UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE
TELECOMUNICACIÓN Y ELECTRÓNICA

PROYECTO FIN DE CARRERA

Aplicación para terminales Android de detección de señales de limitación de velocidad

TITULACIÓN: Telemática
TUTORA: Dra. Itzíar Alonso González
AUTOR: Isaac Reyes Villa
FECHA: Diciembre 2012

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE
TELECOMUNICACIÓN Y ELECTRÓNICA

PROYECTO FIN DE CARRERA

Aplicación para terminales Android de detección de señales de limitación de velocidad

Presidente/a: Secretario/a: Vocal:

Tutora: Autor:

NOTA:

TITULACIÓN: Telemática
TUTORA: Dra. Itzíar Alonso González
AUTOR: Isaac Reyes Villa
FECHA: Diciembre 2012

Índice general

Índice de figuras

Índice de códigos

Índice de tablas

Parte I.

Memoria

Capítulo 1

Introducción

1.1. Visión por ordenador o visión artificial

La visión por ordenador es un campo que incluye métodos para adquirir, procesar, analizar y comprender imágenes y, en general, datos de grandes dimensiones del mundo real con el fin de producir información numérica o simbólica, por ejemplo, en forma de decisiones. Una de las áreas de desarrollo de este campo ha sido la de duplicar la capacidad de la visión humana de percibir y comprender una imagen mediante medios electrónicos. Esta interpretación de la imagen puede considerarse como la separación de la información simbólica de datos de imagen mediante modelos construidos con la ayuda de la geometría, la física, la estadística, y la teoría del aprendizaje. Igualmente, se describe la visión por ordenador como la tarea de automatizar e integrar una amplia gama de procesos y representaciones para la percepción de la imagen.

Las aplicaciones van desde tareas tales como sistemas industriales de visión artificial que, por ejemplo, comprueban las botellas que pasan por una línea de producción, a la investigación en inteligencia artificial y los ordenadores o robots que lleguen a comprender el mundo que les rodea. Los campos de visión por computador y de procesamiento de imágenes tienen una superposición significativa. La visión por ordenador cubre la tecnología básica de análisis automatizado de imágenes que se utiliza en muchos campos. El procesamiento de imágenes se refiere generalmente a un proceso de combinación de análisis automatizado de imágenes con otros métodos y tecnologías para proporcionar la inspección automatizada y guiado de robots en aplicaciones industriales.

Como disciplina científica, la visión artificial se ocupa de la teoría detrás de los sistemas artificiales que extraen información de las imágenes. Los datos de imagen pueden adoptar diversas formas, tales como secuencias de vídeo, puntos de vista de varias cámaras o datos multidimensionales desde un escáner médico.

Como disciplina tecnológica, la visión artificial trata de aplicar sus teorías y modelos para la construcción de sistemas de visión por computador. Alguno ejemplos de aplicaciones de visión artificial incluyen sistemas para:

- Controlar procesos. Por ejemplo, un robot industrial.
- Navegación. Como en un vehículo autónomo o un robot móvil.
- Detección de eventos. Por ejemplo, para vigilancia o recuento de personas.
- Organización de la información. Como por ejemplo en bases de datos de indexación de imágenes y secuencias de imágenes.
- Modelado de objetos o entornos. Como imágenes médicas o modelado topográfico.
- Interacción. Por ejemplo, la entrada a un dispositivo de interacción humano-máquina.
- Inspección automática. En aplicaciones de fabricación.

Los subdominios de la visión por ordenador incluyen reconstrucción de escenas, detección de eventos, seguimiento por cámara, reconocimiento de objetos, aprendizaje, indexación, estimación de movimiento, y restauración de imágenes.

En la mayoría de las aplicaciones prácticas de visión por ordenador, los equipos están preprogramados para resolver una tarea determinada, pero los métodos basados en el aprendizaje se están convirtiendo en algo cada vez más común.

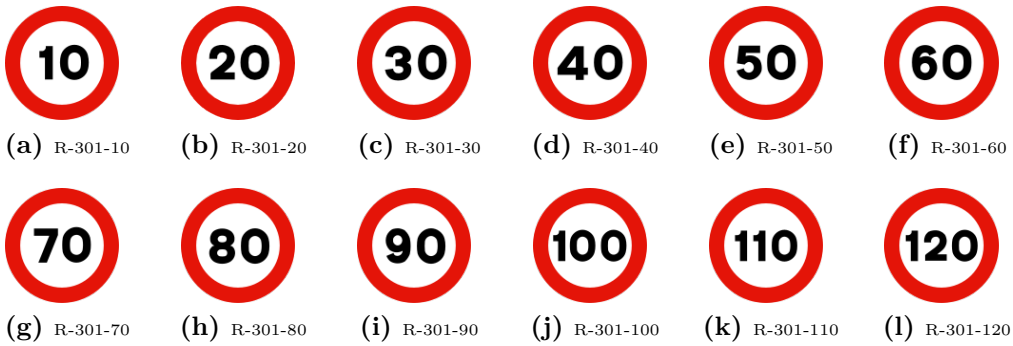
1.2. Señales de tráfico

Las señales de tráfico son los signos usados en la vía pública para proporcionar la información necesaria a los usuarios que transitan por un camino o carretera, en especial a los conductores de vehículos y peatones. Con el aumento del volumen de tráfico desde la década de 1930, muchos países han adoptado signos pictóricos o signos simplificados de otra manera y los han estandarizado para facilitar el tráfico internacional donde las diferencias idiomáticas podrían crear barreras, y en general para ayudar a mejorar la seguridad vial. Dichos signos gráficos utilizan símbolos (a menudo siluetas) en lugar de palabras y suelen basarse en los protocolos internacionales. Esta señalización se desarrolló por primera vez en Europa, y ha sido adoptada por la mayoría de los países en diferentes grados.

En 1968, los países europeos firmaron el tratado de la Convención de Viena sobre la circulación por carretera [1] con el fin de estandarizar las normas de tráfico en los países participantes con la intención de facilitar el tráfico internacional por carretera y aumentar la seguridad vial. Una parte del tratado fue la Convención de Viena sobre señalización vial [2], que define los signos y señales de tráfico. Como resultado, en Europa Occidental las señales de tráfico están bien estandarizadas, aunque todavía hay algunas excepciones específicas de cada país, que en su mayoría datan de la época previa a 1968.

El principio del estándar de señales de tráfico europeo es que se deben utilizar las mismas formas y colores para indicar los mismos fines. Las formas triangulares (fondo blanco o amarillo) se utilizan en señales de advertencia. Además, la convención de Viena permite una forma alternativa de señales de alerta, un rombo, que se utiliza muy poco en Europa (Irlanda). Las señales de prohibición en Europa son redondas con un borde rojo. Las señales informativas y otros signos secundarios son de forma rectangular.

En este proyecto fin de carrera se trabajará con las señales de limitación de velocidad que se engloban en el grupo de señales de prohibición. En España estas señales están definidas por la norma R-301 [3]. En la figura 1.1 se muestran las distintas señales que recoge esta norma.



(a) R-301-10 **(b)** R-301-20 **(c)** R-301-30 **(d)** R-301-40 **(e)** R-301-50 **(f)** R-301-60

(g) R-301-70 **(h)** R-301-80 **(i)** R-301-90 **(j)** R-301-100 **(k)** R-301-110 **(l)** R-301-120

Figura 1.1.: Señales de limitación de velocidad (Norma R-301).

1.3. Terminales móviles y lenguajes de programación

Los teléfonos móviles que se comercializan en la actualidad, los denominados *smartphones*, distan mucho de los modelos de la llamada primera generación que aparecieron allá por los años ochenta. Estos móviles tienen funciones que van mas allá de realizar llamadas o el envío de mensajes cortos de texto. Actualmente, existen móviles

mucho más potentes que la mayor parte de los ordenadores que existan en aquel entonces. Estamos en una época de continuas innovaciones, en una constante y voraz lucha de las compañías tecnológicas más punteras por el mercado de la telefonía, mercado que se ha convertido en prioritario, siempre en pleno auge. Las compañías buscan ofrecer un dispositivo cada vez mas potente, intentando mantener unos precios ajustados, de acuerdo con las leyes de oferta y demanda, así como la competencia entre ellas. La oferta existente en la actualidad esta marcada por los teléfonos con procesadores de doble núcleo (empiezan a aparecer con cuatro núcleos), cámaras de foto con más de cinco mega-píxeles, pantallas de alta definición y multi táctiles.

Para gestionar estos teléfonos inteligentes se requiere de sistemas operativos en constante evolución que permitan controlar las nuevas funcionalidades implementadas. Los sistemas operativos cambian según los fabricantes, destacando los sistemas Android (propiedad de Google Inc.) [4], IOS (propiedad de Apple Inc.) [5], Symbian (propiedad de Nokia desde 2008) [6], Blackberry OS (desarrollado por Research In Motion) [7] y Windows Phone (propiedad de Microsoft) [8]. Todos estos sistemas tienen en común que una parte de su código es accesible para cualquier desarrollador, de manera que es posible la creación de aplicaciones totalmente compatibles con el dispositivo que aproveche aquellas características liberadas por el diseñador del mismo. En este sentido, si hubiera que destacar un sistema operativo entre todos, sera Android, pues se trata de un sistema operativo completamente abierto al desarrollo de aplicaciones y nuevas funcionalidades, sobre todo al incluir un núcleo (*kernel*) basado en el sistema *Linux*, lo que lo convierte en un sistema bastante robusto. El numero de terminales vendidos con este sistema operativo esta creciendo de manera exponencial, ya que las compañas fabricantes de teléfonos móviles ven en el suficiente potencia como para gestionar las nuevas funcionalidades de sus dispositivos así como una gran libertad de personalización debido a su naturaleza abierta.

Entre los distintos lenguajes que se encuentran para programar terminales basados en estos sistemas operativos están: *C*, *C#* y *C++*, *.NET*, *Objetive-C*, *Python*, *Java* y sus distintas variantes (sea *J2EE* o *Java* de *Android*, entre otros). De todos estos lenguajes, para el proyecto fin de carrera que se presenta en esta memoria, se ha elegido *Java* de *Android* debido a que la naturaleza libre de este sistema operativo ha hecho posible que se hayan podido portar las librerías OpenCV [9], haciéndolas accesibles a este tipo de dispositivos.

OpenCV (*Open Source Computer Vision*) es una biblioteca libre de visión artificial desarrollada originalmente por Intel. Su publicación se da bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas. Se ha diseñado para ser computacionalmente eficiente

y con un fuerte enfoque en la creación de aplicaciones en tiempo real. Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos, calibración de cámaras, visión estéreo y visión robótica

1.4. Objetivo del proyecto

En los últimos tiempos se han producido notables mejoras tanto en las infraestructuras viales como en las prestaciones de los vehículos. Esto junto con el continuo aumento del parque móvil que aumenta el número de factores a los que tiene que estar pendiente el conductor origina un aumento de las infracciones por exceso de velocidad, en su mayoría por despiste o por no haber visto a tiempo una señal. El exceso de velocidad es uno de los factores que origina más accidentes graves en nuestro país.

Con esta aplicación se pretende realizar un sistema que sea capaz de detectar e identificar las señales verticales que marcan el límite de velocidad de una vía y comprobar hasta que punto se puede conseguir que el proceso se realice en tiempo real para, en un futuro, poder realizar un sistema integrado de bajo coste que, con ayuda del receptor GPS que incorporan la mayoría de terminales Android, indique al conductor si va a una velocidad excesiva.

1.5. Peticionario

El proyecto fin de carrera presentado en esta memoria ha sido elaborado a petición de la Escuela de Ingeniería de Telecomunicación y Electrónica, para la obtención del título de Ingeniero Técnico de Telecomunicación especialidad Telemática.

1.6. Organización de la memoria

La memoria que analiza la información y el desarrollo de este proyecto se divide en ocho capítulos, el pliego de condiciones y el presupuesto. Además, al final del tomo se presentan unos anexos que muestran la representación UML de algunas clases de OpenCV, el diagrama de flujo de la funcionalidad principal de la aplicación y el diagrama de clases de la aplicación. A continuación se comentan brevemente los capítulos incluidos:

- En el capítulo 1 se realiza una introducción al campo de la visión por ordenador o artificial y a la definición de las señales que en la actualidad rigen el tráfico rodado. Además, se muestran las distintas opciones de sistemas operativos para terminales móviles y los distintos lenguajes de programación empleados para el diseño de aplicaciones en dichos terminales. Igualmente, se especifica el objetivo principal del proyecto.

- En el capítulo 2 se hace un repaso a los distintos estudios realizados en los últimos años en el campo de la visión artificial, prestando mayor atención a aquellos que van en la línea del objetivo de este proyecto fin de carrera. Se explicarán las distintas técnicas de detección por color y por forma y como se emplean en la detección de señales de limitación de velocidad.

- En el capítulo 3 se estudia la librería de visión artificial OpenCV. En el se verán los conceptos fundamentales de la API de la librería y se realizará un estudio de los diversos módulos y sus respectivas clases disponibles para la plataforma Android.

- En el capítulo 4 se realizará un estudio de la plataforma Android visualizando sus fundamentos y los paquetes y clases que serán de interés en la realización del proyecto fin de carrera. Seguidamente se explicará como incluir la biblioteca OpenCV en un proyecto Android y, finalmente se mostrarán algunos ejemplos de aplicaciones simples de Android empleando algunas aplicaciones simples para demostrar el uso de OpenCV.

- En el capítulo 5 se explicará el proceso para la creación del fichero que almacena las características que definen a la señal de limitación de velocidad. Se indicará como se realiza el proceso de captura de datos (grabación de vídeo), la extracción de los frames de esos datos y su procesado para adaptarlos a las herramientas de "entrenamiento" que proporciona OpenCV. Igualmente, se explicará el algoritmo de reconocimiento óptico de caracteres que se empleará en la identificación de los dígitos de la señal de tráfico.

- El capítulo 6 consiste en el análisis previo y funcional de la aplicación. En el análisis previo se trata la solución adoptada siendo desglosada en bloques generales. Estos bloques serán, a su vez, divididos en el análisis funcional, profundizando en las funciones que realiza cada uno.

- En el capítulo 7 se proporciona, por medio del análisis orgánico, una visión pormenorizada de las clases implementadas en el código fuente de la aplicación. Asimismo, se analizan, de manera exhaustiva, las relaciones existentes entre estas

clases. Cada clase sera tratada individualmente, obteniendo una vista completa mediante gráficos UML.

- En el capítulo 8 se muestra el uso de la aplicación indicando la función de las distintas opciones del menú. Además se realizará una evaluación de la aplicación indicando cuales son las condiciones para que se detecte una señal o no. Finalmente se mostrarán las conclusiones a las que se ha llegado una vez probada la aplicación.

- Finalmente se presentan el pliego de condiciones, que especifica que requisitos ha de satisfacer la aplicación, y el presupuesto, que detalla los costes totales del proyecto.

Capítulo 2

Estudio de las Técnicas de Detección de Señales

2.1. Introducción

En los últimos años se han realizado una gran variedad de trabajos e investigaciones en sistemas de reconocimiento de señales de tráfico, algunos de ellos centrados en el objetivo de conseguir sistemas que realicen procesado en tiempo real [10, 11, 12, 13, 14, 15, 16, 17]. La mayoría de estos sistemas de tiempo real consisten en una cámara situada en el parabrisas, que toma imágenes del entorno, y un ordenador (a menudo portátil), que lee y procesa los frames de vídeo de la cámara (véase figura 2.1).

(a) (b)

Figura 2.1.: Sistema de reconocimiento en tiempo real típico. a) Ordenador. b) Cámara.

En este capítulo, se dará una visión general de los sistemas, técnicas y normas que se utilizan para el reconocimiento de señales de limitación de velocidad. Estos sistemas de reconocimiento generalmente constan de dos etapas principales: (1) la detección de la señal y (2) la identificación de los dígitos numerales. La fase de detección se caracteriza por la búsqueda dentro de la imagen de círculos de borde rojo y fondo

blanco con uno o varios elementos negros dentro de el. Están fase será la encargada de determinar si alguna región de la imagen contiene el tipo de señal que se desea encontrar y posteriormente pasar al proceso de identificación sólo la zona de la imagen donde se ha detectado una señal (región de interés[1]). Esta labor se puede realizar utilizando segmentación de color, coincidencia con plantillas, detectores de características preentrenados, SVM (*Support Vector Machines*), o una combinación de varios de estos métodos.

Para la fase de identificación se utilizan métodos de OCR (reconocimiento óptico de caracteres) que se pueden basar en Redes Neuronales Artificiales, modelos difusos y coincidencia local con plantillas o reconocimiento por escaneo de líneas entre otros.

2.2. Espacios de color. Segmentación de color

Normalmente, la detección de círculos en una imagen se considera un proceso de cálculo exigente y costoso, más cuanto mayor sea la imagen. Por esta razón, muchos de los sistemas existentes aplican segmentación de la imagen previamente a la detección de círculos. La etapa de segmentación de color crea umbrales en la imagen basándose en mecanismos y/o espacios de color específicos. La imagen resultante consistirá en regiones con píxeles blancos que indican los objetos con un color similar al que se está buscando. En este caso, similar al rojo del borde de las señales de limitación de velocidad. Para la creación del umbral se han empleado diversos espacios de color.

2.2.1. RGB / Escala de grises

RGB es el espacio de color más directo para el procesamiento de imágenes. Consta de tres canales de color primarios aditivos: rojo, verde y azul (*Red*, *Green*, *Blue*). Sin embargo, rara vez se utiliza en la segmentación de color directa en este ámbito [18]. En [12] y [19], se emplean sistemas que configuran filtros de imagen basados en el espacio de color para extraer tres colores de la imagen dada: rojo, blanco y negro. Esta informaciones de color se almacenan individualmente. La idea de estos algoritmos es comprobar si la región candidata es una señal de limitación de velocidad comprobando si existe una región blanca dentro de una región roja y a su vez una región negra dentro de la blanca. En [20], se convierten las imágenes directamente desde el RGB estándar en una imagen en escala de grises. Realizan esto porque los autores asumen que las áreas blancas dentro de las señales son únicas y suelen aparecer más brillantes

[1]RoI (por sus siglas en inglés)

que otras regiones. Crean un umbral en la imagen en gris para generar una imagen binaria para, a continuación averiguar la región blanca más probable como candidata para los procesos siguientes. En el documento [10] también se hace uso de la imagen en escala de grises al inicio del sistema para obtener los bordes de la imagen y realizar la segmentación de las regiones candidatas.

Uno de los principales inconvenientes del espacio de color RGB es su alta sensibilidad a las condiciones de luz [21, 22, 18, 23]. Por este motivo, el resultado de la segmentación puede variar en diferentes condiciones de luz debidas a las condiciones climáticas y/o las posibles sombras. Este espacio de color hace también imposible el fijar un valor de umbral apropiado.

2.2.2. Espacio de color HSV / HSL / HSI

Uno de los espacios de color más comunes dentro del área de visión por ordenador es HSI[2], también en el campo de reconocimiento e identificación de señales viales [22]. Este espacio de color comprende de tres componentes: Tono, Saturación e Intensidad. Este espacio es muy similar a los espacios HSV[3] y HSL[4], diferenciándose en que en lugar de la intensidad de color, se emplean el valor o la luminosidad respectivamente. Una de las diferencias mas importantes entre HSV y HSL están en que mientras en HSL la componente de saturación va desde el color plenamente saturado hasta el gris equivalente, en HSV se va desde el color saturado hasta el blanco. La otra diferencia radica en que en HSL la luminancia va desde el negro hasta el blanco, dejando en el medio la tonalidad deseada, en HSV el valor (V) va desde el negro hasta el tono. Es decir, realiza sólo la mitad del recorrido.

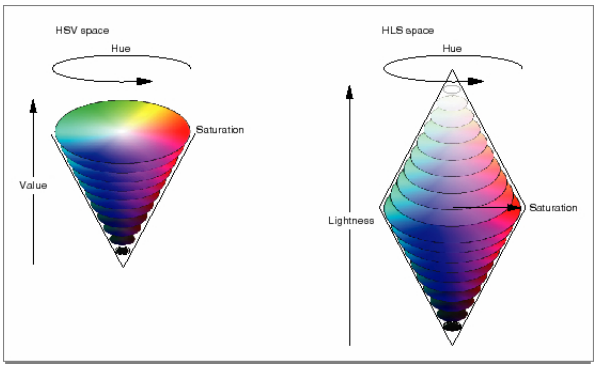


Figura 2.2.: Representación de HSV y HSL.

[2]Hue, Saturation, Intensity (Matiz, Saturación, Intensidad).
[3]Hue, Saturation, Value (Matiz, Saturación, Valor).
[4]Hue, Saturation, Lightness (Matiz, Saturación, Luminosidad).

En [11, 21, 23], primero se convierte la señal original del espacio RGB al HSI y entonces se aplica un valor de umbral fijo para extraer las regiones de color de interés. Los valores de umbral se eligen analizando la información de matiz y saturación de la imagen, normalmente de acuerdo a los histogramas (véase figura 2.3).

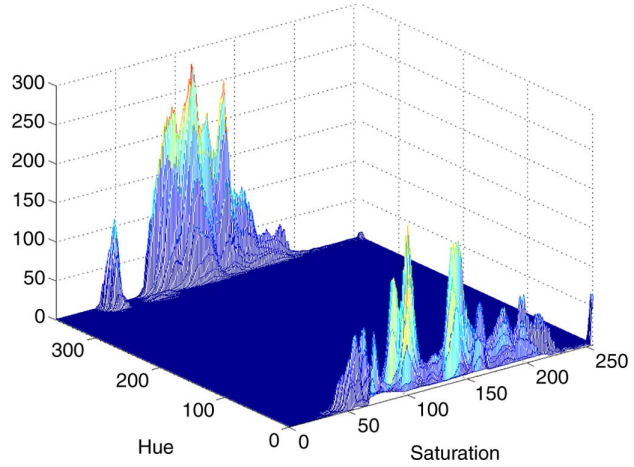


Figura 2.3.: Histograma de matiz y saturación del color rojo [18].

El espacio de color HSI tiene menos variación respecto a las condiciones de luz que RGB y por tanto, como el efecto de la luz puede ser reducido a un nivel aceptable, se puede aplicar un valor de umbral fijo para la segmentación. Un hecho destacable es que no existe una fórmula unificada para la transformación RGB - HSI (RGB - HSV). Cada autor utiliza diferentes fórmulas en sus estudios para convertir la imagen al espacio de color HSI.

2.2.3. Espacio de color YCbCr / YUV[5]

El espacio de color YCbCr se compone de luminancia (Y), crominancia del canal azul (Cb) y crominancia del canal rojo (Cr). La ventaja de este modelo de color está en que la luminancia se presenta separada de la información de crominancia [24]. En [15], se crea un filtro para extraer regiones de color similares al rojo del borde de las señales de limitación de velocidad. El filtro se entrena con imágenes de muestra de señales tomadas en distintas condiciones. En los documentos [13, 24] se convierte la imagen a YCbCr y se crean modelos del espacio de distribución de color (véase figura 2.4) para identificar los valores de umbral óptimos. En estos documentos se emplean dos valores de umbral optimizados para proporcionar un aislamiento robusto de los objetos rojos.

El modelo YCbCr es muy utilizado en el reconocimiento de señales de velocidad, no solo porque es resistente a las variaciones de luz sino que ademas, la fórmula de

[5]Luminancia (Y) y crominancia (UV).

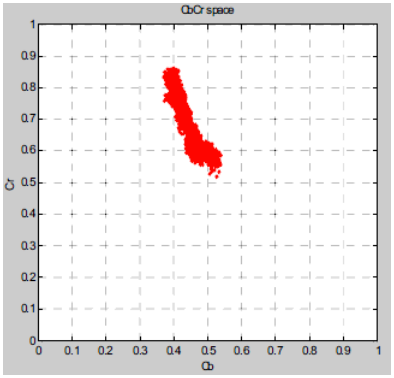


Figura 2.4.: Distribución espacial del color rojo en YCbCr [24].

conversión es directa y la crominancia del canal rojo es una característica a tener muy presente.

2.2.4. Componentes cromáticos / acromáticos

Puesto que el patrón legal establecido para las señales de limitación de velocidad consta principalmente de un círculo blanco rodeado de una circunferencia roja, en los documentos [14, 18, 25] van un paso más allá en la segmentación del panel blanco de la señal empleando la descomposición acromática de las imagen. Proponen tratar el panel blanco y el círculo rojo que lo rodea como un elemento acromático y un elemento cromático respectivamente. Una vez que se ha procesado la segmentación del color rojo basándose en la información de tono, se emplea la ecuación propuesta en [14] para determinar un color acromático / cromático:

$$f(R,G,B) = \frac{(\mid R-G \mid + \mid G-B \mid + \mid B-R \mid)}{3D} \tag{2.1}$$

donde D es el grado de extracción de un color acromático. El valor de D puede variar en diferentes condiciones pero, por norma general se emplea el valor 20, ver [22, 10, 18, 25]. Un color se considera acromático si $f(R,G,B) \leq 1$ y se considera acromático en cualquier otro caso. Atendiendo a este proceso, una región cromática (rojo) grande que contenga a una región acromática (blanco) más pequeña tiene una gran probabilidad de ser una señal de limitación de velocidad y será transferida a la fase de detección de detección de círculos como región de interés. Del mismo modo, en el documento [22], se emplea una variación de la ecuación (2.1) como una de las funciones de energía del sistema para generar la imagen cromática.

La segmentación de color cromática / acromática es una forma eficiente de eliminar la cantidad de regiones pasadas a la fase de detección de círculos y aumenta la precisión de la detección.

2.3. Detección de señales circulares

La detección de círculos se aplica a las RoI[6] generadas en la fase de segmentación anterior para comprobar cuando una región es un círculo. Si lo es, se puede estar seguro de que la RoI es un patrón de la señal de tráfico. Este proceso elimina aquellas regiones que poseen un tamaño similar y características de color similares a las de las señales de limitación de velocidad consiguiendo agilizar el proceso de detección. Un ejemplo de las regiones eliminadas puede ser la señal de "Ceda el Paso" (véase figura 2.5) en la imagen binarizada, que pasa el proceso de segmentación debido a la gran similitud en los colores.

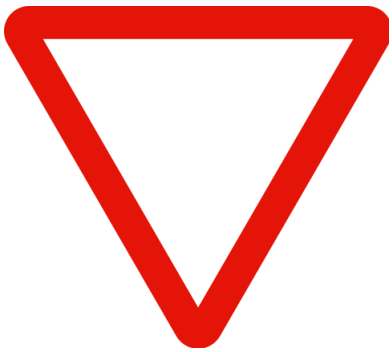


Figura 2.5.: Señal de "Ceda el Paso", muy similar en colores a las señales de limitación de velocidad.

2.3.1. RANSAC para círculos

El algoritmo RANSAC (*Random Sample Consensus*) fue propuesto por Fischler y Bolles en 1981 [26]. Se trata de una de las técnicas de modelo de ajuste más empleadas en el área industrial [27]. Básicamente, el algoritmo toma un conjunto de datos (p.e. el contorno), de forma aleatoria toma un subconjunto de características de muestra (p.e. puntos) e intenta encajarlos en un modelo de forma geométrica (p.e. un círculo). Entonces, compara el modelo con el conjunto de características total. El proceso RANSAC finaliza si hay un número suficiente de características del conjunto de datos que satisfacen el modelo con cierta tolerancia. En [13], se siguen los siguientes pasos para lograr la detección de círculos:

1. Se selecciona un conjunto de $p = 3$ puntos de contorno aleatoriamente (se necesitan tres puntos para determinar un círculo).

2. Se construye un círculo que pase por esos puntos.

[6]Region of Interest (Región de Interés)

3. Se cuenta el número de puntos que coinciden con el círculo dentro de cierta tolerancia ϵ_{max}.

4. Si el número de esos puntos es mayor que cierto valor límite n_{min}, se calcula en círculo de mínimos cuadrados que incluya a todos los puntos. Si no, se repite todo el proceso hasta alcanzar un número máximo de intentos m_{max}.

Como el algoritmo es bastante directo y solo se tienen unas pocas regiones que han pasado las etapa de segmentación de color, se puede considerar la velocidad de procesado suficiente para una detección en tiempo real. En [23] se afirma también que el sistema ha sido probado con éxito bajo varias condiciones climáticas y de luz con un buen rendimiento. Además, el algoritmo RANSAC es capaz de detectar un objeto circular con un contorno incompleto. Esta es una característica extremadamente importante para la detección de señales no solo por las sombras producidas según las condiciones de luz, sino también porque puede haber objetos bloqueando parte de la señal. Esto en muy común en las zonas urbanas, que es donde aparecen con más frecuencia las señales de limitación.

2.3.2. Detección de círculos basada en bordes / gradientes.

El algoritmo basado en borde/gradiente es una técnica de detección de círculos tradicional. Ante todo, en el campo de la detección de señales de velocidad, este algoritmo tiene una ventaja obvia debido a la naturaleza de las señales que constan de dos partes en distintos colores (blanco y rojo). Esto hace que la detección del borde entre las dos partes sea incluso más fácil de realizar. En general, el algoritmo calcula primero los bordes de la imagen en las regiones y entonces, se calcula el gradiente de cada píxel del borde. La base de este algoritmo consiste en comprobar si existe un punto central dentro de la región que sea apuntado por todos los gradientes y que tenga una distancia constante con todos los píxeles del borde [17]. En [22, 15], los autores aplican el algoritmo de detección basado en borde / gradiente original en sus sistemas. En el documento [15] se fija un área de búsqueda alrededor de cada región candidata. Para cada zona de búsqueda, aplican el detector de bordes Canny [28] para extraer la imagen de bordes. Los autores proponen que si un borde es parte de un círculo (p.e. un arco), el centro del círculo debe existir en la línea que atraviesa el borde y que tiene la dirección de su gradiente. Se calcula dicha línea para cada borde de una región y se vota por la línea en el área de búsqueda. Se considera que la región contiene un circulo si hay un pico prominente superior a cierto umbral en la zona. En el documento [22] se aplica una transformada de distancia a los bordes en lugar de considerar directamente la dirección de sus gradientes. Emplean la distancia de Hausdorff [29], que indica cómo

dos formas difieren para comprobar la imagen borde de la región. En [21] se introduce un novedoso principio de votación de bordes para la detección de círculos. En este algoritmo, se divide un círculo en cuatro fases (véase figura 2.6). Utilizan dos operadores

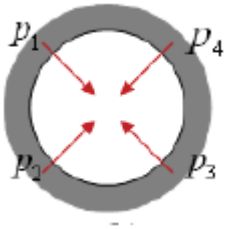


Figura 2.6.: Las cuatro fases de un círculo [21].

3×3 (direcciones horizontal y vertical) para extraer los bordes de la imagen primeramente y entonces, a cada punto en el borde le asignan un valor de fase de acuerdo a sus gradientes utilizando la ecuación (2.2).

$$P_m = \begin{cases} 1, & si\, G_x > 0\ y\ G_y > 0 \\ 2, & si\, G_x < 0\ y\ G_y > 0 \\ 3, & si\, G_x < 0\ y\ G_y < 0 \\ 4, & si\, G_x > 0\ y\ G_y < 0 \\ 0, & en\, otro\, caso. \end{cases} \tag{2.2}$$

Se termina con una matriz de fases de la región y una máscara se variación de bordes, cuyo tamaño es el mismo que el de la matriz de fases. Los valores de la máscara de variación de bordes son $v \mid v\, \epsilon \{1, 2, 3, 4, 0\}$. El patrón de la máscara es similar al de la figura 2.6 y se multiplica con la matriz de fases para realizar la matriz de comparación de bordes. Por último, se suman los valores de la matriz de comparación de bordes para compararlo con el valor umbral T_{mask}. Si la sumatoria es mayor a igual a T_{mask}, se puede confirmar que hay un círculo localizado en la región y que no hay ningún círculo si el valor de la suma es menor. Una vez detectado el círculo, los autores proponen aplicar un principio de voto por el centro para localizar el centro de ese círculo [21] y entonces, extraer dicho círculo. Dado que la detección se basa en el gradiente / borde de la imagen, se tendrá una buena resistencia a las variaciones de las condiciones de luz y una menor sensibilidad al ruido.

2.3.3. Comparación con plantillas circulares

En el campo de la detección y reconocimiento de objetos, la comparación con plantillas se ha considerado siempre como un algoritmo simple y directo. Esta técnica

se ha empleado también en muchos estudios referentes al campo de la detección de señales de tráfico [12, 24, 30, 19, 25]. Básicamente, la comparación con plantillas, es un proceso que compara muestras desconocidas con patrones específicos y calcula las similitudes entre ambos para clasificar la muestra desconocida. En [25], los autores emplean una colección de varías imágenes como plantillas para eliminar la ambigüedad causadas por varios esquemas de selección de plantilla y solamente una plantilla está implicada en la comparación en cada instante de tiempo. La similitud de coincidencia en cada momento se mide por la distancia mínima entre sus funciones tangente. Si la similitud supera el valor umbral, esa muestra será aceptada como una señal de limitación de velocidad. Si no es así, se empleará la siguiente imagen plantilla en la colección para comprobar las coincidencias. Este mismo algoritmo se emplea también en [12] y en [19]. Los autores crean una colección de plantillas con varios tamaños para cada señal de limitación de velocidad (p.e. 32×32, 38×38, 50×50, etc) (ver figura 2.7). Los sistemas rechazan las muestras que no alcanzan una similitud aceptable.

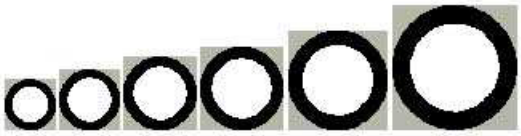


Figura 2.7.: Plantillas de diferentes tamaños [19].

En el documento [24] se proponen algunas mejoras respecto al algoritmo de comparación original. Los autores escalan el tamaño de la muestra para coincidir con el de la plantilla y aplican la comparación entre ambos para identificar la muestra con la máxima correlación cruzada. La correlación describe el teorema de convolución y la posibilidad concomitante de calcular de manera eficiente la correlación en el dominio de la frecuencia empleando la Transformada Rápida de Fourier (FFT[7]). Por ejemplo, se toma una muestra $f(i,j)$ y una plantilla $g(i,j)$ participa en la comparación. El sistema coloca la plantilla en la posición (i,j) de la imagen de muestra para detectar si está presente en ese punto comparando sus niveles de gris. Como es muy raro que dos escalas de grises coincidan exactamente, los autores miden la disparidad de los dos valores. Para obtener todas las localizaciones e instancias de la plantilla aplicadas a la muestra, la plantilla se desplaza sobre la muestra y se hace la medición en cada punto. Por lo tanto, para un patrón (m,n), se emplea la fórmula (2.3) para calcular la correlación cruzada:

$$N(m,n) = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} g(i,j)t(i-m,j-n)}{\left(\sqrt{\sum_{i=0}^{m}\sum_{j=0}^{n} g(i,j)^2} \times \sqrt{\sum_{i=0}^{m}\sum_{j=0}^{n} t(i-m,j-n)^2}\right)} \tag{2.3}$$

[7]Fast Fourier Transform

donde i y j son las coordenadas de un punto en la imagen de muestra. Una vez se han realizado los cálculos para cada posición posible a lo largo del total de la imagen de muestra, se puede determinar el valor local máximo, y compararse con el valor umbral T. El beneficio de este algoritmo mejorado está en que la diferencia de luminancia entre la plantilla y la muestra no afecta al resultado. Este algoritmo se utiliza también en el sistema propuesto por [30]. No obstante, el inconveniente del método de coincidencia con plantillas circulares también es obvio. Es muy sensible a las distorsiones, las cuales se dan frecuentemente según sea las condiciones o al usar distintos tipos de cámaras para monitorizar la escena.

2.3.4. Transformada circular de Hough y sus variaciones

La transformada circular de Hough es un enfoque típico en la detección de círculos [31]. Esta transforma un objeto desde el espacio de imagen al espacio acumulado de Hough para localizar el mayor máximo local. El espacio de Hough es un acumulador que almacena las evidencias de la probabilidad de que exista un círculo empleando la ecuación $(i-a)^2 + (j-b)^2 = r^2$ donde r es el radio dado del círculo y (i, j) y (a, b) son los centros del círculo en la imagen y en el espacio de Hough respectivamente. Sin embargo, la transformada de Hough tiene el inconveniente de la velocidad de procesado ya que siempre se ha considerado un algoritmo costoso en cálculo. A pesar de que todavía están en curso las investigaciones para reducir el coste computacional de este algoritmo, existen muchos sistemas en el campo de la detección de señales de velocidad que emplean este método para detectar la señal debido a su gran precisión y su capacidad de detectar objetos circulares parcialmente ocultos.

En algunos sistemas recientes como en [20, 32], se emplea la transformada de Hough junto con el algoritmo *Adaboost* [33] para detectar las señales de limitación de velocidad. *Adaboost* es un algoritmo de aprendizaje automático potente. Tiene una estructura en cascada que se compone de varias sub-etapas de procesamiento, y en cada etapa recibe un conjunto de características tipo *Haar* [34, 35]. Si la zona de la muestra no puede pasar cualquiera de las fases de sub-procesamiento, será rechazada como una región sin interés. El algoritmo *Adaboost* se entrena con una gran cantidad de patrones positivos y negativos captados en varios entornos y condiciones para crear el detector en cascada. Una muestra es detectada como región candidata para el proceso de transformación circular de Hough si supera la cascada entrenada. La cantidad de regiones candidatas es idealmente pequeña y por lo tanto se puede llevar a cabo la transformada de Hough en un corto tiempo de proceso. El rendimiento de la detección, gracias a este sistema *Adaboost* - Hough, se mejora tanto en aspecto de eficiencia como

de precisión. Además, se ha demostrado que el sistema es poco sensible a condiciones adversas [32].

En [11], los autores proponen una novedosa variante de la transformación de Hough denominada Transformación de Simetría Radial Rápida[8], que también es empleada para detectar señales de limitación de velocidad en [10]. El algoritmo propuesto se ejecuta para kp, donde k es el número de radio discreto buscado y p es la cantidad de píxeles. A diferencia, la transformación de Hough original se ejecuta para kbp, donde p vota en todos los círculos sobre un conjunto discreto de radios k que pueden pasar a través de p, y b viene de la discretización de los contenedores en el gradiente de las tangentes circulares que pueden pasar por este punto. La variante elimina b obteniendo el gradiente del punto del borde directamente de la salida del detector de bordes Sobel. Por lo tanto, se reduce el cálculo de la simetría radial y también se simplifica en una dimensión. Esto resulta en que el sistema se aprovecha de la transformación de Hough con una velocidad de procesado mucho mayor [10]. El algoritmo es apto para procesos en tiempo real ya que según se demuestra en [11], tarda 13.2 ms en detectar los círculos en una imagen de 320×240.

2.3.5. Máquina de Vectores Soporte

La Máquina de Vectores Soporte (SVM[9]) fue presentada por Vapnik [36]. Consta de un conjunto relacionado de mecanismos de aprendizaje supervisados que se emplean para reconocer y analizar patrones. En el campo de reconocimiento de señales verticales, los autores del documento[18] introducen un nuevo algoritmo de detección de formas basados en SVM lineal. Este sistema se ha demostrado también efectivo a la hora de detectar señales de tráfico por [37].

En el sistema, las zonas que pasaron por la segmentación de color son clasificadas por la SVM lineal. La SVM toma las DtBs (Distancias al Margen[10]) como vectores característicos para las entradas. Se toman cuatro vectores DtB para cada zona de muestra. Los DtBs son la distancia desde el borde externo de la zona hasta el margen de su cuadro delimitador [18]. Para obtener un DtB, primero se normaliza cada zona en un recuadro delimitador de tamaño fijo. Un conjunto de líneas de escaneado (20 líneas en este sistema) buscan el borde exterior de la zona partiendo del margen del recuadro envolvente, desde la izquierda hacia la derecha. La línea para cuando alcanza el borde y la siguiente línea comienza el escaneo de la misma manera. Una vez todas las líneas han

[8]Fast Radial Symetry Transform
[9]Support Vector Machine
[10]Distance to Bountary

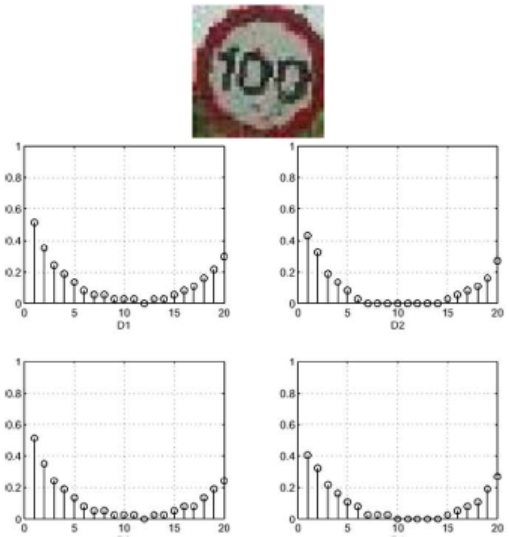


Figura 2.8.: Los cuatro DtBs de una región que contiene una señal de limitación de velocidad (arriba) [18].

realizado el escaneo, se crea el vector DtB basándose en las localizaciones del primer cruce de cada línea. La figura (2.8) muestra un ejemplo de los DtBs de una región de muestra donde $D1$, $D2$, $D3$ y $D4$ representan los DtB superior, derecho, inferior e izquierdo respectivamente. Entonces, estos cuatro vectores DtBs se introducen en las SVM para clasificar si la forma de la zona de la muestra es un círculo o no, mediante la consecución de cuatro votos favorables. La región se rechazara como "forma ruidosa" si el resultado de la votación es menor que un valor de umbral especificado. En su conclusión, los autores de los documentos [37, 18] afirman que el proceso de detección empleando SVM es también invariante respecto a la rotación, la traslación y el escalado, además de su alta eficiencia y precisión.

2.4. Reconocimiento de las señales de limitación de velocidad

Este paso suele ser la última etapa en los sistemas reconocimiento de señales de limitación de velocidad. Las zonas de interés se enviarán al módulo de reconocimiento que es donde se llevará a cabo el Reconocimiento Óptico de Caracteres (OCR[11]). En el ámbito del reconocimiento de señales de limitación de velocidad, sólo existen 10 dígitos que hay que tener en cuenta (0 - 9). Los enfoques comunes utilizan Redes Neuronales Artificiales para clasificar las muestras [21, 13, 24, 19] o por medio de modelos aproximados y emparejamiento de características locales [10, 32]. Asimismo, el método de reconocimiento de caracteres basado en escaneo de líneas se emplea ampliamente en

[11]Optical Character Recognition

la adquisición de documentos y en la lectura de contadores automatizada [38, 39, 40]. Esta técnica se puede emplear también para reconocer los dígitos en las señales de limitación de velocidad. El resultado de este proceso se trata como la lectura de una señal. El sistema avisará al usuario mediante alguna indicación visual y/o acústica. Una vez realizado este paso, se considera finalizado el ciclo de reconocimiento de una señal.

2.4.1. Redes Neuronales Artificiales

Una Red Neuronal Artificial (ANN[12]) es un modelo matemático o computacional que consiste en un grupo interconectado de neuronas artificiales. Procesa los datos mediante un enfoque conexionista, de manera semejante a la extensa red de neuronas del cerebro humano. Actualmente, las ANN son herramientas estadísticas no lineales de modelado de datos y se emplean ampliamente en la detección de patrones en los datos o el modelado de relaciones complejas entre las entradas y salidas.

En los documentos [13, 19, 24] se implementa la etapa de reconocimiento mediante una red neuronal multicapa retroalimentada, también comocidad como *back propagation*. La red retroalimentada básica realiza una transformación no lineal de los datos de entrada con el fin de aproximar los datos de salida. Esta red neuronal contiene tres capas diferentes: La capa de entrada, la capa oculta y la capa de salida [41]. Cada capa consta de un número determinado de nodos. La capa de entrada es pasiva, lo que implica que no modifica los datos, mientras que las otras dos capas son activas. Por ejemplo, cada nodo en la capa de entrada es el valor del color (normalmente 0 para el negro y 1 para el blanco) de cada píxel en una imagen binaria. Cada valor de la capa de entrada se duplica y se envía a todos los nodos de la capa oculta. A esto se le denomina estructura completamente interconectada. La capa oculta normalmente tiene un tamaño del 10 % respecto a la capa de entrada [41]. Los valores que entran en un nodo oculto se multiplican por unos pesos, que son un conjunto de valores predefinidos para los nodos ocultos. Una vez se han multiplicado todos los valores de entrada por el peso, se suman para producir un solo total. Antes de salir del nodo, este total pasa a través de una función matemática no lineal denominada sigmoide que es una curva con forma de 'S' que limita la salida de los nodos. Es decir, la entrada de la sigmoide es un valor entre $-\infty$ y $+\infty$, mientras que la salida del nodo debe ser un valor entre 0 y 1. Cada una de las salidas de la capa oculta se duplica y se aplica a la capa de salida. Los nodos activos de esta capa combinan los valores entrantes para generar las salidas, que normalmente son las posibilidades de las categorías de salida. La categoría de salida con mayor probabilidad es la salida final de esta ANN retroalimentada. Antes de

[12] Artificial Neural Network

poder utilizar una ANN para procesar datos, esta debe ser entrenada mediante el uso de varios algoritmos de entrenamiento [41]. En el documento [24], los autores entrenan la ANN empleando imágenes binarias de los dígitos 0-9 con tamaño 32×32 píxeles, lo que implica que existen 1024 nodos en su capa de entrada. El modelo posee 15 y 10 neuronas en sus capas oculta y de salida respectivamente. Las 10 neuronas de salida representan las 9 señales de limitación de velocidad (15, 30, 40, 50, 60, 70, 80, 90 y 100 Km/h) mas comunes en el país de procedencia de los autores (Malasia), y una neurona de salida no deseada que indica que la entrada no era un dígito (véase figura 2.9). Una

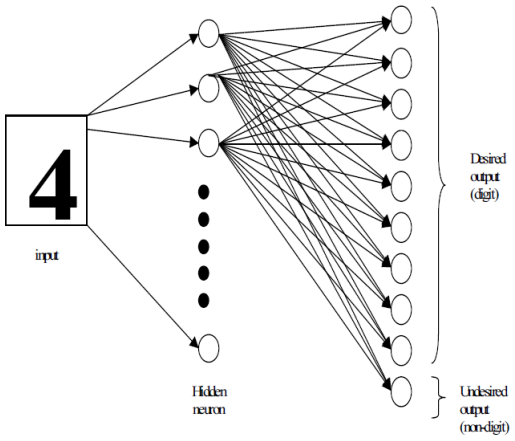


Figura 2.9.: Estructura de ejemplo de una ANN para el reconocimiento de señales de limitación de velocidad [24].

vez todas las neuronas de la capa de salida tienen su resultado, el resultado final de salida puede ser tanto una lectura de señal de limitación (p.e 40 Km/h) o la lectura de un elemento que no es un dígito (p.e. una señal que no es de limitación de velocidad) dependiendo de la mayor probabilidad de parecido de salida. En los documentos [13, 19] se emplea también el mismo modelo de ANN para reconocer las lecturas de las distintas señales de limitación de velocidad. En [13], la capa de entrada a la ANN contiene 400 neuronas (muestras de 20×20 píxeles). El tamaño de la capa oculta se ha fijado por experimentación empírica, que ha demostrado que usando 30 neuronas en la capa oculta se obtienen los mejores resultados. Hay 12 neuronas en la capa de salida que se corresponden con los siguientes tipos de señales: 10, 20, 30, 40, 50, 60, 70, 80, 90 y 100 mph, límites de velocidad nacional (en el Reino Unido) y falso positivo (p. e. no es una señal). Cada valor de salida es la clasificación de probabilidad entre 0 y 1 para la clase de señal correspondiente y el valor mayor indica la salida de la ANN.

En la conclusión de [13, 24, 19], los investigadores afirman que el sistema ha conseguido un rendimiento muy preciso con una capacidad de reconocimiento del 90 al 92 %. La velocidad de reconocimiento podría también ser aceptable para procesado en tiempo real. De todas formas, el coste de un reconocimiento preciso mediante ANN

está en la gran cantidad de muestras de entrenamiento necesarias y en la creación de un modelo de estructura multicapa complejo.

2.4.2. Modelos difusos y emparejamiento de características locales

Al contrario que los enfoques basados en ANN, el emparejamiento con patrones no requiere una toma de muestras y el entrenamiento del modelo. Sin embargo, los algoritmos clásicos de comparación no son aptos para reconocer las leyendas de las señales de limitación de de velocidad debido a las imágenes borrosas o las condiciones climáticas o de iluminación [10, 32]. Para superar estos inconvenientes, los autores de [10] proponen un algoritmo que combina el emparejamiento de patrones con el emparejamiento local de características. Primero, el sistema aplica el método de comparación con patrones difusos para reconocer a grosso modo el carácter numérico de la señal. Entonces, para evitar una mala clasificación y conseguir un funcionamiento robusto, se aplica un proceso de reconocimiento fino que se basa en vectores de características locales. El abordaje de la primera parte del sistema consta de un emparejamiento con patrones empleando la Correlación Cruzada Normalizada como criterio de comparación, la cual ya fue descrita en la sección "2.3.3 Comparación con plantillas circulares". Después de esto, se extraerán las características de la imagen de muestra para crear los vectores de características locales.

La razón para aplicar los vectores de características locales está en poder identificar caracteres similares con desenfoques y diferentes luminancias. Las características de un carácter pueden ser número/posición de huecos [32] y distribución de los píxeles de un carácter (no los píxeles de fondo) en lo alto/bajo de la imagen. En [10] se muestra un ejemplo de como distinguir los caracteres '4' y '6' en una señal de velocidad. Puesto que el número de huecos en '4' y '6' es uno en ambos, la característica que diferencia ambos dígitos es la diferencia entre la distribución de los píxeles blancos (el fondo de una señal de velocidad) en la parte baja de la muestra. Para crear el vector de características local, el sistema busca los píxeles de fondo en la parte baja de la imagen. La búsqueda en una columna termina cuando se encuentra un píxel de carácter (píxel no-fondo). Entonces el sistema cuenta el número de píxeles de fondo encontrados y pasa a la siguiente columna. Una vez se ha realizado la búsqueda en todas las columnas, los recuentos de píxeles de fondo se normalizan al tamaño 20 de acuerdo al orden de búsqueda (p.e. de izquierda a derecha) para obtener el vector de características locales (véase la figura 2.10). El vector de características local del modelo difuso también se puede obtener de la misma manera. Entonces, se podrá obtener la Distancia Euclidea

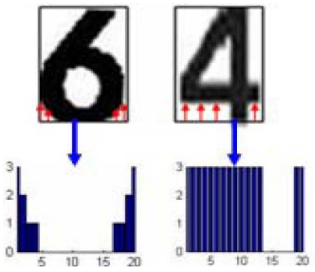


Figura 2.10.: Dos vectores de características para los dígitos '4' (izquierda) y '6' (derecha). Las bandas azules representan en número de píxeles de fondo en cada columna [10].

de los vectores de características locales y el modelo difuso empleando la siguiente ecuación:

$$Distancia(i) = \sqrt{\sum_{j=1}^{20} \left(z_j - z_j^i\right)^2}, \ (i = 4, 6) \tag{2.4}$$

Donde $Z^i = \{z_1^i, ..., z_{20}^i\}$ es el vector de características de modelo difuso i y $Z = \{z_1, ..., z_{20}\}$es el vector local del carácter a ser reconocido. Por lo tanto, la leyenda de la señal de velocidad es finalmente reconocida como el mejor resultado de emparejamiento que tiene la menor distancia Euclidea. De forma similar, el sistema propuesto en el documento [32] utiliza este algoritmo en la etapa de reconocimiento con otras características locales (p.e. la posición de la fila superior / inferior de los píxeles de carácter, el número de columnas a izquierda / derecha a través de los píxeles de carácter).

Este algoritmo mantiene la simplicidad y flexibilidad del emparejamiento con patrones. Se mejora la fiabilidad y se aumenta el rendimiento del reconocimiento de dígitos.

2.4.3. Reconocimiento de dígitos basado en escaneo de líneas

La técnica de OCR basada en escaneo de líneas se emplea ampliamente en la lectura de contadores automatizada (p.e. lecturas automatizadas de registro de un amperímetro [38, 39]) y en la adquisición de documentos debido a su simplicidad y a su velocidad de reconocimiento. De hecho, este algoritmo es adecuado para varios propósitos, incluyendo el reconocimiento de señales de limitación de velocidad. La idea del algoritmo es usar múltiples líneas de escaneo verticales y/o horizontales que pasen por la imagen de entrada y marquen los píxeles encontrados que no pertenecen al fondo. Estos píxeles y ciertos patrones relevantes son las características críticas que identificarán al dígito comparándolos con unas matrices de dígitos (0 - 9) predefinidas. El carácter con la mayor similitud con la características críticas será la salida del sistema.

Si la similitud está por debajo de un cierto valor umbral (p.e. 95 %), se considera que el carácter no es un dígito.

En el documento [40] se da un ejemplo detallado de como trabaja el algoritmo. Para empezar, el sistema detecta los margenes izquierdo (X_{min}), derecho (X_{max}), superior (Y_{min}) e inferior (Y_{max}) de la imagen del carácter. Entonces, se dibuja una línea de escaneo vertical a través del punto medio del ancho de la imagen ($X = X_{min} + [X_{max} - X_{min}]/2$). Todos los píxeles no de fondo que encuentre la línea de escaneo serán marcados como $1, 2, ..., n$, desde arriba hacia abajo (véase figura 2.11). Por ejemplo, hay 6 puntos críticos para las imágenes que contienen los dígitos '2' y '8'.

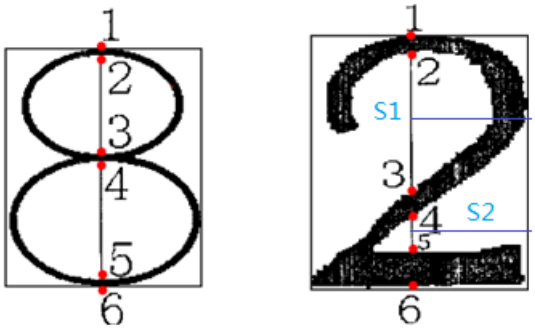


Figura 2.11.: Distribución de los puntos críticos encontrados por la línea de escaneo vertical y dos líneas de escaneo horizontal para los dígitos '8' y '2' [40].

Una vez se marcan todos los puntos críticos, el algoritmo determinará el carácter de la siguiente manera:

1. Si sólo hay dos puntos críticos (número mínimo de puntos posible), el patrón en la imagen de entrada es el dígito **'1'**.

2. Si el número de puntos críticos es mayor que seis, el patrón no se corresponde con ningún dígito (p.e. se retorna "nulo").

3. En otro caso, si el número de puntos críticos es mayor que 3, para todos los puntos pares, excepto el último, se realiza un rastreo del borde interior. Se marca la *característica 1* = "verdadero" si el primer rastreo es exitoso (se trata de un borde cerrado) y se marca la *característica 2* = "verdadero" si ocurre lo mismo para el otro punto. (p.e. para el patrón '8', el primer rastreo empiece en el punto nº 2 y el segundo en el punto nº 4 y tanto *característica 1* como *característica 2* se marcaran como 'verdadero'. De igual forma, para el patrón '2', ambas se marcarán como 'falso').

4. Si el número de puntos críticos es seis, *característica 1* = "verdadero" y *característica 2* = "verdadero", el patrón es el dígito **'8'**.

5. Si el número de puntos críticos es seis, *característica 1* = "verdadero" y *característica 2* = "falso", el patrón es el dígito **'9'**.

6. Si el número de puntos críticos es seis, *característica 1* = "falso" y *característica 2* = "verdadero", el patrón es el dígito **'6'**.

7. Si el número de puntos críticos es cuatro, el punto crítico nº 4 no coincide con Y_{min} *y característica 1* = "verdadero", el patrón es el dígito **'4'**.

8. Si el número de puntos críticos es cuatro, el punto crítico nº 4 coincide con Y_{min} *y característica 1* = "verdadero", el patrón es el dígito **'0'**.

9. Si el número de puntos críticos es cuatro, *característica 1* = "falso" y *característica 2* = "falso", el patrón es el dígito **'7'**.

10. En otro caso, si el número de puntos críticos es seis, *característica 1* = "falso" y *característica 2* = "falso", se crean dos líneas de escaneo horizontales (hacia la derecha) que pasen por el punto medio entre los puntos nº 2 y nº3 y por el punto medio entre los puntos nº 4 y nº5 respectivamente (véase figura 2.11). Se marca *característica 3* = "verdadero" si $S1$ encuentra algún píxel que no pertenezca al fondo y *característica 4* = "verdadero" si no lo encuentra $S2$. Si no, se marcan como "falso".

11. Si el número de puntos críticos es seis, *característica 1* = "falso", *característica 2* = "falso", *característica 3* = "falso" y *característica 4* = "verdadero", el patrón es el dígito **'5'**.

12. Si el número de puntos críticos es seis, *característica 1* = "falso", *característica 2* = "falso", *característica 3* = "verdadero" y *característica 4* = "verdadero", el patrón es el dígito **'3'**.

13. Si el número de puntos críticos es seis, *característica 1* = "falso", *característica 2* = "falso", *característica 3* = "verdadero" y *característica 4* = "falso", el patrón es el dígito **'2'**.

14. Si no se cumple nada de lo anterior, el patrón en la imagen de entrada no pertenece a un dígito. Se devuelve "nulo".

Con las 14 condiciones anteriores, se puede identificar el patrón de una imagen de entrada tanto como un carácter de dígito o un carácter no de dígito. Puesto que la leyenda de las señales de limitación pueden variar en número de dígitos (p.e dos dígitos para 50Km/h y tres para 120Km/h) pero el último es siempre 0 y el primero en el caso

de tres dígitos siempre es un 1, se puede simplificar el sistema identificando solamente el dígito que es más variante (el primero si hay 2 dígitos y el segundo si hay 3).

Algunas de las características obvias de este algoritmos son su rápida velocidad de proceso y la fuerte resistencia a las imágenes borrosas y el cambio de tipo de fuente. El concepto de este algoritmo es directo y fácil de implementar. Se emplea una técnica similar en los sistemas propuestos por [38, 39]. La diferencia está en que en [39], los autores aplican un proceso de adelgazamiento de la imagen y eliminación de rebabas a la imagen de entrada antes de dibujar las líneas de escaneo con el fin de reducir el efecto del ruido. Este paso puede considerarse innecesario ya que el ancho de las tres líneas de escaneo es siempre de un píxel.

Capítulo 3

Estudio de las librerías OpenCV

3.1. Introducción

OpenCV (*Open Source Computer Vision*) es una biblioteca libre de visión artificial desarrollada originalmente por Intel. Su publicación se da bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas. Está desarrollada en código C/C++ optimizado y puede hacer uso de varios procesadores a la vez. Posee interfaces en C++, C, Python y Java y admite su uso en Windows, Linux, Android y Mac OS.

OpenCV se ha diseñado para ser computacionalmente eficiente y con un fuerte enfoque en la creación de aplicaciones en tiempo real. Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos, calibración de cámaras, visión estéreo y visión robótica (véase figura 3.1).

OpenCV posee estructura modular, lo que implica que incluye distintas librerías estáticas o compartidas. Están disponibles los siguientes módulos:

- ***core*** - un módulo compacto que define las estructuras básicas de datos, incluyendo la matriz multidimensional `Mat` y las funciones básicas que son utilizadas por los demás módulos.

- ***imgproc*** - un módulo de procesado de imagen que contiene tanto filtrado lineal como no linear, transformaciones geométricas de la imagen (cambio de tamaño, deformación afín y de perspectiva, remapeado genérico basado en tablas), conversión del espacio de color, histogramas, etcétera.

- ***video*** - módulo de análisis de vídeo que incluye estimadores de movimiento, sustracción de fondo, y algoritmos de seguimiento de objetos.

- ***calib3d*** - algoritmos básicos de geometría multivisión, calibración de cámaras estéreo o simples, estimación de la posición de un objeto, algoritmos de correspondencia estéreo, y elementos para la reconstrucción 3D.

- ***features2d*** - detectores de características destacadas, descriptores, y emparejadores de descriptores.

- ***objdetect*** - detección de objetos e instancias de las clases predefinidas (por ejemplo, caras, ojos, jarras, personas, coches y demás).

- ***highgui*** - una interfaz de fácil uso para la captura de vídeo, codecs de imagen y vídeo, así como capacidad para la creación de interfaces de usuario simples.

- ***android*** - conversión de bitmaps Android a la matriz `Mat` propia de OpenCV así como la carga de ficheros de datos propios de OpenCV.

- ***utils*** **-** conversión de tipos de datos desde `Mat` a los distintos tipos de datos del sistema.

- ***ml*** **-** métodos de aprendizaje automático.

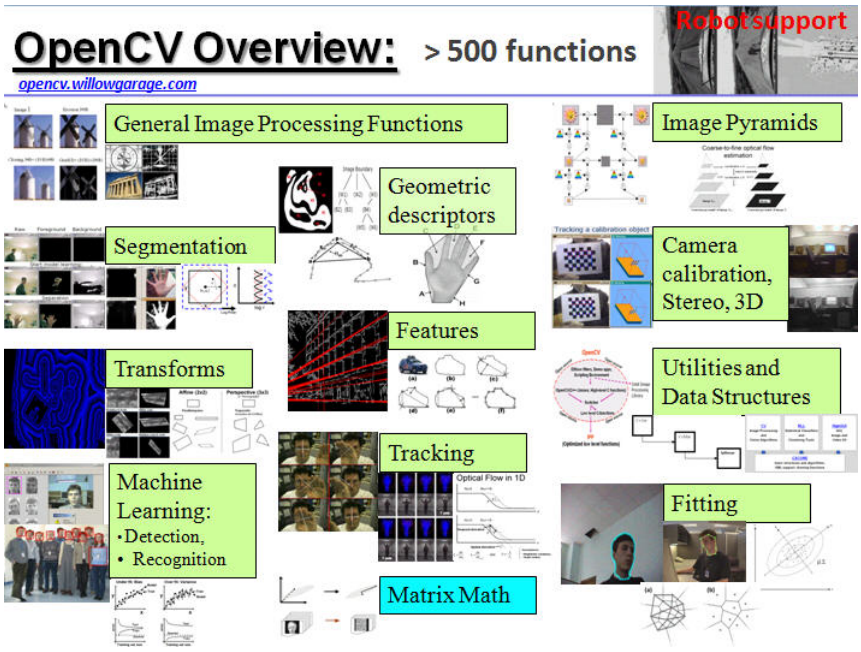


Figura 3.1.: Rango de aplicaciones de OpenCV [42].

3.2. Conceptos fundamentales de la API

3.2.1. Espacio de nombres `cv`

Todas las clases y funciones de OpenCV están emplazadas dentro del espacio de nombres `cv`. Por lo tanto, para acceder a esta funcionalidad se usa el especificador `cv::` o la directiva `using namespace cv;` (ver código 3.1).

```
#include "opencv2/core/core.hpp"
...
cv::Mat H = cv::findHomography(points1, points2, CV_RANSAC, 5);
...
```

```
#include "opencv2/core/core.hpp"
using namespace cv;
...
Mat H = findHomography(points1, points2, CV_RANSAC, 5 );
...
```

Código 3.1: Usos del espacio de nombres `cv`.

3.2.2. Gestión automática de memoria

OpenCV maneja toda la memoria de forma automática. En primer lugar, `std::vector`, `Mat`, y otras estructuras de datos usadas por las funciones y métodos tienen destructores que cancelan la asignación de los buffers de memoria subyacente cuando es necesario. Esto significa que los destructores no siempre liberan los buffers como en el caso de `Mat`. Se tiene en cuenta el posible reparto de datos. Un destructor disminuye el contador de referencia asociado al buffer de la matriz de datos. El buffer es liberado si y solo si el contador de referencia llega a cero, es decir, cuando no queden estructuras apuntando al buffer. De la misma forma, cuando se copia una instancia de `Mat`, realmente no se hace la copia, sino que se incrementa el contador de referencia para memorizar que hay otro propietario de los mismos datos. En su lugar, existe el método `Mat::clone` que si realiza una copia completa de la matriz. Véase un ejemplo en el código 3.2.

```
// se crea una gran matriz de 8Mb
Mat A(1000, 1000, CV_64F);
// se crea un nuevo cabezero para la misma matriz;
// esta es una operación instantánea, independientemente del tamaño de
```

```
// la matriz.
Mat B = A;
// se crea un nuevo cabezero para la 3ra fila de A; no se copian los
// datos
Mat C = B.row(3);
// ahora se crea una copia separada de la matriz
Mat D = B.clone();
// se copia la 5a fila de B en C, es decir, se copia la 5a fila de A
// en la 3a fila A.
B.row(5).copyTo(C);
// ahora hagamos que A y D compartan los datos; despues de esto la
// versión modificada de A todavía está referenciada por B y C.
A = D;
// ahora se hace de B una matriz vacía (que no hace referencia a ningún
// búfer de memoria), pero la versión modificada de  A seguirá estando
// referenciada por C, a pesar que C es sólo una fila de la matriz A
// original
B.release();
// por último, se hace una copia completa de  C. como resultado, la
// gran matriz modificada será liberada, puesto que ya no está
// referenciada por nadie.
C = C.clone();
```

Código 3.2: Ejemplo de la gestión de memoria en OpenCV.

Por lo tanto, el uso de `Mat` y otras estructuras básicas es simple. Pero, ¿qué sucede con las clases de alto nivel o incluso los tipos de datos de usuario creados sin tener en cuenta la gestión automática de memoria? Para eso, OpenCV ofrece la plantilla de clase `Ptr<>`, que es similar a `std::shared_ptr` de C++. Por tanto, en lugar de usar punteros planos:

```
T* ptr = new T(...);
```

Se puede utilizar:

```
Ptr<T> ptr = new T(...);
```

Eso es, `Ptr<T> ptr` encapsula un puntero en una instancia `T` y un contador de referencia asociado con el puntero.

3.2.3. Asignación automática de los datos de salida

OpenCV libera la memoria de forma automática, así como también asigna automáticamente memoria para los parámetros de salida de las funciones la mayor parte

del tiempo. Por tanto, si una función tiene uno o más vectores de entrada (instancias `cv::Mat`) y varios vectores de salida, los vectores de salida se asignan y liberan de forma automática. El tamaño y tipo de los vectores de salida se determinan a partir del tamaño y tipo de los vectores de entrada. Si es necesario, las funciones usan parámetros adicionales que ayudan a descubrir las propiedades de la matriz de salida. Por ejemplo, en el código 3.3, el vector `frame` es asignado directamente por el operador `>>` desde que se conocen la resolución del *frame* de vídeo y la profundidad de bit al modulo de captura de vídeo. El vector `edges` es asignado automáticamente por la función `cvtColor`. Tiene el mismo tamaño y la misma profundidad de bit que el vector de entrada. El número de canales es debido a que se pasa el código de conversión de color `CV_BGR2GRAY`, lo que se traduce en una conversión de color a escala de grises. Notese que `frame` y `edges` se asignan solamente durante la primera iteración del bucle ya que los sucesivos frames de vídeo tendrán la misma resolución. Si por algún motivo se cambia la resolución del vídeo, los vectores serán reasignados automáticamente.El elemento clave de esta tecnología es el método `Mat::create`, que tiene como entrada el tamaño y tipo deseados del vector. Si el vector ya tiene el tamaño y tipo especificado, el método no hace nada. Si no, libera los datos asignados previamente, si existen (esto implica decrementar el contador de referencia y compararlo con cero), y entonces asignar un nuevo buffer con el tamaño requerido. La mayor parte de las funciones llama al método `Mat::create` para cada vector de salida, y entonces se implementa la asignación automática de los datos de salida.

Algunas excepciones notables a este esquema son `cv::mixChannels`, `cv::RNG::fill`, y algunos otros métodos y funciones. No son capaces de asignar la matriz de salida, por lo que habrá que hacerlo de antemano.

```
#include "cv.h"
#include "highgui.h"

using namespace cv;

int main(int, char**)
{
    VideoCapture cap(0);
    if(!cap.isOpened()) return -1;

    Mat frame, edges;
    namedWindow("edges",1);
    for(;;)
    {
        cap >> frame;
        cvtColor(frame, edges, CV_BGR2GRAY);
        GaussianBlur(edges, edges, Size(7,7), 1.5, 1.5);
```

```
        Canny(edges, edges, 0, 30, 3);
19      imshow("edges", edges);
        if(waitKey(30) >= 0) break;
    }
    return 0;
}
```

Código 3.3: Ejemplo de asignación automática de los datos de salida.

3.3. Aritmética de saturación

Como librería de visión artificial, OpenCV se encuentra con píxeles de imágenes codificados de forma compacta, 8 ó 16 bit por canal, y que por tanto tienen un rango de valores limitados. Además, ciertas operaciones con imágenes, como conversiones del espacio de color, ajustes de brillo/contraste, cambios de intensidad, interpolaciones complejas (bi-cúbica, Lanczos [43]) pueden producir valores que estén fuera del rango disponible. Si se almacenan solo los 8 (16) bits menos significativos del resultado, pueden aparecer artefactos visuales (*banding*, posterización, emborronamiento o ruidos) que afectaran al posterior análisis de la imagen. Para solucionar este problema, se utiliza la llamada *aritmética de saturación*. Por ejemplo, para guardar `r`, resultado de una operación, en una imagen de 8 bits, se busca el valor más cercano dentro del rango `0..255`:

$$I(x,y) = min(max(round(r), 0), 255) \tag{3.1}$$

Se aplican reglas similares a los tipos 8 bits con signo y 16 bits con o sin signo. Esta semántica se utiliza en toda la librería. En C++ se hace usando las funciones `saturare_cast<>` que se parecen a las operaciones *cast* estándar de C++. A continuación se muestra una implementación de la fórmula (3.1):

```
I.at<uchar>(y,x) = saturate_cast<uchar>(r);
```

donde `cv::uchar` es un tipo entero sin signo de 8 bits de OpenCV.

3.3.1. Tipos de píxel fijo. Uso limitado de plantillas

Las plantillas son una gran característica de C++ que permiten la implementación de estructuras de datos y algoritmos muy potentes, eficientes y a la vez seguros. Sin

embargo, el uso extensivo de las plantillas acarrea un incremento en el tiempo de compilación y en el tamaño del código. Además, es difícil separar una interfaz y su implementación cuando se utilizan plantillas de forma exclusiva. Eso puede estar bien para algoritmos básicos pero no es bueno para librerías de visión artificial donde un solo algoritmo puede alcanzar miles de líneas de código. Por esto y también para simplificar el desarrollo de enlaces a otros lenguajes, como Python, Java, Matlab que no tienen plantillas en absoluto o tienen capacidades limitadas para plantillas, la actual implementación de OpenCV se basa en polimorfismo y en envío en tiempo de ejecución sobre plantillas. En aquellos casos en los que el envío en tiempo de ejecución puede resultar muy lento (como en el caso de los operadores de acceso a píxeles), imposible (como en la implementación genérica de punteros), o simplemente no es conveniente (p.e. en la conversión exacta de un tipo primitivo a otro mediante `saturate_cast<>()`), la actual implementación introduce plantillas reducidas de clases, métodos y funciones. En cualquier caso el uso de plantillas en la actual versión de OpenCV está limitado.

En consecuencia, hay conjunto limitado y fijo de datos primitivos sobre los que la librería puede operar. Es decir, los elementos de los vectores deben tener uno de los siguientes tipos:

- 8 bits sin signo (`uchar`)
- 8 bits con signo (`schar`)
- entero de 16 bits sin signo (`ushort`)
- entero de 16 bits con signo (`short`)
- entero de 32 bits con signo (`int`)
- punto flotante de 32 bits (`float`)
- punto flotante de 64 bits (`double`)
- una tupla[1] de varios elementos donde todos los componentes tienen en mismo tipo (uno de los anteriores). Los vectores cuyos elementos son tuplas, son conocidos comos vectores multicanal, en oposición a los vectores monocanal, cuyos elementos son valores escalares. El numero posible de canales está definido por la constante `CV_CN_MAX`, que actualmente esta definida a 512.

Para estos tipos básicos, se aplica la siguiente enumeración:

```
enum { CV_8U=0, CV_8S=1, CV_16U=2, CV_16S=3, CV_32S=4, CV_32F=5, CV_64F=6 };
```

[1] secuencia ordenada de objetos

Los tipos de datos multicanal (`n-channel`) se pueden especificar usando las siguientes opciones:

- constantes `CV_8UC1 ... CV_64FC4` (para un número de canales de 1 a 4).
- macros `CV_8UC(n) ... CV_64FC(n)` o `CV_MAKETYPE(CV_8U, n) ... CV_MAKETYPE (CV_64F, n)` cuando el número de canales es mayor que 4 o es desconocido en tiempo de compilación.

En el código (3.4), se muestran algunos ejemplos del uso algunos datos primitivos.

```
Mat mtx(3, 3, CV_32F); // crea una matriz 3x3 de punto flotante
Mat cmtx(10, 1, CV_64FC2); // crea una matriz 10x1 de 2-canales en punto
                           // flotante(vector complejo de 10-elementos)
Mat img(Size(1920, 1080), CV_8UC3); // crea una imagen de 3-canales (color)
                                    // de 1920 columnas y 1080 filas.
Mat grayscale(image.size(), CV_MAKETYPE(image.depth(), 1)); // crea una imagen de
                                                            // 1-canal del mismo
                                                            // tamaño y el mismo
                                                            // tipo de canal que img
```

Código 3.4: Ejemplo de uso de los tipos de datos de OpenCV.

No se pueden construir o procesar vectores con elementos más complejos utilizando OpenCV. Además, cada función o método solo puede manejar un subgrupo de todos los tipos de vectores posibles. Por lo general, cuanto más complejo sea el algoritmo, más pequeño será el subgrupo de formatos soportado. A continuación se ven algunos ejemplos típicos de dichas limitaciones:

- El algoritmo de detección de caras trabaja solo con imágenes de 8 bits a color o en escala de grises.
- Las funciones de álgebra lineal y muchos de los algoritmos de aprendizaje automático solo funcionan con vectores de punto flotante.
- Las funciones básicas, como `CV::add`, soportan todos los tipos.
- Las funciones de conversión del espacio de color soportan los tipos 8 bits sin signo, 16 bits sin signo y punto flotante de 32 bits.

Los subgrupos de tipos soportados por cada función han sido definidos a partir de las necesidades prácticas y se pueden ampliar en el futuro.

3.3.2. Vectores de Entrada y Salida

Muchas funciones de OpenCV procesan grandes vectores bidimensionales o multidimensionales. Normalmente, dichas funciones usan `cpp:class:Mat` como parámetros,

pero en algunos casos es conveniente usar `std::vector<>` (Para un conjunto de puntos, por ejemplo) o `Matx<>` (para la matriz de homografía 3×3). Para evitar muchos duplicados en el API, se han introducido clases *proxy* especiales. La case *proxy* básica es `InputArray`. Se usa para pasar vectores de solo lectura como entrada a una función. La clase derivada de `InputArray`, `OutputArray` se utiliza para especificar los vectores de salida de la función. Normalmente, no hay que preocuparse de estos tipos de datos intermedios (igualmente no hay que declarar variables de estos tipo de forma explícita), todo se realiza de forma automática. Se puede suponer que en lugar de `InputArray/OutputArray` siempre se puede usar `Mat`, `std::vector<>`, `Matx<>`, `Vec<>` o `Scalar`. Cuando una función posee vectores de entrada o salida opcionales y no se tiene o no se necesita alguno, se pasa `cv::noArray()`.

3.3.3. Manejo de errores

OpenCV usa excepciones para indicar los errores críticos. Cuando el dato de entrada tiene un formato correcto y pertenece al rango de valores especificado, pero el algoritmo no puede finalizar por algún motivo (por ejemplo, el algoritmo de optimización no converge), este devuelve un código de error especial (por lo general, una variable booleana).

Las excepciones pueden ser instancias de la clase `cv::Exception` o sus derivadas. A la vez, `cv::Exception` es un derivado de `std::Exception`. Lo que puede ser manejado con elegancia en el código usando otros componentes de la biblioteca estándar C++.

La excepción es lanzada típicamente ya sea usando la macro `CV_Error(errcode, description)`, o su variante impresa `CV_Error_(Errcode, printf-spec, (printf-args))`, o a través de la macro `CV_Assert(condition)` que chequea la condición y lanza una excepción cuando no se cumple. Para código de rendimiento crítico, existe `CV_DbgAssert(condition)` que sólo se mantiene en la configuración de depuración. Gracias al manejo automático de la memoria, todos los buffers intermedios se liberan automáticamente en caso de un error repentino. Solo hay que añadir una declaración "*try*" para capturar las excepciones, si es necesario (véase un ejemplo en el código 3.5).

```
try
{
    ... // call OpenCV
}
catch( cv::Exception& e )
{
    const char* err_msg = e.what();
```

```
    std::cout << "exception caught: " << err_msg << std::endl;
}
```

Código 3.5: Ejemplo de manejo de excepciones en OpenCV.

3.4. Módulos de OpenCV

En esta sección se hará un estudio introductorio de los diferentes módulos de OpenCV prestando mayor atención a los módulos y métodos que serán de interés en la creación de la aplicación.

3.4.1. android

Este módulo proporciona posibilidad de leer ficheros con formato Android en OpenCV o exportar datos con formato OpenCV a Android. También se encarga de la conversión de `Bitmap` Android a `Mat` OpenCV y viceversa. Sólo contiene una clase:

`Utils` Contiene los métodos citados anteriormente. Véase figura (3.2).

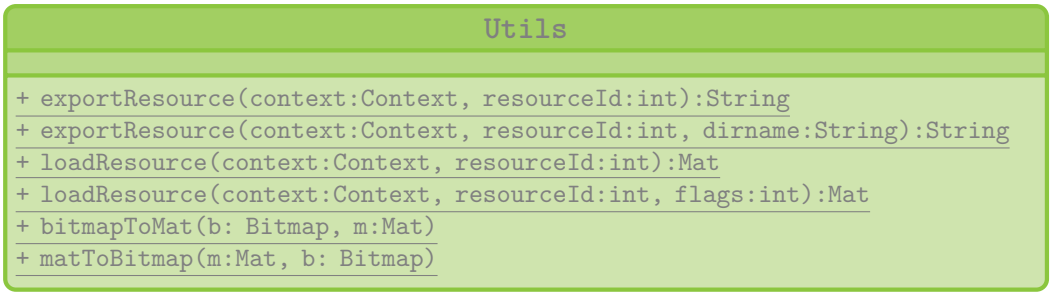


Figura 3.2.: Representación UML de la clase `Utils`.

3.4.2. core. La funcionalidad básica

El módulo *core* proporciona toda la funcionalidad básica que es común a todos los módulos. Consta de diversas clases:

`Algorithm` Se trata de una clase base para todos los algoritmos más o menos complejos en OpenCV (figura 3.3). Sobre todo para las clases de algoritmos que pueden poseer múltiples implementaciones. La clase proporciona las características siguientes para todas sus clases derivadas:

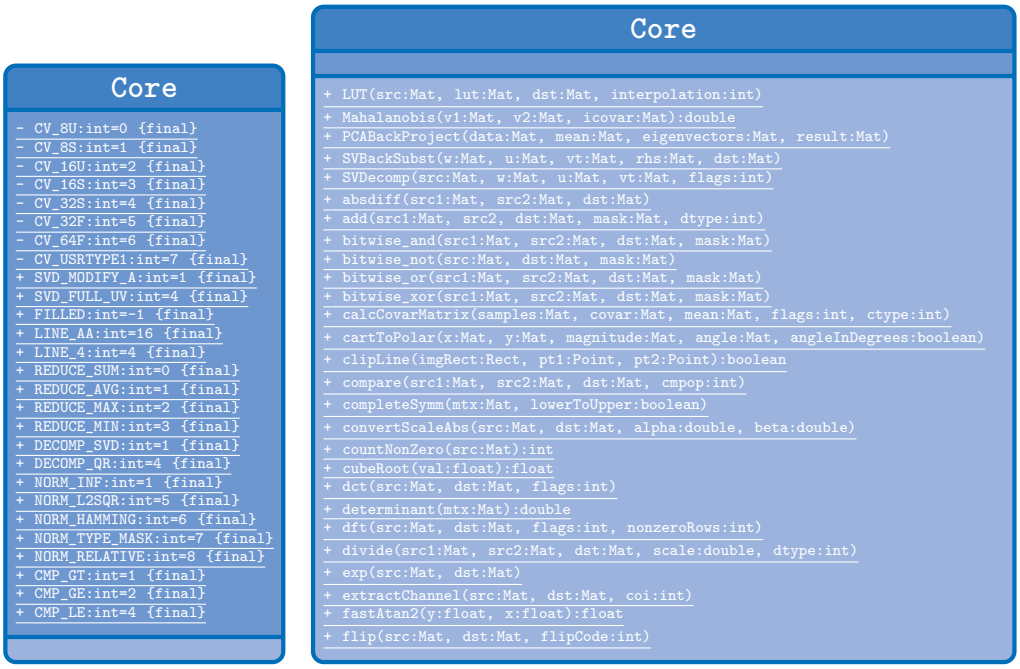
Figura 3.3.: Representación UML de la clase `Algorithm`.

- El denominado "constructor virtual". Es decir, cada derivado de `Algorithm` se registra al inicio del programa y se puede obtener la lista de algoritmos registrados y crear una instancia de un algoritmo determinado por su nombre.
- Ajuste/recuperación de los parámetros del algoritmo por su nombre.
- La lectura y escritura de los parámetros desde/hacia archivos XML o YAML. Cada derivado de `Algorithm` puede guardar todos sus parámetros y luego leerlos de nuevo. No hay necesidad de volver a implementarlos cada vez.

`Core` Contiene las operaciones básicas sobre matrices, como la suma (`add`) y la multiplicación (`multiply`), operaciones bit a bit (`bitwise_and`), etc. Posee también los métodos para dibujar (`rectangle`, `line`) o incluir texto (`putText`) sobre las matrices. Igualmente, posee de algunas utilidades o funciones del sistema (`getNumberOfCPUs`, `getTickCount`). En la figura (3.4) se ve una parte de la clase *core*. Para ver la clase al completo, consulte el anexo A.

`CvException` Manejo de las excepciones.

`CvType` Plantilla de clase "característica" para los tipos de datos primitivos de OpenCV. Los tipos de datos primitivos de OpenCV son `unsigned char`, `bool`, `signed char`, `unsigned short`, `signed short`, `int`, `float`, `double`, o una tupla de valores de alguno de estos tipos, donde todos los valores de la lista tienen el mismo tipo. Cualquier tipo primitivo de la la lista puede definirse por un identificador en la forma `CV_<profundidad de bit>{U|S|F}C(<numero de canales>)`, por ejemplo, `uchar` → `CV8UC1`, tupla de 3 elementos punto flotante→ `CV_32FC3`, etcétera. Una estructura universal de OpenCV que es capaz de almacenar una sola



(a) Atributos **(b)** Métodos

Figura 3.4.: Representación UML de la clase `Core`.

instancia de un tipo de datos primitivo es **`Vec`**. Para almacenar múltiples instancias de cada tipo, se utiliza `std::vector`, `Mat`, `Mat_`, `SparseMat`, `SparseMat_`, o cualquier otro contenedor que pueda almacenar instancias de `Vec`. El propósito principal de esta clase es el de convertir la información en tiempo de compilación en tipos de datos compatibles con OpenCV. En la figura (3.5b) se muestran algunos de los tipos de datos. El resto se puede consultar en el anexo A.

`Mat` Representa un vector numérico n-dimensional de uno o varios canales. Se puede usar para almacenar vectores o matrices de valores reales o complejos, imágenes a color o en escala de grises, volúmenes voxel, campos vectoriales, nubes de puntos, tensores, histogramas (aunque los histogramas con gran número de canales deberían se almacenados en `SparseMat`). La disposición de los datos de la matriz M se define mediante el vector `M.step()`, con el fin de que la dirección de elemento $(i_0, ..., i_{M.dims-1})$, donde $0 \leq i_k < N.size[k]$, sea calculada como:

$$addr\left(M_{i_o,...,i_{dims-1}}\right) = M.data + M.step[0] * i_0 + M.step[1] * i_1 + ...+ \\ +M.step[M.dims - 1] * i_{M.dims-1} \tag{3.2}$$

En el caso de un array bidimensional, la anterior fórmula se reduce a:

$$addr(M_{i,j}) = M.data + M.step[0] * i + M.step[1] * j \tag{3.3}$$

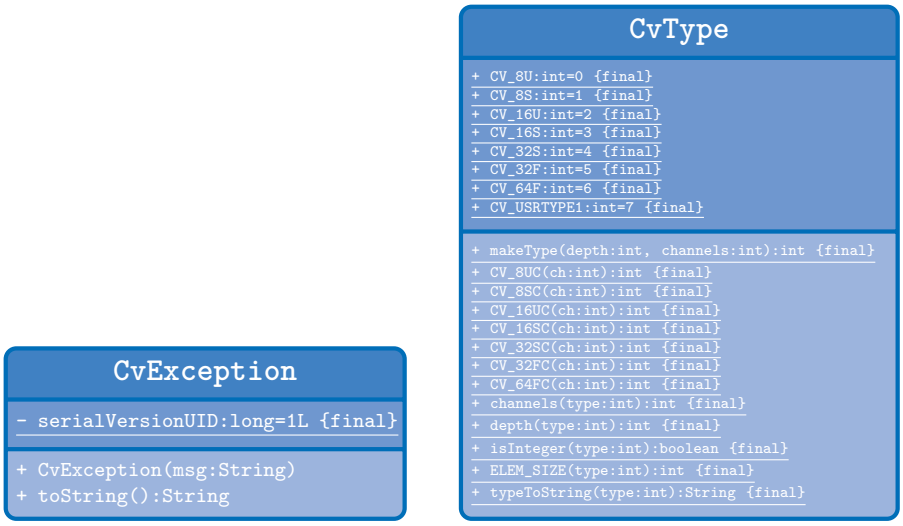


Figura 3.5.: Representación UML de la clase `CvException` (a) y de `CvType` (b).

Nótese que se indica `M.step[i]>=M.step[i+1]` (en realidad, `M.step[i]>=M.step[i+1]*M.size[i+1]`). Esto significa que las matrices bidimensionales se almacenan fila a fila, las de tres dimensiones se almacenas plano a plano, y así sucesivamente. `M.steps[M.dims-1]` es mínima y siempre es igual al tamaño del elemento `M.elemSize()`.

Por lo tanto, la estructura de datos en `Mat` es totalmente compatible con la mayoría de los tipos grandes de arrays de los kits de desarrollo y SDKs estándar. Debido a esta compatibilidad, es posible crear una cabecera `Mat` para los datos asignados por el usuario y procesarlos usando OpenCV.

De la clase `Mat` derivan a su vez las siguientes clases:

`MatOfByte` Para matrices con elementos de 8 bits sin signo (`8UC(x)`).

`MatOfDMatch` Para matrices de 4 canales con elementos de 32 bits en punto flotante (`32FC4`).

`MatOfDouble` Para matrices con elementos de 64 bits en punto flotante (`64FC(x)`).

`MatOfFloat` Para matrices con elementos de 32 bits en punto flotante (`64FC1`).

`MatOfFloat4` Para matrices de 4 canales con elementos de 32 bits en punto flotante (`64FC4`).

`MatOfFloat6` Para matrices de 6 canales con elementos de 32 bits en punto flotante (`64FC6`).

MatOfInt Para matrices con elementos enteros de 32 bits con signo (32SC1).

MatOfInt4 Para matrices de 4 canales con elementos enteros de 32 bits con signo (32SC4).

MatOfKeyPoint Para almacenar puntos claves sus elementos son de 32 bits en punto flotante y posee 7 canales (32FC4).

MatOfPoint Almacena las coordenadas de los puntos como elementos enteros de 32 bits con signo y en 2 canales (32SC2), uno para la coordenada en x y otro para la coordenada en y.

MatOfPoint2f Almacena las coordenadas de los puntos como elementos en punto flotante de 32 bits en 2 canales (32FC2).

MatOfPoint3 Almacena las coordenadas de los puntos espaciales como elementos enteros de 32 bits con signo y en 3 canales (32SC3), uno para la coordenada en x, otro para la coordenada en y y otro para la coordenada en z.

MatOfPoint3f Almacena las coordenadas espaciales de los puntos como elementos en punto flotante de 32 bits en 3 canales (32FC3).

MatOfRect Almacena rectángulos como elementos enteros de 32 bits con signo y en 4 canales (32SC4), 2 canales para las coordenadas x e y de la esquina superior izquierda y los otros 2 para el alto y el ancho del rectángulo.

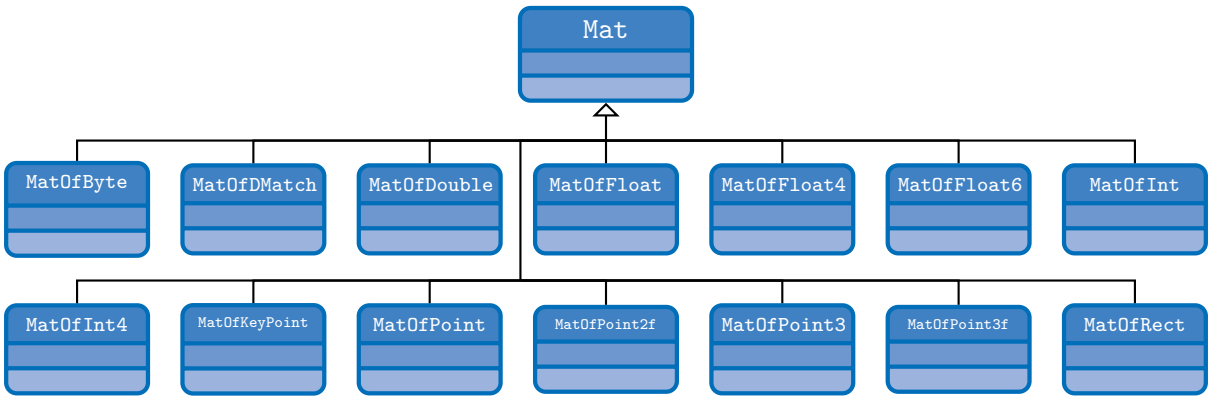


Figura 3.6.: Diagrama de herencia de la clase Mat. El contenido de las clases se muestra en el anexo A.

Point Clase para puntos 2D definidos por sus coordenadas x e y. Existe un operador *cast* para convertir las coordenadas del punto al tipo especificado. La conversión de coordenadas en punto flotante a entera se realiza mediante redondeo. Comúnmente, la conversión utiliza esta operación para cada una de las coordenadas. Soporta aritmética vectorial y las operaciones de comparación (figura 3.7a).

Point3 Clase para los puntos 3D especificados por sus coordenadas x, y, z. De forma similar a `Point`, las coordenadas de los puntos 3D se pueden convertir a otros tipos. La aritmética vectorial y las operaciones de comparación están también soportadas (figura 3.7b).

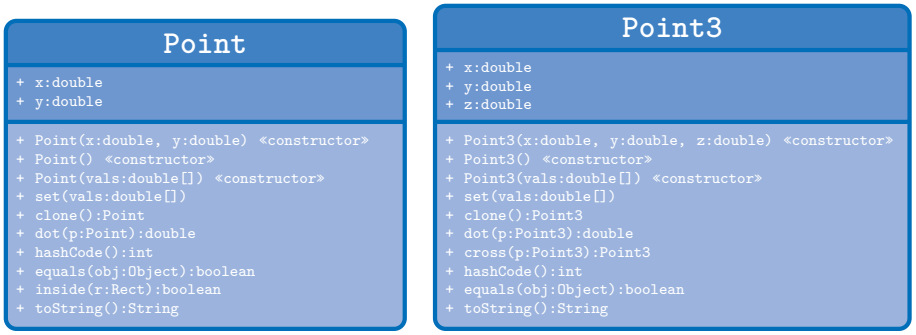


Figura 3.7.: Representación UML de las clases `Point` (a) y `Point3` (b).

Range Clase que especifica una subsecuencia (fragmento) de una secuencia (figura 3.8). Se utiliza para especificar la extensión de una fila o una columna en una matriz (`Mat`) y para muchos otros propósitos. `start` es el margen izquierdo del rango y es inclusivo, `end` es el margen derecho y es exclusivo. Como intervalo semiabiero, normalmente se denota como [*start*, *end*).

El método estático `Range::all()` retorna una variable especial que significa "la secuencia completa" o " "el rango completo". Todos los métodos y funciones de OpenCV que posean `Range` soportarán este valor especial `Range::all()`.

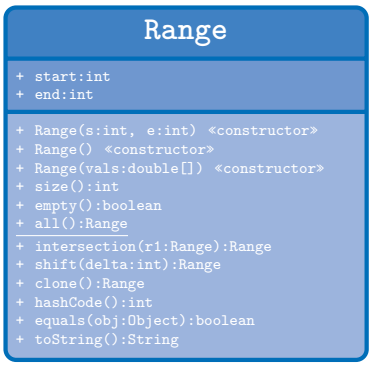


Figura 3.8.: Representación UML de la clase `Range`

Rect Clase para rectángulos 2D (figura 3.9a), descrita por los siguientes parámetros:

- Coordenadas de la esquina superior izquierda. Esta es la interpretación por defecto de `Rect::x` y de `Rect::y` en OpenCV. Por lo tanto en los algoritmos se tendrá en consideración que x e y se cuentan a partir de la esquina superior izquierda.

- Ancho y alto del rectángulo.

OpenCV normalmente asume que el límite superior e izquierdo del rectángulo son inclusivos, mientras que el derecho y el inferior no lo son.. Por ejemplo, el método `Rect_::contains` devuelve `true` si

$$x \leq pt.x < x + width,\ y \leq pt.y < y + height \tag{3.4}$$

Además de los miembros de la clase, se implementan las siguientes operaciones sobre rectángulos:

- `rect = rect` $\pm$ `point` (movimiento de un rectángulo por cierto desplazamiento).
- `rect = rect` $\pm$ `size` (expansión o contracción de un rectángulo cierta cantidad).
- `rect += point`, `rect -= point`, `rect += size`, `rect -= size` (operaciones de aumento).
- `rect = rect1 & rect2` (intersección de rectángulos).
- `rect = rect1 | rect2` (rectángulo de área mínima que contiene a `rect1` y `rect2`).
- `rect &= rect1`, `rect |= rect1` (y sus respectivas operaciones de aumento).
- `rect == rect1`, `rect != rect1` (comparación de rectángulos).

`RotatedRect` Clase para los rectángulos rotados especificados por su centro, tamaño y la rotación en grados (figura 3.9b).

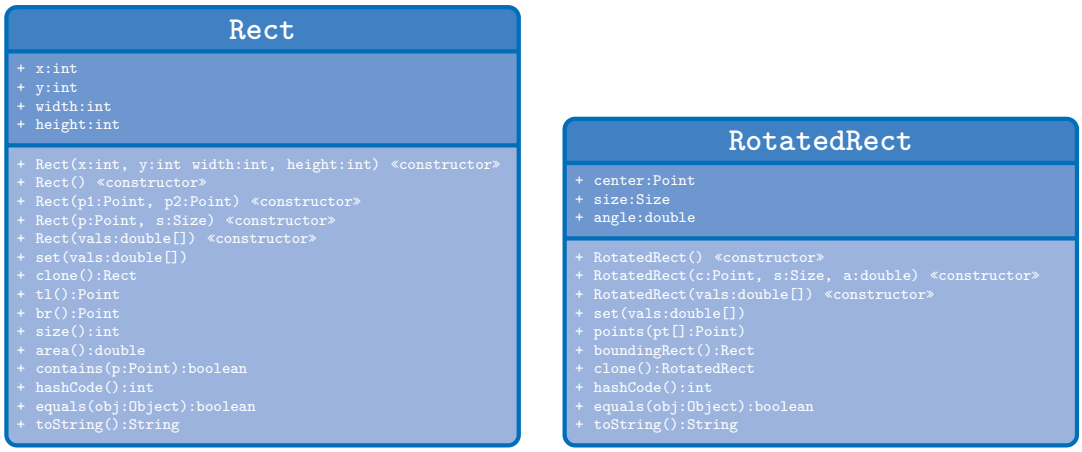


Figura 3.9.: Representación UML de las clases `Rect` (a) y `RotatedRect` (b).

`Scalar` Clase para vectores de 4 elementos derivada de `Vec` (figura 3.10a). Al derivar de `Vec<_Tp,4>`, `Scalar` puede ser utilizado sólo como vector de 4 elementos. El tipo `Scalar` es utilizado ampliamente para pasar valores de píxeles (p.e. los colores).

`Size` Clase que se utiliza para especificar el tamaño de una imagen o rectángulo. La clase tiene dos miembros denominados `width` y `height`. Dispone del mismo juego de instrucciones aritméticas y de comparación que `Point` (figura 3.10b) .

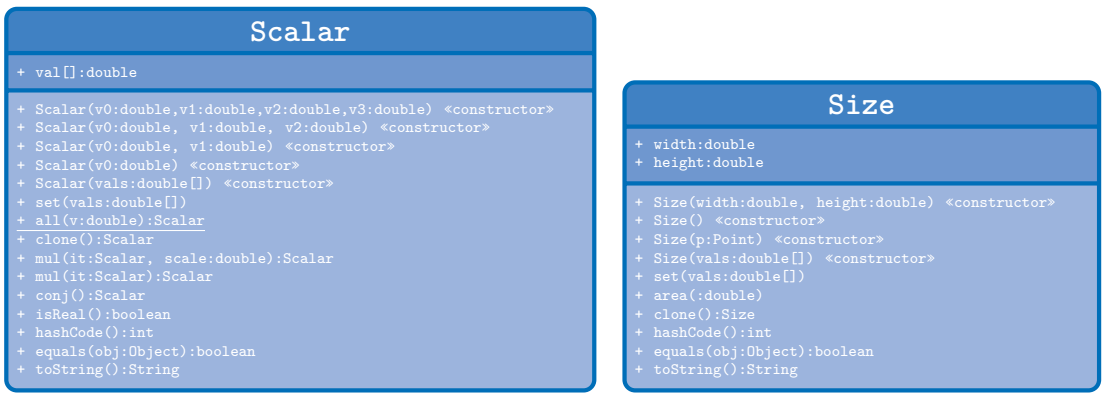


Figura 3.10.: Representación UML de las clases `Scalar` (a) y `Size` (b).

`TermCriteria` Clase que define los criterios de finalización de los algoritmos iterativos (figura 3.11).

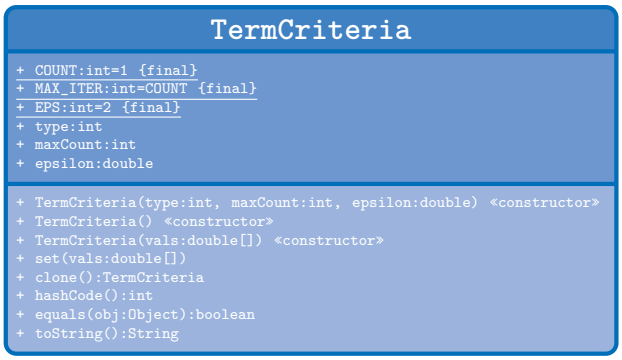


Figura 3.11.: Representación UML de la clase `TermCriteria`.

3.4.3. imgproc. Procesado de imágenes

En este módulo están incluidas las operaciones de procesamiento de imágenes. Según su aplicación, estas operaciones se clasifican en diversos tipos:

- Filtrado de imágenes: Se utilizan para realizar varias operaciones de filtrado lineal o no lineal sobre imágenes 2D (representados como `Mat()`s). Esto significa que por cada localización (x, y) de un píxel en la imagen fuente (normalmente rectangular), se considera y usa su vecindad para calcular la respuesta. En el caso de un filtro lineal, es una suma ponderada de los valores de los píxeles. En el caso de operaciones morfológicas, son los valores mínimo o máximo, etcétera. La respuesta calculada se almacena en la imagen destino en la misma posición (x, y). Esto implica que la señal de salida tendrá el mismo tamaño que la de entrada. Normalmente, las funciones soportan arrays multicanal, en cuyo caso cada canal es procesado independientemente. Por lo tanto, la señal de salida tendrá el mismo número de canales que la de entrada.

  Otra característica común de las funciones y clases descritas en esta sección es que, al contrario de las funciones aritméticas simples, necesitan extrapolar valores da algunos píxeles no existentes. Por ejemplo, si se quiere suavizar una imagen usando un filtro Gaussiano 3×3, entonces, cuando se procesan la píxeles más a al izquierda en cada fila, se necesitan píxeles a la izquierda de estos, e decir, fuera de la imagen. Se puede dejar que esos píxeles sean los mismos que los de más a la izquierda (método de extrapolación de "borde replicado"), o asumir que los píxeles no existentes son ceros (método de extrapolación de "borde constante"), etcétera. OpenCV permite especificar el método de extrapolación.

- Transformaciones de la imagen. Estas pueden ser:

  - Geométricas: Se realizan sobre imágenes 2D. No cambian el contenido de la imagen sino la cuadrícula de píxeles y mapean dicha cuadrícula deformada

en la imagen destino. De hecho, para evitar artefactos de muestreo (como el *aliasing*), el mapeo se realiza en orden inverso, de destino a origen. Es decir, por cada píxel (x, y) de la imagen destino, las funciones calculan las coordenadas del correspondiente píxel "donante" en la imagen origen y copian su valor:

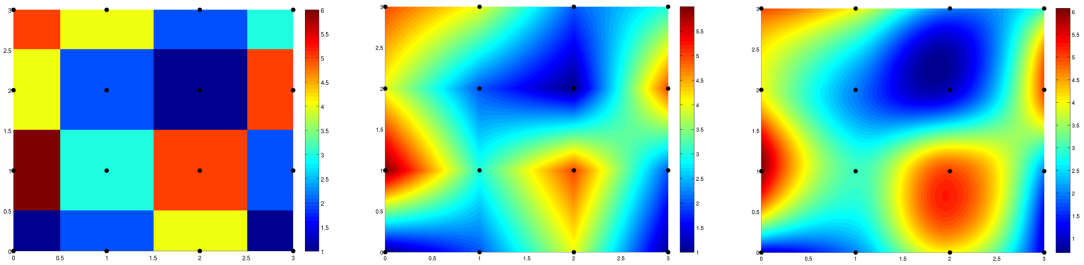
$$\texttt{dst}(x, y) = \texttt{src}(f_x(x, y),\ f_y(x, y)) \tag{3.5}$$

En el caso de que se especifique un mapeo directo $< g_x, g_y >: src \rightarrow dst$, las funciones de OpenCV primero calculan el correspondiente mapeo inverso $< f_x, f_y >: dst \rightarrow src$ y entonces usan la fórmula (3.5).

Las implementaciones actuales de las transformaciones geométricas, necesitan resolver los problemas principales con la fórmula anterior:

- La extrapolación de píxeles no existentes: De forma similar a las funciones de filtrado descritas en el punto anterior, para algunos (x, y), uno de los $f_x(x, y)$, o $f_y(x, y)$, o incluso ambos, pueden caer fuera de la imagen. En este caso, se necesita emplear algún método de extrapolación. OpenCV proporciona las mismas opciones de extrapolación que para las funciones de filtrado. Adicionalmente, proporciona el método `BORDER_TRANSPARENT` que hace que los píxeles correspondientes en la imagen destino no sean modificados.

- La interpolación de los valores del píxel. Normalmente $f_x(x, y)$ y $f_y(x, y)$ son números punto flotante. Esto significa que $< f_x,\ f_y >$ puede ser una transformación afín o de perspectiva, o una corrección de la distorsión radial de las lentes, etcétera. Por tanto, se necesita obtener un valor de píxel en coordenadas fraccionales. En el caso más simple, las coordenadas se pueden redondear a las coordenadas enteras más cercanas y usar los píxeles correspondientes. A esto se le conoce como interpolación por el vecino más cercano [44]. Sin embargo, se pueden obtener mejores resultados empleando métodos de interpolación más sofisticados [45], donde se introduce una función polinómica en un entorno del píxel calculado $(f_x(x, y),\ f_y(x, y))$, y entonces se toma en valor del polinomio en $(f_x(x, y),\ f_y(x, y))$ como el valor del píxel interpolado. En OpenCV se puede escoger entre varios métodos de interpolación. En la figura (3.12) se muestran varios ejemplos de interpolación.



(a) Vecino más cercano. **(b)** Bilineal. **(c)** Bicúbica.

Figura 3.12.: Métodos de interpolación.

- Misceláneas: Son aquellas que no afectan a la geometría de la imagen, como: aplicar umbrales adaptativos o fijos a los píxeles de una imagen (figura 3.13) , conversiones del espacio de color, cáculo de la distancia al pixel cero, relleno

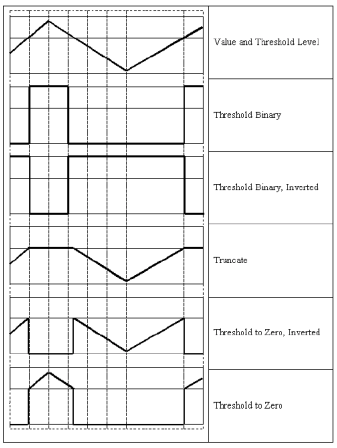


Figura 3.13.: Umbrales disponibles en OpenCv.

de la imagen, restauración de una región de la imagen, cálculo de la integral de una imagen (figura 3.14), segmentar una imagen aplicando el algoritmo Watershed [46] o el algoritmo GrabCut [47].

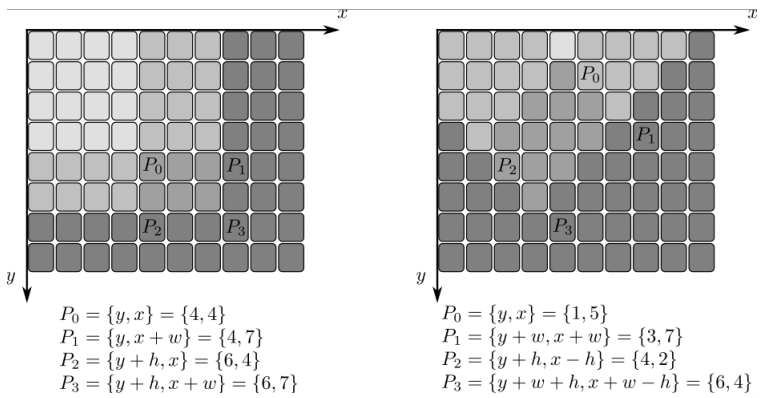


Figura 3.14.: Cálculo de la integral de un rectángulo derecho `Rect(3,3,3,2)` y de un rectángulo rotado `Rect(5,1,2,3)`.

- Histogramas: Un histograma es una representación gráfica de una variable en forma de barras, donde la superficie de cada barra es proporcional a la frecuencia de los valores representados. OpenCV permite calcular los histogramas de una imagen, calcular su retroproyección, comparar histogramas, calcular la distancia de "mínimo trabajo" [48] entre dos configuraciones de puntos ponderadas, ecualizar el histograma de una imagen a escala de grises o localizar una plantilla dentro de una imagen mediante la comparación de histogramas (figura 3.15).

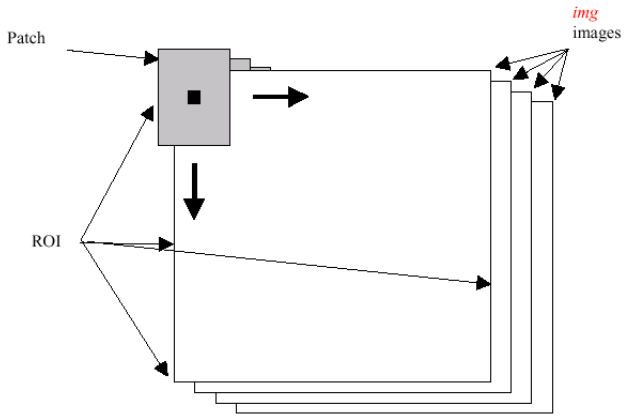


Figura 3.15.: Busqueda de una plantilla dentro de una imagen.

- Análisis estructural y descriptores de forma: Permite realizar operaciones sobre las formas de una imagen tales como localizar contornos en una imagen binaria, dibujar líneas de contorno o contornos rellenos, aproximar una curvas poligonales, aproximar cadenas de *Freeman* [49] con una curva poligonal, calcular el perímetro y el área de un contorno o el largo de una curva así como el rectángulo vertical de delimitación de un conjunto de puntos, halla la envolvente convexa de un conjunto de puntos, encuentra a los defectos de convexidad de un contorno (figura 3.16), ajustar una elipse alrededor de un conjunto de puntos 2D o una línea

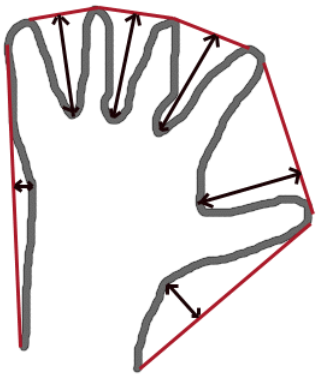


Figura 3.16.: Defectos de convexidad del contorno de una mano.

a un conjunto de puntos 2D o 3D, comprobar la convexidad de un contorno,

encontrar el rectángulo rotado o el círculo de menor área que encierra el conjunto de puntos 2D, comparar dos formas, determinar cuando un punto está dentro de un contorno, fuera, o se encuentra en un borde (o coincide con un vértice) (figura 3.17)

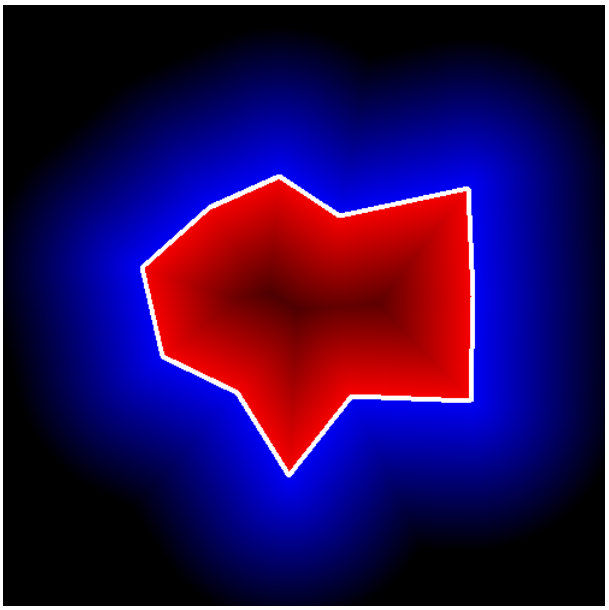


Figura 3.17.: Salida de una función en la que cada píxel se ha testado contra el contorno (en blanco).

Igualmente, permite conocer la estructura de la imagen mediante el cálculo de diversos momentos:

- Momentos hasta el de tercer orden de un polígono o forma rasterizada [50]. En el caso de una imagen rasterizada, los momentos espaciales `Moments` :: $\mathtt{m}_{ji}$ se calculan como:

$$\mathtt{m}_{ji} = \sum_{x,y} (\mathtt{array}(x,y) \cdot x^j \cdot y^i) \tag{3.6}$$

  Los momentos centrales `Moments` :: $\mathtt{mu}_{ji}$ se calculan como:

$$\mathtt{mu}_{ji} = \sum_{x,y} (\mathtt{array}(x,y) \cdot (x - \overline{x})^j \cdot (y - \overline{y})^i) \tag{3.7}$$

  donde $(\overline{x}, \overline{y})$ es el centro de gravedad:

$$\overline{x} = \frac{m_{10}}{m_{00}},\ \overline{y} = \frac{m_{01}}{m_{00}} \tag{3.8}$$

Los momentos centrales normalizados `Moments` :: $\texttt{nu}_{ji}$ se calculan como:

$$\texttt{nu}_{ji} = \frac{\texttt{mu}_{ji}}{m_{00}^{(i+j)/2+1}} \tag{3.9}$$

Los momentos de un contorno se definen de la misma forma, pero se calculan usando la fórmula de Green [51]. Por tanto, debido a una limitada resolución de rasterizado, los momentos calculados para un contorno son ligeramente diferentes a los momentos calculados para el mismo contorno rasterizado.

- Invariantes Hu [52, 53](hasta siete) definidas como:

$$\begin{aligned}
hu[0] &= \eta_{20} + \eta_{02} \\
hu[1] &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
hu[2] &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
hu[3] &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\
hu[4] &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + \\
&\quad +(3\eta_{21} + \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
hu[5] &= (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \\
hu[6] &= (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] - \\
&\quad -(\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]
\end{aligned} \tag{3.10}$$

donde η_{ji} se corresponde con `Moments` :: $\texttt{nu}_{ji}$.

Está demostrado que estos valores permanecen invariantes ante el escalado de imagen, la rotación y la reflexión a excepción del séptimo, cuyo signo cambia en la reflexión. Esta invarianza está demostrada suponiendo resoluciones de imagen infinitas. En el caso de imágenes rasterizadas, las invariantes Hu calculadas para las imágenes originales y las transformadas son un poco diferentes.

- Subdivisiones planas: La subdivisión planar es la división de un plano en un conjunto de regiones no solapadas (facetas) que cubren el plano completamente. La subdivisión está constituida por un conjunto de puntos 2D, donde los puntos están unidos entre sí y forman un gráfico planar, que, junto con unos cuantos bordes que conectan los puntos de subdivisión exteriores (es decir, puntos de envolvente convexa) con el infinito, subdivide el plano en facetas por sus bordes.

  Para cada subdivisión, hay una subdivisión dual en la cual las facetas y los puntos (vértices de la subdivisión) intercambian sus papeles. Esto significa que

una faceta es tratada como un vértice (denominado punto virtual en adelante) de la subdivisión dual y los vértices de la subdivisión original se convierten en facetas. En la figura (3.18), se muestran las subdivisiones originales con línea sólida y la subdivisión dual con línea de puntos.

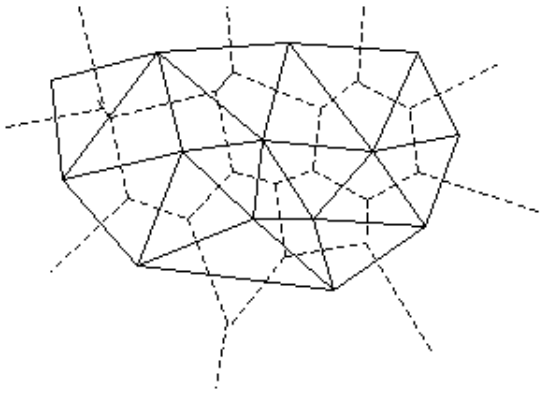
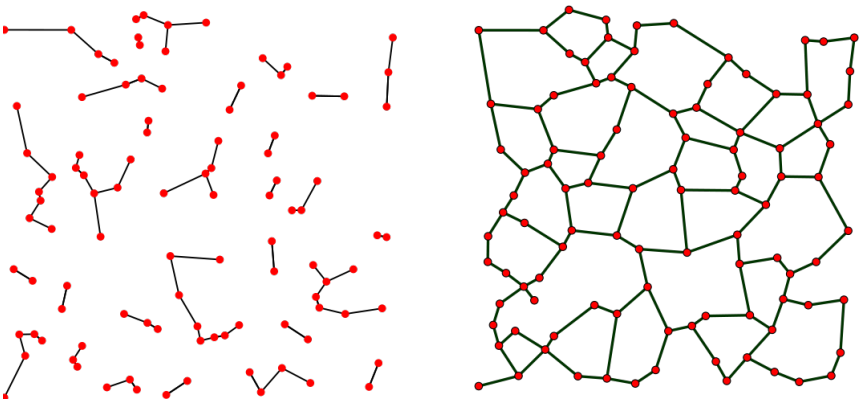


Figura 3.18.: Subdivisión planar.

OpenCV subdivide el plano en triángulos empleando el algoritmo de Delaunay [54]. La subdivisión se realiza de forma iterativa empezando por un triángulo ficticio que se sabe que contiene todos los puntos de la subdivisión. En este caso, la subdivisión dual es un diagrama de Voronoi del conjunto de puntos 2D. Las subdivisiones pueden ser utilizadas para la transformación 3D a trozos de un plano, morphing, localización rápida de puntos en el plano, crear gráficas especiales (como NNG (figura 3.19a), o RNG (figura 3.19b)).

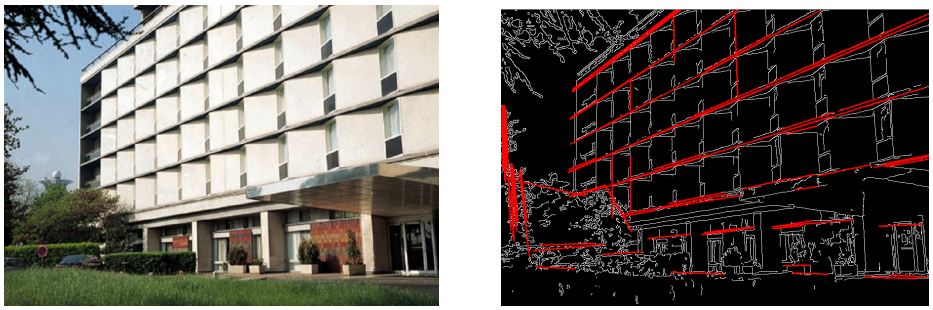


(a) NNG **(b)** RNG

Figura 3.19.: Gráficas especiales de la subdivisión planar.

- Análisis de movimiento y seguimiento de objetos.
- Detección de características: Permite encontrar bordes en una imagen empleando el algoritmo Canny [28], calcular los autovalores y autovectores [55] de bloques

de la imagen o el algoritmo de Harris [56] para la detección de esquinas, localizar círculos o líneas en una imagen en escala de grises empleando la transformada de Hough [57] o segmentos de líneas en una imagen binaria empleando la transformada de Hough probabilística (figura 3.20).



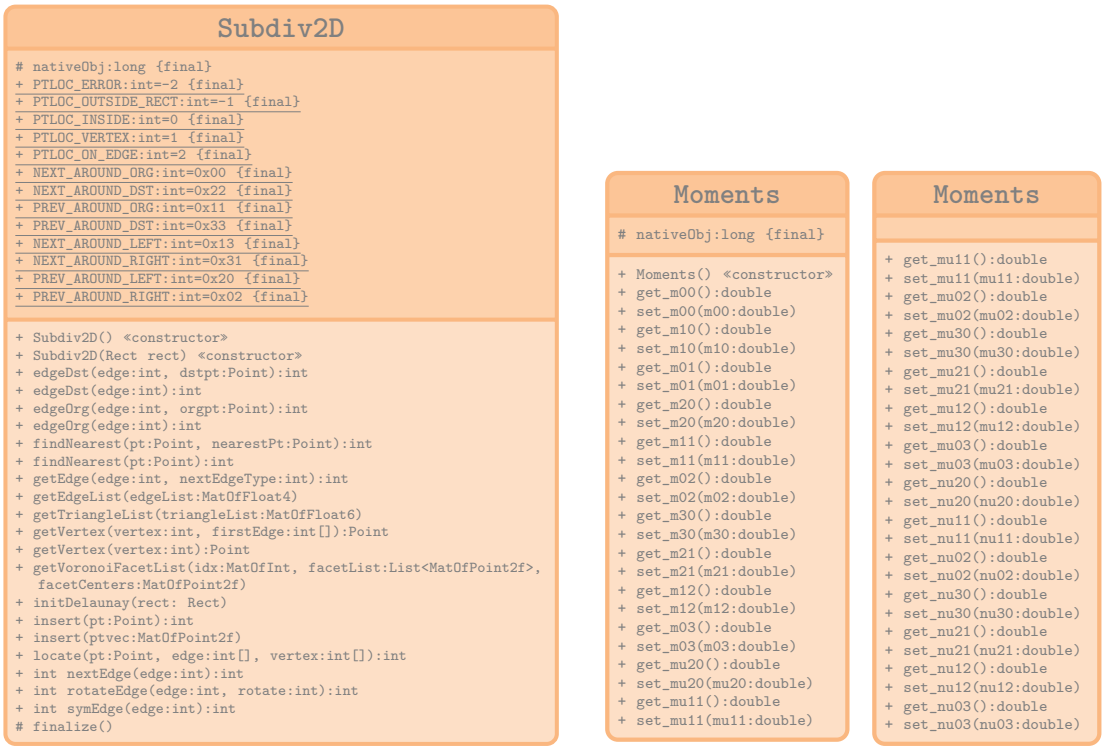
(a) Imagen original. **(b)** Resultado.

Figura 3.20.: Ejemplo de la transformada de Hough probabilística.

El módulo *imgproc* consta de las siguientes clases:

`Subdiv2D` Define los atributos y métodos que se emplean en la subdivisión planar (figura 3.21a).

`Moments` Permite leer o actualizar los distintos momentos desde Android ya que en OpenCV siempre se almacenan de forma nativa (C\C++). (Figura 3.21b).



(a) **(b)**

Figura 3.21.: Representación UML de las clases `Subdiv2D` (a) y `Moments` (b).

Imgproc Define el resto de métodos y atributos que se emplean en las demás operaciones de procesado de imágenes. En la figura (3.22) se muestra un resumen de la clase (para ver la clase al completo consulte el anexo B).

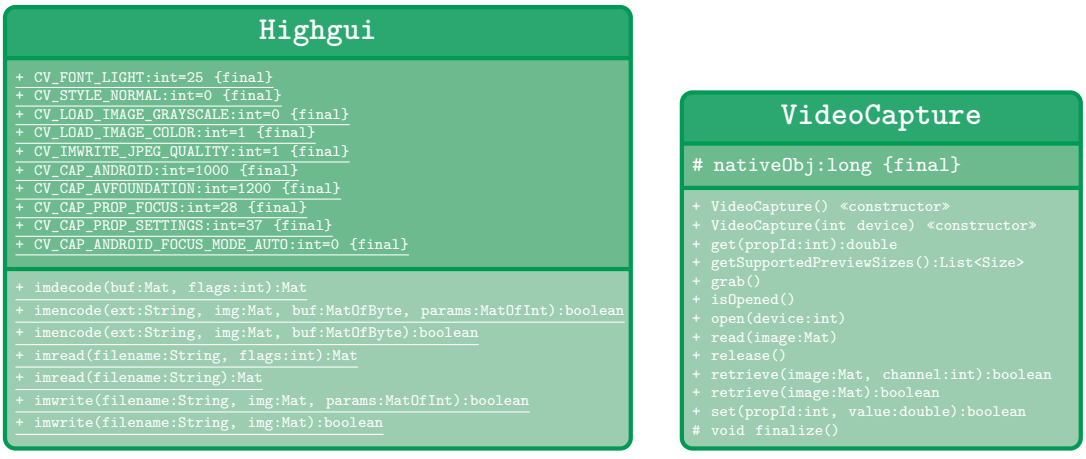
Figura 3.22.: Representación UML de la clase `Imgproc`.

3.4.4. highgui. Medios de E/S

Este es el módulo que se encarga de la lectura y escritura de imágenes y video. Consta de dos clases:

Highgui Se encarga de la lectura y escritura de las imágenes en buffers o ficheros. En la figura (3.23a) se muestra la clase con alguno de sus atributos. Veáse la clase al completo en el anexo C.

VideoCapture Se encarga de la configuración y captura de imágenes desde la cámara (figura 3.23b).



(a) **(b)**

Figura 3.23.: Representación UML de la clase `Highgui` (a) y `VideoCapture` (b).

3.4.5. video. Análisis de movimiento y seguimiento de objetos

Este módulo contiene las siguientes clases:

`BackgroundSubtractor` Clase base para segmentación trasera/frontal. Se emplea únicamente para definir la interfaz común de toda la familia de algoritmos de segmentación trasera/frontal (figura 3.24).

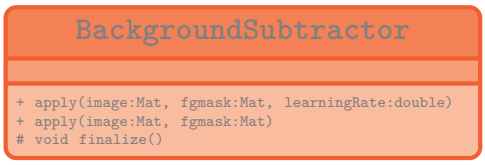


Figura 3.24.: Representación UML de la clase `BackgroundSubtractor`.

`BackgroundSubtractorMOG` Implementa el algoritmo de segmentación trasera/frontal basado en mezcla Gaussiana descrito en [58] (figura 3.25).

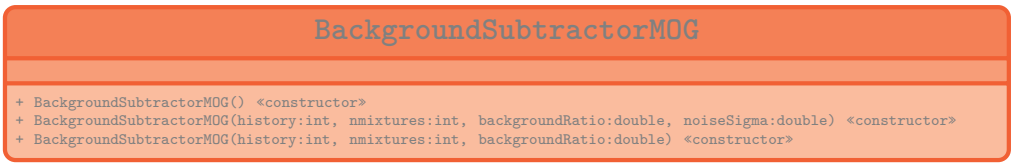


Figura 3.25.: Representación UML de la clase `BackgroundSubtractorMOG`.

`KalmanFilter` La clase implementa un filtro Kalman estándar [59, 60] (figura 3.26).

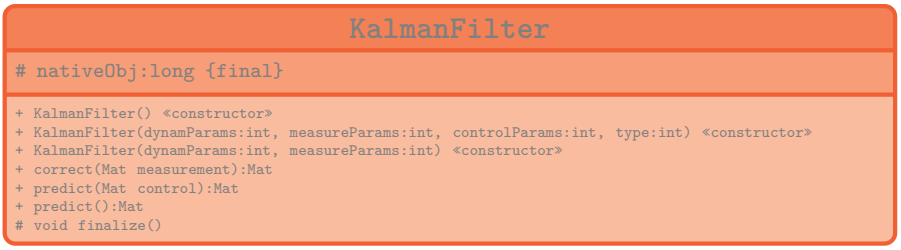


Figura 3.26.: Representación UML de la clase `KalmanFilter`.

`Video` Contiene el resto de métodos para el análisis de movimiento y seguimiento de objetos (figura 3.27).

Figura 3.27.: Representación UML de la clase `Video`.

3.4.6. features2d. Framework de características 2D

Define los distintos métodos de detección y descripción de características. Posee las siguientes clases:

`Keypoint` Los detectores de características en OpenCV tienen contenedores con una interfaz común que le permite cambiar fácilmente entre diferentes algoritmos para resolver el mismo problema. Esta clase se encarga de definir la estructura de datos para detectores de puntos importantes (figura 3.28).

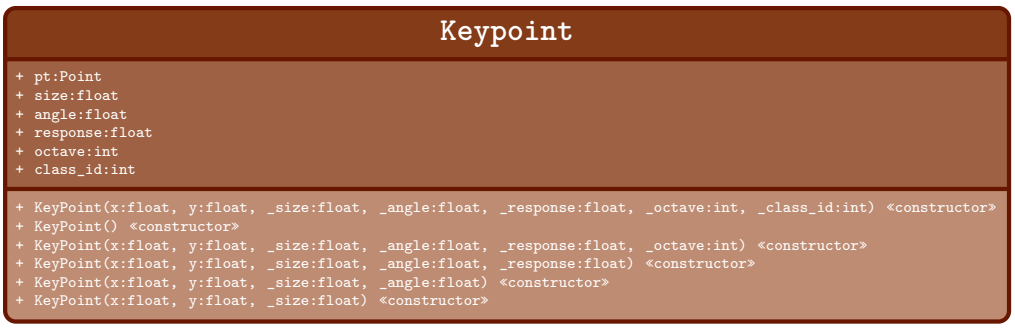


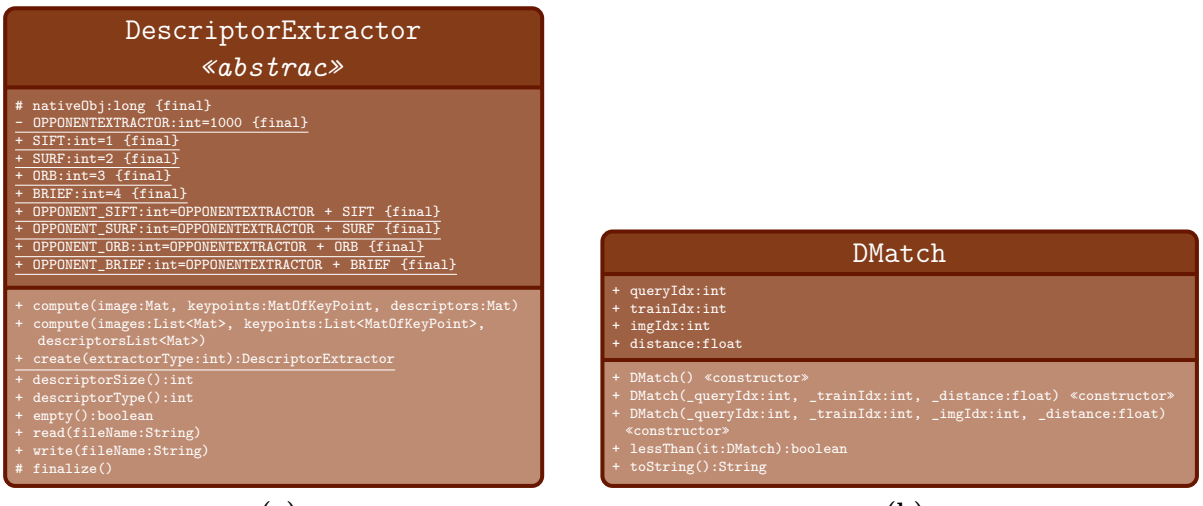
Figura 3.28.: Representación UML de la clase `Keypoint`.

`FeatureDetector` Clase abstracta para los detectores de características en imágenes 2D. Todos los objetos que implementan detectores de puntos clave heredan la interfaz `FeatureDetector` (figura 3.29).

Figura 3.29.: Representación UML de la clase `FeatureDetector`.

`DescriptorExtractor` Clase abstracta base para el cálculo de descriptores para los puntos clave de la imagen. Los extractores de descriptores de puntos clave en OpenCV tienen envolventes con una interfaz común que permite alternar entre diferentes algoritmos que resuelven el mismo problema. Esta sección está dedicada a calcular los descriptores representados como vectores en un espacio multidimensional. Todos los objetos que implementan el vector de extractores de descriptores heredan la interfaz `DescriptionExtractor` (figura 3.30a).

`DMatch` Clase para hacer coincidir los descriptores de puntos clave: índice descriptor de consulta, índice descriptor de entrenamiento, el índice de la imagen de entrenamiento, y la distancia entre los descriptores (figura 3.30b).



(a) **(b)**

Figura 3.30.: Representación UML de las clases `DescriptorExtractor` (a) y `DMatch` (b).

`DescriptorMatcher` Clase abstracta base para hacer coincidir descriptores de puntos clave. Tiene dos grupos de métodos de igualación: Para hacer coincidir los descriptores de una imagen con otra imagen o con un conjunto de imágenes. Todos los objetos que implementan el vector de igualadores de descriptores heredan la interfaz `DescriptionMatcher` (figura 3.31).

Figura 3.31.: Representación UML de la clase `DescriptorMatcher`.

`GenericDescriptorMatcher` Interfaz abstracta para extraer y comparar un descriptor de puntos clave. Para este propósito están también `DescriptorExtractor` y `DescriptorMatcher` pero sus interfaces están destinadas a descriptores representados como vectores en un espacio multidimensional. `GenericDescriptorMatcher` es una interfaz mas genérica para los descriptores. `DescriptorMatcher` y `GenericDescriptorMatcher` tienen dos grupos de métodos de comparación: para comparar puntos clave de una imagen con otra imagen o con un conjunto de imágenes (figura 3.32).

`Features2d` Se encarga de dibujar las coincidencias en la imagen destino (figura 3.33).

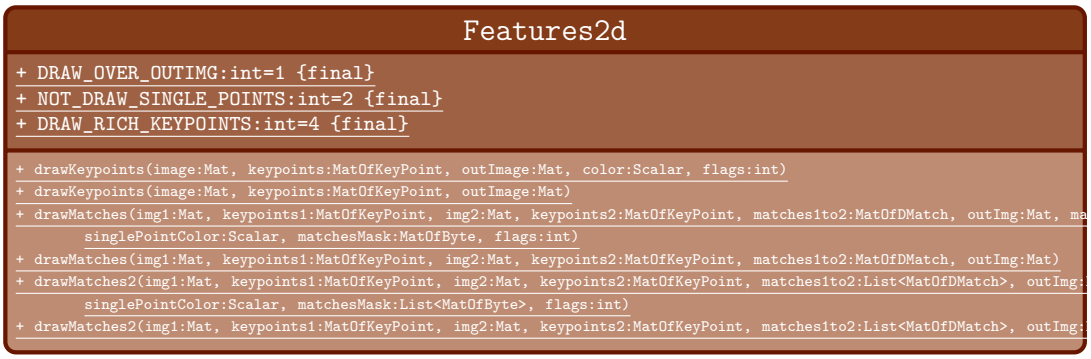


Figura 3.33.: Representación UML de la clase `Features2d`.

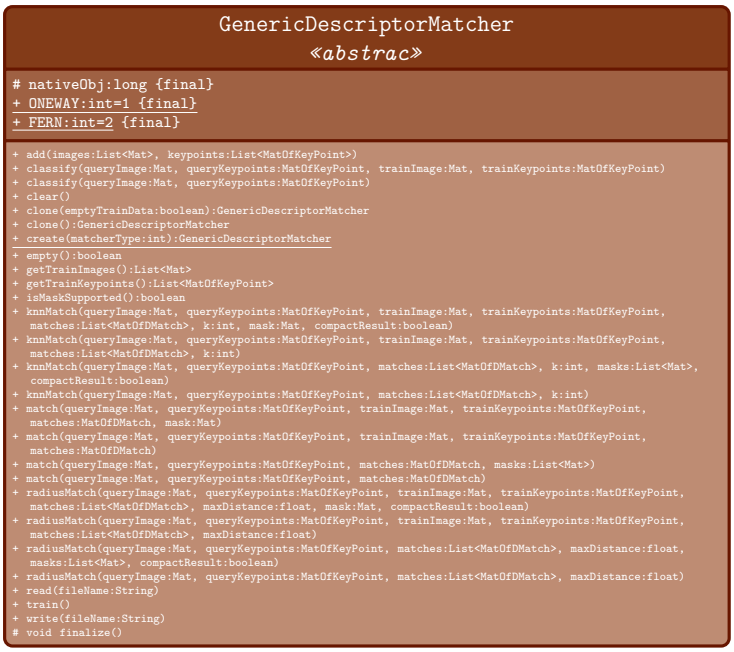


Figura 3.32.: Representación UML de la clase `GenericDescriptorMatcher`.

3.4.7. objdetect. Detección de objetos

Se encarga de la detección de objetos mediante el uso de clasificadores en cascada. OpenCV implementa los detectores *Haar-like*, propuesto por Paul Viola [35] y mejorado por Rainer Lienhart [34]; *LBP* (*Local Binary Patterns*) [61]; y *HOG* (*Histogram of Oriented Gradients*) [62]. Consta de las siguientes clases:

`CascadeClassifier` Clase para calcular los valores de las características en los clasificadores en cascada *Haar-like* y *LBP* (figure 3.34).

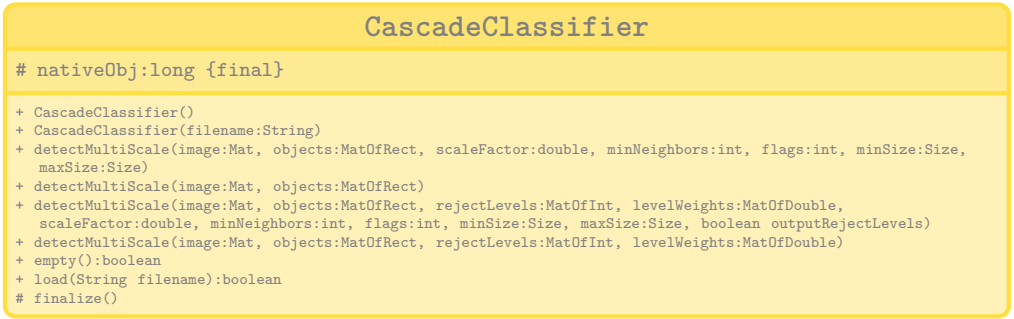


Figura 3.34.: Representación UML de la clase `CascadeClassifier`.

`HOGDescriptor` Clase para calcular los valores de las características en los clasificadores en cascada *HOG* (figura 3.35).

Figura 3.35.: Representación UML de la clase `HOGDescriptor`.

`Objdetect` Otros métodos para la detección de objetos (figura 3.36).

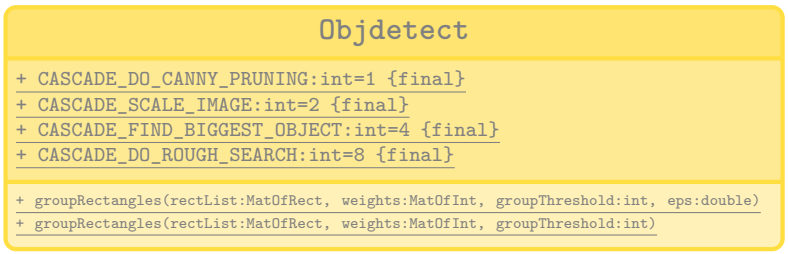


Figura 3.36.: Representación UML de la clase `Objdetect`.

3.4.8. ml. Aprendizaje automático

Machine Learning Library (MLL) es un conjunto de clases y funciones para la clasificación estadística, la regresión, y la agrupación de datos. Aunque los algoritmos tienen diferentes conjuntos de características (como la capacidad de manejar mediciones desaparecidas o las variables categóricas de entrada), hay algo de base común entre las clases. Este módulo tiene varias clases que están agrupadas según la funcionalidad que

aportan o el algoritmo de aprendizaje automático que implementan. El contenido de las clases se puede consultar en el anexo D.

3.4.8.1. Modelos estadísticos

Supone la base del aprendizaje automático. Se especifica en:

`CvStatModel` Clase base para modelos estadísticos en ML.

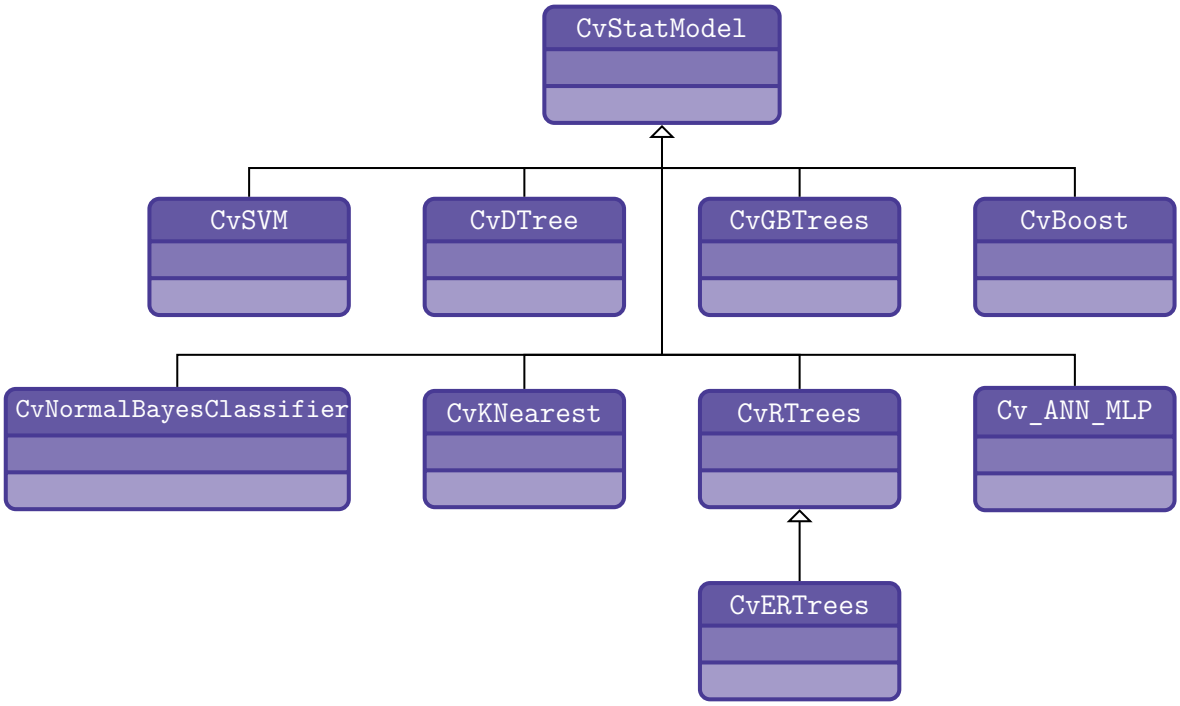


Figura 3.37.: Mecanismo de herencia de `CvStatModel`.

3.4.8.2. Clasificador normal de Bayes

Este modelo de clasificación simple asume que el factor de características de cada clase tiene una distribución normal (aunque, no necesariamente independientemente distribuida). Por lo tanto, se asume que la función de distribución de datos total es una mezcla Gaussiana, un componente por clase. Utilizando los datos de entrenamiento, el algoritmo estima los vectores promedio y las matrices de covarianza para cada clase, y luego los emplea para la predicción. Se desarrolla mediante la clase:

`CvNormalBayesClassifier` Clasificador de Bayes para datos distribuidos de forma normal.

3.4.8.3. K-vecinos más cercanos

El algoritmo almacena todas las muestras de entrenamiento y pronostica la respuesta a una nueva muestra analizando un cierto número (K) de los vecinos más cercanos de la muestra utilizando escrutinio, calculando sumas ponderadas, etcétera. A veces se hace referencia a este método como "aprendizaje por ejemplos" debido a que para la predicción busca el vector de características con una respuesta conocida que está más cerca del vector dado. Se implementa en la clase:

`CvKNearest` La clase implementa el modelo K-vecinos más cercanos.

3.4.8.4. Máquinas de Vectores de Soporte

Originalmente, las máquinas de vectores de soporte (SVM[2]) fueron una técnica para construir clasificadores binarios (2 clases) óptimos. Posteriormente, la técnica se extendió a problemas de regresión y agrupamiento. SVM es un caso parcial de los métodos basados en núcleos. Mapea vectores de características en un espacio dimensional superior con una función del núcleo y construye una función de discriminación óptima en ese espacio o un hiperplano óptimo que encaja dentro de los datos de entrenamiento. En el caso de SVM, no se define el núcleo de manera explícita. En su lugar, se necesita definir una distancia entre dos puntos cualquiera en el hiperespacio.

La solución es óptima, lo que significa que el margen entre el hiperplano de separación y los vectores de características más cercanos de las dos clases (en el caso de un clasificador de 2 clases) es máxima. Los vectores de características que están más cercanos al hiperplano son denominados vectores de soporte, lo que significa que la posición de otros vectores no afecta al hiperplano.

La implementación de SVM en OpenCV esta basada en [63].

Consta de las siguientes clases:

`CvParamGrid` La estructura representa el rango de la rejilla logarítmica de parámetros `CvStatModel`. Se utiliza para optimizar la precisión de `CvStatModel` mediante la variación de los parámetros del modelo, la estimación de la precisión se calcula mediante validación cruzada.

La rejilla determina la siguiente la siguiente secuencia de iteración de los valores del parámetro `CvStatModel:`

[2]Support Vector Machines

$$\left(min\_val, min\_val * step, min\_val * step^2, \ldots, min\_val * step^n\right) \quad (3.11)$$

donde n es es el máximo índice satisfactorio.

La rejilla es logarítmica, por lo tanto `step` deberá ser siempre mayor que 1.

`CvSVMParams` Parámetros de entrenamiento de SVM. La estructura debe inicializarse y pasarse al método de entrenamiento de `CvSVM`.

`CvSVM` Máquinas de Vectores de Soporte.

3.4.8.5. Árboles de decisión

Las clases ML mostradas en esta sección implementan los algoritmos de Árboles de Clasificación y Regresión descritos en [64].

Un árbol de decisión es un árbol binario (árbol donde cualquier nodo no hoja tiene dos nodos hijos). Puede ser utilizado tanto para clasificación como para regresión. Para clasificación, cada hoja del árbol se marca con una etiqueta de clase; múltiples hojas pueden tener la misma etiqueta. Para regresión, también se asigna una constante a cada hoja del árbol, por lo que la función de aproximación por tramos es constante.

Predicción con árboles de decisión. Para alcanzar un nodo hoja y obtener una respuesta para el vector de características de entrada, el procedimiento de predicción comienza con el nodo raíz. Para cada nodo no-hoja el procedimiento va a la izquierda (selecciona el nodo hijo izquierdo como siguiente nodo observado) o a la derecha basándose en el valor de cierta variable cuyo índice está almacenado en el nodo observado. Son posibles las siguientes variables:

- Variables ordenadas: El valor de la variable se compara con un umbral que también está almacenado en el nodo. Si el valor es menor que el umbral, el procedimiento va a la izquierda. En otro caso, va a la derecha. Por ejemplo, si en peso es menor a 1 kilogramo, el procedimiento va a la izquierda y si no, a la derecha.
- Variables categóricas: El valor de una variable discreta se analiza para ver si pertenece a un determinado subconjunto de los valores (también almacenados en el nodo) de un conjunto limitado de valores que puede tomar la variable. Si lo es, el procedimiento va a la izquierda. En otro caso, va a la derecha. Por ejemplo, si el color es rojo o verde, va al izquierda, si no a la derecha.

Por lo tanto, en cada nodo se emplea un par de entidades (`variable_index, decision_rule(threshold subset)`). A este par se le denomina como una *división* (división en la variable `variable_index`). Una vez se alcanza un nodo hoja, se emplea el valor asignado a dicho nodo como salida del procedimiento de predicción.

A veces, ciertas características del vector de entrada se pierden (por ejemplo, en la oscuridad es difícil determinar el color de un objeto), y el procedimiento de predicción puede bloquearse en cierto nodo (en el ejemplo mencionado, si el nodo está dividido por color). Para evitar estas situaciones, los árboles de decisión utilizan las llamadas *divisiones sustitutas*. Es decir, además de la mejor división "primaria", cada nodo del árbol también puede dividirse por otra u otras variables con aproximadamente los mismos resultados.

Entrenamiento de los árboles de decisión. El árbol se construye de forma recursiva, empezando desde el nodo raíz. Todos los datos de entrenamiento (vectores de características y sus respuestas) se emplean para dividir el nodo raíz. En cada nodo se encuentra la regla de decisión óptima (la mejor división "primaria") basada en algún criterio. En aprendizaje automático, los criterios de "pureza" `gini` [65] se emplean para clasificación y la suma de errores cuadráticos se utiliza para la regresión. Entonces, si es necesario, se buscan las divisiones sustitutas. Se asemejan a los resultados de la división primaria en los datos de entrenamiento. Todos los datos se dividen empleando las divisiones primarias y sustitutas (tal como se hace en el procedimiento de predicción) entre los nodos hijos izquierdo y derecho. A continuación, el procedimiento de forma recursiva divide ambos nodos. En cada nodo el procedimiento recursivo puede detenerse (deja de dividir más el nodo) en uno de los siguientes casos:

- La profundidad de ramas del árbol construidas ha alcanzado el valor máximo especificado.
- El número de muestras de entrenamiento en el nodo es menor que el umbral especificado cuando no es estadísticamente representativo para dividir más el nodo.
- Todas las muestras en el nodo pertenecen a la misma clase o, en el caso de la regresión, la variación es muy pequeña.
- La mejor división encontrada no proporciona ninguna mejoría notable comparada con una elección aleatoria.

Cuando el árbol está construido, si es necesario, puede podarse utilizando un procedimiento de validación cruzada. Es decir, algunas ramas del árbol que pueden conducir

al sobreajuste del modelo se cortan. En general, este procedimiento sólo se aplica a los árboles de decisión autónomos. Normalmente los ensambladores de árboles crean árboles que son lo suficientemente pequeños y utilizan sus propios sistemas de protección contra el sobreajuste.

Importancia de la variable. Además de la predicción que es un uso obvio de los árboles de decisión, el árbol se puede usar también para vários análisis de datos. Una de las propiedades esenciales de los algoritmos de árboles de decisión construidos es una capacidad de calcular la importancia (poder de decisión relativo) de cada variable. Por ejemplo, en un filtro de spam que emplea un conjunto de palabras aparecidas en el mensaje como vector de características, se puede emplear la calificación de importancia de la variable para determinar las palabras más "indicadoras de spam" y por lo tanto mantener el diccionario en un tamaño razonable.

La importancia de cada la variable se calcula en todas las divisiones de dicha variable en el árbol, tanto primarias como sustitutas. Por lo tanto, para calcular la importancia de la variable correctamente, las divisiones sustitutas deben esta autorizadas en los parámetros de entrenamiento, incluso si no hay datos faltantes.

Esta sección consta de las siguientes clases:

`CvDTreeParams` La estructura almacena todos los parámetros del árbol de decisión. Se puede inicializar mediante el constructor por defecto y así anular los parámetros directamente antes del entrenamiento, o la estructura puede ser inicializada completamente utilizando la variante avanzada del constructor.

`CvDTree` La clase implementa un árbol de decisión simple que se puede utilizar de manera aislada o como clase base en en conjuntos de árboles (véase Potenciación (en la página siguiente) y Árboles Aleatorios (en la página 72)).

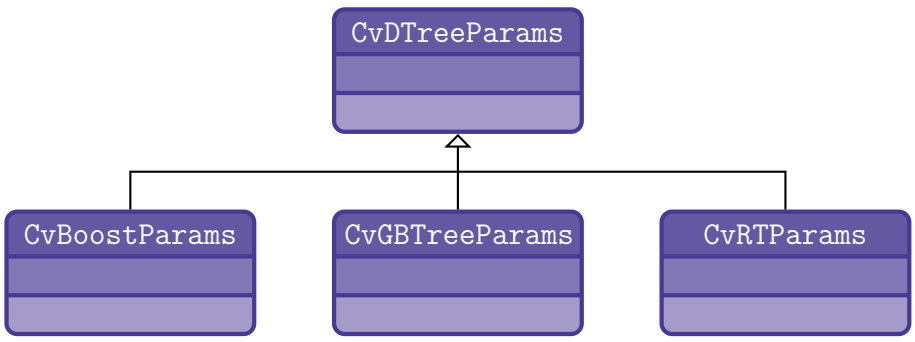


Figura 3.38.: Mecanismo de herencia de `CvDTreeParams`.

3.4.8.6. Potenciación

Una de las tareas comunes del aprendizaje automático es el aprendizaje supervisado. En el aprendizaje supervisado, la meta es aprender la relación funcional $F : y = F(x)$ entre la entrada x y la salida y. La predicción de salida cualitativa se denomina "clasificación", mientras que la predicción cuantitativa se denomina "regresión".

La potenciación es un concepto de aprendizaje poderoso que proporciona una solución a la tarea de de aprendizaje supervisado de clasificación. Combina el rendimiento de muchos clasificadores "débiles" para producir una asociación potente [66]. Un clasificador débil sólo necesita ser mejor que el azar, y por eso puede ser muy simple y computacionalmente poco costoso. De todas formas, muchos de ellos combinan de manera eficiente los resultados en un clasificador fuerte que a menudo supera a clasificadores fuertes más "monolíticos" como las *SVMs* o las Redes Neuronales.

Los árboles de decisión son los clasificadores débiles más utilizados en los esquemas de potenciación. A menudo los árboles de decisión con sólo un nodo de división por árbol (llamados `stumps`[3]) son suficientes.

El modelo potenciado se basa en N ejemplos de entrenamiento $(x_i, y_i)\, 1N$ con $x_i \in R^k$ e $y_i \in -1, +1$. x_i es un vector de K componentes. Cada componente codifica una característica relevante para la tarea de aprendizaje a la mano. La salida de dos clases deseada se codifica como -1 y $+1$.

Las diferentes variantes de potenciación se conocen como *AdaBoost* Discreta, *AdaBoost* Real, *LogitBoost*, y *AdaBoost* Suave [67]. Todas ellas son similares en cuanto a su estructura general. Por lo tanto, nos centraremos sólo en el algoritmo estándar de dos clases *AdaBoost* Discreto, descrito a continuación. Inicialmente se le asigna el mismo peso a cada muestra (paso 2). Entonces, se entrena un clasificador débil $f_m(x)$ en los datos ponderados (paso 3a). Se calcula su error de entrenamiento ponderado y su factor de escalado c_m (paso 3b). Se aumenta el peso de los datos clasificados erróneamente (paso 3c). Se normalizan todos los pesos, y el proceso para encontrar el siguiente clasificador débil continua durante otras $M - 1$ veces. El clasificador $F(x)$ final es el signo de la suma ponderada de los clasificadores débiles individuales (paso 4).

[3] tocones

Algoritmo Adaboost Discreto de dos clases:

1. Establece N ejemplos $(x_i, y_i)\, 1N$ con $x_i \in R^k$ e $y_i \in -1, +1$.
2. Asigna los pesos como $w_i = {}^1/_N$, $i = 1, \ldots, N$.
3. Repite $m = 1, 2, \ldots, M$ veces:
   a) Monta el clasificador $f_m(x) \in -1, +1$, utilizando los pesos w_i en los datos de entrenamiento.
   b) Calcula $err_m = E_w\left[1_{(y \neq f_m(x))}\right]$, $c_m = \log\left({}^{(1-err_m)}/_{err_m}\right)$.
   c) Fija $w_i \Leftarrow w_i \exp\left[c_m 1_{(y_i \neq f_m(x_i))}\right]$, $i = 1, 2, \ldots, N$ y re normaliza de forma que $\sum i w_i = 1$.
4. Clasifica las nuevas muestras x utilizando la fórmula $sign\left(\sum m = 1 M c_m f_m(x)\,.\right)$

Para reducir el tiempo de compilación para los modelos potenciados sin una perdida sustancial de precisión, se puede emplear la técnica de recorte por influencia. A medida que avanza el algoritmo de entrenamiento y el número de árboles en el conjunto es mayor, un mayor número de los ejemplos de entrenamiento se clasifican de forma correcta y con mayor confianza, así que dichas muestras reciben pesos menores en las iteraciones siguientes. Los ejemplos con un peso relativo muy bajo tienen un impacto pequeño en el entrenamiento del clasificador débil. Por lo tanto, dichos ejemplo se pueden excluir durante el entrenamiento del clasificador débil sin tener mucho efecto en el clasificador inducido. Este proceso se controla con el parámetro `weight_trim_rate`. Sólo las muestras con la fracción síntesis `weght_trim_rate` del peso total se utilizan en la formación del clasificador débil. Hay que tener en cuenta que los pesos de **todas** las muestras de entrenamiento se recalculan en cada iteración del entrenamiento. Las muestras eliminadas en una iteración en particular se pueden usar de nuevo para aprender alguno de los clasificadores débiles más adelante [67].

Para la realización de este algoritmo, el módulo ml proporciona las siguientes clases:

`CvBoostParams` Parámetros de entrenamiento de Potenciación. La estructura deriva de
`CvDTreeParams` (figura 3.38) pero no todos los parámetros del árbol de decisión están soportados. En concreto, no se admite la validación cruzada. Todos los parámetros son públicos. Se pueden inicializar mediante un constructor y entonces anular algunos de ellos directamente si se desea.

`CvBoost` Árbol clasificador potenciado derivado de `CvStatModel`.

3.4.8.7. Árboles Potenciados Degradados

Árboles Potenciados Degradados (GBT[4]) es un algoritmo generalizado de potenciación introducido por Jerome Friedman [68]. En contraste con el algoritmo *AdaBoost.M1*, *GBT* puede hacer frente a problemas de clasificación multiclase y de regresión. Además, se puede utilizar cualquier función de pérdida diferencial, algunas de las más populares están implementadas. El uso de árboles de decisión (`CvDTree`) como aprendices base permite procesar variables ordenadas y categóricas.

Entrenamiento del modelo GBT. El modelo GBT representa un conjunto de árboles de regresión simples construidos de manera codiciosa. El procedimiento de entrenamiento es un proceso iterativo similar a la optimización numérica mediante el método de gradiente descendente. El resumen de pérdidas del conjunto de entrenamiento depende solamente de las predicciones de los modelos actuales para las muestras de entrenamiento, en otras palabras $\sum_{i=1}^{N} L\left(y_i, F\left(x_i\right)\right) \equiv \mathcal{L}\left(F\left(x_1\right), F\left(x_2\right), \ldots, F\left(x_N\right)\right) \equiv \mathcal{L}\left(F\right)$. y el gradiente $\mathcal{L}\left(F\right)$ puede ser calculado como sigue:

$$grad\left(\mathcal{L}\left(F\right)\right) = \left(\frac{\partial L\left(y_1, F\left(x_1\right)\right)}{\partial F\left(x_1\right)}, \frac{\partial L\left(y_2, F\left(x_2\right)\right)}{\partial F\left(x_2\right)}, \cdots, \frac{\partial L\left(y_N, F\left(x_N\right)\right)}{\partial F\left(x_N\right)}\right) \tag{3.12}$$

En cada paso del entrenamiento, se construye un único árbol de regresión para predecir un vector de componentes antigradiente. La longitud del paso se calcula de acuerdo a la función de pérdida y por separado para cada región determinada por la hoja de árbol. Se puede eliminar cambiando directamente los valores de las hojas.

A continuación se muestra el esquema principal del proceso de entrenamiento:

1. Localizar el mejor modelo constante.
2. Para i en $[1, M]$:
   a) Calcular el antigradiente.
   b) Desarrollar un árbol de regresión para predecir los componentes antigradiente.
   c) Cambiar valores en las hojas del árbol.
   d) Añadir el árbol al modelo.

[4]Gradient Boosted Trees

Para problemas de regresión están implementadas las siguientes funciones de pérdida:

- Pérdida cuadrática (`SQUARED_LOSS`): $L(y, f(x)) = \frac{1}{2}(y - f(x))^2$
- Pérdida absoluta (`ABSOLUTE_LOSS`): $L(y, f(x)) = \mid y - f(x) \mid$
- Pérdida de Huber (`HUBER_LOSS`):
  $$L(y, f(x)) = \begin{cases} \delta \cdot \left(\mid y - f(x) \mid -\frac{\delta}{2} \right) & : \mid y - f(x) \mid > \delta \\ \frac{1}{2} \cdot (y - f(x))^2 & : \mid y - f(x) \mid \leq \delta \end{cases}$$
  donde δ es la estimación $\alpha - cuantil$ de $\mid y - f(x) \mid$. En la implementación actual, $\alpha = 0{,}2$.

Para problemas de clasificación están implementadas las siguientes funciones de pérdida:

- Pérdida de desviación o de entropía cruzada (`DEVIANCE_LOSS`): Se crean K funciones, una función para cada clase de salida, y $L(y, f_1(x), \cdots, f_K(x)) = -\sum_{k=0}^{K} 1(y = k) \cdot \ln p_k(x)$, donde $p_k(x) = \frac{\exp f_k(x)}{\sum_{i=1}^{K} \exp f_i(x)}$ es la estimación de la probabilidad de $y = k$.

Como resultado se obtiene el siguiente modelo:

$$f(x) = f_0 + \nu \cdot \sum_{i=1}^{M} T_i(x) \tag{3.13}$$

donde f_0 es la suposición inicial (el mejor modelo constante) y ν es un parámetro de regulación del intervalo $(0, 1]$, en adelante denominado *contracción*.

Predicción con el modelo GBT. Para obtener la predicción del modelo *GBT*, se necesita calcular la suma de las respuestas de todos los árboles en el conjunto. Para problemas de regresión, esta es la respuesta. Para problemas de clasificación, el resultado es $\arg\max_{i=1..K}(f_i(x))$.

Se emplean las siguientes clases para implementar el algoritmo:

`CvGBtreesParams` Parámetros de entrenamiento de *GBT*. La estructura contiene parámetros por cada árbol de decisión individual en el conjunto, así como las características del modelo completo. La estructura deriva de `CvDTreeParams` (figura 3.38) pero no todos los parámetros de los árboles de decisión están permitidos: no se usa la validación cruzada, la poda, y la prioridad de clases.

`CvGBTrees` La clase implementa el modelo del árbol potenciado degradado tal y como se describió anteriormente.

3.4.8.8. Árboles aleatorios

Los árboles aleatorios fueron introducidos por Leo Breinan y Adele Cutler [69]. El algoritmo soporta problemas tanto de regresión como de clasificación. Árboles aleatorios es una colección (conjunto) de árboles predictores a los que se denominará "bosque" más adelante en esta sección (el término también fue propuesto por L. Breiman). La clasificación funciona como sigue: el clasificador de árboles aleatorios toma el vector de características de entrada, lo clasifica con todos los árboles del bosque, y devuelve la etiqueta de clase que haya recibido la mayoría de los "votos". En el caso de una regresión, la respuesta del clasificador es la media de todas las respuestas de todos los árboles en el bosque.

Todos los árboles se entrenan con los mismos parámetros pero sobre conjuntos de entrenamiento diferentes. Estos conjuntos se generan a partir del conjunto de entrenamiento original utilizando el procedimiento de arranque: para cada conjunto de entrenamiento, se selecciona aleatoriamente el mismo número de vectores que en el conjunto original (`=N`). Los vectores se elijen con reemplazamiento. Es decir, algunos vectores pueden aparecer más de una vez y otros no aparecer. En cada nodo de cada árbol entrenado, no se emplean todas las variables para determinar la mejor división, sino un subconjunto aleatorio de ellas. Con cada nodo se genera un nuevo subconjunto. Sin embargo, su tamaño es fijo para todos los nodos y todos los árboles. Es un parámetro de entrenamiento fijo a $\sqrt{número\_de\_variables}$ por defecto. Ninguno de los árboles construidos se poda.

En árboles aleatorios no hay necesidad de ningún procedimiento de estimación de la precisión, como validación cruzada o de arranque, o un conjunto de test separado para obtener una estimación del error de entrenamiento. El error es estimado durante el entrenamiento. Cuando el conjunto de entrenamiento para el árbol actual es dibujado por el muestreo con reemplazo, algunos vectores se dejan fuera (llamado datos *oob* (*out-of-bag*)). El tamaño de los datos *oob* es aproximadamente `N/3`. El error de clasificación se estima utilizando estos datos *oob* como sigue:

1. Obtener una predicción para cada vector, que es oob en relación con el árbol i-ésimo, utilizando el mismo árbol i-ésimo.

2. Después de haber entrenado todos los árboles, para cada vector que alguna vez ha sido *oob*, localizar su clase "ganadora" (la clase que obtuvo la mayoría de los votos en los árboles donde el vector era *oob*) y compararla con la respuesta cierta base.

3. Calcular el error de clasificación estimado como una relación del número de vectores oob mal clasificados para todos los vectores en los datos originales. en el caso

de la regresión, el error *oob* se calcula como el error cuadrático para la diferencia de vectores *oob* dividido por el número total de vectores.

OpenCV dispone de las siguientes clases para aplicar este algoritmo:

`CvRTParams` Parámetros de entrenamiento de los árboles aleatorios. El conjunto de parámetros de entrenamiento para el bosque es uno de los parámetros de entrenamiento para un árbol individual. No obstante, los árboles aleatorios no necesitan todas las funcionalidades/características de los árboles de decisión. Lo más notable, los árboles no se podan, por lo que los parámetros de validación cruzada no se utilizan.

`CvRTrees` La clase implementa el predictor de bosque aleatorio.

3.4.8.9. Árboles extremadamente aleatorios

Los árboles extremamente aleatorios fueron introducidos por Pierre Geurts, Damien Ernst y Louis Wehenkel en el artículo [70]. El algoritmo de crecimiento de árboles aleatorios es similar a árboles aleatorios (bosque aleatorio), pero existen dos diferencias:

1. Los árboles extremadamente aleatorios no aplican el procedimiento de embolsado para construir las muestras de entrenamiento para cada árbol. Se emplea el mismo conjunto de entrenamiento de entrada para entrenar cada árbol.

2. Los árboles extremamente aleatorios escogen una división de nodo muy extrema (se escogen aleatoriamente un índice de variable y un valor de división variable), mientras que en los árboles aleatorios se localiza la mejor división (la óptima según el índice de la variable y del valor de división variable) entre los subconjuntos aleatorios de variables.

`CvERTrees` La clase implementa el algoritmo de árboles extremamente aleatorios. `CvERTrees` se hereda de `CvRTrees` (figura 3.37) y tiene la misma interfaz. Para fijar los parámetros de entrenamiento de los árboles extremadamente aleatorios se usa la clase `CvRTParams` (en esta página).

3.4.8.10. Maximización de expectativas

El algoritmo de maximización de expectativas (EM) estima los parámetros de la función de densidad de probabilidad multivariante en la forma de una distribución mixta Gaussiana con un número determinado de mezclas.

Considérese el conjunto de vectores de N características $\{x_1, x_2, \ldots, x_N\}$ de un espacio Euclideo d-dimensional extraídos de una mezcla Gaussiana:

$$p\left(x; a_k, S_k, \pi_k\right) = \sum_{k=1}^{m} \pi_k p_k\left(x\right),\ \pi_k \geq 0,\ \sum_{k=1}^{m} \pi_k, \tag{3.14}$$

$$p_k\left(x\right) = \varphi\left(x; a_k, S_k\right) = \frac{1}{\left(2\pi\right)^{d/2} \mid S_k \mid^{1/2}} \exp\left\{-\frac{1}{2}\left(x - a_k\right)^{\mathrm{T}} S_k^{-1}\left(x - a_k\right)\right\} \tag{3.15}$$

donde m es el número de muestras, p_k es la distribución de densidad normal con media a_k y matriz de covarianzas S_k, π_k es el pero de la mezcla k-ésima. Dados el número de mezclas M y las muestras x_i, $i = 1..N$ el algoritmo localiza las estimaciones de máxima verosimilitud (MLE) de todos los parámetros de mezcla, es decir, a_k, S_k y π_k:

$$L\left(x, \theta\right) = logp\left(x, \theta\right) = \sum_{i=1}^{N} log\left(\sum_{k=1}^{m} \pi_k p_k\left(x\right)\right) \rightarrow \max_{\theta \in \Theta} \tag{3.16}$$

$$\Theta = \left\{\left(a_k, S_k, \pi_k\right) : a_k \in \mathbb{R}^d, S_k = S_k^{\mathrm{T}} > 0, S_k \in \mathbb{R}^{d \times d}, \pi_k \geq 0, \sum_{k=1}^{m} \pi_k = 1\right\} \tag{3.17}$$

El algoritmo EM es un procedimiento iterativo. Cada iteración incluye dos pasos. En el primer paso (paso de expectativa o paso E), se halla una probabilidad $p_{i,k}$ (denotada α_{ki} en la fórmula a continuación) de que la muestra `i` pertenezca a la mezcla `k` empleando las estimaciones de los parámetros de mezcla disponibles en el momento:

$$\alpha_{ki} = \frac{\pi_k \varphi\left(x; a_k, S_k\right)}{\sum_{j=1}^{m} \pi_j \varphi\left(x; a_j, S_j\right)} \tag{3.18}$$

En el segundo paso (paso de maximización o paso M), se afinan las estimaciones de los parámetros de mezcla empleando las probabilidades calculadas:

$$\pi_k = \frac{1}{N} \sum_{i=1}^{N} \alpha_{ki},\ a_k = \frac{\sum_{i=1}^{N} \alpha_{ki} x_i}{\sum_{i=1}^{N} \alpha_{ki}},\ S_k = \frac{\sum_{i=1}^{N} \alpha_{ki}\left(x_i - a_k\right)\left(x_i - a_k\right)^{\mathrm{T}}}{\sum_{i=1}^{N} \alpha_{ki}} \tag{3.19}$$

De forma alternativa, el algoritmo puede comenzar con el paso M cuando se pueden proporcionar los valores iniciales de $p_{i,k}$. Otra alternativa cuando se desconoce

$p_{i,k}$ es usar un algoritmo de agrupación más simple para pre-agrupar las muestras de entrada y por lo tanto obtener el $p_{i,k}$ inicial. A menudo se emplea al algoritmo `kmeans()` para este propósito.

Uno de los problemas de EM están en el gran número de parámetros a estimar. La mayoría de los parámetros residen en matrices de covarianzas, que son elementos $d \times d$ donde d es la dimensionalidad del espacio de características. Sin embargo, en muchos problemas prácticos, las matrices de covarianza están próximas a la diagonal o incluso a $\mu_k * I$, donde I es una matriz identidad y μ_k es un parámetro de "escalado" dependiente de la mezcla. Por lo tanto, un esquema de calculo robusto podría comenzar con limitaciones severas en las matrices de covarianzas y luego utilizar los parámetros estimados como entradas para un problema de optimización menos limitado (a menudo una matriz de covarianza diagonal ya es una aproximación bastante buena).

OpenCV lo define en la siguiente clase:

`EM` La clase implementa el algoritmo EM tal como se describió anteriormente.

3.4.8.11. Redes neuronales

ML implementa redes neuronales artificiales prealimentadas (feed-forward [71]) o, más concretamente, perceptrones multicapa (MLP), el tipo de redes neuronales comúnmente más usado. MLP consiste en una capa de entrada, una capa capa de salida, y una o más capas escondidas. Cada capa de MLP incluye una o más neuronas direccionalmente vinculadas con las neuronas de las capas anterior o posterior. En la figura (3.39) se muestra un perceptrón de 3 capas con tres entradas, dos salidas, y la capa escondida que contiene 5 neuronas.

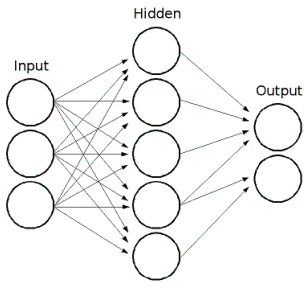


Figura 3.39.: Perceptrón de 3 capas con tres entradas, dos salidas.

Todas las neuronas en MLP son similares. Cada una de ellas tiene diversos puntos de entrada (toma las valores de salida de distintas neuronas en la capa anterior como entrada) y varios vínculos de salida (pasa la respuesta a distintas neuronas en

la siguiente capa). Los valores recuperados de la capa anterior se suman con diversos pesos, individual para cada neurona, más el término de sesgo. La suma se transforma utilizando la función de activación f que también puede ser diferente para diferentes neuronas.

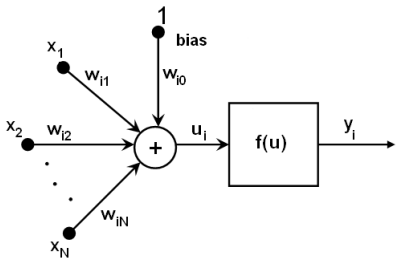


Figura 3.40.: Modelo de neurona.

En otras palabras, dadas las salidas x_j de la capa n, las salidas y_i de la capa $n+1$ se calculan como:

$$u_i = \sum_j \left(w_{i,j}^{n+1} * x_j \right) + w_{i,bias}^{n+1} \tag{3.20}$$

$$y_i = f\left(u_i\right) \tag{3.21}$$

Se pueden emplear distintas funciones de activación. ML implementa tres funciones estándar:

- Función identidad (`IDENTITY`): $f\left(x\right) = x$
- Sigmoide simétrica (`SIGMOID_SYM`): $f\left(x\right) = \frac{\beta * \left(1 - e^{-\alpha x}\right)}{\left(1 + e^{-\alpha x}\right)}$, que es la opción por defecto para MLP. La sigmoide estándar con $\beta = 1$, $\alpha = 1$ se muestra en la figura (3.41)

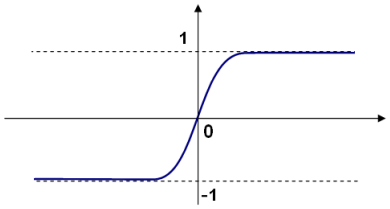


Figura 3.41.: Sigmoide estándar.

- Función Gaussiana (`GAUSSIAN`): $f\left(x\right) = \beta e^{-\alpha x * x}$, que de momento no está completamente soportada.

En ML, todas las neuronas tienen las mismas funciones de activación, con los mismos parámetros libres (α, β) que son especificados por el usuario y que no son alterados por el algoritmo de entrenamiento.

Así, toda la red entrenada funciona como sigue:

1. Toma el vector de características como entrada. El tamaño del vector es igual al tamaño de la capa de entrada.

2. Pasa los valores como entrada a la primera capa escondida.

3. Calcula las salidas de la capa oculta utilizando los pesos y las funciones de activación.

4. Pasa las salidas capa a capa hasta calcular la capa de salida.

Por lo tanto, para calcular la red, es necesario conocer todos los pesos $w_{i,j}^{n+1}$. Los pesos son calculados por el algoritmo de entrenamiento. El algoritmo toma un conjunto de entrenamiento, múltiples vectores de entrada con sus correspondientes vectores de salida, y de forma iterativa ajusta los pesos para permitir a la red que de la respuesta deseada a los vectores de entrada suministrados.

Cuanto mayor es el tamaño de la red (el número de capas escondidas y sus tamaños), mayor es el potencial de flexibilidad de la red. El error en el conjunto de entrenamiento puede hacerse arbitrariamente pequeño. Pero a su vez la red enseñada también "aprende" el ruido presente en el conjunto de entrenamiento, por lo que el error en el conjunto de entrenamiento normalmente comienza a aumentar una vez el tamaño de la red alcanza un límite. Además, las redes más grandes tardan mucho más en entrenarse que las pequeñas, por lo que es razonable preprocesar los datos, utilizando `PCA::operator()` o alguna técnica similar, y entrenar una red más pequeña sólo en las características esenciales.

Otra característica de MPL es su incapacidad para manejar datos categóricos tal cual. De todas formas, existe una solución. Si cierta característica en la capa de entrada o salida (en el caso de un clasificador de n clases para $n > 2$) es categórica y puede tomar $M > 2$ posibles valores, toma sentido representarla como una tupla binaria de M elementos, donde el elemento i-ésimo es 1 si y solo si la característica es igual al valor i-ésimo de los M posibles. Esto aumenta el tamaño de la capa de entrada/salida pero acelera la convergencia del algoritmo de entrenamiento a la vez que permite valores "difusos" de dichas variables, es decir, una tupla de probabilidades en lugar de un valor fijo.

ML implementa dos algoritmos para entrenar MLP. El primer algoritmo es un algoritmo clásico de propagación hacia atrás secuencial aleatorio. El segundo (por defecto) es un algoritmo de lotes RPROP [72].

Para implementar estos algoritmos en OpenCV, se emplean las siguientes clases:

`CvANN_MLP_TrainParams` Parámetros del algoritmo de entrenamiento MLP. Se puede inicializar la estructura con un constructor o se pueden ajustar los parámetros individuales una vez creada la estructura.

`CvANN_MLP` Modelo MLP. Al contrario que en muchos otros modelos ML que se construyen y entrenan a la vez, en el modelo MLP estos pasos están separados. Primero, se crea una red con la topología especificada utilizando el constructor no por defecto o el método `create`. Todos los pesos se fijan a cero. Entonces, se entrena la red utilizando un conjunto de vectores de entrada y de salida. El proceso de entrenamiento se puede repetir más de una vez, es decir, los pesos se pueden ir ajustando basándose en nuevos datos de entrenamiento.

3.4.9. utils

Se trata de un módulo exclusivo de Android. Sólo cuenta con la siguiente clase:

`Converters` Proporciona los métodos de conversión de todas las variables de la clase `Mat` de OpenCV a las clases `List` y `ArrayList` de Java y viceversa.

Figura 3.42.: Representación UML de la clase `Converters`.

Capítulo 4

Introducción a la programación en Android. Inclusión de OpenCV

4.1. Introducción

En este capítulo se realizará un estudio de la plataforma Android visualizando sus fundamentos y los paquetes y clases que serán de interés en la realización del proyecto fin de carrera. Seguidamente se explicará como incluir la biblioteca OpenCV en un proyecto Android y, finalmente se mostrarán algunos ejemplos de aplicaciones simples de Android empleando algunas aplicaciones simples para demostrar el uso de OpenCV.

4.2. Qué es Android

Android es una pila de software de Google, en concreto de la *Open Handset Alliance* [73] (formada aproximadamente por 30 organizaciones cuya intención es mejorar la calidad en el mercado y una oferta de telefonía competitiva abierta, integral y gratuita). Android posee un sistema operativo basado en Linux para dispositivos móviles (teléfonos *smartphones* y últimamente *tablets*). Ademas, incluye una completa interfaz de usuario, aplicaciones, bibliotecas de código, estructuras para aplicaciones, compatibilidad multimedia e incluso las funcionalidades del dispositivo. Aunque internamente los componentes del sistema operativo están escritos en C o C++, el desarrollo de aplicaciones se realiza en el lenguaje Java.

Android permite desarrollar cualquier aplicación empleando la totalidad de funcionalidades de las que dispone el dispositivo e incorporar dicha aplicación de forma sencilla, como si se tratase de una aplicación ya disponible previamente. Quizás por todo esto, la popularidad de Android se ha disparado, convirtiéndose en el sistema operativo para móviles más empleado en el mundo con una cuota de mercado del 59 %,

superando al IOS (23 %), Blackberry (sistema operativo de RIM), 6.4 %, o Symbian (sistema operativo de Nokia), 6.8 % [74]. Esto deja claro que se trata de un sistema con el mayor potencial de desarrollo en el mundo de la telefonía móvil y un crecimiento exponencial a tener en cuenta.

4.2.1. Google Play

Acorde a este crecimiento de unidades vendidas, ha surgido un nuevo negocio: *el desarrollo de aplicaciones*. Las facilidades que da el sistema para diseñar las aplicaciones y las posibilidades que ofrece, han motivado a diferentes empresas desarrolladoras a aprovechar el tirón de la venta de dispositivos para crear un mercado muy productivo. Para ofrecer un servicio de ventas sencillo y único en el que cada desarrollador pueda ofertar su aplicación, Google creó, al igual que Apple con su *AppStore*, *Google Play* (anteriormente conocido como "Android Market").

Las empresas han visto en el desarrollo de aplicaciones sencillas una manera de sacar grandes beneficios, pues su bajo costo de desarrollo permite insertarlas a un precio reducido en el mercado. Justamente por esto, estas aplicaciones encabezan las listas de descargas. Con un bajo precio, muchos usuarios descargan la aplicación, por lo que el beneficio de la empresa no viene de vender pocas unidades a un gran precio, sino de realizar miles de facturaciones a un precio bajo. Con el numero de dispositivos que existen, y con las previsiones de crecimiento, parece ser la opción mas adecuada para sacar beneficios.

4.3. Fundamentos de la programación en Android

Las aplicaciones Android están escritas en el lenguaje de programación Java, siendo el SDK de Android el que compila todo el código, con todos los datos y recursos, en un paquete Android, generando un archivo con extensión *.apk*. Cualquier código dentro de un archivo *.apk* se considera una aplicación y es el archivo que se usa para instalar la aplicación en el terminal móvil.

Una vez instalada en el dispositivo, cada aplicación se ejecuta bajo su propia área de seguridad, es decir, cada aplicación se considerara un usuario diferente, asignando los permisos necesarios de manera independiente (ejecución aislada).

Esto favorece el principio del mínimo privilegio, es decir, cada aplicación, por defecto, tiene acceso solo a los componentes que necesite para ejecutarse, creando un

entorno muy seguro en el cual una aplicación no puede acceder a partes del sistema a las que no se le haya dado permiso [75].

4.3.1. Componentes

Las aplicaciones en Android pueden estar formadas por cuatro tipos de bloques o componentes principales. Toda aplicación es una combinación de uno o más de estos componentes, declarados de forma explícita en un fichero con formato XML denominado *AndroidManifest.xml* [76], junto a otros datos asociados como valores globales, clases que implementa, datos que puede manejar, permisos, entre otros. Cada componente se utiliza para un propósito concreto, con ciclos de vida distintos. Los cuatro tipos de componentes en los que puede dividirse una aplicación para Android son: "Activity", "Service", "BroadcastReceiver" y "ContentProvider".

4.3.1.1. *Activity*

El elemento *Activity* [77] representa una ventana con una interfaz de usuario, una pantalla visible. Una aplicación de Android puede tener mas de un elemento *Activity*, de manera que una aplicación externa puede acceder a dicha aplicación para diferentes acciones. Un ejemplo podría ser una aplicación de correo que tenga una actividad para mostrar la bandeja de entrada, otra para redactar nuevos e-mails y una tercera para la bandeja de salida. Si esta aplicación lo permite, una aplicación externa como puede ser el *browser* puede acceder a la actividad de redactar correo para, por ejemplo, enviar una imagen por correo.

El elemento *Activity* utiliza una o varias vistas (*View*) para presentar al usuario los elementos de la interfaz. Las actividades en Android corresponden con clases que derivan de la clase del mismo nombre *Activity*.

4.3.1.2. *Service*

El elemento *Service* [78] es un componente que se ejecuta en segundo plano (*background*), por tanto no dispone de interfaz de usuario. El usuario no interactúa con los servicios, sino que son manejados por las aplicaciones. Se utiliza, por ejemplo, para llevar a cabo operaciones de larga duración, como puede ser una sincronizan de datos que se ejecute de fondo de manera continuada, o la reproducción de una lista de música con el reproductor, o la obtención de los parámetros de un GPS para enviar a un servidor de localización.

Android distingue dos tipos de servicios: *locales* y *remotos*. Los servicios locales son aquellos que forman parte de la propia aplicación, es decir que son servicios privados únicamente accesibles desde la aplicación. Por otro lado, los servicios remotos son aquellos servicios que ofrece una aplicación instalada en el terminal a cualquier otra aplicación que lo requiera, de manera publica. Asociado a este concepto, se definen dos posibles roles: el de controlador del servicio (encargado de arrancar y parar el servicio) y el cliente del servicio (aquel que se conecta al servicio para realizar peticiones al mismo).

Los servicios en Android corresponden con clases que derivan de la clase *Service*. Por lo tanto, la clase creada debe heredar los métodos de "Service" para sobrecargarlos posteriormente. Los métodos necesarios para que un servicio funcione correctamente son *onStartCommand*, *onBind*, *onCreate* y *onDestroy*. Así:

- onStartCommand: el sistema llama a este método cuando otro componente, como una actividad, pide que el servicio se inicie, llamando al método *start-Service()*. Una vez que se llama a este método, el servicio se inicia y se ejecuta en segundo plano de forma indefinida.

- onBind: el sistema llama a este método cuando otro componente quiere enlazar con el servicio, llamando al método *bindService()*. En la implementación de este método, se debe proporcionar una interfaz que los clientes utilizan para comunicarse con el servicio, mediante la devolución de un *IBinder*. Siempre se debe implementar este método, por lo que si no se utiliza, se debe devolver *null*.

- onCreate: el sistema llama a este método para crear el servicio, realizando una sola vez los procedimientos de instalación. Si el servicio se esta ejecutando, este método no se invoca.

- onDestroy: el sistema llama a este método cuando el servicio ya no es necesario y se puede eliminar del sistema. Este servicio se debe implementar para garantizar los recursos. Esta es la ultima llamada que recibe el servicio.

El sistema de servicios de Android, permite que un servicio en funcionamiento notifique al usuario algunos eventos mediante mensajes emergentes en la interfaz de usuario o la barra de notificaciones, que proporciona un icono con el fin de realizar una acción por parte del usuario a través de esta. Por lo general, una notificación mostrada en la barra de estado es la mejor técnica cuando un trabajo de fondo ha terminado (por ejemplo un aviso de descarga de un archivo completada) y el usuario puede entonces actuar en consecuencia. Cuando el usuario selecciona la notificación de la vista ampliada, la notificación puede iniciar una actividad (por ejemplo, para ver el archivo descargado).

En este proyecto fin de carrera se usa un aviso en la barra de notificaciones para indicar que el servicio esta activo (de forma que el usuario sepa que la aplicación esta usando el GPS y la red WIFI para comunicarse).

4.3.1.3. *BroadcastReceiver*

El elemento *BroadcastReceiver* es aquel que responde a la emisión de un mensaje en el sistema, como por ejemplo una llamada de teléfono o un mensaje de texto entrante. Por tanto, se utiliza para lanzar alguna ejecución dentro de la aplicación actual cuando se produzca un evento semejante a los mencionados.

Para que un objeto de la clase *BroadcastReceiver* funcione, no es necesario que la aplicación en cuestión sea la aplicación activa en el momento de producirse el evento. El sistema lanzara la aplicación si es necesario cuando el evento monitorizado tenga lugar. Este componente se implementa a través de una clase de nombre *BroadcastReceiver* [79].

4.3.1.4. *ContentProvider*

El componente *ContentProvider* es capaz de gestionar, almacenar y mostrar datos a otras aplicaciones en el entorno Android. Una clase que implemente este componente contendrá una serie de métodos que permite almacenar, recuperar, actualizar y compartir los datos de una aplicación. Su uso es similar al de SQL en otras plataformas. El sistema Android proporciona un proveedor de contenido que gestiona la información de los contactos del usuario, información disponible para cualquier otra actividad.

Un proveedor de contenido se implementa como una subclase de *ContentProvider* [80] y debe aplicar un conjunto de estándares de APIs que permita a otras aplicaciones realizar transacciones.

4.3.2. ¿Que es un intent?

Un *intent* es la descripción abstracta de una operación que se va a llevar a cabo. Se crea a partir de un objeto de la clase *Intent* [81], el cual define un mensaje a activar o un componente específico. Un *intent* puede ser:

- *Explícito*: se crea el intent indicando el nombre de la clase correspondiente a la actividad y el paquete, para posteriormente llamar al método *startActivity*.

- *Implícito*: no se indica el nombre específico de la clase de la actividad, sino que una vez creado el *intent*, se establecen una serie de criterios, dejando la elección de la actividad al sistema.

Un objeto de tipo *Intent* esta formado por fragmentos de información que describen la acción o servicio deseados, por lo que sus atributos suelen ser verbos como *VIEW*, *PICK* o *EDIT*. Esto es muy útil, pues al lanzar el *intent*, una aplicación puede delegar el trabajo en otra, de forma que el sistema se encarga de buscar que aplicación entre las instaladas es la que puede llevar a cabo la acción solicitada. Por ejemplo, abrir una URL en algún navegador Web, o escribir un correo electrónico desde algún cliente de correo. Por lo tanto, es posible utilizar un objeto *intent* para iniciar o dar nuevas tareas a componentes en el programa. Así:

- Para iniciar una actividad se pasa un *intent* a los métodos *startActivity()* o *startActivityForResult* (en el caso de que se requiera la devolución de un resultado).

- Para iniciar un servicio se usa el método *startService* o se puede enlazar con *bindService* (pasandole un *intent*).

- Para iniciar una notificación se usan los métodos *sendBroadcast()*, *sendOrderedBroadcast()*, o *sendStickyBroadcast()*.

4.3.3. Archivo AndroidManifest

Este archivo representa la información de implementación de una aplicación Android. Describe todos los componentes de la aplicación (*Activity*, *Service*, *BroadcastReceiver* y *ContentProvider*), de manera que el sistema sabe que componentes hay y bajo que condiciones se ejecutaran. Por otra parte, determina los permisos, tanto de la propia aplicación para acceder a las partes protegidas del API, como para el acceso de otras aplicaciones a las funcionalidades de la aplicación desarrollada. Ademas, a través de este archivo se deben indicar las bibliotecas con las que debe enlazarse la aplicación [76].

El archivo *AndroidManifest* es un archivo XML, en el que se pueden encontrar las siguientes clausulas:

- *manifest*: es el nodo raíz bajo el que se declaran todos los contenidos del manifiesto. Incluye distintos atributos, como son el numero de versión del programa y el nombre del paquete que servirá para referenciar a la aplicación en Google Play.

- *uses-permission*: etiqueta con la que se declaran los requisitos de seguridad para que la aplicación pueda ser desplegada correctamente. Es decir, los permisos otorgados a una aplicación para realizar las tareas requeridas. Con esta etiqueta se consigue que el sistema Android sea mas robusto y seguro, pues una aplicación solo puede utilizar aquellas funcionalidades críticas para las que tiene permiso. Son necesarias, por ejemplo para permitir utilizar el sensor GPS o conectarse a Internet, entre otros. La siguiente sentencia permite a la aplicación hacer uso del sensor GPS:

```
<uses-permission
    Android:name=\Android.permission.ACCESS FINE LOCATION"
/>
```

- *permission*: define la restricción a los servicios del sistema Android para garantizar la integridad del sistema.

- *application*: se trata del elemento raíz que enumera los componentes básicos de la aplicación (*Activity*, *Service*, *BroadcastReceiver* y *ContentProvider*).

- *activity*: representa un componente *Activity* encargado de representar una acción concreta (normalmente asociado a una interfaz de usuario). Contiene los campos "action", "category", "data" y "meta-data" que definen los datos y la acción de la actividad. En el manifiesto existen tantos elementos activity como componentes *Activity* haya en la aplicación.

- *receiver*: este elemento representa un componente *BroadcastReceiver*. Permite a la aplicación recibir *intents* que emiten el sistema u otras aplicaciones.

- *service*: se corresponde con un componente *Service* encargado de ejecutar una acción en *background*. Todos los servicios deben ser representados en el archivo *manifest*, de forma que el que no este declarado no sera visible para el sistema y nunca sera ejecutado.

- *provider*: representa un componente *ContentProvider*, utilizado en la aplicación para almacenar y compartir datos.

- *intent-filter*: declara un *Intent Filter*, es decir, especifica los tipos de intents que una actividad, un servicio o un receptor puede responder. Una vez lanzado un *Intent*, el sistema Android utiliza los *Intent Filter* declarados en el *AndroidManifest.xml* para filtrar y decidir, comparando en cada uno de ellos el valor del elemento *intent-filter* y averigua que aplicación es la mas optima para hacerse cargo de la acción requerida.

En el código 4.1 se muestra un ejemplo de archivo *AndroidManifest.xml* para una determinada aplicación. En este archivo se puede observar que la aplicación esta formada solo por una actividad llamada *HolaMundo*.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:Android="http://schemas.Android.com/apk/res/Android"
    package="Android.pruebas.holaMundo"
    Android:versionCode="1"
    Android:versionName="1.0">
    <application
        Android:icon="@drawable/icon"
        Android:label="@string/app_name">
        <activity
            Android:name=".HolaMundo"
            Android:label="@string/app_name">
            <intent-filter>
                <action Android:name="Android.intent.action.MAIN" />
                <category Android:name="Android.intent.category.
                    LAUNCHER"
                    />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Código 4.1: Ejemplo de *AndroidManifest.xml*

4.3.4. Interfaz de Usuario

La interfaz de usuario en Android [82] se define con un archivo XML que debe encontrarse en la ruta */res/layout* del proyecto, mientras que el comportamiento se especifica en el código fuente de la aplicación (en el directorio */src* del proyecto). El diseño de la plantilla o *layout* de la aplicación determina la posición y orientación que tomaran los diferentes elementos que se añadirán a las ventanas.

Existe una amplia variedad de *layouts*, dependiendo de como sean posicionados los elementos en pantalla, de los cuales, los mas importante son: *LinearLayout*, *FrameLayout*, *TableLayout* y *RelativeLayout* [83]. Asimismo, se permite al usuario la creación de *layouts* propios, mediante la clase *CustomLayout*.

Común a estos layouts, están los elementos que van a formar parte de ellos, es decir, que constituyen las ventanas de la aplicación. Este tipo de componentes son clases derivadas de la clase *View*, que define las características fundamentales para que

un componente pueda presentarse en pantalla. Los principales elementos que se pueden añadir son:

- *Gallery*: se utiliza para desplegar listados de imágenes en un componente con *scroll*.
- *GridView*: despliega una tabla con *scroll* de m columnas y n filas.
- *ListView*: despliega una lista con *scroll* de una columna.
- *Button*: coloca un botón en la pantalla para realizar una acción (evento) sobre el, que sera evaluada en el momento.
- *EditText*: campo de texto evitable.
- *TextView*: etiqueta en la que se representa un texto.
- *Spinner*: muestra una lista, de manera que al pulsar sobre ella, se muestra un desplegable con todas las opciones disponibles, quedando fijada la seleccionada posteriormente.
- *MapView*: representa un mapa de Google.

Todos estos elementos tienen unos parámetros en común que deben ser especificados en el archivo XML para crearlos de manera correcta y poder referenciarlos en el código Java. Para establecer las características de estos elementos se emplea la nomenclatura habitual de XML, "Android:parámetro". Así, cada elemento declarado, que sea referenciado desde el programa Java, debe contener como mínimo tres parámetros:

- *id*: identificador de control, que servirá para identificar en el código al elemento. Para que al compilar se genere la constante correspondiente en la clase *R.java*, el identificador debe estar precedido por "@+id/".
- *layout_heigth* y *layout_width*: donde se especifican las dimensiones con respecto al layout que lo contiene. Normalmente tomara dos valores: "wrap_content", si las dimensiones se ajustan al contenido, o "fill_parent", si el ancho o alto se ajusta al tamaño del *layout* contenedor.

Del mismo modo, dependiendo del elemento creado, existe la posibilidad de incluir otros parámetros que sirvan para definir la interfaz, como puede ser el *background*. En el código 4.2 se utiliza el lenguaje XML para representar objetos *TextView*, *EditText* y *Button*, indicando los parámetros comunes y otros, como la alineación con respecto a la plantilla.

```
<TextView
    Android:id="@+id/label"
    Android:layout_width="fill_parent"
    Android:layout_height="wrap_content"
    Android:text="@string/hello" />
<EditText
    Android:id="@+id/entry"
    Android:layout_width="fill_parent"
    Android:layout_height="wrap_content"
    Android:background="@Android:drawable/editbox_background"
    Android:layout_below="@id/label" />
<Button
    Android:id="@+id/uno"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"
    Android:text="@string/uno"
    Android:layout_below="@id/entry"
    Android:layout_alignParentRight="true"/>
```

Código 4.2: Representación de objetos *TextView*, *EditText* y *Button*.

Posteriormente, para mostrar la interfaz diseñada en el terminal móvil, se debe llamar al método *setContentView(R.layout.main)*, dentro de la actividad diseñada. Se utiliza *R.layout.main* si los elementos gráficos están declarados dentro del archivo *main.xml* del paquete *res/layout*. En el código 4.3 se muestra el archivo *main.xml* correspondiente a la aplicación *Hola Mundo*. En este código se puede observar que la ventana de la aplicación solo tiene una etiqueta cuyo texto es "Hola mundo Android".

```
1  <?xml versión="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:Android="http://schemas.Android.com/apk/res/
       Android"
3      Android:orientation="vertical"
4      Android:layout_width="fill_parent"
5      Android:layout_height="fill_parent">
6      <TextView
7          Android:layout_width="fill_parent"
8          Android:layout_height="wrap_content"
9          Android:text="Hola mundo Android"
10     />
11 </LinearLayout>
```

Código 4.3: Archivo main.xml de la aplicación Hola Mundo.

4.3.5. Paquetes importantes en el desarrollo de aplicaciones

Android esta formado por una serie de bibliotecas de APIs en las que se encuentran las clases principales, con las que se consigue una mayor eficiencia en la utilización de los recursos del sistema. Estas bibliotecas vienen incorporadas al SDK de Android, y permiten que sea posible controlar los sensores (acelerómetro, GPS, giroscopio, entre otros), el contenido en pantalla, el hardware (como la batería y la cámara) y la conexión, entre otros. A continuación se indican algunos de los paquetes mas importantes de Android.

4.3.5.1. El paquete *Android.app*

Este paquete contiene las clases de alto nivel que sirven como modelo de aplicación. Concretamente, en este paquete están definidos dos componentes: *Activity* y *Service*. Este paquete también define algunas utilidades para el desarrollo de aplicaciones, como pueden ser las notificaciones y la barra de acción [84].

4.3.5.2. El paquete *Android.widget*

El paquete *widget* contiene la mayoría de los elementos visuales de la interfaz gráfica disponibles para la aplicación. Ademas de definir la interfaz en el archivo XML, hay que crear el objeto correspondiente para recibir los valores del *layout* XML [85].

4.3.5.3. El paquete *Android.view*

Este paquete proporciona clases que sirven para manejar la distribución de la pantalla y la interacción con el usuario [86]. Las clases más importantes de este paquete son las que se encargan del manejo de la pantalla:

- *android.view.SurfaceHolder*: Interfaz abstracta a una superficie de pantalla. Le permite controlar el tamaño de la superficie y formato, editar los píxeles en la superficie, y vigilar los cambios en la superficie. Esta interfaz suele estar disponible a través de la clase *SurfaceView*.

- *android.view.SurfaceView*: Proporciona una superficie de dibujo dedicada incrustada dentro de una jerarquía de vistas. Permite controlar el formato de esta superficie y, si se quiere, su tamaño. SurfaceView se encarga de la colocación de la superficie en el lugar correcto en la pantalla

4.3.5.4. El paquete *Android.content*

El paquete *content* contiene clases para el acceso y publicación de datos en el dispositivo. Incluye tres categorías principales de APIs [87]:

- *Android.content*: para compartir contenidos entre diferentes aplicaciones. Contiene las clases necesarias para declarar un *BroadcastReceiver*, *ContentProvider*, *ContentResolver*, *Intent* e *IntentFilter*.
- *Android.content.pm*: para el acceso a información sobre un paquete Android de una aplicación, incluida información sobre sus actividades, permisos, servicios, firmas y proveedores.
- *Android.content.res*: para acceder a los datos asociados a los recursos en una aplicación. Contiene las clases necesarias para acceder a los recursos y a los detalles de configuración que afectan al comportamiento de la aplicación.

4.3.5.5. El paquete *Android.location*

El paquete *location* proporciona los medios necesarios para controlar el receptor GPS y obtener los datos referentes a la localización [88]. Contiene las siguientes clases:

- *Address*: representa una dirección en una versión simplificada de *xAL* (*eXtensible Address Language*).
- *Criteria*: indica los criterios a seguir para la utilización de un determinado proveedor de localización.
- *Geocoder*: contiene los controladores para la geocodificación y geocodificación inversa.
- *GpsSatellite*: representa el estado actual del satélite GPS.
- *GpsStatus*: representa el estado actual del GPS.
- *Location*: representa una localización geográfica obtenida por el sensor en un determinado tiempo.
- *LocationManager*: proporciona acceso a los servicios de localización del sistema.
- *LocationProvider*: proporciona informes periódicos sobre la localización geográfica de un dispositivo.

4.3.5.6. El paquete *Android.os*

Paquete principal del sistema Android, con el se accede a los servicios del mismo, el paso de mensajes y la comunicación entre procesos en el dispositivo. Contiene clases para controlar la carga de la batería, manejar los procesos del sistema y el vibrador, entre otros [89].

4.3.5.7. El paquete *Android.graphichs*

Proporciona herramientas gráficas de bajo nivel tales como *canvas*, filtros de color, puntos y rectángulos que permiten manejar directamente dibujos en la pantalla [90]. Se trata de un paquete importante en la realización del proyecto fin de carrera ya que contiene a la clase *Bitmap* que es la clase que emplea OpenCV para convertir las imágenes internas (*Mat*) en imágenes que Android pueda mostrar por pantalla.

4.3.5.8. El paquete *Android.util*

Proporciona métodos comunes de utilidad tales como la manipulación de fecha/hora, los codificadores y decodificadores de base64, métodos de conversión numéricos y de cadenas de texto, y utilidades XML [91]. Su utilidad en la realización del proyecto radica en que OpenCV emplea ficheros XML para almacenar los descriptores previamente calculados.

4.3.5.9. El paquete *Android.hardware*

Proporciona soporte para funciones de hardware, como la cámara y otros sensores. Hay que tener presente que no todos los dispositivos con Android soportan todas las características de *hardware*, por lo que se debe declarar el hardware que requiere la aplicación utilizando el elemento manifest `<uses-feature>`.

4.4. Integración de las librerías OpenCV

En esta sección se verá como preparar el entorno de desarrollo para Android y como integrar la biblioteca OpenCV.

4.4.1. Configuración del entorno de desarrollo para Android

Es necesario instalar las siguientes herramientas:

1. Sun JDK 6.

   Sólo es necesario visitar la página de descargas de Java SE[1] y descargar la versión apropiada para el sistema operativo que se esté empleando en el equipo de desarrollo. En este caso, *Windows*.

2. SDK de Android.

   Obtener la última versión del SDK en http://developer.android.com/sdk/index.html y seguir las instrucciones de la guía de instalación de Google [92].

3. Componentes del SDK de Android.

   Se necesita tener instalados los siguientes componentes:

   - *Android SDK Tools, revision14* o superior.
   - *SDK Platform Android 3.0, API 11* (también conocida como *android-11*).

     La plataforma mínima de la API Java de OpenCV es Android 2.2 (API 8). Para poder compilar con éxito algunos ejemplos la plataforma objetivo debe estar establecida a Android 3.0 (API 11) o superior (figura 4.1). Esto no impedirá que se ejecuten en Android 2.2+.

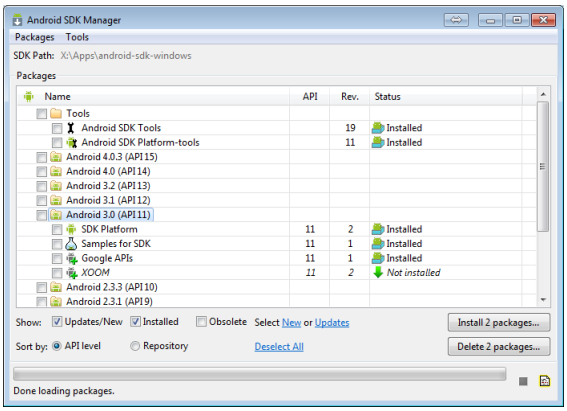


Figura 4.1.: *Manager* del SDK de Android.

4. IDE Eclipse.

   Para OpenCV 2.4.x se recomienda usar Eclipse 3.7 (Índigo) o versiones posteriores. Si Eclipse no está instalado, se puede encontrar en su página de descarga[2].

[1] http://www.oracle.com/technetwork/java/javase/downloads/index.html
[2] http://www.eclipse.org/downloads/

5. *Plugin* ADT para Eclipse.

   Para instalar el *plugin* hay que seguir las instrucciones se que dan en la página de desarrollo de Android (http://developer.android.com/sdk/installing/installing-adt.html).

4.4.2. Obtener el paquete OpenCV para desarrollo en Android

1. Ir a la página de descarga de OpenCV en *SourceForge*[3] y descargar la versión más reciente.

2. Crear una nueva carpeta para el desarrollo Android+OpenCV . Por ejemplo, el directorio `C:\Work\android-opencv\`. Para evitar posibles problemas con el NDK se recomienda no usar espacios en la ruta.

3. Descomprimir el paquete OpenCV en ese directorio.

4.4.3. Abrir la biblioteca OpenCV y sus ejemplos en Eclipse

1. Iniciar Eclipse y seleccionar la localización del *workspace* (figura 4.2).

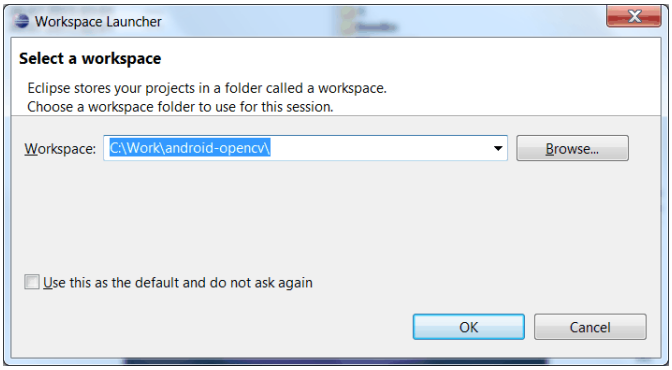


Figura 4.2.: Selección del workspace en Eclipse.

2. Configurar el plugin ADT (si es necesario).

   En la mayoría de los casos, el plugin ADT localiza automáticamente el SDK Android , pero en alguna ocasión falla y muestra la pantalla de la figura 4.3. En ese caso, se selecciona la opción *Use existing SDKs*, se busca el directorio del SDK Android y se hace click en *Finish*.

[3] http://sourceforge.net/projects/opencvlibrary/files/opencv-android/

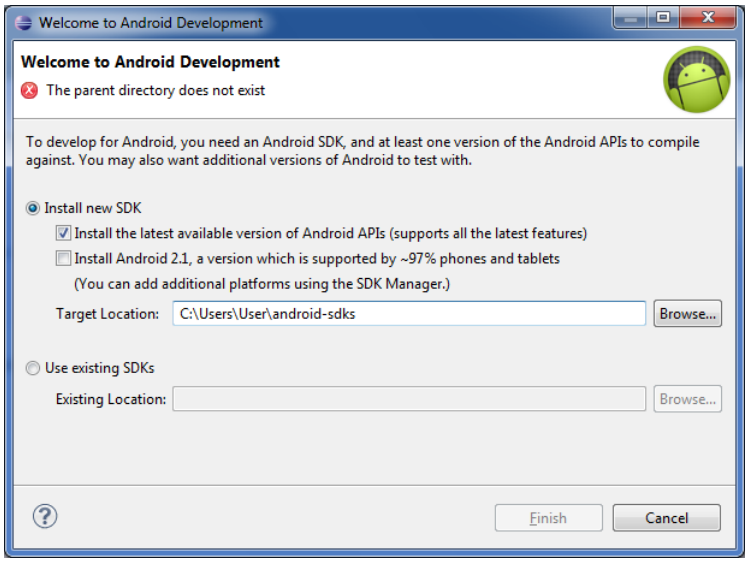


Figura 4.3.: Error en la detección del SDK Android.

3. Importar OpenCV y sus ejemplos al *workspace*.

   La biblioteca OpenCV se distribuye como un *Android Library Project* listo para usar. Simplemente hay que referenciarla en los proyectos.

   Cada ejemplo incluido en el paquete OpenCV es un proyecto Android al uso que ya referencia a la las librerías OpenCV. Para importar OpenCV y sus ejemplos al *workspace* se siguen los siguientes pasos:

   - Se hace click derecho en la ventana *Package Explorer* y se selecciona la opción *Import...* del menú contexto (figura 4.4).

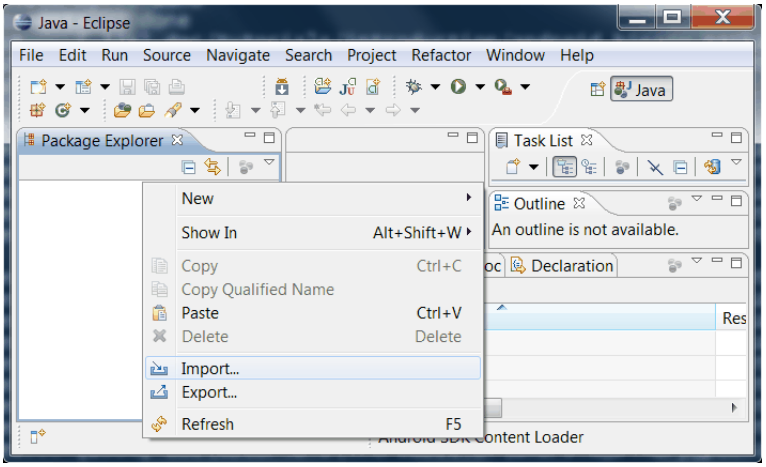


Figura 4.4.: Importar OpenCV al *workspace*. Paso 1.

   - En el panel principal se selecciona *General* ▷ *Existing Projects into Workspace* y se presiona el botón *Next* (figura 4.5).

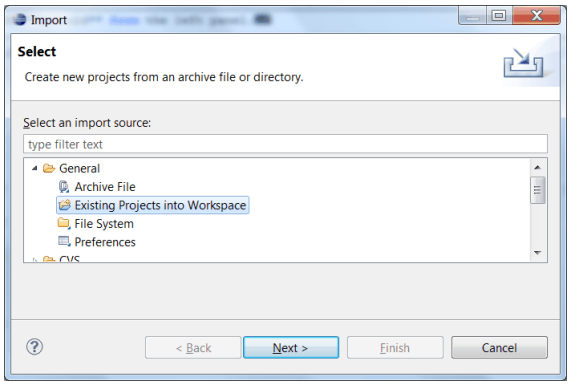


Figura 4.5.: Importar OpenCV al *workspace*. Paso 2.

- En *Select root directory* del panel principal se localiza el directorio del paquete OpenCV (figura 4.6). Si se creó el workspace en el directorio del paquete, sólo será necesario presionar el botón *Browse...* e inmediatamente cerrar el diálogo de selección de directorio con el botón *OK*. Eclipse localizará automáticamente la biblioteca OpenCV y sus ejemplos.

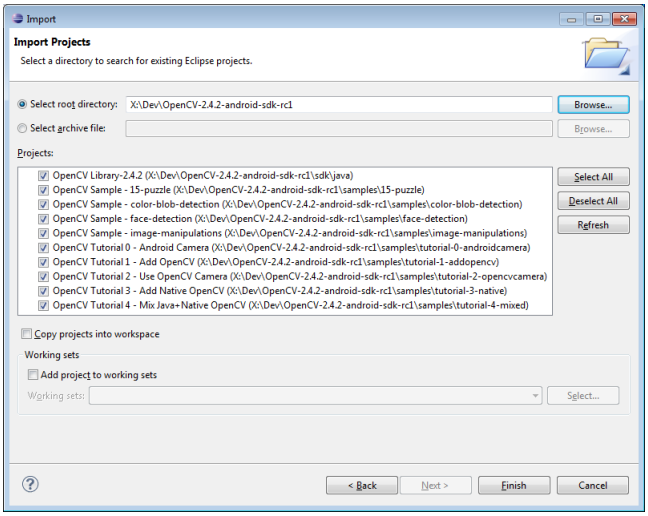


Figura 4.6.: Importar OpenCV al *workspace*. Paso 3.

- Se selecciona el botón *Finish* para completar la operación de importación.

  Una vez pinchado el botón *Finish*,Eclipse cargará todos los proyectos en el workspace. E indicará varios errores (figura 4.7). Sin embargo, todos esos errores son sólo falsas alarmas.

  Para hacer que Eclipse entienda que no hay ningún error es selecciona la biblioteca OpenCV en *Package Explorer* (click derecho del ratón) y se presiona la tecla *F5*. Entonces se selecciona algún ejemplo (excepto el primer ejemplo en *Tutorial Base* y en *Tutorial Advanced*) y se presiona *F5* también.

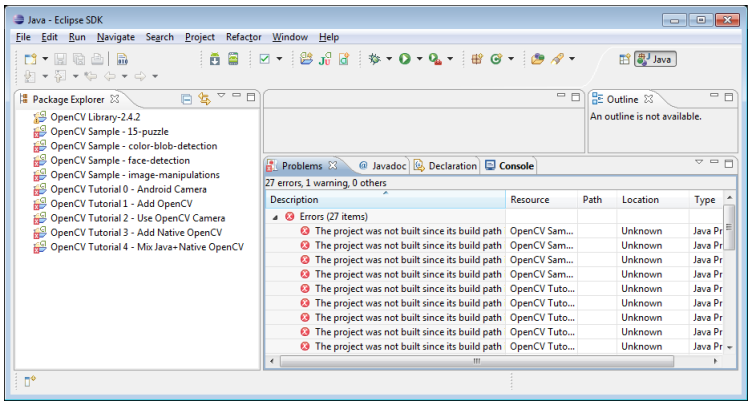


Figura 4.7.: Errores al importar OpenCV.

En algunos casos, esos errores desaparecen una vez se selecciona *Project* ▷*Clean...* ▷*Clean all* ▷*OK*.

Algunas veces se necesitarán algunas manipulaciones más avanzadas:

- Los proyectos proporcionados están configurados para *android-11* y esta plataforma puede no estar disponible en la instalación del SDK de Android. Después de hacer click derecho en cualquier proyecto, se selecciona *Properties* y luego *Android* en el panel izquierdo. Se selecciona una API Nivel 11 o superior (figura 4.8).

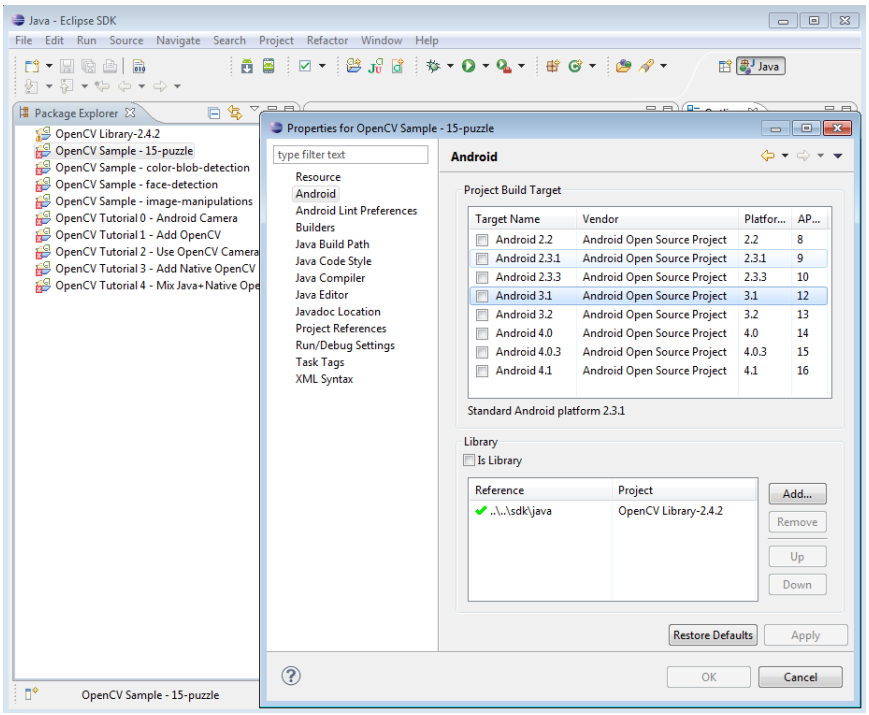


Figura 4.8.: Selección de nivel del API.

Después de esta operación, Eclipse reconstruirá el *workspace* y los iconos de error irán desapareciendo uno tras otro. Una vez Eclipse complete la reconstrucción se tendrá un *workspace* limpio y sin errores (figura 4.9).

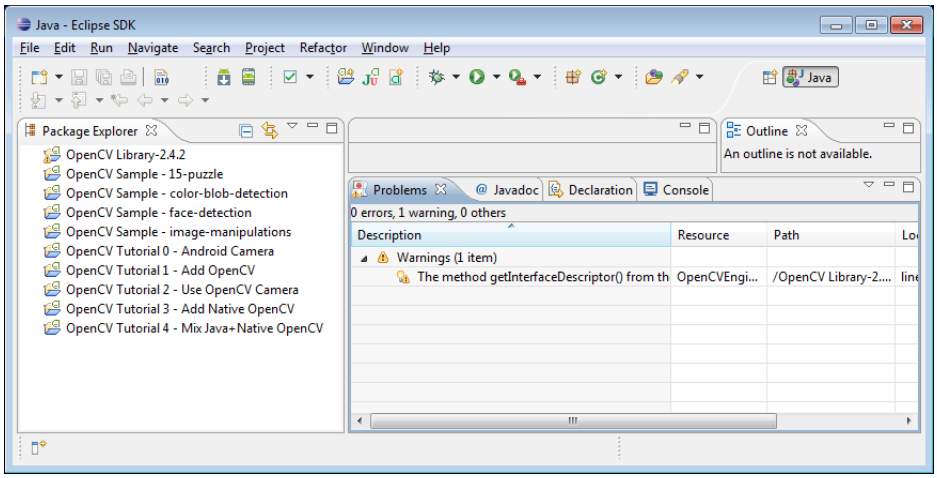


Figura 4.9.: *Workspace* sin errores.

4.4.4. Como usar la biblioteca OpenCV en una aplicación

Para usar OpenCV en un proyecto se deben realizar los siguientes pasos:

1. Añadir la biblioteca OpenCV al *workspace*. Ir a *File* ▷*Import* ▷*Existing project in your workspace*, presionar le botón *Browse* y seleccionar la ruta del SDK OpenCV (figura 4.10).

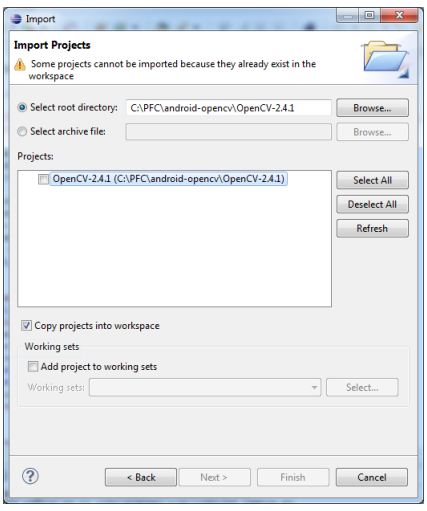


Figura 4.10.: Añadir la biblioteca OpenCV.

2. En el proyecto de la aplicación añadir una referencia al SDK Java de OpenCV. En Project ▷Properties ▷Android ▷Library ▷Add seleccionar la librería OpenCV (figura 4.11).

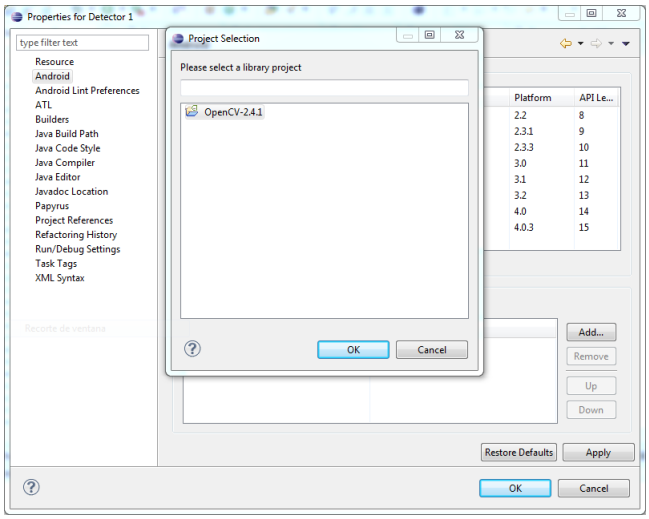


Figura 4.11.: Selección de la librería OpenCV en un Proyecto.

3. Copiar las librerías nativas al directorio del proyecto en la carpeta `libs/target_arch/` (figura 4.12).

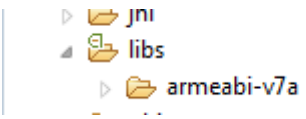


Figura 4.12.: Directorio de las librerías nativas de OpenCV.

Una vez añadidas las dependencias con la biblioteca OpenCV, Eclipse copiará automáticamente todas las librerías necesarias al paquete de la aplicación. Para poder usar la funcionalidad de OpenCV se necesita añadir el código de inicialización de la biblioteca OpenCV antes de usar cualquier código específico de OpenCV, por ejemplo, en la sección *static* de la clase *Activity* (código 4.4).

```
static {
    if (!OpenCVLoader.initDebug()) {
        // Report initialization error
    }
}
```

Código 4.4: Inicialización de OpenCV.

Si la aplicación incluye otras librerías nativas dependientes de OpenCV es necesario inicializar OpenCV antes que estas (código 4.5).

```
static {
    if (OpenCVLoader.initDebug()) {
        System.loadLibrary("my_super_lib1");
```

```
        System.loadLibrary("my_super_lib2");
    } else {
        // Report initialization error
    }
}
```

Código 4.5: Inicialización de otras librerías nativas.

4.4.5. Usando código C++ OpenCV con el paquete binario Android

Normalmente, al crear aplicaciones Android todo el código está escrito en Java. Pero a veces no es suficiente y hay que ir al nivel nativo y escribir algunas partes de la aplicación en C/C++. Esto es especialmente importante cuando se tiene algo de código visión por computador que está escrito en C++ y utiliza OpenCV, y se desea reutilizar en una aplicación para Android, pero no se desea volver a escribir el código C++ a Java. En este caso, es necesario utilizar JNI, un *framework* Java para interactuar con código nativo. Esto implica que se debe agregar una clase Java con métodos nativos que expongan su funcionalidad C++ a la parte Java de la aplicación Android.

4.4.5.1. Configuración del NDK Android

Para compilar código C++ para la plataforma Android se necesita el Kit de Desarrollo Nativo de Android (NDK). La versión más reciente del NDK se puede obtener de su página de descarga[4]. Para instalar el NDK Android simplemente hay que extraer el fichero descargado en alguna carpeta del ordenador de desarrollo [93].

4.4.5.2. Estructura de una aplicación Android

Por lo general, el código de una aplicación Android tiene la estructura siguiente:

- `directorio raíz del proyecto/`
  - `jni/`
  - `libs/`
  - `res/`

[4]http://developer.android.com/tools/sdk/ndk/index.html

- `src/`
- `AndroidManifest.xml`
- `project.properties`
- `... otros ficheros ...`

donde

- la carpeta `src` contiene el código Java de la aplicación,
- la carpeta `res` contiene los recursos de la aplicación (imágenes, ficheros XML que describen el *layout* de la UI , etc),
- la carpeta `libs` contendrá las librerías nativas una vez compiladas,
- y la carpeta `jni` contiene el código fuente C/C++ de la aplicación y los *scripts* de compilación del NDK *(Android.mk* y *Application.mk*).

  Estos scripts controlan el proceso de compilación C++ (están escritos en lenguaje *Makefile*).

Igualmente, carpeta raíz debe contener los siguientes archivos:

- El archivo *AndroidManifest.xml* proporciona información esencial sobre la aplicación al sistema Android (nombre de la Aplicación, nombre del paquete principal de la aplicación, componentes de la aplicación, permisos requeridos, etc).

  Se puede crear usando el asistente de Eclipse o la herramienta del SDK Android.

- *project.properties* es un fichero de texto que contiene información acerca de la plataforma Android objetivo (número de API) y otros detalles de compilación.

  Este fichero es generado por Eclipse o puede ser creado usándola herramienta del SDK Android.

4.4.5.3. Compilando una aplicación con parte nativa C++ desde la línea de comando

Esta es la forma estándar para compilar la parte C++ de una aplicación Android:

1. Abrir la consola e ir a la carpeta raíz de la aplicación Android.

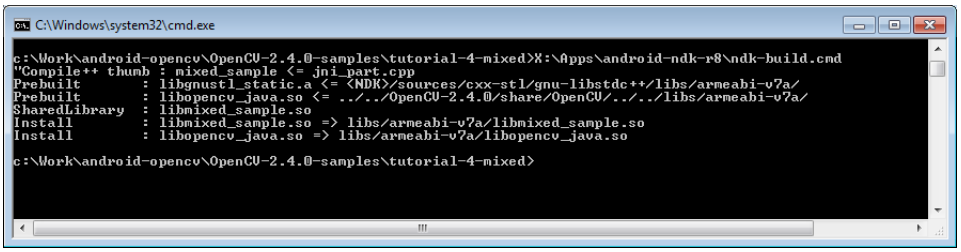


Figura 4.13.: Ejecución de `ndk-build`.

```
cd <directorio raíz del proyecto>/
```

2. Ejecutar el siguiente comando:

```
<ruta del NDK>/ndk-build
```

3. Una vez ejecutado este comando, se compila la parte C++ del código fuente.

   Después de esto, se puede (re)compilar la parte Java de la aplicación (usando tanto Eclipse como la herramienta de compilación *ant*).

Nota: Se pueden fijar algunos parámetros de ndk-build:

Compilación detallada.

```
<ruta del NDK>/ndk-build V=1
```

Recompilar todo.

```
<ruta del NDK>/ndk-build -B
```

4.4.5.4. Compilando una aplicación con parte nativa C++ desde Eclipse

Hay varias maneras posibles para integrar la compilación de código C++ por parte del NDK de Android en el proceso de compilación de Eclipse. Se recomienda la basada en *Eclipse CDT Builder* [94].

Es importante asegurarse de que Eclipse tiene instalado el plugin CDT. Se hace entrando en *Help ▷About Eclipse SDK* y presionando en el botón *Installation Details* (figura 4.14).

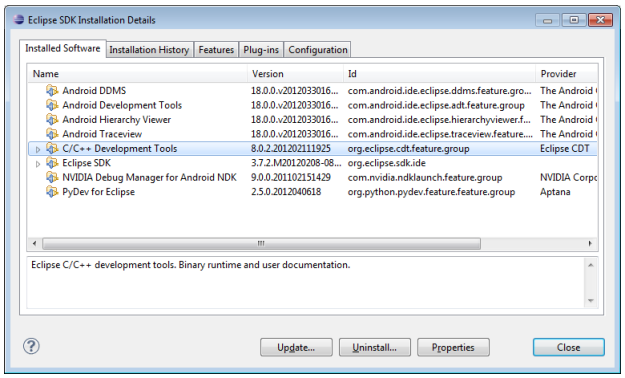


Figura 4.14.: Eclipse *Installation Details*.

Para instalar el plugin CDT se usa el menú *Help* ▷*Install New Software...*, y entonces se introduce la URL del repositorio de CDT (http://download.eclipse.org/tools/cdt/releases/indigo) tal como se muestra en la figura 4.15 y se pulsa el botón *Add...*, se le da un nombre (p.e. CDT) y se hace *click* en *OK*.

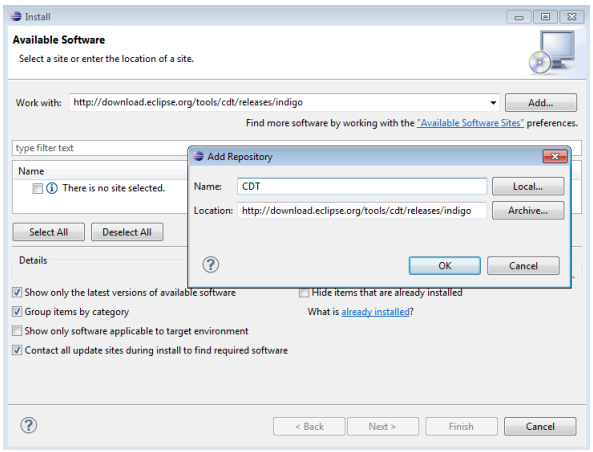


Figura 4.15.: Instalación del *plugin* ADT.

Con instalar *CDT Main Features* suele ser suficiente (figura 4.16).

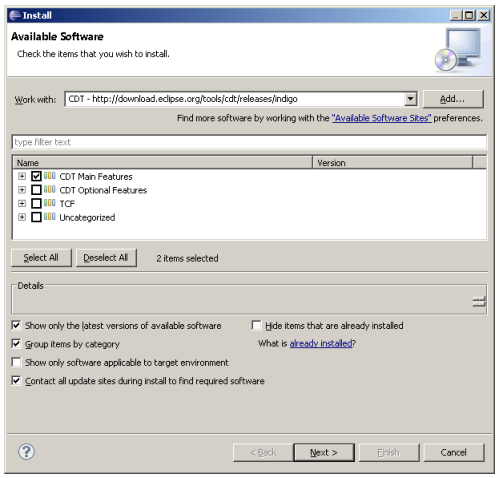


Figura 4.16.: Selección de elementos del plugin CDT.

1. Se define la variable de entorno *NDKROOT* que contiene la ruta al NDK de Android en el sistema (p.e. "`X:\Apps\android-ndk-r8`" o "`/opt/android-ndk-r8`").

2. *CDT Builder* está configurado para sistemas Windows, en Linux o MacOS se abre *Project Properties* en los proyectos que tengan parte JNI, se selecciona *C/C++ Build* en el panel izquierdo, se elimina "*.cmd*" y se deja "*${NDKROOT}/ndk-build*" en la caja de edición *Build command* y por último se pulsa *OK* (figura 4.17).

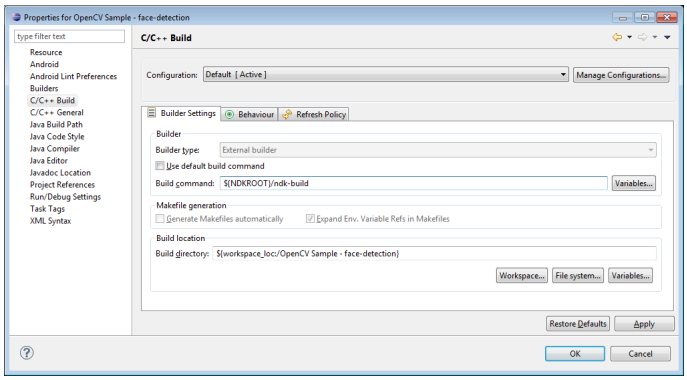


Figura 4.17.: Configuración del *plugin* CDT.

3. Se emplea *Project* ▷*Clean...* para estar seguro de que se invoca el compilador NDK al compilar el proyecto (figura 4.18).

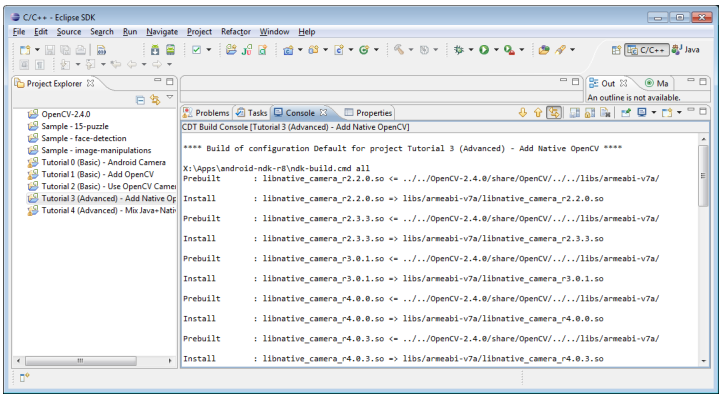


Figura 4.18.: Compilación del código nativo.

4.4.5.5. Estructura de los scripts *Android.mk* y *Application.mk*

El *script Android.mk* normalmente tiene la estructura mostrada en el código 4.6.

```
LOCAL_PATH := $(call my-dir)
```

```
include $(CLEAR_VARS)
LOCAL_MODULE    := <nombre_del_modulo>
LOCAL_SRC_FILES := <lista de los ficheros de proyecto .c y .cpp>
<algún nombre de variable> := <algún valor de variable>
...
<algún nombre de variable> := <algún valor de variable>

include $(BUILD_SHARED_LIBRARY)
```

Código 4.6: Script *Android.mk*.

Este es el fichero mínimo *Android.mk*, que compila el código fuente C++ de una aplicación Android. Notese que las dos primeras líneas y la última línea son obligatorias en cualquier *Android.m*k.

Normalmente, el fichero *Application.mk* es opcional, pero en el caso de un proyecto que usa OpenCV, donde se usan STL y excepciones en C++, también debe escribirse. El en código 4.7 se muestra un ejemplo de *Application.mk*.

```
APP_STL := gnustl_static
APP_CPPFLAGS := -frtti -fexceptions
APP_ABI := armeabi-v7a
```

Código 4.7: Ejemplo de *Application.mk*.

4.5. Ejemplos de aplicaciones para Android

4.5.1. Aplicación que muestra la velocidad mediante en uso del GPS

Esta aplicación muestra un velocímetro simple mediante el uso de GPS. En las líneas de 71 a 118 del cuadro código 4.8 se muestra como acceder al GPS creando e inicializando un *LocationManager*, como recibir actualizaciones de posición mediante un *LocationListener* o como crear un *AlertDialog* que informe al usuario si el GPS no está activo al lanzar la aplicación y solicite de este que lo active (ver figura 4.19). Igualmente, se muestra como activar el *PowerManager* y crear un *wakelock* que evita que se apague la pantalla al iniciar la aplicación (líneas 46 a 49) y como eliminarlo una vez se sale de la aplicación (línea 69). También se muestra como presentar la velocidad y posición (latitud y longitud) mediante simples *TextView* (líneas *119 a 134*) o mediante un *TextView* personalizado (líneas 51 a 60) y como hacer que este *TextView* cambie

de color según se este por debajo (en verde), próximo (en amarillo) o por encima (en rojo) de una velocidad determinada (líneas 145 a 159). En el código 4.9 se muestra la definición en lenguaje XML de la interfaz de la aplicación. La figura 4.20 muestra el velocímetro en funcionamiento.

Figura 4.19.: *AlertDialog* que solicita la activación del GPS.

```
1  package irv.Fuente.TTF;
2
3  import java.math.BigDecimal;
4  import java.math.RoundingMode;
5
6  import android.app.Activity;
7  import android.app.AlertDialog;
8  import android.content.Context;
9  import android.content.DialogInterface;
10 import android.content.Intent;
11 import android.graphics.Color;
12 import android.graphics.Typeface;
13 import android.location.Location;
14 import android.location.LocationListener;
15 import android.location.LocationManager;
16 import android.os.Bundle;
17 import android.os.Handler;
18 import android.os.PowerManager;
19 import android.widget.TextView;
20 import android.widget.Toast;
21
22 public class FuenteTTF extends Activity {
23
24     protected PowerManager.WakeLock wakelock;
25
26     public final int MULTIPLICADOR_HORA = 3600;
27     public final double MULTIPLICADOR_UNIDAD = 0.001;
28     public final int MAX_LIMIT = 151;
29     public final int MAX_DELAY = 500;
30     private TextView velo1;
31     private TextView velo2;
32     private TextView tvLon;
33     private TextView tvLat;
34     private TextView tvVel;
35     private Handler mHandler;
36     private double velGPS;
37     private int valor2;
38     private LocationManager locManager;
39     private LocationListener locListener;
40     private int velocidad = 50;
41
42     /** Called when the activity is first created. */
```

```
43      @Override
        public void onCreate(Bundle savedInstanceState) {
45          super.onCreate(savedInstanceState);
            //Evitar que la pantalla se apague
47          final PowerManager pm=(PowerManager)getSystemService(Context.POWER_SERVICE);
            this.wakelock=pm.newWakeLock(PowerManager.SCREEN_DIM_WAKE_LOCK, "etiqueta");
49          wakelock.acquire();
            setContentView(R.layout.main);
51          // Configuración Velocimetro
            velo1 = (TextView) findViewById(R.id.velocidad1);
53          velo2 = (TextView) findViewById(R.id.velocidad2);
            //Asignar Tipo de Fuente
55          Typeface lcdFont = Typeface.createFromAsset(getAssets(), "fonts/LED.ttf");
            velo1.setTypeface(lcdFont);
57          velo2.setTypeface(lcdFont);
            //Valor Inicial
59          velo1.setText("---");
            //Fin Configuración Velocímetro
61          tvLon = (TextView) findViewById(R.id.Longitud);
            tvLat = (TextView) findViewById(R.id.Latitud);
63          tvVel = (TextView) findViewById(R.id.velo);
            comenzarLocalizacion();
65      }
        @Override
67      protected void onDestroy(){
            super.onDestroy();
69          this.wakelock.release();
        }
71      private void comenzarLocalizacion() {
            //Obtenemos una referencia al LocationManager
73          locManager = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
            //Obtenemos la última posición conocida
75          Location loc =locManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
            //Mostramos la última posición conocida
77          mostrarPosicion(loc);
            //Nos registramos para recibir actualizaciones de la posición
79          locListener = new LocationListener() {
                public void onLocationChanged(Location location) {
81                  mostrarPosicion(location);
                }
83              public void onProviderDisabled(String provider){
                    Toast.makeText(getApplicationContext(), "Provider OFF", Toast.
                        LENGTH_SHORT).show();
85                  AlertDialog.Builder ad = new AlertDialog.Builder(FuenteTTF.this);
                    ad.setCancelable(false);
87                  ad.setTitle("ATENCION");
                    ad.setMessage("Es necesario activar el GPS"+"\n"+"¿Desea hacerlo?");
89                  ad.setPositiveButton("SI", new DialogInterface.OnClickListener() {

91                      @Override
                        public void onClick(DialogInterface dialog, int which) {
93                          Intent intent = new Intent(android.provider.Settings.
                                ACTION_LOCATION_SOURCE_SETTINGS);
                            startActivity(intent);
95                          dialog.dismiss();
                        }
97                  });
                    ad.setNegativeButton("NO", new DialogInterface.OnClickListener() {
99
```

```
                    @Override
101                 public void onClick(DialogInterface dialog, int which) {
                        dialog.dismiss();
103                 }
                });
105             ad.create();
                try {
107                 ad.show();
                } catch(Exception e){
109             }
            }
111         public void onProviderEnabled(String provider){
                Toast.makeText(getApplicationContext(), "Provider ON", Toast.
                    LENGTH_SHORT).show();
113         }
            public void onStatusChanged(String provider, int status, Bundle extras){
115         }
        };
117     locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
            locListener);
    }
119 private void mostrarPosicion(Location loc) {
        if(loc != null)
121     {
            tvLat.setText("Latitud: " + String.valueOf(loc.getLatitude()));
123         tvLon.setText("Longitud: " + String.valueOf(loc.getLongitude()));
            tvVel.setText("Velocidad: " + String.valueOf(loc.getSpeed()));
125         velGPS = loc.getSpeed();
            double velGPSTrunk = redondeoDecimal(convertirVelocidad(velGPS),1);
127         mostrarVelocimetro(velGPSTrunk, velocidad);
        }
129     else
        {
131         tvLat.setText("Latitud: (sin_datos)");
            tvLon.setText("Longitud: (sin_datos)");
133     }
    }
135 private double convertirVelocidad(double velocidad){
        return ((velocidad * MULTIPLICADOR_HORA) * MULTIPLICADOR_UNIDAD);
137 }
    private double redondeoDecimal(double valor, final int posicionDecimal)
139 {
        BigDecimal bd = new BigDecimal(valor);
141     bd = bd.setScale(posicionDecimal, RoundingMode.HALF_DOWN);
        valor = bd.doubleValue();
143     return valor;
    }
145 private void mostrarVelocimetro(double velocidadGps, int velocidadMaxima){
        if (velocidadGps <= (velocidadMaxima*0.9)){
147         velo1.setTextColor(Color.parseColor("#00FF00"));
            velo2.setTextColor(Color.parseColor("#00FF00"));
149     } else {
            if (velocidadGps <= (velocidadMaxima*1.05)) {
151             velo1.setTextColor(Color.parseColor("#FFFF00"));
                velo2.setTextColor(Color.parseColor("#FFFF00"));
153         } else {
                velo1.setTextColor(Color.parseColor("#FF0000"));
155             velo2.setTextColor(Color.parseColor("#FF0000"));
            }
```

```
157         }
            velo1.setText(String.valueOf(velocidadGps));
159     }
}
```

Código 4.8: Velocímetro usando el GPS del terminal.

Figura 4.20.: Velocímetro GPS en funcionamiento.

```
<?xml version="1.0" encoding="utf-8"?>
2     <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
          android"
          android:id="@+id/layout_base"
4         android:layout_width="fill_parent"
          android:layout_height="fill_parent"
6         android:gravity="fill_horizontal" >

8         <TextView
              android:id="@+id/velocidad1"
10            android:layout_width="wrap_content"
              android:layout_height="wrap_content"
12            android:layout_centerInParent="true"
              android:layout_centerVertical="true"
14            android:layout_marginLeft="100dp"
              android:text="120"
16            android:textAppearance="?android:attr/textAppearanceLarge"
              android:textColor="#32cd32"
18            android:textSize="150sp" />
          <TextView
20            android:id="@+id/Longitud"
              android:layout_width="wrap_content"
22            android:layout_height="wrap_content"
              android:layout_alignParentBottom="true"
24            android:layout_alignParentLeft="true"
              android:text="Long: " />
26        <TextView
              android:id="@+id/Latitud"
28            android:layout_width="wrap_content"
              android:layout_height="wrap_content"
30            android:layout_above="@+id/Longitud"
              android:layout_alignParentLeft="true"
```

```
32              android:layout_marginBottom="10dp"
                android:text="Lat: " />
34          <TextView
                android:id="@+id/velo"
36              android:layout_width="wrap_content"
                android:layout_height="wrap_content"
38              android:layout_above="@+id/Latitud"
                android:layout_alignParentLeft="true"
40              android:layout_marginBottom="10dp"
                android:text="Velo:" />
42          <TextView
                android:id="@+id/velocidad2"
44              android:layout_width="wrap_content"
                android:layout_height="wrap_content"
46              android:layout_alignTop="@+id/velocidad1"
                android:layout_toRightOf="@+id/velocidad1"
48              android:text="Km/h"
                android:textAppearance="?android:attr/textAppearanceSmall"
50              android:textColor="#32cd32"
                android:textSize="30sp" />
52      </RelativeLayout>
```

Código 4.9: *main.xml* del velocímetro GPS.

4.5.2. Aplicación que muestra como usar la cámara del dispositivo.

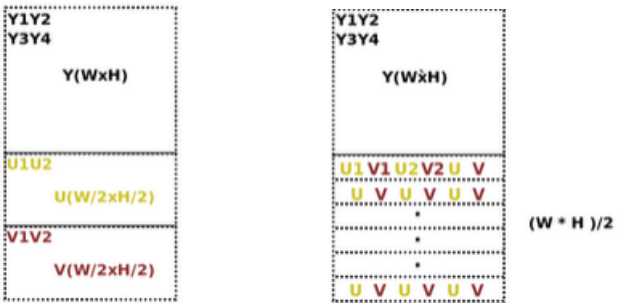
Esta aplicación muestra por pantalla lo que ve la cámara tanto en color como en blanco y negro. Consta de 3 clases:

- *SampleViewBase.java* (código 4.10): Es la clase base. Se encarga de acceder a la cámara y configurarla y de crear un elemento *SurfaceView* con su correspondiente *SurfaceHolder* (véase el apartado 4.3.5.3) que muestra la previsualización por la pantalla.

- *Sample0View* (código 4.11): Esta clase hereda de la anterior y es la encargada de hacer las conversiones de a RGB y a escala de grises. Igualmente, crea un objeto *Bitmap* que es el que mostrará el resultado por pantalla a través del correspondiente *SurfaceView*.

  Hay que tener en cuenta que por defecto, todos las cámaras de los dispositivos Android, capturan las imágenes en formato YUV420SP y que a partir del nivel 12 de la API también se soporta el formato YUV420P. Ambos son formatos YCbCr

(visto en 2.2.3) en los que los valores de luminancia (Y) del píxel se codifican en un bloque de 2×2 (Y11, Y12,Y21, Y22). En YUV420P después del bloque de luminancia se codifican los valores de U en otro bloque y a continuación un bloque más con los valores de V (figura 4.21a). En YUV420SP, los valores de U y de V se codifican en un solo bloque entrelazados (figura 4.21b). El código muestra como convertir las imágenes YUV420SP a escala de grises (líneas 30-34) y a RGB (líneas 35-56).



(a) YUV420 Planar. **(b)** YUV420 Semi-Planar.

Figura 4.21.: Formatos de previsualización en Android.

- *Sample0Base* (código 4.12): Esta clase deriva de *Activity* y es la que se encarga de manejar a actividad de la aplicación.

```
1  package org.opencv.samples.tutorial0;
2
3  import java.io.IOException;
4  import java.util.List;
5
6  import android.content.Context;
7  import android.graphics.Bitmap;
8  import android.graphics.Canvas;
9  import android.graphics.ImageFormat;
10 import android.graphics.SurfaceTexture;
11 import android.hardware.Camera;
12 import android.hardware.Camera.PreviewCallback;
13 import android.os.Build;
14 import android.util.Log;
15 import android.view.SurfaceHolder;
16 import android.view.SurfaceView;
17
18 public abstract class SampleViewBase extends SurfaceView implements SurfaceHolder.
       Callback, Runnable {
19     private static final String TAG = "Sample::SurfaceView";
20
21     private Camera          mCamera;
22     private SurfaceHolder   mHolder;
23     private int             mFrameWidth;
24     private int             mFrameHeight;
25     private byte[]          mFrame;
26     private boolean         mThreadRun;
27     private byte[]          mBuffer;
28
```

```
    public SampleViewBase(Context context) {
30      super(context);
        mHolder = getHolder();
32      mHolder.addCallback(this);
        Log.i(TAG, "Instantiated new " + this.getClass());
34  }
    public int getFrameWidth() {
36      return mFrameWidth;
    }
38  public int getFrameHeight() {
        return mFrameHeight;
40  }
    public void setPreview() throws IOException {
42      if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB)
            mCamera.setPreviewTexture( new SurfaceTexture(10) );
44      else
        mCamera.setPreviewDisplay(null);
46  }
    public boolean openCamera() {
48      Log.i(TAG, "openCamera");
        releaseCamera();
50      mCamera = Camera.open();
        if(mCamera == null) {
52          Log.e(TAG, "Can't open camera!");
            return false;
54      }
        mCamera.setPreviewCallbackWithBuffer(new PreviewCallback() {
56          public void onPreviewFrame(byte[] data, Camera camera) {
                synchronized (SampleViewBase.this) {
58                  System.arraycopy(data, 0, mFrame, 0, data.length);
                    SampleViewBase.this.notify();
60              }
                camera.addCallbackBuffer(mBuffer);
62          }
        });
64      return true;
    }
66  public void releaseCamera() {
        Log.i(TAG, "releaseCamera");
68      mThreadRun = false;
        synchronized (this) {
70          if (mCamera != null) {
                mCamera.stopPreview();
72              mCamera.setPreviewCallback(null);
                mCamera.release();
74              mCamera = null;
            }
76      }
        onPreviewStopped();
78  }
    public void setupCamera(int width, int height) {
80      Log.i(TAG, "setupCamera");
        synchronized (this) {
82          if (mCamera != null) {
                Camera.Parameters params = mCamera.getParameters();
84              List<Camera.Size> sizes = params.getSupportedPreviewSizes();
                mFrameWidth = width;
86              mFrameHeight = height;
                // selección del tamaño óptimo de la vista previa de la cámara
```

```
88                {
                      int  minDiff = Integer.MAX_VALUE;
90                    for (Camera.Size size : sizes) {
                          if (Math.abs(size.height - height) < minDiff) {
92                            mFrameWidth = size.width;
                              mFrameHeight = size.height;
94                            minDiff = Math.abs(size.height - height);
                          }
96                    }
                  }
98                params.setPreviewSize(getFrameWidth(), getFrameHeight());
                  List<String> FocusModes = params.getSupportedFocusModes();
100               if (FocusModes.contains(Camera.Parameters.FOCUS_MODE_CONTINUOUS_VIDEO)
                      )
                  {
102                   params.setFocusMode(Camera.Parameters.FOCUS_MODE_CONTINUOUS_VIDEO)
                          ;
                  }
104
                  mCamera.setParameters(params);
106
                  /* Ahora se asigna el buffer */
108               params = mCamera.getParameters();
                  int size = params.getPreviewSize().width * params.getPreviewSize().
                      height;
110               size  = size * ImageFormat.getBitsPerPixel(params.getPreviewFormat())
                      / 8;
                  mBuffer = new byte[size];
112               /* El buffer donde se copiara el frame inicial */
                  mFrame = new byte [size];
114               mCamera.addCallbackBuffer(mBuffer);
                  try {
116                   setPreview();
                  } catch (IOException e) {
118                   Log.e(TAG, "mCamera.setPreviewDisplay/setPreviewTexture fails: " +
                          e);
                  }
120               /* Notificar que la vista previa va a ponerse en marcha y proporcionar
                       el tamaño de la previsualización */
                  onPreviewStarted(params.getPreviewSize().width, params.getPreviewSize
                      ().height);
122               /* Ya podemos iniciar una previsualización */
                  mCamera.startPreview();
124           }
          }
126   }
      public void surfaceChanged(SurfaceHolder _holder, int format, int width, int
          height) {
128       Log.i(TAG, "surfaceChanged");
          setupCamera(width, height);
130   }
      public void surfaceCreated(SurfaceHolder holder) {
132       Log.i(TAG, "surfaceCreated");
          (new Thread(this)).start();
134   }
      public void surfaceDestroyed(SurfaceHolder holder) {
136       Log.i(TAG, "surfaceDestroyed");
          releaseCamera();
138   }
```

```
    /* El mapa de bits devuelto por este método debe ser propiedad del hijo y liberado
         en onPreviewStopped () */
    protected abstract Bitmap processFrame(byte[] data);
    /**
    * Este método se llama cuando el proceso de vista previa se está iniciando.
    * Se le llama antes de entregar el primer fotograma y de qu processFrame sea
        llamado.
    * Se le llama con los parámetros ancho y alto del proceso de vista previa.
    * Se puede utilizar para preparar los datos necesarios durante el procesamiento de
         frames.
    * @param previewWidth - la anchura de los fotogramas de vista previa que serán
        entregados por processFrame.
    * @param previewHeight - la altura de los fotogramas de vista previa que serán
        entregados por processFrame.
    */
    protected abstract void onPreviewStarted(int previewWidtd, int previewHeight);
    /**
    * Este método se llama cuando se detiene la previsualización.
    * Cuando se llama a este método se cierra la previsualización y se finaliza todo
        el procesamiento de fotogramas.
    * Si el objeto Bitmap devuelto por processFrame se almacena en caché - es un buen
        momento para reciclarlo.
    * Todos los demás recursos utilizados durante la vista previa pueden ser liberados
        .
    */
    protected abstract void onPreviewStopped();

    public void run() {
        mThreadRun = true;
        Log.i(TAG, "Starting processing thread");
        while (mThreadRun) {
            Bitmap bmp = null;
            synchronized (this) {
                try {
                    this.wait();
                    bmp = processFrame(mFrame);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            if (bmp != null) {
                Canvas canvas = mHolder.lockCanvas();
                if (canvas != null) {
                    canvas.drawBitmap(bmp, (canvas.getWidth() - getFrameWidth()) / 2,
                        (canvas.getHeight() - getFrameHeight()) / 2, null);
                    mHolder.unlockCanvasAndPost(canvas);
                }
            }
        }
        Log.i(TAG, "Finishing processing thread");
    }
}
```

Código 4.10: *SampleViewBase.java*

```
package org.opencv.samples.tutorial0;

import android.content.Context;
import android.graphics.Bitmap;
```

```
5  import android.util.Log;
7  class Sample0View extends SampleViewBase {
       private static final String TAG = "Sample0View";
9
       int         mSize;
11     int[]       mRGBA;
       private Bitmap mBitmap;
13     private int mViewMode;
15     public static final int     VIEW_MODE_RGBA = 0;
       public static final int     VIEW_MODE_GRAY = 1;
17
       public Sample0View(Context context) {
19         super(context);
           mSize = 0;
21         mViewMode = VIEW_MODE_RGBA;
       }
23     @Override
       protected Bitmap processFrame(byte[] data) {
25         int frameSize = getFrameWidth() * getFrameHeight();
27         int[] rgba = mRGBA;
29         final int view_mode = mViewMode;
           if (view_mode == VIEW_MODE_GRAY) {
31             for (int i = 0; i < frameSize; i++) {
                   int y = (0xff & ((int) data[i]));
33                 rgba[i] = 0xff000000 + (y << 16) + (y << 8) + y;
               }
35         } else if (view_mode == VIEW_MODE_RGBA) {
               for (int i = 0; i < getFrameHeight(); i++)
37                 for (int j = 0; j < getFrameWidth(); j++) {
                       int index = i * getFrameWidth() + j;
39                     int supply_index = frameSize + (i >> 1) * getFrameWidth() + (j &
                           ~1);
                       int y = (0xff & ((int) data[index]));
41                     int u = (0xff & ((int) data[supply_index + 0]));
                       int v = (0xff & ((int) data[supply_index + 1]));
43                     y = y < 16 ? 16 : y;
45                     float y_conv = 1.164f * (y - 16);
                       int r = Math.round(y_conv + 1.596f * (v - 128));
47                     int g = Math.round(y_conv - 0.813f * (v - 128) - 0.391f * (u -
                           128));
                       int b = Math.round(y_conv + 2.018f * (u - 128));
49
                       r = r < 0 ? 0 : (r > 255 ? 255 : r);
51                     g = g < 0 ? 0 : (g > 255 ? 255 : g);
                       b = b < 0 ? 0 : (b > 255 ? 255 : b);
53
                       rgba[i * getFrameWidth() + j] = 0xff000000 + (b << 16) + (g << 8)
                           + r;
55                 }
           }
57         mBitmap.setPixels(rgba, 0/* offset */, getFrameWidth() /* stride */, 0, 0,
               getFrameWidth(), getFrameHeight());
           return mBitmap;
59     }
```

```
    @Override
61  protected void onPreviewStarted(int previewWidth, int previewHeight) {
        Log.i(TAG, "onPreviewStarted("+previewWidth+", "+previewHeight+")");
63      /* Create a bitmap that will be used through to calculate the image to */
        mBitmap = Bitmap.createBitmap(previewWidth, previewHeight, Bitmap.Config.
            ARGB_8888);
65      mRGBA = new int[previewWidth * previewHeight];
    }
67  @Override
    protected void onPreviewStopped() {
69      Log.i(TAG, "onPreviewStopped");
        if(mBitmap != null) {
71          mBitmap.recycle();
            mBitmap = null;
73      }
        if(mRGBA != null) {
75          mRGBA = null;
        }
77  }
    public void setViewMode(int viewMode) {
79      Log.i(TAG, "setViewMode("+viewMode+")");
        mViewMode = viewMode;
81  }
}
```

Código 4.11: *Sample0View.java*

```
package org.opencv.samples.tutorial0;
2
import android.app.Activity;
4 import android.app.AlertDialog;
import android.content.DialogInterface;
6 import android.os.Bundle;
import android.util.Log;
8 import android.view.Menu;
import android.view.MenuItem;
10 import android.view.Window;

12 public class Sample0Base extends Activity {
    private static final String TAG = "Sample::Activity";
14
    private MenuItem       mItemPreviewRGBA;
16  private MenuItem       mItemPreviewGray;
    private Sample0Viev mView;
18
    public Sample0Base() {
20      Log.i(TAG, "Instantiated new " + this.getClass());
    }
22  @Override
    protected void onPause() {
24      Log.i(TAG, "onPause");
        super.onPause();
26      mView.releaseCamera();
    }
28  @Override
    protected void onResume() {
30      Log.i(TAG, "onResume");
        super.onResume();
32      if( !mView.openCamera() ) {
```

```
            AlertDialog ad = new AlertDialog.Builder(this).create();
            ad.setCancelable(false); // This blocks the 'BACK' button
            ad.setMessage("Fatal error: can't open camera!");
            ad.setButton("OK", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int which) {
                    dialog.dismiss();
                    finish();
                }
            });
            ad.show();
        }
    }
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        Log.i(TAG, "onCreate");
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        mView = new Sample0View(this);
        setContentView(mView);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menú) {
        Log.i(TAG, "onCreateOptionsMenu");
        mItemPreviewRGBA = menú.add("Preview RGBA");
        mItemPreviewGray = menú.add("Preview GRAY");
        return true;
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        Log.i(TAG, "Menu Item selected " + item);
        if (item == mItemPreviewRGBA)
            mView.setViewMode(Sample0View.VIEW_MODE_RGBA);
        else if (item == mItemPreviewGray)
            mView.setViewMode(Sample0View.VIEW_MODE_GRAY);
        return true;
    }
}
```

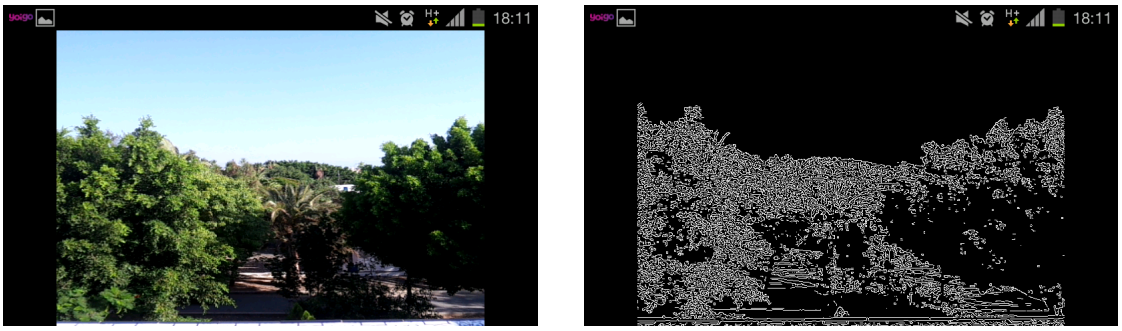
Código 4.12: *Sample0Base.java*

4.5.3. Aplicación que integra OpenCV.

Al igual que la aplicación anterior, esta muestra por pantalla lo que ve la cámara tanto en color como en blanco y negro. Además muestra el resultado de la aplicación del algoritmo Canny de detección de bordes [28] a la imagen de entrada. Consta de 3 clases:

- *SampleViewBase.java* (código 4.10): Es la misma que la de la aplicación anterior.
- *Sample1View.java* (código 4.13): Al igual que en la aplicación anterior, esta clase es la encargada de hacer las conversiones de a RGB y a escala de grises pero, a

diferencia de la anterior, la conversión del formato YUV a escala de grises y a RGB se realiza mediante el método de OpenCV *Imgproc.cvtColor* (líneas 71 y 74 respectivamente). También incorpora una llamada al algoritmo de detección de bordes Canny (línea 77). En la figura 4.22 se muestra un ejemplo de este algoritmo.



(a) Imagen original. **(b)** Imagen resultado.

Figura 4.22.: Ejemplo del algoritmo Canny.

- *Sample1java.java* (código 4.14): Esta clase deriva de *Activity* y es la que se encarga de manejar a actividad de la aplicación.

```
1  package org.opencv.samples.tutorial1;
2
3  import org.opencv.android.Utils;
4  import org.opencv.core.CvType;
5  import org.opencv.core.Mat;
6  import org.opencv.imgproc.Imgproc;
7
8  import android.content.Context;
9  import android.graphics.Bitmap;
10 import android.util.AttributeSet;
11 import android.util.Log;
12
13 class Sample1View extends SampleViewBase {
14
15     public static final int VIEW_MODE_RGBA  = 0;
16     public static final int VIEW_MODE_GRAY  = 1;
17     public static final int VIEW_MODE_CANNY = 2;
18
19     private Mat mYuv;
20     private Mat mRgba;
21     private Mat mGraySubmat;
22     private Mat mIntermediateMat;
23     private Bitmap mBitmap;
24     private int mViewMode;
25
26     public Sample1View(Context context, AttributeSet attrs) {
27         super(context, attrs);
28         mViewMode = VIEW_MODE_RGBA;
29     }
30     @Override
31     protected void onPreviewStarted(int previewWidth, int previewHeight) {
```

```
32          synchronized (this) {
                //Inicializar las Mats antes de su uso
34              mYuv = new Mat(getFrameHeight() + getFrameHeight() / 2, getFrameWidth(),
                    CvType.CV_8UC1);
                mGraySubmat = mYuv.submat(0, getFrameHeight(), 0, getFrameWidth());
36              mRgba = new Mat();
                mIntermediateMat = new Mat();
38              mBitmap = Bitmap.createBitmap(previewWidth, previewHeight, Bitmap.Config.
                    ARGB_8888);
                }
40      }
        @Override
42      protected void onPreviewStopped() {
            if(mBitmap != null) {
44              mBitmap.recycle();
            }
46          synchronized (this) {
                // Liberar las Mats de forma explícita
48              if (mYuv != null)
                    mYuv.release();
50              if (mRgba != null)
                    mRgba.release();
52              if (mGraySubmat != null)
                    mGraySubmat.release();
54              if (mIntermediateMat != null)
                    mIntermediateMat.release();
56
                mYuv = null;
58              mRgba = null;
                mGraySubmat = null;
60              mIntermediateMat = null;
            }
62      }
        @Override
64      protected Bitmap processFrame(byte[] data) {
            mYuv.put(0, 0, data);
66
            final int viewMode = mViewMode;
68
            switch (viewMode) {
70          case VIEW_MODE_GRAY:
                Imgproc.cvtColor(mGraySubmat, mRgba, Imgproc.COLOR_GRAY2RGBA, 4);
72              break;
            case VIEW_MODE_RGBA:
74              Imgproc.cvtColor(mYuv, mRgba, Imgproc.COLOR_YUV420sp2RGB, 4);
                break;
76          case VIEW_MODE_CANNY:
                Imgproc.Canny(mGraySubmat, mIntermediateMat, 80, 100);
78              Imgproc.cvtColor(mIntermediateMat, mRgba, Imgproc.COLOR_GRAY2BGRA, 4);
                break;
80          }

82          Bitmap bmp = mBitmap;

84          try {
                Utils.matToBitmap(mRgba, bmp);
86          } catch(Exception e) {
                Log.e("org.opencv.samples.tutorial1", "Utils.matToBitmap() throws an
                    exception: " + e.getMessage());
```

```
88              bmp.recycle();
                bmp = null;
90          }
            return bmp;
92      }
        public void setViewMode(int viewMode) {
94          mViewMode = viewMode;
        }
96  }
```

Código 4.13: *Sample1View.java*

```
    package org.opencv.samples.tutorial1;
2
    import android.app.Activity;
4   import android.app.AlertDialog;
    import android.content.DialogInterface;
6   import android.os.Bundle;
    import android.util.Log;
8   import android.view.Menu;
    import android.view.MenuItem;
10  import android.view.Window;

12  public class Sample1Java extends Activity {
        private static final String TAG = "Sample::Activity";
14
        private MenuItem        mItemPreviewRGBA;
16      private MenuItem        mItemPreviewGray;
        private MenuItem        mItemPreviewCanny;
18      private Sample1View  mView;

20      public Sample1Java() {
            Log.i(TAG, "Instantiated new " + this.getClass());
22      }
        @Override
24      protected void onPause() {
            Log.i(TAG, "onPause");
26          super.onPause();
            mView.releaseCamera();
28      }
        @Override
30      protected void onResume() {
            Log.i(TAG, "onResume");
32          super.onResume();
            if( !mView.openCamera() ) {
34              AlertDialog ad = new AlertDialog.Builder(this).create();
                ad.setCancelable(false); // This blocks the 'BACK' button
36              ad.setMessage("Fatal error: can't open camera!");
                ad.setButton("OK", new DialogInterface.OnClickListener() {
38                  public void onClick(DialogInterface dialog, int which) {
                        dialog.dismiss();
40                      finish();
                    }
42              });
                ad.show();
44          }
        }
46
```

```
    /** Called when the activity is first created. */
48  @Override
    public void onCreate(Bundle savedInstanceState) {
50      Log.i(TAG, "onCreate");
        super.onCreate(savedInstanceState);
52      requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.main);
54      mView = (Sample1View) findViewById(R.id.surfaceView1);
    }
56  @Override
    public boolean onCreateOptionsMenu(Menu menú) {
58      Log.i(TAG, "onCreateOptionsMenu");
        mItemPreviewRGBA = menú.add("Preview RGBA");
60      mItemPreviewGray = menú.add("Preview GRAY");
        mItemPreviewCanny = menú.add("Canny");
62      return true;
    }
64  @Override
    public boolean onOptionsItemSelected(MenuItem item) {
66      Log.i(TAG, "Menu Item selected " + item);
        if (item == mItemPreviewRGBA) {
68          mView.setViewMode(Sample1View.VIEW_MODE_RGBA);
        } else if (item == mItemPreviewGray) {
70          mView.setViewMode(Sample1View.VIEW_MODE_GRAY);
        } else if (item == mItemPreviewCanny) {
72          mView.setViewMode(Sample1View.VIEW_MODE_CANNY);
        }
74      return true;
    }
76  }
```

Código 4.14: *Sample1Java.java*

4.5.4. Aplicación que accede a la cámara del móvil de manera nativa.

Esta aplicación tiene la misma funcionalidad que la anterior. Muestra la previsualización en RGB y en escala de grises, además del algoritmo Canny. La diferencia radica en que accede a la cámara del terminal de forma nativa a través de OpenCV y no de la API de Android. Esto acelera muchas operaciones sobre la imagen, como por ejemplo el cambio de espacio de color, lo que incide en el rendimiento. Al igual que las anteriores esta aplicación cuenta con tres clases:

- *SampleCvViewBase.java* (código 4.15): En esta clase es donde radica la diferencia respecto a las dos aplicaciones anteriores. En lugar de acceder a la cámara mediante un objeto de la clase *Camera* propio de Android (código 4.10, línea 21), se accede directamente al hardware mediante un objeto de la clase *VideoCapture* de OpenCV (línea 20).

- *Sample2View.java* (código 4.16): Esta clase realiza la misma función que la clase *Sample1View.java* de la aplicación anterior (en página 116).
- *Sample2NativeCamera.java* (código 4.17): Es la clase encargada de gestionar la funcionalidad de la aplicación (hereda de *Activity*).

```
package org.opencv.samples.tutorial2;

import java.util.List;

import org.opencv.core.Size;
import org.opencv.highgui.VideoCapture;
import org.opencv.highgui.Highgui;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.util.Log;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public abstract class SampleCvViewBase extends SurfaceView implements SurfaceHolder.
    Callback, Runnable {
    private static final String TAG = "Sample::SurfaceView";

    private SurfaceHolder   mHolder;
    private VideoCapture     mCamera;

    public SampleCvViewBase(Context context) {
        super(context);
        mHolder = getHolder();
        mHolder.addCallback(this);
        Log.i(TAG, "Instantiated new " + this.getClass());
    }
    public boolean openCamera() {
        Log.i(TAG, "openCamera");
        synchronized (this) {
            releaseCamera();
            mCamera = new VideoCapture(Highgui.CV_CAP_ANDROID);
            if (!mCamera.isOpened()) {
                mCamera.release();
                mCamera = null;
                Log.e(TAG, "Failed to open native camera");
                return false;
            }
        }
        return true;
    }
    public void releaseCamera() {
        Log.i(TAG, "releaseCamera");
        synchronized (this) {
            if (mCamera != null) {
                    mCamera.release();
                    mCamera = null;
            }
```

```
        }
50  }

52  public void setupCamera(int width, int height) {
        Log.i(TAG, "setupCamera("+width+", "+height+")");
54      synchronized (this) {
            if (mCamera != null && mCamera.isOpened()) {
56              List<Size> sizes = mCamera.getSupportedPreviewSizes();
                int mFrameWidth = width;
58              int mFrameHeight = height;
                // selecting optimal camera preview size
60              {
                    double minDiff = Double.MAX_VALUE;
62                  for (Size size : sizes) {
                        if (Math.abs(size.height - height) < minDiff) {
64                          mFrameWidth = (int) size.width;
                            mFrameHeight = (int) size.height;
66                          minDiff = Math.abs(size.height - height);
                        }
68                  }
                }
70              mCamera.set(Highgui.CV_CAP_PROP_FRAME_WIDTH, mFrameWidth);
                mCamera.set(Highgui.CV_CAP_PROP_FRAME_HEIGHT, mFrameHeight);
72          }
        }
74  }
    public void surfaceChanged(SurfaceHolder _holder, int format, int width, int
        height) {
76      Log.i(TAG, "surfaceChanged");
        setupCamera(width, height);
78  }
    public void surfaceCreated(SurfaceHolder holder) {
80      Log.i(TAG, "surfaceCreated");
        (new Thread(this)).start();
82  }
    public void surfaceDestroyed(SurfaceHolder holder) {
84      Log.i(TAG, "surfaceDestroyed");
        releaseCamera();
86  }
    protected abstract Bitmap processFrame(VideoCapture capture);
88

    public void run() {
90      Log.i(TAG, "Starting processing thread");
        while (true) {
92          Bitmap bmp = null;
            synchronized (this) {
94              if (mCamera == null)
                    break;
96              if (!mCamera.grab()) {
                    Log.e(TAG, "mCamera.grab() failed");
98                  break;
                }
100             bmp = processFrame(mCamera);
            }
102         if (bmp != null) {
                Canvas canvas = mHolder.lockCanvas();
104             if (canvas != null) {
                    canvas.drawBitmap(bmp, (canvas.getWidth() - bmp.getWidth()) / 2, (
                        canvas.getHeight() - bmp.getHeight()) / 2, null);
```

```
106                     mHolder.unlockCanvasAndPost(canvas);
                    }
108                 bmp.recycle();
                }
110         }
            Log.i(TAG, "Finishing processing thread");
112     }
}
```

Código 4.15: *SampleCvViewBase.java*

```
1  package org.opencv.samples.tutorial2;

3  import org.opencv.android.Utils;
   import org.opencv.core.Core;
5  import org.opencv.core.Mat;
   import org.opencv.core.Point;
7  import org.opencv.core.Scalar;
   import org.opencv.highgui.Highgui;
9  import org.opencv.highgui.VideoCapture;
   import org.opencv.imgproc.Imgproc;
11
   import android.content.Context;
13 import android.graphics.Bitmap;
   import android.util.Log;
15 import android.view.SurfaceHolder;

17 class Sample2View extends SampleCvViewBase {
       private Mat mRgba;
19     private Mat mGray;
       private Mat mIntermediateMat;
21
       public Sample2View(Context context) {
23         super(context);
       }
25     @Override
       public void surfaceCreated(SurfaceHolder holder) {
27         synchronized (this) {
               // initialize Mats before usage
29             mGray = new Mat();
               mRgba = new Mat();
31             mIntermediateMat = new Mat();
           }
33         super.surfaceCreated(holder);
       }
35     @Override
       protected Bitmap processFrame(VideoCapture capture) {
37         switch (Sample2NativeCamera.viewMode) {
           case Sample2NativeCamera.VIEW_MODE_GRAY:
39             capture.retrieve(mGray, Highgui.CV_CAP_ANDROID_GREY_FRAME);
               Imgproc.cvtColor(mGray, mRgba, Imgproc.COLOR_GRAY2RGBA, 4);
41             break;
           case Sample2NativeCamera.VIEW_MODE_RGBA:
43             capture.retrieve(mRgba, Highgui.CV_CAP_ANDROID_COLOR_FRAME_RGBA);
               Core.putText(mRgba, "OpenCV + Android", new Point(10, 100), 3, 2, new
                   Scalar(255, 0, 0, 255), 3);
```

```
45              break;
        case Sample2NativeCamera.VIEW_MODE_CANNY:
47              capture.retrieve(mGray, Highgui.CV_CAP_ANDROID_GREY_FRAME);
                Imgproc.Canny(mGray, mIntermediateMat, 80, 100);
49              Imgproc.cvtColor(mIntermediateMat, mRgba, Imgproc.COLOR_GRAY2BGRA, 4);
                break;
51          }

53          Bitmap bmp = Bitmap.createBitmap(mRgba.cols(), mRgba.rows(), Bitmap.Config.
                ARGB_8888);

55          try {
                Utils.matToBitmap(mRgba, bmp);
57              return bmp;
            } catch(Exception e) {
59              Log.e("org.opencv.samples.tutorial2", "Utils.matToBitmap() throws an
                    exception: " + e.getMessage());
                bmp.recycle();
61              return null;
            }
63      }
        @Override
65      public void run() {
            super.run();
67          synchronized (this) {
                // Explicitly deallocate Mats
69              if (mRgba != null)
                    mRgba.release();
71              if (mGray != null)
                    mGray.release();
73              if (mIntermediateMat != null)
                    mIntermediateMat.release();
75

                mRgba = null;
77              mGray = null;
                mIntermediateMat = null;
79          }
        }
81  }
```

Código 4.16: *Sample2View.java*

```
1   package org.opencv.samples.tutorial2;

3   import android.app.Activity;
    import android.app.AlertDialog;
5   import android.content.DialogInterface;
    import android.os.Bundle;
7   import android.util.Log;
    import android.view.Menu;
9   import android.view.MenuItem;
    import android.view.Window;
11
    public class Sample2NativeCamera extends Activity {
13      private static final String TAG = "Sample::Activity";
```

```
15      public static final int      VIEW_MODE_RGBA  = 0;
        public static final int      VIEW_MODE_GRAY  = 1;
17      public static final int      VIEW_MODE_CANNY = 2;

19      private MenuItem             mItemPreviewRGBA;
        private MenuItem             mItemPreviewGray;
21      private MenuItem             mItemPreviewCanny;

23      public static int            viewMode = VIEW_MODE_RGBA;

25      private Sample2View          mView;

27      public Sample2NativeCamera() {
            Log.i(TAG, "Instantiated new " + this.getClass());
29      }
        @Override
31      protected void onPause() {
            Log.i(TAG, "onPause");
33          super.onPause();
            mView.releaseCamera();
35      }
        @Override
37      protected void onResume() {
            Log.i(TAG, "onResume");
39          super.onResume();
            if( !mView.openCamera() ) {
41              AlertDialog ad = new AlertDialog.Builder(this).create();
                ad.setCancelable(false); // This blocks the 'BACK' button
43              ad.setMessage("Fatal error: can't open camera!");
                ad.setButton("OK", new DialogInterface.OnClickListener() {
45                  public void onClick(DialogInterface dialog, int which) {
                        dialog.dismiss();
47                      finish();
                    }
49              });
                ad.show();
51          }
        }
53      /** Called when the activity is first created. */
        @Override
55      public void onCreate(Bundle savedInstanceState) {
            Log.i(TAG, "onCreate");
57          super.onCreate(savedInstanceState);
            requestWindowFeature(Window.FEATURE_NO_TITLE);
59          mView = new Sample2View(this);
            setContentView(mView);
61      }
        @Override
63      public boolean onCreateOptionsMenu(Menu menú) {
            Log.i(TAG, "onCreateOptionsMenu");
65          mItemPreviewRGBA = menú.add("Preview RGBA");
            mItemPreviewGray = menú.add("Preview GRAY");
67          mItemPreviewCanny = menú.add("Canny");
            return true;
69      }
        @Override
71      public boolean onOptionsItemSelected(MenuItem item) {
            Log.i(TAG, "Menu Item selected " + item);
73          if (item == mItemPreviewRGBA)
```

```
            viewMode = VIEW_MODE_RGBA;
        else if (item == mItemPreviewGray)
            viewMode = VIEW_MODE_GRAY;
        else if (item == mItemPreviewCanny)
            viewMode = VIEW_MODE_CANNY;
        return true;
    }
}
```

Código 4.17: *Sample2NativeCamera.java*

Capítulo 5

Aprendizaje Automático y Reconocimiento Óptico de Caracteres

5.1. Introducción

En este capítulo se explicará el proceso para la creación del fichero que almacena las características que definen a la señal de limitación de velocidad. Se verá como se ha realizado el proceso de captura de datos (grabación de vídeo), la extracción de los frames de esos datos y su procesado para adaptarlos a las herramientas de "entrenamiento" que proporciona OpenCV.

Igualmente, se explicará el algoritmo de reconocimiento óptico de caracteres que se empleará en la identificación de los dígitos de la señal de tráfico. Este algoritmo es una variante del algoritmo de escaneo de líneas presentado en la sección 2.4.3 del capitulo 2 (Técnicas de Detección de Señales).

5.2. Aprendizaje Automático. Entrenamiento de Clasificadores en Cascada

Tal como se vio en el capítulo 3 (Estudio de las librerías OpenCV), OpenCV cuenta con módulos para detección de objetos (apartado 3.4.7) y para aprendizaje automático (apartado 3.4.8). En esta sección se mostrará el proceso seguido para entrenar un clasificador en cascada, tanto en la preparación de los datos como en la ejecución de la aplicación de entrenamiento.

OpenCV proporciona dos aplicaciones para entrenar un clasificador en cascada: `opencv_haartraining` y `opencv_traincascade`. `opencv_traincascade` es la versión

más actual y está desarrollada en C++ en conformidad con la API OpenCV 2.x. No obstante, la mayor diferencia entre ambas aplicaciones es que `opencv_traincascade` soporta tanto características tipo *Haar* [34] como *LBP* (*Local Binary Patterns*) [61]. Las características *LBP* son de tipo entero en contraposición con las características tipo *Haar*, que son de punto flotante. Por lo tanto, tanto el entrenamiento como la detección con características LBP son varias veces más rápidos que con características tipo *Haar*. En cuanto a la calidad de detección de *LBP* y *Haar*, éstas dependen del entrenamiento: tanto de los datos como de los parámetros de entrenamiento. Es posible entrenar un clasificador basado en *LBP* que proporcione prácticamente la misma calidad que uno basado en *Haar*.

`opencv_traincascade` y `opencv_haartraining` almacenan el clasificador entrenado en diferentes formatos de archivo. La interfaz de cascadas de detección más actual soporta ambos formatos (véase [95]). `opencv_traincascade` puede guardar (exportar) la cascada entrenada en el formato antiguo.

Además hay algunas utilidades auxiliares relacionadas con el entrenamiento.

- `opencv_createsamples`: Produce un conjunto de datos positivos que es reconocido tanto por `opencv_traincascade` como por `opencv_haartraining`. La salida es un fichero con extensión **.vec*, que es un formato binario que contiene imágenes.
- `opencv_performance`: Se emplea para medir la calidad de los clasificadores. De momento, sólo funciona con los clasificadores entrenados por `opencv_haartraining`.

5.2.1. Preparación de los datos de entrenamiento

El proceso de entrenamiento necesita un conjunto de muestras. Existen dos tipos de muestras: positivas y negativas. La muestras negativas corresponden a imágenes donde no aparece el objeto a identificar y las positivas corresponden a imágenes donde el objeto está presente. El conjunto de muestras negativas se debe preparar manualmente, mientras que el conjunto de muestras positivas se crea con la utilidad `opencv_createsamples`.

5.2.1.1. Muestras negativas

Las muestras negativas proceden de imágenes arbitrarias. Estas imágenes no deben contener los objetos a detectar. Las muestras negativas se enumeran en un archivo

especial. Es un archivo de texto en el que cada línea contiene el nombre de la imagen de muestra negativa relativo al directorio del archivo de descripción (véase figura 5.1). Este archivo debe crearse manualmente. A las muestras negativas también se les denomina imágenes o muestras de fondo. Las imágenes pueden ser de diferentes tamaños pero, cada imagen debe tener al menos el tamaño de la ventana de entrenamiento ya que estas imágenes se submuestrean al tamaño de entrenamiento.

Estructura del directorio:

```
/img
        img1.jpg
        img2.jpg
bg.txt
```

Fichero `bg.txt`:

```
img/img1.jpg
img/img2.jpg
```

Figura 5.1.: Ejemplo del directorio y fichero de muestras negativas.

5.2.1.2. Muestras positivas

Las muestras positivas son creadas por la utilidad `opencv_createsamples`. Pueden crearse a partir de una sola imagen con el objeto o de una colección de imágenes previamente marcadas.

Hay que tener en cuenta que se necesita un gran conjunto de datos de muestras positivas antes de entregárselas a la utilidad `opencv_createsamples`, ya que esta sólo aplica la transformación de perspectiva. Por ejemplo, puede bastar sólo con una muestra positiva de un objeto absolutamente rígido como un logotipo, pero definitivamente se necesitan cientos y hasta miles de muestras positivas para las señales de tráfico.

Por lo tanto, en el caso de proporcionar una sola imagen de muestra, se crea el conjunto de imágenes positivas aplicándole a dicha imagen rotaciones aleatorias, cambiándole la intensidad y colocándola sobre distintos fondos. La cantidad y el intervalo de la aleatoriedad puede ser controlado por los argumentos de línea de comandos de la herramienta `opencv_createsamples`.

Estos argumentos de línea de comandos son:

- `-vec <vec_file_name>`: Nombre del archivo de salida que contiene las muestras positivas para el entrenamiento.
- `-img <image_file_name>`: Imagen fuente del objeto (por ejemplo, un logotipo).

- `-bg <background_file_name>`: Archivo de descripción del fondo, contiene una lista de imágenes que se utilizan como fondo para las versiones del objeto distorsionadas de forma aleatoria.
- `-num <number_of_samples>`: Número de muestras positivas a generar.
- `-bgcolor <background_color>`: Color que indica el fondo transparente (actualmente se suponen imágenes en escala de grises). Como pueden aparecer artefactos de compresión, la cantidad de tolerancia de color puede ser especificado por `-bgthresh`. Todos los píxeles comprendidos entre `bgcolor-bgthresh` y `bgcolor+bgthresh` se interpretan como transparentes.
- `-bgthresh <background_color_threshold>`
- `-inv`: Si se especifica, los colores se invierten.
- `-randinv`: Si se especifica, los colores se invierten de forma aleatoria.
- `-maxidev <max_intensity_deviation>`: Desviación máxima de intensidad de los píxeles en las muestras de primer plano.
- `-maxxangle <max_x_rotation_angle>`
- `-maxyangle <max_y_rotation_angle>`
- `-maxxangle <max_z_rotation_angle>`: Los ángulos de rotación máximos deben ser dados en radianes.
- `-show`: Opción útil de depuración. Si se especifica, se mostrará cada muestra. Al pulsar la tecla *Esc* continuará el proceso de creación de las muestras sin mostrarlas.
- `-w <sample_width>`: Ancho (en píxeles) de las muestras de salida.
- `-h <sample_height>`: Altura (en píxeles) de las muestras de salida.

El procedimiento siguiente permite crear una instancia del objeto de muestra: La imagen fuente se hace girar al azar alrededor de los tres ejes. El ángulo elegido está limitado por `-max#angle`[1]. Entonces los píxeles que tienen la intensidad en el rango [`bgcolor-bgthresh`; `bgcolor+bgthresh`] se interpretan como transparentes. Se añade ruido blanco a las intensidades de primer plano. Si se especifica la opción `-inv` entonces se invierten las intensidades de los píxeles en primer plano. Si se especifica la opción

[1]Donde # es x, y o `z` según el eje que se rote.

`-randinv` entonces el algoritmo selecciona al azar si la inversión debe ser aplicada a la muestra. Por último, la imagen obtenida se coloca en un fondo seleccionado arbitrariamente del archivo de descripción de fondos, se le cambia de tamaño al especificado por `-w` y `-h` y se almacenan en el archivo **.vec*, especificado por la opción `-vec` de la línea de comando.

Las muestras positivas pueden obtenerse también a partir de una colección de imágenes previamente marcadas. Esta colección se describe mediante un archivo de texto similar al archivo de descripción de muestras negativas. Cada línea de este fichero se corresponde con una imagen. El primer elemento de la línea es al nombre del fichero. Le sigue el número de instancias del objeto. Los siguientes números son las coordenadas de los rectángulos que delimitan los objetos (x, y, anchura, altura).

La figura 5.2 muestra un ejemplo de una colección de muestras positivas. La imagen `Img1.jpg` contiene una instancia de objeto con las siguientes coordenadas del rectángulo delimitador: (140, 100, 45, 45). La imagen `Img2.jpg` contiene una instancia de objeto.

Estructura del directorio:

```
/img
        img1.jpg
        img2.jpg
info.dat
```

Fichero `bg.txt`:

```
img/img1.jpg  1  140 100 45 45
img/img2.jpg  2  100 200 50 50   50 30 25 25
```

Figura 5.2.: Ejemplo del directorio y fichero de muestras positivas.

Con el fin de crear muestras positivas de dicha recolección, se debe especificar el argumento `-info` en lugar de `-img`:

- `-info <collection_file_name>`: Fichero de descripción de la colección de imágenes marcadas.

El esquema de la creación de muestras en este caso es el siguiente. Las instancias de objetos se toman de las imágenes. Luego se redimensionan al tamaño de las muestras objetivo y se almacenan en el fichero de salida **.vec*. No se aplica distorsión, por lo que los únicos argumentos que serán empleados son `-w`, `-h`, `-show` y `-num`.

La utilidad `opencv_createsamples` puede utilizarse para examinar las muestras almacenadas en el archivo de muestras positivas. Para hacer esto solo deben especificarse los parámetros `-vec`, `-w` y `-h`.

5.2.2. Entrenamiento de cascadas

El siguiente paso es el entrenamiento del clasificador. Como se mencionó anteriormente, se puede utilizar `opencv_traincascade` u `opencv_haartraining` para entrenar un clasificador en cascada, pero sólo se describirá el más reciente `opencv_traincascade`.

Los argumentos de la línea de comandos de la aplicación `opencv_traincascade` están agrupados por objetivos:

1. Argumentos comunes:
   - `-data <cascade_dir_name>`: Donde debe ser almacenado el clasificador entrenado.
   - `-vec <vec_file_name>`: Fichero **.vec* con las muestras positivas (creado por la utilidad `opencv_createsamples`).
   - `-bg <background_file_name>`: Fichero de descripción de fondo (muestras negativas).
   - `-numPos <number_of_positive_samples>`
   - `-numNeg <number_of_negative_samples>`: Número de muestras positivas / negativas utilizadas en la formación de cada etapa del clasificador.
   - `-numStages <number_of_stages>`: Número de etapas de la cascada que serán entrenadas.
   - `-precalcValBufSize <precalculated_vals_buffer_size_in_Mb>`: Tamaño del buffer para los valores característicos precalculados (en Mb).
   - `-precalcIdxBufSize <precalculated_idxs_buffer_size_in_Mb>`: Tamaño de buffer de índices de características precalculados (en Mb). Cuanta más memoria tenga, más rápido será el proceso de formación.
   - `-baseFormatSave`: Este argumento se emplea en caso de características tipo *Haar*. Si se especifica, la cascada se guardará en el formato antiguo.
2. Parámetros de cascada:
   - `-stageType <BOOST(default)>`: Tipo de etapas. De momento sólo impulsado se admiten como un tipo de etapa los clasificadores potenciados (véase en la página 68).

- `-featureType<{HAAR(default), LBP}>`: Tipo de características: HAAR - características tipo *Haar*, LBP - patrones locales binarios.
- `-w <sampleWidth>`
- `-h <sampleHeight>`: Tamaño de las muestras de entrenamiento (en píxeles). Debe tener exactamente los mismos valores utilizados durante la creación de las muestras de entrenamiento (utilidad `opencv_createsamples`).

3. Parámetros del clasificador potenciado:

   - `bt <{DAB, RAB, LB, GAB(default)}>`: Tipo de clasificador potenciado: DAB - Discrete AdaBoost, RAB - Real AdaBoost, LB - LogitBoost, GAB - Gentle AdaBoost.
   - `-minHitRate <min_hit_rate>`: Tasa de éxito mínima deseada para cada etapa del clasificador. La tasa de éxito total se puede estimar como (`min_hit_rate ^ number_of_stages`).
   - `-maxFalseAlarmRate <max_false_alarm_rate>`: Tasa de falsa alarma máxima deseada para cada etapa del clasificador. La tasa de falsa alarma global puede calcularse como (`max_false_alarm_rate ^ number_of_stages`).
   - `-weightTrimRate <weight_trim_rate>`: Especifica si se debe utilizar el recorte y su peso. 0.95 suele ser un valor óptimo.
   - `-maxDepth <max_depth_of_weak_tree>`: Profundidad máxima del árbol débil. Una opción común es 1.
   - `-maxWeakCount <max_weak_tree_count>`: Conteo máximo de árboles débiles para todas las etapas en cascada. El clasificador potenciado (etapa) tendrá tantas árboles débiles ($\leq$`maxWeakCount`), según sea necesario para alcanzar el valor -`maxFalseAlarmRate` indicado.

4. Parámetros de las características tipo *Haar*:

   - `-mode <BASIC (default) | CORE | ALL>`: Selecciona el tipo de características *Haar* que se utilizarán en el entrenamiento. `BASIC` utiliza sólo las características verticales, mientras que `ALL` emplea el conjunto completo de características verticales y rotadas 45 grados.

5. Parámetros de patrones locales binarios:

   LBP no tiene parámetros.

Después de que la aplicación opencv_traincascade haya terminado su trabajo, la cascada entrenada se guardará en el archivo *cascade.xml* en la carpeta que fue especificada como parámetro `-data`. Los demás archivos de esta carpeta se crean por si se interrumpe el entrenamiento, por lo que se pueden borrar una vez finalizado este.

5.2.3. Entrenamiento de la cascada de la aplicación

En esta sección se explican los pasos seguidos para obtener un clasificador en cascada para su empleo en el aplicación de este proyecto fin de carrera. Desde la toma de muestras hasta el resultado final.

5.2.3.1. Toma y preparación de las muestras

El proceso de toma de muestras se tomaron vídeos con un terminal móvil situado en un soporte adherido al parabrisas de un vehículo y se empleo la aplicación que este proporciona (figura 5.3).

Figura 5.3.: Terminal móvil situado en un soporte para coche.

Se tomaron en total 12 vídeos de aproximadamente 20 minutos de duración en los que se intentó abarcar distintas condiciones climáticas y de luz (tomados a distintas horas del día). En la figura 5.4 se muestran los distintos vídeos tomados. Estos vídeos están disponibles en el CD adjunto al proyecto fin de carrera.

Para extraer los frames de estos vídeos se emplea la aplicación de libre distribución "HaarTraning Positive Builder" desarrollada por *Prodigy Productions LLC*[2]. Esta aplicación carga un vídeo y permite reproducirlo fotograma a fotograma e indicar si en

[2]http://www.prodigyproductionsllc.com

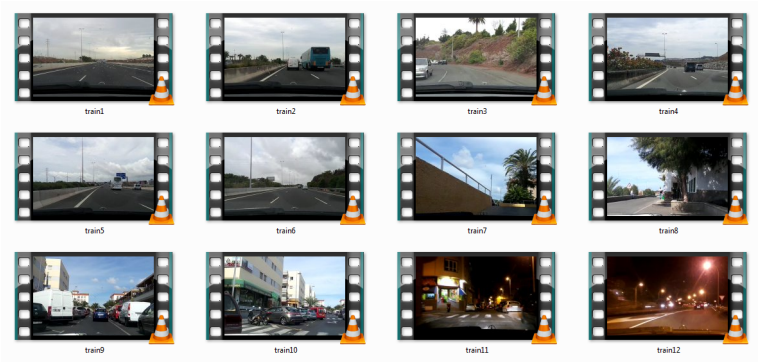


Figura 5.4.: Vídeos para la realización de las muestras.

la imagen aparece una señal objetivo e indicar sus coordenadas para crear el fichero de definición de muestras positivas y el directorio con las imágenes tal y como se explicó en el apartado 5.2.1.2. La figura 5.5 muestra un ejemplo de este paso.

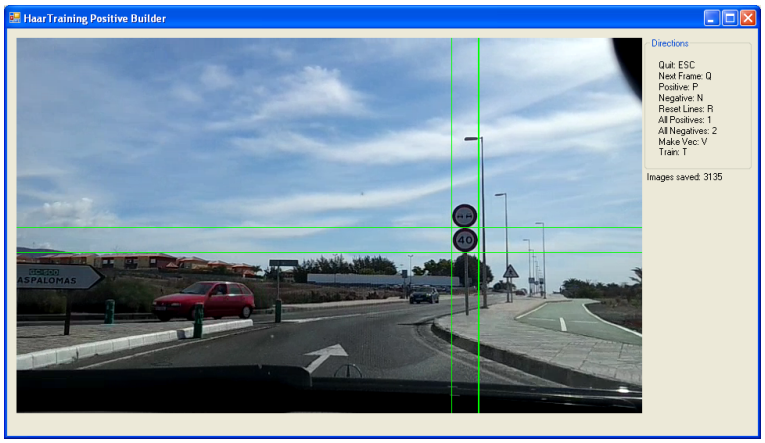


Figura 5.5.: Extracción de una muestra positiva con la aplicación *HaarTraning Positive Builder*.

De igual manera, si no aparece ninguna señal, se puede identificar la imagen como una muestra negativa y crear la estructura definida en el apartado 5.2.1.1. Este caso se muestra en la figura 5.6.

Para esta aplicación se extrajeron más de 3000 muestras positivas y más de 6000 muestras negativas. Estas muestras, así como sus correspondientes ficheros de definición están disponibles en el CD adjunto al proyecto.

5.2.3.2. Entrenamiento del clasificador en cascada

En el proceso de entrenamiento se han empleado las muestras citadas anteriormente. En primer lugar se ha empleado la utilidad `opencv_createsamples` a la que se le han entregado 3000 muestras positivas y su correspondiente fichero descriptor para crear el correspondiente fichero de vectores que contiene las plantillas de las señales de

Figura 5.6.: Extracción de una muestra negativa con la aplicación *HaarTraning Positive Builder*.

tráfico. Para esta aplicación, la utilidad `opencv_createsamples` se ejecutó empleando los siguientes parámetros:

- Fichero de descripción de la colección de imágenes marcadas: *positives.txt*
- Nombre del archivo de salida que contiene las muestras positivas para el entrenamiento: *muestras231.vec*
- Número de muestras positivas a generar: *3000*
- Color de fondo: *0*
- Umbral del color de fondo: *80*
- Inversión de colores: *FALSO*
- Desviación máxima de intensidad de los píxeles en las muestras de primer plano: *40*
- Ángulo de rotación máximo en el eje x: *1.1 radianes*
- Ángulo de rotación máximo en el eje y: *1.1 radianes*
- Ángulo de rotación máximo en el eje z: *1.1 radianes*
- Ancho de las muestras de salida: *24 píxeles*
- Altura de las muestras de salida: *24 píxeles*

En la figura 5.7 se muestra la ejecución de la utilidad `opencv_createsamples` así como una de las señales extraída de uno de las imágenes de muestra.

Una vez obtenido el fichero de vectores se ha procedido a crear el clasificador mediante la aplicación `opencv_traincascade` descrita anteriormente. Se emplearon los siguientes parámetros:

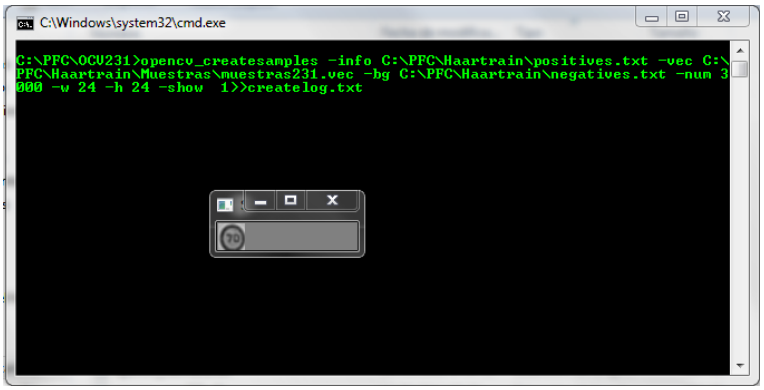


Figura 5.7.: Ejecución de la utilidad `opencv_createsamples`.

- Directorio de almacenamiento el clasificador entrenado: *\CascLBP*
- Fichero con las muestras positivas: *muestras231.vec*
- Fichero de descripción de fondo (muestras negativas): `negatives.txt`
- Número de muestras positivas: *3000*
- Número de muestras negativas: *6000*
- Número de etapas de la cascada que serán entrenadas: *20*
- Tamaño del buffer para los valores característicos precalculados: *1024Mb*
- Tamaño del buffer para los índices precalculados: *1024Mb*
- Tipo de etapas: *BOOST*
- Tipo de característica: *LBP*
- Ancho de las muestras de entrenamiento: *24 píxeles*
- Altura de las muestras de entrenamiento: *24 píxeles*
- Tipo de clasificador potenciado: *GAB - Gentle AdaBoost*
- Tasa de acierto mínima deseada para cada etapa del clasificador: *0.995*
- Tasa de falsa alarma máxima deseada para cada etapa del clasificador: *0.5*
- Peso del recorte: *0.95*
- Profundidad máxima del árbol débil: *1*
- Número máximo de árboles débiles para todas las etapas en cascada: *100*

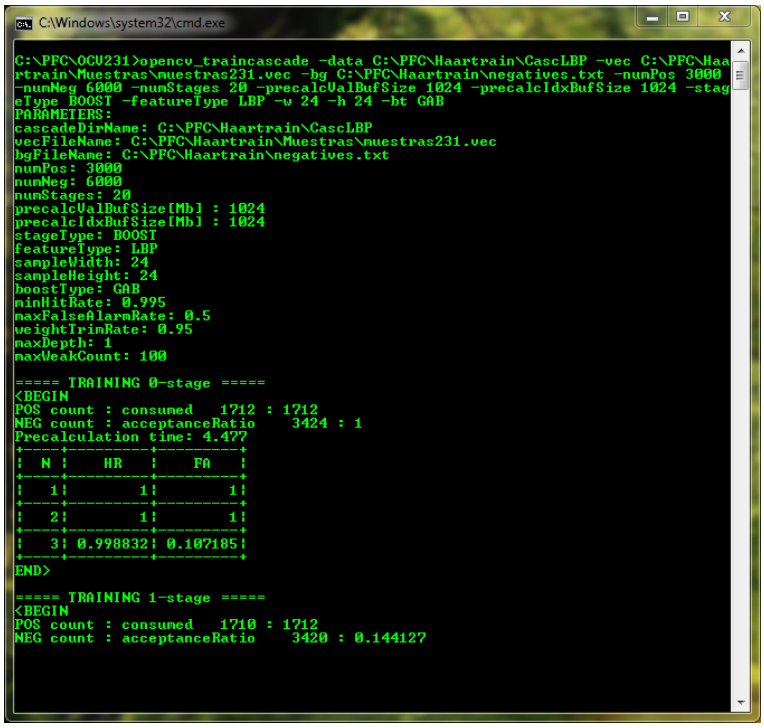


Figura 5.8.: Ejecución de la aplicación `opencv_traincascade`.

La ejecución de la aplicación `opencv_traincascade` con estos parámetros se muestra en la figura 5.8.

En la figura 5.8 también se ve como la aplicación `opencv_traincascade` entrega los resultados de cada etapa después de calcularla. Estos son sus significados:

- `POS count : consumed` : Indica el número de muestras positivas tomadas del vector y el número de ellas que se han empleado.
- `NEG count : acceptanceRatio` : Indica el número de muestras positivas tomadas y el porcentaje de estas que se aceptan.
- `Precalculation time`: Indica el tiempo empleado en precalcular la etapa (en segundos). El proceso de precalculado de la etapa comprende el cálculo de los dos grupos de valores superiores.
- Tabla:
  - `N`: Indica el número de división. De aquí se deduce el número de ramas que tendrá cada etapa ya que $n^o\, de\, ramas = 2^N$.
  - `HR`: Tasa de acierto (*HitRate*).
  - `FA`: Tasa de falsa alarma (*FalseAlarm*).

Como se ve, para cada etapa, la aplicación crea nuevas ramas del árbol de decisión hasta que las tasas de acierto y falsa alarma alcanzan valores dentro de los límites indicados en los parámetros ($\geq 0{,}995$ y $\leq 0{,}5$ respectivamente). En el CD anexo a este proyecto conjuntamente a los demás datos del entrenamiento se encuentra el fichero *trainlog.txt* en el que se pueden ver los resultados de los cálculos de cada una de las etapas.

Para completar el entrenamiento del clasificador con estos parámetros, la aplicación empleó alrededor de 24h en un equipo con procesador de 4 núcleos a 3.4GHZ y 4Gb de memoria RAM.

5.3. Reconocimiento de caracteres

Para este proyecto fin de carrera se implementa una modificación del algoritmo basado en líneas de escaneo que se presentó en el capítulo 2 (página 26). El algoritmo realiza el siguiente proceso teniendo en cuenta que la región candidata que entra al algoritmo tiene siempre un tamaño fijo (30×40 píxeles en este caso) para poder acelerar el procesado:

1. Se traza una línea vertical en el centro de la región candidata y se cuenta en número de intersecciones, cambios de color de blanco a negro o viceversa, que aparecen sobre dicha línea. Para que la región pueda ser considerada como un número, la línea de escaneo sólo puede tener 2 (figura 5.9b), 4 (figura 5.9a) o 6 (figura 5.9c) intersecciones. El valor del recuento se guarda en una variable denominada *#IC*.

2. Se traza una línea en la mediatriz entre el borde izquierdo de la región candidata y la línea de escaneo realizada en el paso anterior y se cuenta el número de intersecciones. En este caso, los posibles valores son 2 (figura 5.9a), 4 (figura 5.9c). El valor del recuento se almacena en una variable denominada *#II*.

3. Se traza una línea en la mediatriz entre el borde derecho de la región candidata y la línea de escaneo realizada en el paso 1 y se cuenta el número de intersecciones. En este caso, los posibles valores son 2 (figura 5.9b), 4 (figura 5.9f). El valor del recuento se almacena en una variable denominada *#ID*.

4. Se determina si existe un lóbulo (área cerrada) en la parte superior del dígito. Para hacerlo, se traza una línea perpendicular a la línea de escaneo central que corte a esta en el punto intermedio entre la 2ª y la 3ª intersección. Se recorre esta línea horizontal desde el punto de corte con la línea vertical hasta el margen

izquierdo. Si a lo largo de esa línea se produce alguna intersección, como ocurre por ejemplo en la figura 5.9h, se considera que el área esta cerrada y, si no se produce ninguna intersección (figura 5.9c), se considera que el área está abierta. El resultado se almacena en la variable booleana *L1C* que indica si el área está cerrada (verdadero) o no (falso).

Hay que tener en cuenta que si el área está cerrada por la izquierda y abierta por la derecha (figura5.9e), el algoritmo considerará igualmente el área como cerrada. Se ha tomado esta determinación para ahorrar variables y cálculos ya que, al analizar el algoritmo, se ha determinado que en el caso de este proyecto fin de carrera, al tener que identificar solamente 10 dígitos y aplicando el resto de variables, el único elemento diferenciador en lo que a los lóbulos se refiere es que el lóbulo superior esté abierto o no por la izquierda.

5. Se determina si las intersecciones primera y última de la línea de escaneo central están cerca de los margenes superior e inferior de la región candidata respectivamente. Se considera que se está cerca del margen si el espacio entre este y la intersección es menor o igual a 3 píxeles. Si esta condición se cumple para ambos puntos, se indicará estableciendo la variable booleana *CCM* a verdadero (figura 5.9a). En el caso de que al menos uno de los puntos no cumpla la condición se marcará la variable a falso (figura 5.9a).

6. Se determina si las intersecciones primera y última de la línea de escaneo izquierda están cerca de los margenes superior e inferior de la región candidata respectivamente. Se considera que se está cerca del margen si el espacio entre este y la intersección es menor o igual a 10 píxeles. Si esta condición se cumple para ambos puntos, se indicará estableciendo la variable booleana *ICM* a verdadero (figura 5.9c). En el caso de que al menos uno de los puntos no cumpla la condición se marcará la variable a falso (figura 5.9d).

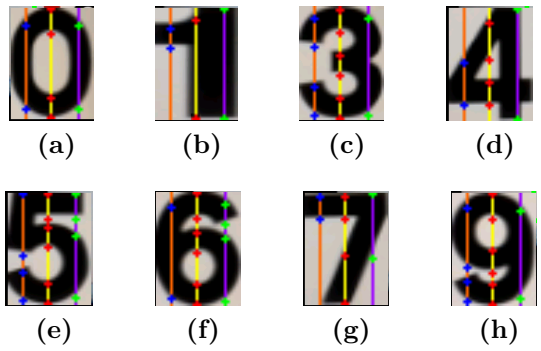


Figura 5.9.: Ejemplos de líneas de escaneo.

7. Se determina si las intersecciones primera y última de la línea de escaneo derecha están cerca de los margenes superior e inferior de la región candidata respecti-

vamente. Se considera que se está cerca del margen si el espacio entre este y la intersección es menor o igual a 10 píxeles. Si esta condición se cumple para ambos puntos, se indicará estableciendo la variable booleana *DCM* a verdadero (figura 5.9c). En el caso de que al menos uno de los puntos no cumpla la condición se marcará la variable a falso (figura 5.9g).

8. Una vez hallados estos valores, se procederá a calcular las probabilidades según la tabla 5.1. Cada variable tiene una importancia (peso) que se agregará al contador de cada dígito si cumple la condición fijada en dicha tabla.

   Estos pesos están especificados en la tabla mediante colores: Rojo (25 %); azul (20 %); verde (10 %); blanco (5 %);

Dígito	#IC	#II	#ID	L1C	CCM	ICM	DCM
0	4	2	2	V	V	V	V
1	2	2	2	F	V	F	V
2	4	4	4	F	V	V	V
3	6	4	2	F	V	V	V
4	4	2	2	V	F	F	V
5	6	4	4	V	V	V	V
6	6	2	4	F	V	V	V
7	4	2	2	F	V	F	F
8	6	2	2	V	V	V	V
9	6	4	2	V	V	V	V

Tabla 5.1.: Valores de los parámetros de las líneas de escaneo para cada dígito.

Por ejemplo, para la figura 5.9c el algoritmo devolverá los valores *#IC=6*, *#II=4*, *#ID=2*, *L1C=falso*, *CCM=verdadero*, *ICM=verdadero* y *DCM=verdadero* y como se puede comprobar en la tabla 5.1, estos valores sólo identifican al número “3”.

Capítulo 6

Análisis Previo y Funcional

6.1. Análisis previo

El objetivo del análisis previo consiste en obtener una visión aproximada de la aplicación a desarrollar, los costes y plazos de desarrollo y los recursos que se van a utilizar. Para ello se han de identificar los bloques generales que componen el proyecto y las relaciones existentes entre ellos.

Por medio de este análisis se puede determinar los objetivos que ha de cumplir el proyecto, as como sus restricciones, demostrando la viabilidad y estudiando su situación actual, sus requisitos y limitaciones. Para ello se han de seguir los siguientes pasos:

- ***Análisis del problema.*** Se han de identificar los objetivos a cumplir y la situación actual del problema.
- ***Solución adoptada.*** Se desarrollara basándose en un diagrama modelo y en los requisitos y restricciones de la aplicación a desarrollar.
- ***Implementación de la solución.*** Se han de identificar los recursos a emplear, así como determinar los plazos y costos necesarios.

6.1.1. Análisis del problema

En los últimos años, tras el auge de los teléfonos móviles inteligentes (*smartphones*), se ha producido un fuerte incremento en la capacidad de los procesadores que incorporan estos dispositivos, convirtiéndolos casi en pequeños ordenadores de bolsillo. Igualmente se ha producido un gran avance en el campo de la visión artificial. Campo en el que algunos investigadores han conseguido crear sistemas de reconocimiento de

señales en tiempo real sobre ordenadores potentes (como por ejemplo en [10] o en [12]) e incluso algunos fabricantes de automóviles han desarrollado sistemas específicos para incorporarlos a sus vehículos (como el galardonado sistema "Opel Eye"[1]). El objetivo de este proyecto fin de carrera está en desarrollar una aplicación sobre la plataforma Android que detecte e identifique las señales de limitación de velocidad definidas en el código de circulación español y comprobar si es posible que el reconocimiento sea en tiempo real en esta plataforma. Esta aplicación está enfocada al usuario final y sólo se iniciará cuando este seleccione el icono correspondiente en la pantalla de aplicaciones de terminal.

6.1.2. Solución adoptada

Para alcanzar el objetivo propuesto, se divide el problema en tres bloques principales tal y como se muestra en la figura 6.1. A continuación, se explica, en detalle, el análisis propuesto en la figura:

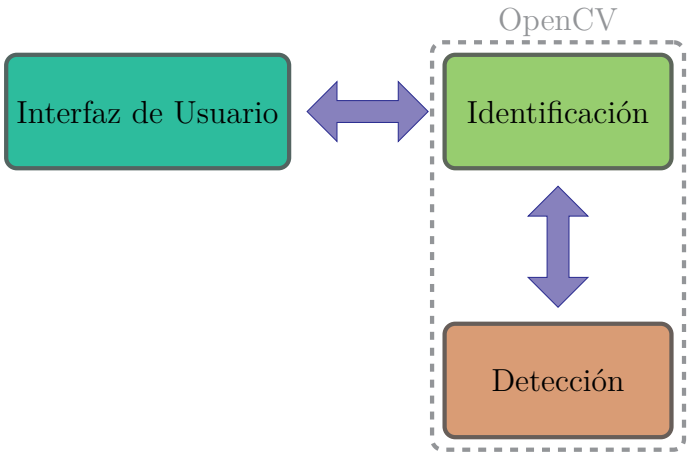


Figura 6.1.: Análisis previo de la aplicación.

- ***Detección.*** Se encarga de detectar las señales de limitación de velocidad y definir el área de la imagen en la que la señal candidata está presente. Se comunica directamente con el bloque de identificación.
- ***Identificación.*** Recibe las regiones candidatas del bloque de detección y realiza todas operaciones sobre la imagen para identificar la señal. Se comunica por un lado con el bloque de detección y por otro con la interfaz de usuario.
- ***Interfaz de usuario.*** Permite la interacción con el usuario y se encarga de gestionar las peticiones de este al sistema. Se comunica directamente el bloque de detección.

[1] http://es.euroncap.com/es/rewards/opel_eye.aspx

6.1.3. Implementación de la solución

Para la implementación de la solución final adoptada se empleará la última versión de la herramienta Android SDK y la biblioteca de visión artificial OpenCV. Debido a las restricciones de OpenCV, las pruebas se realizarán directamente sobre un terminal real y no mediante el emulador del SDK.

La duración estimada para la realización del proyecto es de siete meses, siguiendo la siguiente secuencia:

- Estudio y familiarización con las herramientas de programación de la plataforma Android (mes 1).
- Estudio del API de las librerías OpenCV, haciendo hincapié en las funciones que serán de utilidad en la realización del proyecto (mes 2).
- Estudio y procedimientos para el adiestramiento de máquinas (Machine Learning) que proporciona OpenCV y selección del método a utilizar (meses 3 y 4).
- Toma de imágenes de muestra y tratamiento para su posterior uso en el proceso de aprendizaje (mes 4).
- Inicio de los procesos de aprendizaje (a partir del 5º mes en paralelo al resto de actividades).
- Análisis previo del proyecto donde se especifique a nivel de bloques generales, cada una de las partes que lo componen. Análisis de los parámetros de configuración del sistema y definición de la interfaz de usuario (mes 5).
- Análisis funcional del proyecto donde se especifique, partiendo de los bloques generales, cada una de las partes que lo componen, desarrollando cada bloque general en sub-bloques (mes 5).
- Realización del código de la aplicación y análisis orgánico del proyecto donde se especifique, partiendo de los bloques del análisis funcional, cada una de las partes que lo componen, desarrollando cada bloque en sus correspondientes clases, atributos y métodos. Utilizar UML para describir cada clase desarrollada (meses 6 al 8).
- Finalización de la memoria (mes 9).

6.2. Análisis funcional

El objetivo del análisis funcional es profundizar en el modelo de bloques visto en el análisis previo. Para ello se tratara cada uno de los bloques que componen la aplicación por separado, profundizando en los sub-bloques que lo constituyen. Los distintos bloques del análisis previo se subdividen tal y como se indica a continuación.

6.2.1. Bloque Interfaz de usuario

Bloque que se encarga de la interacción con el usuario. Se comunica con el bloque *Identificación* para indicar que etapa de la identificación se quiere mostrar y recibir el resultado para mostrarlo. Este bloque de divide en dos sub-bloques, tal y como se muestra en la figura 6.2, que pasaran a describirse a continuación:

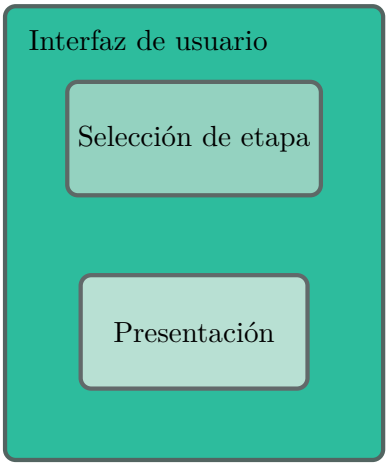


Figura 6.2.: Análisis funcional del bloque *Interfaz de usuario*.

- **Sub-bloque Selección de etapa**: sub-bloque que envía al bloque *Identificación* la solicitud del usuario respecto a que etapa de la detección desea visualizar.
- **Sub-bloque Presentación:** sub-bloque que se encarga de mostrar por pantalla la previsualización de la cámara así como el resultado de la etapa seleccionada por el usuario. Asimismo, mostrará la última señal detectada.

6.2.2. Bloque Detección

Este bloque es el encargado de delimitar la región de la imagen. La fase de detección es la más exigente en cuanto al coste computacional. Por eso se aísla esta parte quedando sólo con un sub-bloque, como se ve en la figura 6.3. Este sub-bloque se comentará a continuación. Sólo se comunica con el bloque *Identificación*.

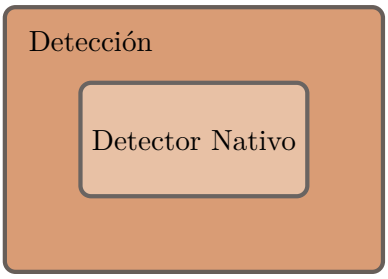


Figura 6.3.: Análisis funcional del bloque *Detección*.

- **Sub-bloque Detector Nativo**: Este sub-bloque es el encargado de explorar los frames de entrada de la cámara en busca posibles señales de limitación de velocidad y de extraer las regiones candidatas para pasárselas al bloque *Identificación*.

6.2.3. Bloque Identificación

Este bloque es el que se encarga de identificar las regiones candidatas como señales de limitación de velocidad y de "leer" la velocidad concreta. Se comunica con el bloque *Interfaz de usuario* para recibir la etapa a calcular seleccionada por el usuario y para devolverle los resultados. Igualmente, activa el bloque *Detección* y recibe de el las regiones de interés. Como se muestra en la figura 6.4, consta de dos sub-bloques:

Figura 6.4.: Análisis funcional del bloque *Identificación*.

- **Sub-bloque Extracción**: Este sub-bloque será el encargado de aislar los posibles dígitos de la señal de limitación de velocidad.

- **Sub-bloque OCR**: Este sub-bloque recibirá los posibles dígitos extraídos por el sub-bloque anterior y les realizará un reconocimiento óptico de caracteres con el fin de identificar si se trata de un número o no y, en el caso de serlo concretar que señal de limitación de velocidad es.

Capítulo 7

Análisis Orgánico

7.1. Introducción

El análisis orgánico es la fase de diseño de la aplicación en la cual se traducen, a lenguaje comprensible por el ordenador, las decisiones tomadas en las fases anteriores de diseño. De este modo, en este punto del proceso, se trabajara directamente con el lenguaje de programación seleccionado y con conceptos más especializados y técnicos.

Asimismo, en este ultimo análisis se definirá con un mayor grado de profundidad el funcionamiento global del sistema, pasándose a detallar, a nivel de clases de la aplicación, cada uno de los bloques definidos en los análisis anteriores, profundizando en su funcionamiento interno y describiendo cada una de las clases que lo conforman. Así, en este capítulo se especificará cada una de las clases que componen la aplicación utilizando colores (para distinguir los bloques a los que pertenecen) y el diagrama de herencia (de que clases se deriva y que interfaces se implementan). A continuación, se presenta para cada clase individual, su representación en formato UML, especificando sus atributos y métodos (el diagrama de clases de toda la aplicación se puede encontrar en el anexo E). Debido a limitaciones de OpenCV, que requiere que todas las clases que lo utilizan estén en un mismo paquete, la aplicación constará únicamente de un paquete. Aún así, para facilitar la comprensión, las representaciones UML de las clases se colorearán según la clasificación que se hizo en el análisis previo (figura 6.1 en la página 144).

7.2. Análisis del sistema

Antes de analizar cada clase realizada, y para facilitar la comprensión del trabajo en su conjunto, se presenta una visión global del sistema a través de la figura 7.1, en

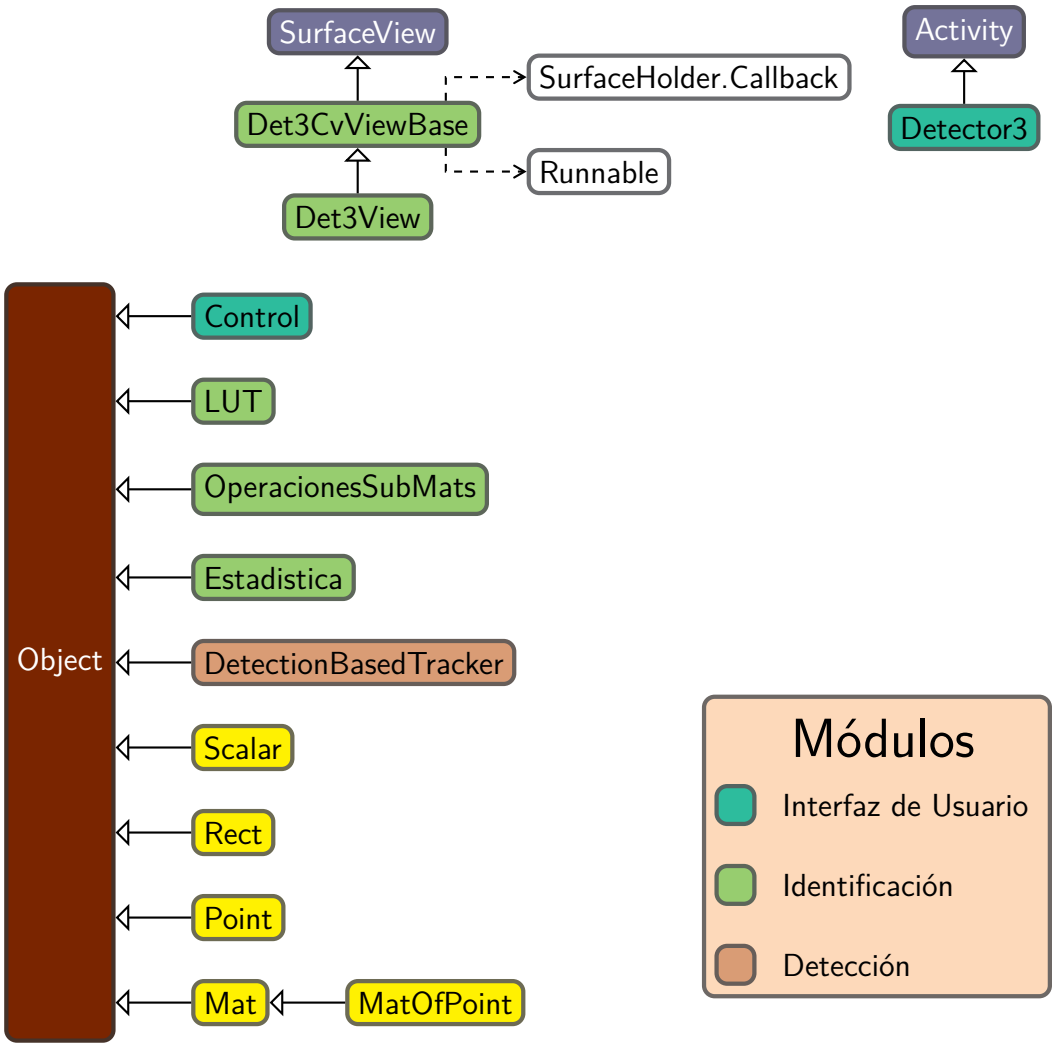


Figura 7.1.: Diagrama de herencia de clases de la aplicación.

la que se muestra el diagrama de herencia de clases (presentándose de que clase se deriva y que interfaces implementan cada una de las clases de la aplicación). En esta figura puede observarse como se han utilizado una serie de colores para cada una de las clases, que define el módulo al que pertenece (utilizándose un mismo color para todas las clases relacionadas con un determinado bloque). En esta figura se pueden ver las diferentes clases e interfaces de la aplicación para el terminal móvil. Ademas, todas las clases e interfaces propias de Android vendrán definidas en color azul y las clases de OpenCV por el color amarillo.

A continuación se presentan las clases que conforman la aplicación, describiéndolas según el módulo al que pertenecen. Indicar que para cada clase se presenta su esquema UML, donde aparecerán sus atributos y métodos. La nomenclatura usada es la siguiente: el símbolo + significa publico, el símbolo - significa privado, el símbolo # significa protegido y un atributo o método subrayado significa que es estático. Los métodos abstractos se representarán en cursiva.

7.3. Análisis de la aplicación

En este apartado se presentan cada una de las clases desarrolladas en la aplicación, ayudados por sus esquemas UML. De este modo, se definirá toda la jerarquía de clases de la aplicación del terminal móvil.

7.3.1. Módulo Interfaz de Usuario

Este módulo contiene las clases que se encargan de interactuar con el usuario, bien para que este decida que etapa de la detección desea visualizar, o bien para mostrarle a este los resultados. Este módulo lo componen las siguientes clases: *Detector3* y *Control*.

7.3.1.1. Clase *Detector3*

Esta clase deriva de *Activity* y se encarga de inicializar la aplicación. En la figura 7.2 se muestra la representación UML de la clase.

Atributos:

- `public static final int OP_NORMAL = 0`: Constante que define opción de mostrar solamente la etapa de detección.
- `public static final int OP_VALUE = 1`: Constante que define opción de mostrar el canal *Value* de HSV.
- `public static final int OP_UMBRAL = 2`: Constante que define opción de mostrar las partes en negro de la imagen.
- `public static final int OP_CONTORNO = 3`: Constante que define opción de mostrar los contornos negros que se han localizado.
- `public static final int OP_NUMEROS = 4`: Constante que define opción de mostrar los posibles dígitos.
- `public static final int OP_ORDENADOS = 5`: Constante que define opción de mostrar los posibles dígitos ordenados de izquierda a derecha.
- `public static final int OP_CRUCES = 6`: Constante que define opción de mostrar los dígitos con sus respectivas líneas de escaneo.

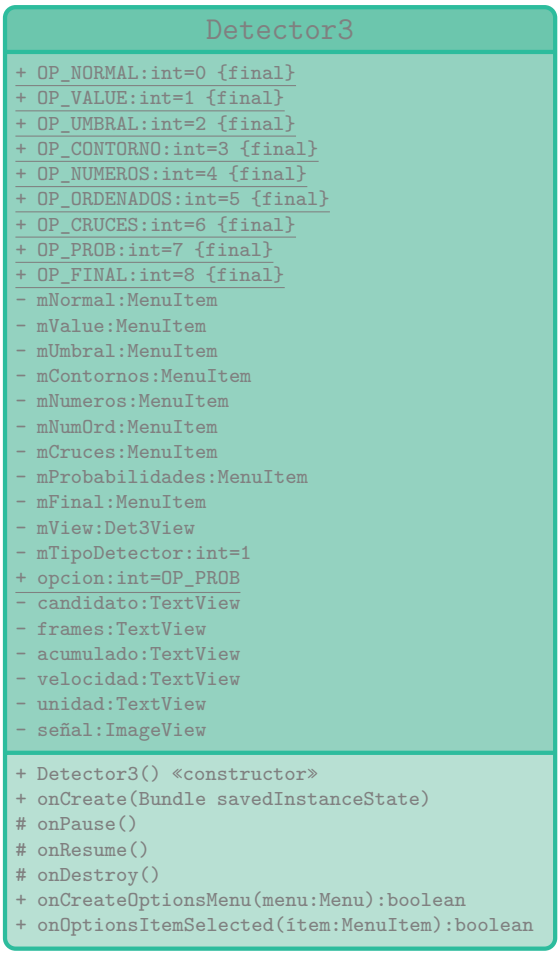


Figura 7.2.: Representación UML de la clase *Detector3*.

- **`public static final int OP_PROB = 7`**: Constante que define opción de mostrar el resultado de la identificación.
- **`public static final int OP_FINAL = 8`**: Constante que define opción de mostrar un velocímetro en lugar de la previsualización de la cámara.
- **`private MenuItem mNormal`**: Objeto que identifica el ítem del menú para mostrar solamente la etapa de detección.
- **`private MenuItem mValue`**: Objeto que identifica el ítem del menú para mostrar el canal *Value* de HSV.
- **`private MenuItem mUmbral`**: Objeto que identifica el ítem del menú para mostrar las partes en negro de la imagen.
- **`private MenuItem mContornos`**: Objeto que identifica el ítem del menú para mostrar los contornos negros que se han localizado.
- **`private MenuItem mNumeros`**: Objeto que identifica el ítem del menú para mostrar los posibles dígitos.

- **`private MenuItem mNumOrd`**: Objeto que identifica el ítem del menú para mostrar los posibles dígitos ordenados de izquierda a derecha.
- **`private MenuItem mCruces`**: Objeto que identifica el ítem del menú para mostrar los dígitos con sus respectivas líneas de escaneo.
- **`private MenuItem mProbabilidades:`** Objeto que identifica el ítem del menú para mostrar el resultado de la identificación.
- **`private MenuItem mFinal`**: Objeto que identifica el ítem del menú para mostrar un velocímetro en lugar de la previsualización de la cámara.
- **`private Det3View mView`**: Objeto de la clase que realiza la identificación.
- **`private int mTipoDetector = 1`**: Define que tipo de detector se va a emplear.
- **`public static int opcion = OP_PROB`**: Almacena la opción que ha seleccionado el usuario. La aplicación se inicia por defecto con la opción de mostrar el resultado de la identificación.
- **`private TextView candidato`**: Objeto que muestra el posible resultado.
- **`private TextView frames:`** Objeto que muestra el número de frames de una iteración.
- **`private TextView acumulado`**: Objeto que muestra cuantas identificaciones se han dado en una iteración.
- **`private TextView velocidad`**: Objeto que muestra la velocidad del velocímetro.
- **`private TextView unidad:`** Objeto que muestra la unidad de velocidad (Km/h).
- **`private ImageView señal`**: Objeto que muestra la imagen de la señal identificada.

Métodos:

- **`public Detector3()`**: Método constructor de la clase.
- **`public void onCreate(Bundle savedInstanceState)`**: Método que se encarga de presentar la interfaz de la aplicación. Inicializa los distintos objetos *TextView* de la aplicación y los objetos *ImageView* y *Det3View*. Además, evita que la pantalla se apague.

- **`protected void onDestroy()`**: Método que destruye todos los procesos relacionados con la aplicación.
- **`protected void onPause()`**: Apaga la cámara cuando la aplicación pasa a segundo plano.
- **`protected void onResume()`**: Reinicia la cámara cuando la aplicación vuelve a primer plano.
- **`public boolean onCreateOptionsMenu(Menu menu)`**: Añade las distintas opciones al menú de la aplicación.
- **`public boolean onOptionsItemSelected(MenuItem ítem)`**: Marca la opción del menú seleccionada por el usuario.

7.3.1.2. Clase *Control*

Esta clase es la encargada de actualizar la interfaz de usuario mostrando los diferentes elementos según la opción que haya seleccionado el usuario. En la figura 7.3 se muestra la representación UML de la clase.

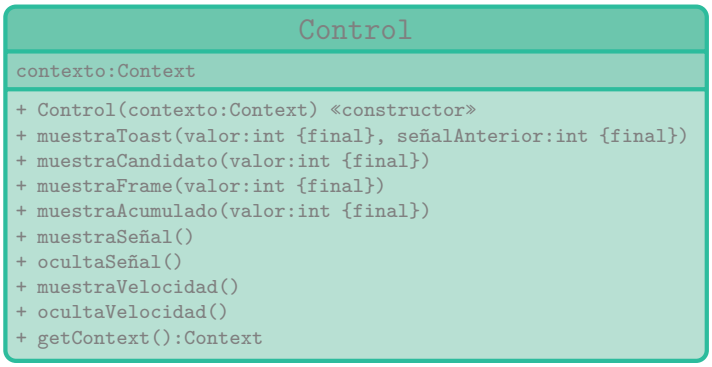


Figura 7.3.: Representación UML de la clase *Control*.

Atributos:

- **`Context contexto`**: Almacena el contexto de la aplicación.

Métodos:

- **`public Control(Context contexto)`**: Método constructor de la clase.

- **`public void muestraToast(final int valor,final int señalAnterior)`**: Muestra un *Toast*[1] de Android con la señal identificada si no es la misma que la anterior señal identificada. Igualmente actualiza la imagen de la señal de la interfaz de usuario.

- **`public void muestraCandidato(final int valor)`**: Actualiza el *TextView* que muestra el posible valor de la señal.

- **`public void muestraFrame(final int valor)`**: Actualiza el *TextView* que muestra el frame de la iteración actual.

- **`public void muestraAcumulado(final int valor)`**: Actualiza el *TextView* que muestra las veces que se ha identificado una señal en la iteración.

- **`public void muestraSeñal()`**: Hace que el *ImageView* que muestra la última señal identificada se muestre en pantalla.

- **`public void ocultaSeñal()`**: Hace que el *ImageView* que muestra la última señal identificada desaparezca de la pantalla.

- **`public void muestraVelocidad()`**: Hace que los *TextViews* que muestran la velocidad y la unidad de medida se muestren por pantalla

- **`public void ocultaVelocidad()`**: Hace que los *TextViews* que muestran la velocidad y la unidad de medida desaparezcan de la pantalla.

- **`public Context getContext()`**: Devuelve el contexto de la aplicación.

7.3.2. Módulo Identificación

7.3.2.1. Clase *Det3CvViewBase*

Clase abstracta que deriva de *SurfaceView*. Implementa las interfaces *SurfaceHolder.Callback* y *Runnable*. La interfaz *SurfaceHolder.Callback* se implementa para recibir información acerca de los cambios en la superficie y se fija con el método *SurfaceHolder.addCallback*. La interfaz Runnable representa un comando que puede ser ejecutado. En la figura 7.4 se muestra la representación UML de la clase.

[1] Ventana emergente que puede mostrar texto y/o imágenes y que permanece en pantalla un tiempo determinado.

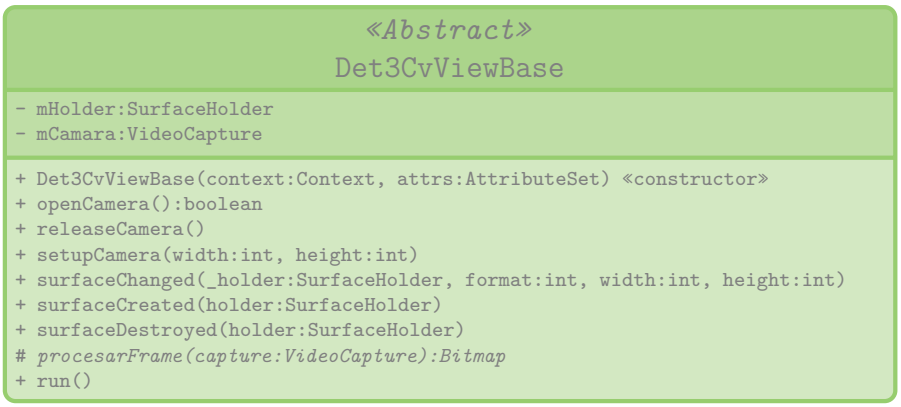


Figura 7.4.: Representación UML de la clase *Det3CvViewBase*.

Atributos:

- **`private SurfaceHolder mHolder`**: Objeto que gestiona la superficie de previsualización de la pantalla.
- **`private VideoCapture mCamara`**: Objeto de la clase VideoCapture de la biblioteca OpenCV. Se encarga de la configuración y captura de imágenes desde la cámara.

Métodos:

- **`public Det3CvViewBase(Context context, AttributeSet attrs)`**: Constructor de la clase, Inicializa el soporte (*Holder*) de la superficie de previsualización (*SurfaceView*).
- **`public boolean openCamera()`**: Crea un objeto mCamara de la clase VideoCapture. Indica si la acción se pudo realizar (*true*) o no (*false*).
- **`public void releaseCamera()`**: Destruye el objeto mCamara de la clase VideoCapture.
- **`public void setupCamera(int width, int height)`**: Fija el tamaño al que la cámara tomará las imágenes. Por defecto y por motivos de rendimiento, se fijará este tamaño a 320×240 píxeles.
- **`public void surfaceChanged(SurfaceHolder _holder, int format, int width, int height)`**: Se llama inmediatamente después de que se han realizado cambios estructurales (formato o tamaño) a la superficie.
- **`public void surfaceCreated(SurfaceHolder holder)`**: Se llama inmediatamente después de que la superficie se crea por primera vez.

- `public void surfaceDestroyed(SurfaceHolder holder)`: Se llama justo antes de que la superficie sea destruida. Invoca al método *releaseCamera()*.
- `protected abstract Bitmap procesarFrame(VideoCapture capture)`: Declaración del método que será definido en las clases hijas.
- `public void run()`: Inicia la ejecución de la parte activa del código de la clase.

7.3.2.2. Clase *Det3View*

Esta clase hereda de la clase *Det3CvViewBase*. Se trata de la clase principal de la aplicación ya que es la que aporta toda la funcionalidad y la que crea y emplea todos los objetos y métodos para la detección e identificación de las señales de limitación de velocidad. En la figura 7.5 se muestra la representación UML de la clase.

Atributos:

- `public Mat mRgba`: Objeto de la clase *Mat* de OpenCV (en la página 42) que contiene el frame de vídeo en color captado por la cámara.
- `private Mat mGris`: Objeto de la clase *Mat* de OpenCV que contiene el frame de vídeo en escala de grises captado por la cámara.
- `private File mCascadeFile`: Objeto que representa al fichero XML que almacena el clasificador entrenado en el proceso de aprendizaje automático.
- `private CascadeClassifier mJavaDetector`: Objeto que carga el clasificador entrenado.
- `private DetectionBasedTracker mNativeDetector`: Objeto que representa al detector implementado de forma nativa.
- `private static final Scalar COLOR_VERDE = new Scalar(0, 255, 0, 255)`: Objeto de la clase *Scalar* de OpenCV (en la página 47) que define el color verde.
- `private static final Scalar COLOR_AMARILLO = new Scalar(255, 255, 0, 255)`: Objeto de la clase *Scalar* de OpenCV que define el color amarillo.
- `private static final Scalar COLOR_ROJO = new Scalar(255, 0, 0, 255)`: Objeto de la clase *Scalar* de OpenCV que define el color rojo.

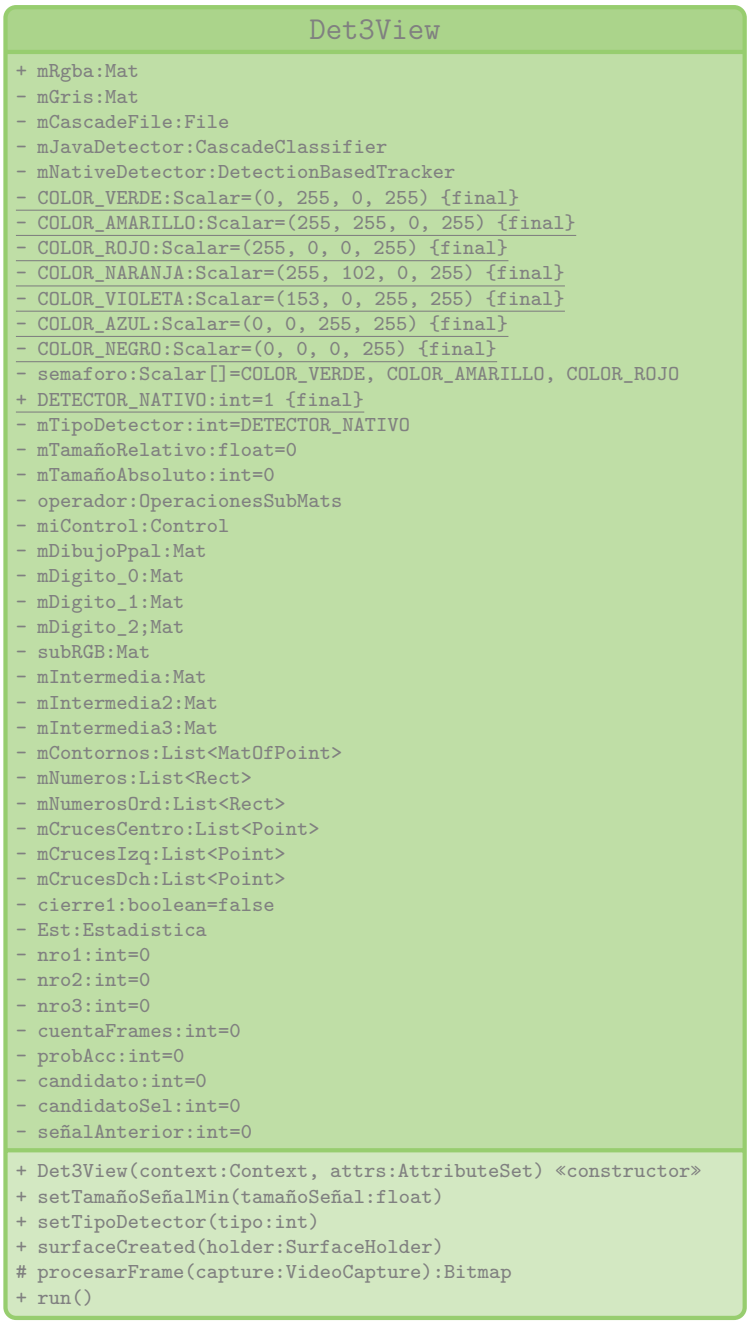


Figura 7.5.: Representación UML de la clase *Det3View*.

- **`private static final Scalar COLOR_NARANJA = new Scalar(255, 102, 0, 255)`**: Objeto de la clase *Scalar* de OpenCV que define el color naranja.
- **`private static final Scalar COLOR_VIOLETA = new Scalar(153, 0, 255, 255)`**: Objeto de la clase *Scalar* de OpenCV que define el color violeta.
- **`private static final Scalar COLOR_AZUL = new Scalar(0, 0, 255, 255)`**: Objeto de la clase *Scalar* de OpenCV que define el color azul.
- **`private static final Scalar COLOR_NEGRO = new Scalar(0, 0, 0, 255)`**: Objeto de la clase *Scalar* de OpenCV que define el color negro.

- **`private Scalar[] semaforo = {COLOR_VERDE, COLOR_AMARILLO, COLOR_ROJO}`**: Vector de objetos de tipo *Scalar* que se emplea para mostrar el orden del los posibles dígitos.
- **`public static final int DETECTOR_NATIVO = 1`**: Constante que representa al detector nativo.
- **`private int mTipoDetector = DETECTOR_NATIVO`**: Indica que tipo de detector se va a emplear. En el caso de la aplicación será siempre el detector nativo.
- **`private float mTamañoRelativo = 0`**: Indica el tamaño mínimo que tiene que tener la señal respecto a la imagen completa para ser detectada.
- **`private int mTamañoAbsoluto = 0`**: Indica el tamaño absoluto de la señal. Se calcula en función del tamaño relativo.
- **`private OperacionesSubMats operador`**: Objeto que contiene los métodos para procesar los diferentes *frames*.
- **`private Control miControl`**: Objeto para interactuar con la interfaz de usuario.
- **`private Mat mDibujoPpal`**: Objeto de tipo *Mat* donde se muestran los resultados parciales del proceso de identificación.
- **`private Mat mDigito_0`**: Objeto de tipo *Mat* donde se muestra el primer dígito de la señal con las líneas de escaneo.
- **`private Mat mDigito_1`**: Objeto de tipo *Mat* donde se muestra el segundo dígito de la señal con las líneas de escaneo.
- **`private Mat mDigito_2`**: Objeto de tipo *Mat* donde se muestra el tercer dígito de la señal con las líneas de escaneo.
- **`private Mat subRGB`**: Objeto de tipo *Mat* que contiene una parte del frame original (*mRgba*) que ha sido extraída según los resultados entregados por el objeto *mNativeDetector*.
- **`private Mat mIntermedia`**: Objeto en el que se realizarán las transformaciones de la matriz subMat. Algunos métodos de OpenCV modifican al contenido de la matriz de entrada por lo que es necesario utilizar matrices intermedias.
- **`private Mat mIntermedia2`**: Igual que el anterior, se crea otra matriz intermedia en el caso de la identificación del carácter, en este caso para el segundo dígito.

- **`private Mat mIntermedia3`**: Igual que el anterior, en este caso para el tercer dígito.
- **`private List<MatOfPoint> mContornos`**: Lista con objetos de tipo *MatOfPoint* (en la página 44) que almacena la información de los distintos contornos localizados en la matriz de entrada. Cada objeto *MatOfPoint* almacena un contorno.
- **`private List<Rect> mNumeros`**: Lista con objetos de tipo *Rect* (en la página 45) que contiene las regiones de interés de los distintos contornos almacenados en *mContornos*. Cada objeto *Rect* se corresponde con un *MatOfPoint* de *mContornos*. Sólo se tomarán los contornos que están situados en torno a la bisectriz horizontal de la imagen ya que es la zona donde se encuentran los números en las señales de limitación de velocidad.
- **`private List<Rect> mNumerosOrd`**: La lista anterior pero con los objetos *Rect* ordenados según las regiones estén situadas de izquierda a derecha en la imagen (el primero la más a la izquierda y el último la más a la derecha).
- **`private List<Point> mCrucesCentro`**: Lista de objetos tipo *Point* (en la página 44) que almacena los puntos de corte de la línea de escaneo central en el reconocimiento de caracteres.
- **`private List<Point> mCrucesIzq`**: Lista de objetos tipo *Point* que almacena los puntos de corte de la línea de escaneo izquierda en el reconocimiento de caracteres.
- **`private List<Point> mCrucesDch`**: Lista de objetos tipo *Point* que almacena los puntos de corte de la línea de escaneo derecha en el reconocimiento de caracteres.
- **`private boolean cierre1 = false`**: Indica si la parte superior del dígito es un área cerrada.
- **`private Estadistica Est = new Estadistica()`**: Objeto que contiene los atributos y métodos para calcular la probabilidad acumulada de un dígito.
- **`private int nro1 = 0`**: Indica el valor estimado del primer dígito.
- **`private int nro2 = 0`**: Indica el valor estimado del segundo dígito.
- **`private int nro3 = 0`**: Indica el valor estimado del tercer dígito.
- **`private int cuentaFrames = 0`**: Indica los frames que se han procesado en la iteración.

- **`private int probAcc = 0`**: Indica en cuantos frames de la iteración se ha identificado un valor concreto.

- **`private int candidato = 0`**: El posible valor de velocidad de la señal.

- **`private int candidatoSel = 0`**: El valor identificado. Se tiene cuando *candidato* ha alcanzado al menos un número determinado de *probAcc* en una iteración.

- **`private int señalAnterior = 0`**: Almacena el valor de la última señal identificada.

Métodos:

- **`public Det3View(Context context, AttributeSet attrs)`**: Método constructor de la clase. Cuando se invoca se encarga de crear el objeto *mCascadeFile* a partir del fichero XML creado durante el proceso automático. Igualmente crea los objetos *mJavaDetector* y *mNativeDetector* pasandoles la información contenida en *mCascadeFile*.

- **`public void setTamañoSeñalMin(float tamañoSeñal)`**: Fija el tamaño mínimo que debe tener la señal en la imagen para ser detectada.

- **`public void setTipoDetector(int tipo)`**: Fija el detector que se va a emplear.

- **`public void surfaceCreated(SurfaceHolder holder)`**: Inicializa los objetos de tipo *Mat* que se emplearán en la aplicación. Es conveniente asignar a los objetos tipo Mat un espacio de memoria desde el principio y liberarlo al salir de la aplicación ya que la asignación y liberalización de memoria para este tipo de objetos supone una gran carga para el procesador.

- **`protected Bitmap procesarFrame(VideoCapture captura)`**: Es el método que aglutina la operativa principal de la aplicación. Se encarga de gestionar los distintos objetos y de realizar las distintas operaciones sobre el *frame* actual según la opción de visualización seleccionada por el usuario. Igualmente transfiere la información a mostrar por la interfaz de usuario a través de un objeto de la clase *Control*. La operativa de este método se muestra en el diagrama presente en el anexo F.

- **`public void run()`**: Vacía y libera de la memoria los objetos tipo *Mat*.

7.3.2.3. Clase *OperacionesSubMats*

Esta clase contiene los atributos y métodos necesarios para aislar los distintos dígitos de la señal de limitación de velocidad. En la figura 7.6 se muestra la representación UML de la clase.

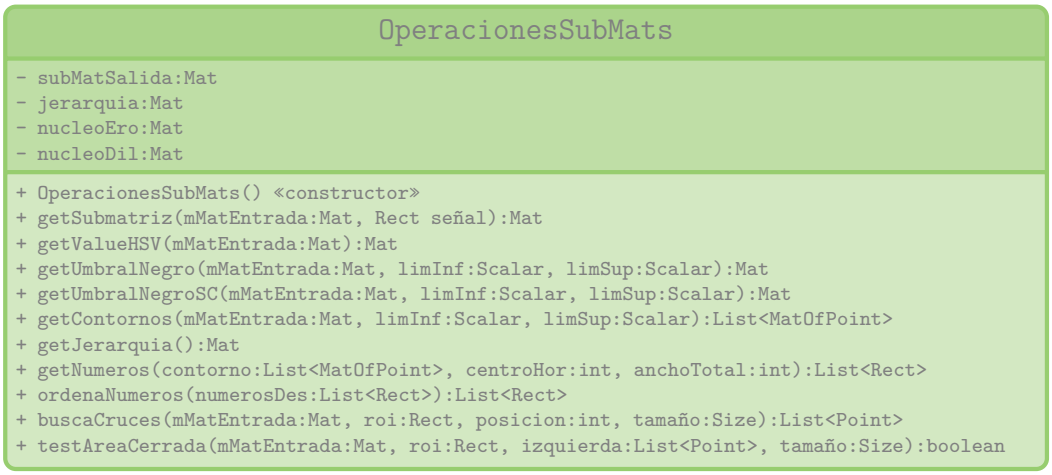


Figura 7.6.: Representación UML de la clase *OperacionesSubMats*.

Atributos:

- **`private Mat subMatSalida = new Mat()`**: Objeto sobre el que se harán las operaciones y que es el que devolverán algunos métodos como resultado.

- **`private Mat jerarquia = new Mat()`**: Matriz de salida opcional del método *findContours* [96] de OpenCV que contiene información acerca de la topología de la imagen.

- **`private Mat nucleoEro = Imgproc.getStructuringElement(Imgproc.MORPH_ERODE, new Size(2,2), new Point(1,1))`**: Matriz de tamaño 2×2 que se utiliza en el proceso de erosión [97].

- **`private Mat nucleoDil = Imgproc.getStructuringElement(Imgproc.MORPH_DILATE, new Size(3,3), new Point(1,1))`**: Matriz de tamaño 3×3 que se utiliza en el proceso de dilatación [97].

  Para obtener más información acerca del funcionamiento del método *getStructuringElement* consulte la referencia de la API de OpenCV [98].

Métodos:

- **`public OperacionesSubMats()`**: Método constructor de la clase.
- **`public Mat getSubmatriz(Mat mMatEntrada, Rect señal)`**: Entrega la submatriz que contiene a la posible señal. Controla que esa submatriz esté dentro de la imagen.
- **`public Mat getValueHSV(Mat mMatEntrada)`**: Convierte la imagen al espacio de color HSV y extrae el canal *Value* que es el que indica la intensidad de la imagen. La matriz de salida se vuelve a convertir al espacio de color RGB para poder mostrarla por pantalla.
- **`public Mat getUmbralNegro(Mat mMatEntrada, Scalar limInf, Scalar limSup)`**: Convierte la imagen al espacio de color HSV y extrae el canal Value. A partir de este canal filtra la imagen para extraer las regiones de color negro (los dígitos de las señales son de ese color) obteniendo una imagen binaria en la que las zonas donde está presente el color negro se representan con un 0 y el resto con un 1. A esta imagen se le aplican filtrados de erosión y dilatación para suavizar los bordes. La imagen de salida se vuelve a convertir al espacio de color RGB para poder mostrarla por pantalla.
- **`public Mat getUmbralNegroSC(Mat mMatEntrada, Scalar limInf, Scalar limSup)`**: Realiza la misma operación que el método anterior pero, en este caso, no vuelve a convertir la imagen al espacio de color RGB sino que devuelve la imagen binaria para que sea empleada en otros procesos de la identificación.
- **`public List<MatOfPoint> getContornos(Mat mMatEntrada, Scalar limInf, Scalar limSup)`**: A partir de la imagen binaria, halla todos los contornos (áreas en blanco) y devuelve una lista en la que cada elemento contiene los puntos claves que describen al contorno.
- **`public Mat getJerarquia()`**: El método que localiza los contornos crea una matriz auxiliar que contiene la jerarquia[2] de los contornos. Este método permite obtener dicha jerarquia a través del objeto de la clase.
- **`public List<Rect> getNumeros(List<MatOfPoint> contorno, int centroHor, int anchoTotal)`**: Comprueba que los contornos están centrados horizontalmente (los dígitos en las señales lo están) y a partir de la información de los puntos clave de cada contorno, genera rectángulos horizontales que contiene

[2]La jerarquia marca todos los contornos exteriores como padres e indica si dentro de estos hay otros contornos marcándolos como hijos.

a cada uno de los contornos centrados horizontalmente (los posibles dígitos de la señal).

- **`public List<Rect> ordenaNumeros(List<Rect> numerosDes)`**: La lista de rectángulos generada por el método anterior no tiene porque estar necesariamente ordenada. Este método se encarga de que los posibles dígitos estén en orden. Es decir, que el primer rectángulo corresponda al contorno de la izquierda, que el segundo rectángulo corresponda con el contorno central (en el caso de velocidades de 3 dígitos) o el de la derecha (en el caso de señales con 2 dígitos) y que el tercer rectángulo corresponda al contorno de la derecha (únicamente en señales con tres dígitos).

- **`public List<Point> buscaCruces(Mat mMatEntrada, Rect roi, int posicion, Size tamaño)`**: Devuelve los puntos de cruce de las líneas de escaneo. Este método ha de llamarse tres veces para cada dígito, una por cada línea de escaneo que se realice (central, izquierda y derecha).

- **`public boolean testAreaCerrada(Mat mMatEntrada, Rect roi, List<Point> izquierda, Size tamaño)`**: Comprueba si el lóbulo superior del dígito está cerrado, como ocurre por ejemplo con el 9, o abierto, como en el caso del 3.

7.3.2.4. Clase *LUT*

Esta clase representa a la tabla de búsqueda para el reconocimiento de los caracteres (números en este caso). En la figura 7.7 se muestra la representación UML de la clase.

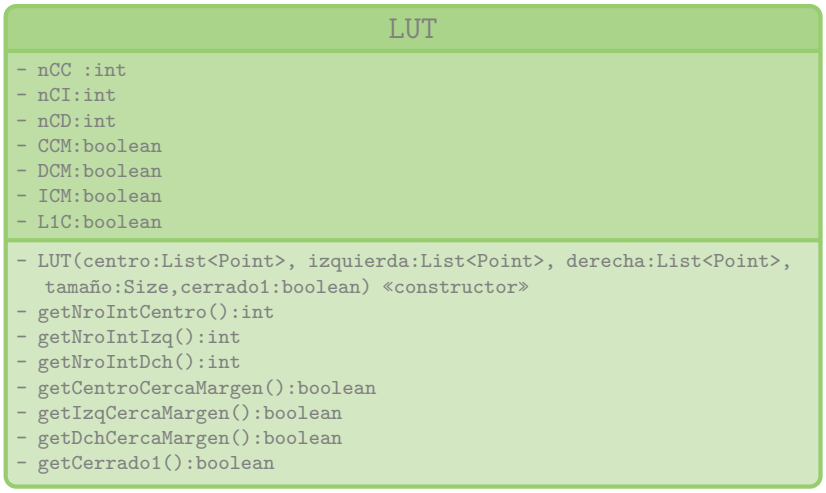


Figura 7.7.: Representación UML de la clase *LUT*.

Atributos:

- **`private int nCC`**: Número de intersecciones en la línea de escaneo central.
- **`private int nCI`**: Número de intersecciones en la línea de escaneo izquierda.
- **`private int nCD`**: Número de intersecciones en la línea de escaneo derecha.
- **`private boolean CCM`**: Indica si las intersecciones primera y última de la línea de escaneo central están cerca de los margenes superior e inferior respectivamente.
- **`private boolean DCM`**: Indica si las intersecciones primera y última de la línea de escaneo derecha están cerca de los margenes superior e inferior respectivamente.
- **`private boolean ICM`**: Indica si las intersecciones primera y última de la línea de escaneo izquierda están cerca de los margenes superior e inferior respectivamente.
- **`private boolean L1C`**: Indica si el primer lóbulo (centro[1]-centro[2]) está cerrado.

Métodos:

- **`public LUT(List<Point> centro, List<Point> izquierda, List<Point> derecha, Size tamaño, boolean cerrado1)`**: Método constructor de la clase. Cuando se crea un objeto *LUT*, el constructor "rellena" la tabla en función de las líneas de escaneo centro, izquierda y derecha y de si el lóbulo superior está abierto o no.
- **`public int getNroIntCentro()`**: Entrega el número de intersecciones en la línea de escaneo central.
- **`public int getNroIntIzq()`**: Entrega el número de intersecciones en la línea de escaneo izquierda.
- **`public int getNroIntDch()`**: Entrega el número de intersecciones en la línea de escaneo derecha.
- **`public boolean getCentroCercaMargen()`**: Entrega el valor del atributo *CCM*.
- **`public boolean getIzqCercaMargen()`**: Entrega el valor del atributo *DCM*.
- **`public boolean getDchCercaMargen()`**: Entrega el valor del atributo *ICM*.
- **`public boolean getCerrado1()`**: Entrega el valor del atributo *L1C*.

7.3.2.5. Clase *Estadistica*

Esta clase es la encargada de calcular las probabilidades de cada dígito en función de los valores de la tabla de búsqueda. En la figura 7.8 se muestra la representación UML de la clase.

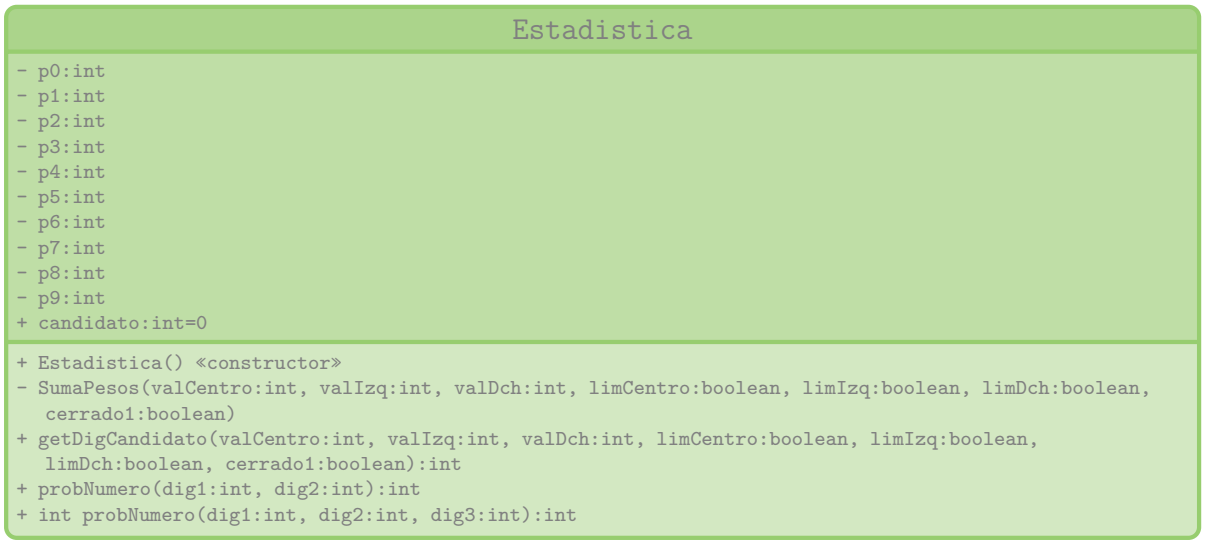


Figura 7.8.: Representación UML de la clase *Estadistica*.

Atributos:

- **`private int p0`**: Probabilidad de que el dígito sea un 0.
- **`private int p1`**: Probabilidad de que el dígito sea un 1.
- **`private int p2`**: Probabilidad de que el dígito sea un 2.
- **`private int p3`**: Probabilidad de que el dígito sea un 3.
- **`private int p4`**: Probabilidad de que el dígito sea un 4.
- **`private int p5`**: Probabilidad de que el dígito sea un 5.
- **`private int p6`**: Probabilidad de que el dígito sea un 6.
- **`private int p7`**: Probabilidad de que el dígito sea un 7.
- **`private int p8`**: Probabilidad de que el dígito sea un 8.
- **`private int p9`**: Probabilidad de que el dígito sea un 9.
- **`public int candidato = 0`**: Valor posible de la señal.

Métodos:

- **`public Estadistica()`**: Método constructor de la clase.
- **`private void SumaPesos(int valCentro, int valIzq, int valDch, boolean limCentro, boolean limIzq, boolean limDch, boolean cerrado1)`**: Suma a cada probabilidad p0 ∼p9 el peso que le corresponda según lo determinen los valores de entrada.
- **`public int getDigCandidato(int valCentro, int valIzq, int valDch, boolean limCentro, boolean limIzq, boolean limDch, boolean cerrado1)`**: Devuelve el número que tiene el valor de probabilidad más alto según el cálculo del método anterior.
- **`public int probNumero(int dig1, int dig2)`**: Devuelve la posible lectura de la señal para el caso de que esta tenga 2 dígitos. Controla que el segundo dígito sea un "0".
- **`public int probNumero(int dig1, int dig2, int dig3)`**: Devuelve la posible lectura de la señal para el caso de que esta tenga 3 dígitos. Controla que el primer dígito sea un "1" y el tercero sea un "0".

7.3.3. Módulo Detección

7.3.3.1. Clase *DetectionBasedTracker*

La clase implementa un mecanismo de seguimiento que define el uso de cascadas *Haar* o *LBP* en segundo plano para detectar objetos. La clase es una plantilla para el uso de código C++ . En la figura 7.9 se muestra la representación UML de la clase.

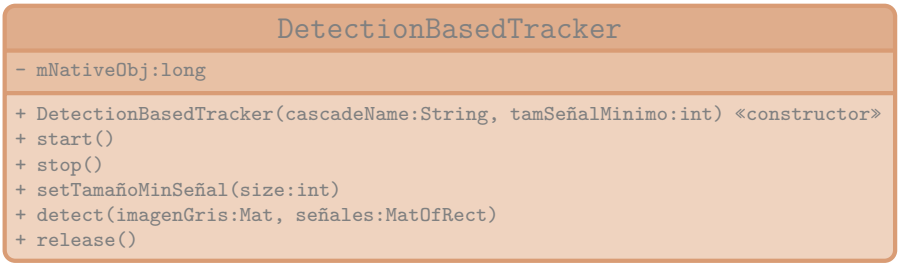


Figura 7.9.: Representación UML de la clase *DetectionBasedTracker*.

Atributos:

- **`private long mNativeObj`**: Identificador del objeto nativo.

Métodos:

- `public DetectionBasedTracker(String cascadeName, int tamSeñalMinimo)`: Método constructor de la clase.
- `public void start()`: Arranca el detector nativo.
- `public void stop()`: Detiene el detector nativo.
- `public void setTamañoMinSeñal(int size)`: Fija el tamaño mínimo de la señal.
- `public void detect(Mat imagenGris, MatOfRect señales)`: Procesa el frame en escala de grises y devuelve las regiones de interés donde se han localizado candidatos.
- `public void release()`: Elimina el objeto nativo y pone su indicador a cero.

Capítulo 8

Uso de la Aplicación

8.1. Introducción

El objetivo principal de esta aplicación es comprobar la capacidad que posee un terminal móvil para detectar e identificar un determinado grupo de señales de tráfico (de limitación de velocidad). Además se pretende comprobar la capacidad de tiempo real de este tipo de dispositivos.

Para ejecutar la aplicación se debe pulsar sobre el icono correspondiente (figura 8.1) en la pantalla de aplicaciones del terminal. Al iniciarse, la aplicación se inicia en modo normal. Es decir, se muestra la funcionalidad completa del detector.

Figura 8.1.: Icono de la aplicación.

Pulsando en la tecla menú se accede a las distintas opciones de visualización: *Sólo Localización*; *Mostrar Canal Value*; *Mostrar Regiones en Negro*; *Mostrar Contornos Localizados*; *Mostrar Posibles Números* (figura 8.2a). Al presionar la opción "más", se muestran el resto de opciones: *Ordenar Posibles Números*; *Mostrar Líneas de Escaneo*; *Modo Normal* (figura 8.2b).

La funcionalidad de cada opción será explicada en los siguientes apartados de este capítulo.

(a) **(b)**

Figura 8.2.: Opciones de Menú.

8.2. Opciones del menú

8.2.1. Sólo Localización

Al marcar esta opción, la aplicación mostrará si se ha detectado o no una señal candidata. En el caso de que se encuentre algún candidato, lo indicará marcándolo con un rectángulo verde en la imagen principal. Además, mostrará la región aislada en una ventana complementaria (figura 8.3).

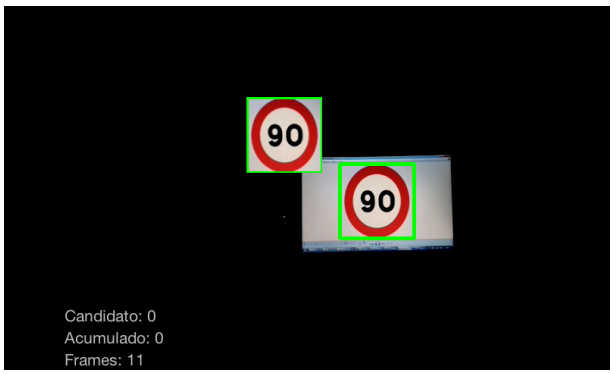


Figura 8.3.: Localización de una señal.

8.2.2. Mostrar Canal Value

Esta opción sirve para convertir la región candidata al formato de color HSV y mostrar el canal *Value* de esta representación. El canal *Value* representa la intensidad de la imagen. El resultado se mostrará en una ventana complementaria, mientras que en la ventana principal se mostrara la imagen original con la región remarcada en verde (figura 8.4). El empleo del formato HSV se debe a que esta representación de color es más tolerante a los cambios lumínicos.

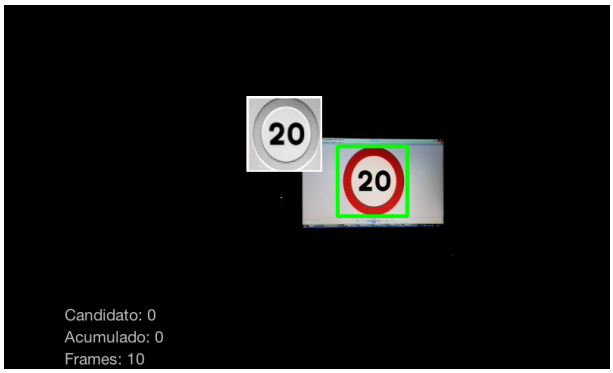


Figura 8.4.: Muestra del canal *Value*.

8.2.3. Mostrar Regiones en Negro

Teniendo el canal Value, es fácil localizar las regiones en negro de una imagen. Con esta opción, se muestra en la ventana auxiliar el resultado de filtrar y binarizar el canal Value. En la imagen binaria aparecen las zonas que tienen color negro en blanco y el resto en negro (figura 8.5).

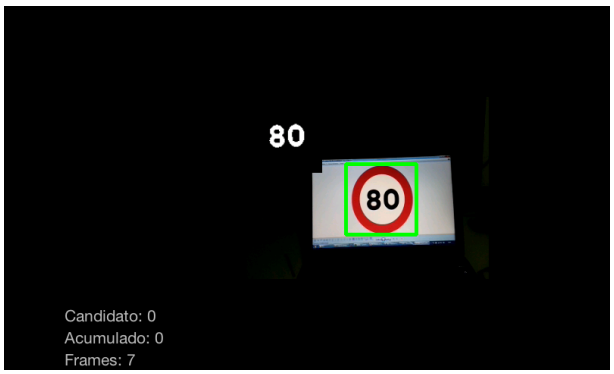


Figura 8.5.: Imagen binarizada del filtrado de color negro.

8.2.4. Mostrar Contornos Localizados

Una vez se tiene una imagen binarizada, es sencillo aislar las regiones donde puede existir un dígito (siempre son negros) y difinir la región de la imagen que ocupan. Al seleccionar esta opción, en la ventana auxiliar salen las distintas regiones que contienen el color negro. Estas regiones se mostrarán en color amarillo (figura 8.6).

8.2.5. Mostrar Posibles Números

Como se vio en la figura 8.6, a veces según la incidencia de la luz o las posibles sombras, se localizan otras regiones en negro en zonas donde por la definición de las

Figura 8.6.: Localización de los distintos contornos.

señales de limitación de velocidad no pueden aparecer los números. Al elegir esta opción del menú, se eliminan los contornos que no están centrados en la región y se delimita el área rectangular que comprende a cada contorno y se enumeran estas áreas. En la imagen auxiliar aparece la imagen de la señal con rectángulos que contienen a cada uno de los posibles números (figura 8.7). Los rectángulos tendrán distintos colores para indicar en que posición se extrajo el contorno. Verde para la primera, amarillo para la segunda y rojo para la tercera posición.

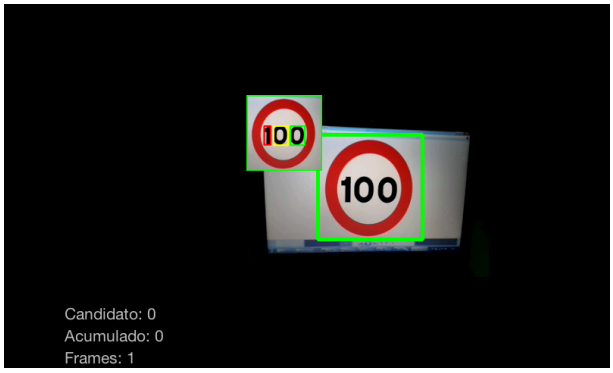


Figura 8.7.: Marcado de los posibles números.

8.2.6. Ordenar Posibles Números

La extracción de contornos no sigue ningún criterio respecto a la posición de estos en la imagen. Por este motivo los posibles números no tienen porque estar ordenados, como se vio en la figura 8.7. Al seleccionar esta opción se ordenan los números según la posición que tiene el contorno en la imagen. En la ventana suplementaria aparecen los números marcados con los rectángulos de colores siguiendo la estructura de un semáforo (figura 8.8).

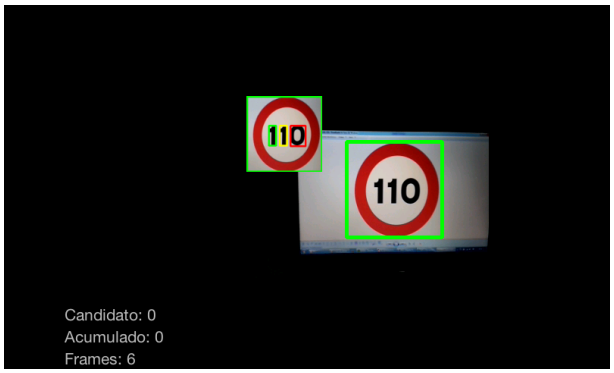


Figura 8.8.: Posibles números ordenados.

8.2.7. Mostrar Líneas de Escaneo

Una vez se tienen aislados a los posibles candidatos se les realiza a cada uno en reconocimiento óptico. Al seleccionar esta opción se crea una ventana auxiliar para cada posible número y en ella se muestra el dígito con las tres líneas de escaneo superpuestas y los correspondientes puntos que indican las intersecciones (figura 8.8).

Figura 8.9.: Candidatos con sus respectivas líneas de escaneo.

8.2.8. Modo Normal

Al seleccionar esta opción, la aplicación realiza la operativa completa. Es decir, sigue todos los pasos hasta identificar la señal. En la pantalla se muestra diversa información. En la esquina superior izquierda aparece una imagen con la última señal identificada, y en la esquina inferior izquierda aparecen distintas líneas de texto:

- Candidato: En el caso de que se esté identificando una señal, se muestra al posible valor de esta.
- Acumulado: Indica cuantas veces se ha identificado el valor candidato en una secuencia de frames.

- Frames: indica el frame de la secuencia de se está procesando (de 1 a 20).

En la figura 8.10 se muestra como la en la identificación anterior se detecto la señal de limitación a 30 Km/h y se ha detectado la actual señal como de 70 Km/h en 10 ocasiones. Cuando la aplicación detecta una señal diferente, lo indica mostrando esta señal como ventana emergente que permanece unos instantes en pantalla (figura 8.11). Se considera que una señal está correctamente identificada cuando se obtiene el mismo valor en el reconocimiento óptico de caracteres en al menos 10 frames de cada ciclo de 20.

Figura 8.10.: Proceso de identificación.

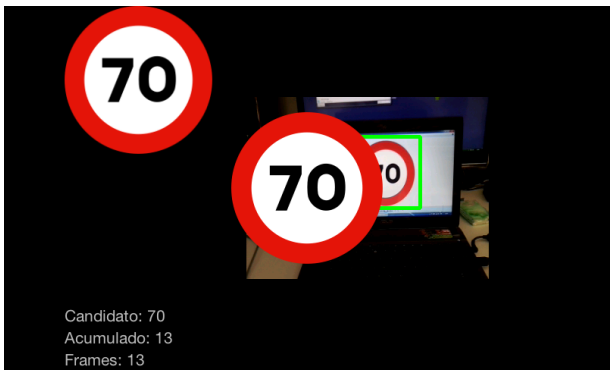


Figura 8.11.: Señal identificada.

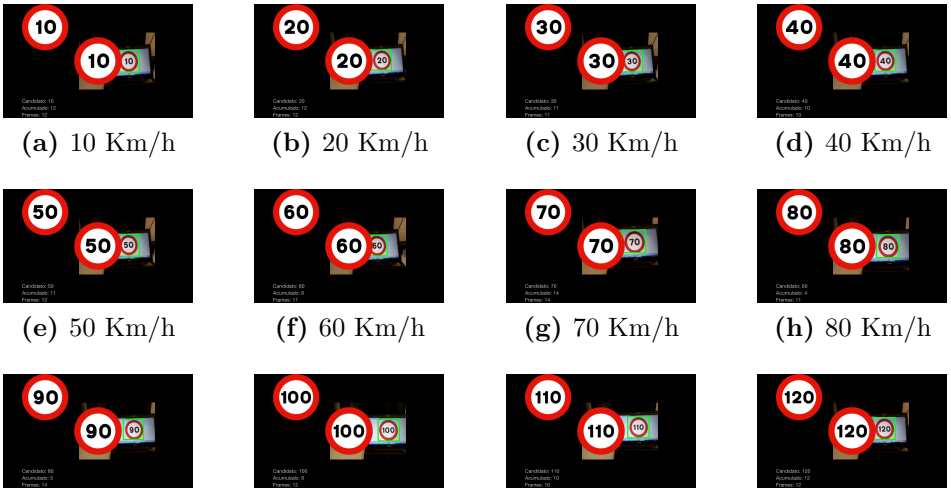
8.3. Evaluación de la aplicación

En este apartado se detalla el proceso de pruebas de la aplicación determinando primero si es capaz de reconocer las señales de limitación de velocidad y a continuación, si la aplicación descarta las demás señales. Para estas pruebas, se emplean imágenes en la pantalla de un ordenador hacia las que se apunta con la cámara del terminal móvil. También se comprobarán las condiciones en las que se pueden dar falsos positivos o malas identificaciones.

8.3.1. Casos positivos

Para esta prueba se emplean las imágenes de las señales que se utilizaron para crear las notificaciones de la aplicación. Estas imágenes representan las distintas señales de limitación de velocidad vigentes en España (10 - 120 Km/h).

En la figura 8.12 se muestra el resultado de mostrar a la cámara de terminal las distintas señales de tráfico.



(a) 10 Km/h **(b)** 20 Km/h **(c)** 30 Km/h **(d)** 40 Km/h

(e) 50 Km/h **(f)** 60 Km/h **(g)** 70 Km/h **(h)** 80 Km/h

(i) 90 Km/h **(j)** 100 Km/h **(k)** 110 Km/h **(l)** 120 Km/h

Figura 8.12.: Identificación positiva de señales.

8.3.2. Casos negativos

En este apartado, se ve como la aplicación descarta las señales de tráfico restantes. Estas señales se dividen en los siguientes tipos:

- Señales que son semejantes en colores pero no en forma: En este caso estarían comprendidas las señales de indicación de peligro, que poseen un fondo blanco con borde rojo y un anagrama en negro pero, que tienen forma triangular. En este grupo también estaría comprendida la señal de ceda el paso. En la figura se muestran algunos casos donde estas señales son mostradas a la cámara y no son reconocidas. En este caso es el proceso de detección el que descarta las señales ya que el identificador está entrenado para reconocer señales circulares.

- Señales circulares que no tienen el mismo patrón de color: Esto incluye tanto a las señales de obligatoriedad, que son circulares con fondo azul y anagramas en blanco (figura 8.14a), como a las señales de fin de prohibición o restricción,

(a) Ceda el Paso. **(b)** Obras. **(c)** Subida con fuerte pendiente.

Figura 8.13.: Descarte de señales con igual patrón de color y distinta forma.

que son blancas con anagramas en gris atravesadas por líneas diagonales negras (figura 8.14b) o bien azules con anagrama en blanco atravesadas por una línea diagonal roja (figura 8.14c). Como se ve en la figura, al no tener el borde exterior, estas señales tampoco son detectadas.

(a) Velocidad mínima. **(b)** Fin de la limitación de velocidad. **(c)** Fin de vía reservada para ciclos.

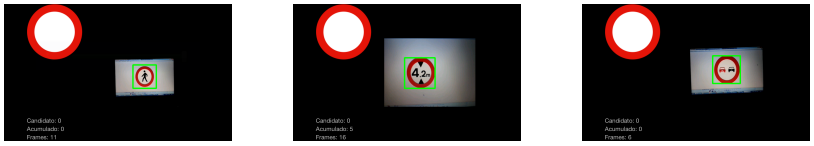
Figura 8.14.: Descarte de señales con forma circular y distintos colores.

- Resto de señales de prohibición: Se corresponde con las señales circulares con borde rojo que no son señales de limitación de velocidad. En este grupo se engloban:

  - las señales de prohibición de entrada, con borde rojo, fondo blanco y dibujo en negro (figura 8.15a). Como se ve, son detectadas como posibles candidatos, pero no pasan la etapa de identificación. Es decir, el reconocimiento óptico de caracteres no reconoce ningún numero.

  - las señales de restricción de paso, similares a las anteriores con posibilidad de contener números (figura 8.15b). En estos casos el reconocimiento óptico falla al haber letras en las leyendas.

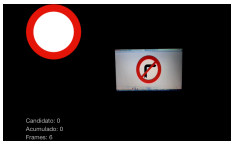
  - otras señales de prohibición o restricción como pueden ser aquellas que tienen borde rojo sobre fondo blanco y mas de un ideograma (figura 8.15c), o aquellas que están atravesadas por una o dos líneas diagonales rojas. Bien sean con fondo blanco (figura 8.15d) o con fondo azul (figura 8.15e). En el primer caso, la señal es rechazada al no identificarse los ideogramas como números. En los otros dos, la línea diagonal hace que el detector no identifique la señal como candidata.



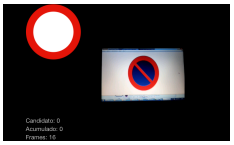
(a) Entrada prohibida a peatones.

(b) Limitación de altura.

(c) Adelantamiento prohibido.



(d) Giro a la derecha prohibido.

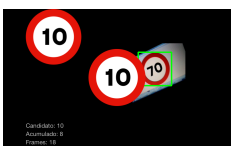


(e) Zona de estacionamiento limitado.

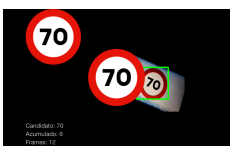
Figura 8.15.: Otras señales rechazadas.

8.3.3. Falsos positivos e identificaciones incorrectas.

Se dan casos en los que pueden producirse identificaciones erróneas o falsos positivos. Esto último es muy improbable debido a la naturaleza redundante y a lo restrictivo del rango de caracteres reconocibles (sólo números del 0 al 9) de la aplicación. Es decir, es muy improbable que la aplicación detecte como señal de limitación de velocidad si realmente no lo es. Lo que si puede suceder es que se identifiquen incorrectamente algunas señales de limitación de velocidad al determinar la aplicación que el número que esta contiene es otro distinto al que realmente hay. Este caso se da sobre todo si la señal está inclinada y su eje horizontal no está alineado con el eje horizontal de la cámara del terminal móvil. El ángulo de inclinación puede ser más crítico en algunos dígitos e incluso afectar más o menos según la inclinación sea en sentido horario o anti-horario. En la figura 8.16 se ve como para una misma señal, una inclinación en sentido anti-horario produce una identificación incorrecta (figura 8.16a) y sin embargo una inclinación en sentido horario no hace que se produzca error en la detección (figura 8.16b).



(a)



(b)

Figura 8.16.: Detección de señales inclinadas.

8.4. Conclusiones

Como se ha demostrado, es posible emplear los terminales móviles actuales para detectar e identificar señales de tráfico. Sin embargo, la capacidad de procesado y la escasa calidad de las cámaras de muchos de los terminales móviles del mercado hacen que, de momento, estos dispositivos no puedan ser empleados en aplicaciones autónomas que permitan identificar en tiempo real las señales desde el interior de un vehículo en movimiento. A esto tampoco ayuda el que la versión para Android de OpenCV esté aún en un estadio inicial y que no se tengan optimizaciones para los distintos tipos de procesadores que aparecen en estos dispositivos. Por lo tanto, y a la espera de como evolucionen en el futuro tanto los terminales móviles (en cuanto a calidad de imagen y capacidad de proceso) como la librería OpenCV (en cuanto a optimizaciones y nuevos algoritmos más ligeros), las aplicaciones del tipo a la realizada en este proyecto fin de carrera solo tendrán validez como ejercicio de estudio y demostración de capacidades y no como aplicaciones comerciales.

Bibliografía

[1] Wikipedia. *Vienna Convention on Road Traffic.*
Disponible en: `http://en.wikipedia.org/wiki/Vienna_Convention_on_Road_Traffic`.
Última visita: Noviembre 2012.

[2] Wikipedia. *Vienna Convention on Road Signs and Signals.*
Disponible en: `http://en.wikipedia.org/wiki/Vienna_Convention_on_Road_Signs_and_Signals`.
Última visita: Noviembre 2012.

[3] *SEÑALES VERTICALES DE CIRCULACIÓN*, tomo II. Catálogo y Significado de las Señales. (MOPT. Ministerio de Obras Públicas y Transporte . Dirección General de Carreteras.). **Junio 1992**, URL `http://www.fomento.gob.es/NR/rdonlyres/172DDE1A-9B93-49E0-BB1E-8F99CABD0D81/55739/1110950.pdf`.

[4] *Página oficial de Android.*
Disponible en: `http://www.android.com/`.
Última visita: Noviembre 2012.

[5] *Página oficial de iOS.*
Disponible en: `http://www.apple.com/es/iphone/ios/`.
Última visita: Noviembre 2012.

[6] *Página oficial de Symbian.*
Disponible en: `http://symbian.nokia.com/`.
Última visita: Noviembre 2012.

[7] *Página oficial de RIM.*
Disponible en: `http://www.rim.com/`.
Última visita: Noviembre 2012.

[8] *Página oficial de Windows Phone.*
Disponible en: `http://www.microsoft.com/windowsphone/es-es/default.`

aspx.
Última visita: Noviembre 2012.

[9] *Página oficial de OpenCV.*
Disponible en: `http://opencv.org/`.
Última visita: Noviembre 2012.

[10] Liu, W., et al. *Real-Time Speed Limit Sign Detection and Recognition from Image Sequences.*
Proceedings of the 2010 International Conference on Artificial Intelligence and Computational Intelligence, tomo 1 de *AICI '10* (
IEEE Computer Society, Washington, DC, USA), págs. 262–267. **2010**.

[11] Barnes, N. y Zelinsky, A. *Real-time radial symmetry for speed sign detection.*
Intelligent Vehicle, IEEE Symposium. **2004**.

[12] Broggi, A., et al. *Real Time Road Signs Recognition.*
Intelligent Vehicle, IEEE Symposium. págs. 981–986. **2007**.

[13] Eichner, M.L. y Breckon, T.P. *Integrated speed limit detection and recognition from real-time video.*
Intelligent Vehicle, IEEE Symposium. págs. 626–631. **2008**.

[14] Liu, H., Liu, D., y Xin, J. *Real-time recognition of road traffic sign in motion image based on genetic algorithm.*
Machine Learning and Cybernetics. **2002**.

[15] Miura, J., Kanda, T., y Shirai, Y. *An active vision system for real-time traffic sign recognition.*
International Conference on Intelligent Transportation. **2000**.

[16] Muller-Schneiders, S., Nunn, C., y Meuter, M. *Performance evaluation of a real time traffic sign recognition system.*
Intelligent Vehicle, IEEE Symposium. págs. 79–84. **2008**.

[17] Wu, J., et al. *Real-time Robust Algorithm for Circle Object Detection.*
Proceedings of the 2008 The 9th International Conference for Young Computer Scientists, ICYCS '08
(IEEE Computer Society, Washington, DC, USA), págs. 1722–1727. **2008**.

[18] Maldonado-Bascón, S., et al. *Road-Sign Detection and Recognition Based on Support Vector Machines.*
IEEE Transactions on Intelligent Transportation Systems, tomo 8; págs. 264–278. **2007**.

[19] Tørresen, J., Bakke, J.W., y Sekanina, L. *Efficient Recognition of Speed Limit Signs.*
International Conference on Intelligent Transportation. **2004**.

[20] Huang, Y.S. y Lee, Y.S. *Detection and recognition of speed limit signs.*
Computer Symposium (ICS); págs. 107–112. **2010**.

[21] Chiang, H.H., et al. *Road speed sign recognition using edge-voting principle and learning vector quantization network.* **2010**.

[22] Escalera, A.D.L., et al. *Visual sign information extraction and identification by deformable models for intelligent vehicles.*
IEEE Transactions on Intelligent Transportation Systems, tomo 5; págs. 57–68. **2004**.

[23] Srinonchat, J. *Efficient detection of speed limit signs within obscure environment.*
IET Conference Publications, tomo 2010(CP569); págs. 311–314. **2010**.

[24] Ishak, K.A. y Sani, M.M. *A Speed limit Sign Recognition System Using Artificial Neural Network.*
Student Conference on Research and Development. **2006**.

[25] Xu, S. *Robust traffic sign shape recognition using geometric matching.*
Iet Intelligent Transport Systems, tomo 3. **2009**.

[26] Fischler, M.A. y Bolles, R.C. *Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography.*
Commun. ACM, tomo 24(6); págs. 381–395. **jun 1981**.

[27] Stache, N.C. y Zimmer, H. *Robust Circle Fitting in Industrial Vision for Process Control of Laser Welding.*
Proceedings of the 11th International Student Conference on Electrical Engineering POSTER (Prague). **2007**.

[28] Canny, J. *A Computational Approach to Edge Detection.*
IEEE Trans. Pattern Anal. Mach. Intell., tomo 8(6); págs. 679–698. **jun 1986**.

[29] Huttenlocher, D.P., et al. *Comparing Images Using the Hausdorff Distance.*
IEEE Transactions on Pattern Analysis and Machine Intelligence, tomo 15; págs. 850–863. **1993**.

[30] Siogkas, G.K. y Dermatas, E.S. *Detection, Tracking and Classification of Road Signs in Adverse Conditions.*
MELECON - IEEE Mediterranean Electrotechnical Conference. **2006**.

[31] Kerbyson, D.J. y Atherton, T.J. *Circle Detection Using Hough Transform Filters.* **1995**.

[32] Wang, Y., Shi, M., y Wu, T. *A Method of Fast and Robust for Traffic Sign Recognition.*
Proceedings of the 2009 Fifth International Conference on Image and Graphics, ICIG '09 (IEEE Computer Society, Washington, DC, USA), págs. 891–895. **2009**.

[33] Baro, X. y Vitria, J. *Fast Traffic sign detection on greyscale images.*
Recent Advances in Artificial Intelligence Research and Development; págs. 69 – 76. **2004**.

[34] Lienhart, R. y Maydt, J. *An extended set of Haar-like features for rapid object detection.*
International Conference on Image Processing, tomo 1. págs. 900–903. **2002**.

[35] Viola, P.A. y Jones, M.J. *Rapid Object Detection using a Boosted Cascade of Simple Features.*
Computer Vision and Pattern Recognition, tomo 1. págs. 511–518. **2001**.

[36] Vapnik, V. *An overview of statistical learning theory.*
IEEE Transactions on Neural Networks; págs. 988–999. **1999**.

[37] Gil-Jimenez, P., et al. *Traffic sign shape classification based on Support Vector Machines and the FFT of the signature of blobs.*
Intelligent Vehicles Symposium, IEEE. págs. 375–380. **2007**.

[38] Castells-Rufas, D. y Carrabina, J. *Camera-based Digit Recognition System.* **2006**.

[39] Li, Y. y Qian, H. *Automatic Recognition System for Numeric Characters on Ammeter Dial Plate.*
Proceedings of the 2008 The 9th International Conference for Young Computer Scientists, ICYCS '08
(IEEE Computer Society, Washington, DC, USA), págs. 913–918. **2008**.

[40] Xu Han-wei, W.C. *A New Algorithm for Numeral Recognition.*
Surveying and Mapping of Geology and Mineral Resources, tomo 2; pág. 31. **2000**.

[41] Bishop, C.M. *Neural networks for pattern recognition* (Oxford University Press). **1996**.

[42] *OpenCV Wiki*.
Disponible en: `http://opencv.willowgarage.com/wiki/`.
Última visita: Noviembre 2012.

[43] Wikipedia. *Lanczos algorithm*.
Disponible en: `http://en.wikipedia.org/wiki/Lanczos_algorithm`.
Última visita: Noviembre 2012.

[44] Wikipedia. *Nearest-neighbor interpolation*.
Disponible en: `http://en.wikipedia.org/wiki/Nearest-neighbor_interpolation`.
Última visita: Noviembre 2012.

[45] Wikipedia. *Multivariate interpolation*.
Disponible en: `http://en.wikipedia.org/wiki/Multivariate_interpolation`.
Última visita: Noviembre 2012.

[46] Wikipedia. *Segmentación (procesamiento de imágenes)*.
Disponible en: `http://es.wikipedia.org/wiki/Segmentaci%C3%B3n_(procesamiento_de_im%C3%A1genes)#Transformaci.C3.B3n_divisoria_.28watershed.29`.
Última visita: Noviembre 2012.

[47] C. Rother, V. Kolmogorov, A.B.M.B. *Image and Video Editing. GrabCut*.
Disponible en: `http://research.microsoft.com/en-us/um/cambridge/projects/visionimagevideoediting/segmentation/grabcut.htm`.
Última visita: Noviembre 2012.

[48] Wikipedia. *Earth mover's distance*.
Disponible en: `http://en.wikipedia.org/wiki/Earth_mover%27s_distance`.
Última visita: Noviembre 2012.

[49] Freeman, H. *On the Encoding of Arbitrary Geometric Configurations*.
IEEE Transactions on Electronic Computers, tomo EC-10; págs. 260–268. **1961**.

[50] Wikipedia. *Rasterización*.
Disponible en: `http://es.wikipedia.org/wiki/Rasterizaci%C3%B3n`.
Última visita: Noviembre 2012.

[51] Wikipedia. *Teorema de Green.*
Disponible en: `http://es.wikipedia.org/wiki/Teorema_de_Green`.
Última visita: Noviembre 2012.

[52] Hu, M.K. *Visual pattern recognition by moment invariants.*
Information Theory, IRE Transactions on, tomo 8(2); págs. 179–187. **Febrero 1962**.

[53] Wikipedia. *Image moment.*
Disponible en: `http://en.wikipedia.org/wiki/Image_moment#Rotation_invariant_moments`.
Última visita: Noviembre 2012.

[54] Wikipedia. *Triangulación de Delaunay.*
Disponible en: `http://es.wikipedia.org/wiki/Triangulaci%C3%B3n_de_Delaunay`.
Última visita: Noviembre 2012.

[55] Wikipedia. *Vector propio y valor propio.*
Disponible en: `http://es.wikipedia.org/wiki/Vector_propio_y_valor_propio`.
Última visita: Noviembre 2012.

[56] Harris, C. y Stephens, M. *A Combined Corner and Edge Detector.*
Proceedings of the 4th Alvey Vision Conference. págs. 147–151. **1988**.

[57] Wikipedia. *Transformada de Hough.*
Disponible en: `http://es.wikipedia.org/wiki/Transformada_de_Hough`.
Última visita: Noviembre 2012.

[58] Butler, D.E., Bove, Jr., V.M., y Sridharan, S. *Real-time adaptive foreground/background segmentation.*
EURASIP J. Appl. Signal Process., tomo 2005; págs. 2292–2304. **jan 2005**.

[59] Welch, G. y Bishop, G. *An Introduction to the Kalman Filter*. Informe técnico. **1995**.

[60] Wikipedia. *Kalman filter.*
Disponible en: `http://en.wikipedia.org/wiki/Kalman_filter`.
Última visita: Noviembre 2012.

[61] Liao, S., et al. *Learning Multi-scale Block Local Binary Patterns for Face Recognition.*

Advances in Biometrics, (Editado por S.W. Lee y S. Li), tomo 4642 de *Lecture Notes in Computer Science* (
Springer Berlin / Heidelberg), págs. 828–837. **2007**.

[62] Wikipedia. *Histogram of oriented gradients*.
Disponible en: `http://en.wikipedia.org/wiki/Histogram_of_oriented_gradients`.
Última visita: Noviembre 2012.

[63] Chang, C.C. y Lin, C.J. *LIBSVM: A library for support vector machines*.
ACM Trans. Intell. Syst. Technol., tomo 2(3); págs. 27:1–27:27. **may 2011**.

[64] Breiman, L. *Classification and regression trees* (
Wadsworth International Group). **1984**.

[65] Wikipedia. *Coeficiente de Gini*.
Disponible en: `http://es.wikipedia.org/wiki/Coeficiente_de_Gini`.
Última visita: Noviembre 2012.

[66] Hastie, T., Tibshirani, R., y Friedman, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*.
Springer Series in Statistics (
Springer). **2003**.

[67] Friedman, J., Hastie, T., y Tibshirani, R. *Additive Logistic Regression: a Statistical View of Boosting*.
The Annals of Statistics, tomo 38(2). **2000**.

[68] Friedman, J.H. *Greedy Function Approximation: A Gradient Boosting Machine*.
Annals of Statistics, tomo 29; págs. 1189–1232. **2000**.

[69] Breiman, L. y Cutler, A. *Random Forests*.
Disponible en: `http://www.stat.berkeley.edu/users/breiman/RandomForests/`.
Última visita: Noviembre 2012.

[70] Geurts, P., Ernst, D., y Wehenkel, L. *Extremely randomized trees*.
Mach. Learn., tomo 63(1); págs. 3–42. **apr 2006**.

[71] Wikipedia. *Feed-forward*.
Disponible en: `http://es.wikipedia.org/wiki/Feed-forward`.
Última visita: Noviembre 2012.

[72] Wikipedia. *Rprop*.
Disponible en: `http://en.wikipedia.org/wiki/Rprop`.
Última visita: Noviembre 2012.

[73] *Página oficial sobre Open Handset Alliance*.
Disponible en: `http://www.openhandsetalliance.com/index.html`.
Última visita: Noviembre 2012.

[74] Ortiz, D. *Android domina el mercado de smartphones mundial con un 59 % de cuota*.
Disponible en: `http://www.xatakandroid.com/mercado/android-domina-el-mercado-de-smartphones-mundial-con-un-59-de-cuota`.
Última visita: Noviembre 2012.

[75] Alcalde, A. *Fundamentos aplicaciones Android*.
Disponible en: `http://www.elbauldelprogramador.com/programacion/fundamentos-aplicaciones-android-parte/`.
Última visita: Noviembre 2012.

[76] *Documentación sobre el archivo AndroidManifest*.
Disponible en: `http://developer.android.com/guide/topics/manifest/manifest-intro.html`.
Última visita: Noviembre 2012.

[77] *Activities en Android*.
Disponible en: `http://developer.android.com/guide/components/activities.html`.
Última visita: Noviembre 2012.

[78] *Services en Android*.
Disponible en: `http://developer.android.com/guide/components/services.html`.
Última visita: Noviembre 2012.

[79] *Especificación de la clase BroadcastReceiver*.
Disponible en: `http://developer.android.com/reference/android/content/BroadcastReceiver.html`.
Última visita: Noviembre 2012.

[80] *Especificación de la clase ContentProvider*.
Disponible en: `http://developer.android.com/reference/android/content/`

`ContentProvider.html`.
Última visita: Noviembre 2012.

[81] *Especificación de la clase Intent.*
Disponible en: `http://developer.android.com/reference/android/content/Intent.html`.
Última visita: Noviembre 2012.

[82] *Interfaz de usuario en Android.*
Disponible en: `http://developer.android.com/guide/topics/ui/index.html`.
Última visita: Noviembre 2012.

[83] *Especificación sobre Layouts.*
Disponible en: `http://developer.android.com/guide/topics/ui/declaring-layout.html`.
Última visita: Noviembre 2012.

[84] *Descripción del paquete android.app.*
Disponible en: `http://developer.android.com/reference/android/app/package-summary.html`.
Última visita: Noviembre 2012.

[85] *Descripción del paquete android.widget.*
Disponible en: `http://developer.android.com/reference/android/widget/package-summary.html`.
Última visita: Noviembre 2012.

[86] *Descripción del paquete android.view.*
Disponible en: `http://developer.android.com/reference/android/view/package-summary.html`.
Última visita: Noviembre 2012.

[87] *Descripción del paquete android.content.*
Disponible en: `http://developer.android.com/reference/android/content/package-summary.html`.
Última visita: Noviembre 2012.

[88] *Descripción del paquete android.location.*
Disponible en: `http://developer.android.com/reference/android/location/Location.html`.
Última visita: Noviembre 2012.

[89] *Descripción del paquete android.os.*
Disponible en: `http://developer.android.com/reference/android/os/package-summary.html`.
Última visita: Noviembre 2012.

[90] *Descripción del paquete android.graphics.*
Disponible en: `http://developer.android.com/reference/android/graphics/package-summary.html`.
Última visita: Noviembre 2012.

[91] *Descripción del paquete android.util.*
Disponible en: `http://developer.android.com/reference/android/util/package-summary.html`.
Última visita: Noviembre 2012.

[92] *Guía de Instalación del SDK de Android.*
Disponible en: `http://developer.android.com/sdk/installing/index.html`.
Última visita: Noviembre 2012.

[93] *Documentación del NDK Android.*
Disponible en: `http://developer.android.com/tools/sdk/ndk/index.html`.
Última visita: Noviembre 2012.

[94] *Documentación plugin CDT.*
Disponible en: `http://eclipse.org/cdt/`.
Última visita: Noviembre 2012.

[95] OpenCV. *API Reference; objdetect. Object Detection.*
Disponible en: `http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html#cascadeclassifier`.
Última visita: Noviembre 2012.

[96] OpenCV. *API Reference; imgproc. Image Processing.*
Disponible en: `http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#findcontours`.
Última visita: Noviembre 2012.

[97] OpenCV. *Tutorials. Eroding and Dilating.*
Disponible en: `http://docs.opencv.org/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html`.
Última visita: Noviembre 2012.

[98] OpenCV. *API Reference; imgproc. Image Processing.*
Disponible en: `http://docs.opencv.org/modules/imgproc/doc/filtering.html#getstructuringelement`.
Última visita: Noviembre 2012.

Parte II.

Pliego de Condiciones

Pliego de condiciones

Introducción

A continuación se especificarán los requisitos técnicos que permitirán la correcta ejecución de la aplicación en el terminal móvil. Para ello se establecerán las diferentes condiciones referentes al hardware y al software, as como la instalación de la aplicación. Por ultimo, se expondrá la licencia de uso del software.

Requisitos

El hardware requerido ha de consistir en un terminal móvil con cámara integrada. Ademas, debe tratarse de un terminal con el sistema operativo Android, de Google, en su versión 2.2 Froyo o superior. Según las pruebas realizadas, se garantiza el correcto funcionamiento del sistema con los terminales Samsung Galaxy SII y HTC Desire.

Condiciones de instalación

Para la instalación de la aplicación desarrollada, se debe copiar el archivo *Detector.apk* en la memoria externa (*sdcard* o memoria SD) del terminal móvil, en una carpeta cualquiera (se recomienda la carpeta *Aplicaciones*). La aplicación para el terminal móvil se encuentra en el directorio `/Codigo/Aplicacion` del disco compacto adjunto a esta memoria,

La copia puede llevarse a cabo mediante USB, *Bluetooth* o WIFI. Si bien, para el primero de ellos, se realiza la copia directamente, mediante el propio explorador de archivos del ordenador, en el caso del *Bluetooth* o WIFI es necesario un gestor de ficheros para la transferencia.

Una vez copiado el archivo *apk* en la memoria externa del terminal móvil, se procede a la instalación. Para ello se necesita que el usuario haya descargado e instalado previamente, a través de *Google Play*, cualquier explorador de archivos, tal como *Apk Installer*, *Root Explorer*, entre otros. Así, se ejecuta el explorador de archivos en el terminal móvil y se entra en la carpeta donde se copió el archivo *Detector.apk*. Al seleccionar el archivo ha de salir un menú contextual en el que se sugieren varias opciones. Entre ellas, *Instalar*. Si se elige *Instalar*, aparecerá una ventana que explica al usuario los permisos que concede a la aplicación. Si el usuario esta de acuerdo, se elige *aceptar*, con lo que la instalación queda realizada.

Licencia del programa

Normas generales

Esta aplicación es propiedad de la Universidad de Las Palmas de Gran Canaria y su uso ha de estar sujeto a los términos y condiciones establecidas en esta licencia del programa, aceptándose todas sus cláusulas. El uso de esta aplicación ha de ser bajo la expresa autorización del autor o de la tutora de este proyecto o de la Escuela de Ingeniera de Telecomunicación y Electrónica la Universidad de Las Palmas de Gran Canaria.

Derechos de autor

La aplicación y la documentación están protegidas por la ley de Propiedad Intelectual aplicable, as como por las disposiciones de los tratados internacionales. En consecuencia, el usuario podrá usar copia de esta aplicación, así como del código fuente de programación y de la documentación, siempre bajo la autorización del autor o la tutora de este proyecto o de la Escuela de Ingeniera de Telecomunicación y Electrónica de la Universidad de Las Palmas de Gran Canaria.

Garantía

Se garantiza el correcto funcionamiento de la aplicación en el momento de la instalación de acuerdo con las especificaciones vistas con anterioridad. La instalación de la aplicación no acarrea la aparición de defectos en los dispositivos que cumplan con las especificaciones técnicas.

Con la única excepción de lo expresamente expuesto en el párrafo anterior, la aplicación ha sido desarrollada sin garantías de ninguna clase. El autor no asegura, garantiza o realiza ninguna declaración respecto al uso o los resultados derivados de la utilización del programa o de la documentación. Tampoco se garantiza que la operación del programa sea ininterrumpida o sin errores. En ningún caso serán el autor, la tutora o la Escuela de Ingeniera de Telecomunicación y Electrónica de la Universidad de Las Palmas de Gran Canaria responsables de los perjuicios directos, indirectos, incidentales, ejemplarios o consiguientes gastos, lucro cesante, perdida de ahorros, interrupción de negocios, perdida de información comercial o de negocio o cualquier otra perdida que resulte del uso o de la incapacidad de usar el programa o la documentación. El usuario conoce y acepta que los derechos de licencia reflejan esta asignación de riesgo como el resto de clausulas y restricciones.

Otras consideraciones

La fiabilidad de operación del programa puede verse afectada por factores adversos a los que se denominan “fallas del sistema". En estos se incluyen errores en el funcionamiento del hardware del dispositivo, sistema operativo o entorno del mismo, compiladores o software de desarrollo usado para realizar la aplicación, software externo utilizado para el funcionamiento de la misma, errores de instalación, problemas de compatibilidad de software y hardware, fallos o funcionamiento incorrectos de equipos de control, fallas por uso o errores por parte del usuario de la aplicación.

En el supuesto de que cualquier disposición de esta licencia sea declarada total o parcialmente invalida, la clausula afectada será modificada convenientemente de manera que sea ejecutable y una vez modificada, plenamente eficaz, permaneciendo el resto de este contrato en plena vigencia.

Esta licencia se regirá por las leyes de España. El usuario o licenciatario acepta la jurisdicción exclusiva de los tribunales de este país en relación con cualquier disputa que pudiera derivarse de la presente licencia.

Parte III.

Presupuesto

Presupuesto

Introducción

En este capítulo se estimarán los gastos generados por el proyecto fin de carrera presentado en esta memoria. Este presupuesto se ha obtenido según las indicaciones del COITT (Colegio Oficial de Ingenieros Técnicos de Telecomunicación) y la Asociación Española de Ingenieros de Telecomunicación. Asimismo, el presupuesto presentado se divide en las siguientes partes:

- Tarifa de honorarios por tiempo empleado.
- Amortización de los equipos empleados.
  - Amortización del material hardware.
  - Amortización del material software.
- Coste de acceso a Internet.
- Redacción de la documentación.
- Derechos de visado.
- Gastos de tramitación y envío.

Tarifa de honorarios por tiempo empleado

Este concepto contabiliza los gastos correspondientes a la mano de obra. Para su calculo se ha utilizado la ultima formula del COITT, que en su día se conocía como "Propuesta de Baremos Orientativos para el Calculo de Honorarios"[1]. Hay que comentar que en la actualidad, todos los colegios profesionales han recibido una nota del

[1]Dirección web: http://www.coitt.es

Ministerio de Economía y Hacienda en la que se recuerda que, siguiendo directivas europeas, se deben eliminar los baremos orientativos de honorarios que tradicionalmente se venían aplicando. En esta nota informativa se indica que, los honorarios son libres y responden al libre acuerdo entre el profesional y el cliente.

La formula propuesta por el COITT (antes de la nota) es:

$$H = (14,48 \times Hn) + (20,27 \times He)$$

Siendo:

- **H**: honorarios.
- **Hn**: honorarios en jornada laboral normal.
- **He**: honorarios fuera de la jornada laboral normal.

En el proyecto presentado en esta memoria, se ha empleado un periodo aproximado de **10** meses de trabajo, realizados por un ingeniero de software, trabajando **6** horas diarias, en horario laboral normal.

Distribución de la contemporizador del proyecto

El proyecto se divide en cuatro etapas bien diferenciadas, que se pasan a definir:

Documentación

Esta primera etapa comprende el tiempo empleado en la recopilación de la información necesaria para alcanzar los conocimientos necesarios para la realización de la aplicación. De igual manera, esta etapa comprende el tiempo necesario para la valoración de las ventajas y desventajas de los lenguajes de programación disponibles para la realización del proyecto, as como el proceso de documentación. Asimismo, una vez escogido el lenguaje de programación Android, se ha realizado un estudio exhaustivo del mismo, así como de las herramientas necesarias para su empleo.

Otros aspectos de documentación que se han tenido en cuenta son los siguientes:

- Estudio de la API de la biblioteca OpenCV.
- Estudio del entorno de diseño lógico de documentos LaTeX.

Definición

Esta segunda etapa consiste en la definición funcional y constitutiva de cada una de las partes del proyecto. En ella se definirán los objetivos de cada parte y las interrelaciones entre ellas.

Análisis

En esta etapa se establecen los módulos definitivos de los que dispondrá la aplicación, determinando sus funcionalidades. De esta manera se consigue una visión global puramente descriptiva de la aplicación, analizándose los objetivos de cada módulo del sistema.

Codificación

En esta etapa se realiza la transcripción del código que satisface las necesidades y objetivos vistos en la etapa de análisis. Esta fase incluye la depuración y las pruebas pertinentes de las prestaciones del código.

A la vez que se han llevado a cabo las etapas anteriores, se ha ido elaborando la memoria final explicativa del proceso. Este ultimo punto se contempla en la sección *Redacción de la documentación.*

Calculo de tarifa de honorarios por tiempo empleado

En la tabla 1 se detalla el tiempo empleado para cada una de las etapas anteriormente descritas y se realiza el cálculo de los honorarios.

Etapa	**Horas laborales**	**Honorarios**
Documentación	240	3.475,20 €
Definición	90	1.303,20 €
Análisis	180	2.606,40 €
Codificación	570	8.253,60 €
Total		**15.638,40 €**

Tabla 1: Honorarios por tiempo empleado.

Por lo que sumadas cada una de las etapas, el calculo de honorarios por tiempo empleado asciende a QUINCE MIL SEISCIENTOS TREINTA Y OCHO EUROS CON CUARENTA CÉNTIMOS DE EURO.

Amortización de los equipos empleados

Dentro de este concepto se considera tanto la amortización del hardware como del software empleado en la realización del proyecto. De este modo se estipula el coste de amortización para un período de 3 años, utilizando un sistema de amortización lineal o constante. En este sistema, se supone que el inmovilizado material se deprecia de forma constante a lo largo de su vida útil. La cuota de amortización anual se calcula haciendo uso de la siguiente formula:

$$C = \frac{V_{ad} - V}{N}$$

Donde:

- **C**: cuota de amortización anual.
- $\mathbf{V_{ad}}$: valor de la adquisición.
- **V**: valor residual.
- **N**: numero de años de vida útil de la adquisición.

Siendo el valor residual el valor teórico que se supone tendrá el elemento en cuestión después de su vida útil, teniendo en cuenta los índices de depreciación actual. En el caso del hardware y del software son 3 años (al 33 % de depreciación máximo por año).

Amortización del material hardware

Debido a que el proyecto se ha elaborado en un periodo inferior a 3 años, que es el periodo en que se calcula la amortización de material hardware, se realizara una amortización equiparable al periodo de duración del mismo. Según esto, se obtienen los gastos expuestos en la tabla 2.

Por tanto, el coste total de hardware asciende a QUINIENTOS CINCUENTA Y CINCO EUROS Y CUARENTA Y CUATRO CÉNTIMOS DE EURO.

Descripción	Valor de adquisición	Tiempo de uso	Coste anual	Total
Ordenador Portatil Asus U33J, Intel Core i3-M370 2,40GHz, 4Gb de RAM	794,40 €	9 meses	264,80 €	198,60 €
Ordenador Personal AMD Phenom II X4 965 3,40GHz, 4 Gb de RAM	879,90 €	9 meses	293,30 €	219,97 €
Telefono movil Samsung Galaxy SII	457,50 €	9 meses	152,50 €	114,37 €
Impresora HP F4280	90,00 €	9 meses	30,00 €	22,50 €
Total	2221,80€	-	740,60 €	**555,44 €**

Tabla 2: Precios y costes de amortización del hardware.

Amortización del material software

El coste total del software utilizado es la suma de las herramientas software empleadas para la realización del proyecto (tabla 3). De esta manera se describe cada uno de las herramientas utilizadas y el coste supuesto para su utilización usando la formula de amortización anteriormente citada.

Descripción	Valor de adquisición	Tiempo de uso	Coste anual	Total
Microsoft Windows 7	285,00 €	9 meses	95,00 €	71,25 €
Eclipse Indigo IDE	0,00 €	9 meses	0,00 €	0,00 €
SDK Android	0,00 €	9 meses	0,00 €	0,00 €
API OpenCV	0,00 €	9 meses	0,00 €	0,00 €
LyX	0,00 €	9 meses	0,00 €	0,00 €
Total	285,00 €	-	95,00 €	**71,25 €**

Tabla 3: Precios y costes de amortización del software.

Por tanto, el coste total de software asciende a SETENTA Y UN EUROS Y VEINTICINCO CÉNTIMOS DE EURO.

Coste de acceso a Internet

Para la conexión a Internet se dispone de una solución ADSL a 10Mbps, cuyo coste mensual es de 29,90 euros. Debido a que se ha usado dicha conexión durante cada uno de las etapas de creación del mismo (9 meses), el coste de acceso a Internet se traduce a un total de DOSCIENTOS SESENTA Y NUEVE EUROS Y DIEZ CÉNTIMOS DE EURO.

Redacción de la documentación

Haciendo uso del punto 1 de la recomendación del COITT, que hace referencia al valor monetario de la redacción de proyectos y trabajos en general, se aplicara la formula siguiente para determinar el coste asociado a la redacción de la memoria del proyecto:

$$R = 0,05 \times P$$

Donde:

- **R**: coste de la redacción.
- **P**: presupuesto.

El valor de P se obtiene sumando los costes de las secciones anteriores tal y como muestra la tabla 4.

Concepto	**Coste**
Honorarios por tiempo empleado	15.638,40 €
Amortización del Hardware	555,44 €
Amortización del Software	71,25 €
Coste de acceso a Internet	269,10 €
Total	**16.534,19 €**

Tabla 4: Precios y costes de la ejecución del proyecto mas la amortización y acceso a Internet.

De este modo, se obtiene que:

$$R = 0,05 \times 16{,}534,19 = 826,71$$

Al coste de redacción obtenido hasta el momento se le deben añadir otros gastos, quedando así el importe final de redacción del proyecto como se describe en la tabla 5.

Concepto	Coste
Redacción del proyecto	826,71 €
Papel de impresión	18,25 €
Tinta de impresión color	19,90 €
Tinta de impresión negra	16,90 €
Encuadernación	9,00 €
DVD-R 4.7GB Traxdata	1,95 €
Total	**892,71 €**

Tabla 5: Coste final de redacción del proyecto.

Por lo tanto la redacción total del proyecto asciende a un total de OCHOCIENTOS NOVENTA Y DOS EUROS Y SETENTA Y UN CÉNTIMOS DE EURO.

Derechos de visado

El COITT establece para la redacción de proyectos y trabajos en general los derechos de visado, que, a partir del 1 de Marzo del 2007 se calculan de acuerdo con la siguiente ecuación:

$$V = 0,007 \times P \times C$$

Donde:

- **V**: coste del visado.
- **P**: presupuesto.
- C: coeficiente reductor en función del presupuesto.

El valor del coeficiente se obtiene de la tabla 6, mientras que el valor de P es el valor total del presupuesto, que se pasa a calcular en la tabla 7.

Por lo tanto, el valor del visado sera de:

$$V = 0,007 \times 17{,}426,90 \times 1 = 121,99$$

Coste del presupuesto (€)	Factor de Correlación (C)
Hasta 30.050	1
Exceso de 30.050 hasta 60.101	0.9
Exceso de 60.101 hasta 90.151	0.8
Exceso de 90.151 hasta 120.202	0.7
Exceso de 120.202 hasta 150.253	0.65
Exceso de 150.253 hasta 300.506	0.60
Exceso de 300.506 hasta 450.759	0.55
Exceso de 450.759 hasta 601.012	0.50
Exceso de 601.012 hasta 901.518	0.40
Exceso de 901.518 hasta 1.202.024	0.40
...	

Tabla 6: Tabla de coeficientes para el cálculo del visado.

Concepto	Coste
Coste del tiempo empleado en la ejecución, amortización y acceso a Internet	16.534,19 €
Redacción del proyecto	892,71 €
Total	**17.426,90 €**

Tabla 7: Valor total de P para el calculo del visado (Presupuesto base).

Los costes de derechos de visado del proyecto ascienden a un total de CIENTO VEINTIÚN EUROS Y NOVENTA Y NUEVE CÉNTIMOS DE EURO.

Gastos de tramitación y envío

Los gastos de tramitación y envío según la tarifa vigente asciende a NUEVE EUROS por cada documento visado digital.

Presupuesto antes de impuestos

Sumando todos los conceptos calculados hasta el momento, se obtiene el presupuesto, sin incluir los impuestos, que se muestra en la tabla 8.

Concepto	Coste
Presupuesto base	17.426,90 €
Derechos de visado	121,99 €
Gastos de tramitación y envío	9,00 €
Total	**17.557,89 €**

Tabla 8: Presupuesto total sin impuestos.

El presupuesto calculado, antes de incluir los impuestos, asciende a DIECISIETE MIL QUINIENTOS CINCUENTA Y SIETE EUROS Y OCHENTA Y NUEVE CÉNTIMOS DE EURO.

Presupuesto incluyendo impuestos

Al presupuesto calculado anteriormente hay que incluirle un 7 % de IGIC obteniendo el coste del presupuesto final (tabla 9).

Concepto	Coste
Total (sin IGIC)	**17.557,89 €**
IGIC (7 %)	1229,05 €
Total	**18.786,94 €**

Tabla 9: Presupuesto total.

Por tanto, el presupuesto total, incluyendo impuestos, asciende a la cantidad de DIECIOCHO MIL SETECIENTOS OCHENTA Y SEIS EUROS Y NOVENTA Y CUATRO CÉNTIMOS DE EURO.

Las Palmas de Gran Canaria, Diciembre de 2012

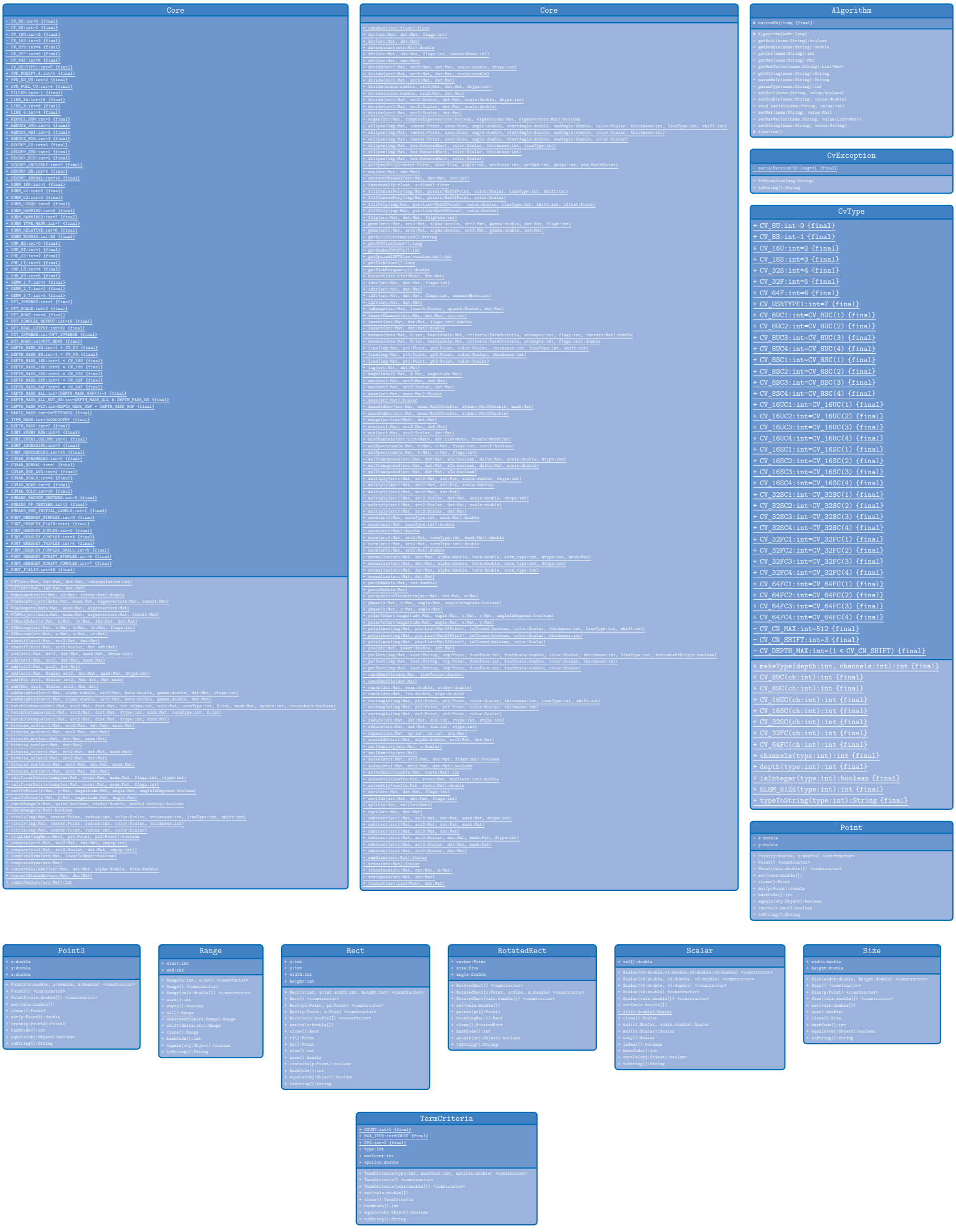
Fdo. Isaac Reyes Villa

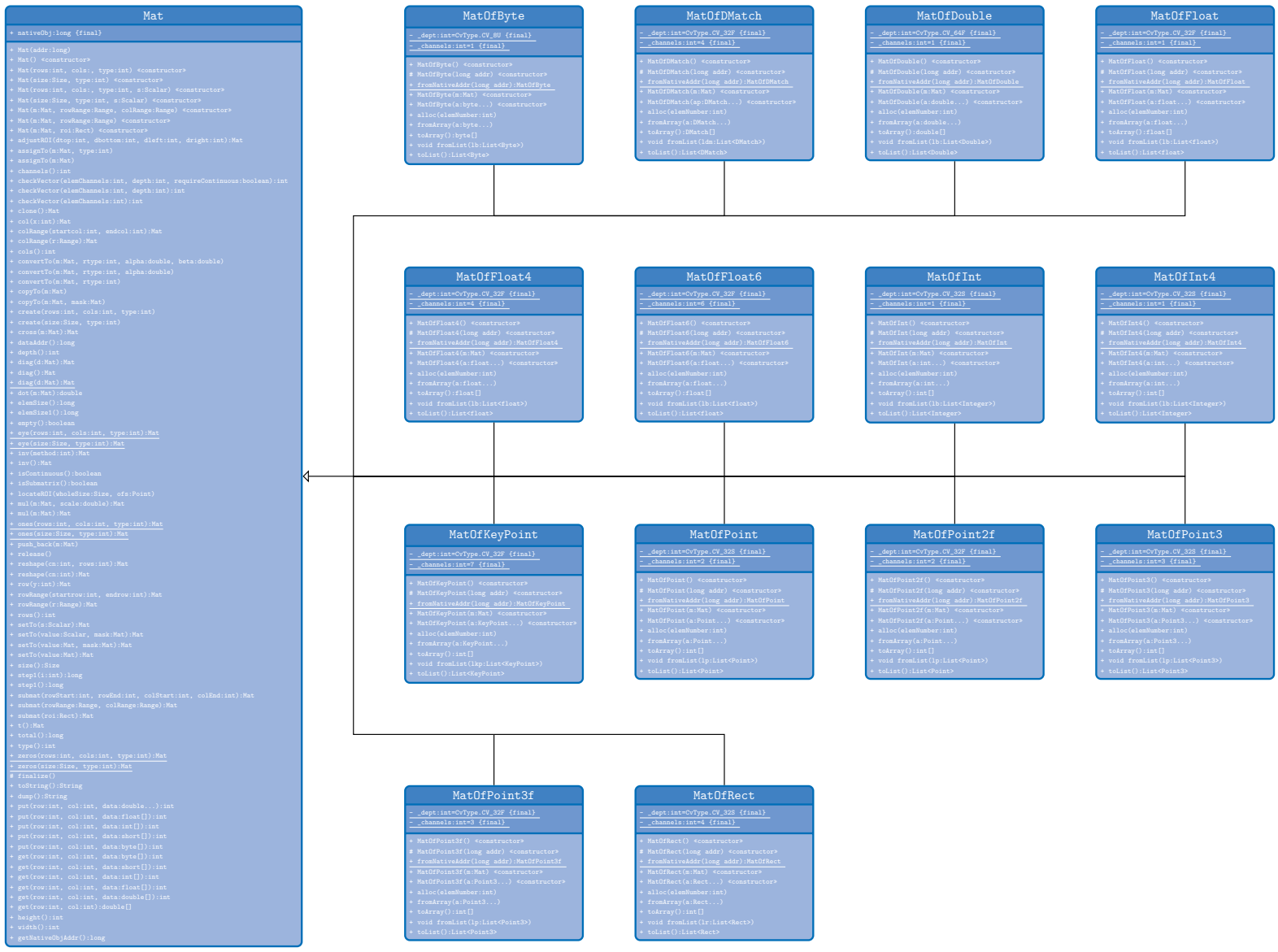
Parte IV.

Anexos

Anexo A

Módulo core





Anexo B

Módulo `imgproc`.

Imgproc

- IPL_BORDER_CONSTANT:int=0 {final}
- IPL_BORDER_REPLICATE:int=1 {final}
- IPL_BORDER_REFLECT:int=2 {final}
- IPL_BORDER_WRAP:int=3 {final}
- IPL_BORDER_REFLECT_101:int=4 {final}
- IPL_BORDER_TRANSPARENT:int=5 {final}
- CV_INTER_NN:int=0 {final}
- CV_INTER_LINEAR:int=1 {final}
- CV_INTER_CUBIC:int=2 {final}
- CV_INTER_AREA:int=3 {final}
- CV_INTER_LANCZOS4:int=4 {final}
- CV_MOP_ERODE:int=0 {final}
- CV_MOP_DILATE:int=1 {final}
- CV_MOP_OPEN:int=2 {final}
- CV_MOP_CLOSE:int=3 {final}
- CV_MOP_GRADIENT:int=4 {final}
- CV_MOP_TOPHAT:int=5 {final}
- CV_MOP_BLACKHAT:int=6 {final}
- CV_RETR_EXTERNAL:int=0 {final}
- CV_RETR_LIST:int=1 {final}
- CV_RETR_CCOMP:int=2 {final}
- CV_RETR_TREE:int=3 {final}
- CV_RETR_FLOODFILL:int=4 {final}
- CV_CHAIN_APPROX_NONE:int=1 {final}
- CV_CHAIN_APPROX_SIMPLE:int=2 {final}
- CV_CHAIN_APPROX_TC89_L1:int=3 {final}
- CV_CHAIN_APPROX_TC89_KCOS:int=4 {final}
- CV_THRESH_BINARY:int=0 {final}
- CV_THRESH_BINARY_INV:int=1 {final}
- CV_THRESH_TRUNC:int=2 {final}
- CV_THRESH_TOZERO:int=3 {final}
- CV_THRESH_TOZERO_INV:int=4 {final}
- CV_THRESH_MASK:int=7 {final}
- CV_THRESH_OTSU:int=8 {final}
+ CV_BLUR_NO_SCALE:int=0 {final}
+ CV_BLUR:int=1 {final}
+ CV_GAUSSIAN:int=2 {final}
+ CV_MEDIAN:int=3 {final}
+ CV_BILATERAL:int=4 {final}
+ CV_GAUSSIAN_5x5:int=7 {final}
+ CV_SCHARR:int=-1 {final}
+ CV_MAX_SOBEL_KSIZE:int=7 {final}
+ CV_WARP_FILL_OUTLIERS:int=8 {final}
+ CV_WARP_INVERSE_MAP:int=16 {final}
+ CV_SHAPE_RECT:int=0 {final}
+ CV_SHAPE_CROSS:int=1 {final}
+ CV_SHAPE_ELLIPSE:int=2 {final}
+ CV_SHAPE_CUSTOM:int=100 {final}
+ CV_CHAIN_CODE:int=0 {final}
+ CV_LINK_RUNS:int=5 {final}
+ CV_POLY_APPROX_DP:int=0 {final}
+ CV_CONTOURS_MATCH_I1:int=1 {final}
+ CV_CONTOURS_MATCH_I2:int=2 {final}
+ CV_CONTOURS_MATCH_I3:int=3 {final}
+ CV_CLOCKWISE:int=1 {final}
+ CV_COUNTER_CLOCKWISE:int=2 {final}
+ CV_COMP_CORREL:int=0 {final}
+ CV_COMP_CHISQR:int=1 {final}
+ CV_COMP_INTERSECT:int=2 {final}
+ CV_COMP_BHATTACHARYYA:int=3 {final}
+ CV_DIST_MASK_3:int=3 {final}
+ CV_DIST_MASK_5:int=5 {final}
+ CV_DIST_MASK_PRECISE:int=0 {final}
+ CV_DIST_LABEL_CCOMP:int=0 {final}
+ CV_DIST_LABEL_PIXEL:int=1 {final}
+ CV_DIST_USER:int=-1 {final}
+ CV_DIST_L1:int=1 {final}
+ CV_DIST_L2:int=2 {final}
+ CV_DIST_C:int=3 {final}
+ CV_DIST_L12:int=4 {final}
+ CV_DIST_FAIR:int=5 {final}
+ CV_DIST_WELSCH:int=6 {final}
+ CV_DIST_HUBER:int=7 {final}
+ CV_CANNY_L2_GRADIENT:int=(1 « 31) {final}
+ CV_HOUGH_STANDARD:int=0 {final}
+ CV_HOUGH_PROBABILISTIC:int=1 {final}
+ CV_HOUGH_MULTI_SCALE:int=2 {final}
+ CV_HOUGH_GRADIENT:int=3 {final}
+ BORDER_REPLICATE:int=IPL_BORDER_REPLICATE {final}
+ BORDER_CONSTANT:int=IPL_BORDER_CONSTANT {final}
+ BORDER_REFLECT:int=IPL_BORDER_REFLECT {final}
+ BORDER_WRAP:int=IPL_BORDER_WRAP {final}
+ BORDER_REFLECT_101:int=IPL_BORDER_REFLECT_101 {final}
+ BORDER_REFLECT101:int=BORDER_REFLECT_101 {final}
+ BORDER_TRANSPARENT:int=IPL_BORDER_TRANSPARENT {final}
+ BORDER_DEFAULT:int=BORDER_REFLECT_101 {final}
+ BORDER_ISOLATED:int=16 {final}
+ KERNEL_GENERAL:int=0 {final}
+ KERNEL_SYMMETRICAL:int=1 {final}
+ KERNEL_ASYMMETRICAL:int=2 {final}
+ KERNEL_SMOOTH:int=4 {final}
+ KERNEL_INTEGER:int=8 {final}
+ MORPH_ERODE:int=CV_MOP_ERODE {final}
+ MORPH_DILATE:int=CV_MOP_DILATE {final}
+ MORPH_OPEN:int=CV_MOP_OPEN {final}
+ MORPH_CLOSE:int=CV_MOP_CLOSE {final}
+ MORPH_GRADIENT:int=CV_MOP_GRADIENT {final}
+ MORPH_TOPHAT:int=CV_MOP_TOPHAT {final}
+ MORPH_BLACKHAT:int=CV_MOP_BLACKHAT {final}
+ MORPH_RECT:int=0 {final}
+ MORPH_CROSS:int=1 {final}
+ MORPH_ELLIPSE:int=2 {final}
+ INTER_NEAREST:int=CV_INTER_NN {final}
+ INTER_LINEAR:int=CV_INTER_LINEAR {final}
+ INTER_CUBIC:int=CV_INTER_CUBIC {final}
+ INTER_AREA:int=CV_INTER_AREA {final}
+ INTER_LANCZOS4:int=CV_INTER_LANCZOS4 {final}
+ INTER_MAX:int=7 {final}
+ WARP_INVERSE_MAP:int=CV_WARP_INVERSE_MAP {final}
+ INTER_BITS:int=5 {final}
+ INTER_BITS2:int=INTER_BITS*2 {final}
+ INTER_TAB_SIZE:int=(1«INTER_BITS) {final}
+ INTER_TAB_SIZE2:int=INTER_TAB_SIZE*INTER_TAB_SIZE {final}
+ THRESH_BINARY:int=CV_THRESH_BINARY {final}
+ THRESH_BINARY_INV:int=CV_THRESH_BINARY_INV {final}
+ THRESH_TRUNC:int=CV_THRESH_TRUNC {final}
+ THRESH_TOZERO:int=CV_THRESH_TOZERO {final}
+ THRESH_TOZERO_INV:int=CV_THRESH_TOZERO_INV {final}
+ THRESH_MASK:int=CV_THRESH_MASK {final}
+ THRESH_OTSU:int=CV_THRESH_OTSU {final}
+ ADAPTIVE_THRESH_MEAN_C:int=0 {final}
+ ADAPTIVE_THRESH_GAUSSIAN_C:int=1 {final}
+ PROJ_SPHERICAL_ORTHO:int=0 {final}
+ PROJ_SPHERICAL_EQRECT:int=1 {final}
+ GC_BGD:int=0 {final}
+ GC_FGD:int=1 {final}
+ GC_PR_BGD:int=2 {final}
+ GC_PR_FGD:int=3 {final}
+ GC_INIT_WITH_RECT:int=0 {final}
+ GC_INIT_WITH_MASK:int=1 {final}
+ GC_EVAL:int=2 {final}
+ DIST_LABEL_CCOMP:int=0 {final}
+ DIST_LABEL_PIXEL:int=1 {final}
+ FLOODFILL_FIXED_RANGE:int=1 « 16 {final}
+ FLOODFILL_MASK_ONLY:int=1 « 17 {final}
+ COLOR_BGR2BGRA:int=0 {final}
+ COLOR_RGB2RGBA:int=COLOR_BGR2BGRA {final}
+ COLOR_BGRA2BGR:int=1 {final}
+ COLOR_RGBA2RGB:int=COLOR_BGRA2BGR {final}
+ COLOR_BGR2RGBA:int=2 {final}
+ COLOR_RGB2BGRA:int=COLOR_BGR2RGBA {final}
+ COLOR_RGBA2BGR:int=3 {final}
+ COLOR_BGRA2RGB:int=COLOR_RGBA2BGR {final}
+ COLOR_BGR2RGB:int=4 {final}

Imgproc

+ COLOR_RGB2BGR:int=COLOR_BGR2RGB {final}
+ COLOR_BGRA2RGBA:int=5 {final}
+ COLOR_RGBA2BGRA:int=COLOR_BGRA2RGBA {final}
+ COLOR_BGR2GRAY:int=6 {final}
+ COLOR_RGB2GRAY:int=7 {final}
+ COLOR_GRAY2BGR:int=8 {final}
+ COLOR_GRAY2RGB:int=COLOR_GRAY2BGR {final}
+ COLOR_GRAY2BGRA:int=9 {final}
+ COLOR_GRAY2RGBA:int=COLOR_GRAY2BGRA {final}
+ COLOR_BGRA2GRAY:int=10 {final}
+ COLOR_RGBA2GRAY:int=11 {final}
+ COLOR_BGR2BGR565:int=12 {final}
+ COLOR_RGB2BGR565:int=13 {final}
+ COLOR_BGR5652BGR:int=14 {final}
+ COLOR_BGR5652RGB:int=15 {final}
+ COLOR_BGRA2BGR565:int=16 {final}
+ COLOR_RGBA2BGR565:int=17 {final}
+ COLOR_BGR5652BGRA:int=18 {final}
+ COLOR_BGR5652RGBA:int=19 {final}
+ COLOR_GRAY2BGR565:int=20 {final}
+ COLOR_BGR5652GRAY:int=21 {final}
+ COLOR_BGR2BGR555:int=22 {final}
+ COLOR_RGB2BGR555:int=23 {final}
+ COLOR_BGR5552BGR:int=24 {final}
+ COLOR_BGR5552RGB:int=25 {final}
+ COLOR_BGRA2BGR555:int=26 {final}
+ COLOR_RGBA2BGR555:int=27 {final}
+ COLOR_BGR5552BGRA:int=28 {final}
+ COLOR_BGR5552RGBA:int=29 {final}
+ COLOR_GRAY2BGR555:int=30 {final}
+ COLOR_BGR5552GRAY:int=31 {final}
+ COLOR_BGR2XYZ:int=32 {final}
+ COLOR_RGB2XYZ:int=33 {final}
+ COLOR_XYZ2BGR:int=34 {final}
+ COLOR_XYZ2RGB:int=35 {final}
+ COLOR_BGR2YCrCb:int=36 {final}
+ COLOR_RGB2YCrCb:int=37 {final}
+ COLOR_YCrCb2BGR:int=38 {final}
+ COLOR_YCrCb2RGB:int=39 {final}
+ COLOR_BGR2HSV:int=40 {final}
+ COLOR_RGB2HSV:int=41 {final}
+ COLOR_BGR2Lab:int=44 {final}
+ COLOR_RGB2Lab:int=45 {final}
+ COLOR_BayerBG2BGR:int=46 {final}
+ COLOR_BayerGB2BGR:int=47 {final}
+ COLOR_BayerRG2BGR:int=48 {final}
+ COLOR_BayerGR2BGR:int=49 {final}
+ COLOR_BayerBG2RGB:int=COLOR_BayerRG2BGR {final}
+ COLOR_BayerGB2RGB:int=COLOR_BayerGR2BGR {final}
+ COLOR_BayerRG2RGB:int=COLOR_BayerBG2BGR {final}
+ COLOR_BayerGR2RGB:int=COLOR_BayerGB2BGR {final}
+ COLOR_BGR2Luv:int=50 {final}
+ COLOR_RGB2Luv:int=51 {final}
+ COLOR_BGR2HLS:int=52 {final}
+ COLOR_RGB2HLS:int=53 {final}
+ COLOR_HSV2BGR:int=54 {final}
+ COLOR_HSV2RGB:int=55 {final}
+ COLOR_Lab2BGR:int=56 {final}
+ COLOR_Lab2RGB:int=57 {final}
+ COLOR_Luv2BGR:int=58 {final}
+ COLOR_Luv2RGB:int=59 {final}
+ COLOR_HLS2BGR:int=60 {final}
+ COLOR_HLS2RGB:int=61 {final}
+ COLOR_BayerBG2BGR_VNG:int=62 {final}
+ COLOR_BayerGB2BGR_VNG:int=63 {final}
+ COLOR_BayerRG2BGR_VNG:int=64 {final}
+ COLOR_BayerGR2BGR_VNG:int=65 {final}
+ COLOR_BayerBG2RGB_VNG:int=COLOR_BayerRG2BGR_VNG {final}
+ COLOR_BayerGB2RGB_VNG:int=COLOR_BayerGR2BGR_VNG {final}
+ COLOR_BayerRG2RGB_VNG:int=COLOR_BayerBG2BGR_VNG {final}
+ COLOR_BayerGR2RGB_VNG:int=COLOR_BayerGB2BGR_VNG {final}
+ COLOR_BGR2HSV_FULL:int=66 {final}
+ COLOR_RGB2HSV_FULL:int=67 {final}
+ COLOR_BGR2HLS_FULL:int=68 {final}
+ COLOR_RGB2HLS_FULL:int=69 {final}
+ COLOR_HSV2BGR_FULL:int=70 {final}
+ COLOR_HSV2RGB_FULL:int=71 {final}
+ COLOR_HLS2BGR_FULL:int=72 {final}
+ COLOR_HLS2RGB_FULL:int=73 {final}
+ COLOR_LBGR2Lab:int=74 {final}
+ COLOR_LRGB2Lab:int=75 {final}
+ COLOR_LBGR2Luv:int=76 {final}
+ COLOR_LRGB2Luv:int=77 {final}
+ COLOR_Lab2LBGR:int=78 {final}
+ COLOR_Lab2LRGB:int=79 {final}
+ COLOR_Luv2LBGR:int=80 {final}
+ COLOR_Luv2LRGB:int=81 {final}
+ COLOR_BGR2YUV:int=82 {final}
+ COLOR_RGB2YUV:int=83 {final}
+ COLOR_YUV2BGR:int=84 {final}
+ COLOR_YUV2RGB:int=85 {final}
+ COLOR_BayerBG2GRAY:int=86 {final}
+ COLOR_BayerGB2GRAY:int=87 {final}
+ COLOR_BayerRG2GRAY:int=88 {final}
+ COLOR_BayerGR2GRAY:int=89 {final}
+ COLOR_YUV2BGR_NV12:int=90 {final}
+ COLOR_YUV2BGR_NV12:int=91 {final}
+ COLOR_YUV2BGR_NV21:int=92 {final}
+ COLOR_YUV2BGR_NV21:int=93 {final}
+ COLOR_YUV420sp2RGB:int=COLOR_YUV2RGB_NV21 {final}
+ COLOR_YUV420sp2BGR:int=COLOR_YUV2BGR_NV21 {final}
+ COLOR_YUV2RGBA_NV12:int=94 {final}
+ COLOR_YUV2BGRA_NV12:int=95 {final}
+ COLOR_YUV2RGBA_NV21:int=96 {final}
+ COLOR_YUV2BGRA_NV21:int=97 {final}
+ COLOR_YUV420sp2RGBA:int=COLOR_YUV2RGBA_NV21 {final}
+ COLOR_YUV420sp2BGRA:int=COLOR_YUV2BGRA_NV21 {final}
+ COLOR_YUV2RGB_YV12:int=98 {final}
+ COLOR_YUV2BGR_YV12:int=99 {final}
+ COLOR_YUV2RGB_IYUV:int=100 {final}
+ COLOR_YUV2BGR_IYUV:int=101 {final}
+ COLOR_YUV2RGB_I420:int=COLOR_YUV2RGB_IYUV {final}
+ COLOR_YUV2BGR_I420:int=COLOR_YUV2BGR_IYUV {final}
+ COLOR_YUV420p2RGB:int=COLOR_YUV2RGB_YV12 {final}
+ COLOR_YUV420p2BGR:int=COLOR_YUV2BGR_YV12 {final}
+ COLOR_YUV2RGBA_YV12:int=102 {final}
+ COLOR_YUV2BGRA_YV12:int=103 {final}
+ COLOR_YUV2RGBA_IYUV:int=104 {final}
+ COLOR_YUV2BGRA_IYUV:int=105 {final}
+ COLOR_YUV2RGBA_I420:int=COLOR_YUV2RGBA_IYUV {final}
+ COLOR_YUV2BGRA_I420:int=COLOR_YUV2BGRA_IYUV {final}
+ COLOR_YUV420p2RGBA:int=COLOR_YUV2RGBA_YV12 {final}
+ COLOR_YUV420p2BGRA:int=COLOR_YUV2BGRA_YV12 {final}
+ COLOR_YUV2GRAY_420:int=106 {final}
+ COLOR_YUV2GRAY_NV21:int=COLOR_YUV2GRAY_420 {final}
+ COLOR_YUV2GRAY_NV12:int=COLOR_YUV2GRAY_420 {final}
+ COLOR_YUV2GRAY_YV12:int=COLOR_YUV2GRAY_420 {final}
+ COLOR_YUV2GRAY_IYUV:int=COLOR_YUV2GRAY_420 {final}
+ COLOR_YUV2GRAY_I420:int=COLOR_YUV2GRAY_420 {final}
+ COLOR_YUV420sp2GRAY:int=COLOR_YUV2GRAY_420 {final}
+ COLOR_YUV420p2GRAY:int=COLOR_YUV2GRAY_420 {final}
+ COLOR_YUV2RGB_UYVY:int=107 {final}
+ COLOR_YUV2BGR_UYVY:int=108 {final}
+ COLOR_YUV2RGB_Y422:int=COLOR_YUV2RGB_UYVY {final}
+ COLOR_YUV2BGR_Y422:int=COLOR_YUV2BGR_UYVY {final}
+ COLOR_YUV2RGB_UYNV:int=COLOR_YUV2RGB_UYVY {final}

Imgproc

+ COLOR_YUV2BGR_UYNV:int=COLOR_YUV2BGR_UYVY {final}
+ COLOR_YUV2RGBA_UYVY:int=111 {final}
+ COLOR_YUV2BGRA_UYVY:int=112 {final}
+ COLOR_YUV2RGBA_Y422:int=COLOR_YUV2RGBA_UYVY {final}
+ COLOR_YUV2BGRA_Y422:int=COLOR_YUV2BGRA_UYVY {final}
+ COLOR_YUV2RGBA_UYNV:int=COLOR_YUV2RGBA_UYVY {final}
+ COLOR_YUV2BGRA_UYNV:int=COLOR_YUV2BGRA_UYVY {final}
+ COLOR_YUV2RGB_YUY2:int=115 {final}
+ COLOR_YUV2BGR_YUY2:int=116 {final}
+ COLOR_YUV2RGB_YVYU:int=117 {final}
+ COLOR_YUV2BGR_YVYU:int=118 {final}
+ COLOR_YUV2RGB_YUYV:int=COLOR_YUV2RGB_YUY2 {final}
+ COLOR_YUV2BGR_YUYV:int=COLOR_YUV2BGR_YUY2 {final}
+ COLOR_YUV2RGB_YUNV:int=COLOR_YUV2RGB_YUY2 {final}
+ COLOR_YUV2BGR_YUNV:int=COLOR_YUV2BGR_YUY2 {final}
+ COLOR_YUV2RGBA_YUY2:int=119 {final}
+ COLOR_YUV2BGRA_YUY2:int=120 {final}
+ COLOR_YUV2RGBA_YVYU:int=121 {final}
+ COLOR_YUV2BGRA_YVYU:int=122 {final}
+ COLOR_YUV2RGBA_YUYV:int=COLOR_YUV2RGBA_YUY2 {final}
+ COLOR_YUV2BGRA_YUYV:int=COLOR_YUV2BGRA_YUY2 {final}
+ COLOR_YUV2RGBA_YUNV:int=COLOR_YUV2RGBA_YUY2 {final}
+ COLOR_YUV2BGRA_YUNV:int=COLOR_YUV2BGRA_YUY2 {final}
+ COLOR_YUV2GRAY_UYVY:int=123 {final}
+ COLOR_YUV2GRAY_YUY2:int=124 {final}
+ COLOR_YUV2GRAY_Y422:int=COLOR_YUV2GRAY_UYVY {final}
+ COLOR_YUV2GRAY_UYNV:int=COLOR_YUV2GRAY_UYVY {final}
+ COLOR_YUV2GRAY_YVYU:int=COLOR_YUV2GRAY_YUY2 {final}
+ COLOR_YUV2GRAY_YUYV:int=COLOR_YUV2GRAY_YUY2 {final}
+ COLOR_YUV2GRAY_YUNV:int=COLOR_YUV2GRAY_YUY2 {final}
+ COLOR_COLORCVT_MAX:int=125 {final}
+ TM_SQDIFF:int=0 {final}
+ TM_SQDIFF_NORMED:int=1 {final}
+ TM_CCORR:int=2 {final}
+ TM_CCORR_NORMED:int=3 {final}
+ TM_CCOEFF:int=4 {final}
+ TM_CCOEFF_NORMED:int=5 {final}
+ RETR_EXTERNAL:int=CV_RETR_EXTERNAL {final}
+ RETR_LIST:int=CV_RETR_LIST {final}
+ RETR_CCOMP:int=CV_RETR_CCOMP {final}
+ RETR_TREE:int=CV_RETR_TREE {final}
+ RETR_FLOODFILL:int=CV_RETR_FLOODFILL {final}
+ CHAIN_APPROX_NONE:int=CV_CHAIN_APPROX_NONE {final}
+ CHAIN_APPROX_SIMPLE:int=CV_CHAIN_APPROX_SIMPLE {final}
+ CHAIN_APPROX_TC89_L1:int=CV_CHAIN_APPROX_TC89_L1 {final}
+ CHAIN_APPROX_TC89_KCOS:int=CV_CHAIN_APPROX_TC89_KCOS {final}

+ Canny(image:Mat, edges:Mat, threshold1:double, threshold2:double, apertureSize:int, L2gradient:boolean)
+ Canny(image:Mat, edges:Mat, threshold1:double, threshold2:double)
+ GaussianBlur(src:Mat, dst:Mat, ksize:Size, sigmaX:double, sigmaY:double, borderType:int)
+ GaussianBlur(src:Mat, dst:Mat, ksize:Size, sigmaX:double, sigmaY:double)
+ GaussianBlur(src:Mat, dst:Mat, ksize:Size, sigmaX:double)
+ HoughCircles(image:Mat, circles:Mat, method:int, dp:double, minDist:double, param1:double, param2:double, minRadius:int , maxRadius:int)
+ HoughCircles(image:Mat, circles:Mat, method:int, dp:double, minDist:double)
+ HoughLines(image:Mat, lines:Mat, rho:double, theta:double, threshold:int, srn:double, stn:double)
+ HoughLines(image:Mat, lines:Mat, rho:double, theta:double, threshold:int)
+ HoughLinesP(image:Mat, lines:Mat, rho:double, theta:double, threshold:int, minLineLength:double, maxLineGap:double)
+ HoughLinesP(image:Mat, lines:Mat, rho:double, theta:double, threshold:int)
+ HuMoments(m:Moments, hu:Mat)
+ Laplacian(src:Mat, dst:Mat, ddepth:int, ksize:int, scale:double, delta:double, borderType:int)
+ Laplacian(src:Mat, dst:Mat, ddepth:int, ksize:int, scale:double, delta:double)
+ Laplacian(src:Mat, dst:Mat, ddepth:int)
+ PSNR(src1:Mat, src2:Mat):double
+ Scharr(src:Mat, dst:Mat, ddepth:int, dx:int, dy:int, scale:double, delta:double, borderType:int)
+ Scharr(src:Mat, dst:Mat, ddepth:int, dx:int, dy:int, scale:double, delta:double)
+ Scharr(src:Mat, dst:Mat, ddepth:int, dx:int, dy:int)
+ Sobel(src:Mat, dst:Mat, ddepth:int, dx:int, dy:int, ksize:int, scale:double, delta:double, borderType:int)
+ Sobel(src:Mat, dst:Mat, ddepth:int, dx:int, dy:int, ksize:int, scale:double, delta:double)
+ Sobel(src:Mat, dst:Mat, ddepth:int, dx:int, dy:int)
+ accumulate(src:Mat, dst:Mat, mask:Mat)
+ accumulate(src:Mat, dst:Mat)
+ accumulateProduct(src1:Mat, src2:Mat, dst:Mat, mask:Mat)
+ accumulateProduct(src1:Mat, src2:Mat, dst:Mat)
+ accumulateSquare(src:Mat, dst:Mat, mask:Mat)
+ accumulateSquare(src:Mat, dst:Mat)
+ accumulateWeighted(src:Mat, dst:Mat, alpha:double, mask:Mat)
+ accumulateWeighted(src:Mat, dst:Mat, alpha:double)
+ adaptiveThreshold(src:Mat, dst:Mat, double maxValue:, adaptiveMethod:int, thresholdType:int, blockSize:int, C:double)
+ approxPolyDP(curve:MatOfPoint2f, approxCurve:MatOfPoint2f, epsilon:double, closed:boolean)
+ arcLength(curve:MatOfPoint2f, closed:boolean):double
+ bilateralFilter(src:Mat, dst:Mat, d:int, sigmaColor:double, sigmaSpace:double, borderType:int)
+ bilateralFilter(src:Mat, dst:Mat, d:int, sigmaColor:double, sigmaSpace:double)
+ blur(src:Mat, dst:Mat, ksize:Size, anchor:Point, borderType:int)
+ blur(src:Mat, dst:Mat, ksize:Size, anchor:Point)
+ blur(src:Mat, dst:Mat, ksize:Size)
+ borderInterpolate(p:int, len:int, borderType:int):int
+ boundingRect(points:MatOfPoint):Rect
+ boxFilter(src:Mat, dst:Mat, ddepth:int, ksize:Size, anchor:Point, normalize:boolean, borderType:int)
+ boxFilter(src:Mat, dst:Mat, ddepth:int, ksize:Size, anchor:Point, normalize:boolean)
+ boxFilter(src:Mat, dst:Mat, ddepth:int, ksize:Size)
+ calcBackProject(images:List<Mat>, channels:MatOfInt, hist:Mat, dst:Mat, ranges:MatOfFloat, scale:double)
+ calcHist(images:List<Mat>, channels:MatOfInt, mask:Mat, hist:Mat, histSize:MatOfInt, ranges:MatOfFloat, accumulate:boolean)
+ calcHist(images:List<Mat>, channels:MatOfInt, mask:Mat, hist:Mat, histSize:MatOfInt, ranges:MatOfFloat)
+ compareHist(H1:, H2:Mat, method:int):double
+ contourArea(contour:Mat, oriented:boolean):double
+ contourArea(contour:Mat):double
+ convertMaps(map1:Mat, map2:Mat, dstmap1:Mat, dstmap2:Mat, dstmap1type:int, nninterpolation:boolean)
+ convertMaps(map1:Mat, map2:Mat, dstmap1:Mat, dstmap2:Mat, dstmap1type:int)
+ convexHull(points:MatOfPoint, hull:MatOfInt, clockwise:boolean)
+ convexHull(points:MatOfPoint, hull:MatOfInt)
+ convexityDefects(contour:MatOfPoint, convexhull:MatOfInt, convexityDefects:MatOfInt4)
+ copyMakeBorder(src:Mat, dst:Mat, top:int, bottom:int, left:int, right:int, borderType:int, value:Scalar)
+ copyMakeBorder(src:Mat, dst:Mat, top:int, bottom:int, left:int, right:int, borderType:int)
+ cornerEigenValsAndVecs(src:Mat, dst:Mat, blockSize:int, ksize:int, borderType:int)
+ cornerEigenValsAndVecs(src:Mat, dst:Mat, blockSize:int, ksize:int)
+ cornerHarris(src:Mat, dst:Mat, blockSize:int, ksize:int, k:double, borderType:int)
+ cornerHarris(src:Mat, dst:Mat, blockSize:int, ksize:int, k:double)
+ cornerMinEigenVal(src:Mat, dst:Mat, blockSize:int, ksize:int, borderType:int)
+ cornerMinEigenVal(src:Mat, dst:Mat, blockSize:int, ksize:int)
+ cornerMinEigenVal(src:Mat, dst:Mat, blockSize:int)
+ cornerSubPix(image:Mat, corners:MatOfPoint2f, winSize:Size, zeroZone:Size, criteria:TermCriteria)
+ createHanningWindow(dst:Mat, winSize:Size, type:int)
+ cvtColor(src:Mat, dst:Mat, code:int, dstCn:int)
+ cvtColor(src:Mat, dst:Mat, code:int)
+ dilate(src:Mat, dst:Mat, kernel:Mat, anchor:Point, iterations:int, borderType:int, borderValue:Scalar)
+ dilate(src:Mat, dst:Mat, kernel:Mat, anchor:Point, iterations:int)
+ dilate(src:Mat, dst:Mat, kernel:Mat)
+ distanceTransform(src:Mat, dst:Mat, distanceType:int, maskSize:int)
+ distanceTransformWithLabels(src:Mat, dst:Mat, labels:Mat, distanceType:int, maskSize:int, labelType:int)
+ distanceTransformWithLabels(src:Mat, dst:Mat, labels:Mat, distanceType:int, maskSize:int)
+ drawContours(image:Mat, contours:List<MatOfPoint>, contourIdx:int, color:Scalar, thickness:int, lineType:int, hierarchy:Mat, maxLevel:int, offset:Point)
+ drawContours(image:Mat, contours:List<MatOfPoint>, contourIdx:int, color:Scalar, thickness:int)
+ drawContours(image:Mat, contours:List<MatOfPoint>, contourIdx:int, color:Scalar)
+ equalizeHist(src:Mat, dst:Mat)

Imgproc

+ erode(src:Mat, dst:Mat, kernel:Mat, anchor:Point, iterations:int, borderType:int, borderValue:Scalar)
+ erode(src:Mat, dst:Mat, kernel:Mat, anchor:Point, iterations:int)
+ erode(src:Mat, dst:Mat, kernel:Mat)
+ filter2D(src:Mat, dst:Mat, ddepth:int, kernel:Mat, anchor:Point, delta:double, borderType:int)
+ filter2D(src:Mat, dst:Mat, ddepth:int, kernel:Mat, anchor:Point, delta:double)
+ filter2D(src:Mat, dst:Mat, ddepth:int, kernel:Mat)
+ findContours(image:Mat, contours:List<MatOfPoint>, hierarchy:Mat, mode:int, method:int, Point offset:)
+ findContours(image:Mat, contours:List<MatOfPoint>, hierarchy:Mat, mode:int, method:int)
+ fitEllipse(points:MatOfPoint2f):RotatedRect
+ fitLine(points:Mat, line:Mat, distType:int, param:double, reps:double, aeps:double)
+ floodFill(image:Mat, mask:Mat, seedPoint:Point, newVal:Scalar, rect:Rect, loDiff:Scalar, upDiff:Scalar, flags:int):int
+ floodFill(image:Mat, mask:Mat, seedPoint:Point, newVal:Scalar):int
+ getAffineTransform(src:MatOfPoint2f, dst:MatOfPoint2f):Mat
+ getDefaultNewCameraMatrix(cameraMatrix:Mat, imgsize:Size, centerPrincipalPoint:boolean):Mat
+ getDefaultNewCameraMatrix(cameraMatrix:Mat):Mat
+ getDerivKernels(kx:Mat, ky:Mat, dx:int, dy:int, ksize:int, normalize:boolean, ktype:int)
+ getDerivKernels(kx:Mat, ky:Mat, dx:int, dy:int, ksize:int)
+ getGaborKernel(ksize:Size, sigma:double, theta:double, lambd:double, gamma:double, psi:double, ktype:int):Mat
+ getGaborKernel(ksize:Size, sigma:double, theta:double, lambd:double, gamma:double):Mat
+ getGaussianKernel(ksize:int, sigma:double, ktype:int):Mat
+ getGaussianKernel(ksize:int, sigma:double):Mat
+ getPerspectiveTransform(src:Mat, dst:Mat):Mat
+ getRectSubPix(image:Mat, patchSize:Size, center:Point, patch:Mat, patchType:int)
+ getRectSubPix(image:Mat, patchSize:Size, center:Point, patch:Mat)
+ getRotationMatrix2D(center:Point, angle:double, scale:double):Mat
+ getStructuringElement(shape:int, ksize:Size, anchor:Point):Mat
+ getStructuringElement(shape:int, ksize:Size):Mat
+ goodFeaturesToTrack(image:Mat, corners:MatOfPoint, maxCorners:int, qualityLevel:double, minDistance:double, mask:Mat, blockSize:int, useHarrisDetector:boolean, k:double)
+ goodFeaturesToTrack(image:Mat, corners:MatOfPoint, maxCorners:int, qualityLevel:double, minDistance:double)
+ grabCut(img:Mat, mask:Mat, rect:Rect, bgdModel:Mat, fgdModel:Mat, iterCount:int, mode:int)
+ grabCut(img:Mat, mask:Mat, rect:Rect, bgdModel:Mat, fgdModel:Mat, iterCount:int)
+ initUndistortRectifyMap(cameraMatrix:Mat, distCoeffs:Mat , R:Mat, newCameraMatrix:Mat, size:Size, m1type:int, map1:Mat, map2:Mat)
+ initWideAngleProjMap(cameraMatrix:Mat, distCoeffs:Mat, imageSize:Size, destImageWidth:int, m1type:int, map1:Mat, map2:Mat, projType:int, alpha:double):float
+ initWideAngleProjMap(cameraMatrix:Mat, distCoeffs:Mat, imageSize:Size, destImageWidth:int, m1type:int, map1:Mat, map2:Mat):float
+ integral(src:Mat, sum:Mat, sdepth:int)
+ integral(src:Mat, sum:Mat)
+ integral2(src:Mat, sum:Mat, sqsum:Mat, sdepth:int)
+ integral2(src:Mat, sum:Mat, sqsum:Mat)
+ integral3(src:Mat, sum:Mat, sqsum:Mat, tilted:Mat, sdepth:int)
+ integral3(src:Mat, sum:Mat, sqsum:Mat, tilted:Mat)
+ intersectConvexConvex(_p1:Mat, _p2:Mat, _p12:Mat, handleNested:boolean):float
+ intersectConvexConvex(_p1:Mat, _p2:Mat, _p12:Mat):float
+ invertAffineTransform(M:Mat, iM:Mat)
+ isContourConvex(contour:MatOfPoint):boolean
+ matchShapes(contour1:Mat, contour2:Mat, method:int, parameter:double):double
+ matchTemplate(image:Mat, templ:Mat, result:Mat, method:int)
+ medianBlur(src:Mat, dst:Mat, ksize:int)
+ minAreaRect(points:MatOfPoint2f):RotatedRect
+ minEnclosingCircle(points:MatOfPoint2f, center:Point, radius:float[])
+ moments(array:Mat, binaryImage:boolean):Moments
+ moments(array:Mat):Moments
+ morphologyEx(src:Mat, dst:Mat, op:int, kernel:Mat, anchor:Point, iterations:int, borderType:int, borderValue:Scalar)
+ morphologyEx(src:Mat, dst:Mat, op:int, kernel:Mat, anchor:Point, iterations:int)
+ morphologyEx(src:Mat, dst:Mat, op:int, kernel:Mat)
+ phaseCorrelate(src1:Mat, src2:Mat, window:Mat):Point
+ phaseCorrelate(src1:Mat, src2:Mat):Point
+ pointPolygonTest(contour:MatOfPoint2f, pt:Point, measureDist:boolean):double
+ preCornerDetect(src:Mat, dst:Mat, ksize:int, borderType:int)
+ preCornerDetect(src:Mat, dst:Mat, ksize:int)
+ pyrDown(src:Mat, dst:Mat, dstsize:Size, borderType:int)
+ pyrDown(src:Mat, dst:Mat, dstsize:Size)
+ pyrDown(src:Mat, dst:Mat)
+ pyrMeanShiftFiltering(src:Mat, dst:Mat, sp:double, sr:double, maxLevel:int, termcrit:TermCriteria)
+ pyrMeanShiftFiltering(src:Mat, dst:Mat, sp:double, sr:double)
+ pyrUp(src:Mat, dst:Mat, dstsize:Size, borderType:int)
+ pyrUp(src:Mat, dst:Mat, dstsize:Size)
+ pyrUp(src:Mat, dst:Mat)
+ remap(src:Mat, dst:Mat, map1:Mat, map2:Mat, interpolation:int, borderMode:int, borderValue:Scalar)
+ remap(src:Mat, dst:Mat, map1:Mat, map2:Mat, interpolation:int)
+ resize(src:Mat, dst:Mat, Size dsize:, fx:double, fy:double, interpolation:int)
+ resize(src:Mat, dst:Mat, Size dsize:)
+ sepFilter2D(src:Mat, dst:Mat, ddepth:int, kernelX:Mat, kernelY:Mat, anchor:Point, delta:double, borderType:int)
+ sepFilter2D(src:Mat, dst:Mat, ddepth:int, kernelX:Mat, kernelY:Mat, anchor:Point, delta:double)
+ sepFilter2D(src:Mat, dst:Mat, ddepth:int, kernelX:Mat, kernelY:Mat)
+ threshold(src:Mat, dst:Mat, thresh:double, maxval:double, type:int):double
+ undistort(src:Mat, dst:Mat, cameraMatrix:Mat, distCoeffs:Mat, newCameraMatrix:Mat)
+ undistort(src:Mat, dst:Mat, cameraMatrix:Mat, distCoeffs:Mat)
+ undistortPoints(src:MatOfPoint2f, dst:MatOfPoint2f, cameraMatrix:Mat, distCoeffs:Mat, R:Mat, P:Mat)
+ undistortPoints(src:MatOfPoint2f, dst:MatOfPoint2f, cameraMatrix:Mat, distCoeffs:Mat)
+ warpAffine(src:Mat, dst:Mat, Mat M:, dsize:Size, flags:int, borderMode:int, borderValue:Scalar)
+ warpAffine(src:Mat, dst:Mat, Mat M:, dsize:Size, flags:int)
+ warpAffine(src:Mat, dst:Mat, Mat M:, dsize:Size)
+ warpPerspective(src:Mat, dst:Mat, Mat M:, dsize:Size, flags:int, borderMode:int, borderValue:Scalar)
+ warpPerspective(src:Mat, dst:Mat, Mat M:, dsize:Size, flags:int)
+ warpPerspective(src:Mat, dst:Mat, Mat M:, dsize:Size)
+ watershed(image:Mat, markers:Mat)

Moments

nativeObj:long {final}

+ Moments() «constructor»
+ get_m00():double
+ set_m00(m00:double)
+ get_m10():double
+ set_m10(m10:double)
+ get_m01():double
+ set_m01(m01:double)
+ get_m20():double
+ set_m20(m20:double)
+ get_m11():double
+ set_m11(m11:double)
+ get_m02():double
+ set_m02(m02:double)
+ get_m30():double
+ set_m30(m30:double)
+ get_m21():double
+ set_m21(m21:double)
+ get_m12():double
+ set_m12(m12:double)
+ get_m03():double
+ set_m03(m03:double)
+ get_mu20():double
+ set_mu20(mu20:double)
+ get_mu11():double
+ set_mu11(mu11:double)
+ get_mu11():double
+ set_mu11(mu11:double)
+ get_mu02():double
+ set_mu02(mu02:double)
+ get_mu30():double
+ set_mu30(mu30:double)
+ get_mu21():double
+ set_mu21(mu21:double)
+ get_mu12():double
+ set_mu12(mu12:double)
+ get_mu03():double
+ set_mu03(mu03:double)
+ get_nu20():double
+ set_nu20(nu20:double)
+ get_nu11():double
+ set_nu11(nu11:double)
+ get_nu02():double
+ set_nu02(nu02:double)
+ get_nu30():double
+ set_nu30(nu30:double)
+ get_nu21():double
+ set_nu21(nu21:double)
+ get_nu12():double
+ set_nu12(nu12:double)
+ get_nu03():double
+ set_nu03(nu03:double)

Subdiv2D

nativeObj:long {final}
+ PTLOC_ERROR:int=-2 {final}
+ PTLOC_OUTSIDE_RECT:int=-1 {final}
+ PTLOC_INSIDE:int=0 {final}
+ PTLOC_VERTEX:int=1 {final}
+ PTLOC_ON_EDGE:int=2 {final}
+ NEXT_AROUND_ORG:int=0x00 {final}
+ NEXT_AROUND_DST:int=0x22 {final}
+ PREV_AROUND_ORG:int=0x11 {final}
+ PREV_AROUND_DST:int=0x33 {final}
+ NEXT_AROUND_LEFT:int=0x13 {final}
+ NEXT_AROUND_RIGHT:int=0x31 {final}
+ PREV_AROUND_LEFT:int=0x20 {final}
+ PREV_AROUND_RIGHT:int=0x02 {final}

+ Subdiv2D() «constructor»
+ Subdiv2D(Rect rect) «constructor»
+ edgeDst(edge:int, dstpt:Point):int
+ edgeDst(edge:int):int
+ edgeOrg(edge:int, orgpt:Point):int
+ edgeOrg(edge:int):int
+ findNearest(pt:Point, nearestPt:Point):int
+ findNearest(pt:Point):int
+ getEdge(edge:int, nextEdgeType:int):int
+ getEdgeList(edgeList:MatOfFloat4)
+ getTriangleList(triangleList:MatOfFloat6)
+ getVertex(vertex:int, firstEdge:int[]):Point
+ getVertex(vertex:int):Point
+ getVoronoiFacetList(idx:MatOfInt, facetList:List<MatOfPoint2f>, facetCenters:MatOfPoint2f)
+ initDelaunay(rect: Rect)
+ insert(pt:Point):int
+ insert(ptvec:MatOfPoint2f)
+ locate(pt:Point, edge:int[], vertex:int[]):int
+ int nextEdge(edge:int):int
+ int rotateEdge(edge:int, rotate:int):int
+ int symEdge(edge:int):int
# finalize()

Anexo C

Módulo highgui.

Highgui

+ CV_FONT_LIGHT:int=25 {final}
+ CV_FONT_NORMAL:int=50 {final}
+ CV_FONT_DEMIBOLD:int=63 {final}
+ CV_FONT_BOLD:int=75 {final}
+ CV_FONT_BLACK:int=87 {final}
+ CV_STYLE_NORMAL:int=0 {final}
+ CV_STYLE_ITALIC:int=1 {final}
+ CV_STYLE_OBLIQUE:int=2 {final}
+ CV_LOAD_IMAGE_UNCHANGED:int=-1 {final}
+ CV_LOAD_IMAGE_GRAYSCALE:int=0 {final}
+ CV_LOAD_IMAGE_COLOR:int=1 {final}
+ CV_LOAD_IMAGE_ANYDEPTH:int=2 {final}
+ CV_LOAD_IMAGE_ANYCOLOR:int=4 {final}
+ CV_IMWRITE_JPEG_QUALITY:int=1 {final}
+ CV_IMWRITE_PNG_COMPRESSION:int=16 {final}
+ CV_IMWRITE_PNG_STRATEGY:int=17 {final}
+ CV_IMWRITE_PNG_STRATEGY_DEFAULT:int=0 {final}
+ CV_IMWRITE_PNG_STRATEGY_FILTERED:int=1 {final}
+ CV_IMWRITE_PNG_STRATEGY_HUFFMAN_ONLY:int=2 {final}
+ CV_IMWRITE_PNG_STRATEGY_RLE:int=3 {final}
+ CV_IMWRITE_PNG_STRATEGY_FIXED:int=4 {final}
+ CV_IMWRITE_PXM_BINARY:int=32 {final}
+ CV_CVTIMG_FLIP:int=1 {final}
+ CV_CVTIMG_SWAP_RB:int=2 {final}
+ CV_CAP_ANDROID:int=1000 {final}
+ CV_CAP_XIAPI:int=1100 {final}
+ CV_CAP_AVFOUNDATION:int=1200 {final}
+ CV_CAP_PROP_FRAME_WIDTH:int=3 {final}
+ CV_CAP_PROP_FRAME_HEIGHT:int=4 {final}
+ CV_CAP_PROP_ZOOM:int=27 {final}
+ CV_CAP_PROP_FOCUS:int=28 {final}
+ CV_CAP_PROP_GUID:int=29 {final}
+ CV_CAP_PROP_ISO_SPEED:int=30 {final}
+ CV_CAP_PROP_BACKLIGHT:int=32 {final}
+ CV_CAP_PROP_PAN:int=33 {final}
+ CV_CAP_PROP_TILT:int=34 {final}
+ CV_CAP_PROP_ROLL:int=35 {final}
+ CV_CAP_PROP_IRIS:int=36 {final}
+ CV_CAP_PROP_SETTINGS:int=37 {final}
+ CV_CAP_PROP_AUTOGRAB:int=1024 {final}
+ CV_CAP_PROP_PREVIEW_FORMAT:int=1026 {final}
+ CV_CAP_PROP_XI_DOWNSAMPLING:int=400 {final}
+ CV_CAP_PROP_XI_DATA_FORMAT:int=401 {final}
+ CV_CAP_PROP_XI_OFFSET_X:int=402 {final}
+ CV_CAP_PROP_XI_OFFSET_Y:int=403 {final}
+ CV_CAP_PROP_XI_TRG_SOURCE:int=404 {final}
+ CV_CAP_PROP_XI_TRG_SOFTWARE:int=405 {final}
+ CV_CAP_PROP_XI_GPI_SELECTOR:int=406 {final}
+ CV_CAP_PROP_XI_GPI_MODE:int=407 {final}
+ CV_CAP_PROP_XI_GPI_LEVEL:int=408 {final}
+ CV_CAP_PROP_XI_GPO_SELECTOR:int=409 {final}
+ CV_CAP_PROP_XI_GPO_MODE:int=410 {final}
+ CV_CAP_PROP_XI_LED_SELECTOR:int=411 {final}
+ CV_CAP_PROP_XI_LED_MODE:int=412 {final}
+ CV_CAP_PROP_XI_MANUAL_WB:int=413 {final}
+ CV_CAP_PROP_XI_AUTO_WB:int=414 {final}
+ CV_CAP_PROP_XI_AEAG:int=415 {final}
+ CV_CAP_PROP_XI_EXP_PRIORITY:int=416 {final}
+ CV_CAP_PROP_XI_AE_MAX_LIMIT:int=417 {final}
+ CV_CAP_PROP_XI_AG_MAX_LIMIT:int=418 {final}
+ CV_CAP_PROP_XI_AEAG_LEVEL:int=419 {final}
+ CV_CAP_PROP_XI_TIMEOUT:int=420 {final}
+ CV_CAP_PROP_ANDROID_FLASH_MODE:int=8001 {final}
+ CV_CAP_PROP_ANDROID_FOCUS_MODE:int=8002 {final}
+ CV_CAP_PROP_ANDROID_WHITE_BALANCE:int=8003 {final}
+ CV_CAP_PROP_ANDROID_ANTIBANDING:int=8004 {final}
+ CV_CAP_PROP_ANDROID_FOCAL_LENGTH:int=8005 {final}
+ CV_CAP_PROP_ANDROID_FOCUS_DISTANCE_NEAR:int=8006 {final}
+ CV_CAP_PROP_ANDROID_FOCUS_DISTANCE_OPTIMAL:int=8007 {final}
+ CV_CAP_PROP_ANDROID_FOCUS_DISTANCE_FAR:int=8008 {final}
+ CV_CAP_PROP_IOS_DEVICE_FOCUS:int=9001 {final}
+ CV_CAP_PROP_IOS_DEVICE_EXPOSURE:int=9002 {final}
+ CV_CAP_PROP_IOS_DEVICE_FLASH:int=9003 {final}
+ CV_CAP_PROP_IOS_DEVICE_WHITEBALANCE:int=9004 {final}
+ CV_CAP_PROP_IOS_DEVICE_TORCH:int=9005 {final}
+ CV_CAP_ANDROID_COLOR_FRAME_BGR:int=0 {final}
+ CV_CAP_ANDROID_COLOR_FRAME:int=CV_CAP_ANDROID_COLOR_FRAME_BGR {final}
+ CV_CAP_ANDROID_GREY_FRAME:int=1 {final}
+ CV_CAP_ANDROID_COLOR_FRAME_RGB:int=2 {final}
+ CV_CAP_ANDROID_COLOR_FRAME_BGRA:int=3 {final}
+ CV_CAP_ANDROID_COLOR_FRAME_RGBA:int=4 {final}
+ CV_CAP_ANDROID_FLASH_MODE_AUTO:int=0 {final}
+ CV_CAP_ANDROID_FLASH_MODE_OFF:int=0+1 {final}
+ CV_CAP_ANDROID_FLASH_MODE_ON:int=0+2 {final}
+ CV_CAP_ANDROID_FLASH_MODE_RED_EYE:int=0+3 {final}

Highgui

+ CV_CAP_ANDROID_FLASH_MODE_TORCH:int=0+4 {final}
+ CV_CAP_ANDROID_FOCUS_MODE_AUTO:int=0 {final}
+ CV_CAP_ANDROID_FOCUS_MODE_CONTINUOUS_VIDEO:int=0+1 {final}
+ CV_CAP_ANDROID_FOCUS_MODE_EDOF:int=0+2 {final}
+ CV_CAP_ANDROID_FOCUS_MODE_FIXED:int=0+3 {final}
+ CV_CAP_ANDROID_FOCUS_MODE_INFINITY:int=0+4 {final}
+ CV_CAP_ANDROID_FOCUS_MODE_MACRO:int=0+5 {final}
+ CV_CAP_ANDROID_WHITE_BALANCE_AUTO:int=0 {final}
+ CV_CAP_ANDROID_WHITE_BALANCE_CLOUDY_DAYLIGHT:int=0+1 {final}
+ CV_CAP_ANDROID_WHITE_BALANCE_DAYLIGHT:int=0+2 {final}
+ CV_CAP_ANDROID_WHITE_BALANCE_FLUORESCENT:int=0+3 {final}
+ CV_CAP_ANDROID_WHITE_BALANCE_INCANDESCENT:int=0+4 {final}
+ CV_CAP_ANDROID_WHITE_BALANCE_SHADE:int=0+5 {final}
+ CV_CAP_ANDROID_WHITE_BALANCE_TWILIGHT:int=0+6 {final}
+ CV_CAP_ANDROID_WHITE_BALANCE_WARM_FLUORESCENT:int=0+7 {final}
+ CV_CAP_ANDROID_ANTIBANDING_50HZ:int=0 {final}
+ CV_CAP_ANDROID_ANTIBANDING_60HZ:int=0+1 {final}
+ CV_CAP_ANDROID_ANTIBANDING_AUTO:int=0+2 {final}
+ CV_CAP_ANDROID_ANTIBANDING_OFF:int=0+3 {final}
+ IMREAD_UNCHANGED:int=-1 {final}
+ IMREAD_GRAYSCALE:int=0 {final}
+ IMREAD_COLOR:int=1 {final}
+ IMREAD_ANYDEPTH:int=2 {final}
+ IMREAD_ANYCOLOR:int=4 {final}
+ IMWRITE_JPEG_QUALITY:int=1 {final}
+ IMWRITE_PNG_COMPRESSION:int=16 {final}
+ IMWRITE_PNG_STRATEGY:int=17 {final}
+ IMWRITE_PNG_STRATEGY_DEFAULT:int=0 {final}
+ IMWRITE_PNG_STRATEGY_FILTERED:int=1 {final}
+ IMWRITE_PNG_STRATEGY_HUFFMAN_ONLY:int=2 {final}
+ IMWRITE_PNG_STRATEGY_RLE:int=3 {final}
+ IMWRITE_PNG_STRATEGY_FIXED:int=4 {final}
+ IMWRITE_PXM_BINARY:int=32 {final}

+ imdecode(buf:Mat, flags:int):Mat
+ imencode(ext:String, img:Mat, buf:MatOfByte, params:MatOfInt):boolean
+ imencode(ext:String, img:Mat, buf:MatOfByte):boolean
+ imread(filename:String, flags:int):Mat
+ imread(filename:String):Mat
+ imwrite(filename:String, img:Mat, params:MatOfInt):boolean
+ imwrite(filename:String, img:Mat):boolean

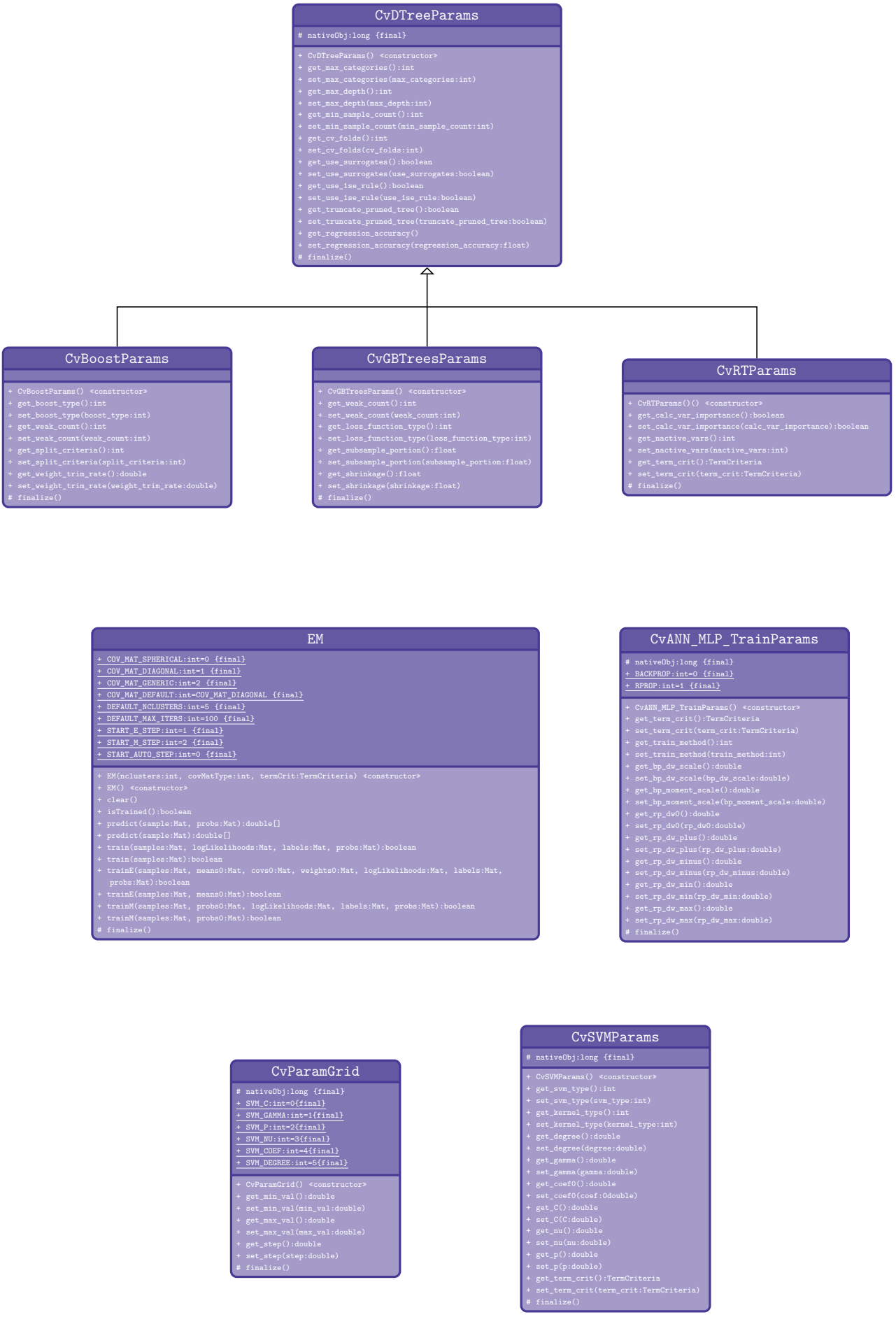
VideoCapture

nativeObj:long {final}

+ VideoCapture() «constructor»
+ VideoCapture(int device) «constructor»
+ get(propId:int):double
+ getSupportedPreviewSizes():List<Size>
+ grab()
+ isOpened()
+ open(device:int)
+ read(image:Mat)
+ release()
+ retrieve(image:Mat, channel:int):boolean
+ retrieve(image:Mat):boolean
+ set(propId:int, value:double):boolean
# void finalize()

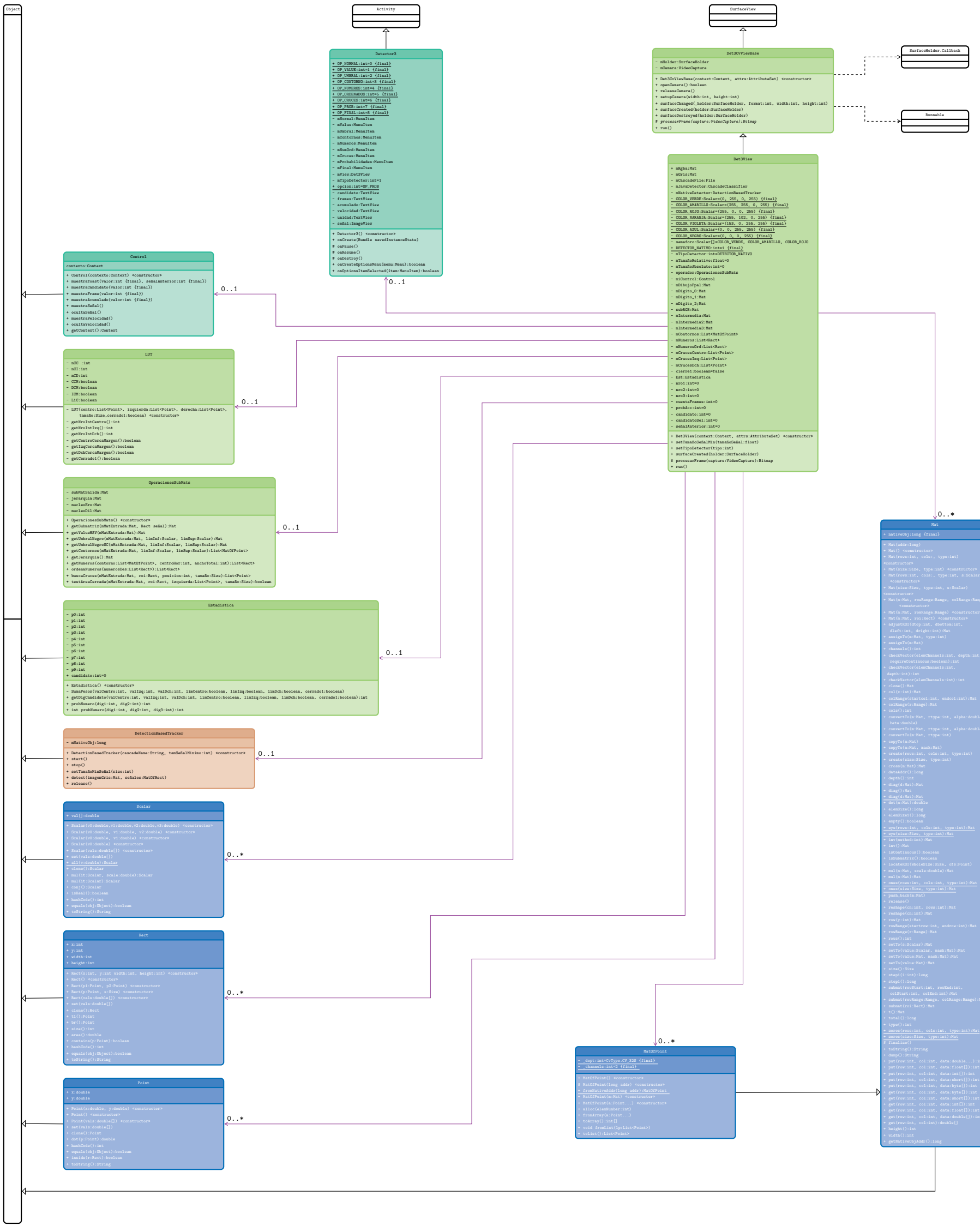
ANEXO D

Módulo `ml`



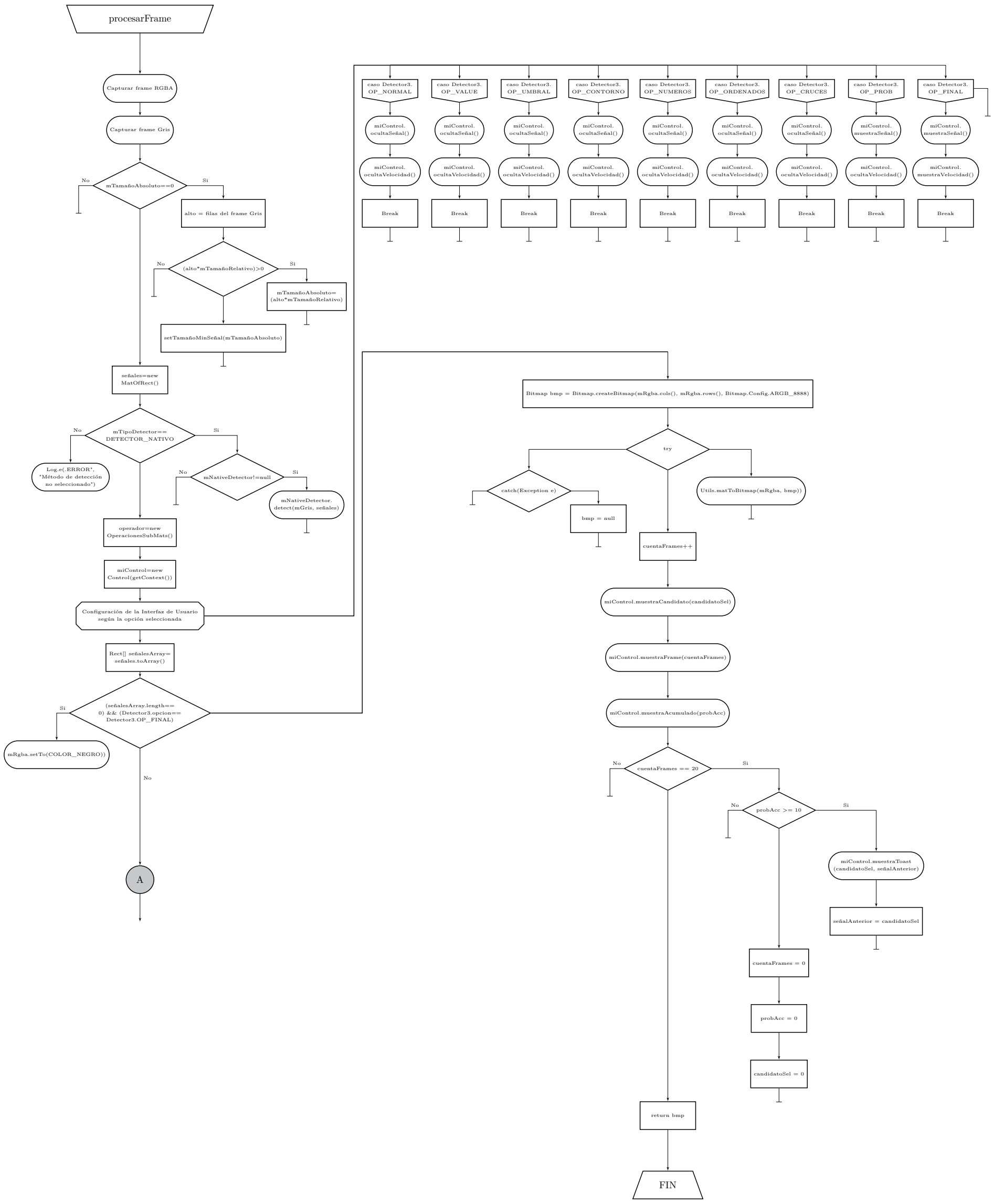
Anexo E

Diagrama de clases de la aplicación



Anexo F

Diagrama de flujo del método `procesarFrame`



B

Caso Detector3.OP_CRUCES

subRGB = operador.getSubmatriz(mRgba, subalboArray[j])

subRGB != null

No

Si

subRGB.copyTo(mIntermedia)

subRGB.copyTo(mIntermedia2)

subRGB.copyTo(mIntermedia3)

mContorno = operador.getContorno(mIntermedia, new Scalar(60))

int centroSubMat = (int) mIntermedia.row(j/2

int anchoSubMat = mIntermedia.cols()

Break

mNumeros = operador.getNumeros(mContorno, centroSubMat, anchoSubMat)

mIntermedia2 = operador.getUnitariNegroS(mIntermedia2, new Scalar(60))

mNumerosOrd = operador.ordenaNumeros(mNumeros)

mNumerosOrd != null

No

Si

for (int ind = 0; ind<mNumerosOrd.size(); ind++)

int inicio = 0

Size tamaño = new Size(30, 40)

int ctrDigito = 14

int izqDigito = 5

int dchDigito = 24

Según ind

Caso 0

mIntermedia = operador.getSubmatriz(mIntermedia3, mNumerosOrd.get(ind))

Imgproc.resize(mIntermedia, mIntermedia, tamaño)

Se extraen los 3 vectores de cruce verticales

Se dibujan las 3 líneas verticales

Se dibujan las intersecciones

mDigito_0 = mRgba.submat(mIntermedia.height()*3, mIntermedia.height()*4, inicio, inicio+(mIntermedia.width()*3))

Imgproc.resize(mIntermedia, mDigito_0, mDigito_0.size())

Break

Caso 1

mIntermedia = operador.getSubmatriz(mIntermedia3, mNumerosOrd.get(ind))

Imgproc.resize(mIntermedia, mIntermedia, tamaño)

Se extraen los 3 vectores de cruce verticales

Se dibujan las 3 líneas verticales

Se dibujan las intersecciones

inicio = mDigito_0.width() + 10

mDigito_1 = mRgba.submat(mRgba.height(), (mIntermedia.height()*3), mRgba.height(), inicio, inicio+(mIntermedia.width()*3))

Imgproc.resize(mIntermedia, mDigito_1, mDigito_1.size())

Break

Caso 2

mIntermedia = operador.getSubmatriz(mIntermedia3, mNumerosOrd.get(ind))

Imgproc.resize(mIntermedia, mIntermedia, tamaño)

Se extraen los 3 vectores de cruce verticales

Se dibujan las 3 líneas verticales

Se dibujan las intersecciones

inicio = mDigito_0.width() + mDigito_1.width() + 10

mDigito_2 = mRgba.submat(mRgba.height(), (mIntermedia.height()*3), mRgba.height(), inicio, inicio+(mIntermedia.width()*3))

Imgproc.resize(mIntermedia, mDigito_2, mDigito_2.size())

Break

Caso Detector3.OP_PROB

subRGB = operador.getSubmatriz(mRgba, subalboArray[j])

cierre1 = false

Est = new Estadistica()

nro1 = 0

nro2 = 0

nro3 = 0

subRGB != null

No

Si

Break

Caso Detector3.OP_FINAL

subRGB = operador.getSubmatriz(mRgba, subalboArray[j])

cierre1 = false

Est = new Estadistica()

nro1 = 0

nro2 = 0

nro3 = 0

subRGB != null

No

Si

mRgba.setTo(COLOR_NEGRO)

Break

subRGB.copyTo(mIntermedia)

int anchoSubMat = mIntermedia.cols()

mNumeros = operador.getNumeros(mContorno, centroSubMat, anchoSubMat)

mNumerosOrd = operador.ordenaNumeros(mNumeros)

mNumerosOrd != null

No

Si

for (int ind = 0; ind<mNumerosOrd.size(); ind++)

digitos = 0

digitos == 2

No

Si

candidatoSel = candidato

probAcc++

candidatoSel == candidato

No

Si

candidatoSel = candidato

probAcc++

candidatoSel == 0

No

Si

candidatoSel = candidato

probAcc++

Est.probNumero(nro1, nro2)

Según ind

Caso 0

Se extraen los 3 vectores de cruce verticales

Detección de cierre

Crear LUT prop1

nro1 = Est.getDigCandidato (prop1.getNvolutasCentro(), prop1.getNvolantes(), prop1.getNvolantDch(), prop1.getIzqCercaMargen(), prop1.getDchCercaMargen(), prop1.getCerrado1())

Caso 1

Se extraen los 3 vectores de cruce verticales

Detección de cierre

Crear LUT prop2

nro1 = Est.getDigCandidato (prop2.getNvolutasCentro(), prop2.getNvolantes(), prop2.getNvolantDch(), prop2.getIzqCercaMargen(), prop2.getDchCercaMargen(), prop2.getCerrado1())

Caso 2

Se extraen los 3 vectores de cruce verticales

Detección de cierre

Crear LUT prop3

nro1 = Est.getDigCandidato (prop3.getNvolutasCentro(), prop3.getNvolantes(), prop3.getNvolantDch(), prop3.getIzqCercaMargen(), prop3.getDchCercaMargen(), prop3.getCerrado1())

digitos++

Break