculo de los descriptores. El parámetro *locations* contiene los puntos hallados.
- **setSVMDetector** (detector): configura el detector SVM interno mediante los coeficientes del parámetro detector. Una vez configurado el detector, es posible realizar detecciones con las funciones *detect* y *detectMultiScale*.
- **detect** (img, found_locations, hit_threshold, win_stride): Esta función sólo se puede utilizar en caso de que se haya configurado el detector SVM interno. Realiza una inspección de izquierda a derecha y de arriba a abajo y devuelve en el array *found_locations* las localizaciones de los objetos hallados en la imagen *img* sin reescalarla, por lo que sólo detectará objetos del tamaño que especifica el atributo de clase win_*size*. El parámetro *found_locations* devuelve la coordenada superior-izquierda de los objetos detectados. Se devuelve esa coordenada en concreto para facilitar el encuadramiento del objeto mediante un rectángulo. *hit_threshold* se trata de un parámetro libre que configura la distancia entre las muestras de entrada y el plano clasificador del detector SVM. Normalmente vale 0 y viene especificado por los coeficientes del detector en la última posición. *Win_stride* es el desplazamiento vertical y horizontal de la ventana de detección, que tendrá las mismas dimensiones que el atributo de clase *win_size*. Cuanto menor sea el desplazamiento más precisa será la detección pero más se tardará en recorrer toda la imagen.

- **detectMultiScale** (img, found_locations, hit_threshold, win_stride, scale0, group_threshold): realiza una detección en la imagen *img* igual que en la función

detect anterior, salvo con la particularidad de que se realizan reescalados de la ventana de detección para detectar objetos más grandes. Esto es al contrario que en el caso de procesado de imágenes en pirámide, donde lo que se reescala es la imagen de entrada, no la ventana deslizante. Los parámetros *found_locations*, *hit_threshold* y *win_stride* son los mismos que en la función *detect*. El parámetro scale0 indica el factor de escalado de la ventana de búsqueda. El párametro *group_threshold* es un factor de umbral de similaridad para realizar una única elección entre las varias posibles detecciones del mismo objeto.

- **getDescriptorSize** (): devuelve el tamaño de del descriptor que se calcula de la siguiente manera: *Número de bloques x Celdas por bloque x Divisiones por Celda*. A mayor tamaño de descriptor mayor estabilidad pero también mayor tiempo de cálculo.

Tras extraer las muestras con la ventana deslizante el siguiente paso es calcular los descriptores con HOG con la función **compute**. Esto funcionamiento se puede observar en negrita en el siguiente pseudocódigo de la solución:

```
SE CALCULAN Nº DE VECES QUE SE PUEDE REDUCIR LA IMAGEN -> ITERACIONES

PARA I = 1, MIENTRAS QUE I MENOR ITERACIONES ENTONCES

	/* EN LA PRIMERA ITERACIÓN NO SE REALIZA EL REESCALADO */
	SI I MAYOR QUE 1 ENTONCES
		DIEZMAR DIMENSIONES DE LA IMAGEN SEGÚN UN FACTOR
	FIN SI

	PARA Y = 0, MIENTRAS QUEDE IMAGEN POR RECORRER EN VERTICAL ENTONCES

		PARA X = 0, MIENTRAS QUEDE IMAGEN POR RECORRER EN HORIZONTAL ENTONCES

			DEFINIR RECTANGULO CON TAMAÑOS IGUAL A VENTANA DESLIZANTE

			RECORTAR VENTANA DE LA IMAGEN ESCANEADA EN LA COORDENADA X, Y

			/* HOG*/
			CALCULAR DESCRIPTORES DE LA VENTANA

			OPERACIONES CON SVM

			X = X + PASO_HORIZONTAL

		FIN PARA

		Y = Y + PASO_VERTICAL

	FIN PARA

FIN PARA
```

El siguiente paso sería la clasificación de dichas características en vehículo o no vehículo, cuyo cometido corresponde al submódulo de SVM.

SVM

El funcionamiento de SVM en profundidad y en nivel de detalle viene explicado en el apartado anterior 2.2.3. En esta parte se comentará sus aspectos relativos a la implementación de la solución propuesta.

La función de este módulo será la de clasificar los descriptores HOG de las imágenes de entrada en muestra positiva (vehículo) o en muestra negativa (no vehículo). No obstante, y como es sabido, para poder realizar esta clasificación es necesario que el detector SVM sea previamente entrenado para que pueda discernir entre muestras positivas y negativas. Además, esta parte de la solución es de las más críticas e importantes. Si se realiza una mala clasificación, se pueden dar dos tipos de errores:

- **Falso Negativo**: una muestra positiva se clasifica como muestra negativa.
- **Falso Positivo**: una muestra negativa se clasifica como muestra negativa.

La manera de entrenar el detector es administrándole un grupo de muestras representativas de cada conjunto. Para este caso concreto, donde se va a clasificar vehículos como muestras positivas, se necesita imágenes donde aparezcan vehículos en la mayor de las condiciones posibles, es decir, vehículos de diferentes colores, en diferentes posiciones, de diferentes alturas y estructuras, con distintas iluminaciones, etc… Se debe evitar que aparezcan elementos como las líneas de la carretera, partes de peatones, semáforos o calzadas en la medida de lo posible. Al igual que con las muestras positivas, es necesario recalcar que en estas muestras negativas no aparezcan los objetos de interés, como partes de coche o mitades de vehículos. En la figura 29, a la izquierda, se muestra dos ejemplos de lo que pueden ser dos buenas muestras positivas. En la derecha, dos ejemplos de muestras negativas.

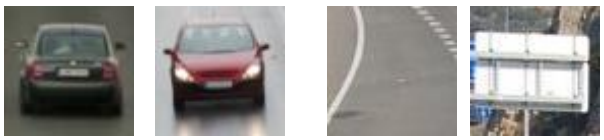


Figura 29. Muestras positivas (a la izquierda). Muestras negativas (a la derecha). Fuente propia.

Para realizar la implementación de este módulo de clasificación se ha utilizado la clase *CvSVM* de OpenCV del paquete *ml* (Machine Learning) que modela un clasificador SVM totalmente configurable en todos sus parámetros mediante código, ya que para ello utiliza la estructura auxiliar *CvSVMParams*. A su vez, la implementación de la clase SVM está basada en la librería LibSVM [38], una librería externa optimizada para este tipo de propósitos. La estructura *CvSVMParams* contiene los siguientes parámetros para su constructor:

- **Número de clases:** Configura el tipo de SVM. Determina el número de clases que clasificará. En este caso se utilizará un SVM de **dos clases**.
- **Tipo de kernel:** Configura el núcleo discriminatorio del SVM. Se trata de la figura que modelará el hiperplano para hacer la discriminación en la clasificación. En este caso, se utilizará **un núcleo lineal**.
- **Parámetro C:** Este parámetro regula la separación entre las dos clases. Su valor será de 0.01.

Una vez constituidos los principales parámetro para la clasificación es necesario inicializar el objeto SVM de la clase *CvSvm*, cuyos principales métodos son los siguientes:

- **Train** (trainData, labels, params): Esta función entrena al objeto SVM para la clasificación con los parámetros almacenados en *params*. Devuelve *true* en caso de que el entrenamiento haya sido exitoso. Los datos de entrenamiento se almacenan en la variable matriz *trainData*. Para identificar de qué clase es cada dato en la matriz *trainData* se utiliza la variable *labels*, donde se "etiqueta" cada dato de la matriz de datos con la clase con la que guarda correspondencia.

- **Predict** (sample, returnDFVal): Esta función devuelve la clase a la que pertenece la muestra de entrada almacenada en *sample* y que se pretende clasificar. Si se configura el parámetro *returnDFVal* a *true*, la función devolverá la llamada función de decisión que es la distancia de la muestra al margen hallado en el entrenamiento.

- **get_support_vector ():** Devuelve el número de vectores de soporte.

La implementación de esta parte en solución se realiza únicamente insertando las partes pertinentes (creación del clasificador, el entrenamiento de mismo y la clasificación de las muestras) que se pueden ver en negrita en el pseudocódigo de la solución:

```
CREAR CLASIFICADOR SVM
OBTENER MUESTRAS POSITIVAS Y NEGATIVAS
SE CALCULAN LOS DESCRIPTORES HOG DE LAS MUESTRAS POSITIVAS Y SE ETIQUETAN COMO -
SE CALCULAN LOS DESCRIPTORES HOG DE LAS MUESTRAS NEGATIVAS Y SE ETIQUETAN COMO +
SE ENTRENA EL CLASIFICADOR CON LAS MUESTRAS POSITIVAS Y NEGATIVAS
SE CALCULAN Nº DE VECES QUE SE PUEDE REDUCIR LA IMAGEN -> ITERACIONES
PARA I = 1, MIENTRAS QUE I MENOR ITERACIONES ENTONCES
       /* EN LA PRIMERA ITERACIÓN NO SE REALIZA EL REESCALADO */
       SI I MAYOR QUE 1 ENTONCES
              DIEZMAR DIMENSIONES DE LA IMAGEN SEGÚN UN FACTOR
       FIN SI
       PARA Y = 0, MIENTRAS QUEDE IMAGEN POR RECORRER EN VERTICAL ENTONCES
```

```
            PARA X = 0, MIENTRAS QUEDE IMAGEN POR RECORRER EN HORIZONTAL ENTONCES

                DEFINIR RECTANGULO CON TAMAÑOS IGUAL A VENTANA DESLIZANTE

                RECORTAR VENTANA DE LA IMAGEN ESCANEADA EN LA COORDENADA X, Y

                CALCULAR DESCRIPTORES DE LA VENTANA

                CLASIFICAR DESCRIPTORES

                SI ES MUESTRA POSITIVA ENTONCES
                    SE ALMACENA LA IMAGEN EN UN VECTOR DE POSITIVOS
                    SE RECUADRA LA MUESTRA EN UN RECTANGULO VERDE
                FIN SI

                X = X + PASO_HORIZONTAL

            FIN PARA

            Y = Y + PASO_VERTICAL

        FIN PARA

FIN PARA
```

A la salida del módulo de SVM se obtendrá finalmente si la imagen hallada por la ventana deslizante y la pirámide se corresponden con un vehículo o no. En este momento ya se ha realizado la clasificación de características y, en adelante, da comienzo el seguimiento de las mismas a través de la escena.

3.4.2 Seguimiento de vehículos

El siguiente módulo tras la extracción y detección de vehículo es el seguimiento de los mismos. Este se realiza a través de dos submódulos: detección de esquinas, que se encarga de recopilar ciertos puntos de las detecciones y flujo óptico, que realiza el seguimiento de dichos puntos durante el transcurso de los frames.

Detector de esquinas

En esta solución se ha optado por realizar un seguimiento a través de los puntos más representativos del objeto. Estos puntos pueden corresponderse con variaciones grandes y bruscas de luminancia en la imagen, es decir, las **esquinas** del objeto. Además los vehículos cuentan con cierta forma y disposición de sus aristas que es bastante representativa y diferente del resto de los objetos en la escena.

Para hallar las esquinas de un objeto se ha recurrido a la función *goodFeaturesToTrack* que OpenCV proporciona. Esta función está basada en el algoritmo detector de esquinas de J. Shi y C. Tomasi., que a su vez es una particularización del detector de Harris. El detector de esquinas de Shi y Tomasi está explicado en detalle en el apartado 2.2.1.

La función *goodFeaturesToTrack* utilizada tiene los siguientes parámetros:

- **image**: Imagen de entrada en escala de grises en la que se va a realizar la detección de esquinas.
- **corners**: Vector parámetro de salida que contiene las coordenadas de las esquinas detectadas en *image*.
- **maxCorners**: Número máximo de esquinas que se pueden detectar. Su valor límite es de 20.
- **qualityLevel**: Parámetro que se utiliza para controlar la calidad de las esquinas detectadas. Este valor es multiplicado por el resultado de la función de Harris o por el valor mínimo del autovalor (ver parámetro *userHarrisDetector*). Aquellas esquinas con valor menor a dicha multiplicación son rechazadas. Se establece su valor 0.001, el recomendado por la documentación.
- **minDistance**: mínima distancia posible entre las esquinas encontradas.
- **mask**: Máscara opcional para delimitar una región de interés.
- **blockSize:** Tamaño del bloque para hallar la matriz de covarianza sobre cada píxel vecino.
- **useHarrisDetector:** Si su valor es *true* se utiliza el detector de Harris previamente. Si es *false* se utiliza la detección de esquinas por el autovalor mínimo.
- **k:** Parámetro libre del detector de Harris.

Como muestra del funcionamiento de esta función en la figura 30 se muestra la imagen de un coche resultado de la clasificación positiva del módulo SVM y donde se resalta en rojo las esquinas del mismo halladas.

Figura 30. Puntos hallados por el detector de Shi-Tomasi. La imagen se ha aumentado en tamaño para mejor visualización. Fuente propia.

Se puede observar en la figura 30 como no todas las esquinas del coche son halladas. Por ejemplo, los bordes superiores del coche y el retrovisor de la derecha no han sido reconocidos como esquinas. Esto puede deberse a una mala configuración de los parámetros antes descritos.

La implementación de esta fase en la solución se muestra en el pseudocódigo siguiente, donde se muestra en negrita lo relativo a las detecciones de la esquinas. Únicamente es reseñable el hecho de calcular y almacenar las coordenadas de las esquinas para cada imagen positiva obtenida del clasificador

```
CREAR CLASIFICADOR SVM

OBTENER MUESTRAS POSITIVAS Y NEGATIVAS

SE CALCULAN LOS DESCRIPTORES HOG DE LAS MUESTRAS POSITIVAS Y SE ETIQUETAN COMO -

SE CALCULAN LOS DESCRIPTORES HOG DE LAS MUESTRAS NEGATIVAS Y SE ETIQUETAN COMO +

SE ENTRENA EL CLASIFICADOR CON LAS MUESTRAS POSITIVAS Y NEGATIVAS

SE CALCULAN Nº DE VECES QUE SE PUEDE REDUCIR LA IMAGEN -> ITERACIONES

PARA I = 1, MIENTRAS QUE I MENOR ITERACIONES ENTONCES

        /* EN LA PRIMERA ITERACIÓN NO SE REALIZA EL REESCALADO */
        SI I MAYOR QUE 1 ENTONCES
                DIEZMAR DIMENSIONES DE LA IMAGEN SEGÚN UN FACTOR
        FIN SI

        PARA Y = 0, MIENTRAS QUEDE IMAGEN POR RECORRER EN VERTICAL ENTONCES

                PARA X = 0, MIENTRAS QUEDE IMAGEN POR RECORRER EN HORIZONTAL ENTONCES

                        DEFINIR RECTANGULO CON TAMAÑOS IGUAL A VENTANA DESLIZANTE

                        RECORTAR VENTANA DE LA IMAGEN ESCANEADA EN LA COORDENADA X, Y

                        CALCULAR DESCRIPTORES DE LA VENTANA

                        CLASIFICAR DESCRIPTORES

                        SI ES MUESTRA POSITIVA ENTONCES
                                SE ALMACENA LA IMAGEN EN UN VECTOR DE POSITIVOS
                                SE RECUADRA LA MUESTRA EN UN RECTANGULO VERDE
                        FIN SI

                        X = X + PASO_HORIZONTAL

                FIN PARA

                Y = Y + PASO_VERTICAL

        FIN PARA

FIN PARA

CREAR VECTOR DE COORDENADAS DE ESQUINA PARA FRAME ACTUAL

PARA CADA IMAGEN DEL VECTOR DE POSITIVOS HACER

        CALCULAR LAS ESQUINAS DE LA IMAGEN

        ALMACENAR LAS ESQUINAS EN UN VECTOR DE ESQUINAS

FIN PARA
```

Tracking por Flujo Óptico

El procedimiento utilizado para realizar el seguimiento es el algoritmo de **Flujo Óptico** de Lukas-Kanade piramidal, cuyo funcionamiento se ha desarrollado en detalle en el

apartado 2.3.1. Aquí se detallan sólo los aspectos funcionales relativos a la implementación.

Para implementar el seguimiento por flujo óptico en esta solución propuesta se recurre a las librerías de OpenCV. OpenCV permite calcular el flujo óptico de Lukas-Kanade piramidal mediante la función *calcOpticalFlowPyrLK* del paquete *videoAnalysis*. Los parámetros de esta función son los siguientes:

- **prevImg**: Imagen de entrada en escala de grises de dónde se han obtenido los puntos (párametro *prevPts*) que se buscaran en la siguiente imagen (párametro *nextImg*).
- **nextImg**: Imagen de entrada en escala de grises siguiente a *prevImg* donde el algoritmo tendrá que buscar los puntos de *prevPts*.
- **prevPts**: Puntos de entrada cuyas coordenadas se corresponde a los objetos que se desean buscar en *nextImg*.
- **nextPts**: Parámetro de salida que se corresponde con los puntos equivalentes a **prevPts** y que han sido hallados en *nextImg*.
- **status y err**: Variables de salida utilizadas para el control de errores.
- **winSize**: Tamaño de la ventana deslizante para el algoritmo de imágenes piramidales.
- **maxLevel**: Número de imágenes piramidales utilizadas.

Por tanto esta función calcula el flujo óptico entre dos imágenes de entrada consiguiendo la estimación del posible movimiento de los objetos previamente segmentados. Las imágenes *prevImg* y *nextImg* no tienen por qué ser consecutivas, si bien se recomienda que lo sean para un cálculo preciso. En ocasiones los objetos en una imagen se mueven de forma tan lenta que apenas generan cambios entre frames inmediatamente consecutivos, lo que lleva a no generar flujo óptico y no estimación del movimiento, y por ende, el no seguimiento de dicho objeto. Por otro lado, evitar el procesamiento frame a frame disminuye los tiempos de ejecución.

En la figura 31 se puede ver dicho funcionamiento. En azul se muestran los puntos (*prevPts*) hallados con el detector de esquinas en el frame *prevImage*. En rojo los puntos estimados (*nextPts)* por el flujo óptico calculado en el frame actual, es decir, *nextImage*.

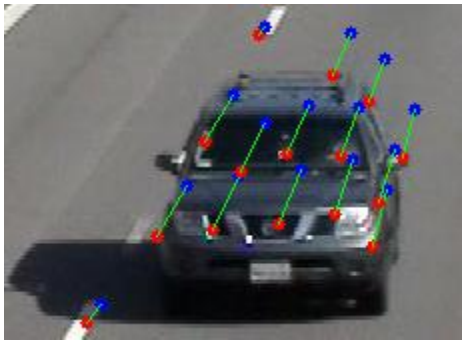


Figura 31. Ejemplo de estimación de flujo óptico. Fuente propia.

De cara a esta solución propuesta, hace falta actualizar la estructura de los tracks de seguimientos previamente inicializada de manera constante. Durante la actualización de la estructura de tracks, se calcula el flujo óptico en el nuevo frame, partiendo de los puntos característicos obtenidos del track anterior en caso de que los hubiera. Es decir, **se calcula un flujo óptico para cada detección de vehículo hallada**, ya que cada detección tiene sus propios puntos característicos. Es decir, si se han detectado cuatro vehículos en la imagen, se hallará el flujo óptico para cada uno de los cuatro. Dentro de la escena, durante la actualización de los tracks se pueden encontrar tres casos:

- No se ha realizado tracking del vehículo anteriormente.
- Se ha realizado tracking del vehículo anteriormente y se actualiza su nueva posición en el frame.
- El vehículo ha desaparecido.

Para implementar en la solución este parte de tracking se realizan las inserciones pertinentes en el pseudocódigo de la aplicación.

```
CREAR CLASIFICADOR SVM

OBTENER MUESTRAS POSITIVAS Y NEGATIVAS

SE CALCULAN LOS DESCRIPTORES HOG DE LAS MUESTRAS POSITIVAS Y SE ETIQUETAN COMO -

SE CALCULAN LOS DESCRIPTORES HOG DE LAS MUESTRAS NEGATIVAS Y SE ETIQUETAN COMO +

SE ENTRENA EL CLASIFICADOR CON LAS MUESTRAS POSITIVAS Y NEGATIVAS

SE CALCULAN Nº DE VECES QUE SE PUEDE REDUCIR LA IMAGEN -> ITERACIONES

PARA I = 1, MIENTRAS QUE I MENOR ITERACIONES ENTONCES

       /* EN LA PRIMERA ITERACIÓN NO SE REALIZA EL REESCALADO */
       SI I MAYOR QUE 1 ENTONCES
              DIEZMAR DIMENSIONES DE LA IMAGEN SEGÚN UN FACTOR
       FIN SI

       PARA Y = 0, MIENTRAS QUEDE IMAGEN POR RECORRER EN VERTICAL ENTONCES

              PARA X = 0, MIENTRAS QUEDE IMAGEN POR RECORRER EN HORIZONTAL ENTONCES

                     DEFINIR RECTANGULO CON TAMAÑOS IGUAL A VENTANA DESLIZANTE

                     RECORTAR VENTANA DE LA IMAGEN ESCANEADA EN LA COORDENADA X, Y
```

```
                        CALCULAR DESCRIPTORES DE LA VENTANA

                        CLASIFICAR DESCRIPTORES

                        SI ES MUESTRA POSITIVA ENTONCES
                                SE ALMACENA LA IMAGEN EN UN VECTOR DE POSITIVOS
                                SE RECUADRA LA MUESTRA EN UN RECTANGULO VERDE
                        FIN SI

                        X = X + PASO_HORIZONTAL

                FIN PARA

                Y = Y + PASO_VERTICAL

        FIN PARA
FIN PARA

CREAR VECTOR DE COORDENADAS DE ESQUINA PARA FRAME ACTUAL

PARA CADA IMAGEN DE VECTOR_POSITIVOS HACER

        CALCULAR LAS ESQUINAS DE LA IMAGEN

        ALMACENAR LAS ESQUINAS EN UN VECTOR_ESQUINAS

FIN PARA

SI NO ES EL PRIMER FRAME HACER

        PARA CADA IMAGEN DEL VECTOR DE POSITIVOS HACER

                CALCULAR FLUJO ÓPTICO CON FRAME ANTERIOR -> VECTOR_ESQUINAS_ANTERIOR

                COMPARAR VECTOR_ESQUINAS CON VECTOR_ESQUINAS_ANTERIOR

                ACTUALIZAR ESTRUCTURA DE TRACKS (ELIMINAR)

        FIN PARA

FIN SI
```

3.4.3 Pruebas y experimentación

Una vez implementada la solución se decidió hacer pruebas para medir la eficiencia de su funcionamiento, y más concretamente la detección. El procedimiento de experimentación fue ir probando cada módulo secuencialmente uno detrás de otro. El motivo de obrar de esta manera es la dependencia de cada módulo de su antecesor: si no se realiza una buena detección o esta provoca muchos fallos, el módulo de seguimiento cometerá todos los errores que se vengan arrastrando ante las malas detecciones.

Así pues las primeras pruebas se realizaron probando el módulo de extracción y detección de vehículos sobre pequeñas muestras de pocos frames (figura 32). El objetivo de este procedimiento es medir la calidad de la detección de vehículos, que será clave para después realizar un buen seguimiento.

Figura 32. Vídeo utilizado parar probar la eficiencia de detección. Fuente [39].

En la tabla 1 se observan los resultados de la prueba de detección realizada.

Fuente	Frames	Verdaderos positivos	Falsos positivos	Falsos negativos	Tiempo de ejecución
Vídeo	21	32	17	7	6.7 segundos

Tabla 1. Pruebas realizadas para medir la eficiencia de la detección de la solución

En el caso del vídeo, se puede observar que se realiza una detección algo mejorable si se ajustan los parámetros del clasificador y de los descriptores HOG. Por otro lado el tiempo de ejecución alcanza límites insostenibles, ya que apenas un segundo de vídeo tarda seis veces más en procesarlo. Este aspecto es clave pues al realizar la experimentación con el módulo de seguimiento este sufre aún más demora al intentar realizar el tracking de las detecciones.

3.4.4 Ventajas y desventajas de la solución

Una vez realizada la implementación de la solución y sus pruebas de funcionamiento pertinentes se divisaron varios aspectos sobre el funcionamiento de la misma y los cuales se comentan a continuación.

En primer lugar, y como ya se ha comentado, OpenCV ofrece la posibilidad de realizar la detección por SVM de forma interna en la misma clase donde se obtienen los descriptores HOG (*HogDescriptor*) mediante la función *detectMultiscale*. Al utilizar la función *detectMultiscale* se observó que el funcionamiento era incorrecto además de lento. Apenas se producían detecciones, por lo que la tasa de falsos negativos era demasiada alta. También el tiempo de ejecución era demasiado elevado para el procesado de cada frame, por lo que el uso de esta función era prácticamente inviable. Se propuso el uso de librerías externas que modelarán un detector externo SVM como *LibSVM*, pero finalmente se optó por la utilización de la clase *Svm* propia de OpenCV siendo está mucho más completa y robusta respecto a configuración que la opción

interna utilizada antes, que requería instalación externa. Tras esta elección, la detección mejoró respecto a la tasa de falsos negativos y falsos positivos, y el tiempo de ejecución disminuyó aunque de forma muy reducida.

Aunque se haya conseguido que la extracción y detección de vehículos sea buena, también es necesario configurar los parámetros para cada caso en concreto, lo que resta independencia al proceso teniendo que ser configurado para cada situación. Más allá de eso, se encontraron inconvenientes en el módulo de seguimiento. El cálculo del flujo óptico realizado para los puntos de cada vehículo detectado es costoso y hace que aumente aún más el tiempo de ejecución. Si hay demasiadas detecciones porque hay muchos vehículos en la vía el algoritmo tendrá que calcular demasiados flujos ópticos y será inviable. Además, por otro lado, debido a la tipología de las imágenes de carreteras, vehículos y vías que se están procesando, se cometen errores a la hora de que el algoritmo encuentre correspondencia entre los puntos de un frame n a los puntos de un frame n+1, ya que hay muchos elementos en la escena que pese a pertenecer a objetos distintos se parecen mucho. Esto conlleva a que el algoritmo asigne de forma errónea la posición final del vehículo en los frames siguientes, dando como resultado un mal encuadre del vehículo durante el seguimiento del mismo.

Por otro lado, la estructura de seguimiento de los vehículos en la etapa de diseño resultó inestable ante un número elevado de entradas y salidas de vehículos llevando incluso al colapso de la aplicación en algunas escenas. Esto se produce porque aunque el vehículo hubiese salido de escena el algoritmo de tracking seguía asignando los puntos del vehículo que había salido a cualquier otro objeto de la escena, dando lugar a falsos positivos y entorpeciendo la ejecución de la actualización del tracking al no liberar memoria.

En vista de los aspectos comentados, **esta solución no es la mejor opción para un control vehicular fluido en tiempo real**, siendo esto último un requisito primordial. No obstante, si es posible un mejor funcionamiento de esta opción con un hardware y un equipo más potente, ya que no hay que olvidar que pese a que la detección es lenta, si es cierto que es fiable permitiendo además la detección con cámara no estática.

3.5 Sustracción de background y seguimiento por modelos de apariencia

La segunda solución propuesta está basada en la sustracción de background como método de extracción y detección de vehículos, y el seguimiento de los mismos por modelos de apariencia como método de tracking. La estructura de procesado de la solución propuesta se muestra en la figura 33.

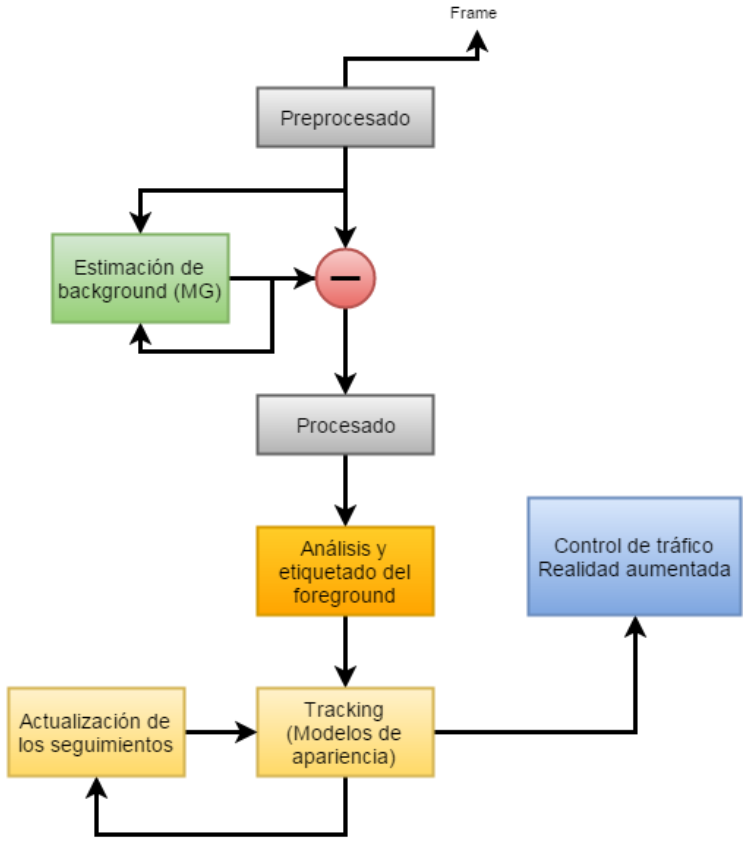


Figura 33. Estructura de procesado de la segunda solución propuesta

3.5.1 Preprocesado

Para realizar la sustracción de background correctamente es necesario adecuar las imágenes de entrada al módulo. Para ello se opta por un preprocesado. Este consiste, en primer lugar, en disminuir el tamaño de los frames a una resolución más llevadera para el procesado posterior pero sin comprometer la detección de anomalías mediante la función *imresize* de OpenCV. Con este cambio de resolución se buscar evitar saturar el procesamiento y reducir el tiempo de ejecución. En segundo lugar se realiza un **desenfoque gaussiano** sobre la imagen, como se observa en la figura 34. El objetivo de este desenfoque es eliminar el posible ruido residual que aparece en los vídeos debido a las componentes de alta frecuencia [40]. Estas componentes pueden complicar la estimación del background ya que los píxeles afectados por el ruido de alta frecuencia a menudo pasan como objetos en movimiento debido a la oscilación que los provocan. Este desenfoque se realiza con la función *blur*, proporcionada de nuevo por las librerías de OpenCV, y un ejemplo de ello se puede observar en la figura 34.

Figura 34. Ejemplo de desenfoque gaussiano. Fuente [40].

3.5.2 Estimación y sustracción de background

Una vez procesado el frame, este pasa a ingresar en el módulo de estimación y sustracción de background. Para estimar el background se ha optado por utilizar el algoritmo de **Mezclas de Gaussianas**, cuyo funcionamiento se puede encontrar detallado en el apartado 2.2.4. La elección de este tipo de algoritmo de sustracción por encima de los otros propuestos se fundamenta en que la técnica de mezclas de gaussianas ofrece una actualización constante del background teniendo en cuenta los frames anteriores. Esto se traduce, como se ha explicado anteriormente, en que si por ejemplo un vehículo para y estaciona en la escena y no se vuelve a mover, pasará a ser parte del background y no se reconocerá. También pasan a ser parte del background aquellos objetos del fondo que se mueven de forma cíclica y repetitiva en pequeñas áreas, como pueden ser árboles o banderas agitadas por el viento.

Para realizar la estimación de background por mezclas de gaussianas OpenCV ofrece las siguientes clases y estructuras de su paquete *video*, que contiene algoritmos y técnicas para el análisis de movimiento y seguimiento:

- ***MOG (Mixtures of Gaussians)***: Se trata de un método basado en la modelación progresiva y constante del background a través del uso de varias funciones de densidad de tipo gaussiana. La codificación de esta clase está basada en el algoritmo de la publicación [41].
- ***MOG2*****:** Es el método elegido para desarrollar la solución propuesta en este caso. Sigue la misma filosofía que el método del punto anterior sólo que incorpora mejoras como la detección y posible eliminación de sombras. Su funcionamiento está basado en la publicación [42].
- ***GMG*****:** En las versiones más recientes de OpenCV este algoritmo ha caído en desuso aconsejando la utilización de los dos anteriores.

Más concretamente, la clase *MOG2* contiene los siguientes atributos y funciones principales:

- **bgmodel**: Variable que contiene el modelo de background estimado y que se actualiza para cada frame.
- **nframes**: Número de frames anteriores los cuáles se tendrán en cuenta a la hora de estimar el background.
- **nmixtures**: Número de componentes gaussianas permitido para cada píxel. Suele oscilar entre 3 y 5 componentes.
- **varThreshold**: Se corresponde con el parámetro *k* explicado anteriormente. Es un umbral que se aplica sobre la distancia que decide si un píxel se corresponde con el modelo background hallado o no. Regula la sensibilidad del algoritmo a la hora de estimar el background.
- **bShadowDetection**: Flag que regula la detección de sombras si su valor es *true*.
- **nShadowDetection:** Es el valor de luminancia comprendido entre 0 y 255 que se le dará a las sombras detectadas en el foreground final en caso de que el flag *bShadowDetection* esté activado. Para una eliminación total de las sombras esta variable valdrá 0.
- **operator** (image, foreground): Función principal de la clase que devuelve el foreground tras realizar estimar un background (con los parámetros anteriores configurados) y sustraerlo a la imagen de entrada *image*. El foreground es una imagen binaria donde los elementos en movimiento son representados en blanco frente al background que es representado en negro.
- **getBackgroundImage ():** Función que devuelve una imagen de la estimación de background realizada en el momento que es llamada.

Por tanto, la clase *MOG2* a través de su función *operator* permitirá obtener la máscara de foreground con los vehículos en movimiento de la vía. Es relevante destacar la detección de sombras incorporada en esta clase (parámetro *bShadowDetection* a *true*). La mayoría de los objetos proyectan sombras sobre el suelo o paredes en mayor o menor medida dependiendo desde dónde les incida la luz. Además los píxeles de estas sombras se mueven conjuntamente con el objeto, por lo que se identificarán como foreground, como se muestra en la figura 36 izquierda. Es importante intentar eliminar dichas sombras porque en el resultado final pasarán como falsos positivos, aumentando así la tasa de error. Dado que la clase elegida ofrece la posibilidad de eliminar dichas sombras, se hará uso de esta opción tal y como se observa en la figura 35 derecha donde se ha eliminado las sombras detectadas en la figura 35, izquierda.

Figura 35. A la izquierda, componente segmentada con sombras detectadas en gris. A la derecha, componente segmentada pero con sombras eliminadas. Fuente [39].

En estas máscaras los objetos segmentados aparecen como componentes en forma de "manchas" blancas (*blobs* es el término en inglés utilizado por la mayoría de las publicaciones) sobre una imagen en escala de grises.

Si hay partes de los objetos segmentados que comparten colores con el background estimado, puede ocurrir que estas partes se confundan como background y que dichas componentes aparezcan como fragmentadas o discontinuas con agujeros en sus interiores, como se muestra en la figura 36 izquierda. En un caso más crítico, si este problema no se soluciona es posible que en etapas posteriores de procesado un vehículo sea detectado como dos vehículos debido a esta fragmentación. Para solucionar este problema y conseguir que estas manchas no estén tan fragmentadas y si más uniformes, se realiza un **procesado** a la máscara de foreground que consiste en las operaciones morfológicas de **dilatación** y **erosión**. En ocasiones, solo es necesario realizar una dilatación sin necesidad de alguna otra operación para que las componentes segmentadas estén algo más cerradas por dentro, tal y como se muestra en la figura 36 derecha. La función *dilate* de OpenCV permite realizar una dilatación morfológica sencilla en una imagen binaria como la que contiene la máscara de foreground.

Figura 36. A la izquierda, componente segmentada sin aplicar dilatación morfológica. A la derecha, la misma componente pero dilatada morfológicamente

Por tanto, el objetivo de este procesado de la máscara de foreground es adecuarla y perfeccionar al máximo posible para la siguiente etapa, donde se realizará el análisis y etiquetado de las componentes de está máscara.

3.5.3 Análisis y etiquetado del foreground

Tras realizar la extracción del foreground y su posterior procesamiento, es necesario identificar cada componente extraída e identificarla de forma única del resto. Esta tarea de etiquetado (*labeling* es el término utilizado en la mayoría de las publicaciones) es una fase preparatoria para el posterior seguimiento de los vehículos a través de las escenas que se suceden.

Para realizar la identificación y etiquetado de las componentes se ha propuesto la utilización de las librerías ***cvBlob*** [34]. Dentro de las librerías se puede encontrar un módulo de análisis y etiquetado de componentes basado en el seguimiento del contorno que está basado en el propuesto en la publicación [35]. Su funcionamiento es el siguiente:

La imagen se recorre de arriba a abajo y de izquierda a derecha buscando aquellos píxeles de color negro que representarán los objetos del foreground sobre un fondo color blanco. Esto último, sin embargo, se encuentra invertido en la aplicación: los objetos segmentados estarán en blanco y el background en negro. Pero se ha decidido mantener el criterio original de la publicación en esta explicación para que haya cierta correspondencia con las figuras mostradas y con la publicación referenciada.

De forma general, se puede dividir el método en cuatro grandes pasos que están representados en la figura 37:

A. Cuando se encuentra un punto A de un contorno externo se realiza un seguimiento a través de todo el contorno hasta que se llega de nuevo a A. Todos estos puntos se les asigna la misma etiqueta que al punto A.

B. Para cada punto A' del contorno externo A previamente hallado se seguirá escaneando en la misma línea horizontal (de izquierda a derecha) para encontrar todos los píxeles negros interiores y asignarlos a la misma etiqueta que A' y A.

C. Cuando se llega a un punto de un contorno interno, al que se llamará B, se le asigna la misma etiqueta que al contorno externo. Además a todos los puntos que conforman el contorno interno se le asignará la misma etiqueta que al punto B.

D. Para cada punto B' del contorno interno B previamente hallado se seguirá escaneando en la misma línea horizontal (de izquierda a derecha) para encontrar todos los píxeles negros interiores y asignarlos a la misma etiqueta que B' y B.

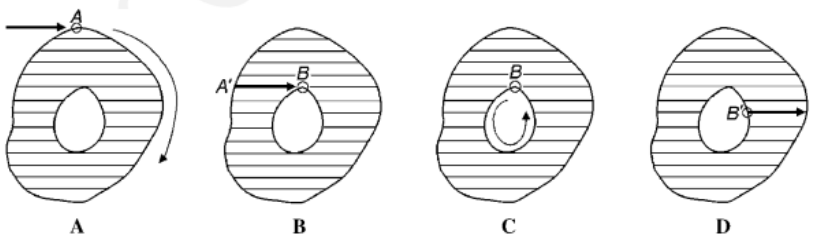


Figura 37. Escaneo de una región en 4 pasos. Fuente [41].

Estos cuatro pasos pueden ser fácilmente reducidos a tres. Por simplicidad, en la figuras se establece que cuando píxel negro vale 1 está desetiquetado y vale ∆ cuando está etiquetado. Los tres pasos serían los siguientes para un punto P dado, tomando como referencia la figura 38:

I. Si el punto P está desetiquetado y el píxel superior es blanco, entonces P debe pertenecer a un contorno exterior. En este caso, se le asigna una etiqueta C a ese punto y se ejecuta un algoritmo de seguimiento de contorno (que se detalla más adelante) para encontrar dicho contorno externo y asignarle la misma etiqueta C.

II. Si el punto bajo P es blanco y no está marcado, P debe de pertenecer a un nuevo contorno interior. En este momento hay dos posibilidades:

    a. En caso de que P ya esté etiquetado, P pertenecería a un contorno interior, tal y como se muestra en la figura 38.A

    b. En caso de que P no esté etiquetado, se le asignaría la misma etiqueta que su punto anterior (su píxel vecino a su izquierda N), tal y como se muestra en la figura 38.B. En cualquier caso, se procede como en el paso I anterior, es decir, se ejecuta un algoritmo de seguimiento de contorno para encontrar el contorno interno y asignarle la misma etiqueta que al resto de los puntos del contorno.

III. En caso de que P no cumpla las características del paso 1 o 2, se le asigna la misma etiqueta que su punto anterior.

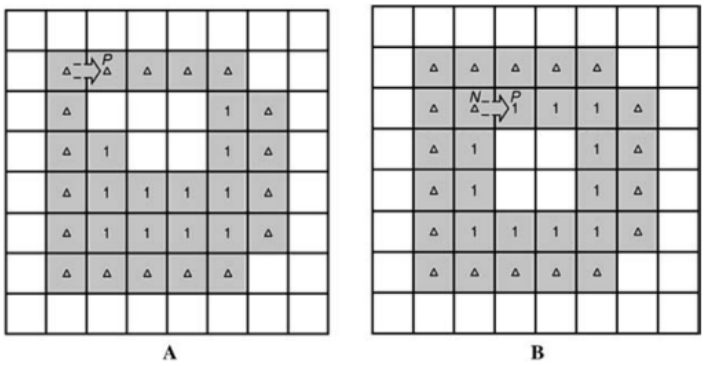


Figura 38. Etiquetado de cada píxel. Fuente [41].

Además, en el momento en el que el punto de una esquina de un contorno externo es hallado se marcan sus píxeles circundantes con un número negativo. De esa manera, como se ilustra en la figura 39, los píxeles blancos alrededor del píxel Q se marcan negativos mientras se escanea esa línea, incluyendo el píxel blanco de debajo de Q. No obstante, los píxeles blancos encontrados bajo el primer punto de un contorno interior (punto P en la figura) no serán marcados negativamente hasta que se complete el contorno entero.

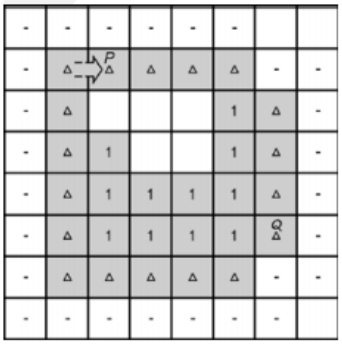


Figura 39. Etiquetado de cada pixel. Fuente [41].

El mecanismo que se está utilizando para encontrar un contorno entero desde un punto S dado es el llamado "*contour tracing*" o seguimiento de contorno. Pero antes de realizar "*countour tracing*" se ejecuta otro procedimiento llamado "*tracer*" o trazador. Si el trazador identifica el punto S como un punto aislado, se alcanza el final del mecanismo de seguimiento de contorno. En cualquier otro caso, el seguimiento de contorno devolverá el punto siguiente al punto S, llamado punto T. Entonces se procede a ejecutar el trazador para encontrar el punto siguiente a T hasta que se cumplan dos condiciones:

1. El trazador devuelve el punto inicial S.
2. El siguiente punto a S es el punto T.

En la figura 40 se puede ver un ejemplo de cómo funciona este seguimiento de contorno. El camino trazado en este caso sería STUTSVWVS y de esta forma quedaría definido el contorno.

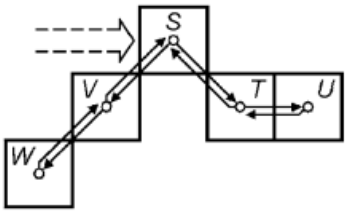


Figura 40. Funcionamiento del *"contour tracing"*. Fuente [41].

Dentro del seguimiento del contorno, el trazador ("*tracer*") decide a qué píxel dirigirse para ir definiendo el contorno. Su objetivo es encontrar el siguiente punto del contorno entre sus 8 píxeles circundantes. Dado un punto P, sus ocho píxeles circundantes son asignado con índices desde el número 0 hasta el 7 como se muestra en la figura 41.A.

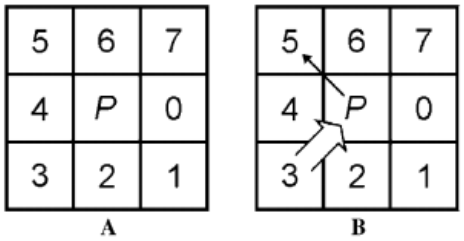


Figura 41. Orden de elección de píxeles del trazador. Fuente [41].

La búsqueda del píxel se hará en el sentido de las agujas del reloj desde un punto inicial que se establece de la siguiente forma.

Si P es el punto de comienzo de un contorno, la posición 7 será la inicial sabiendo que el punto por encima de P es un píxel blanco y su siguiente posición sería la 7. Sin embargo, cuando P es el comienzo de un contorno interno la posición inicial sería la 3, ya que el punto por debajo de P sería un píxel blanco y su siguiente punto sería el de la posición 3. Por otro lado, si existe un punto de contorno anterior y toma la posición 3, por ejemplo, entonces la búsqueda comenzará en el píxel número 5 ya que ya se habrá pasado por el píxel número 4, como se muestra en la figura 41.B. Como norma general, si P no es el inicio de un contorno se establecerá el punto inicial de búsqueda del siguiente punto mediante la fórmula d + 2 (mod 8), siendo d el número de la posición del anterior punto del contorno. Una vez establecido el punto inicial de búsqueda, se comienza a localizar el siguiente píxel negro en el sentido de las agujas del reloj. Si no se encuentra ningún píxel negro se estará ante un punto aislado. Durante este proceso se realiza el procesado de marcado negativo de aquellos píxeles blancos.

Las variables y funciones más importantes relativas a este algoritmo de la librería cvBlob son:

- **Struct CvBlob**: Se trata de una estructura que contiene la información sobre una componente (*blob*). Dentro de esta estructura los atributos más importantes son:
  - **label**: Contiene la etiqueta única de la componente a la que ha sido asignada.
  - **centroid:** Almacena la coordenada del centro de la componente.
  - **contour e internalContours**: Almacena la cadena de contornos interiores y exteriores halladas para esa componente.

- **Map <CvLabel, CvBlob>**: Se trata de un mapa (estructura compuesta de clave y valor) que contiene parejas de etiquetas y componentes. Cada entrada del mapa constituye una pareja siendo la etiqueta el identificador de cada componente. Es la estructura principal que se irá actualizando en cada frame cuando se realice el etiquetado de las componentes del foreground.

- **cvLabel** (img, imgOut, blobs): Función principal y la más importante del algoritmo que realiza el etiquetado de todas las componentes encontradas en la imagen binaria *img*, tal y como se ha detallado anteriormente. Devuelve las etiquetas en la variable *blobs* y la imagen con las componentes etiquetadas en la variable *imgOut*.

- **cvFilterLabels** (imgIn, imgOut, blobs): Función que devuelve dibujadas en la imagen *imgOut* las componentes etiquetadas pasadas en el parámetro *blobs*, partiendo de la imagen de entrada *imgIn*.

- **cvGetLabel** (img, x y): Función que devuelve la etiqueta de la imagen *img* situada en las coordenadas *x* e *y*.

- **cvFilterByArea** (blobs, minArea, maxArea): Función que filtra aquellas componentes almacenadas en el parámetro *blobs* que tengan un área comprendida entre los parámetros *minArea* y *maxArea*.

- **cvRenderBlob** (imgLabel, blob, imgSource, imgDest, mode, color, alpha): Dibuja la componente o la información de la componente almacenada en *blob* sobre la imagen *imgDest* de un determinado color con posibilidad de transparencia. Se puede dibujar su centro, su ángulo o su área.

Un ejemplo del uso de estas funciones se puede observar en la figura 42, donde se compara la máscara de foreground obtenida en imagen binaria (figura 42, imagen de arriba) y sus correspondientes componentes tras el análisis (figura 42, imagen de abajo). Las componentes se han filtrado previamente con la función *cvFilterByArea* y se han dibujado por colores con *cvRenderBlob*.

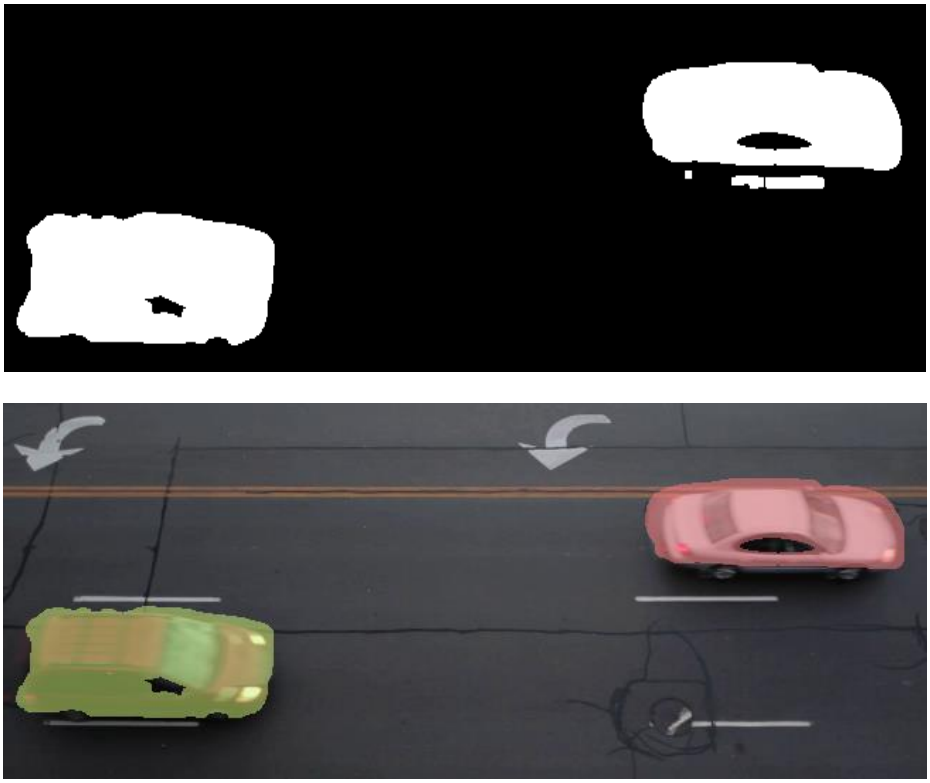


Figura 42. Arriba, máscara de foreground obtenida. Abajo, el análisis de las componentes de dicho foreground con filtro por área. Fuente [39].

Al final de todo este proceso todas las componentes conexas segmentadas del frame habrán sido analizadas y etiquetadas. Además aquellas componentes con pequeños contornos interiores que no se han conseguido cerrar el procesado anterior son etiquetadas al igual que el contorno exterior. Estas etiquetas serán importantes de cara al posterior módulo de seguimiento de los vehículos.

3.5.4 Tracking mediante modelos de apariencia

Tras el etiquetado de las componentes del foreground el siguiente paso es el seguimiento de las mismas a través de las escenas que se van sucediendo. De nuevo las librerías *cvBlob* ofrecen un sistema de tracking de modelos de apariencia con manejo de oclusión basado en la publicación referenciada [36].

En primer lugar, se entiende por oclusión, en el entorno de tracking de objetos, como la interrupción del seguimiento del objeto cuando este queda cubierto total o parcialmente por otro objeto que no se está siguiendo o que forma parte del background. Un ejemplo de oclusión se puede ver en la figura 43, donde se puede ver como una furgoneta oculta de forma parcial el seguimiento del grupo de las personas que se está realizando.

Figura 43. A la izquierda, componentes de las furgonetas y de las personas. A la derecha, oclusión de una furgoneta en el seguimiento de personas. Fuente [36].

De forma resumida, el método propone un algoritmo de tracking basado en la estructura que se muestra en la figura 44.

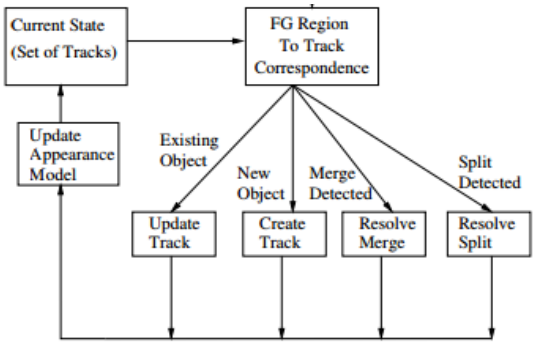


Figura 44. Estructura del sistema de tracking propuesto. Fuente [36].

En primer lugar se realiza la ya comentada sustracción de background para pasar después al análisis de las componentes obtenidas de la sustracción previa. Este análisis analiza cada una de las componentes del foreground halladas y las agrupa, las etiqueta y las identifica individualmente. De esta manera se obtendría un reconocimiento y una identificación de los objetos a seguir en la escena en el frame actual. Una vez hallados los objetos a seguir en la escena se deberá comprobar sus correspondencia, es decir, si se trata de un objeto reconocido en frames anteriores, si es un objeto nuevo o si se trata de una mezcla de varios objetos que se unen (*merge*) o se separan (*split*). Es en estos dos últimos casos donde se estaría dando la llamada oclusión.

Para hacer esta correspondencia se propone que cada región de foreground esté definida por un rectángulo cuyo centro coincida con el centro de dicha región. Dentro de ese rectángulo habrá una máscara que indica aquellos píxeles del rectángulo que pertenecen al foreground. En cada frame, el algoritmo intentará asociar cada rectángulo a algún 'track' ya existente, que también está representado con un rectángulo. Esta asociación se realizará mediante la construcción de una matriz de distancias entre los rectángulos del frame actual (columnas) y los rectángulos del 'tracks' (filas). Este criterio

de distancias se muestra en la figura 45. La distancia entre dos rectángulos A y B se define como la menor distancia desde el centro de A hasta el punto más próximo de B, o viceversa (figura 45, izquierda). En el caso de que el centro de algún rectángulo se encuentre dentro de la región del otro rectángulo la distancia será cero (figura 45, derecha).

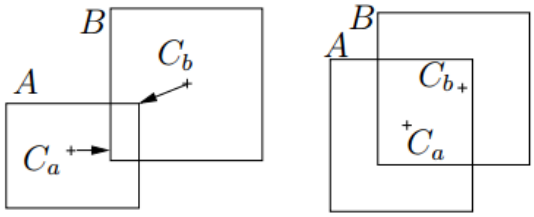


Figura 45. Criterio de distancias para el sistema de tracking propuesto. Fuente [36].

Llegados a este punto la matriz de distancias se binariza dando lugar a una matriz que realiza una correspondencia entre 'tracks' y regiones de foreground halladas. A la vista de dicha matriz se puede sacar en claro tres resultados posibles:

- Una matriz con la mayoría de sus elementos a no cero se corresponderá con una situación donde los 'tracks' y los rectángulos de foreground están ampliamente separados. Por ello, las columnas con resultados a cero dará lugar a la creación de nuevos 'tracks' ya que no se corresponde con ninguno de los existentes. Las filas con resultados a cero concuerdan con 'tracks' ya existentes pero que ya no son visibles
- En el caso de que varios 'tracks' se correspondan con un único objeto del foreground (valor a cero) se estará ante una situación donde varios objetos se están mezclando (merge), dando lugar a una posible oclusión. Esto dará lugar a que una columna tenga más de un resultado con no-cero.
- En el caso de que un solo 'track' se corresponda con varios objetos del foreground se estará ante una situación donde dos objetos se están dividiendo (Split). Por ello, una fila tendrá más de un resultado con no cero. Cuando esto ocurre y la distancia entre 'tracks' y objetos es la suficiente, se crearán nuevos 'tracks' correspondientes a los objetos que se han separado.

Un ejemplo de estas matrices se puede encontrar en la figura 46, donde el 1 marcará la correspondencia y el 0 la no correspondencia. La nomenclatura T se corresponde a los 'tracks' ya existentes, y la nomenclatura F se corresponde con las regiones del foreground.

- En el ejemplo A, se puede observar que cada región hallada en ese frame se corresponde un único 'track' de los ya existentes, por ello, en este caso sólo se actualizaría la información de dichos 'tracks'.
- En el ejemplo B, se observa que hay dos tracks (T1 y T2) para una región del foreground (F1). Esto es una situación de mezcla (merge) de objetos porque coinciden en la misma región del foreground. Además se observa que para la región F2 no hay ningún 'track' asignado, lo que daría lugar a la creación del mismo.
- En el ejemplo C, se observa que para un mismo 'track' (T1) hay asignado dos regiones del foreground (F1 y F2). Esto da lugar a una situación de división (Split) donde dos objetos que estuvieron en la misma región de foreground se han comenzado a separar, aunque de momento siguen estando asignados al mismo 'track'. Por otro lado se observa que el 'track' T2 no tiene ninguna región asignada, por tanto, se corresponde con un 'track' que ya no es visible porque está desapareciendo.

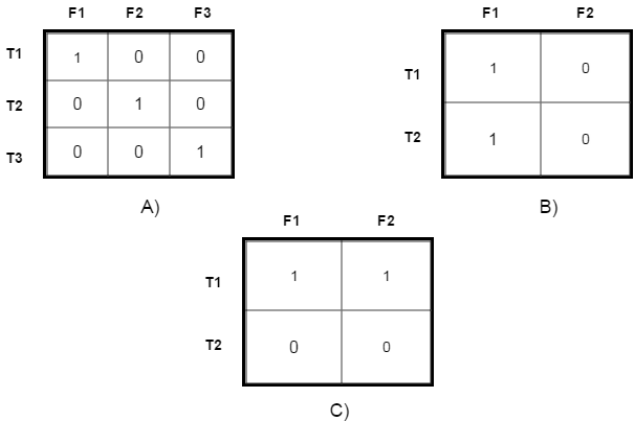


Figura 46. Posibles casos en la matriz de correspondencia del sistema de tracking propuesto. Fuente propia.

Una vez que un nuevo 'track' es creado, un modelo de apariencia de las mismas medidas que el rectángulo correspondiente a la región del foreground es establecido. El modelo de apariencia estará basado en un modelo de color RGB junto a una máscara de probabilidad. El modelo de color muestra la apariencia de cada píxel del objeto, mientras que la máscara de probabilidad registra la probabilidad de que dicho píxel pertenezca al objeto. Inicialmente, cuando el 'track' es creado, el modelo se inicializa copiando los píxeles del rectángulo del foreground y estableciendo las probabilidades a 0.4 a aquellos píxeles que se correspondan al objeto y 0 a los que no se correspondan.

Este modelo se irá actualizando en cada frame siempre y cuando el objeto continúe correspondiéndose con ese track. El modelo de color es actualizado mediante la mezcla de los valores de los píxeles y probabilidades del frame actual con los del frame anterior. Esta mezcla se puede regular mediante un factor cuyo valor suele ser 0,95. En el caso de

que aparezcan problemas de oclusión cuando en las regiones de foreground los objetos se mezclan se utilizarían dichos modelos de apariencia para resolver la ambigüedad siendo en parte posible el seguimiento del objeto una vez terminada la oclusión.

Dentro de las librerías *cvBlob* las funciones y atributos utilizados para implementar este modelo de tracking han sido las siguientes:

- **struct Track**: Se trata de una estructura que contiene la información esencial del seguimiento de un solo objeto. Los atributos más importantes de esta estructura son:
  - **id**: Identificación única del track sobre el resto. Pertenece a la clase *cvIDTrack*.
  - **minx, miny, maxx, maxy**: Coordenadas que definen el rectángulo que engloba las componentes del track en cuestión.
  - **centroid**: Coordenadas del centro del rectángulo que engloba al objeto que se sigue.
  - **lifetime**: Número de frames durante los cuales el objeto ha sido seguido.
  - **inactive**: Número de frames que indica cuánto tiempo ha estado perdido el objeto.
  - **active:** Número de frames que indica cuánto tiempo ha estado el objeto activo desde el último período inactivo.

- **map Tracks:** Mapa que contiene parejas de identificaciones de tracks y los propios tracks. Cada entrada del mapa es una pareja compuesta por una estructura de track y su identificación única de la clase *cvIDTrack*. Este mapa se actualiza en cada frame con el seguimiento de las posiciones de cada objeto mediante la función *cvUpdateTracks*.

- **cvUpdateTracks** (blobs, tracks, thDistance, thInactive, thActive): Función principal e importante del algoritmo que actualiza el mapa de tracks (parámetro *tracks*) basándose en las componentes detectadas (parámetro *blobs*). El parámetro *thDistance* configura la distancia máxima para determinar cuándo un track y una componente coinciden. El parámetro *thInactive* configura el número máximo de frames que un objeto puede estar sin ser seguido antes de ser borrado. Cuando un track se vuelve inactivo pero ha estado activo menos frames que los que marca el parámetro *thActive*, el track es eliminado.

- **cvRenderTracks** (tracks, imgSource, imgDest, mode, font): Función que dibuja la información sobre los tracks almacenados en la variable *tracks*, en la imagen *imgDest*, dependiendo del modo almacenado en la variable *mode*. Según el

modo se puede dibujar el rectángulo que engloba al objeto que se está siguiendo o la identificación del track en forma de número según la fuente configurada en la variable *font*, siendo posibles ambas opciones. Además, esta función dibuja de color distinto aquellos cuadrados que se corresponden a tracks que se encuentran inactivos o que se han perdido durante un tiempo estipulado en la variable *thInactive*.

Un ejemplo del uso de estas funciones se puede encontrar en la figura 47, donde mediante la función *cvRenderTracks* se han dibujado los rectángulos de los objetos que se están siguiendo de acuerdo a la coincidencia de los tracks con las componentes conexas etiquetadas.

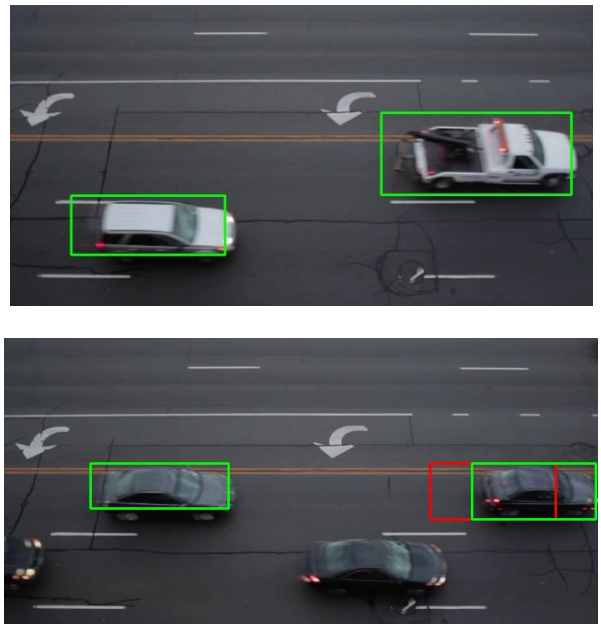


Figura 47.Demostración del uso de la función *cvRenderTracks*. Arriba, tracks realizados correctamente en verde. Abajo, track perdido o inactivo en rojo. Fuente [39].

3.5.5 Control del tránsito vehículo y Realidad Aumentada.

Tras realizar el seguimiento de todos los elementos vehiculares de la escena, se puede comenzar con el control del tránsito de vehículos en la vía. Del módulo de tracking se ha obtenido las coordenadas de los vehículos y, en concreto, la de su centro (parámetro *centroid* de la estructura de tracks). Esta coordenada del centro será muy importante de cara al control de vehículos, pues representa el centro de masas de la detección, concepto muy cercano y similar al centro de masas del propio vehículo.

Por control del tránsito vehicular se entiende la observación de los vehículos que circulan por vía en busca de un comportamiento anómalo, siendo esto cualquier

comportamiento que se salga de la tónica habitual dentro un entorno propio establecido y controlado (autovía, carretera urbana, parking, etc…). Para realizar el control del tránsito vehicular se ha tenido en cuenta algunas situaciones anómalas como puede ser la entrada o salida de un vehículo de una zona determinada o el control de la velocidad de dichos vehículos cuando está por encima del límite establecido. Por ello, se ha desarrollado en la aplicación el control de las siguientes situaciones: salida y entrada del vehículo de una zona previamente delimitada, control de velocidad de los vehículos, control del estacionamiento de un vehículo en una zona delimitada y control de la entrada y salida de vehículos en una zona con conteo de los mismos.

Es necesario comentar previamente ciertos aspectos a la hora de desarrollar el control de los vehículos en la aplicación. En primer lugar, hay que señalar el hecho de que una imagen es el resultado de mapear elementos de tres dimensiones sobre una superficie de dos, perdiéndose así la coordenada de profundidad en ese paso. En segundo lugar, hay que señalar también la manera en la que la cámara enfoca al objetivo, generando así una perspectiva. Si la cámara muestra un plano o ángulo picado (figura 48) los objetos en la imagen que se encuentran más lejos del objetivo se muestran más pequeños que los cercanos, que se mostrarán más grandes.

Figura 48. Ejemplo de plano contrapicado en una carretera. Los coches del fondo de la escena se muestran más pequeños que los del frente. Esta deformación se debe a la perspectiva. Fuente propia.

En la figura 49 se supone un elemento A en movimiento (como puede ser un vehículo o cualquier objeto) situado al fondo de la escena y que se mueve tanto en horizontal como en vertical. Se supone además otro elemento B de iguales dimensiones que el elemento A, también en movimiento y que se sitúa en primer plano de la escena, más cerca del objetivo y que también se mueve en horizontal y en vertical. Aunque ambos elementos se muevan la misma cantidad de píxeles en la misma dirección en sentido contrario, en la realidad los dos elementos no se han movido la misma distancia en metros. Esto es debido al mapeo anteriormente comentado y al tipo de perspectiva con el que se enfoca a los vehículos.

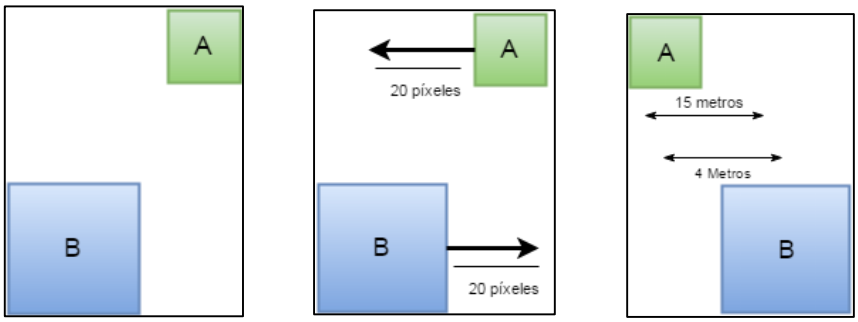


Figura 49. A la izquierda, dos elementos (A y B) de las mismas dimensiones situados a distinta distancias del objetivo. En el centro, los objetos se mueven una misma distancia en la imagen (20 píxeles). A la derecha, los elementos se han movido la misma cantidad de píxeles pero no de metros en la realidad. El elemento A se mueve una distancia mayor que el elemento B al estar mas lejos debido a la perspectiva.

Es importante valorar estos aspectos sobre todo si lo que se pretende es estimar la velocidad de los vehículos en la vía. Para realizar este tipo de acciones no servirán imágenes tomadas con una cámara en plano picado porque se realiza una falsa estimación de la distancia recorrida por los elementos del vehículo, y por tanto, una falsa estimación de la velocidad. Para estos casos, es más correcto utilizar planos horizontales o supinos. En estos planos hay relación igual entre la distancia recorrida por un objeto en la imagen y en la realidad, y todos los objetos de la escena se muestran a la misma distancia del objetivo, como se muestra en la figura 50.

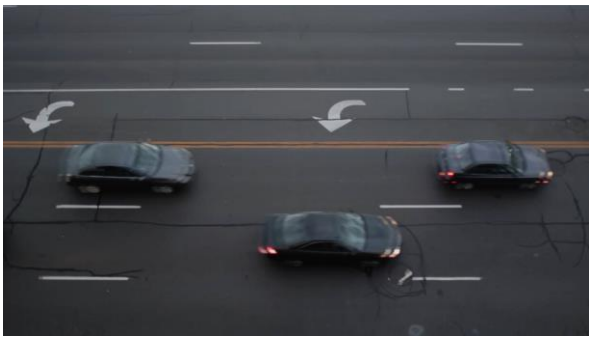


Figura 50. Plano horizontal o supino. No existe deformación del tamaño de los objetos en la escena. Fuente [39].

La implementación de cada control anteriormente explicado se detalla en el pseudocódigo del apartado 3.6.1, a continuación se muestran aspectos del funcionamiento no relativos a la codificación:

1. **Control del vehículo en una zona previamente delimitada.** El uso de esta opción es aplicable cuando se desea controlar cierta zona de interés en una vía. Esto puede ser, por ejemplo, para avisar cuando un vehículo entra en un desvío o un tramo de la carretera que por diversas razones está prohibido el acceso; o bien, puede servir para conocer la llegada de un vehículo que se esperaba a una zona concreta de la vía. Antes de comenzar con la detección de vehículos, el usuario

delimita con el ratón una zona de puntos. Estos puntos al unirse formarán un polígono cuya área será la estudiada y que se mostrará en un color transparente. Después se realiza la detección y el seguimiento de todos los vehículos que entran en la escena mediante la función *cvUpdateTracks*, pero no se dibujan sus detecciones en rectángulos. Cuando las coordenadas del centro del track de un vehículo (parámetro *centroid*) entran en el área delimitada, se dibuja su detección en un rectángulo con la función *cvRenderTracks* y además, el área delimitada cambia de color rojo a verde junto a un indicador de texto ("DETECTADO" o "NO DETECTADO") para avisar al usuario de que uno o varios vehículos han entrado en la zona. En la figura 51 se puede observar este funcionamiento.

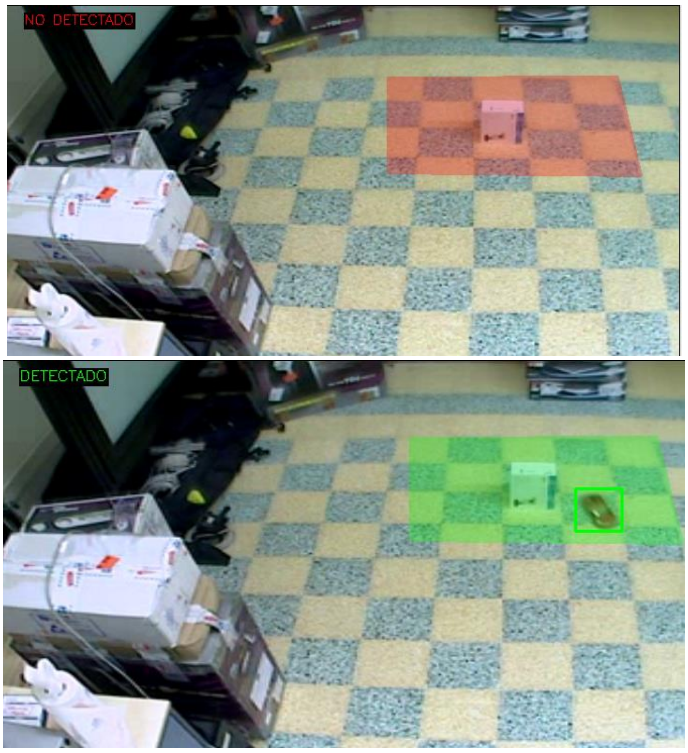


Figura 51. Arriba, zona vacía sin ningún vehículo en su interior. Abajo, la zona delimitada con un vehículo en su interior. Fuente [43].

2. **Control de la velocidad de los vehículos.** El control de la velocidad en espacios de tráfico es algo fundamental para garantizar la seguridad en la vía. Mediante esta opción se pretende que el usuario monitorice la velocidad de los vehículos de la escena. En primer lugar, se actualiza el estado del seguimiento de todos los vehículos previamente detectados en el frame actual con la función *cvUpdateTracks*. Si previamente no se tienen registros de los tracks del frame anterior se crea una estructura con los datos recién adquiridos y se pasa al siguiente frame. Una vez que se tengan datos de los tracks del frame actual y del

anterior, se puede comenzar con el cálculo de la velocidad. Para cada track de cada vehículo se dibuja su detección con *cvRenderTracks* y se obtiene la distancia avanzada por su centro (parámetro *centroid*) entre los dos frames. Como se sabe, la definición de velocidad de un objeto es el cociente entre la distancia recorrida entre dos puntos y el tiempo que tarda en recorrerla. La velocidad del vehículo se puede hallar dividiendo dicha distancia (en píxeles) entre el tiempo entre frames tal y como indica la ecuación 33, donde el *V* es la velocidad del vehículo, *C* las coordenadas del centro de la detección y *t* el frame actual:

$$V = |C_t - C_{t-1}| * fps \qquad (33)$$

Un aspecto importante a destacar es el resultado dimensional de la ecuación 33, dado en *píxeles/frame*, ya que el cálculo se realiza frame por frame. Una muestra del funcionamiento se puede observar en la figura 52.

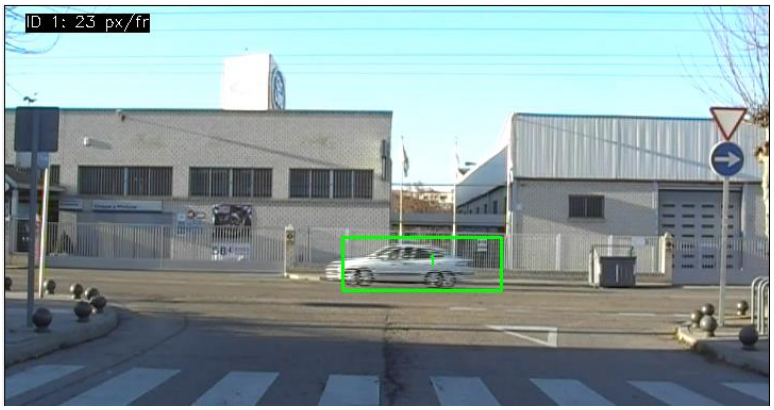


Figura 52. Funcionamiento del control de la velocidad de los vehículos. Fuente propia.

A primera vista, y sin ningún tipo de dato adicional, no es posible obtener las distancias en metros recorridas por los vehículos en la escena, únicamente es posible mostrar la velocidad en la cantidad de píxeles recorridos por el vehículo en un segundo. No obstante se puede encontrar una correspondencia aproximada entre las medidas *pixeles* y *metros*. Para ello es necesario conocer el ángulo horizontal de visión de la cámara y la distancia a la vía que se está enfocando. Mediante cálculos trigonométricos como se observan en las ecuaciones 34 y 35, y en la figura 53, se obtiene las dimensiones de la vía que se está enfocando.

$$\mathrm{Tan}\left(\frac{A_h}{2}\right) = \frac{l/2}{d} = \frac{l}{2d} \qquad (34)$$

$$d = \frac{l}{2} \arctan\left(\frac{A_h}{2}\right) \qquad (35)$$

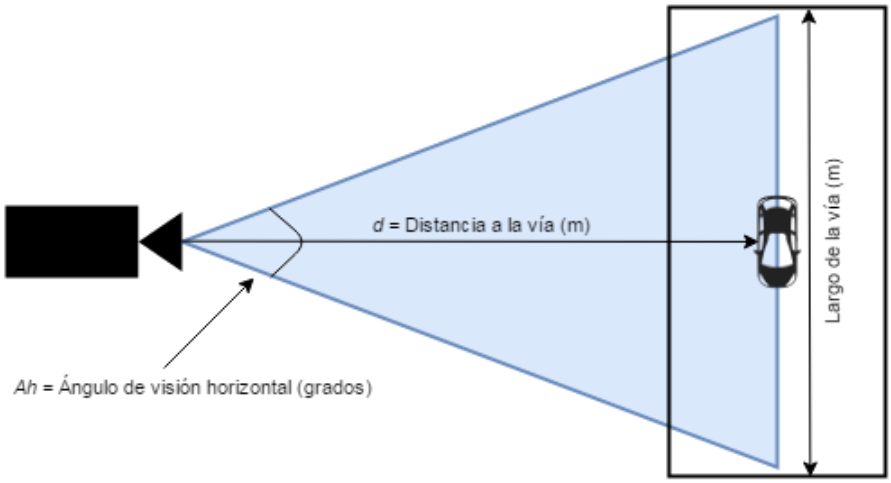


Figura 53: Vista aérea de la toma de imágenes en una vía.

Donde A_h es el ángulo de visión horizontal en grados, l el largo de la vía que engloba la cámara por donde circulan los vehículos y d la distancia entre la cámara y la vía en cuestión. Una vez obtenida la distancia a la vía se puede encontrar una correspondencia entre metros y píxeles, ya que **la distancia $\boldsymbol{l}$ se corresponde con la anchura del frame**. Estos cálculos sólo son válidos si la posición de la cámara se encuentra en el mismo plano horizontal que los vehículos y la vía. Además, es imprescindible realizar la toma con planos supinos u horizontales, como se ha indicado anteriormente, ya que de otra manera la perspectiva no permitiría hallar una relación constante píxeles/metro. Es importante destacar que este método no deja de ser una aproximación o planteamiento y está sujeto a condiciones ideales como que la cámara esté a la misma altura que los vehículos que circulan por la vía y la línea de enfoque sea perpendicular a dichos vehículos. Por lo tanto, no se trata de una manera fehaciente de encontrar la correspondencia entre píxeles y metros en la imagen.

3. **Control del estacionamiento de un vehículo en una zona delimitada.** El uso del algoritmo de Mezclas de Gaussianas que realiza una adaptación constante del background permite averiguar cuando un vehículo ha desaparecido del foreground, y por tanto, cuando ha dejado de moverse. Se puede entender el fin de movimiento como dos sucesos posibles: el vehículo se ha parado o el vehículo ha desaparecido de la escena. La diferencia entre ambos sucesos es que durante la parada del vehículo el seguimiento del vehículo desaparecerá de forma progresiva y estará durante un tiempo inactivo. Detectando esa desaparición progresiva de la detección dentro de una zona concreta, se puede obtener el

control de estacionamiento de dicho vehículo en dicha zona. Para ello, será necesario en primer lugar que el usuario escoja una serie de puntos que delimiten la zona de estacionamiento. Después se realizará el seguimiento de todos los vehículos de la escena con la función *cvUpdateTracks*. Cuando un vehículo entre dentro de la zona delimitada (parámetro *centroid* de la detección dentro del área) un indicador de texto pasará de "VACIO" a "APARCANDO" y la zona de aparcamiento cambiará de rojo a azul para avisar al usuario de que hay un vehículo que se dispone a realizar el aparcamiento, además se dibujará la detección del vehículos con *cvRenderTracks*. Una vez que el vehículo haya parado y el algoritmo haya asimilado que forma parte del background su detección desaparecerá progresivamente y su seguimiento pasará al estado inactivo. Cuando esto ocurra el indicador de texto pasará de "APARCANDO" a "APARCADO" y la zona de aparcamiento cambiará del color azul a verde. De esta forma se avisará al usuario de que el vehículo finalmente ha estacionado en la zona y su área cambiará de color. Este proceso se puede observar en la figura 54.

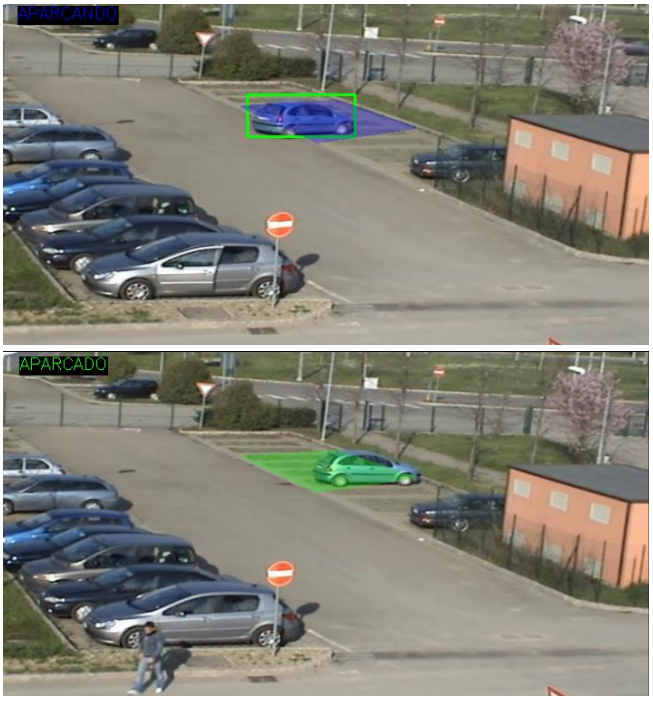


Figura 54. De arriba a abajo, fase control del estacionamiento. Fuente [43].

Una vez que el seguimiento haya pasado al estado inactivo, el proceso esperará un tiempo determinado en los cuales se aguardará a que los integrantes del vehículo salgan y abandonen la zona delimitada. La espera se puede realizar con las funciones de la librería *time* implícita en las librerías de C++. Esto se realiza dado que el movimiento de las personas saliendo del coche provoca la reactivación del algoritmo provocando un error en la detección, ya que el vehículo no ha abandonado la zona, sólo sus integrantes. Una vez terminado este tiempo, la aplicación pasará a un estado de reposo donde buscará en cada frame una reactivación del movimiento en la zona de aparcamiento mediante la función *cvUpdateTracks*, con la intención de detectar cuando el vehículo abandona la plaza de aparcamiento delimitada. Si esto ocurre, cuando el vehículo se empiece a mover, el indicador de texto pasará de "APARCADO" a "APARCANDO" y el color de la zona de verde a azul. Además se dibujará la detección del vehículo de nuevo con la función *cvRenderTracks*. Finalmente cuando el vehículo abandone de forma total la plaza de aparcamiento el indicador de texto tomará el valor de "VACIO" y la zona delimitada tomará el valor de rojo. Una muestra de este funcionamiento se puede observar en la figura 55.

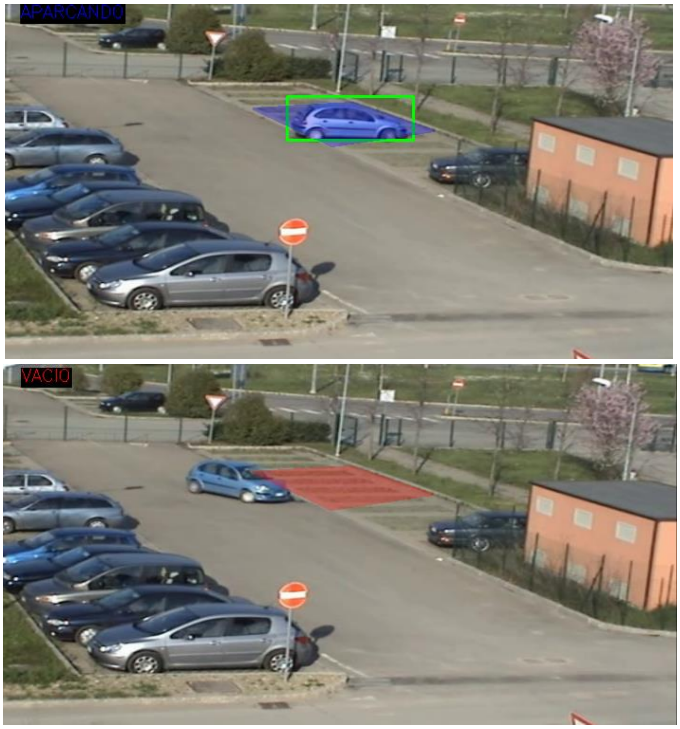


Figura 55. De arriba a abajo, fases del control de estacionamiento cuando un vehículo abandona la plaza de estacionamiento. Fuente [43].

4. **Control de la entrada y salida de una zona con conteo de vehículos:** El control de entrada y salida y de conteo de vehículo de una zona controlada resulta casi indispensable a la hora de realizar tareas de vigilancia y seguridad en zonas como parkings o aparcamientos privados. Para realizar este control el usuario establece tres zonas tal y como se muestra en la figura 56.

Figura 56. Las tres zonas para el control de entrada/salida y de conteo de vehículos Fuente [43].

La zona roja corresponde al lugar final de estacionamiento de los vehículos que entran y el lugar de partida de los vehículos que salen. Las zonas verde y azul responden a la siguiente lógica: cuando un vehículo atraviesa primero la zona verde (zona 1) y después la zona azul (zona 2) se notifica como una entrada y el contador de vehículos aumenta en uno; sin embargo, cuando un vehículo

atraviesa primero la zona azul (zona 2) y después la zona verde (zona 1) se notifica como una salida y el contador disminuye en 1. Por ello se realiza la detección y el seguimiento de todos los vehículos en la zona (mediante la función *cvUpdateTracks*) prestando especial atención a aquellos *tracks* cuyo parámetro *centroid* quede dentro de alguna de estas dos zonas. En el momento en el que el parámetro *centroid* del *track* de un vehículo quede dentro de alguna de las zonas su identificación (parámetro *ID*) se almacena en un vector propio del área (ya sea la 1 o la 2) y se dibuja su detección con *cvRenderUpdates*. Con estos vectores, se lleva el control de los vehículos que atraviesan cada zona. En cada frame se buscará vehículos en ambas zonas y se revisará si el vehículo está en el vector de la otra área correspondiente. Es decir, si un vehículo entra en la zona 1 se buscará dicho vehículo en el vector de la zona 2 para comprobar si previamente ha pasado por esa zona 2. Y si un vehículo entra en la zona 2 se buscará dicho vehículo en el vector de la zona 1 para comprobar si previamente ha pasado por esa zona 1. De esta manera es posible establecer si un vehículo está saliendo o está entrando de la zona delimitada, y de esa manera, aumentar o disminuir el contador.

3.5.6 Presentación de resultados: Realidad Aumentada.

Una vez realizado el control del tráfico vehicular es necesario mostrar al usuario toda la información recopilada sobre el mismo. Esta información se mostrará de forma visual con indicadores, zonas coloreadas y elementos superpuestos a la imagen para ofrecer el añadido de información al usuario sin que en ningún momento deje de visualizar la escena. En las figuras anteriores del apartado anterior se pueden ver ejemplo de dichas representaciones de la realidad aumentada.

Para generar realidad aumentada OpenCV dispone de multitud de funciones y clases. No obstante, en primer lugar, se explicarán aquellas más recurrentes y utilizadas en la implementación:

- ***imshow***. Se trata de la función que permite representar por la pantalla una imagen en cualquier espacio de color tratada anteriormente en una ventana emergente.
- ***imresize***. Reescala la imagen a las proporciones especificadas, permitiendo varios modos de reescalado.
- ***rectangle***. Inserta un rectángulo de dimensiones específicas en las coordenadas dadas por el usuario. El rectángulo es totalmente configurable en tamaño, color, relleno y anchura del trazo.

- ***circle****.* Inserta un círculo de dimensiones específicas en las coordenadas dadas por el usuario. El círculo es configurable en su tamaño, color, si está relleno o no y su anchura del trazo.
- ***line***. Inserta una línea entre dos coordenadas de punto dadas por el usuario. La línea es configurable en su color y su anchura del trazo.
- ***drawContours****.* Dibuja los contornos cerrados que se pasan por parámetros. La función permite además rellenar la figura contorneada y parametrizar su color y su trazo. Muy útil para dibujar áreas en una imagen.
- ***addWeighted***. Realiza una mezcla entre dos imágenes pasadas por parámetro teniendo en cuenta un parámetro *alpha* que regula la cantidad de mezcla de una imagen con la otra. Muy útil para añadir un canal *alpha* a las imágenes y volver transparentes ciertas zonas coloreadas como se muestra en las figuras X o X.
- ***putText****.* Inserta un texto de color y fuentes especificada en la imagen en la coordenada dada por el usuario. Útil para mostrar información sobre las variables por pantalla
- ***waitKey****.* Aunque no realiza ninguna función visual en la imagen, este método es importante ya que tiene que ser llamado en cada frame para refresca la imagen que se va mostrando por pantalla.

A continuación se explican cómo se han utilizado estas funciones anteriores en cada opción de control elegida por el usuario:

- **Control de todos los vehículos de la escena.** En este caso, tras recopilar toda la información del tracking de los vehículos lo único que queda es mostrar dicha información en pantalla. Por ello cada detección de cada vehículo viene dada por un rectángulo que se mostrará en la imagen mediante la función *rectangle*.

- **Control de vehículos en una zona delimitada.** Para dibujar la zona delimitada se han utilizado la función *drawContours* con los contornos elegidos por el usuario con el ratón. La transparencia de los mismos se ha conseguido con la función *addWeighted* que también se encarga de cambiar su color cuando un vehículo penetra la zona. El texto en la parte superior izquierda de la pantalla se ha conseguido con la función *putText* y las detecciones de las mismas con la función *rectangle*.

- **Control de velocidad de los vehículos.** Las detecciones de los vehículos se realizan de nuevo con la función *rectangle*. La velocidad mostrada de cada vehículo y la identificación propia junto a su detección se muestran con la función *putText*.

- **Control de estacionamiento y control de entrada y salida con conteo.** En ambos casos, las zonas de control se dibujan, se colorean y se transparentan gracias a las funciones *drawContours* y *addWeighted*. La información en texto sobre el estado del aparcamiento se muestra mediante la función *putText*. Igualmente, la información sobre el estado del contador se muestra también con la función *putText*. Las detecciones de los vehículos en ambos casos se muestran con la función *rectangle*.

3.5.7 Ventajas y desventajas de la solución

Se han realizado varias pruebas de implementación sobre esta solución antes de optar por el desarrollo final. El desarrollo de dichas pruebas ha permitido obtener una perspectiva más general a la hora de realizar una implementación definitiva. Durante el desarrollo se podido observar las distintas ventajas e inconvenientes que tiene esta solución, además de realizar una comparación con la solución anterior de HOG + SVM.

En primer lugar, se puede señalar como ventajas los siguientes aspectos del método:

- La sustracción de background como método de extracción y detección de características supone una elección acertada para el objetivo que este proyecto persigue. Este tipo de método son utilizados para extraer y estimar el movimiento en imágenes estáticas, donde la cámara no se mueve. Dado que el objetivo en un principio es la detección de vehículos, la sustracción de background permite obtener únicamente los elementos que se mueven en la vía que serán, en este caso, los coches y demás vehículos en movimiento. Por tanto la extracción se realiza de forma casi exclusiva a estos elementos, evitando por tanto la adquisición de falsos positivos.

- Dentro de los métodos de sustracción de background, la técnica de Mezclas de Gaussianas con detección de sombras permite modelar un background variable a cambios en el escenario. Algo ideal para este tipo de tareas de vigilancia. Esto permite que el background se actualice de forma constante consiguiendo que no sea demasiado sensible a cambios de luminancia o a elementos que quedan estáticos después de estar en movimiento; todo ello sin realizar una gran carga computacional.

- Dado que la extracción y la detección de los objetos se realiza de forma notablemente eficiente, facilita el trabajo para las posteriores etapas como el etiquetado y el tracking.

- La extracción y el tratamiento del foreground es una de las técnicas más tratadas y estudiadas en visión artificial. Esto hace que sea posible disponer de multitud de librerías y métodos ya implementados para su uso directo. Este es el caso de la librería CvBlob, donde se puede encontrar los algoritmos de etiquetado de componentes conexas del foreground y de tracking utilizados en la implementación final. Esto ha permitido un desarrollo inicial mucho más llevadero y organizado.

- El método de tracking propuesto (modelos de apariencia) permite estimar de forma más adecuada el seguimiento de cada vehículo. Las siluetas de los vehículos prácticamente permanecen invariantes durante gran parte del tiempo, lo que facilita el seguimiento de las detecciones.

- Dado que se han utilizado métodos y librerías estables y bien definidas, la aplicación funciona de manera óptima con tiempos de ejecución reducidos. Como normal general, la mayoría de cámaras para vigilancia de tráfico no superan los 25 o 30 frames por segundo, por tanto, si el sistema que ejecuta la aplicación cumple ciertos requisitos mínimos de hardware la aplicación puede funcionar en tiempo real.

Respecto a las desventajas, se pueden destacar las siguientes:

- Pese a que Mezclas de Gaussianas es un método que actualiza el background según una tasa de aprendizaje para evitar que la sustracción no sea demasiado sensible, el algoritmo lo sigue siendo a cambios muy bruscos de iluminación, pequeños movimientos de cámara o por ejemplo, en ambientes nocturnos. Por tanto, la situación de la cámara en una vía de tráfico puede ocasionar pequeños movimientos debido al viento, o bruscos cambios de luminancia por los reflejos de las luces en las superficies de los vehículos.

- Un cambio brusco y grande en el background (por ejemplo, el movimiento repentino de la cámara) puede ocasionar una mala estimación del mismo. Dado que el método de Mezclas de Gaussianas estima el background teniendo en cuenta un factor de memoria de backgrounds anteriores, puede darse el caso de que el sistema entre en una situación inestable con una bajada grande de frames por segundo, aunque consiga más tarde recuperarse.

- En términos generales, es difícil estimar con certeza y de forma simple la velocidad de un objeto en un vídeo debido sobre todo a la perspectiva generada

al pasar de un entorno de tres dimensiones a uno sólo de dos. Por lo tanto, se añade cierta dificultad a la hora de toma de imágenes desde la cámara, que debe ser desde un ángulo y perspectiva correctos y con ciertas especificaciones como se ha comentado anteriormente en este mismo capítulo.

- El algoritmo de tracking propuesto en esta solución permite el manejo de ciertos niveles de oclusión y de mezcla de dos o más objetos simultáneos durante el tracking. No obstante, para períodos largos de oclusión junto a una alta tasa de actualización puede ocasionar problemas en el tracking de los objetos haciendo que los seguimientos se vuelvan inactivos demasiado pronto.

A la vista de las ventajas y desventajas anteriores se comparó esta solución con la descrita anteriormente de HOG, SVM y Flujo Óptico. Finalmente **se decide optar por el desarrollo y la implementación final de la aplicación utilizando esta solución de sustracción de background, análisis de componentes conexas y tracking mediante modelos de apariencia**. Esto es debido principalmente a que con la solución primera no se consiguieron resultados suficientemente válidos en las pruebas de detección como para optar por ella en un desarrollo final; además, y como se detalla en el capítulo siguiente de pruebas, la segunda solución demuestra su validez cómo método para este tipo de aplicaciones.

3.6 Desarrollo de la aplicación

3.6.1 Implementación de la lógica y de la interfaz gráfica

En fases iniciales se optó primeramente por el desarrollo de la aplicación en consola sin ningún tipo de interfaz gráfica más allá de ventanas que muestran las detecciones y procesamiento final. De esta manera, el desarrollo fue fácilmente adaptable cuando se daba el caso de que la lógica o el diseño de la aplicación sufrían modificaciones, a fin de mejorar su funcionamiento.

En esta primera parte se ideó una lógica sencilla general y de grandes rasgos. El objetivo fue el desarrollo secuencial de cada parte para después ir añadiendo sucesivamente las partes restantes. Cada parte, a su vez, se dividió en partes o tareas más pequeñas para simplificar el desarrollo.

Como objetivo principal, la aplicación primeramente debía realizar la detección y el seguimiento de los vehículos en una imagen de una vía. Este objetivo se dividió a su vez en tres objetivos más:

1) Extracción del foreground mediante mezclas gaussianas.
2) Análisis y etiquetado de las componentes y regiones del foreground.
3) Seguimiento de dichas componentes y regiones a través de frame sucesivos.

Una vez codificada la extracción del foreground, se pasó a la codificación del análisis del etiquetado de componentes, y así, hasta llegar al tracking final. La deconstrucción del objetivo principal en pequeñas tareas, como las tres anteriormente comentadas, dota de cierta independencia al desarrollo pudiendo modificar fácilmente si es necesario cada parte del objetivo separándola de las demás.

En términos de pseudocódigo este objetivo sería de la siguiente forma:

```
INICIO
	INICIAR FUENTE DE IMÁGENES (VIDEO/CAMARA)
	DECLARAR E INICIALIZAR VARIABLES PARA IMÁGENES
	CREAR Y CONFIGURAR SUSTRACTOR DE BACKGROUND
	DECLARAR E INICIALIZAR VARIABLES PARA EL ANÁLISIS DE COMPONENTES
	DECLARAR E INICIALIZAR VARIABLES PARA EL TRACKING
	MIENTRAS
		CAPTURAR FRAME
		SI FRAME VACIO O FIN DE FUENTE ENTONCES
			BREAK MIENTRAS
		FIN SI
		// 1) EXTRACCIÓN DE FOREGROUND
		REDIMENSIONAR FRAME
		DESENFOCAR FRAME
		EXTRAER FOREGROUND CON SUSTRACTOR DE BACKGROUND
		DILATAR MORFOLÓGICAMENTE EL FOREGROUND
		// 2) ANÁLISIS Y ETIQUETADO DE COMPONENTES
		EXTRACCION Y ETIQUETADO DE COMPONENTES DEL FOREGROUND
		FILTRAR COMPONENTES POR AREA
		// 3) SEGUIMIENTO DE COMPONENTES
		SI SE INICIÓ ESTRUCTURAS DE TRACKING
			ACTUALIZAR ESTRUCTURAS DE TRACKING
		SINO
			INICIAR ESTRUCTURAS DE TRACKING
```

```
            FIN SI

            DIBUJAR RECTANGULOS DE TRACKING

        MOSTRAR IMAGEN

        FIN MIENTRAS

FIN
```

Una vez cumplido este objetivo, se sigue con la codificación de las opciones del control del tráfico anteriormente explicadas. Las opciones a codificar son las siguientes:

1) Detección de vehículos general.
2) Control de vehículos en una zona delimitada.
3) Control de estacionamiento de vehículos en una zona delimitada.
4) Control de velocidad de vehículos.
5) Control de entrada y salida con conteo de vehículos de una zona delimitada.

Para codificar estas opciones únicamente es necesario insertarlas en el código anterior del objetivo principal en el bucle principal, quedando de esta manera completa de manera total la lógica de la aplicación. El pseudocódigo de la aplicación se puede ver en el anexo adjunto al final de este documento. De esta manera quedaría a falta únicamente de dotar a la aplicación de una interfaz gráfica que permita al usuario un manejo sencillo y accesible.

La interfaz gráfica de usuario debía cumplir con ciertos requisitos de accesibilidad y sencillez. La interfaz debe ser intuitiva para el usuario, y dada la naturaleza del proyecto debe mostrar las imágenes de manera clara. La disposición y el aspecto de los elementos destinados a configurar los procesos han de ser simple y austeros, sin estorbarse entre ellos.

Como ya se ha comentado anteriormente, para el diseño y la implementación de esta interfaz se ha optado por el software Qt Creator, que otorga cierta facilidad y sencillez a la hora de implementar pequeñas interfaces gráficas como la de esta aplicación. En la figura 57 se muestra el aspecto de la ventana principal. Se ha optado por un diseño de una única ventana, donde se encuentra conjuntamente un apartado para mostrar las imágenes procesadas y justo debajo las opciones para cargar la fuente (ya sea de una cámara externa o un archivo de vídeo) y las opciones de control de tránsito vehicular.

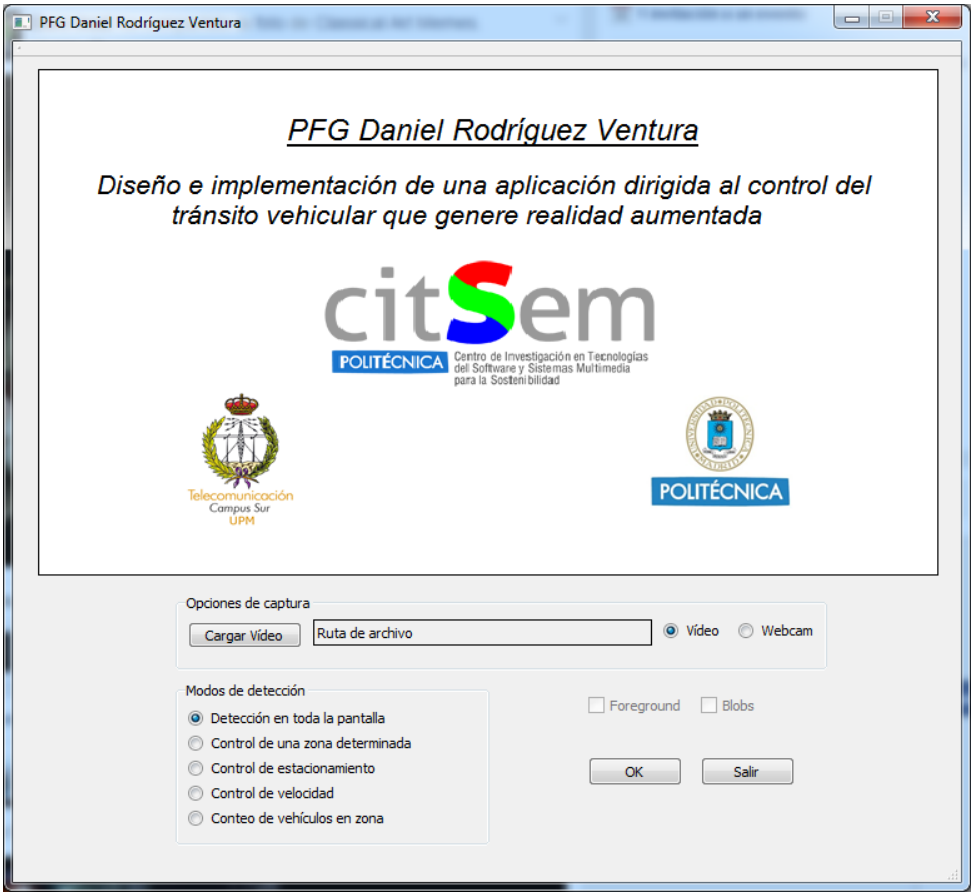


Figura 57. Aspecto principal de la interfaz gráfica.

Dada la figura 57 los elementos de la interfaz con su función asociada son los siguientes:

1. **Visor**: es el objeto principal donde se muestra las imágenes procesadas de las detecciones y el control de tráfico vehicular según la opción marcada por el usuario. Se modela mediante las clases nativas *QLabel* y *QPixmap*. Sus dimensiones son 720 por 405 píxeles, ya que se trata de una resolución que permite mostrar las imágenes a un tamaño adecuado sin comprometer demasiado al tiempo de procesamiento de la imagen, que es proporcional a la cantidad de píxeles de la misma. En la parte superior del visor se mostrará la información en texto si la opción escogida así lo requiero, como es el caso del contador de vehículos en una zona. Además, en la parte inferior izquierda se mostrará la tasa de frames por segundos (*fps*) en todo momento.

2. **Botones "Vídeo" y "Cámara":** permite conmutar entre los dos posibles tipos de fuente de imágenes. Si el usuario pulsa el botón de "Vídeo" se habilitarán el botón de "Cargar Video" y el campo de texto "Ruta de archivo". Si el usuario pulsa el botón de "Cámara", el botón de "Cargar Video" y el campo "Ruta de archivo" se bloquearán y en el visor aparecerá las imágenes de la cámara si está

disponible. Si no hay ninguna cámara disponible se mostrará un aviso. De forma predeterminada, el botón "Vídeo" siempre está seleccionado.

3. **Botón "Cargar Vídeo":** al pulsarlo se abre una ventana de distribución de ficheros en carpetas para que el usuario elija uno. Los formatos disponibles son .avi, .mpg, .mpeg y .mp4. Una vez elegido el archivo se volverá a la ventana principal. En el campo de texto "Ruta de archivo" aparecerá la ubicación en disco del fichero elegido y este comenzará a reproducirse en el visor a modo de previsualización.

4. **Opciones de control del tránsito vehicular:** se trata de un conjunto de cinco botones seleccionables que permite al usuario elegir entre las opciones de modos de control anteriormente explicadas. Dependiendo de la opción escogida se ejecutará una parte correspondiente del proceso principal.

5. **Botones "Ok" y "Salir":** al pulsar el botón "Ok" la aplicación ejecuta el proceso principal en base a la fuente de vídeo elegida y a la opción escogida por el usuario y muestra por el visor las imágenes procesadas. Si se pulsa de nuevo, se pausará el proceso y se volverá al estado inicial anterior. El botón "Salir" permite al usuario cerrar la aplicación en cualquier momento, ya sea en ejecución del procesamiento o no. En el caso de que ninguna fuente de imágenes haya sido elegida previamente, al pulsar el botón la aplicación mostrará un mensaje de advertencia. En la figura 58 se muestra el aspecto aplicación tras presionar el botón "Ok".

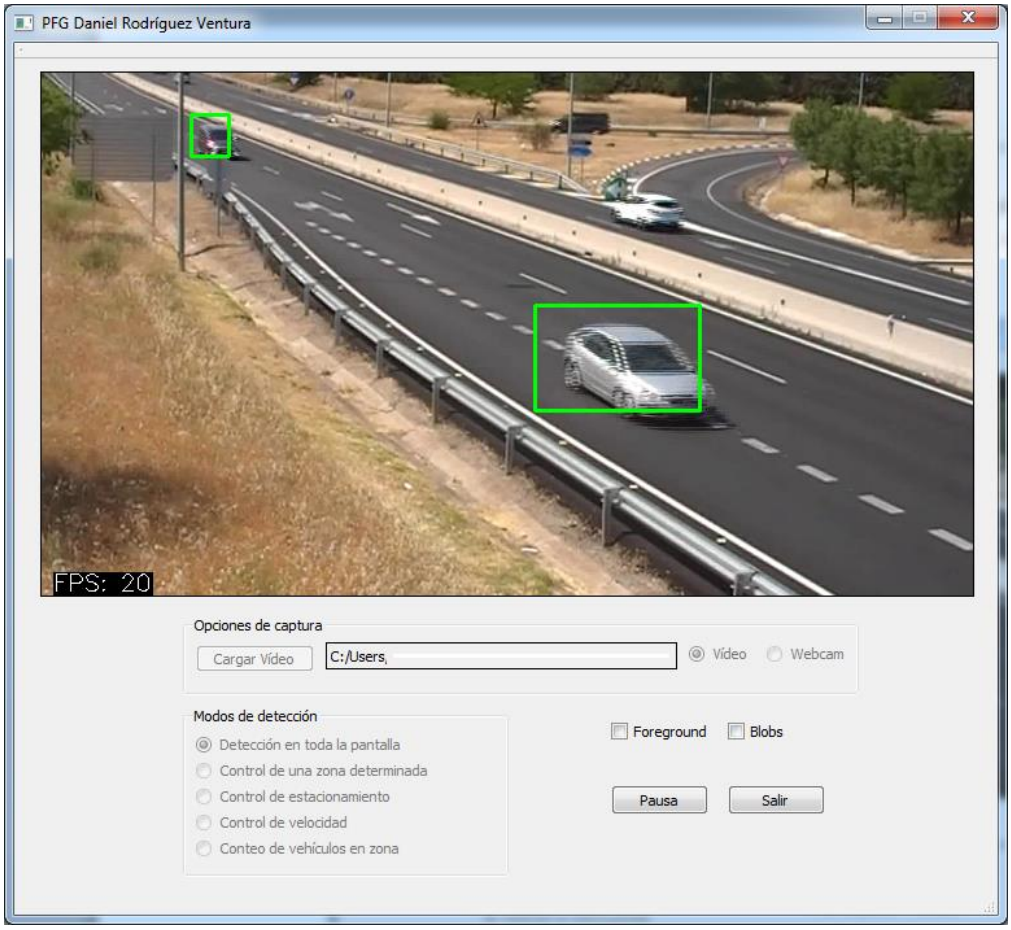


Figura 58. Aspecto de la aplicación al pulsar el botón "Ok".

6. **Casillas "Foreground" y "Blobs":** son casillas que se desbloquean cuando el usuario pulsa el botón "Ok" y empieza el procesamiento. Al pulsar la casilla "Foreground" se abre una ventana emergente donde se muestra la máscara de *foreground* (imagen binaria) obtenida de la fuente escogida. En el caso de pulsar la casilla "Blobs" aparece otra ventana emergente donde se muestran las componentes conexas encontradas en el *foreground* dibujadas en distintos colores y filtradas por área. Ambas ventanas pueden funcionar simultáneamente. De esta manera se ofrece el usuario un extra de información sobre el procesamiento de las imágenes, dando una visión general sobre ciertas partes del proceso principal que a priori son invisibles en la aplicación. En las figuras 59 y 60 se pueden ver un ejemplo del funcionamiento de cada casilla con sus respectivas ventanas.

Figura 59. Ventana "Foreground" que muestra en una imagen binaria el foreground que se está procesando. Fuente [39].

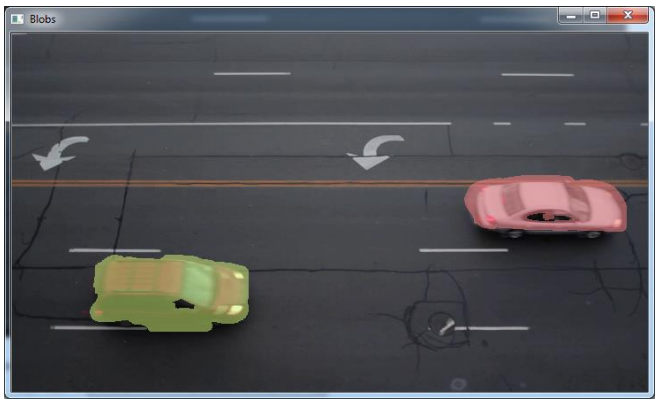


Figura 60. Ventana "Blobs" que muestra las componentes conexas analizadas en distintos colores según la máscara de foreground obtenida. Fuente [39].

Aparte de estos objetos de la ventana principal, la aplicación cuenta con otros elementos de configuración. Para los modos de control donde el usuario deba delimitar una zona en concreto en la escena (modos 2, 3 y 5), al pulsar el botón “Ok” aparecerán dos ventanas emergentes una de ellas con una imagen estática de la fuente seleccionada y otra con instrucciones. En la ventana de la imagen, el usuario deberá seguir las instrucciones para delimitar, mediante el uso del ratón, una o varias zonas de interés para el modo de control. En la figura 61 se muestran ambas ventanas en funcionamiento.

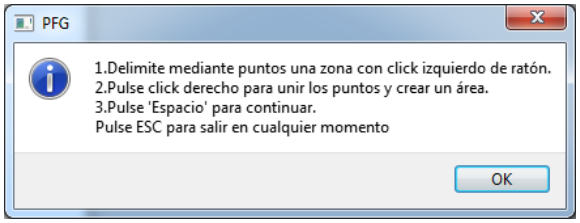


Figura 61. Ventanas para delimitar zonas de interés en la escena, junto a las instrucciones. Fuente [43].

3.6.2 Diagrama general de la aplicación

En la figura 62 se encuentra el diagrama general de la aplicación. En él se detalla el flujo de operaciones y la lógica que sigue el programa según las decisiones de usuario.

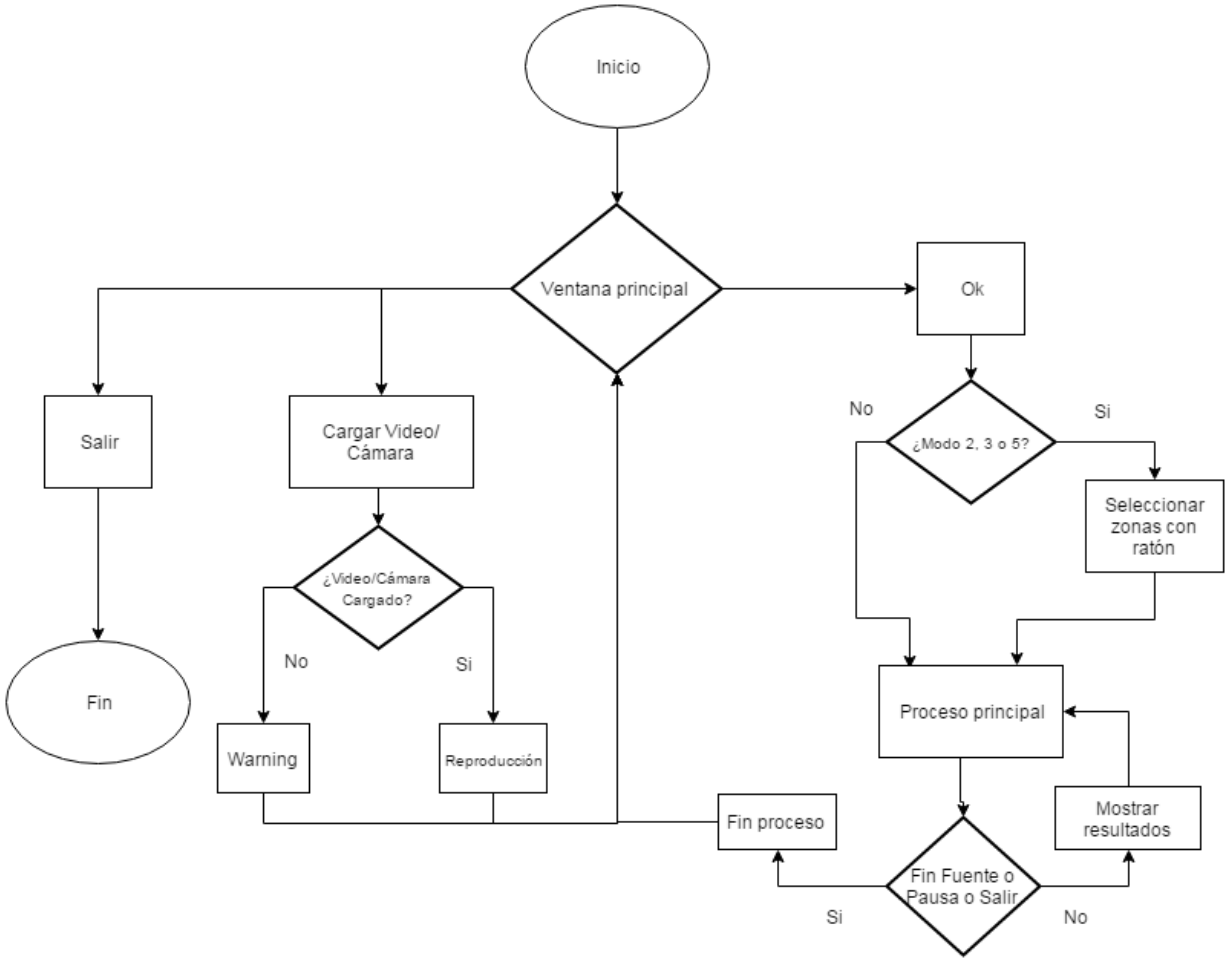


Figura 62. Esquema general de la aplicación

3.6.3 Estructura técnica de la aplicación

En este apartado se presenta los aspectos técnicos del diseño de la aplicación como es la distribución de sus ficheros, así como sus clases y funciones más importantes.

El despliegue de archivos utilizados durante el desarrollo de la aplicación en Qt Creator es el que se muestra en la siguiente figura 63.

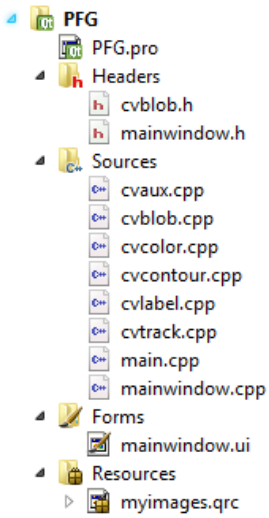


Figura 63. Distribución de ficheros en Qt Creator

Así pues, el proyecto "PFG" tendría los siguientes archivos:

- **PFG.pro:** Se trata de un archivo propio de configuración para proyectos de Qt Creator. En este archivo se encuentran especificados el uso de las librerías de OpenCV y CvBlob, así como los nombres de los formularios y los archivos fuente .cpp necesarios.

- **Carpeta "Headers":** Contiene los ficheros de cabecera con extensión .h. Estos archivos contienen declaraciones de clases, variables, estructuras y funciones propias. Dentro de esta carpeta se encuentra:
  - *cvblob.h*: contiene toda la información de definición de clases, estructuras, librerías, constantes y funciones de la librería CvBlob.
  - *mainwindow.h*: contiene la definición de la clase MainWindow que es la clase que modela la ventana principal de la aplicación. Aquí se ha definido las cabeceras de los métodos que posteriormente modelan el funcionamiento y la lógica de cada elemento de la interfaz gráfica. En la tabla 2 se muestran todas las funciones declaradas en este fichero con una descripción de su uso.

Función	Descripción
on_cargarButton_clicked	Asociada al botón de cargar vídeo desde fichero. Permite al usuario elegir un archivo de video para procesarlo mediante una ventana de selección. Si la carga se realiza correctamente, lanza el *timer* para previsualizar el vídeo en el visor.
on_webcamButton_clicked	Asociada al botón de elegir una cámara como fuente principal de imágenes. Si existe un dispostivo disponible se lanza un *timer* para previsualizar sus imágenes por el visor.
on_radioButton_clicked	Asociada al botón para elegir carga desde fichero. Interrumpe la carga de imágenes desde cámara, si se está realizando.
on_okButton_clicked	Asociada al botón "Ok". Realiza validaciones sobre las opciones elegidas por el usuario y lanza el *timer* del proceso principal.
timerLoopWebCamVisual	Proceso asociado a un *timer* que es lanzado por el botón "Cámara" y que muestra las imágenes de la cámara, si están disponibles.
timerLoopVideoVisual	Proceso asociado a un *timer* que es lanzado por el botón "Cargar Vídeo" y que muestra las imágenes del fichero, si están disponibles.
timerLoopPrincipal	Proceso asociado al *timer* principal que es lanzado por el botón "Ok". Realiza el procesado principal de la aplicación y muestra los resultados por el visor.
cargarImagenInicial	Carga en el visor una imagen inicial a modo de inicio o una pantalla de carga según su parámetro de entrada.
liberarRecursos	Limpia las variables del proceso principal una vez terminado.
on_checkBoxForeground_stateChanged	Asociada a la casilla "Foreground". Cuando la casilla cambia de estado se abre o se cierra la ventana que muestra la máscara de *foreground* obtenida.
on_checkBoxBlobs_stateChanged	Asociada a la casilla "Blobs". Cuando la casilla cambia de estado se abre o se cierra la ventana que muestra las componentes conexas obtenidas del proceso principal.
activarBloqueo	Asociada al botón "Ok". Bloquea y desbloquea los elementos de cargar video o cámara y los modos de detección durante el proceso principal.
setLabel	Inserta un texto con fondo negro en la imagen pasada por parámetro.
loadFromQrc	Carga un recurso de la estructura Qrc de la carpeta "Resources". Se utiliza para obtener las imágenes para la función *cargarImagenInicial*.
mouseHandler	Proceso interno, asociado al botón "Ok" que se llama cuando el usuario ha elegido los modos 2,3 o 5. Permite al usuario, mediante el ratón y teclado, elegir una o varias zonas de interés de la escena para su posterior procesado.

Tabla 2. Funciones declarada y descripción de su uso.

- **Carpeta "Sources":** Contiene los ficheros de extensión .cpp que componen el grueso de la programación de la aplicación. Estos archivos contienen el funcionamiento y la programación en crudo de las funciones y clases definidas en los ficheros de cabecera anteriores. En la tabla 3 se especifica los archivos que contiene y una explicación del contenido de cada uno de ellos.

Fichero	Contenido
cvaux.cpp	Asociada a la librería CvBlob. Contiene funciones auxiliares que utilizan los procesos tales como calcular distancias de un punto a una línea o de punto a punto.
cvblob.cpp	Asociada a la librería CvBlob .Contiene las funciones para definir, dibujar y calcular las componentes conexas.
cvcolor.cpp	Asociada a la librería CvBlob. Contiene una función que calcula el color medio de una componente que se pasa por parámetro.
cvcontour.cpp	Asociada a la librería CvBlob. Contiene métodos para el cálculo de los contornos a partir de un imagen binaria con una máscara de foreground.
cvlabel.cpp	Asociada a la librería CvBlob. Modela una clase con sus funciones para etiquetar a cada componente hallada durante el análisis.
cvtrack.cpp	Asociada a la librería CvBlob. Modela una clase con sus funciones para definir un objeto de tracking.
main.cpp	Lanza la aplicación. Crea un formulario, un objeto de la clase Mainwindow y lo ejecuta.
mainwindow.cpp	Contiene las funciones programas para realizar la lógica de la interacción del usuario con la aplicación así como la lógica del procesado principal de las imágenes.

Tabla 3. Ficheros .cpp con descripción de su contenido.

- **Carpeta "Forms":** Contiene los formularios de la aplicación. Los formularios modelan las ventanas gráficas y sus elementos de la propia aplicación. En este caso solo contiene el formulario *mainwindow.ui* ya que la aplicación sólo cuenta con una ventana principal. El archivo *mainwindow.ui* contiene el código en xml que crea todos los elementos visuales (botones, etiquetas, etc…) de la interfaz gráfica y los dota de un identificador único que servirá para modelar su comportamiento mediante las funciones de los archivos .h y .cpp.

- **Carpeta "Resources":** Contiene el almacenamiento de los recursos externos utilizados en la aplicación. Estos recursos pueden ser archivos de cualquier tipo. En este caso se trata de las imágenes que se muestran en el inicio de la aplicación o cuando se está cargando algún recurso.

Cabe destacar la importancia de los archivos "main.cpp" y "mainwindow.cpp". El primero constituye la función *main*, la cual sirve como punto de partida de la ejecución del programa. Cuando el programa se ejecuta se llama a la función *main*, esta crea un objeto *QApplication* y un objeto *MainWindow*. Después la función otorga al usuario el control de la aplicación utilizando el método *exec* del objeto *QApplication*. A partir de este momento el usuario controla la ejecución del programa a través de la interacción

con el formulario *mainwindow.ui* cuyos elementos están asociados a las funciones de *mainwindow.cpp*.

Dentro del fichero *mainwindow.cpp* las funciones más importantes son las siguientes:

- **Constructor**: En el constructor se definen los parámetros predeterminados que aparecerán cuando el usuario ejecute el programa así como los ajustes principales. Es mencionable aquí el uso de la clase *QTimer*. La clase *QTimer* modela un objeto *timer* que se ha de conectar con una función. Cuando el *timer* es lanzado, su función asociada se ejecuta una vez cada *t* milisegundos, siendo *t* un parámetro propio del *timer*. En el constructor se conecta el *timer* a la función *timerLoopPrincipal*. Una vez conectado se asocia el lanzamiento del *timer* a la acción de, por ejemplo, presionar un botón. Una vez lanzado la función *timerLoopPrincipal* se ejecuta para cada frame siendo tu parámetro *t* igual a su tasa de *frames* por segundo, ya que el proceso se realiza para cada frame. Cuando el proceso acaba el *timer* se para. Además de las configuraciones de los *timers*, en el constructor también se conectan acciones como la acción de cerrar la aplicación.

- **on_*okButton*_clicked:** Se trata de la función asociada al pulsar el botón "Ok" de la ventana principal. Cuando el botón es pulsado se realizan validaciones sobre la fuente de imágenes elegida para evitar errores en el procesamiento. Una vez realizada las validaciones, y según el modo de control elegido, se pasa a preparar el lanzamiento del proceso principal. Si se han elegido los modos 2, 3 y 5, será necesario que el usuario elija, mediante un proceso con el ratón, una o varias zonas de interés. Cuando todo se encuentre listo, se lanza el *timer* asociado al proceso principal. Si se vuelve a pulsar el botón "Ok" durante el proceso principal, se accederá a una parte de la función que para el *timer* y devuelve el control de la aplicación.

- ***timerLoopPrincipal*:** Se trata de la función que se ejecuta cuando se lanza el *timer* desde la función *on_okButton_clicked*. Constituye el grueso de la ejecución del programa ya que dentro de ella se hace todo el procesamiento de las imágenes (*frame* por *frame)* de principio a fin. Esto es desde la adecuación de la imagen para su tratamiento posterior hasta mostrar finalmente los resultados en el visor. Como ya se ha comentado, esta función es lanzada por el *timer* cada *t* milisegundos siendo *t* el parámetro la tasa de *frames* final. Como solución de compromiso para mantener un buen flujo de *frames* sin comprometer demasiado el tiempo de ejecución, se establece este parámetro *t* en 400 milisegundos, limitando así la tasa a 25 *frames* por segundo.

3.6.4 Especificaciones técnicas de desarrollo y ejecución

El desarrollo de la aplicación en todas sus fases se ha realizado en un terminal con las siguientes características:

- Procesador: Intel Core i7-4790K 4.0 GH
- Memoria RAM: 32.00 GB.
- Sistema operativo: Windows 7 de 64 bits.

No obstante, para la ejecución de la aplicación se establecen los siguientes requisitos mínimos:

- Procesador: Inter Core i3
- Memoria RAM: 4.00 GB.
- Sistema operativo: Windows 7 de 64 bits o superiores.

4. Experimentación y pruebas

En este capítulo se realiza la experimentación de la aplicación poniendo a prueba su uso desde varios puntos de vista, entre ellos, el de la precisión o el de la eficiencia.

Se realizaron pruebas en numerosos ambientes y situaciones. No obstante hay que tener en cuenta los objetivos que se tuvieron al diseñar la aplicación. Estos dictan que la detección se hará de día y en ambientes controlados de tránsito de vehículos. Teniendo esto en cuenta quedan excluidos de las pruebas aquellas realizadas en ambientes nocturnos y de escasa iluminación. Por otro lado, también quedan fuera de las baterías de pruebas aquellas realizadas con cámara en movimiento. Dado que la aplicación finalmente ha hecho uso de algoritmos de sustracción de background, no es posible realizar detección ya que si la escena está constantemente en movimiento nunca se realizará una sustracción precisa.

Una de las mejores maneras de evaluar la precisión de la aplicación en la detección sería evaluar *frame* por *frame* en cada prueba la tasa de falsos positivos y falsos negativos. Dado el volumen de fotogramas manejados en cada prueba es muy poco práctico realizar una valoración de esta manera. Debido a esta razón, se han realizado pruebas para hallar la precisión en la detección general y para hallar la eficiencia de cada tipo de control disponible en la aplicación. Por ello, se ha dispuesto de una batería de pruebas realizada con una serie de vídeos encontrados en Internet así como algunos grabados por fuente propia.

En este capítulo es necesario definir un concepto importante que ayuda a valorar la eficiencia de la aplicación, tal y como se muestra en el documento referenciado [44]. Se trata de la matriz de confusión que se muestra en la figura 64.

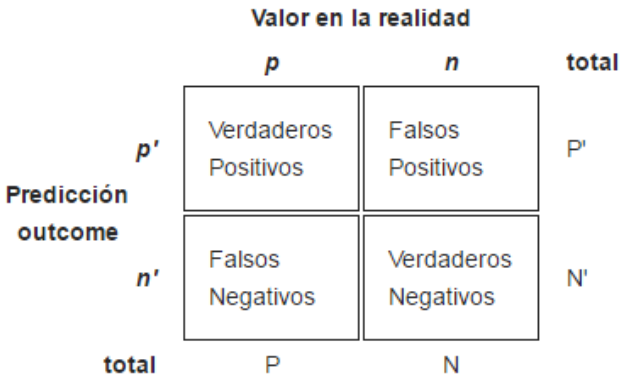


Figura 64. Matriz de confusión para la valoración del algoritmo de detección de vehículos [44]

Como se observa en la figura 65, se realizarán predicciones teniendo en cuenta el comportamiento de la aplicación. Las predicciones pueden tomar valores negativos o

positivos dependiendo de su tipo. Si la predicción coincide con el valor del suceso en la realidad entonces se hablará de un suceso verdadero pudiéndose ser Verdadero Positivo (VP) o Verdadero Negativo (VN). En cambio, si la predicción no coincide se trata de un suceso falso, ya sea Falso Positivo (FP) o Falso Negativo (FN).

Para el caso concreto de la detección de vehículos la matriz de confusión se explicaría de la siguiente manera:

- **Falso positivo:** la aplicación detecta un vehículo pero este no se encuentra en escena.
- **Falso negativo:** la aplicación no detecta un vehículo que si se encuentra en escena.
- **Verdadero positivo:** la aplicación detecta un vehículo en la escena cuando realmente lo hay.
- **Verdadero negativo:** la aplicación no detecta un vehículo en la escena cuando realmente no lo hay.

Por otro lado, cuando se esté valorando la eficiencia de la aplicación en el caso del control de tráfico en sus distintas modalidades, la matriz de confusión se entiende de manera distinta. En este caso toma relevancia el concepto de suceso en el contexto de control de tráfico. Se entiende por suceso como un evento que activa o modifica el mecanismo de control de tráfico de la escena. Así pues, la matriz de confusión se explicaría de la siguiente manera:

- **Falso positivo:** la aplicación detecta un suceso pero este no ocurre en escena.
- **Falso negativo:** la aplicación no detecta un suceso que si ocurre en escena.
- **Verdadero positivo:** la aplicación detecta un suceso en la escena cuando realmente ocurre.
- **Verdadero negativo:** la aplicación no detecta un suceso en la escena cuando realmente no ocurre.

Además también se hará uso de la curva *ROC* la cual permitirá dar una idea sobre la eficacia de la aplicación y su funcionamiento. Para el cálculo de la curva son necesarios dos valores más hallados en base a los explicados anteriormente. Estos valores se encuentran en las ecuaciones 36 y 37.

- Ratio de verdaderos positivos (VPR)

$$VPR = \frac{VP}{VP + FN} \qquad (36)$$

- Ratio de falsos positivos (VPR)

$$FPR = \frac{FP}{FP + VN} \qquad (37)$$

Para hallar la curva ROC esta se representará en una gráfica tomando los valores de FPR como los valores de abscisas y VPR como los valores de ordenadas.

4.1 Detección de vehículos en varios escenarios

Para realizar las pruebas de funcionamiento y eficiencia de la detección de vehículos en general de la aplicación se han utilizado varios vídeos.

El primero de ellos es una escena donde un coche por control de remoto se mueve por el suelo de una habitación donde a veces queda ocultado por una caja, provocando oclusión, como aparece en la figura 65.

Figura 65. Escena de coche por control remoto. Fuente [43].

El segundo de ellos es una escena donde se muestra una carretera urbana de doble sentido en la cual los vehículos van circulando, como se muestra en la figura 66. Al ser un plano cercano los vehículos pasan poco tiempo en escena, lo que puede dificultar la detección a priori.

Figura 66. Escena de carretera de doble sentido. Plano horizontal superior. Fuente [43].

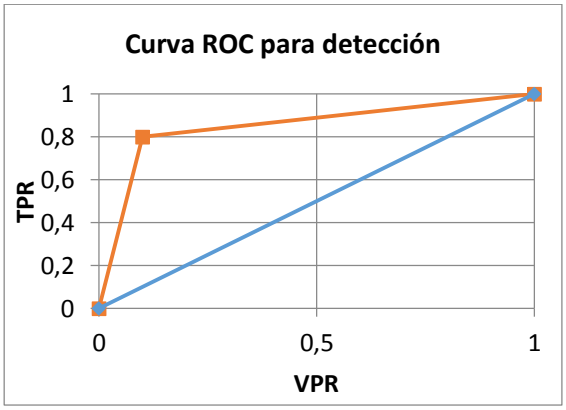
Los resultados de la prueba se muestran en la tabla 4.

Video (duración)	Frames	VN	VP	FP	FN	Comentarios
Coche control remoto (37 s)	945	145	800	18	2	Un solo vehículo con oclusión. Imagen algo distorsionada por ruido digital
Vista Horizontal Arriba (30 s)	725	500	225	39	4	Numerosos vehículos con poco tiempo en escena. Plano cercano.

Tabla 4. Resultados de pruebas de detección

En ambos casos se puede observar que se obtiene una gran tasa de verdaderos positivos, por lo que el reconocimiento, la detección y el seguimiento son fiables con baja tasa de errores.

En vista de estos resultados, y dando lugar a un VPR medio de 0,92 y un FRP medio de 0,085, la curva ROC quedaría tal y como se muestra en la gráfica 1.



Gráfica 1. Curva ROC resultado de medir la eficiencia de la detección

A la vista de la curva ROC (color naranja) de la gráfica 1, se puede observar que esta queda por encima de la línea de corte (color azul), considerando el corte como el límite de una eficiencia media. Por tanto, la aplicación es eficiente a la hora de hacer la detección de vehículos, lo que facilitará los controles posteriores sobre dichas detecciones.

En este apartado es relevante comentar la detección de las posibles sombras en la escena, dado que la detección de las mismas se puede interpretar como falsos positivos. Como ya se ha comentado anteriormente, las sombras en movimiento puede interpretarse como objetos en movimiento ya que al fin y al cabo, se trata de píxeles cuyos valores varían de gran manera a lo largo del tiempo. No obstante, el algoritmo de sustracción de background empleado en la aplicación permite (en cierta parte) eliminar de la máscara de foreground las sombras de los objetos en movimiento. Esto se controla mediante un parámetro del objeto sustractor de background llamado *fTau*. Este

parámetro funciona como un valor umbral, donde si un pixel es *fTau* veces más oscuro que el pixel considerado como foreground, entonces dicho pixel no es tratado como una sombra.

4.2 Control de vehículos en una zona de interés

Para probar la eficiencia de este tipo de control se tomará como suceso positivo que el vehículo esté dentro de la zona delimitada y negativo que el vehículo esté fuera.

En primer lugar se realizarán pruebas sobre el vídeo del coche de control remoto, donde situaremos la zona de interés tal y como se muestra en la figura 67.

Figura 67. Colocación de la zona de interés en la primera prueba. Fuente [43].

En segundo lugar se realizará de nuevo la misma prueba en un vídeo tomado desde un plano horizontal a altura media, donde los vehículos circulan en ambos sentidos, tal y como se ve en la figura 68.

Figura 68. Colocación de la zona de interés en la segunda prueba. Fuente propia.

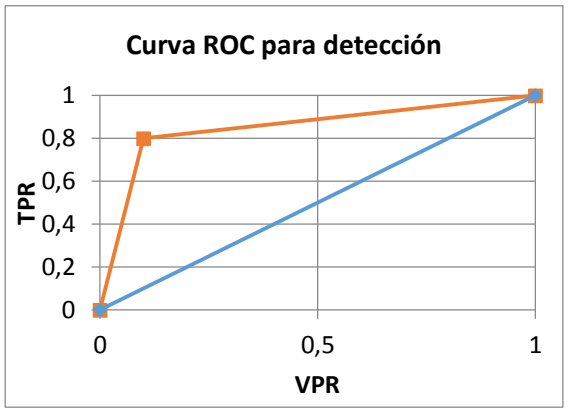
Los resultados de las pruebas se muestran en la tabla 5.

Video (duración)	Frames	VN	VP	FP	FN	Comentarios

Coche control remoto (37 s)	945	557	383	2	3	Un solo vehículo con oclusión. Imagen algo distorsionada por ruido digital
Vista Horizontal Media (30 s)	545	520	22	2	1	Numerosos vehículos con poco tiempo en escena. Plano medio en horizontal paralelo a la vía.

Tabla 5. Resultado de las pruebas en el control de vehículos en una zona de interés

Dando como resultado un VPR medio de 0,99 y un FPR medio de 0,003, la curva ROC para la eficiencia de este control queda como se muestra en la gráfica 2.



Gráfica 2. Curva ROC para la eficiencia del control de vehículos en una zona determinada.

Dados lo anteriormente calculado en las pruebas de detección es lógico que el resultado al medir la eficiencia de esta prueba sean buenos. Esto es así ya que el control de la zona depende directamente de que se realice o no una detección y seguimiento correctos en dicha zona.

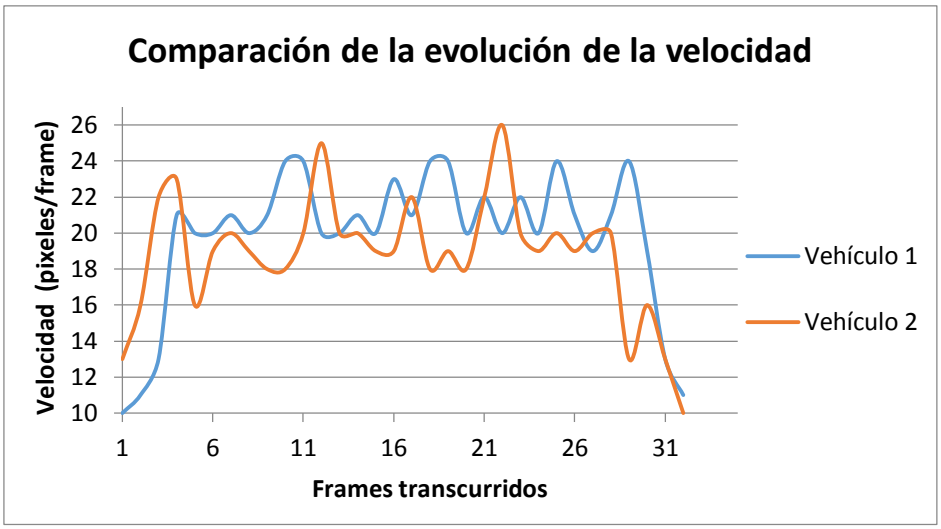
4.3 Control de velocidad de vehículos

Para hallar la eficiencia del control de velocidad de los vehículos detectado no es útil basar la prueba mediante la matriz de confusión como se ha hecho hasta ahora. Una forma de comprobar que la velocidad mostrada por pantalla coincide con la velocidad real del vehículo es observando que la velocidad calculada se mantiene relativamente constante para aquellos vehículos que durante la escena apenas experimentan aceleración.

Para verificar esto se escogerá un vídeo donde, frame a frame, se observará la evolución de la velocidad de un vehículo en su paso de por la escena. El vídeo está tomado desde un plano horizontal a altura media, donde los vehículos circulan en ambos sentidos, tal y como se ve en la figura 69.

Figura 69. Escenario de prueba para la eficiencia del control de la velocidad. Fuente propia.

En la gráfica 3 se observa la comparación entre la evolución de la velocidad en píxeles por frame de los dos primeros vehículos que aparecen en la escena.



Gráfica 3. Evolución de la velocidad

A la vista del gráfico se puede observar que a partir del frame 6, la velocidad se mantiene constante dentro de un ±2 pixeles/frame. Se debe exceptuar la parte inicial y final de gráfico, que coincide con los momentos en el que los vehículos aún no han entrado ni ha salido de la escena del todo. Por tanto, se puede llegar a la conclusión de que visto que en el vídeo ningún vehículo acelera ni frena de manera pronunciada y la velocidad se muestra más o menos constante, la velocidad en la aplicación se muestra de igual manera constante.

4.4 Control de estacionamiento en un área delimitada

Para hallar la eficiencia del control del estacionamiento se entenderá como suceso positivo los siguientes dos hechos:

- El vehículo entra en la zona de estacionamiento y esta cambia su estado a "ocupado" mientras queda a la espera de que se mueva el vehículo de nuevo.
- El vehículo entra en la zona de estacionamiento y esta cambia su estado a "ocupado", queda en reposo en busca de movimiento y cuando el vehículo finalmente se mueva y salga de la zona, se cambia el estado a "vacío".

Sus correspondientes en negativo serían así:

- El vehículo no entra en la zona de estacionamiento o.
- El vehículo al entrar en la zona de estacionamiento es detectado, pero tras estar en reposo no vuelve a salir de la zona.

Para evaluar estas situaciones se probará la aplicación con una escena de un parking en el que entran dos vehículos. Uno de ellos permanece en el parking hasta el final del vídeo pero el otro, tras permanecer quieto en la zona, vuelve a salir. Una toma de la escena se puede observar en la figura 70.

Figura 70. Escena utilizada para evaluar la eficiencia del control de estacionamiento. Fuente [43].

Los resultados de las pruebas se pueden observar en la tabla 6.

Video (duración)	Frames	VN	VP	Comentarios
Parking Coche 1 (35s)	875	375	500	Vehículo que entra al parking y queda estacionado hasta fin de vídeo.
Parking Coche 2 (1:10 mins)	1750	875	875	Vehículo que entra al parking, estaciona y se mantiene quieto hasta que finalmente liberal la zona.

Tabla 6. Resultados de la pruebas para la medida de la eficiencia del control de estacionamiento

En la figura 71 y 72 se pueden ver las secuencias resultado de ambas pruebas.

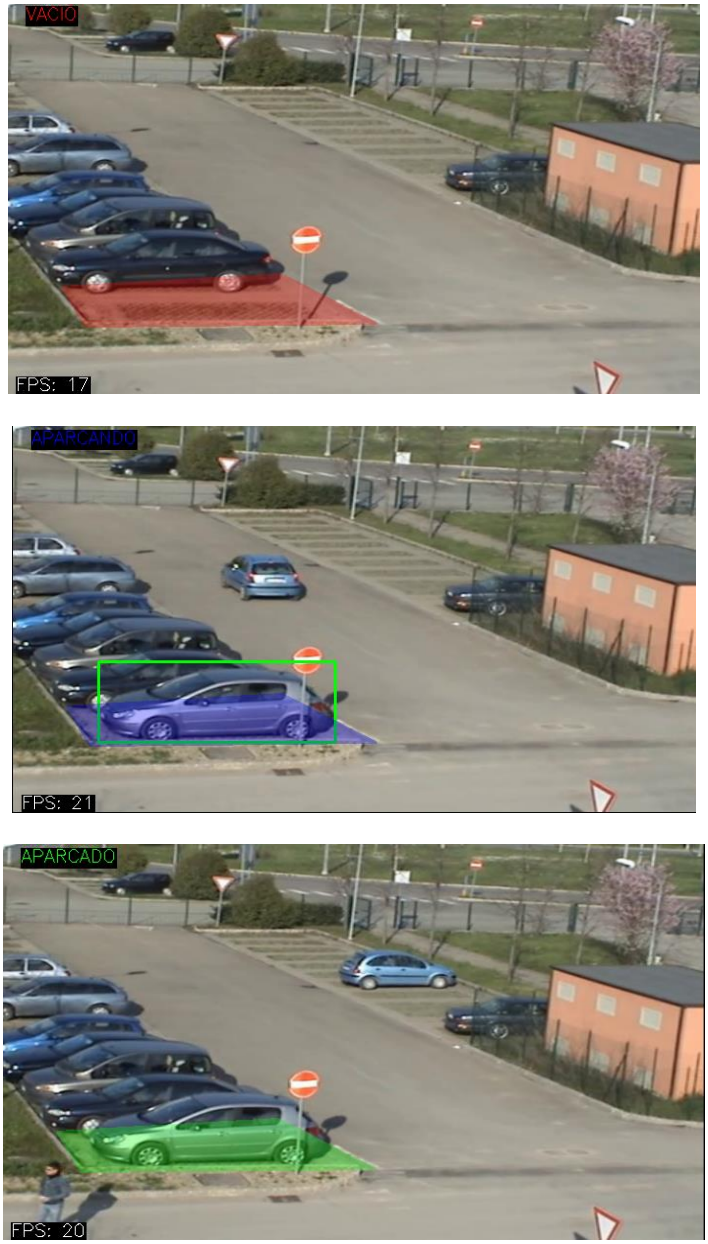


Figura 71. Secuencia resultado de la primera prueba. De arriba abajo, se puede observar como la zona va cambiando de color de vacío a aparcando cuando el vehículo entra, quedando finalmente como ocupado. Fuente [43].

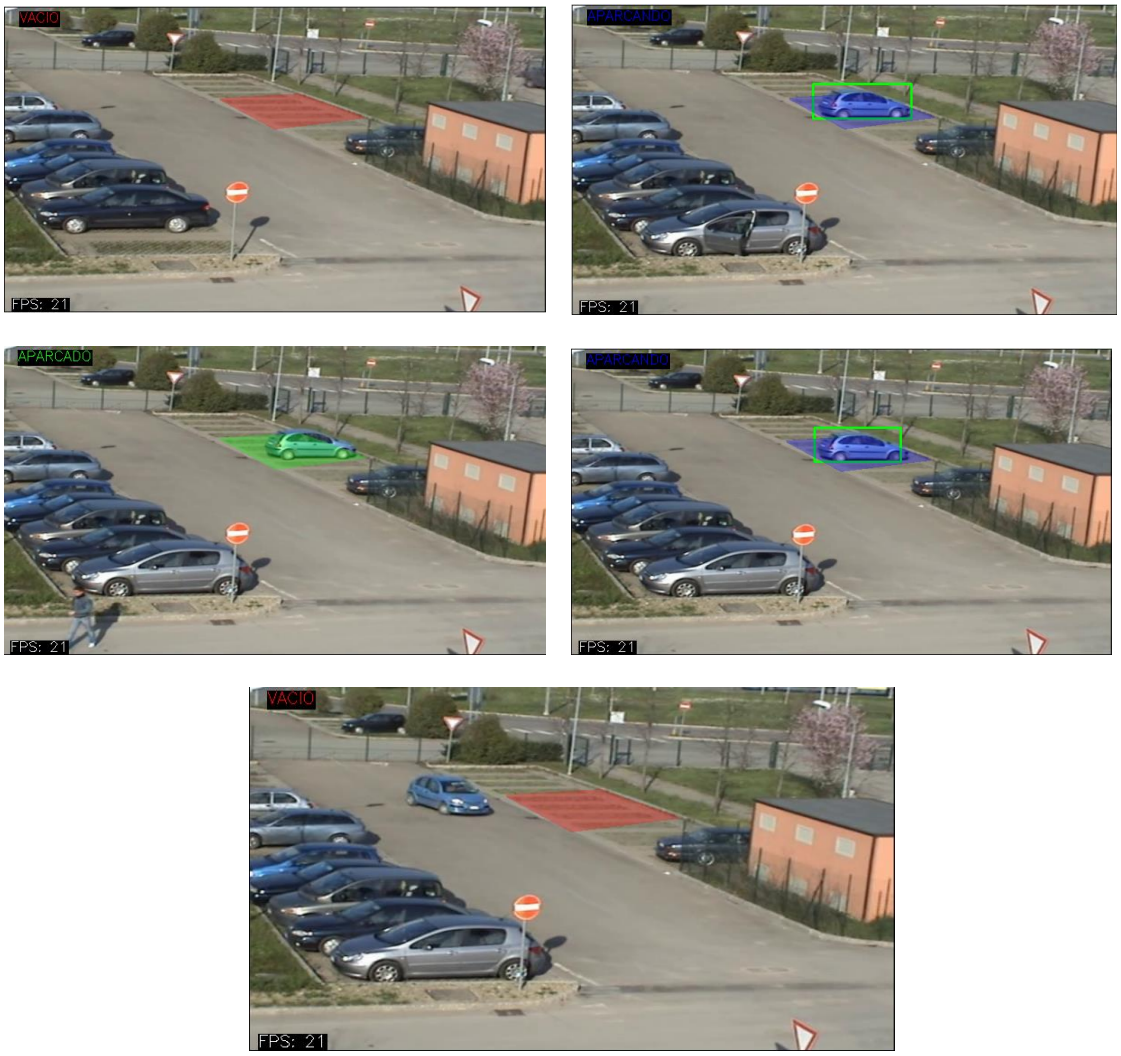


Figura 72. De izquierda a derecha y de arriba a abajo, secuencia resultado de la segunda prueba. Se observa claramente como el vehículo reanuda su marcha tras haber estado estacionado en la zona, provocando el cambio de estado y de color de la misma. Fuente [43].

A la vista de los resultados obtenidos se puede observar el funcionamiento correcto del control de estacionamiento implementado.

Un detalle importante a tener en cuenta, y que se ha podido ver en la primera prueba, es que una vez estacionado el vehículo en la zona los pasajeros y conductor suelen tardar varios segundos en salir. Esto se traduce en pequeños movimientos en los píxeles al fin y al cabo que pueden ser reconocidos, lo que puede provocar, a priori, que la aplicación crea que el vehículo esté intentando salir de la zona, provocando falsos positivos. Para evitar esto se ha implementado el siguiente mecanismo. Al pararse el vehículo, habrá ausencia de movimiento y por ende de detección, por tanto, se le dejará de reconocer. En ese instante comienza un contador de tiempo de veinte segundos. Durante este tiempo se ignorará cualquier tipo de movimiento que haya en la zona de aparcamiento.

De esta manera los ocupantes del vehículo bajaran del mismo y su movimiento no será detectado. Pasados esos segundos se volverá a detectar movimiento y de esta manera se podrá saber si el vehículo va a dejar la zona.

4.5 Control de entrada/salida y conteo de vehículos en una zona

Para evaluar la eficiencia de este control se entenderá como suceso positivo que un coche, ya sea al entrar o a salir de la zona delimitada, atraviese ambas zonas previas 1 y 2, y además se modifique el contador (ya sea aumentando o disminuyendo).

El escenario de pruebas es el mismo vídeo que en el caso anterior, ya que es una zona de parking en la que hay dos entradas y una salida.

En la tabla 7 se puede observar los resultados de las pruebas.

Video (duración)	Frames	VN	VP	Comentarios
Parking Coche 2 (1:19 mins)	1975	1800	175	Dos vehículos entran a la zona de parking. Tras unos segundos, uno se queda dentro de la zona pero el otro la abandona

Tabla 7. Resultados de la pruebas para la medida de la eficiencia del control de entrada/salida y conteo

La secuencia resultado de la prueba se puede observar en la figura 73.

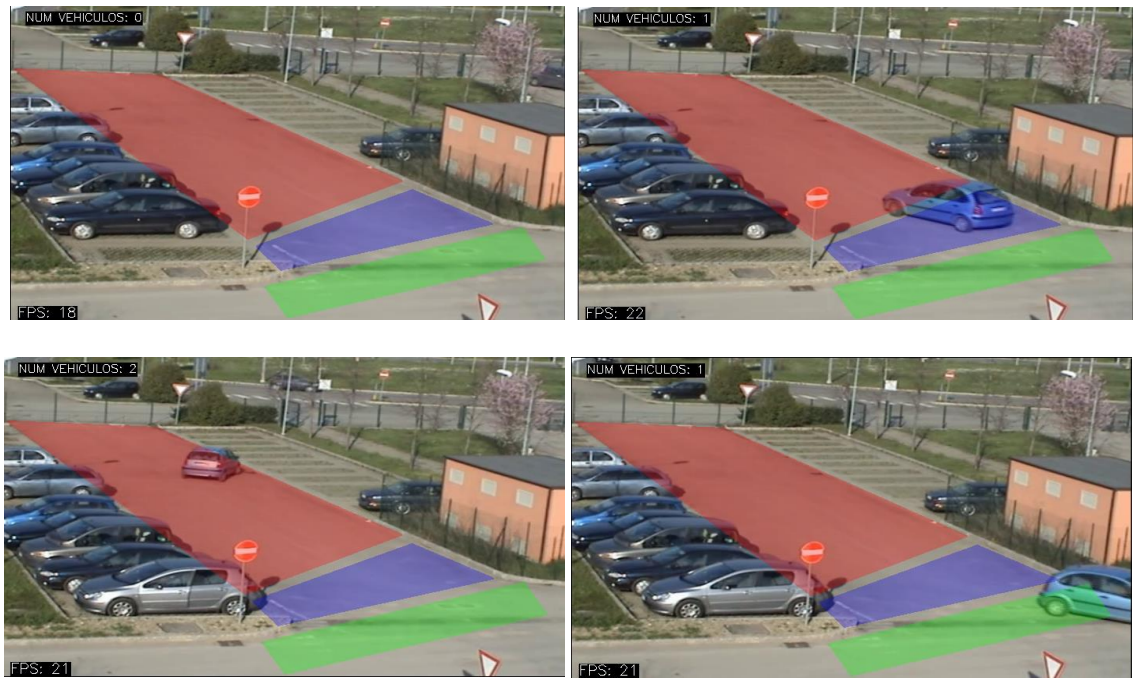


Figura 73. De arriba a abajo y de izquierda a derecha, secuencia seguida por la aplicación al probar la eficiencia del control de entrada/salida y conteo. Se puede observar como aumenta y disminuye el contador de vehículos. Fuente [43].

A la vista de los resultados, se puede observar el buen funcionamiento del control tanto en la entrada como en la salida de los vehículos. De nuevo cobra vital importancia los buenos resultados obtenidos anteriormente en las pruebas de detección, ya que influye enormemente en el reconocimiento de los vehículos en las zonas de entrada y salida.

5. Conclusiones y líneas de trabajo futuro

5.1 Conclusiones

En esta memoria del Proyecto de Fin de Grado que se presenta, se ha desarrollado una aplicación que permite realizar tareas de vigilancia y control de tráfico vehicular a partir de una fuente de imágenes o cámara mediante técnicas de visión artificial y realidad aumentada. La implementación se ha basado principalmente en tres algoritmos: sustracción de background por Mezclas Gaussianas, análisis y etiquetado de las componentes del foreground halladas y seguimiento de las mismas por Modelos de Apariencia. En la parte técnica, y desde el principio del proyecto, la implementación se ha basado en la elección de las librerías de OpenCV como sustento principal, así como librerías adicionales de cvBlob.

En las primeras partes del desarrollo del proyecto se optó por el estudio de la técnica HOG + SVM + Flujo Óptico. Tras el estudio, se realizó una serie de implementaciones por aplicación de consola. No se obtuvieron los resultados adecuados ya que no se cumplían los requisitos establecidos de tasa baja de errores y fluidez en la aplicación. Por tanto, se decide no continuar con el desarrollo por esa parte y cambiar de estrategia.

Se elige estudiar la técnica de Sustracción de Background + Análisis y Etiquetado de Componente + Tracking por Modelos de Apariencia. Gracias a la sustracción de background, la aplicación puede detectar el movimiento en la escena que se asocia a los vehículos que circulan por ella, lo cual es muy provechoso dada la naturaleza del proyecto. El análisis y etiquetado de las componentes de foreground extraídas facilitan el seguimiento por modelos de apariencia ya que cada vehículo queda debidamente etiquetado en cada momento. Tras realizar una serie de implementaciones sencilla se decide optar por dicha opción ya que cumplía gran parte de los requisitos impuestos.

Durante la implementación final de la opción escogida, se diseñó el código final y se implementa junto a una interfaz gráfica desarrollada en C++ en el framework Qt Creator. Es en esta etapa donde además se desarrollan las opciones de control del tránsito vehicular. Por otro lado, la interfaz permite operar al usuario de forma fácil e intuitiva y además, mostrar los elementos de detección y control de Realidad Aumentada de forma muy visual.

El último paso en la elaboración de este proyecto fue la experimentación y las pruebas de eficiencia de la aplicación. En ellas se obtuvieron, en general, buenos resultados en la detección y en la eficiencia del control de las distintas situaciones planteadas con los sets de vídeo probados.

El resultado final es una aplicación de vigilancia dedicada al control de distintas situaciones en varios escenarios de tránsito vehicular añadiendo elementos visuales de realidad aumentada para la alerta y la interacción del usuario, funcionando además en una tasa de frames por segundo adecuada tomando en cuenta los requisitos mínimos técnicos. Aunque se hayan alcanzado los requisitos funcionales planteados en el desarrollo, en el siguiente capítulo se muestran posibles mejoras de la aplicación así como líneas de trabajo futuro.

5.2 Trabajo futuro

Dada la naturaleza de la aplicación implementada durante el desarrollo de este proyecto es posible la ampliación de su alcance mediante las siguientes mejoras:

- **Cálculo de la velocidad real**. Uno de los aspectos de la opción del control de la velocidad es que esta viene dada en píxeles/frame. Aunque en el apartado 3.5.5 se especifica una manera de mostrar la velocidad en metros por segundo, no se trata de un método fiable. Por ello se propone como trabajo futuro encontrar una manera de obtener la equivalencia pixel/metro en imágenes de vídeo para el cálculo de la velocidad en esta aplicación.

- **Varias zonas de aparcamiento.** La opción del control de estacionamiento sólo contempla la posibilidad de un vehículo. Una posible mejora sería la posibilidad de controlar varias zonas de aparcamiento simultáneamente.

- **Clasificación de tipo de vehículos y personas por tamaño.** Otra posible mejora de la aplicación es ofrecer una clasificación de los vehículos detectados por tamaño distintos (por ejemplo: turismos, camiones, motocicletas), pudiendo ser posible además la detección e identificación de personas en la escena para ambientes de tráfico urbano.

- **Mejora de control de entrada y salida.** El diseño de dicha opción está pensado para sólo ciertos modelos de parking o aparcamientos donde la entrada y la salida se encuentran en la misma zona. Se propone por ello mejorar el control teniendo en cuenta que muchas de estos aparcamientos tienen situadas en distintos lugares su salida y su entrada.

6. Bibliografía

[1] BOE, «https://www.boe.es,» BOE, Agosto 1997. [En línea]. Available: https://www.boe.es/buscar/doc.php?id=BOE-A-1997-17574.

[2] L. G. S. a. G. C. Stockman, Computer Vision, Prentice Hall, 2001.

[3] T. Morris, Computer Vision and Image Processing, Palgrave Macmillan, 2004.

[4] Quercus S.A, «Quercus,» 12 Julio 2015. [En línea]. Available: http://quercus.biz/birdwatch%C2%AE-red-light.

[5] Universidad Rey Juan Carlos, «Vicerrectorado de Investigación de la URJC,» URJC, 25 Febrero 2015. [En línea]. Available: http://www.ucci.urjc.es/autobuses-urbanos-nuevos-agentes-de-movilidad/.

[6] ASSET, «Project ASSET Road,» Proyecto de la U.E, 25 Enero 2009. [En línea]. Available: http://www.project-asset.com/.

[7] Tecnocarreteras, «www.tecnocarreteras.es,» Tecnocarreteras, 22 Diciembre 2011. [En línea]. Available: https://www.tecnocarreteras.es/2011/12/22/asset-el-sistema-de-vigilancia-total-de-la-carretera/.

[8] Tecnocarreteras, «Tecnocarreteras,» Tecnocarreteras, 8 Enero 2012. [En línea]. Available: https://www.tecnocarreteras.es/2012/01/08/sistema-de-vision-artificial-que-informa-de-la-densidad-de-trafico-en-cada-zona/.

[9] Invaringenieria, «www.invaringenieria.com,» Invar, 4 Noviembre 2011. [En línea]. Available: http://www.invaringenieria.com/vision-artificial/sistemas-invar.

[10] C. H. y. M. Stephens, «A combined corner and edge detector,» de *Proceedings of the 4th Alvey Vision Conference 147–151*, 1988.

[11] J. S. a. C. Tomasi., «Good Features to Track.,» *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* pp. 593-600, 1994.

[12] OpenCV, «OpenCV Documentation Harris Detector,» Intel, 10 Noviembre 2014. [En línea]. Available: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html .

[13] Universidad de Granada, «http://www.ugr.es/~rpaya/documentos/Teleco/Fund-Mat02.pdf,» Universidad de Granada. Telecomunicaciones, 2 Febrero 2009. [En línea]. Available: http://www.ugr.es/~rpaya/documentos/Teleco/Fund-Mat02.pdf.

[14] N. D. y. B. Triggs, «Histograms of Oriented Gradients for Human Detection,» de *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* , San Diego, USA, 2005.

[15] Coursera, «https://class.coursera.org/deteccionobjetos-001,» Coursera, 30 Abril 2015. [En línea].

[16] E. J. Carmona., «http://www.ia.uned.es/~ejcarmona/publicaciones/[2013-Carmona]%20SVM.pdf,» 2013. [En línea]. Available: http://www.ia.uned.es/~ejcarmona/publicaciones/[2013-Carmona]%20SVM.pdf.

[17] J. L. A. Castro, «http://web.archive.org/web/20140801145654/http://www.gts.tsc.uvigo.es/~jalba/doctorado/SVM.pdf,» 14 Junio 2013. [En línea]. Available: http://web.archive.org/web/20140801145654/http://www.gts.tsc.uvigo.es/~jalba/doctorado/SVM.pdf.

[18] Wikipedia, «Support Vector Machine,» Wikipedia, 27 Julio 2002. [En línea]. Available: https://en.wikipedia.org/wiki/Support_vector_machine.

[19] «OpenCV SVM Documentation,» Mayo 2015. [En línea]. Available: http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html.

[20] F. P. B. H. A. V. Thierry Bouwmans, Background Modeling and Foreground Detection for Video Surveillance, CRC Press, 2014.

[21] B. Tamersoy, «http://www.cs.utexas.edu/~grauman/courses/fall2009/slides/lecture9_background.pdf,» 29 Septiembre 2009. [En línea]. Available: http://www.cs.utexas.edu/~grauman/courses/fall2009/slides/lecture9_background.pdf.

[22] M. Piccardi, «Background subtraction techniques: a review,» de *2004 IEEE International Conference on Systems, Man and Cybernetics*, 2004.

[23] O. J. T. K. Yaser Sheikh, «Background Subtraction for Freely Moving Cameras,» de *2009 IEEE 12th International Conference on Computer Vision*, Kyoto, 2009.

[24] N. Otsu, «A Tlreshold Selection Method from Gray-Level Histograms,» *IEEE Transactions on Systems, Man, and Cybernetics ,* vol. 9, nº 1, pp. 62-66, 1979.

[25] A. A. T. D. A. P. Christopher Wren, «Pfinder: real-time tracking of the human body,» *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 19, p. 5, Julio 1997.

[26] C. Wren, A. Azarbayejani, T. Darrell y A. Pentland, «Adaptive background mixture models for real-time tracking,» de *IEEE Computer Society Conference on Computer Vision and Pattern Recognition.*, (July 1997).

[27] O. J. M. S. Alper Yilmaz, «Object Tracking: A Survey,» *Journal ACM Computing Surveys (CSUR),* vol. 38, nº 4, p. 45, 2006.

[28] G. L. Dengsheng Zhang, «Segmentation of moving objects in image sequence: A review,» *Circuits, Systems and Signal Processing ,* vol. 20, nº 2, pp. 143-183, 2003.

[29] R. E. Kalman, «A New Approach to Linear Filtering and Prediction Problems,» *Transactions of the ASME - Journal of Basic Engineering ,* vol. 82, pp. 35-45, 1960.

[30] T. K. Bruce D. Lucas, «An Iterative Image Registration Technique with an Application to Stereo Vision,» de *International Joint Conference on Artificial Intelligence*, 674–679, 1981.

[31] Y. Cheng, «Mean Shift, Mode Seeking, and Clustering,» *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 17, nº 8, pp. 790-799, Agosto 1995.

[32] G. R. Bradski, «Real time face and object tracking as a component of a perceptual user interface,» de *Applications of Computer Vision, 1998. WACV '98.*, Princeton, NJ, 1998.

[33] OpenCV, «OpenCV Documentation,» Intel, 10 Noviembre 2014. [En línea]. Available: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html .

[34] C. Carnero, «cvBlob,» 2008. [En línea]. Available: http://cvblob.googlecode.com.

[35] C.-J. C. a. C.-J. L. Fu Chang, «A linear-time component-labeling algorithm using contour tracing technique,» *Computer Vision and Image Understanding,* vol. 93, nº 2, pp. 206-220, 2004.

[36] A. H. Y.-L. T. L. B. S. P. R. B. Andrew Senior, «Appearance models for occlusion handling,» *Image and Vision Computing,* vol. 24, nº 11, p. 1233–1243, 2006.

[37] IIPImage, «http://iipimage.sourceforge.net/documentation/images/,» 2014. [En línea]. Available: http://iipimage.sourceforge.net/documentation/images/.

[38] C.-C. C. a. C.-J. Lin, «LIBSVM: a library for support vector machines,» *ACM Transactions on Intelligent Systems and Technology,* vol. 2, pp. 1-27, 2001.

[39] Beachfront B-Roll, «http://www.beachfrontbroll.com/2012/01/1-clip-needs-frog.html,» 24 Enero 2012. [En línea]. Available: http://www.beachfrontbroll.com/2012/01/1-clip-needs-frog.html.

[40] OpenCV, «http://docs.opencv.org/3.1.0/d4/d13/tutorial_py_filtering.html#gsc.tab=0,» Febrero 2015. [En línea]. Available: http://docs.opencv.org/3.1.0/d4/d13/tutorial_py_filtering.html#gsc.tab=0.

[41] Z. Zivkovic, «Improved adaptive Gaussian mixture model for background subtraction,» de *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference*, 204.

[42] R. B. P. Kaewtrakulpong, «An Improved Adaptive Background Mixture Model for Realtime Tracking with Shadow Detection,» de *2nd European Workshop on Advanced Video Based Surveillance Systems*, 2001.

[43] R. C. R. Vezzani, «Video Surveillance Online Repository (ViSOR): an integrated framework,» *Multimedia Tools and Applications,* vol. 50, nº 2, pp. 359-380, 2010.

[44] S. M. Lopez H., «Detección y seguimiento de objetos con cámaras de movimiento.,» Septiembre 2011. [En línea]. Available: http://arantxa.ii.uam.es/~jms/pfcsteleco/lecturas/20110930HectorLopezParedes.pdf.

[45] Wikipedia, «https://en.wikipedia.org/,» Wikipedia, 15 Octubre 2012. [En línea]. Available: https://en.wikipedia.org/wiki/Computer_vision.

[46] G. R. W. R., «Digital Image Processing,» de *Digital Image Processing*, Edit. Prentice Hall, 2001, 2001, pp. Capítulos 9-11.

[47] B. E. Boser, I. M. Guyon y V. N. Vapnik, «A training algorithm for optimal margin classifiers,» de *Proceedings of the fifth annual workshop on Computational learning theory*, 1992.

[48] Wikipedia, «Wikipedia,» Wikipedia, 20 Septiembre 2011. [En línea]. Available: https://en.wikipedia.org/wiki/Computer_vision.

[49] Wikipedia, «https://en.wikipedia.org/wiki/Gradient,» Wikipedia, 17 Agosto 2001. [En línea]. Available: https://en.wikipedia.org/wiki/Gradient.

[50] A. G. Castrando y U. P. d. Catalunya, «Simulador de algoritmos de estimación de background con aplicaciones docentes,» 15 de enero de 2014.

Anexo 1: Instalación de librerías de OpenCV e integración en Visual Studio 2013 y en Qt Creator

Instalación de OpenCV

A continuación se detalla el procedimiento para instalar las librerías de OpenCV e integrarlas en el framework de Microsoft Visual Studio 2013.

1. Descargar las librerías de OpenCV para Windows desde su página web en la sección "Downloads" (http://opencv.org/downloads.html). Instalar las librerías en una ruta conocida, como por ejemplo: C:\opencv\.

Figura 74. Descarga de las librerías de OpenCV

2. Una vez descargadas e instaladas las librerías es necesario añadir la ruta de las librerías dinámicas a las variables de sistema pertinentes. En "Mi PC" > "Propiedades" > "Configuración avanzada del sistema" > "Avanzado" > "Variables de entorno". En la variable *Path* se añade al final tras un punto y coma las rutas de las librerías dinámicas. En este caso "C:\opencv\build\x86\vc12\bin;" en el caso de una estructura de 32 bits y "C:\opencv\build\x64\vc12\bin;" en el caso de 64.

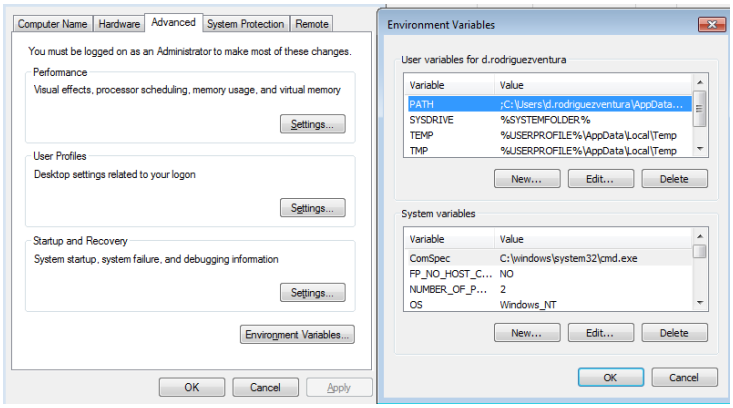


Figura 75. Ventaba de variables de entorno

Integración en Microsoft Visual Studio

3. A continuación, se inicia Microsoft Visual 2013 para iniciar la integración de las librerías en el código. Tras iniciarse, se crea un nuevo proyecto, en “Nuevo Proyecto” > “Aplicación de consola win32”. En “View” > “Other Windows” > “Property Manager” se obtiene una ventana de configuración del modo de depuración (Debug) y lanzamiento (Release).

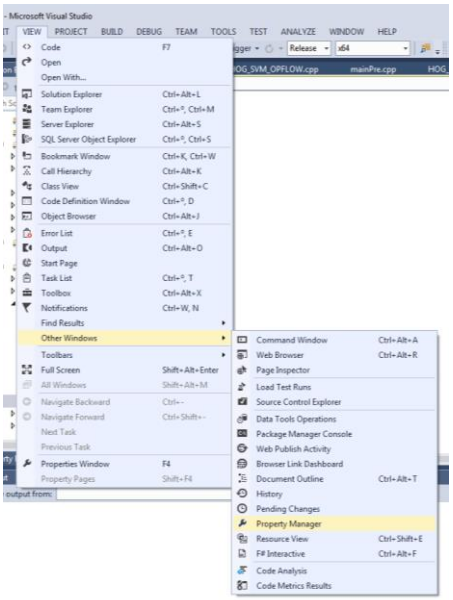


Figura 76. Búsqueda de la ventana Property Manager

4. En la ventana de “Property Manager” se pulsa con el botón derecho del ratón en “Debug | Win32” y se pulsa “Add New Project Property Sheet”. En este caso, se puede nombrar el archivo como “OPENCV_DEBUG.props”.Una vez creado el

archivo se selecciona con el botón derecho y se selecciona “Properties”. Aparecerá un menú con el aspecto de la figura 80.

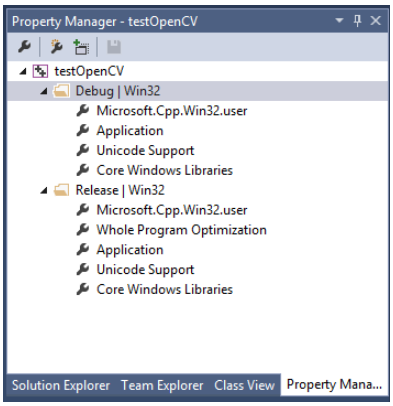


Figura 77. Ventana Property Manager

5. Se añaden los siguientes valores en los siguientes campos, sin borrar los ya existentes. En caso de que la estructura sea de 64 bits en vez de 32, solo es necesario cambiar en las rutas “x86” por “x64”.
    - VC++ Directories
      - Executable Directories: C:\opencv\build\x86\vc12\bin
      - Library Directories: C:\opencv\build\x86\vc12\lib
    - C/C++
      - Additional Include Directories: C:\opencv\build\include
    - Linker > General
      - Additional Library Directories: C:\opencv\build\x86\vc12\lib
    - Linker > Input
      - Additional Dependencies:
        opencv_core2410d.lib
        opencv_highgui2410d.lib
        opencv_imgproc2410d.lib
        opencv_video2410d.lib
        opencv_features2d2410d.lib

6. De nuevo, en la ventana de “Property Manager” se pulsa con el botón derecho del ratón en “Release | Win32” y se pulsa “Add New Project Property Sheet”. En este caso, se puede nombrar el archivo como “OPENCV_RELEASE.props”. Una vez creado el archivo se selecciona con el botón derecho y se selecciona “Properties”. Se añaden de nuevo los valores en los siguientes campos, sin borrar

los ya existentes. En caso de que la estructura sea de 64 bits en vez de 32, solo es necesario cambiar en las rutas “x86” por “x64”.

- VC++ Directories
  - Executable Directories: C:\opencv\build\x86\vc12\bin
  - Library Directories: C:\opencv\build\x86\vc12\lib
- C/C++
  - Additional Include Directories: C:\opencv\build\include
- Linker > General
  - Additional Library Directories: C:\opencv\build\x86\vc12\lib
- Linker > Input
  - Additional Dependencies:
    opencv_core2410.lib
    opencv_highgui2410.lib
    opencv_imgproc2410.lib
    opencv_video2410.lib
    opencv_features2d2410.lib

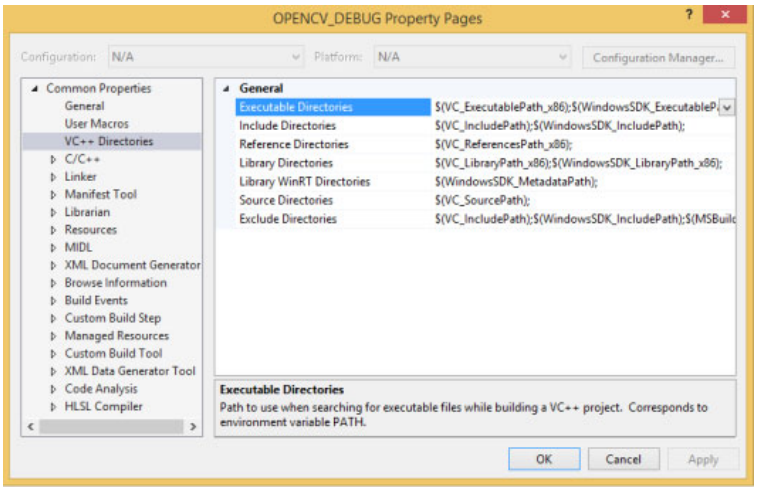


Figura 78. Aspecto de los ficheros de propiedades.

La instalación y la integración de las librerías ya está realizada y se pueden añadir al código mediante sentencias como *“#include <opencv2/core/core.hpp>”* o *“#include <opencv2/highgui/highgui.hpp>”*. Además el uso de las funciones de OpenCV puede simplificarse declarando espacio de nombres (namespace) como *cv* ya que las funciones, clases y estructuras de OpenCV responden a dicho espacio.

Integración en Qt Creator

7. Se inicia Qt Creator tras descargarlo e instalarlo. Se crea un nuevo proyecto "*Qt Widgets Application*". Se elige una localización y su nombre. Se escribe el nombre de la clase principal tal y como se desee. Se elige entre estructura de 64 o 32 bits y finalmente se pulsa "*Finish*".

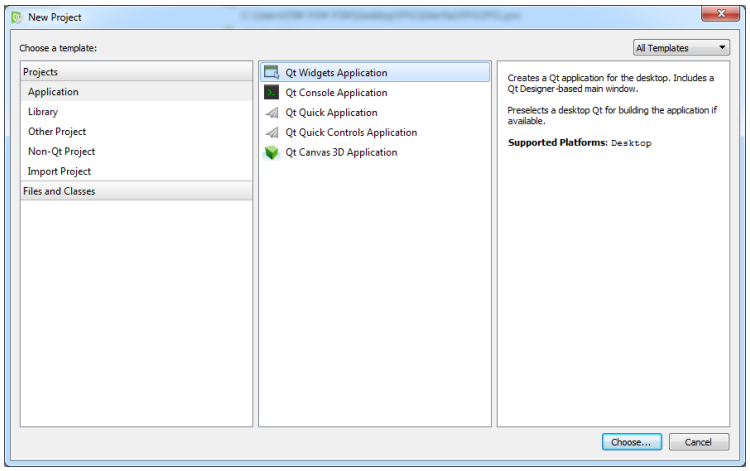


Figura 79. Menú de "New Project" en Qt Creator.

8. Se abre el archivo con extensión .pro en el visor de archivos de la izquierda. Dado que se trata del archivo de configuración del proyecto, se edita para que recoja las rutas de las librerías, archivos de cabecera y demás archivos .cpp necesarios. Para el caso que concierne, el archivo .pro quedaría de forma parecida como se muestra en la figura 81.
    - En la etiqueta "*Include Path*" se debe poner "*C:\opencv\build\include*".
    - En la etiqueta "*Libs*" se debe poner la ruta "*C:\opencv\build\x86\vc12\bin*" o "*C:\opencv\build\x64\vc12\bin*" dependiendo del tipo de estructura de 32 o 64 bits, así como añadir las librerías pertinentes:
        - opencv_core2410.lib
        - opencv_highgui2410.lib
        - opencv_imgproc2410.lib
        - opencv_video2410.lib
        - opencv_feature2d2410.lib
    - En la etiqueta "*Headers*" se debe de añadir la ruta de los archivos de cabecera .h, o bien únicamente poner su nombre y dejar el archivo en el escritorio raíz del proyecto.

- En la etiqueta "*Sources*" se debe añadir las rutas o los nombres de aquellos archivos con extensión .cpp que sean necesarios ya sea porque están incluidos dentro de la cabecera o cualquiera otra razón.

```
#-------------------------------------------------

QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = PFG
TEMPLATE = app


INCLUDEPATH += C:\\opencv\\build\\include

LIBS += -LC:\\opencv\\build\\x64\\vc12\\lib \
    -lopencv_core2410 \
    -lopencv_highgui2410 \
    -lopencv_imgproc2410 \
    -lopencv_features2d2410 \
    -lopencv_video2410 \
    -lopencv_calib3d2410

CONFIG += no_lflags_merge

SOURCES += main.cpp\
        mainwindow.cpp \
        cvcolor.cpp \
        cvcontour.cpp \
        cvlabel.cpp \
        cvblob.cpp  \
        cvtrack.cpp  \
        cvaux.cpp

HEADERS  += mainwindow.h\
            cvblob.h

FORMS    += mainwindow.ui

```

Figura 80. Aspecto del archivo .pro.

9. Tras realizar esos cambios y guardarlos, se selecciona la carpeta del proyecto con el click derecho para abrir el menú y se pulsa "*Run qmake*". Esto recargará el proyecto con la configuración contenida en el archivo .pro antes editado. De esta manera quedaría concluida la integración de las librerías.

Anexo 2: Pseudocódigo de la aplicación

```
INICIO

	INICIAR FUENTE DE IMÁGENES (VIDEO/CAMARA)

	DECLARAR E INICIALIZAR VARIABLES PARA IMÁGENES

	CREAR Y CONFIGURAR SUSTRACTOR DE BACKGROUND

	DECLARAR E INICIALIZAR VARIABLES PARA EL ANÁLISIS DE COMPONENTES

	DECLARAR E INICIALIZAR VARIABLES PARA EL TRACKING

	SI OPCION ELEGIDA IGUAL A 2 ENTONCES
		OBTENER COORDENADAS DE LA ZONA DELIMITADA
	FIN SI

	SI OPCION ELEGIDA IGUAL A 3 ENTONCES
		OBTENER COORDENADAS DE LA ZONA DE ESTACIONAMIENTO
	FIN SI

	SI OPCION ELEGIDA IGUAL A 5 ENTONCES
		OBTENER COORDENADAS DE LA ZONAS 1, 2 Y 3
	FIN SI


	MIENTRAS

		CAPTURAR FRAME

		SI FRAME VACIO O FIN DE FUENTE ENTONCES

			BREAK MIENTRAS

		FIN SI

		REDIMENSIONAR FRAME

		DESENFOCAR FRAME

		EXTRAER FOREGROUND CON SUSTRACTOR DE BACKGROUND

		DILATAR MORFOLÓGICAMENTE EL FOREGROUND

		EXTRACCION Y ETIQUETADO DE COMPONENTES DEL FOREGROUND

		FILTRAR COMPONENTES POR AREA

		//OPCION 1) DETECCION GENERAL

		SI OPCION ELEGIDA IGUAL A 1 ENTONCES

			ACTUALIZAR ESTRUCTURAS DE TRACKING

			DIBUJAR RECTANGULOS DE TRACKING

		FIN SI

		// OPCION 2) CONTROL DE VEHICULOS EN ZONA

		SI OPCION ELEGIDA IGUAL A 2 ENTONCES

			VARIABLE DETECTADO IGUAL FALSO
```

```
ACTUALIZAR ESTRUCTURAS DE TRACKING
PARA CADA COMPONENTE HALLADA
        OBTENER CENTRO DE COMPONENTE
        SI CENTRO ESTÁ DENTRO DE ZONA DELIMITADA
                AÑADIR COMPONENTE A VARIABLE 'BLOBS'
                VARIABLE DETECTADO IGUAL A VERDADERO
        FIN SI
FIN PARA
PARA CADA COMPONENTE EN 'BLOBS'
        OBTENER TRACK ASOCIADO A COMPONENTE
        DIBUJAR RECTANGULO ASOCIADO A TRACK
FIN PARA
SI DETECTADO IGUAL A VERDADERO
        DIBUJAR ZONA CON COLOR VERDE
        CAMBIAR MENSAJE A "DETECTADO"
SINO
        DIBUJAR ZONA CON COLOR ROJO
        CAMBIAR MENSAJE A "NO DETECTADO"
FIN SI
FIN SI
//OPCION 3) CONTROL DE ESTACIONAMIENTO
SI OPCION ELEGIDA IGUAL A 3 ENTONCES
        ACTUALIZAR ESCTRUCTURA DE TRACKS
        OBTENER ZONA DE APARCAMIENTO
        SI EL COCHE VA A APARCAR ENTONCES
                SI EL COCHE AUN NO HA APARCADO ENTONCES
                        PARA CADA TRACK HACER
                                OBTENER CENTRO DE CADA TRACK
                                SI CENTRO ESTÁ DENTRO DE ZONA
                                        COCHE ESTA APARCANDO
                                SINO
                                        ZONA VACIA
```

```
                                FIN SI
                        FIN PARA
                FIN SI
                SI EL COCHE ESTÁ APARCANDO ENTONCES
                        SE OBTIENE EL CENTRO DE SU TRACK
                        SI CENTRO ES DISTINTO DE NULL ENTONCES
                                COCHE SIGUE APARCANDO
                        SINO
                                COCHE YA APARCÓ
                                ZONA OCUPADA
                                COMIENZA CONTADOR DE TIEMPO
                        FIN SI
                FIN SI
        FIN SI
        SI EL COCHE YA APARCÓ ENTONCES
                SI EL COCHE SE MANTIENE QUIETO ENTONCES
                        SI CONTADOR TIEMPO MENOR QUE 30 SEGUNDOS
                                SEGUIR CONTANDO
                        SINO
                                SE BUSCAN TRACKS EN LA ZONA
                                SI HAY TRACKS ENTONCES
                                        COCHE SALIENDO
                                        DIBUJAR RECTANGULO DE TRACKS
                                FIN SI
                        FIN SI
                FIN SI
                SI COCHE SALIENDO DEL APARCAMIENTO ENTONCES
                        OBTENER CENTRO DE TRACK
                        SI CENTRO DE TRACK DENTRO DE ZONA
                                COCHE AÚN SALIENDO DE LA ZONA
                        SI NO
                                COCHE SALIÓ DE LA ZONA
```

```
                                        ZONA VACIA
                                FIN SI
                        FIN SI
                FIN SI
                SI COCHE ENTRANDO O SALIENDO DE LA ZONA ENTONCES
                        DIBUJAR ZONA AZUL
                        CAMBIAR MENSAJE A 'APARCANDO'
                FIN SI
                SI ZONA VACIA ENTONCES
                        DIBUJAR ZONA ROJA
                        CAMBIAR MENSAJE A 'VACIO'
                SINO
                        DIBUJAR ZONA VERDE
                        CAMBIAR MENSAJE A 'APARCADO'
                FIN SI
        FIN SI
        //OPCION 4) CONTROL DE VELOCIDAD
        SI OPCION ELEGIDA IGUAL A 4 ENTONCES
                INICIAR MAPA DE VELOCIDADES
                ACTUALIZAR ESTRUCTURA DE TRACKS_ACTUALES
                SI MAPA DE TRACKS_ANTERIORES ESTA VACIO ENTONCES
                        RELLENAR MAPA DE TRACKS_ANTERIORES
                SI NO
                        PARA CADA TRACK DE TRACKS_ANTERIORES HACER
                                BUSCAR TRACK EN TRACKS_ACTUALES
                                SI ENCUENTRA
                                        CALCULAR DISTANCIA ENTRE CENTROS
                                        HALLAR VELOCIDAD
                                        AÑADIR VELOCIDAD A MAPA VELOCIDADES
                                FIN SI
                        FIN PARA
                        DIBUJAR RECTANGULOS E ID ASOCIADOS A CADA TRACK
```

```
                MOSTRAR VELOCIDAD DE CADA ID EN PANTALLA
                BORRAR MAPA TRACKS_ANTERIORES
                BORRAR MAPA DE VELOCIDADES
                RELLENAR TRACKS_ANTERIORES CON TRACKS_ACTUALES
        FIN SI
FIN SI

//OPCION 5) CONTROL ENTRADA/SALIDA CON CONTEO
SI OPCION ELEGIDA IGUAL A 5 ENTONCES
        ACTUALIZAR ESTRUCTURA DE TRACKS
        PARA CADA TRACK DE LA ESTRUCTURA DE TRACKS
                OBTENER CENTRO DE TRACK
                SI CENTRO DE TRACK ESTÁ EN ZONA 1 ENTONCES
                        BUSCAR TRACK EN TRACKS_ZONA1
                        SI NO EXISTE ENTONCES
                                BUSCAR TRACK EN TRACKS_ZONA2
                                SI EXISTE ENTONCES
                                        NOTIFICAR SALIDA
                                        DISMINUIR CONTADOR
                                        BORRAR TRACK DE TRACKS_ZONA2
                                SINO
                                     INSERTAR TRACK EN TRACKS_ZONA1
                                FIN SI
                        FIN SI
                FIN SI
                SI CENTRO DE TRACK ESTÁ EN ZONA 2 ENTONCES
                        BUSCAR TRACK EN TRACKS_ZONA2
                        SI NO EXISTE ENTONCES
                                BUSCAR TRACK EN TRACKS_ZONA1
                                SI EXISTE ENTONCES
                                        NOTIFICAR ENTRADA
                                        AUMENTAR CONTADOR
```

```
                                                BORRAR TRACK DE TRACKS_ZONA1
                                            SINO
                                                 INSERTAR TRACK EN TRACKS_ZONA2
                                            FIN SI
                                        FIN SI
                                    FIN SI
                                FIN PARA
                            FIN SI
        MOSTRAR IMAGEN
        FIN MIENTRAS
        LIMPIAR VARIABLES
        CERRAR PROCESOS
FIN
```