**UNIVERSIDAD DE ALCALÁ**

**Escuela Politécnica Superior**

**INGENIERÍA DE TELECOMUNICACIÓN**

Universidad de Alcalá

**Trabajo Fin de Carrera**

# SISTEMA DE MONITORIZACIÓN Y CONTROL DE TRÁFICO EN CARRETERA

Pablo Fernández Alcantarilla
Diciembre 2006

**UNIVERSIDAD DE ALCALÁ**

**Escuela Politécnica Superior**

**INGENIERÍA DE TELECOMUNICACIÓN**

Universidad de Alcalá

**Trabajo Fin de Carrera**

# SISTEMA DE MONITORIZACIÓN Y CONTROL DE TRÁFICO EN CARRETERA

**Autor:** Pablo Fernández Alcantarilla

**Director:** Dr. D. Miguel Ángel Sotelo Vázquez

**TRIBUNAL:**

| | |
|---|---|
| **Presidente:** | D. Pedro Revenga de Toro. |
| **Vocal 1º:** | D. Ricardo García López. |
| **Vocal 2º:** | D. Miguel Ángel Sotelo Vázquez. |
| **CALIFICACIÓN:** | ..... |
| **FECHA:** | ..... |

*No dejes que tu fuego se extinga, chispa a chispa irremplazables, en los pantanos deshauciados de lo incompleto, del todavía no, del absolutamente no.*

*No dejes que perezca el héroe que hay en tu alma, en una frustración solitaria por la vida que merecías, pero nunca has podido alcanzar.*

*El mundo que anhelas puede conseguirse, existe, es real, es posible, y es TUYO.*

*Ayn Rand.*

# Agradecimientos

Me gustaría dar las gracias al profesor Dr. D. Miguel Ángel Sotelo por haberme dado la oportunidad de realizar este trabajo fin de carrera y por toda la ayuda recibida para que pudiera completar mi formación académica en Suecia así como por el tiempo e interés dedicado en la realización del presente trabajo.

También quisiera tener unas palabras de agradecimiento para Daniel Pizarro y Jesús Nuevo, por su gran ayuda, sus sugerencias y su colaboración en el desarrollo de este proyecto. No me puedo olvidar de agradecer al profesor D. Göran Salerud y D. Fernando Seoane la ayuda que me dieron para poder estar en Suecia y finalizar mi proyecto aquí.

Sin duda, a las personas que más tengo que agradecer son mis padres y mi hermano, por todo el apoyo que me han dado durante todos estos años de estudio, ya que sin su ayuda y comprensión no hubiera sido posible llegar hasta aquí. También agradecer su apoyo e interés al resto de mi familia, tíos, primos y mi abuela.

Especialmente quisiera dedicar este trabajo a mi abuelo dónde quiera que esté. Siempre confiaste en mí, siempre me preguntabas que cuándo acababa la carrera. Ya la he acabado, por unos meses no has podido verlo, no me pude dar más prisa en acabarla. Va por ti, allá dónde estés, disfrútalo.

No podría olvidarme de una chica que hace como 5 años cogió a un chaval lleno de ilusiones y le enseñó la universidad. Han pasado 5 años desde entonces, y esa misma chica realizó la matrícula de este proyecto y lo llevó a encuadernar. Gracias Marta.

Quisiera agradecer el apoyo y la compañía de todos mis amigos y compañeros de clase con quiénes he compartido muy buenos momentos, fiestas, viajes, autobuses, exámenes y demás durante estos 5 años. En especial quiero dar las gracias a Roberto por ser mi compañero de fatigas en tantas cosas, a la gente del Laboratorio 22.4 de Electrónica, y a los pesados vecinos del laboratorio de al lado. Gracias a todos por el buen tiempo que hemos pasado juntos.

También quiero acordarme de toda la gente que he conocido en Suecia, que con sus comentarios y ánimos me han ayudado a finalizar el proyecto, además de pasar unos meses inolvidables con todos ellos.

Por supuesto que tampoco puedo olvidarme de mis amigos de toda la vida, Sergi, Java, Kepa y Nieves por su amistad, por todo el apoyo recibido durante todo este tiempo y por estar siempre ahí en los malos y en los buenos momentos.

Por último quisiera agradecer a todos los buenos profesores de los que he tenido el privilegio de aprender porque gente como ellos hace que estudiar tenga sentido.

Gracias a todos.

*N*oviembre 2006, Linköping, Suecia.

# Índice general

# Índice de figuras

# Parte I

# Resumen

# Resumen

El objetivo de este proyecto es el desarrollo de una aplicación capaz de detectar y clasificar vehículos, además de realizarles un seguimiento a partir del procesado de vídeos de secuencias de tráfico diurnas.

Para tal propósito, se colocó una cámara en un puente para poder captar diferentes secuencias de vídeo para su posterior procesado. Por lo tanto, para el desarrollo del proyecto se han utilizado vídeos grabados previamente, aunque el sistema es capaz de procesar nuevos vídeos así como de trabajar en tiempo real.

El entorno de desarrollo elegido fue el sistema operativo ***Linux***. Se utilizaron las librerías de código libre para visión por computador ***OpenCV***. Para el desarrollo de la interfaz de usuario se utilizó el programa ***Glade***.

El procesado de las imágenes de vídeo, consta fundamentalmente de 5 niveles que son:

1. Resta de fondo.
2. Clustering.
3. Tracking *(Filtro de Kalman)*.
4. Modelado 3D - 2D.
5. Visualización de resultados.

*Palabras Clave: Resta de Fondo, Clustering, Tracking, Filtro de Kalman, Modelado 3D.*

# Abstract

The purpose of this project is the development of an application able to detect and classify vehicles and also a tracking from processing road daylight traffic video sequences.

For such intention, a camera in a bridge was placed to be able to record different sequences from video for its later processing. Therefore, for the development of the project videos recorded previously have been used, although the system is able to process new videos as well as to work in real time.

The project was developed in ***Linux***. Computer Vision libraries ***OpenCV*** were used to implement the algorithms. Also ***Glade*** was used for the user interface design.

The video processing, consists fundamentally of 5 levels that are:

1. Background Substraction.
2. Clustering.
3. Tracking *(Kalman Filter)*.
4. 3D - 2D Modelling.
5. Visualization of results.

*Keywords: Background Substraction, Clustering, Tracking, Kalman Filter, 3D Modelling*.

# Parte II

# Memoria

# Capítulo 1

# Introducción

Actualmente los accidentes de tráfico son una de las principales causas de muerte en España. Esto se debe en la mayoría de las ocasiones a falta de preparación de los conductores así como a despistes. Durante los últimos años, se está trabajando mucho en aumentar la seguridad y la fiabilidad de los vehículos así como el uso de otros dispositivos que facilitan la conducción.

En los períodos de vacaciones normalmente podemos ver muchas campañas de la ***Dirección General de Tráfico*** intentando sensibilizar a la gente sobre el problema de conducir indebidamente y sus consecuencias.

Cuando estamos conduciendo, una de las informaciones más importantes que nos pueden dar, es cómo se encuentra el tráfico en una determinada vía o carretera. Y por otro lado a la ***Dirección General de Tráfico*** le interesa recolectar datos sobre el volumen de tráfico en las carreteras.

Con este proyecto, se intenta conseguir una aplicación capaz de lograr el conteo de vehículos en una determinada vía y que realize a cada uno de los vehículos detectados un seguimiento durante unos pocos metros, y todo esto con ***una sola cámara de vídeo***. Este será el objetivo principal, sin embargo a pesar de que el número de vehículos es una valiosa información, podemos obtener información muy importante como puede ser la velocidad de cada vehículo, así como distancias entre los vehículos, ampliando de esta manera las aplicaciones del proyecto.

Posteriormente, toda esta información puede ser utilizada por otros sistemas de información como ***GPS***[1] o de detección como ***RADAR***[2], así como la densidad de vehículos, le puede servir a tráfico para realizar estudios sobre el nivel de congestión de las vías a lo largo del día.

[1] Global Positioning System
[2] Radio Detection and Ranging

En este proyecto, se ha trabajado exclusivamente en la ***detección y seguimiento de vehículos durante el día***. Por lo tanto, forma parte de un proyecto global que conlleva otra serie de proyectos como puede ser la detección durante la noche en la cuál, debido a las condiciones de luminosidad, la detección se realiza principalmente fijándose en las luces de los faros de los coches.

La memoria del proyecto para lograr una explicación detallada del mismo, está estructurada de la siguiente manera:

1. **Estado del Arte:** Se comentarán otros proyectos comerciales así como artículos científicos que han servido de pequeña base e inspiración para el desarrollo del proyecto.
2. **Entorno de Desarrollo:** Se analizará la plataforma de desarrollo utilizada ***Linux***, así como las librerías y programas necesarios para el correcto funcionamiento del proyecto.
3. **Descripción del Algoritmo:** Se describirá las distintas partes que componen el algoritmo del proyecto, realizando un estudio teórico de cada una de las partes o niveles.
4. **Estructura de Programación:** Se intentará a modo de diagrama de bloques explicar el desarrollo de los algoritmos sin entrar en cuestiones referentes al código del programa.
5. **Resultados:** Se analizarán los resultados obtenidos en cada uno de los vídeos bajo test con los que se ha trabajado.
6. **Conclusiones y Futuros Mejoras**: Se exponen las principales conclusiones obtenidas después de analizar los resultados y se dan una serie de ideas para futuras mejoras en la aplicación.

Finalmente, se incluirán fuera del apartado de la **Memoria**, los siguientes capítulos referentes al proyecto:

- Presupuesto del Proyecto.
- Pliego de Condiciones.
- Manual de Usuario.
- Bibliografía.

# Capítulo 2

# Estado del Arte

El uso de sistemas de monitorización y control de tráfico en carretera es en estos momentos, uno de los principales puntos de trabajo para detectar posibles congestiones de tráfico y servir de ayuda a otros tipos de servicios más extendidos como pueden ser el ***RADAR*** o el ***GPS***.

Actualmente, en España no existe ningún proyecto comercial basado en visión artificial capaz de realizar este control de tráfico. Los sistemas actuales de conteo de vehículos son más caros y de mayor dificultad de implementación ya que suelen ser algún tipo de sensores colocados debajo de la carretera como pueden ser los ***Lazos Inductivos***. Por lo tanto el mantenimiento de estos últimos sistemas es bastante más caro que el de una simple cámara de vídeo.

A nivel internacional sí que existen proyectos comerciales como ***Traficon*** o ***Citilog***, así como numerosos artículos científicos en diferentes publicaciones que han constituido una base importante en el desarrollo de este proyecto.

De manera general, las ventajas que presentan estos tipos de sistemas basados en visión sobre los sensores comerciales, son las siguientes:

- Solución más económica que el uso de sensores.
- Mantenimiento menos costoso.
- Solución reconfigurable mediante software.
- Ofrecen una mayor variedad de datos de tráfico.
- Aplicaciones de control de tráfico interrelacionadas como detección de incidentes, conducción en sentido contrario...
- Acceso a las grabaciones de vídeo.

## 2.1. Sistemas Comerciales

### 2.1.1. Sensores

Existen diversas técnicas y procedimientos para poder realizar el conteo y la clasificación de vehículos (en función del número de ejes) con sensores a nivel de calzada o introducidos en el pavimento. Disponen de una inteligencia menor que los sistemas basados en visión artificial, pero sin embargo su uso está más extendido aunque en declive con el paso del tiempo y el avance en la tecnología.

Dependiendo del tipo de aplicación, del tipo de tráfico (tráfico libre, arranque y parada, etc) y de la precisión que se requiera, se utilizará un tipo de sensor u otro. Las técnicas más utilizadas son:

- **Manguera neumática:** Son sensores de eje que detectan el paso del vehículo debido al cambio de presión que se genera. Pueden contar y clasificar, pero sólo para uso temporal y con tránsitos fluidos. Cuando la manguera atraviesa varios carriles debe estar protegida en los carriles en los que no se esté realizando la medida. También es importante cuidar bien la fijación de la manguera a la calzada para evitar que los vehículos la rompan o la arrastren.

Figura 2.1: *Manguera Neumática*

- **Lazos inductivos:** No son sensores de ejes. Se pueden utilizar en forma temporal o permanente, siendo esta ultima la más normal. Detectan el paso del vehículo por variación de la masa magnética sobre el lazo. No detectan con detalle el eje, de modo que no pueden diferenciar el eje doble o el eje triple de un eje sencillo. Son económicos. Permiten clasificar vehículos, pero sin precisar exactamente el número de ejes.
- **Sensores piezoeléctricos:** Son sensores de ejes. Un par de sensores piezoeléctricos colocados a una distancia conocida permiten calcular la velocidad del vehículo y clasificarlo con precisión. Se utilizan normalmente junto con un lazo inductivo situado entre ellos para poder separar unos vehículos de otros, en configuración piezo-lazo-piezo. También pueden utilizarse dos lazos inductivos con un sensor piezoeléc-

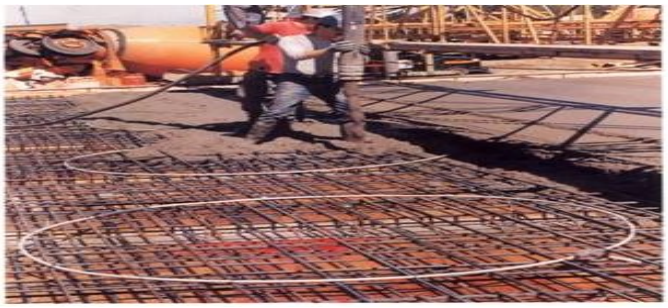

Figura 2.2: *Lazos Inductivos*

trico en configuración lazo-piezo-lazo, pero la primera (piezo-lazo-piezo) es la que proporciona mayor precisión. Detectan el paso del vehículo en base a la carga eléctrica que se genera en el material piezoeléctrico cuando es pisado por una rueda. Su campo de aplicación va desde 1 Km/h hasta 180 Km/h.

- **Sensores en base a cables de fibra óptica:** Son sensores de ejes que detectan el paso del vehículo por la variación de la conductividad óptica de un cable de fibra óptica pisado por la rueda del vehículo. Su utilización es análoga a la de los sensores piezoeléctricos, pero cubriendo velocidades desde 0 Km/h, por lo que son los únicos sensores que permiten clasificar en base al número de ejes en situaciones de circulación a vuelta de rueda, arranque-parada, etc...

### 2.1.2. Citilog

Citilog es una compañía con bases en Estados Unidos y Europa, siendo uno de los líderes en soluciones de visión artificial para la industria del transporte. Citilog desarrolla diversos proyectos y distribuye comercialmente sistemas de detección de incidentes para autopistas, puentes, túneles e intersecciones.

Sus principales áreas de trabajo son:

- Detección automática de incidentes.
- Vigilancia de vídeo avanzada.
- Seguridad.
- Almacenamiento de datos de tráfico (Traffic Data Collection).
- Control de intersecciones.
- Detección con cámaras PTZ[1].

[1] Pant Till Zoom

Actualmente, más de 300 autopistas, puentes y túneles están equipados con sistemas de detección de incidentes basados en visión desarrollados por Citilog a lo largo del mundo, convirtiéndose casi en un tipo de estándar en el campo de los Sistemas Inteligentes del Transporte. Sus sistemas han sido probados con éxito en prestigiosas infraestructuras tales como los 4 túneles de Manhattan en Nueva York o el puente Millau en Francia.

Figura 2.3: *Citilog Video Detection Systems*

Con los detectores de tráfico de Citilog, se puede obtener una buena solución para obtener datos sobre la velocidad de los coches así como la densidad de tráfico existente en una determinada autovía.

El sistema utiliza diferentes algoritmos en procesado de imágenes para extraer automáticamente del vídeo los datos de tráfico deseados. Primero se utilizan algoritmos de reconocimiento de objetos para posteriormente realizarles un tracking sobre un área de trabajo determinada sin límites de carriles o de velocidad de vehículos. Los datos pueden ser almacenados y posteriormente enviados a algún centro de supervisión de tráfico mediante distintos tipos de conexiones (serie, IP o wireless).


Figura 2.4: *Citilog Trafic Data Collection*

### 2.1.3. Traficon

Los detectores Traficon ofrecen datos de tráfico (volumen, velocidad, ...) que posteriormente pueden ser usados para otros fines tales como:

- **Detección automática de incidentes:** Rápida detección de vehículos detenidos o conductores suicidas, aumentan la velocidad de intervención y de esta manera salvan vidas.
- **Control de flujo:** La congestión dentro y a través de las grandes áreas metropolitanas continúa aumentando y limitando la movilidad. El control exacto del promedio de la velocidad de flujo ayuda a distinguir diferentes niveles de servicio (p.ej. fluido, denso, congestionado, atascos).


Figura 2.5: *Traficon Video Detection*

Figura 2.6: *Traficon Trafic Data Collection*

## 2.2. Artículos Académicos

Se van a comentar 2 artículos acádemicos publicados en el IEEE, que constituyeron un buen punto de partida para la realización del proyecto fin de carrera. No obstante, para el desarrollo del proyecto no se ha seguido ningún artículo determinado, si no que se han tomado ideas y detalles de diversos artículos.

### 2.2.1. Tracking All Traffic

**Autores:** *Rita Cucchiara, Massimo Piccardi y Paola Mello.*

Este artículo presenta un sistema capaz de detectar vehículos en escenarios de tráfico urbano a través de visión. Divide el desarrollo entre procesado de imagen de bajo nivel (extracción de los datos bajo diversas condiciones de iluminación) y el módulo de alto nivel (con mayor inteligencia y capaz de detectar los vehículos). La arquitectura del sistema con los dos módulos, la podemos ver en la figura 2.7.

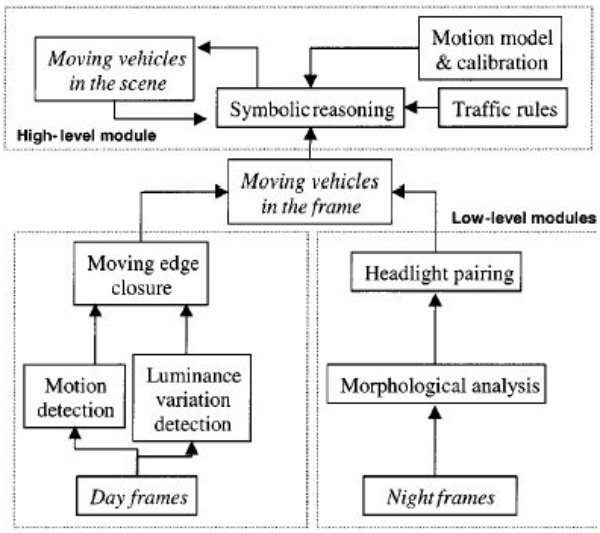


Figura 2.7: *Arquitectura del sistema*

El control del tráfico en este tipo de sistemas debe adaptarse a diferentes entornos, tales como clima, condiciones de iluminación... En particular, las escenas de tráfico urbano son especialmente complicadas debido a la gran variabilidad que presenta el fondo (***background***) de la imagen.

Como el sistema debe de trabajar en diferentes ambientes de iluminación y durante las 24 horas del día, se implementan dos módulos distintos: uno para los frames correspondientes al día (***day frames***) y otro para los frames correspondientes a la noche (***night frames***).

Los algoritmos utilizados durante el ***día*** son:

- ***Resta de fondo (Background Substraction):*** Detección en la imagen de los puntos en movimiento, mediante diferencia entre el frame bajo estudio con respecto al background o fondo. Si la diferencia es mayor que un determinado umbral adaptativo, el punto será clasificado como de movimiento, y en caso contrario será clasificado como fondo.
- Detección de puntos de alto contraste en la imagen (puntos con un gradiente elevado) como posibles bordes de los objetos del primer plano.
- Mediante operaciones morfológicas entre los puntos de movimiento y los puntos de borde se intenta extraer los objetos móviles.
- De entre los posibles candidatos, se seleccionan aquellos objetos que pueden ser vehículos desde el punto de vista del modelo en la escena.

En la figura 2.8 podemos ver el resultado de los algoritmos para detección de vehículos durante el ***día***:

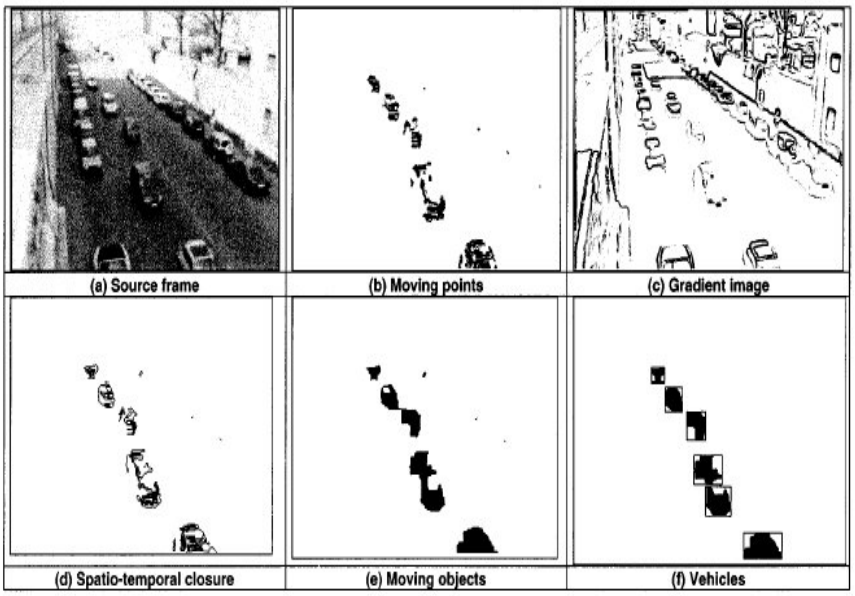


Figura 2.8: *Detección de vehículos durante el día*

Para la deteccíón de vehículos durante la ***noche*** hay que tener en cuenta que las condiciones de iluminación son totalmente distintas a las del día. Por lo tanto durante la noche, bajo condiciones de baja iluminación, para poder detectar los coches se utilizan las luces

de los faros delanteros. Los algoritmos utilizados durante la ***noche*** tienen como objetivo fundamental, identificar los vehículos según los pares de luces. Los principales algoritmos son:

- Umbralizado de la imagen mediante histograma, para obtener de una manera cómoda la separación entre los objetos y el fondo.
- Matching entre la imagen y una plantilla de luces de faros escalada con respecto a la región de la imagen.
- Correlación cruzada de los pares de luces.

En la figura 2.9 podemos ver el resultado de los algoritmos para detección de vehículos durante la ***noche***:

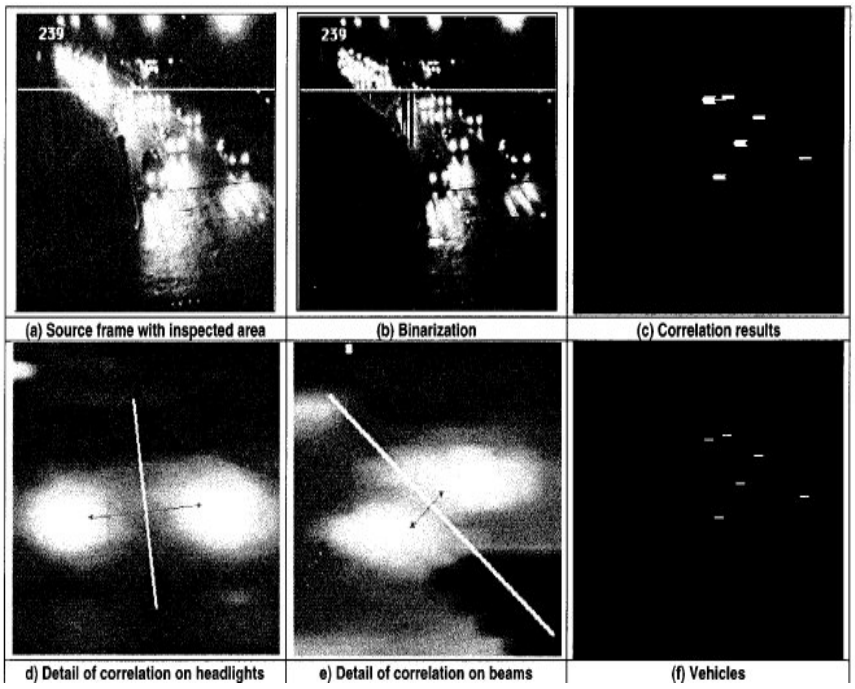


Figura 2.9: *Detección de vehículos durante la noche*

Cabe destacar que se desaconseja la utilización de la técnica de ***Flujo Óptico*** que a pesar de que permite obtener con gran exactitud la dirección y el movimiento de cada uno de los puntos de la imagen, requiere mucha carga computacional y no cumpliría los requisitos para un sistema de tiempo real.

Los resultados de este artículo, podemos verlos en la figura 2.10. A pesar de que no se puede observar gran cosa en las imágenes, y de que habría que ver los resultados del sistema en condiciones más desfavorables, el resultado es muy bueno, consiguiendo un

background realmente bueno.

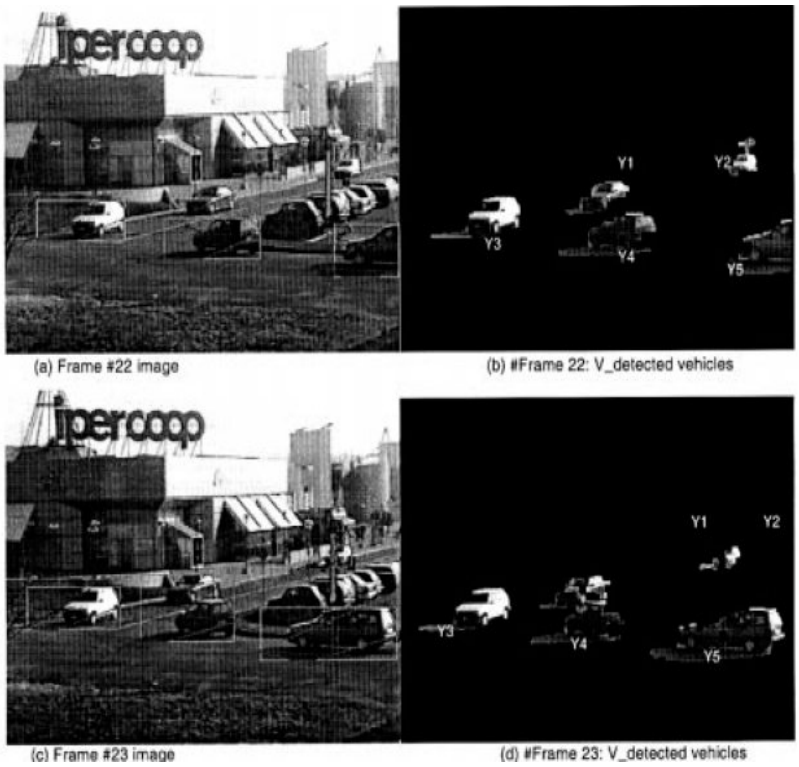


Figura 2.10: *Resultados de vehículos detectados*

### 2.2.2. Image Analysis and Rule-Based Reasoning for a Traffic Monitoring System

**Autores:** *Benjamin Maurin, Osama Masoud y Nikolaos P.Papanikolopoulos.*

La principal diferencia de este artículo con el anterior, es que en este último si utilizan el ***Flujo Óptico*** en combinación con la ***Resta de Fondo*** para poder clasificar los puntos de la imagen en fondo o movimiento.

El objeto de estudio de este artículo es la detección y seguimiento tanto de personas como de vehículos en entornos urbanos, cosa que es complicada ya que hay muchos objetos moviéndose (vehículos y peatones), y la detección de peatones en exteriores es también compleja.

Las principales etapas de las que se compone el procesado de las imágenes son:

- **Detección:** Para detectar el movimiento se realiza una combinación entre la información obtenida a partir del ***Flujo Óptico*** y la ***Resta de Fondo***.
- **Clustering:** En esta etapa, se deben agrupar todos aquellos píxeles que tengan características similares. La idea principal, es agrupar todos aquellos píxeles que su flujo óptico en la misma dirección sea mayor o menor que un determinado umbral.
- **Tracking:** Es la etapa de seguimiento y es una de las partes más esenciales de todo el proceso. Para realizar el tracking, se utiliza el ***Filtro de Kalman***.
- **Visualización:** Es la etapa final del proceso en la que se procede a la presentación en pantalla de los vehículos y personas detectados.

El proceso completo se puede ver de modo esquemático en la figura 2.11:

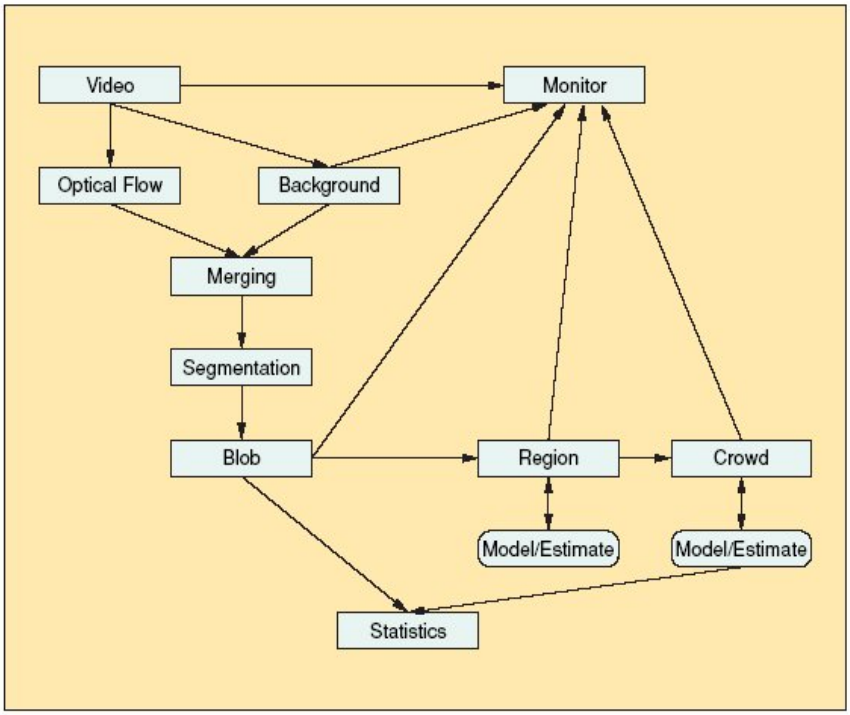


Figura 2.11: *Arquitectura del sistema*

Los resultados finales se pueden ver en la figura 2.12, aunque no se pueden apreciar muy bien los objetos debido a que la calidad de las imágenes disponibles no es del todo buena, pero si que se puede observar bastante bien la detección de peatones.

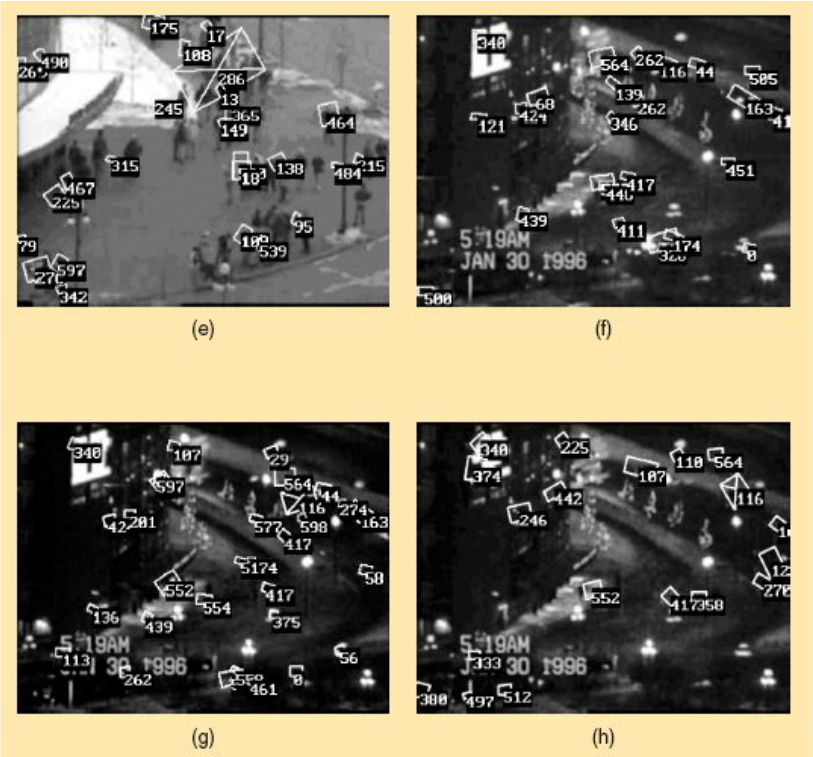


Figura 2.12: *Resultados de vehículos detectados*

## 2.3. Conclusiones

La base de partida para este proyecto no es ninguno de los ejemplos que se han mostrado anteriormente, ya que algunos difieren un poco del objetivo de este proyecto. Se han tomado ideas básicas de los ejemplos anteriores así como de otra serie de artículos que vienen referenciados en la sección ***Bibliografía***.

Así como para la realización de los algoritmos para este proyecto no se ha seguido ningún artículo en concreto, si no que se ha partido desde una base teórica y a partir de esta, se han implementado según las características de nuestra aplicación.

# Capítulo 3

# Entorno de Desarrollo

En este apartado se pretende detallar el software más importante que ha sido utilizado, para que un usuario pueda ejecutar el proyecto en una computadora y la aplicación funcione correctemente.

Dentro del entorno de desarrollo utilizado, los programas más importantes son los siguientes (todos ellos de ***código abierto***):

- Sistema Operativo: ***Linux***.
- Librerías para visión por computador: ***OpenCV***.
- Desarrollo de interfaz gráfica: ***Glade***.

A continuación, se comentan cada uno de los distintos puntos anteriores. También se incluyen ejemplos de cómo compilar el código para una correcta instalación de todos los programas necesarios para el correcto funcionamiento de la aplicación, y aunque no sea necesario para la compilación ni ejecución del proyecto, se ha añadido una breve guía de usuario que intenta explicar cómo construir una interfaz gráfica de usuario de manera sencilla con ***Glade***.

## 3.1. Linux

Para el desarrollo de este proyecto, se ha utilizado el sistema operativo ***Linux***. Linux es a su vez la denominación de un sistema operativo y el nombre de un núcleo. Linux es software libre (o código abierto), lo que quiere decir que el código está disponible al público y que cualquier persona con los conocimientos infórmaticos adecuados, puede libremente estudiarlo, usarlo, cambiarlo y distribuirlo. Esta es una de las principales razones por las cuales se ha desarrollado este proyecto en Linux, el uso de ***Software Libre*** le da al

programador una gran versatilidad sobre los programas a utilizar.

La marca ***Linux*** es propiedad de *Linus Torvalds*[1] y se define como *un sistema operativo para computadoras que facilita su uso y operación.*

Se supone que el usuario final presenta conocimientos básicos de como manejarse en un sistema Linux, por lo tanto para evitar una extensión innecesaria en la memoria del proyecto, solamente se comentarán las *distribuciones* utilizadas así como las ventajas de usar Linux.

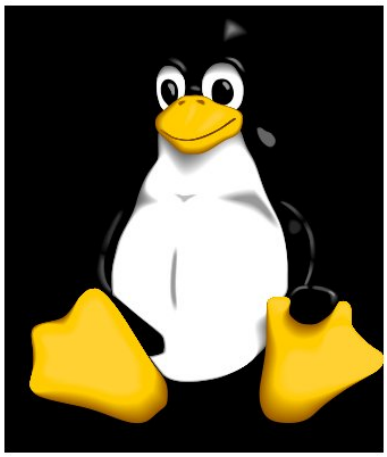

Figura 3.1: *Sistema Operativo Linux*

### 3.1.1. Distribuciones Linux

Una distribución ***Linux*** o ***GNU/Linux*** es un conjunto de aplicaciones que ofrecen distintas mejoras para instalar un sistema Linux. Son distintas versiones que por lo general, se diferencian únicamente en las herramientas de configuración y sistemas de paquetes de software a instalar, pero tienen el mismo núcleo o *kernel*.

Existen multitud de distribuciones de Linux. Cada una de ellas puede incluir multitud de programas adicionales (de código abierto o no). Estos programas adicionales pueden ser para instalar más fácilmente el sistema, más aplicaciones, entornos gráficos distintos, paquetes ofimáticos, servidores web...

La base de cada distribución incluye el núcleo Linux, con las bibliotecas y herramien-

[1] creador del *kernel* (núcleo) del sistema operativo GNU/Linux.

tas del proyecto GNU y de muchos otros proyectos como *BSD*[2].

Para mantener las interfaces gráficas, normalmente se utilizan la plataforma *XFree86* o la *Xorg*.

En el desarrollo de este proyecto, se utilizaron dos distribuciones distintas de Linux:

1. ***Knoppix 3.7***: Knoppix es una distribución de Linux basada en *Debian* y utilizando *KDE*[3].

   Esta distribución fue utilizada durante la primera fase de desarrollo del proyecto bajo un ordenador de sobremesa Pentium 4 CPU 3 GHz 992 Mb RAM.


Figura 3.2: *Knoppix 3.7*

2. ***Fedora Core 3***: Fedora Core (Fedora Linux) es una distribución Linux desarrollada por la comunidad Fedora y promovida por la compañía norteamericana *Red Hat*.

   Se utilizó esta distribución durante la fase final de desarrollo del proyecto bajo un ordenador portátil PENTIUM 4 CPU 2.66 GHz 512 MB RAM.

[2] Berkeley Software Distribution
[3] K Desktop Environment, entorno de escritorio gráfico


Figura 3.3: *Fedora Core 3*

### 3.1.2. Ventajas de Linux

El proyecto se podía haber desarrollado desde un principio bajo el sistema operativo ***Windows***, ya que las librerías de visión ***OpenCV*** también están disponibles para este sistema operativo y además son gratuitas.

Sin embargo, se prefirió utilizar ***Linux*** por las siguientes ventajas:

1. La principal ventaja de Linux, es que es ***software libre*** y que la mayoría de los programas bajo Linux son código abierto, con lo cual se abren muchas posibilidades a los desarrolladores y a los usuarios finales.

2. Todas las aplicaciones y paquetes necesarios para el desarrollo de la aplicación son gratis, con lo cuál se reduce el coste total del proyecto.

3. Presenta una gran ***flexibilidad*** lo que ha permitido que sea utilizado en sistemas muy diversos y en distintas arquitecturas: computadoras personales, supercomputadores, dispositivos portátiles...

4. Existen ***Grupos de Usuarios de Linux*** por todas las partes del planeta, con lo que muchas veces se puede obtener ayuda gratuita sobre diversos problemas que ocurran.

5. Presenta aplicaciones específicas para desarrollo de aplicaciones en C, ya que cuenta con herramientas propias de Linux para compilar como son ***gcc***, ***g++***. Cabe destacar la herramienta ***Makefile***, por lo que simplemente se necesita un editor de textos y un

compilador, a diferencia de otros entornos de programación en Windows.

Un ***Makefile*** es un script que se ejecuta con la orden ***make***. Este comando nos ayuda a compilar nuestros programas. Las principales ventajas que presenta son:

- Detecta que archivos son necesarios de recompilar. Si estamos trabajando con un programa con multitud de archivos fuente y solamente modificamos uno de ellos, al ejecutar el comando ***make***, solamente se recompilarán los ficheros que se hayan modificado y los que dependan del fichero modificado.
- Podemos incluir en los comandos de compilación todos los parámetros necesarios para incluir librerías, ficheros de cabecera .h, ficheros objeto... Con lo cuál no hace falta cada vez que queramos compilar saberse de memoria miles de comandos y opciones de compilación, o al menos con hacerlo una sola vez es suficiente.

6. Se puede utilizar un programa sencillo y gratuito para crear una interfaz gráfica de usuario como ***Glade***.
7. La mayoría de las distribuciones cuentan con sencillos gestores de descarga para ir actualizando los paquetes y reinstalarlos o instalar nuevos paquetes.

Por otro lado, la principal ***desventaja*** que presenta Linux, es que para poder aprovechar al máximo todas sus posibilidades, son necesarios unos conocimientos avanzados de informática.

## 3.2. OpenCV

La principal librería utilizada en el desarrollo de los algoritmos de tratamiento de imágenes, es ***OpenCV***.


Figura 3.4: *OpenCV*

Se trata de una librería de código abierto desarrollado por ***Intel***. Funciona bajo ordenadores personales que estén basados en la arquitectura ***Intel*** y es totalmente integrable en aplicaciones escritas en C y C++.

Implementa una gran variedad de funciones de alto nivel para el desarrollo de aplicaciones en Visión por Computador así como muchos tipos de datos (matrices, imágenes, árboles ...) dando mucha facilidad de diseño al programador a la hora de implementar los algoritmos. Dispone de más de 350 funciones y algoritmos de alto nivel. Algunas de sus principales aplicaciones son:

- Operaciones básicas.
- Análisis de imagen.
- Reconocimiento de objetos.
- Detección de objetos.
- Calibración.
- Reconstrucción 3D.
- Seguimiento.

### 3.2.1. Estructura de OpenCV

***OpenCV*** se puede estructurar en distintas partes cada una con una funcionalidad determinada. Cada una de las partes tienen su respectiva cabecera (.h) y son:

- ***CxCore:*** Contiene estructuras de datos básicos, así como funciones matemáticas y para dibujar líneas, rectángulos...
- ***CvReference:*** Contiene funciones especiales, como gradientes, histogramas, calibración...
- ***CvAux:*** Funciones en fase de prueba o experimentales.
- ***HighGui:*** Funciones de interfaz gráfica (interfaz del ratón) y funciones de vídeo y cámara.

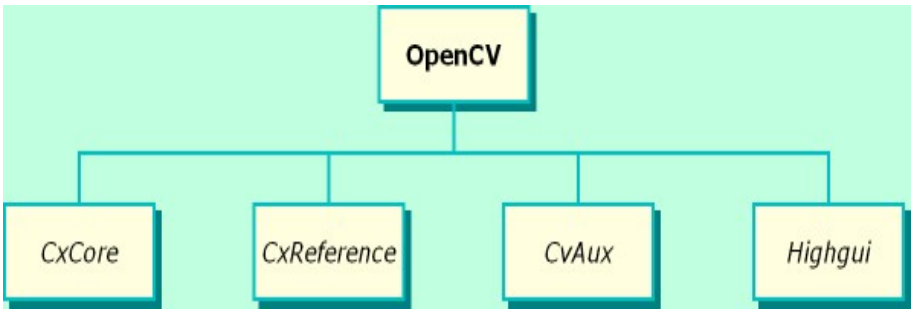


Figura 3.5: *Estructura de OpenCV*

### 3.2.2. Compilación e Instalación de OpenCV con Soporte para AVIs

En este apartado de la memoria, comentaremos como se ha de realizar la instalación de las librerías OpenCV para que la aplicación funcione correctamente.

En primer lugar, se necesitan los siguientes paquetes para que OpenCV se encuentre correctamente instalado:

- gtk +2.x
- libjpeg
- zlib
- libpng
- libtiff
- v4l
- dc1394 & raw1394
- ffmpeg

Para el proyecto, el único paquete que no necesita ser instalado es *dc1394 & raw1394*. Dependiendo de la aplicación que desarrollemos podremos no necesitar alguno de los paquetes anteriores.

Para instalar los paquetes anteriores, se puede acudir al gestor de descargas de la distribución Linux con la que se este trabajando, no obstante, muchos de estos paquetes ya se encuentrar instalados en algunas distribuciones. También nos los podemos descargar desde páginas de Internet como *www.sourceforge.net*.

Sin embargo, existen paquetes que no suelen venir en las distribuciones típicas de Linux por lo que tenemos que descargarlos de Internet, e incluso en ocasiones, en algunos de ellos es necesario realizar ciertas modificaciones. Los paquetes que se comentan a continuación, están incluidos en el CD adjunto al proyecto.

En este apartado de la memoria, nos centraremos en la compilación e instalación de las librerías OpenCV con soporte para poder trabajar con vídeos AVI. Los pasos a seguir para una correcta instalación y posterior compilación son:

### 3.2.3. Instalación y compilación de ffmpeg.

Las librerías ***ffmpeg*** son necesarias para poder descomprimir vídeo y son un requisito para la instalación de OpenCV si deseamos trabajar con vídeos.

Lo primero que tenemos que hacer es descomprimir el paquete por lo que en modo usuario ejecutaremos la instrucción:

**tar -zxvf ffmpeg-0.4.9-pre1.tar.gz**

Con la instrucción anterior se descomprimirá el archivo en un directorio. Entrando en el directorio principal creado por la instrucción *tar*, podemos configurar el software tecleando:

**./configure**

Pasamos a compilar el software tecleando:

**make**

Nos pasamos a modo **root** tecleando **su** e introduciendo la contraseña de supervisor.Ahora podemos realizar la instalación del software tecleando:

**make install**

Y cuando haya finalizado el comando anterior:

**make installlib**

Si obtenemos algún tipo de error en el paso anterior, realizamos el siguiente enlace y volvemos al paso anterior:

**ln -s libavformat libav**

### 3.2.4. Instalación y compilación de OpenCV.

Una vez que tenemos instalada la librería ffmpeg y el resto de paquetes necesarios, podemos pasar a la instalación de OpenCV.

Lo primero que tenemos que hacer es descomprimir el paquete por lo que en modo usuario ejecutaremos la instrucción:

**tar -zxvf opencv-0.9.6.tar.gz**

Con la instrucción anterior se descomprimirá el archivo en un directorio. Entrando en el directorio principal creado por la instrucción *tar*, podemos configurar el software tecleando:

**./configure**

Pasamos a compilar el software tecleando:

**make**

Nos pasamos a modo **root** tecleando **su** e introduciendo la contraseña de supervisor.Ahora podemos realizar la instalación del software tecleando:

**make install**

Seguimos en modo **root** y modificamos el path de las librerías añadiendo en el fichero **/etc/ld.so.conf** la línea **/usr/local/lib**. Salvamos el fichero anterior y tecleamos:

**ldconfig -v**

En este momento ya estamos listos para poder compilar ejemplos y ejecutarlos, no obstante si utilizamos un Makefile, tendremos que editarlo correctamente como se puede observar más adelante en la sección de la memoria dedicada a la elaboración del Makefile.

Por último, cabe destacar que la instalación se ha realizado para la versión 0.9.6 de OpenCV. Si se usan otras versiones distintas o se trabaja con una distribución de Linux diferente, es posible que en algún punto de la instalación anterior se obtenga algún tipo de error y sea necesario modificar algunas cosas.

## 3.3. Glade

***Glade*** es una aplicación gráfica basada en las líbrerias ***GTK*** que nos permite crear interfaces de usuario o *GUI*[4] de una manera bastante sencilla, además de ser una herramienta que se encuentra disponible de manera gratuita en la mayoría de las distribuciones Linux.

Las principales características de ***Glade*** son:

- Soporte para casi todos los *widgets*[5] de GTK.
- Software Libre.
- Disponibilidad en la mayoría de las distribuciones Linux.
- Posibilidad de generar el código en diversos lenguajes como: C, C++, Ada95, Perl y Phyton.
- La interfaz de usuario se almacena en XML[6], lo que permite una fácil integración de herramientas externas.
- Permite crear automáticamente los archivos fuente del proyecto, así como otros archivos de configuración y el ***Makefile*** para poder compilar de manera sencilla los archivos del proyecto cada vez que sean modificados.
- La interfaz puede ser llamada dinámicamente utilizando la librería *libglade*.
- Fácil manejo de los *widgets* a partir de funciones de las librarías GTK y GDK.
- Es una aplicación utilizada por muchos desarrolladores en el mundo, por la tanto muchas veces se puede obtener ayuda on-line.
- Compatible con otros entornos de programación como *Anjuta*.

A continuación se comentará brevemente como realizar una sencilla interfaz de usuario y así poder explicar el funcionamiento básico de ***Glade***.

[4] Graphical User Interface
[5] componente gráfico o control con el cuál lel usuario interactua
[6] lenguaje de marcas extensible

### 3.3.1. Construcción de la Interfaz de Usuario

1. Lo primero de todo es ejecutar el programa. Lo podemos hacer desde la línea de comandos escribiendo *glade* o bien desde el menú de inicio seleccionando el programa.

2. Una vez ejecutado el programa veremos la ventana de la figura 3.6, en la que seleccionaremos ***Nuevo***. Tras haber hecho esto se nos preguntará que tipo de proyecto queremos crear, si queremos crear una aplicación *Gnome* o una aplicación *GTK+ 2.0*. En el caso de este proyecto, se realizó una aplicación *GTK+ 2.0*.


Figura 3.6: *Menú Principal*

La ventana anterior, será la que controle nuestro proyecto. En ***Opciones*** podremos cambiar las opciones del proyecto, el nombre de los archivos, el uso de *libglade*, el lenguaje de programación... Por defecto, no se deben modificar ninguna de las opciones, solamente cambiar el nombre del proyecto y el directorio donde queremos que se encuentre el proyecto. El *Menú de Opciones del Proyecto*, podemos verlo en la figura 3.7.

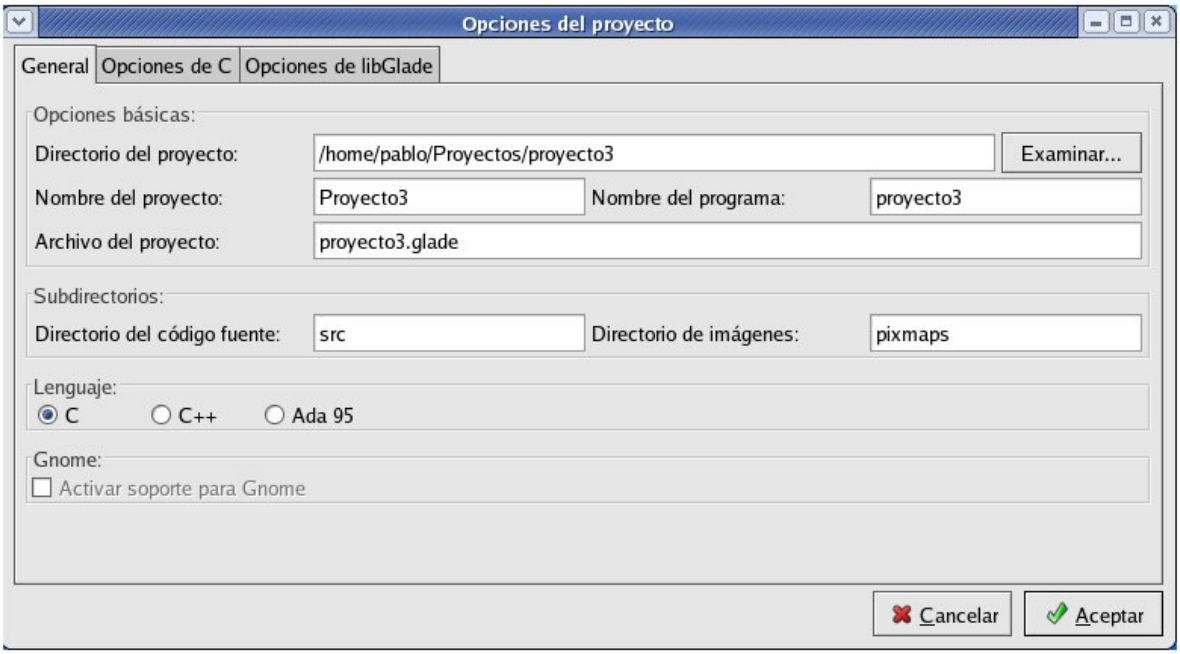


Figura 3.7: *Opciones del Proyecto*

3. Ahora ya podemos empezar a rellenar nuestro proyecto. Lo primero que necesitaremos es un lugar para colocar los ***widgets***, para lo que seleccionaremos en la *paleta de widgets* el icono ventana. La paleta nos permite añadir ***widgets*** (pequeños artefactos) a nuestra aplicación. Podemos verlo en la figura 3.8.


Figura 3.8: *Paleta de Widgets*

4. Ya tenemos una nueva ventana lista para poder crear nuestra interfaz. Podemos cambiar algunos parámetros de nuestra ventana tales como nombre, dimensiones, etc. en la ventana de ***Propiedades*** y añadirle algunas ***señales***, cosa que veremos más adelante.

   Esta ventana, va a ser el contenedor principal de nuestra aplicación, ya que sobre ella debemos de ir empaquetando el resto de widgets que insertemos en nuestra aplicación.

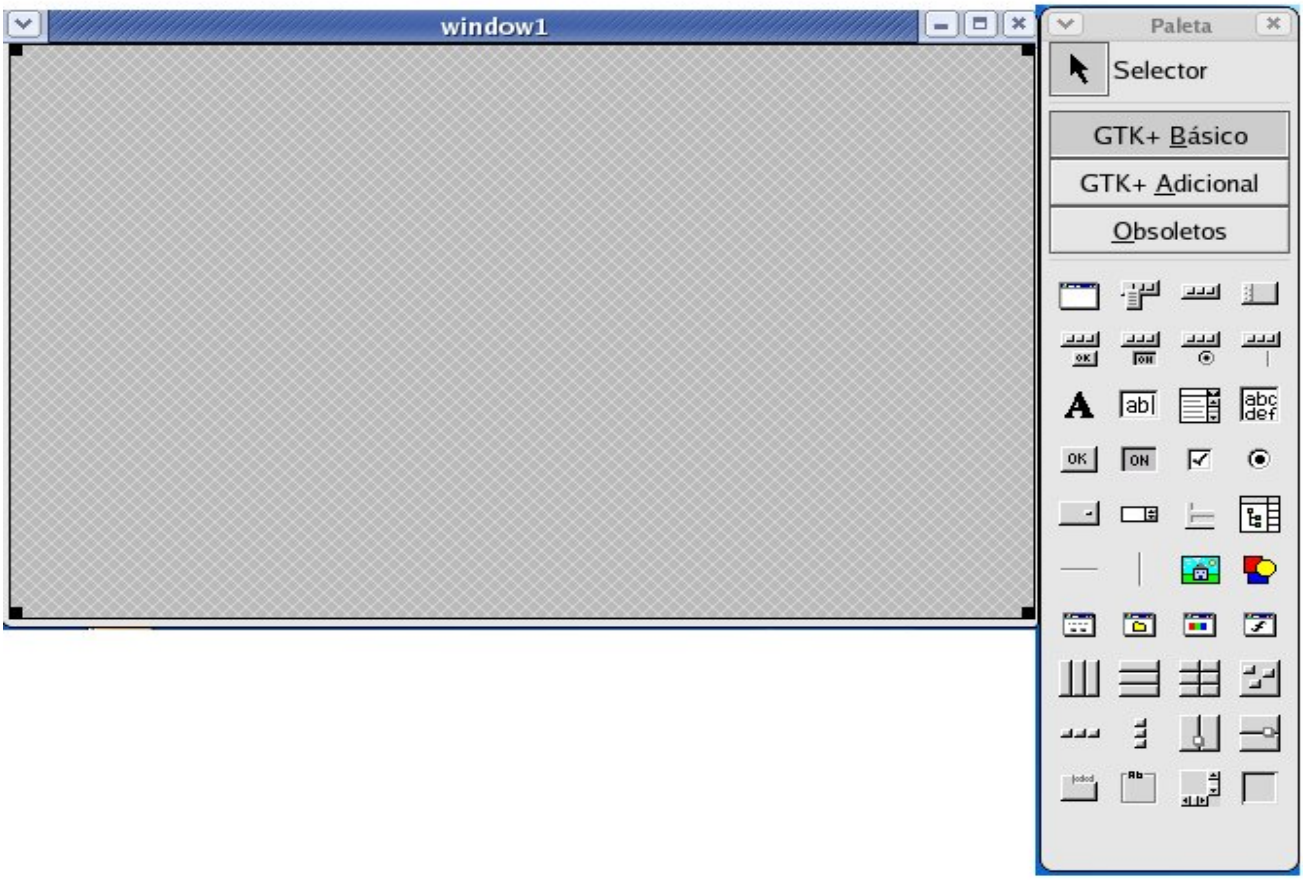


Figura 3.9: *Ventana Principal y Paleta de Widgets*

5. Ahora tenemos que añadir algunos botones y etiquetas, pero no podemos hacerlo libremente. En ***Glade*** los ***widgets*** son empaquetados. Al principio lo de empaquetar widgets nos puede parecer algo engorroso, sin embargo, es bastante útil como por ejemplo para realizar un cambio automático del tamaño de la ventana. Ya que normalmente cuando un usuario cambia el tamaño de una aplicación le gusta que los widgets de la ventana incrementen también su tamaño para aprovechar mejor el espacio y del mismo modo si se reduce el tamaño de la aplicación. Esto se logra automáticamente con el empaquetado sin que el desarrollador tenga que escribir código adicional para los cambios de tamaño.

Para poder hacer el empaquetado, tenemos que crear cajas o contenedores, ya que una de las características más importantes de ***Glade*** es que los ***widgets*** deben estar dentro de contenedores.

Los contenedores son invisibles, es decir, no pueden ser vistos en tiempo de ejecución, sin embargo tienen un efecto directo en la aplicación.

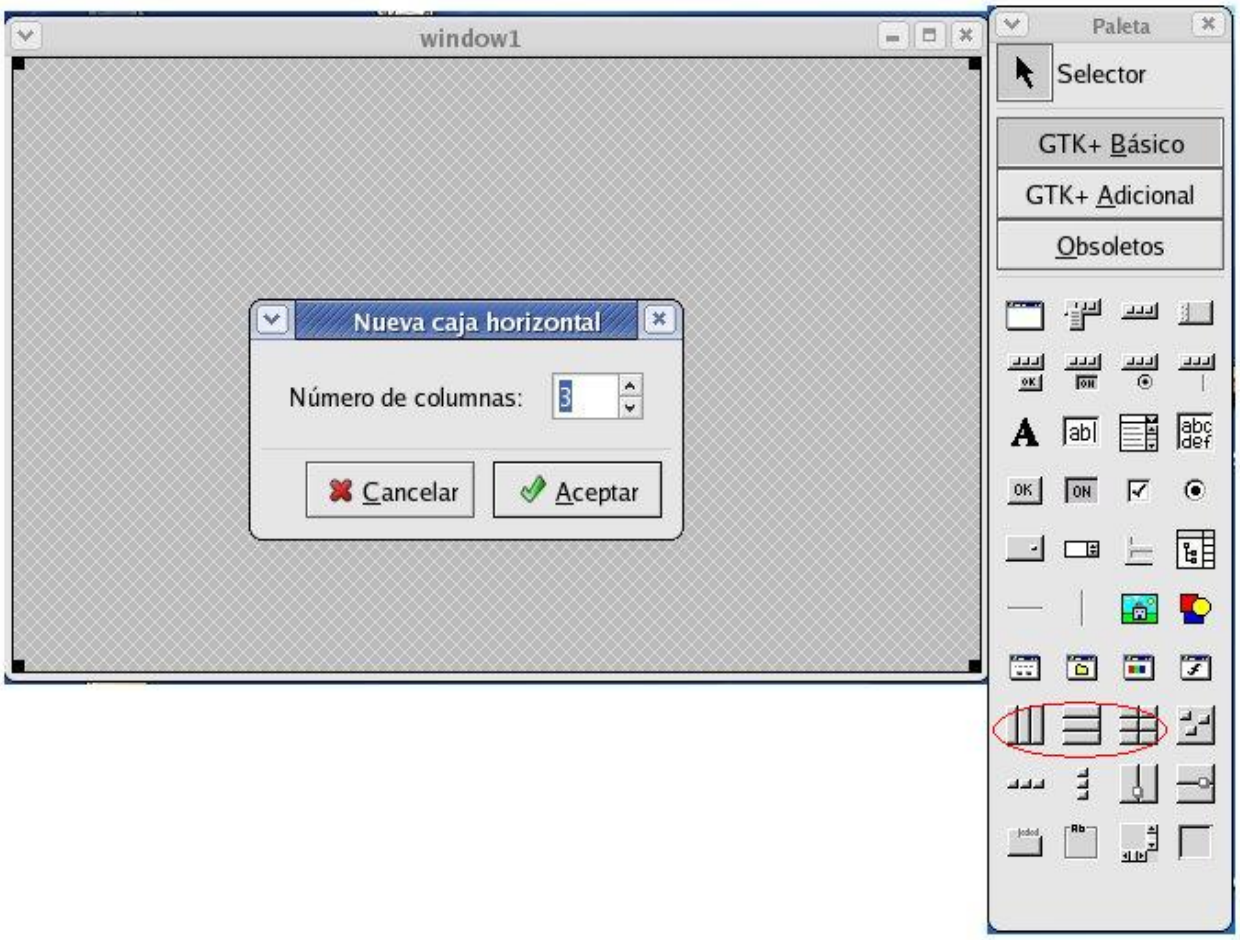


Figura 3.10: *Introduciendo Contenedores*

6. Después de ir insertando *widgets* entre ellos etiquetas, botones, dibujos, gráficas y demás, un primer resultado se puede ver en la figura 3.11.

   Para que se vean mejor los botones y la aplicación tenga una cara más bonita, se pueden tocar las propiedades de *Empaquetado*, en concreto *Expandir* y *Rellenar*. De esta forma conseguiremos que la aplicación se vea con un aspecto mejor y más profesional.


Figura 3.11: *Ejemplo de Interfaz de Usuario*

7. Ahora deberíamos añadir alguna ***señal***. Por señal entendemos cualquier evento que ocurre en un determinado *widget* y ante el cual nuestra aplicación puede responder. Para añadir señales a un widget, tenemos que irnos al campo *Propiedades* y allí a *Señales*. Elegimos la señal que queremos y le damos a añadir. Podemos ver el proceso en la figura 3.12.

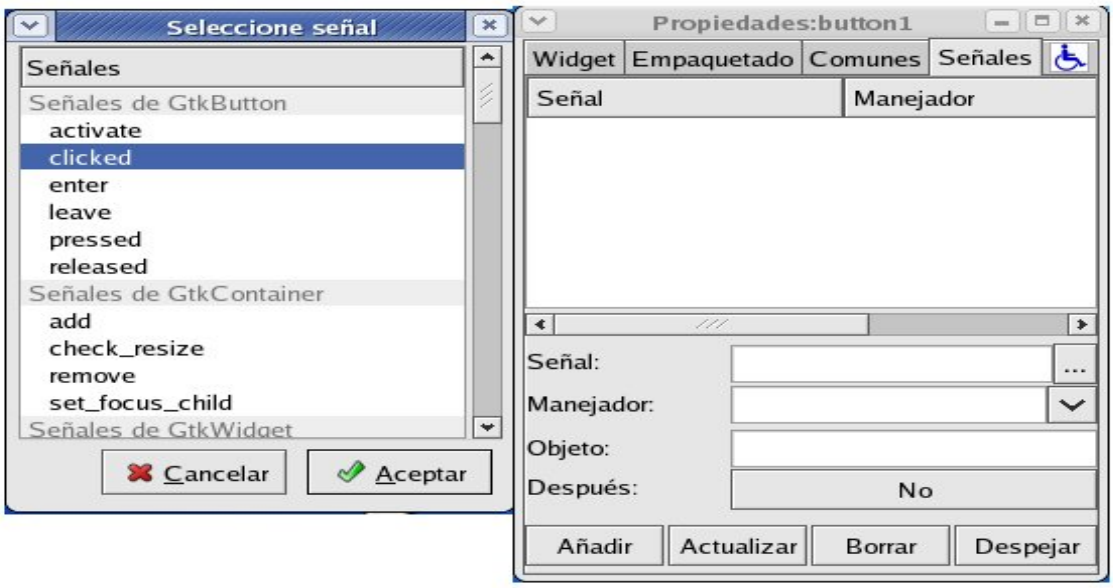


Figura 3.12: *Señales*

8. A medida que vamos incrementando el número de *widgets* en nuestra interfaz, es conveniente hacer uso del ***Árbol de Widgets*** en el que podemos ver la organización de todos los widgets de nuestra aplicación, facilitándonos el acceso a los mismos.

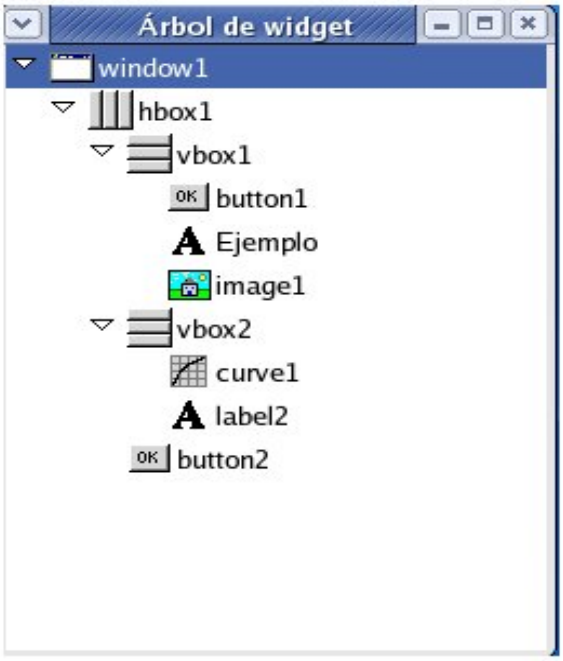


Figura 3.13: *Árbol de Widgets*

9. Luego se pueden seguir añadiendo *widgets* y mejorando el aspecto de la interfaz gráfica de usuario hasta que la tengamos completa.

   Podemos ver el aspecto final de la interfaz de usuario del proyecto en la figura 3.14.

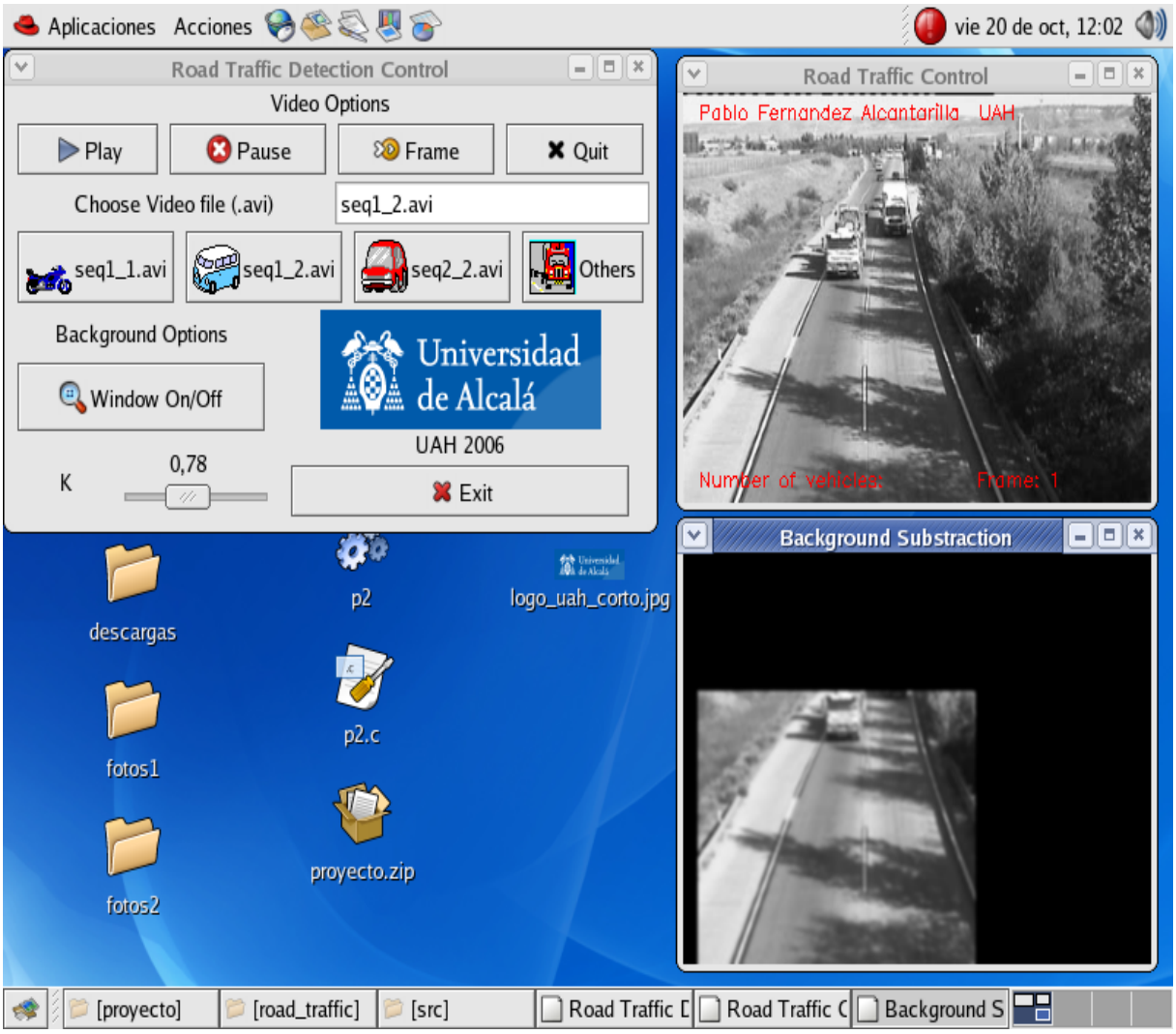


Figura 3.14: *Resultado Final de la Interfaz de Usuario*

10. Por fin ya está terminado el esqueleto de la interfaz gráfica. Pero esto no vale para mucho si no le damos funcionalidad a las señales de los widgets.

    Antes de escribir código, pulsamos en el ***Menú Principal*** el botón ***Construir*** para que Glade escriba el código de la interfaz de usuario asociado a nuestra aplicación.

    Por defecto, los ficheros que nos genera Glade son los siguientes:

    - **/src/interface.c**, **/src/interface.h**: Funciones necesarias referentes a la creación del interfaz gráfico. No deben modificarse, ya que *Glade* sobreescribe estos archivos al modificar el proyecto.
    - **/src/callbacks.c**, **/src/callbacks.h**: Son funciones encargadas del manejo de señales, y en el archivo .C se escribirá el código C necesario para dar una respuesta ante los eventos que se produzcan.
    - **/src/support.c**, **/src/support.h**: Funciones de soporte que no deben ser editadas.
    - **/src/main.c**, **/src/main.h**: En estos archivos se encuentran aquellas funciones que son ejecutadas antes de la creación de la ventana principal. Se puede editar.

    Es conveniente realizar una observación acerca de los archivos anteriores. Glade es una herramienta que permite que se puedan añadir widgets sin modificar el resto del programa. Esto es, si ya hemos escrito código en el archivo callbacks.c, Glade solamente escribirá el código necesario para el nuevo widget que se ha añadido. Sin embargo, si queremos borrar alguna función para algún widget que hemos eliminado, tendremos que modificar los archivos de código manualmente.

    Para el manejar los widgets mediante lenguaje C, se dispone de las siguientes librerías:

    - **GTK+**: Contienen funciones para poder acceder y controlar los widgets.
    - **GDK**: Contienen funciones que están más enfocadas hacia la construcción de los widgets.

    Se puede buscar más información acerca de estas librerías en las referencias [20], [21], así como en numerosos sitios en Internet.

    Si se desea profundizar más en la programación en Glade, se pueden consultar también las siguientes referencias: [18], [19].

11. El último paso antes de poder ejecutar nuestra aplicación es compilar el código. Si todavía Glade no nos ha generado ningún Makefile, utilizamos el comando ./autogen.sh y posteriormente en el directorio /src ejecutamos make para poder compilar el proyecto. Y finalmente, dentro del directorio ./src podemos ejecutar nuestro programa con el nombre que la hayamos dado.

## 3.4. Makefile

A la hora de compilar el proyecto, se tienen que realizar una serie de modificaciones para que la compilación no genere ningún error. Para editar el Makefile tenemos 2 opciones:

1. Usar el Makefile generado por Glade y realizar modificaciones.
2. Realizar un Makefile nuevo.

### 3.4.1. Makefile Creado por Glade

Si elegimos la primera opción Glade generará un Makefile automáticamente. Pero tenemos que realizar una serie de modificaciones al archivo, ya que necesitamos compilar el archivo **v4l.cpp** *Video for Linux* para poder trabajar con vídeos, y por defecto en el Makefile el compilador es de lenguaje C. A su vez, también tendremos que indicar los *paths* o caminos en donde se encuentran las librerías de OpenCV.

Se deben realizar las siguientes modificaciones:

- **CC**: cambiamos gcc por **g++**.
- **CPP**: cambiamos gcc -E por **g++ -E**.
- **ac_ct_cc**: gcc por **g++**.

Y ahora tenemos que incluir los *paths* correspondientes a las funciones de OpenCV:

- **INCLUDES**: Añadimos **-I/usr/local/include/opencv**.
- **road_traffic_LDADD**: Añadimos **-lopencv -lcvaux -lhighgui**.

Y finalmente en los campos **SOURCES HEADERS** y **OBJECTS**, añadimos los ficheros que falten de nuestro proyecto.

### 3.4.2. Makefile Creado para el Proyecto

Debido a que muchas veces el usuario no tiene toda la capacidad deseada para entender todos los pasos y comandos del Makefile que genera Glade y para evitar errores en la compilación, en este proyecto se optó por realizar un sencillo Makefile en unas pocas líneas de código. El Makefile se puede ver a continuación:

```
OPENCV_LDFLAG = -Wl,--export-dynamic -L/usr/local/lib -lgtk-x11-2.0\
          -lgdk-x11-2.0 -latk-1.0 -lgdk_pixbuf-2.0\
          -lm -lpangoxft-1.0 -lpangox-1.0 -lpango-1.0\
           -lgobject-2.0 -lgmodule-2.0 -ldl -lglib-2.0 -lcxcore\
           -lcv -lhighgui -lcvaux

LDFLAGS  = $(OPENCV_LDFLAG) -lm

OPENCV_INFLAG =  -DXTHREADS -I/usr/include/gtk-2.0\
 -I/usr/lib/gtk-2.0/include -I/usr/X11R6/include\
 -I/usr/include/atk-1.0 -I/usr/include/pango-1.0\
     -I/usr/include/freetype2 -I/usr/include/glib-2.0\
     -I/usr/lib/glib-2.0/include -I/usr/local/include/opencv

INCFLAGS = $(OPENCV_INFLAG)

CC = g++
CFLAGS = -O

SRCS = callbacks.c main.c interface.c support.c v4l.cpp opencv_avi2.c
HDRS = callbacks.h interface.h support.h v4l.h opencv_avi2.h\
opencv_capture.h kalmanf.h
OBJS = callbacks.o main.o interface.o support.o v4l.o opencv_avi2.o
TARGET = road_traffic


#
# compile
#
.c.o:
$(CC) $(INCFLAGS) $(CFLAGS)  -c -g $<
#
# link
#
$(TARGET) : $(OBJS)
$(CC) -g -o $@ $(OBJS) $(LDFLAGS)

#
```

```
# compare time
#
#opencv_capture.o : opencv_capture.c $(HDRS)
v4l.o : v4l.cpp $(HDRS)

clean:
rm -f $(OBJS) $(TARGET) *~
```

# Capítulo 4

# Descripción del Algoritmo

En esta sección, se explicarán cada una de las fases más importantes de las que consta el algoritmo desarrollado para el proyecto. Podemos ver un esquema del algoritmo en la figura 4.1:

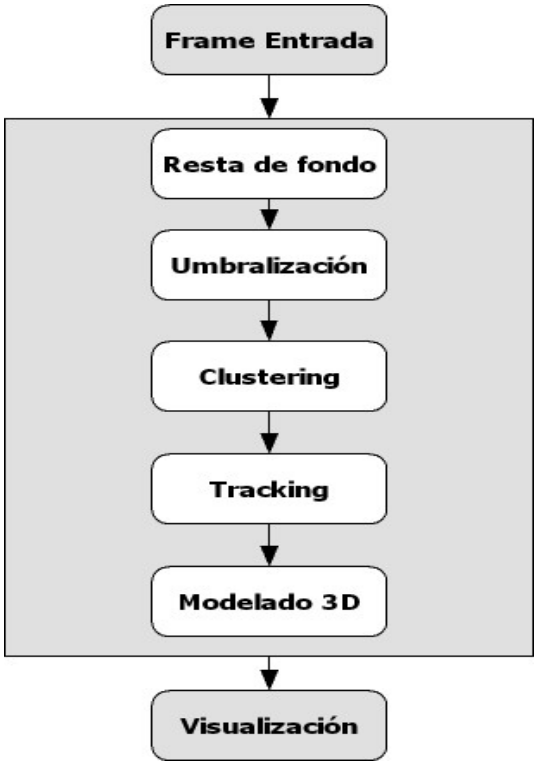


Figura 4.1: *Esquema del Algoritmo*

Básicamente, cada una de las etapas anteriores consiste en lo siguiente:

1. **Frame de Entrada:** En la fase de entrada obtenemos un frame del vídeo *.avi* bajo estudio. La aplicación trabaja con frames en **escala de grises** reduciendo de esta manera el coste computacional.
2. **Resta de Fondo:** En esta fase se realiza una ***segmentación*** de la imagen de entrada. Por un lado tenemos que clasificar que píxeles corresponden a un primer plano y cuales corresponden al fondo. Este análisis se realiza mediante la obtención de parámetros estadísticos de los píxeles.
3. **Umbralización:** Simplemente se umbraliza la imagen obtenida tras aplicar la ***Resta de Fondo*** y se aplican operadores morfológicos. Aquellos píxeles que pertenezcan al fondo se les dará un valor 0, y aquellos que sean de primer plano un valor 1 (255).
4. **Clustering:** Se agrupan los distintos píxeles según condiciones de proximidad y de vecindad entre píxeles. Lo que se pretende es obtener de entre todos los píxeles correspondientes a un primer plano, cuáles de ellos pertenecen a un mismo *cluster* y de ese modo a partir del número de clusters que tengamos poder obtener una idea aproximada del número de vehículos en la imagen. También se realizan operaciones de *limpieza* de clusters.
5. **Tracking:** Se realiza un seguimiento a cada uno de los vehículos detectados. Se realiza un filtrado de la posición en 2D sobre la que se construye el modelo 3D utilizando el ***Filtro de Kalman*** para conseguir suavizar los resultados finales.
6. **Modelado 3D:** Para cada uno de los clusters obtenidos, se realiza un modelado 3D. Se consideran conocidas las dimensiones de los vehículos en 3D, así como la posición y la calibración estimada de la cámara en la escena. Por lo tanto, a partir de un punto de cada uno de los clusters en 2D, se proyectará ese punto en 3D y se construirá el modelo en 3D a partir de ese punto, para posteriormente volverlo a proyectar en 2D.
7. **Visualización:** Finalmente se muestran los resultados de todo el proceso anterior en una nueva imagen.

A continuación se comentarán con amplio detalle las fases más importantes del algoritmo. La explicación de cada una de estas fases, seguirá el siguiente esquema:

- Introducción.
- Algoritmo Desarrollado.
- Esquema de Programación.

Posteriormente, en otro capítulo de la memoria se comentarán los resultados obtenidos.

## 4.1. Resta de Fondo

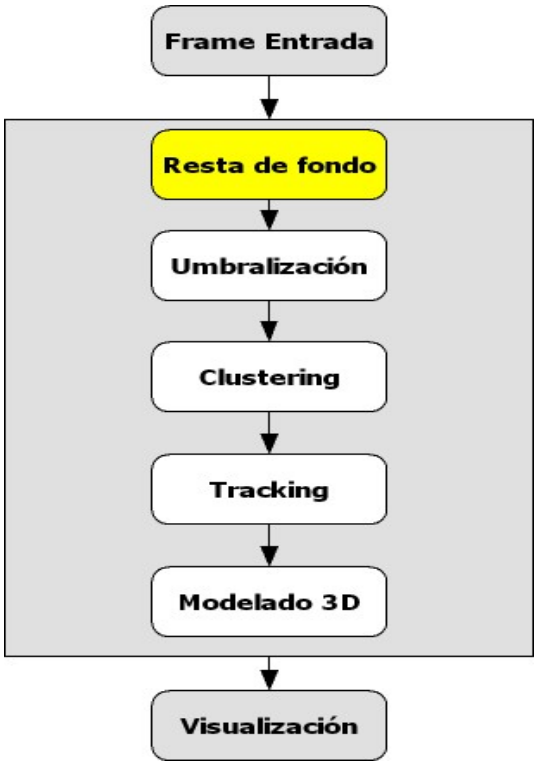


Figura 4.2: *Esquema del Algoritmo: Resta de Fondo*

### 4.1.1. Introducción

En muchas aplicaciones de visión por computador, los objetos aparecen sobre un fondo que es bastante estable y cambia poco con el tiempo. Por ejemplo, si observamos durante un tiempo una carretera, podemos darnos cuenta de que el fondo permanece casi constante y podemos separar mentalmente lo que es el fondo de la imagen y los objetos en movimiento.

La idea básica es la siguiente: **Si el fondo permanece constante podemos estimar los vehículos en movimiento restándole a la imagen original el fondo.** De este modo podríamos ser capaces de contar los vehículos que circulan en una determinada carretera. Es lo que se conoce como ***Resta de Fondo*** o ***Background Substraction***.

En este tipo de aplicaciones, se puede obtener una buena segmentación de la imagen si se le resta a la imagen bajo estudio un estimación aproximada del fondo. Por lo tanto el principal objetivo va a ser: **obtener una buena estimación del fondo**.

Sin embargo, esto en la práctica no resulta tan sencillo, ya que tenemos una serie de problemas como pueden ser:

- No se mueven sólo los vehículos.
- Tenemos árboles, vegetación, peatones...
- Problemas con las sombras, que en ocasiones pueden ocultar otros vehículos.
- Condiciones metereológicas variables.
- Condiciones de luminosidad también variables.
- Diferentes tipos de vehículos con tamaños muy diferentes como camiones, coches o motos.

Una de las consideraciones importantes que se tienen que tener en cuenta, es que en el ***inicio*** de la ***Resta de Fondo los vehículos no pueden estar parados***, ya que serían clasificados como fondo, cuando en realidad son de primer plano. Por lo tanto es conveniente que al iniciar el algoritmo no existan muchos vehículos parados en la carretera.

La manera de estimar el fondo, será tomando estadísticos de cada uno de los píxeles que forman parte del área de trabajo.

Luego se compara el valor absoluto de la diferencia entre el valor de gris del píxel actual y el valor medio de gris para ese píxel en el modelo de fondo con un umbral determinado. Si la diferencia es mayor que el umbral, el píxel pertenece al primer plano, y en caso contrario el píxel es de fondo.

El mayor problema, va a estar en calcular ese umbral adaptativo al entorno, en conocer la regla de decisión que nos permita clasificar a un píxel como fondo o como primer plano.

### 4.1.2. Algoritmo Desarrollado

Se calcularon dos umbrales distintos para obtener la regla de decisión. Comentaremos los dos intentos, a pesar de que el primero de ellos no fue satisfactorio pero tuvo gran importancia en el desarrollo del algoritmo definitivo.

Ambos algoritmos tienen un tiempo transitorio ajustable mediante software, en el cual se toman estadísticos de todos los píxeles del área de trabajo para poder estimar un modelo de fondo. En este proyecto, el transitorio tiene una duración de 4 segundos (200 Frames). Después del transitorio inicial, ya se pasa a clasificar píxeles como fondo o primer plano.

**Primer Intento**

Durante las primeras pruebas para obtener el ***fondo*** o ***background*** se realizó un análisis estadístico con **validación por desviación típica con respecto a la media**.

Cuando el valor de gris de un píxel sea menor que la desviación típica pasará a formar parte del fondo a no ser que estemos haciendo un seguimiento previo para evitar utilizar dichos puntos en el cálculo del fondo.

$(p_{i,j} - E[p_{i,j}]) < k \cdot \sigma_{i,j} \Rightarrow$ *Ese píxel pertenece al fondo*.

$(p_{i,j} - E[p_{i,j}]) > k \cdot \sigma_{i,j} \Rightarrow$ *Ese píxel pertenece al primer plano*.

Siendo *k* un parámetro de ajuste. Consideramos inicialmente k = 1. **Los píxeles que no pertenezcan al fondo no se computan para el cálculo del mismo**.

En la figura 4.3 podemos ver el **resultado** obtenido, observando las imágenes de 4 frames consecutivos de uno de los vídeos de prueba. Como podemos ver, el resultado no es admisible. Existen diversas razones por las cuáles el resultado no es bueno:

- Imagen muy ruidosa.
- Necesitamos un umbral más grande que la desviación típica.
- Los píxels que no son de fondo tienen unas desviaciones respecto a la media grandes.

Entre las posibles soluciones a estos problemas podemos realizar distintas alternativas: suavizado, utilizar un umbral mayor, etc.

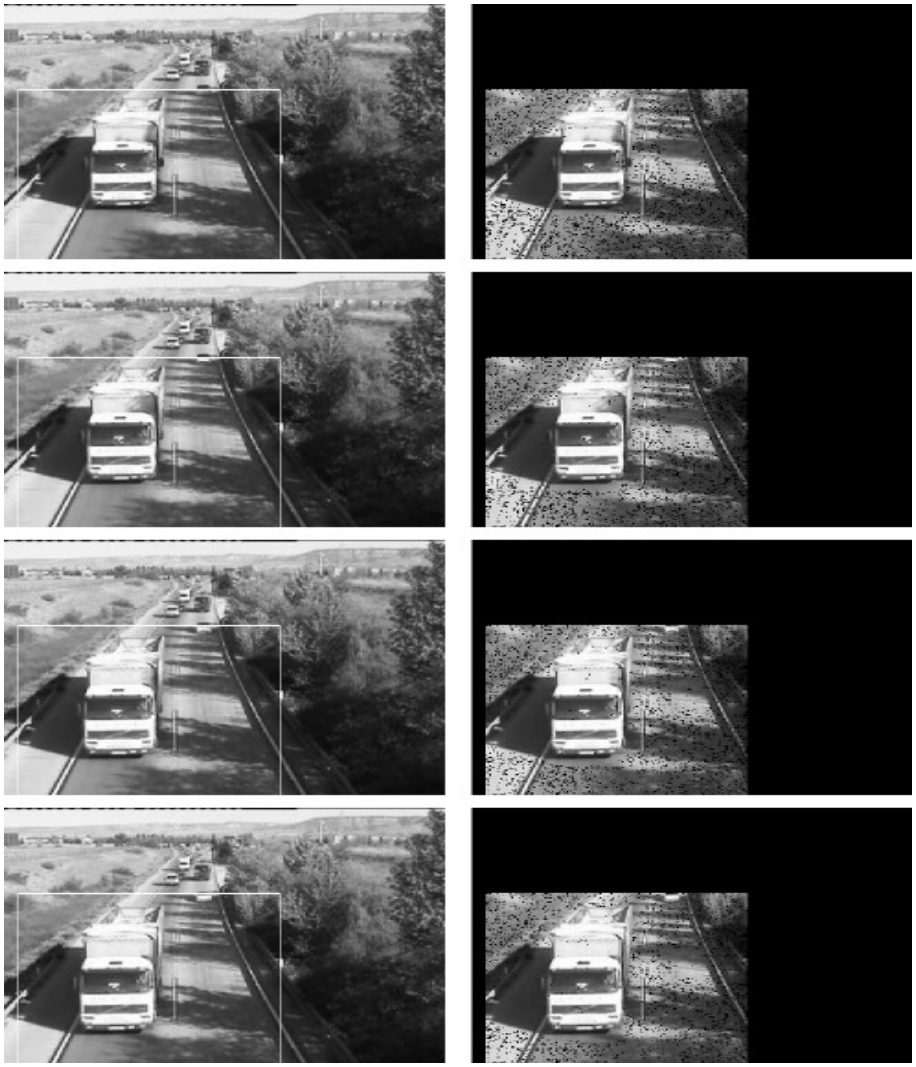

Figura 4.3: *Resultados del Primer Intento*

**Segundo Intento**

En esta ocasión, se introdujeron mejoras para poder obtener una ***Resta de Fondo*** aceptable que nos permita trabajar con una mayor facilidad en los niveles superiores del algoritmo.

Las mejoras que se han introducido son las siguientes:

- **Suavizado Gaussiano.**
- **NTU** (Número de Unidad de Textura).
- **Nuevo umbral de decisión y mejora del algoritmo**.

Con la combinación de la NTU y el análisis estadístico del valor de gris del píxel, obtendremos el fondo:


Figura 4.4: *Esquema del Decisor*

Ahora pasamos a explicar cada una de las novedades que se han incorporado al algoritmo:

- **Suavizado Gaussiano.**: Se implementó un filtro gaussiano de 5x5 para reducir el ruido y otros efectos no deseados utilizando funciones de OpenCV. Las máscaras paso bajo de suavizado producen un emborronamiento o difuminado de los detalles de la imagen como se puede ver en la figura 4.5:


Figura 4.5: *Efecto de aplicar Suavizado Gaussiano 5 x 5*

- **NTU**(Número de Unidad de Textura): El Número de Unidad de Textura nos da un indicador de la textura que presenta un píxel en una región determinada.

  Para su cálculo, utilizamos una ventana de 3x3. Para un pixel x, su NTU se calcula con la siguiente expresión:

| | | |
|---|---|---|
| 0 | 1 | 2 |
| 7 | X | 3 |
| 6 | 5 | 4 |

$$NTU_x = \sum_{i=0}^{7} I_i \cdot 2^i$$

Figura 4.6: *Cálculo de la NTU*

La NTU por sí sola, no ofrece mucho significado en esta aplicación. Pero la **diferencia** en valor absoluto de NTU para un píxel entre 2 frames consecutivos si puede ser un parámetro importante que nos ayude a clasificar los píxeles entre fondo o primer plano.

- **Nuevo umbral de decisión y mejora del algoritmo**: Para cada uno de los píxeles dentro de la ventana de trabajo se tomarán los siguientes estadísticos y parámetros:

  *Media*: $\mu$

  *Valor Cuadrático Medio*: $P_x$

  *Diferencia de NTU's*: $\Delta NTU = NTU_1 - NTU_2$

  Para poder realizar un ***umbral adaptativo***, incorporamos un parámetro más a nuestro algoritmo $\alpha$. Este nuevo parámetro toma valores en un intervalo determinado:

  $$\boxed{0.25 \leq \alpha \leq 0.6}$$

  1. Inicialmente vale 0.5.
  2. Si ese píxel es de fondo, incrementamos $\alpha$ en 0.01.
  3. Si ese píxel es de primer plano, decrementamos $\alpha$ en 0.05.
  4. Si llegamos a los valores finales 0.25 o 0.6 mantenemos el mismo valor.

  De este modo, **si un píxel pertenece al fondo**, muy probablemente este mismo píxel pertenezca al fondo en el frame siguiente. **Incrementamos el umbral**.

  Al igual que **si un píxel es de primer plano**, muy probablemente este píxel pertenezca al fondo en el siguiente frame. **Decrementamos el umbral**.

  Tras varios ajustes experimentales se llegó a la obtención de un umbral que proporciona una buena estimación del fondo para el correcto desarrollo del proyecto. Finalmente el umbral para decidir si un píxel es de fondo o de primer plano, nos queda de la siguiente manera:

$$\boxed{(x - \mu_x) < k \cdot \alpha \cdot \sqrt{P_x} \qquad y \qquad \Delta NTU < 1500}$$

1. Si se cumplen ambas condiciones: **Ese píxel pertenece al fondo**.

2. Si no se cumplen ambas: **Ese píxel pertenece al primer plano**. No se computa para el cálculo de los estadísticos del fondo.

El **parámetro k** es un parámetro de ajuste con el que podemos incrementar o decrementar el umbral. Si incrementamos *k*, el umbral será mayor por lo tanto más estricto y menos puntos pasarán a formar parte del primer plano, y viceversa si decrementamos *k*. Por defecto k = 0.78, aunque el usuario puede ajustarlo manualmente a través de la interfaz gráfica en un determinado intervalo de valores.

### 4.1.3. Esquema de Programación

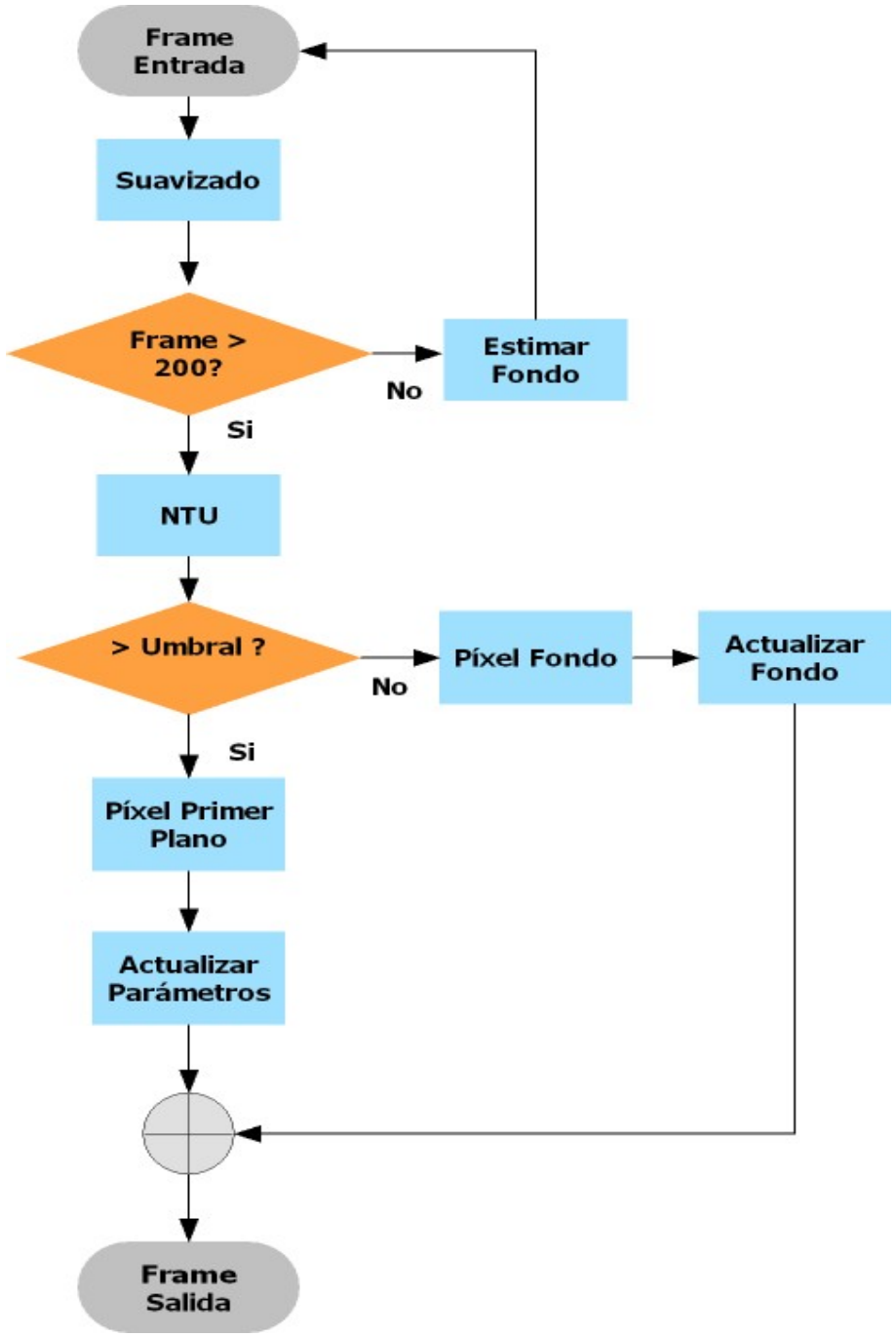


Figura 4.7: *Esquema de Programación: Resta de Fondo*

### 4.1.4. Resultados

A continuación, se muestran los resultados de la **Resta de Fondo** para tres valores distintos del *parámetro k* con el objetivo de ver la importancia que tiene este parámetro y ver como afectan sus variaciones, ya que como se ha comentado, la **Resta de Fondo** juega un papel muy importante en el resultado final de la aplicación.

Se muestran 5 frames de una de las secuencias de prueba para 3 valores distintos del *parámetro k*. Se han elegido tres valores distintos de *k* para ver sus efectos (el valor por defecto, y el valor máximo y mínimo que se le puede dar al parámetro). En la aplicación final el usuario puede ir cambiando el valor de este parámetro. Las conclusiones para cada uno de los valores de *k* son:

- **k = 0.25**: El valor del umbral es un valor muy bajo lo que conlleva que muchos píxeles pasen a formar parte del primer plano. No es un valor aceptable para realizar una correcta detección de los vehículos ya que el umbral debe ser más grande. Ver figura 4.8.
- **k = 0.78**: Es el valor por defecto del umbral, obteniendo buenos resultados. Sin embargo si por cualquier circunstancia existen efectos no deseados como por ejemplo movimiento de las hojas de los árboles, etc., se puede incrementar el umbral de tal manera que pasen menos píxeles a formar parte del primer plano. Ver figura 4.9.
- **k = 1.5**: Con este valor, tenemos un umbral alto con lo cual solamente pasan a formar parte del primer plano aquellos píxeles que tengan variaciones muy grandes con respecto a la estimación realizada del fondo. Si las condiciones ambientales son muy variables, existe mucho viento, mucho movimiento de hojas de los árboles, etc., el algoritmo consigue eliminar estos problemas, pero sin embargo algunos vehículos como en el ejemplo la motocicleta no pueden ser detectados. Ver figura 4.10.

**K = 0.25**

Figura 4.8: *Resultados Resta de Fondo: k = 0.25*

**K = 0.78**

Figura 4.9: *Resultados Resta de Fondo: k = 0.78*

**K = 1.5**

Figura 4.10: *Resultados Resta de Fondo: k = 1.5*

## 4.2. Umbralización

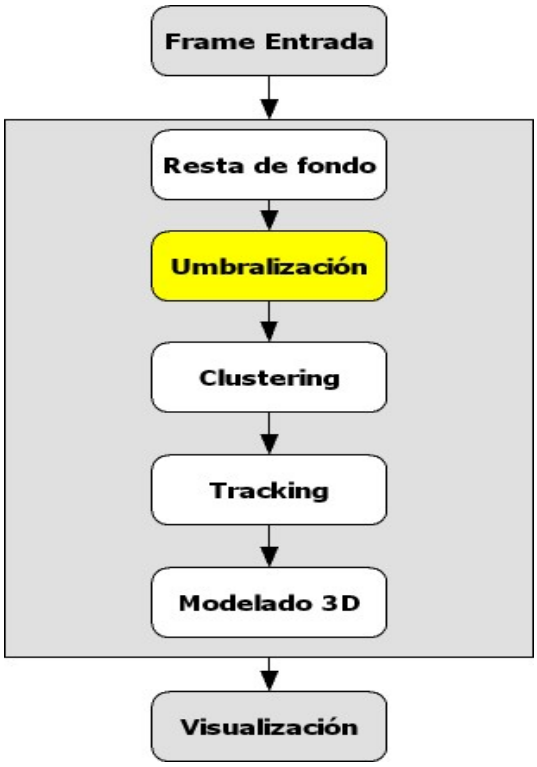


Figura 4.11: *Esquema del Algoritmo: Umbralización*

### 4.2.1. Introducción

La imagen resultante de la ***Resta de Fondo*** se compone de muchos píxeles que han sido clasificados como fondo o primer plano. Los píxeles que son de fondo, presentan un valor de gris igual a 0. Pero los píxeles pertenecientes al primer plano presentan valores de gris dentro del rango 1 - 255. Para operar con mayor comodidad en el resto de niveles del algoritmo principal, pasamos a umbralizar la imagen.

Umbralizar la imagen significa que aquellos píxeles que pertenezcan al primer plano les daremos un valor de gris de 255 (1 lógico) y aquellos píxeles que pertenezcan al fondo les daremos un valor de gris de 0 (0 lógico). De este modo en nuestra imagen umbralizada tendremos solamente dos colores o valores, blanco y negro.

**Operadores Morfológicos**

Los operadores morfológicos son operadores que se utilizan solamente con imágenes binarias, para intentar reducir efectos de ruido en la imagen o para mejorar la calidad de la misma, aplicando diferentes máscaras o núcleos según la aplicación que estemos desarrollando.

Algunos de los núcleos más utilizados los podemos ver en la figura 4.13:

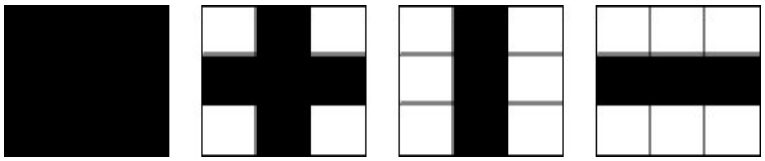

Figura 4.12: *Operadores Morfológicos: Núcleos*

Las operaciones más importantes son dos:

- **Dilatación**: Esta operación provoca un aumento del área ocupada por la figura original en la imagen tras aplicar el *kernel*.
- **Erosión**: Esta operación provoca una disminución del área ocupada por la figura original en la imagen tras aplicar el *kernel*.

En nuestra aplicación, utilizamos la siguiente máscara 3x3:

| | | |
|---|---|---|
| a | d | f |
| b | X | g |
| c | e | h |

Figura 4.13: *Operadores Morfológicos: Máscara 3x3*

Para un píxel central X, calculamos el número de unos que tenemos en su vecindad y operamos de la siguiente manera:

- Si el número de unos en la máscara es mayor que 4, ponemos el píxel X a uno.
- Si el número de unos en la máscara es menor que 4, ponemos el píxel X a cero.

De este modo si por ejemplo tenemos en el píxel central un uno, y todos sus vecinos tienen ceros, pondremos el valor del píxel central a cero ya que seguramente el que tuviera valor uno sería debido a problemas de ruido.

## 4.3. Clustering

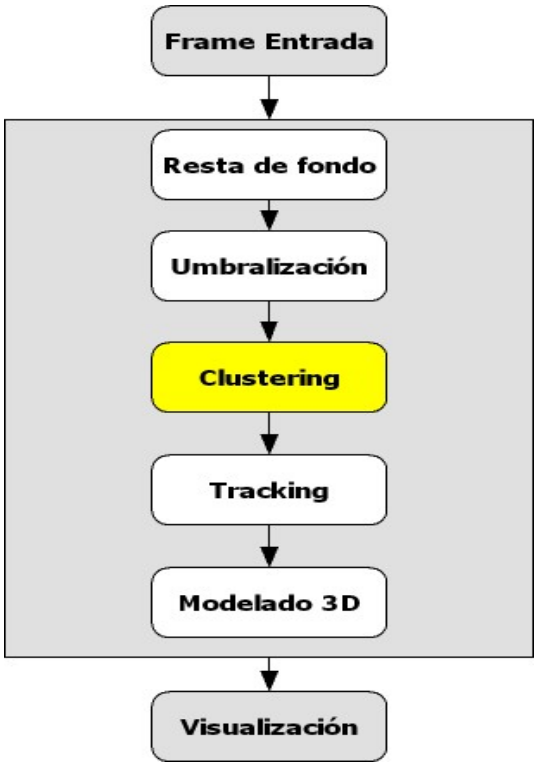


Figura 4.14: *Esquema del Algoritmo: Clustering*

### 4.3.1. Introducción

En esta fase del algoritmo tras una primera segmentación con la ***Resta de Fondo*** nos encontramos con muchos datos, muchos píxeles que hay que saber interpretar. Para poder detectar vehículos y tomar medidas sobre estos es necesario agrupar todos estos píxeles en objetos, lo que se conoce como ***Clustering***. En nuestro caso, tenemos que agrupar todos los píxeles que pertenezcan a un mismo vehículo.

Una vez que tengamos un conjunto de píxeles agrupados como un vehículo, podemos tomar medidas sobre al área del conjunto o ***cluster***, momentos de primer orden, de segundo orden, orientación... Por lo tanto en nuestra aplicación que se centra en la detección y el seguimiento de vehículos, el clustering juega un papel fundamental.

Para la clasificación, se pueden usar diversas características de cada píxel, como por ejemplo:

- Intensidad en distintos espacios de color.

- Intensidad o valor de gris.
- Nivel de gris en otras imágenes.
- Características locales como varianza.
- Proximidad, entornos de vecindad.
- Bordes, líneas, contornos, formas, ángulos...

Además, dependiendo del tipo de aplicación se pueden fijar de antemano un número fijo de clases o no.

Sin embargo el clustering o el agrupamiento no es tan sencillo como parece ya que hay que tener en cuenta muchos aspectos y problemas. Algunos de ellos pueden ser:

- Agrupar varios vehículos en uno sólo.
- Obtener clusters que sean acordes con vehículos, no es posible obtener un cluster que tenga mucho más ancho que alto por ejemplo.
- Los píxeles pertenecientes a las sombras, pueden ocasionar problemas. Por ejemplo, podemos tener una imagen en la que aparezcan varios vehículos unidos por puntos de sombra.
- Clusters o conjuntos muy pequeños pueden ser debidos a que la imagen presenta mucho ruido.

Existen muchas técnicas de clustering, la elección de una u otra depende principalmente de la aplicación que se quiera desarrollar. El algoritmo de clustering desarrollado para este proyecto es un híbrido entre varios algoritmos de clustering, diseñado con el objetivo de la detección de vehículos.

A continuación se comentan brevemente dos algoritmos muy populares de clustering. No obstante, si se quiere profundizar en el tema se puede buscar bien en Internet o en la bibliografía la referencia [1].

### KNN (K Nearest Neighbour)

Es un algoritmo muy sencillo pero que presenta buenos resultados. Básicamente se compone de los siguientes pasos:

1. Partimos de un conjunto de puntos.
2. Se define una distancia entre cada uno de los puntos que componen nuestro conjunto inicial. La distancia puede ser calculada mediante la distancia euclídea o cualquier otro tipo de distancia.

$$\boxed{d_{x,y} = \sqrt{(x_i - y_i)^2 + (x_j - y_j)^2}}$$

3. Con las distancias anteriores se construye una tabla de distancias.
4. Según la tabla de distancias se clasifica cada uno de los puntos del conjunto inicial en *k* conjuntos o vecinos.

### K-Medias (K Means)

En este algoritmo de clustering se suponen conocidos los centros iniciales de cada uno de los *k* clusters. Se tiene también una función discriminante o función distancia al igual que en el algoritmo anterior. El algoritmo tiene los siguientes pasos:

1. Partimos de un conjunto inicial de puntos.
2. Se escogen *k* puntos para que sean los centros de los clusters.
3. Se itera el algoritmo hasta que los centros de los clusters no cambien.
   *a*) Se coloca cada punto del conjunto en el cluster aquel que la distancia entre el centro del cluster y el punto sea mínima.
   *b*) Hay que asegurarse de que cada uno de los *k* clusters tengan algún elemento, es decir, que no se encuentren vacíos.
   *c*) Se reemplazan los centros de los clusters por la media de cada uno de los puntos de cada uno de los clusters.
4. Una vez que el algoritmo converja, ya tendremos clasificados cada uno de los puntos en los distintos *k* conjuntos o clusters.

### 4.3.2. Algoritmo Desarrollado

El algoritmo de **clustering** desarrollado consta de dos fases principales:

- Una primera fase de agrupamiento o clustering donde se agrupan los píxeles en los distintos clusters.
- Una segunda fase de **Limpieza de Clusters** en la que alguno de los clusters procedentes de la fase anterior son descartados ya que no cumplen algún requisito.

El algoritmo desarrollado es similar al **KNN** pero teniendo en cuenta una serie de consideraciones importantes, ya que los clusters que se formen deben corresponderse con vehículos. Algunas de estas consideraciones son:

- **Perspectiva**: Hay que tener en cuenta que el tamaño de los vehículos va aumentando según se acercan en la cámara, lo que en la imagen se corresponde con un mayor tamaño según aumenta la coordenada y en píxeles.
- **Separación en x,y**: A la hora de calcular la distancia de un píxel del cluster a otro píxel de los que queden sin agrupar, hay que dar un margen de valores en las coordenadas x e y. Por ejemplo, si estamos calculando la distancia entre la esquina superior derecha de un cluster y un píxel que se encuentra a la izquierda una **distancia x** no demasiado grande y en una **distancia y** distinta, este punto deberá pertenecer al mismo cluster ya que es el mismo vehículo. También hay que tener cuidado con la separación en x, ya que si se da mucho margen, debido a la presencia de las sombras, si estas son muy pronunciadas se pueden agrupar dos vehículos en un mismo cluster.
- **Tamaño de Cluster**: Solamente serán válidos aquellos clusters que tengan un tamaño mínimo. Los que no cumplan esta condición serán descartados en el proceso de limpieza.
- **Relación Ancho/Alto de Cluster**: Esta relación tiene que estar dentro de un rango de valores, ya que no existen vehículos mucho más anchos que altos y viceversa.

#### Clustering

El algoritmo de clustering consta de los siguientes pasos:

1. Inicialmente tenemos un **conjunto de píxeles no vacío** procedentes de los niveles inferiores del sistema (Resta de fondo y Umbralización).
2. Se crea un cluster a partir del primer punto del conjunto de píxeles. Cada vez que se incorpore un píxel a un cluster, se elimina este punto del conjunto de píxeles.
3. Se calcula la distancia entre el último píxel perteneciente al cluster y el siguiente píxel del conjunto. Se calculan individualmente las distancias en x,y en valor absoluto:

$$\boxed{d_x = |x_{cluster} - x_{conjunto}|} \qquad \boxed{d_y = |y_{cluster} - y_{conjunto}|}$$

4. Una vez que se han calculado las distancias, se calculan los límites para las coordenadas x,y. Es decir, el píxel del conjunto pertenecerá al cluster, si las distancias entre el píxel del conjunto y el del cluster están dentro de unos márgenes. En función de la coordenada y se dejan los siguientes márgenes de distancia en x y en y:

   - $y < 80: \ k_x = 10 \, ; \, k_y = 2$
   - $y < 120: \ k_x = 20 \, ; \, k_y = 4$
   - $y \geq 120: \ k_x = 30 \, ; \, k_y = 8$

   Mientras no se cambie de coordenada y, a la vez que se van añadiendo píxeles al cluster, se va estimando para una y determinada el ancho o la longitud en x. Si la coordenada y es distinta entre el píxel del cluster y el píxel del conjunto, el márgen de distancia en x se incrementa en la estimación del ancho para esa coordenada y. Esto se realiza para solventar el problema de que por ejemplo tengamos un píxel del cluster que se corresponda con un borde del vehículo en el lado derecho y el siguiente píxel de la lista aparezca a la izquierda de este último, por lo que se debe incrementar el márgen en la coordenada x.

5. Se itera el algoritmo anterior hasta que se llegue al final del conjunto de píxeles. Si todavía quedan píxeles sin agrupar, se vuelve a repetir el algoritmo creando un nuevo cluster.

6. El algoritmo finaliza cuando no queda ningún píxel del conjunto inicial sin agrupar en ningún cluster.

#### Medidas sobre los Clusters

Una vez que se han formado los clusters, se toman medidas de parámetros sobre estos. Luego estas medidas serán utilizadas en el proceso de limpieza de clusters. Las medidas que se toman sobre cada uno de los clusters son:

- **Área**: El número total de píxeles que componen el cluster.
- **Centroide**: Coincide con el centro de masas de un objeto en el que cada uno de los píxeles aporta la misma masa. Es la media en x e y de las coordenadas del cluster.

$$\boxed{\overline{x} = \frac{1}{N} \cdot \sum_{x \in Cluster} x} \qquad \boxed{\overline{y} = \frac{1}{N} \cdot \sum_{y \in Cluster} y}$$

- **Estimación del Alto y Ancho**: A partir de los primeros y últimos píxeles de cada cluster se realiza una estimación del alto y ancho del mismo.
- **Momentos Centrales Normalizados**: Se calculan los momentos centrales normalizados $\mu_{11}$, $\mu_{20}$ y $\mu_{02}$ a partir de la siguiente expresión:

$$\boxed{\mu_{p,q} = \sum_x \sum_y x^p \cdot y^q}$$

- **Orientación**: Se calcula la orientación del cluster a partir de los momentos anteriores:

$$\boxed{\theta = \frac{1}{2} \cdot arg(\frac{2 \cdot \mu_{11}}{\mu_{20} - \mu_{02}})}$$

**Limpieza de Clusters**

En esta etapa hay que comprobar cuales de los clusters anteriores son válidos y cuáles no. Para ello se realiza un proceso de limpieza que básicamente consiste en:

- **Tamaño**: Si el cluster tiene un tamaño muy pequeño en comparación con el resto, lo eliminamos ya que seguramente no corresponda a ningún vehículo, si no posiblemente a píxeles sueltos o de ruido o sombras.
- **Relación Alto/Ancho**: Esta relación debe estar comprendida entre un valor mínimo y un valor máximo. Si la relación no se encuentra entre estos valores, el cluster se elimina.

Una vez acabado este proceso, los clusters que no hayan sido descartados pasan a los niveles superiores de **Modelado 3D** y **Tracking**.

### 4.3.3. Esquema de Programación

**Clustering**

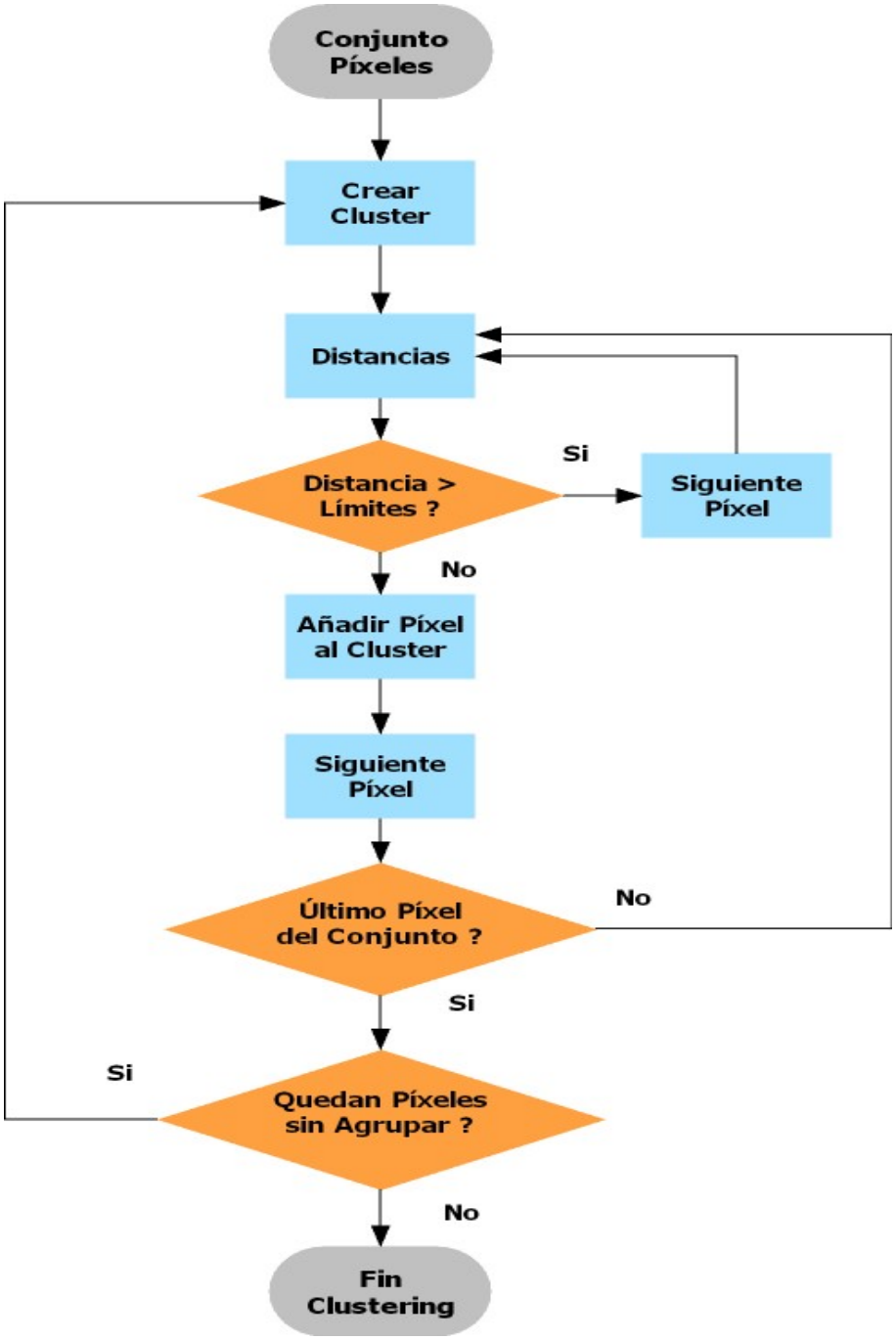


Figura 4.15: *Esquema de Programación: Clustering*

**Limpieza de Clusters**

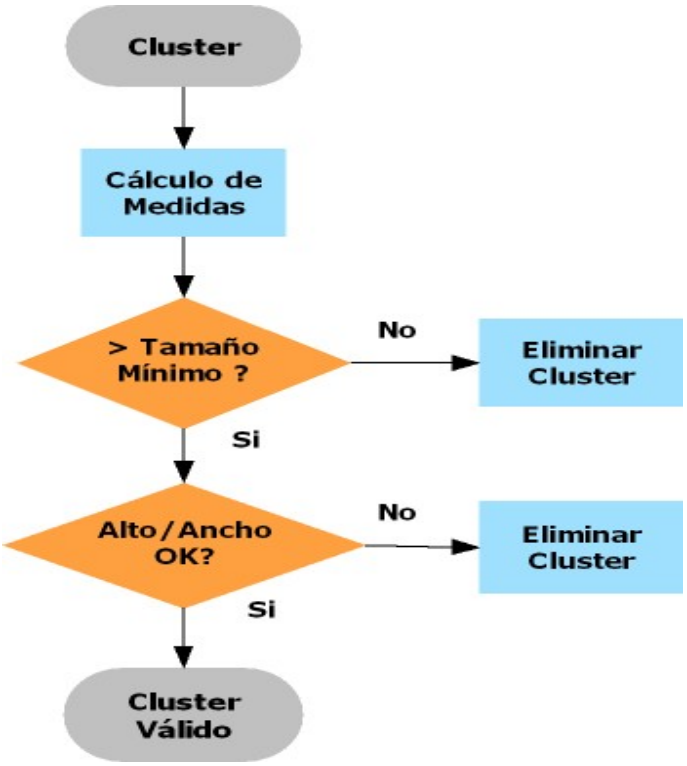


Figura 4.16: *Esquema de Programación: Limpieza de Clusters*

## 4.4. Tracking

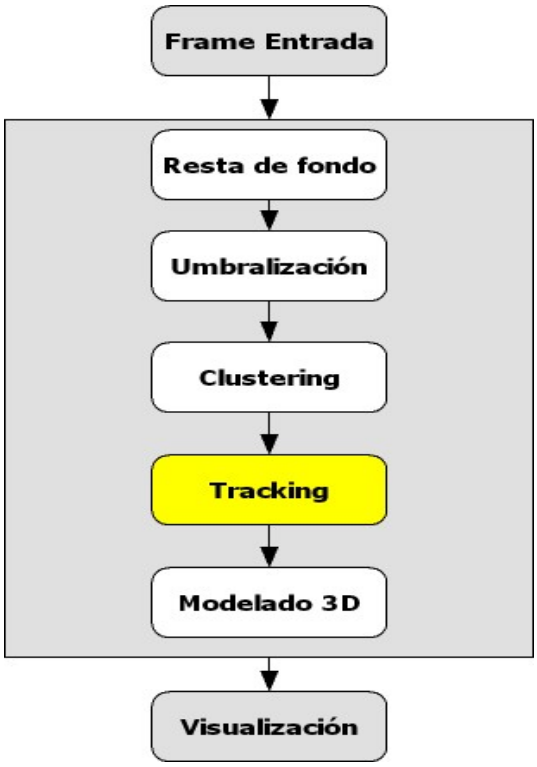


Figura 4.17: *Esquema del Algoritmo: Tracking*

### 4.4.1. Introducción

En esta fase del algoritmo se realiza un seguimiento a cada uno de los centroides de los clusters. A la hora de realizar el seguimiento se utiliza el llamado **Filtro de Kalman** para corregir variaciones e imprecisiones en los datos obtenidos procedentes de los clusters.

Sin embargo, lo más complicado de esta fase consiste en **relacionar o emparejar a cada cluster con su respectivo *tracking***. Llamamos **tracking** a una estructura de datos que contiene la posición del centroide tras aplicar el Filtro de Kalman. Ya que podemos tener algunos problemas como por ejemplo:

- En un frame determinado un cluster al que se estaba realizando un seguimiento desaparezca.
- En un frame determinado un cluster al que se estaba realizando un seguimiento se encuentre asociado a dos vehículos.

- Debido a la presencia de sombras o de cambios de iluminación en la escena, el centroide de un cluster entre un frame y el siguiente se encuentre muy desplazado.

A partir de un único punto en 2D en este caso el centroide de cada cluster, se genera posteriormente el modelo 3D completo. Por lo tanto, **el vector de estado del Filtro de Kalman consistirá en la posición *x,y* correspondiente al centroide de cada cluster**:

$$Vector\ Estado = \begin{pmatrix} x_{centroide} \\ y_{centroide} \end{pmatrix}$$

**Filtro de Kalman**

El **Filtro de Kalman** es un algoritmo desarrollado por *Rudolf Emil Kalman* basado en variables de estado que se utiliza para poder estimar el estado de un sistema dinámico lineal cuando esté se encuentra sometido a ruido blanco aditivo.

En el caso de un sistema discreto, el modelo en variables de estado presenta las siguientes ecuaciones:

$$x_{k+1} = A \cdot x_k + B \cdot u_k + w_k$$

$$y_k = C \cdot x_k + D \cdot u_k + v_k$$

Siendo en las ecuaciones anteriores:

- $x_k$: el vector de estado.
- $y_k$: la salida del sistema.
- $u_k$: la entrada del sistema.
- A, B, C, D: las matrices características del sistema.
- $w_k$ ruido blanco de media cero y de varianza $Q_k$, asociado a incertidumbres del modelo y a las variables.
- $v_k$ ruido blanco de media cero y de varianza $Q_k$, asociado a ruido de medida.
- En la mayoría de los sistemas, la salida no depende directamente de la entrada, por lo que la matriz C suele considerarse nula.

El Filtro de Kalman nos permite estimar el estado $x_k$ a partir de las mediciones anteriores de u, y, Q, V y las estimaciones anteriores de x a través de una fase de **predicción** y otra de **corrección**. Se puede consultar con mayor detalle la teoría del Filtro de Kalman en la referencia [5].

### 4.4.2. Algoritmo Desarrollado

Como se ha comentado el principal problema de esta fase es emparejar cluster y *tracking*, ya que las funciones que realizan el **Filtro de Kalman** no tienen demasiado problema al estar el filtro implementado en **OpenCV**. Por lo tanto la principal dificultad será emparejar correctamente cluster y *tracking*. El algoritmo consta de las siguientes fases:

1. **Fase de Iniciación**: Para cada uno de los clusters nuevos que llegan a esta fase se inicia el Filtro de Kalman creando lo que llamaremos *tracking* por cada uno de los clusters. Una vez que el filtro ha sido iniciado se procede al seguimiento de cada uno de los clusters.

2. Una vez que el filtro ha sido inicializado en los siguientes frames hay que **emparejar cada uno de los clusters con su respectivo *tracking***. Por cada uno de los *tracking* que tengamos se buscará aquel cluster cuya distancia entre ambos sea menor que un umbral determinado que depende de la coordenada *y* del centroide. Esta umbral dinámico es el siguiente:

$$\boxed{Umbral_{\,distancia} = y_{\,Tracking} \,/\, 8}$$

3. Se utiliza una variable llamada **CL (Cluster Lost)** que se incrementa cuando no se encuentra el cluster correspondiente.

   *a*) **Si se encuentra el cluster asociado**, se realiza una actualización del Filtro de Kalman con los datos del cluster. Se pone la variable CL a 0 para ese *tracking*.

   *b*) **Si no se encuentra el cluster asociado**, se realiza una estimación de la posición y se actualiza el Filtro de Kalman. A su vez se incrementa la variable CL en una unidad.

4. El seguimiento se sigue realizando a cada uno de los clusters hasta que alguno llegue al **Fin de Tracking** y se finalice el Filtro de Kalman liberando su respectivo *tracking*. Se puede llegar al Fin de Tracking por las siguientes razones:

   *a*) La variable CL es superior a 10.

   *b*) La posición de alguno de los *tracking* está fuera de los límites de la ventana de trabajo.

### 4.4.3. Esquema de Programación

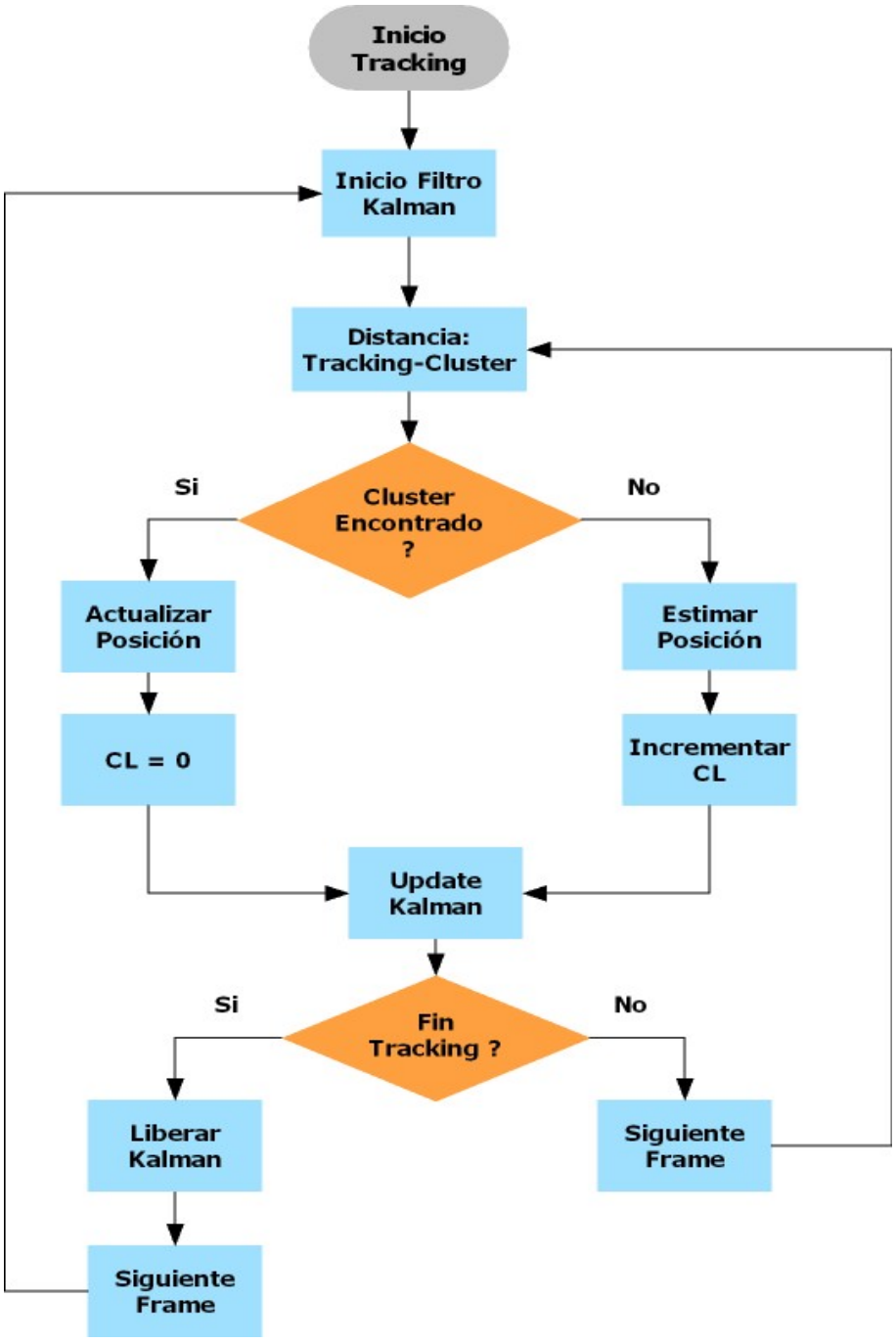


Figura 4.18: *Esquema de Programación: Tracking*

## 4.5. Modelado 3D

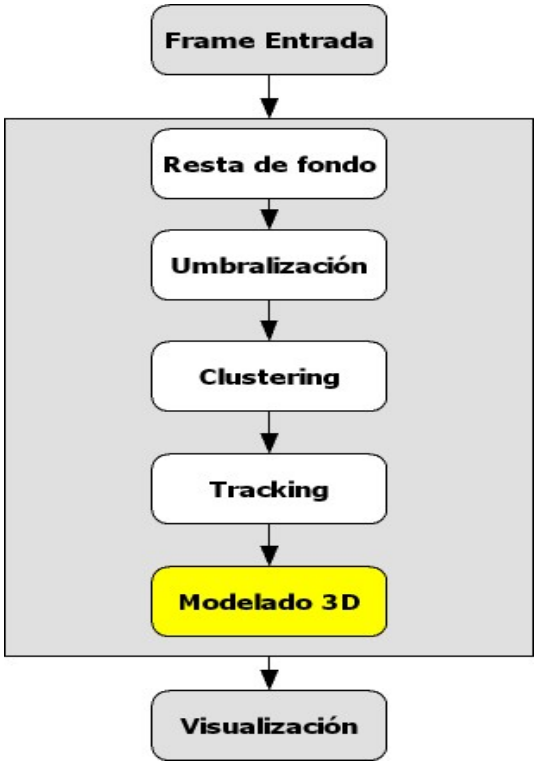


Figura 4.19: *Esquema del Algoritmo: Modelado 3D*

### 4.5.1. Introducción

Después de las fases anteriores, se llega a la fase final del algoritmo: el **Modelado 3D** y su posterior visualización. Los puntos del mundo 3D se traducen en proyecciones 2D a través de la óptica de la cámara.

El problema es que a partir de un punto en 2D tenemos que construir un modelo en 3D, con lo cuál se hace indispensable conocer los parámetros intrínsecos y extrínsecos de la cámara con la que hemos trabajado, es decir realizar el **calibrado de la cámara**.

Por **calibrado** entendemos el proceso de estimar los parámetros ópticos propios de la cámara (**parámetros intrínsecos**) y su posición y orientación en el mundo real (**parámetros extrínsecos**). Sin realizar el calibrado no podremos obtener relaciones métricas del mundo real a partir de las imágenes de la cámara.

Una vez que tengamos calibrada la cámara, hay que proyectar el píxel en 2D en el

mundo 3D y obtener sus coordenadas x,y,z. A partir de este punto en 3D y suponiendo conocidas las dimensiones de los vehículos se construye el modelo en 3D, para finalmente volver a proyectarlo en 2D.

El objetivo que se persigue es que en la aplicación final, alrededor de los vehículos que han sido detectados se muestre el modelo 3D proyectado. Esta es una manera original y elegante de presentar los resultados finales.

### 4.5.2. Calibración de la Cámara

El calibrado de la cámara se ha realizado de manera empírica a través de ensayos de prueba y error. No se ha utilizado ningún método de calibración específico. Si no que se ha ido probando con distintos valores de parámetros hasta que los resultados fueron coherentes, por lo tanto habrá ciertos errores debidos a la calibración. El modelo de cámara utilizado ha sido el modelo **pin-hole**. Se puede ver en la figura 4.20 el planteamiento del problema:

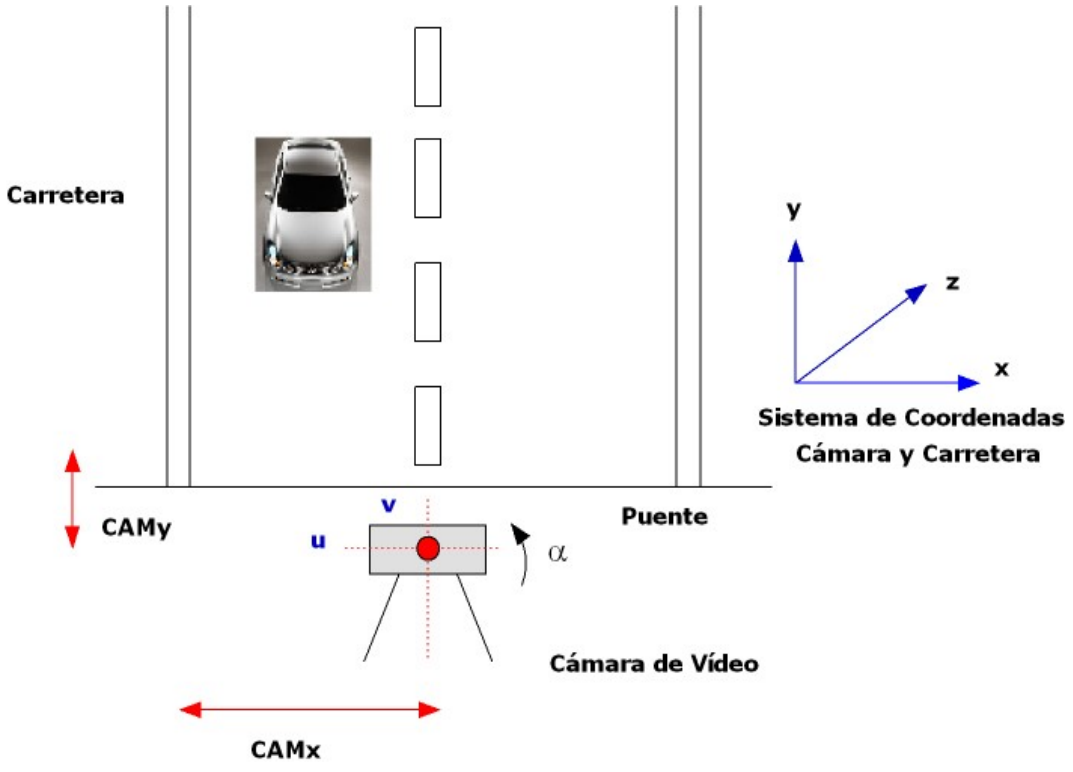


Figura 4.20: *Planteamiento del Problema*

**Parámetros intrínsecos de la cámara**:

- **Distancias Focales**: FU = 400, FV = -700.
- **Desplazamiento en píxels**: $u_0$ = -160, $v_0$ = -120.

**Parámetros extrínsecos de la cámara**:

- **Desplazamiento en x**: CAMx = -6 m.
- **Desplazamiento en y**: Diferencia de alturas entre la carretera y la cámara. CAMy = -5 m.
- **Matriz de Rotación R**: Sólo rotamos en el eje x un ángulo $\alpha$ o ángulo de elevación. $\alpha$ = -10.2º.

Aplicando Geometría Proyectiva, la **Matriz de Proyección** resultante tiene la siguiente expresión:

$$\begin{pmatrix} \lambda \cdot u \\ \lambda \cdot v \\ \lambda \end{pmatrix} = \begin{pmatrix} FU & 0 & -u_0 \\ 0 & FV & -v_0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & CAMx \\ 0 & cos\alpha & -sen\alpha & CAMy \\ 0 & sen\alpha & cos\alpha & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Y despejando en la expresión anterior obtenemos las coordenadas de los píxeles u,v en el plano imagen. Usaremos estas ecuaciones para obtener la **proyección de los puntos en el mundo 3D en el mundo 2D**. ($3D \Rightarrow 2D$).

$$u = \frac{FU \cdot (x + CAMx)}{sen\alpha \cdot y + cos\alpha \cdot z} - u_0$$

$$v = \frac{FV \cdot (cos\alpha \cdot y - sen\alpha \cdot z + CAMy)}{sen\alpha \cdot y + cos\alpha \cdot z} - v_0$$

A partir de las ecuaciones anteriores se pueden obtener las ecuaciones de **Proyección Inversa**. Usaremos estas ecuaciones para obtener la **proyección de los puntos en el mundo 2D en el mundo 3D**. ($2D \Rightarrow 3D$).

$$x = \frac{(sen\alpha \cdot y + cos\alpha \cdot z) \cdot (u + u_0) - FU \cdot CAMx}{FU}$$

$$z = \frac{FV \cdot (cos\alpha \cdot y + CAMy) - y \cdot (v + v_0) \cdot sen\alpha}{(v + v_0) \cdot cos\alpha + FV \cdot sen\alpha}$$

En las ecuaciones anteriores no aparece como incógnita el valor de la coordenada **y**, ya que esta es la única coordenada conocida, puesto que es la altura a la que se encuentra el centroide de cada uno de los vehículos.

### 4.5.3. Modelo 3D

Las dimensiones de los modelos de los vehículos en 3D **se suponen más o menos conocidas** aunque pueden diferir algo con respecto a las dimensiones que tienen los vehículos en la realidad. Las dimensiones de cada uno de los vehículos son:

| **Vehículo** | $\Delta X$ | $\Delta Y$ | $\Delta Z$ |
|---|---|---|---|
| *Turismos* | 1.6 m | 0.8 m | 3 m |
| *Camiones* | 2.5 m | 1.6 m | 7 m |

Se genera un modelo 3D para cada uno de los clusters resultantes, a partir del vector de estado obtenido a la salida del Filtro de Kalman que se corresponde con el centroide filtrado de cada uno de los clusters.

Podemos ver una aproximación del resultado que se persigue en la figura 4.21

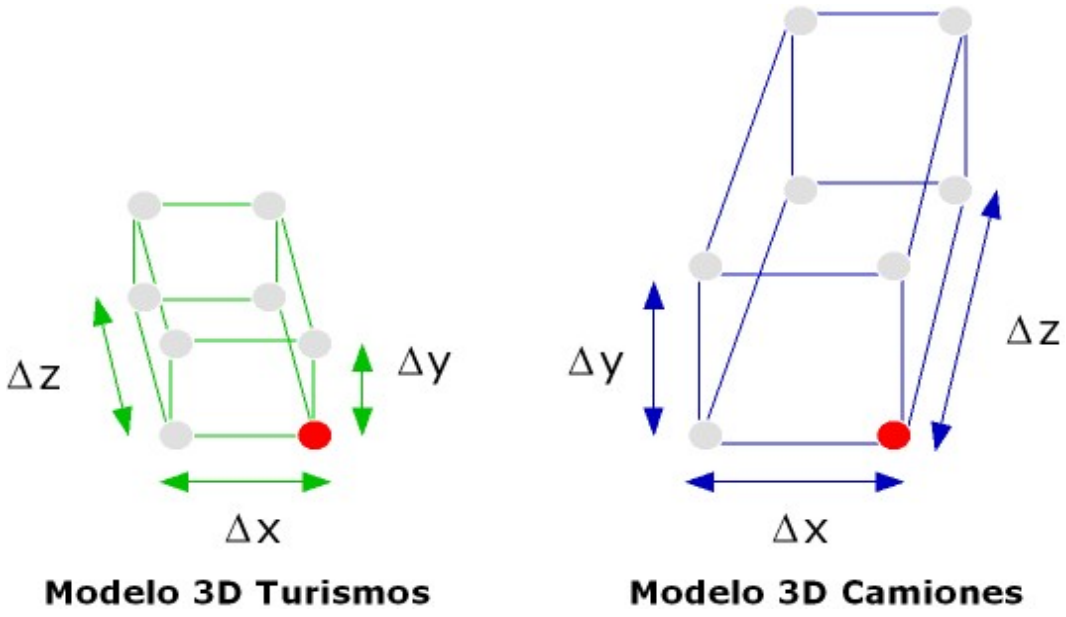


Figura 4.21: *Modelos 3D de Vehículos*

Los pasos necesarios para generar el modelo 3D son los siguientes:

1. **Clasificación**: Se compara el tamaño del cluster con un umbral para clasificarlo como turismo o como camión.

2. **Proyección Centroide 2D-3D**: Se proyecta el centroide a 3D a partir de las **Ecuaciones de Proyección Inversa**. Dependiendo de si el cluster se ha clasificado como camión o como turismo la **coordenada y** es distinta:

   *a*) **Turismo**: $y = 0{,}5\,m$

   *b*) **Camión**: $y = 0{,}7\,m$

3. **Generar Modelo 3D**: Una vez que se ha obtenido la proyección en 3D del centroide, calculamos la posición en 3D de la **esquina inferior derecha**. A partir de este punto generamos el resto del modelo según las dimensiones del vehículo que corresponda. La elección de este punto es arbitraria, se puede construir el modelo por cualquier otro punto que se elija. En total, sólo es necesario conocer las coordenadas en 3D de 8 puntos (vértices) ya que estamos trabajando con modelos prismáticos.

4. **Proyección Modelo 3D-2D**: Para cada uno de los 8 puntos anteriores se aplica la **Matriz de Proyección** para proyectar cada uno de los puntos a 2D.

5. **Visualización**: Se unen los puntos del modelo en 2D mediante líneas. En función del tipo de vehículo el modelo tendrá un color diferente:

   *a*) **Azul** si el vehículo es clasificado como **Camión**.

   *b*) **Verde** si el vehículo es clasificado como **Turismo**.

En principio el sistema solamente distingue entre turismos y camiones, aunque también podría clasificar vehículos como motos, pero debido a la falta de motos en los vídeos de prueba, esta opción no está completamente depurada.

### 4.5.4. Esquema de Programación

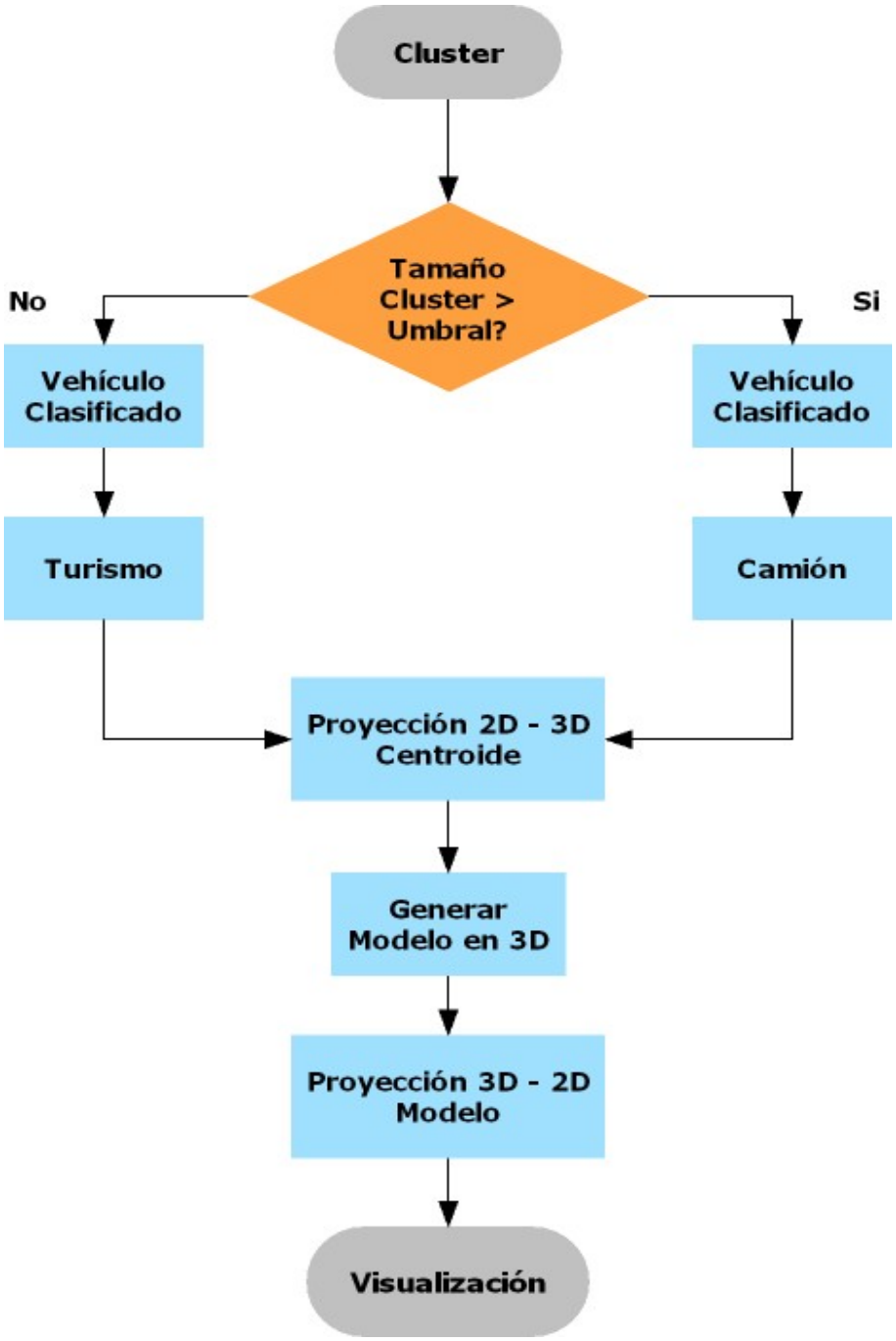


Figura 4.22: *Esquema de Programación: Modelado 3D*

# Capítulo 5

# Resultados

En este apartado se muestran las secuencias de los vídeos de salida de la aplicación, y se muestran resultados sobre el número de vehículos detectados, posición, velocidad y se miden los tiempos de ejecución para cada una de las secuencias, para poder tener una aproximación de la potencia de cálculo necesaria si hubiera que implementar la aplicación en tiempo real.

En el CD adjunto con el proyecto se incluyen los vídeos de salida para poder ver los resultados sin necesidad de ejecutar la aplicación.

Se han analizado tres secuencias de vídeo todas ellas correspondientes a la *Nacional II* que une Madrid y Barcelona a la altura de *Alcalá de Henares* a las 5 de la tarde de un día laborable durante el mes de Octubre. Las principales características de cada una de las secuencias son:

- **Secuencia 1**: Vista frontal de los vehículos, presencia de vegetación y sus respectivas sombras, tráfico con turismos y numerosos camiones de grandes dimensiones y cabe destacar la presencia de una motocicleta.
- **Secuencia 2**: Mismas características que la secuencia anterior.
- **Secuencia 3**: Vista trasera de los vehículos, sin apenas vegetación y tráfico con turismos y camiones.

Para cada una de las secuencias anteriores se muestran los siguientes resultados:

- **Secuencias de Salida**: Se muestran 30 frames de la secuencia de salida.
- **Número de Vehículos Detectados por Frame**: El número de vehículos detectados por cada uno de los frames del vídeo. El número de vehículos detectados se corresponde con el número de **clusters** obtenidos.

- **Tiempo de Procesado por Frame**: El tiempo de cómputo que lleva el algoritmo completo por frame.
- **Posición**: Gráficas de la posición del centroide de los vehículos.
- **Velocidad**: Histograma para poder ver la velocidad media de los vehículos en cada uno de los vídeos.
- **Porcentajes de Detección**: Se calcula mediante inspección visual el número de vehículos en cada uno de los vídeos, y posteriormente calculamos el número de vehículos que han sido detectados. Por vehículo detectado se entiende aquel vehículo que presente un modelo 3D sobre él durante al menos unos frames consecutivos.

Los resultados dependen de 2 factores que el usuario puede modificar durante el transcurso de la aplicación:

- **Ventana de Trabajo**: En las secuencias que se muestran a continuación se ha escogido una ventana de trabajo lo suficientemente grande para poder comprobar el funcionamiento del algoritmo.
- **Parámetro k**: Por defecto se ha escogido k = 0.78, aunque en las secuencias 1 y 2, el resultado puede mejorar si se incrementa el valor de k.

## 5.1. Secuencia 1

### 5.1.1. Secuencias de Salida

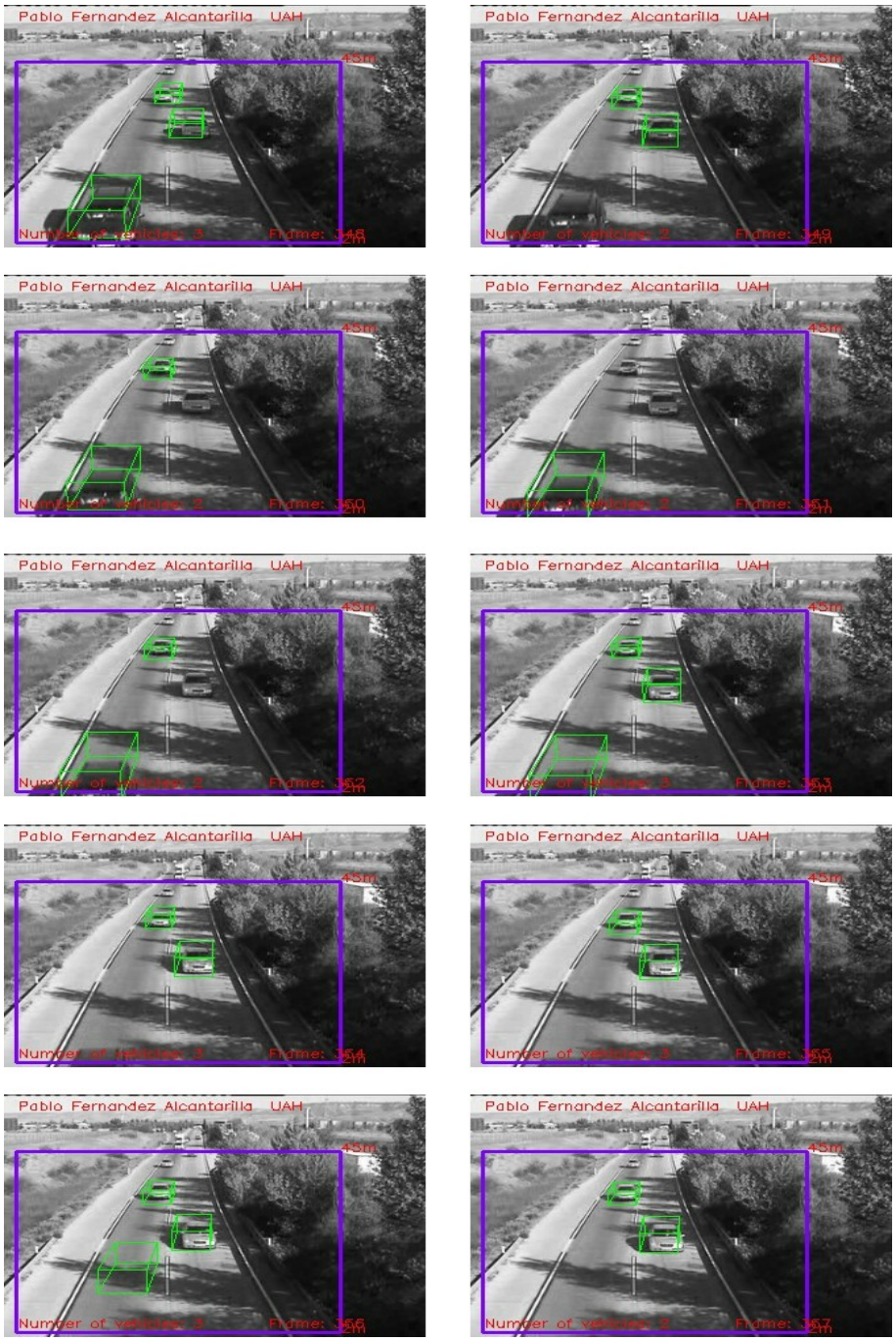


Figura 5.1: *Secuencia de Salida 1: Frames 348 - 357*

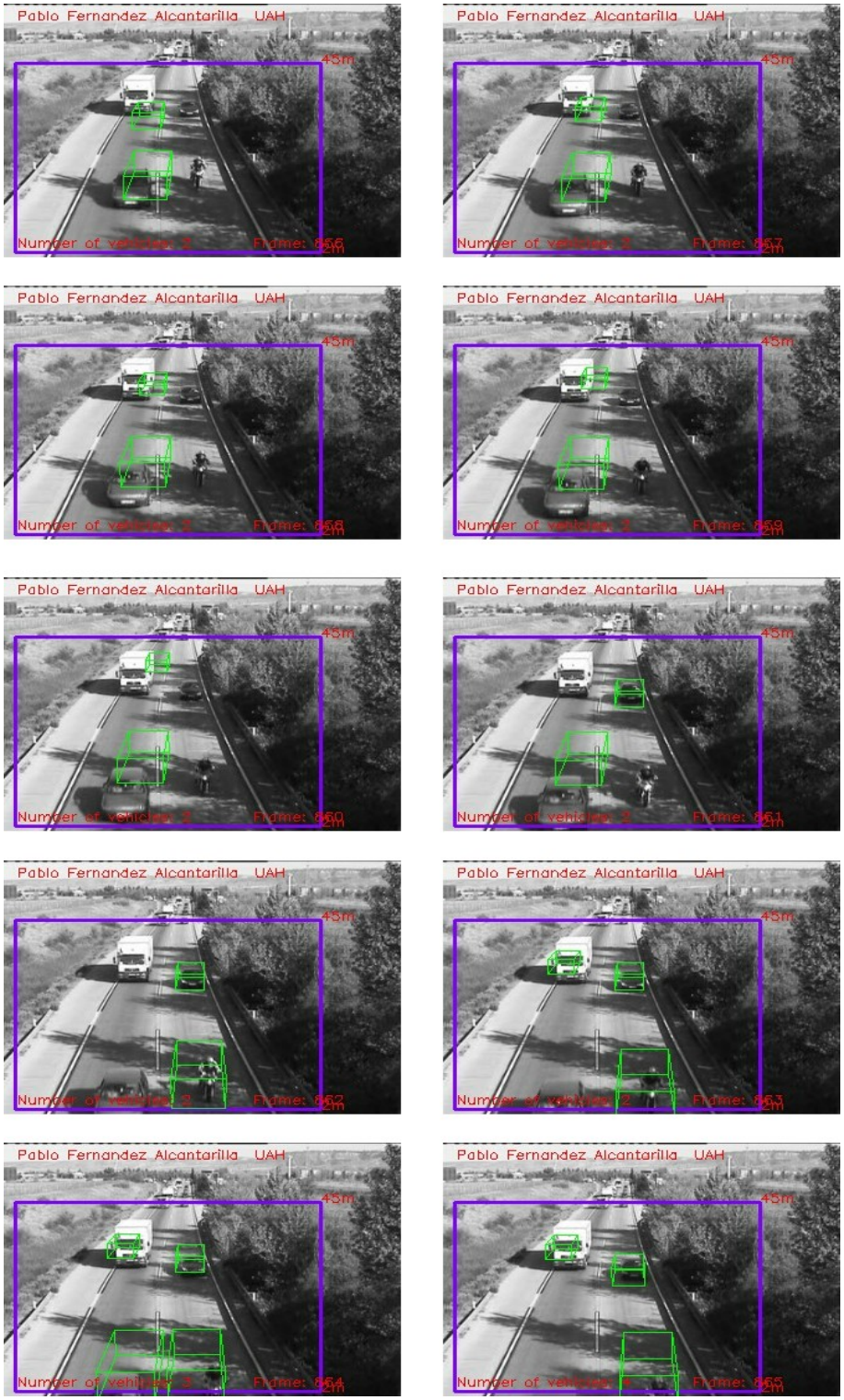


Figura 5.2: *Secuencia de Salida 1: Frames 856 - 865*

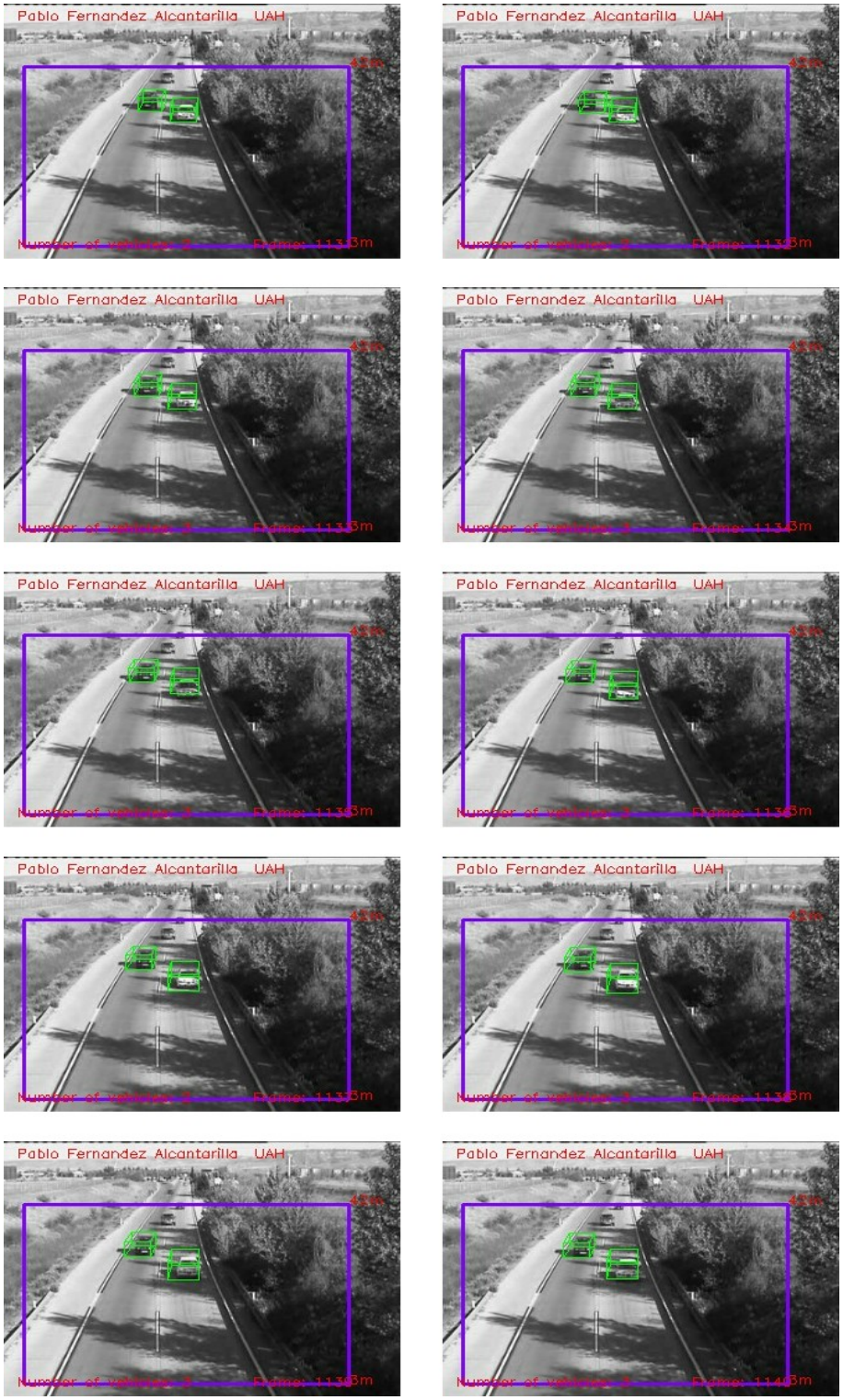


Figura 5.3: *Secuencia de Salida 1: Frames 1131 - 1140*

### 5.1.2. Número de Vehículos Detectados y Tiempo de Procesado

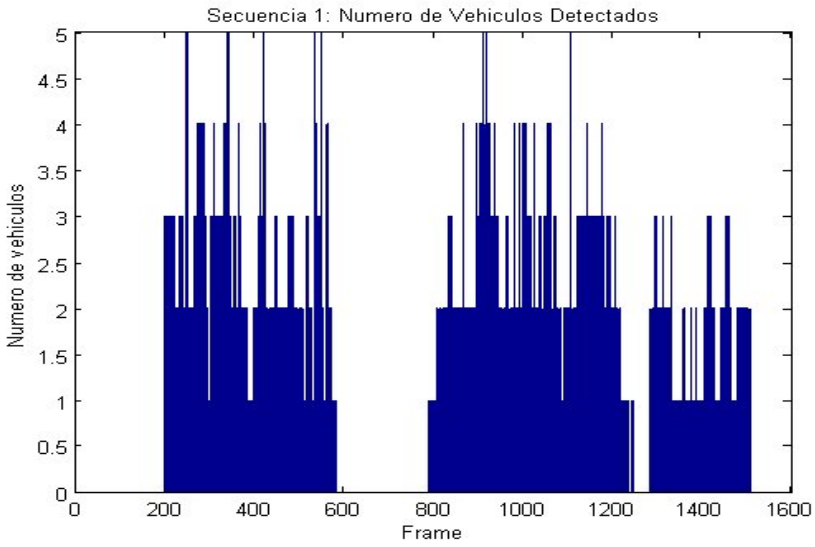


Figura 5.4: *Secuencia 1: Número de Vehículos Detectados*

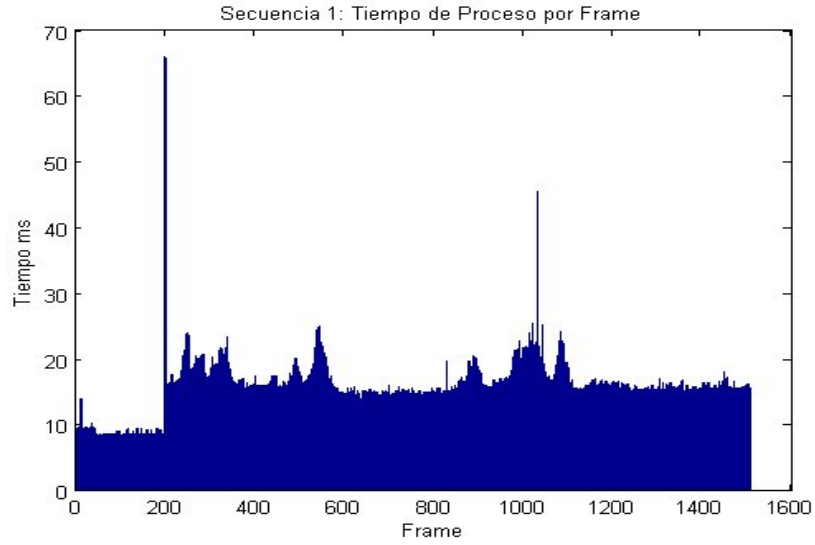


Figura 5.5: *Secuencia 1: Tiempo de Procesado*

### 5.1.3. Posición y Velocidad

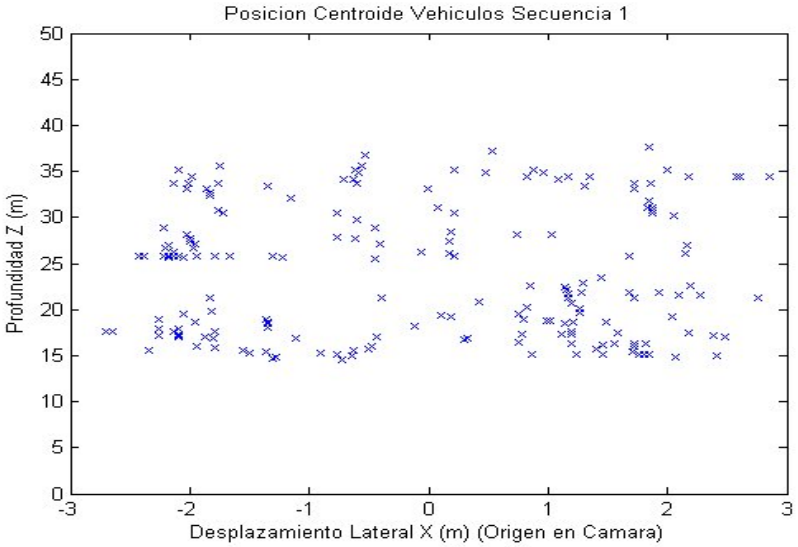


Figura 5.6: *Secuencia 1: Posición del Centroide*

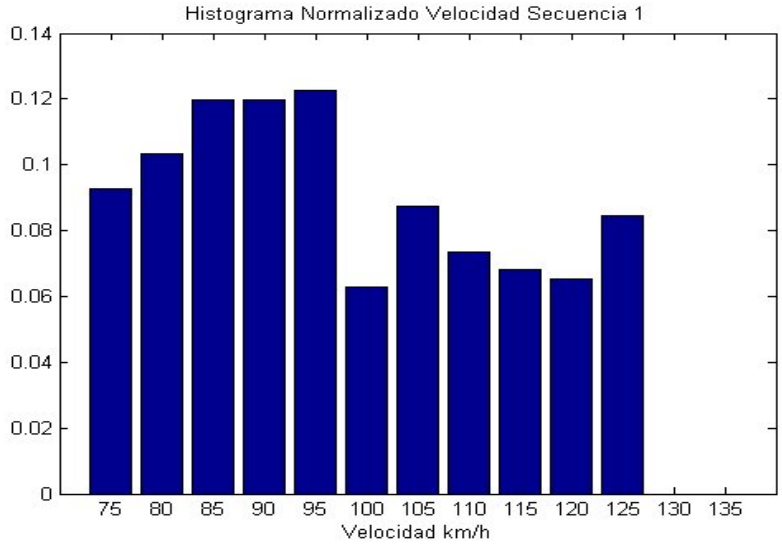


Figura 5.7: *Secuencia 1: Histograma de Velocidad*

## 5.2. Secuencia 2

### 5.2.1. Secuencias de Salida

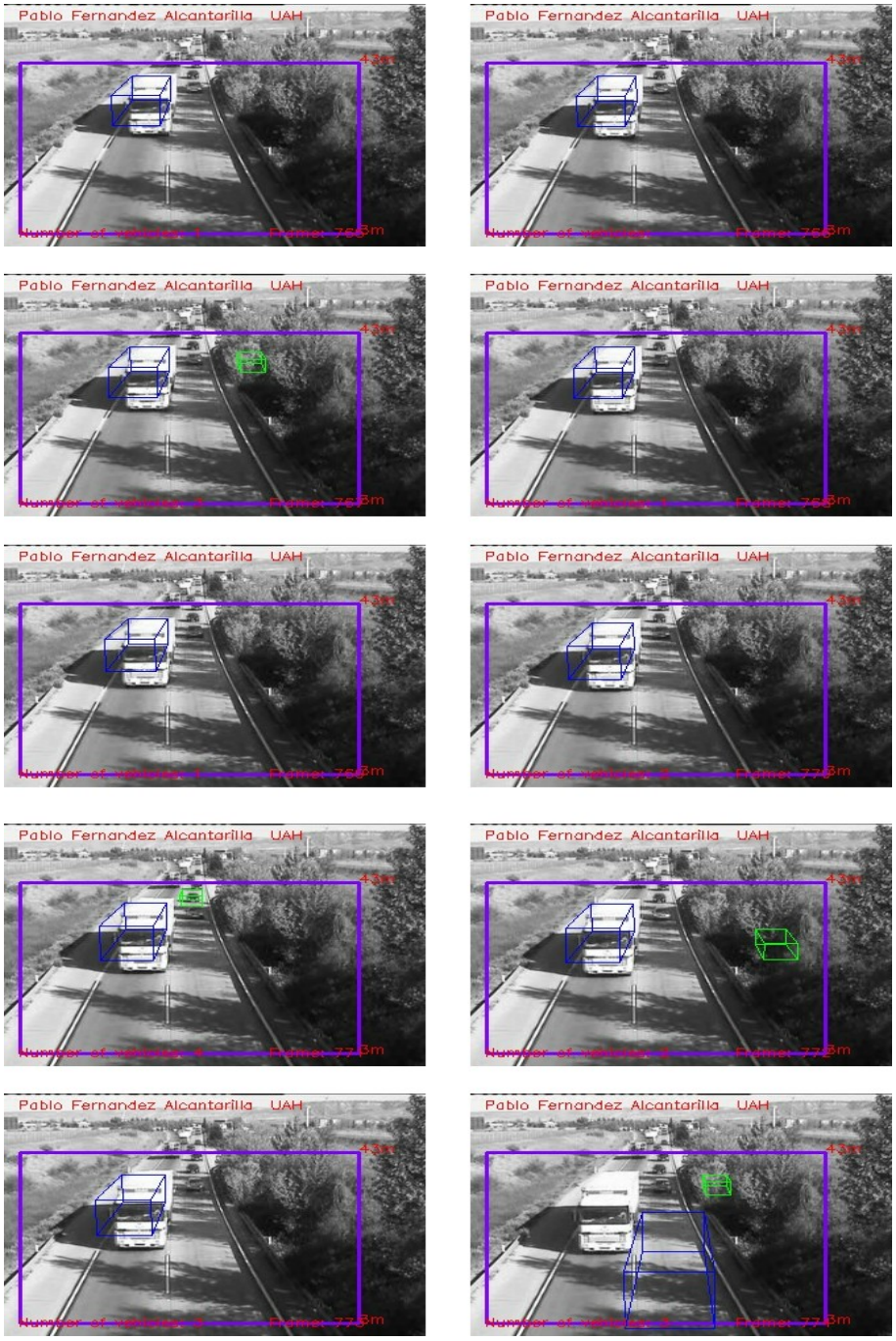


Figura 5.8: *Secuencia de Salida 2: Frames 258 - 267*

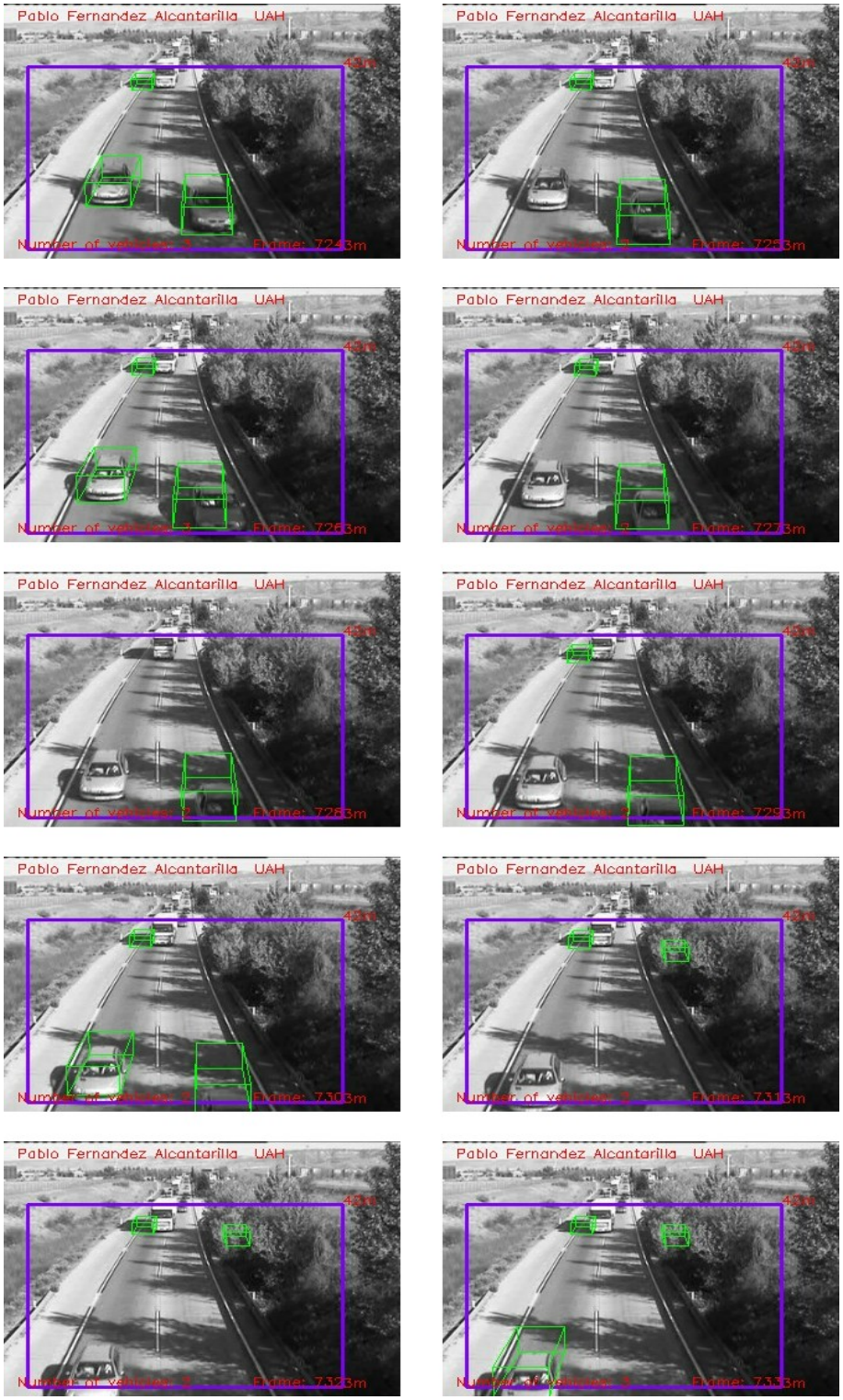


Figura 5.9: *Secuencia de Salida 2: Frames 724 - 733*

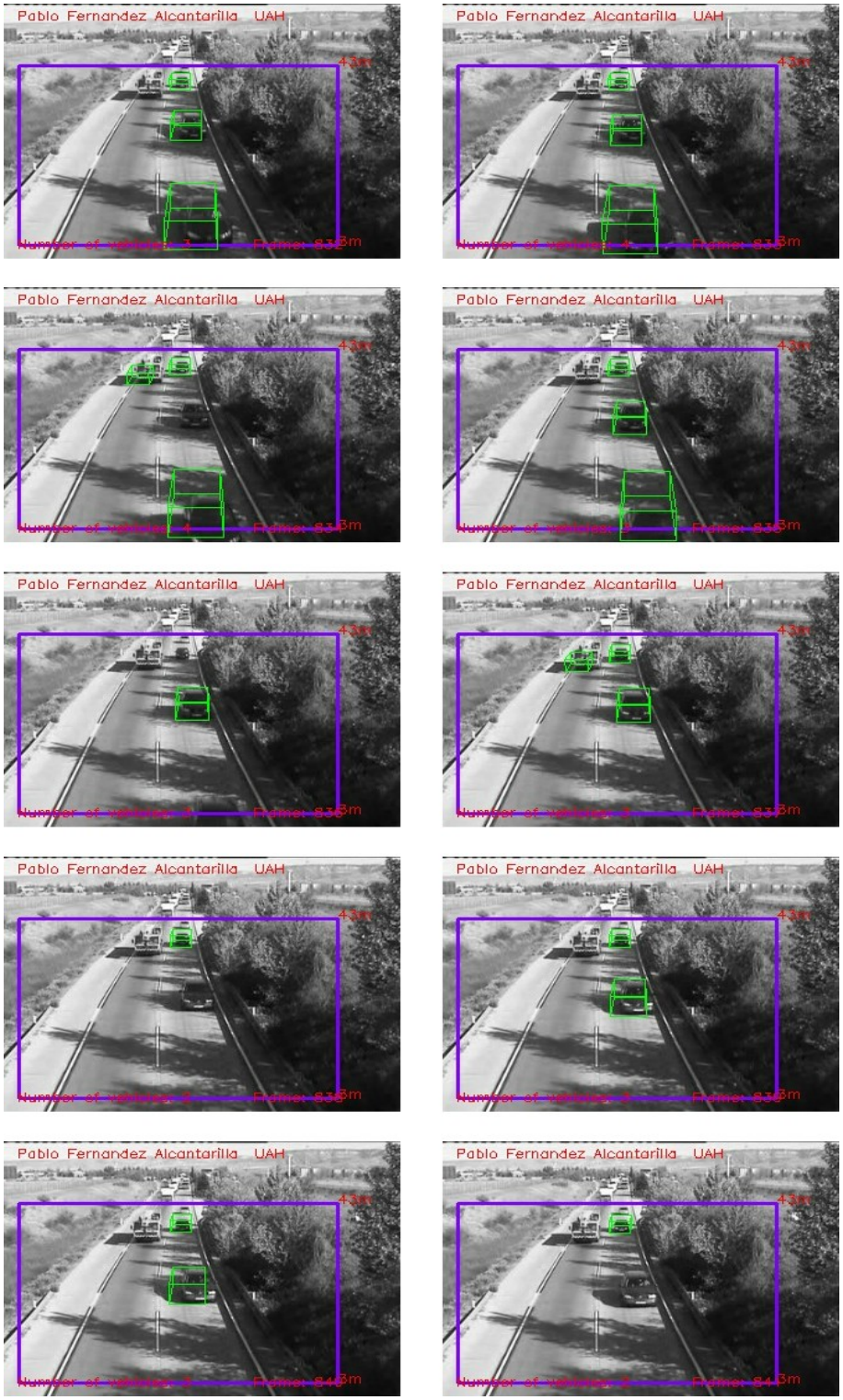


Figura 5.10: *Secuencia de Salida 2: Frames 832 - 841*

### 5.2.2. Número de Vehículos Detectados y Tiempo de Procesado

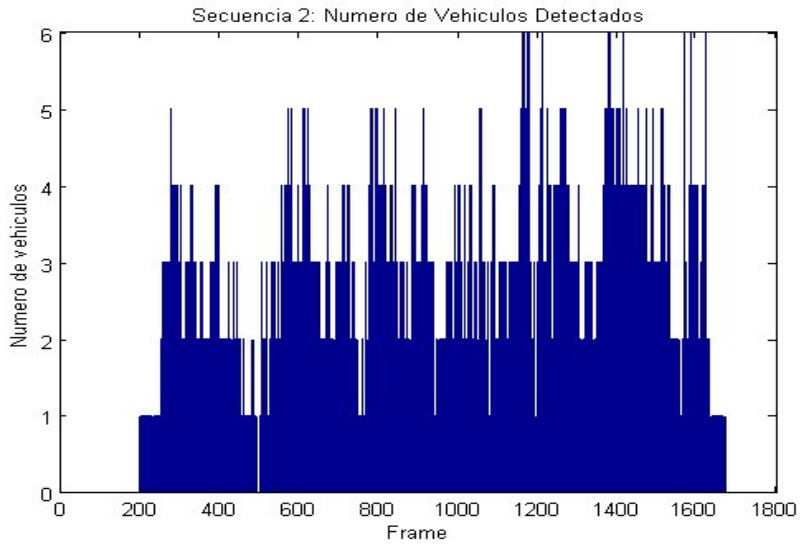


Figura 5.11: *Secuencia 2: Número de Vehículos Detectados*

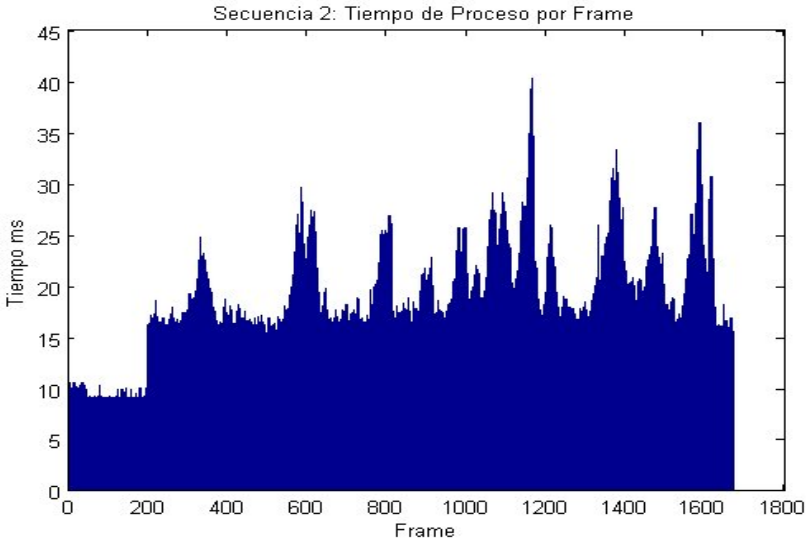


Figura 5.12: *Secuencia 2: Tiempo de Procesado*

### 5.2.3. Posición y Velocidad

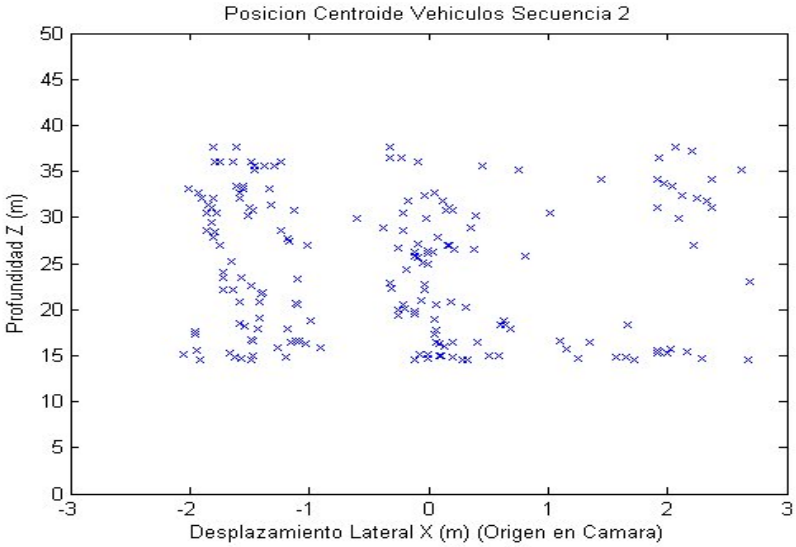


Figura 5.13: *Secuencia 2: Posición del Centroide*

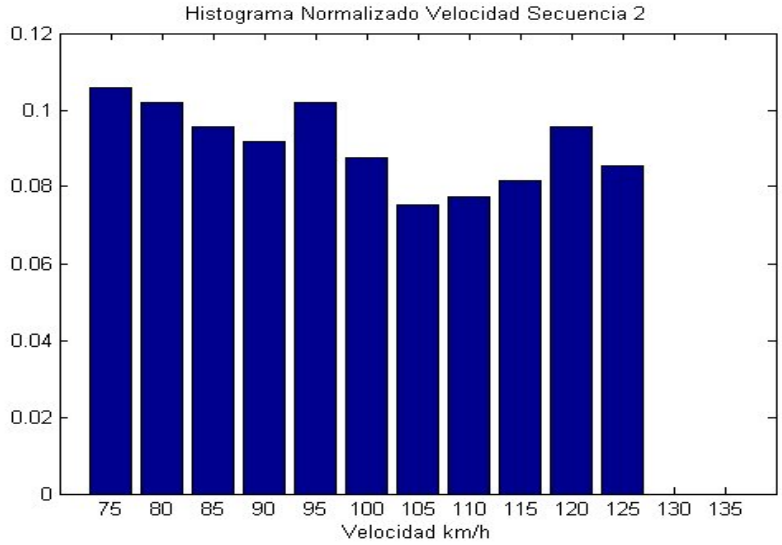


Figura 5.14: *Secuencia 2: Histograma de Velocidad*

## 5.3. Secuencia 3

### 5.3.1. Secuencias de Salida

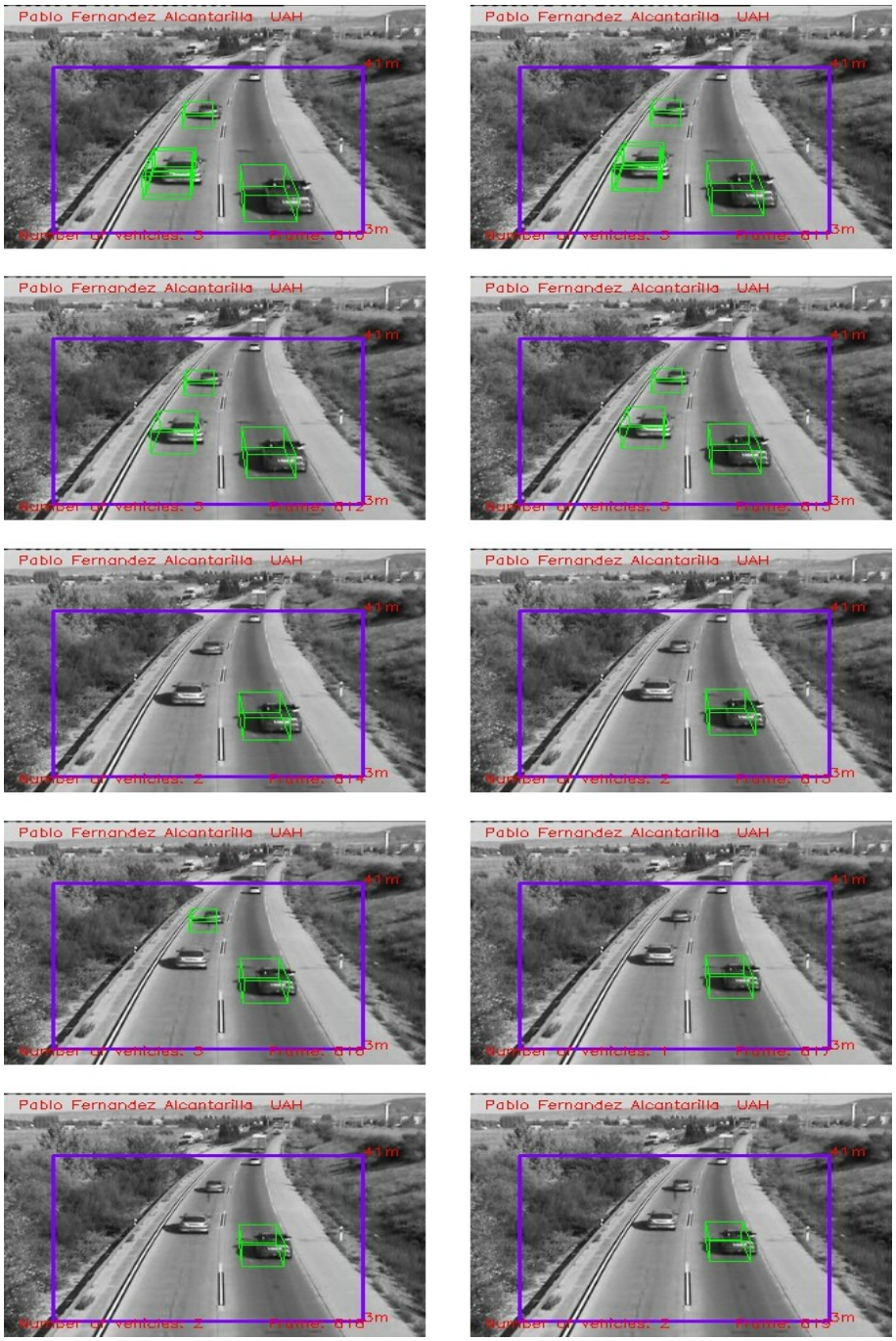


Figura 5.15: *Secuencia de Salida 3: Frames 810 - 819*

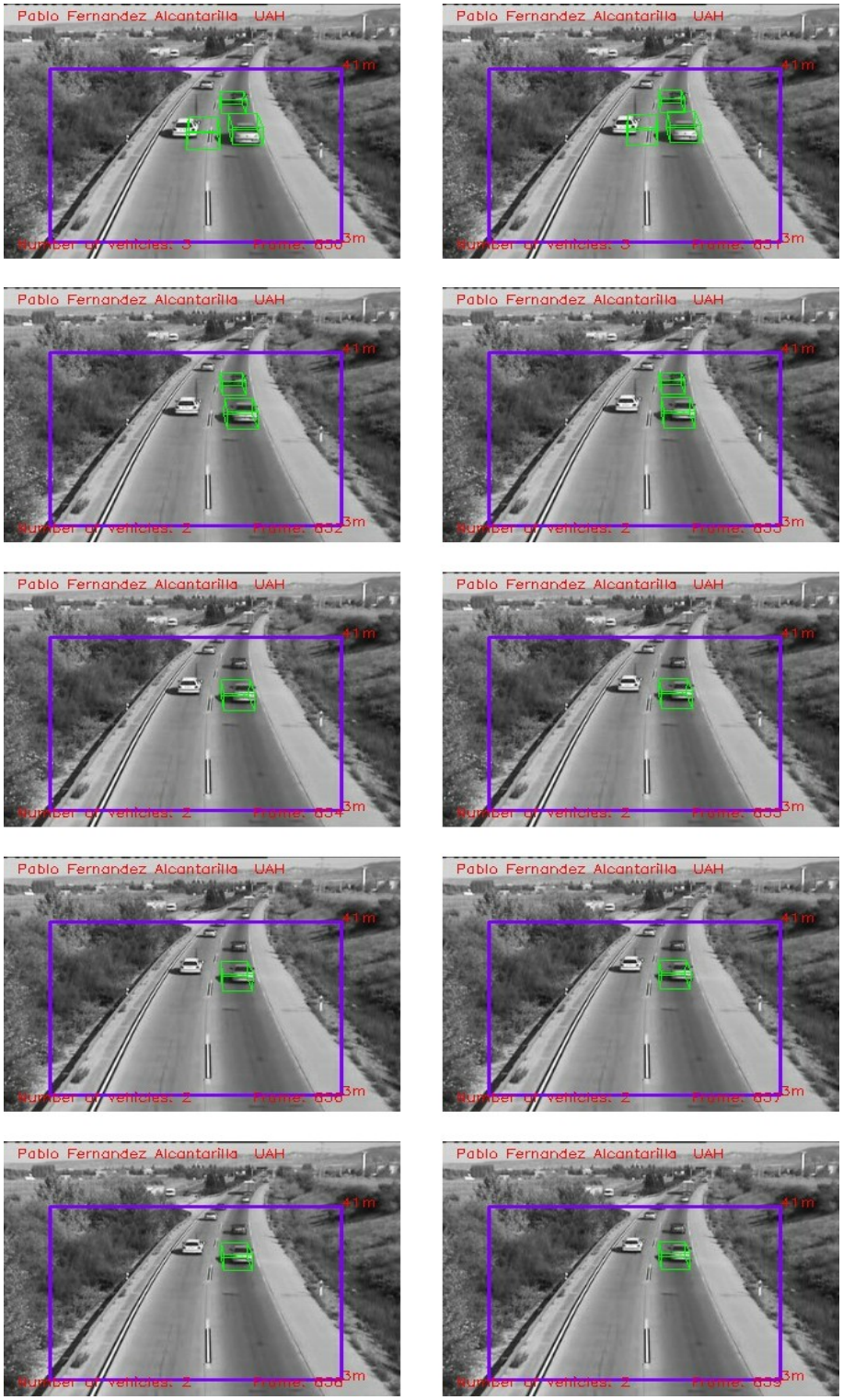


Figura 5.16: *Secuencia de Salida 3: Frames 850 - 859*

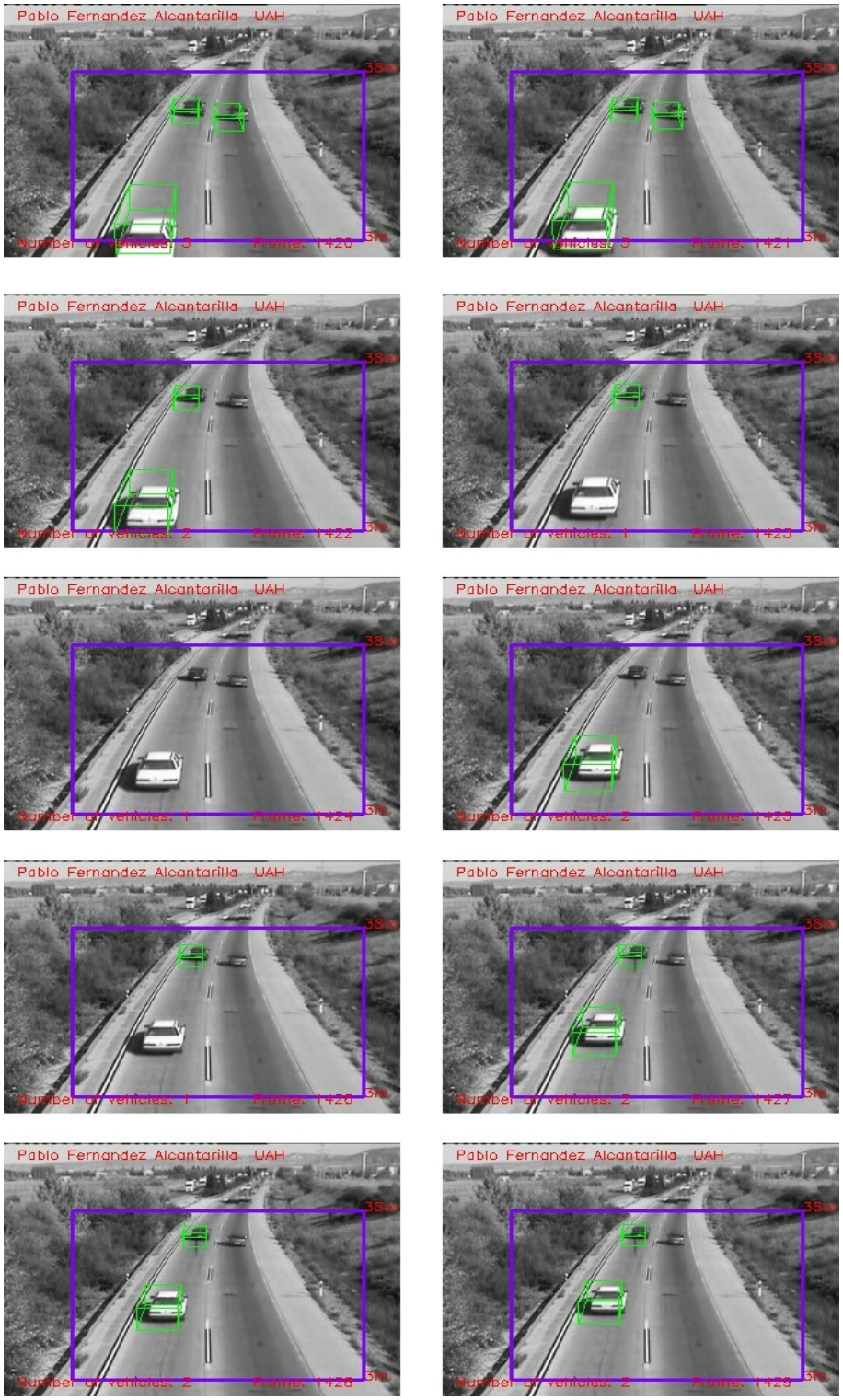


Figura 5.17: *Secuencia de Salida 3: Frames 903 - 912*

### 5.3.2. Número de Vehículos Detectados y Tiempo de Procesado

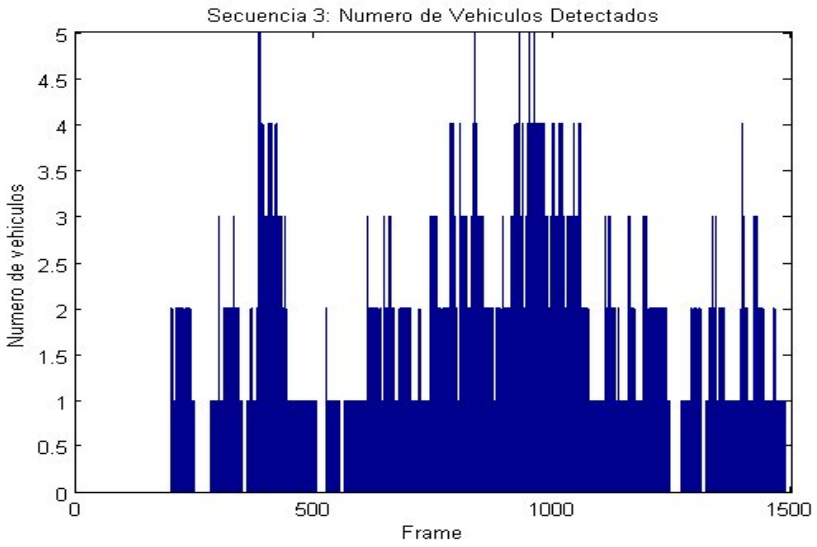


Figura 5.18: *Secuencia 3: Número de Vehículos Detectados*

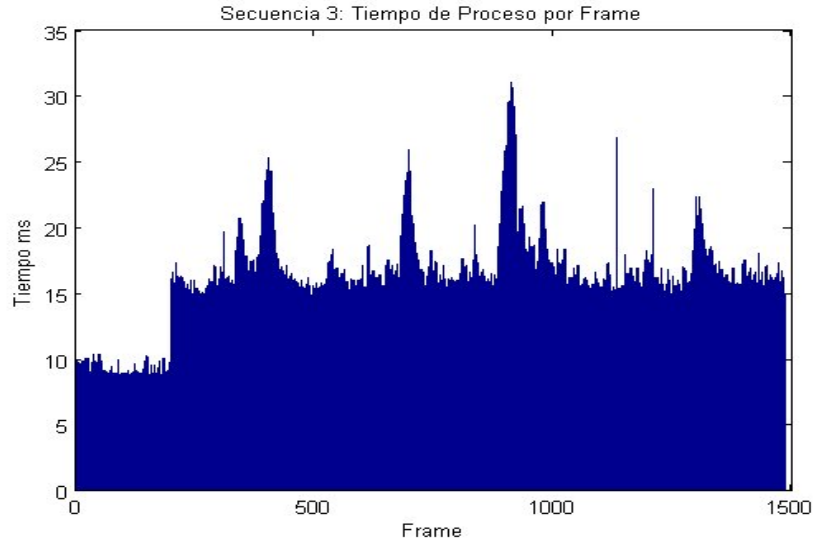


Figura 5.19: *Secuencia 3: Tiempo de Procesado*

### 5.3.3. Posición y Velocidad

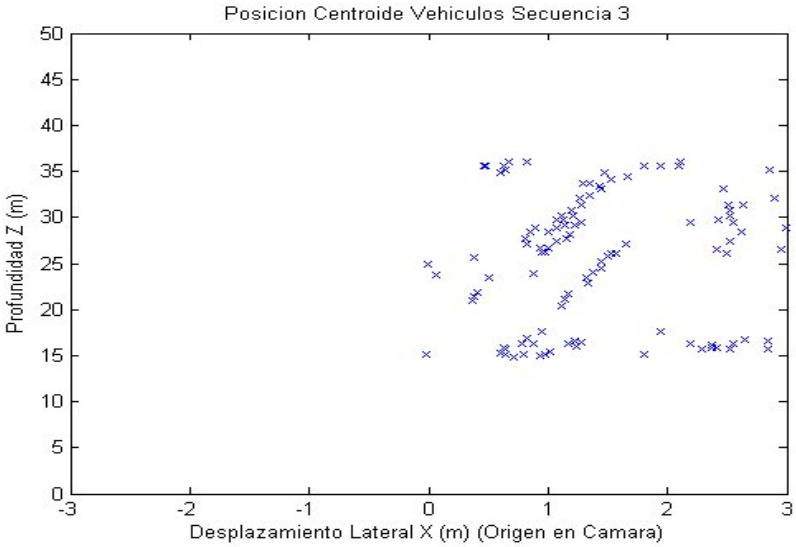


Figura 5.20: *Secuencia 3: Posición del Centroide*

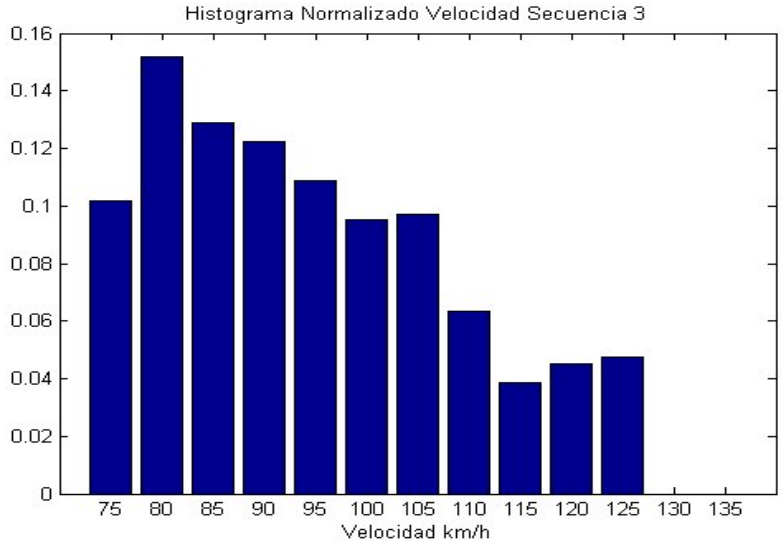


Figura 5.21: *Secuencia 3: Histograma de Velocidad*

## 5.4. Porcentajes de Detección

En este apartado se muestra una tabla con los resultados de la detección de vehículos para cada una de las secuencias. Como ya se ha comentado antes, se considera un **vehículo detectado** cuando sobre él se ha generado un modelo 3D.

Para calcular el **Porcentaje de Detección**, primeramente se computa mediante inspección visual el número de vehículos que aparecen en la secuencia, sin tener en cuenta el tiempo de inicialización del algoritmo. Una vez hecho esto, se computa mediante inspección visual el número de vehículos que han sido detectados y se calcula el porcentaje.

Un parámetro de especial interés es la **Densidad de Vehículos**. Se calcula como el valor medio de los vehículos que aparecen por cada frame de la secuencia. Este parámetro nos puede dar una idea de lo congestionada que se puede encontrar una calle, carretera o autopista.

| **Secuencia** | **Duración** | **Número Vehículos** | **Porcentaje Detección** | **Densidad Vehículos** |
|---|---|---|---|---|
| Secuencia 1 | 1:01 | 42 | 100 % | 1.4904 |
| Secuencia 2 | 1:07 | 60 | 95 % | 2.2589 |
| Secuencia 3 | 1:00 | 39 | 97.43 % | 1.5331 |

# Capítulo 6

# Conclusiones y Futuras Mejoras

La principal característica del algoritmo, es que **es capaz de detectar a un elevado porcentaje de vehículos**. Sin embargo, presenta algunos problemas a la hora de realizar el seguimiento al vehículo detectado en toda la ventana de trabajo, así como con algunos camiones demasiado grandes. Por lo tanto, se puede concluir que el algoritmo funciona mejor cuando las condiciones de la carretera son buenas y no existe demasiada congestión de tráfico. Algunas de las conclusiones a destacar son:

- Los resultados finales como ya se ha comentado dependen principalmente de tres parámetros:
    1. **Transitorio Inicial**: Durante este período el algoritmo toma datos para estimar el fondo y posteriormente poder clasificar un píxel en fondo o primer plano. Para que el algoritmo funcione correctamente, es necesario que en este transitorio inicial no aparezcan en el vídeo demasiados vehículos o se encuentren en un atasco por ejemplo. También se puede escoger un tiempo lo suficientemente grande para poder estimar correctamente el fondo.
    2. **Parámetro k**: Este parámetro se puede ir variando en el transcurso de la aplicación en función de los resultados para intentar mejorarlos.
    3. **Área de Trabajo**: Se debe escoger un área de trabajo lo suficientemente grande, para obtener buenos resultados. Tampoco se debe escoger toda la imagen, ya que la aplicación se ralentizaría considerablemente.
- Existen pequeños errores debidos a la **calibración de la cámara**, ya que el proceso de calibración se realizó hasta que se consiguió un ajuste bueno de los modelos 3D. Por lo tanto, si se calibrara correctamente la cámara se obtendría una mayor precisión a la hora de medir resultados de posición y velocidad.
- El algoritmo funciona muy bien en los últimos metros, cuando los vehículos se encuentran más cercanos a la cámara, siendo capaz de detectar y de realizar un seguimiento durante algunos frames a vehículos tan pequeños como una motocicleta. Podemos ver un ejemplo en la figura 6.1 en la que se ha detectado una motocicleta en una región con numerosas sombras:


Figura 6.1: *Conclusiones: Funcionamiento Algoritmo Últimos Metros*

- En ocasiones, debido a la existencia de viento o cualquier otra alteración metereológica, si el umbral de la Resta de Fondo es demasiado bajo, podemos obtener clusters no deseados como podemos ver en la figura 6.2. Una forma de solucionar este problema, es aumentando el umbral.

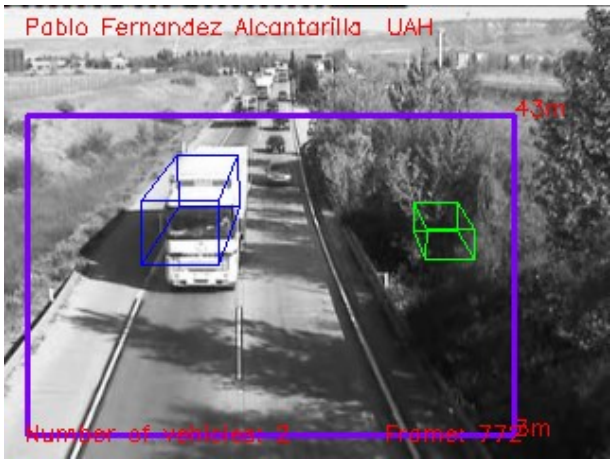


Figura 6.2: *Conclusiones: Problemas con la Vegetación*

- Se pueden tener problemas debido a que no se libere correctamente el tracking de un vehículo cuando este sale fuera de los límites de la ventana de trabajo. Cuando ocurra esto, se puede observar durante un par de frames la existencia de un modelo 3D *perdido* como podemos ver en la figura 6.3.


Figura 6.3: *Conclusiones: Problemas con los Modelos Perdidos*

- Existen problemas cuando hay que distinguir dos vehículos que se encuentran muy cerca y existen oclusiones debido a las sombras dificultando el clustering y la posterior detección y seguimiento, como se puede ver en la figura 6.4.


Figura 6.4: *Conclusiones: Vehículos Demasiado Cerca*

- El algoritmo funciona bastante bien detectando vehículos que están circulando en paralelo como podemos ver en la figura 6.5.
- Sin embargo, el algoritmo puede tener problemas cuando circulan camiones en paralelo debido a su elevado tamaño como podemos ver en la figura 6.6.

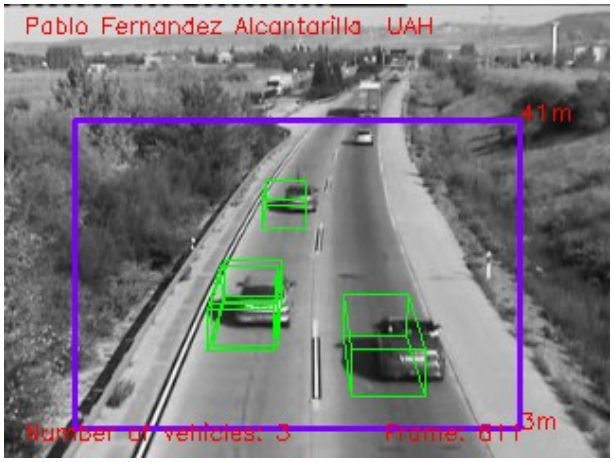


Figura 6.5: *Conclusiones: Detección de Coches en Paralelo*

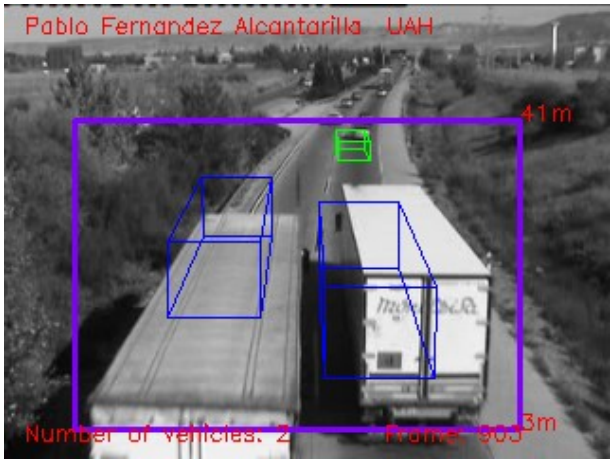


Figura 6.6: *Conclusiones: Detección de Camiones en Paralelo*

## 6.1. Futuras Mejoras

En este apartado se comentan posibles mejoras a realizar así como orientaciones que puedan servir a futuros proyectos similares o basados en este mismo proyecto. Hay que tener en cuenta que el proyecto realizado constituye una buena base para proyectos de una mayor envergadura sobre detección y control de vehículos en carretera como puede ser por ejemplo una tesis doctoral. Por lo tanto, se necesitan uno o dos años de trabajo sobre el tema para depurar los resultados.

Se pueden introducir muchas mejoras al proyecto. Una de sus principales características es la estructura del algoritmo principal compuesto por distintos niveles. En cada uno de estos niveles se pueden introducir muchas mejoras, siendo los niveles con mayor margen

de mejora la **Resta de Fondo** y el **Clustering**. Algunas mejoras que se pueden introducir son:

- **Supresión de Sombras**: La implementación de un algoritmo de detección de sombras es muy importante, ya que como hemos visto en la Resta de Fondo pasan muchos píxeles pertenecientes a sombras. Con un algoritmo de este tipo tendríamos una categoría más para poder clasificar a los píxeles. Podríamos clasificarlos según estos tres tipos:

  - Fondo.
  - Primer Plano.
  - Sombra.

  Si se consiguen eliminar correctamente las sombras el clustering resultará mucho más fácil, y no existirán oclusiones o solapamientos entre vehículos debido a las sombras.

- **Clustering con Mayor Inteligencia**: Realizar un clustering que no sólo se fije en la distancia entre los píxeles, si no que sea capaz de reconocer ciertos patrones de vehículos o esté basado en funciones de decisión bayesianas, y de este modo sea más sencillo poder clasificar los vehículos como camiones, turismos o motocicletas.

- **Calibración de la Cámara**: Si se desean obtener datos con precisión de posiciones, distancias, velocidades, etc... se ha de calibrar correctamente la cámara para poder conocer con exactitud sus parámetros intrínsecos.

- **Seguimiento y Modelado 3D con Orientación**: Esta es una mejora muy sencilla de realizar. Si incorporamos al modelado 3D parámetros sobre la orientación del vehículo, seríamos capaces de seguir al vehículo y que estuviera correctamente modelado por ejemplo en curvas, rotondas...

- **Modelado 3D**: Se puede mejorar los modelos 3D a partir de prototipos de modelos de vehículos que se adapten mejor que un cubo al coche o a partir de modelos generados en otros programas de desarrollo gráfico como *OpenGL* o *3D Studio*.

- **Mejora en la Velocidad**: Si se logra detectar la velocidad de los vehículos con una precisión elevada, esto podría de ser de gran ayuda para sistemas tipo RADAR utilizados para detectar excesos de velocidad.

- **Distancias**: Consiguiendo una buena precisión en el clustering y conociendo la calibración de la cámara, se pueden obtener parámetros importantes como por ejemplo distancias entre vehículos.

- **Integración del Sistema en Entornos Urbanos**: Un objetivo claro, es la integración de esta aplicación en entornos urbanos en donde estén presentes tanto vehículos como peatones, y el sistema sea capaz de detectar y clasificar a los objetos como vehículos o peatones.

# Parte III

# Presupuesto

# Capítulo 7

# Presupuesto del Proyecto

En este apartado podemos ver el desglose de todos los costes relacionados con el desarrollo del proyecto. Podemos dividir los costes en los referentes a especificaciones y requerimientos técnicos, y en los referentes a horas de trabajo humano en el desarrollo del proyecto.

## 7.1. Costes de Ordenador y Complementos

| DESCRIPCIÓN | VALOR | CANTIDAD | PRECIO |
|---|---|---|---|
| ORDENADOR PENTIUM 4 CPU 3 GHz 992 MB RAM | 699.00 € | 1 | 699.00 € |
| RATÓN ÓPTICO PS2 2 BOTONES + RUEDA | 12.00 € | 1 | 12.00 € |

**Subtotal:** ..... **711.00 €**

## 7.2. Costes de Sistema de Visión

| DESCRIPCIÓN | VALOR | CANTIDAD | PRECIO |
|---|---|---|---|
| TARJETA CAPTURADORA DE VÍDEO AVERTV | 49.99 € | 1 | 49.99 € |
| CÁMARA DE VÍDEO SONY XC-73 | 385.00 € | 1 | 385.00 € |
| TRÍPODE DE PIE | 24.99 € | 1 | 24.99 € |

**Subtotal:** ..... **459.98 €**

## 7.3. Costes de Software de Desarrollo del Proyecto

| DESCRIPCIÓN | VALOR | CANTIDAD | PRECIO |
|---|---|---|---|
| SISTEMA OPERATIVO LINUX: FEDORA CORE 3.0 | – | 1 | – |
| OPENCV 0.9.6 | – | 1 | – |
| FFMPEG | – | 1 | – |
| GLADE | – | 1 | – |

**Subtotal:** ..... **0 €**

El resto de software utilizado no se desglosa en el presupuesto, ya que son de libre distribución y muchos de ellos se incluyen en la distribución de Linux utilizada ***Fedora Core 3***.

## 7.4. Costes de Software de Elaboración de Documentos

| DESCRIPCIÓN | VALOR | CANTIDAD | PRECIO |
|---|---|---|---|
| MIKTEX | – | 1 | – |
| TEXNIC CENTER | – | 1 | – |
| THE GIMP | – | 1 | – |
| SMARTDRAW 7 | 155.95 € | 1 | 155.95 € |

**Subtotal:** ..... **155.95 €**

## 7.5. Costes de Horas de Diseño

| DESCRIPCIÓN | VALOR | CANTIDAD | PRECIO |
|---|---|---|---|
| HORAS DISEÑO DEL ALGORITMO | 25.00 € | 380 | 9500.00 € |
| HORAS DISEÑO INTERFAZ GRÁFICA | 25.00 € | 20 | 500.00 € |
| HORAS MECANOGRAFÍA | 15.00 € | 100 | 1500.00 € |

**Subtotal:** ..... **11500.00 €**

A la hora de realizar el cálculo de horas de desarrollo del proyecto, se ha estimado una dedicación a tiempo completo de 4 meses.

## 7.6. Costes de Honorarios por Redacción y Desarrollo

Los costes de Honorarios por Redacción y Desarrollo se estiman en un 13 % del coste total del proyecto sin considerar impuestos.

**Subtotal:** ..... **1685.62 €**

## 7.7. Coste Total del Proyecto

| CONCEPTO | PRECIO |
|---|---|
| COSTE DEL PROYECTO | 12966.33 € |
| COSTE DE HONORARIOS | 1685.62 € |

**TOTAL:** ..... **14651.95 €**

**IVA 16 %:** ..... **2344.31 €**

**TOTAL IVA INCLUIDO:** ..... **16996.26 €**

El Importe Total del proyecto suma la cantidad de:

**Dieciséis Mil Novecientos Noventa y Seis Euros, con Veintiséis Céntimos.**

Alcalá de Henares: 8, Noviembre, 2006

El Ingeniero

# Parte IV

# Pliego de Condiciones

# Pliego de Condiciones

Para que el proyecto se ejecute correctamente no sólo hay que tener en cuenta la correcta ejecución de la aplicación, si no que también el sistema debe ser capaz de ejecutar la aplicación en un tiempo determinado, de tal modo que el sistema no ralentize a la aplicación. Si se desea que la aplicación funcione en **tiempo real** el sistema debe ser lo suficientemente rápido para cumplir las restricciones temporales.

Según lo anterior, las características técnicas que debe cumplir el equipo en dónde se ejecute la aplicación, son las siguientes:

**Características Hardware**:

- La potencia de cálculo mínima necesaria es la de un equipo con procesador Intel Pentium III a 866 MHz con memoria RAM suficiente para garantizar una velocidad mínima de ejecución de 20 fps.
- Cualquier equipo compatible con una potencia de cálculo superior a la anterior también es válido.

**Características Software**:

- Sistema Operativo Linux con kernel 2.4.21 o superior.
- Entorno de ventanas X Window correctamente instalado y configurado.

Una vez comprobado que el sistema cumple los requisitos anteriores, habrá que instalar el software que se adjunta con este proyecto de la forma que se ha explicado en la memoria en el capítulo **Entorno de Desarrollo**.

Para ejecutar la aplicación se debe arrancar el entorno X Window y ejecutar el programa desde el mismo entorno o bien abriendo un intérprete de comandos desde el entorno de ventanas.

# Parte V

# Manual de Usuario

# Capítulo 8

# Manual de Usuario

En este Manual de Usuario se explican todos los pasos necesarios para que un usuario sin experiencia previa sea capaz de ejecutar la aplicación. Lo primero de todo es tener el sistema correctamente configurado como se explicó en la sección **Entorno de Desarrollo**. Una vez que el sistema ha sido configurado y tenemos la aplicación en nuestra computadora podemos pasar a ejecutarla.

## 8.1. Ejecutando la Aplicación

Para ejecutar la aplicación podemos teclear desde el directorio donde se encuentre la aplicación mediante línea de comandos **./road_traffic** o bien pulsando el icono road_traffic que podemos ver en la figura 8.1.

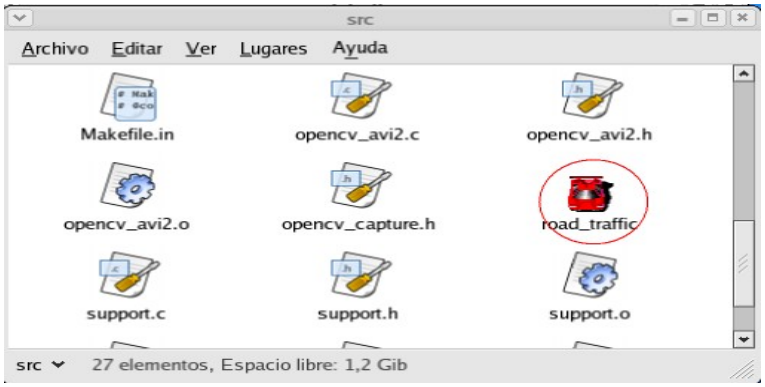


Figura 8.1: *Iniciando la Aplicación road_traffic*

## 8.2. Menú Principal Interfaz de Usuario

Tras haber iniciado la aplicación, nos aparecerá en pantalla el menú principal de nuestra interfaz de usuario:

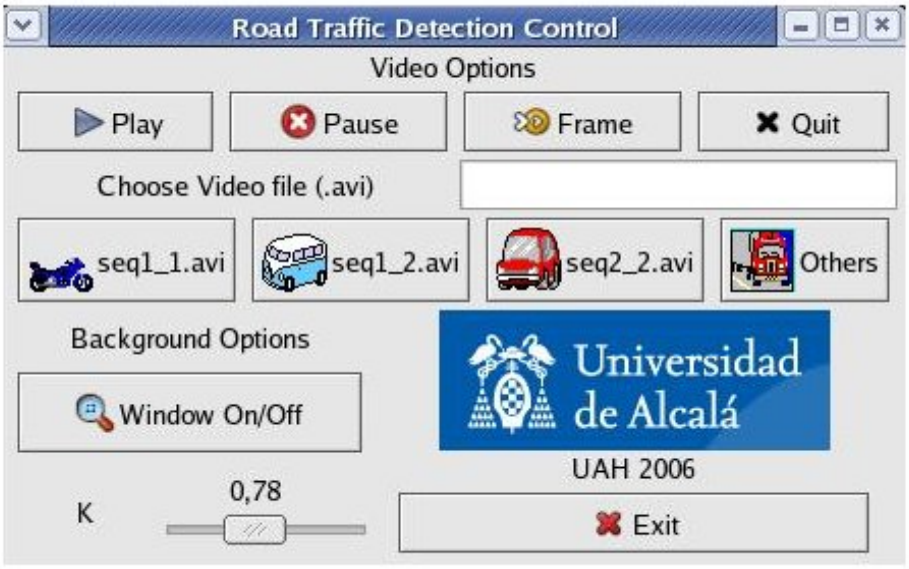


Figura 8.2: *Interfaz de Usuario: Menú 1*

Como podemos ver en la figura 8.2, tenemos dos bloques en la interfaz. Ahora pasamos a comentar para que sirve cada uno de ellos.

- **Video Options**: Son opciones básicas para el manejo de los vídeos.

  1. **Play**: Pulsando sobre este botón reanudaremos la aplicación tras haber realizado una pausa presionando sobre el botón **Pause**.
  2. **Pause**: Pulsando sobre este botón realizaremos una pausa en la aplicación. Existen otras formas de realizar pausas en la aplicación como veremos más adelante.
  3. **Frame**: Pulsando este botón entraremos en **Modo Frame**. Es decir, se analiza un frame y la aplicación se detiene. Para volver a analizar el siguiente frame pulsamos sobre el botón **Play**. Para volver a modo de reproducción continuo, hay que pulsar otra vez sobre el botón **Frame** y pulsar **Play**.
  4. **Quit**: Este botón sirve para eliminar las ventanas en dónde se muestran los resultados de la aplicación y volver al menú principal.

5. **seq1_1.avi**: Pulsando sobre este botón seleccionaremos el vídeo seq1_1.avi. Podemos ver el vídeo seleccionado en la etiqueta de entrada/salida de texto a la derecha de *Choose .avi video file*.
6. **seq1_2.avi**: Pulsando sobre este botón seleccionaremos el vídeo seq1_2.avi.
7. **seq2_2.avi**: Pulsando sobre este botón seleccionaremos el vídeo seq2_2.avi.
8. **Others**: Pulsando sobre este botón seleccionaremos otro vídeo de prueba distinto de los tres anteriores por si en un futuro se siguen realizando pruebas con esta aplicación poder utilizar la misma interfaz de usuario. Para ello tendremos que introducir previamente el nombre del archivo de vídeo que deseemos en la etiqueta de entrada/salida de texto a la derecha de *Choose .avi video file*. Si el vídeo no se encuentra, podremos ver un mensaje de error en la etiqueta de entrada/salida de texto.

- **Background Options**: Son opciones para controlar parámetros de la Resta de Fondo.

  1. **Window On/Off**: Pulsando este botón podremos quitar o hacer visible la ventana de resultados de la Resta de Fondo. Por defecto esta ventana se encuentra visible.
  2. **k**: Podemos seleccionar un valor del intervalo 0.25 - 1.5 para modificar el parámetro $k$ del umbral de la Resta de Fondo. Por defecto $k$ vale 0.78.

- **Exit**: Pulsando este botón, saldremos de la aplicación. Se recomienda pulsar previamente el botón **Quit** si no se han eliminado las ventanas de resultados. También para poder salir de la aplicación se puede pulsar sobre el icono en forma de cruz en la esquina superior derecha del menú principal.

Estos son los comandos del menú principal. Como se ha podido observar su manejo es muy fácil e intuitivo. Ahora sólo queda seleccionar alguno de los vídeos de prueba y disfrutar de la aplicación.

## 8.3. Ventanas de Resultados

Una vez que hemos seleccionado un vídeo, nos aparecen dos ventanas nuevas, una llamada **Road Traffic Control** y la otra **Background Substraction**. Podemos ver estas nuevas ventanas en la figura 8.3:

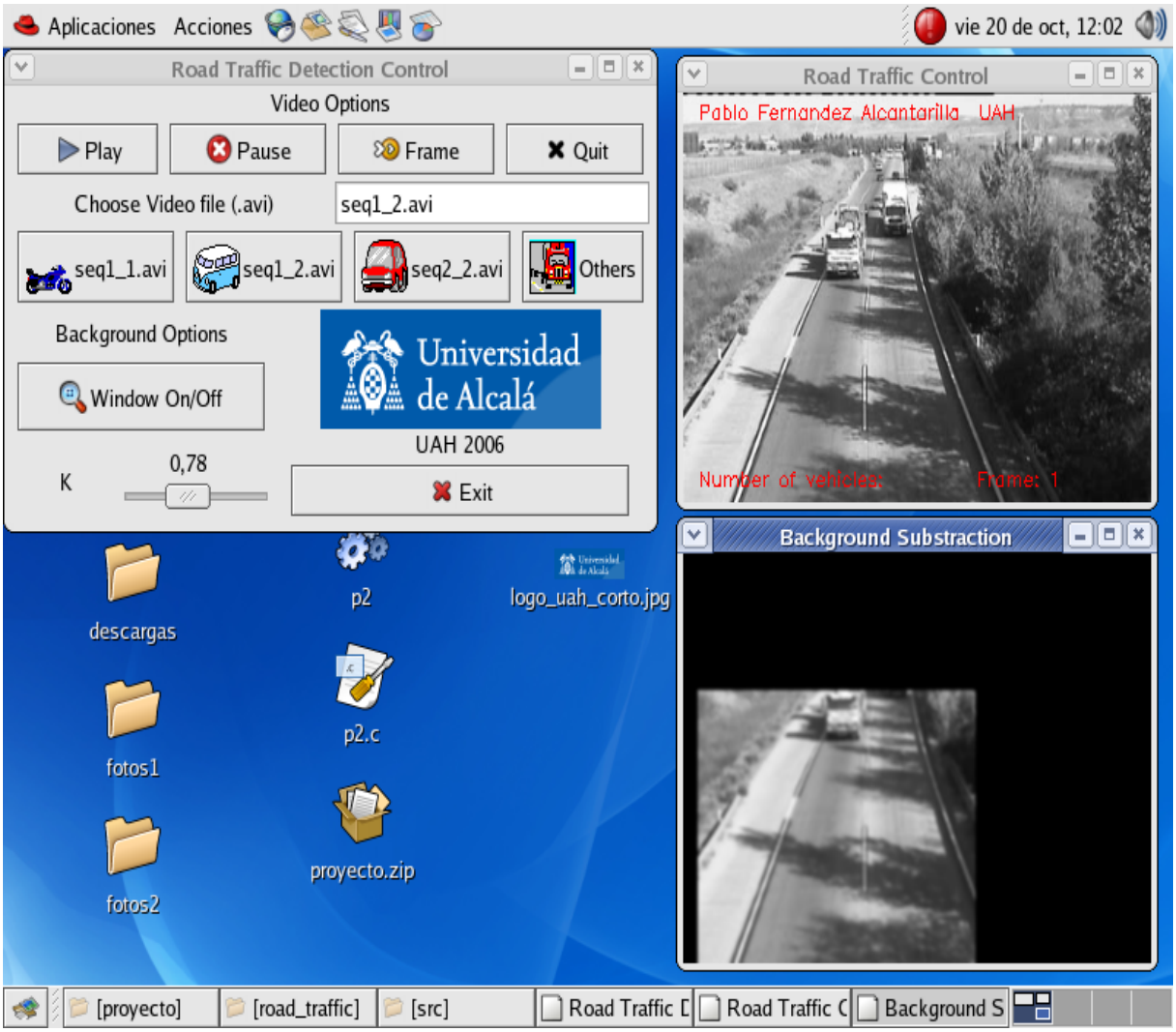


Figura 8.3: *Interfaz de Usuario: Ventanas de Resultados*

Ahora tenemos que seleccionar el **área o ventana de trabajo**. Nuestra ventana de trabajo va a ser un rectángulo de tamaño variable por lo que solamente tendremos que introducir dos esquinas del rectángulo. Para hacer esto hacemos click con el **botón izquierdo del ratón** sobre la ventana **Road Traffic Control** y nos aparecerá una cruz morada en la ventana como podemos ver en la figura 8.4:

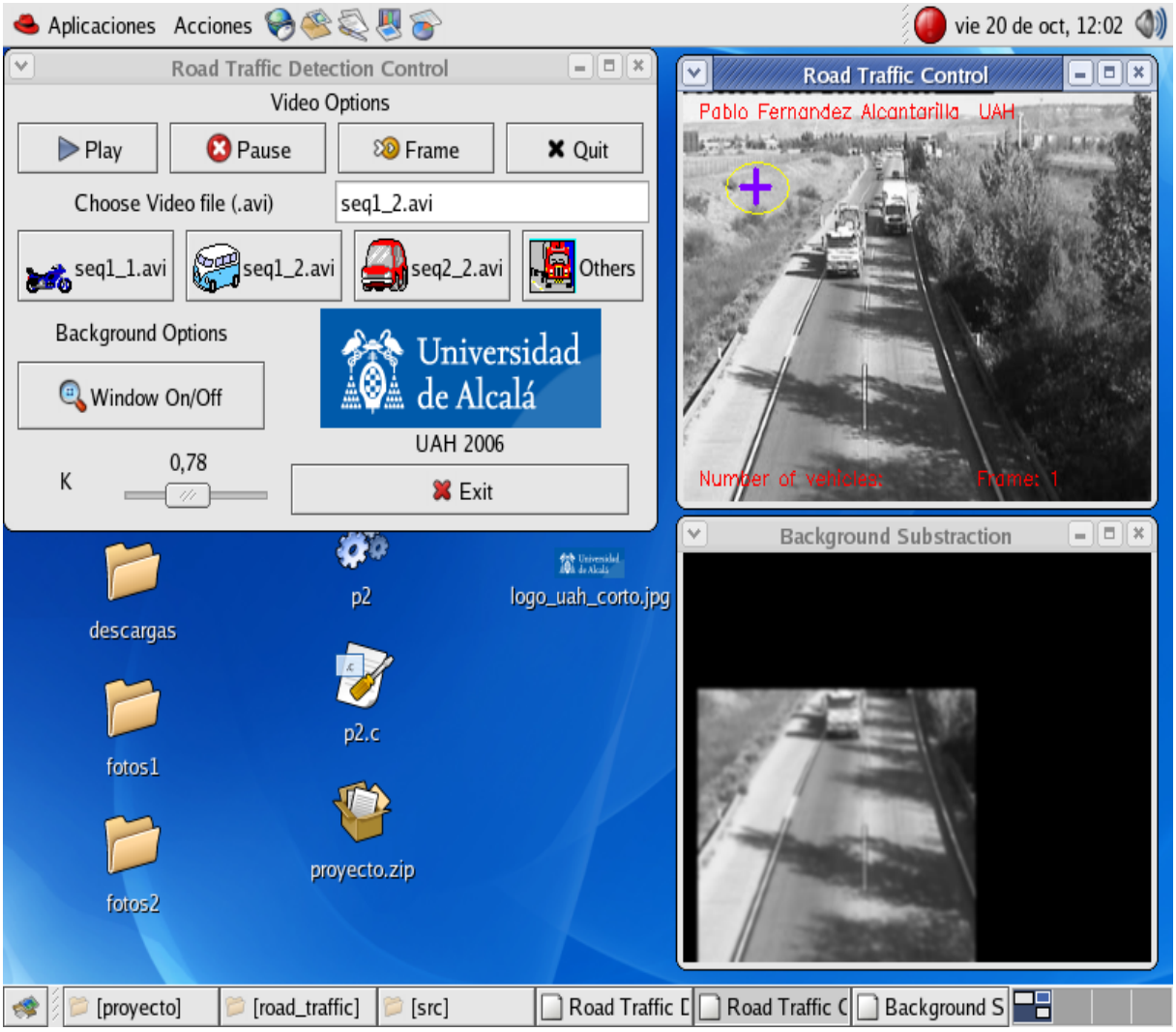


Figura 8.4: *Interfaz de Usuario: Selección de Ventana de Trabajo*

Una vez que hemos seleccionado una esquina de nuestra ventana de trabajo tenemos que seleccionar la otra esquina de nuestra ventana rectangular. Para hacer esto volvemos a repetir el paso para la esquina anterior, y esta vez hacemos click con el **botón derecho del ratón**.

Tras haber seleccionado la ventana de trabajo, la aplicación comienza a procesar el vídeo seleccionado. Después de los 4 segundos que dura el transitorio inicial, se muestran los resultados como se puede ver en la figura 8.5:

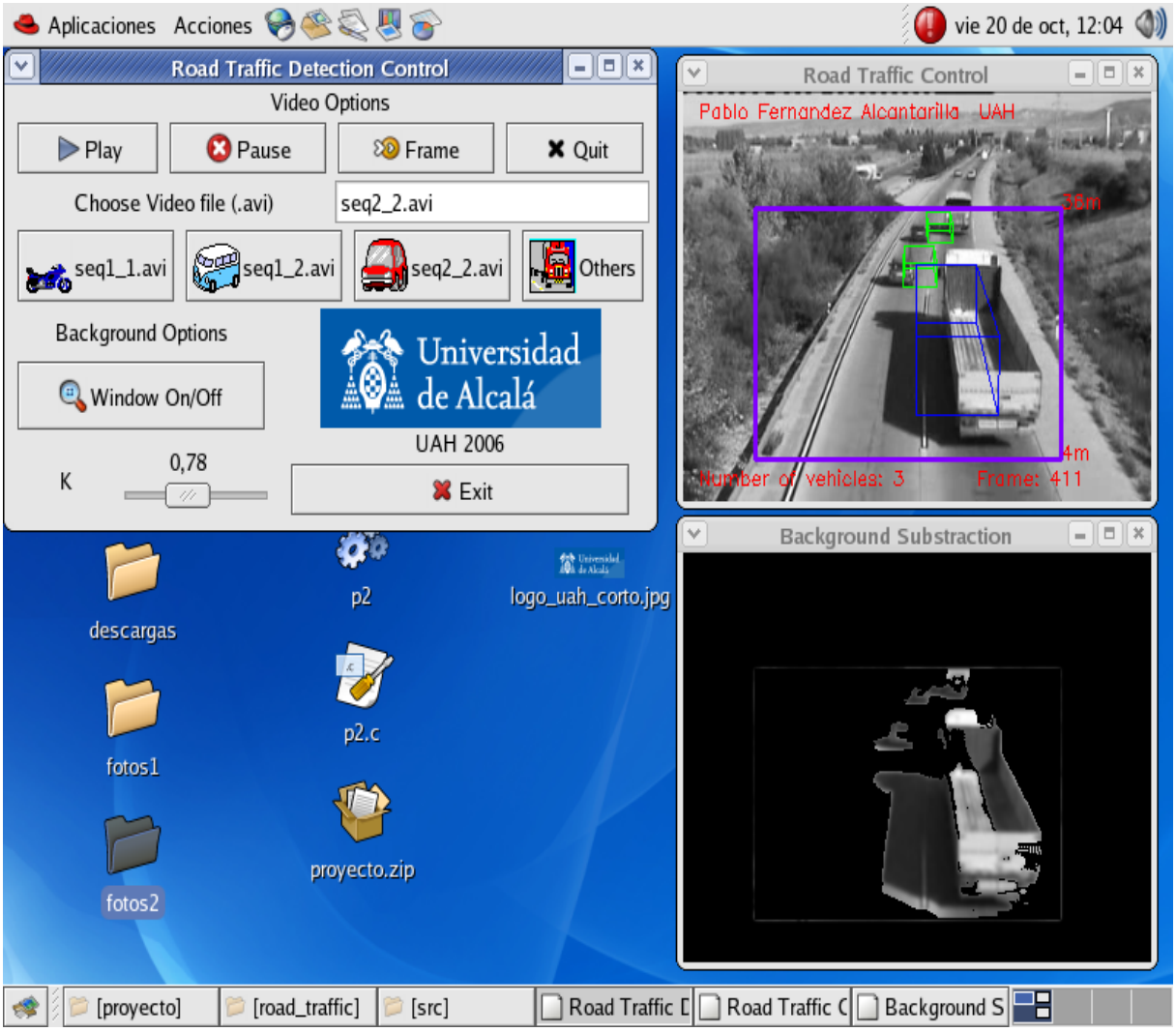


Figura 8.5: *Interfaz de Usuario: Resultados*

En la ventana **Road Traffic Control** podemos ver los resultados de la detección y el seguimiento de los vehículos. En la línea inferior de la ventana podemos ver el número de vehículos detectados en ese frame, así como el número de frame.

En la ventana **Background Substraction** podemos ver los resultados tras aplicar la **Resta de Fondo**. Esta ventana si se desea se puede deshabilitar con el botón **Window On/Off**, no obstante en ocasiones es buena tenerla visible por si queremos modificar en tiempo real el parámetro ***k*** y ver los resultados.

Cada vez que seleccionemos una nueva ventana de trabajo comenzará de nuevo el transitorio inicial para esa ventana de trabajo.

Una vez que finalice el procesado del vídeo seleccionado volveremos al menú principal donde podremos seleccionar otro vídeo o finalizar la aplicación.

### 8.3.1. Opciones de la Ventana: Road Traffic Control

Como se ha comentado anteriormente, existen otras formas diferentes de realizar una pausa en la aplicación desde el menú principal pulsando el botón **Pause**.

Podemos realizar las siguientes acciones pulsando sobre la ventana **Road Traffic Control** (siempre que se encuentre seleccionada):

1. Estando seleccionada la ventana **Road Traffic Control** podemos pulsar una tecla cualquiera para realizar una pausa. Para luego volver a continuar la aplicación podemos volver a pulsar una tecla en esta ventana o bien desde el menú principal con el botón **Play**.
2. Estando seleccionada la ventana **Road Traffic Control** podemos hacer doble click con cualquier botón del ratón para volver a seleccionar una nueva ventana de trabajo. No obstante se recomienda que para seleccionar una nueva ventana de trabajo se pulse previamente el botón **Pause** y luego se seleccione la ventana de trabajo.

# Bibliografía


[1] ***Visión por Computador: Imágenes digitales y aplicaciones.*** G. Pajares y J. M. De la Cruz. Ed. Ra-Ma.

[2] ***Open Source Computer Vision Library. Reference Manual.*** Intel Corporation.

[3] ***Image Analysis and Rule-Based Reasoning for a Traffic Monitoring System.*** Rita Cucchiara, Massimo Piccardi y Paola Mello. IEEE Transactions on Intelligent Transportation Systems. VOL I, NO 2, Junio 2000.

[4] ***Tracking All Traffic. Computer Vision for Monitoring Vehicles, Individuals and Crowds.*** Benjamin Maurin, Osama Masoud y Nikolaos P. Papanikolopoulos. IEEE Robotics and Automation Magazine, Marzo 2005.

[5] ***An Introduction to the Kalman Filter.*** Grey Welch y Gary Bishop.

[6] ***Citilog Video Detection Systems.*** *http://www.citilog.com/index_en.php*.

[7] ***Traficon Video Detection.*** http://www.traficon.com/.

[8] ***Técnicas de Gestión de Tráfico.*** *http://www.sistemasdepesaje.com/sensores-en-pavimento.html*.

[9] ***An algorithm to Estimate Mean Traffic Speed Using Uncalibrated Cameras.*** Daniel J.Dailey, F.W.Cathey y Suree Pumrin.

[10] ***Performance of Optical Flow Techniques.*** J. L. Barron, D.J. Fleet y S.S. Beauchemin.

[11] ***Proyecto Fin de Carrera: Diseño de un Sistema Señalador para Presentaciones.*** D. David Millán Escrivá y D. Manuel Agustí i Melchor.

[12] ***OpenCV Library.*** *http://opencvlibrary.sourceforge.net/*.

[13] ***CXCORE Reference Manual.*** *http://opencvlibrary.sourceforge.net/CxCore*.

[14] ***CV Reference Manual.*** *http://opencvlibrary.sourceforge.net/CvReference*.

[15] ***HighGUI Reference Manual.*** *http://opencvlibrary.sourceforge.net/HighGui*.

[16] ***Manual de Latex.*** *http://www.fceia.unr.edu.ar/lcc/cdrom/Instalaciones/LaTex/latex.html*.

[17] ***Wikipedia: La Enciclopedia Libre.*** *http://es.wikipedia.org/wiki/Portada*.

[18] ***Desarrollo de Aplicaciones Científicas con Glade.*** Francisco Domínguez-Adame. *http://valbuena.fis.ucm.es/ adame/programacion*.

[19] ***Desarrollando Aplicaciones Gnome con Glade. Tutorial básico de Glade.*** *http://eddy.writelinux.com/spanish/*.

[20] ***GTK+ Reference Manual.*** *http://developer.gnome.org/doc/API/gtk/index.html*.

[21] ***GDK Reference Manual.*** *http://developer.gnome.org/doc/API/gdk/index.html*.