INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Escuela superior de Ingeniería Informática

Curso Académico 2014/2015

Proyecto Fin de Carrera

Seguimiento visual de vehículos en escenarios de baja visibilidad

Autor: Daniel Gómez Gómez

Tutor: Jose María Cañas Plaza

Agradecimientos

En primer lugar, agradecer a mi familia todo el apoyo y cariño recibido no solo durante mi formación, sino durante toda mi vida. Por animarme siempre a estudiar lo que me gustara y a luchar por conseguir todo lo que me proponga.

Gracias a Helena por todo el apoyo que me ha brindado cuando más lo necesitaba. Por su cariño, por su ternura y por su capacidad de hacerme olvidar los problemas. Porque con ella todo es más fácil.

También quisiera agradecer a mi tutor Jose María toda la infinita paciencia que ha demostrado conmigo, por su esfuerzo y por el conocimiento que me ha aportado en el área de la Visión Computacional.

Tampoco puedo olvidar agradecer la paciencia que han tenido mis amigos por estar tan desaparecido últimamente y todo el apoyo recibido por ellos todo este tiempo.

Resumen

La Visión Computacional es un área lleno de potencial, en el que por ejemplo un programa es capaz de reconocer situaciones concretas en un vídeo o en una fotografía. Estas herramientas son cada vez más utilizadas en diferentes ámbitos, como por ejemplo en la monitorización de tráfico rodado.

El objetivo de este Proyecto Fin de Carrera es desarrollar una aplicación que detecte y contabilice vehículos mediante el procesamiento de las imágenes de vídeos de tráfico grabados por una única cámara en entornos de visibilidad reducida.

La aplicación consta de varias fases de procesamiento de imágenes: aprendizaje de fondo, detección, seguimiento y conteo. La detección sirve para que el algoritmo tenga una referencia de los elementos de interés existentes en un fotograma. Para efectúar el conteo de vehículos, es necesario realizar un seguimiento a los elementos de interés, detectados previamente, a través de los sucesivos fotogramas del vídeo. El seguimiento consiste en identificar y relacionar, mediante diversos criterios, vehículos durante varios fotogramas. Se han estudiado e implementado diferentes técnicas para estas fases de procesamiento visual.

Para validar empíricamente la aplicación, se ha recopilado un conjunto de vídeos de tráfico rodado en condiciones de visibilidad reducida, ya sea por estar grabados en baja resolución, por la proliferación de sombras, por estar situada la cámara en una posición que favorece las apariciones de oclusiones parciales o por ser nocturnos.

En la realización de experimentos con estas imágenes se obtiene una alta tasa de aciertos en la mayoría de diferentes escenarios, respecto al número de vehículos observados y los que el programa contabiliza. Además se logra en la mayoría de los casos un coste computacional razonable.

Índice general

Capítulo 1

Introducción

Desde la aparición de la computación se ha buscado la automatización de procesos tediosos, largos y con altas probabilidades de caer en el error para un humano. Desde la gestión automática de datos y su almacenamiento, hasta cálculos matemáticos de todo tipo. En el vasto área del procesamiento de datos, es realmente interesante el análisis del entorno que nos rodea mediante recogida de información de diversos tipos. Un sistema puede reaccionar de una forma u otra dependiendo del estímulo que le llegue a través de sensores, con una mínima o nula intervención del usuario en el proceso.

La emulación de la percepción humana abre todo un abanico de posibilidades en la creación de programas y sistemas informáticos. Lograr que un algoritmo, por ejemplo, reconozca formas, contornos o colores, y pueda señalarlos, da pie a realizar cosas cada vez más complejas, como la identificación de vehículos concretos o el reconocimiento de situaciones en una imagen o un vídeo. El número de aplicaciones que se puede dar a esto es enorme.

Este Proyecto Fin de Carrera se centra en la Visión Computacional y su aplicación en la monitorización de tráfico, en concreto, en el conteo y seguimiento de vehículos que pasan por una sección de carretera en unas condiciones de visibilidad reducida, ya sea escasa iluminación o ángulos de cámara donde se den más oclusiones de lo normal.

1.1. Visión artificial

La vista es, sin duda, el sentido que más información del entorno nos aporta. El cerebro procesa e interpreta las imágenes que el ojo recoge, pudiendo reaccionar en caso de que así se requiera. La disciplina que investiga y desarrolla software cuyo objetivo es la extracción y análisis de información de una imagen se denomina “Visión Artificial”. Ya que ayuda a

un sistema a la toma de decisiones y debe ser capaz de adaptarse a los cambios en una imagen, se clasifica como un subcampo de la Inteligencia Artificial.

El inicio de la Visión Artificial lo marca la presentación de un estudio cuyo autor es Larry Roberts, uno de los ideólogos de ARPANET, el sistema de comunicación precursor de Internet. El estudio de Roberts [17], presentado en 1963, muestra cómo un programa, a partir de una fotografía de una figura, reconstruye ésta en diferentes perspectivas. Como ilustra la imagen 1.1, el programa detecta los bordes de la figura y, construyendo un modelo 3D, lo manipula para poder mostrarla desde distintos ángulos.

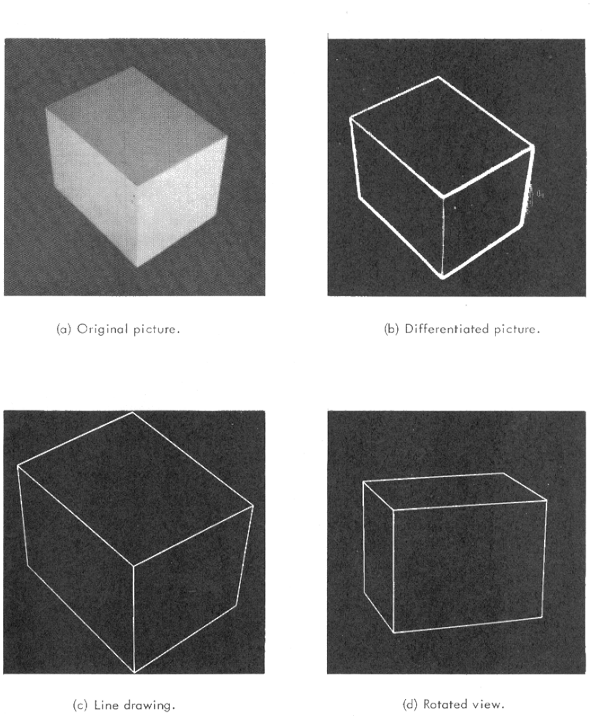


Figura 1.1: Muestra del funcionamiento del programa de Larry Roberts

Más tarde, con la mejora de las cámaras a lo largo del tiempo se pudo incorporar al ordenador. Junto con el aumento de la potencia de los procesadores, posibilitó diseñar algoritmos más potentes y usarlos en tiempo real en imágenes y vídeos. Éstos últimos tienen una dificultad añadida: el sistema que se desarrolle tiene que dar continuidad y coherencia a los datos que se obtengan entre los distintos frames de un vídeo.

En la Visión Artificial se utilizan técnicas tales como reconocimiento de patrones, geometría de proyección y procesamiento de imágenes, entre otros.

Las posibilidades que tiene este campo son inmensas. Las diversas aplicaciones que tiene en la vida cotidiana lo demuestran. A continuación mostramos algunos de los ámbitos más comunes donde se usa la Visión Artificial:

- **OCR:** El Reconocimiento Óptico de Caracteres (u *Optical Character Recognition*, OCR, en inglés) es capaz de detectar cifras y letras en fotos y vídeo y digitalizarlos, pudiendo hacer corresponder cada símbolo en la imagen en su equivalente en ASCII. Ejemplos de utilización de este sistema está en el lector automático de matrículas que hay en la mayoría de los aparcamientos de pago o en las aplicaciones de escáner portátil para *smartphones* donde se utiliza este sistema para digitalizar textos y documentos a través de las cámaras que suelen venir integradas en estos dispositivos, como el de la figura 1.2.

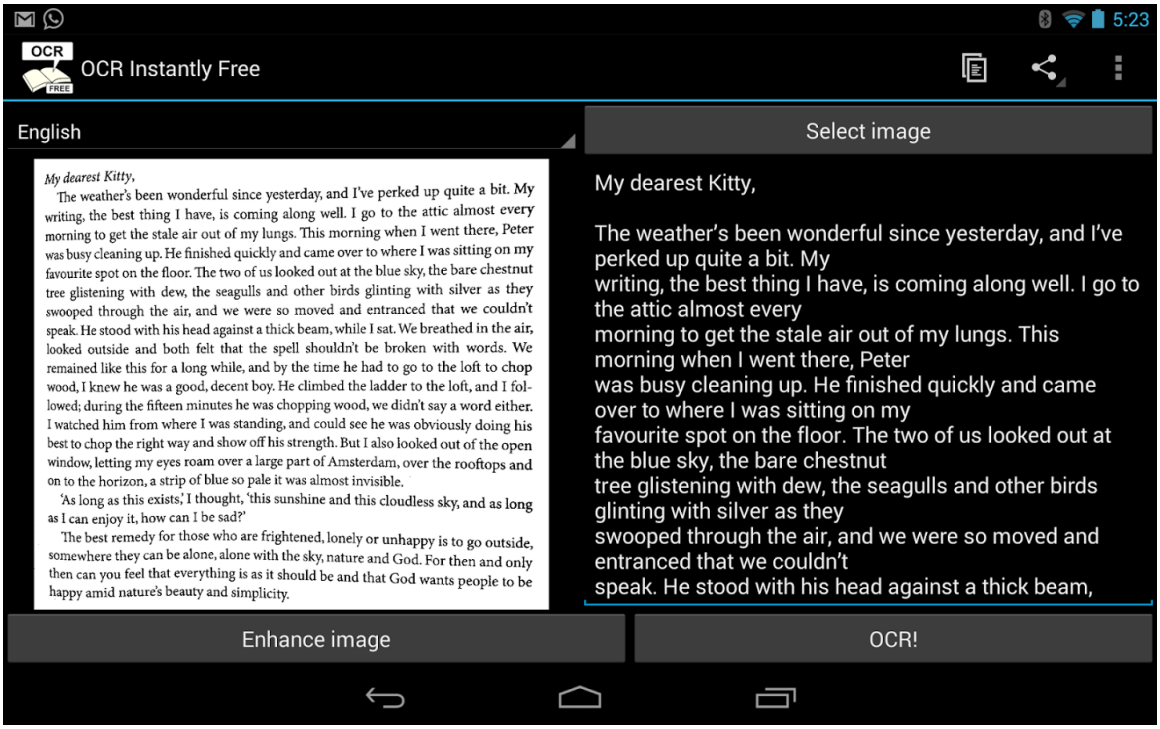


Figura 1.2: Una de las aplicaciones de Android que se basan en el sistema OCR

- **Código QR:** El código de Respuesta Rápida (o *Quick Responde code*, en inglés) es un sistema de codificación en el que se puede albergar información de texto plano de diversa índole. Usualmente, el código se presenta como la imagen de una matriz de puntos bidimensional, como el de la figura 1.3. Para descodificarlo, un programa lector de QR detecta el código a través de la cámara cuando se enfoca al símbolo y puede actuar de diversa forma, dependiendo de la información que albergue el QR. Por ejemplo, si es una dirección de Internet, el programa abrirá el navegador para dirigirse a ella.

  Lo ideó en 1994 la compañía japonesa, Denso Wave, para poder registrar más fácilmente inventario en un almacén. En la actualidad se utiliza sobre todo en publicidad, tarjetas de visita y gestión de inventarios. Este sistema facilita el almacenaje de información en dispositivos móviles y el sistema de decodificación es en tiempo real y casi instantáneo, por ello ha tenido tan buena acogida y gran expansión entre la sociedad de consumo.

Figura 1.3: Apariencia de un código QR

- **Biometría:** Existen una serie de características físicas únicas en cada ser humano que pueden ser aprovechadas en el ámbito de la seguridad, como sustitución o refuerzo de las contraseñas. Las huellas dactilares o la morfología de la voz son dos buenos ejemplos de ello.

  Centrándonos en las lecturas biométricas que utilizan Visión Artificial, la más destacable es el reconocimiento facial. Existen diversas técnicas para analizar las facciones de un rostro y digitalizarlos de tal manera que se puedan comparar rasgos para discernir si pertenecen a una persona concreta o no. Dichas técnicas van desde localizar patrones geométricos en un rostro, hasta realizar proyecciones geométricas de una cara, creando un mapa facial a partir del procesamiento de sus facciones, pudiendo utilizarlo como un identificador único para un individuo (Figura 1.4). La mayor

desventaja del reconocimiento facial es que, por el momento, tiende a fallar si existe alguna variación del ángulo en el que se muestra un rostro y si hay cambios en la iluminación en ese momento. También puede fallar si el rostro sufre cambios físicos, como la aparición o desaparición de barba, llevar gafas, etc...

Figura 1.4: Mapa facial creado a partir de detección de patrones

Otro apartado digno de mención dentro de la biometría usando Visión Artificial es el reconocimiento de iris. Durante estos análisis, se captura una imagen del iris usando una cámara de alta resolución en un entorno bien iluminado. La imagen pasa por una etapa de preprocesado, para obtener una imagen más nítida. Después, se extrae las zonas de interés para el análisis del anillo del iris, exclusivamente, de la manera en la que se muestra en la figura 1.5. Tras obtener la imagen con el área que interesa para su análisis, se vuelve a procesar para mejorar la nitidez de la imagen y acentúar las características del iris. La siguiente etapa analiza el iris en sí, construyendo una plantilla por proyección de los datos obtenidos. Y por último se compara la plantilla obtenida con la que debería existir previamente, y juzgar si el iris analizado pertenece a una persona en concreto. Como principal desventaja de este sistema, es que para realizar bien el reconocimiento, la distancia de la cámara no puede superar los dos metros. Además la persona a identificar tiene que estar unos momentos mirando a la cámara y con la cabeza quieta, con lo que debe cooperar para que el análisis del iris sea correcto.

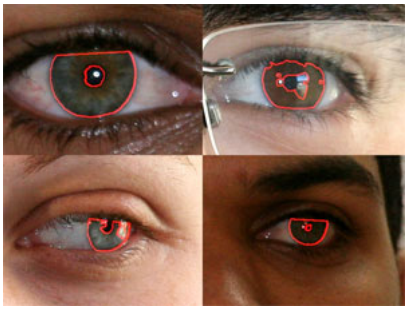


Figura 1.5: Detección de áreas de interés del iris

- **Realidad aumentada:** Son una serie de técnicas en las que, a través de una cámara, se analiza el medio en el que uno se encuentra, se procesa y se añaden elementos virtuales en tiempo real, tanto en dos como en tres dimensiones. Las aplicaciones basadas en Realidad Aumentada empiezan a despuntar en varios sectores. Por ejemplo, en el campo de los videojuegos tenemos *Table Top Tanks* para la consola portátil *PS Vita*. El juego utiliza tarjetas especiales como puntos de referencia para construir el escenario en un entorno 3D, integrándose con el mundo real. Como puede apreciarse en la figura 1.6, el juego también detecta elementos reales con los que se pueden interactuar dentro de él.

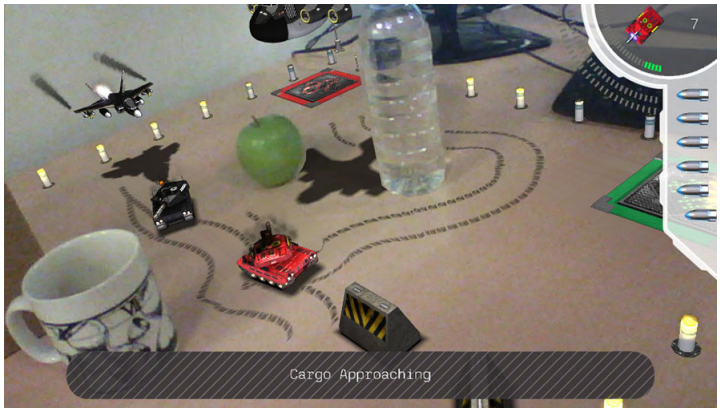


Figura 1.6: Apariencia del juego *Table Top Tanks*

En el sector turístico, existen aplicaciones para dispositivos portátiles en el que el programa puede reconocer monumentos o edificios singulares de una ciudad en tiempo real, ayudándose de coordenadas GPS, y mostrar información relevante del lugar en el momento. Un ejemplo es la aplicación *Guía Madrid 5D* lanzada por el Ayuntamiento de Madrid, donde se utilizan también métodos de Realidad Aumentada para dar información de interés histórico y turístico, tal y como muestra la figura 1.7.

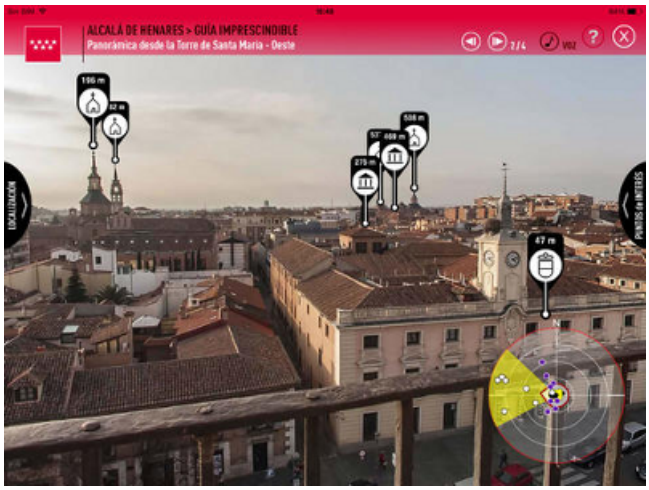


Figura 1.7: Funcionamiento de *Guía Madrid 5D* en Alcalá de Henares

En el mundo de la publicidad, la Realidad Aumentada es cada vez más utilizada en las campañas de publicidad y promociones de productos. De esta manera se llama más la atención sobre el público y además se logra que éste interactúe de forma activa con el anuncio. En las imágenes contenidas en la figura 1.8 se pueden observar un par de ejemplos.

(a) Muestra tridimensional de un modelo de coche

(b) Mujer probándose virtualmente un zapato

Figura 1.8: Aplicaciones de Realidad Aumentada en publicidad

Por último, en el ámbito educativo también está entrando tímidamente esta técnica. En algunas aulas ya se trabaja con aplicaciones basadas en Realidad Aumentada para reforzar los conocimientos del alumno, mostrando modelos en 3D cuando se enfoca con la cámara de un dispositivo a tarjetas especiales, por ejemplo (Figura 1.9).

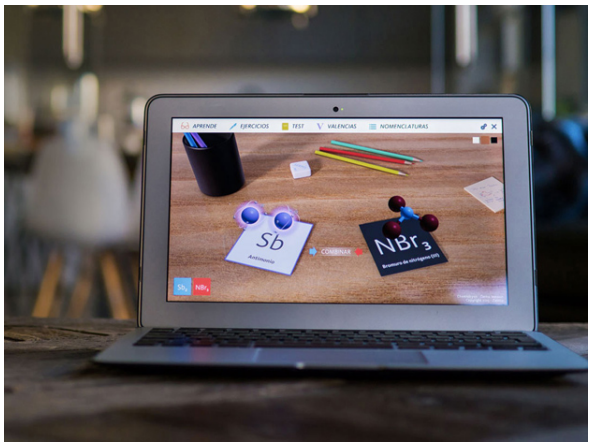


Figura 1.9: Representación tridimensional de moléculas en una clase

Poco a poco, empresas apuestan cada vez más fuerte por esta técnica. Por ejemplo, *Google* lanzó *Google Glass*, unas gafas con una cámara incorporada que pueden analizar el entorno y presentar información en tiempo real respecto a donde señale la cámara. Este dispositivo, mostrado en la figura 1.10, puede servir en todos los distintos ámbitos anteriormente explicados.

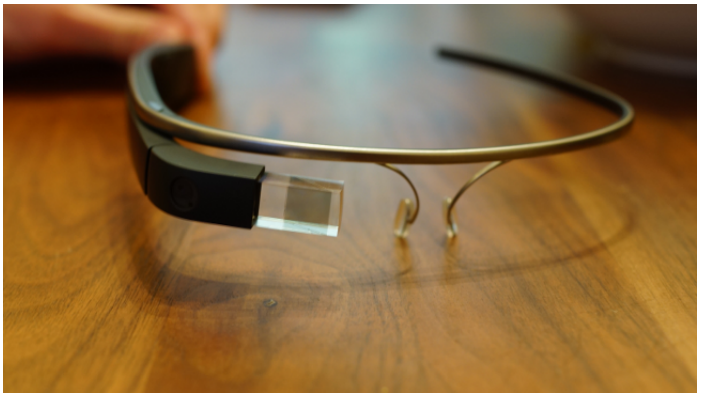


Figura 1.10: Imagen de Google Glass

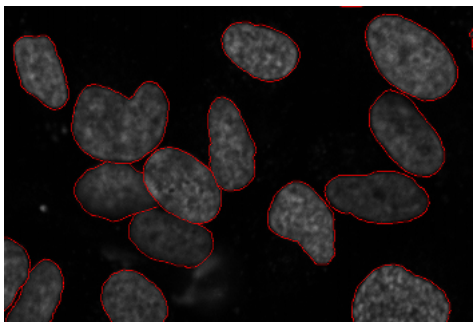
Existen más aplicaciones de la Visión Artificial, como en el control de calidad del sector industrial, para revisar que cierto producto tiene las formas y las medidas exactas. O en la industria del envasado y el embotellado, donde se revisa mediante la Visión Artificial y de manera automática, la correcta colocación del producto en todos los envases de manera además idéntica, permitiendo de este modo aumentar la rapidez de esta actividad y, por tanto, la productividad (Figura 1.12a).

En el campo de la Investigación también existen sistemas de apoyo basados en la Visión Artificial. En la Robótica, sin ir más lejos. El sistema de Visión Artificial de los sistemas robóticos que utilizan cámaras de vídeo como receptor de estímulos del exterior es esencial para que el robot pueda actuar en cada caso. Un ejemplo es la programación de drones, aportándoles autonomía en sus decisiones para ir entre dos puntos sin chocar con ningún obstáculo. En la figura 1.11 podemos ver una muestra del funcionamiento del análisis del entorno del sistema de navegación de uno de estos drones.

Figura 1.11: Sistema de navegación de un drone autónomo

Existen muchos más campos donde las técnicas de la Visión Artificial son realmente esenciales. Esto sólo es una muestra de lo integradas que están ya estas técnicas en nuestro día a día. Un futuro muy interesante aguarda mientras se van desarrollando cada vez mejores sistemas basados en la Visión Artificial.

(a) Control de calidad del envasado de medicamentos



(b) Detección visual automática de células al microscopio

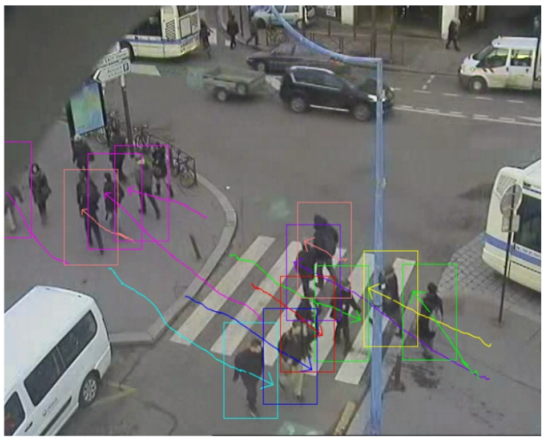
Figura 1.12: Existe un sinfín de aplicaciones para la Visión Artificial

1.2. Seguimiento visual de objetos

Una vez que tenemos el sistema para detectar objetos de interés en una imagen, podemos usar estas mismas técnicas en un vídeo. Éste se compone de un conjunto de imágenes, ordenadas de tal manera, que al mirar sucesivamente 24 de estas imágenes en un segundo cada vez, se transmite la sensación de movimiento. Así, existe una correlación entre sí en este conjunto de imágenes, aportando coherencia.

Cuando se quieren detectar elementos de interés en un vídeo, suele ser necesario hacer un seguimiento de estos objetos a lo largo de todos los fotogramas. El seguimiento visual de objetos persigue este objetivo: desarrollar técnicas para identificar objetos de interés en un vídeo y relacionarlos entre sus distintos *frames*, dándoles una identidad única.

El sector de la seguridad es el que más aprovecha las ventajas que brinda el seguimiento visual de objetos. En la videovigilancia, por ejemplo, donde una persona tiene que estar pendiente de las cámaras por si ocurre algún problema. Con el seguimiento visual de objetos, se puede lograr que un programa informático haga un seguimiento de las personas que capte la cámara y hasta tener la capacidad de detectar ciertas situaciones y avisar de ellas, todo esto en tiempo real. Por ejemplo, como en las aplicaciones mostradas en la figura 1.13, puede registrarse la densidad de gente que transita una calle a cierta hora del día, extraer el número de gente que pasea en cierta zona de un centro comercial o detectar en tiempo real si una persona mayor se cae al suelo, como en el proyecto *Eldercare* del grupo de robótica de la URJC[**?**].



(a) Seguimiento de personas, dibujando su recorrido

(b) Programa *ElderCare* detectando una persona caída en el suelo

Figura 1.13: Aplicaciones en la vida real utilizando Seguimiento Visual de Objetos

1.2.1. Seguimiento visual de objetos en el control del tráfico

Estas técnicas se utilizan también en la monitorización del tráfico en las carreteras. Aprovechando que existen cámaras instaladas en distintos puntos de las principales carreteras, se pueden desarrollar sistemas basados en el seguimiento visual de objetos para detectar automáticamente situaciones de riesgo para el tráfico, o recoger datos estadísticos sobre el flujo de vehículos en un tramo de una vía concreta a distintas horas del día, por poner dos ejemplos.

Distintos países están empezando a adoptar sistemas automatizados en sus carreteras. Buscan así aumentar la eficacia en la gestión del tráfico y minimizar lo máximo posible el tiempo de respuesta ante situaciones de riesgo para la integridad o la vida de los conductores. En Estados Unidos, por ejemplo, existe la ITS (*Intelligent Transportation System*)[1], una iniciativa del Departamento Federal de Transportes, que apoya y recoge a nivel nacional todos los proyectos de desarrollo tecnológico de sistemas de ayuda al control del tráfico.

Figura 1.14: Dispositivo basado en radar que recoge diversa información del tráfico en tiempo real

No obstante, el seguimiento visual de objetos no son los únicos datos que se recogen en un sistema de estas características. La Visión Artificial tan solo complementa a un conjunto de información, obtenida de diversas fuentes. Como muestra de lo que se expone es el sistema *SunGuide*[12], implantado en el Estado de Florida para el apoyo de la tarea de los operadores de control de tráfico. Éste se vale también, entre otras, de tecnología radar para la detección de vehículos y calcular en tiempo real la velocidad y sentido del mismo, con la capacidad de realizarlo en varios carriles, procesando cada uno por separado (Figura 1.14). En *SunGuide*, el seguimiento visual de objetos se utiliza concretamente para detectar vehículos parados en la imagen proporcionada por cámaras de tráfico. De esta

[1]Web oficial de la ITS: `http://www.its.dot.gov/`

forma, un operador no tiene que estar pendiente de tantas pantallas, disminuyendo la posibilidad de que no se vea a tiempo un hecho de gravedad ocurrido en alguna carretera. El sistema, tras detectar un coche parado, emite un aviso para que el operador lo revise. Por ello es importante también minimizar el número de falsas alarmas cuando se utiliza una funcionalidad basada en este área de la Visión Artificial.

Siempre pueden haber falsos positivos en sistemas basados en seguimiento visual de objetos, como muestra los de la figura 1.15. Por ello, en sistemas complejos, es buena idea tener más dispositivos recogiendo datos del entorno. Tanto los sensores fijos como los montados en otros vehículos, junto con la información transmitida por el cuerpo de seguridad, aportan más datos que permite confirmar, o rechazar, la existencia de un hecho relevante en una carretera. Aunque la inversión que hay que realizar para la adquisición y montaje de un mayor número de sensores es obviamente más alta.

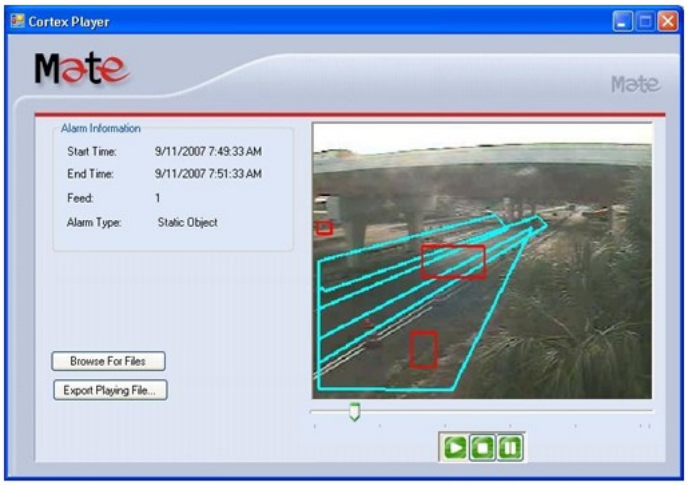


Figura 1.15: Falsas alarmas en una aplicación detectora de situaciones peligrosas

En España, el procesamiento de imágenes de vídeo para apoyo de sistemas automatizados en control de tráfico aún está poco extendido, limitado a tareas muy concretas pero útiles. La detección automática en vídeo de vehículos saltándose semáforos en rojo es una de ellas[3], como el que se puede ver en la figura 1.16. Al utilizarse una cámara inteligente, se puede aprovechar su uso para analizar y extraer otros datos, como la densidad del tráfico o la velocidad media estimada en esa vía. La cámara instalada suele tener incorporada iluminación para escenas nocturnas o momentos con poca luz, un procesador para obtener resultados en tiempo real, y soporte a sistemas de red de comunicación, para poder enviar estos datos.

Figura 1.16: Sistema de control de semáforo en rojo en la ciudad de Madrid

Existen sistemas de Visión Artificial que no son fijos, como hemos visto anteriormente. También existen sistemas móviles montados en vehículos. El sistema BusVigía, desarrollado y patentado por la URJC[7], incorpora una cámara de vídeo en los propios autobuses, como se ve en la imagen de la figura 1.17, y detectan en tiempo real posibles infracciones de vehículos dentro del carril bus. La cámara registra la matrícula de los infractores en diversas condiciones climáticas y lumínicas.

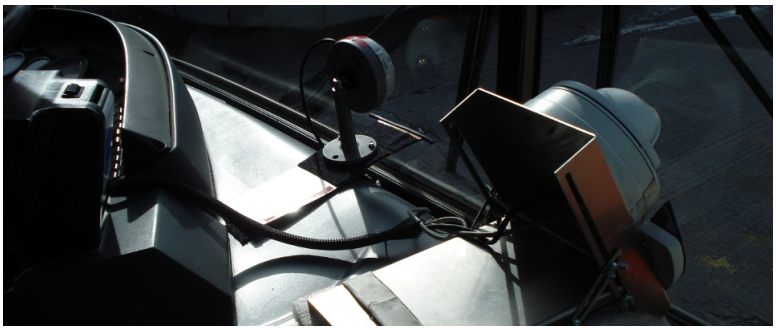


Figura 1.17: Cámara en un autobús utilizando el sistema BusVigía

Estas aplicaciones orientadas al control del tráfico tienen mucha proyección en el futuro, ya que cada vez se usan más y más. Y están presentes en los grandes proyectos urbanos del futuro, como el de CIUDAD2020[2]. Este proyecto de I+D pretende la modernización de las ciudades, permitiendo al ciudadano acceder en tiempo real a la mayor cantidad de información posible sobre los servicios públicos y mejorando tecnológicamente el transporte y sus infraestructuras, respetando a su vez al medio ambiente. CIUDAD2020 es un proyecto liderado por Indra y donde colaboran empresas privadas y organismos de investigación tales como la Universidad Politécnica de Madrid, la Universidad de Alcalá de Henares y la Universidad Carlos III. El Centro para el Desarrollo Tecnológico Industrial, perteneciente al Ministerio de Economía y Competitividad de España, cofinancia esta iniciativa.

[2]Web del proyecto CIUDAD2020: `http://www.innprontaciudad2020.es/`

Algunas de las aplicaciones pensadas para CIUDAD2020 están basadas en el procesamiento de imágenes de vídeo. Por ejemplo, están desarrollando un programa para el control del tráfico en intersecciones apoyado en la Visión Artificial[19]. Se está empezando a probar el software en el vídeo que proporcionan cada una de diversas cámaras de tráfico que ya hay instaladas. Hay que tener en cuenta que cada cámara tiene su propio ángulo de inclinación y distancia, sumado con el problema de la iluminación y las condiciones metereológicas, hace difícil calibrar la imagen que proporciona el vídeo para empezar a procesar las imágenes. Por ello, dentro de este programa, se está desarrollando un sistema de autocalibración, usando elementos artificiales que aparecen en la imagen como referencia (señales, semáforos, pasos de peatones, edificios...).

Una vez calibrada la imagen, el *software*, tal y como se ve en la figura 1.18, puede detectar vehículos, peatones y ciclistas, con los índices del flujo de tráfico de cada uno. De este modo, se pueden recoger datos estadísticos sobre en qué entradas y salidas de una intersección hay más densidad de tráfico y detectar incidentes. Esta información podrán usarla los sistemas de control de tráfico, que decidirán el tiempo de apertura y cierre de los semáforos o avisar automáticamente de algún incidente producido.

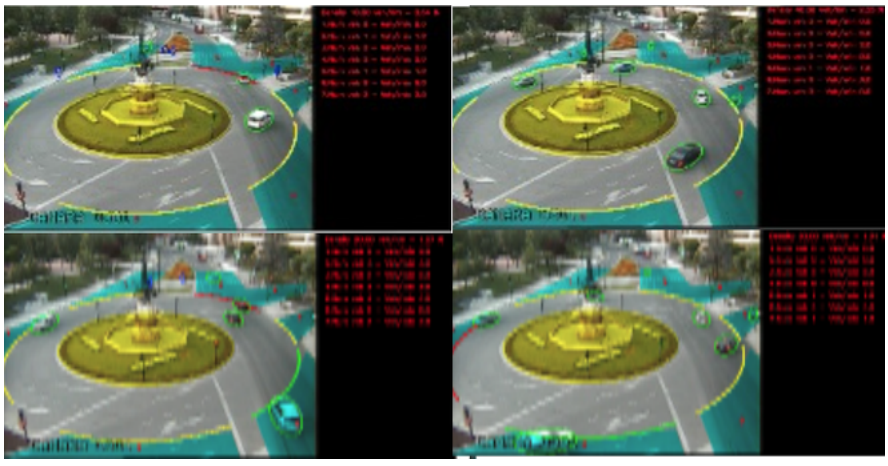


Figura 1.18: Programa de monitorización de flujo de tráfico probado en una rotonda

En conclusión, la tecnología de seguimiento visual es tremendamente útil como sistema de apoyo a la gestión del tráfico. Este sistema sigue mejorando cada vez más, pudiendo reducir notablemente la intervención humana en la supervisión de las cámaras de tráfico en busca de algún incidente. Esta tecnología permitirá reducir el tiempo de respuesta ante incidentes en carretera y a mejorar la eficacia en el control del flujo del tráfico.

1.3. Seguimiento visual de objetos en el Grupo de Robótica de la URJC

En el Grupo de Robótica de la URJC, existe una serie de trabajos previos que estudian temas relacionados con el seguimiento visual de objetos, donde los algoritmos que se utilizan han ayudado al aprendizaje de técnicas, las cuales han sido de ayuda para desarrollar el sistema.

1.3.1. Carspeed

En el Proyecto Fin de Carrera de Víctor Hidalgo[25] se utilizan diversos algoritmos clásicos para la fase de detección y seguimiento de los vehículos que aparecen en vídeo. Este sistema utiliza un algoritmo evolutivo, el cual propone varios elementos candidatos de la imagen, uno por cada velocidad hipotética calculada, para decidir cuál de ellos será el coche en la siguiente iteración. Para ello, les va asignando una medida de calidad y, basándose en ella, criba los resultados hasta encontrar el elemento más prometedor.

Para obtener esta medida de calidad en cada elemento, se implementa una función `Salud`. En ella se sopesa el tamaño del vehículo en cada iteración para obtener una estimación de la velocidad del vehículo, que será de utilidad posteriormente para realizar cada seguimiento. Para poder discernir dichas medidas, esta función se apoya en el uso de la imagen diferencias entre el fotograma actual y la imagen de fondo aprendida, calculando la diferencia de tamaño entre el fotograma actual y el previo.

El proceso de `Carspeed` se compone de las siguientes fases: rectificación de la imagen, detección de nuevos vehículos y seguimiento, pudiendo solaparse estas dos etapas en algunos casos.

Para la etapa de detección de vehículos, existe un área preestablecida de la imagen donde se utiliza también el sistema de imagen diferencia para detectar movimiento (Ver figura 1.19). Se ha decidido hacerlo de esta manera para hacer llevadero el coste computacional, ya que hay que aplicar la función `Salud` a todo píxel de la imagen donde se detecte movimiento. De esta forma se comienzan a elegir los elementos de interés.

La etapa de seguimiento consiste en actualizar la posición de cada elemento, luego recalcular con la función `Salud` la velocidad estimada y hacer una criba de los elementos en la imagen cuyo valor de `Salud` no alcance un valor mínimo. Al final de ésta, el elemento

con el mejor valor de `Salud` es la que se selecciona. Típicamente solo queda una de entre todas las velocidades hipotetizadas.

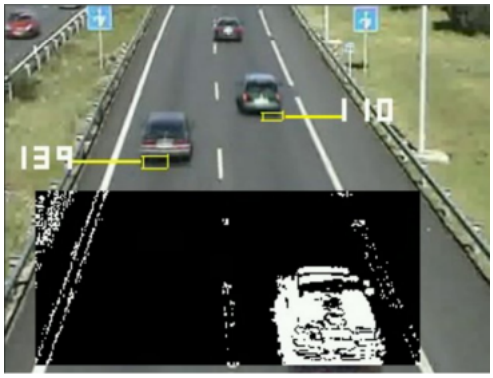


Figura 1.19: Detección de vehículos y estimación de velocidad en `Carspeed`

Disponer de una estimación de la velocidad de un vehículo y ser capaces de estimar la posición de éste en cada momento usando estos datos es una gran ventaja, ya que se pueden descartar rápidamente falsos positivos durante la etapa de seguimiento. Utilizando este sistema, el seguimiento de los vehículos parece ser bastante consistente.

Por contra, como principales inconvenientes, `Carspeed` no funciona bien en vídeos con condiciones pobres de visibilidad (con lluvia o vídeos nocturnos) y confunde a vehículos largos, como autobuses o camiones grandes, como si fueran más de uno.

1.3.2. Seguimiento de objetos con PixelTrack

En este Trabajo Fin de Máster[10] se explican diversos métodos clásicos de seguimiento de objetos, a modo introductorio, y un método que es la evolución natural de estos algoritmos: *PixelTrack*[24]. A diferencia del algoritmo de seguimiento clásico donde se requiere un modelo segmentado de fondo para realizar los cálculos, en *PixelTrack* se va creando dinámicamente. La principal ventaja de esta característica es que permite el uso de cámaras móviles.

Este sistema utiliza modelos de Hough para detectar un objeto de interés y determinar su centro de masas. También detecta los cambios de forma utilizando el modelo de segmentación, calculado previamente. Obteniendo estos parámetros, el algoritmo puede establecer un seguimiento visual de los elementos de interés en los distintos fotogramas, con bastante precisión. En la figura 1.20 se puede encontrar una muestra del proceso de seguimiento utilizando este sistema.

Por contra, los mayores inconvenientes que tiene respecto a los algoritmos clásicos son:

Figura 1.20: Proceso de seguimiento usando *PixelTrack*

- Cada vez que se inicia el algoritmo hay que establecer un área de búsqueda para indicar al programa la posición del objeto que estamos interesados para su seguimiento.
- Una vez ya arrancado el programa, en cada iteración el área de búsqueda empieza siendo mayor para asegurarse de que no se pierde y esto tiene un coste computacional elevado. Esto es debido a que para calcular el centro de masas debe procesar los datos de todos los píxeles del área seleccionada más de una vez.

1.3.3. Seguimiento de múltiples objetos

Este trabajo analiza el funcionamiento de diversos métodos para la detección y seguimiento de puntos de interés en un vídeo[18]. Entre los métodos para detección de puntos de interés más destacables se encuentran:

- **Filtrado de color HSV**: Consiste en seleccionar aquellos píxeles que están dentro del rango del filtro de color seleccionado, convertir el resultado a escala de grises y detectar contornos en la imagen resultante. De esta manera se pueden detectar objetos de interés de un determinado color en una imagen.
- **Segmentación de fondo:** Este sistema elimina el fondo de un vídeo, que en una cámara fija es estático, y dejar solo los objetos móviles, que en este caso serán los objetos de interés.

Para el seguimiento visual de objetos de interés se analizan los siguientes tres sistemas. Todos ellos están basados en la predicción matemática de los puntos con más posibilidades

de pertenecer a uno u otro objeto en seguimiento (o *track*). Cada uno difiere del otro en la manera de asignar la nueva posición calculada al objeto en seguimiento:

- **Filtro de Kalman:** En este método, se elige el punto más cercano al centroide del objeto en seguimiento.
- **Filtro de asociación de datos por probabilidad (PDAF):** Llegado el momento de relacionar los puntos, el procedimiento calcula la probabilidad de cada uno de ellos de pertenecer a uno de los *tracks* y se queda con los que poseen una probabilidad mayor a un valor umbral (que se calcula para cada punto obtenido con el PDAF y en cada iteración). Esto quiere decir que un *track* puede tener más de un punto relacionado. En este trabajo, el propio método que define el valor umbral es el que decide qué punto es la nueva localización del *track* estableciendo la probabilidad de que se desplace ahí en la iteración actual, quedándose con el que mayor valor tenga.
- **Filtro fusionado de asociación de datos por probabilidad (JPDAF):** El método tiene la misma mecánica que el PDAF, con la salvedad que sirve para procesar varios *tracks* y no solo uno. El sistema devuelve una matriz Ω, donde vienen los resultados para cada *track*. Procesando de nuevo esta matriz con el método de valor umbral, obtenemos la posición actualizada del *track* en la iteración en curso. En la figura 1.21 se muestra este algoritmo en funcionamiento, siguiendo a varios elementos a la vez. En este caso, personas.

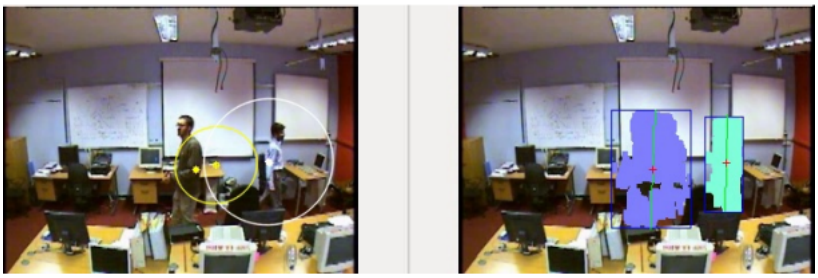


Figura 1.21: Funcionamiento del algoritmo JPDAF siguiendo a personas

Según las pruebas realizadas en este trabajo, el Filtro de Kalman es bastante inestable en sus resultados, dependiendo del sistema de detección de puntos de interés que se use. No funciona bien en imágenes con mucho ruido. Los que mejor funcionan son PDAF y JPDAF, incluso en situaciones de oclusión, donde se predice con bastante acierto el movimiento del objeto de interés. No obstante, JPDAF es bastante engorroso para trabajar con él debido a lo complicado de sus cálculos y eventualmente puede llegar a requerir bastante capacidad de cómputo.

1.3.4. ElderCare

ElderCare es un sistema basado en visión computacional, pensado para la asistencia y cuidados a gente mayor. Está implementado sobre JdeRobot por el grupo de Robótica de la Universidad Rey Juan Carlos[5, 22, 13, 21].

Este programa, de forma totalmente autónoma, realiza un análisis simultáneo de las imágenes captadas desde diversas cámaras de una habitación concreta. De esta forma, se puede realizar un seguimiento en 3 dimensiones a una o varias personas a través de una habitación, como muestra la figura 1.22. El sistema también puede detectar si una persona cae al suelo y en ese caso, si se mantiene inmóvil durante un tiempo determinado, salta una alarma.

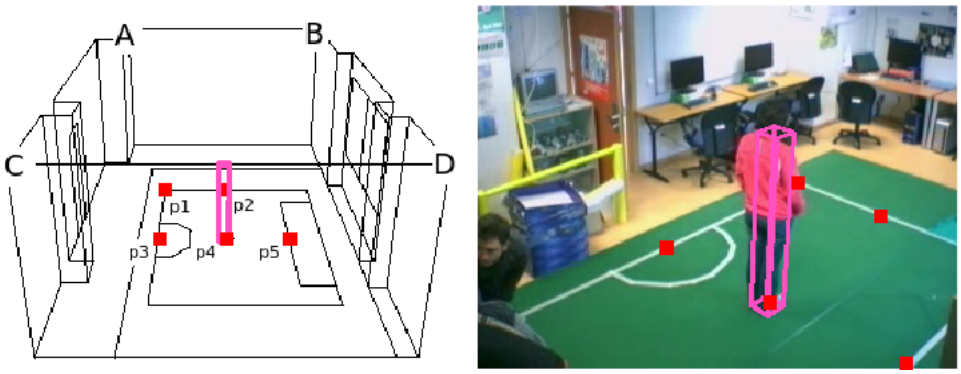


Figura 1.22: Estimación de la posición en 3D de una persona en una habitación utilizando *ElderCare*

ElderCare se basa en diversos algoritmos de seguimiento evolutivos, tanto en 2D como en 3D, utilizando técnicas de cálculo de puntos característicos, filtrado de imágenes por color en el modelo HSV y más tipos de filtros, junto con técnicas basadas en cálculos geométricos para el seguimiento en 3D, con el fin de aportar robustez al proceso de seguimiento y obtener resultados fiables.

1.3.5. TrafficMonitor

TrafficMonitor es una aplicación desarrollada sobre JdeRobot por el grupo de Robótica de la Universidad Rey Juan Carlos[20] cuya funcionalidad principal es contar y catalogar los vehículos que circulan por una vía, así como estimar la velocidad a la que van, a partir de los datos de vídeo que el software recoge (Ver figura 1.23).

El programa pasa por dos etapas: detección y seguimiento. En la etapa de detección, se detectan los nuevos vehículos que aparecen en el fotograma actual y la etapa de seguimien-

to comienza a continuación, relacionando cada vehículo detectado con los de fotogramas anteriores. Junto a estos procesos, también se calcula la posición de los coches en el plano de la carretera para ayudar a estimar su velocidad, para lo cual se realiza una homografía geométrica sobre el mencionado plano de la carretera y el plano de la imagen.

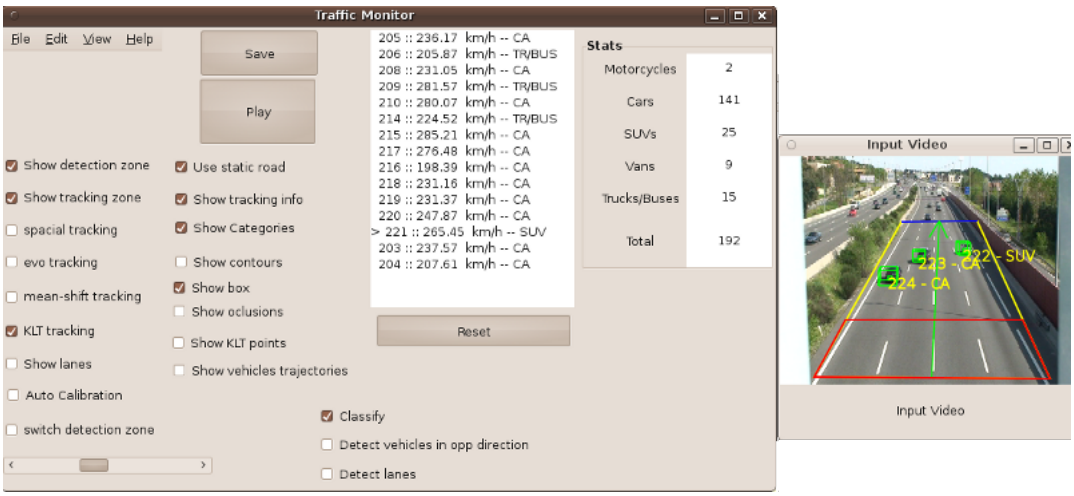


Figura 1.23: Aspecto de la interfaz de TrafficMonitor durante su funcionamiento

Para un buen funcionamiento de este software es necesario grabar la carretera por encima de ella, con buenas condiciones de iluminación y en la misma dirección y sentido que el tráfico. No obstante, ya que este trabajo se ha basado en la idea de este software, ha sido de valiosa ayuda en el transcurso de su desarrollo.

En el siguiente capítulo se fijarán las metas, se explicará lo que se quiere conseguir durante el transcurso de este trabajo y se expondrá la planificación del mismo. Después, se expondrá qué librerías y herramientas informáticas se han utilizado para ayudar a desarrollar el sistema. Más adelante, se detallará la solución propuesta al problema. En el capítulo 5, se pondrá a prueba el sistema con un conjunto de vídeos con distintos escenarios de visibilidad reducida, y se analizarán los resultados obtenidos. Por último, en el capítulo 6, se comprobará si la solución propuesta ha cumplido los objetivos fijados.

Capítulo 2

Objetivos

Tras exponer el contexto en el cual este Proyecto se sitúa, se explicará cuál es el fin de este trabajo, junto con los requisitos de la aplicación desarrollada y la metodología empleada en el proceso.

2.1. Descripción del problema

El objetivo final de este Proyecto Fin de Carrera es diseñar y desarrollar un *software* capaz de detectar los vehículos que pasan por una carretera y contarlos, bajo condiciones de visibilidad no ideales. El programa recogerá imágenes de vídeo de una sola cámara, siendo ésta la única fuente de datos. Tras la implementación, se realizará una serie de experimentos con diferentes vídeos para comparar los resultados arrojados por el sistema desarrollado con los datos reales, y comprobar así la eficacia de nuestro algoritmo.

Este objetivo final podría dividirse en tres subobjetivos, a los que llegaremos progresivamente en orden:

1. **Detección de nuevos vehículos:** El *software* deberá ser capaz de detectar vehículos en cada uno de los *frames* del vídeo. Para ello se han estudiado técnicas de visión artificial y se han integrado en el sistema con el objetivo de lograr esta meta.

2. **Seguimiento visual de vehículos y contabilización:** En cada fotograma del vídeo, el programa tiene que relacionar cada vehículo detectado en el fotograma anterior con los que se detecten en el que se encuentre. De esta forma, no se contabiliza un vehículo que ya ha sido detectado anteriormente y que sigue presentándose en la imagen. Se implementarán y probarán varias alternativas de sistemas de seguimiento

visual de vehículos, entre otros, un sistema comparador de descriptores de puntos de interés para hacer más eficiente este punto. También se ideará un sistema de conteo de vehículos: una vez se siga a un vehículo un tiempo determinado, se incrementará el contador global y se asignará el número resultante al vehículo que cumpla esta condición.

3. **Validación experimental:** Se reunirá un conjunto representativo de vídeos con condiciones variadas de iluminación, posición de la cámara y resolución. Se realizará una batería de pruebas con estas grabaciones como entrada de datos. Después se comparará cada uno de los resultados arrojados por el sistema con los reales. De esta forma se podrá comprobar la precisión que posee el algoritmo implementado en vídeos con visibilidad reducida de distinta índole.

2.2. Requisitos

Durante el desarrollo del sistema para la consecución de los objetivos previamente descritos, es necesario que el *software* cumpla también una serie de requisitos:

- El *software* deberá ser capaz de recoger datos de vídeo de una fuente cualquiera, de forma que esta información posibilite ser almacenada y procesada para extraer los resultados que el algoritmo necesite. La cámara que tome las imágenes deberá permanecer en una posición fija enfocando al tráfico.
- El sistema deberá realizar sus cálculos en tiempo real y de manera fluida. El procesamiento de las imágenes no tendrá un coste computacional elevado. El programa deberá ser capaz de ser ejecutado en procesadores poco potentes. De esta forma, posibilitaremos que un supuesto montaje del sistema resulte más económico.
- El lenguaje de programación que se utilizará para el desarrollo del proyecto será C++.
- El sistema operativo sobre el que transcurrirá el desarrollo del proyecto será GNU/Linux, dado que es lo utilizado en el laboratorio de Robótica.
- La arquitectura del sistema se diseñará en base a la utilizada en JdeRobot, en concreto, al módulo `basic_component`. Se utilizará como referencia este diseño, la cual se basa en la ejecución multihilo. El *software* ejecutará concurrentemente dos hebras: una se encargará de los procesos relacionados con la interfaz de usuario y la otra los procesos que tienen que ver con el control y la API. De esta manera, se

podrán gestionar los recursos de una manera más eficiente durante la ejecución del sistema.

2.3. Metodología

Todo proceso de desarrollo *software* debe someterse a una metodología. De esta forma se pueden identificar y organizar todas las etapas en dicho proceso de desarrollo. Si se encuentra la metodología idónea, se puede llegar a obtener mayor rendimiento del tiempo dedicado a la creación de un *software*, pudiendo incluso llegar a optimizar hipotéticos costes en el desarrollo de un proyecto.

La metodología utilizada en este proyecto fin de carrera está basada en el modelo incremental. Consiste en agrupar una serie de tareas en distintas etapas en las que se busca conseguir un pequeño hito dentro del desarrollo del sistema. La figura 2.1 puede ayudarnos a comprobar que la estructura básica de cada etapa repite a la anterior.

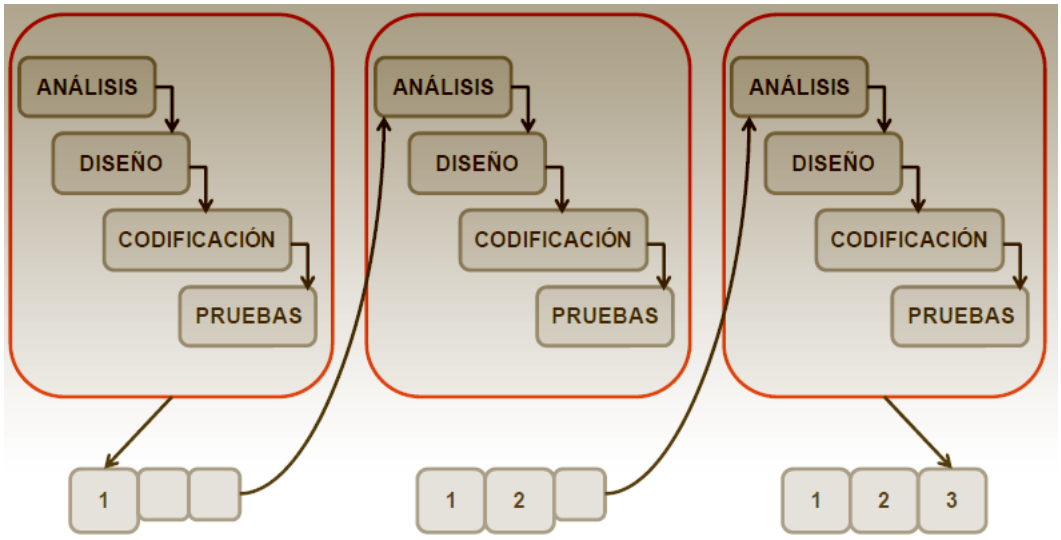


Figura 2.1: Esquema del desarrollo incremental

La mayor ventaja de este modelo es que permite añadir cada vez más funcionalidad a un programa en desarrollo, a partir de la versión más reciente de éste, cumpliendo poco a poco con los objetivos que se hayan establecido. También posibilita añadir nuevas funcionalidades al sistema, una vez acabado. Otra de los beneficios de este modelo es que permite probar el *software* desde etapas muy tempranas, detectar errores y corregirlos en alguna de las siguientes etapas. Esto permite a los desarrolladores adaptarse a la situación que se presenta con cada etapa, pudiendo incluso rectificar la planificación de trabajo del proyecto sin muchos inconvenientes, si fuera preciso.

Dada la flexibilidad que proporciona este modelo, es lo más adecuado para un proyecto de este tipo, ya que permite la inclusión de requisitos que en un principio no estaban incluidos cuando se establecieron la lista de objetivos y requerimientos que debía tener este sistema.

Se mantendrá un registro en la página web del grupo de Robótica en formato *wiki* durante todo el proceso de desarrollo[4], donde se explicará cada avance conseguido e integrado en el sistema, se publicarán vídeos e imágenes de cada funcionalidad nueva implementada y se mostrarán los resultados. También se llevarán a cabo diversas reuniones de seguimiento con el tutor, las cuales servirán para llevar a buen puerto la consecución de los objetivos establecidos para este proyecto.

El proceso de desarrollo constará de las siguientes etapas:

1. **Aprendizaje del entorno JdeRobot:** Para empezar a abordar este proyecto, se instalará JdeRobot y se empezará a estudiar su arquitectura y el funcionamiento de sus módulos más significativos. Se instalarán y ejecutarán los componentes *Cameraserver* y *Cameraview* para comenzar a entender su funcionamiento y la interacción entre módulos del entorno. También se probará el componente JdeRobot en los simuladores de entorno *Gazebo* y *Stage/Player* dedicado al control telemático o programado de robots virtuales en escenarios pregenerados.

2. **Aprendizaje de técnicas de tratamiento de imágenes:** Se instalarán manualmente las librerías de *OpenCV*. Se probarán los componentes *Opencvdemo* y *Colortuner* de JdeRobot. Se leerá el código de *Opencvdemo* para comprender el funcionamiento de todos los métodos de *OpenCV* que se utilizarán en este componente, entre los que se incluyen el flujo óptico y un detector de esquinas. Se programará también un pequeño componente para experimentar con los sistemas de extracción de descriptores de imagen, que aportan las librerías de *OpenCV*: *SIFT*, *SURF* y *ORB*. De esta forma, se analizará el método de utilización de estos sistemas en el código y observaremos los resultados de cada uno utilizando las dos mismas imágenes. También se estudiarán e implementarán diversas técnicas de segmentación de fondo, utilizando aprendizaje exponencial y mezcla de Gaussianas, entre otras.

3. **Estudio de las herramientas de desarrollo:** Se aprenderá el funcionamiento interno de los componentes *Opencvdemo* y *colortuner*. Para profundizar y aplicar lo aprendido, se procederá a adaptar estos dos componentes de JdeRobot a los métodos de la librería de *OpenCV* de la versión 2.3.4.1. Para ello, se necesitará adaptar al lenguaje C++ el componente *Opencvdemo* para que funcione en esta versión de la librería. En este punto, se comprobará el funcionamiento de *ICE* al tener que

comunicarse *Opencvdemo* con el componente *cameraserver*. También se aprenderá a utilizar la herramienta *Make* y a configurar la compilación de código mediante los *makefile*.

4. **Análisis de TrafficMonitor y aprendizaje sobre la utilización de herramientas gráficas:** Se revisará el código de *TrafficMonitor*[20] para fijar la atención en la mecánica de su funcionamiento. Se aprenderá sobre GLADE y GTK+. También se aprenderá cómo gestionar las señales que se lanzan desde el GUI, usando métodos de GTK+.

5. **Implementación de los algoritmos de detección y seguimiento:** Se utilizarán los métodos de la librería `cvBlob` para realizar la detección de vehículos nuevos, aunque antes se tendrán que adaptar dichos métodos para poder ser utilizados en C++. Se implementará un algoritmo de seguimiento propio, que se apoyará en el tamaño de *blobs*, su distancia y en la comparación de descriptores de los puntos de interés de cada *blob* encontrado entre cada fotograma consecutivo.

6. **Experimentos:** Finalmente, se realizará una recopilación de vídeos donde se recoge el flujo de tráfico en condiciones de visibilidad reducida y óptima, tanto diurnos como nocturnos. Se efectuarán diferentes pruebas con cada uno de estos vídeos. Mediante una comparativa de los resultados obtenidos y los reales, se reflejarán las impresiones de los datos resultantes en el apartado correspondiente de la memoria.

Capítulo 3

Infraestructura

En el desarrollo de este Proyecto Fin de carrera nos hemos ayudado de herramientas y librerías que conforma el esqueleto para nuestro programa. Desde el diseño de la propia interfaz, pasando por comunicar componentes del sistema, hasta procesamientos intermedios para los datos de la imagen. En este capítulo detallaremos todo lo que hemos utilizado para hacer realidad nuestro componente.

El lenguaje que hemos utilizado para crear este programa ha sido C++ ya que nos permite adaptarnos con más facilidad al lenguaje en el que está implementada la plataforma *software* en la que nos apoyaremos.

Para hacer posible la gestión de distintas hilos de ejecución hemos hecho uso de la librería *pthreads* (`POSIX threads`). Ésta provee distintas utilidades para crear y gestionar varios hilos de ejecución en *software*. Nuestro componente utiliza una hebra para gestionar todo lo relacionado con el GUI y otra para administrar la API. El cometido de la API es ejecutar las funciones necesarias según la selección del usuario en la interfaz, pasando los resultados de estas funciones de vuelta al GUI para mostrarlo por pantalla. Para el paso de datos entre la hebra GUI y la hebra API utilizamos variables compartidas y controlamos el acceso a las mismas mediante variables de control *mutex* para evitar condiciones de carrera.

3.1. Herramientas para la interfaz gráfica

Para poder seleccionar entre las distintas opciones que el componente ofrece, necesitamos implementar un medio con el que el usuario interactúe. Por ello hemos desarrollado un GUI (*Graphical User Interface*) con el que seleccionar distintas opciones de visualización

y modificar parámetros dentro de cada una en tiempo real. Para ello hemos utilizado las librerías GTK+ (*The Gimp Toolkit*)[1] [2]de la versión 2.4, compatibles con C++ y simplificando nuestra tarea al poder usar el mismo lenguaje que utiliza nuestro componente.

3.1.1. GTK+

Llamado originalmente *Gimp Toolkit* por haber sido creadas originalmente para el desarrollo del programa de edición de imagen Gimp, GTK es un conjunto de librerías diseñadas para llevar a cabo el desarrollo de interfaces gráficas en distintos entornos de escritorio. Está adaptado para lenguajes como C, C++, C#, Fortran, Java, Ruby, Perl, PHP o Python.

Los desarrolladores de este compendio de librerías fueron los equipos de GTK+ y GNOME y está disponible bajo licencia LGPL. Las librerías de las que se compone son:

- **Glib:** Aporta la infraestructura de datos de GTK+ para todas las plataformas de lenguaje anteriormente mencionadas.
- **GTK:** Librería que contiene todos los métodos y definición de variables para los *widgets* de una interfaz.
- **GDK:** Actúa de intermediario entre el Servidor X y GTK+, ayudando al renderizado de los elementos del GUI para mostrarlos por pantalla.
- **ATK:** Ofrece métodos y elementos específicos, diseñados para construir interfaces o añadir elementos adaptadas a personas discapacitadas.
- **Pango:** Alberga tanto el diseño como el renderizado del texto para la interfaz. Aquí se recogen todos los caracteres especiales e internacionales.
- **Cairo:** Implementada como API independiente y no sólo como librería. Proporciona los métodos para *renderizar* imágenes basadas en gráficos vectoriales y puede utilizar aceleración por hardware cuando está disponible.

Los elementos que componen la ventana de un GUI se denominan *widget*s. Internamente, es posible implementar los efectos que tendrá cada uno de nuestros clicks en cada uno de los *widget* gestionando los eventos (*signals*) que lanzan.

Cada uno de los *widgets* hereda los métodos de un tipo de datos primigenio que es Gtk::Object y después va especializándose, dependiendo del tipo de *widget* que sea (botón,

[1]Web: `http://www.gtk.org/`

barra deslizante...), adquiriendo los métodos de cada tipo de datos del que procede (Figura 3.1).

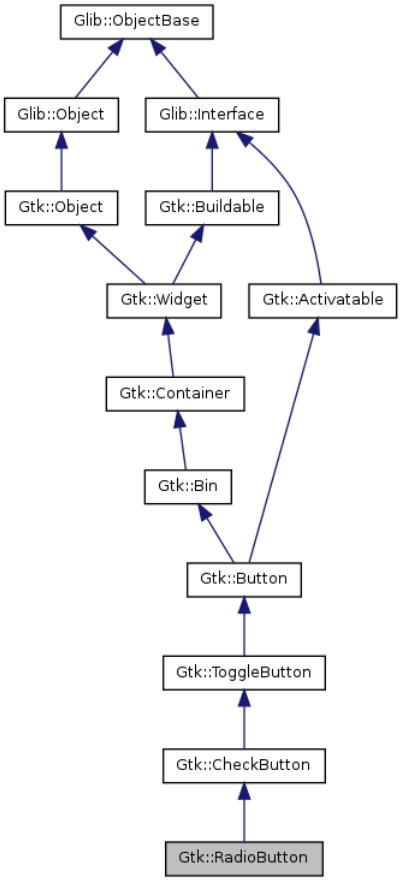


Figura 3.1: Ejemplo de herencia en el tipo Gtk::RadioButton

Los eventos que puede activar cada uno de los *widgets* se denominan *signals*. Cuando el usuario realiza alguna acción sobre un *widget* determinado tales como clickar en el área que ocupa este elemento, arrastrar con el ratón el cursor, activar o desactivar un botón... se pueden recoger en nuestro código y establecer las acciones que deseemos que realice nuestro componente.

De este modo podemos hacer que nuestro software reaccione como queramos dependiendo de las acciones que realice el usuario a través de la interfaz que hemos construido y mostrar los resultados por pantalla a través del GUI, mediante el *widget* correspondiente, únicamente utilizando las herramientas que nos brinda GTK+, facilitándonos la tarea de desarrollo de esta parte del componente.

3.1.2. Glade

Glade es una herramienta visual para el diseño de interfaces gráficas con GTK+. Dicha herramienta provee al diseñador la manera de seleccionar y colocar directamente los elementos de un GUI y previsualizar el estado del mismo. También permite modificar un amplio ámbito de parámetros de dichos elementos en el propio editor gráfico. En la figura 3.2 se puede ver una captura de la ventana principal de Glade.

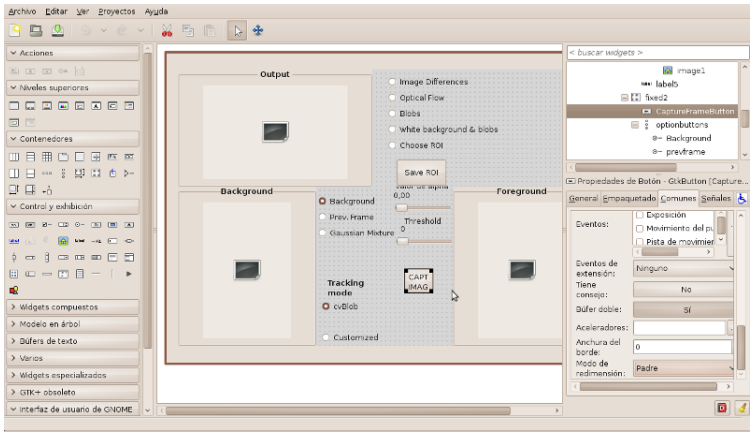


Figura 3.2: Glade ejecutándose en Ubuntu 10.04

Su funcionamiento es muy simple e intuitivo: Seleccionamos el tipo de ventana donde colocaremos los componentes de nuestra interfaz. Después establecemos los contenedores donde pondremos nuestros *widgets* o elementos del GUI, ya necesitemos compartimentar dentro de una misma ventana o establecer toda la superficie de la ventana como espacio útil. Después podemos establecer qué *widgets* colocar en estos espacios, dónde y sus dimensiones. A través de unas pestañas, a la derecha, podremos editar los parámetros de cada *widget* tales como si se prefiere que sea visible para el usuario desde el principio o no, establecer límites numéricos en las barras deslizantes, etiquetar los botones...

La interfaz que diseñemos en Glade es independiente del lenguaje con el que estemos trabajando. No genera ningún código fuente pero sí un archivo XML, con el que tendremos la posibilidad de acceder a cada elemento del GUI, asignar datos a elementos o modificar parámetros desde nuestro código fuente.

3.2. JdeRobot

JdeRobot es un conjunto de utilidades especializadas en robótica, domótica y visión artificial desarrolladas por el grupo de Robótica de la Universidad Rey Juan Carlos. El código se encuentra bajo licencia GNU GPL versión 3.

Esta plataforma consta de varios componentes que pueden funcionar independientemente y también unos con otros, pudiendo construir de esta forma un sistema donde podemos elegir qué utilidades necesitamos y tener la posibilidad de que se comuniquen entre sí. Entre estos elementos encontramos los que simplemente recogen datos del mundo real y los digitalizan (*drivers*), y otros que realizan cálculos y procesan estos datos. La comunicación entre estos componentes se realiza mediante interfaces ICE.

Internet Communications Engine o ICE es un *middleware* cuya funcionalidad principal es posibilitar la comunicación entre procesos de una o distintas máquinas, con independencia del lenguaje de programación con el que se hayan implementado. Se utiliza en JdeRobot, para la comunicación entre los distintos componentes y utilidades. Esto aporta al desarrollador una infraestructura sólida mediante el uso de interfaces para posibilitar la interconexión entre los distintos procesos que se requiera tener ejecutando y los sensores que recogen datos o actuadores de un robot (cámaras de vídeo, infrarrojos, motores, etc...). Para el componente desarrollado se ha utilizado la versión 3.3.

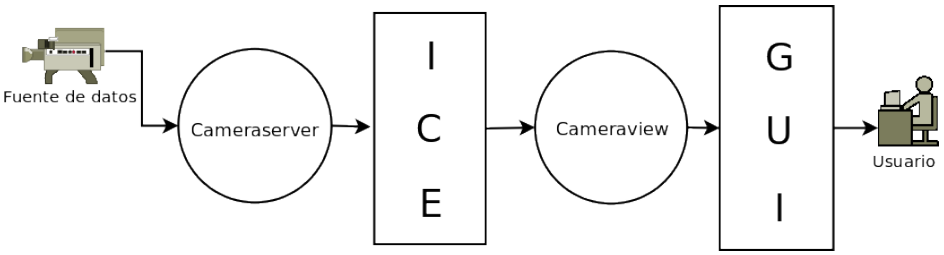


Figura 3.3: Diagrama del flujo de datos del componente Cameraview

Se incluyen librerías para ayudar al desarrollo de diversas herramientas especializadas en el ámbito de JdeRobot. También el soporte para diferentes sensores y motores de robots, sensores *Kinect*, *Wiimote*, dispositivos X10, así como para cálculos geométricos y otros métodos para ayudar a procesar imágenes para visión artificial. Más utilidades que ofrece JdeRobot son los simuladores Stage y Gazebo que permiten simular un escenario concreto para un robot y comprobar su comportamiento en él.

Mención especial merece para este PFC el componente `cameraserver`, cuyo objetivo es actúar de servidor de datos de vídeo para que cualquier software pueda hacer uso de dichos datos. La característica más destacada es que este programa puede configurarse fácilmente para recoger estos datos en crudo desde una cámara, de forma directa, desde un fíchero de vídeo ubicado tanto en un disco duro local o en la web. Ha simplificado enormemente la tarea de poder servir y recoger datos de vídeo para que el programa implementado pudiera procesarlos. En la figura 3.3 se puede ver un ejemplo del funcionamiento de `cameraview`, que requiere estar en comunicación con `cameraserver`.

3.3. Librerías de visión

En cualquier proyecto software es indispensable la utilización de distintas librerías que facilitan los cálculos intermedios que se necesitan para obtener cierto resultado, sea final o no. A continuación se señalan las más importantes para el desarrollo de este proyecto:

3.3.1. OpenCV

OpenCV (*Open Source Computer Vision Library*) es un compendio de librerías especializadas en visión artificial bajo licencia BSD. Originalmente fue creado por Intel, y lanzada su versión alfa en el año 1999, fue adquirido por Willow Garage y finalmente se unió a dar soporte a OpenCV la empresa Itseez.[2]

OpenCV fue creado para otorgar una infraestructura común independiente del entorno de desarrollo y ayudar a desarrollar aplicaciones especializadas en visión computacional, así como aumentar la eficiencia en los procesos de cálculo y hacer hincapié en el soporte a las aplicaciones de procesamiento en tiempo real. Dispone de interfaces compatibles con C++, C, Java y Python y está soportado en Windows, Linux, Mac OS, iOS y Android.

OpenCV tiene una estructura modular, lo que significa que cada librería está clasificada en las siguientes categorías:

- **core:** Aquí se definen los tipos de datos y la infraestructura básica de OpenCV en cuanto a estructuras de datos y funciones primarias.
- **imgproc:** Todas las librerías ubicadas en esta categoría están relacionadas con el tratamiento de imágenes como las operaciones de filtrado, transformaciones geométricas, conversiones de sistemas de color, etc...
- **video:** Procedimientos relacionados con el análisis de vídeo tales como estimación de movimiento, extracción del fondo y sistemas de seguimiento de objetos.
- **calib3d:** Algoritmos relacionados con la calibración de cámaras, simples o estereoscópicas, estimación de posición de objetos y cálculos geométricos básicos en vista múltiple.
- **features2d:** Funciones especializadas en detección de puntos de interés, cálculo de descriptores y emparejadores de descriptores.

[2]Web de OpenCV:http://opencv.org/

- **objdetect:** Detección de objetos y su reconocimiento y clasificación en clases predefinidas.
- **highgui:** Proporciona un método sencillo para captura de vídeo, códecs para imagen y vídeo y procedimientos básicos para desarrollo de interfaces.
- **gpu:** Conjunto de algoritmos de las anteriores categorías pero permitiendo aceleración por GPU.

La versión de OpenCV utilizada en el desarrollo del sistema contador de coches ha sido la 2.3.4.1 y ha sido de muchísima ayuda, desde la utilización del tipo de dato `cv::Mat` para almacenar y procesar cada fotograma en forma de matriz de datos, hasta las operaciones de procesamiento de imágenes incluidas en las librerías.

3.3.2. cvBlob

Librería diseñada y desarrollada por Cristóbal Carnero Liñán en 2008[3] dedicada a detectar objetos de interés en una imagen en forma de *blobs* y gestionarlos de distintas maneras, apoyándose con OpenCV. Entre las funciones que se pueden encontrar en esta librería está por ejemplo el mostrar por pantalla los *blobs* que se encuentran en cada fotograma o hacer un seguimiento de los *blobs* que aparecen en pantalla. `CvBlob` se encuentra disponible bajo licencia GPL. Estas funciones están escritas en C y se ha tenido que adaptar mínimamente al código del sistema contador de coches para que utilizaran la funcionalidad en C++ de OpenCV, como en el resto del programa desarrollado. La versión utilizada de esta librería ha sido la 0.10.4.

El algoritmo que detecta *blobs* en el procedimiento `cvLabel` incluido en esta librería está basado en el trabajo de Fu Chang, Chun-Jen Chen y Chi-Jen Lu [14] en el que en una sola pasada de imagen pueden recoger tanto los *blobs* de una imagen como sus contornos. De hecho, detectando los contornos de una imagen el algoritmo va conformando los *blobs* en un algoritmo de cuatro pasos y haciendo una sola pasada. Su mecánica es como podemos ver en la imagen 3.4:

1. Una vez obtenemos los contornos de una imagen, la función recorre la imagen binaria de arriba a abajo y de izquierda a derecha. Cuando encuentra un punto perteneciente a uno de los contornos sin etiqueta, le asigna una nueva y única a todos los puntos que lo conforman.

[3]Web de cvBlob: `http://code.google.com/p/cvblob/`

2. Si en los siguientes barridos nos encontramos con un punto B que forma parte de un contorno que ya ha sido etiquetado, todos los puntos sin etiqueta que nos encontremos dentro de él, como el punto C, se le asignará la misma que la del punto B.

3. Una vez etiquetado el punto C, se recorrerá todos los puntos que conforman este nuevo contorno para asignarles la misma etiqueta que la del punto C.

4. Cuando se encuentre un punto D ya etiquetado de un contorno contenido dentro de otro, el algoritmo recorrerá horizontalmente hacia la derecha la imagen y asignará a los puntos en negro que encuentre la misma etiqueta que D hasta llegar al contorno exterior. De esta forma, aunque la figura conste de varios contornos internos, el algoritmo lo identificará como un solo componente o *blob*.

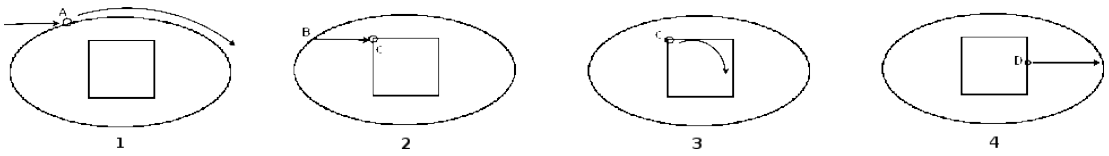


Figura 3.4: Pasos del algoritmo de la función cvLabel

En cuanto al algoritmo de seguimiento de blobs utilizado por esta librería en la función `cvUpdateTracks`, está basado en el trabajo *Appearance Models for Occlusion Handling* [23].

El sistema que adopta `cvBlob` para seguimiento de *blobs* es como sigue: Crea una matriz de adyacencias con todas las etiquetas de los *blobs* ya detectados en fotogramas anteriores (o `tracks`) y calculando en cada iteración del vídeo la distancia entre el centroide de cada uno y el punto más cercano de cada *blob* detectado en el fotograma actual. Se prefirió descartar la distancia euclídea por la dificultad que aporta ésta en la distinción de la contraposición y fusión de dos *blobs* muy cercanos. Consultando los valores de la matriz en cada iteración en base al identificador de cada `track`, puede discernirse si es un nuevo *blob* que ha aparecido en el fotograma o relacionar un *blob* con un `track` del anterior fotograma, además de poder consultar la distancia que separa a un *blob* y un `track` y cuantos elementos están relacionados entre sí con un bajísimo coste computacional ya que se tiene acceso directo a estos datos en la matriz.

El algoritmo de seguimiento de `cvBlob` ha servido de mucha ayuda para el desarrollo de un sistema de seguimiento propio, así como la función de detección de *blobs*, que se usa también en el programa.

3.4. Herramientas de desarrollo

Durante el desarrollo del componente se han utilizado herramientas que han servido para facilitar el trabajo y que merecen ser destacadas.

3.4.1. Make

Es una utilidad GNU que determina automáticamente qué ficheros de código fuente necesitan ser compilados y lanza los comandos necesarios para hacerlo, con ayuda de un fichero de texto llamado *makefile* donde se definen los parámetros de compilación que el usuario requiera. *Make* puede ejecutarse siempre y cuando el compilador del lenguaje con el que estemos trabajando pueda correr en `shell`. Fue implementada por Richard Stallman y Roland McGrath.

En el llamado *makefile* se fijan todos los parámetros que se necesitan para compilar el código, incluido los enlaces a librerías dinámicas y nombres de los ficheros resultantes de la compilación. También *make* admite otras facilidades como poder interpretar nociones de `bash` en el fichero: uso de variables, ejecuciones de comandos en segundo plano... lo que lo convierte en una herramienta potente.

A continuación se añade un ejemplo de makefile:

```
PROGRAM = car_counter
PROGRAM_FILE = car_counter.cpp gui2.cpp control.cpp API.cpp
    trackscc.cpp cvBlob/cvaux.cpp cvBlob/cvblob.cpp cvBlob/
    cvcolor.cpp cvBlob/cvcontour.cpp cvBlob/cvlabel.cpp cvBlob/
    cvtrack.cpp FLAGS = -g -Wall -o
JDEROBOTDIR = /usr/local CXXFLAGS = -I$(JDEROBOTDIR)/include/
    jderobot $(shell pkg-config gtkmm-2.4 libglademm-2.4 opencv
    --cflags)
LDFLAGS = -I$(JDEROBOTDIR)/include/jderobot -I$(JDEROBOTDIR)/lib
    /jderobot -I/usr/include/gearbox -I/usr/local/include/opencv2
      -I/usr/include/gtk-2.0 -I/usr/lib/gearbox  $(shell pkg-
    config gtkmm-2.4 libglademm-2.4 --cflags --libs)
LDADD = -L$(JDEROBOTDIR)/lib/jderobot  -lJderobotInterfaces -
    lopencv_core -lprogeo -lopencv_video -lopencv_features2d -
    lopencv_highgui -lopencv_imgproc -lJderobotUtil -lJderobotIce
     -DGLADE_DIR=.
CC = g++
carcounter:
$(CC) $(FLAGS) $(PROGRAM) $(PROGRAM_FILE) $(LDFLAGS) $(LDADD)
```

Gracias a toda la automatización que aporta la herramienta, simplifica y aclara mucho el proceso de compilación para el programador.

3.4.2. Subversion

Subversion o SVN es un sistema de control de versiones desarrollado por la empresa CollabNet y lanzado con licencia de software libre en Octubre del 2000 para desbancar al sistema más utilizado en aquel momento mejorando sus defectos: *Concurrent Versions System* (CVS). Actualmente se encarga de su soporte la empresa Apache.

Este sistema es de gran utilidad para los desarrolladores ya que permite llevar un histórico de todas las versiones de código fuente y/o documentación que se vaya implementando y poder compartirlo fácilmente *online* con el resto del equipo, si lo hubiera. Todos los cambios que se hagan en un fichero se reflejarán como cambios atómicos en Subversion, esto es, no subirá el fichero entero modificado, solo los cambios. Esto es la

principal mejora con su antiguo competidor CVS, ya que en este sistema si el proceso de subida se interrumpía por cualquier motivo podía desembocar en corrupción de ficheros.

SVN posee el mismo sistema de ficheros básico para todos los proyectos. Una vez que establecemos la ubicación del `root` del proyecto, se crea una estructura básica de directorios con tres carpetas:

- **Trunk:** Este es el directorio para la versión de desarrollo en curso y válida.
- **Tags:** Alberga las distintas versiones desarrolladas y cerradas del proyecto.
- **Branches:** Donde se ubican los desarrollos paralelos a lo que se está realizando en *trunk*.

Este tipo de programas se han hecho muy populares en el ámbito de desarrollo de software libre ya que permite compartir código online de una forma cómoda y organizada, controlando el número de versión cada vez que un miembro del equipo desarrolle algo nuevo y poniéndolo a disposición de quien posea permisos para acceder a los ficheros publicados.

Capítulo 4

Descripción Informática

En este capítulo se expondrá con detalle la solución propuesta al problema que se ha planteado anteriormente. Esto es, llevar la cuenta de vehículos que transcurren por una carretera mediante técnicas de Visión Artificial en distintas situaciones. Éstas pueden implicar desde una posición de cámara poco favorable hasta condiciones pobres de iluminación.

`Car_counter` es el componente desarrollado, pensado para aportar una solución que logre alcanzar resultados satisfactorios. Se mostrará su diseño y se explicará en qué librerías y algoritmos nos hemos apoyado. También se expondrá la mecánica de su funcionamiento: los parámetros que recibe en cada una de las opciones en que se estructura el componente y qué es lo que dibuja y muestra en la imagen de salida y en las de control. Se describirán los métodos que se han creado para las diferentes opciones que el componente posee para mostrar diferentes resultados.

4.1. Diseño

El componente ha sido pensado para recoger datos de un vídeo, fotograma a fotograma, procesar los datos del vídeo según haya sido la elección del usuario y devolver el resultado en el vídeo de salida integrado en la interfaz del software. El modelo de caja negra del componente se encuentra detallado a continuación, en la figura 4.1:

El componente se ha desarrollado modularmente y ello ha posibilitado que el usuario, mediante la interfaz gráfica, pueda seleccionar y comprobar en el vídeo de salida cuál es el resultado de cada uno de los procesamientos intermedios por los que pasa cada fotograma,

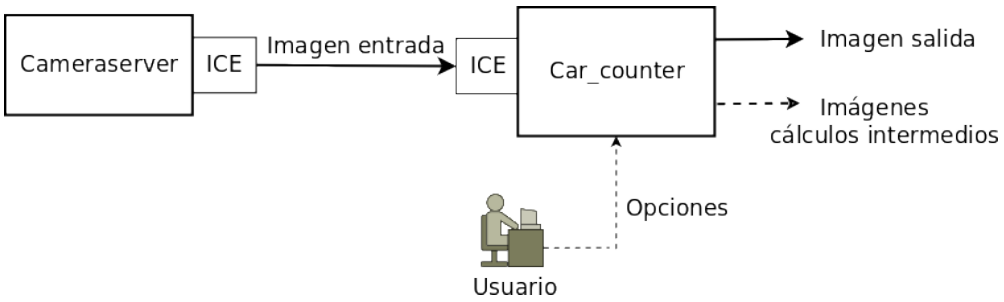


Figura 4.1: Diagrama de entradas y salidas del componente `Car_counter`

esto es, por ejemplo, fondo aprendido e imagen de diferencias, usando distintas técnicas, entre otros.

Se ha usado una estructura de dos hebras. Una ejecuta el GUI y la otra la obtención de imágenes y su procesamiento. Las variables compartidas entre estas dos hebras de procesamiento están protegidas mediante exclusión mutua (`mutex`) para evitar condiciones de carrera.

Se ha desarrollado un algoritmo, y una estructura de datos en el que se apoya aquel, basado en el que aparece implementado en las librerías de `cvBlob` y se ha mejorado, pudiendo analizar vídeos cuya visibilidad no es la óptima para ello. Este algoritmo procesa la imagen en 4 etapas:

1. Segmentación de fondo.
2. Detección de *blobs* en imagen.
3. Proceso de seguimiento que se divide en los siguientes pasos:
   a) Primer emparejamiento de *blobs* y `Tracks` por área y distancia entre un frame y otro.
   b) Detección de puntos característicos de cada *blob* y `Track`.
   c) Describir estos puntos de interés (es decir, calcular los descriptores asociados a dichos puntos).
   d) Segundo emparejamiento de los puntos de interés entre ambas estructuras.
4. Actualización del número de vehículos contados y asignación de número a cada blob detectado y reconocido en distintos fotogramas seguidos.

Todo este proceso puede verse reflejado en el diagrama de bloques plasmado en la figura 4.2.

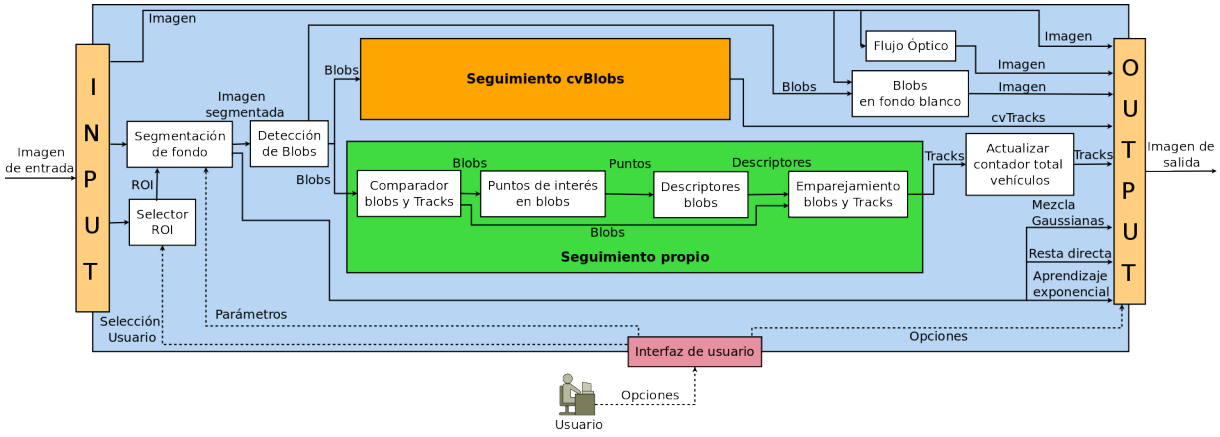


Figura 4.2: Diagrama de bloques de funcionamiento del componente `Car_counter`

Para comparar el funcionamiento de varias opciones, el componente da a elegir al usuario si quiere ejecutar el seguimiento de *blobs* de `cvBlob` o el que se ha desarrollado expresamente y da a elegir qué variante de segmentación de fondo se quiere utilizar, ya sea por mezcla de Gaussianas o diferencia con el aprendizaje de fondo.

4.2. Región de Interés

Una Región de Interés o ROI por sus siglas en inglés (*Region Of Interest*) es una zona de una imagen, generalmente seleccionada por el usuario, donde se quiere procesar algo de manera independiente a los demás datos del resto de la imagen.

Por motivos de optimización y eficiencia, es apropiado que el componente ofrezca al usuario la opción de seleccionar una región del vídeo para contabilizar los vehículos que pasen por esa sección, y no que se procese el vídeo completo, lo cual la mayoría de las veces no tiene sentido.

Para poder incluir esta funcionalidad, se aprovecha una característica del tipo de datos `cv::Mat`, perteneciente a la librería OpenCV, utilizado para guardar los datos de cada imagen de los fotogramas del vídeo.

Este tipo de datos posee un constructor, el cual hace posible extraer una Región de Interés de forma rectangular de otra variable de tipo `cv::Mat` ya existente. Dicho constructor se detalla a continuación:

```
Mat::Mat(const Mat& m, const Rect& roi)
```

- `m` - Variable donde se almacena la imagen de la que queremos extraer su Región de Interés.

- `roi` - Variable del tipo `cv::Rect`, en el cual se detallan las coordenadas donde se ubican los vértices del rectángulo.

El tipo `cv::Rect` consta simplemente de una tupla de 2 puntos, que cada uno a su vez consta de otra tupla de 2 elementos, con sus coordenadas en las ordenadas y las abscisas. El primer punto corresponde al vértice de la esquina superior izquierda del rectángulo y el segundo punto al vértice de la esquina inferior derecha.

La Región de Interés extraída con este método tiene una particularidad, y es importante tenerla en cuenta: los datos de la variable `cv::Mat` resultante de este método está vinculada a la variable original de donde se extrae el ROI.

Una vez que se ha procesado la matriz con la imagen y ésta utiliza una Región de Interés, para asignar los resultados al lugar correcto de la matriz original es necesario trasladar el sistema de coordenadas de la Región de Interés a la matriz original, utilizando las coordenadas del rectángulo seleccionado por el usuario como referencia. Para ello, solo es necesario sumar las coordenadas del vértice correspondiente a la esquina superior izquierda del ROI para trasladar correctamente los resultados a la matriz original. Como puede verse en la figura 4.3, la imagen de la derecha corresponde a la imagen del ROI, que es la que se procesa. Los resultados se obtienen en base al sistema de coordenadas del ROI y, por tanto, hay que trasladarlas a la matriz original, que es la que correspondería a la imagen de la izquierda. El traslado de resultados de una matriz a otra se realiza correctamente.

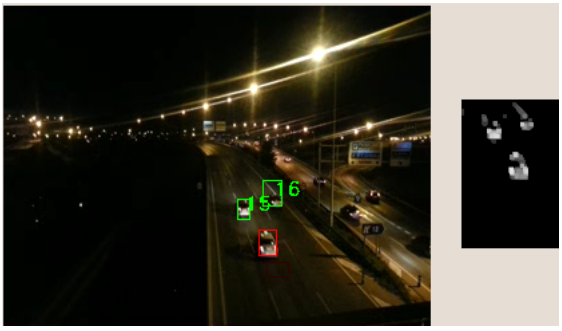


Figura 4.3: Aspecto del ROI y traslado de resultados a la imagen original

4.3. Segmentación de fondo

En cuestiones de tratamiento de imágenes, es esencial reducir cuanto se pueda el coste computacional para ejecutar un componente puesto que nos interesa presentar los resultados que nos da el algoritmo en tiempo real de reproducción del vídeo. También interesa facilitar la tarea a funciones como detectores de blobs o de puntos de interés para afinar y obtener mejores datos del proceso. Es por ello que se realiza antes de nada una segmentación de fondo de la imagen para descartar las zonas que no tienen interés ninguno para nuestros objetivos.

En este componente se han incluido tres tipos de segmentación de fondo, lo cual sirve para comprobar su funcionamiento y también nos apoyaremos en alguno de ellos para realizar la detección y seguimiento posterior de vehículos.

4.3.1. Mezcla de Gaussianas

La librería OpenCV aporta varios procedimientos para la segmentación de fondo, entre los cuales se encuentra la segmentación mediante mezcla de Gaussianas. Basado en el algoritmo desarrollado por Zoran Zivkovic[26][27], esta función permite la detección de objetos en primer plano de vídeos así como la extracción del fondo. También puede incluir en la detección las sombras pertenecientes a dichos objetos.

Esta técnica se basa en un modelo estadístico, el cual detecta puntos de interés en una imagen para después agruparlos en diferentes conjuntos. La acumulación de puntos cuya distribución se ajuste más a una distribución Gaussiana conformarán estos conjuntos, los cuales serán los objetos de interés en una imagen (Figura 4.4).

Este método tiene en cuenta los cambios de iluminación de la escena. A medida que se va reproduciendo el vídeo, el algoritmo de OpenCV va clasificando la información de la imagen según los detecte como objetos en primer plano y el fondo.

Posee un problema importante y es que su coste computacional es muy elevado. Además, el resultado que se obtiene con el uso de esta técnica en el algoritmo de seguimiento visual de objetos desarrollado no fue nada satisfactorio en las pruebas que se realizaron con él.

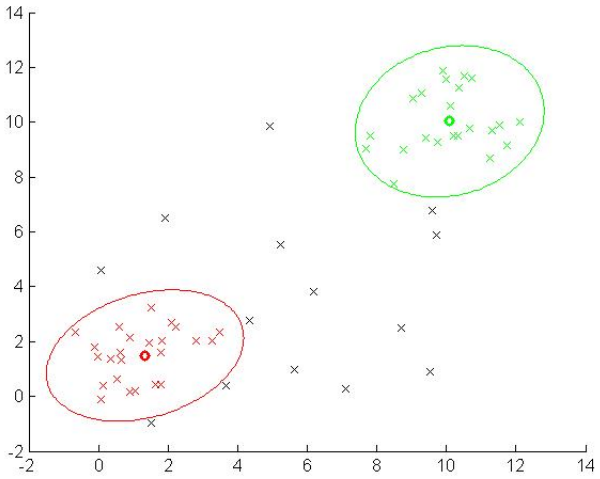


Figura 4.4: Agrupación de puntos de interés por Mezcla de Gaussianas

4.3.2. Resta directa

Las imágenes con las que se va a trabajar se obtienen a través de una cámara fija. Por tanto, los valores de los píxeles del fondo serán estáticos, o variarán muy poco. Con esta premisa en mente, se puede observar que restando los valores de los píxeles de dos fotogramas consecutivos, y desechando los píxeles que no lleguen a cierto umbral, se puede extraer objetos móviles en primer plano, eliminando el fondo. En la figura 4.5, se puede comprobar el resultado de aplicar la siguiente ecuación a un vídeo, correspondiente a esta técnica.

$$PrimerPlano(x,y)_t = |Fotograma(x,y)_{t-1} - Fotograma(x,y)_t|$$

Figura 4.5: Resultado de aplicar Resta Directa a un vídeo de tráfico nocturno

Si el usuario no establece ningún umbral mínimo de diferencia, el resultado muestra cualquier tipo de variación entre los dos fotogramas. Por tanto, cualquier movimiento

por pequeño que sea es detectado con este método y, en cálculos posteriores del proceso, podrían detectarse falsos positivos.

4.3.3. Aprendizaje exponencial de fondo

En este método de segmentación de fondo se descarta los objetos móviles o cambiantes paulatinamente en los distintos fotogramas y permanece lo que aparece constante, lo que en el caso que nos ocupa sería la carretera, mediante un cálculo de la media de valores de luminosidad obtenida a partir de los distintos cuadros del vídeo a través del tiempo.

Aplicando un proceso acumulativo dependiente de un valor α, será posible obtener las partes estáticas de una imagen, es decir, el fondo. Para realizarlo, se aplica la siguiente ecuación a cada píxel de la imagen:

$$Fondo(x,y)_t = \alpha * Fondo(x,y)_{t-1} + (1-\alpha) * FotogramaActual(x,y)_t$$

El usuario puede alterar este valor α mediante una barra deslizante de la interfaz de usuario. Cuanto más pequeño sea este valor, más cercano será el resultado al fotograma original. Si se incrementa a valores cercanos al 1 entonces se comprobará cómo quedan sólo visibles en la imagen las partes estáticas. Este valor podrá ser modificado mientras el algoritmo se ejecuta para poder afinar los resultados que el componente devuelve. En la interfaz hay reservado un hueco para mostrar la imagen calculada tras la segmentación de fondo, de esta forma el usuario podrá controlar mejor lo que recibe el detector de blobs.

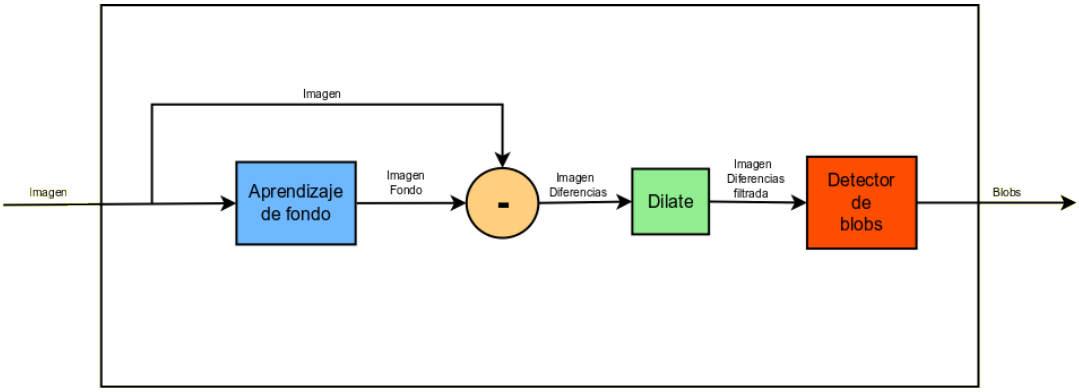


Figura 4.6: Cálculo Segmentación de Fondo

Una vez se obtiene el fondo aprendido, se puede calcular lo que se llama "imagen diferencias" para obtener los objetos en primer plano. Para ello, se resta el valor de cada píxel de la imagen del fondo aprendido al de cada píxel correspondiente de la imagen

normal. El resultado, si es menor que cierto umbral, se le asigna el valor 0. Si es mayor, se le asigna este valor diferencia. (Figura 4.6)

$$ImagenDiferencias(x,y)_t = \begin{cases} |FrameActual(x,y)_t - Fondo(x,y)_t| < Umbral, & 0 \\ |FrameActual(x,y)_t - Fondo(x,y)_t| \geq Umbral, & |FrameActual(x,y)_t - Fondo(x,y)_t| \end{cases}$$

Al obtener la imagen diferencias, se aplica una operación de filtración de imagen, `dilate`, ubicado en la librería de OpenCV, para que los contornos de los objetos móviles que salen en la imagen resultante se vean más definidos, en busca de un mejor resultado en los procesamientos posteriores. Se trata de una dilatación de la imagen, que consiste en un crecimiento de píxeles. A continuación se detalla la función utilizada:

```
void dilate(InputArray src, OutputArray dst, InputArray element,
            Point anchor=Point(-1,-1), int iterations=1,
            int borderType=BORDER_CONSTANT,
            const Scalar& borderValue=morphologyDefaultBorderValue() )
```

- `src` – Imagen que se pasará como parámetro a la función.
- `dst` – Variable de la imagen destino del mismo tamaño y tipo que `src`.
- `element` – Elemento estructural a utilizar en la dilatación de la imagen. Si se usa cv::Mat() como parámetro, se utilizará una matriz 3x3 rectangular.
- `anchor` – Posición dentro de la estructura element, para realizar los cálculos de dilatación. El valor por defecto es (-1, -1), la cual es la posición central de la estructura.
- `iterations` – Número de veces que se aplicará dilatación en la imagen.
- `borderType` – Método de extrapolación del píxel.
- `borderValue` – Valor del borde en caso de seleccionar BORDER_CONSTANT en el parámetro anterior.

Al obtener la imagen diferencias tras aplicar `dilate()` se habrá realizado la segmentación de fondo, como podemos ver en la figura 4.7.

(a) Imagen Original

(b) Fondo aprendido

(c) Imagen Diferencias

Figura 4.7: Segmentación de fondo

4.4. Obtención de blobs

Una vez conseguidos los resultados dados por la segmentación de fondo, se pasan a la función `cvLabel` para la detección de blobs. Esta función pertenece a la librería `cvBlob` y su algoritmo está basado en el artículo "*A linear-time component-labeling algorithm using contour tracing technique*"[14]. Éste se basa en un sistema de detección y seguimiento de contornos del objeto que se detecta en la imagen e identifica y etiqueta el área interna de cada componente detectado en forma de blobs. La descripción de la cabecera de la función corresponde a la siguiente:

```
unsigned int cvLabel(IplImage const *img, IplImage *imgOut, CvBlobs &blobs);
```

- `img` es la imagen de entrada que se pasará al procedimiento.
- `imgOut` será la imagen que devuelve, aunque no se dará ningún uso en el algoritmo, ya que solo interesan los blobs.
- `blobs` es la variable donde se devuelven todos los blobs encontrados por el procedimiento. Esta variable está basada en el tipo `std::map` y cada elemento de esta estructura es donde se encuentran los datos de cada blob.

- La función devuelve un entero correspondiente al número de píxeles que ha etiquetado en blobs.

Una vez se obtiene la variable `blobs`, si no está vacía, se pueden reprocesar áreas de interés en la imagen a partir de estos datos o dibujar en la imagen lo devuelto por la función `cvLabel,` tal y como se ve en la figura 4.8.

Figura 4.8: Dibujos de blobs numerados en imagen

4.5. Seguimiento visual de vehículos y conteo

Una vez tengamos el resultado final de la segmentación de fondo de un fotograma, se puede hallar los `blobs`, u objetos de interés en pantalla, y realizar un seguimiento del mismo, identificando cada `blob` en el transcurso de la ejecución de las diferentes iteraciones del algoritmo con el objeto con el que se corresponde y que aparece en pantalla. El resultado de dicho proceso de seguimiento se irá almacenando en una estructura de datos llamada `track`.

Para ayudarse a hacer el proceso de correspondencia entre `blobs` y `tracks`, el algoritmo implementado compara el tamaño y cercanía entre los `blobs` del fotograma `t-1` y `t`, para después hallar sus puntos de interés y comprobar que efectivamente se está siguiendo el mismo objeto en `t` que en el fotograma `t-1`.

Aprovechando la característica del programa en la que se identifica numéricamente a cada `track`, se incluye la funcionalidad de conteo de vehículos dentro del propio sistema.

Para poder comparar el funcionamiento del algoritmo de la librería `cvBlob` y del algoritmo implementado, se ha incluido en la interfaz la opción de escoger ejecutar entre uno y otro durante la reproducción de un mismo vídeo.

Los dos algoritmos se han incluido en el sistema desarrollado como dos soluciones alternativas al problema del seguimiento visual de vehículos: el procedimiento `cvUpdateTracks`, incluido en `cvBlob`, y el algoritmo propio implementado: `UpdateTracks`.

4.5.1. Algoritmo de seguimiento en cvBlob

La librería `cvBlob` incluye un pequeño algoritmo de seguimiento, que trabaja con los *blobs* que se han detectado previamente. El planteamiento de este algoritmo es realizar el seguimiento relacionando un mismo *blob* en distintos fotogramas, utilizando nada más que la distancia recorrida entre un fotograma y otro. Este procedimiento trabaja con una matriz de distancias entre *blobs*, detectados en el fotograma actual, y los *tracks*, la cual se denomina matriz de adyacencias. Esta matriz se utiliza para cribar los resultados obtenidos optimizando el consumo de recursos del sistema, pudiendo así distinguir entre elementos de interés reconocidos de anteriores fotogramas y nuevos elementos detectados que aparecen por primera vez de manera rápida.

La función de la librería cvBlob tiene el siguiente aspecto:

```
void cvUpdateTracks(CvBlobs const &b, CvTracks &t,
                    const double thDistance,
                    const unsigned int thInactive,
                    const unsigned int thActive=0);
```

- `b` - Variable donde se encuentran los *blobs* detectados
- `t` - Variable donde se encuentran los *tracks*
- `thDistance` - Máxima distancia de separación entre un *blob* y un *track* para ser considerados el mismo elemento. Si este valor se rebasa, se considera que ese blob y ese track no son el mismo.
- `thInactive` - Número máximo de fotogramas en los que un *track* puede quedarse inactivo.
- `thActive` - Si un *track* pasa a inactivo, pero ha estado activo menos cantidad de fotogramas que el indicado en este argumento, se elimina.

El algoritmo hace uso de acumuladores para cada *blob* y *track*, relacionados con sus identificadores. Cada acumulador se incrementa cada vez que se detecta que una distancia entre

un *blob* instántaneo y un *track* en la matriz es menor que la distancia umbral, establecida en los parámetros de la función. De esta forma, cada acumulador sirve como un indicador que muestra cuántas veces se ha cumplido la condición de la distancia para un *blob* o *track* concreto.

En la figura 4.9 se puede observar la estructura de la matriz de adyacencias. Tras las líneas, se sitúan los acumuladores para cada *track* y *blob* detectado en el fotograma actual. En la parte superior y en el extremo izquierdo, en negrita, pueden verse los distintos identificadores para cada *blob* y *track* activos en ese momento.

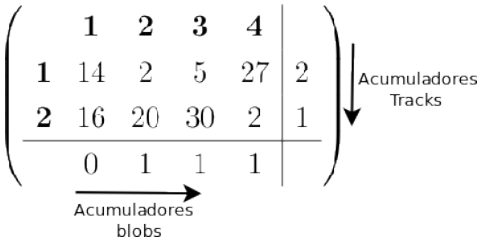


Figura 4.9: Representación de matriz de adyacencias utilizada por `cvBlob`

Una vez tenemos construida dicha matriz, ésta se analiza elemento a elemento y se crea una lista de *blobs* y otra de *tracks*. Dependiendo de si su acumulador es mayor que 0 o no, el *blob* o el *track* pasa a formar parte de la lista correspondiente de posibles candidatos aptos para ser emparejados. El algoritmo busca entonces en la lista de *tracks* el elemento con mayor área y lo empareja con el *blob* de mayor área que encuentra en la otra lista.

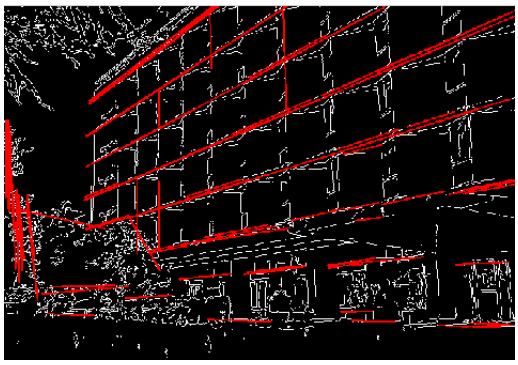
Sin embargo, los resultados que arroja este sistema no son del todo satisfactorios. Este algoritmo no realiza un seguimiento preciso de los vehículos. Por ejemplo, si dos vehículos están juntos, es frecuente que el sistema asigne el mismo *track* a uno o a otro en fotogramas consecutivos.

Otro problema es que tampoco es útil para realizar el conteo de los vehículos que circulan. El sistema no utiliza ninguna variable para almacenar el identificador del último track detectado. Cuando el algoritmo no detecta *tracks* ni *blobs* en el fotograma, la cuenta vuelve a reiniciarse.

4.5.2. Detectores de puntos de interés

En una imagen interesa detectar ciertos patrones, figuras con una cierta forma, un cierto tamaño... Para ello hay diferentes algoritmos para cada problema y la detección de cada figura, ya sea por puntos o bordes, como la transformada de Hough, utilizada para detectar líneas rectas y otras figuras geométricas detectando sus bordes (Fig 4.10). En

este apartado se hará una pequeña introducción a diversos métodos probados en nuestra aplicación.



(a) Detección de líneas

(b) Detección de círculos

Figura 4.10: Resultado de detección de distintos criterios en imágenes usando la Transformada de Hough

- **Shi-Tomasi:** Solución presentada por Jianbo Shi y Carlo Tomasi para la detección de esquinas dentro de una imagen[16] e implementada como la función *goodFeaturesToTrack* en las librerías OpenCV.
  Este algoritmo utiliza medidas de calidad en todos los puntos a nivel de píxel en la imagen de entrada. Estas medidas es el cálculo del valor propio(λ) de las submatrices, que por defecto tienen tamaño 3x3 en la función, que forma ésta en cada píxel de la imagen con sus contiguos. Utilizando estas medidas, la función criba sus resultados para devolver en un vector ordenado de mayor a menor calidad los puntos más prometedores dada una imagen. En la figura 4.11 podemos ver un ejemplo de una imagen donde se ha dibujado el contenido del vector que ha devuelto la función *goodFeaturesToTrack*, basada en el algoritmo de Shi-Tomasi.
  En el programa implementado, hemos hecho uso de una clase en OpenCV, cuyo algoritmo se basa en la búsqueda de puntos de interés utilizando Shi-Tomasi en los blobs detectados. La especificación utilizada es la siguiente:

```
GoodFeaturesToTrackDetector( int maxCorners=1000, double
    qualityLevel=0.01, double minDistance=1, int blockSize=3,
    bool useHarrisDetector=false, double k=0.04 );
```

  `maxCorners` – Número máximo de esquinas que devolver. Si el número de esquinas en la imagen es mayor que este parámetro, se recogen las de mejor calidad.

  `qualityLevel` – Parámetro que especifica el índice de calidad mínimo aceptado en las esquinas detectadas. Este índice se multiplica a la mejor medida de calidad de

esquina encontrada. Todo aquel punto que tenga una medida de calidad inferior a este resultado, es descartado.

`minDistance` – Mínima distancia euclideana posible entre las esquinas encontradas.

`blockSize` – Tamaño que tendrán las submatrices que se formarán alrededor de cada píxel de la imagen para el cálculo de las medidas de calidad, tal y como se ha explicado anteriormente.

`useHarrisDetector` – Parámetro que indica si usar un detector Harris[1][6] para obtener las medidas de calidad de cada píxel de la imagen, en detrimento del cálculo de autovalores de Shi-Tomasi.

`k` – Parámetro correspondiente a la ecuación que utiliza el detector Harris para cada píxel de la imagen: $dst(x, y) = \det M^{(x,y)} - k \cdot (\mathrm{tr} M^{(x,y)})^2$.

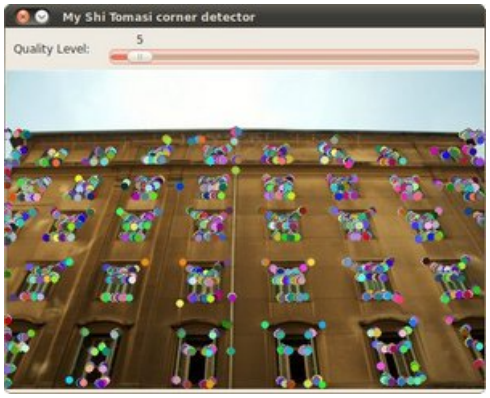


Figura 4.11: Resultado en la aplicación del algoritmo Shi-Tomasi

[1]Algoritmo alternativo para detectar esquinas sin usar los valores propios como medida de calidad.

- **SIFT:** *Scale Invariant Feature Transform[8]* es un algoritmo desarrollado para detectar puntos en una imagen que permanecen invariables a cambios de escala, traslaciones y rotaciones en la imagen donde aparecen.
  Este procedimiento utiliza filtros de Gaussianas para hallar los puntos de interés en una imagen y un filtrado por interpolación de píxeles cercanos por diferencia de Gaussianas para ajustar la posición de los puntos detectados. También se realizan diferentes cálculos para descartar puntos que se detecten de forma dudosa a lo largo de un borde o que sean sensibles al ruido en una imagen (y por tanto propensos a que cambien) y para el cálculo posterior de descriptores, que se explicará en el siguiente subapartado.
  En la librería OpenCV viene incluida una clase para detectar puntos de interés usando este sistema: *SiftFeatureDetector*. En la imagen 4.12 podemos ver un ejemplo de la detección de puntos de interés usando SIFT.

Figura 4.12: Detección de puntos de interés usando SIFT

- **SURF:** *Speeded Up Robust Features[15]* tiene el mismo cometido que el sistema de SIFT. De hecho, SURF está parcialmente basado en SIFT pero es más rápido que éste y más robusto frente al análisis de la transformación de una misma imagen. Su cálculo está basado en determinantes de matrices Hessianas en detrimento de las diferencias de Gaussianas, método usado por SIFT.
  Existe una clase en la librería OpenCV que posibilita detectar puntos de interés utilizando este sistema. La clase se llama *SurfFeatureDetector*. En la figura 4.13 se pueden comprobar los resultados de la utilización de SURF para la detección de puntos de interés.

Figura 4.13: Detección de puntos de interés usando SURF

- **ORB:** *Oriented FAST and Rotated BRIEF[11]* fue un algoritmo desarrollado por trabajadores de OpenCV para la detección de puntos de interés.
  Dentro de ORB, se usa FAST (*Features from Accelerated Segment Test*)[9], un detector de esquinas. Dicho algoritmo utiliza círculos que engloba 16 píxeles para determinar si un píxel es una esquina o no, comparando su brillo con un determinado número de píxeles contiguos al píxel que se está examinando. Si este píxel resulta ser el más brillante o el más oscuro de sus contiguos, es clasificado como una esquina. ORB utiliza FAST sin modificaciones, salvo que al final aplica una criba calculando la calidad de las esquinas detectadas y quedándose con los N mejores valores.
  La librería OpenCV posee la clase *OrbFeatureDetector*, la cual utiliza ORB para encontrar puntos de interés en una imagen. Un ejemplo de la detección de puntos de interés puede verse en la figura 4.14.

Figura 4.14: Detección de puntos de interés usando ORB

4.5.3. Cálculo de descriptores

Los descriptores de los puntos de interés son metadatos referentes a estos tales como orientación, tamaño... útiles en etapas posteriores, para posibles emparejamientos de puntos de interés de dos imágenes, por ejemplo. Se describen a continuación los algoritmos más importantes que calculan estos descriptores, incluidos todos en las librerías OpenCV:

- **SIFT:** Una vez obtenidos los puntos de interés, se toman los puntos colindantes a estos puntos de interés y se calcula la magnitud y dirección del gradiente. Con estos datos se realiza un histograma de estas direcciones combinado con la magnitud del gradiente. El mayor valor representado en el histograma nos indica la orientación del punto de interés. Utilizando estos datos, se organiza para cada punto bloques alrededor de sí de 16x16. A su vez, se dividen en sub-bloques de 4x4 y para cada uno se crea un histograma de orientaciones, que es lo que utiliza SIFT como descriptores para sus puntos de interés. La figura 4.15 explica muy bien lo expuesto en este punto.

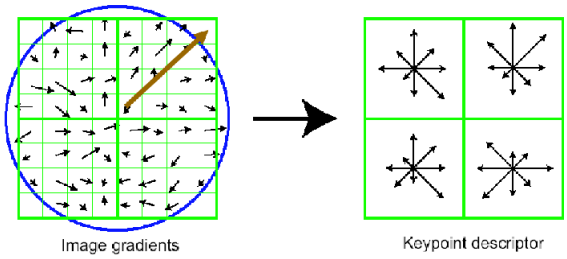


Figura 4.15: Descriptores calculados por SIFT

- **SURF:** Con el vector de los puntos de interés, SURF obtiene la orientación de la siguiente forma: Realiza los cálculos de la transformada de Haar tanto para la coordenada x como para la y del punto de interés que está tratando. Estos cálculos son sensibles a la escala que se esté tratando, la cual viene dada por el propio punto de interés. Una vez que obtenemos los resultados, los cuales están estructurados en vectores, uno por ordenada y otro por la abscisa, se suman y en este vector resultado se obtiene la orientación del punto de interés. Después, se divide el área colindante donde se ha encontrado el punto de interés en cuadrados de 4x4 píxeles y orientados según lo calculado anteriormente y se realiza en cada subárea otra transformada de Haar. Se obtiene finalmente de cada subregión un vector donde se consigue el resultado del sumatorio de los cálculos de la transformada en el eje de abscisas y en el de ordenadas, tal y como ilustra la figura 4.16. Esto se hace así para mantener robustos los cálculos aunque se reescale la imagen.

Nótese que el proceso que resulta en un vector de 4 valores en cada subregión se asemeja al proceso de calcular el histograma de orientaciones visto en SIFT.
Este sistema de obtención de descriptores es el elegido para la implementación del programa. Los resultados que arroja este método para el problema que nos ocupa son mejores que los demás que ofrece OpenCV.

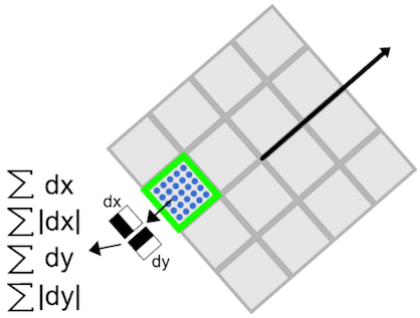


Figura 4.16: Orientación en SURF

- **ORB:** Este procedimiento se basa en una versión del algoritmo BRIEF (*Binary Robust Independent Elementary Features*) para el cálculo de los descriptores. El funcionamiento de BRIEF se basa en la idea de que los datos de una imagen se pueden clasificar a partir de la intensidad de cada uno de sus píxeles. Para ello, el algoritmo coge secciones de tamaño aleatorio de la imagen, suaviza la imagen aplicando un filtrado Gaussiano y realiza la criba píxel por píxel. Este método lo hace sensible a rotaciones de la imagen. ORB mejora esta tara de BRIEF, dando orientación a los puntos de interés, poniendo como centro a cada uno de su área colindante y calculando un vector de orientación a partir de la intensidad de los píxeles que le rodean. Para mejorar la independencia de estos cálculos a la rotación de la imagen se toma este área en forma circular. Después, cuando llega a la ejecución del algoritmo BRIEF mejorado, lo único que hace ORB es orientar BRIEF en base a los cálculos previos de la orientación de los puntos de interés.

4.5.4. Emparejadores

Una vez obtenidos los puntos de interés de dos imágenes y sus descriptores, se pueden usar emparejadores para relacionar puntos entre las dos imágenes. La idea es que, dados los conjuntos de descriptores de los puntos de interés, el sistema reconozca similitudes en los puntos de una y otra imagen.

Una cosa importante a tener en cuenta es que los emparejadores deben trabajar con conjuntos de descriptores obtenidos de la misma forma, ya sea con SIFT, SURF u ORB. Esto se debe a que cada método construye su vector de descriptores utilizando una norma distinta, ya que el tamaño de cada elemento del vector puede cambiar, dependiendo del proceso utilizado para obtener el conjunto de descriptores. Estas normas pueden ser la Euclídea (L1), la de Manhattan (L2) o la de Hamming.

La librería OpenCV aporta dos emparejadores:

- **BruteForceMatcher:** Este emparejador simplemente busca en cada descriptor del primer conjunto que se le pasa por parámetro, el descriptor del segundo conjunto más parecido en sus características, esto es, más cercanos entre los datos que alberga cada descriptor. Un simple algoritmo de fuerza bruta. En la figura 4.17 se puede ver cómo relaciona este emparejador los puntos de interés de dos imágenes con sus descriptores, obtenidas ambas cosas mediante SURF.

- **FlannBasedMatcher:** Este sistema está pensado para una colección grande de descriptores y llama a sus propios procedimientos de búsqueda de los descriptores más similares. Es un procedimiento pensado para ejecutarse más rápido que un procedimiento de fuerza bruta, como el anterior, en el caso de que tengamos un número de descriptores bastante elevado.

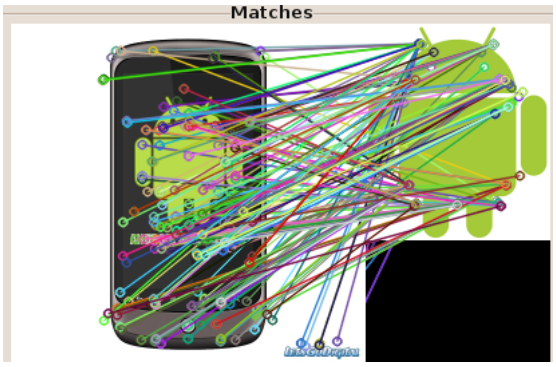


Figura 4.17: Puntos de interés obtenidos por SURF, emparejados

En el programa `Car_counter`, se utiliza *BruteForceMatcher* para realizar el emparejado de descriptores. Tras acabar este proceso, se obtiene el tamaño del vector resultante para conocer cuántas coincidencias entre el blob y el track de los dos fotogramas han habido, solo si estos dos elementos cumplen la condición de ser lo bastante similares en tamaño y estar lo suficientemente cerca el uno del otro.

4.5.5. Contador de vehículos

Junto con todas las funcionalidades de la aplicación, se incluye también un algoritmo propio de seguimiento, cuyo papel final es el de contador de vehículos. La función que contiene este algoritmo se llama `UpdateTracks` y su especificación es la siguiente:

```
void UpdateTracks (cv::Mat const &frameGray, cvb::CvBlobs const &blobs,
                   Tracks &tracks, double minDistance,
                   unsigned int minAreaDif, unsigned int maxtinactive,
                   unsigned int activeframes)
```

- `frameGray` - El fotograma actual del vídeo, en escala de grises.
- `blobs` - Estructura donde se almacenan los *blobs* detectados en el fotograma actual.
- `tracks` - Estructura donde se almacenan y donde se actualizarán, si procede, los *tracks*.
- `minDistance` - Distancia máxima a la que deben estar un *blob* y un *track* para empezar a ser considerados el mismo elemento.
- `minAreaDif` - Diferencia de área máxima entre un *blob* y un *track* para ser considerados el mismo elemento. Si se rebasa esta diferencia, se considera que no son el mismo elemento.
- `maxtinactive` - Máximo tiempo que se le permite a un *track* estar inactivo. Pasado este tiempo, se elimina si sigue estando inactivo.
- `activeframes` - Si un *track* pasa a estado inactivo, y el tiempo que ha estado activo es menor que el indicado en este parámetro, el *track* se elimina.

La idea del algoritmo implementado es la siguiente: dadas la imagen en escala de grises del fotograma actual del vídeo, los *blobs* detectados y los *tracks* almacenados hasta el momento, se recorre toda la estructura que almacena los *blobs* y compara cada uno con cada *track* de la otra estructura. Primero se calcula la distancia en píxeles que los separa desde los centros de cada elemento y la diferencia de sus áreas. Se comparan con los valores `minAreaDif` y `minDistance` pasados como parámetros a la función y si cumplen las dos condiciones, se hallan los puntos de interés y descriptores del *blob* en cuestión y se empareja con los descriptores del *track*. Se almacena el número de emparejamientos que ha habido entre los descriptores de los dos elementos.
Aquí terminaría la iteración del bucle, y el algoritmo seguirá comparando estos mismos

valores con todos los *tracks* de la estructura. Si en algún momento se encontrara un *track* que estuviera más cercano aún, se volvería a repetir la operación de emparejado de descriptores con este *track*, y sólo si el número de emparejamientos es mayor que con el anterior *track*, el *track* actual pasaría a ser el candidato más probable para actualizar su posición al del *blob*.

Para que un *track* sea identificado con un *blob* y le sea asignado un número, y por tanto, sea contabilizado, tiene que pasarse más de 4 fotogramas siendo relacionado con un *blob*. De esta forma, el mismo contador es útil para poder etiquetar a cada vehículo detectado en el vídeo y mostrarlo por pantalla con su identificador correspondiente, como ya se ha mostrado en la imagen 4.8.

Si un *blob* no es relacionado con ningún *track* al final, es convertido en un nuevo *track* y almacenado para las comprobaciones de los siguientes fotogramas. Cada *track* tiene almacenado en una estructura interna sus descriptores, calculados en su momento. De este modo, se aprovechan mejor los recursos y el coste computacional es menor.

Para calcular los puntos de interés de un *blob* o un *track*, se necesita la variable donde se almacena la imagen en escala de grises del fotograma actual. Usando las coordenadas del elemento del que se quieran obtener sus puntos característicos, se extrae de la imagen usando el constructor de `cv::Mat` para crear una Región de Interés, ésta se amplía con `cv::Resize` para aumentar la imagen de tamaño y se procesa para obtener dichos puntos.

4.6. Interfaz gráfica de usuario

Para que el usuario pueda interaccionar con la aplicación y ver los resultados en tiempo real es preciso diseñar y construir una interfaz fácil de manejar y que ayude a abstraer al usuario lo máximo posible de los pormenores del funcionamiento interno del programa, así como aportarle flexibilidad a la hora de modificar parámetros. Para ello se ha implementado una interfaz usando `Gtk` para el componente. La interfaz gráfica se divide en varios módulos y éstos aparecen según se vaya necesitando al hacer click en las diferentes opciones (Figura 4.18).

La interfaz da la posibilidad de mostrar al usuario cómo funcionan los cálculos intermedios utilizados para calcular blobs, tales como la segmentación de fondo, y el uso de varios métodos de realizarlo, si procede. Cada apartado puede mostrar varios de los subapartados detallados a continuación:

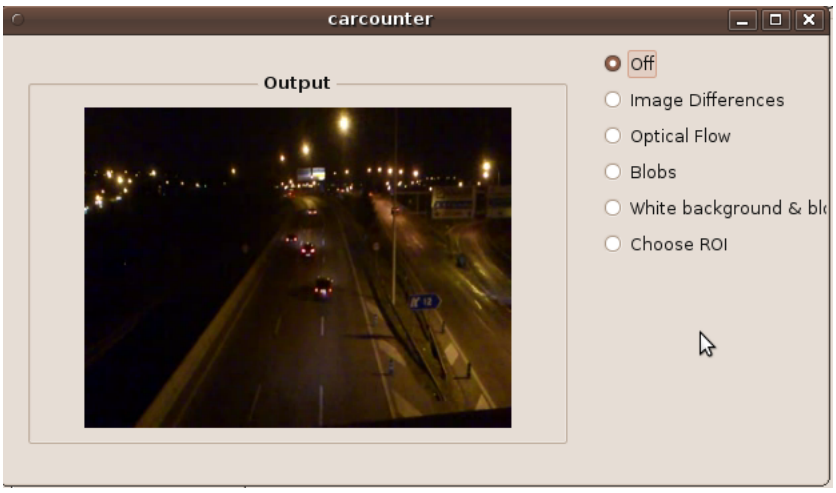


Figura 4.18: Interfaz de Car Counter

- En el subapartado de "*Background*", se utilizará la segmentación de fondo y por tanto aparecerán barras deslizantes referentes al cambio del valor α y al umbral mínimo de la diferencia por píxel entre la imagen y el fondo aprendido. También se muestra otra ventana donde se puede ver el resultado intermedio del aprendizaje del fondo y un botón de "*Capturar pantalla*" para capturar el frame actual y reiniciar así el aprendizaje de fondo.
- El subapartado "*Prev Frame*" solo tendrá la barra deslizante del umbral mínimo de diferencia ya que no utiliza aprendizaje de fondo y tan solo resta los valores, píxel por píxel, del anterior frame. La ventana donde se puede ver el aprendizaje de fondo está oculta si esta opción se activa.
- El subapartado "*Gaussian Mixture*" no requiere de parámetros determinados por el usuario, con lo que no aparecen barras pero sí la ventana donde se muestra el fondo aprendido y el botón "*Capturar pantalla*", del que ya hemos hablado en el primer punto.

Estos subapartados pueden verse al activar la opción "*Image Differences*", como puede verse en la figura 4.19.

4.6.1. Configurador de ROI

Para optimizar mejor nuestros resultados, es conveniente establecer ROIs (*Region Of Interest*) en la imagen a analizar. Se ha implementado un sistema en la interfaz que permite al usuario seleccionar y guardar una región rectangular personalizada que se utilizará en la función de detección de blobs, si hubiera alguna.

Figura 4.19: Subapartados relativos a la segmentación de fondo

La interfaz mostrará el recuadro azul cuando se trata de una selección temporal y rojo cuando está mostrando un ROI previamente guardado en memoria, tal y como se puede ver en la figura 4.20.

También se ha incluido dentro del código del GUI un pequeño sistema de seguridad para evitar que se seleccione ninguna coordenada fuera del rango de la resolución del vídeo. Si esto ocurre, internamente se deja seleccionado el mayor o el menor valor de la coordenada, dependiendo de si el usuario ha seleccionado algo por exceso o por defecto, quedando siempre dentro del rango de la resolución del vídeo y evitando errores en tiempo de ejecución.

4.6.2. Detector de blobs y seguimiento

La opción de detección de blobs, objeto principal del componente, aparte de mostrar la opción de elegir entre calcular la segmentación de fondo de manera exponencial o por mezcla de Gaussianas para la obtención de blobs en el paso posterior del algoritmo, también muestra el resultado tras el proceso de la segmentación de fondo, esto es, los objetos en primer plano. Si se ha seleccionado previamente un ROI, entonces este recuadro pasará a tener el tamaño de la región que hayamos configurado y se verán los objetos en primer plano solo de ese área.

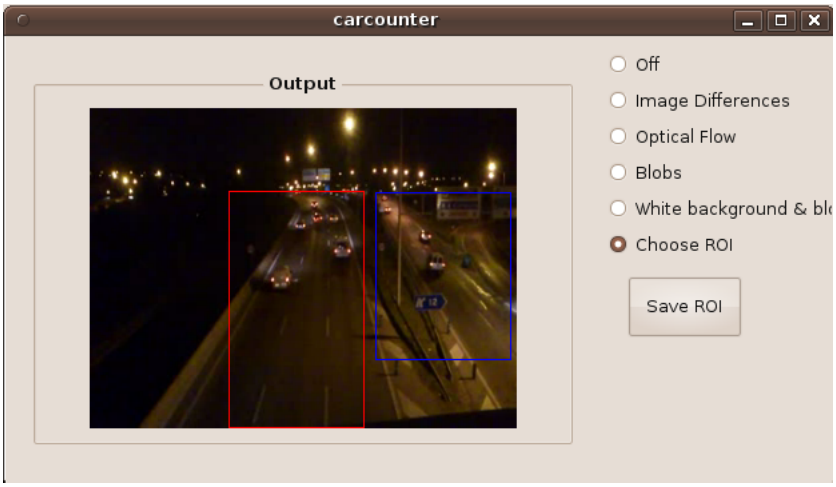


Figura 4.20: Interfaz para selección del ROI

Para poder comparar entre uno y otro, también se ha implementado una opción en el que tenemos la posibilidad de seleccionar el modo de seguimiento que se desee utilizar. En la opción "*cvBlob*", se usará el modo de seguimiento que nos proporciona la librería `cvBlob`. Si se elige en cambio "*Customized*", el componente utilizará el algoritmo de seguimiento que se ha desarrollado para este proyecto.

Observar en la imagen 4.21 que al utilizar segmentación de fondo con aprendizaje de fondo exponencial en los cálculos intermedios (en el momento de hacer la captura de pantalla está seleccionado el subapartado "*Background*"), en la interfaz también aparecen las barras deslizantes y el botón de "*Capturar Imagen*" de los que ya se ha hablado en el apartado 3.

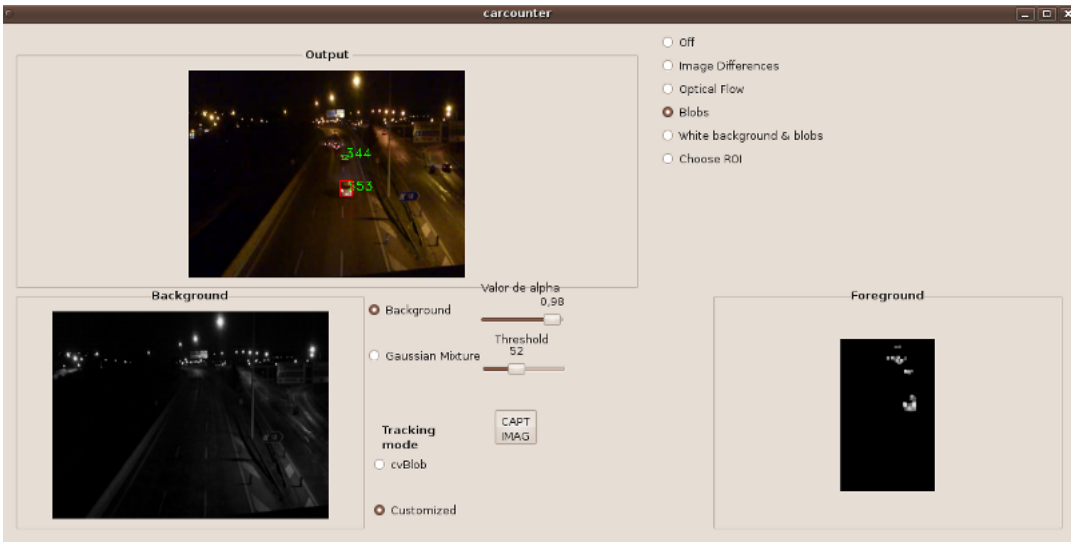


Figura 4.21: Interfaz en detector de blobs

Capítulo 5

Experimentos

En este capítulo se explicarán las diferentes pruebas que haremos al programa desarrollado y se recogerá su resultado. Se utilizarán 8 vídeos grabados en distintos entornos y condiciones de iluminación para comprobar la robustez y fiabilidad del algoritmo implementado, así como obtener el coste computacional del programa durante su ejecución. Una vez lo hayamos hecho, se tendrán los datos para juzgar cómo de buena es la solución que se ha propuesto para el problema expuesto.

Realizando estos experimentos, se busca que el programa reconozca los vehículos de la manera más exacta posible, los contabilice y les asigne un identificador único en forma de número, hasta que el vehículo se pierda de vista o salga del plano.

5.1. Entorno de pruebas y base de datos de vídeos de tráfico

Este Proyecto Fin de Carrera ha sido analizado y probado en un HP Compaq Presario V6500 con un procesador Intel Centrino Duo a 2.0 Ghz, 2 GB de memoria RAM, una tarjeta gráfica integrada Intel Extreme Graphics y Ubuntu 10.04.

También se ha utilizado las cámara de los móviles Sony Ericsson Xperia Neo V y Samsung Galaxy S3 grabando en HD 720p y el *software* FormatFactory para convertir vídeos a un formato y a una resolución que tolere el programa (MPG a 320x240) con el objeto de realizar pruebas con lo grabado por la cámara. En el periodo de aprendizaje de JdeRobot, también se utilizó la cámara Logitech QuickCam Pro 9000 trabajando de igual forma con una resolución de 320x240.

Las secuencias de vídeo fueron tomadas desde una posición fija sobre dos autovías en Madrid, en diferentes momentos del día y por la noche, para así conseguir distintas configuraciones de iluminación, tanto ideales como no ideales. También se recogieron grabaciones de cámaras de tráfico reales via *streaming* mediante la web de tráfico de Nevada[1], tanto por el día como por la noche, de la ciudad de Reno, en el Estado de Nevada, en Estados Unidos. De esta forma se incluyen en la batería de pruebas del programa vídeos con el formato, la resolución y la ubicación de una cámara de tráfico funcional. Todas las grabaciones tienen una duración de 2 minutos.

5.2. Metodología

Antes de iniciar cada prueba, se entrará al menú de selección de ROI para elegir la Región de Interés que mejor se adapte al vídeo en cuestión. De este modo, el algoritmo se centrará en la sección de carretera que interesa en el vídeo para el conteo de vehículos y minimizará los falsos positivos. Las coordenadas de los puntos del ROI quedarán registradas en el apartado de cada prueba. También se contabilizarán manualmente los vehículos que van apareciendo en el vídeo

A cada vídeo se le aplicará un aprendizaje exponencial de fondo, como método de segmentación de fondo. Es por ello que, dependiendo de las condiciones de iluminación de cada vídeo, los valores α y umbral mínimo de diferencia serán ajustados en cada prueba para optimizar los resultados del programa. El funcionamiento del aprendizaje exponencial de fondo, así como sus parámetros, se ha explicado en el apartado 4.3.3.

El ajuste de dichos parámetros ha sido establecida en cada caso mediante ensayo y error, buscando los mejores resultados para cada caso. Una vez fijados, durante la ejecución del programa en cada prueba, estos valores permanecerán constantes en todo momento. Los parámetros controlan la visibilidad de los objetos en movimiento en la imagen. Dependiendo de sus ajustes, se requiere mayor o menor cantidad de movimiento para que aparezca un objeto en la imagen resultante tras la segmentación de fondo, dependiendo también de la iluminación de cada vídeo.

Del mismo modo, para los procedimientos `cvLabel` y `UpdateTracks` también se han fijado los mismos parámetros para todas las pruebas. Para `cvLabel`, se han filtrado los blobs cuyo área fuera menor que 50 y mayores que 400. Para la función `UpdateTracks`, que se encarga del seguimiento de los blobs, realizará el seguimiento a los blobs cuya distancia entre fotogramas sea menor a 20 píxeles, que el área máxima de diferencia sea 300 y el blob esté activo como máximo 30 fotogramas.

Para medir y registrar el coste computacional del programa mientras se está ejecutando, se ha utilizado *Oprofile*[1]. Esta herramienta permite registrar la carga computacional de un programa concreto o del sistema entero, en una amplia variedad de parámetros del PC, a nivel de contadores de evento del procesador. Permite configurarlo para monitorizar varios de estos contadores, incluso por núcleo, y recoger estadísticas. También posibilita guardar e identificar varias sesiones distintas en el disco duro, si así se desea. Una vez analizado el rendimiento de un programa, puede mostrar los datos recolectados por pantalla, y ordenarlos bajo una amplia variedad de parámetros, mostrando sólo lo que queremos ver en cada momento. Por ejemplo, posibilita observar todos los símbolos utilizados por un proceso concreto. De este modo se puede saber qué función está utilizando más el procesador dentro de un proceso.

Los eventos del procesador que contabilizará *Oprofile* en todas las pruebas son `CPU_CLK_UNHALTED:30000`, para contabilizar el tiempo que un proceso está usando el procesador, y `L2_LINES_IN:30000` para contabilizar los accesos a la memoria principal que se hace desde la CPU (en concreto desde el nivel 2 de la memoria caché). La cifra que hay a continuación del nombre de los eventos indica a *Oprofile* que incremente su contador de muestreos cada vez que detecte que han habido, en este caso, 30000 eventos de un tipo concreto en el procesador.

5.3. Experimento con vídeo diurno

El primer vídeo con el que se comienza esta tanda de pruebas es con un vídeo diurno normal, en condiciones ideales de visibilidad y una óptima posición de la cámara: en la parte central de la carretera (Fig 5.1).

Figura 5.1: Vídeo diurno

[1]Web de *Oprofile*: `oprofile.sourceforge.net`

Durante la grabación del vídeo se contabilizaron de forma manual **115 coches** que pasan ante la cámara.

Se configura una Región de Interés cuyas coordenadas del vértice superior izquierda son (93,68) y las del vértice inferior derecha son (250,254).

Estableciendo como parámetros $\alpha = 0,92$ y el valor umbral a 56, se obtiene el conteo de **107 coches**. Un 93 % de precisión respecto a los resultados reales.

Se han observado algunos problemas durante el transcurso de las pruebas con este vídeo:

- Los resultados sufren una considerable variación, dependiendo dónde se seleccione la Región de Interés. Esto se debe a que el algoritmo tarda un poco en detectar el *blob* y completar el proceso de seguimiento y reconocimiento. Si la Región de Interés no es lo suficientemente alargada, deja sin contar algunos vehículos que detecta tarde.
- Cuando los coches se agrupan, el algoritmo detecta al grupo entero como un solo coche. Esto ocurre sobre todo cuando la agrupación está lejos de la cámara, como ilustra la figura 5.2. Afecta al conteo final de vehículos del algoritmo.

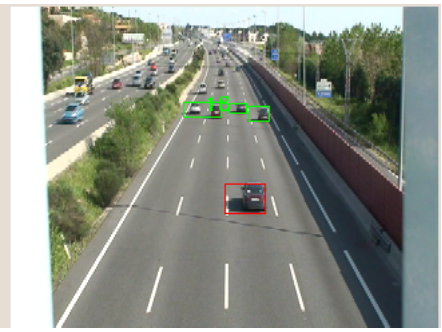


Figura 5.2: Agrupación de 2 vehículos detectados como un *blob*. (*Blob* 18)

- No se detectan del todo bien los vehículos grandes, como los camiones o los autobuses. Es un problema intrínseco del detector de *blobs* utilizado, ya que distingue partes inconexas del mismo vehículo y considera que son dos o más vehículos, variando la cuenta total. No es un problema que aparezca siempre, depende sobre todo de las sombras que arroje el propio vehículo o si éste aparece en la pantalla de forma oblicua o recta.

En cuanto a coste computacional, el nivel de carga de la CPU al ejecutar el programa ha sido del 48,7 % y el nivel de uso de la memoria ha llegado al 33 %.

5.4. Experimento con vídeo diurno con oclusiones parciales

En este vídeo se ha grabado el tráfico a la misma altura que el anterior, pero más desplazado hacia el arcén, como se puede comprobar en la figura . De este modo, se dan casos de oclusiones parciales entre los vehículos cuando se adelantan y podemos probar la eficacia del algoritmo en este escenario.

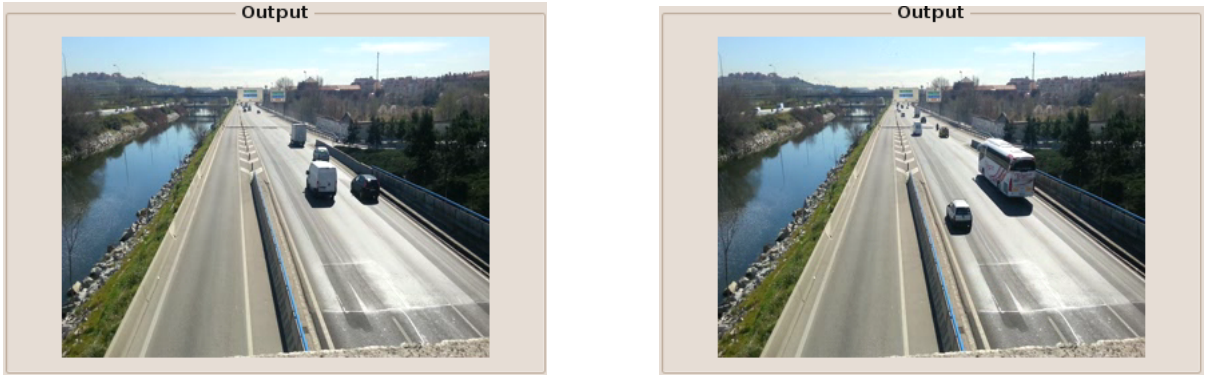


Figura 5.3: Fotogramas del vídeo diurno con oclusiones parciales

Se contabilizan de forma manual **104 coches** que capta la cámara en la región del vídeo que vamos a analizar.

Se selecciona una Región de Interés, cuyas coordenadas del vértice superior izquierda son (142,79) y las coordenadas del vértice inferior derecha son (319,239).

Se fijan los parámetros de la segmentación de fondo en $\alpha = 0,95$ y valor umbral en 92. Se obtienen al final del vídeo la cuenta de **95 coches**. En este vídeo, el programa ha alcanzado una precisión del 91 %.

Los problemas encontrados en este vídeo son prácticamente los mismos que en el anterior caso:

- Debido a las oclusiones parciales, el programa no contabiliza dos coches cuando uno está adelantando a otro. Lo contabiliza solo como uno. Es decir, en este vídeo, el problema con las agrupaciones de coche se acentúa un poco más.
- Los vehículos grandes tienen un problema añadido con la sombra que arrojan. En este vídeo, los vehículos grandes siempre se contabilizan por 2, ya que el programa detecta sus sombras y su techo como dos vehículos diferentes, como ocurre en la situación recogida en la figura 5.4. Esto también se debe a la diferencia de área

entre estos *blobs*, ya que los techos de estos vehículos son más alargados que la sombra que arrojan por detrás. Por lo tanto, se obtienen falsos positivos.

Figura 5.4: Furgoneta siendo detectada dos veces, en el *blob* 73 y en su sombra

- Por el brillo del sol, se advierte más fragmentación de *blobs* dentro del área de los vehículos, donde debería ser todo un *blob*, aunque no llega a afectar al cómputo final. Es debido al reflejo del sol en partes concretas de los vehículos.

Una vez expuestos estos problemas, se hace de manifiesto que la precisión calculada no es fiable debido a la proliferación de falsos positivos, aunque en realidad se estima que no se aleja mucho de ese porcentaje.

La carga de la CPU observada ha sido del 67 % mientras se ejecutaba el programa. La carga de uso de memoria ha sido del 54,2 %. El coste computacional ha sido más elevado que el caso anterior, ya que al haber mucha más iluminación, se detectan muchos más puntos de interés, y el algoritmo debe procesarlos.

5.5. Experimento con vídeo diurno con oclusiones parciales y sombras

En esta grabación se ha querido analizar el funcionamiento del sistema en las mismas condiciones que la sección anterior, solo que en estas imágenes la carretera está cubierta por sombras de una manera no uniforme, como se puede apreciar en la figura 5.5. Se quiere medir la robustez del algoritmo al tener que recorrer los vehículos un tramo con claroscuros.

En este vídeo vamos a contabilizar sólo los coches del carril derecho. Primero hacemos el conteo de forma manual. Se cuentan **60 coches**.

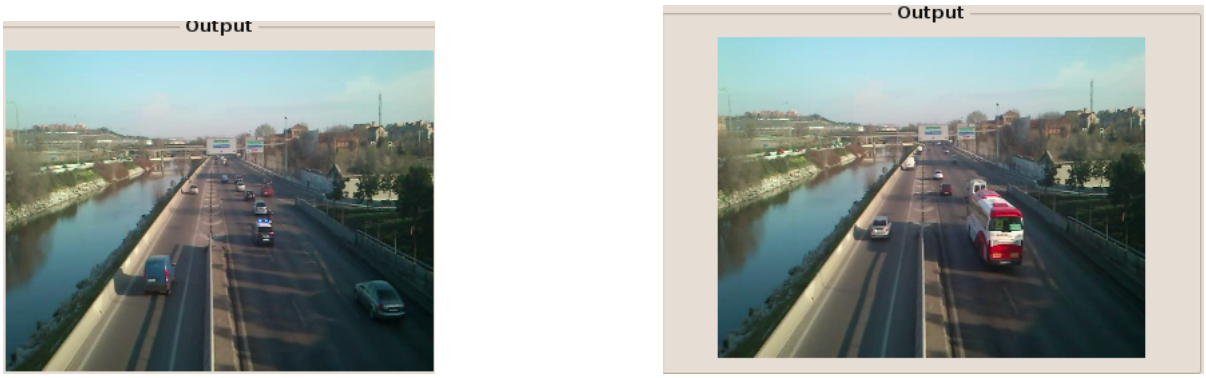


Figura 5.5: Vídeo con oclusiones parciales y sombras en el asfalto

Se elige una Región de Interés correspondiente al carril derecho. Se selecciona una con la esquina superior izquierda en las coordenadas (160,121) y la esquina inferior derecha en (319,239).

Se decide los valores de configuración para la segmentación de fondo, siendo $\alpha = 0,95$ y el umbral 31. Tras la ejecución del programa, éste contabiliza **47 vehículos**, lo cual indica una precisión en el conteo del 78 %.

Comprobamos que la precisión es menor en este caso, debido a los problemas que exponemos a continuación:

- Los claroscuros en la carretera provocan que el sistema pierda el seguimiento del vehículo detectado, al no cumplir el tiempo de seguimiento seguido necesario del *track* para contabilizarlo. Durante la segmentación de fondo, el contorno del vehículo se solapa con las sombras e imposibilita el seguimiento.

- El movimiento de las sombras sobre la carretera aumenta el número de puntos de interés a calcular, y por lo tanto, aumentan los cálculos que el programa tiene que efectuar. Estos cálculos ralentizan todo el análisis, el cual no permite completar el tiempo de seguimiento para contabilizar el vehículo. También confunden al programa cuando un vehículo atraviesa esta zona donde las sombras se mueven e interfiere con el proceso de detección de blobs. Una pequeña muestra de lo que se explica aquí se recoge en la figura 5.6.

- En este vídeo no se detectan las motos. Este tipo de vehículo tan pequeño pasa inadvertido en la detección de *blobs*, al fundirse con las sombras cuando recorre estos tramos.

- Tiene problemas para detectar también los vehículos grandes, desde autobuses hasta furgonetas (Figura 5.6b). Debido a la mezcla de problemas de los puntos anteriores,

no se cumple el tiempo de seguimiento mínimo para que este tipo de vehículos se contabilicen.

(a) Fragmentación de blobs

(b) Dificultad en detección de vehículos grandes

Figura 5.6: Las sombras en el asfalto provocan problemas en el seguimiento de vehículos

Tras examinar los anteriores puntos, podríamos afirmar que el mayor problema en este entorno es la confusión que le genera al programa el detectar como elementos de interés a las sombras en el asfalto. Los vehículos se mezclan con estas sombras, provocando finalmente que no se contabilicen correctamente en el cómputo final, durante la etapa de seguimiento.

La carga que el procesador ha soportado durante la ejecución del programa con este escenario ha sido del 57,7 %. La carga de la memoria principal ha sido del 44,4 %.

5.6. Experimento con vídeo nocturno

Con la aportación de este vídeo, se quiere comprobar cómo reacciona el programa ante un escenario con poca iluminación. La única iluminación la aporta el alumbrado público. La cámara está situada a la misma altura que en el resto de vídeos y la posición de la cámara es teóricamente la óptima para conseguir los mejores resultados: en la posición central. En la figura 5.7 se puede encontrar una captura de un fotograma de dicho vídeo.

Se contabilizan **103 coches** de forma manual en la grabación.

La Región de Interés seleccionada tiene el punto de su esquina superior izquierda en las coordenadas (104,74) y el punto de su esquina inferior derecha en (203,257).

Se configura la segmentación de fondo con los siguientes valores: $\alpha = 0,99$ y umbral 52. El conteo por el programa arroja una cifra de **96 coches**, con lo que se ha alcanzado en este caso una precisión del 93 %.

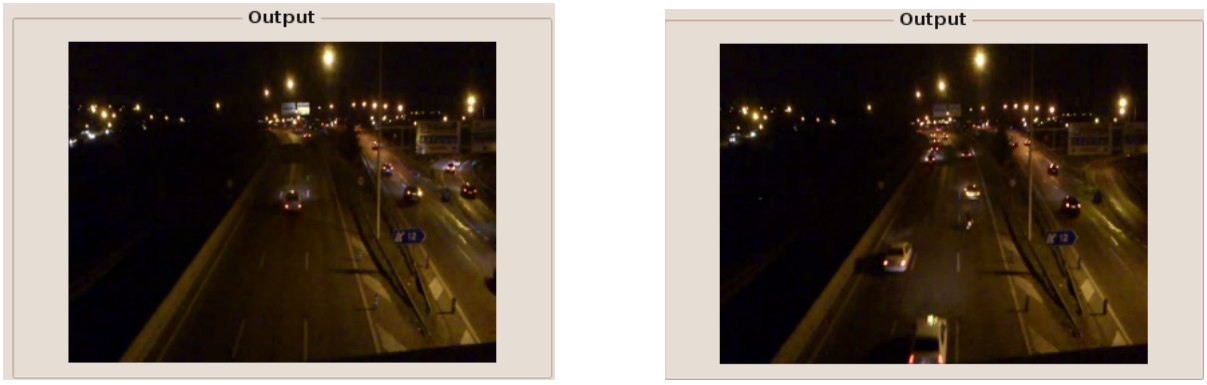


Figura 5.7: Escenario nocturno

Los problemas que surgen ante este escenario son distintos respecto a los diurnos:

- El programa tiene dificultad para detectar los coches de color oscuro, al confundirse con el color del asfalto. Tras realizar la segmentación, se observa cómo en la imagen que se usa para detectar blobs, los vehículos oscuros aparecen bastante difuminados. Es necesario establecer con cuidado los parámetros de la segmentación de fondo, para que el programa pueda detectar de la forma más óptima posible los coches de este tipo.
- Los vehículos pequeños, como las motocicletas, no se detectan en ningún caso. Se han probado diferentes configuraciones para la segmentación de fondo para paliarlo, pero en todas ellas se ha obtenido el mismo resultado negativo.
- En algunas ocasiones, la luz de los faros de los coches pueden hacer que se produzcan falsos positivos. Mediante el brillo en las señales de tráfico o en los arcenes que producen las luces de los coches, el sistema puede detectarlo como un vehículo más y contabilizarlo (Fig 5.8), aunque esto rara vez ocurre, ya que se necesita un tiempo mínimo de seguimiento para que el objeto se contabilice. En cambio, en los vehículos que tienen activos los faros de larga distancia y su luz es más intensa, sí suele aparecer con más frecuencia una mayor fragmentación de *blobs* y falsos positivos, ya que el detector puede confundir el brillo de las luces del coche, que puede ir muy por delante de él, con un vehículo.

Por contra, y respecto a este último punto, las luces de los coches, tanto los faros como los pilotos traseros, en la mayoría de los casos sirven de guía al programa para detectar al vehículo en cuestión en este tipo de entorno.

Mientras duraba la ejecución del programa, la carga del procesador fue del 61,8 % y el uso de la memoria fue de un 41,4 %.

Figura 5.8: El *blob* 15 es un falso positivo por el brillo de los faros del vehículo de detrás

5.7. Experimento con vídeo nocturno con oclusiones parciales

Este escenario posee una iluminación escasa, proveniente tan solo del alumbrado público. La única dificultad añadida a este vídeo es que la cámara desde la que se ha grabado se ha colocado en una posición no ideal, que permite oclusiones parciales en momentos como en el que un coche adelante a otro. La cámara está situada en una posición elevada, pero cercana al arcén. En la figura 5.9 se puede comprobar lo descrito.

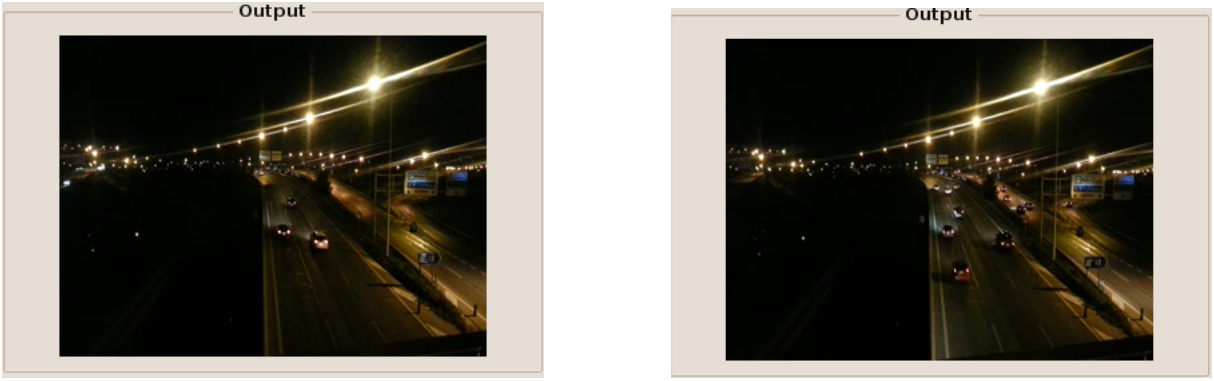


Figura 5.9: Vídeo nocturno con casos de oclusiones parciales

Tras contabilizar manualmente los coches que pasan durante la grabación, se observan **77 coches**.

Se selecciona como Región de Interés el área rectangular, comprendido entre la esquina superior izquierda en las coordenadas (154,116) y la esquina inferior derecha en (209,271).

Los parámetros de configuración para la segmentación de fondo que mejor han funcionado para este vídeo ha sido el que $\alpha = 0,96$ y el umbral es 36. El resultado final que arroja el programa es de **75 vehículos**. Es decir, obtenemos una precisión del 97 %.

Los problemas que se ha encontrado el algoritmo en este vídeo son los siguientes:

- El sistema puede confundir el reflejo de las luces de los faros con un vehículo y lo contabiliza como tal, dando falsos positivos. Por ejemplo, el reflejo de las luces en el arcén de la carretera o en el asfalto, esto último si el coche en cuestión tiene activas las luces de larga distancia y éstas son muy brillantes. La problemática es la misma que la explicada en el vídeo nocturno normal.
- En ocasiones, cuando dos o varios coches se agrupan, el programa los detecta como un solo vehículo, realizando mal el conteo al final de la ejecución del programa, tal y como puede verse en la figura 5.10. Desgraciadamente es un problema del detector de *blobs*, al considerar el grupo de coches como un único área de interés. Este mismo problema también ocurre con el vídeo diurno.

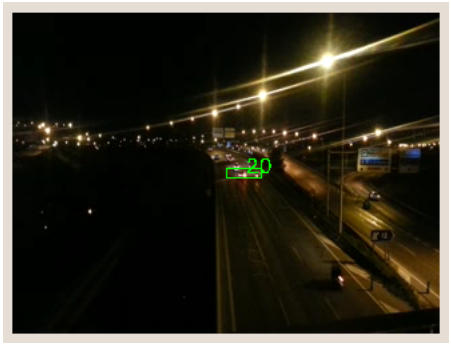


Figura 5.10: Agrupación de dos coches detectados como uno solo

- En algunos casos, se puede apreciar fragmentación de *blobs*, es decir, se reconoce como varios elementos algo que debería ser solo uno. La mayoría de las veces el sistema no llega a contabilizarlos, ya que no llegan a cumplir el tiempo mínimo de seguimiento para ser identificados. Pero sí que existen falsos positivos debidos a este problema. Por suerte, el número de veces que ocurre esto son muy pocas. Esto es debido a los reflejos que desprende el vehículo por la iluminación de las farolas y sus propios faros.

Aún teniendo algunos falsos positivos, la precisión que se ha alcanzado en este vídeo es bastante alta. En este vídeo, se hace evidente lo esencial que resulta encontrar los parámetros de configuración para la segmentación de fondo más apropiados para cada caso, y para cada vídeo en particular.

La carga del procesador durante la ejecución del programa con este vídeo ha sido del 64 %, y la carga del uso de la memoria ha sido del 50,3 %. No se aprecia mucha diferencia respecto al vídeo nocturno de la sección anterior.

5.8. Experimento con vídeo nocturno con oclusiones parciales e iluminación nula

La dificultad añadida en este vídeo es que, aparte de tener que lidiar con las oclusiones parciales, el algoritmo tendrá que desenvolverse en un entorno en el que la iluminación es prácticamente inexistente. Es de preveer que, durante la ejecución del programa, tanto los faros como los pilotos traseros de los vehículos jugarán un importante papel en su detección. El aspecto que presentará el vídeo puede verse en la captura del fotograma de la figura 5.11.

Figura 5.11: La única fuente de luz en este vídeo será la que emitan los vehículos

Para esta prueba, solo contabilizaremos los coches del carril derecho. Ignoraremos el carril de incorporación de la izquierda. Se contabilizan manualmente **135 vehículos**.

Se establece la Región de Interés en (160, 101) para las coordenadas del vértice superior izquierda y (314, 238) para las del vértice inferior derecha.

Se fijan los valores de configuración de la segmentación de fondo en $\alpha = 0,96$ y el umbral en 56. El programa arroja un resultado de **128 vehículos** contabilizados. Obtenemos, por tanto, una precisión del 94 %.

Los problemas observados durante la ejecución del programa son los siguientes:

- La problemática de la agrupación de *blobs* vuelve a hacer acto de presencia en este vídeo, por la existencia de las oclusiones parciales. En este caso, no parece haber un incremento en el número de casos de agrupación y podría decirse que se mantiene.
- Se dan casos de fragmentación de *blobs*. Debido al problema que trae consigo el mayor contraste en la imagen entre la luz de los vehículos y la oscuridad de fondo, esta es una de las consecuencias. El programa detecta reflejos u otros faros y los contabiliza

como vehículos. Comparándolo con los demás vídeos nocturnos, la ocurrencia de casos de fragmentación es mayor, como se puede comprobar en la figura 5.12.

- Debido al alto coste computacional que supone procesar cada imagen de este vídeo, la reproducción se ralentiza de tal manera que, en ocasiones, los fotogramas saltan sin procesarse, lo que afecta al seguimiento de los vehículos y a su conteo. Esto ocurre por la gran cantidad de elementos de interés que aparecen por pantalla, ya que la segmentación de fondo se vuelve muy sensible al movimiento dentro del vídeo, por el aumento del contraste entre las luces de los coches y la oscuridad del fondo.

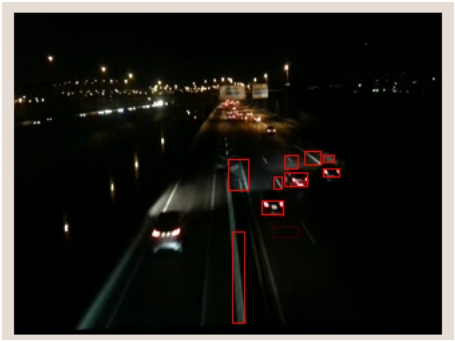


Figura 5.12: Elevada fragmentación de *blobs* durante la fase de detección de vehículos

Precisamente por el aumento de este contraste, se observa un incremento de los problemas en casi todos los ámbitos. El problema con los faros de los coches, ya descrito en anteriores secciones referente a los vídeos nocturnos, se acentúa. Se incrementan los casos de falsos positivos debido a que los reflejos de la luz de los faros son más potentes y aparecen más brillantes en las grabaciones en entornos oscuros.

En cuanto al coste computacional, las mediciones que se han realizado son reveladoras. La carga de la CPU es de un 75 % mientras se ejecuta el programa, mientras que la carga de la memoria es de un 53 %. El procesador aguanta una carga considerable de trabajo, y eso al final se resiente en los resultados del programa.

Es por ello que quizá, tras lo expuesto, la fiabilidad de los resultados que muestra el programa en este vídeo se resienta un poco. No obstante, la precisión del conteo del programa en este escenario se mantiene por encima del 90 %.

5.9. Experimento con vídeo nocturno de baja resolución

Como se ha explicado en capítulos anteriores, en países como Estados Unidos es posible acceder via web y en tiempo real, a las cámaras de tráfico de determinadas ciudades. Aprovechando esta coyuntura, se probará el funcionamiento del programa implementado con la grabación recogida por una cámara de tráfico real.

Figura 5.13: Fotogramas del vídeo nocturno grabado por una cámara de tráfico

El vídeo capturado es de baja calidad debido a la propia calidad de la emisión, como se puede apreciar en la figura 5.13. Es interesante probar el programa con este tipo de grabaciones, ya que hasta ahora se han utilizado vídeos grabados en alta definición. Se comprobará cuánto afecta la calidad del vídeo al resultado final que se obtenga del programa. El sistema también deberá detectar vehículos bajo la escasa iluminación que aporta el alumbrado público al entorno.

Contabilizamos manualmente **58 vehículos** en la carretera de la derecha, que es la parte seleccionada donde el programa contabilizará los coches.

Se configura, por tanto, la esquina superior izquierda de la Región de Interés en las coordenadas (154,104) y la esquina inferior derecha en (319,239).

Ha sido complicado encontrar unos valores de configuración adecuados para la segmentación de fondo de este vídeo, por problemas que más adelante se explicarán. Se establece un valor $\alpha = 0,99$ y un umbral a 108. Tras la ejecución del programa, se obtiene la cuenta de **39 vehículos**, un 67 % de precisión.

Se puede observar la diferencia significativa entre la cifra real de vehículos que pasan realmente y los que detecta el programa. Los problemas que se exponen a continuación afectan en gran medida al resultado obtenido:

- Uno de los problemas que surgen en los demás vídeos nocturnos, también aparecen aquí. Los vehículos con colores oscuros resultan más difíciles de detectar por el programa, ocasionando que no se contabilicen. Por fortuna, en la mayoría de los casos, el vehículo acaba siendo detectado por la luz de sus pilotos traseros o los faros del vehículo.

- Algunos vehículos se contabilizan dos veces. En contados momentos, el sistema detecta un vehículo por la luz de los pilotos traseros, realiza exitosamente el proceso de seguimiento y se le contabiliza para, a continuación, detectar el brillo de los faros delanteros de este mismo vehículo y delimitar un área de *blob* mucho mayor que la detectada inicialmente. Así, al superar la diferencia mínima de área entre los dos *blobs*, y a pesar de ser el mismo vehículo, el sistema lo detecta como dos elementos distintos. Este problema se manifiesta un número muy limitado de veces.

- En un par de ocasiones durante el vídeo, se puede apreciar que hay momentos en los que alguno de los faros de los vehículos que circulan en el otro sentido por el carril de la izquierda, deslumbra a la cámara y provoca que se desenfoque durante unos breves instantes. Durante estos momentos, el proceso de detección de vehículos deja de funcionar. Esto ocurre porque las propiedades de la imagen se ven alteradas durante los momentos en los que la cámara tarda en volverse a autoajustar y las imágenes que obtiene la segmentación de fondo no son las apropiadas para poder detectar ningún *blob*. Como es obvio, mientras ocurre esto, no se contabiliza ningún vehículo.

- La calidad de las imágenes del vídeo afectan a la detección de *blobs*. Se aprecia una mayor fragmentación de éstos en la etapa de detección, aumentando sobremanera el coste computacional. La fragmentación es debido al aumento de puntos de interés causado, entre otras cosas, por el reflejo del brillo de los faros de los coches. Tras la segmentación de fondo, también se aprecia en las imágenes resultantes que durante el movimiento de los brillos de los faros de los coches, éstos aparecen algo desdibujados debido a la calidad de las imágenes, y eso confunde al detector de *blobs*, que en vez de asignar un *blob* a todo el área, cree reconocer varios objetos de interés dentro del brillo de las luces de los vehículos, como ilustra perfectamente la figura 5.14. Esto parece un defecto propio del detector de *blobs* cuando trabaja con imágenes de baja resolución.

- El altísimo coste computacional, del que se hacía referencia en el punto anterior, provoca que se ralentice en exceso el procesamiento de imágenes. Esto al final desemboca en que al programa no le de tiempo a analizar todos los fotogramas del vídeo, y se salte alguno. La consecuencia de esto es que el proceso de seguimiento

se resiente y, con ello, el conteo de vehículos. Debido a la mayor fragmentación de *blobs* que existe en este vídeo, como se ha explicado en el punto anterior, el cálculo de puntos de interés de cada *blob* junto con sus descriptores incrementa exponencialmente el coste computacional. Esto se hace evidente en el momento en el que en la carretera se junta un grupo de unos 5 vehículos y se observa que el programa no puede procesar todos los fotogramas, ya que se advierten saltos en la pantalla de la interfaz donde se muestran los resultados del programa.

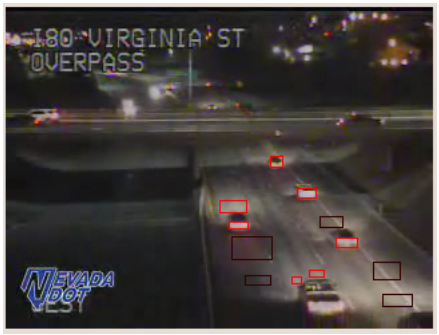


Figura 5.14: Elevado número de casos de fragmentación

Este último punto se hace evidente cuando echamos un vistazo a los porcentajes de carga de la CPU, que llegan hasta el 79,6 %. La carga de uso de la memoria también alcanza unos porcentajes bastante elevados: 52,8 %.

Sin duda uno de los problemas más importantes es que la carga de trabajo del procesador, resultante del procesamiento de imágenes, afecte finalmente al cómputo final del conteo de vehículos. Observando estos problemas en su conjunto, se puede entender mejor el bajo nivel de precisión que ha obtenido el programa en este vídeo.

5.10. Experimento con vídeo diurno de baja resolución

Al igual que en la sección anterior, se probará un vídeo captado por la misma cámara de tráfico, pero durante el día, como se puede ver en la figura 5.15. Esta grabación también posee una calidad inferior de imagen, así que se comprobará si presenta tantos problemas como el vídeo nocturno que anteriormente se ha analizado.

Se procede a contabilizar manualmente los vehículos de la carretera de la derecha, que será la Región de Interés escogida para que el programa la procese. Se observan **69 coches**.

Figura 5.15: Fotogramas de grabación diurna realizada por una cámara de tráfico

Se configura la Región de Interés en el lado derecho, cuyo vértice de la esquina superior izquierda está en las coordenadas (154,79) y el vértice de la esquina inferior derecha está en (319,239).

Se ajustan los parámetros de configuración de la segmentación de fondo en $\alpha = 0,92$ y umbral en 45. Tras el análisis de la grabación, el algoritmo arroja un resultado de **65 coches**. En este entorno se ha obtenido un 94 % de precisión en los resultados.

Es destacable la robustez con la que en este vídeo se desenvuelve el programa, no habiendo cometido casi ningún error. No obstante, se han encontrado algunos problemas:

- La cámara está en una posición que favorece la aparición de algunas oclusiones parciales entre los vehículos. Como en los anteriores vídeos en los que sucede esto, aparecen casos de aglomeración de *blobs*, como el de la figura 5.16. Esto ocurre cuando el detector de *blobs* detecta a las agrupaciones de 2 o más coches como uno solo. Como es evidente, afecta al resultado final que arroja el programa. En el vídeo, solo ocurre en los pocos casos en los que existen oclusiones, pero el programa falla en cada uno de ellos.

Figura 5.16: Dos vehículos detectados como uno, en el *blob* 58

- El algoritmo encuentra dificultades en la detección de vehículos grandes, como camiones o autobuses. Los resultados que se obtienen de la segmentación de fondo confunden al detector de *blobs*, reconociendo partes inconexas del vehículo como elementos independientes, dando como resultado la fragmentación de *blobs* dentro de lo que debería ser el vehículo. Cada uno de estos fragmentos es considerado como un coche y se contabiliza por separado o directamente no se llega a contabilizar, ya que al recorrer una poca distancia, el detector de *blobs* puede no volverlos a considerar como elementos de interés. Cuando esto ocurre, la sombra del vehículo pasa a ser un elemento de interés siendo detectado como un *blob*, pero el tiempo de seguimiento no es suficiente para contabilizarlo, y este se pierde.

Un hecho destacable durante la ejecución del programa es que puede verse que el algoritmo detecta las sombras de los vehículos de la carretera de la izquierda, que van en sentido contrario al de la carretera que se está analizando. Se puede apreciar cómo no se realiza el seguimiento de estas sombras, que se detectan en *blobs*, y sí a los vehículos de la carretera en la que se está realizando el conteo. De hecho, existe alguna ocasión, como en el momento recogido en la figura 5.17, en la que estos *blobs* se cruzan y no existe confusión por parte del programa. Esto simplemente prueba que el algoritmo funciona como debería, contabilizando solo los vehículos que se alejan, y no confundiendo los *blobs* cuando se cruzan, sin afectar al proceso de seguimiento y conteo que se está realizando.

Figura 5.17: Seguimiento correcto del coche, cruzándose con la sombra del camión

La carga que ha soportado el procesador y la memoria entra dentro de la normalidad. La carga de la CPU ha llegado hasta el 42 % y la carga del uso de memoria al 26,9 %.

5.11. Análisis

Una vez se han recogido todos los datos tras experimentar con el programa a través de 8 vídeos, se comentarán y compararán estos resultados.

Existe un problema común en todos los vídeos analizados: lo que se ha llamado "agrupación de *blobs*". Este problema consiste en que se detecta como un *blob* una agrupación de vehículos, cuando debería detectarse un *blob* por cada vehículo. Al ocurrir en todos los vídeos, no se puede achacar a una causa externa o a una particularidad ambiental en las grabaciones, tales como captar las imágenes de una posición en la que se recogen oclusiones parciales o una pobre iluminación. No obstante, es de reseñar que este error solo ocurre en los grupos de coches que están más alejados de la cámara. La clave parece estar en el detector de blobs escogido, que no parece funcionar bien en este tipo de situaciones. La solución que nos proporciona la librería `cvBlob` sufre este defecto de forma nativa.

En los vídeos diurnos, todos tienen la particularidad de la dificultad de realizar un buen seguimiento a los vehículos grandes. En este caso, el contorno que se obtiene de la segmentación de fondo para los camiones y autobuses no aparece muy definido y eso puede confundir al detector de *blobs*, que no lo ve como un solo vehículo, sino que detecta las partes más visibles del vehículo y lo hace como elementos independientes, como si fueran coches. Es un defecto propio del resultado que proporciona la segmentación de fondo, utilizando previamente aprendizaje exponencial de fondo, y que dependiendo de ciertos factores como la iluminación, se puede obtener poca definición en los contornos de los elementos móviles de las imágenes. Esto sumado a que el detector de blobs, al basarse en la búsqueda de contornos dentro de la imagen, cuando encuentra contornos dentro de los contornos los detecta y asigna a cada una de estas partes del vehículo un *blob*, fomentando la aparición de fragmentación de *blobs* y pudiendo afectar finalmente al resultado final del conteo que realiza el programa. La imagen de la figura 5.18 puede apoyar lo anteriormente expuesto.

Figura 5.18: Fragmentación de *blobs* en vehículos grandes y el resultado de la segmentación de fondo

Otra situación frecuente en los vídeos diurnos son las sombras que arrojan los propios vehículos u otros elementos situados a los lados de la carretera. Si estas sombras en los vehículos son muy alargadas o grandes puede provocar confusión en el detector de *blobs* y confundirlo con otro coche, contabilizándolo como tal. Aunque esto no ocurre en muchas

ocasiones y no arruina el procentaje de precisión en el conteo de vehículos. En cambio, las sombras que arrojan a la calzada elementos como los árboles sí que puede ser un problema grave, a efectos de coste computacional y, en última instancia, afectar gravemente a la precisión de conteo del programa. Si estos elementos se mueven por la acción del viento, por ejemplo, dejan rastros en la segmentación de fondo, y por tanto, afecta de modo reseñable al procesamiento de imágenes, como ocurre en el vídeo diurno con oclusiones parciales y sombras. En la figura 5.19 se puede encontrar una imagen de la segmentación de fondo de uno de los fotogramas del vídeo, en el cual se puede advertir pequeños píxeles sueltos que indican rastros de movimiento pertenecientes a las sombras en el asfalto, y prueba lo expuesto en este párrafo.

Figura 5.19: Rastros de movimiento de sombras en la segmentación de fondo

En cuanto a los vídeos nocturnos, la dificultad que más veces ha aparecido en este tipo de imágenes son los malos resultados en la detección de los coches de color oscuro. Este tipo de vehículos cuesta mucho detectarlos, ya que su presencia se camufla con el color del asfalto y la segmentación de fondo no logra construir un contorno lo suficientemente claro como para que sea detectado como un *blob*. Es por ello que la mayoría de las veces, cuando un coche oscuro es detectado y contabilizado, es por la luz de sus faros y la de sus pilotos traseros, aunque esto no sea una garantía. Aproximadamente la mitad de las veces, un vehículo tipo turismo de color oscuro pasa desapercibido para el programa, porque ni la luz de los faros ni la de los pilotos traseros es suficientemente brillante ni grande como para ser detectada como un *blob*.

Lo mismo pasa con los vehículos pequeños, como las motos. Debido a su tamaño, el detector de blobs no logra detectar el paso de una moto porque logra camuflarse con el color del asfalto, y la luz que emite tanto su faro como el piloto trasero no es suficiente para poder ser detectada. Desgraciadamente, esto ocurre con todos los vídeos nocturnos de tráfico y no parece que con el sistema implementado se pueda solucionar de algún modo. Si se modifican los ajustes de segmentación de fondo para que se detecten las motos, los falsos positivos empiezan a proliferar en los resultados, ya que sale reflejado todo reflejo y luz que pueda moverse o cambiar durante la grabación.

Precisamente el brillo de los faros de los vehículos en este entorno puede llegar a ser motivo de fallos en el programa. Estos brillos sirven, a menudo, de guía al detector de *blobs* para que pueda reconocer vehículos en las imágenes y contabilizarlos, cosa que de otra manera resultaría imposible. Pero otras veces, los reflejos y el brillo de los faros de larga distancia provocan que éstos sean contabilizados como si fueran un coche más, influyendo al final en el conteo total de vehículos. El único motivo por el que ocurre esto es que el detector de *blobs* no es capaz de distinguir cuándo lo que se mueve es un vehículo o no, limitándose a señalar agrupaciones de píxeles donde hay movimiento, como demuestran las imágenes de la figura 5.20. Por lo tanto y, en principio, es complicado evitar este problema con el sistema implementado.

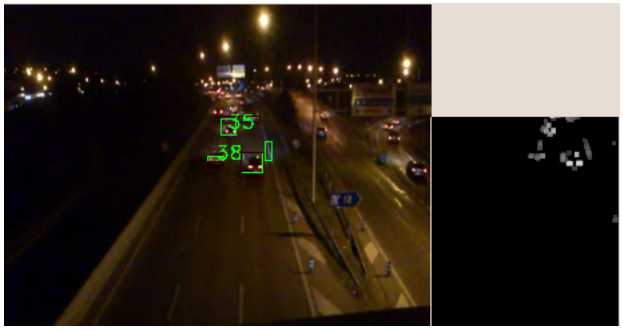


Figura 5.20: Los reflejos de los faros de los coches pueden generar falsos positivos

En las grabaciones capturadas de las cámaras de tráfico de Nevada, es evidente que la baja resolución de las imágenes ha afectado al procesamiento de éstas, sobre todo en las imágenes nocturnas. En este caso, la calidad del vídeo provoca una mayor pixelización en el resultado de segmentación de fondo, mostrando más actividad y, por tanto, obligando al programa a realizar más cálculos, procesando puntos de interés y descriptores (Figura 5.21). El alto coste computacional de estas operaciones provoca tal ralentización que algunos fotogramas del vídeo se quedan sin procesar, afectando al seguimiento visual de objetos que el programa está realizando. La solución pasaría por encontrar unos parámetros de configuración para la segmentación de fondo de tal forma que se detectaran menos elementos de interés, pero esto no ha sido posible. Cuando se han encontrado unos valores que no requerían a los procesos un gran coste computacional, el detector de *blobs* no detectaba a la mayoría de los vehículos, por no distinguir las luces de éstos en la oscuridad al encontrarse poco definidos. Es probable que la solución pase por requerir que las imágenes nocturnas deban tener una resolución mínima para conseguir unos resultados más cercanos a la realidad, en cuanto al conteo de vehículos.

Salvo dos excepciones, puede observarse que la precisión respecto al conteo de vehículos realizado por el programa y la cifra real de vehículos ronda entre el 92 % y el 97 %, un porcentaje muy elevado de acierto. Una de las excepciones es el vídeo nocturno de la

Figura 5.21: Pixelización elevada en el resultado de la segmentación de fondo

cámara de tráfico de Nevada, el cual alcanzó un 67 % de precisión, por los motivos ya expuestos. La otra excepción se corresponde con el vídeo diurno con oclusiones parciales y sombras, que tuvo una precisión en el conteo del 72 %. Esta precisión tan baja es a causa del movimiento de las sombras sobre la carretera, que a su vez provoca fallos en el proceso de seguimiento y en el de detección de *blobs*, afectando al conteo de vehículos. Las explicaciones de las causas de esta precisión están expuestas en la sección correspondiente de este experimento.

Respecto al coste computacional, no hay sorpresas. Las veces que el programa ha consumido más recursos ha sido cuanto más *blobs* se detectaban y, por lo tanto, más puntos de interés había que procesar. Las pruebas que más carga han aportado al sistema ha sido el vídeo nocturno de la cámara de tráfico en Nevada y el vídeo nocturno con oclusiones parciales e iluminación nula, por los motivos que ya se han explicado con anterioridad. El porcentaje de carga en los demás vídeos está entre los valores del 50 % y el 60 %, unos valores que permiten la ejecución del programa sin ralentizaciones graves.

Capítulo 6

Conclusiones y trabajos futuros

Tras proponer y describir una solución a los problemas planteados al inicio de este Proyecto, a continuación se ponderará si se han cumplido los objetivos propuestos al principio de esta memoria. También se sugerirán futuras líneas de investigación relacionadas con este trabajo.

6.1. Conclusiones

Tal y como se describió en el capítulo 2, el objetivo principal de este Proyecto Fin de Carrera es desarrollar un programa capaz de detectar y contabilizar los vehículos que pasan por una carretera bajo condiciones de visibilidad no ideales. Vistos los resultados de los experimentos en el capítulo 5, se puede afirmar que los hemos cumplido en su mayor parte. La precisión de los resultados que obtenemos en casi todos los vídeos no bajan del 92 %.

Este objetivo estaba dividido en 3 subobjetivos: detección de vehículos en la imagen, seguimiento visual de vehículos y su conteo y validación experimental.

Para el cumplimiento del primer subobjetivo, se ha integrado el programa desarrollado dentro de la plataforma JdeRobot, bajo la estructura de `basic_component` y ayudándose de `cameraserver`. También se ha apoyado en diversas funciones de la librería OpenCV. En el programa, se ha incluido el detector de *blobs* `cvLabel` de la librería `cvBlob` y se ha implementado varias alternativas para el segmentador de fondo, para el cual se ha utilizado una técnica de aprendizaje exponencial de fondo, diferencia directa de imágenes y mezcla de Gaussianas. Para el segmentador de fondo con aprendizaje exponencial de fondo se han elegido los mejores parámetros, mediante ensayo y error, para obtener imágenes

donde sólo aparecieran el contorno de objetos móviles lo más definidos posibles. Después se ha aprendido a obtener los mejores resultados posibles con la función `cvLabel`, tanto manipulando los parámetros de la función como filtrando los datos que se obtienen de él. Este subobjetivo se ha conseguido, detectando la gran mayoría de los vehículos y filtrando los blobs dentro de la imagen que no nos interesaban.

Una vez obtenidos resultados satisfactorios en el subobjetivo anterior, se procedió a implementar el seguimiento visual de los vehículos y el conteo. Para ello, se desarrolla un sistema de seguimiento de *blobs*, donde en cada *blob* se calculan puntos de interés y sus descriptores, ambos utilizando el sistema SURF de OpenCV, para proporcionar robustez al algoritmo y evitar fallos reforzando el proceso de seguimiento, evitando que dos coches puedan confundirse entre sí en la etapa de seguimiento. Además de los puntos de interés y los descriptores, también se compara fotograma a fotograma la distancia que separa a los centros de los blobs candidatos, la diferencia de sus áreas y el tiempo que están activos en pantalla. De este modo, se obtiene muy buenos resultados en el seguimiento de los vehículos en diferentes entornos, aunque no en todos. Especialmente en casos extremos de vídeos con baja resolución y visibilidad reducida.

En cuanto al tercer subobjetivo, se preparó una batería de pruebas de 8 vídeos con distintas características de entorno para probar la robustez del algoritmo y la fiabilidad de los datos que arroja. También se utilizó la herramienta *Oprofile*, para realizar mediciones en detalle de la ocupación del procesador y la carga de la memoria principal durante la ejecución. Las mediciones realizadas revelan que en 6 de los 8 vídeos la precisión de los resultados ronda entre el 92 % y el 97 %, demostrando así que aunque no haga un seguimiento perfecto de vehículos, sí arroja muy buenos resultados en la mayoría de entornos con visibilidad reducida. También hay que tener en cuenta los problemas que ha demostrado tener el programa desarrollado en las distintas pruebas, tales como fragmentación de *blobs* y la existencia de falsos positivos, por diversas razones, explicadas en el capítulo 5.

Respecto a los requisitos del programa, también puede decirse que se cumplen según lo fijado:

- El *software* es capaz de recoger datos de vídeo de una fuente cualquiera. Gracias al componente `cameraserver`, el cual está captando datos de vídeo de la fuente que sea que se le configure y se lo transmite al sistema desarrollado, el origen de los datos de vídeo es opaco para el programa. Los vídeos recopilados durante la fase de Experimentos están grabados desde una posición fija y enfocando al tráfico.
- Salvo en los casos extremos de visibilidad reducida, durante el procesamiento de las imágenes en tiempo real el coste computacional ha sido muy razonable y en

general no han existido ralentizaciones que pudieran afectar al sistema de conteo de vehículos.

- El lenguaje utilizado para implementar el componente ha sido C++.
- El componente se ha construido siguiendo la estructura del componente de JdeRobot `basic_component`, el cual tiene dos hebras de procesamiento independientes que se encargan de una funcionalidad específica, controlando una la parte operacional y la otra de los procesos relacionados con el GUI. También el código se ha separado en distintas partes, dependiendo de su funcionalidad: API, control y GUI.

6.2. Trabajos futuros

El trabajo en este Proyecto Fin de Carrera abre las puertas a otras vías de investigación, pudiendo mejorar el sistema desarrollado o dando pie a otros nuevos, apoyándose en este:

- Mejorar el sistema de detección de los blobs, basándose en sistemas tales como el flujo óptico o una mejora del detector de contornos, para aumentar la precisión de los resultados del programa.
- Una buena mejora sería añadir un estimador de velocidad a lo ya implementado. Por ejemplo, ampliar `Carspeed`[25] para poder usarlo con entornos de visibilidad reducida.
- Un mejor sistema para detectar vehículos en entornos de escasa iluminación, que permita detectar vehículos de pequeño tamaño, como las motos.
- Un autoajuste de parámetros para la segmentación de fondo o un juego de parámetros válido para un conjunto amplio de vídeos de entrada con distintas condiciones de visibilidad. La idea es minimizar la intervención del usuario en el procesamiento de vídeos.

Bibliografía

[1] *Página de la Nevada Department of Transportation.* `http://www.nevadadot.com`.

[2] *Web de GTK.* `http://www.gtk.org/`.

[3] *Birdwatch, un detector de vehículos que se saltan el semáforo en rojo muy fácil de implantar*, Tecnocarreteras, (2011).

[4] *Wiki del desarrollo de CarCounter.* `http://jderobot.org/Dgomezg-itis`, 2014.

[5] Antonio Pineda, *Aplicación de seguridad basada en visión.* Proyecto Fin de Carrera, Universidad Rey Juan Carlos, 2006.

[6] Chris Harris y Mike Stephens, *A combined corner and edge detector*, en Proceedings of the 4th Alvey Vision Conference, 1988, pp. 147–151.

[7] David Fernández López, Juan José Pantrigo Fernández, y Antonio Sanz Montemayor, *Bus Vigía: Análisis en tiempo real de la circulación para el control de un Carril-Bus.* Proyecto fin de carrera, Universidad Rey Juan Carlos, 2010.

[8] David G. Lowe, *Object Recognition from Local Scale-Invariant Features*, en Proceedings of the International Conference on Computer Vision, 1999, pp. 1150–1157.

[9] Edward Rosten y Tom Drummond, *Fusing Points and Lines for High Performance Tracking*, en IEEE International Conference on Computer Vision, vol. 2, 2005, pp. 1508–1511.

[10] Eloy Montero del Río, *Análisis de algoritmos para seguimiento visual de objetos*, Proyecto fin de máster, Universidad Rey Juan Carlos, 2014.

[11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, y Gary Bradski, *ORB: an efficient alternative to SIFT or SURF.* Willow Garage, 2011.

[12] Florida Department of Transportation, *SunGuide Software.* `http://sunguidesoftware.com/managing-traffic`.

[13] Francisco Rivas, *Algoritmo evolutivo para detección y seguimiento de personas en 3D con sensores RGB-D*, Proyecto fin de máster, Universidad Rey Juan Carlos, 2014.

[14] Fu Chang, Chun-Jen Chen, y Chi-Jen Lu, *A linear-time component-labeling algorithm using contour tracing technique*, Computer Vision and Image Understanding, (2003).

[15] Herbert Bay, Tinne Tuytelaars, y Luc Van Gool, *SURF: Speeded Robust Features*. 9th European Conference on Computer Vision, 8 Mayo 2006.

[16] Jianbo Shi y Carlo Tomasi, *Good Features to Track*. IEEE Conference on Computer Vision and Pattern Recognition, Junio 1994.

[17] Lawrence G. Roberts, *Machine Perception Of Three-Dimensional Solids*, Tesis de doctorado, MIT, 22 Mayo 1963.

[18] Luis Menéndez García, *Visual tracking of multiple objects in cluttered environments*, Proyecto fin de máster, Universidad Rey Juan Carlos, 2014.

[19] María Teresa de Diego, Laura Tordera, Miriam Gómez, Bárbara Fernández, Pablo Hyam, Rafael Fando, Miguel Ángel Sotelo, y Ignacio Parra, *Visión artificial para el control de tráfico en rotondas de la Ciudad 2020*, www.innprontaciudad2020.es, (2013).

[20] Redouane Kachach, *Clasificación automática de vehículos basada en visión*, Proyecto fin de máster, Universidad Rey Juan Carlos, 2010.

[21] Roberto Calvo, *Sistema distribuido de video-vigilancia basado en Android*, Proyecto fin de máster, Universidad Rey Juan Carlos, 2010.

[22] Sara Marugán Alonso, *Seguimiento visual de personas mediante evolución de primitivas volumétricas*. Proyecto fin de carrera, Universidad Rey Juan Carlos, 2010.

[23] A. Senior, A. Hampapur, Y.-L. Tian, L. Brown, S. Pankanti, y R. Bolle, *Appearance Models for Occlusion Handling*, Image and Vision Computing, (2006).

[24] Stefan Duffner y Christophe Garcia, *Pixeltrack: a fast adaptive algorithm for tracking non-rigid objects*. Proceedings of the International Conference on Computer Vision (ICCV), 2013.

[25] Víctor Hidalgo, *Detección visual de vehículos en la plataforma jdec*. Proyecto fin de carrera, Universidad Rey Juan Carlos, 2008.

[26] Zoran Zivkovic, *Improved adaptive Gaussian mixture model for background subtraction*. International Conference Pattern Recognition, 2004.

[27] Zoran Zivkovic y F. van der Heijden, *Pattern Recognition Letters*, 2006, ch. Efficient Adaptive Density Estimation per Image Pixel for the Task of Background Subtraction, pp. 773–780.