GRADO EN INGENIERIA EN TECNOLOGÍA
DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Universidad del País Vasco Euskal Herriko Unibertsitatea

BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

SISTEMA DE CONTROL ÓPTICO DE VEHÍCULOS BASADO EN CASCADAS DE HAAR

Alumno: Dueñas, Blanco, Asier

Director: Espinosa, Acereda, Koldo

Curso: 2017-2018

Fecha: 23, 07, 2018

Resumen

Este trabajo describe un sistema de detección de coches en entornos del mundo real basado en visión artificial, campo que actualmente está en auge. El sistema usa las *cascadas de Haar*, método inicialmente introducido por Paul Viola y Michael Jones, que es conocido por su rápido procesamiento y buenas tasas de detección; sobre las librerías de programación *OpenCV*, las cuales son las más utilizadas en el campo de la visión artificial. El proceso requiere un conjunto de datos representativos a usar para el entrenamiento y la validación, incluyendo conjuntos positivos (presencia del objeto a detectar) y negativos (ausencia de dicho objeto) de imágenes. Por lo tanto, varias imágenes ejemplo de coches fueron etiquetadas a mano para el entrenamiento y con el propósito de realizar cálculos de rendimiento. Los resultados preliminares muestran que el método puede ser muy eficaz para detectar automóviles a altas tasas y mostrar capacidades de generalización. A pesar de algunas faltas de detección, debido a que este método es bastante rápido, puede servir para propósitos como el de este proyecto. Dicho propósito es el estudio del flujo de coches en un punto de una autopista, de tal manera que se pueda conocer el uso que se le da a la misma, así como prevenir retenciones y, por lo tanto, evitar posibles accidentes de tráfico y mejorar la circulación.

Palabras clave: Características Haar, Visión Artificial, Detección Automática de Coches, OpenCV.

Abstract

This paper describes a system designed for detection of cars in real-world environments. It is based on computer vision, a field that is currently booming. The system uses the Haar cascades method firstly introduced by Paul Viola and Michael Jones, which is known for its fast processing and good detection rates; under the OpenCV programming libraries, which are the most used in the field of computer vision. The process requires representative data sets to be used for training and validation, including positive (presence of the object to detect) and negative (absence of the object to detect) image samples. Therefore, several example images of cars were hand labelled for training and performance calculation purposes. Preliminary results show that the method can be very effective to detect cars at fast rates and show generalization capabilities. In spite of some lack of detection, because this method is quite fast, it can serve for purposes like the one of this project. This purpose is the study of the flow of cars at a point on a highway, so that it can be known the use that is given to it, as well as prevent retentions and, therefore, avoid possible traffic accidents and improve the circulation.

Key words: Haar Features, Computer Vision, Automatic Detection of Cars, OpenCV.

Laburpena

Lan honek, ikusmen-artifizialean oinarritutako eta mundu errealean kokatutako autoen detekzioa gauzatzen du. Sistemak Haar Urjauziak erabiltzen ditu, hasiera batean Paul Viola eta Michael Jones-ek ikasitako arloa, bere prozesamendu azkar eta detekzio tasa altuengatik. OpenCV liburutegi ezagunak erabiliko dira metodo honen aplikaziorako. Prozesuak datu-multzo esanguratsu baten beharra du entrenamendua eta ondorengo balidazio prozesua osatzeko, datu-multzo positibo bat eta negatibo bat behar dituelarik. Horregatik, autoen irudi batzuk eskuz etiketatuta izan ziren ondorengo entrenamendurako eta errendimendu kalkuluak egiteko finarekin. Lehenengo emaitzek adierazten duten moduan, metodo honek autoak detektatzeko tasa altuak eta orokortze kapazitateak eskaintzen ditu. Detekzio falta batzuk ditu bere prozesamendu azkarraren ondorioz, baina proiektu honen antzeko lanetarako erabilgarria da. Lan honen helburua, autobide baten puntu bateko autoen fluxuaren ikasketa da. Horrela auto-ilarak eta hauen ondorioz gertatu daitezkeen istripuak galarazteko helburuarekin.

Giltza-hitz: Haar Ezaugarriak, Ikusmen Artifiziala, Autoen Detekzio Automatikoa, OpenCV.

Índice

Lista de ilustraciones

Lista de figuras

Lista de tablas

Memoria

1. Introducción

La visión artificial es un campo tecnológico que está en auge en la última década. Desde el punto de vista práctico, el inicio de la vida de la visión artificial fue marcado por Larry Roberts, el cual, en 1961 creó un programa que podía "ver" una estructura de bloques, analizar su contenido y reproducirla desde otra perspectiva, demostrando así a los espectadores que esa información visual que había sido enviada al ordenador por una cámara, había sido procesada adecuadamente por él. [1]

Este trabajo pretende dar un uso práctico a esta tecnología, mostrando su capacidad como herramienta. Para ello, se ha realizado un prototipo de detección de objetos. Pero, antes de nada, hay que comprender qué es y de dónde viene la visión artificial.

Existe mucho miedo hacia la inteligencia artificial (IA), infundido por la ciencia ficción, en la que se muestran sistemas electrónicos inteligentes amenazando con la destrucción o incluso teniendo sentimientos. También existe mucho temor social hacia la automatización y la robotización, además de la tan importante privacidad. Todos estos miedos son lógicos, pero se debe partir de la consideración de que la tecnología se desarrolla bajo un marco ético y, por supuesto, que las personas son siempre la prioridad de los desarrolladores. Por ello, es importante recalcar que la IA se desarrolla para ayudar a la vida humana.

Para comenzar a comprender la visión artificial se necesita entender la propia naturaleza de la tecnología en cuestión. Como todos los sistemas de inteligencia artificial, su objetivo es el de recrear, en la medida de lo posible, una función humana, en este caso, la visión.

La visión es uno de los sentidos más importantes de los seres humanos. La mayoría de las tareas del cerebro son empleadas en el análisis de la información visual. Casi todas las disciplinas científicas hacen uso de la visión para llevar a cabo sus estudios, es una actividad inconsciente y aún hoy, es muy complicado saber cómo se produce con exactitud.

La fotografía, con todas sus técnicas desarrolladas a lo largo del tiempo desde su invención en 1826, se puede considerar como la primera aproximación a la recreación de este sentido humano. Sin embargo, no es hasta los años 80, junto con la revolución electrónica, cuando estas técnicas confluyen y dan cuerpo de conocimiento propio. Con la llegada de las cámaras de vídeo CCD y los microprocesadores comienza a ser factible una tecnología como la visión artificial. Esta tecnología nace con la pretensión de capturar, de manera automatizada, la información visual del entorno físico para extraer características visuales relevantes y, desde aquel momento, no se ha dejado de investigar en el campo. Actualmente, es una tecnología creciente y en la que se va innovando cada día. [2]

La visión artificial engloba múltiples aspectos de la ciencia, por lo que se considera un sistema tecnológico complejo. Las tres tecnologías más importantes que conforman la visión artificial son la visión por computadora, la visión de máquina y el procesamiento de imágenes.

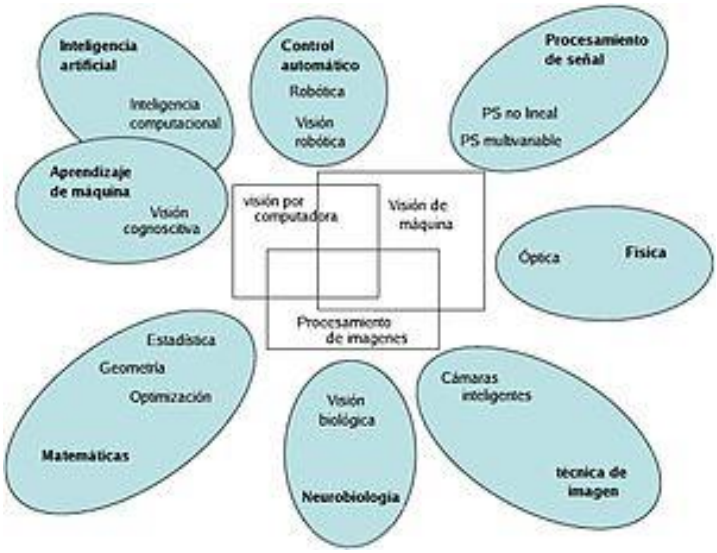

Ilustración 1: Diagrama de visión artificial. [3]

Tal y como es el mundo de la programación, hay infinitas maneras de desarrollar un sistema de visión artificial. En este proyecto se han utilizado las técnicas y librerías de programación más populares para desarrollar un sistema de reconocimiento de objetos.

En cuanto al propósito de este proyecto, se trata de la aplicación de la visión artificial a la seguridad vial. Dado que la seguridad en las carreteras es un tema de gran importancia, sobre todo teniendo en cuenta los numerosos accidentes de tráfico que hay en la actualidad. El sistema desarrollado pretende ayudar a disminuir los accidentes mediante la observación del tráfico y creación de estadísticos para el control de las carreteras.

A lo largo de este documento se encontrarán tanto las cuestiones técnicas como las prácticas, así como el código y los resultados de las pruebas realizadas con el prototipo desarrollado. Además, se explicará la organización y la metodología seguida en la realización del proyecto mediante diagramas temporales.

El sistema desarrollado ha sido testado y el lector podrá comprobar los resultados prácticos del mismo. También se incluirán en este documento los entresijos técnicos, sobre todo de programación, a tener en cuenta a la hora de realizar un sistema de este tipo. Por último, se incluirá un manual de usuario y de instalación para que el lector pueda utilizarlo, comprenderlo y realizar un prototipo similar. Se explicará la utilización de las librerías utilizadas, las cuales son muy útiles para programar un sistema de visión artificial.

2. Contexto

2.1 Dispositivos de visión artificial

Los orígenes de la visión artificial están encuadrados en el desarrollo de la robótica, y ambas tecnologías van de la mano. Durante la segunda mitad del siglo veinte, creció la investigación y desarrollo de nuevos campos relacionados con la dotación de sentidos humanos a robots.

A finales de los años veinte, Gustav Tauschek patentó en Alemania bajo el nombre de "Reading Machine" el primer dispositivo capaz de reconocer caracteres. Tanto a él como a Paul Handel se les concedió la patente de OCR (Optical Character Recognition) en Estados Unidos.

El primer sistema de este tipo en comercializarse fue instalado en Readers Digest en 1955, posteriormente donado al museo Smithsonian para su exhibición. El segundo sistema comercial se utilizó con propósitos de facturación, para leer impresiones en tarjetas de crédito. Éste fue vendido a la empresa californiana Standard Oil Company.

En 1969, Charles Rosen dirigió la creación del robot "Shake" dentro de un proyecto creado por la DARPA (Defense Advanced Research Projects). Éste fue el primer robot capaz de obtener imágenes de su entorno y tomar decisiones en base a la información que recababa con su cámara. Hoy en día está expuesto en un museo como el gran avance en esta área, teniendo en cuenta la precariedad de hardware que existía por aquel entonces.

En otros países como México, se han realizado investigaciones más enfocadas a la robótica. Algunos de estos proyectos guardan importantes similitudes con el desarrollado en este trabajo. Por ejemplo, en el Instituto Politécnico Nacional en 2010 se desarrolló un sistema de identificación de placas vehiculares mediante técnicas de visión por computadora. El lenguaje de programación utilizado entonces fue el lenguaje C, utilizando la librería *MemAccess*. En la Universidad de Colima en 2003 se desarrolló un sistema de tecnología computacional para la identificación de vehículos automóviles. Esta vez se utilizó *LabView* como herramienta y entorno para desarrollar la programación, además de una aplicación gráfica llamada *IMAQ* para el muestreo de imágenes y las mediciones. [4]

Tras varias décadas de desarrollo, las investigaciones actuales sobre visión artificial guardan una gran cercanía con el procesamiento de imágenes. Se enfocan, principalmente, al desarrollo matemático que permite trabajar de diferentes maneras con el reconocimiento de objetos.

Para analizar los sistemas de visión artificial actuales, se pueden dividir en tres categorías: sensores de visión, cámaras inteligentes y sistemas de visión artificial. [5]

- Sistemas de visión avanzados

Este tipo de sistemas está dirigido a aplicaciones muy complejas, como los sistemas de visión que se integran en la maquinaria de fabricación. Depende de los requerimientos del sistema, tales como la potencia de cálculo o el número de elementos a analizar en cada imagen, estos dispositivos pueden estar integrados por cámaras de alta resolución, matriciales o lineales, o incluir varias cámaras en un mismo sistema.

El software que utilizan estos sistemas se basan en librerías de programación de alto nivel. Su precio depende de la complejidad de la aplicación.

- Cámaras inteligentes y sistemas de visión

Estos sistemas son un paso adelante respecto a los sensores de visión. Son fáciles de utilizar, incorporan mayor potencia de cálculo, mayor resolución y mayor rango de aplicación. Las cámaras inteligentes incluyen, en muy poco espacio, un sensor de captura, la memoria de almacenamiento, el procesador y los mecanismos de entrada y salida. Sin embargo, normalmente requieren de una placa de entrada-salida adicional para posibilitar su conexión al resto de sistemas de automatización.

Los sistemas de visión son algo diferentes a las cámaras inteligentes, ya que su sensor y la memoria de la cámara se sitúan en un cabezal remoto y muy pequeño, a parte del elemento de proceso, que contiene todo lo demás. Una de las ventajas que proporciona este sistema es la de reducir el coste en aplicaciones donde se requiera varias tomas del mismo objetivo, ya que se pueden conectar varios cabezales de visión remotos en un mismo sistema de proceso.

- Cámaras de visión artificial

Las cámaras de visión artificial industriales más utilizadas en los sistemas de visión artificial son Gigabit Ethernet, USB2, USB3, Cameralink, Cameralink HS, Coaxpress y Firewire.

En los sistemas de visión artificial, la función de las cámaras de visión es capturar la imagen proyectada en el sensor, vía las ópticas, para poder transferirla a un sistema electrónico. Las cámaras que se utilizan en visión artificial requieren una serie de características específicas, como el control del disparo de la cámara, que permite capturar el objetivo que pasa por delante de la cámara en la posición exacta requerida.

Aunque el uso de estos sistemas es más bien industrial, esta tecnología de reconocimiento de objetos se puede usar en muchos ámbitos.

2.2 Implementaciones de la visión artificial

La visión artificial puede servir para solucionar problemas en diversos aspectos y áreas. Por ejemplo, puede ser utilizada para ayudar a la medicina, detectando anomalías médicas. También puede ayudar a la criminología, recabando datos mediante el reconocimiento de huellas dactilares o firmas en documentos. Su aplicación más general está en la industria, donde se utiliza para clasificación, control de calidad, conteo de unidades, etc.

Este trabajo se centrará en las aplicaciones que pueda tener la visión artificial dentro del ámbito del transporte, ya que es el uso que se le dará al sistema desarrollado.

Los dispositivos que utilizan la visión artificial en las carreteras son las cámaras de tráfico. Se pueden encontrar cámaras de tráfico tanto en las autopistas como en las carreteras dentro de

las ciudades, y el rango de uso de éstas es muy amplio. El sistema desarrollado en este proyecto hará uso de dichas cámaras para ayudar a controlar el tráfico.

Las cámaras que graban un punto de una carretera o autopista se suelen utilizar para el control del tráfico, de tal manera que se pueda prevenir una situación de congestión y reducir la siniestralidad en las carreteras. Estas cámaras posibilitan crear estadísticos para identificar las zonas con más afluencia de vehículos y evitar grandes aglomeraciones. Gracias a esta información, se pueden enviar mensajes de aviso de tráfico, o bien utilizarla para regular el tráfico con limitaciones de velocidad.

También existen cámaras inteligentes en los pasos de nivel, con las cuales se puede detectar de manera inmediata si hay un vehículo obstruyendo la vía, por ejemplo. Ante esta circunstancia, se activa una alarma, que se gestiona desde un puesto de mando, se realiza una verificación automática y se activan los protocolos de seguridad. Una empresa emergente de Bilbao, Begirale Controlling Risk, ha instalado un sistema pionero de visión con inteligencia artificial en seis pasos a nivel de la red ferroviaria vasca. Esta empresa utiliza tecnologías de visión artificial e inteligencia artificial para procesar en tiempo real el vídeo proveniente de una cámara que está enfocando al paso a nivel. [6]

Un ejemplo del uso de las cámaras de tráfico dentro de la ciudad es el empleado por la ciudad de Barcelona. Concretamente, se trata un sistema de visión artificial para el control de accesos a Ramblas (con 8 puntos de control con cámaras de lectura de matrículas), el sistema de fotorrojo (con 17 instalaciones en cruces semafóricos y 37 pre-instalaciones preparadas) y el centro de control para la gestión, visualización y denuncias. Además, se incluyen también el sistema de control de gálibo en la calle Magdalenas y el sistema de radar por tramos en la Ronda de Dalt. [7]

La más novedosa aplicación de esta tecnología en el sistema de carreteras español es la de la detección del uso del cinturón de seguridad. El funcionamiento de estas aplicaciones es la más pura descripción de cómo trabaja un sistema de visión artificial: las cámaras realizan una fotografía de cada coche en circulación, esa fotografía es analizada mediante un software capaz de identificar la presencia - o no - del cinturón de seguridad, si el software determina que el conductor no lleva cinturón de seguridad o que existe duda automáticamente pasa esa fotografía a un sistema donde será revisada por un agente para determinar si se procede con la sanción. Este tipo de cámaras también pueden ser capaces de detectar cuándo se usa el teléfono móvil al volante, al igual que pueden emplear los datos del vehículo fotografiado para descubrir si tiene el seguro o la ITV en vigor. [8]

2.3 Necesidad de ayuda en la seguridad vial

En estos momentos, la Dirección General de Tráfico (DGT) controla 13.000 kilómetros de carreteras, del total de 165.000 que hay en España, mediante cámaras y sensores. Las estadísticas muestran que el número de muertos en la carretera se ha reducido considerablemente medida tras medida de seguridad. Según el anuario estadístico de la Dirección General de Tráfico (DGT), en 2015 fallecieron 71 personas en 2.200 accidentes con víctimas. Echando la vista atrás casi cuatro décadas, hasta 1976, se encuentra que el número de accidentes es bastante cercano (2.318), pero hubo muchos más muertos, hasta 220. [9]

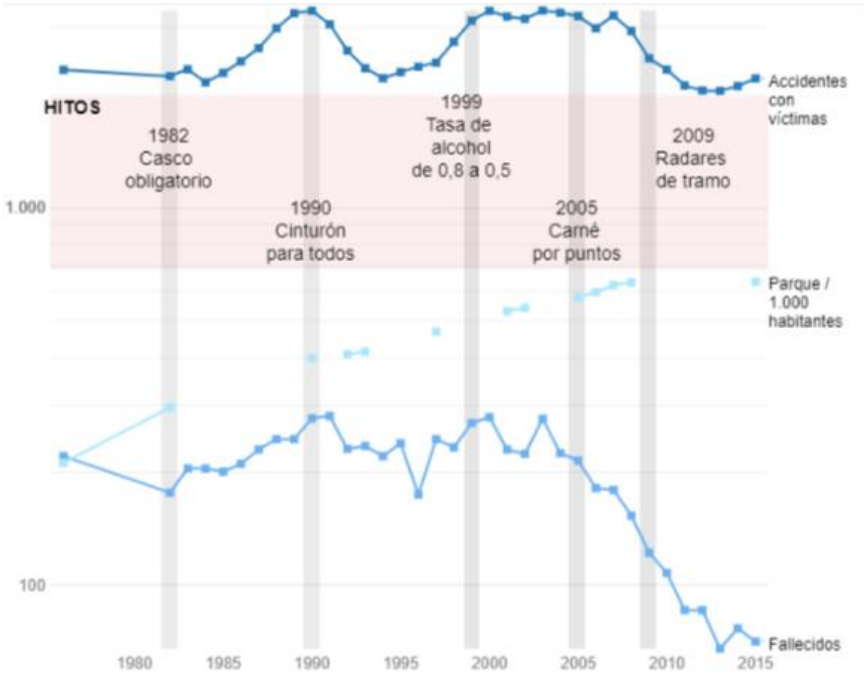

Ilustración 2: Evolución de los accidentes de tráfico y las muertes con las medidas de seguridad. [9]

También hay que tener en cuenta que el número de vehículos circulantes hoy en día es muchísimo mayor.

Las primeras medidas fueron las de la obligatoriedad de llevar casco cuando se circula en moto y la de la extensión del cinturón de seguridad para todos los pasajeros del vehículo (antes sólo el conductor debía llevarlo). Más tarde se redujo la tasa de alcohol, pero sin duda la medida más efectiva fue la del carnet por puntos.

A partir de 2009 se empezó a hacer uso de la tecnología, con la implantación de radares y el uso de cámaras de tráfico.

Aun y todo, las tasas de accidentes de tráfico muestran que se siguen necesitando nuevas formas de regular el tráfico. La tecnología de la visión artificial es de gran ayuda en este ámbito.

2.4 El futuro de la visión artificial en las carreteras

Actualmente hay varios proyectos en marcha que posibilitarán dar un paso hacia delante en la implantación de los sistemas de visión artificial en la seguridad vial.

Uno de ellos es la cámara de visión 360º junto con un sistema de visión artificial, que proporciona una gestión integral de todo lo que ocurre en una intersección, detectando los vehículos y peatones que se encuentran en cada vía, y pudiendo alertar en tiempo real al gestor de la carretera cuando se detecta cualquier problema. Además, el sistema generará informes de todo lo ocurrido, lo grabará y registrará todo. Se trata de una única cámara, fácil de instalar y de un coste muy económico, pero que ofrece un gran servicio gracias al software de visión y gestión que integra. [10]

Este sistema forma parte del proyecto "Ciudad2020" de la empresa Indra, en colaboración con muchas empresas importantes. El proyecto está desarrollado en base a lo que se ha dado en conocer como el "Internet de las cosas", que consiste en dotar a los elementos que participan en la relación ciudadano-ciudad de la tecnología que permita conectar entre sí estos objetos e intercambiar información (señales, vehículos, semáforos, teléfonos móviles, y demás dispositivos) para ofrecer las mejores soluciones a los requisitos de comunicación permanente entre los ciudadanos con los diferentes elementos que configuran la ciudad del futuro. [11]

También se encuentra en desarrollo el proyecto DGT 3.0, denominado coloquialmente como "el Gran Hermano" de las carreteras españolas, en referencia al programa televisivo. La DGT aspira a convertirse en el receptor y proveedor de datos del tráfico, con el objetivo de reducir a cero la cifra de accidentes de tráfico. Así, la misión que tendrá esta plataforma es la de mantener conectados en tiempo real a los distintos usuarios de la vía, ofreciéndoles en todo momento información de tráfico en tiempo real. Para ello, hará uso de numerosas cámaras de visión artificial, y de una red de comunicación para el intercambio de información vial. [12]

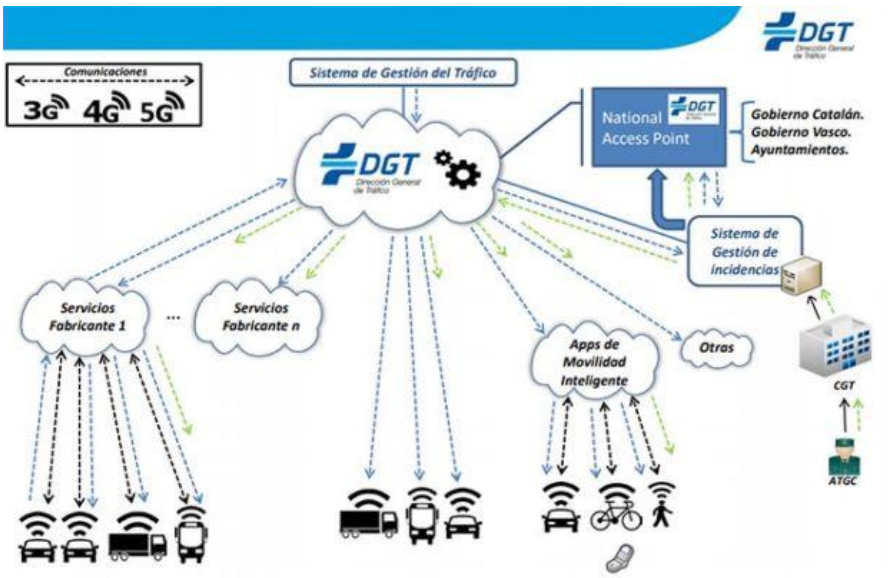

Ilustración 3: Sistema de gestión de tráfico DGT 3.0. [12]

Se trabajará con terceros para la gestión y el almacenamiento de los datos en la nube, por lo que la DGT perderá el monopolio del control de la información. Aun así, se confía mucho en este proyecto.

3. Objetivos y alcance del trabajo

El principal objetivo de este proyecto es crear un sistema de detección de vehículos en movimiento. Las características de este sistema se establecen de tal manera que pueda ser de utilidad para el control del tráfico en las carreteras y autopistas.

Además, la tecnología utilizada en la creación de este sistema se basa en las *cascadas de Haar*. El sistema trabajará bajo los algoritmos de esta tecnología, partiendo de un vídeo de entrada, y dando como resultado la misma entrada, pero con encuadres que identifican los vehículos. También proporcionará estadísticos tales como el número de vehículos circulando en cada "frame" del vídeo, o la media de vehículos circulantes en un tramo de la carretera.

Otro de los objetivos de este trabajo es mostrar el procedimiento seguido para la realización de un sistema de este tipo. De esta manera, puede servir de modelo para desarrollar sistemas más complejos, o puede ser más fácilmente implementado en otros sistemas.

El sistema está diseñado para realizar dos funciones importantes:

- Para conseguir la detección de los vehículos se han utilizado librerías de programación *OpenCV* para el desarrollo del clasificador que posibilita la detección del objeto en cuestión. Se expondrá más adelante el método seguido en la utilización de dichas librerías con el objetivo previamente expuesto.
- Para utilizar el clasificador en el vídeo de entrada, además de la visualización del resultado y de los estadísticos, se utilizará un código de elaboración propia, programado en lenguaje C. Se añadirá este código en un anexo con el objetivo de que el lector comprenda el funcionamiento del sistema.

En este proyecto se abordarán estos dos procedimientos, intentando dar solución a los problemas de tráfico a partir de la tecnología de visión artificial.

El alcance de este trabajo puede ser más extenso de lo que a priori parece. Atendiendo a la primera función expuesta previamente, la creación de un clasificador mediante las librerías de *OpenCV*, se puede aplicar este procedimiento prácticamente a todo el rango de aplicación de la visión artificial. Haciendo uso del anexo explicativo y tomando como referencia el método seguido en este proyecto, se puede desarrollar un clasificador para cualquier objeto que se desee detectar. Partiendo de ésto, este proyecto puede ser de utilidad en infinidad de situaciones en las que se necesite detectar artificialmente la presencia de un objeto.

En cuanto a seguridad vial, que es el principal objetivo y aplicación del sistema desarrollado, se puede aplicar en cualquier carretera, autopista o autovía que disponga de una cámara. A partir de las imágenes grabadas por dichas cámaras, se obtendrán los resultados anteriormente expuestos.

También se puede aplicar individualmente a vehículos. Este es el caso de los emergentes vehículos inteligentes. Éstos están equipados con una cámara que lleva integrado un software similar al desarrollado en este proyecto. Por lo tanto, podría servir para la inclusión en vehículos no inteligentes de una cámara que use el sistema desarrollado en este trabajo.

Ilustración 4: Identificación de otro vehículo por parte de uno inteligente. [13]

En rasgos generales, los objetivos de este proyecto son los siguientes:

- Estudiar las herramientas necesarias para generar clasificadores.
- Analizar la estructura de datos correcta para el uso de los clasificadores.
- Diseñar la aplicación que use el clasificador generado y aporte la información necesaria.
- Utilizar esa información visual y estadística para ayudar al control de las carreteras.

4. Beneficios que aporta el trabajo

Partiendo del contexto de este proyecto y de sus posibles implementaciones, se pueden explicar los beneficios que presentaría a nivel técnico, económico y social. Se debe tener en cuenta también el alcance descrito en el apartado anterior.

4.1 Técnicos

A nivel técnico, este sistema presenta dos propiedades características: el proceso de creación de un clasificador que permita la detección de objetos y la generación de resultados mediante la aplicación que permite utilizar dicho clasificador.

En cuanto a la creación del clasificador, este trabajo presenta un método aplicable a otros proyectos, o bien modificable. Se ha trabajado con las librerías de programación *OpenCV*, con su función *traincascade*, la cual trabaja con un conjunto de imágenes de muestra y da como resultado el clasificador. Dicha función tiene varios parámetros que pueden ser modificados dependiendo de las necesidades del programador (velocidad de procesamiento, calidad y precisión del clasificador resultante …). Como se verá en el apartado Estado del arte, existen diferentes algoritmos de aprendizaje para la detección de objetos. Para este proyecto se ha utilizado el algoritmo basado en las *cascadas de Haar* desarrollado por Paul Viola y Michael Jones, ya que ha demostrado ser el más efectivo en relación eficacia – rapidez de procesamiento (se explicará en el apartado Estado del arte). Teniendo en cuenta la limitación de los recursos de los que se ha dispuesto en la realización de este proyecto (un ordenador portátil personal, ver más detalles en el apartado Descripción de Requerimientos Hardware), se puede demostrar que no es necesario disponer de grandes equipos o infraestructuras para desarrollar un sistema como este. Esto supone un gran beneficio a nivel técnico.

Además, gracias a los desarrolladores y programadores del software libre *OpenCV*, se tiene a disposición de todo el mundo el código y los programas como el utilizado para este proyecto. Por lo tanto, este proyecto puede servir de referencia para desarrollar un clasificador en cualquier contexto, sin la necesidad de un hardware excesivamente potente.

Por otra parte, se necesitará desarrollar un código que permita el uso del clasificador. En este proyecto, como ya se ha dicho anteriormente, se ha escrito un código en lenguaje C que utiliza el clasificador para obtener una detección visual (encuadre de los vehículos en el vídeo) y estadística (conteo del número de vehículos). A diferencia de la mayoría de la mayoría de sistemas de este tipo, que se quedan en la identificación visual, esto supone un plus en cuanto a beneficios técnicos, ya que amplía la funcionalidad del dispositivo.

Otra de las ventajas técnicas con las que cuenta este sistema es la posibilidad de trabajar con cualquier vídeo. Es decir, no es necesario desarrollar un sistema para cada cámara de tráfico, sino que los vídeos de cualquier cámara son cargados por el sistema y se trabaja con ellos. Además, gracias a la funcionalidad de las librerías de *OpenCV* utilizadas, el formato de los vídeos puede ser de un amplio rango. Esto dota al sistema de una polivalencia técnica muy útil.

4.2 Económicos

En cuanto a los aspectos económicos, el desarrollo del sistema es completamente libre de costos, al igual que la implementación del mismo (únicamente habría que tener en cuenta el sueldo del programador). Tan solo se debe tener en cuenta el equipo necesario para la realización del programa. Éste tampoco es necesario que sea de una gran potencia de procesamiento, pero sí que debe cumplir unos mínimos (ver más detalles en el apartado Descripción de Requerimientos Hardware).

La mayor parte del presupuesto estaría destinada a la infraestructura de cámaras en las carreteras y al espacio de almacenamiento de los vídeos. Este último puede ser local (en discos duros, pero esto es inviable ya que se trabaja con grandes volúmenes de datos) o en servidores en internet. Para ello sería necesario comprar los discos duros o alquilar los servidores. Además, es necesaria una gestión de la base de datos para la organización de todos los vídeos.

También habría que tener en cuenta el coste de posibles averías de la infraestructura.

En general, es un sistema económicamente muy rentable si se dispone de la infraestructura necesaria para su implementación. Ofrece una funcionalidad muy beneficiosa a un coste muy bajo. Con una inversión inicial no muy generosa se haría realidad la implementación de este sistema y su uso inmediato. Además, mirando al futuro, la infraestructura puede ser utilizada para actualizaciones del sistema desarrollado, por lo que el coste a futuro es nulo.

En caso de no disponer de ningún recurso de los necesarios, sería bastante caro construir todo el sistema (ver el apartado de Análisis de Costes), pero este sistema está pensado para implementarse en la red de las cámaras de tráfico ya existente.

4.3 Sociales

El aspecto social es el más destacable en cuanto a los beneficios que aporta este trabajo. Es evidente la necesidad de seguridad en las carreteras, como se ha expuesto con anterioridad en este documento. A lo largo de estos últimos años se han implantado muchas medidas de seguridad en el tráfico por carretera, pero aun así sigue habiendo muchos accidentes. Este proyecto pretende servir como ayuda para prevenirlos.

La mayoría de las carreteras, sobre todo las autopistas, cuentan con un sistema de cámaras en varios tramos de su recorrido. Dichos tramos suelen ser los más concurridos o los denominados “tramos de concentración de accidentes”, es decir, los tramos donde se han registrado varios accidentes y que se consideran peligrosos. Por ejemplo, un tipo de estos tramos son los túneles. Las cámaras de tráfico han sido utilizadas sobre todo en la vigilancia de los túneles, donde brindan un servicio insuperable debido al grado de riesgo que representan las vías cubiertas y de difícil acceso en caso de accidentes. [14]

Mediante el uso de este sistema utilizando dichas cámaras, es posible controlar la circulación en dichos tramos de las carreteras, identificar congestiones de tráfico accidentes y de esta manera poder avisar de las retenciones que ocasionarán dichos fenómenos. Esto ayudará mucho a reducir los muchos accidentes de tráfico que tiñen de rojo las carreteras.

La implantación de las cámaras de tráfico se realizó principalmente para detectar infractores, cosa que disgustó e incluso enfadó mucho a los ciudadanos. No obstante, la sanción sigue siendo el elemento más persuasivo para los conductores.

En cuanto a las cámaras de tráfico dentro de la ciudad, estas son muy útiles tanto para la vigilancia de la circulación de los vehículos como para la seguridad de los peatones. Una de las cosas más importantes que ha traído esta nueva tecnología es que, por una parte, los peatones se sienten mucho más seguros en las calles, y por otra, los conductores de la ciudad han comenzado a interiorizar que la reducción de la velocidad en el municipio está consiguiendo salvar un importante número de vidas. [15]

A través de cámaras y un sistema informático de Big Data se podría monitorizar y estudiar qué zonas de la ciudad son las más congestionadas en cada momento, la dirección y el sentido de los atascos, un posible accidente, etcétera.

De esta manera, se podría conseguir una gestión del tráfico de una forma totalmente autónoma sin la necesidad de intervención de un humano. Configurando los diferentes semáforos que pudieran existir o advirtiendo a los usuarios de la vía mediante carteles luminosos o mensajes a los smartphones de los peatones si la señal GPS indica que se encuentran cerca de la zona colapsada. Todo ello con el objetivo de evitar el “embudo” que suponen las retenciones en las ciudades en determinadas franjas horarias del día o por una incidencia en el tráfico. [16]

Esta faceta también la satisfacen las cámaras instaladas individualmente en vehículos.

5. Descripción de requerimientos

Para desarrollar un sistema de visión artificial es necesario contar con una serie de requerimientos. Como ya se ha dicho anteriormente en este documento, el desarrollo de un sistema de este tipo se puede desglosar en dos etapas: la creación de un clasificador y la creación de una aplicación para el uso de este.

En este apartado se expondrán los requerimientos necesarios para realizar estas etapas y, por consiguiente, crear un sistema de visión artificial. Estos requerimientos se pueden dividir en hardware y software. Se analizarán ambas para el caso de un sistema genérico de detección de objetos y para el caso particular del prototipo desarrollado en este trabajo.

Además, se verán otras herramientas, técnicas y recursos utilizados en el desarrollo de sistemas de este tipo.

Por último, se analizarán los componentes del sistema a nivel funcional.

5.1 Requerimientos de los sistemas genéricos de detección de objetos

5.1.1 Requerimientos hardware

En cuanto a la creación del clasificador que posibilita la detección del objeto de estudio, es necesario un hardware computacional relativamente potente, ya que se trabajará con volúmenes de datos grandes y el coste de ejecución de la aplicación será alto. Se trabajará con un conjunto de imágenes muy grande para "entrenar" el clasificador. Este "entrenamiento" es computacionalmente intensivo. Para llevarlo a cabo, será necesario un procesador apto para ello (con un procesador *Intel* i5 o cualquiera de 1.7 GHz por lo menos será suficiente), una buena memoria RAM (4 GB como mínimo), un disco duro con capacidad suficiente para guardar cierta información obtenida por el sistema, además del conjunto de imágenes de muestra utilizadas en el entrenamiento (se requerirán alrededor de 2 GB de memoria) y, por último, una tarjeta gráfica de al menos 1 GB.

5.1.2 Requerimientos software

El software necesario para realizar un proyecto de este tipo puede variar según las preferencias del desarrollador.

Empecemos con el sistema operativo del ordenador utilizado. Éste va a depender de cómo programemos las aplicaciones necesarias para crear el sistema. Se pueden usar todos los sistemas existentes. Para cada uno, se hará uso de las librerías que puedan utilizarse en cada sistema.

Si no se va a hacer uso de ninguna librería, es decir, se va a programar todo el sistema desde cero, entonces se valorarán los programas utilizados para realizar la programación. Por ejemplo, *MATLAB* o *Gedit* son los más utilizados. *MATLAB* es comúnmente utilizado bajo el

sistema operativo Windows, mientras que *Gedit* es el editor de textos de los sistemas *Linux*, basados en *Unix*.

En cuanto a la gestión de los datos utilizados en el sistema, ésta se podrá mantener utilizando memorias físicas o servidores on-line. En el primero de los casos, estaríamos hablando de un requerimiento hardware como una memoria externa, un disco duro interno o externo, etc. En el segundo de los casos, sería necesario utilizar una base de datos en la nube. Para ello, el software más utilizado es Oracle, Microsoft SQL Server, Sybase, etc.

5.2 Requerimientos particulares del sistema de detección de vehículos desarrollado

5.2.1 Requerimientos hardware

Particularmente, para el sistema de detección de vehículos desarrollado en este proyecto se ha utilizado un ordenador portátil *Asus X555LDB*, con un procesador *Intel* i7 de 2.4 GHz, una tarjeta gráfica *NVIDIA GEFORCE 820M* de 2 GB y una memoria muy superior a los 1.7 GB requeridos en el desarrollo.

5.2.2 Requerimientos software

En cuanto al software utilizado, se ha trabajado con el sistema operativo *Ubuntu* 16.04.1, distribución de *Linux*. Se ha programado en lenguaje C++ haciendo uso del ejecutor de comandos del sistema y de la librería *OpenCV* para la primera etapa del proyecto, y utilizando el editor de textos *Gedit* para la segunda etapa.

5.3 Otros requerimientos

Tanto para los sistemas genéricos como particularmente para el desarrollado en este trabajo, hay una serie de requerimientos extra que no han sido clasificados anteriormente entre el software o el hardware.

Para realizar cualquier detección de objetos es necesario disponer de un conjunto de imágenes para realizar el entrenamiento del clasificador. Este conjunto debe ser bastante grande, compuesto por imágenes que contengan al objeto en cuestión y otras que no lo contengan.

También es recomendable disponer de copias de seguridad de los vídeos utilizados como entrada al sistema y de los resultados, para que en caso de ruptura o fallo del sistema no se perdiera la información.

Además, se necesita una conexión a internet para la gestión descentralizada de la información con la que trabaja el sistema, además de para conseguir las imágenes de entrenamiento y para los posibles usos del sistema. Es algo imprescindible.

5.4 Componentes del sistema a nivel funcional

En cuanto a la funcionalidad del dispositivo desarrollado, la disposición de sus componentes sigue el siguiente esquema:

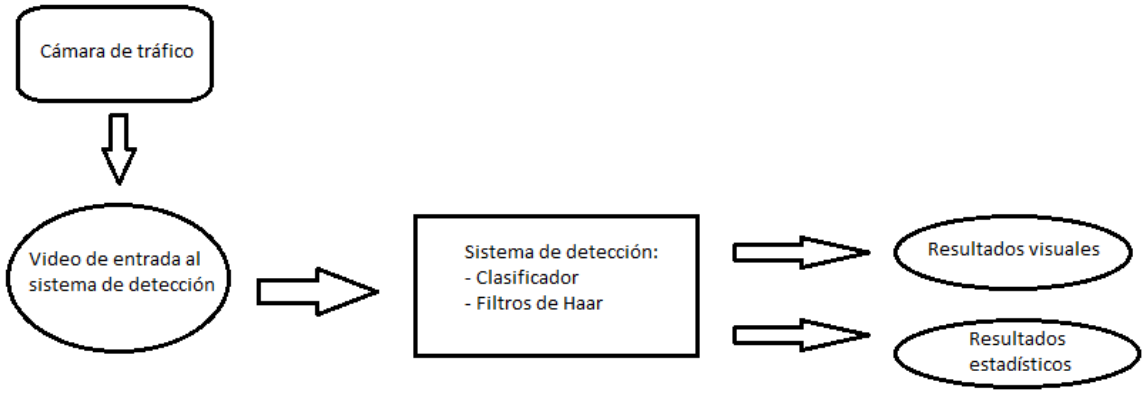

Ilustración 5: Esquema funcional del sistema

6. Estado del arte

En este apartado vamos a analizar la tecnología actual en lo referente a la detección de objetos mediante visión artificial. Se expondrán algunos de los diferentes métodos existentes para desarrollar un sistema de este tipo y se analizará su funcionamiento.

6.1 Clasificadores

El aspecto más importante en la detección de objetos es la anotación o etiquetado de imágenes. Este proceso es el encargado de identificar lo que a la visión artificial se le proporciona como entrada. Para conseguir que este proceso llegue a buen puerto, es necesario que la máquina aprenda a identificar, es decir, que disponga de la información necesaria para identificar el objeto en cuestión. Esta información se obtiene de los clasificadores.

Ya se ha recalcado anteriormente la importancia de los clasificadores dentro de un sistema de detección de objetos, pero no se ha descrito lo que son en concreto.

Un clasificador es un algoritmo que implementa una forma de clasificar. Esta definición hace referencia a la clasificación estadística, donde a partir una entrada de información se debe obtener como salida la categorización de ella. Si se habla del funcionamiento de un clasificador, tiene otros factores que afectan a la categorización o clasificación de la información.

En este proyecto se trata la clasificación de imágenes, pero también intervienen otros procesos antes de etiquetar las imágenes.

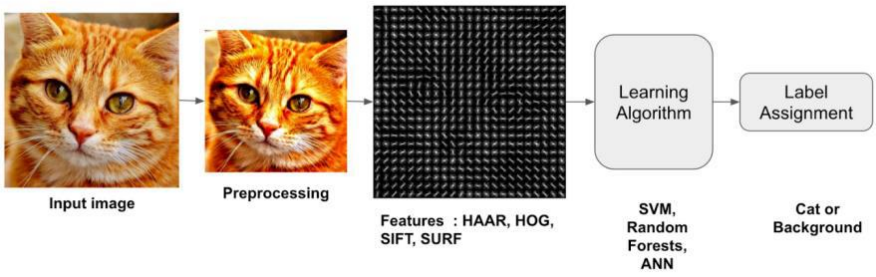

Ilustración 6: Proceso de etiquetado de una imagen. [17]

El proceso inicia con una imagen como entrada la cual se deberá clasificar, pero antes esta imagen se ha de pre-procesar, normalmente definiendo un determinado tamaño y/o transformando la imagen a escala de grises.

Una vez la imagen es apta se aplicará un descriptor (algunos ejemplos de descriptores son: HAAR, HOG, SIFT, SURF, etc.) que permitirá obtener una gran variedad de características de la imagen. A partir de estas características, se emplea un clasificador que va a etiquetar las imágenes según su algoritmo y las características que le resulten útil de la imagen. [18]

Actualmente existen una gran variedad de clasificadores según los métodos y algoritmos que emplean. A continuación, se expondrán los más comunes:

6.1.1 Cascadas de Haar. Método de Viola – Jones

El sistema de detección de objetos de Viola-Jones es el primer sistema de detección de objetos en ofrecer tasas competitivas para la detección de objetos en tiempo real propuesto en 2001 por Paul Viola y Michael Jones [19]. Éste será el método seguido en este proyecto.

A pesar de que pueden ser entrenados para detectar una variedad de clases de objetos, fue motivada principalmente por el problema de la detección de rostros.

Características y evaluación

Las características en las que se basa el sistema de detección general se fundamentan en la suma de los píxeles de la imagen dentro de áreas rectangulares. Como tales, tienen cierto parecido a las funciones base Haar [20], que han sido utilizadas con anterioridad en el ámbito de la detección de objeto. Sin embargo, dado que todas las características utilizadas por Viola y Jones se basan en más de un área rectangular, son, por lo general, más complejas.

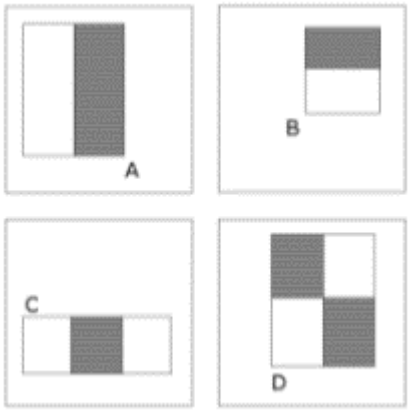

Ilustración 7: Características "rectángulo". [21]

En la ilustración anterior se pueden identificar cuatro tipos de las mencionadas características "rectángulo".

El valor de una característica "de dos rectángulos" (A, B) es la diferencia entre la suma de los píxeles dentro de dos regiones rectangulares. Las regiones tienen el mismo tamaño y forma y están horizontal o verticalmente adyacentes.

La característica "de tres rectángulos" (C) calcula la suma dentro de los dos rectángulos exteriores y se le resta a la suma en el rectángulo central.

Por último, una característica "de cuatro rectángulos" (D) calcula la diferencia entre pares diagonales de los rectángulos.

Como es de esperar, los elementos rectangulares de este tipo son bastante primitivos en comparación con otras alternativas como filtros orientables. Sin embargo, con el uso de una representación de imagen denominada "imagen integral", los elementos rectangulares se pueden evaluar en tiempo constante, lo que le da a este método una ventaja considerable con respecto a la velocidad de sus "familiares" más sofisticados.

Imagen Integral

Las características "rectángulo" se pueden calcular muy rápidamente usando una representación intermedia de la imagen denominada imagen integral [22]. La imagen integral en el lugar *(x,y)* contiene la suma de los píxeles situados arriba y a la izquierda de *(x,y):*

$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x', y')$$

Donde *ii(x,y)* es la imagen integral y la imagen original es *i(x,y).*

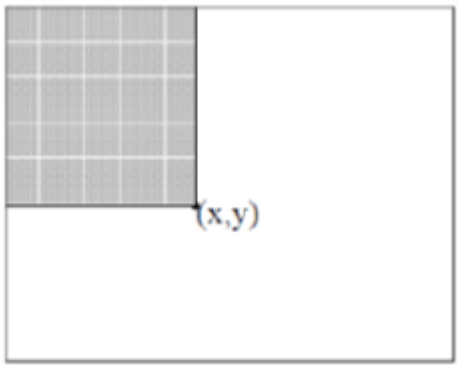

Ilustración 8: Imagen integral. [22]

Arquitectura en Cascada

La evaluación de los clasificadores buenos generados por el proceso de aprendizaje se puede hacer rápidamente, pero no es lo suficientemente rápido como para ejecutarse en tiempo real. Por este motivo, los clasificadores robustos están dispuestos en una cascada por orden de complejidad, donde cada clasificador sucesivo es entrenado sólo en las muestras seleccionadas que pasan a través de los clasificadores anteriores. Si en cualquier etapa de la cascada un clasificador rechaza la sub-ventana bajo la inspección, no se lleva a cabo ningún tratamiento posterior y se continúa buscando en la siguiente sub-ventana.

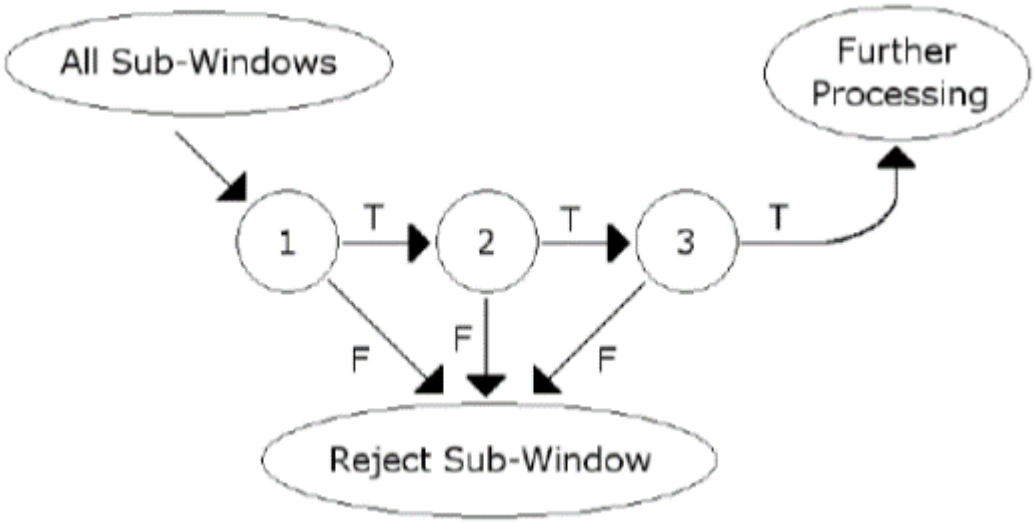

Ilustración 9: Esquema de arquitectura en cascada. [23]

La arquitectura en cascada tiene implicaciones interesantes para el rendimiento de los clasificadores individuales. Debido a la activación de cada clasificador, depende totalmente de la conducta de su predecesor. La tasa de falsos positivos para una cascada entera es:

$$F = \prod_{i=1}^{K} fi$$

Del mismo modo, la tasa de detección es:

$$D = \prod_{i=1}^{K} di$$

Por lo tanto, para que coincida con las tasas de falsos positivos normalmente conseguidos por los otros detectores, cada clasificador puede llegar a tener resultados sorprendentemente pobres. Por ejemplo, para una cascada de 32 etapas, para lograr una disminución de la tasa de falsos positivos de un 60%, cada clasificador puede llegar a alcanzar una tasa de falsos positivos de alrededor del 65%. Al mismo tiempo, sin embargo, cada clasificador tiene que ser excepcionalmente capaz de alcanzar tasas de detección adecuadas. Por ejemplo, para lograr una tasa de detección de alrededor del 90%, cada clasificador de la mencionada cascada necesita alcanzar una tasa de detección de aproximadamente 99.7%. [23]

6.1.2 Local Binary Patterns

El operador de textura Local Binary Patterns (LBP) fue introducido por primera vez como una medida complementaria para el contraste de la imagen local. La primera versión operaba con los ocho vecinos de un píxel, utilizando el valor del píxel central como umbral. Un código LBP para un "vecindario" ha sido producido multiplicando los valores límite con las ponderaciones asignadas a los píxeles correspondientes y sumando el resultado.

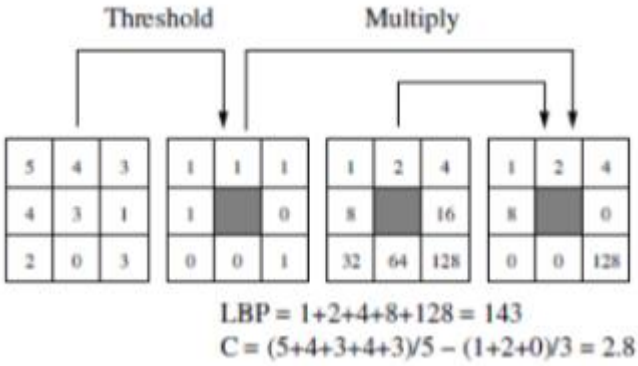

Ilustración 10: Ejemplo de cálculo de código LBP original y una medida de contraste. [24]

Dado que el LBP es, por definición, invariante a los cambios mono-tónicos en escala de grises, se complementó con una medida independiente de contraste local. La ilustración anterior muestra cómo se ha obtenido la medida de contraste (C). El promedio de los niveles de gris por debajo del píxel central se resta del de los niveles de gris igual o por encima que el píxel central. [24]

En la forma actual del operador LBP de la versión básica, la definición original se extiende a "vecindarios" circulares arbitrarios. Se han desarrollado varias variantes de esta idea básica. La idea básica es, sin embargo, la misma: un código binario que describe el patrón de la textura local es construido mediante el umbral de un "vecindario" por el valor de gris de su centro. El operador está relacionado con muchos métodos de análisis de texturas conocidos.

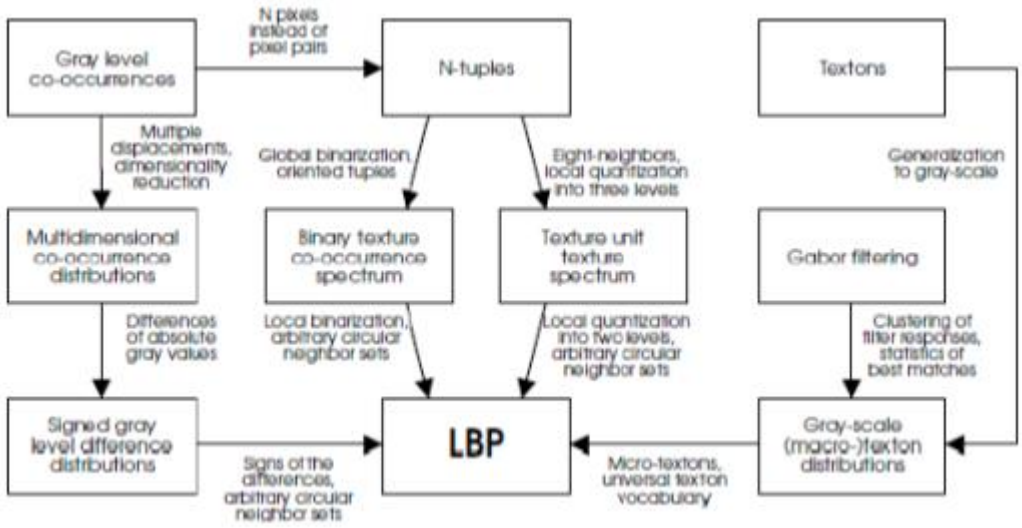

Ilustración 11: LBP en el campo de los operadores de análisis de texturas. [25]

6.1.3 Distancia Euclídea

La distancia Euclídea no es un método de clasificación de imágenes como tal, sino que es un método de cálculo que se empleará para el cálculo de las diferencias (distancias) entre imágenes. Este método se puede aplicar al modelo de "Vecino más Cercano" para seleccionar

las imágenes más aptas para la imagen de entrada. No se entrará a explicar con profundidad el método de "Vecino más Cercano" ya que es demasiado extenso.

Las reglas de clasificación basadas en distancia son las siguientes: [26]

Sea *d: E x E → R* una métrica y sea *y = y(x)* un punto de *E*; teniendo en cuenta el caso de Vecino más Cercano "1-NN", sea P_c un conjunto de prototipos representante de la clase *c*:

$$x \in c` \leftrightarrow \exists\, p \in Pc`: d(y,p) \leq d(y,p') \forall p' \in Pc, 1 \leq c \leq C, c \neq c'$$

En caso de empate, se decide, entre las empatadas, la clase con mayor número de prototipos. Teniendo en cuenta el caso de la Distancia Euclídea, dicha distancia será:

$$d(P1,P2) = \sqrt{(x2-x1)^2 + (y2-y1)^2}$$

En resumen, se clasifica *y* en la clase a la que pertenezcan la mayoría de sus *k* vecinos más próximos.

6.1.4 Análisis de Componentes Principales

En estadística, el Análisis de Componentes Principales o ACP (en inglés Principal Component Analysis o PCA) es una técnica utilizada para reducir la dimensionalidad de un conjunto de datos. Intuitivamente, la técnica sirve para hallar las causas de la variabilidad de un conjunto de datos y ordenarlas por importancia.

Técnicamente, el PCA busca la proyección según la cual los datos queden mejor representados en términos de mínimos cuadrados. Este método se emplea sobre todo en análisis exploratorio de datos y para construir modelos predictivos.

El PCA construye una transformación lineal que escoge un nuevo sistema de coordenadas para el conjunto original de datos en el cual la varianza de mayor tamaño del conjunto de datos es capturada en el primer eje, denominado el Primer Componente Principal, la segunda varianza más grande es el segundo eje, y así sucesivamente. Para construir esta transformación lineal debe construirse primero la matriz de covarianza o matriz de coeficientes de correlación. Debido a la simetría de esta matriz, existe una base completa de vectores propios de la misma. La transformación que lleva de las antiguas coordenadas a las coordenadas de la nueva base es, precisamente, la transformación lineal necesaria para reducir la dimensionalidad de datos. Además, las coordenadas en la nueva base dan la composición en factores subyacentes de los datos iniciales.

Una de las ventajas del PCA para reducir la dimensionalidad de un grupo de datos es que retiene aquellas características del conjunto de datos que contribuyen más a su varianza, manteniendo un orden de bajo nivel de los componentes principales e ignorando los de alto nivel. El objetivo es que esos componentes de bajo orden a veces contienen el aspecto más importante de esa información. [27]

6.1.5 Máquinas de Vectores de Soporte

Las máquinas de Vectores Soporte (en inglés, Support Vector Machines o SVMs) son un conjunto de algoritmos de aprendizaje supervisado desarrollados por Vladimir Vapnik y su equipo en los laboratorios AT&T.

Estos métodos están propiamente relacionados con problemas de clasificación y regresión. Dado un conjunto de ejemplos de entrenamiento o muestras, podemos etiquetar las clases y entrenar una SVM para construir un modelo que prediga la clase de una nueva muestra. Intuitivamente, una SVM es un modelo que representa a los puntos de muestra en el espacio, separando las clases por un espacio lo más amplio posible. Cuando las nuevas muestras se ponen en correspondencia con dicho modelo, en función de su proximidad pueden ser clasificadas a una u otra clase.

Más formalmente, una SVM construye un hiperplano o conjunto de hiperplanos en un espacio de dimensionalidad muy alta (o incluso infinita) que puede ser utilizado en problemas de clasificación o regresión. Una buena separación entre las clases permitirá una clasificación correcta.

Dado un conjunto de puntos, subconjunto de un conjunto mayor (espacio), en el que cada uno de ellos pertenece a una de dos posibles categorías, un algoritmo basado en SVM construye un modelo capaz de predecir si un punto nuevo pertenece a una categoría o a la otra.

Como en la mayoría de los métodos de clasificación supervisada, los datos de entrada son vistos como un vector p-dimensional. La SVM busca un hiperplano que separe de forma óptima a los puntos de una clase de la de otra, que eventualmente han podido ser previamente proyectados a un espacio de dimensionalidad superior. En ese concepto de "separación óptima" es donde reside la característica fundamental de las SVM: este tipo de algoritmos buscan el hiperplano que tenga la máxima distancia, o margen, con los puntos que estén más cerca de él mismo. Por eso, también a veces se conoce a las SVM como clasificadores de margen máximo. De esta forma, los puntos del vector que sean etiquetados con una categoría estarán a un lado del hiperplano y los casos que se encuentren en la otra categoría estarán al otro lado.

En la literatura de las SVM, se llama atributo a la variable predictora y característica a un atributo transformado que es usado para definir el hiperplano. La elección de la representación más adecuada del universo estudiado se realiza mediante un proceso denominado selección de características. Al vector formado por los puntos más cercanos al hiperplano se le llama vector de soporte.

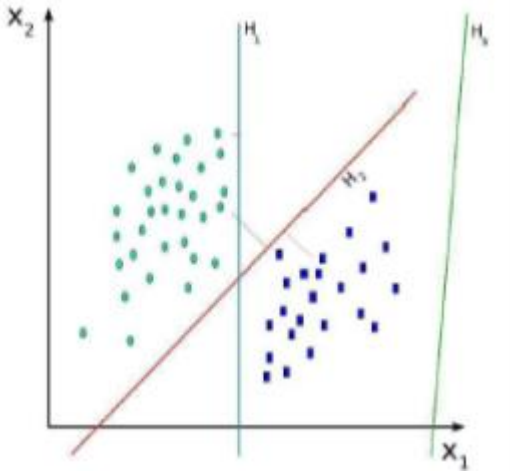

Ilustración 12: Ejemplo de clasificación SVM. [28]

La representación de los datos a clasificar se realiza en el plano x-y. El algoritmo SVM trata de encontrar un hiperplano unidimensional (una recta) que una a las variables predictoras y constituya el límite que define si un elemento de entrada pertenece a una categoría o a la otra. Existe un número infinito de posibles hiperplanos que realicen la clasificación, pero, ¿cuál es el mejor y cómo lo definimos? La mejor solución es aquella que permita un margen máximo entre los elementos de las dos categorías.

Idealmente, el modelo basado en SVM debería producir un hiperplano que separe completamente los datos del universo estudiado en dos categorías. Sin embargo, una separación perfecta no siempre es posible y, si lo es, el resultado del modelo no puede ser generalizado para otros datos. Esto se conoce como sobreajuste (overfitting). Con el fin de permitir cierta flexibilidad, las SVM manejan un parámetro *C* que controla la compensación entre errores de entrenamiento y los márgenes rígidos, creando así un margen blando (soft margin) que permita algunos errores en la clasificación, a la vez que los penaliza. [29]

6.2 OpenCV

Una de las herramientas utilizadas en este proyecto y en la mayoría de los proyectos de visión artificial es *OpenCV*. *OpenCV* es una librería de visión por computador de código abierto. Está escrito en C y C++ y corre bajo Linux, Windows y Mac OS X. Hay un desarrollo activo de las interfaces para Python (ya disponible), Ruby, Matlab y otros lenguajes.

OpenCV ha sido diseñado para la eficiencia computacional y con un fuerte enfoque en aplicaciones de tiempo real. Está escrito en C optimizado y puede aprovechar los procesadores multinúcleo.

Uno de los objetivos de OpenCV es proporcionar una infraestructura de visión artificial fácil de utilizar, que ayuda a crear rápidamente aplicaciones de visión bastante sofisticadas. La librería OpenCV contiene más de 500 funciones que abarcan muchas áreas de la visión, incluyendo el reconocimiento de productos de fábricas, imágenes médicas, seguridad, interfaz de usuario, calibración de cámara, visión estéreo y robótica. Debido a que la visión por computador y el aprendizaje automático aparecen a menudo juntos, OpenCV contiene también una completa Librería de Aprendizaje Automático o, en inglés, Machine Learning Library (MLL). Esta librería se centra en el reconocimiento de patrones estadísticos y de

agrupamiento. La MLL es de gran utilidad para las tareas de visión que se encuentran en el núcleo de la misión de OpenCV, pero es lo suficientemente general como para ser utilizada para cualquier problema de aprendizaje automático. [30]

6.3 SimpleCV

Otra de las herramientas disponibles para trabajar con proyectos de visión artificial es SimpleCV. SimpleCV es un "framework" de código abierto para desarrollar aplicaciones de visión por computadora. Con el podremos tener acceso a una gran cantidad de librerías de Computer Vision como OpenCV, sin tener que aprender con profundidad sobre formatos de archivos, espacio de colores, manejo de buffer y otros. [31]

Está escrita en lenguaje Python. Debido a la naturaleza de Python, puede ejecutar "scripts" o usar un "shell" interactivo para hacer cosas de visión por computadora y tareas relacionadas.

6.4 Accord.NET

Una variante más que se puede usar en trabajos como este es Accord.NET. Accord.NET es un "framework" para la computación científica. El marco comprende un conjunto de librerías que están disponibles en el código fuente, así como a través de instaladores ejecutables y paquetes *NuGet*. Las principales áreas cubiertas incluyen álgebra lineal numérica, optimización numérica, estadísticas, aprendizaje automático, redes neuronales artificiales, procesamiento de señales e imágenes y bibliotecas de soporte (como trazado y visualización de gráficas).

El proyecto se creó originalmente para ampliar las capacidades de AForge.NET Framework, pero desde entonces ha incorporado AForge.NET dentro de sí mismo. Las versiones más recientes unieron ambos marcos bajo el nombre de Accord.NET. [32]

6.5 MatLab

Otra de las herramientas utilizadas en proyectos de este tipo es MatLab. Matrix Laboratory (MatLab) es un software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M). Está disponible para las plataformas Unix, Windows y Apple Mac OS X.

Entre sus prestaciones básicas se hallan:

- Manipulación de matrices.
- Representación de datos y funciones.
- Implementación de algoritmos.
- Creación de interfaces de usuario (GUI).
- Comunicación con programas en otros lenguajes y con otros dispositivos hardware.

El paquete MATLAB dispone de dos herramientas adicionales que expanden sus prestaciones, a saber, Simulink (plataforma de simulación multidominio) y GUIDE (editor de interfaces de usuario). Además, se pueden ampliar las capacidades de MATLAB con los toolboxes; y las de Simulink con los blocksets.

Es un software muy utilizado en universidades y centros de investigación y desarrollo. En los últimos años, ha aumentado el número de prestaciones, como la de programar directamente procesadores digitales de señal o crear código VHDL (lenguaje utilizado para describir circuitos digitales).

MATLAB es un programa de cálculo numérico orientado a matrices. Por lo tanto, será más eficiente si se diseñan los algoritmos en términos de matrices y vectores. [23]

6.6 Gedit

La herramienta principal y la más básica utilizada para escribir programación es el editor de textos. En el caso de las distribuciones de *Linux*, el editor de textos del sistema es Gedit.

Además de las funcionalidades básicas que son habituales en un editor de texto, como copiar, cortar y pegar texto, imprimir, etc., Gedit incorpora coloreado de sintaxis para diversos lenguajes de programación y marcado. Gedit también posee pestañas en su interfaz para editar múltiples archivos a la vez. Puede editar archivos de manera remota usando la biblioteca GVFS. Otras características orientadas al código incluyen numeración de líneas, resaltado de la línea actual, identificación automática y copiado de seguridad del archivo. [33]

Además, Gedit incluye un corrector ortográfico multilenguaje y un flexible sistema de plugins que permite añadir características a la aplicación. Además de los complementos incluidos en Gedit, hay más disponibles para descargar.

La aplicación está diseñada para funcionar sobre X Window System, usando las librerías GTK+ y GNOME. El editor está integrado con el resto de GNOME, soportando el arrastrado de archivos entre Nautilus hacia el cuerpo principal de la aplicación, uso del framework de impresión del gestor de GNOME y su sistema integrado de ayuda.

7. Análisis de alternativas

A continuación, se analizarán las posibles alternativas que se pueden seguir para la realización del sistema. Partiendo de lo expuesto en el apartado de Estado del Arte, se valorarán las alternativas según los criterios pertinentes.

Ya que la etapa más importante en la creación de un sistema de reconocimiento de objetos basado en visión artificial es la de la creación del clasificador, se analizarán los posibles caminos para conseguir desarrollarlo.

Se deben de tener en cuenta dos aspectos a la hora de entrenar el clasificador: el algoritmo de reconocimiento seguido y los parámetros del entrenamiento.

7.1 Algoritmos de reconocimiento

Como ya se ha explicado en el apartado de Estado del arte, es necesario utilizar un algoritmo de reconocimiento, denominado descriptor, para extraer de la imagen sus características principales. A continuación, se analizarán los diferentes algoritmos expuestos previamente.

Para analizar los resultados, se valorarán según los siguientes criterios:

- TPR (True Positive Ratio): Ratio de detección positiva correcta del objeto. Relación entre los positivos correctos detectados (*TP*) y el total de los positivos detectados.

$$TPR = \frac{TP}{Total\ Positives} \cdot 100$$

- Precision / Hit Rate: Tasa de acierto. Relación entre los positivos correctos detectados y el total de las detecciones (positivas y negativas (*FP*)).

$$Hit\ Rate = \frac{TP}{TP + FP} \cdot 100$$

- ATpx (Average Time per Pixel): Relación entre el tiempo de procesamiento y el total de los pixels.

$$ATpx = \frac{Total\ Processing\ Time}{Total\ Pixel\ Amount}$$

A continuación se mostrará un análisis cuyos resultados son perfectamente aplicables al sistema de este proyecto [34]. En este análisis se utilizarán los mismos parámetros de entrenamiento del clasificador para todos los algoritmos, de tal manera que se pueda analizar cada uno en las mismas condiciones. Se utilizarán los valores por defecto de dichos parámetros.

Los resultados que se verán a continuación muestran que las cascadas Haar y las cascadas LBP tienen una tasa de acierto similar, de alrededor de un 80%. Sin embargo, la tasa precisión de estos algoritmos es diferente; en el caso de las cascadas LBP es de al menos 78% mientras que las cascadas Haar tienen una precisión de un 50%. En cuanto a rapidez de procesamiento, ambos tienen un comportamiento similar en caso de completarse todas las etapas propuestas.

Tabla 1: Análisis de las cascadas Haar. [34]

Win.	Stages	TP	TPR	FP	FN	Prec.	ATpx (ms)
32px	20	40	80%	47	9	46%	$4{,}11{\cdot}10^{-4}$
	25	40	80%	36	9	52,6%	$4{,}81{\cdot}10^{-4}$
	30	40	80%	37	9	51,9%	$3{,}76{\cdot}10^{-4}$
	35	41	82%	39	8	51,3%	$3{,}61{\cdot}10^{-4}$
	40	43	86%	60	7	43%	$2{,}43{\cdot}10^{-4}$
50px	20	39	78%	13	11	75%	$37{,}9{\cdot}10^{-4}$
	25	40	80%	10	10	80%	$48{,}1{\cdot}10^{-4}$
	30	39	78%	12	11	76,5%	$48{,}66{\cdot}10^{-4}$
	35	39	78%	11	11	78%	$36{,}16{\cdot}10^{-4}$
	40	40	80%	11	10	78,4%	$33{,}67{\cdot}10^{-4}$

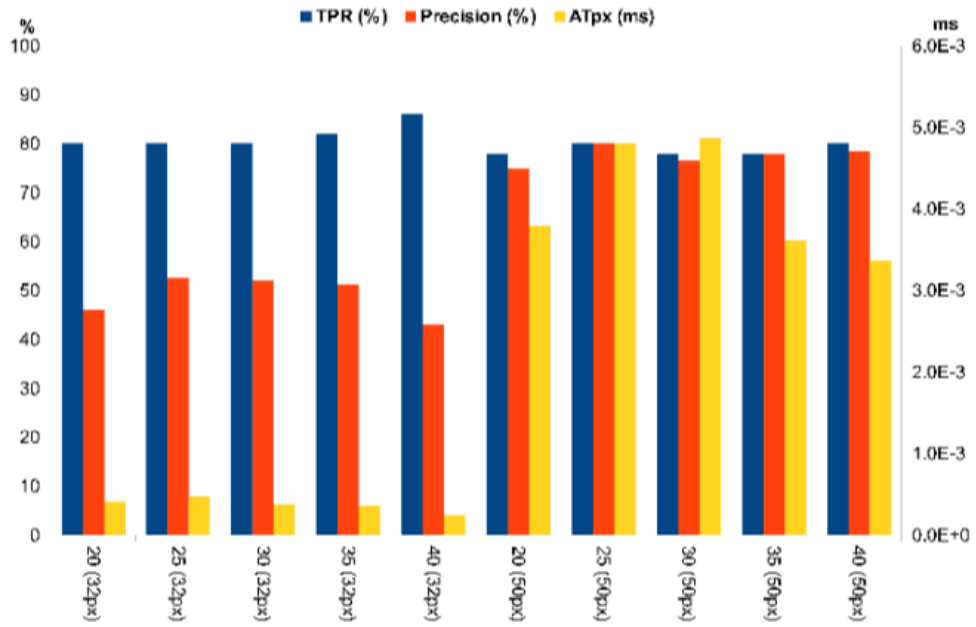

Figura 1: Resultados de las cascadas Haar. [34]

Tabla 2: Análisis de las cascadas LBP. [34]

Win.	Stages	TP	TPR	FP	FN	Prec.	ATpx (ms)
32px	20	39	78%	11	11	78%	$5 \cdot 10^{-4}$
	25	39	78%	11	11	78%	$2{,}85 \cdot 10^{-4}$
	30	39	78%	11	11	78%	$1{,}69 \cdot 10^{-4}$
	35	40	80%	11	10	78,4%	$1{,}11 \cdot 10^{-4}$
	40	38	76%	24	12	61,3%	$0{,}78 \cdot 10^{-4}$
50px	20	39	78%	11	11	78%	$40{,}8 \cdot 10^{-4}$
	25	39	78%	11	11	78%	$26{,}4 \cdot 10^{-4}$
	30	40	80%	10	10	80%	$15{,}3 \cdot 10^{-4}$
	35	40	80%	10	10	80%	$8{,}82 \cdot 10^{-4}$
	40	40	80%	10	10	80%	$5{,}35 \cdot 10^{-4}$

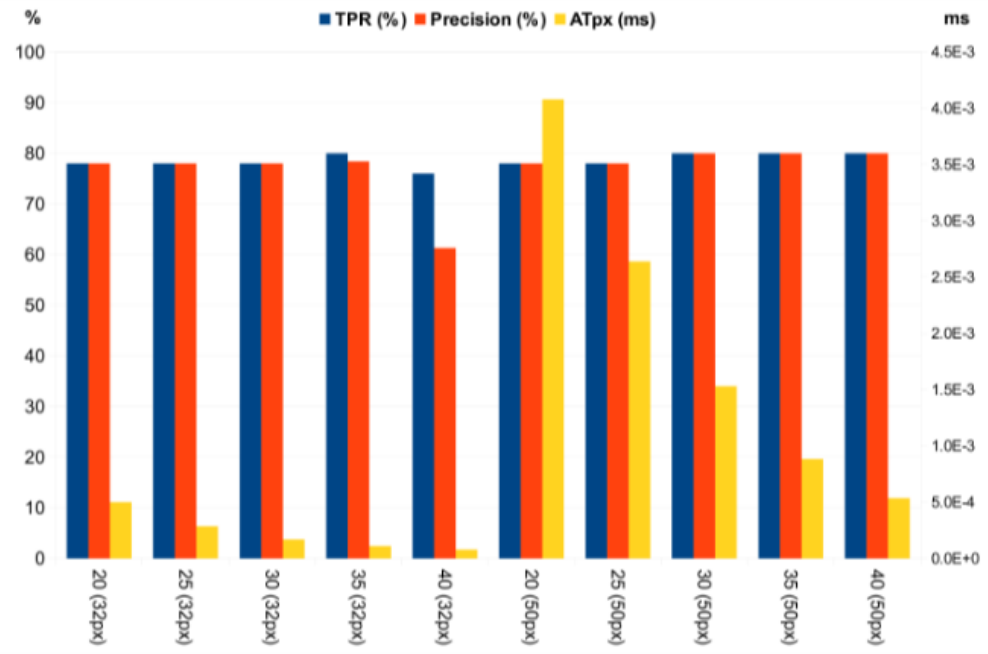

Figura 2: Resultados de las cascadas LBP. [34]

En cuanto a la configuración HOG + SVM, es evidente que tiene unos resultados bastante diferentes de los dos anteriores algoritmos. Atendiendo a la tasa de falsos positivos podemos observar que los resultados con mejor tasa de acierto también tienen la peor precisión. La varianza de estos parámetros es excesivamente grande, desde un 24% hasta un 94% en el caso del Hit Rate y desde un 4% hasta un 51% en el caso de la precisión. Por otro lado, se observa que la velocidad de procesamiento es mayor que la de las anteriores observaciones.

Se reconocieron la mayoría de los objetos, pero hubo demasiadas detecciones incorrectas. Esta configuración no es productiva, ya que sería necesario utilizar otro método adicional para separar los falsos positivos de los positivos correctos para conseguir unos resultados

aceptables. Es por esto por lo que este algoritmo ha sido descartado para la realización de este proyecto.

Tabla 3: Análisis de la configuración HOG + SVM. [34]

Win.	Kernel	TP	TPR	FP	FN	Prec.	ATpx (ms)
32 px	Linear	47	94%	961	3	4,66%	$273{,}5{\cdot}10^{-4}$
	RBF	45	90%	992	5	4,34%	$845{,}1{\cdot}10^{-4}$
	Sigmoidal	46	92%	923	4	4,75%	$661{,}1{\cdot}10^{-4}$
64 px	Linear	23	46%	27	27	46%	$625{,}5{\cdot}10^{-4}$
	RBF	12	24%	41	38	22,64%	$3463{\cdot}10^{-4}$
	Sigmoidal	23	46%	27	27	46%	$3081{,}6{\cdot}10^{-4}$
128 px	Linear	27	54%	26	23	50,94%	$792{,}5{\cdot}10^{-4}$
	RBF	24	48%	95	26	20,17%	$15523{,}6{\cdot}10^{-4}$
	Sigmoidal	23	46%	27	27	46%	$10913{,}7{\cdot}10^{-4}$

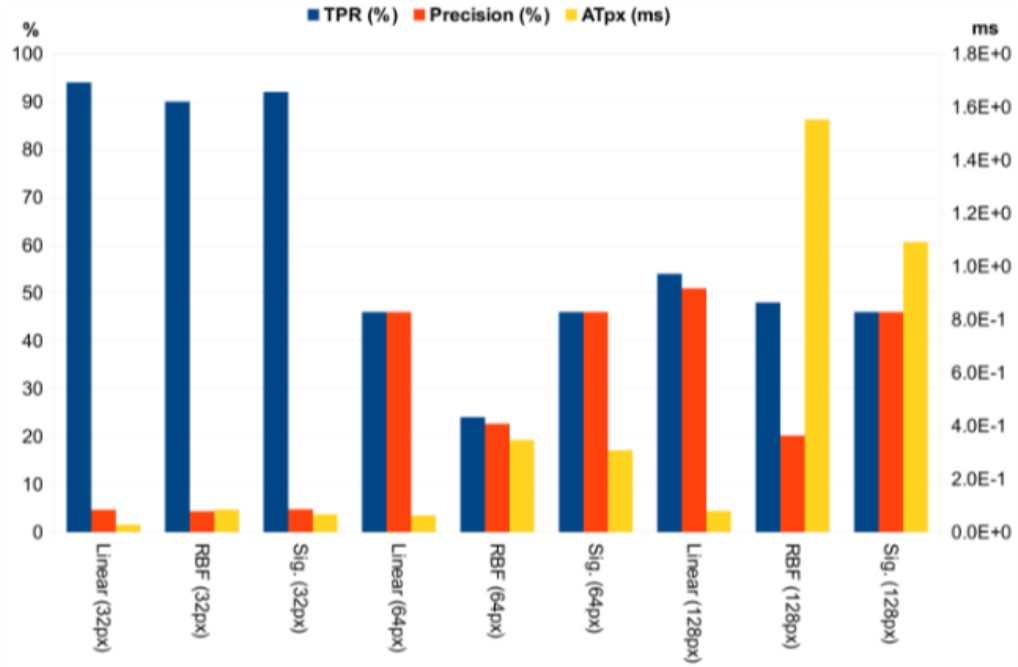

Figura 3: Resultados de la configuración HOG + SVM. [34]

Faltaría por analizar la configuración PCA + SVM, pero se ha realizado el estudio sobre esta ya que teóricamente los resultados son aún peores que los de la configuración HOG + SVM. En las siguientes figuras se puede observar los resultados de esta configuración, de la anterior y los del método SVM en particular, además de la línea morada que simula el comportamiento de los dos algoritmos analizados primeramente. [35]

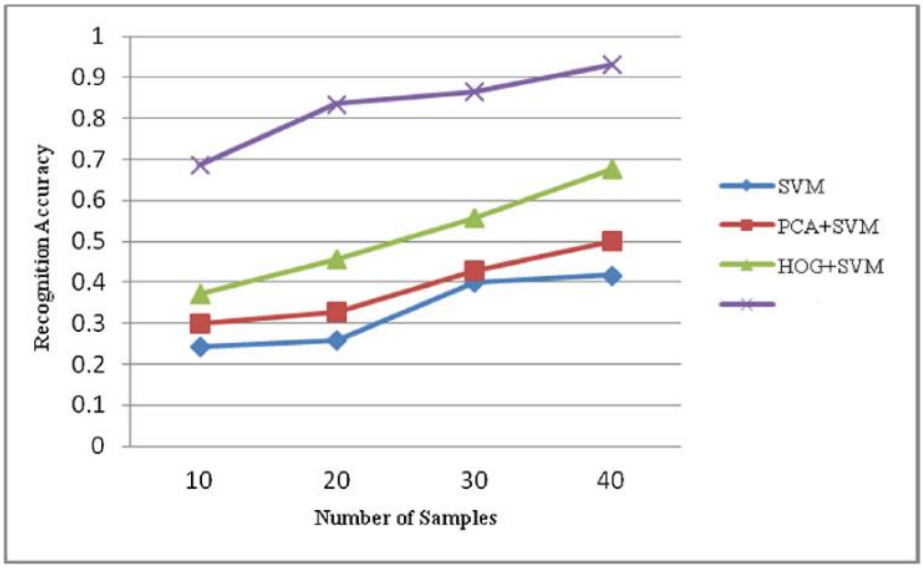

Figura 4: Resultados PCA + SVM en comparación con HOG + SVM y el resto de los métodos. [35]

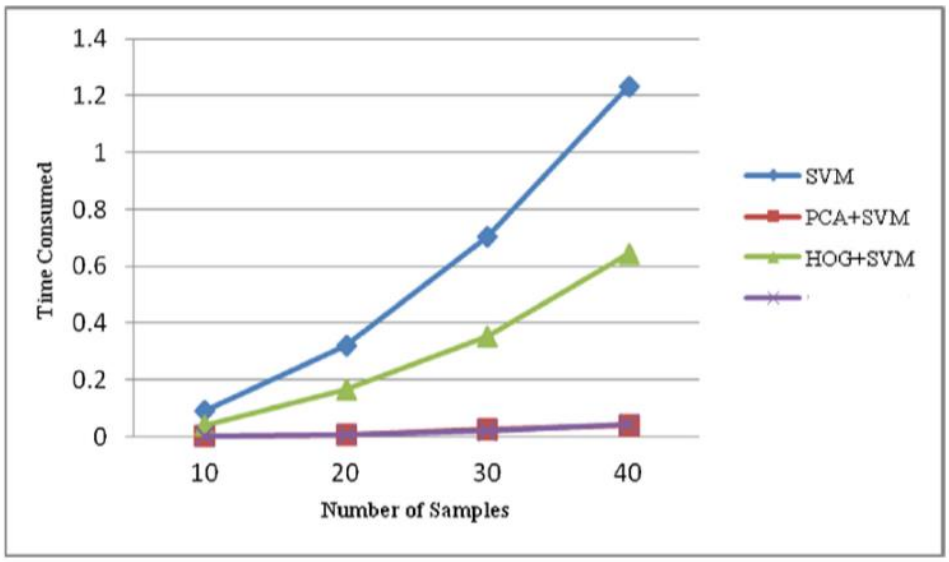

Figura 5: Velocidad de procesamiento de PCA + SVM en comparación con HOG + SVM y el resto de los métodos. [35]

Atendiendo a estos resultados, queda claro que la decisión acerca de qué algoritmo utilizar se reduce a las cascadas Haar o las cascadas LBP.

Los resultados experimentales muestran que las cascadas LBP son ligeramente mejores, pero para la realización de este proyecto se ha decidido utilizar las cascadas de Haar, ya que los resultados de otros investigadores indican que es preferible usar este método.

El método de cascadas de Haar está más desarrollado y, como se verá a continuación, permite operar con un rango muy amplio de parámetros que se pueden ajustar dependiendo del conjunto de datos de entrenamiento con el que se trabaje.

7.2 Librerías de programación

En cuanto a la decisión de utilizar las librerías de *OpenCV* en lugar de sus similares *SimpleCV* o *Accord.NET*, esta decisión fue marcada por la recomendación del tutor. Al ser utilizado como punto de partida para comenzar con el proyecto, las decisiones posteriores se tomaron a partir de lo que más convenía para el desarrollo del sistema con las librerías de *OpenCV*.

7.3 Plataforma de programación

Las plataformas de programación se eligen, por norma general, a gusto del programador. En el caso de este proyecto, para la programación de la aplicación que hace uso del clasificador se ha elegido programar en lenguaje C++ sobre el editor de textos *Gedit* en la distribución *Ubuntu* del sistema operativo *Linux*. Para el desarrollo del clasificador se ha trabajado en el ejecutor de comandos del sistema.

No se entrará a analizar el resto de alternativas, ya que no hay un criterio de decisión unánime ni datos estadísticos objetivos sobre este tema. Es cuestión de comodidad y de costumbrismo. En el caso que concierne a este trabajo, se ha estado programando bajo las condiciones elegidas durante toda la docencia de la universidad. Es por ello por lo que se ha decidido seguir utilizando la plataforma descrita.

7.4 Parámetros para la creación del clasificador

Como se verá más adelante, la etapa del proyecto que más costosa ha resultado ha sido la elección de los parámetros a utilizar para el desarrollo del clasificador mediante el método de las cascadas Haar. Estos parámetros son los correspondientes a la herramienta de *OpenCV traincascade*.

Para empezar, se describirán los parámetros que van a ser estudiados:

- Número de muestras positivas (Npos): Determina el número de muestras positivas a utilizar en el proceso de entrenamiento. Puede ir desde 0 hasta el número de imágenes positivas disponibles.
- Número de muestras negativas (Nneg): Análogo al anterior, pero con las muestras que no contienen el objeto.
- Número de etapas (Nstages): número de etapas totales a completar en el proceso en cascada de entrenamiento. En teoría, cuantas más etapas mejor será el entrenamiento.
- Tasa de acierto mínima (MHRate): Mínimo Hit Rate (número de objetos detectados en relación con los totales presentes) requerido para el clasificador.
- Alarma de máxima tasa de falsos (MFARate): Decide el máximo número de falsos positivos que se permiten detectar en el entrenamiento.

- Tasa de ajuste de peso (WTRate): Dentro de la etapa de la cascada, las WTRate muestras con menor peso serán descartadas.
- Cuenta de debilidad (Wcount): Establece la cantidad máxima de árboles débiles que deben participar en cada etapa del entrenamiento.

También será necesario describir los criterios de evaluación que se van a utilizar. Algunos son iguales o similares a los utilizados anteriormente:

- Tiempo de entrenamiento (TT): Tiempo requerido para entrenar el clasificador, medido en segundos.
- Tiempo de detección (DT): Tiempo requerido por el clasificador para realizar una detección, medido en segundos/Mpixel.
- Tasa de falsos positivos (FPR): tasa de detección errónea.
- Tasa de positivos correctos (TPR): tasa de detección correcta.
- Índice de positivos correctos (TPI): TPR relativizado a cada imagen o frame.

A continuación, se mostrará un análisis del efecto de las variaciones de dichos parámetros en los resultados [36]:

- Variación de Npos:

A medida que aumenta el valor de Npos, el TT disminuye. Esta disminución es a costa de un aumento de FPR y una disminución del TPI. Por otro lado, TPR muestra un comportamiento casi constante. El tiempo de detección casi no se ve afectado por las variaciones en Npos. En un caso especial donde Npos es 100 y otros parámetros se mantienen por defecto, el entrenamiento se detiene en la octava etapa cuando el MFARate alcanza su valor máximo establecido. Existe una compensación entre TT y TPI. En conclusión, se considera que el valor óptimo de Npos debe estar aproximadamente entre 68% y 85% del total de positivos disponibles.

- Variación de Nneg:

Con el aumento en Nneg, TT muestra un crecimiento constante, mientras que DT casi permanece constante. Nneg tiene un efecto muy bajo en el TRI inicialmente, pero muestra un incremento gradual cuando Nneg aumenta. Como el resto de los parámetros aún no están optimizados, el efecto de Nneg no puede ser observado a gran escala. Se puede concluir que el valor óptimo de Nneg es entre 90% y 98% de los negativos disponibles.

- Variación de Nstages:

A medida que Nstages aumenta, DT muestra un incremento muy gradual después de cada etapa del entrenamiento. Por otro lado, el valor promedio de TPI aumenta y, por lo tanto, el valor promedio de FPR disminuye. Similarmente a DT, con el aumento en Nstages, el

valor de TT aumenta. El límite del valor de Nstages dependerá de las características hardware de la máquina que se encargue del entrenamiento. Para un ordenador de las características del disponible, dicho valor se estima que estará entre 15 y 25.

- Variación de MHRate:

Cuanto menor sea el valor de MHRate, más muestras se desperdician para cada etapa, dejando sin muestras a las etapas superiores a 5 o 6. TT aumenta hasta que MHRate alcanza un valor de 0,8 desde 0,5 y luego por cada incremento en MHRate, TT disminuye. Hasta que MHRate es igual a 0.9, los clasificadores entrenados alcanzan la MFARate permitida en la etapa 5. Sólo los clasificadores entrenados exitosamente tienen un valor MHRate de 0.95 y 0.999 y, cuanto más alto es el MHRate, menor es el TT y mayor es el TPR. En conclusión, se puede decir que el valor óptimo de MHRate se encuentra aproximadamente entre 0.98 y 0.999.

- Variación de MFARate:

TPR muestra un fuerte aumento cuando MFARate está en el rango de 0.2 a 0.3 y se mantiene alto hasta que MFARate está entre 0.3 y 0.9. TT casi permanece constante, pero disminuye gradualmente hasta que MFARate tiene un valor de entre 0.4 y 0.9. Inesperadamente, TT cae abruptamente y DT aumenta para los valores 0,95 y 0,99; y el entrenamiento se detiene en la etapa 8 (se alcanza la MFARate). TPI y FPR casi permanecen constantes, pero FPR muestra una depresión profunda cuando el valor de MFARate se encuentra entre 0.3 y 0.5. El valor óptimo de MFARate estará entre 0.3 y 0.5, pero para un mayor número de etapas, se prefiere un valor más bajo de MFARate.

- Variación de WTRate:

Con el aumento en WTRate, TT inicialmente permanece constante, pero aumenta después de que WTRate cruce el valor de 0.4. Parece que FPR aumenta después de este punto, pero antes de ello TPR es muy bajo; esto indica que el número de detecciones es muy bajo hasta que WTRate se encuentra por debajo de 0.4. DT cae abruptamente una vez que WTRate cruza el valor 0.5. Una vez que WTRate alcanza un valor de 0.9, el DT continúa cayendo, el FRP disminuye, el TPI aumenta y el TPR continúa en un valor de 1. Por lo tanto, el valor óptimo de WTRate se encuentra entre 0.9 y 0.99.

- Variación de Wcount:

A medida que el valor de Wcount aumenta, el valor de TT aumenta constantemente. El beneficio del aumento en Wcount es que FPR disminuye y TPI aumenta. Pero cuando la cuenta cruza el valor 130, la TPR se reduce. DT tiende a disminuir hasta que el valor de Wcount se encuentra entre 10 y 70, pero no se ve afectado una vez que la cuenta cruza el valor 70. El valor óptimo de Wcount está entre 80 y 120.

Para concluir este análisis, se exponen a continuación los valores utilizados en el proceso de entrenamiento del clasificador utilizado en este proyecto:

Tabla 4: Parámetros utilizados en el entrenamiento

PARÁMETRO	VALOR
Npos	1390
Nneg	3523
Nstages	20
MHRate	0.995
MFARate	0.5
WTRate	0.95^C
Wcount	100

8. Descripción del sistema desarrollado

En este apartado se describirá la realización del proyecto de una forma lineal, explicando cómo puede materializarse un sistema de visión artificial basado en la tecnología explicada en los apartados anteriores para la creación de un sistema de detección de objetos, concretamente, de detección de vehículos.

Se identificarán las diferentes etapas necesarias para la creación del prototipo, indicando en cada una de ellas los pasos seguidos.

Para una mejor comprensión por parte del lector, esta sección se complementa con los anexos explicativos y de código. Se recomienda encarecidamente leerlos.

8.1 Proceso de entrenamiento. Generación del clasificador

La primera etapa del proyecto consiste en "entrenar" el clasificador que, con su información, permitirá la detección de los vehículos. En cuestiones prácticas, el clasificador es un archivo de extensión .xml. El objetivo de esta etapa es conseguir dicho archivo. El esquema que sigue este proceso es el siguiente:

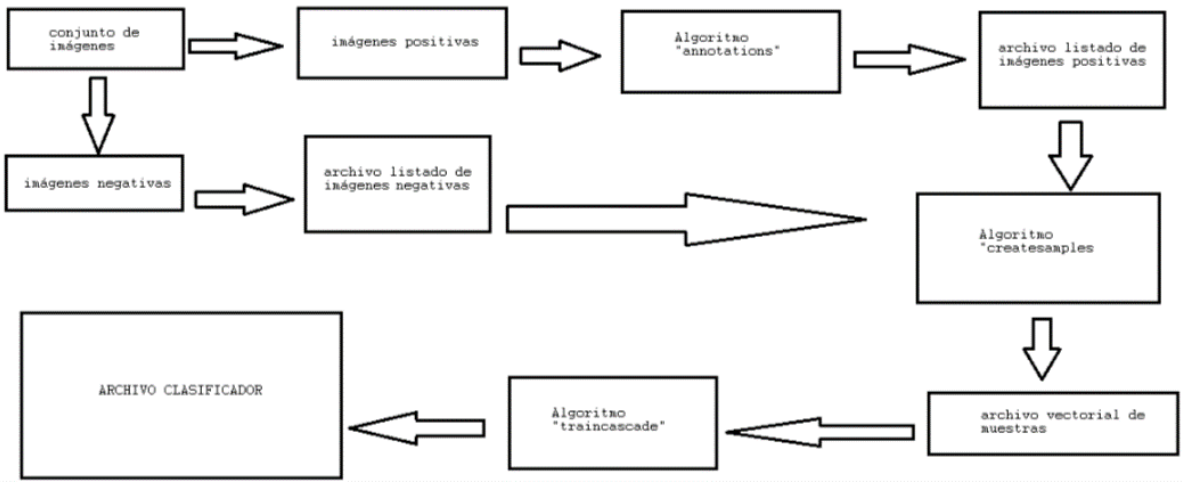

Ilustración 13: Esquema del entrenamiento del clasificador

8.1.1 Conjunto de datos de entrenamiento

Para cualquier tipo de objeto, es necesario construir dos conjuntos de imágenes independientes, ya que se desea crear una cascada Haar, basada en el método de Viola-Jones.

Para empezar, se necesita un "set" de imágenes bastante grande con el que trabajar, algunas positivas (con presencia de vehículos) y otras negativas (sin presencia de vehículos). En este proyecto, se ha creado una base de datos con imágenes de vehículos en carreteras en diferentes situaciones para garantizar la robustez y variedad del entrenamiento.

1. Conjunto de muestras positivas:

En este conjunto se encuentran las imágenes que contienen muestras del objeto que se desea detectar, es decir, aquel para el que se diseña el clasificador. En el caso de este proyecto, se utilizarán imágenes de las partes traseras de vehículos, ya que es lo que las cámaras de tráfico captan. Este conjunto de imágenes es necesario para que se genere el conjunto de imágenes de entrenamiento en el formato esperado por quien realice efectivamente el cálculo del clasificador.

Para conseguir dicho conjunto de muestras, inicialmente es necesario obtener un archivo de texto que contenga un listado de las rutas de las imágenes que contienen vehículos, seguidas de las coordenadas del vehículo dentro de la imagen. La estructura de las coordenadas es la siguiente: [Inicio en abscisas, inicio en ordenadas, longitud en abscisas, longitud en ordenadas]. La estructura del archivo listado esperado será la siguiente:

Nombre de la imagen.extensión	*Nº de objetos presentes en la imagen*	*Coordenadas del primer objeto*	*Coordenadas del segundo objeto*

```
image_0451_t.jpg 3 68 43 56 45 20 34 53 22 2 26 27 20
image_0452_t.jpg 2 57 45 84 58 0 36 28 20
image_0453_t.jpg 1 56 44 84 54
image_0454_t.jpg 1 51 44 86 51
image_0455_t.jpg 1 48 46 93 59
image_0456_t.jpg 1 53 47 81 49
image_0457_t.jpg 1 64 45 64 45
image_0458_t.jpg 1 27 46 91 57
image_0459_t.jpg 1 28 47 91 67
image_0460_t.jpg 2 50 41 61 41 25 38 30 21
image_0461_t.jpg 2 45 42 77 53 8 37 42 29
image_0462_t.jpg 2 32 47 96 63 1 41 42 25
image_0463_t.jpg 1 53 40 71 49
image_0464_t.jpg 1 55 42 67 44
image_0465_t.jpg 1 55 40 68 46
image_0466_t.jpg 1 52 40 61 48
image_0467_t.jpg 1 38 45 81 52
image_0468_t.jpg 1 42 46 78 53
image_0469_t.jpg 1 1 42 44 39
image_0470_t.jpg 1 56 41 56 43
image_0471_t.jpg 1 44 44 73 50
image_0472_t.jpg 1 27 47 93 58
image_0473_t.jpg 1 87 47 61 41
image_0474_t.jpg 1 47 44 81 55
```

Ilustración 14: Ejemplo del archivo listado de imágenes positivas

Para conseguir dicho archivo, se partirá del conjunto de imágenes positivas y se utilizará la función de *OpenCV* "annotations" desde el ejecutor de comandos del sistema, mediante el siguiente comando:

opencv_annotations –images directorio –annotations archivo_listado.txt

Una vez ejecutado dicho comando, se abrirá una ventana por cada imagen presente en el directorio especificado en la cual se deberá encuadrar el vehículo o los vehículos presentes, haciendo uno del ratón del ordenador o bien del touchpad.

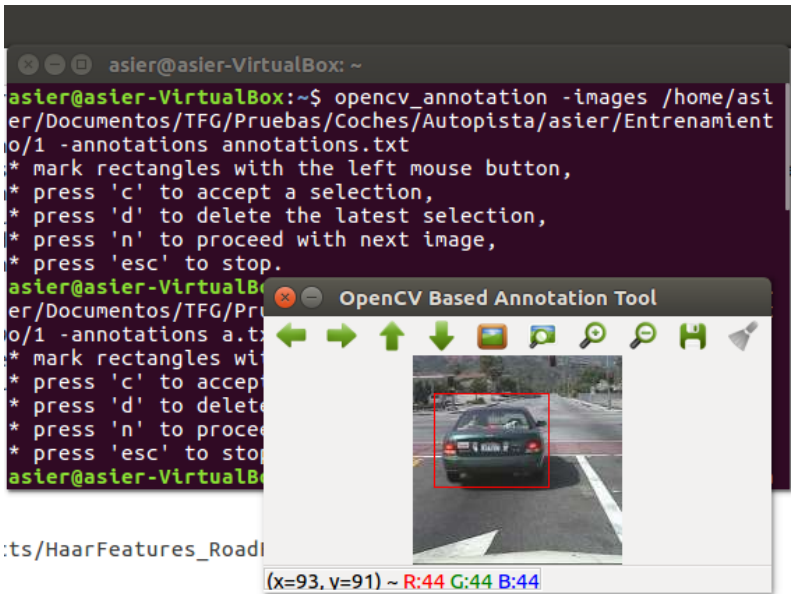

Ilustración 15: Ejemplo de uso de la herramienta annotations de OpenCV

Dado que muchas de las imágenes del conjunto utilizado eran similares, es decir, el vehículo se encontraba en coordenadas prácticamente iguales dentro de cada imagen, se puede sustituir el laborioso procedimiento anterior por un atajo. Dicho atajo consiste en anotar una imagen utilizando el método anterior y, con el resto de imágenes similares, copiar las coordenadas utilizando el siguiente comando:

find carpeta_imagenes_similares –iname '' –exec echo \{\} coordenanas \; > archivo_listado.txt*

De esta manera conseguimos el archivo de texto con el listado de las imágenes y las coordenadas del vehículo dentro de ellas.

2. Conjunto de muestras negativas:

De forma análoga a las muestras positivas, se usará un conjunto de imágenes sin presencia del patrón a buscar (vehículos).

Esta vez no será necesario hacer uso de ninguna herramienta para conseguir el archivo de texto con el listado de las imágenes negativas. Simplemente se ejecutará el siguiente comando:

find carpeta_imagenes_negativas –iname '' > archivo_lista_negativas.txt*

Se obtendrá de esta manera un archivo similar al obtenido para las imágenes positivas, con la diferencia de que en este caso únicamente se listarán los nombres o rutas de las imágenes.

```
neg/image_1068_t.jpg
neg/image_0949_t.jpg
neg/image_0067_t.jpg
neg/image_0695_t.jpg
neg/image_0725_t.jpg
neg/image_0234_t.jpg
neg/image_0354_t.jpg
neg/image_0842_t.jpg
neg/image_0317_t.jpg
neg/image_0343_t.jpg
neg/image_0855_t.jpg
neg/image_1027_t.jpg
neg/image_0023_t.jpg
neg/image_1132_t.jpg
neg/image_0107_t.jpg
neg/image_0982_t.jpg
neg/image_0754_t.jpg
```

Ilustración 16: Ejemplo del archivo listado de imágenes negativas

8.1.2 Creación de muestras

Una vez se dispone de los archivos listado de las imágenes positivas y negativas, además de dichas imágenes, se deberá crear otro archivo, este de formato vectorial y extensión .vec, que transformará la información anterior en un formato con el que pueda trabajar la herramienta que se utilizará más adelante para la creación del clasificador.

Para ello se hará uso de la herramienta *createsamples* de *OpenCV*.

Las imágenes proporcionadas no necesariamente deben ser todas del mismo tamaño, pero se sugiere que sean de dimensiones similares; será *createsamples* quien las escale en base a los parámetros *w* y *h* proporcionados.

createsamples permite además distorsionar las muestras que tenemos para obtener un número mayor de muestras positivas a partir de un grupo inicial de imágenes más reducido. Sin embargo, en este proyecto no se utilizará esta funcionalidad, ya que ofrece peores resultados.

El funcionamiento de esta herramienta es el siguiente: *createsamples* toma la lista de muestras positivas contenidas en el fichero de texto correspondiente, usándose como "background" para comparación las imágenes negativas contenidas en su fichero de texto. El comando ejecutado en el ejecutor de comandos del sistema es el siguiente:

opencv_createsamples –info archivo_listado_positivas.txt –num 1662 –bg archivo_listado_negativas.txt –vec archivo_vectorial_muestras.vec –w 20 –h 20

El parámetro *num* se refiere al número de anotaciones de las que se dispone.

Se ha podido comprobar el correcto funcionamiento de los pasos seguidos hasta ahora utilizando el comando:

opencv_createsamples –vec archivo_vectorial_muestras.vec –w 20 –h 20

Ejecutándolo, se visualizan las muestras recortadas de los vehículos.

8.1.3 Creación del archivo clasificador

Una vez se dispone del archivo vectorial que contiene la información con la que se va a trabajar, se hará uso de la herramienta *traincascade* de *OpenCV* para conseguir el archivo de extensión .xml, es decir, el clasificador.

Como ya se explicó en el anterior apartado de Análisis de Alternativas, la herramienta *traincascade* es muy compleja y tiene muchos parámetros que variar para conseguir un buen resultado.

El comando finalmente utilizado es el siguiente:

opencv_traincascade –data carpeta_destino –vec archivo_vectorial_muestras.vec –bg archivo_listado_negativas.txt –numStages 20 –nsplits 1 –minhitrate 0.995 –maxfalsealarm 0.5 –numPos 1390 –numNeg 3523 –w 20 –h 20 –stageType BOOST –featureType HAAR –bt GAB –weighTrimRate 0.95^C

El "output" que se genera al ejecutar el comando es recursivo para cada etapa del entrenamiento hasta llegar a las 20 indicadas, o bien hasta que alguna alarma salte (maxfalsealarm, minhitrate, etc). La estructura de la ejecución de esta herramienta es la siguiente:

```
asier@asier-VirtualBox:~/Documentos/TFG/Pruebas/Coches/Autopista/asier/Entrenamiento$ opencv_traincascade -data data_80 -vec cars_roar_f.vec -
bg bg_80.txt -numS
tages 20 -nsplits 1 -minhitrate 0.995 -maxfalsealarm 0.5 -numPos 1390 -numNeg 3523 -w 20 -h 20 -stageType BOOST -featureType HAAR -bt GAB -wei
ghtTrimRate 0.95^C
PARAMETERS:
cascadeDirName: data_80
vecFileName: cars_roar_f.vec
bgFileName: bg_80.txt
numPos: 1390
numNeg: 3523
numStages: 20
precalcValBufSize[Mb] : 1024
precalcIdxBufSize[Mb] : 1024
acceptanceRatioBreakValue : -1
stageType: BOOST
featureType: HAAR
sampleWidth: 20
sampleHeight: 20
boostType: GAB
minHitRate: 0.995
maxFalseAlarmRate: 0.5
weightTrimRate: 0.95
maxDepth: 1
maxWeakCount: 100
mode: BASIC
Number of unique features given windowSize [20,20] : 78460

===== TRAINING 0-stage =====
<BEGIN
POS count : consumed   1390 : 1390
NEG count : acceptanceRatio    3523 : 1
Precalculation time: 50
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
```

Ilustración 17: Ejecución de traincascade (1)

```
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
|   2|        1|        1|
+----+---------+---------+
|   3|        1|        1|
+----+---------+---------+
|   4| 0.997122| 0.705649|
+----+---------+---------+
|   5| 0.995683| 0.512064|
+----+---------+---------+
|   6| 0.996403| 0.457281|
+----+---------+---------+
END>
```

Ilustración 18: Ejecución de traincascade (2)

De esta manera se consigue el archivo clasificador que se utilizará para la detección de vehículos en la aplicación que se desea. La estructura del archivo .xml generado es la siguiente:

```
−<cascade>
   <stageType>BOOST</stageType>
   <featureType>HAAR</featureType>
   <height>20</height>
   <width>20</width>
  −<stageParams>
     <boostType>GAB</boostType>
     <minHitRate>9.9500000476837158e-01</minHitRate>
     <maxFalseAlarm>5.0000000000000000e-01</maxFalseAlarm>
     <weightTrimRate>9.4999998807907104e-01</weightTrimRate>
     <maxDepth>1</maxDepth>
     <maxWeakCount>100</maxWeakCount>
   </stageParams>
  −<featureParams>
     <maxCatCount>0</maxCatCount>
     <featSize>1</featSize>
     <mode>BASIC</mode>
   </featureParams>
   <stageNum>19</stageNum>
  −<stages>
     <!-- stage 0 -->
   −<_>
      <maxWeakCount>6</maxWeakCount>
      <stageThreshold>-2.0655322074890137e+00</stageThreshold>
     −<weakClassifiers>
      −<_>
         <internalNodes> 0 -1 144 -9.8064422607421875e-02</internalNodes>
        −<leafValues>
           3.5280282609164715e-03 -9.0605163574218750e-01
```

Ilustración 19: Extracto del archivo clasificador.xml

8.2 Desarrollo de la aplicación

Una vez habiendo creado el archivo clasificador, se dispone de la información necesaria para realizar la detección de vehículos. La siguiente etapa del proyecto corresponde al desarrollo de la aplicación que hará uso de dicho archivo. El esquema del funcionamiento de esta etapa es el siguiente:

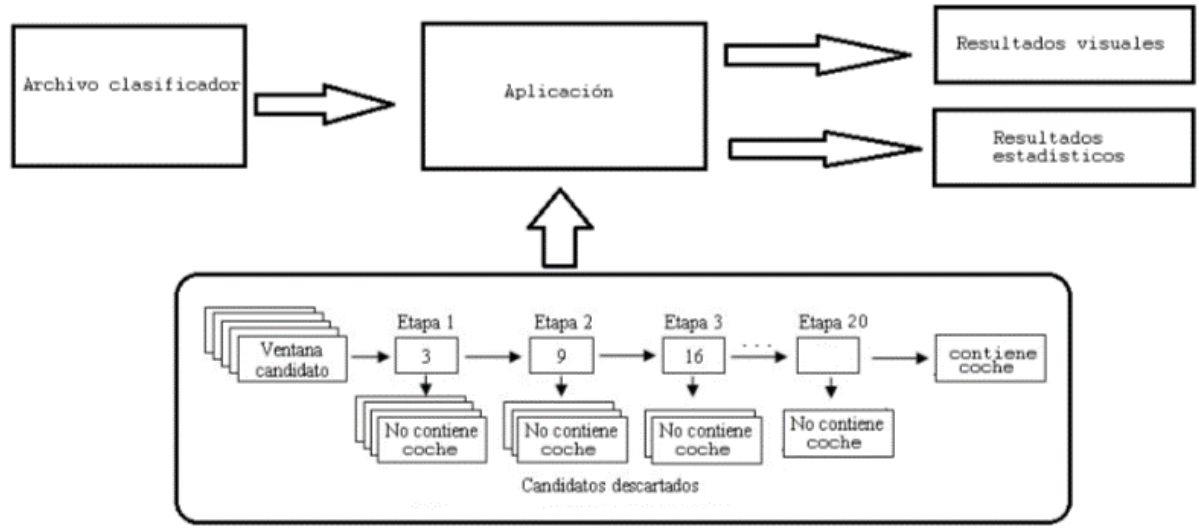

Ilustración 20: Esquema gráfico del desarrollo y funcionalidad de la aplicación

El código de la aplicación se puede encontrar uno de los anexos al final del documento.

La aplicación tomará como entrada el archivo clasificador de extensión .xml, además de un vídeo sobre el que trabajar. Las pruebas se han realizado sobre un vídeo grabado por una cámara de autopista. Las funcionalidades de esta aplicación son las siguientes:

- Por una parte, como todo sistema de detección de objetos, dispone de la funcionalidad visual de encuadrar los vehículos detectados. Esto se consigue mediante la captura consecutiva de los "frames" del vídeo; en cada uno de ellos, se aplicará el clasificador y se encuadrarán los vehículos.
- Por otra parte, también genera algunos estadísticos a partir de la información de las detecciones. Para cada frame, en el ejecutor de comandos aparecerá la siguiente información:

  "Total: *x* coches circulando."

  Además de informar instantáneamente (frame por frame, pero los frames se capturan cada milisegundo) del estado de la carretera en cuanto a número de vehículos circulando, esta aplicación también dará un resultado al final del vídeo. Dicho resultado es el de informar del número medio de coches que circulan por el tramo de carretera capturado por el vídeo. Tiene el siguiente formato (*x.xxx* representa un número decimal, que se redondeará por exceso a *x*):

  "Número medio de coches circulando simultáneamente por este tramo: $x.xxx \rightarrow x$

- Por último, cabe destacar la posibilidad de pausar el funcionamiento del programa y reanudarlo en cualquier momento. Esta funcionalidad se ha implementado con el fin de poder analizar visualmente el estado de la carretera y estudiar los estadísticos en el instante que se desee.

  Para ello, basta con pulsar la tecla "espacio" para pausar y reanudar el vídeo.

  Además, existe la posibilidad de terminar la ejecución del programa antes de que el vídeo acabe. Esto se consigue pulsando la tecla "escape".

Los resultados de la ejecución de la aplicación se podrán observar posteriormente en el apartado pertinente.

En otro de los anexos se adjunta un manual de usuario donde se explica cómo realizar la ejecución del programa.

Metodología seguida y desarrollo del proyecto

1. Diagrama de Gantt/cronograma

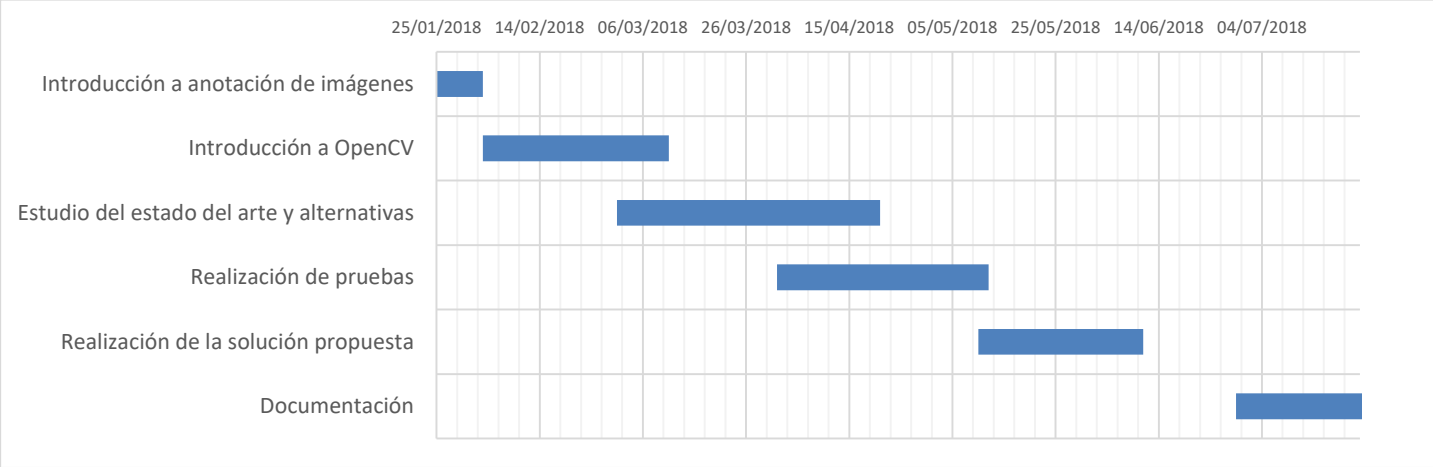

Ilustración 21: Diagrama de Gant

2. Descripción de tareas. Plan de proyecto y planificación

2.1 Introducción a anotación de imágenes

El proyecto se inició con el proceso de documentación sobre la visión artificial, en concreto sobre la anotación de imágenes. Este proceso se realizó entre el 25 de enero y el 2 de febrero.

Ya que no se disponía de conocimientos acerca de este ámbito tecnológico, ni tampoco se conocían fuentes de recursos informativos, se comenzó leyendo una tesis proporcionada por el profesor. Dicha tesis trataba sobre la anotación de imágenes basada en el método de "vecinos más cercanos". Gracias a ello se pudieron adquirir unas nociones básicas de cómo funciona un sistema de este tipo.

2.2 Introducción a *OpenCV*

La siguiente etapa en la realización del proyecto fue documentarse, instalar y curiosear con la librería *OpenCV*. La duración de esta etapa fue desde el 3 de febrero hasta el 10 de marzo.

Por recomendación del profesor, se decidió utilizar la herramienta *OpenCV* para la realización del prototipo. Ya que no se conocía dicha herramienta ni su funcionamiento, se debió buscar información, sobre todo vídeos explicativos, para conocer en profundidad el funcionamiento y las aplicaciones de la librería.

Se estudiaron las diferentes funciones de la librería, su funcionamiento y cómo aplicarlas para desarrollar el proyecto (cómo utilizarlas en la programación de la aplicación).

2.3 Estudio del estado del arte y alternativas

Cuando se estimó que los conocimientos acerca del funcionamiento de *OpenCV* eran suficientes, se procedió al análisis del estado del arte. Este proceso comenzó el 1 de marzo y acabó el 20 de abril.

Dentro de este proceso se estudiaron las posibles maneras de desarrollar el sistema de detección de vehículos. Como se ha podido ver en los anteriores apartados Estado del arte y Análisis de alternativas, este proceso fue muy largo y laborioso.

Sobre todo, se invirtió mucho tiempo en el estudio y la decisión de los parámetros de la función *traincascade*.

Gran parte de este proceso se solapó con la realización de pruebas, ya que se trató de un proceso de aprendizaje de "prueba y error".

Además, se buscaron diferentes proyectos similares, de tal manera que se pudieras adquirir unas nociones básicas de cómo se realizaba la programación de una aplicación que hiciera uso de un clasificador para la detección de objetos.

2.4 Realización de pruebas

La realización de pruebas comprendió desde el 1 de abril hasta el 10 de mayo.

A la vez que se estudiaba la influencia de los parámetros de las funciones de *OpenCV* en el proceso de entrenamiento del clasificador, se realizaron diversas pruebas para comprobar los resultados.

Los objetos elegidos para realizar dichas pruebas fueron, en orden cronológico: balones (sin éxito), peatones (sin éxito), aviones (buenos resultados), motos (buenos resultados), coches (buenos resultados).

Estas pruebas se realizaron utilizando una imagen como entrada, en vez de un vídeo.

En esta etapa tuvieron lugar dos procedimientos: la obtención de conjuntos de imágenes de entrenamiento para cada uno de los objetos de prueba, lo cual requirió mucho tiempo, y el desarrollo (programación) de una aplicación que permitiera la prueba de los clasificadores en una imagen de entrada.

2.5 Realización de la solución propuesta

Tras la documentación y la realización de pruebas, una vez sabiendo cómo desarrollar un clasificador efectivo, se procedió a la realización del sistema final. Esta etapa duró desde el 10 de mayo hasta el 10 de junio.

Esta fase comprendió el desarrollo efectivo del clasificador y la programación de la aplicación que hace uso de éste. Se desarrollaron actualizaciones de mejora tanto del clasificador como de la aplicación, dando finalmente lugar al sistema definitivo.

También se realizaron diversas pruebas con vídeos en diferentes condiciones para testar el rendimiento del sistema.

2.6 Documentación

Por último, se realizó el proceso de documentación y desarrollo del documento de texto que recoge todo el proyecto, desde el 29 de junio hasta el 23 de julio.

Los apartaos de Introducción, Contexto, Objetivos y Alcance del Trabajo y Beneficios del Trabajo fueron documentados en estos plazos, mientras que la información del resto de los apartados tiene su origen en los procesos prácticos previamente descritos.

3. Descripción de los resultados

A continuación, se ilustrarán los resultados del funcionamiento del sistema desarrollado.

Dado que el sistema funciona con un vídeo y los resultados se muestran sobre éste en tiempo real, se incluirán algunas capturas en diferentes momentos de la ejecución del vídeo para examinar los resultados.

Se podrá observar tanto la detección de los vehículos sobre el vídeo, como las estadísticas proporcionadas por la aplicación.

Los resultados se muestran sobre un vídeo de entrada captado por las cámaras de tráfico de una autopista estadounidense. Se eligió este vídeo ya que fue el utilizado para la realización de las pruebas de varios sistemas como éste en algunos proyectos que se tomaron como ejemplo.

Ilustración 22: Resultado (1)

Ilustración 23: Resutado (2)

Ilustración 24: Resultado (3)

Ilustración 25: Resultado (4)

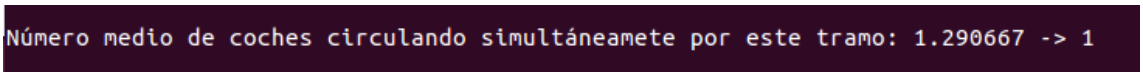

Ilustración 26: Resultado estadístico final

Como se puede observar en los resultados, la efectividad del sistema no es del 100%. Hay algunos vehículos que no son detectados, o que son detectados en algunas posiciones y en otras no; es decir, que la detección no acompaña en todo momento al vehículo.

También se puede observar que los vehículos que no son coches, como los camiones, no son detectados. Esto se debe al entrenamiento del clasificador, para el cual se utilizaron imágenes de coches únicamente.

Por otro lado, vemos cómo los resultados estadísticos simultáneos reflejan a la perfección los resultados visuales en cada instante.

Por último, se puede observar el resultado estadístico final, que nos indica la afluencia que tiene ese tramo de la autopista.

En general, se pueden considerar unos buenos resultados, ya que para el desarrollo de un sistema de detección perfecto se necesitaría mucho más tiempo, unos recursos hardware mucho más potentes y un conjunto de imágenes mucho más amplio y adecuado.

Análisis de Costes

Para realizar el análisis del coste de realización, implementación y puesta en funcionamiento del sistema desarrollado en este proyecto, se tomará como punto de partida el sistema sin desarrollar. Además, se seleccionará el sistema de almacenaje de los vídeos en servidores en la nube, en vez de en memorias físicas.

En primer lugar, se analizará el personal necesario. Se deberá tener en cuenta el salario del programador en relación al número de horas que se necesiten para desarrollar el sistema de detección. También se deberá contratar personal para el análisis de los resultados.

Tabla 5: Salarios

Trabajo	**Nº de trabajadores**	**Nº de horas**	**Salario (€/hora)**	**Total (€)**
Programador	1	200	20 €	4000 €
Analista	2	500	12 €	6000 €
Director de proyecto	1	200	25 €	5000 €
Total				15000 €

Teniendo en cuenta que la Seguridad Social es el 31% del salario de los trabajadores, al anterior total habría que sumarle:

$$15000 \cdot \frac{31}{100} = 4650$$

Por lo tanto, el coste total de los salarios de los trabajadores sería de:

$$15000 + 4650 = 19650 \text{ €}$$

Por otra parte, hay que tener en cuenta los gastos materiales asociados al proyecto:

Tabla 6: Gastos materiales

Concepto	Cantidad	Importe (€)	Total (€)
Ordenadores	2	3000 €	6000 €
Material de oficina: Mesas, sillas, etc.	15	(Medio): 150 €	2250 €
Cámaras de tráfico	10	2000 €	20000 €
Total			28250 €

Por último, se añadirán otros costes relacionados con los alquileres, servicios, etc. Para calcular el total, se estudiará el uso del sistema durante un año:

Tabla 7: Gastos en alquileres, servicios, etc.

Concepto	Cantidad/mes	Importe (€/mes)	Total (€)
Alquiler de servidores	1	100 €	1200 €
Arrendamientos	1	1000 €	12000 €
Servicio de mantenimiento y reparación	1	400 €	4800 €
Servicios profesionales (asesoramiento, declaraciones de la renta, etc)	4	250 €	12000 €
Suministros (agua, gas, luz, electricidad)	4	(Media): 30 €	1440 €
Total			31440 €

Por lo tanto, el importe total acumulado para el desarrollo, implementación y uso durante un año del sistema que describe este proyecto sería el siguiente:

$$19650 + 28250 + 31440 = 79340 \text{ €}$$

Dado que es una cantidad muy alta, se pediría un préstamo bancario de unos 90000 €. Teniendo en cuenta el porcentaje de comisión que cobrará la entidad bancaria (alrededor de un 3% para jóvenes emprendedores), el coste añadido sería el siguiente:

$$90000 \cdot \frac{3}{100} = 2700 \text{ €}$$

Finalmente, el coste total sería de:

$$79340 + 2700 = 82040 \text{ €}$$

Conclusiones

Este proyecto demuestra la utilidad de la visión artificial en el ámbito de la seguridad vial. También demuestra que es posible la realización de un sistema de detección de objetos con unas condiciones iniciales y recursos no demasiado exigentes.

Además, la utilización de la librería *OpenCV* hace del laborioso proceso de entrenamiento del clasificador algo más comprensible y fácil. Asimismo, el análisis de los parámetros realizado posibilita la reducción del tiempo de procesamiento y la adaptación a los resultados deseados del comportamiento y el entrenamiento del clasificador.

Con el prototipo desarrollado se consigue el control visual de los vehículos circulantes, así como la información necesaria para analizar el tráfico por el tramo de la autopista que capta la cámara. Esto permite mejorar el control sobre las carreteras y puede ayudar a dar avisos para evitar retenciones o accidentes.

En definitiva, este proyecto cumple con sus objetivos ayudando a la seguridad vial y mostrando una manera asequible de crear un sistema de detección de objetos. A partir de este proyecto se puede desarrollar un sistema de seguridad más complejo, o simplemente utilizarse para los fines propuestos, ya que es perfectamente válido.

Los resultados visuales obtenidos no son 100% efectivos, hay algunos falsos positivos y algunas faltas de detección, pero son aceptables para el fin propuesto. Además, los resultados estadísticos representan a la perfección lo que los resultados visuales muestran, por lo que no hay pérdida de información.

En cuanto a la implementación de este sistema, se requerirían algunos recursos como la gestión de la información y el personal dedicado al análisis de ésta, pero estos aspectos quedan fuera de este trabajo. Simplemente se indica la viabilidad de una posible implementación en el sistema de seguridad vial actual.

El carácter económico de una posible implantación del sistema en una red de cámaras existente (con los recursos ya adquiridos, es decir, sin coste adicional) lo hace aún más atractivo.

De cara al futuro, dado que la tecnología avanza a pasos desmesurados y cada vez se introduce más en nuestra vida diaria, se han podido ver algunas implementaciones de este sistema en lo que se denomina “ciudad inteligente”. El prototipo desarrollado no es una tecnología obsoleta ni mucho menos. La visión artificial está en auge.

Referencias

[1] [En línea]. Available: https://visartblog.wordpress.com/2013/05/02/resumen-historia-de-la-vision-artificial/.

[2] [En línea]. Available: http://faicoblog.blogspot.com/2016/02/de-donde-viene-la-vision-artificial.html.

[3] [En línea]. Available: https://es.wikipedia.org/wiki/Visi%C3%B3n_artificial.

[4] C. Morales y I. Nava, "Detector Automático de Franqueamiento de Señal de Maniobra para el Sistema de Transporte Colectivo Metro mediante Algoritmos de Visión Artificial", 2015.

[5] [En línea]. Available: https://blog.infaimon.com/camara-vision-artificial-tecnologias-modelos/.

[6] [En línea]. Available: http://revista.dgt.es/es/investigacion/innovacion/2017/1226camaras-inteligentes-en-pasos-a-nivel.shtml#.Wz3YeNIzbIU.

[7] [En línea]. Available: http://www.sice.com/actualidad/sice-mantendra-los-sistemas-de-vision-artificial-y-de-control-de-infracciones-de-trafico-de-la-ciudad-de-barcelona.

[8] [En línea]. Available: http://www.lasexta.com/motor/noticias/como-funcionan-las-camaras-que-detectan-el-uso-de-cinturon-y-cuando-empezaran-a-multar_201709125a9606570cf2586cf842e081.html.

[9] [En línea]. Available: https://www.heraldo.es/noticias/aragon/2017/03/21/cuatro-decadas-seguridad-vial-mismos-accidentes-pero-cientos-muertos-menos-1165379-300.html.

[10] [En línea]. Available: https://www.tecnocarreteras.es/2015/04/23/una-camara-con-vision-de-360o-y-un-sistema-de-vision-artificial-para-gestionar-integralmente-todo-lo-que-ocurre-en-una-interseccion/.

[11] [En línea]. Available: https://www.tecnocarreteras.es/2012/06/07/ciudad-2020-disenando-la-ciudad-inteligente-del-futuro/.

[12] [En línea]. Available: https://www.merca2.es/dgt-datos-accidentes/.

[13] [En línea]. Available: https://www.sapo.pt/noticias/motores/bosch-apresenta-veiculo-autonomo-inteligente_58d32089a6841bff25d89b0f.

[14] [En línea]. Available: http://www.radio.cz/es/rubrica/notas/mas-camaras-de-trafico-para-la-seguridad-vial.

[15] [En línea]. Available: https://www.tecnocarreteras.es/2016/05/27/beneficios-los-radares-velocidad-las-camaras-semaforos-reducir-numero-accidentes-la-ciudad/.

[16] [En línea]. Available: http://www.circulaseguro.com/aspectos-diseno-una-ciudad-pueden-reducir-los-accidentes-trafico/.

[17] [En línea]. Available: https://www.learnopencv.com/image-recognition-and-object-detection-part1/.

[18] A. I. Fava Fernandez, "Big data y Deep Learning para la detección de objetos en imágenes", 2017.

[19] P. Viola y M. Jones, "Rapid object detection using a boosted cascade of simple features", 2001.

[20] [En línea]. Available: https://en.wikipedia.org/wiki/Haar-like_feature.

[21] S. Figueroa Sánchez, 2015. [En línea]. Available: http://200.10.19.208/REPOS/COMPUTERVISION/Semana07.html.

[22] M. Chang, "Object Detection, Method of Viola and Jones", 2014.

[23] F. Maniscalo Martín, "Identificación visual en ambientes subacuáticos", 2011.

[24] [En línea]. Available: https://www.researchgate.net/figure/Visualization-of-calculating-a-LBP-pattern-value-for-a-given-pixel-location-25_fig1_262346344.

[25] D. He y L. Wang, "Texture Unit, Texture Spectrum, And Texture Analysis", 1990.

[26] F. Hernandez, 2015. [En línea]. Available: https://www.researchgate.net/publication/267820077_El_Concepto_de_Distancia_y_su_Aplicacion_en_Estadistica_Multivariada.

[27] [En línea]. Available: https://es.wikipedia.org/wiki/An%C3%A1lisis_de_componentes_principales.

[28] [En línea]. Available: https://es.wikipedia.org/wiki/M%C3%A1quinas_de_vectores_de_soporte.

[29] A. Castro y J. Luis, "Máquinas de Vectores de Soporte (SVM)", 2014.

[30] [En línea]. Available: https://es.wikipedia.org/wiki/OpenCV.

[31] [En línea]. Available: https://stackoverflow.com/questions/21215896/the-difference-between-simplecv-and-opencv.

[32] [En línea]. Available: https://alternativeto.net/software/accord-net-framework/.

[33] A. Min, "Gedit: a powerful, underrated text editor for everybody", 2008.

[34] j. Cruz, E. Shiguemori y L. Guimaraes, "A comparison of Haar-like, LBP and HOG approaches to concrete and asphalt runway detection in high resolution imagery", 2015.

[35] L. Xiang-Yu y L. Zhen-Xian, "Face recognition based on HOG and fast PCA Algorithm", 2017.

[36] A. Kummar Annamraju y A. Deep Singh, "Analysis and Optimization of Parameters used in Training a Cascade Classifier", 2015.

[37] [En línea]. Available: https://www.learnopencv.com/install-opencv3-on-ubuntu/.

Anexo 1: Código

En este anexo se adjunta el código de la aplicación que hace uso del clasificador desarrollado y de la que se obtienen los resultados tanto visuales como estadísticos.

No se añadirá el código de las funciones de la librería *OpenCV* ya que se pueden encontrar en su página web oficial y no han sido modificadas. Además, la explicación de su funcionamiento y sus parámetros ya ha sido comentada con anterioridad.

Aplicación de seguridad vial

```
1. #include "opencv2/objdetect.hpp"
2. #include "opencv2/videoio.hpp"
3. #include "opencv2/highgui.hpp"
4. #include "opencv2/imgproc.hpp"
5.
6. #include <iostream>
7. #include <stdio.h>
8. #include <math.h>
9.
10.      using namespace std;
11.      using namespace cv;
12.
13.      /** Cabeceros de las funciones */
14.      void detectaYMuestra( Mat frame );
15.
16.      /** Variables globales */
17.      String nombre_clasificador;
18.      String nombre_video;
19.      CascadeClassifier cascada;
20.      String ventana resultado = "Autopista - Deteccion de coches";
21.      const int tecla_espacio = 32;    //Tecla espacio para pausar el vídeo
22.      const int tecla_escape = 27;     //Tecla escape para salir del vídeo
23.      float contador_coches = 0;       //Variable para contar los coches que
   hay en el frame
24.      float contador_frames = 0;       //Variable para contar el número de
   frames
25.
26.
27.      /** Función main */
28.      int main( int argc, const char** argv )
29.      {
30.          nombre_clasificador = "/home/asier/Documentos/TFG/Pruebas/Coches/
   Autopista/asier/cascades/2/cascade.xml";
31.          nombre_video = "/home/asier/Documentos/TFG/Pruebas/Coches/Autopis
   ta/asier/videos/video2.avi";
32.          VideoCapture video;
33.          Mat frame;
34.
35.          //-- 1. Cargar el clasificador
36.          if( !cascada.load( nombre_clasificador ) ){ printf("--(!)Error
   cargando el clasificador\n"); return -1; };
37.
```

```
38.           //-- 2. Leer el stream del video
39.           video.open(nombre_video);
40.           if ( ! video.isOpened() ) { printf("--(!)Error abriendo el
   video\n"); return -1; };
41.
42.           int tecla = 0;
43.           while ( video.read(frame) )
44.           {
45.               contador_frames ++;
46.               if( frame.empty() )
47.               {
48.                   printf(" --(!) No hay frame -- Break!");
49.                   break;
50.               }
51.
52.               //-- 3. Aplicar el clasificador al frame
53.               detectaYMuestra( frame );
54.
55.               tecla = cvWaitKey(33);  //Para poder utilizar escape y
   espacio
56.
57.               if(tecla == tecla_espacio)
58.               tecla = cvWaitKey(0);   //Parar el vídeo
59.
60.               if(tecla == tecla_escape)
61.               break;                  //Salir
62.
63.           }
64.
65.           //Liberamos recursos
66.           video.release();
67.           destroyAllWindows();
68.
69.           //Resultados
70.           float media_coches = contador_coches/contador_frames;
71.           printf("\n\nNúmero medio de coches circulando simultáneamete por
   este tramo: %f -> %d\n\n", media_coches, (int)round(media_coches));
72.
73.           return 0;
74.       }
75.
76.       /** Función detectaYMuestra */
77.       void detectaYMuestra( Mat frame )
78.       {
79.           std::vector<Rect> coches;   //vector contenedor de los coches que
   hay en el frame
80.           Mat frame_gray;
81.
82.           cvtColor( frame, frame_gray, COLOR_BGR2GRAY );      //Cambiamos a
   escala de grises el frame
83.           equalizeHist( frame_gray, frame_gray );     /*Normaliza el brillo
   y sube el contraste de la imagen:
84.                                                       1º Calcula el
   histograma del primer input
85.                                                       2º Lo normaliza para
   que la suma de los contenedores del histograma de 255
86.                                                       3º Hace la integral
   del histograma
```

```
87.                                                     4° Transforma la
   imagen usando la integral como una tabla de búsqueda.*/
88.
89.          //-- Detección de coches
90.          cascada.detectMultiScale( frame_gray, coches, 1.1, 2, 0|CASCADE_S
   CALE_IMAGE, Size(20, 20) );
91.
92.          std::cout << "Total: " << coches.size() << " coches
   circulando." << std::endl;      //Número de vehículos capturados en el
   frame
93.          contador_coches = contador_coches + coches.size();
94.
95.          for ( size_t i = 0; i < coches.size(); i++ )
96.          {
97.              rectangle(frame, cvPoint(cvRound(coches[i].x), cvRound(coches
   [i].y)),
98.                                cvPoint(cvRound(coches[i].x + coches[i].width-
   1), cvRound(coches[i].y + coches[i].height-1)),
99.                                Scalar( 0, 0, 255 ), 2, 8, 0);
100.
101.             Mat carsROI = frame_gray( coches[i] );
102.         }
103.
104.         //-- Mostrar el resultado
105.         imshow( ventana resultado, frame );
106.     }
```

Anexo 2: Manuales de usuario y de instalación

Instalación de *OpenCV*

Se puede descargar la librería desde su página oficial: https://opencv.org/releases.html.

Una vez descargada se debe proceder a la instalación de la misma. Se explicará el proceso de instalación para la plataforma Ubuntu.

Dentro de la descarga se encuentra un documento de texto llamado "INSTALL" en el que se pueden encontrar unas instrucciones a seguir para la instalación.

Sin embargo, para la instalación de *OpenCV* en este proyecto se han seguido los pasos de un tutorial [37]. Se trata de la ejecución de los siguientes comandos:

```
1. sudo apt-get update
2. sudo apt-get upgrade
3.
4. Remove any previous installations of x264</h3>
5. sudo apt-get remove x264 libx264-dev
6.
7. We will Install dependencies now
8.
9. sudo apt-get install build-essential checkinstall cmake pkg-config yasm
10.      sudo apt-get install git gfortran
11.      sudo apt-get install libjpeg8-dev libjasper-dev libpng12-dev
12.
13.      # If you are using Ubuntu 14.04
14.      sudo apt-get install libtiff4-dev
15.      # If you are using Ubuntu 16.04
16.      sudo apt-get install libtiff5-dev
17.
18.      sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev
   libdc1394-22-dev
19.      sudo apt-get install libxine2-dev libv4l-dev
20.      sudo apt-get install libgstreamer0.10-dev libgstreamer-plugins-
   base0.10-dev
21.      sudo apt-get install qt5-default libgtk2.0-dev libtbb-dev
22.      sudo apt-get install libatlas-base-dev
23.      sudo apt-get install libfaac-dev libmp3lame-dev libtheora-dev
24.      sudo apt-get install libvorbis-dev libxvidcore-dev
25.      sudo apt-get install libopencore-amrnb-dev libopencore-amrwb-dev
26.      sudo apt-get install x264 v4l-utils
27.
28.      sudo apt-get install python-dev python-pip python3-dev python3-pip
29.      sudo -H pip2 install -U pip numpy
30.      sudo -H pip3 install -U pip numpy
31.
32.      # Install virtual environment
33.      sudo pip2 install virtualenv virtualenvwrapper
34.      sudo pip3 install virtualenv virtualenvwrapper
35.      echo "# Virtual Environment Wrapper"  >> ~/.bashrc
36.      echo "source /usr/local/bin/virtualenvwrapper.sh" >> ~/.bashrc
37.      source ~/.bashrc
38.
```

```
39.       ############ For Python 2 ############
40.       # create virtual environment
41.       mkvirtualenv facecourse-py2 -p python2
42.       workon facecourse-py2
43.
44.       # now install python libraries within this virtual environment
45.       pip install numpy scipy matplotlib scikit-image scikit-learn ipython
46.
47.       # quit virtual environment
48.       deactivate
49.       #####################################
50.
51.       ############ For Python 3 ############
52.       # create virtual environment
53.       mkvirtualenv facecourse-py3 -p python3
54.       workon facecourse-py3
55.
56.       # now install python libraries within this virtual environment
57.       pip install numpy scipy matplotlib scikit-image scikit-learn ipython
58.
59.       # quit virtual environment
60.       deactivate
61.       #####################################
62.
63.       git clone https://github.com/opencv/opencv.git
64.       cd opencv
65.       git checkout 3.3.1
66.       cd ..
67.
68.       git clone https://github.com/opencv/opencv_contrib.git
69.       cd opencv_contrib
70.       git checkout 3.3.1
71.       cd ..
72.
73.       cd opencv
74.       mkdir build
75.       cd build
76.
77.       cmake -D CMAKE_BUILD_TYPE=RELEASE \
78.             -D CMAKE_INSTALL_PREFIX=/usr/local \
79.             -D INSTALL_C_EXAMPLES=ON \
80.             -D INSTALL_PYTHON_EXAMPLES=ON \
81.             -D WITH_TBB=ON \
82.             -D WITH_V4L=ON \
83.             -D WITH_QT=ON \
84.             -D WITH_OPENGL=ON \
85.             -D OPENCV_EXTRA_MODULES_PATH=../../opencv_contrib/modules \
86.             -D BUILD_EXAMPLES=ON ..
87.
88.       # find out number of CPU cores in your machine
89.       nproc
90.       # substitute 4 by output of nproc
91.       make -j4
92.       sudo make install
93.       sudo sh -c 'echo "/usr/local/lib" >> /etc/ld.so.conf.d/opencv.conf'
94.       sudo ldconfig
95.
```

Manual de usuario de la aplicación

A continuación, se explicará la utilización del sistema desarrollado:

Lo primero que se necesita hacer es compilar el código. Para ello se ejecutará el siguiente comando en el ejecutor de comandos del sistema:

g++ -std=c++11 codigo.cpp 'pkg-config –libs –cflags opencv' –o codigo

De esta manera se crea un ejecutable de nombre *codigo*. Para ejecutar la aplicación simplemente se deberá ejecutar el ejecutable *codigo*:

./codigo

El programa se ejecutará y se podrán ver los resultados. Si se desea pausar o salir de la ejecución del programa, basta con pulsar las teclas ESPACIO (pausa) o ESCAPE (salir).