UNIVERSIDAD POLITECNICA DE VALENCIA

ESCUELA POLITECNICA SUPERIOR DE GANDIA

Grado en Ing. Sist. de Telecom., Sonido e Imagen

Implementación de software para la gestión de un parking mediante el uso de técnicas de visión artificial

TRABAJO FINAL DE GRADO

Autor/a:

Eric Beaucamps Santofimia

Tutor/a:

José Ignacio Herranz Herruzo

RESUMEN

En éste proyecto veremos cómo utilizar diferentes técnicas de visión artificial con la intención de gestionar la salida y entrada de vehículos en un parking, para ello entrenaremos un clasificador en cascada con el fin de detectar coches desde una determinada perspectiva. Entrenar el clasificador requiere al menos 100 imágenes del objeto a detectar, es por eso que antes del entrenamiento es necesaria una fase de recopilación y procesado de imágenes.

Una vez entrenado el clasificador, tendremos que desarrollar un algoritmo que permita detectar y clasificar los coches con el fin de obtener información acerca de la ubicación y la cantidad de éstos.

Con el fin de utilizar la aplicación de una forma cómoda, se desarrollará una interfaz gráfica que nos permita visualizar tanto la detección desde las cámaras, como la información obtenida gracias a la detección previa.

Llevaremos a cabo el proyecto utilizando como leguaje de programación Java y mediante las librerías de OpenCV, las cuales son de las más completas y utilizadas en visión artificial.

PALABRAS CLAVE

Visión Artificial, Descriptores *HAAR*, OpenCV, Entrenamiento, Coche.

ABSTRACT

In this project we will see how to use different artificial vision techniques with the intention of managing the exit and entry of vehicles in a parking, for this we will train a cascade classifier in order to detect cars from a certain perspective. Training the classifier requires at least 200 images of the object to be detected, that is why a phase of image collection and processing is necessary before training.

Once the classifier has been trained, we will have to develop an algorithm that allows us to detect and classify the cars in order to obtain information about the location and the quantity of these.

In order to use the application in a comfortable way, a graphic interface will be developed that allows us to visualize both the detection from the camera and the information obtained thanks to the previous detection.

We will carry out the project using Java programming language and OpenCV libraries, which are the most complete and used in artificial vision.

KEY WORDS

Artificial Vision, *HAAR* Descriptors, OpenCV, Training, Car.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGO

1 INTRODUCCIÓN

1.1 PRESENTACIÓN

Es frecuente ir a un parking y encontrarnos con sensores de movimiento iluminados con leds, estos dispositivos están colocados en cada plaza de aparcamiento con el fin de controlar la entrada y salida de vehículo.¿Cómo podríamos realizar la misma función pero evitando la instalación masiva de sensores? Utilizando un pequeño número de cámaras, mayor o menor en función del diseño del parking, y aplicando diversas técnicas de visión artificial podemos conseguir el mismo resultado y además obtener información adicional, como las dimensiones del vehículo.

En la actualidad las técnicas de aprendizaje automático, las cuales consisten en detección de patrones mediante algoritmos matemáticos, se utilizan eventualmente en diferentes ámbitos. Las aplicaciones pueden ser muy variadas, desde control de calidad en una cadena de producciones (como por ejemplo control de anomalías en piezas), hasta reconocimiento del habla, motores de búsqueda, diagnósticos médicos, robótica, etc....

En este proyecto se utilizarán las técnicas de aprendizaje automático aplicadas a la visión artificial para la detección de coches con el fin de gestionar las plazas de un parking, el aparcamiento del centro comercial Plaza Mayor en Gandía será el seleccionado para realizar las pruebas.

Una vez entrenado un clasificador decente y desarrollado un algoritmo que nos permita detectar los coches, el software debe devolver información sobre el movimiento de los coches: plazas libres, plazas ocupadas, salida y entrada de coches.

Todos los pasos seguidos y técnicas aplicadas serán explicados a lo largo de este trabajo.

1.2 OBJETIVOS DEL PROYECTO

El objetivo principal del proyecto es desarrollar un programa que permita detectar coches mediante una cámara de video y que a partir de los coches detectados se obtenga diferente información acerca de su posición, pero para llegar a esto habrá que ir cumpliendo objetivos a menor escala:

- Recopilar al menos 100 fotografías de los objetos a detectar, en nuestros tendremos que capturar imágenes de los coches del parking a gestionar y desde la misma perspectiva desde la que posteriormente detectaremos los coches.
- Entrenar el clasificador en cascada con las imágenes previamente recopiladas.
- Desarrollar un software que nos permita aplicar el clasificador para detectar los coches y recopilar información de esto.
- Desarrollar las diferentes funcionalidades del programa: Conteo de coches, entrada y salida, plazas libres, ubicación de los coches.

Para conseguir los resultados esperados ha sido necesaria una fase previa de investigación acera de las diferentes técnicas empleadas para detección de objetos que incluye obtener información, comparar las diferentes técnicas y decidir la opción más óptima para el proyecto.

1.3 FASES DEL PROYECTO

La elaboración del proyecto pasará por diferentes fases las cuales están relacionadas con los objetivos a cumplir enunciados en el punto anterior.

Fase 1. La fase inicial consiste en buscar información e investigar para buscar la opción más óptima a la hora de desarrollar nuestro programa.

La mayor parte del esfuerzo ha sido dedicado a elegir el método más adecuado para entrenar el clasificador y el tipo de descriptores que utilizaremos para ello. Pero también ha sido necesario elegir el lenguaje y la IDE que se utilizará, en éste caso el lenguaje elegido ha sido Java y la IDE Net Beans, el motivo de ésta elección es que me encuentro más familiarizado con éstas herramientas, pero posiblemente la mejor opción objetivamente hubiese sido Visual Studio y Python.

Cabe decir que la búsqueda de información, a pesar de ser más intensa en la fase inicial del proyecto a sido constante a lo largo de todo el proceso.

Fase 2.Una vez investigado acerca de las técnicas requeridas para desarrollar el detector, el siguiente paso ha sido familiarizarme con la librerías a utilizar, en éste caso he decidido trabajar con OpenCV, uno de los framework más utilizados en el mundo para proyectos de visión artificial. Los motivos de elegir OpenCV han sido, por un lado que es totalmente gratuito y muy accesible, y por otro lado la gran cantidad de funcionalidades que posee y la calidad de éstas mismas.

Al ser la primera vez que trabajaba con OpenCV ésta segunda fase ha consistido en documentarme con libros, tutoriales y manuales para aprender a manejarme con las clases de OpenCV.

Fase 3. Para entrenar el clasificador de una forma adecuada es necesario poseer al menos 100 imágenes del objeto a detectar.

En mi caso los objetos a detectar son los coches aparcados en el parking del centro comercial Plaza mayor (Gandía), es por eso que ha sido necesario ir recolectando imágenes de éstos mismos.

Además de recolectar las imágenes, éstas tienen que tener unas dimensiones y ciertas características concretas, por eso será necesario procesarlas para que posean las características necesarias, para ello he empleado Matlab.

Fase 4. Una vez que disponemos de los elementos necesarios toca entrenar el clasificador, para ello es necesario pasar por diferentes fases:

- Crear las imágenes de muestra y almacenarlas en un archivo de tipo vec.
- Ajustar ciertos parámetros como el Ratio de Error o la permisividad ante un fallo.
- Por último, entrenar el clasificador, lo cuál requiere de al menos 1 hora.

Para poder realizar éstos paso se han utilizado diferentes funcionalidades que posee OpenCV.

Fase 5. Ya tenemos un clasificador decente, ahora será necesario desarrollar un software que capture un video frame a frame y que a cada uno de esto frames se le aplique el clasificador para comprobar si aparece algún coche en la imagen. Una vez detectados los coches tendremos que diseñar un algoritmo que permita obtener información a partir de las detecciones.

Para tener un algoritmo que funcione correctamente ha sido necesario realizar una infinidad de pruebas con el fin de ajustar al máximo los diferentes parámetros y conseguir un funcionamiento óptimo.

Fase 6. Fase final del proyecto, la cuál ha consiste simplemente en pulir el código para evitar redundancias y que funcione lo mejor posible y por último llevar a cabo la redacción de ésta memoria.

Fase	Tiempo requerido
1. Investigación	50 h
2. Familiarización con OpenCV	50 h
3. Recolección de imágenes	20 h (repartidas en varios meses)
4. Entrenamiento	50 h
5. Desarrollo del software	70 h
6. Pulir código y redactar	70 h

Tabla 1. Fases del proyecto.

2 MARCO TEÓRICO

En el siguiente punto se pasarán a explicar, de una forma resumida, diferentes conceptos relacionados con la detección de objetos. Para ello me he basado en diferentes fuentes y en mi propia experiencia a la hora de poner en práctica dichos conceptos.

2.1 INTRODUCCIÓN A LA DETECCIÓN DE OBJETOS

La detección de objetos basada en visión por computador no es tan fácil como pueda parecer a priori, como seres humanos si nos presentan una imagen y nos piden buscar objetos en ella lo hacemos con relativa sencillez, sin embargo, para un sistema automático esto no es tan sencillo. Únicamente a partir de la última década se han empezado a obtener resultados que son útiles para resolver ciertas aplicaciones, de hecho la detección de objetos sigue siendo un campo de investigación en la visión por computador.

Las aplicaciones que pueden tener la detección de objetos son muy variadas y pueden ser casi infinitas, pero la mayoría suelen estar relacionadas con sistemas automáticos. Algunas de éstas aplicaciones son:

- El reconocimiento óptico de caracteres (ROC), que consiste en la digitalización de textos, los cuales se identifican automáticamente a partir de una imagen símbolos o caracteres que pertenecen a un determinado alfabeto, para luego almacenarlos en forma de datos. Así podremos interactuar con estos mediante un programa de edición de texto o similar.

- Control de calidad en cadenas automáticas. En algunas empresas que requieren de una cadena de montaje para automatizar todos los procesos, se utilizan cámaras con el fin de detectar imperfecciones en los productos manufacturados.

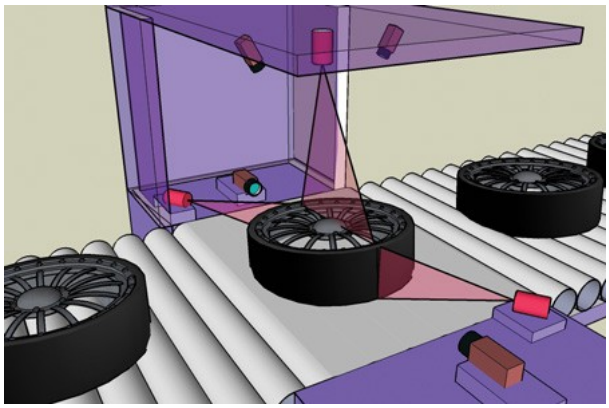

Ilustración1.Ejemplo de sistema automático de control de calidad

El objetivo es modelar los objetos con el objetivo de discriminar unos de otros, e inclusive dentro del propio tipo objeto diferenciar las diferentes varianzas que se pueden dar tanto en el objeto (ej. Dentro del grupo de objetos “coches” permitir diferentes colores y tamaños) como en el entorno (ej. En la imagen a analizar sea de día o de noche).

Las diferentes fases por las que pasará el proceso de detección de objetos en nuestro caso será el representado en el siguiente diagrama:

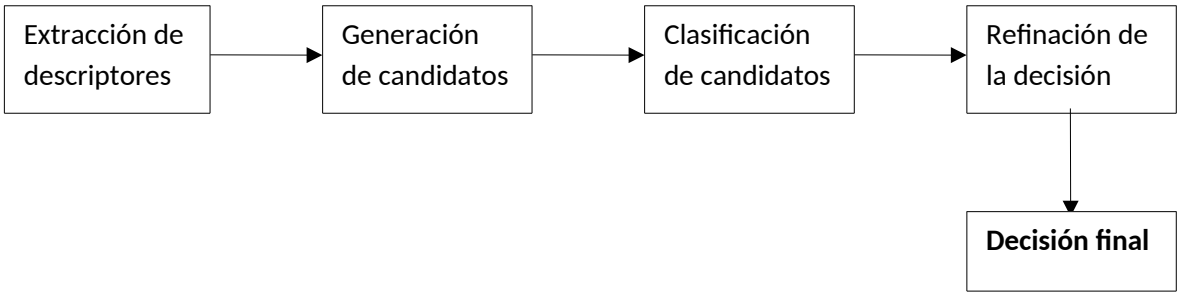

Ilustración 2.Tabla resumen de fases para la detección de objetos

La primera fase consiste en la extracción de una serie de características llamadas descriptores basados en las diferentes relaciones entre un pixel y sus vecinos, como la diferencia de intensidad. Una vez extraídas las diferentes características se generará una serie de candidatos, los cuales pasarán por un módulo de clasificación que decidirá si representan un objeto a detectar o no. En muchas ocasiones se realizan detecciones redundantes por lo que es necesario refinar el resultado.

Todo lo enunciado será explicado detalladamente en los siguientes puntos.

2.2 DESCRIPTORES

Para poder clasificar o detectar los objetos de una imagen es necesario medir sus cualidades mediante la extracción de ciertos parámetros sencillos que nos aporte la información necesaria para distinguir los objetos a detectar.

De ésta forma cada objeto vendrá caracterizado por un vector de valores numéricos el cual posteriormente utilizará el clasificador para discernir si en una región de pixeles de la imagen aparece uno de los objetos a detectar.

Cabe decir que los descriptores a pesar de reflejar información más o menos simple, han de cumplir dos características mínimas:

- Unicidad: es decir la información acerca del objeto que deseamos detectar lo haga totalmente diferenciable del resto de objetos, que los parámetros que hacen particular a un objeto no se repitan con los demás.
- Invariancia: que la información no se vea perjudicada al modificar las dimensiones de la imagen.

Existen una gran variedad de descriptores, pero los más empleados son los siguientes:

- **Tamaño**: se basan en el tamaño de los objetos en función de la distancia entre píxeles, se puede obtener diferente información de éstos descriptores, como sería el área, perímetro, diámetro. El principal problema que presentan es la difícil escalabilidad, ya que te obliga a conocer las dimensiones reales del objeto a detectar o de algún otro como referencia

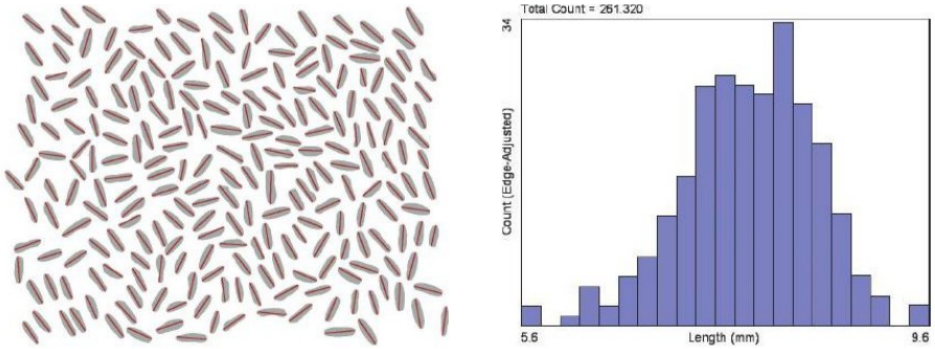

Ilustración 3. Histograma del dímetro.

- **Forma**: ésta clase de descriptores son los más adecuados para distinguir formas entre sí. Podemos encontrar 2 tipo diferentes: los de frontera y de región. Los descriptores de frontera describen la trayectoria del contorno del

objeto, por ejemplo se puede representar el borde del objeto en forma de función unidimensional (firma). En ésta categoría también se encuentran los descriptores de Fourier los cuales tratan el borde de un objeto como si fueran puntos de un plano complejo.

Algunos de los algoritmos utilizados para extraer éstos descriptores:Region-basedShape Descriptor (RSD), Contour-basedShape Descriptor (CSD), 3-D Shape Descriptor (3-D SD)

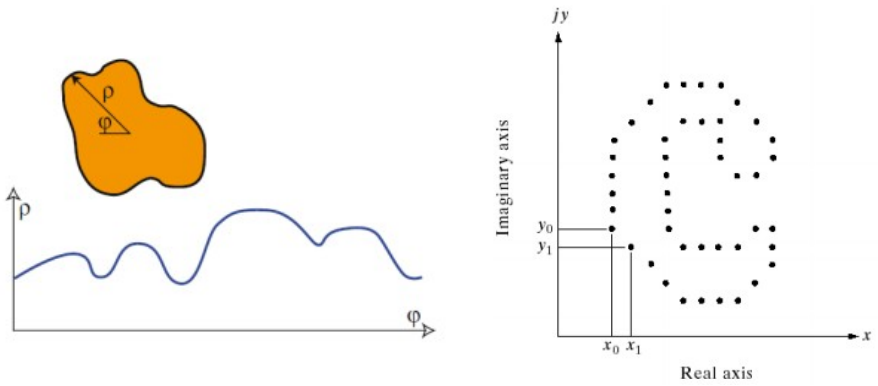

Ilustración 4. Representación de un objeto mediante firmay ejemplo descriptores de Fourier.

- **Color:** otros descriptores muy utilizados son los basados en las diferencias de color entre diferentes regiones de la imagen. El principal problema que muestran estos descriptores son las diferencias de iluminación, ya que si una región de la imagen está más iluminada que otra el percepción del color será diferente.

  Algunos de los algoritmos empleados para extraer éstos descriptores: Dominant Color Descriptor (DCD, Scalable Color Descriptor (SCD), Color Structure Descriptor (CSD), Color Layout Descriptor (CLD), Group of frame (GoF) o Group-of-pictures (GoP).

2.3 DESCRIPTORES AVANZADOS

Los descriptores analizados anteriormente son descriptores básicos y mas o menos fáciles de obtener, pero la información que nos ofrecen a veces no es suficiente o simplemente la situación hace que sean imposibles de extraer, debido a una amplia variedad de objetos a detectar, escenas variantes en un video, fallos de iluminación, imposibilidad de realizar segmentación por umbralización[1]Es por eso que destacan algunos descriptores más complejos que a pesar de ser más duros de procesar nos ofrecen una mayor calidad y robustez frente a problemas varios.

En los siguientes puntos veremos dos de los descriptores avanzados que más se utilizan hoy en día, *HAAR* y *HOG*. En el caso de éste proyecto se han empleado descriptores *HAAR*, es por ello que éstos serán analizados con más detalle.

2.3.1 DESCRIPTORES HOG

Éste tipo de descriptores se basan en el cálculo del vector gradiente para discernir si existe un borde y por lo tanto un objeto en la imagen.

El vector gradiente se cálculo en cada pixel y nos indica la dirección del mayor cambio de intensidad El resultado muestra cómo de abruptamente o suavemente cambia una imagen en cada punto analizado, y en consecuencia, cuán probable es que éste represente un borde en la imagen, y capturando la orientación a la que tiende ese borde.

Ilustración5.Gradiente de la imagen de un peatón.

Como podemos apreciar en esta imagen del gradiente, los valores altos no están indicando que existen cambios de intensidad en la imagen, y como hemos explicado

[1]**Segmentación por umbralización**:La umbralización es una técnica de segmentación ampliamente utilizada en las aplicaciones industriales. Se emplea cuando hay una clara diferencia entre los objetos a extraer respecto del fondo de la escena.

antes, los puntos en los que existe un cambio de intensidad más abrupto coinciden con el contorno de los objetos.

El detector *HOG* permite aprovechar de una forma muy eficiente la información del gradiente a partir de combinar esta información en forma de histogramas locales, que se calculan en celdas de pequeño tamaño y que se distribuyen de forma uniforme por toda la imagen como podemos ver en el siguiente ejemplo.

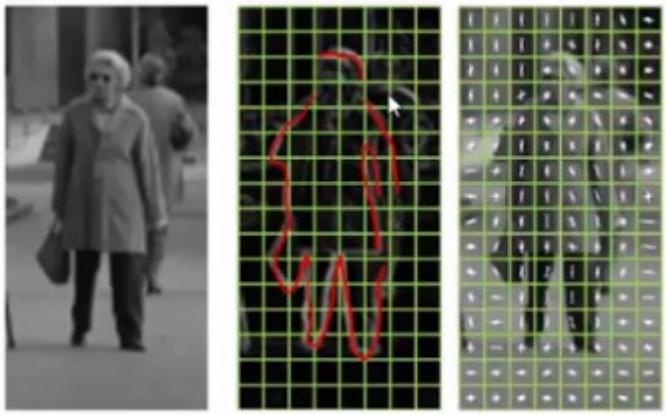

Ilustración 6. Celdas locales e histogramas por celda

Como vemos en la *Ilustración 6,* los histogramas nos proporcionan información de los contornos que dominan en cada una de las posiciones de la imagen, en las piernas los contornos verticales son los predominantes y en cambio en los hombros predomina la orientación diagonal. Esta información nos permite distinguir la forma de los objetos de la imagen lo cual es una muy una buena base para empezar a detectarlos.

Cómo último paso, los histogramas que se calculan localmente en diferentes posiciones de la imagen se agrupan en bloques de un tamaño un poco mayor y estos bloques sirven para normalizar la representación final y para hacerla más invariante ante cambios de iluminación y distorsiones en la imagen. Así, la representación final será la concatenación de la representación de todos estos bloques.

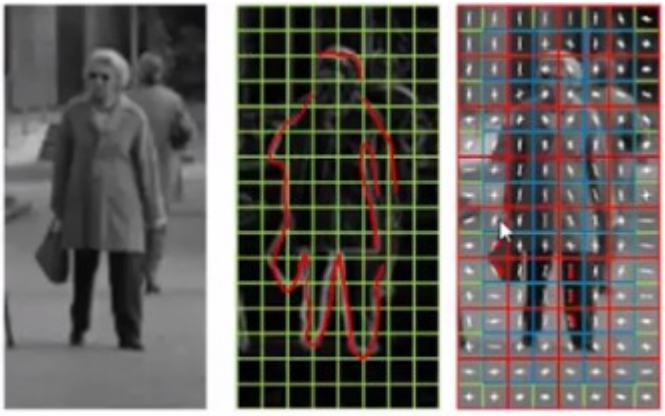

Ilustración7. Histogramas agrupados en bloques mayores

Algunas de las aplicaciones para las que los descriptores *HOG* que funcionan de forma óptima:

- Reconocimiento de gestos.
- Reconocimiento de personas.
- Reconocimiento de caracteres.

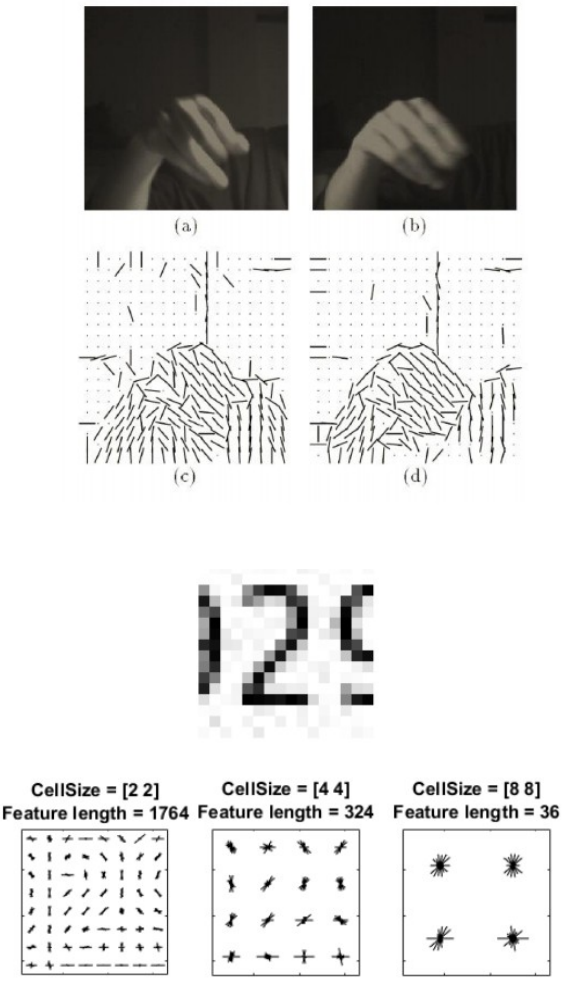

Ilustración 8.Ejemplos de reconocimiento de gestos y reconocimiento de caracteres.

2.3.2 DESCRIPTORES HAAR

Los descriptores *HAAR* son unos de los más eficientes que existen en la actualidad debido a su bajo coste computacional. Las características de *HAAR* se obtienen después de aplicar un conjunto de filtros por toda la imagen a diferentes escalas, de esta manera vamos a tener un número muy elevado de características, del orden de más de 100.000 por imagen.

El clasificador va a tener que tratar con este número tan elevado de características y para ello vamos a utilizar *Adaboost*, que como veremos es un método que nos permite mientras aprendemos también determinar y seleccionar las características que son más relevantes para la clasificación

.

El cálculo del descriptor va a serun poco diferente a lo que hemos visto con *HOG,* que se basa en calcular características locales de cada pixely después, estas características se agregaban en bloques utilizando histogramas. El cálculo del descriptor *HAAR* se basa en aplicar un conjunto de filtros, los que se conocen como filtros de *HAAR*,por toda la imagen a diferentes escalas.

Después en vez de agregar y combinar el resultado de todos estos filtros de alguna forma similar *HOG*, en este caso el resultado de aplicar cada filtro en cada posición y escala va a ser una componente del vector final de características.

Ilustración 9.FiltrosHAAR.

En ésta imagen podemos ver ejemplo de los filtros *HAAR* utilizados, éstos se aplican por toda la imagen en todas las posibles posiciones.

Los rectángulos que están marcados en negro representan aquella parte del filtro que van a darnos una contribución positiva al resultado final, mientras que los rectángulos marcados en blanco son aquellas partes del filtro que nos van a dar una contribución negativa al resultado final del filtro. Así cuando aplicamos un filtro en una posición dada en una imagen, el resultado final va a ser la diferencia entre la suma de intensidad de todos los pixeles que están situados en el rectángulo negro menos la suma de intensidad de todos los pixeles que están situados en el rectángulo blanco. En la *Ilustración 10*el cálculo sería: (200+200+250+250) – (100+100+50+50) = 600.

200	200	100	100	200	200	100	100
250	250	50	50	250	250	50	50
255	255	255	255	100	100	100	100
255	255	255	255	100	100	100	100
200	200	100	100	200	200	100	100
250	250	50	50	250	250	50	50
255	255	255	255	100	100	200	200
255	255	255	255	100	100	250	250

Ilustración 10.Ejemplo aplicación filtro HAAR.

El resultado puede requerir de normalización por el tamaño de la ventana del filtro para conseguir que los valores sean invariantes a cambios en los tamaños de los objetos y también para que características aplicadas a diferentes escalas en toda la imagen tengan valores comparables. Igualmente, y de forma previa puede ser necesario normalizar la imagen para conseguir también invariancia a cambios de iluminación.

Cada uno de estos filtros se aplicará en diferentes tamaños posiciones de la imagen, así vamos a poder detectar cambios de intensidad en cualquier posición de la imagen a diferentes escalas y orientaciones (vertical, horizontal). La aplicación de otros filtros en otras partes de la imagen nos va a permitir detectar otros tipos de cambios de contraste en diferentes zonas de la imagen. Vamos a ver un ejemplo de re escalado de los filtros:

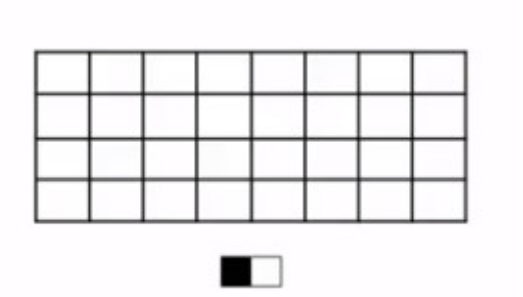

Ilustración 11.Re escalado de filtros.

Utilizando éste filtro de la *Ilustración 11* como ejemplo vamos a poder aplicarlo en la dirección horizontal cuatro escalas diferentes ya que el tamaño del rectángulo negro tiene que ser siempre igual al tamaño del rectángulo blanco y por lo tanto en la dirección horizontal solo va a ser posible generar escalas en tamaños dos, cuatro, seis y ocho para este filtro.

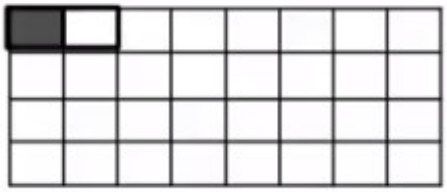

Imagen 12. Re-escalado de filtros 1.

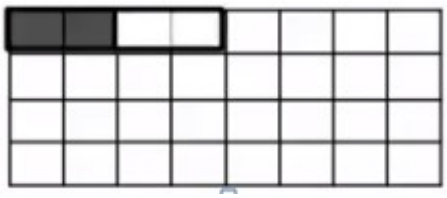

Imagen 13. Re-escalado de filtros 2

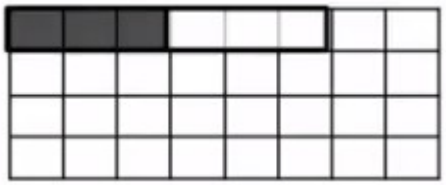

Imagen 14. Re-escalado de filtros 3.

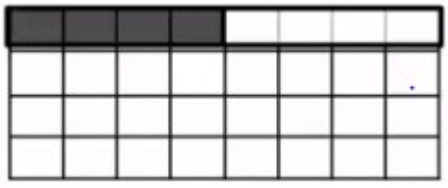

Imagen 15.Re-escalado de filtros 4.

Igualmente podemos escalar el filtro en la dirección vertical, y para cada nueva escala en vertical podemos volver a aplicar el mismo escalado que en horizontal. Así para esta imagen y con este filtro vamos a poder obtener un total de 16 escalas.

El resultado de aplicar cada uno de los filtros en cada una de estas escalas de posiciones es una de las características de *HAAR*. De esta forma la descripción global de la imagen es la concatenación de todas las características y podemos deducir solo con este ejemplo tan simple que el número de características y por lo tanto la dimensionalidad final del descriptor será muy elevada.

2.4 CLASIFICADORES

Una vez extraídos los descriptores, es necesario clasificar la información obtenida con el fin de discernir si los candidatos de una imagen representan el objeto a detectar. Para ello se emplean clasificadores los cuales implementan diferentes algoritmos para el procesamiento de la información obtenida.

Algunos de los clasificadores más utilizados son:

- *Adaboost*.
- SVM
- K-Vecinos

A continuación, se pasará a explicar los diferentes algoritmos de clasificación, pero en concreto nos centraremos en el funcionamiento de *Adaboost* ya que será el modelo de decisión empleado en este proyecto.

2.4.1 K-VECINOS

A continuación, pasaremos a explicar el proceso que sigue el algoritmo de K-vecinos.

Para empezar, como en muchos otros modelos, se necesitará de un conjunto previamente clasificado, para asignar el objeto a detectar a la clase a la cual pertenece el elemento más próximo del conjunto de entrenamiento. En la clasificación pueden surgir problemas de transición entre clases o elementos discordantes, como la mala segmentación o el ruido.

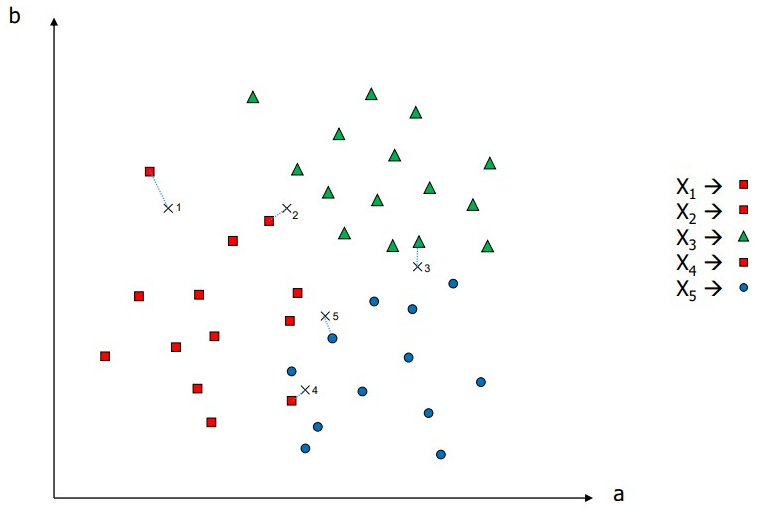

Ilustración 16. Distribución de características.

Como vemos en la *Ilustración 16*, el algoritmo:

1. Calcula la distancia desde el punto a clasificar a todos los puntos de entrenamiento.
2. Ordena las distancias.
3. Escoge las K menores.
4. Clasificar el punto en la clase predominante.

En cuanto a las ventajas e inconvenientes tener en cuenta que es un modelo intuitivo y fácil de implementar además que las regiones de decisión se pueden adaptar a cualquier forma. Por otro lado, dentro de los inconvenientes encontramos el almacenamiento de datos de entrenamiento en la fase de clasificación o la carga computacional de la clasificación.

2.4.2 SVM

Teniendo un conjunto de vectores, los cuales representan la información de diferentes clases (como podrían ser un conjunto de peatones y el fondo):

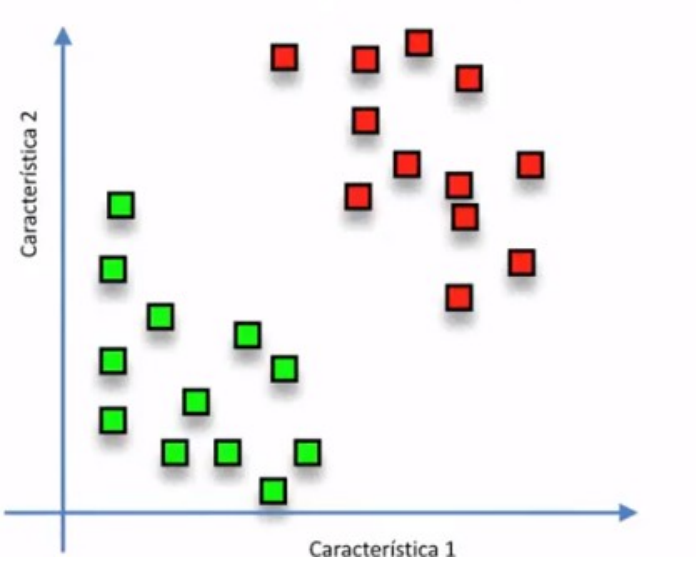

Ilustración 17.Representacion de características.

Los clasificadores lineales buscan como solución un “hiperplano” que separe las dos clases de la forma más óptima posible. En el caso concreto de dos dimensiones como el que ilustramos aquí, estas fronteras de separación van a ser siempre una línea, para tres dimensiones la solución va a ser un plano, y para dimensiones mayores o iguales a tres la solución general va a ser un “hiperplano”.

Para afrontar el problema de separación de clases encontramos diferentes modelos, por un lado tenemos lo que se conoce como modelos generativos que tratan de construir estas fronteras de separación entre clases a partir de estimar la función de densidad de probabilidad que podemos asociar a cada una de las clases (por ejemplo los clasificadores de Bayes).Por otro lado tenemos lo que conocemos como modelos discriminativos que tratan de clasificar las muestras sin tener que generar estas funciones de densidad de probabilidad de las clases, y por lo tanto encuentra la frontera a partir de un conjunto de entrenamiento que sea suficientemente representativo, dentro de ésta categoría encontramos las *Support Vector Machine* (SVM).

Los *support vector machines* o también de forma abreviada SVM son sistemas de clasificación binarios, es decir, nos permiten distinguir entre dos clases. Las características principales de los SVM es que son clasificadores lineales cuya solución se basa en encontrar un "hiperplano" que separe las dos clases a partir de unos vectores determinados que es lo que conocemos como vectores de soporte, los cuales serán algunos de los vectores del conjunto, los cuales poseerán unas características especiales.

Para elegir los vectores de soporte tendremos en cuenta el margen entre los planos que los contiene, como vemos en la *Ilustración 18,* sea máxima.

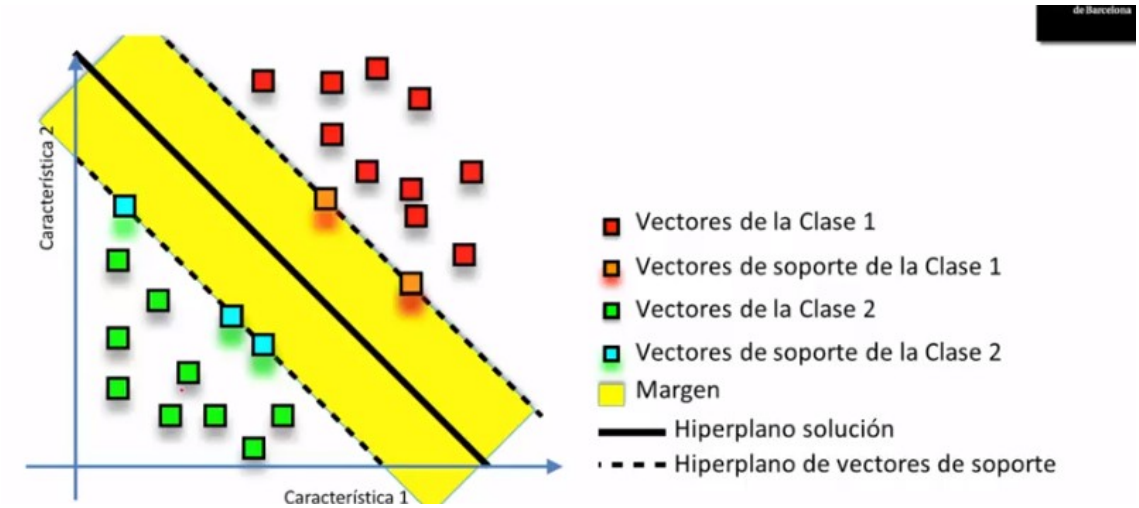

Ilustración 18. Clasificación SVM de vectores.

Una vez determinados los vectores de soporte de las diferentes clases para calcular el "hiperplano" que separe las clases realizará un problema de optimización, o sea, distancia media entre los "hiperplanos" que contienen los vectores de soporte.Por lo tanto podemos decir que la solución del *support vector machines* realmente es una solución a un problema de optimización en el que lo que buscamos optimizar es el margen entre las dos clases.

Dentro de las ventajas de SVM destacan su versatilidad y la gran variedad de implementaciones disponibles. Por otro lado, cabe destacar su mala escalabilidad, ya que el aprendizaje puede ser largo para problemas grandes.

2.4.3 ADABOOST

A continuación, pasaremos a analizar el método de clasificación de*Adaboost*, el cuál ha sido el empleado en éste proyecto debido a ciertas prestaciones que veremos en éste apartado.

Adaboost es un método de clasificación el cual nos permite tratar de forma muy eficiente con un número muy elevado de características durante el proceso de aprendizaje, lo cual nos vendrá estupendamente a la hora de trabajar con descriptores *HAAR*, ya que éstos exigen el procesamiento de una gran cantidad de características.

La base de éste clasificador es la combinación de diferentes clasificadores muy simples, pero que sin embargo la todos combinados nos va a acabar dando como resultado final un clasificador global queminimizará el error de clasificación. Por otro lado, cada clasificador va a dar un peso relativo a cada ejemplo para aprender dependiendo del resultado de los pasos anteriores. Así para cada nuevo clasificador simple que se va a aprender, se va a incrementar el peso de aquellos ejemplos que se han clasificado mal en los pasos anteriores mientras que se va a disminuir el peso de aquellos ejemplos que están bien clasificados, de ésta forma los clasificadores se irán centrando en aquellos ejemplos que no se han clasificado en paso anteriores. Veamos un ejemplo gráfico:

1. Tenemos un conjunto de entrenamiento con dos clases de características diferentes y vemos que es imposible trazar una línea que separe perfectamente a las dos.

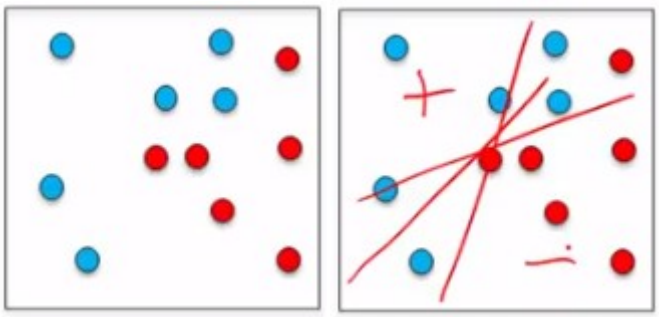

Ilustración 19. Conjunto de entrenamiento de dos clases características diferentes.

2. Aplicamos un clasificador simple que marcará un umbral, y todas las muestras que superen ese umbral se separarán de las que no lo hagan.

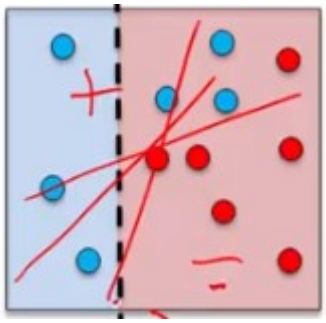

Ilustración 20. Clasificación de descriptores 1.

3. Cómo vemos hay tres muestras que han sido mal clasificadas, por lo que éstas recibirán un peso relativo mayor mientras que al resto un peso menor, de forma que la clasificación se ve condicionada por este hecho, como vemos en la imagen.

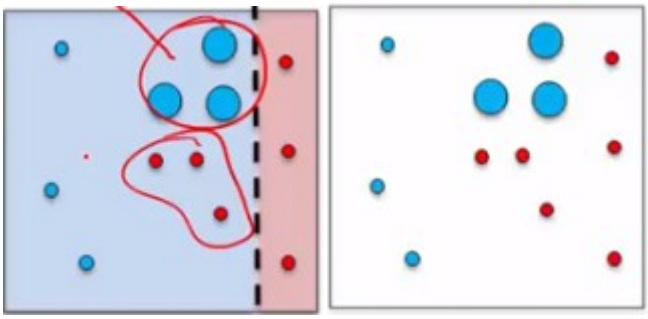

Ilustración 21. Clasificación de descriptores 2.

4. Después del último clasificador se han quedado tres muestras clasificadas, por lo que repetiremos el proceso anterior.

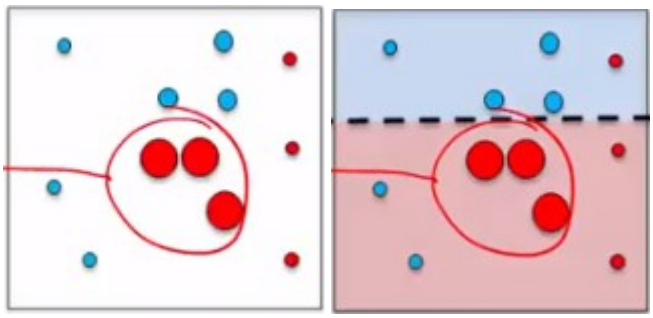

Ilustración 22. Clasificación de descriptores 3.

5. Finalmente todas las clasificaciones se agrupan en un clasificador global, de forma que las regiones donde exista una mayor cantidad de muestras de las clases A serán clasificadas como regiones A y lo mismo con la B.

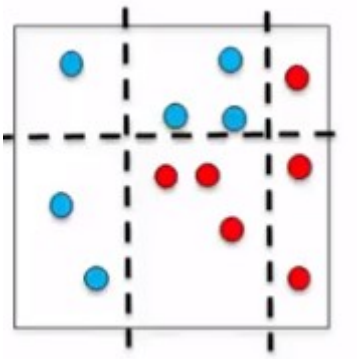

Ilustración 23. Clasificación de descriptores 4.

Una vez visto el funcionamiento de los clasificadores simples, vamos a pasar a analizar el comportamiento de la cascada de clasificadores completa.

Para detectar un coche en una imagen, tendremos que pasar una gran cantidad de ventanas a lo largo y ancho de la imagen, de las cuales solo un porcentaje muy pequeño encerrarán al coche o coches. Por lo que el objetivo va a ser descartar el mayor número posible de ventanaslas cuales no tienen un coche, para poder concentrar el máximo esfuerzo en las ventanas que tienen una mayor probabilidad.

La cascada de clasificadores nos va a permitir alcanzar este objetivo mediante una combinación secuencial de clasificadores, de forma que una imagensolo será detectada como coche si realmente es reconocida de forma correcta como coche por todos los clasificadores de la cascada. Con que uno de estos clasificadores rechace la imagen como cochela imagen quedará rechazada. De esta forma el primer clasificador va a recibir como entrada todas lasposibles ventanas de una imagen, y todas aquellas que rechace este primer clasificador quedarán descartadas y las que este clasificador acepte se pasarán como entrada al segundo clasificador. Este proceso se va a ir repitiendo hasta llegar al último clasificador donde solo las imágenes que sean reconocidas como coche por este último clasificador y por lo tanto, que también habrán sido reconocidas como coche por todos los anteriores, van a ser la detección final de que va a producir el detector.

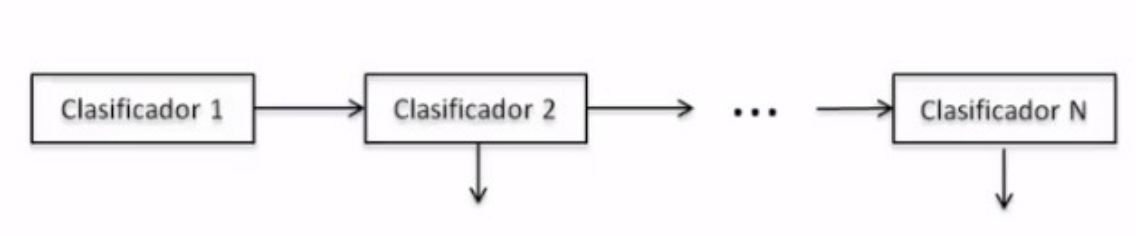

Ilustración 24. Cascada de "n" clasificadores.

Para conseguir un funcionamiento más óptimo de la cascada podemos fijar un objetivo de rendimiento diferente a cada clasificador en relación al número máximo de falsas detecciones que permitiremos para cada clasificador y alnúmero mínimo de detecciones correctas que debamos exigir. Ajuste el cuál veremos más adelante en la fase de entrenamiento del detector.

Por último, antes de terminar con *Adaboost* vamos a ver las principales modificaciones que realizaremos a las imágenes utilizadas para entrenar los clasificadores:

- Para reducir la variabilidad de todas estas imágenes todas ellas se normalizan a un tamaño fijo de 24 por 24 pixeles. Por lo tanto, esta va a ser la escala mínima que podremos utilizar como base para la detección de loscoches.

- Por otro lado, durante el proceso de aprendizaje también va a ser necesario normalizar las imágenes para reducir el efecto de iluminaciones diferentes.

3 DESARROLLO DEL PROYECTO

Una vez analizados todos los conceptos teóricos pasaremos a ver el desarrollo del proyecto paso a paso y dividido en diferentes apartados los cuales son enunciados en la *Tabla 2*:

	Fase	Sub fase
1	Creación de una biblioteca de imágenes	• Recopilación • Pre- procesado
2	Entrenamiento del clasificador	• Proceso entrenar • Errores/solución
3	Desarrollo del software	• Algoritmo detección • Algoritmo gestión • Interfaz gráfica

Tabla 2.Fases y Sub fases del proyecto.

Cabe tener en cuenta que se está obviando una fase inicial de investigación para determinar el método de trabajo a seguir y aclarar conceptos técnicos, además de una fase de familiarización con las librerías de OpenCV.

Fase 1.Creción de una biblioteca de imágenes.

Una vez pasada a fase de investigación se decide emplear descriptores *HAAR* debido a su alta eficiencia y su bajo coste computacional. El método óptimo de clasificación para éste tipo de descriptores es *Adaboost* ya que nos permite manejar la gran cantidad de características extraídas en los descriptores *HAAR* y nos permite mientras aprendemos también determinar y seleccionar las características que son más relevantes para la clasificación.

Para poder empezar a entrenar nuestro clasificador será necesario disponer de una biblioteca de imágenes con al menos 100 imágenes de los objetos a detectar, que en nuestro caso serán los coches de un parking, para ello seguiremos dos pasos: un primer paso de recopilación de la imágenes y a continuación el procesado de éstas para que dispongan de las características necesarias a la hora de entrenar el clasificador.

- **Recopilación**

El primer planteamiento de éste proyecto tenía como objetivo la gestión del parking exterior del CRAI (Universidad Politécnica de Valencia, campus de Gandía), por lo que empecé a recopilar dichas imágenes, pero pronto empezaron a surgir problemas.Por un lado la sombra proyectada por el propio edificio del CRAI generaba diferencias de iluminación entre las diferentes plazas, además de la sombra de los árboles, lo cual dificultaba la extracción de características debido cambios de intensidad en los píxeles de la imagen. Al fin y al cabo la iluminación es algo “controlable” ya que en función de la hora del día puedes conseguir resultados diferentes, pero el problema principal fueros los arboles del parking, ya que éstos dificultaban altamente la visión de la mayoría de las plazas. La primera solución que se planteo fue la de gestionar el parking con diferentes cámaras, las cuales nos permitiesen obtener diferentes ángulos desde los cuales visionar todas las plazas:

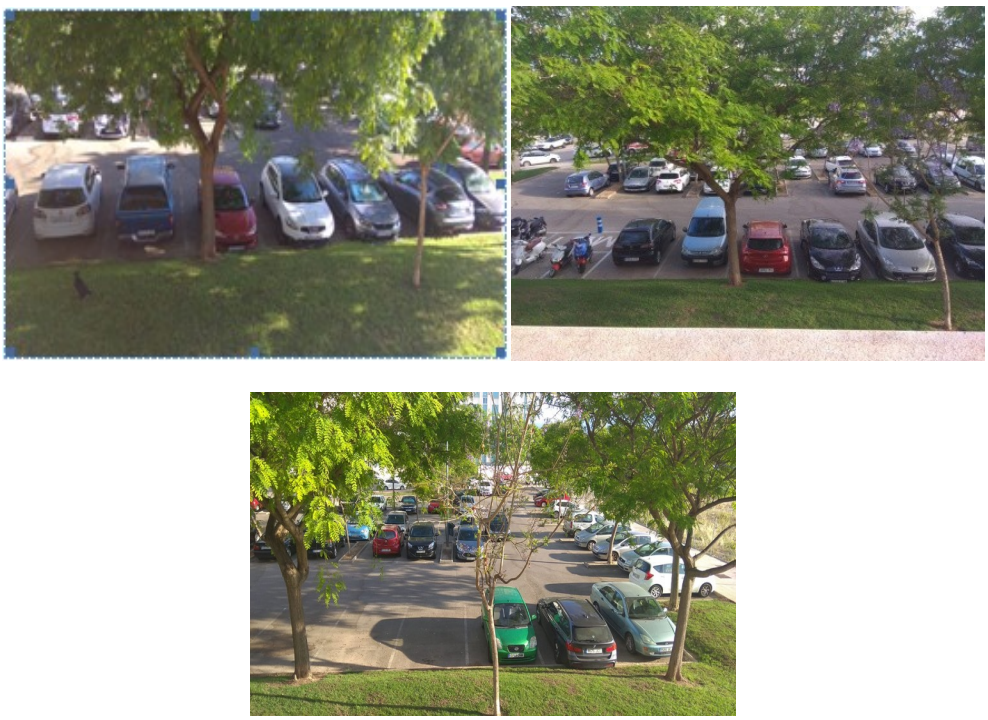

Ilustración 25. Foto del parking del CRAI (EPSG).

A pesar de utilizar diferente cámara, cubrir el 100% de las plazas es imposible o requiere un número demasiado alto de cámaras, lo que dificultaría en gran medida la gestión de éstas mismas. Es por ello que se planteo otra solución, gestionar simplemente la fila de plazas que nos ofreciese la mejor visión, en éste caso se trata de la fila de plazas indicada en la imagen.

Ilustración 26. Fila especificada, parking CRAI.

Pero a pesar de tener una visión más o menos clara de las plazas seguían existiendo cierta obstaculización por los árboles, además de una diferencia de iluminación generada por las sombras proyectadas por éstos mismos. A parte, la gestión de una sola fila de plazas no cumplía mis expectativas de gestionar un alto número de plazas.

La última solución, que ha sido la que finalmente se ha puesto en práctica, fue el cambio de parking, una solución bastante radical exigida por las circunstancias, ya que al fin y al cabo el objetivo de este proyecto demostrar el funcionamiento del software no la gestión de un parking en concreto.

Una vez tanteadas varias opciones decidí elegir elparking exterior del centro comercial Plaza Mayor en Gandía ya que ofrece una visión bastante clara y no cuenta con diferencias de iluminación preocupantes.

Ilustración 27.Parking del centro comercial Plaza Mayor (Gandía).

El único problema de esta ubicación es la alta afluencia de camiones y furgonetas lo cual limita de cierta manera la visión de todas las plazas, además de que implica entrenar el clasificador de forma que detecte tanto camiones y furgonetas como coches.

Así que de esta forma los siguientes días me dediqué a recopilar diferentes imágenes de los vehículos de dicho parking, lo cual dio como resultado un total de 125 fotos de vehículos.

Ilustración 28. Imágenes de coches sin procesar.

- **Procesado**

Una vez recolectada las imágenes suficientes, éstas tendrán que pasar por un proceso de modificación de sus características con el fin de hacerlas más eficaces en el posterior entrenamiento.

Para empezar, modificaremos el tamaño de las imágenes de forma que todas tengan la misma escala y pasaremos la imagen a escala de grises, para ello decido emplear Matlab debido a que es un programa con el que estoy familiarizado.

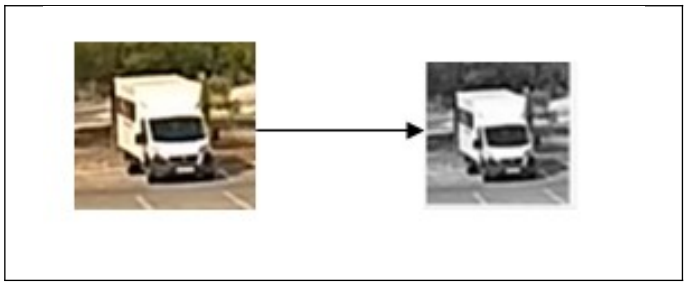

Ilustración 29. Transformación tras procesado mediante Matlab.

```
A = imread('tf20.jpg');
cd=rgb2gray(A);
j=imresize(cd,[50 50]);

imwrite(j,'tf20m.bmp');

imshow(j);
```

Código 1.Código Matlab para procesado.

Una vez realizadas estas modificaciones pasaremos a utilizar la aplicación que incorpora OpenCV llamada “créate samples”, la cual procesa las imágenes de forma que les aplica un re escalado y corrección de la iluminación y las agrupaen un vector de muestras positivas de forma que pueda ser utilizado en posteriores fases.

Para ello, lo primero que haremos será preparar los elementos necesarios:

- Todas las imágenes positivas (con positivas quiero decir las imágenes de los objetos a detectar) agrupadas en una carpeta.
- Un archivo de texto donde se indique el nombre del directorio donde se encuentran las imágenes positivas, el nombre de cada una y una serie de parámetros similares a los del siguiente ejemplo.
-

```
pos/pos-0.pgm 1 0 0 50 50
pos/pos-84.pgm 1 0 0 50 50
```

Ilustración 30. Ejemplo de lista de imágenes positivas.

Los parámetros que siguen al nombre de la imagen representan número de objetos en la imagen y las dimensiones de ésta.

Una vez tenemos todos los elementos requeridos, desde la terminal, entraremos en la carpeta “opencv” y seguiremos la ruta “C:\user\opencv\build\x64\vc14\bin”, donde se encuentra la aplicación “create_samples”.

```
C:\Users\ericb\OpenCV\build\x64\vc14\bin>opencv_createsamples -info C:\Users\ericb\Desktop\cascadas\cars3.info -num 125
-w 48 -h 24 -vec cars.vec
```

Ilustración 31. Captura de terminal 1 (Comando "create_samples").

Introduciremos el comando que aparece en la imagen, donde introducimos a la aplicación:

- El archivo de texto antes nombrado.
- El número de imágenes que vamos a introducir.
- Las dimensiones de las muestras que obtendremos a las salida de la aplicación
- El nombre del archivo ".vec" donde se agruparán las muestras
- NOTA: importante poner siempre las rutas de los archivos.

Ejecutamos la aplicación y obtendremos el siguiente resultado:

```
Info file name: C:\Users\ericb\Desktop\cascadas\cars3.info
Img file name: (NULL)
Vec file name: cars.vec
BG  file name: (NULL)
Num: 125
BG color: 0
BG threshold: 80
Invert: FALSE
Max intensity deviation: 40
Max x angle: 1.1
Max y angle: 1.1
Max z angle: 0.5
Show samples: FALSE
Width: 48
Height: 24
Max Scale: -1
RNG Seed: 12345
Create training samples from images collection...
Done. Created 125 samples
```

Ilustración 32. Captura de terminal 2 (Resultado de comando "create_samples").

Para visualizar las muestras obtenida introduciremos el siguiente comando:

```
C:\Users\ericb\OpenCV\build\x64\vc14\bin>opencv_createsamples -vec C:\Users\ericb\Desktop\cascadas\cars.vec -w 48 -h 24
```

Ilustración 33. Captura de terminal 3 (Comando "create_samples" para visualizar muestras).

Lo que nos dará como resultado la visualización de las muestras, después de todo el procesamiento:

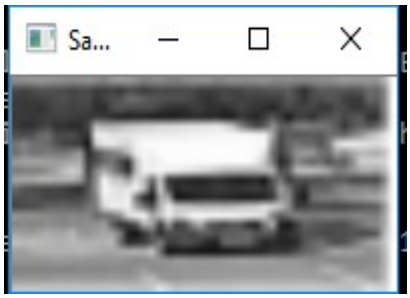

Ilustración 33. Captura de terminal 4 (Muestra resultante del comando "create_samples").

En las siguientes fases utilizaremos las muestras obtenidas en ésta fase para entrenar nuestro clasificador *Adaboost*.

Fase 2. Entrenamiento del clasificador.

En ésta fase veremos el proceso seguido para entrenar el clasificador en cascada de *Adaboost*, para ello emplearemos la herramienta "opencv_traincascade" que nos ofrece OpenCV, la cual incluye varias funcionalidades como elegir el cualel número de etapas por las que pasarán los candidatos antes de ser elegidos como objetos a detectar o variar algunos parámetros que veremos continuación.

Para empezar a entrenar el clasificador es necesario disponer de ciertos requisitos:

- Vector con las muestras positivas elaborado en la fase anterior.

- Un conjunto de muestras negativas, que en éste caso constara de unas 100 imágenes de objetos que no sean coches, las cuales serán utilizadas por el clasificador para determinar las características que definen los coches. Cabe decir que éstas imágenes las obtuve de una base de datos de OpenCV por lo que no necesité aplicarles un preprocesado.

- Por último un archivo de texto que recoja los nombres de todas las imágenes negativas y el directorio donde se encuentran, siguiendo la siguiente estructura:

```
neg/neg-119.pgm
neg/neg-198.pgm
neg/neg-155.pgm
```

Ilustración 34. Ejemplo de lista bg.

Una vez recopilados todos los requisitos vamos a ver el comportamiento de la funcionalidad “opencv_traincascade”.

La aplicación aplicará los filtros *HAAR*, mencionados en anteriormente, de diferentes tamaño y orientaciones en todas las imágenes positivas, de forma que almacenará las diferencias de intensidad en forma de descriptor que utilizará para generar las diferentes etapas de decisión por las que tendrán que pasar los candidatos para discernir si se tratan del objeto a detectar.

El comando utilizado para llamar a la aplicación es el siguiente:

```
C:\Users\ericb\OpenCV\build\x64\vc14\bin>opencv_traincascade -data data -vec C:\Users\ericb\Desktop\cascadas\cars.vec -bg C:\Users\ericb\Desktop\bg.
txt -numPos 125 -numNeg 219 -numStages 5 -w 48 -h 24 -featureType HAAR
```

Ilustración 35. Captura de terminal 5 (Comando “train_samples”).

Vamos a ver los elementos que introducimos en el comando:

- - data: representa la carpeta donde se van a ir volcando los datos (las diferentes etapas y la cascada completa).
- - vec: introducimos el vector de muestras positivas extraído gracias a “opencv_createsamples”.
- - bg : introducimos el archivo de texto con la ubicación de las imágenes negativas.
- -numPos: número de muestras positivas utilizadas en el entrenamiento para cada etapa de clasificador.
- -numNeg: Número de muestras negativas utilizadas en el entrenamiento para cada etapa de clasificador.
- -numStages: Número de etapas de cascada para entrenar.
- -featureType: Indicar el tipo de descriptores que utilizaremos, en nuestro caso *HAAR*.
- -w:Ancho de las muestras de entrenamiento (en píxeles). Debe tener exactamente el mismo valor que el utilizado durante la creación de las muestras positivas.
- -h: Alto de las muestras de entrenamiento (en píxeles). Debe tener exactamente el mismo valor que el utilizado durante la creación de las muestras positivas.

Una vez ejecutado el comando el sistema comenzará con el entrenamiento etapa a etapa de forma que nos aparecerá la siguiente información.

```
===== TRAINING 1-stage =====
<BEGIN
POS count : consumed   125 : 125
NEG count : acceptanceRatio    219 : 0.286275
Precalculation time: 6.08
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
|   2|        1|        1|
+----+---------+---------+
|   3|        1| 0.351598|
+----+---------+---------+
END>
Training until now has taken 0 days 0 hours 0 minutes 26 seconds.
```

Ilustración38. Captura de terminal 4 (Muestra resultante del comando "create_samples").

De forma que podemos visualizar el tiempo que ha tardado en entrenar la primera etapa, las veces que se han consumido las muestras positivas acompañadas por los resultados de *hitrate*ymax false alarmrate, los cuales determinará el comportamiento del sistema de entrenamiento, de forma que si tenemos un hit rate de 0,95 se aceptarán 5 de 1000 imágenes positivas erróneas durante el periodo de entrenamiento, y*maxalarmrate*representará el porcentaje máximo de imágenes negativas mal clasificadas.

En la primera prueba que hice me salió el siguiente mensaje:

```
===== TRAINING 4-stage =====
<BEGIN
POS count : consumed   125 : 125
NEG count : acceptanceRatio    3 : 0.0234375
Required leaf false alarm rate achieved. Branch training terminated.
```

Ilustración 37. Captura de terminal 4 (Muestra resultante del comando "train_samples").

Lo cual significa que se ha alcanzado la false alarm máxima y que el proceso de entrenamiento se termina. Para solucionar esto hay dos posibles soluciones, la más adecuada sería aumentar el número de muestras positivas o bien endurecer las

condiciones de entrenamiento subiendo el valor de *hit rate y maxalarmrate*. En mi caso seguí aumentando las muestras positivas hasta que me fue posible y a continuación aumente el *hit rate* a 0.98 y *max false alarm* a 0.7, de forma que introduciendo el siguiente comando conseguimos el resultado deseado.

```
C:\Users\ericb\OpenCV\build\x64\vc14\bin>opencv_traincascade -data C:\Users\ericb\Desktop\data -vec C:\Users\ericb\Desktop\cascadas\cars.vec -bg C:\
Users\ericb\Desktop\bg.txt -numPos 125 -numNeg 219 -numStages 5 -minHitRate 0.98 -maxFalseAlarm 0.7 -w 48 -h 24 -featureType HAAR
```

Ilustración 38. Captura de terminal 4 (Comando "create_samples" modificado).

Al terminar el entrenamiento dispondremos en la carpeta data de la cascada completa la cuál utilizaremos en la siguiente fase para desarrollar el detector.

Fase 3. Desarrollo del software.

Una vez llegados a este punto del proyecto es el momento de implementar la aplicación. Las funcionalidades de la aplicación serán:

- Detección de coches.
- Gestión de las plazas del parking
- Generación de un informe con los movimientos realizados en el parking.

Para ello emplearemos dos algoritmos, por un lado, el algoritmo de detección de coches y por otro la gestión de las plazas de parking. Antes de pasar a ver los algoritmos es necesario explicar de una forma superficial el funcionamiento y estructura de la aplicación.

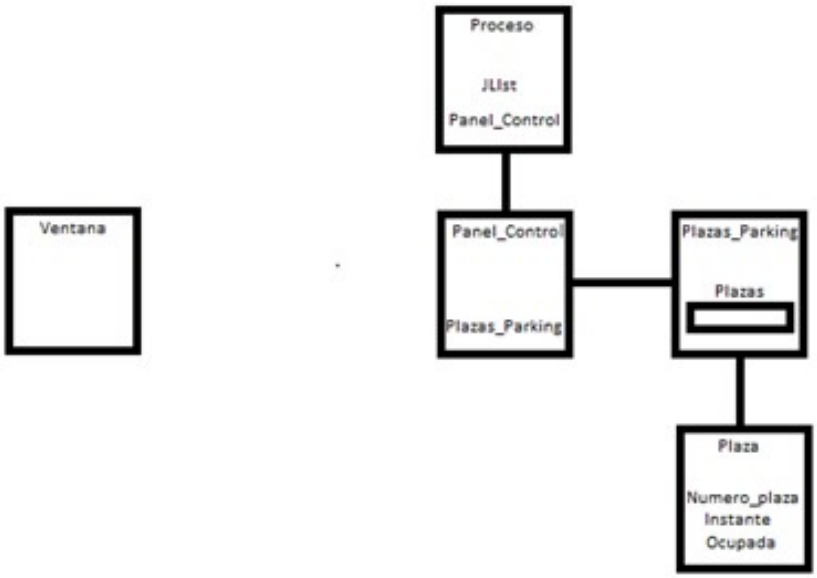

Ilustración 39. Diagrama de clases de la aplicación.

Como trabajaremos con un lenguaje orientado a objetos organizaremos la aplicación en clases, las cuales aparecen representadas en el diagramade la *Ilustración 39*. Paraempezar, veremos una breve explicación de cada clase:

- **Plaza:** los objetos de la clase “Plaza” serán utilizados para gestionar las plazas del parking, es por eso que como atributos contendrá un entero que nos indicara el número de la plaza, un objeto de tipo *Calendar*que nosalmacenará el instante en que se ocupa o desocupa la plaza y un *booleano* que pasará a *true* cuando la plaza esté ocupada.

- **Plazas_Parking:**esta clase contendrá como atributo un*ArrayList* de objeto de tipo Plaza e implementará métodos para manipular a este. Algunos de los métodos con los que cuenta son: ocuparPlaza, liberarPlaza, anyadirPlaza, eliminarPlaza, vaciar, Buscar, etc…

- **Ventana:** en cuanto a la interfaz he decidido separarla del resto del código para trabajar de una forma más ordenada, por lo que la clase “Ventana” contendrá todo el código referente al apartado gráfico. El diseño de la interfaz divide la pantalla en cuatro paneles, tres de los cuales serán implementados en ésta clase, el inferior, superior y principal.

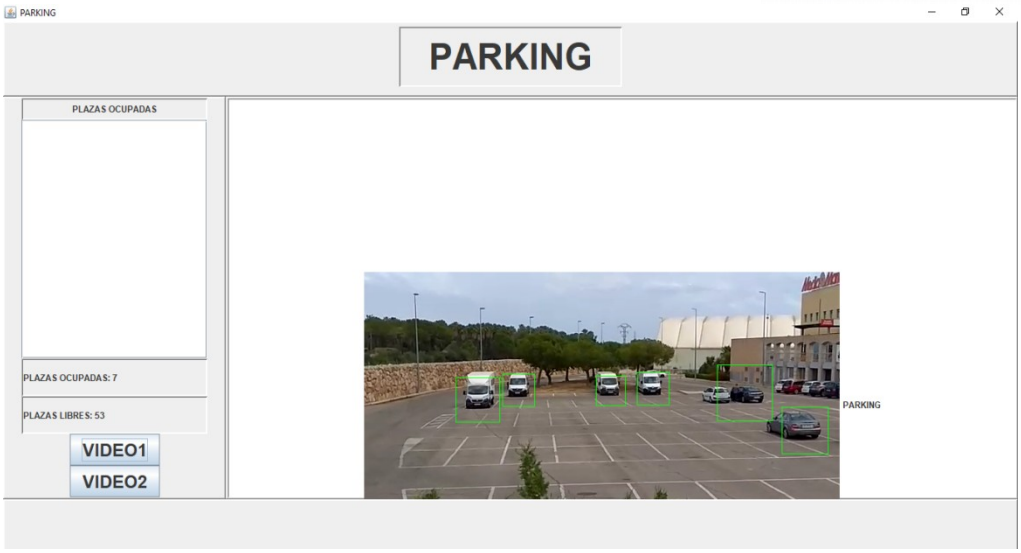

Ilustración 40. Interfaz de la aplicación.

Tanto el panel inferior como el superior son meramente decorativos, y en el panel principal dibujaremos el video con las detecciones.

- **Panel control:** En la clase anterior he mencionado que solo se implementaban tres de los paneles, el cuarto lo implementa la clase “Panel_control”. He decidido separar este panel de los demás ya que la clase contendrá una lista*JList*con los movimientos del párking, los cuales serán mostrados en pantalla junto con el número de plazas ocupadas y el de plazas libres y un objeto de tipo “Plaza_parking” para poder almacenar el número de plazas ocupadas.

  La clase implementará varios métodos, entre ellos encontramos dos imprescindibles:

  - Video/escogerVideo: estos métodos buscarán la ruta del video que seleccionemos y se lo pasarán a la clase “Proceso” para ser tratado. Para ello se empleará una pantalla de tipo buscador que almacenará el directorio del video, que se le será pasado al método VideoCapture.

```
1.  protected void Video (int v)
2.      {
3.         String s= null;
4.
5.         if (v == 1)
6.         {
7.             s= this.escogerVideo();
8.
9.             // Lanzar hilo con el vídeo seleccionado.
10.
11.            if (s != null)
12.            {
13.                                                        Proceso
    p= new Proceso (s, parent.getLabelImagen(), this);
14.            }
15.        }
16.     }
```

Código 1. Método Video.

```
1.  protected StringescogerVideo ()
2.      {
3.           // Muestra un cuadro de diálogo para escoger
    un fichero de vídeo.
4.          // En caso de éxito, devuelve la ruta completa
    del fichero.
5.          // Devuelve null en cualquier otro caso.
6.
7.
8.          // Selección de la ruta por defecto.
9.
10.         File f= new File (".");
11.         String ruta= null;
12.
13.         try
14.         {
15.                    ruta= f.getCanonicalPath();//  +  "\\
    Vídeos";
16.         }
17.         catch (IOException ex)
18.         {
19.             ruta= null;
20.         }
21.
22.          FileDialog d= new FileDialog (parent, "ESCOGER
    VÍDEO", FileDialog.LOAD);
23.         d.setDirectory (ruta);
24.         d.setVisible (true);
25.
26.         String carpeta= d.getDirectory();
27.
28.         if (carpeta != null)
29.         {
30.             return (carpeta + d.getFile ());
31.         }
32.         return null;
33.     }
```

Código 2. Método escogerVideo.

- generarInforme: creará un archivo de texto mediante las utilidades *FileWriter* y *FilePinter,* que se almecenará en la carpeta “Informes”.

```
1.  public voidgenerarInforme ()
2.      {
3.           // Genera un fichero de texto con los movimientos
    en las plazas
4.          // realizados durante la ejecución del vídeo.
5.
6.
7.          // Nombre del fichero en la carpeta "Informes".

8.
9.          Calendar c= Calendar.getInstance();
10.                                                          String
    nom_fichero= "Informe_" + String.format ("%1$tY_%1$tm_
    %1$td_%1$tH_%1$tM_%1$tS.txt", c);
11.         String ruta= "";
12.
13.         File f= new File (".");
14.
15.         try
16.         {
17.                              ruta= f.getCanonicalPath() + "\\
    Informes\\" + nom_fichero;
18.         }
19.         catch (IOException ex)
20.         {
21.             ruta= nom_fichero;
22.         }
23.
24.
25.         FileWriterfw= null;
26.           //s=  s  +  "  >>>  Plaza  "  +  p.getNumero()  +  "
    ocupada";
27.
28.         try
29.         {
30.            fw= new FileWriter (ruta);
31.            PrintWriterinforme= new PrintWriter (fw);
32.
33.
34.            // ENCABEZADO.
35.
36.
     informe.println ("====================================");
37.              informe.println ("   INFORME  DE  OCUPACION  DEL
    PARKING");
38.
     informe.println ("====================================\n\
    n");
39.
40.            String s="";
41.                                                      s= "FECHA:
    " + String.format("%1$td/%1$tm/%1$tY %1$tT\n\n", c);
42.            informe.println (s);
43.
44.
45.
46.            // MOVIMIENTO DE PLAZAS.
47.
```

```
48.            informe.println ("MOVIMIENTO DE PLAZAS");
49.            informe.println ("--------------------\n");
50.
51.            s= "";
52.
53.
     for (inti= 0; i <= this.historial.ocupacion() - 1; i++)
54.            {
55.
     s= String.format ("%1$td/%1$tm/%1$tY %1$tT", this.historia
    l.plaza (i).getInstante ());
56.                                        s= s + "    >>>    Plaza
    " + this.historial.plaza(i).getNumero();
57.                if (this.historial.plaza (i).estaOcupada())
58.                {
59.                    s= s + " ocupada.";
60.                }
61.                else
62.                {
63.                    s= s + " libre.";
64.                }
65.
66.                informe.println (s);
67.            }
68.
69.            informe.println ("\nFIN DE MOVIMIENTOS.");
70.
71.
72.            informe.close ();
73.            fw.close ();
74.         }
75.         catch (IOException ex)
76.         {
77.             // Mostrar error generando el informe.
78.                  JOptionPane.showMessageDialog(null, "Se  ha
    producido     un     error     generando     el     informe
    " + nom_fichero , "ERROR                              GENERANDO
    INFORME", JOptionPane.ERROR_MESSAGE);
79.         }
80.     }
81. }
```

Código 3. Método generarInforme.

- **Proceso**: por último, encontramos la clase “Proceso” la cuál contendrá tanto el algoritmo de detección como el de gestión. A continuación, se explicará más detalladamente.

Algoritmo de detección.

Teniendo en cuenta que hemos grabado un video previamente, emulando la cámara de seguridad de un párking, detectaremos todos los coches que aparecen en la imagen de forma más o menos eficiente.Para ello emplearemos las funcionalidades que nos ofrecen las librerías de *OpenCV,* las cuales nos permitirán aplicar el clasificador en cascada previamente entrenado. El lenguaje de programación será Java y el IDE *NetBeans*.

A continuación, se pasará a explicar el funcionamiento del software paso a paso apoyándonos en capturas extraídas del código fuente de la aplicación.

Por motivos de optimización introduciremos el algoritmo de detección en forma de proceso, dentro de un*thread,* de manera que podremos controlar la ejecución de éste proceso.Para ello :

- Nuestra clase implementará la interfaz*Runable,* para así poder utilizar el método *run.*

```
public class Proceso implements Runnable
```

Código 4. Proceso implementa a Runnable.

- En el constructor declararemos e iniciaremos el *Thread*que ejecutará nuestro proceso

```
1.          Thread th= new Thread (this);
2.          Stringname= th.getName();
3.          th.start ();
```

Código 5. Crear un hilo.

- Implementaremos nuestro proceso detector dentro del *método run* de*Runable*

```
1.  public void run ()
2.       {
3.         this.source= new VideoCapture (this.fichero);
4.
5.         if (this.source.isOpened())
6.         {
7.                 while(this.source.read (this.frame))
8.                 {
9.                     if (!this.frame.empty())
10.                    {
11.
12.                faceDetector.detectMultiScale(frame, faceDetections);
13.            for (Rectrect : faceDetections.toArray()) {
14. Imgproc.rectangle(frame, new Point(rect.x, rect.y), new Point(rect.x
    + rect.width, rect.y + rect.height),
         new Scalar(0, 255,0))
```

Código 6. Proceso de detección.

Como se puede apreciar en el código, capturaremos el video que queramos utilizar para la detección utilizando el método *VideoCapture* (la mayoría de los métodos que utilizaremos para la detección serán propios de las librerías de OpenCV). Una vez tengamos el video, mediante el método *read*lofragmentaremos en*frames,* los cuales serán utilizados uno a uno por*detectMultiScale,* al que introduciremos, aparte de los *frames*delobjeto*carDetections,* un*ArrayList*quealamacenará cada una de las decciones.

Para poder colocar recuadros verdes que contengan los objetos detectados, recorreremos el *ArrayList*de detecciones y extraeremos las coordenadas de cada objeto, las cuales consistirán de las posición de la esquina superior izquierda(‘x’),posición de la esquina inferior derecha(‘y’), la altura (‘heght’), y la anchura (‘width’). La extracción de las coordenadas será muy importante en la siguiente fase para desarrollar el algoritmo de gestión de plazas.

Por último, utilizaremos el método *rectangle* de la clase *Imgproc*para pintar los cuadros verdes de forma que utilizará las coordenadas para calcular las dimensiones de éstos.

```
faceDetector.detectMultiScale(frame, carDetections);
                            for (Rectrect : carDetections.toArray()) {
                                    //Clasificador c= new Clasificador
(this.cx, this.cy,this.cw,this.ch,this.x);

Imgproc.rectangle(frame, new Point(rect.x, rect.y), new Point(rect.x + r
ect.width, rect.y + rect.height),
                                            new Scalar(0, 255,0));

```

Código 7. Generar cuadros en los objetos.

Ilustración 41. Ejemplo de recuadros verdes.

Para terminar con el proceso de detección, almacenaremos el frame de video con sus respectivos cuadros verdes en forma de imagen, la cuál será mostrada en pantalla.

```
 Image imagen= HighGui.toBufferedImage(this.frame);
ImageIcon icono= new ImageIcon (imagen);
 this.label.setIcon (icono);
```

Código 8. Dibujar resultado de la detección en pantalla.

Una vez realizadas las detecciones, pasaremos a analizar el comportamiento del algoritmo de gestión que aprovechará la información obtenida por el algoritmo anterior para así poder almacenar las plazas ocupadas.

Algoritmo de gestión de plazas.

Como se ha mencionado anteriormente, la gestión de las plazas se realizará también en la clase “Proceso”, pero para éste necesitaremos implementar más métodos a parte de *run*. Si apreciamos en el diagrama de clases anterior podemos ver que “Proceso” contiene un objeto de tipo “Panel_control” el cuál será empleado por éste algoritmo.

Para empezar necesitamos saber donde se la ubicación de los objetos detectados, para ello recorreremos el ArrayList carDetections en busca de las coordenadas extraidas en el algoritmo anterior.

```
for (Rectrect : faceDetections.toArray()) {

Imgproc.rectangle(frame, new Point(rect.x, rect.y), new Point(rect.x
+ rect.width, rect.y + rect.height),
      new Scalar(0, 255,0));}
```

Código 9. Bucle que recorre el array detecciones.

Mediante los métodos“movimiento”, “contador” y “clasificar”clasificaremos los objetos detectados según sus coordenadas.

```
1.  public voidclasificar(){
2.
3. if (tamaño <60){
4.
    if(this.cx>310 && this.cx<330 && this.cy>135 && this.cy<145){
5.           this.t1=this.t1+1;
6.        }
7.
    if(this.cx>360 && this.cx<380 && this.cy>135 && this.cy<145){
8.             this.t2=this.t2+1;
9.        }
10.
     if(this.cx>390 && this.cx<410 && this.cy>135 && this.cy<145){
11.                  this.t3=this.t3+1;
```

Código 10.Método clasificar.

```
1.  public void contador (){
2.    if (this.t1==5 && this.p1==false){
3.
4.        this.control.ocupar_plaza(46,true);
```

Código 11. Método contador.

```
1.  }
2.        public voidmovimiento(){
3.
4.              if(this.p1==true){
5.              this.c1=this.c1+1;
6.              int cc1=this.t1-this.c1;
7.                                  if(cc46<-60){
8.                        this.control.ocupar_plaza(46,false);
```

Código 12. Método movimiento.

Los tres métodos emplearán a parte de las coordenadas, tres contadores por plaza que nos permitirá capturar el movimiento de los coches.

El método “clasificar” activará el contador “tn” de la plaza en cuestión cuando reciba las coordenadas pertinentes. El motivo de utilizar un contador es debido a que en algunos frames el clasificador confunde algunos objetos con coches es por eso que el objeto detectado deberá mantenerse por varios *frames* para ser declarado como coche.

Una vez activado el contador cuando se haya detectado en cinco frames éste pasará la condición de “contador” que mediante el objeto tipo “Panel_control” y el método “ocupar_plaza” marcará la plaza en cuestión con un *true*.

Cabe tener en cuenta que en ocasiones, cuando dos coches se encuentran muy próximos el clasificador lo clasifica como uno solo pero más grande.

Ilustración 42. Ejemplo de error por proximidad de coches.

Con el fin de solucionar éste problema en el “clasificar” se añade una condición que en función del tamaño del objeto los clasificará de una manera u otra. El tamaño del objeto es calculado gracias a las coordenadas y el método “área”.

```
1. public intarea( inth,int w){
2.  intarea= h*w;
3. return area;
4. }
```

Código 13. Método área.

El método movimiento activará el contador “cn” que de por sí irá 5 unidades por detrás de “tn”, si a la vez que contamos, almacenamos la resta “tn - cn” cuando este llegue a -60 sabremos que el contador “tn” ha dejdo de contar y que por lo tanto el coche ya no ocupa la plaza. El contador se poner a -60 ya que algunos frames no se detectan todos los coches y de esta forma hacemos el algoritmo más robusto frente a éstos errores.

Por último decir que como es obvio, los métodos se llamarán dentro del bucle que recorre “carDetections” para así realizar la clasificación objeto a objeto.

```
1. for (Rectrect : faceDetections.toArray()) {
2.
3.
   Imgproc.rectangle(frame, new Point(rect.x, rect.y), new Point(rect.x + r
   ect.width, rect.y + rect.height),
4.                                          new Scalar(0, 255,0));
5.                          /*
6.                          this.cy=rect.y;
7.                          this.cw=rect.width;
8.                          this.ch=rect.height;
9.
10.;
11.                         clasificar();
12.                         contador();
13.x                         movimiento();
```

Código 14. Llamada a los métodos.

Una vez clasificados todos los vehículos en sus pertinentes plazas se mostrará en pantalla tantos los movimientos realizados junto con la fecha y el momento. Además, como hemos visto antes, se generará un informe con estos movimientos para así tenerlos registrados.

```
==================================
  INFORME DE OCUPACION DEL PARKING
==================================
FECHA: 03/09/2018 19:03:48
MOVIMIENTO DE PLAZAS
--------------------
03/09/2018 19:03:36 >>> Plaza 20 ocupada.
03/09/2018 19:03:36 >>> Plaza 7 ocupada.
03/09/2018 19:03:36 >>> Plaza 5 ocupada.
03/09/2018 19:03:37 >>> Plaza 1 ocupada.
03/09/2018 19:03:37 >>> Plaza 29 ocupada.
03/09/2018 19:03:37 >>> Plaza 30 ocupada.
03/09/2018 19:03:37 >>> Plaza 62 ocupada.
FIN DE MOVIMIENTOS.
```

Ilustración 43. Ejemplo de informe.

4. APLICACIONES FUTURAS.

En este apartado hablaremos de como se podría desarrollar el proyecto en un futuro y se dividirá en dos apartados. Hablaremos tanto de las funcionalidades futuras de la aplicación como de los cometidos que podría desempeñar si se utilizase como herramienta.

En cuanto a las funcionalidades de la aplicación, lo que más interesante me parece sería incluir detección y lectura de matrículas aplicando diferentes técnicas de visión artificial, y junto con un algoritmo de seguimiento de coches poder asociar a una plaza la matrícula del coche aparcado. De ésta forma podríamos no solo tener controlada a la gente que aparca en nuestro parking, además incluir funcionalidades como ayuda para encontrar coches en el parking, levantar la valla a los coches autorizados, registro de incidencias asociadas a una matrícula etc…

Otra funcionalidad que podríamos añadir se podrían basar en obtener información un poco más avanzada de los coches, como tamaño, forma, color, marca lo que nos permitiría tener una mayor precisión en la gestión de los coches. Pero lo más interesante en cuanto a extraer información más avanzada podría ser detectar la orientación de los coches y de esta forma saber si está mal aparcado.

Vamos a poner un ejemplo de una aplicación con todas estas funcionalidades. Un usuario autorizado para aparcar llega a su parking habitual, se acerca a la valla y como la matricula es detectada y se encuentra en la base de datos como autorizada la valla se sube automáticamente. El usuario busca en la aplicación las plazas libres y a parca su coche, una vez vuelva al coche utilizará de nuevo la aplicación para recordar la ubicación del coche. Por otro lado, el encargado de gestionar el parking detecta que le usuario en cuestión ha aparcado el coche ocupando dos plazas ya que el sistema le ha notificado que un coche con unas características concretas (marca, color, tamaño) ha aparcado incorrectamente. Al conocer la matrícula, si se trata de un usuario conocido se le comunicará personalmente, si no lo es un mensaje automático puede sonar por los megáfonos describiendo las características del coche y la infracción.

Estas han sido algunas funcionalidades que podríamos incluir a nuestra aplicación, ahora veremos como podríamos utilizarla tal cual la hemos diseñado.

Actualmente encontrar plaza de aparcamiento en ciudades grandes es muy costoso ya que el número de coches en España ha crecido mucho en los últimos años y parece que va a seguir creciendo, es por eso que la aplicación podría jugar un papel clave en este problema. Colocando cámaras de seguridad en farolas y calles y utilizando el software diseñado en este proyecto se podría gestionar todas las plazas públicas de la ciudad y de esta forma conseguir una conducción más cómoda y segura en las ciudades.

Ilustración 44. Noticias del periódico "El país".

Otra aplicación que tendría el software sería la mencionada en el inicio del proyecto. Sustituir a los sensores de movimiento de los parkings privados.

5. CONCLUSIÓN.

Una vez finalizado el proyecto podemos decir que el resultado has sido un éxito, ya que se han conseguido cumplir los objetivos que nos marcamos en un inicio. Para ello me ha sido necesario no solo aprender una gran cantidad de conocimientos informáticos y sobretodo relacionados sobretodo con la programación orientada a objetos, si no que además he tenido que informarme acerca de campos bastante experimentales como la visión artificial o *machine learning.*

Pero a pesar de que los resultados han sido exitosos, la aplicación aun necesitaría mejorar en muchos aspectos. Uno de los mayores problemas que se me han presentado ha sido la falta de precisión del clasificador. A pesar de que el detector detecta la mayoría de los coches, en muchas ocasiones se deja alguno, o detecta coches fantasmas. Este es debido en parte al poco volumen de la biblioteca de imágenes, ya que la mayoría de los detectores profesionales utilizan en torno a 1000 imágenes, y a la calidad de ésta misma.

Por otro lado, para gestionar el 100% de las plazas del parking es necesaria la utilización de varias cámaras, ya que en muchas ocasiones los propios coches tapan la perspectiva del coche de atrás. Pero el mayor problema que presenta la utilización de varias cámaras es la sincronización de éstas mismas, ya que deberían trabajar sobre la misma línea de tiempo y este proceso en el planteamiento parece bastante complejo.

En definitiva, la aplicación requiere de mucho mas tiempo y conocimientos para ser 100% productiva. Aun así, en este proyecto se ha realizado un prototipo que consigue unos resultados bastante buenos, y que sin duda es una perfecta base para empezar un proyecto más ambicioso.

En cuanto a mí, personalmente me he llevado una gran cantidad de conocimientos técnicos, que sin duda me serán de gran ayuda para orientar mi carrera profesional.

6. BIBLIOGRAFÍA.

- Algorithms for Image Processing and Computer Vision (OpenCV) - 2nd Edition. (J.R Parker).
- OpenCV 3.0 Computer Vision with Java (D.L Baggio).
- OpenCV 2 Computer Vision Application Programming Cookbook (R Laganiere)
- A Practical Introduction to Computer Vision with OpenCV(Dawson/Howe).
- PracticalOpenCV (Brahmbhatt).
- Tutoriales de OpenCV (https://es.scribd.com/document/239928365/OpenCV-OpenCV-Documentacion-Tutoriales-2-4-9).
- Documentación OpenCV (https://opencv.org/).
- Temario de la asignatura "Tratamiento digital de imagen y video" (JoseIgnacionHerranz).
- Fragmentos de código (github.com).
- DESARROLLO DE UNA APLICACIÓN MÓVIL PARA LE DETECCIÓN Y CLASIFICACIÓN DE HOJAS DE ÁRBOLES (Jaume Blanco Alambiaga).
- Visión por computador (J Vélez Serrano).