UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO FIN DE GRADO

IMPLEMENTACIÓN DE CLASIFICACIÓN DE OBJETOS MEDIANTE VISIÓN POR COMPUTADOR

Autor: Alejandro Fernández González

Tutor: Aurelio Ponz Vila

RESUMEN

A pesar de los grandes avances tecnológicos en seguridad y confort en el sector de la automoción, actualmente, todavía son muchas las vidas que se pierden a causa de los accidentes de tráfico. Estos avances no se centran únicamente en la mejora o eficiencia de los elementos de seguridad del vehículo para, así, lograr la máxima reducción del daño que pudiera ocasionar, sino que también se centran en la anticipación de estos accidentes.

En el presente proyecto se busca esto último, tener la capacidad de poder anticiparse a estos accidentes, más concretamente a los accidentes en los que estén implicados como victimas peatones o ciclistas. El objetivo del proyecto es diseñar un algoritmo basado en visión computacional que sea capaz de detectar, de manera rápida y precisa, a un peatón o ciclista que se encuentre en el ángulo de visión de la cámara del vehículo, de tal forma que el conductor tenga constancia en todo momento de que se encuentra delante de él.

Para la detección se ha utilizado una técnica que se basa en el Histograma de Gradientes Orientados (HOG). Se trata de una técnica que gracias a su invariancia a cambios lumínicos o posturas ofrece unos resultados robustos. Para poder ofrecer el nivel máximo de precisión y un menor tiempo de ejecución se necesita realizar un estudio de los distintos parámetros del sistema. Para el desarrollo de este algoritmo se ha recurrido a las librerías OpenCV, que ofrecen una gran comodidad y versatilidad en el procesamiento de imágenes y visión artificial.

ABSTRACT

Despite great technology breakthroughs in security and comfort in the automotive sector, there are still many people who lose their lives in car accidents. Not only those innovations are focused on improvements and efficiency of vehicles' safety to get the least damage, but also on crash-anticipation systems.

The latter is what this project is about, more specifically about situations where victims are pedestrians or cyclists. Its goal consists of designing an algorithm, based on computational vision, with the capacity to quickly and accurately detect pedestrians or cyclists in front of the vehicle, helping the driver to always be aware of what is in front.

For the detection, a technique based on Histogram Oriented Gradients (HOG) has been used. It gives robust results thanks to the invariance to lighting or position changes. In order to provide the highest level of accuracy and the lowest run time a study of the parameters of the system is needed. OpenCV libraries have been employed for the development of this algorithm, great simplicity and versatility in image processing and artificial vision being its benefits.

ÍNDICE

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

Agradecimientos

En primer lugar, agradezco a Aurelio Ponz Vila, el tutor de este proyecto, la paciencia que ha tenido ante mis continuas preguntas y por saber desatascar el trabajo en sus malos momentos. A mi familia, sin la cual no hubiera llegado hasta aquí. Y a todos los componentes del Laboratorio de Sistemas Inteligentes que tanto me han ayudado en este proyecto.

Acrónimos

- **OpenCV:** Open source Computer Vision
- **GCC:** GNU Compiler Collection
- **QML:** QT Meta Lenguage
- **MinGW:** Minimalist GNU for Windows
- **MSVC:** Microsoft Visual C++
- **SVM:** Support Vector Machine (Máquina de soporte vectorial)
- **ADAS:** Advanced Driver Assistance Systems
- **HOG:** Histogram of Oriented Gradients (Histograma de gradientes orientados)

1. INTRODUCCIÓN

A medida que los años avanzan los productos electrónicos están teniendo cada vez una mayor importancia en nuestras vidas. Estos productos consiguen simplificarnos las labores cotidianas de modo que hemos llegado a un nivel de dependencia tal que a día de hoy es imposible concebir cualquier ejercicio cotidiano sin la ayuda de estos.

El uso de estos productos se ha ido extendiendo al ámbito del automovilismo generando sistemas de ayuda a la conducción. La inclusión de estos sistemas se debe no solo a un tema de simple comodidad para el conductor y sus ocupantes, sino por ayuda a la reducción de accidentes que se producen a diario por el uso de automóviles. El nivel de desarrollo está tan adelantado que actualmente llevan a cabo labores que para la mayoría de la población eran impensables hace solo unos pocos años, como por ejemplo la detección de peatones o la conducción autónoma de los coches.

Los sistemas de visión han pasado de verse como algo futurista a una realidad en los últimos años, gracias a librerías gratuitas que han facilitado el acceso a todo el mundo como el caso de las OpenCV.

En este proyecto se ha decidido usar estas librerías para que el programa sea capaz de distinguir entre peatones o gente paseando en bicicleta mediante la visión por computador.

La primera parte del proyecto se basará en el entrenamiento del sistema para que sea capaz distinguir a gente en bicicleta o peatones, y la segunda será el reconocimiento de ciclistas y peatones y la optimización del algoritmo de detección para que su tiempo de detección sea mínimo. Todo esto se hará con la ayuda de las funciones ya implementadas de OpenCV.

2. PROBLEMÁTICA ACTUAL

A día de hoy, la mayoría de los accidentes en carretera se producen por errores del conductor. Los Sistemas Avanzados de Ayuda a la Conducción (conocidos como ADAS) buscan minimizar al máximo el factor humano en estos accidentes, tratando de detectar y anticiparse a situaciones que podrían comprometer la integridad del vehículo y ocupantes. En la siguiente ilustración se muestra la evolución de los fallecidos en carreteras españolas mediante accidentes de tráfico.

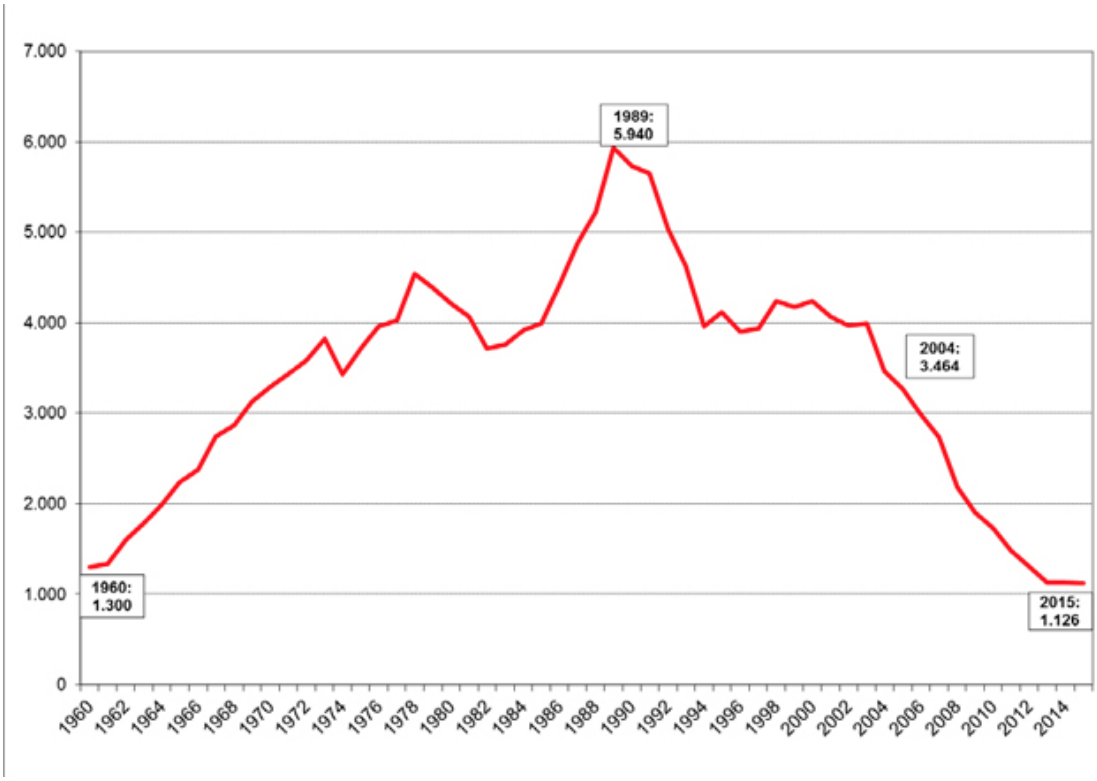

ILUSTRACIÓN 1: EVOLUCIÓN DE FALLECIDOS EN ACCIDENTES DE COCHE EN ESPAÑA DESDE 1960 A 2015 [1]

Se observa en la Ilustración 1, cuyos datos han sido obtenidos de la web de la DGT, que el número de víctimas entre 2004 y 2015 ha disminuido en torno a un 60%. En el año 2015 se ha producido un descenso [2] del 1% en el número de fallecidos y un 2% en heridos hospitalizados y un aumento del 3% en accidentes mortales. Estas cifras de fallecidos representa un mínimo histórico desde 1960, primer año en el que se tienen estadísticas.

Algunos factores que han podido influir en el descenso de accidentes de tráfico son [3] la mejora del estado de la carretera, mayor número de controles policiales, mayores medidas de seguridad en vehículos y campaña de concienciación del uso de sistemas de seguridad entre otros.

4. EVOLUCIÓN DE LA LETALIDAD [1] EN LOS ACCIDENTES DE TRÁFICO CON VÍCTIMAS. SERIE 1993 A 2012.

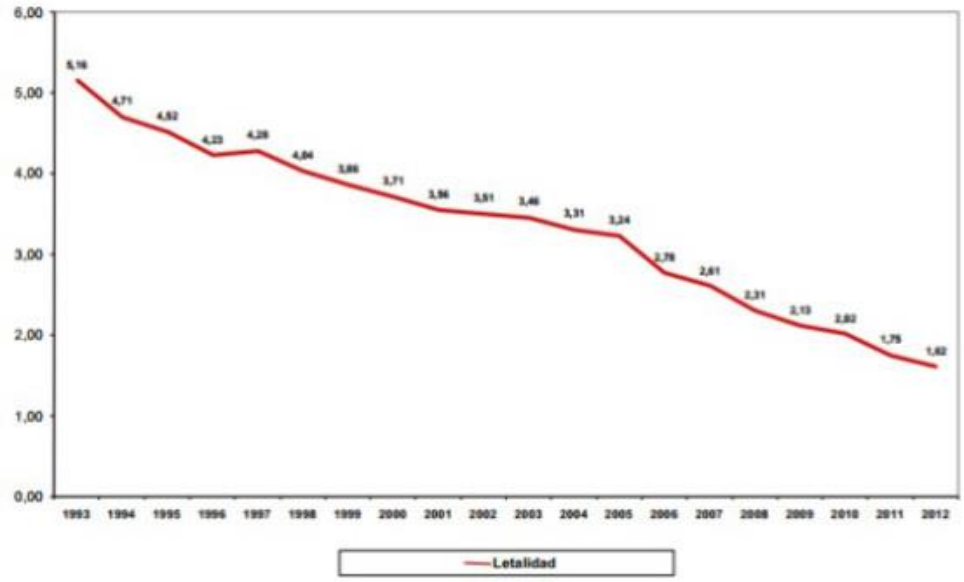

ILUSTRACIÓN 2: EVOLUCIÓN DE LETALIDAD EN ACCIDENTES DE TRÁFICO EN ESPAÑA DESDE 1993 A 2012 [4]

Como vemos en la Ilustración 2 perteneciente al año 2012 que la letalidad de los accidentes de tráfico, siguiendo una línea descendente, cada vez es menor.

Según la Dirección General de Tráfico [5] analizando el número de accidentes según el tipo de vía, podemos decir que es en las vías urbanas donde se producen la mayor parte de los accidentes. Sin embargo, el mayor número de fallecidos se produce en las carreteras convencionales.

Dos de cada tres muertes en un accidente de tráfico se producen en carreteras convencionales, en las que el índice de mortalidad es el doble que en autopistas y autovías. El principal motivo es debido a que se tratan de vías de doble sentido, aumentando así la posibilidad de que se produzcan colisiones con otros vehículos como salida de vía o incluso atropellos a peatones y ciclistas.

ILUSTRACIÓN 3: Nº DE VÍCTIMAS MORTALES POR TIPO DE VÍA Y TIPO DE ACCIDENTE DEL AÑO 2013 EN ESPAÑA [6]

En la Ilustración 3 se muestran los datos sobre los accidentes de tráfico (los datos han sido sacados de la web del Ministerio del Interior), donde se puede ver claramente la mortandad que tiene las carreteras convencionales respecto a las vías de gran capacidad.

Si atendemos, por otra parte, a los atropellos de ciclistas vemos que lejos de reducirse el número de víctimas mortales aumenta respecto años anteriores.

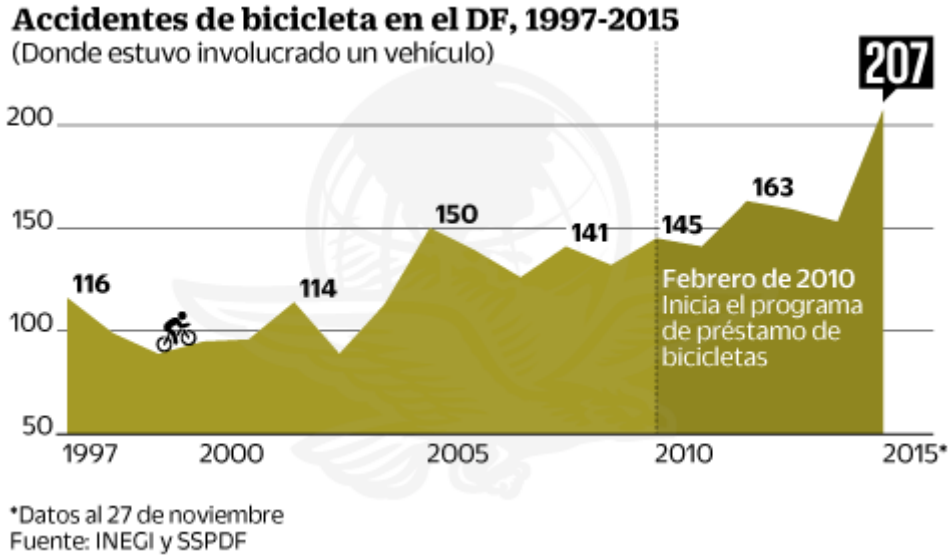

ILUSTRACIÓN 4: EVOLUCIÓN DE ACCIDENTES DE BICICLETAS CON UN VEHÍCULO INVOLUCRADO DESDE 1997 EN ESPAÑA [7]

La causa de estos accidentes [8] corresponden, con un 48%, a impactos fronto-laterales en los que la bicicleta se sitúa en el punto ciego del vehículo. Con un 12 % a impactos laterales en los que el conductor no frena a tiempo cuando el ciclista se cruza en su trayectoria de manera inesperada. Y con un porcentaje menor corresponden los accidentes en los que el conductor no deja la suficiente distancia de seguridad.

3. ESTADO DEL ARTE

En la actualidad están siendo desarrollados sistemas de seguridad muy novedosos, que ayudan tanto a hacer del vehículo un lugar más seguro como generar una conducción más cómoda para su conductor. Estos sistemas pueden estar basados en la visión por computador, ya sea mediante cámaras, infrarrojos o cualquier tipo de sensor electrónico que le permita al coche saber que tiene delante o simplemente componentes electrónicos que mejoren la conducción. A continuación, se comentan algunos ejemplos de estos sistemas.

3.1. Sistemas de detección de la fatiga y falta de concentración al volante

En los últimos tiempos están empezando a verse vehículos que incorporan sistemas que avisan al conductor si estos detectan algún indicio de fatiga o falta de concentración al volante [9]. Estos sistemas son denominados por cada fabricante de una manera diferente, pero, al fin y al cabo, se tratan de sistemas similares que ejercen la misma función. Detectar si el conductor está en óptimas condiciones para que continúe conduciendo.

Existen diversas formas de detectar esta fatiga o falta de concentración. Una de ellas es mediante la introducción de sensores en el volante. Aprenden nuestra dinámica de conducción y cuando no seguimos esa dinámica entienden que estamos fatigados o nos estamos quedando dormidos. En líneas generales el modo de funcionamiento es que al conducir en línea recta no sujetamos el volante fijo, sino que hacemos pequeñas correcciones suaves para mantenernos en el carril. Cuando estamos muy cansados esas correcciones no las hacemos de igual manera o incluso las podemos hacer bruscas. Cuando el sistema detecta esto hace saltar una alarma sonora y muestra un mensaje de texto en la pantalla avisando.

Otro sistema, que hace la misma función, aunque es ligeramente diferente, es la detección de fatiga mediante visión. Esto se basa en la detección de parpadeo del conductor, si es normal o tiene indicios de cansancio, o si el conductor mantiene la vista centrada en la carretera. Esto es posible gracias a la información que nos llegan desde los ojos . Uno de los índices más utilizados para medir la fatiga en tiempo real es el denominado PERCLOS [10] que relaciona el porcentaje de cierre de parpado.

Estos sistemas, en España, han hecho descender el porcentaje de accidentes en los últimos años, siendo, actualmente, la fatiga o el cansancio un 20% de las causas de los accidentes en carretera.

3.2. Detección y reconocimiento de señales de tráfico

El sistema de detección y reconocimiento de señales de tráfico [11] apareció por primera vez en el 2008 en el modelo BMW serie 7 y luego más tarde lo fueron incorporando diversos fabricantes. Generalmente sólo detectan límites de velocidad, aunque, recientemente, estamos viendo distintos fabricantes que ya incorporan sistemas que detectan otras restricciones.

Estos sistemas hacen uso de la visión por computador, la cual hace un tratamiento digital de la imagen que la cámara capta de manera que el ordenador la analiza e interpreta.

En la detección de señales existen dos tipos de aplicaciones, activa y pasiva.

- Aplicación activa: el sistema analiza señales y si detecta que el conductor no reacciona, este actúa en consecuencia. Por ejemplo, si el sistema lee un STOP y conforme a la velocidad y a la distancia a él el conductor no ha reaccionado el sistema frenaría el coche por él para, así, evitar accidentes.
- Aplicación pasiva: Consiste en advertir al conductor de las condiciones de la carretera, ya sea mediante sonidos, imágenes o cualquier otro tipo de señales.

3.3. Detección de cambio de carril involuntario

El 30% de los fallecidos en las carreteras españolas se deben a una salida de la vía. De la necesidad de evitar estos accidentes, que en Europa también generan miles de fallecidos cada año, surgió la alerta por cambio involuntario de carril.

Existen diversos sistemas que ayudan a lograr la detección de este cambio. [12]

El más sencillo y económico es el sistema basado en sensores infrarrojos, aunque también es el menos efectivo para predecir con anticipación este cambio. Un sensor es capaz de detectar el cambio de carril, mediante un haz de luz infrarroja reflejada en la pintura reflectante de las líneas.

Otro muy común es el uso de sistemas basados en visión por computador. El cual se encarga de procesar las imágenes captadas por las cámaras de las líneas de los carriles.

Por último, existe el scanner laser, que es el menos común y el más costoso. La tecnología láser permite la detección de objetos midiendo el tiempo que tarda un pulso de luz en ser reflejado por el objeto. Permite detectar cambios de la reflectividad en el asfalto y reconocer las líneas que delimitan el carril y de los bordes de la carretera. Se trata del método más efectivo para detectar estos cambios, siendo capaz de cubrir distancias superiores a 200 metros.

Los sistemas actuales están pensados para funcionar por encima de los 80 km/h, dado que por debajo de esa velocidad el sistema no es capaz de discriminar eficazmente si se trata de un cambio de carril voluntario o no.

Existen, al igual que con los sistemas de detección de señales, aplicaciones tanto activas como pasivas. Avisando al conductor si se trata de una aplicación pasiva o actuando en consecuencia, si lo requiere, en la aplicación activa.

3.4. Cámara delantera en vehículos grandes

ILUSTRACIÓN 5: CAMIÓN CON CÁMARA DELANTERA [13]

Una de las medidas promovidas por alguien ajeno al sector automovilístico, Samsung, es la de dotar a vehículos de ciertas dimensiones de una cámara delantera para poder reflejarla en su parte trasera. Facilitando a los coches que se encuentren detrás de él la visión de la carretera.

3.5. Advertencia de ángulo muerto en el espejo retrovisor

Otra tecnología que se están incluyendo ya en muchos coches de gama media-alta es la advertencia de ángulo muerto [14] situado en el espejo retrovisor. El funcionamiento se basa en la inclusión de sensores que se encuentran alojados en los retrovisores o en el pilar A y que detecta cuando un vehículo se aproxima por los laterales.

Si en ese momento accionamos los intermitentes para un desplazamiento lateral el sistema nos avisara con un mensaje visual y/o sonoro para avisarnos que un vehículo se aproxima.

3.6. Conducción autónoma

Actualmente se están avanzando mucho en los sistemas de conducción autónoma. Google [15], por su parte, está elaborando un coche el cual se conduce solo. Empezó a operar con coches de conducción autónoma en 2009. Estos coches están equipados con la tecnología más puntera del mercado como lo es un sistema LIDAR. El trabajo conjunto de ambos les permite saber distancias de objetos al igual que generar un detallado mapa 3D de la situación. Los coches cogen esos mapas generados y los combinan con los mapas del mundo, produciendo diferentes tipos de modelos que les permiten conducirse solos. Estos vehículos han avanzado desde su modelo inicial que fue un Toyota Prius o un Lexus RX450h a un modelo mucho más comedido que fue presentado en mayo de 2014 desarrollado por ellos mismos y ensamblado por Rousg Enterprises el cual usa equipamiento procedente de Bosch, ZF Lenksysteme, LG y Continental.

ILUSTRACIÓN 6: PRIMER PROTOTIPO DE COCHE AUTÓNOMO DE GOOGLE

ILUSTRACIÓN 7: PROTOTIPO ACTUAL DEL COCHE AUTÓNOMO DE GOOGLE

.

Tesla [16] también se sumó a la idea de coche autónomo dotando a su Model S de la función Autopilot. No se trata de un coche 100% autónomo al uso, pues únicamente solo obedece a las instrucciones que le ha dictado el conductor. Por ejemplo, permanecerá en un carril, guardando la distancia de seguridad, de manera automática, pero solamente adelantará al coche de delante si el conductor se lo indica con el encendido del intermitente.

ILUSTRACIÓN 8: TESLA MODEL S [16]

3.7. Aparcamiento automático

Un sistema que cada vez está presente en los nuevos coches es el aparcamiento automático [17]. Para poder conseguir este avance, los coches han incorporado numerosos radares, todos ellos en perímetro de su carrocería. Con estos radares analizan el resto de vehículos aparcados a su alrededor para encontrar un hueco vacío en paralelo o batería. Si el conductor desea aparcar en ese hueco únicamente tendrá que presionar un botón de manera que el coche se encargará del manejo del volante dejando al conductor el manejo de los pedales y la introducción de las marchas en función de lo que le vaya indicando el coche. Los sensores de distancia, normalmente de ultrasonidos, en el paragolpes delantero y trasero, evitaran rozar o golpear con otros coches. Este sistema está presente, actualmente, en numerosos coches como por ejemplo el Audi A7, Audi TT, Ford Focus, Ford Mondeo y Opel Corsa entre otros muchos coches.

4. FUNDAMENTOS TEÓRICOS

4.1. Óptica

A continuación, se va a explicar los dos modelos principales que nos permiten la captación de una imagen para poder tener una mejor concepción del tema que se va a tratar. Los modelos en cuestión son el modelo pin-hole y el modelo de lente fine.

4.1.1. Modelo Pin-Hole

Una cámara basada en este modelo consiste en una caja la cual carece de lente y únicamente dispone en la parte delantera de una pequeña apertura, de ahí su nombre (Pin Hole) o en castellano "Agujero de aguja". De otra manera, se trata de una caja con un agujero en la parte delantera, siendo este orificio la entrada de luz a la caja. Cuando la luz de una imagen pasa a través de este agujero se forma una imagen invertida en la película fotográfica situada en el otro extremo dentro de la caja. La imagen que se forma estará invertida a la imagen original. El funcionamiento de este modelo es muy similar al del ojo humano.

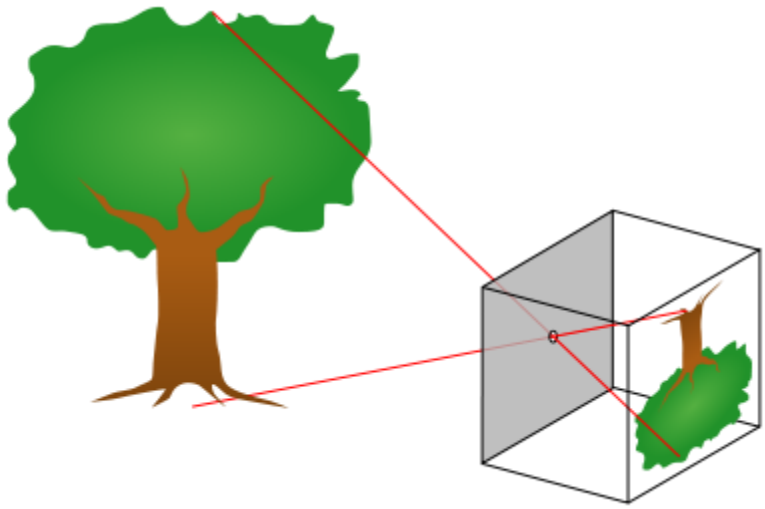

ILUSTRACIÓN 9: CÁMARA PIN-HOLE [18]

El modelo pin-hole describe la relación que existe entre las coordenadas de un punto 3D y su proyección en el plano de la imagen. Las ecuaciones (1) y (2) se obtienen por triangulación a partir de los parámetros que se indican en la Ilustración 10 e Ilustración 11.

$$x = \frac{f}{Z}.X \qquad (1)$$

$$y = \frac{f}{Z}.Y \qquad (2)$$

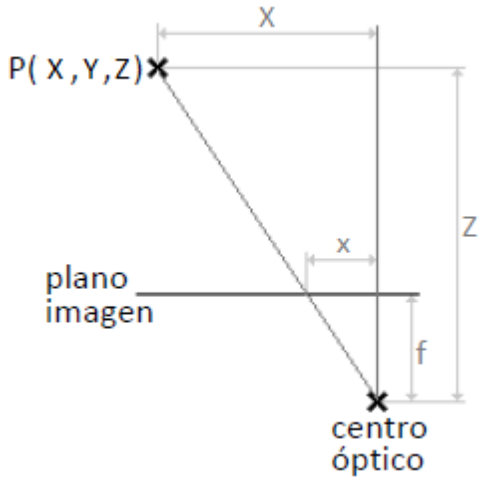

ILUSTRACIÓN 10: ESQUEMA DEL MODELO PIN-HOLE (1)

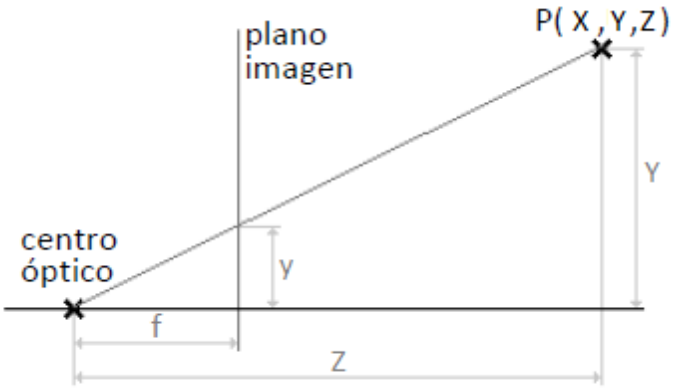

ILUSTRACIÓN 11: ESQUEMA DEL MODELO PIN-HOLE (2)

4.1.2. Modelo lente fina

Este modelo consiste en una lenta que además de ser biconvexa posee un grosor despreciable, lo que le permite concentrar en un punto todos los infinitos rayos luminosos procedentes de un punto del espacio. El plano donde se encuentra el punto en el que se concentran todos los rallos se denomina *plano de formación de la imagen.* Llamamos centro óptico al punto interior de la lente donde corta el eje óptico al plano de simetría de la lente. Todos los rayos paralelos cortan en un punto del eje óptico a una distancia igual a la focal del centro de la lente. La distancia que separa el foco del centro óptico de la lente se denomina distancia focal (f).

Este modelo está inspirado en la fórmula de Gauss de las lentes delgadas con rayos paraxiales.

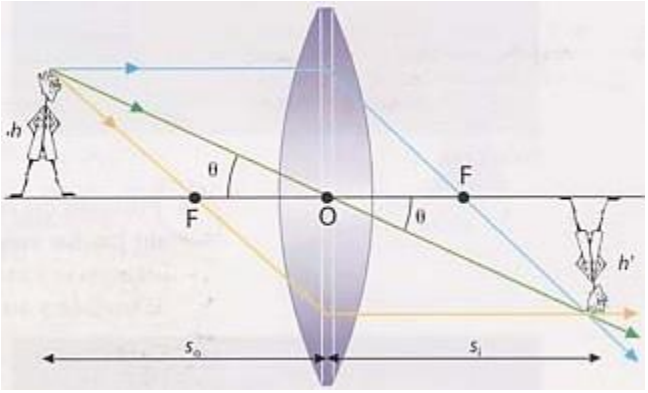

ILUSTRACIÓN 12: MODELO LENTE FINA [19]

4.1.3. Visión estéreo

La visión estéreo se basa en el sistema de captar las imágenes que poseen los seres humanos, en el que, debido a la separación de nuestros ojos, se obtienen dos imágenes con pequeñas diferencias entre ellas, encontraremos las diferencias en la posición relativa de los objetos, lo que denominamos disparidad. El cerebro se encarga de procesar las diferencias entre ambas imágenes y las interpreta de forma que se percibe una sensación de profundidad, cercanía o lejanía de los objetos de nuestro alrededor.

El procedimiento de estas cámaras consiste en capturar dos imágenes de una misma escena, cada imagen es capturada desde una posición diferente, la posición de las cámaras imitará la posición de nuestros ojos, por lo que las imágenes se mostrarán ligeramente desplazadas entre sí.

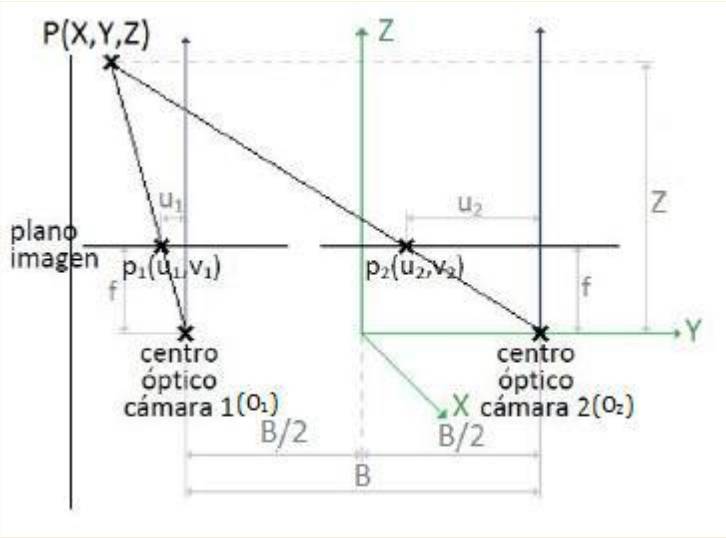

ILUSTRACIÓN 13: REPRESENTACIÓN DE LA PROYECCIÓN ESTEREOSCÓPICA

Atendiendo a un punto de vista computacional, un sistema de visión estereoscópica debe resolver dos tipos de problemas. El primero de ellos es conocido como *correspondencia*. Se conoce como correspondencia estereoscópica a la identificación de un mismo punto u objeto en las dos imágenes capturadas. El segundo problema se denomina *reconstrucción*, cuyo principal objetivo es encontrar las coordenadas del punto u objeto en cuestión. Para calcular las coordenadas X, Y de un punto P y la profundidad Z se ha de recurrir a las ecuaciones del modelo pin-hole (1) y (2) a partir de las cuales se pueden obtener las ecuaciones de un sistema estéreo (3), (4), (5).

$$X = Z\frac{u1}{f} - \frac{B}{2} = Z\frac{u1}{f} + \frac{B}{2} \qquad (3)$$

$$Y = \frac{B \cdot v1}{d} \qquad (4)$$

$$Z = \frac{f \cdot B}{u1 - u2} = \frac{f \cdot B}{d} \qquad (5)$$

Siendo, en las anteriores expresiones, f la distancia focal y del valor de disparidad dado por d=u1-u2.

4.2. Descriptores basados en Histograma de Gradientes Orientados (HOG)

Los descriptores HOG [20] (Histogram of Oriented Gradients) describen la orientación del gradiente de toda una región dada. Se obtienen dividiendo la imagen en celdas, calculando el histograma de la dirección del gradiente o de la orientación de los bordes de cada una de las celdas. Estos histogramas capturan propiedades de forma local dentro de la celda, estas propiedades serán bordes de la imagen. El gradiente en cada pixel se encuentra discreteado en uno de los nueve contenedores de orientación (bins), y cada pixel vota por la orientación de su gradiente. Para mejorar la invariancia frente a cambios de iluminación o sombras se realiza una normalización del contraste en cada uno de los histogramas que ha generado.

Los métodos HOG se fundamentan en las bases teóricas de trabajos previos como Histograma de bordes orientados [Freeman and Roth 1995, Freeman et al], descriptores SIFT2 [Lowe 1004] y reconocimiento de formas [Belongie et al. 2001], entre otros. Sin embargo, el valor extra que presenta los métodos HOG reside en que no se calculan los gradientes sobre un mallado denso, sino que la imagen se divide en bloques que a su vez se dividen en subbloques y se calcula el gradiente y el histograma en cada uno de estos subbloques.

El procedimiento para el cálculo de los descriptores de una imagen cualquiera de tamaño 64x128 puede verse en la siguiente ilustración.

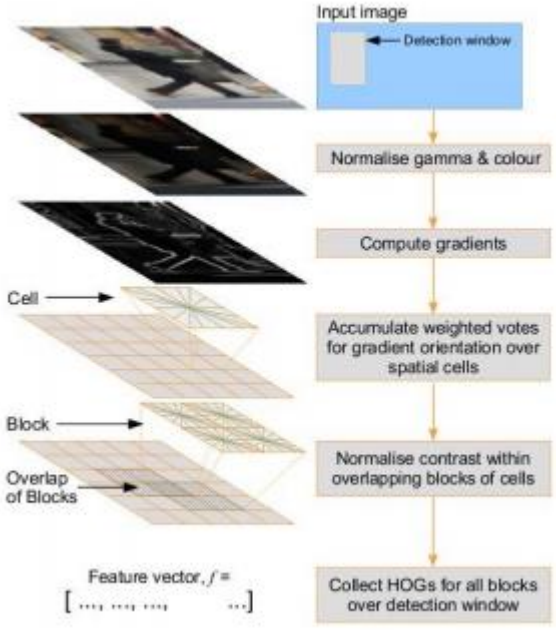

ILUSTRACIÓN 14: PROCESO DE EXTRACCIÓN DE CARACTERÍSTICAS [20]

Como se ve en la Ilustración 14 primero se dispone de una imagen a color que es transformada a escala de grises. Se calculan los gradientes espaciales sobre toda la imagen. Más tarde, se divide la imagen en bloques los cuales se vuelven a dividir en bloques (o celdas). Una vez hechas todas las divisiones pertinentes se calcula los histogramas de los gradientes orientados. Por último, se aplica una ventana gaussiana sobre cada bloque,

almacenándose esta información en el vector de características de la imagen. El proceso se repetirá tantas veces como bloques tenga la imagen.

4.3. Máquinas de soporte vectorial (SVM)

Las máquinas de soporte vectorial [20] son capaces de clasificar muestras en dos posibles conjuntos. En el caso que tratamos los clasificará en "personas en bicicleta" y "no personas en bicicleta" y por otra parte en el otro entrenamiento que hagamos lo clasificará en "personas" y "no personas". Para hacer esta clasificación se necesita, previamente, un entrenamiento de la máquina, facilitándole ejemplos de ciclistas o personas que serán los positivos y ejemplos de no-ciclistas o no-personas (esta elección dependerá del tipo de entrenamiento que estemos haciendo) que serán los negativos. Con los ejemplos de entrenamiento ya adjuntados al programa, el algoritmo de clasificación SVM elaborará una curva M-dimensional que divide ambos conjuntos, obteniendo así el kernel de la máquina. Las dimensiones del espacio dependen del número de componentes de cada vector a clasificar.

Hay diferentes parámetros a la hora de la realización del kernel. Podemos hablar de curvas trazadas con funciones lineales, polinómicas $k(x, x') = (s \cdot (x \cdot x') + c)^d$, exponenciales $k(x, x') = exp(-gamma \cdot ||x - x'||^2)$, de tangente hiperbólica $k(x, x') = tanh(s \cdot (x \cdot x') + c)$ y definir cada uno de sus parámetros, como el grado del polinomio, el valor de gamma, el valor de sigma "s",…

Por todo ello, la elaboración del kernel óptimo se hace tediosa y requiere mucho tiempo. Además, muchas aplicaciones toman parte en máquinas de este tipo, con lo que no hay, a priori, una inclinación por un método u otro.

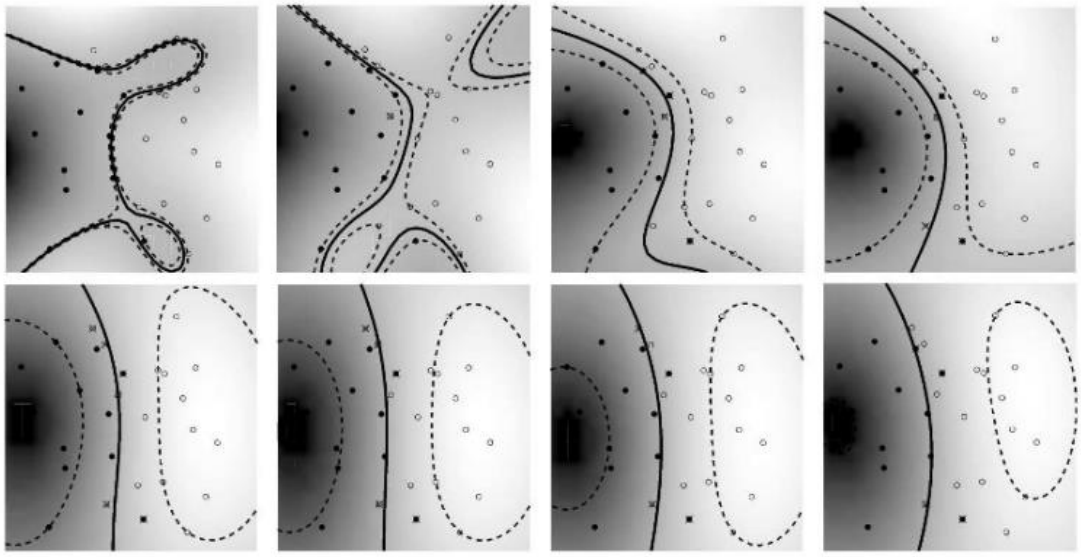

ILUSTRACIÓN 15: EJEMPLO DE FUNCIONAMIENTO DE UN KERNEL GAUSSIANO PARA VARIOS VALORES DEL PARÁMETRO GAMMA, CONSISTENTE EN LA SEPARACIÓN DE DOS CONJUNTOS MUESTRALES (x_1,y_1) (x_M, y_M)∈ X × {± 1} [20]

La Ilustración 15 muestra un ejemplo de funcionamiento de un kernel gaussiano. Los valores tomados de este parámetro van desde *gamma = 0.1* (arriba a la izquierda) a *gamma = 0.8* (abajo a la derecha). Por lo que vemos que cuanto más alto sea el valor de gamma, más puntos permitimos que se encuentren en la zona alojada entre ambos conjuntos.

Una vez que ha sido trazada la línea que separa ambos espacios M-dimensionales, cuando queramos clasificar una nueva muestra no identificada deberemos introducirla en la máquina y representarla como un punto en este espacio. La distancia euclídea de la muestra a la curva trazada en la etapa de entrenamiento de la máquina resultará ser la medida que nos da la predicción sobre la muestra. El signo de esta distancia nos informará la pertenencia de la muestra a un conjunto u otro (en nuestro caso, “ciclistas” o “no ciclistas” y en el otro entrenamiento “personas” o “no personas”). Un signo positivo nos indicará que la muestra clasificada pertenece al primer grupo. Por lo tanto, un signo negativo nos indicará que la muestra pertenece al segundo grupo. De manera que, el módulo de esta distancia nos indicará la probabilidad de que la muestra pertenezca a un conjunto u otro. Por ejemplo, valores de 3.4 indicarán que la muestra pertenece al espacio de “personas” con mayor probabilidad que si la predicción fuera de 0.3. En el segundo caso, la máquina no predice con tanta exactitud la pertenencia de la muestra a un conjunto u otro. Algo que hace que estas máquinas sean muy atractivas es la capacidad que poseen para aprender y la alta precisión en la clasificación de un conjunto muestral de test entre los dos posibles conjuntos. En los últimos tiempos, SVM ha sido muy utilizado para diversas aplicaciones y por diversos investigadores del campo de reconocimiento de formas.

5. DESCRIPCIÓN GENERAL DEL SISTEMA

Para realizar este proyecto se ha hecho uso tanto de elementos hardware, el ordenador que se encargara de entrenar y posteriormente clasificar lo entrenado y una cámara estéreo para grabar la secuencia. Como de elementos software, el lenguaje de programación (C++), así como las librerías que hemos usado (OpenCV).

5.1. Hardware

5.1.1. NVIDIA Jetson TK1

Jetson TK1 [21]es la plataforma de desarrollo embebido de NVIDIA bajo el sistema operativo LINUX. Ofrece un SOC Tegra K1 (Se trata de CPU + GPU + ISP en un solo chip). La Jetson TK1 viene con Linux4Tegra OS preinstalado. Básicamente se trata de Ubuntu 14.04 con controladores preconfigurados).

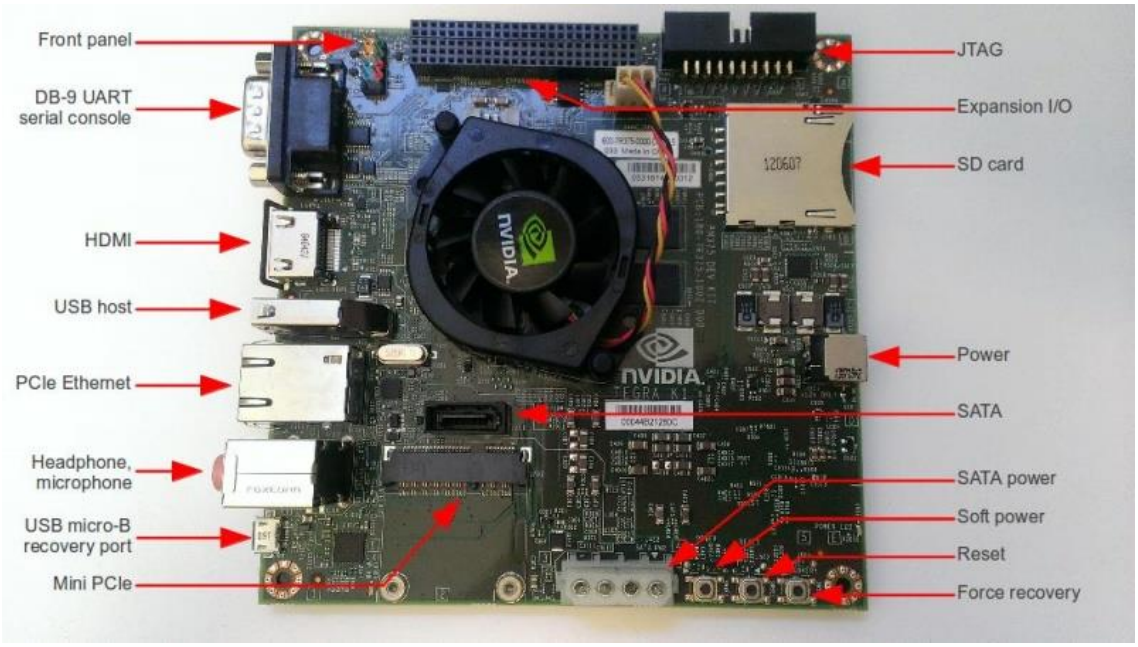

ILUSTRACIÓN 16: JETSON TK1 [21]

Además de la CPU con cuatro núcleos a 2,3 GHz ARM Cortex-A15 y la GPU Tegra TK1 que posee, la Jetson TK1 incluye características similares a las que ofrecería una Raspberry PI, incluyendo, también, características orientadas a PC como son la conexión SATA, mini-PCIe y un ventilador que hace de sistema de refrigeración de la plataforma, permitiendo así una continua operación bajo fuertes cargas de trabajo.

Todas las características de la Jetson TK1 se detallan a continuación:

- **Dimensiones de la placa:** 127mm x 127mm.
- **SOC Tegra K1** (CPU+GPU+ISP en un solo chip, con un consumo energético típico, entre 1 y 5 Vatios):
- **GPU:** NVIDIA Kepler "GK20a" GPU con 192 núcleos CUDA.
- **CPU:** ARM Cortex A15 de cuatro núcleos NVIDIA "4-Plus-1" a 2.32GHz
- **DRAM:** 2GB de memoria DDR3L a 922MHz EMC x16 con 64 bits de ancho de banda.
- **Almacenamiento:** 16 GB de memoria eMMC 4.51.
- **mini-PCIe:** 1 Ranura para mini-PCIe de media altura (permite la conexión de módulos WIFI, SSD en RAID, FireWire o tarjetas Ethernet).
- **Conector SD/MMC:** 1 conector SD/MMC de tamaño normal.
- **USB 3.0:** 1 puerto USB 3.0, A.
- **USB 2.0:** 1 puerto USB 2.0, micro-AB.
- **HDMI:** 1 puerto HDMI de tamaño normal.
- **RS232:** 1 puerto serie RS232.
- **Audio:** 1 codec de audio ALC5639 Realtek con entrada de micrófono (Mic in) y salida de línea (Line out).
- **Ethernet:** 1 puerto de LAN RTL8111GS Realtek GigE.
- **SATA:** 1 puerto de datos SATA (soporta tantos discos de 2.5” como 3,5”).
- **Alimentación:** 1 conector de alimentación 12V DC y un conector de alimentación PC IDE de 54 pines, utilizando PMIC AS3722.
- **Ventilador:** 1 ventilador disipador de calor que funciona a 12 V.

Las siguientes características están disponibles a través de un Puerto de expansión:

- **Puerto LCD:** DP/LVDS
- **Puertos para pantalla táctil:** SPI para pantalla táctil, buses CSI-2 1x4 + 1x1.
- **UART**
- **HSIC**
- **I2C:** 3 puertos
- **GPIO:** 7 x GPIO pines (1.8V)

Conectores del panel frontal:

- Verde - Power LED
- Naranja - HDD LED
- Rojo – Botón de alimentación
- Morado / Azul – Botón de Reset

APIs suportadas de aceleración por Hardware.

- **CUDA** 6.
- **OpenGL** 4.4
- **OpenGL ES** 3.1
- **OpenMAX IL multimedia codec** incluyendo H.264, VC-1 y VP8 a través Gstreamer
- **NPP**
- **OpenCV4Tegra** (NEON + GLSL + quad-core CPU quad-core optimizados)
- **VisionWorks**

5.2. Software

5.2.1. QtCreator

El entorno de desarrollo que se ha decidido utilizar es QtCreator [22]. Se trata de un entorno de desarrollo integrado que cuenta con soporte para C+. QML y ECMAscript. Además, puede ser compilado por cualquier compilador de C ++ estándar compatible como Clang, GCC, CPI, MinGW y MSVC.

Además, las características del editor incluyen el resaltado de sintaxis y autocompletado.

La elección de este programa se debe a que se trata del IDE con el que he tratado más tiempo, por lo tanto, mi adaptación a él fue inmediata.

5.2.2. OpenCV

OpenCV [23] se trata de una biblioteca libre de visión artificial que inicialmente fue desarrollada por Intel en 1999. Actualmente tienen una comunidad de usuarios que superan las 47 mil personas además de un número de descargas estimado de 9 millones. OpenCV fue diseñado para la eficiencia computacional con un fuerte enfoque en aplicaciones en tiempo real. Para la mejor eficiencia de ellas permite tomar procesamiento multi-núcleo que se activa con OpenCL. De manera que permite aprovechar la aceleración por hardware, lo que es ideal para nuestro proyecto. Además, la publicación de OpenCV está bajo una licencia BSD. De manera que es libre de ser usado para uso académico o comercial. Gracias a esto OpenCV es usado en infinidad de aplicaciones. Se utilizan tanto en sistema de seguridad como en cualquier control de procesos que requiera algún tipo de clasificación de objetos.

OpenCV es multiplataforma. Existen versiones para GNU/Linux, Mac OSX y Windows. Además, contiene más de 500 funciones que abarcan una gran fama de áreas en el proceso de visión computacional, como, por ejemplo, el reconocimiento de objetos (también está incluido en esta área el reconocimiento facial), visión estéreo, calibración de cámaras y visón robótica.

Las OpenCV presenta 5 librerías como estructura fundamental:

- **cv**: principales algoritmos

- **cxcore**: estructuras básicas
- **cvaux**: algoritmos más experimentales
- **higui**: GUI and Video I/O
- **ml**: machine learning

Las funciones de OpenCV se centran, principalmente, en la visión por computador. El uso de OpenCV se debe a que dispone de muchas funciones ya programadas que, además, permite el uso de ellas tanto en CPU como GPU. Por lo que me generaba muchas facilidades a la hora de realizar el proyecto.

5.2.3. Librerías Boost C++

Las librerías Boost se tratan de un conjunto de bibliotecas de software libre bajo una licencia de tipo BSD diseñadas para el lenguaje de programación C++. Estas librerías proporcionan apoyo a las tareas y estructuras como algebra lineal, generación de números pseudoaleatorios, expresiones regulares y procesamiento de imágenes entre otros.

Boost contiene más de ochenta bibliotecas individuales.

6. ESTUCTURA DEL PROGRAMA

El programa que nos ocupa se podría dividir en varios módulos o funciones principales.

Primeramente, aunque el programa haya sido concebido con un video, que al fin y al cabo se compone en una sucesión constante de imágenes, está ideado y es perfectamente capaz de ser ejecutado en tiempo real con las imágenes captada a través de una videocámara, además con el aliciente de la incorporación de la aceleración mediante GPU permite un tratado de las imágenes mucho más rápido. Por lo tanto, primero estaría la imagen captada por la cámara. El tratamiento de imágenes se haría en "bruto" no haría falta corregir distorsión alguna pues no estaríamos haciendo uso de una cámara con ojo de pez.

A continuación, se llevaría a cabo la detección de los elementos que interesen. El programa comienza a buscar por la imagen. Primero, empieza con la búsqueda de elementos que se asemejen a una persona montada en una bicicleta para luego continuar la búsqueda de elementos que se asemejen a un peatón y la zona donde se detecte un peatón o una persona montada en bicicleta se guardará como región de interés (ROI).

Finalmente, las detecciones de estos ROIs se llevan a cabo mediante clasificadores, previamente entrenados, que son los encargados de distinguir que tipo de detección se trata para posteriormente mandar un mensaje informando de ello y las características de las detecciones o, si se visualiza en tiempo real, recuadrando de un color distinto los diferentes tipos de detecciones.

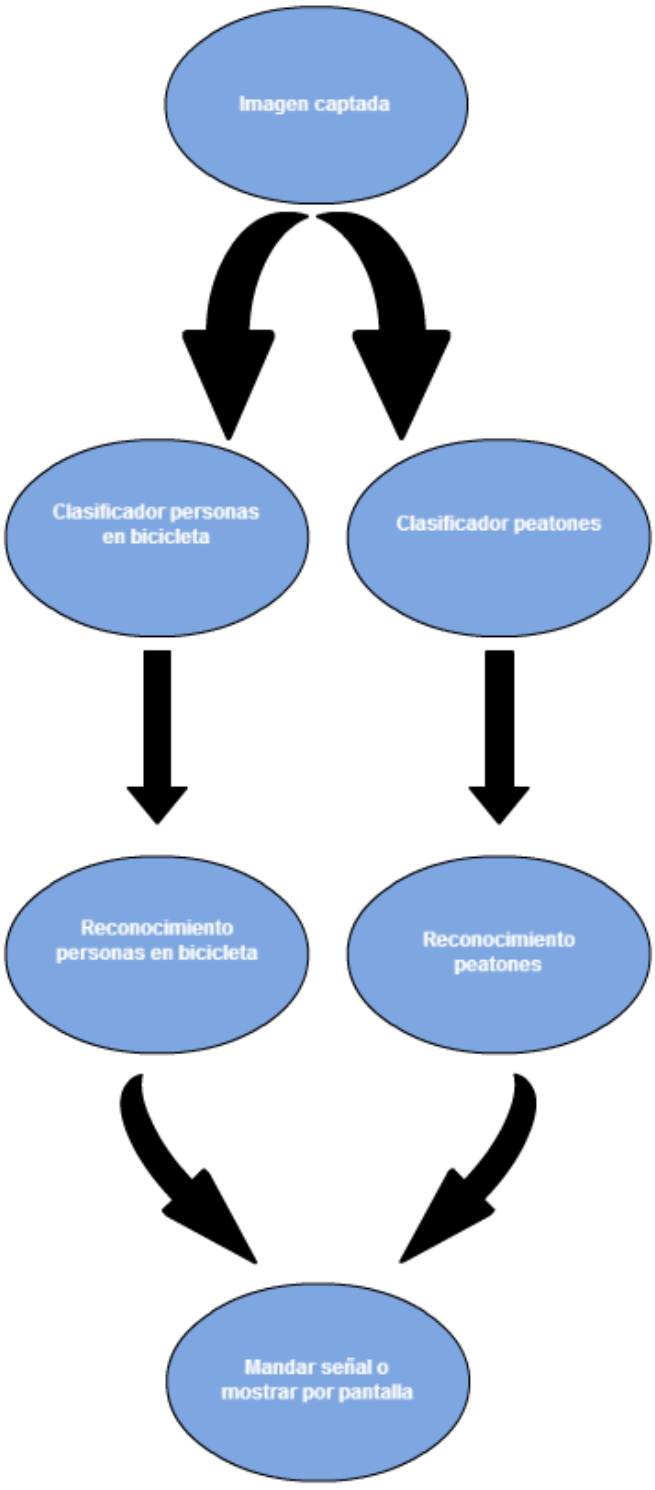

ILUSTRACIÓN 17: ESTRUCTURA DEL PROGRAMA

7. DESARROLLO DEL ALGORITMO

7.1. Desarrollo del clasificador

El reconocimiento ciclistas y peatones se lleva a cabo mediante un clasificador SVM. Para llevarlo a cabo se han tenido que recoger numerosas muestras de las personas en bicicleta y peatones con las que poder entrenar. Se han tenido que hacer dos clasificadores diferentes, uno para las personas en bicicleta y otro para los peatones. Finalmente se ejecuta cada clasificador por separado por cada frame para comprobar la presencia de un peatón o una persona en bicicleta en él.

7.1.1. Preparación de las muestras

Para obtener las muestras positivas de bicicletas han sido procesados varios archivos de video en los que salían bicicletas, para este caso se han utilizado muestras tomadas en el día de la bicicleta de Madrid del año pasado. El recorte de estas muestras ha sido hecho a mano una a una por cada frame. Estos recortes se han hecho mediante un algoritmo que simplificaba los recortes hecho por mi tutor, Aurelio Ponz Vila.

ILUSTRACIÓN 18: EJEMPLO MUESTRA CICLISTA POSITIVA(64X64)

Las muestras negativas no tienen por qué guardar relación alguna con lo que se vaya a entrenar por lo tanto las muestras que se han cogido han sido de cualquier elemento que no fuera una bicicleta, desde edificios, coches, árboles, etc…

ILUSTRACIÓN 19: EJEMPLO MUESTRA CICLISTA NEGATIVA (64X64)

Para el clasificador de bicicletas se han empleado un total de 22.246 muestras de las cuales 18557 eran negativas y 3689 positivas.

A la hora de conseguir las muestras de peatones se intentó con imágenes procedentes de una base de datos de INRIA. Pero estas muestras no eran del todo claras ya que, en los recortes positivos, los cuales recortamos mediante un algoritmo hecho en Python una vez sabiendo las coordenadas de estos, salían varias personas o incluso personas con una silueta no muy clara, por ejemplo, personas con esquís y con el aderezo que ello implica. Con lo que desechamos la opción.

Una vez desechada esta opción procedimos a generar nosotros las muestras. Para ello los integrantes del Laboratorio de Sistemas Inteligentes de la Carlos III nos reunimos en el sótano de la biblioteca del campus de Leganés y en el croma que disponen en una de las habitaciones generamos nosotros las secuencias. Una vez generadas las secuencias los recortes serían mucho más fáciles, ya que las secuencias se generaron a voluntad, y los sujetos permanecían en una marca haciendo gestos los gestos típicos que se hacen al andar. Por lo que únicamente, mediante un script habría que indicarle las coordenadas de los sujetos y saldrían los recortes automáticamente.

ILUSTRACIÓN 20: EJEMPLO MUESTRA PEATONES POSITIVA (64X128)

ILUSTRACIÓN 21: EJEMPLO MUESTRA PEATONES NEGATIVA (64X128)

Para el entrenamiento del clasificador de peatones se han utilizado, inicialmente, 9.880 imágenes positivas y solamente 400 negativas. A lo que hemos tenido que recurrir al boostrapping en imágenes muy variadas como por ejemplo ciudades, paisajes, señales, cualquier lugar donde pudiera haber un falso positivo en la vida real. Tras la realización del boostrapping han salido 12.629 imágenes negativas.

7.1.2. Descripción del código

La aplicación *SVMclassifier.pro* nos servirá para proceder al desarrollo del clasificador.

Durante el desarrollo del programa, el análisis de las imágenes se lleva a cabo con las imágenes que previamente hemos recortado. En este apartado se describirán aquellas líneas de código cuya interpretación pueda resultar más compleja, así como algunas de sus funciones.

```
int main(int argc, char** argv) {

    // <editor-fold defaultstate="collapsed" desc="Init">
    HOGDescriptor hog,hog1; // Use standard parameters here
    hog.winSize = Size(64, 64); // Default training images size as used in paper
    hog1.winSize = Size (64,64);
    // Get the files to train from somewhere
    static vector<string> positiveTrainingImages;
    static vector<string> negativeTrainingImages;
    static vector<string> validExtensions;
    validExtensions.push_back("jpg");
    validExtensions.push_back("png");
    validExtensions.push_back("ppm");
    // </editor-fold>

    // <editor-fold defaultstate="collapsed" desc="Read image files">
    getFilesInDirectory(posSamplesDir, positiveTrainingImages, validExtensions);
    getFilesInDirectory(negSamplesDir, negativeTrainingImages, validExtensions);
```

ILUSTRACIÓN 22: PARTE DEL MAIN DE LA APLICACIÓN SVMCLASSIFIER.PRO

Comenzamos definiendo el descriptor HOG que vamos a utilizar, así como su tamaño. Los vectores declarados servirán para, en el caso de ***validExtension*** para almacenar las extensiones que consideramos validas en las imágenes, y en el caso de ***positiveTrainingImages*** y ***negativeTrainingImages*** servirá para almacenar todas aquellas direcciones de los archivos que vayamos a utilizar para el entrenamiento. En la Ilustración 23 llamamos a la función ***getFilesInDirectory()***:

```
static void getFilesInDirectory(const string& dirName, vector<string>& fileNames, const vector<string>& validExtensions) {
    printf("Opening directory %s\n", dirName.c_str());
    struct dirent* ep;
    size_t extensionLocation;
    DIR* dp = opendir(dirName.c_str());
    if (dp != NULL) {
        while ((ep = readdir(dp))) {
            // Ignore (sub-)directories like . , .. , .svn, etc.
            if (ep->d_type & DT_DIR) {
                continue;
            }
            extensionLocation = string(ep->d_name).find_last_of("."); // Assume the last point marks beginning of extension like file.ext
            // Check if extension is matching the wanted ones
            string tempExt = toLowerCase(string(ep->d_name).substr(extensionLocation + 1));
            if (find(validExtensions.begin(), validExtensions.end(), tempExt) != validExtensions.end()) {
                printf("Found matching data file '%s'\n", ep->d_name);
                fileNames.push_back((string) dirName + ep->d_name);
            } else {
                printf("Found file does not match required file type, skipping: '%s'\n", ep->d_name);
            }
        }
        (void) closedir(dp);
    } else {
        printf("Error opening directory '%s'!\n", dirName.c_str());
    }
    return;
}
```

ILUSTRACIÓN 23: FUNCIÓN GETFILESINDIRECTORY

La función ***getFilesInDirectory()*** se encarga de coger de la dirección que hemos enviado en el primer parámetro de la función todas las direcciones de los archivos que se encuentren

en ese directorio y tengan una extensión que esté incluida en el vector ***validExtensions*** y almacenarlas en el vector que le hemos enviado en el segundo parámetro de la función ***positiveTrainingImages*** o ***negativeTrainingImages***. Por lo tanto, ahora dispondremos de un vector lleno de direcciones a nuestras imágenes.

Más tarde, en el **main()**, procedemos a calcular todas las características HOG de las imágenes positivas y negativas. Para ello procedemos a llamar a la función ***calculateFeaturesFromInput().***

```
    fstream File;
    File.open(featuresFile.c_str(), ios::out);
    if (File.good() && File.is_open()) {
        // Remove following line for libsvm which does not support comments
        // File << "# Use this file to train, e.g. SVMlight by issuing $ svm_learn -i 1 -a weights.txt " << featuresFile.c_str() 
        // Iterate over sample images
        for (unsigned long currentFile = 0; currentFile < overallSamples; ++currentFile) {
            storeCursor();
            vector<float> featureVector;
            // Get positive or negative sample image file path
            const string currentImageFile = (currentFile < positiveTrainingImages.size() ? positiveTrainingImages.at(currentFile)
            // Output progress
            if ( (currentFile+1) % 10 == 0 || (currentFile+1) == overallSamples ) {
                percent = ((currentFile+1) * 100 / overallSamples);
                printf("%5lu (%3.0f%%):\tFile '%s'", (currentFile+1), percent, currentImageFile.c_str());
                fflush(stdout);
                resetCursor();
            }
            // Calculate feature vector from current image file
            calculateFeaturesFromInput(currentImageFile, featureVector, hog);
            if (!featureVector.empty()) {
                /* Put positive or negative sample class to file,
                 * true=positive, false=negative,
                 * and convert positive class to +1 and negative class to -1 for SVMlight
                 */
                File << ((currentFile < positiveTrainingImages.size()) ? "+1" : "-1");
                // Save feature vector components
                for (unsigned int feature = 0; feature < featureVector.size(); ++feature) {
                    File << " " << (feature + 1) << ":" << featureVector.at(feature);
                }
                File << endl;
            }
        }
        printf("\n");
        File.flush();
        File.close();
    } else {
        printf("Error opening file '%s'!\n", featuresFile.c_str());
        return EXIT_FAILURE;
    }
```

ILUSTRACIÓN 24: APERTURA Y EXTRACCIÓN DE CARACTERÍSTICAS DE LAS IMÁGENES

La función ***calculateFeaturesFromInput()*** será la encargada de calcular todas las características HOG de las imágenes, asignándole un valor de +1 si se trata de una imagen positiva y de -1 si se trata de una imagen negativa. A continuación, guarda en un fichero de texto las características de cada una de las imágenes.

```
static void calculateFeaturesFromInput(const string& imageFilename, vector<float>& featureVector, HOGDescriptor& hog) {
    /** for imread flags from openCV documentation,
     * @see http://docs.opencv.org/modules/highgui/doc/reading_and_writing_images_and_video.html?highlight=imread#Mat imread(const string& filename, int flags)
     * @note If you get a compile-time error complaining about following line (esp. imread),
     * you either do not have a current openCV version (>2.0)
     * or the linking order is incorrect, try g++ -o openCVHogTrainer main.cpp `pkg-config --cflags --libs opencv`
     */
    Mat imageData = imread(imageFilename, CV_LOAD_IMAGE_GRAYSCALE);
    if (imageData.empty()) {
        featureVector.clear();
        printf("Error: HOG image '%s' is empty, features calculation skipped!\n", imageFilename.c_str());
        return;
    }
    // Check for mismatching dimensions
    if (imageData.cols != hog.winSize.width || imageData.rows != hog.winSize.height) {
        featureVector.clear();
        printf("Error: Image '%s' dimensions (%u x %u) do not match HOG window size (%u x %u)!\n", imageFilename.c_str(), imageData.cols, imageData.rows, hog.wi
        return;
    }
    vector<Point> locations;
    hog.compute(imageData, featureVector, winStride, trainingPadding, locations);
    imageData.release(); // Release the image again after features are extracted
}
```

ILUSTRACIÓN 25: FUNCIÓN DE EXTRACCIÓN DE CARACATERÍSTICAS HOG

Siguiendo en el **main()** y una vez extraída todas las características HOG de las imágenes procedemos al entrenamiento de ellas. Para el entrenamiento hemos optado a la librería de libSVM en vez de utilizar el entrenamiento que dispone OpenCV ya que consideramos que libSVM hace un entrenamiento más completo haciendo varias iteraciones y cogiendo los valores que mejor se adapten al clasificador.

```
// <editor-fold defaultstate="collapsed" desc="Pass features to machine learning algorithm">
/// Read in and train the calculated feature vectors
printf("Calling %s\n", libSVM::getInstance()->getSVMName());
libSVM::getInstance()->read_problem(const_cast<char*> (featuresFile.c_str()));
libSVM::getInstance()->train(); // Call the core libsvm training procedure
printf("Training done, saving model file!\n");
libSVM::getInstance()->saveModelToFile(svmModelFile);
// </editor-fold>

// <editor-fold defaultstate="collapsed" desc="Generate single detecting feature vector from calculated SVM support vectors and SVM model">
printf("Generating representative single HOG feature vector using svmlight!\n");
vector<float> descriptorVector;
vector<unsigned int> descriptorVectorIndices;
// Generate a single detecting feature vector (v1 | b) from the trained support vectors, for use e.g. with the HOG algorithm
libSVM::getInstance()->getSingleDetectingVector(descriptorVector, descriptorVectorIndices);
// And save the precious to file system
saveDescriptorVectorToFile(descriptorVector, descriptorVectorIndices, descriptorVectorFile);
// </editor-fold>

// <editor-fold defaultstate="collapsed" desc="Test detecting vector">
// Detector detection tolerance threshold
const double hitThreshold = libSVM::getInstance()->getThreshold();
// Set our custom detecting vector

hog.setSVMDetector(descriptorVector);

detectTrainingSetTest(hog, hitThreshold, positiveTrainingImages, negativeTrainingImages);
```

ILUSTRACIÓN 26: ENTRENAMIENTO DEL CLASIFICADOR

En la Ilustración 26 se ha procedido al entrenamiento del clasificador mediante la instancia **libSVM::getInstance()->train();** y una vez entrenado se procederá a guardar el vector de características. Para guardar el vector de características se ha tenido que crear una función aparte de la ya implementada ya que podíamos guardar el modelo mediante **libSVM::getInstance()->saveModelToFile(svmModelFile);** pero a la hora de cargar la instancia **libSVM::getInstance()->loadModelfromFile(String);** daba problemas como "*segmentation fault*" al llegar a una línea de código concreta. Para ello lo que hicimos fue recuperar el vector de características mediante **libSVM::getInstance()->getSingleDetectingVector(descriptorVector, descriptorVectorIndices);** y con la función **saveDescriptorVectorToFile(descriptorVector, descriptorVectorIndices, descriptorVectorFile);** lo guardamos en fichero.

El fichero que contiene el vector de características tiene el siguiente formato:

```
0.0899159 -0.0333578 -0.0164097 0.0370599 -0.0295663 0.0267991
0.00888763 -0.0268793 0.0264154 -0.0325328 -0.00737527
0.0125503 -0.0126974 -0.0440741 -0.0396101 0.0489135 0.0551996
0.0599696 0.0796968 -0.0202416 -0.0335551 0.0375344 -0.0141138
0.0230493 -0.0353518 0.0130616 0.0548612 0.0143717
0.0223019 -0.00217055 -0.0396749 -0.0246661 0.0128766
0.0268712 0.0226469 -0.0117978 0.010533 -0.0141599
0.0555111 -0.0233566 0.0452372 0.043008 0.0528208 -0.0292076
0.010522 -0.043909 0.0217003 0.0135777
0.01259 -0.0307032 -0.00914155 0.0479541 0.00095716 0.0395327
0.0409349 -0.0496584 -0.00405884 -0.00575015 -0.00292358
0.00551998 0.0424871 0.00526624 0.0254037 0.0246704
9.64688e-05 -0.0223747 0.044806 -0.0180505
0.0654439 -0.0052184 0.0420576
0.0155975 -0.0321479 -0.0624077 -0.00288592 -0.0168442
0.012038 -0.0187857 0.0453868 -0.0398265 -0.0161506 0.0396876
0.0413579 0.0822465 -0.00676709 -0.0509568 0.0278793
0.000894766 0.0713519
0.077834 -0.0281932 -0.0172583 -0.0347618 0.0205469 0.0621686
0.051901 0.0561466 -0.00798095 0.030118 0.0735436 -0.0218464
0.070394 -0.0118106 0.0129177 0.0227207 -0.00605897 -0.0412414
0.00378747 0.0476297 0.0605916 0.0767622 0.0722182
0.0107516 -0.00985472 -0.011407 0.0891046 0.0809168 0.0297684
0.000900553 -0.00839941 0.0214738 -0.0143464
0.0306425 -0.000173668 0.0366233 -0.00378718
0.0510071 -0.00141009 -0.00873994 0.0526703
0.0185671 -0.0153682 -0.00348669 0.00167198 0.0233884
0.0468566 0.0210467 -0.0147421 -0.0338843 -0.0204417 0.0707826
```

ILUSTRACIÓN 27: FICHERO DEL VECTOR DE CARACTERÍSTICAS

El clasificador comentado anteriormente ha sido hecho para las bicicletas que tienen un tamaño de 64x64. Este tamaño se lo hemos indicado en la sentencia de **hog.winsize() = Size(64,64)** al inicio del **main()**. Para el entrenamiento del clasificador basado en peatones tendríamos únicamente que cambiar el tamaño por 64x128 y las direcciones a los directorios de las imágenes positivas y negativas.

7.2. Desarrollo del programa de generación de recortes de falsos positivos

Para poder conseguir un mejor entrenamiento requerimos al "boostrap" o "boostrapping". Esto es entrenar al clasificador de manera normal y luego ponerle a clasificar imágenes negativas. Y ahí donde detectara que por características debería haber una bicicleta procederíamos a su recorte pues solo le estaríamos pasando imágenes negativas sin ninguna bicicleta en ella. Una vez recortados procederíamos a su entrenamiento de nuevo, pero con estos falsos positivos como imágenes negativas.

A continuación, explicamos detalladamente el algoritmo.

7.2.1. Descripción del código

Comenzamos con el **main()**

```
int main(int argc , char** argv){

    setDirectoriesAndExtensions();

    HOGDescriptor hog; // Use standard parameters here
    hog.winSize = Size(64,64); // Default training images size as used in paper

   //CARGO EL DESCRIPTOR VECTOR
    vector<float> descriptorVectorNuevo;
    float buffer;
    ifstream ifs ("/media/Secuencias/Alex/TFG/experiment/descriptorvectorLibSVM.dat",ios::binary);
    if(ifs.is_open()){
        cout<<"Abierto!"<<endl;
         while (ifs >> buffer)
              {
                  descriptorVectorNuevo.push_back(buffer);
              }
              ifs.close();
    }
    else cout<<"No se ha podido abrir"<<endl;



    hog.setSVMDetector(descriptorVectorNuevo);

    getFilesInDirectory(negSamplesDir, negativeTrainingImages, validExtensions);
    for(size_t i=0; i<negativeTrainingImages.size();i++){
        cout<<"Error 0"<<endl;

        Mat img = cv::imread(negativeTrainingImages[i]);//,CV_LOAD_IMAGE_GRAYSCALE
        if (img.cols == 0) {
             cout << "Error reading file " << negativeTrainingImages[i] << endl;
             return -1;
        }
        else
        {

        detectTest(hog,img,negativeTrainingImages[i]);
        }



    }
    return EXIT_SUCCESS;
}
```

ILUSTRACIÓN 28: MAIN PROGRAMA FALSEPOSITIVES.PRO

La metodología es la misma que en el programa anterior. En este programa se utiliza la función **setDirectoriesAndExtensions();** que es la que les indica los directorios de las

imágenes y las extensiones que vamos a utilizar. En el anterior programa las sentencias de la función anterior se encontraban en el **main().**

Declaramos un vector de tipo float para almacenar el vector de características y abrimos el archivo y lo guardamos ahí. Una vez cargado el vector de características se lo incluimos al HOG mediante **hog.setSVMDetector(descriptorVectorNuevo);.** Con el vector incluido en el HOG comenzamos a meter los negativos en un vector de strings como hicimos en el anterior programa con **getFilesInDirectory(negSamplesDir, negativeTrainingImages, validExtensions);**. En la sentencia del for empezamos a leerlo uno a uno y para cada imagen llamamos a la función **detectTest();** que será la encargada de detectar dónde están los falsos positivos y recortarnos los recuadros.

```
static void detectTest(const HOGDescriptor& hog,  Mat& imageData, String& nombre) {

//    img - Source image. See HOGDescriptor::detect() for type limitations.
//    found_locations - Detected objects boundaries.
//    hit_threshold - Threshold for the distance between features and SVM classifying plane. See gpu::HOGDescriptor::detect() for details.
//    win_stride - Window stride. It must be a multiple of block stride.
//    padding - Mock parameter to keep the CPU interface compatibility. It must be (0,0). (esto es solo para GPU)
//    scale0 - Coefficient of the detection window increase.
//    group_threshold - Coefficient to regulate the similarity threshold. When detected, some objects can be covered by many rectangles. 0 means not to perform

    vector<Rect> found;
    int groupThreshold = 2;
    Size padding(Size(32, 32));
    Size winStride(Size(8, 8));
    double hitThreshold = 4.15; // tolerance
    hog.detectMultiScale(imageData, found, hitThreshold, winStride, padding, 1.05, groupThreshold);


    //RECORTES DE CUADROS
    if(found.size()!=0){
        for(int i=0;i<found.size();i++){
           if ((found[i].x >0) && (found[i].y >0) &&
                (found[i].width >0) && (found[i].height >0) &&
                ((found[i].width + found[i].x) < imageData.cols ) &&
                ((found[i].height + found[i].y) < imageData.rows ))
            {
            cv::Point a=found[i].tl();//extremo superior izquierdo
            cv::Point b = found[i].br();//extremo inferiorderecho
            cout<<"extremo superior izquierdo"<<a<<endl;
            cout<<"extremo inferior derecho"<<b<<endl;
            cv::Point c=cv::Point((a.x+b.x)/2,(a.y+b.y)/2);
             cout<<"centro"<<c<<endl;
            cout<<"Antiguo extremo superior izquierdo"<< found[i].tl()<<endl;
            cout<<"Antiguo extremo inferior derecho"<< found[i].br()<<endl;
            found[i].tl()=cv::Point(c.x-(b.y-a.y)/4,a.y);
            found[i].br()=cv::Point(c.x+(b.y-a.y)/4,b.y);
            cout<<"nuevo extremo superior izquierdo"<< found[i].tl()<<endl;
            cout<<"nuevo extremo inferior derecho"<< found[i].br()<<endl;
            cv::Mat newimg = imageData(found[i]);
            ostringstream archivo;
            bf::path filepath(nombre);
            cout<<"Basename"<<basename(filepath.string().c_str())<<endl;
            archivo<<"/media/Secuencias/Alex/Falsospositivos/"<<i<<basename(filepath.string().c_str());
            string newfileToSave=archivo.str().c_str();
            cout << "Guardo fichero  " << newfileToSave<< endl;
            cv::imwrite(newfileToSave,newimg);
            }
        }
    }
}
```

ILUSTRACIÓN 29: FUNCIÓN DE MODIFICACIÓN DE RECTÁNGULOS

En la función **detectTest();** que mostramos en la Ilustración 29 le hemos pasado parámetros como el HOG, la imagen correspondiente al vector de strings y el nombre de la imagen. En la función definimos varios parámetros y llamamos a la función **hog.detectMultiScale(imageData, found, hitThreshold, winStride, padding, 1.05, groupThreshold);** que con los parámetros que le hemos pasado se encargará de encontrar esas detenciones que el programa creerá que se trata de detecciones verdaderas cuando no lo son (falsos positivos). La función rellena un vector de Rect (rectángulos de las detecciones) que comenzamos a recorrer y si esos rectángulos se encuentran dentro de la imagen completamente comenzamos a manipularlos. Para ello cogemos las coordenadas de la esquina superior izquierda **tl()** (top left) y la esquina inferior derecha **br()** (botom right). Estas coordenadas las modificamos de tal manera que queden el doble de alto que de ancho para que nos quede una imagen del tamaño que sea, pero con esa proporción. Esta función solo será válida con los peatones ya que tienen un tamaño de 64x128, para las bicicletas habría que modificar la sentencia de cambio de coordenadas. Al haber modificado ya las coordenadas de las esquinas renombramos el archivo con la librería boost cogiendo el

nombre del archivo y añadiéndole un contador para indicar el número de recorte de la imagen y lo guardamos en un directorio diferente. Una vez sacadas todos los recortes de los falsos positivos únicamente quedaría redimensionarlas para tener el mismo tamaño que los demás positivos (64x128) (peatones). Para redimensionarlas nos metemos en la terminal de Linux y ejecutamos la siguiente sentencia

[**for i in *.png; do convert $i.png –resize 64x128! /media/Secuencias/Alex/Resizes/nuevo$i ; done**]

de tal manera que quedarían ya todas las imágenes guardadas con el programa redimensionadas en el directorio indicado.

7.3. Desarrollo del programa de detección

El siguiente y último tiene como función principal mostrar las detecciones de los elementos de interés, entre otras muchas funciones. El **main()** correspondiente a dicho algoritmo se encuentra comentado la mayor parte de él, esto se debe a que se trata de una manera de simplificar los procesos, de modo que si se desea ejecutar otra función sólo se tiene que descomentar dicha parte de código y comentar la anterior para que esto no se ejecute. Cada parte del **main()** está bien diferenciada con comentarios en azul que indican el tipo de función que se lleva a cabo.

7.3.1. Descripción del código

Como en los programas anteriores se explicará detalladamente las partes de códigos que a priori puedan parecer más confusas.

```
int main(int argc, char** argv) {


    static vector<string> imagenes;
    static vector<string> ficheros;
    static vector<string> validExtensions;
    validExtensions.push_back("jpg");
    validExtensions.push_back("png");
    validExtensions.push_back("ppm");
    validExtensions.push_back("txt");

    ///PARA GENERAR LOS ARCHIVOS DE ANOTACIONES O VER RESULTADOS DE IMAGENES
//    getFilesInDirectory(direccionimagenes,imagenes, validExtensions);
//    String dir ="/home/afernandez/Resultados/descriptorvectorBicis.dat";
//    HOGDescriptor hog;
//    hog.winSize=Size(64,64);
//    vector<float> descriptorVectorNuevo=cargavector(dir);//Para cargar el vector
//    hog.setSVMDetector(descriptorVectorNuevo);
//    ///EL VECTOR DE BICIS TIENE 1764 DATOS
//    for(int i=0; i<imagenes.size();i++){
//        Mat img = imread(imagenes[i]);//, CV_LOAD_IMAGE_GRAYSCALE);
//        vector<Rect>  found = detectTest(hog,4.15,img);
//        ostringstream archivo;
//        bf::path filepath(imagenes[i]);
//        bf::path p = bf::path(filepath.stem());//stem() para cogerlo sin extension y filename() con extension
//        archivo<<"/media/Secuencias/Alex/Bicicletas/Textpredictions/"<<p.c_str()<<".txt";
//        string newfileToSave=archivo.str().c_str();
//        fstream ficherosalida;
//        ficherosalida.open(newfileToSave, ios::out);
//        for(int i = 0; i<found.size();i++){
//            double a= (found[i].x+found[i].width/2);///COORD X DEL CENTRO
//            double b= (found[i].y+found[i].height/2);///COORD Y DEL CENTRO
//            ficherosalida<<a<<" "<<b<<" "<<found[i].width<<" "<<found[i].height<<endl;
//        }
//        ficherosalida.flush();
//        ficherosalida.close();
//    }
```

ILUSTRACIÓN 30: MAIN PROGRAMA RECT.PRO (1)

La primera parte del **main()** se compone con la declaración de los vectores que serán los encargados de recoger las direcciones de los archivos y el vector **validExtension** que será el

encargado de almacenar las extensiones válidas para los archivos. Guardamos las direcciones de los archivos como en los programas anteriores con **getFilesInDirectory();**. Una vez guardada las direcciones definimos el tamaño del HOG y cargamos el vector de características, y comenzamos a leer las imágenes. Estas funciones estaban concebidas para ver únicamente si recuadraba bien las detecciones, pero al poseer unos documentos de texto con la posición especifica con la posición de los recortes decidimos guardar unos ficheros con el mismo formato para poder comprobar si estas detecciones eran o no válidas. Para guardarlo lo que hicimos fue con el vector de Rect que nos devolvía la función **detectTest();** recorrerlo y comenzar a guardarlo en el fichero de texto. El vector Rect que nos devuelve la función corresponde a las localizaciones de los rectángulos de las detecciones. El fichero de texto tendrá el siguiente formato

[coordenada x del centro de la imagen; coordenada y del centro de la imagen; ancho; alto].

Por lo que nos quedaría un fichero tal que así.

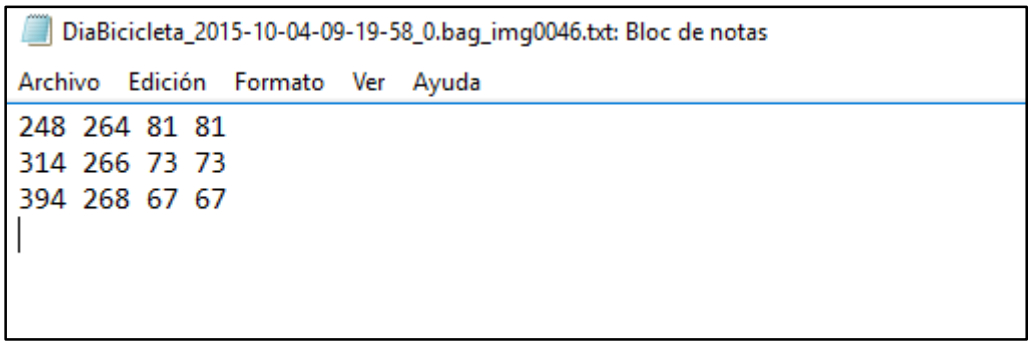

ILUSTRACIÓN 31: RECUADROS DE DETECCIONES

Guardado con el nombre correspondiente a la imagen.

Si lo que queremos es guardar los recuadros que detecte el HOG tendríamos que irnos a la siguiente función del **main().**

```
    ///PARA VER Y GUARDAR LOS RECUADROS
//   getFilesInDirectory(direccionimagenes,imagenes, validExtensions);
//   String dir ="/home/afernandez/Resultados/descriptorvectorBicis.dat";
//   HOGDescriptor hog;
//   hog.winSize=Size(64,64);
//   vector<float> descriptorVectorNuevo=cargavector(dir);//Para cargar el vector
//   hog.setSVMDetector(descriptorVectorNuevo);
//   ///EL VECTOR DE BICIS TIENE 1764 DATOS
//   for(int i=0; i<imagenes.size();i++){
//       Mat img = imread(imagenes[i]);//, CV_LOAD_IMAGE_GRAYSCALE);
//       bf::path filepath(imagenes[i]);
//       bf::path p = bf::path(filepath.filename());//stem() para cogerlo sin extension y filename() con extension
//       ostringstream archivo;
//       archivo<<"/media/Secuencias/Alex/Bicicletas/Comparison/"<<p.c_str();
//       string newfileToSave=archivo.str().c_str();
//       vector<Rect>  found = detectTest(hog,4.15,img);
////       imshow(imagenes[i],img);
////       waitKey(0);
//       imwrite(newfileToSave,img);
```

ILUSTRACIÓN 32: MAIN DEL PROGRAMA RECT.PRO (2)

La cual se encarga de recorrer el vector del directorio de las imágenes e ir analizándolo una a una y guardarlas con los recuadros ya sobre la imagen en el directorio indicado. La función **detecTest()**, como hemos comentado antes, será la encargada de detectar las bicicletas, en este caso, y recuadrarlas llamando a una función dentro de esta función

llamada **showDetections();**. Por lo que nos quedaría en el directorio indicados una tira de imágenes como la siguiente.

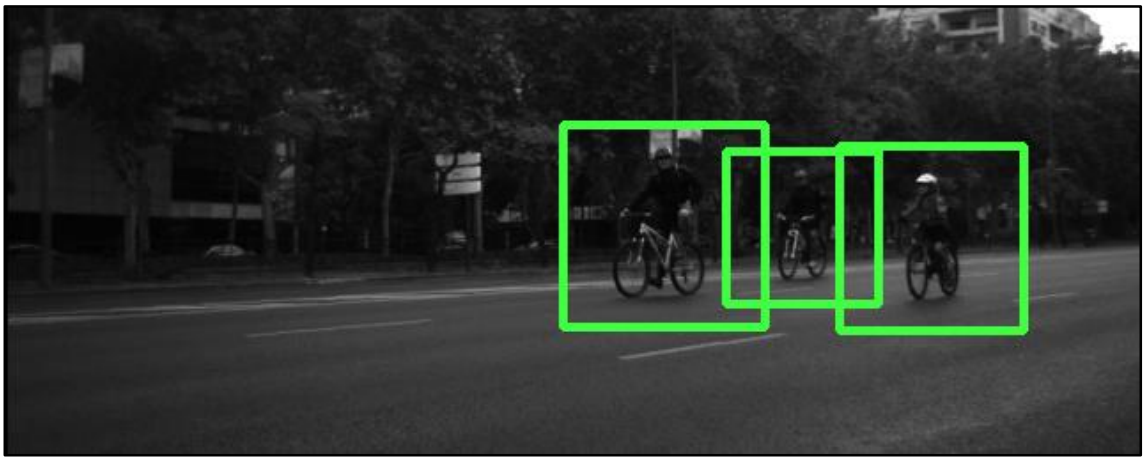

ILUSTRACIÓN 33: DETECCIONES CICLISTAS

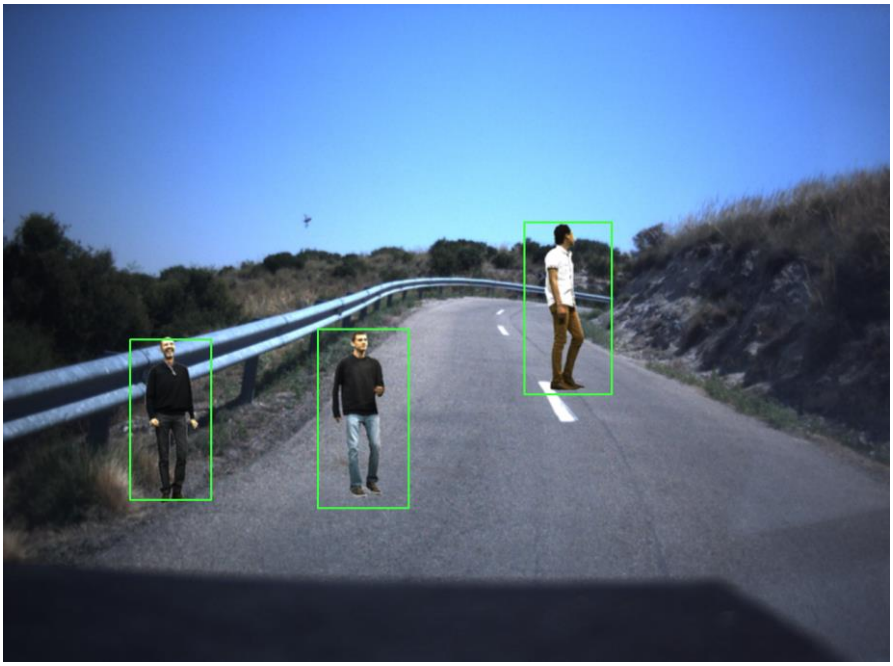

ILUSTRACIÓN 34: DETECCIONES PEATONES

En la Ilustración 33 e Ilustración 34 vemos las detecciones hechas por el HOG..

La siguiente función a comentar será la encargada de comparar los ficheros elaborados anteriormente con las anotaciones de la situación real de las bicicletas.

```
    ///PARA COMPARAR CON ANNOTATIONS
    /// OJO!!AHORA SE GUARDAN COMO tl(),width,height. NO COMO x(centro), y(centro),width, height
//    getFilesInDirectory(direccionficherosannotations,ficheros, validExtensions);
//    comparison(ficheros);
```

ILUSTRACIÓN 35 MAIN DEL PROGRAMA RECT.PRO (3)

En la Ilustración 35 lo primero que hace es guardar la dirección de los ficheros en el vector de strings y llamar a la función **comparison()** pasándole el vector.

```
void comparison (vector<String>& fichero){

    vector<Rect> annotations, predictions;
    vector<double> b;
    String cadena;
    vector<string> linea;

    for(int i=0;i<fichero.size();i++){//fichero e imagenes tienen el mismo tamaño y nombre

        bf::path filepath(fichero[i]);
        bf::path p = bf::path(filepath.filename());//stem() para cogerlo sin extension
        string newfile("/media/Secuencias/Alex/Bicicletas/Textpredictions/");
        string newfile1(p.c_str());
         newfile= newfile + newfile1;

        ///RECTANGULOS DE LOS FICHEROS
        ifstream ar(fichero[i], ios::binary);///FICHERO ANNOTATIONS
            if(ar.is_open()){//Abro fichero y leo las anotaciones
                cout<<" abierto annotations "<<fichero[i]<<endl;
               while(getline(ar,cadena)){//Mientras haya una linea por leer
                   boost::split(linea,cadena, boost::is_any_of(" "));//separo en los espacios
                   for(int j=0;j<linea.size()-1;j++){//desde 0 hasta el tamaño -1 (quito el ultimo 1 que hay)
                        b.push_back(atof(linea[j].c_str()));//Los meto en un vector de doubles
                   }

                   cv::Point tl=cv::Point(((b[0]-b[2]/2)),(b[1]-(b[3]/2)));
                   cv::Point br=cv::Point(((b[0]+b[2]/2)),(b[1]+(b[3]/2)));
                   Rect re(tl,br);
                   annotations.push_back(re);//meto en el vector los rectangulos de un fichero
                   b.clear();//Limpio el vector de doubles
                   linea.clear();//Limpio el vector de strings
                   }
               ar.close();//Cierro fichero
        }
```

ILUSTRACIÓN 36: FUNCIÓN COMPARISON

La primera parte de la función comparison es la encargada de recorrer el directorio de los archivos con las anotaciones, en este caso se trata de los ficheros con las anotaciones de los recortes, las anotaciones donde sabemos con certeza que se encuentran las bicicletas. Lo primero que hacemos es leer el fichero uno a uno y guardar las líneas en un string, recorremos el string y buscamos los lugares donde haya un espacio en blanco para poder separarlo mediante la sentencia Split perteneciente a la biblioteca de boost y guardarlo en un vector de strings. Una vez guardado en ese vector procedemos a guardarlo de nuevo en otro vector, pero esta vez de doubles. Más adelante declaramos dos variables de tipo **cv::Point** a la cual les metemos como valores la esquina superior izquierda (**tl()**) y la esquina inferior derecha (**br()**) a las que las meteremos en el constructor de rect, para ello habrá que hacer una serie de cálculos para determinar esas esquinas, para proceder tras esto a meterlo en un vector de rects. Y así tendríamos almacenadas las coordenadas de los archivos en un vector dentro del programa. Una vez guardadas estas coordenadas correspondiente a una imagen en concreto procedemos a hacer lo mismo para la misma imagen pero con el archivo de las detecciones.

```
    ifstream ar1(newfile, ios::binary);///FICHERO PREDICTIONS CREADO POR TI
        if(ar1.is_open()){//Abro fichero y leo las anotaciones
            cout<<"abierto detected "<<newfile<<endl;
           while(getline(ar1,cadena)){//Mientras haya una linea por leer
               if(cadena==" ")break;
               boost::split(linea,cadena, boost::is_any_of(" "));//separo en los espacios
               for(int j=0;j<linea.size();j++){///NO NECESITA -1!! LO HAS CREADO SIN EL ULTIMO 1
                    b.push_back(atof(linea[j].c_str()));//Los meto en un vector de doubles
               }
               // Rect re1(b[0],b[1],b[2],b[3]);//Defino el nuevo rectangulo con las dimensiones que he sacado(X,Y, Width, Height)
               ///LO DE ABAJO SOLO PARA ANNOTATIONS
               cv::Point tl=cv::Point(((b[0]-b[2]/2)),(b[1]-(b[3]/2)));
               cv::Point br=cv::Point(((b[0]+b[2]/2)),(b[1]+(b[3]/2)));
               Rect re1(tl,br);
               predictions.push_back(re1);//meto en el vector los rectangulos de un fichero
               b.clear();//Limpio el vector de doubles
               linea.clear();//Limpio el vector de strings

               }
           ar1.close();//Cierro fichero
    }
```

ILUSTRACIÓN 37: LECTURA DE FICHEROS DE RECTÁNGULOS

Los pasos a seguir son exactamente los mismos que en el apartado anterior pero únicamente modificando la ruta del archivo que hemos realizado anteriormente. Esta modificación es muy sencilla ya que, a ambos archivos, el de los recortes y el de las detecciones, los hemos llamado de la misma manera, pero los hemos almacenados en directorios diferentes.

Al tener ya para una imagen en concreto los dos vectores de rect, el de los recortes y las detecciones, procedemos a compararlos entre sí con la siguiente función.

```
            ostringstream archivo1;
            archivo1<<"/media/Secuencias/Alex/Bicicletas/Textcomparison/"<<p.c_str();
            p=bf::path (p.stem());
            string newtextfile=archivo1.str().c_str();
            fstream ficherosalida;
            ficherosalida.open(newtextfile, ios::out);
            if(ficherosalida.is_open()){
             for(int k=0;k<annotations.size();k++){
                 if(predictions.size()==0){
                     ficherosalida<<p.c_str()<<";"<<k<<";"<<annotations[k].x<<";"<<annotations[k].y<<";"<<annotations[k].width
                                 <<";"<<annotations[k].height<<";"<<"-1"<<";"<<"-1"<<";"<<"-1"<<";"<<"-1"<<";"<<"-1"<<";"<<"-1"<<endl;
                 }
                 else {
                    for(int l=0;l<predictions.size();l++){
                        Rect r1=(annotations[k] & predictions[l]);
                        Rect r2=(annotations[k] | predictions[l]);
                        float a = (float)r1.area()/(float)r2.area();
                        ficherosalida<<p.c_str()<<";"<<k<<";"<<annotations[k].x<<";"<<annotations[k].y<<";"
                                     <<annotations[k].width<<";"<<annotations[k].height<<";"<<l<<";"<<predictions[l].x
                                    <<";"<<predictions[l].y<<";"<<predictions[l].width<<";"<<predictions[l].height<<";"<<a<<endl;
                    }
                 }
                }
            }

        ficherosalida.flush();
        ficherosalida.close();
    annotations.clear();
    predictions.clear();
    }
}
```

ILUSTRACIÓN 38: COMPARACIÓN DE RECTÁNGULOS

En la Ilustración 38 se procede a la comparación de los dos vectores de rects almacenados. Para ello se procede a la comparación entre todos los elementos de los vectores para posteriormente guardarlos a un fichero de texto y analizar sus datos. El formato de los ficheros será el siguiente:

[nombre del archivo; id_1; tl ()$_1$; $width_1$; $heigth_1$; id_2; tl ()$_2$; $width_2$; $height_2$; intersección/unión]

Siendo id1 y id2 la posición correspondiente al vector que estamos analizando y el ultimo campo (intersección/unión) una relación entre los triángulos analizados para ver cuanta semejanza tiene lo detectado con lo que verdaderamente es.

Los ficheros de texto, por lo tanto, quedarían de la siguiente manera

```
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0307;0;343;206;56;91;0;311;194;125;125;0.326144
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0307;0;343;206;56;91;1;483;203;117;117;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0307;0;343;206;56;91;2;430;217;86;86;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0307;1;454;220;35;69;0;311;194;125;125;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0307;1;454;220;35;69;1;483;203;117;117;0.024236
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0307;1;454;220;35;69;2;430;217;86;86;0.326528
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0307;2;513;225;51;77;0;311;194;125;125;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0307;2;513;225;51;77;1;483;203;117;117;0.286873
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0307;2;513;225;51;77;2;430;217;86;86;0.0200451
```

ILUSTRACIÓN 39: FICHERO DE COMPARACIONES

Como analizar los datos así resultaba muy difícil elaboramos el siguiente programa, aquí mostramos la parte correspondiente al **main().**

```
    ///PARA RECUADRAR DESDE ANNOTATIONS
//    getFilesInDirectory(direccionficherosannotations,ficheros, validExtensions);
//    recuadraannotations(ficheros);
```

ILUSTRACIÓN 40: MAIN DEL PROGRAMA RECT.PRO (4)

Vemos que llama a la función **recuadraannotations();** que es la encargada de coger todos los ficheros con los datos de los rectángulos y representarlos en la imagen.

```
void recuadraannotations(vector<String>& fichero){
    vector<Rect> annotations;
        vector<double> b;
        String cadena;
        vector<string> linea;

        for(int i=0;i<fichero.size();i++){//fichero e imagenes tienen el mismo tamaño y nombre

            bf::path filepath(fichero[i]);
            bf::path p = bf::path(filepath.stem());//stem() para cogerlo sin extension
            string newfile("/media/Secuencias/Alex/Bicicletas/Imagescomparison/");///DESDE DONDE COGE LAS IMAGENES
            string newfile1(p.c_str());
            string ext(".png");
             newfile= newfile + newfile1+ext;

            ///RECTANGULOS DE LOS FICHEROS
            ifstream ar(fichero[i], ios::binary);//Fichero de anotaciones cambiar el nombre
                if(ar.is_open()){//Abro fichero y leo las anotaciones
                    cout<<" Abierto annotations "<<fichero[i]<<endl;
                   while(getline(ar,cadena)){//Mientras haya una linea por leer
                       boost::split(linea,cadena, boost::is_any_of(" "));//separo en los espacios
                       for(int j=0;j<linea.size();j++){//desde 0 hasta el tamaño -1 (quito el ultimo 1 que hay)
                            b.push_back(atof(linea[j].c_str()));//Los meto en un vector de doubles
                       }
                       ///OJO, FICHERO ANNOTATIONS SE LEE CON LO DE ABAJO
                       cv::Point tl=cv::Point(((b[0]-b[2]/2)),(b[1]-(b[3]/2)));
                       cv::Point br=cv::Point(((b[0]+b[2]/2)),(b[1]+(b[3]/2)));
                       Rect re(tl,br);
                       ///OJO, FICHERO DETECTED(CREADO POR TI) SE LEE CON LO DE ABAJO
//                       Rect re(b[0],b[1],b[2],b[3]);//Defino el nuevo rectangulo con las dimensiones que he sacado(X,Y, Width, Height)
                       annotations.push_back(re);//meto en el vector los rectangulos de un fichero
                       b.clear();//Limpio el vector de doubles
                       linea.clear();//Limpio el vector de strings
                       }
                   ar.close();//Cierro fichero
            }
                cout<<newfile<<endl;
                Mat img =imread(newfile);
                showDetections(annotations, img);
                ostringstream archivo;
                archivo<<"/media/Secuencias/Alex/Bicicletas/Imagescomparison/"<<p.c_str();///DONDE LAS VA A GUARDAR
                string newfileToSave=archivo.str().c_str();
                newfileToSave=newfileToSave+ext;
                imwrite(newfileToSave,img);
                annotations.clear();
        }
}
```

ILUSTRACIÓN 41: EXTRACCIÓN Y REPRESENTACIÓN DE LOS DATOS DE LOS FICHEROS

Siguiendo la misma metodología que hemos utilizado para leer los ficheros comenzamos a guardar en vectores los recuadros y representarlos. Esta función se ha de realizar dos veces, la primera para que coja los vectores de los recortes y los represente y la segunda para que coja los recortes de las detecciones y los represente en la imagen que ya había representado anteriormente los recuadros de los recortes. Por lo que nos quedaría una imagen como la siguiente mostrada.

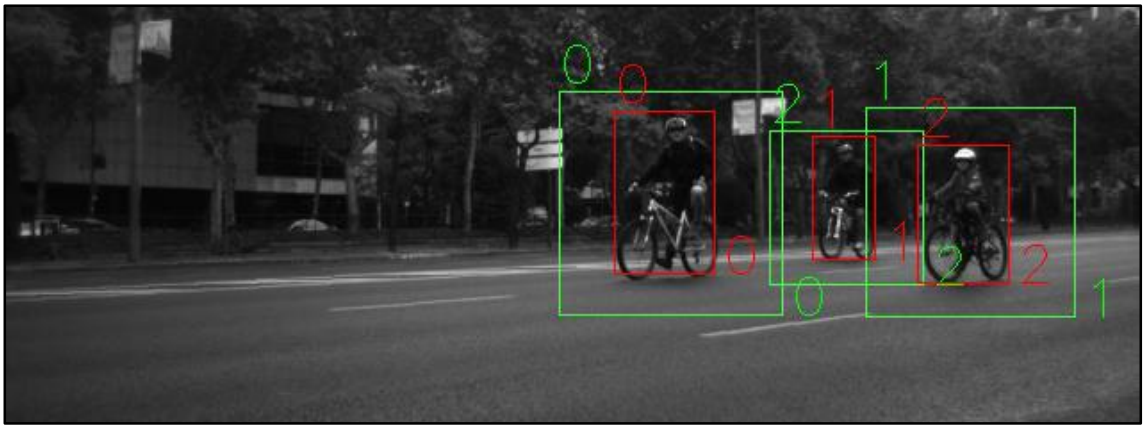

ILUSTRACIÓN 42: REPRESENTACIÓN DE LOS DATOS DEL FICHERO

En la cual en la Ilustración 42 muestra las detecciones hechas (recuadros verdes) y los recuadros de los recortes (recuadros rojos). El número de arriba de los recuadros no es significativo, únicamente nos indica el número de rectángulo empleado para poder luego compararlo con el fichero creado anteriormente.

Se puede dar el caso de que haya algún cuadro que no tenga anotación como el mostrado en la siguiente imagen.

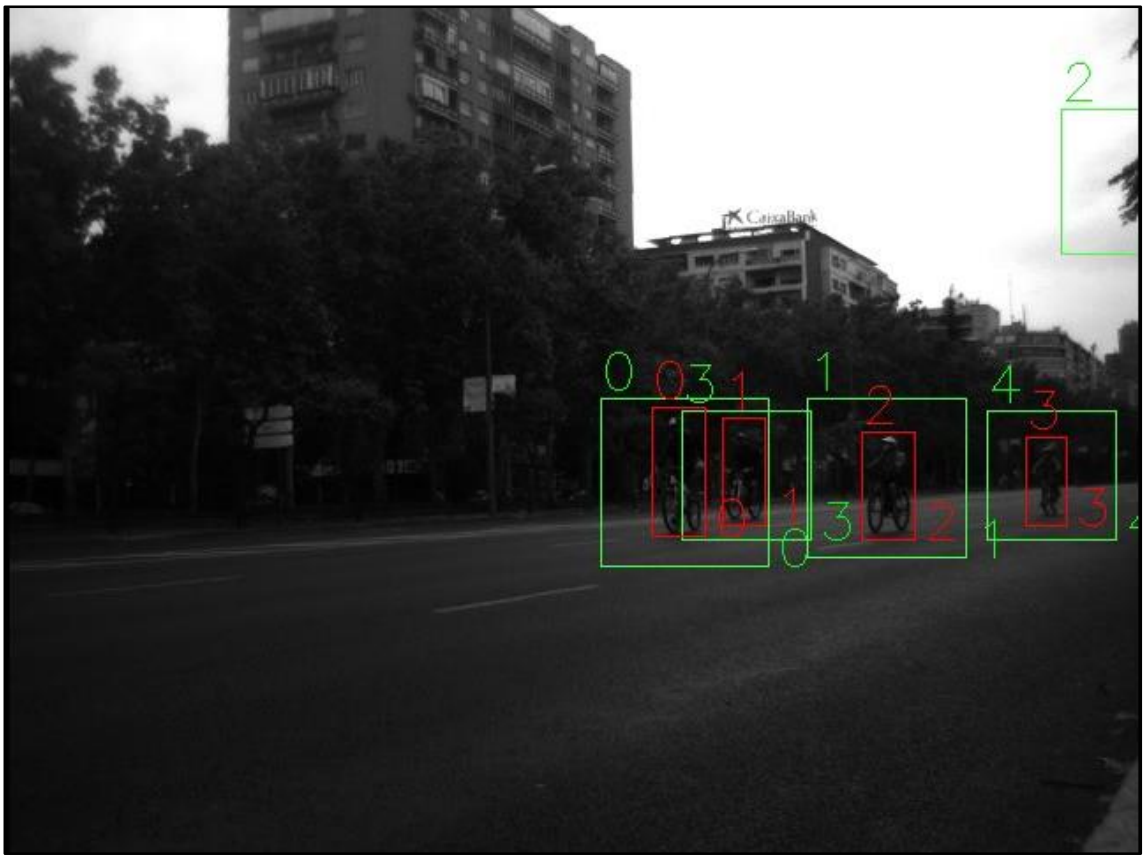

ILUSTRACIÓN 43: REPRESENTACIÓN DE LOS DATOS DEL FICHERO CON UN FALSO POSITIVO

La Ilustración 43 corresponderá con un falso positivo. La manera de ver si existe un falso positivo en los archivos de texto es ver si todas las detecciones están emparejadas. La que

no esté emparejada significará que se trata de un falso positivo. Como muestra la siguiente imagen

```
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0281;0;364;226;30;73;0;334;220;95;95;0.242659
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0281;0;364;226;30;73;1;452;221;90;90;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0281;0;364;226;30;73;2;596;57;82;82;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0281;0;364;226;30;73;3;380;227;73;73;0.153052
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0281;0;364;226;30;73;4;553;227;73;73;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0281;1;404;232;24;61;0;334;220;95;95;0.162216
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0281;1;404;232;24;61;1;452;221;90;90;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0281;1;404;232;24;61;2;596;57;82;82;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0281;1;404;232;24;61;3;380;227;73;73;0.274723
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0281;1;404;232;24;61;4;553;227;73;73;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0281;2;483;240;30;61;0;334;220;95;95;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0281;2;483;240;30;61;1;452;221;90;90;0.225926
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0281;2;483;240;30;61;2;596;57;82;82;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0281;2;483;240;30;61;3;380;227;73;73;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0281;2;483;240;30;61;4;553;227;73;73;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0281;3;576;243;23;50;0;334;220;95;95;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0281;3;576;243;23;50;1;452;221;90;90;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0281;3;576;243;23;50;2;596;57;82;82;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0281;3;576;243;23;50;3;380;227;73;73;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0281;3;576;243;23;50;4;553;227;73;73;0.2158
```

ILUSTRACIÓN 44: FICHERO DE COMPARACIONES CON UN FALSO POSITIVO

Siendo el falso positivo el único recuadro en todas las combinaciones posibles que no tiene ningún valor en la relación entre recuadros

Pero ir analizando cada archivo de texto a mano es muy tedioso por lo que decidimos elaborar la siguiente función que se encargaba de ello.

```
   ///PARA SACAR LOS VERDADEROS POSITIVOS
  getFilesInDirectory(direccioncomparison,ficheros,validExtensions);
  falsepositive(ficheros);

  cout<<"Saliendo"<<endl;
    return EXIT_SUCCESS;
}
```

ILUSTRACIÓN 45: MAIN DEL PROGRAMA RECT.PRO (5)

La función de la Ilustración 45 corresponde al **main()**. A continuación, se explica detalladamente de que se trata.

```
void falsepositive(vector<String> &fichero){

///EN ESTA FUNCION SE LEE ASI
///Rect re(b[0],b[1],b[2],b[3]);//Defino el nuevo rectangulo con las dimensiones que he sacado(tl(), Width, Height)
    vector<double> b;
    String cadena;
    vector<string> linea;
    vector<vector<double>> relaciones;
    fstream archivo,estadisticas;

    for(int i=0;i<fichero.size();i++){//fichero e imagenes tienen el mismo tamaño y nombre
        bf::path filepath(fichero[i]);
        bf::path p = bf::path(filepath.filename());//stem() para cogerlo sin extension
        string newfile("/media/Secuencias/Alex/Bicicletas/Textdeteccionesverdaderas/");
        string newfile1(p.c_str());
         newfile= newfile + newfile1;
         string stat("/media/Secuencias/Alex/Bicicletas/Textstatistics/");
         stat=stat+newfile1;


        ifstream ar(fichero[i], ios::binary);///FICHERO ANNOTATIONS
            if(ar.is_open()){//Abro fichero y leo las anotaciones
                cout<<" abierto annotations "<<fichero[i]<<endl;
               while(getline(ar,cadena)){//Mientras haya una linea por leer
                   boost::split(linea,cadena, boost::is_any_of(";"));//Separo en las comas
                   b.push_back(atof(linea[1].c_str()));
                   b.push_back(atof(linea[6].c_str()));
                   b.push_back(atof(linea[11].c_str()));///HE METIDO EN EL VECTOR DE DOUBLES EL NUMERO DE ANOTACION, NUMERO DE DETECCION Y LA RELACION
                   relaciones.push_back(b);
                   b.clear();//Limpio el vector de doubles
                   linea.clear();//Limpio el vector de strings
                   }
               ar.close();//Cierro fichero
        }
```

ILUSTRACIÓN 46: EXTRACCIÓN DE DATOS

La función se encarga de recoger los datos del archivo de comparación que hemos creado la cual meterá en un vector de doubles únicamente los valores que nos interesan de cada archivo que son el índice de cada detección y recorte y la relación que tienen estos rectángulos. Como únicamente nos interesan los valores de relación que tengan el valor más alto comenzamos a quedarnos únicamente con los mejores con la siguiente sentencia.

```
vector<double> best_value;
vector<int> best_index;
int num_recortes = relaciones[relaciones.size() - 1][0] + 1;//Establezco el numero maximo de recortes
int num_detecciones =relaciones[relaciones.size() -1][1] + 1;
int true_pos=0;
int false_pos=0;
for (int j = 0; j < num_recortes; ++j){
    best_value.push_back(-1);
    best_index.push_back(-1);
}
for(int j=0;j<relaciones.size();j++){
    int recorte_index = relaciones[j][0];
    if ( relaciones[j][2] > best_value[recorte_index]){
        best_value[recorte_index] = relaciones[j][2];//Comparo para los recortes las mejores detecciones
        best_index[recorte_index] = relaciones[j][1];//Comparo para los recortes las mejores detecciones
    }

}///AQUI YA ESTA EL VECTOR RELLENO
```

ILUSTRACIÓN 47: RECOPILACIÓN DE LAS RELACIONES MÁS ALTAS

En la Ilustración 46 muestra cómo se recorre el vector creado y se queda con las mejores relaciones correspondiente a cada id de los recortes. Pero se podía dar el caso de que para dos recortes diferentes la mayor relación fuera para una misma detección como en el siguiente caso.

ILUSTRACIÓN 48: DETECCIÓN CON DOS RECORTES

Que nos daba como archivo el siguiente

```
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0048;0;202;233;45;62;0;188;229;75;75;0.496
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0048;0;202;233;45;62;1;257;228;76;76;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0048;0;202;233;45;62;2;340;230;75;75;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0048;1;269;237;41;56;0;188;229;75;75;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0048;1;269;237;41;56;1;257;228;76;76;0.397507
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0048;1;269;237;41;56;2;340;230;75;75;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0048;2;356;242;37;56;0;188;229;75;75;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0048;2;356;242;37;56;1;257;228;76;76;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0048;2;356;242;37;56;2;340;230;75;75;0.368356
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0048;3;388;250;26;40;0;188;229;75;75;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0048;3;388;250;26;40;1;257;228;76;76;0
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0048;3;388;250;26;40;2;340;230;75;75;0.184889
```

ILUSTRACIÓN 49: FICHERO DE DETECCIÓN CON DOS RECORTES

Por lo que según el código elaborado se quedaría para el recorte 0 y la detección 0 el valor de 0.49, para el recorte 1 y la detección 1 el valor de 0.397507, para el recorte 2 y la detección 2 el valor de 0.368356 y por ultimo para el recorte 3 con la detección 3 el valor de

0.184889 por lo que vemos que se hubiera quedado dos diferentes recortes con una misma detección.

```
int k=0;
do{
    for(int j=k+1;j<best_value.size();j++){
        if(best_index[k]==best_index[j] && best_value[k]>0){
            if(best_value[k]>best_value[j]) best_value[j]=0;
            else best_value[k]=0;
        }
    }k++;
}
while(k<best_value.size());
int comprobacion=0;
for(int j=0;j<best_value.size();j++){
if(best_value[j]>0) comprobacion++;}
```

ILUSTRACIÓN 50: ELECCIÓN DE LA MEJOR RELACIÓN

Así que elaboramos el siguiente bucle que miraba para una misma detección los valores que tenía y ponía el menor valor a 0.

Una vez que tenemos ya los datos que queremos procedemos a imprimirlos a fichero, en este caso se imprimen dos ficheros a la vez. El primero de ellos para la comprobación de los datos como el siguiente,

```
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0048.txt4;3
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0048.txt;0;0;0.496
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0048.txt;1;1;0.397507
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0048.txt;2;2;0.368356
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0048.txt;3;2;0
```

ILUSTRACIÓN 51:FICHERO CON LOS RECORTES Y DETECCIONES

en el que aparecen en la primera línea el número de recortes con el número de detecciones y en el resto de líneas el índice del recorte con la detección correspondiente. Como vemos el recorte 3 con la detección 2 se ha puesto a 0.

En el segundo fichero se han imprimido los datos para las posteriores estadísticas de la siguiente manera,

```
        cout<<fichero[i]<<endl;

        estadisticas.open(stat,ios::out);
        archivo.open(newfile,ios::out);

        archivo<<p.c_str()<<num_recortes<<";"<<num_detecciones<<endl;

    for(int j=0;j<best_value.size();j++){

        archivo<<p.c_str()<<";"<<j<<";"<<best_index[j]<<";"<<best_value[j]<<endl;

        if(best_value[j]>0){//Si es positivo
            if(best_value[j]<0.15 ){
                false_pos++;
            }//Si el mejor valor esta por debajo de 0.15 lo considero falso positivo
            if(best_value[j]>0.15){
                true_pos++;
            }//Si esta por encima lo considero positivo detectado
        }
        cout<<"mejores detecciones "<<j<<" "<<best_index[j]<<" "<<best_value[j]<<endl;
    }
    if(num_recortes<num_detecciones && num_detecciones!=-1) false_pos=false_pos+(num_detecciones-num_recortes);//Si imprimo un numero mayor
    false_pos=num_detecciones-comprobacion;
    estadisticas<<p.c_str()<<";"<<num_recortes<<";"<<true_pos<<";"<<false_pos<<endl;

    estadisticas.flush();
    estadisticas.close();
    archivo.flush();
    archivo.close();
    // }
    relaciones.clear();
```

ILUSTRACIÓN 52: EXTRACCIÓN DE DATOS PARA ESTADÍSTICAS

si la relación es positiva entramos en el siguiente if, si se encuentra entre 0 y 0.15 lo denominamos falso positivo. El valor de 0.15 ha sido el umbral que hemos puesto para considerarlo falso positivo ya que si un recuadro es demasiado grande no lo consideramos como detección buena, como en el siguiente caso.

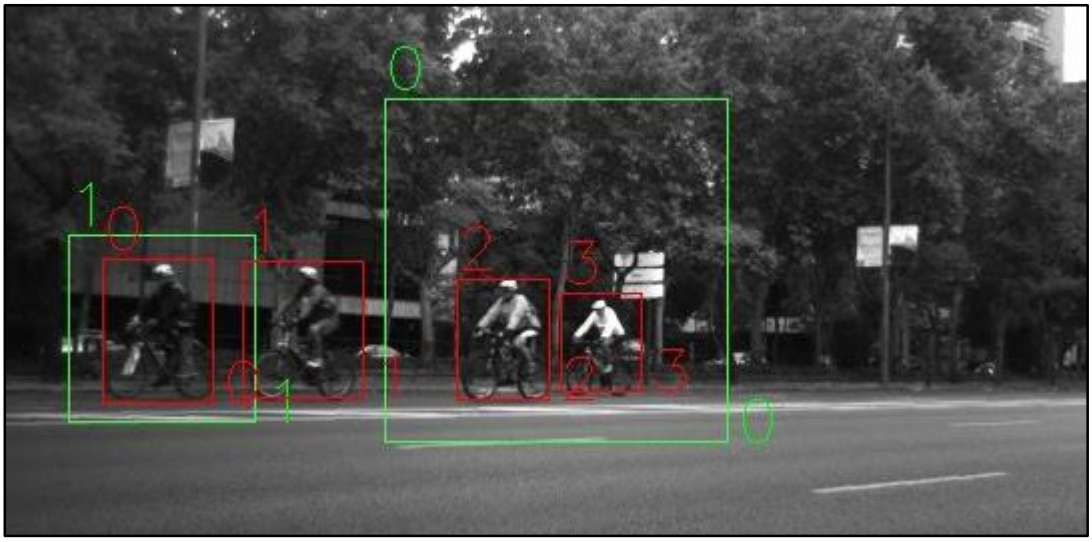

ILUSTRACIÓN 53: EJEMPLO FALSO POSITIVO CICLISTAS

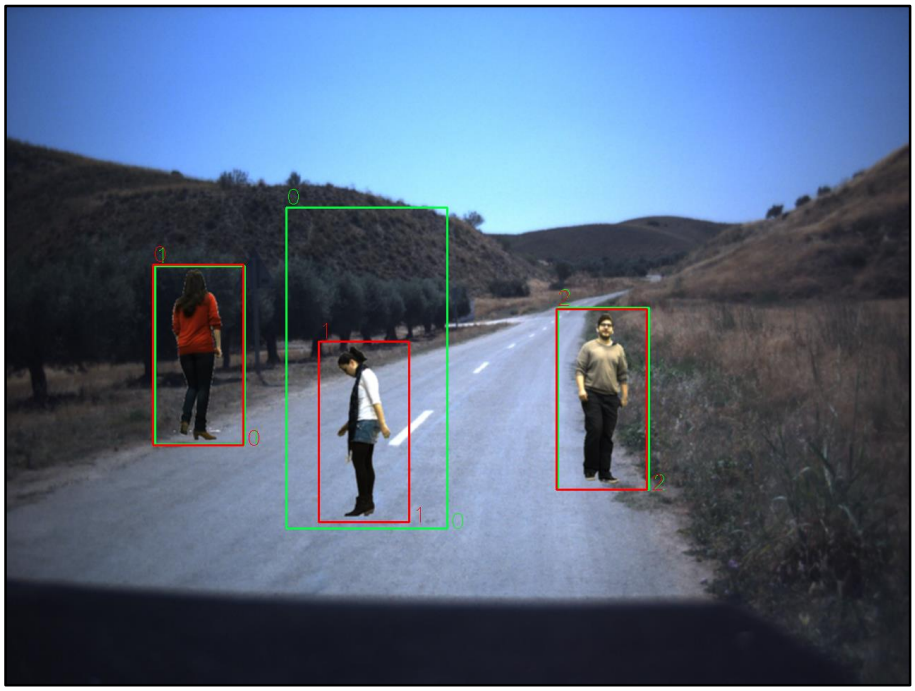

ILUSTRACIÓN 54: EJEMPLO FALSO POSITIVO PEATONES

En la Ilustración 53 e Ilustración 54 vemos que la detección es demasiado grande para considerarla buena, la relación con los recortes tendrá un valor menor que 0.15 por lo que la consideramos falso positivo en el caso de los ciclistas y un valor de 0.55 en el caso de los peatones.

Además, si obtenemos un número mayor de detecciones que de recortes significará que la diferencia entre ambos serán los falsos positivos que haya en la imagen. Con los valores de los positivos detectados y los falsos positivos procedemos a imprimirlos en un fichero de texto que quedará de la siguiente manera.

```
DiaBicicleta_2015-10-04-09-19-58_0.bag_img0058.txt;4;1;1
```

ILUSTRACIÓN 55: FICHERO CON RECORTES, DETECCIONES Y FALSOS POSITIVOS

La Ilustración 55 corresponde a la última imagen mostrada y en él se muestran los datos del nombre de la imagen, número de recortes (bicicletas que sabemos que hay), número de detecciones y falsos positivos.

8. RESULTADOS

En este apartado se analizarán los resultados obtenidos por el algoritmo desarrollado en este proyecto. Para realizar el estudio, en el caso de las bicicletas, se ha analizado la secuencia de imágenes de donde se sacó el conjunto de entrenamiento, la cual teníamos la certeza de cuantas bicicletas había y donde se encontraban. La secuencia cuenta con 284 *frames.* En primer lugar, se va a analizar los resultados obtenidos en el entrenamiento de las bicicletas y más tarde se analizará los resultados obtenidos en la detección de peatones, para luego poder alcanzar una conclusión sobre el algoritmo desarrollado.

8.1. Resultados obtenidos

Para cada uno de los clasificadores se van a calcular las matrices de confusión, la cual permite visualizar el nivel de confusión del clasificador, Precisión, Recall, FP, TP y MR.

La matriz de confusión tendrá el siguiente formato:

		RESULTADO	
		BICICLETA	NO-BICICLETA
REALIDAD	BICICLETA		
	NO-BICICLETA		

TABLA 1: MATRIZ DE CONFUSIÓN

Donde las filas corresponden a las instancias reales de las clases y las columnas son los resultados de la clasificación para la clase.

En el caso de los peatones será igual, pero cambiando el nombre de bicicleta por peatón y el de no-bicicleta por no-peatón.

Los valores de la matriz de 2x2 serán:

- Posición [1][1]: Verdaderos Positivos.
- Posición [1][2]: Falsos Negativos.
- Posición [2][1]: Falsos Positivos.
- Posición [2][2]: Verdaderos Negativos*.

*La posición correspondiente a verdaderos negativos se ha declarado como nula ya que las estadísticas las hemos sacado sobre una secuencia de un video sabiendo donde se posicionaban las bicicletas o peatones y viendo si las detecciones correspondían con esta información, por lo tanto, al analizar una frame completo y no sólo recortes positivos o

negativos la cantidad de verdaderos positivos sería infinita ya que cualquier cosa que no fuera positiva se consideraría un verdadero negativo.

La **precisión** indica el porcentaje de detecciones correctas. Podríamos calificarlo también como la calidad de respuesta del clasificador. La fórmula matemática es la siguiente:

$$Precisión = \frac{TP}{TP + FP}$$

El **recall** o **sensibilidad** indica el porcentaje de bicicletas o peatones que se han clasificado correctamente, dicho con otras palabras, la eficiencia de la clasificación de todos los elementos.

$$Recall = \frac{TP}{TP + FN}$$

El **MR (Miss Rate)** nos proporciona el grado de pérdida de candidatos.

$$MR = \frac{FN}{Nºmuestraspos}$$

Donde:

- **TP: "True Positives"** Número de imágenes clasificadas como positivas, siendo positivas.
- **FP: "False Positives"** Número de imágenes clasificadas erróneamente como positivas, siendo negativas.
- **FN: "False Negatives"** Número de imágenes clasificadas erróneamente como negativas, siendo positivas.
- **Nºmuestraspos:** Número de muestras totales positivas.

8.1.1. Bicicletas

La siguiente prueba consiste en la clasificación de 284 frames, pertenecientes a la secuencia de bicicletas, para poder ver sus resultados.

Matriz de confusión:

		RESULTADO	
		BICICLETA	**NO-BICICLETA**
REALIDAD	**BICICLETA**	537	82
	NO-BICICLETA	86	

TABLA 2:MATRIZ DE CONFUSIÓN DE CICLISTAS

Parámetros:

Precisión (%)	Recall o sensibilidad (%)	MR (Miss Rate) (%)
86,20%	86,75%	13,25%

TABLA 3: PARÁMETROS DE CONFIANZA CICLISTAS

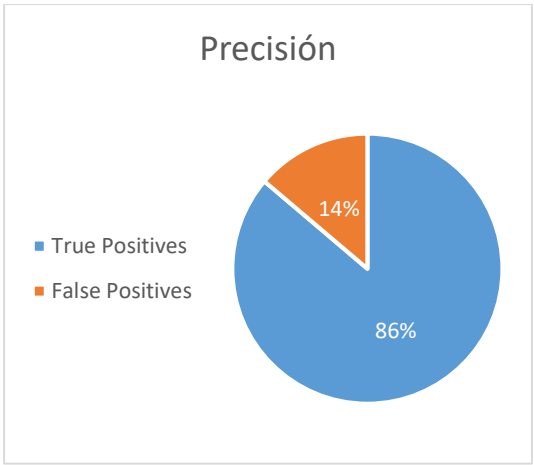

ILUSTRACIÓN 56: GRÁFICA DE PRECISIÓN CICLISTAS

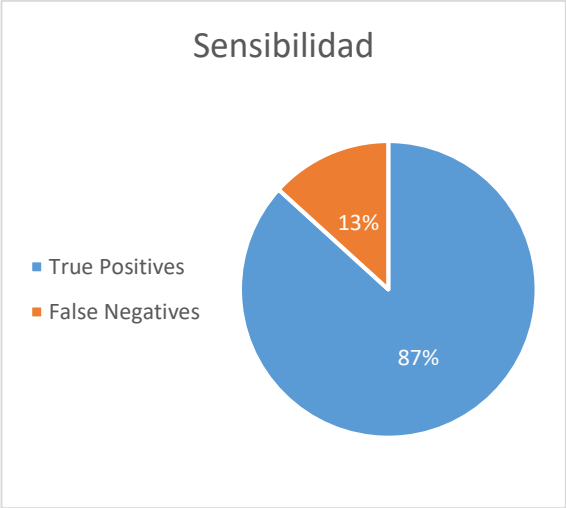

ILUSTRACIÓN 57: GRÁFICA DE SENSIBILIDAD CICLISTAS

Ratios:

True Positive Rate	False Negative Rate
0,87	0,14

TABLA 4: RATIOS CICLISTAS

En estos resultados vemos se obtiene al haber entrenado un clasificador con más de 25 mil imágenes una respuesta bastante buena ya que al analizar los 284 frames hemos detectado 537 bicicletas de 619 posibles otorgándonos un porcentaje del 87% de acierto. Teniendo un 14% de detecciones mal hechas (falsos positivos) y no detectando un 13,25% de bicicletas. Por lo que podemos afirmar que se trata de un clasificador con una alta tasa de acierto.

8.1.2. Peatones

Primera prueba

La siguiente prueba consiste en la clasificación de 479 imágenes en las que aparece diversos peatones.

La matriz de confusión de esta prueba será la siguiente:

Matriz de confusión:

		RESULTADO	
		PEATÓN	**NO-PEATÓN**
REALIDAD	**PEATÓN**	1415	22
	NO-PEATÓN	296	

TABLA 5: MATRIZ DE CONFUSIÓN PEATONES (PRUEBA 1)

Parámetros:

Precision(%)	Recall o sensibilidad	MR (Miss Rate)
82,70%	98,47%	1,53%

TABLA 6: PARÁMETROS DE CONFIANZA PEATONES (PRUEBA 1)

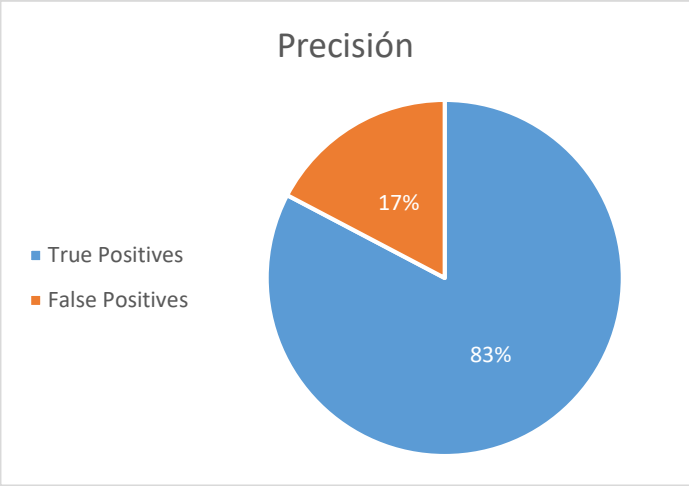

ILUSTRACIÓN 58: GRÁFICA DE PRECISIÓN PEATONES (PRUEBA 1)

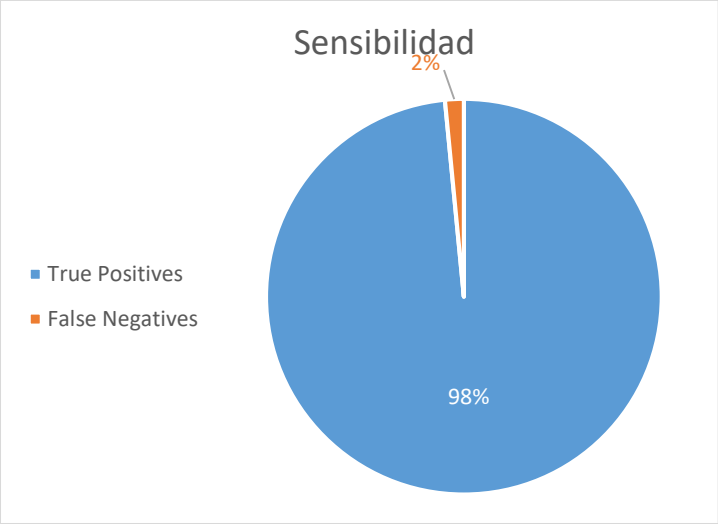

ILUSTRACIÓN 59: GRÁFICA DE SENSIBILIDAD PEATONES (PRUEBA 1)

Ratios:

True Positive Rate	False Negative Rate
0,98	0,17

TABLA 7: RATIOS PEATONES (PRUEBA 1)

Analizando los resultados podemos observar que tras analizar 479 imágenes en los que aparecen 1437 peatones han detectado a un total de 1415 peatones, lo que supone una tasa de acierto del 98,47% teniendo un porcentaje de pérdida del 1,53%. Estos resultados arrojan una precisión del 82,7% teniendo un total de 296 falsos positivos de 1711

detecciones. A priori podríamos decir que se trata de un clasificador fiable, pero al utilizar otro conjunto de detección diferente al de entrenamiento vemos que el número de falsos positivos se dispara alarmantemente. Esto puede deberse a lo explicado a continuación:

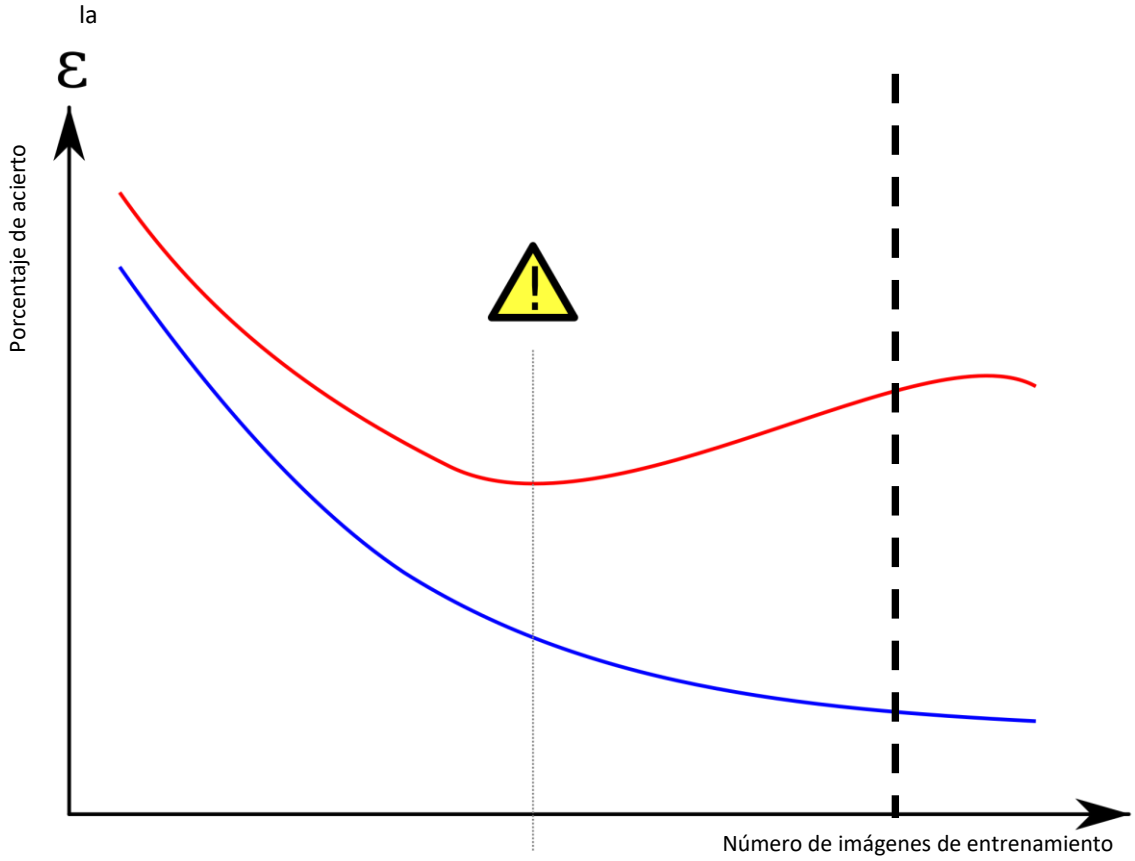

ILUSTRACIÓN 60: GRÁFICA UNDERFITTING/OVERFITTING

Si atendemos a la Ilustración 60 podemos observar que la línea azul, correspondiente a las detecciones con el conjunto de entrenamiento, a medida que se incrementa el número de imágenes de entrenamiento la respuesta de las detecciones es mejor, mientras que con un conjunto diferente al de entrenamiento (línea roja) vemos que llega un punto en el que, lejos de mejorar las detecciones, las empeora. Esto se debe a que a partir de ese punto de equilibro el sistema busca algo tan especifico que no lo logra encontrar en el conjunto que no es el de entrenamiento. Esto se denomina overfitting u overtraining que es el caso en el que nos encontramos, por lo tanto con este entrenamiento nos encontraríamos en la línea discontinua negra.

Segunda prueba

Al observar que los resultados obtenidos mostraban un cierto grado de overfitting nos dispusimos a hacer un entrenamiento diferente al ya realizado. Para ello eliminamos del conjunto de entrenamiento a una persona en concreto para posteriormente pasar el detector en una serie de imágenes donde sólo se encontraba la persona que hemos eliminado del entrenamiento. En estas detecciones la persona en concreto aparecía con vestimenta normal o con un thobe árabe. Por lo tanto, la detección se dificultaba pues en las imágenes no se le aprecian las piernas. A continuación se comentan los resultados obtenidos en las detecciones.

Matriz de confusión:

		RESULTADO	
		PEATÓN	**NO-PEATÓN**
REALIDAD	**PEATÓN**	1329	108
	NO-PEATÓN	489	

TABLA 8: MATRIZ DE CONFUSIÓN PEATONES (PRUEBA 2)

TABLA 9: MATRIZ DE CONFUSIÓN PEATONES (PRUEBA 2)

Parámetros:

Precision(%)	Recall o sensibilidad	MR (Miss Rate)
73,10%	92,48%	7,52%

TABLA 10: PARÁMETROS PEATONES (PRUEBA 2)

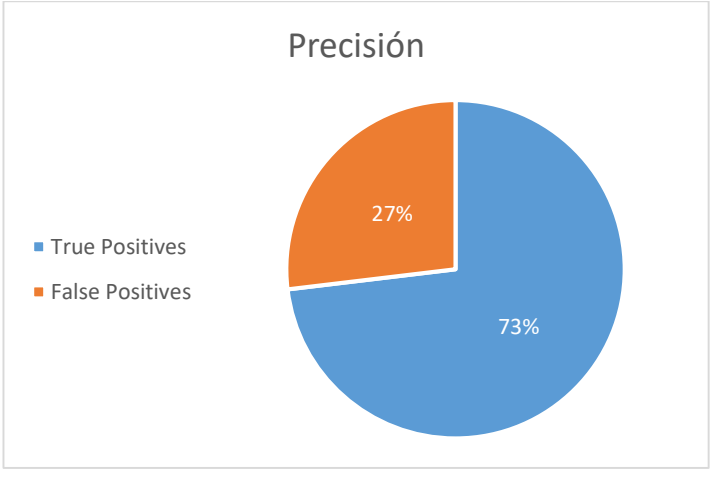

TABLA 11: GRÁFICA PRECISIÓN PEATONES (PRUEBA 2)

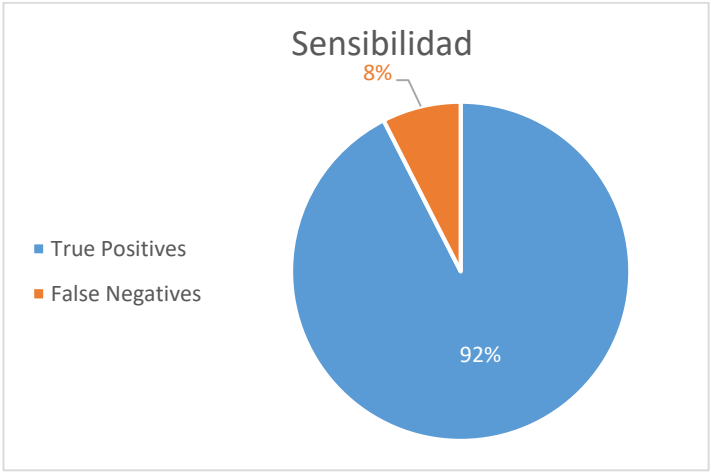

TABLA 12: GRÁFICA SENSIBILIDAD PEATONES (PRUEBA 2)

Ratios:

True Positive Rate	False Negative Rate
0,92	0,27

TABLA 13: RATIOS PEATONES (PRUEBA 2)

Según los datos obtenidos seguimos teniendo una gran fiabilidad ya que la tasa de acierto es de un 92,48% tras analizar 479 imágenes con una existencia total de 1437 positivos. El ratio de pérdida sigue siendo bajo con un valor del 7,52%. En esta prueba hemos podido observar el aumento del número de falsos positivos, ascendiendo a un total de 489. La tarea de reducir este número resulta muy trabajosa ya que la mayoría de los falsos positivos se sitúan o en líneas discontinuas o en torres eléctricas que podríamos las cuales tienen una apariencia similar a las que tendría un peatón, por lo tanto, al realizar el boostrapping e incluirlos en los negativos no adquieren un valor relevante como para poder desecharlo en las detecciones.

9. PRESUPUESTO

De acuerdo con las tablas que libera anualmente la Agencia Tributaria, si atendemos a la tabla liberada en 2015, los equipos electrónicos tienen una amortización del 20%.

El desarrollo del proyecto ha llevado 4 meses de trabajo invirtiendo de media 5,5 horas diarias, por tanto, ese será el tiempo que servirá para calcular el sueldo del desarrollador y las desamortizaciones.

Para hacer el cálculo de la amortización de los elementos usados durante el proyecto se tendrá en cuenta el porcentaje de uso que se les ha dado durante el tiempo de desarrollo del proyecto, puesto que, por ejemplo, la cámara de video ha sido mucho menos usado que el ordenador, por ejemplo. Por lo que el precio de la desamortización quedaría de la siguiente manera:

$$Precio\ de\ desamortiación = Precio * \frac{uso}{100} * coef.amortizacion * \frac{4\ meses}{12\ meses}$$

Elemento	Precio (€)	Uso	Coeficiente de amortización	Precio de desamortización (€)
Jetson TK1	268	100%	0.2	17,86

TABLA 14: AMORTIZACIÓN

Para el coste de personal se ha tenido en cuenta que según el Instituto Nacional de Estadística el sueldo medio de un ingeniero junior es de 40.000€ al año a jornada completa. Por lo tanto, el coste quedaría de la siguiente manera.

$$Salario = \frac{40000 * \frac{5{,}5\ horas}{8\ horas}}{12\ meses} = 2.291{,}66 \frac{€}{mes}\ jornada\ parcial$$

Apellidos, Nombre	Categoría	Tiempo dedicado (meses)	Coste (€/mes)	Coste total (€)
Fernández González, Alejandro	Ingeniero	4	2.291,66	9.166,64

TABLA 15: COSTE DE DESARROLLO

Por lo tanto, los costes totales del proyecto serán:

Elemento	Coste (€)
Material	16,97
Personal	9.166,64
Total	9.184,5

TABLA 16: COSTE TOTAL

10. LINEAS FUTURAS

A lo largo de este Proyecto han ido apareciendo diversos problemas, la mayoría de ellos se pudieron solucionar, por no decir la totalidad. A parte de los problemas surgidos también se pensaron nuevas formas de ampliar este Trabajo de Fin de Grado. A continuación, se comentarán algunas de estas posibles ampliaciones.

Las detecciones de este proyecto se hacen con una cámara de video, existe una posibilidad mucho más versátil ante los cambios de luz, pues en situaciones de poca luz la cámara de video no es capaz de detectar la totalidad de la escena. Por lo tanto, una opción viable sería la incorporación de cámara infrarrojas ya que estas no dependen en gran medida de la iluminación además de ofrecer un menor nivel de ruido, por lo que sería mucho más sencillo la detección de los elementos de interés.

Otra posibilidad sería agregar la detección de otros obstáculos, esto permitiría abarcar toda la gama de situaciones que se pudieran dar en la carretera. Al ya tener entrenado el clasificador con peatones y ciclistas lo ideal sería poder entrenarlo con coches y motocicletas.

Una vez implementado lo comentado anteriormente agregar una ayuda de frenado a este sistema sería muy efectiva, pues, como se ha comentado en el anterior párrafo, podríamos abarcar prácticamente todas las situaciones que se pudieran dar en la carretera y la reacción del sistema automático de frenado sería mucho más fiable.

Algo que ayudaría a la reducción de accidentes en carretera seria el seguimiento de obstáculos. El poder predecirla trayectoria de los obstáculos detectado daría un valor extra a los sistemas de seguridad del automóvil, ya que permitiría anticiparse a posibles situaciones comprometidas para el vehículo y sus ocupantes. Esto sería posible incorporando el filtro Kalman, que permitiría calcular la trayectoria en tiempo real utilizando únicamente las mediciones de entrada actuales.

Un sistema de alerta al conductor complementaría todo lo comentado anteriormente. Este sistema se tendría que activar en casos de extremo peligro para que el conductor pudiera reaccionar ante ellos. Considerando una señal sonora algo perjudicial ya que podría ocasionar algún movimiento brusco de parte del conductor, creo que lo ideal sería la incorporación de algún sistema de vibración en el volante o una señal lumínica reflejada en la luna del coche como se produce en algunos de los sistemas ADAS.

Combinando la visión estéreo con un sistema de detección laser haría que las detecciones realizadas fueran mucho más precisas.

11. CONCLUSIONES

La realización de este trabajo ha supuesto un duro reto al que enfrentarse y muchos días de frustración delante de la pantalla buscando por qué el programa no reconocía, únicamente marcaba como detección el centro de la imagen sea cual fuera. Y, en parte, gracias a eso, a la frustración sufrida, me ha servido para adquirir muchos conocimientos sobre visión por computador, sistemas inteligentes de vehículos… y, sobre todo, la forma de realizar un proyecto, las distintas fases que se van siguiendo (planificación, desarrollo, pruebas) y la manera en la que se trabaja en él. Por lo que, fruto de la frustración sufrida, he podido sacar más cosas buenas que malas del proyecto.

En el proyecto realizado se ha buscado con éxito identificar tanto ciclistas como peatones presentes en una escena mediante visión por computador.

Se optó desde un primer momento por la combinación de HOG-SVM ya que, para la detección de peatones, que es una parte importante del proyecto, la técnica HOG es más efectiva que otro tipo como Haar-Cascade. La técnica Haar-Cascade es más eficaz modelando texturas y no reconociendo objetos.

Además, la utilización del clasificador SVM ofrece mucha versatilidad, ya que permiten el aprendizaje. Esto es, la posibilidad de ser re-entrenadas con aquellas muestras que en otras ocasiones ha clasificado mal, por lo que lo convierte en una maquina más robusta con cada entrenamiento.

Podríamos decir que los resultados del proyecto son bastante satisfactorios. Con una tasa de acierto del 86,2% y con una tasa baja de falsos positivos (14%) en lo referente a las bicicletas. En lo referente a los peatones obtenemos un buen porcentaje de acierto (92,48%) una vez reducida la cantidad de las imágenes positivas ya que si no funcionaría muy bien si las detecciones fueran con el conjunto de entrenamiento, pero a la hora de detectar fuera de ese conjunto el clasificador buscaría algo tan concreto que las detecciones no serían nada buenas.

El coste total del proyecto ha sido de 9.184,5€, tratándose la mayor parte de ello en personal (9.166,64 €). Lo que permitiría que los costes al añadir este sistema a un automóvil fueran muy bajos, más aún teniendo en cuenta que parte de ese coste (ordenador y cámara) puede ser compartido con otros sistemas del coche.

Es decir, con un coste muy bajo se podría dotar en cualquier vehículo un sistema de detección de peatones y ciclistas con una alta tasa de éxito. Esta tecnología permitiría disminuir el número de atropellos.

12. BIBLIOGRAFÍA

Bibliografía para la realización de la memoria

[1] «Perez Tirado» [10/06/16]. Disponible: http://www.pereztirado.com/mortalidad-accidentes-de-trafico-en-2014/.

[2] «DGT» [10/06/16]. Disponible: http://www.dgt.es/es/prensa/notas-de-prensa/2016/20160104-nuevo-minimo-historico-numero-victimas-mortales-accidente-desde-1960.shtml.

[3] L. Cifuentes, «EstrellaDigital» [10/06/16]. Disponible: http://www.estrelladigital.es/articulo/empresas/registra-nuevo-descenso-accidentes-trafico/20160527130711286091.html.

[4] «Bhsinterceptor» [10/06/16]. Disponible: https://bhsinterceptor.wordpress.com/2014/05/23/estadisticas-de-accidentes-de-trafico/.

[5] «DGT» [10/06/16]. Disponible: http://www.dgt.es/revista/num225/files/basic-html/page20.html.

[6] D. Villareal, «DiarioMotor» [10/06/16]. Disponible: http://www.diariomotor.com/2014/01/03/1-128-victimas-mortales-en-accidentes-de-trafico-una-tragica-estadistica-que-sigue-menguando/.

[7] O. Balderas, «eluniversal» [10/06/16]. Disponible: http://www.eluniversal.com.mx/articulo/periodismo-de-investigacion/2015/11/28/se-dispara-el-numero-de-accidentes-ciclistas-sspdf.

[8] Villarramblas, «EnbiciporMadrid» [10/06/16]. Disponible: http://www.enbicipormadrid.es/2013/03/los-5-accidentes-ciclistas-mas.html.

[9] Ibáñez, «circulaseguro» [10/06/16]. Disponible: http://www.circulaseguro.com/sistemas-de-deteccion-de-la-fatiga-y-falta-de-concentracion-al-volante/.

[10] «TechBrief» [10/06/16]. Disponible: http://ntl.bts.gov/lib/10000/10100/10114/tb98-006.pdf.

[11] E. Viso, «circulaseguro» [10/06/16]. Disponible: http://www.circulaseguro.com/que-es-la-deteccion-de-las-senales-de-trafico/.

[12] D. Villareal, «DiarioMotor» [10/06/16]. Disponible: http://www.diariomotor.com/2014/12/11/alerta-cambio-involuntario-carril/.

[13] «Samsung» [10/06/16]. Disponible: https://news.samsung.com/global/the-safety-truck-could-revolutionize-road-safety.

[14] A. García, «frenomotor» [10/06/16]. Disponible: http://frenomotor.com/tecnologia/dentro-de-tu-coche-11.

[15] «Google» [10/06/16]. Disponible: https://www.google.com/selfdrivingcar/.

[16] «infobae» [10/06/16]. Disponible: http://www.infobae.com/2016/01/26/1785647-como-funciona-el-tesla-s-un-auto-100-autonomo.

[17] Ibáñez, «MotorPasionfFuturo» [10/06/16]. Disponible: http://www.motorpasionfuturo.com/ayudas-a-la-conduccion/los-coches-que-se-aparcan-completamente-solos.

[18] «Epixea» [10/06/16]. Disponible: http://www.epixea.com/research/multi-view-coding-thesisse8.html.

[19] «fisicanet» [10/06/16]. Disponible: http://www.fisicanet.com.ar/fisica/ondas/ap17_optica_geometrica.php.

[20] J. Oliver. [10/06/16]. Disponible: https://riunet.upv.es/bitstream/handle/10251/12624/Tesina-Javier_Oliver_Moll.pdf?sequence=1.

[21] «elinux» [10/06/16]. Disponible: http://elinux.org/Jetson_TK1.

Bibliografía para la realización del algoritmo

[22] «qt» [10/06/16]. Disponible: https://www.qt.io/ide/.

[23] «OpenCV» [10/06/16]. Disponible: http://opencv.org/.

[24] «github» [10/06/16]. Disponible: https://github.com/DaHoC/trainHOG/blob/master/libsvm/libsvm.h.

[25] «OpenCV» [10/06/16]. Disponible: http://answers.opencv.org/question/33874/hog-person-detection-and-setting-up-svm-classifiers/.

[26] «OpenCV» [10/06/16]. Disponible: http://answers.opencv.org/question/26818/svm-bias-on-weights-of-positives-and-negatives/.

[27] «github» [10/06/16]. Disponible: https://github.com/Itseez/opencv/blob/master/samples/cpp/train_HOG.cpp.

[28] «stackoverflow» [10/06/16]. Disponible: http://stackoverflow.com/questions/22346690/how-to-use-model-generated-by-libsvm-in-opencv-project/22354047#22354047.

[29] K.-T. Lai, «kuantinglai» [10/06/16]. Disponible: http://kuantinglai.blogspot.com.es/2013/07/using-libsvm-with-opencv-mat.html.

[30] «stackoverflow» [10/06/16]. Disponible: http://stackoverflow.com/questions/34758138/how-to-load-svm-train-data-model-by-libsvm-in-opencv-3-1.

[31] «github» [10/06/16]. Disponible: https://github.com/DaHoC/trainHOG/wiki/trainHOG-Tutorial.