Universidad Carlos III de Madrid
www.uc3m.es

Grado en Ingeniería Electrónica Industrial y Automática

2015-2016

Trabajo Fin de Grado

"Aplicación Android para Seguimiento de Peatones en Entornos Viarios"

Jéssica Cecibel Arrobo Sarango

Tutor:

Fernando García Fernández

Leganés 26 de Septiembre de 2016

Agradecimientos

Quiero agradecer a mis profesores que han ayudado a formarme para poder realizar este proyecto, a mi tutor que ha tenido mucha paciencia conmigo y a mi familia, especialmente a mi marido e hijo que me han animado para seguir adelante.

Resumen

El objetivo principal de este proyecto es desarrollar una aplicación para la detección de peatones en la vía pública para su posterior seguimiento. Con esto se quiere evitar los numerosos accidentes con peatones implicados, que se producen en la actualidad tanto en vías urbanos como interurbanas.

La aplicación se ha desarrollado para funcionar en plataformas Android, debido a que cumple una serie de características: es una plataforma de código abierto y es una de las plataformas móviles más extendidas en el mercado actual, que permiten desarrollar aplicaciones de forma sencilla y que éstas tengan un gran número de usuarios.

Para la detección de peatones se va usar un algoritmo basado en un extractor de características Haar-like, que al aplicarse en cascada permite una mejor detección. Para el seguimiento se hará uso de un filtro de Kalman, asumiendo que el peatón tiene un movimiento rectilíneo uniforme. En ambos algoritmos se ha usado, de apoyo, las librerías OpenCV sobre todo para la obtención y procesamiento de la imágenes tomadas con la cámara del dispositivo móvil.

Abstract

The aim of this project is to develop an application which detects pedestrians on public roads and subsequently track them. The purpose is to avoid the numerous traffic accidents, where pedestrians are involved, occurred nowadays in both urban and interurban roads.

The application has been developed to run on Android platforms, because it have some characteristics that allow to develop applications easily. Android is one of the most widely used mobile platforms on market today, so there are a large number of potential users.

Pedestrians detection will be based on a Haar-like feature extractor, when it is applied on cascade offers a better detection algorithm. To pedestrians tracking, a Kalman filter has been used assuming that the pedestrians has a uniform rectilinear motion. In both algorithms OpenCV libraries has been used for image obtaining and image processing.

Índice general

Índice de figuras

Índice de tablas

Índice de acrónimos

API	Application Programming Interface (Interfaz de Programación de Aplicaciones)
OpenCV	Open source Computer Vision Library (Librería de código abierto de visión artificial)
ADAS	Advanced Driver Assistance System (Sistemas Avanzados de Asistencia en la Conducción)
HOG	Histogram of Oriented Gradients (Histogramas de Gradientes Orientados)
SVM	Support Vector Machine (Máquinas de Vectores Soporte)
AdaBoost	Adaptive Boosting (Aprendizaje Adaptativo)
MIT	Massachusetts Institute of Technology (Instituto de Tecnología de Massachusetts)
Wi-Fi	Wireless Fidelity (Fidelidad inalámbrica)
V2V	Vehicle To Vehicle (de Vehículo a Vehículo)
V2X	Vehicle To Infrastructure (de Vehículo a Infraestructura)
HMI	Human-Machine Interface (Inferfaz Humano-Maquina)
LiDAR	Light Detection And Ranging (Detección y cálculo de Distancias con Laser)

1 Introducción

En la actualidad las nuevas tecnologías están permitiendo mejoras en la sociedad como puede ser la realización de aplicaciones de ayuda en la conducción, ahorro energético, de control de la salud, etc.

La visión por computador ha dado un nuevo enfoque al desarrollo de este tipo de aplicaciones ya muchas de ellas no serían posibles sin el uso de los métodos desarrollados con la visión por computador. Además, el uso cada vez más extendido de los Smartphones permiten que las aplicaciones desarrolladas, para plataformas móviles, sean accesibles para la mayor parte de la población.

1.1 Motivación

En las últimas décadas el uso de vehículos a motor a aumentado, por lo que también ha aumentado la circulación de dichos vehículos. Los vehículos a motor son capaces de moverse a grandes velocidades y por ello cuando se produce un accidente el índice de mortalidad es muy alto, por este motivo la seguridad en la conducción es uno de los principales temas de preocupación en la actualidad. Dentro de la seguridad durante la conducción, uno de los principales problemas es el atropello de peatones, ya que son uno de los colectivos más vulnerables en la carretera. Los accidentes con peatones implicados tienen un gran índice de mortalidad, por lo que un sistemas de detección de peatones, en el vehículo, ayudaría a disminuir el número de este tipo de accidente.

En la actualidad existen algunas aplicaciones de detección de personas, pero éstas están desarrolladas para otros sistema operativos que en su mayoría requiere el uso de un ordenador, y tienen un coste computacional elevado, que debido a sus dimensiones es difícil de situar en el interior de un automóvil. Por lo que es interesante realizar una aplicación este tipo para un dispositivo más portable, como pueden ser los Smartphones, tablets u otros dispositivos ligeros. El uso de un Smartphone o tablet permite que no sea necesario la modificación de la estructura del vehículo, con la instalación de una cámara, un procesador y una pantalla, elementos de los que ya disponen de estos dispositivos; y de este modo se puede usar en una gran variedad de vehículos.

1.2 Objetivos

El principal objetivo de este proyecto es desarrollar una aplicación que sea capaz de detectar y seguir a peatones, en un entorno viario. Con esta aplicación se busca evitar los numerosos accidentes con peatones implicados que se producen cada año en las carreteras. Para cumplir con este objetivo, se divide el proyecto en dos partes en primer lugar la detección y posteriormente el seguimiento del peatón, de forma que se pueda determinar la trayectoria del peatón.

Si se quiere evitar la mayor parte de estos accidentes, se debe llegar al mayor número de usuarios, que puedan conducir un vehículo. Por lo tanto, la aplicación se debe poder usar en una gran variedad de dispositivos Smartphones o tablets. En la actualidad uno de los sistemas operativos más utilizados en Smartphones o tablets, es el sistema operativo Android. Por lo que otro de los objetivos de este proyecto es desarrollar la aplicación en Android. Pero no todos los usuarios tienen acceso a dispositivo de última generación por lo que la aplicación debe poder funcionar en diferentes versiones de Android, incluso en las más antiguas.

Como se ha mencionado en el apartado anterior, se quiere llegar a un gran número de usuarios y no todos ellos tienen grandes conocimientos de las nuevas tecnologías, por lo que otro de los objetivos debe ser que la aplicación sea intuitiva y fácil de usar.

Al usar Smartphones o tablets, existe una limitación en la capacidad de procesamiento que tienen estos dispositivos, motivo por el cual se requiere que la aplicación desarrollada esté depurada, de modo que funcione de una manera fluida y pueda ser usada en tiempo real.

2 Estado del arte

2.1 Sistema operativo Android

Android es una plataforma de código abierto (bajo licencia Apache/MIT) diseñada inicialmente para móviles aunque en la actualidad se emplea en otros dispositivos como tablets, portátiles 2 en 1, relojes, Smart TV o automóviles (Android Auto). Al ser de código abierto es más accesible para los desarrolladores de modo que se crean un mayor número de aplicaciones con respecto a otras plataformas móviles (iOS, Symbian, WindowsPhone, Blackberry).

En sus inicios fue desarrollada por Android Inc. y posteriormente fue comprada por Google. Google junto con Open Handset Alliance (consorcio de compañías de hardware, software y telecomunicaciones) crearon una alianza cuya principal motivación era "acelerar la innovación en los dispositivos móviles y ofrecer a una gran cantidad de consumidores, a precio reducido, una mejor experiencia móvil" (Gargenta, 2011).

Como consecuencia Android ha revolucionado el mercado de móviles duplicando el número de ventas con respecto al segundo más vendido, como demuestra un estudio realizado por Kantar Worldpanel sobre las ventas de Smartphone. Durante el periodo Febrero-Abril 2016 las ventas de dispositivos Android en España alcanzaron casi un 94%, mientras que las ventas de dispositivos iOS no llegaba al 6% y un 1% los dispositivos WindowsPhone. También ha aumentado su cuota de mercado en los principales mercados mundiales (EE.UU., China, Australia, Japón y los 5 principales países europeos) con respecto a las otras plataformas móviles, cuya cuota ha disminuido, como se puede ver en la Tabla 1.

	Abril 2015	Abril 2016	% de Variación
USA			
Android	62.4	67.6	5.2
iOS	33.2	30.7	-2.5
Windows	3.8	1.3	-2.5
Other	0.7	0.4	-0.3
China			
Android	74.0	78.8	4.8
iOS	24.5	20.1	-4.4
Windows	1.0	0.7	-0.3
Other	0.6	0.4	-0.2
Australia			
Android	54.4	64	9.6
iOS	36.3	32.7	-3.6
Windows	7.6	2.8	-4.8
Other	1.7	1.5	-0.2
Japan			
Android	51.6	55.7	4.1
iOS	45.6	43.2	-2.4
Windows	0.2	0.1	-0.1
Other	2.5	1	-1.5
EU5			
Android	70.2	76.0	5.8
iOS	19.3	18.3	-1.0
Windows	9.3	4.8	-4.5
Other	1.2	0.8	-0.3

Tabla 1. Cuotas de ventas de Sistemas Operativos de Smartphones (%) (Guenveur, 2016)

2.1.1 Arquitectura Android

Android posee una arquitectura de 4 capas (Figura 1), cada capa posee características y propósitos propios , relacionadas unas con otras

Como indica la Figura 1, la arquitectura Android está construida sobre un núcleo de Linux, modificado para adaptarse a dispositivos móviles. Al estar basado en Linux, Android posee las ventajas de este sistema operativo, tales como la portabilidad, la seguridad y la flexibilidad.

En el segundo nivel de las capas de Android se encuentran las librerías. En esta capa se encuentran la capa de abstracción de hardware, permitiendo que una misma aplicación funcione en un gran número de dispositivos sin importar su arquitectura física; las librerías nativas, como librerías OpenGL, SQLite, las librerías de servicios de Wi-Fi o telefonía; la consola de procesos y la máquina virtual Dalvik, que es la encargada de interpretar los archivos generados durante la compilación de los programas.

En el tercer nivel se encuentra el marco de trabajo de las aplicaciones. Esta capa es interesante para los desarrolladores de aplicaciones ya que en esta capa se encuentran las librerías Java, los paquetes android.* (en los que se encuentran las características necesarias para construir una aplicación Android). En esta capa también se encuentran manejadores, servicios y proveedores de contenido, que se usan para comunicar las aplicaciones con el ecosistema de Android.

Por último, la cuarta capa, se centra en la ejecución, comunicación y estabilidad de las aplicaciones ya sea preinstaladas por el fabricante o las que se vayan a instalar. A esta capa es la que acceden todos los usuarios, ya tiene un alto nivel de comprensión y simplicidad.

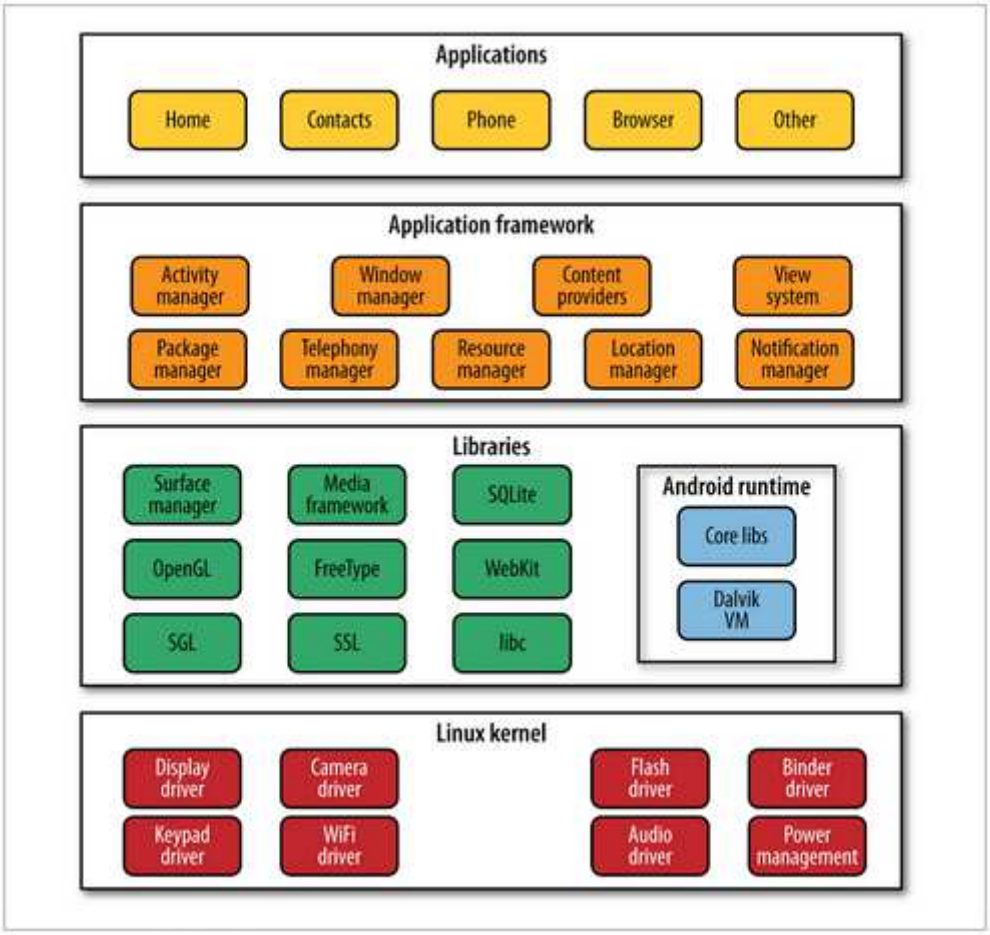

Figura 1. Arquitectura Android

2.1.2 Versiones Android

Como muchos otros software, Android se está actualizando constantemente, y esto se refleja en sus versiones. Las versiones lanzadas por Android llevan el nombre de un postre en inglés siguiendo un orden alfabético, aunque el seguimiento de ellas es algo confuso ya que no tienen un número consecutivo.

La primera versión de Android (1.0) fue lanzada, en versión de pruebas, en Septiembre de 2008 y llevaba el nombre de Apple Pie (Tarta de manzana). Desde entonces este sistema operativo ha tenido varias actualizaciones. Generalmente estas aplicaciones añaden mejoras al sistema o corrigen los fallos que haya tenido.

Cada versión de Android viene acompañada por un nivel API, en el desarrollo de aplicaciones es muy importante el nivel API utilizado ya que de ello depende en que dispositivos puede funcionar la aplicación desarrollada.

Para llegar a la mayor parte de los usuarios se debe usar el nivel API más bajo posible, que permita desarrollar adecuadamente la aplicación y que cubra la mayor cuota de mercado. En la Figura 2. podemos ver que versiones de Android hay actualmente (Junio, 2016) en el mercado y en la Tabla 2. el nivel API que le corresponde.

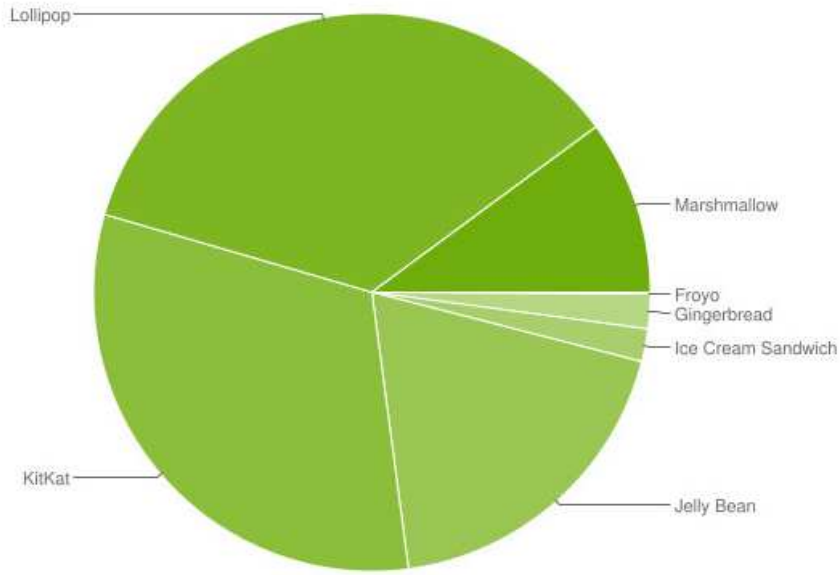

Figura 2. Distribución de las versiones Android (COSMOS, 2016)

Versión	Nombre	API	Distribución (%)
2.2	Froyo	8	0.1
2.3.3 - 2.3.7	Gingerbread	10	2.0
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.9
4.1.x	Jelly Bean	16	6.8
4.2.x		17	9.4
4.3		18	2.7
4.4	KitKat	19	31.6
5.0	Lollipop	21	15.4
5.1		22	20.0
6.0	Marshmallow	23	10.1

Tabla 2. Distribución de las versiones Android (COSMOS, 2016)

2.2 Sistemas ADAS

Los sistemas ADAS son sistemas de ayuda para una conducción mejor y más segura. El objetivo de los sistemas ADAS es ofrecer tecnologías que avisen al conductor de problemas potenciales, eviten colisiones, encendido automático de luces, control de velocidad adaptativo, frenado automático, aviso de salida de carril, reconocimiento de señales de tráfico entre otros; mediante una interfaz humano-maquina (HMI). Un ejemplo de estas aplicaciones se pueden ver en la Figura 3.

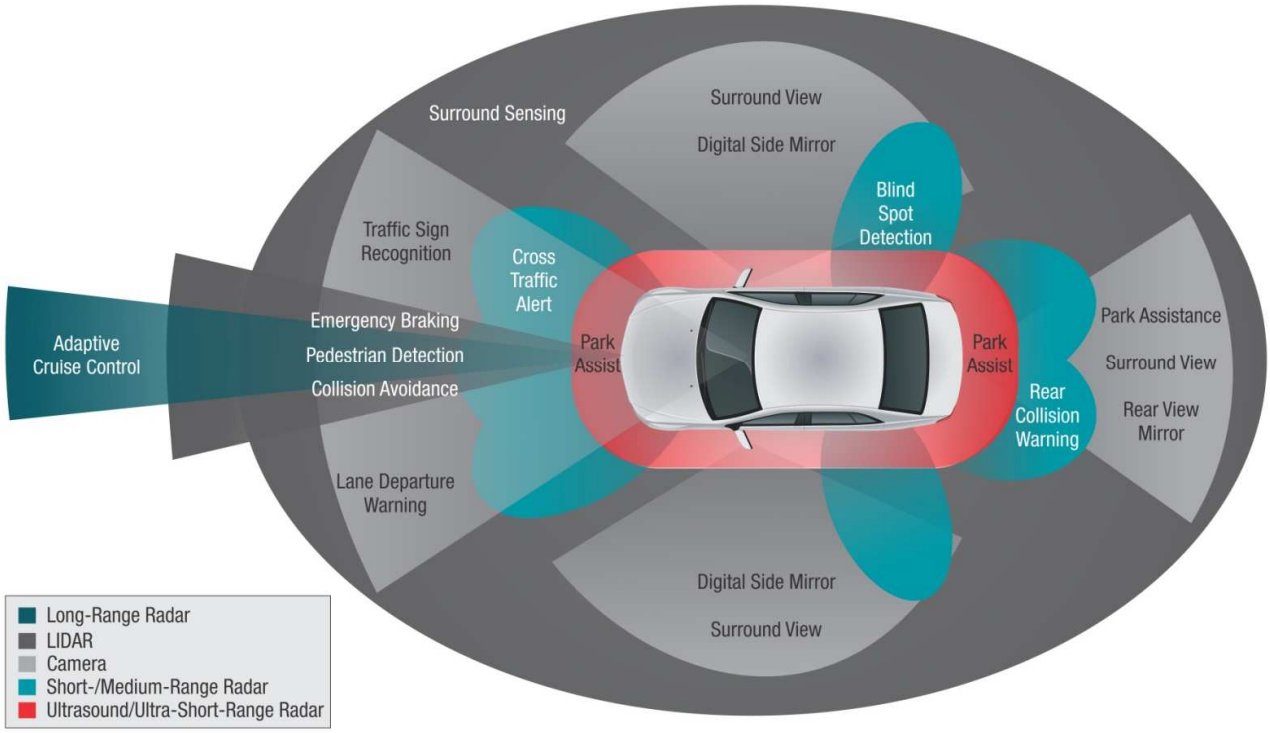

Figura 3. Aplicaciones ADAS

Hay varios tipos de sistemas ADAS disponibles, algunos se pueden incorporar en el interior del automóvil o como paquetes complementarios que se pueden añadir al vehículo. Estos sistemas no solo están disponibles para vehículos de nueva fabricación sino también para que ya están en uso o ya han salido de fabrica.

La forma en que los sistemas ADAS reciben los datos de entrada, para poder desempeñar su papel, puede provenir de distintos sensores como radares, LiDAR o cámaras. Y estos datos se pueden tratar con diferentes algoritmos entre los que se encuentran los de procesamiento de imágenes o visión artificial.

Además de tener una interfaz humano-máquina también se quiere conectar el vehículo con su entorno, por lo que se la siguiente generación de sistemas ADAS está trabajando en la conexión con otros vehículos (V2V) o con otra infraestructura (V2X), ya sea mediante telefonía móvil o WiFi.

Los sistemas ADAS han experimentado un crecimiento muy rápido en el sector dedicado a la electrónica del automóvil, y a la vez que se han desarrollado aplicaciones ADAS también se ha ido aprobando especificaciones que deben cumplir estos sistemas, ya que de muchos de ellos depende la seguridad de personas, como puede ser la ISO 26262.

2.3 Visión Artificial

La visión artificial o visión por computador es un campo científico que busca dotar a los ordenadores de una visión lo más parecida posible a la humana, mediante el uso de algoritmos matemáticos.

El objetivo principal de la Visión Artificial es extraer información de mundo que rodea al ordenador a partir de una imagen obtenida con una cámara. Generalmente para desempeñar esta tarea se siguen los siguientes pasos: adquisición, procesamiento y análisis de la imagen. El reto más importante es extraer información del mundo real y poder reproducirla con números o símbolos que pueda entender una máquina. Para ello se han construido modelos para los objetos del mundo real, ya sean basados en su geometría, comportamiento físico, etc.; también se ha analizado su comportamiento estadístico y desarrollado teorías de aprendizaje para las máquinas.

La forma de resolver un problema varía en función del método que se quiera usar por lo que para cada una de estas etapas, la visión artificial dispone de diversos herramientas o algoritmos, como puede ser la calibración de la cámara, extracción de bordes, detección de objetos, reconstrucción de escenas 3D, seguimiento de objetos, etc.

La visión artificial empezó a desarrollarse en la década de los 60's y el objetivo era imitar la visión humana y dotar a los robots de ella, como punto de partida de la inteligencia artificial.

Lo que distingue la visión artificial, del procesamiento de imagen digital, es que con la visión artificial se busca extraer la estructura tridimensional, a partir de imágenes, con el objetivo de lograr una plena comprensión de la escena.

Los estudios realizados en la década de los 70's fueron bases de muchos de los algoritmos de visión por computador que existen hoy en día, como por ejemplo la extracción de bordes, etiquetado de líneas, flujo óptico o de estimación del movimiento.

En las décadas siguientes se realizó un estudio más riguroso de las matemáticas implicadas en la visión por computador. Algunos de los campos de la visón artificial fueron más estudiados que otros, como por ejemplo, métodos de optimización para la calibración de la cámara, la reconstrucción de escenas en tres dimensiones a partir de imágenes múltiples.

La visión artificial se puede aplicar a numerosas tareas desde sistemas de visión en la industria hasta aplicarla en el desarrollo de la inteligencia artificial de ordenadores y robots, de modo que puedan comprender el mundo que les rodea. A menudo la visión por computador se combina con otras tecnologías para poder resolver tareas complejas. Uno de los campos más destacados, de la visión artificial, es en el campo médico; lo que se busca en esta área es realizar un diagnostico médico a partir de la información que les proporciona una imagen.

Además de la industria, otro de los campos donde la visión tiene un gran rango de uso, es en la industria militar, algunas de las aplicaciones pueden ser la detección de soldados enemigos o la guía de misiles,

En la actualidad el uso de la visión artificial se ha extendido y existe un gran interés por desarrollar algoritmos de visión. Por lo que hay diferentes bibliotecas de visión artificial como pueden ser Torch3Vision, SimpleCV, OpenCV, PCL (C++) o BoofCV(Java) que tienen diferentes algoritmos de visión artificial y son de código abierto.

2.3.1 OpenCV

OpenCV es una librería de código abierto, fue desarrollada inicialmente por Intel. Su primera versión alfa fue lanzada en 1999 y desde entonces se ha usado en una gran cantidad de aplicaciones; como por ejemplo en sistemas de de seguridad basados en la detección de movimiento, control de procesos o sistemas de vigilancia de vídeo. En la actualidad cuenta con más de 2.500 algoritmos optimizados, que pueden ser usados para detección y reconocimiento facial, identificación de objetos, extraer modelos 3D de objetos.

OpenCV invita a sus usuarios a desarrollar o mejorar algoritmos de visión artificial, actualmente cuenta con una comunidad de 47 mil miembros. Al ser de código abierto es usado por universidades, órganos gubernamentales particulares o empresas como Google, Sony o Yahoo, que desean desarrollar aplicaciones de visón por lo que las descargas, de librerías OpenCV, superan los 7 millones. OpenCV es multiplataforma, es decir, se pueden usar en diferentes sistemas operativo como pueden ser: Windows, Linux, Android y Mac.

OpenCV está escrito en lenguaje C++ y su principal interfaz es para C++, pero aun conserva la interfaz en C. Además posee adaptaciones para Python, Java y MATLAB y se están desarrollando adaptaciones para otros lenguajes como pueden cer C#, Perl o Haskell.

Figura 4. Logotipo de OpenCV4Android

OpenCV dispone de una adaptación de su librería para Android, llamada OpenCV4Android (Figura 4.). La primera versión que soportaba Android fue la 2.4.3 y en la actualidad se está trabajando en la versión 3.0. No dispone de las mismas funcionalidades que OpenCV, ya que no todas las clases y funciones están adaptadas a Java y por tanto a Android y algunas de las que están adaptadas no disponen de todas sus funcionalidades. Además para poder utilizar una aplicación que use librerías de OpenCV, en un dispositivo Android, es necesario tener instalado en dicho dispositivo la aplicación *OpenCV Manager*.

2.4 Detección de personas

La detección de personas es un problema clave en la visión por computador, las soluciones propuestas son complejas ya que los humanos son difíciles de caracterizar debido a la variación en la apariencia debido a las ropas, formas de caminar, las condiciones de iluminación comunes en las escenas exteriores. Por ello a pesar de los logros conseguidos en esta tare, la detección de personas ha estado en continúa investigación en los últimos años. Un ejemplo de detección de personas se puede ver en la Figura 5.

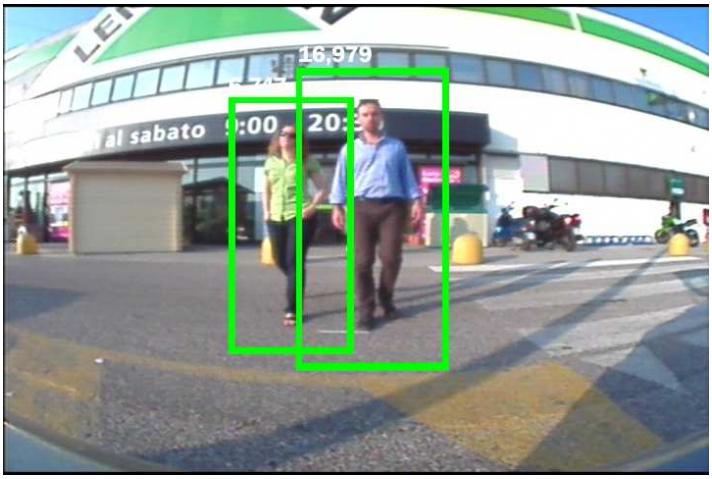

Figura 5. Detección de personas

Se han desarrollado numeroso algoritmos de detección, algunos con mejor resultados que otros y se pueden dividir en 5 tipos:

El enfoque holísticos tiene detectores entrenados para buscar personas en los fotogramas de un video. Algunos de los métodos usan características globales y otros características locales. Una de las propuestas, con un excelente resultado de detección, es la que hicieron (Triggs & Dalal, 2005) cuyo método un descriptor de HOG entrenado con una SVM lineal. Otro método para extraer las características de las personas de forma heurística es usando un clasificador Haar.

En el enfoque basado en "partes", la persona es modelada como un conjunto de sus partes. Para ello primero se generan las características locales de cada parte de la persona, con sus características. La tarea más compleja, en estos algoritmos, es encontrar la forma óptima de unir estas partes. La implementación de este tipo de enfoque sigue un procedimiento estandarizado: primero crea una pirámide de imágenes de muestra, después se extrae las

características de la persona en la escala de cada parte, una vez se tienen las características se realiza la clasificación de las posibles localizaciones de cada parte del modelo y el paso final es generar el conjunto final de personas detectadas.

El enfoque basado en regiones propone una combinación de la detección y la segmentación, conocido con el nombre de Modelo de Forma Implícita (ISM). La apariencia local de la persona se aprende durante el proceso de entrenamiento. En la fase de detección, se busca si las características locales del objeto coinciden con las aprendidas durante el entrenamiento y por cada coincidencia da un voto a la hipótesis de persona. La ventaja de este sistema es que el entrenamiento se puede hacer con un pequeño número de muestras.

El enfoque basado en el movimiento se puede usar solo cuando se cumple ciertas condiciones ambientales, la cámaras debe tener una posición fija, las condiciones de luminosidad debe ser poco variable, etc. Cuando se cumples estas condiciones puede usarse método de extracción de fondo (background subtraction) para detectar personas. Para ello se usa un modelo basado en el movimiento, diferenciando entre el fondo (sin movimiento detectado) y el primer plano (movimiento detectado). El fondo se tiene guardado, y cuando hay una diferencia entre la imagen actual y el fondo guardado se considera que hay movimiento. La imagen que se analiza es la diferencia entre el fondo y el primer plano.

La detección usando cámaras múltiples, es otro de los enfoques dados para tratar la detección de personas. Es un método que integra varias cámaras calibradas para detectar múltiples personas. Con este método el detector produce un mapa con la probabilidad de ocupación y provee la probabilidad de que una celda, con tamaño típico de 25x25 cm este ocupado por una persona

La detección de personas tiene numerosas aplicaciones en la industria del automóvil sobre todo para mejorar las sistemas de seguridad.

2.4.1 Descriptor de HOG

El descriptor de HOG es un clasificador que usa las características locales del objeto para detectar y clasificar objetos, usado en visión por computador y procesamiento de imágenes.

El descriptor de HOG se basa en que un objeto puede ser caracterizado por la distribución de la intensidad del gradiente (orientación de los bordes del objeto). Un descriptor de este tipo es una combinación de los histogramas, de los gradientes orientados de las celdas (n píxeles), en las que se divide una imagen. El resultado después del cálculo del histograma se puede ver en la Figura 6.

El algoritmo de HOG necesita de dos fases, fase de entrenamiento y fase de detección. En la fase entrenamiento se le da al algoritmo de HOG un conjunto de muestras positivas y negativas para que extraiga las características locales de las personas. Una vez se ha entrenado el clasificador, que contiene las características del objeto que se quiere detectar, se puede usar para detectar el objeto buscado en nuevas imágenes.

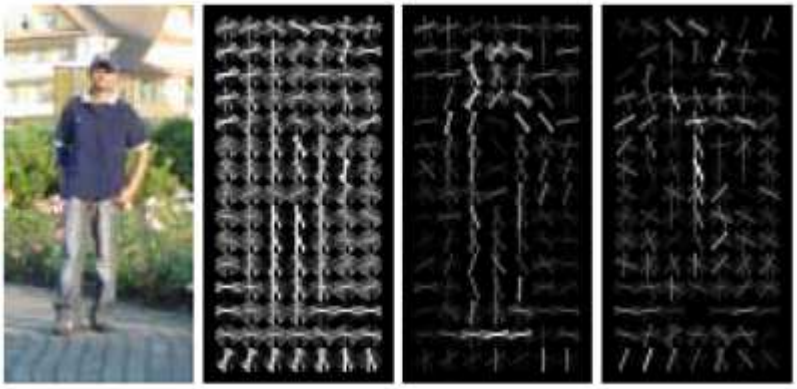

Figura 6. Gradientes orientados de una imagen

La primera vez que se definió un descriptor de HOG fue en la Conferencia en de Visión Artificial y reconocimiento de Patrones del año 2005.

2.4.2 Clasificador de características Haar

Un clasificador de características Haar se basa en aplicar unas características denominadas Haar-like (Figura 7) y se usa para el reconocimiento de objetos. Su nombre proviene de su similitud con el concepto de Haar wavelets, la primera vez que se uso un algoritmo de características Haar-like fue en la detección de caras.

Inicialmente las características Haar-like sólo trabajaban con las intensidades de la imagen, lo que implica un conste computacional muy alto. Las llamadas características Haar-like, propiamente dichas, fueron originalmente desarrollada por (Viola & Jones, 2001) a partir del concepto de los Haar wavelets (secuencia de re-escalado de funciones con forma cuadrada).

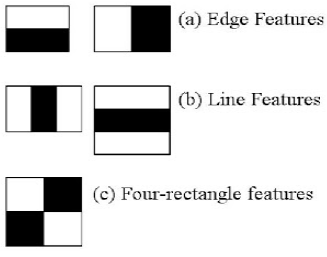

Figura 7. Características Haar-like

Al igual que ocurre con el descriptor de HOG, un clasificador de características Haar necesita ser entrenado por ejemplo con un algoritmo AdaBoost (Freund & Schapire, 1997). Durante el entrenamiento se crean varios clasificadores, denominados débiles, y el conjunto de ellos forma un clasificador fuerte. Los clasificadores débiles están organizados en cascada y se van aplicando, a la imagen que se quiere analizar, de menos restrictivo a más restrictivo.

La principal ventaja de las características Haar-like, sobre otros extractores de características, es su velocidad de cálculo ya que se usan imágenes integrales para obtener la suma de los valores de intensidad. La imagen integral posee una determinadas propiedades que permite reducir el número de operaciones implicadas; en una imagen integral se puede obtener el valor de intensidad que tiene una región concreta conociendo solamente el valor de intensidad de los cuatro píxeles de las esquinas de la región elegida.

2.5 Seguimiento de personas

El seguimiento de personas es otro de los problemas que afronta la visión por computador, una forma simple de definir el seguimiento de personas es considerarlo como la estimación de la trayectoria de la persona en el plano de la imagen. Los campos a los que se puede aplicar el seguimiento de objetos son el reconocimiento basado en el movimiento, vigilancia autónoma, interacción humano-máquina, monitorización del tráfico, navegación de vehículos, etc.

El seguimiento de las objetos en general, y las personas en particular, también tiene sus dificultadas entre ellas se encuentran los cambios abruptos que tiene en su movimiento, cambios del entorno y del objeto mismo (oclusiones), ruido en las imágenes, la iluminación de la escena o los requisitos de procesamiento a tiempo real (Yilmaz, Javed, & Shah, 2006). Es posible hacer el seguimiento de múltiples objetivos pero cuando estos se cruzan entre sí o se juntan, se convierte en un gran reto diferenciar unos de otros.

Hay numerosos métodos que se pueden aplicar en el seguimiento de las personas en función de la representación que se haga de ella como se puede ver en la Figura 8.

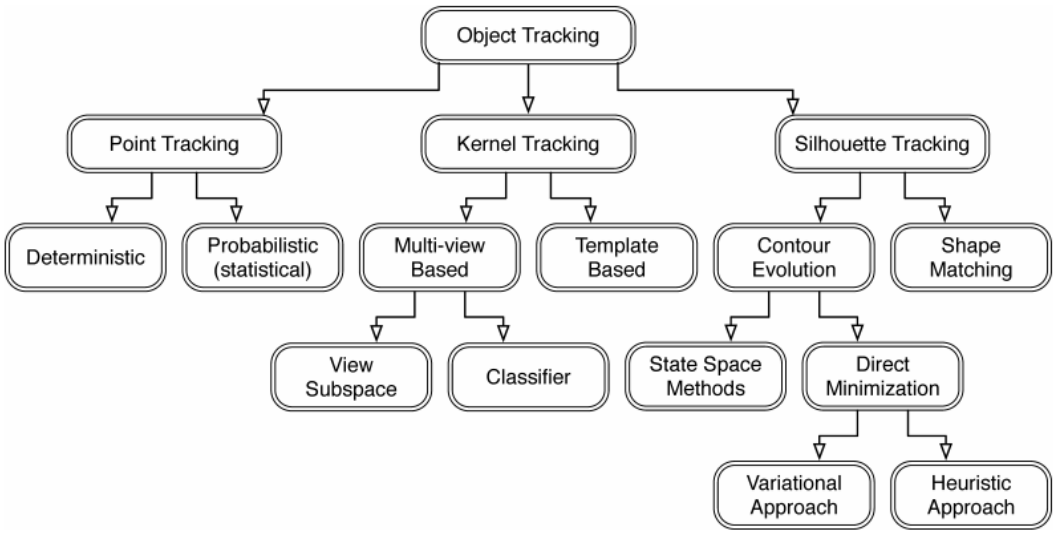

Figura 8. Taxonomía de los métodos de seguimiento

Como se puede ver en la Figura los algoritmos de seguimiento se dividen en tres grupos principales: seguimiento por puntos, seguimiento del núcleo y seguimiento de siluetas.

2.5.1 Seguimiento por puntos

El seguimiento por puntos se usa si la persona se representa mediante puntos o una asociación de puntos, se basa en el estado previo de la persona y requiere de un mecanismo externo para la detección de la persona.

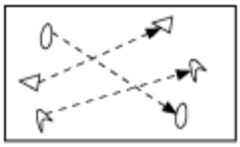

Figura 9. Seguimiento por puntos

Los algoritmos basados en puntos se dividen a su vez en dos grupos: métodos deterministas y métodos probabilísticos .

2.5.1.1 Métodos deterministas

Los métodos deterministas, asignan un coste de correspondencia a un determinado objeto. a cada uno de sus puntos. Un ejemplo de esto método es el algoritmo húngaro. Para asignar estos costes se tienen en cuenta una serie de restricciones.

- Proximidad: se asume que la ubicación del objeto no varía bruscamente de un fotograma a otro
- Velocidad máxima: define la velocidad máxima a la que se mueve el objeto
- Cambios de velocidad pequeños: se asume que los cambios de velocidad del objeto es suave y el movimiento no cambia bruscamente
- Rigidez: se asume que el objeto es rígido, es decir, la distancia entre dos puntos del objeto no va a variar.

2.5.1.2 Métodos probabilísticos

Estos métodos realizan el seguimiento teniendo en cuenta las mediciones del sistemas y la incertidumbre del modelo. Usan el espacio de estado para modelar los objetos, e el vector de estados se modelan la posición, la velocidad o aceleración. Las mediciones del sistema consisten normalmente en la posición del objeto, y se obtienen mediante sensores de video y pueden contener ruido. Además el movimiento de los objetos pueden sufrir perturbaciones aleatorias. Algunos de los algoritmos que usan estos métodos son: el filtro de Kalman o el filtro de partículas.

2.5.1.2.1 Filtro de Kalman

El filtro de Kalman es un conjunto de ecuaciones matemáticas que sirve para identificar el estado oculto o no medible de un sistema dinámico lineal a partir de las mediciones actuales y el estado calculado previamente.

El filtro de Kalman debe su nombre a Rudolf E. Kalman que desarrollo esta teoría en 1960, más adelante Richard S. Bucy (Universidad de California Sur) contribuyo a su desarrollo. Pero fue Stanley F. Schmidt quien implementó por primera vez un Filtro de Kalman, para resolver el problema no-lineal de estimación de la trayectoria en el programa Apollo, encaminado a incorporarse en el ordenador de navegación del Apollo.

El filtro de Kalman fue vital en los sistemas de navegación de los submarinos con misiles nucleares de Marina estadounidense y en los sistemas de guía y navegación de misiles de crucero, como el misil Tomahawk de la Marina estadounidense, y para el lanzamiento de misiles de la Fuerza Aérea estadounidense.

Este filtro tiene diferentes extensiones y generalizaciones que han sido desarrolladas por otros autores como por ejemplo el filtro de Straronovich-Kalman-Bucy. Este filtro es un caso especial para problemas no lineales desarrollado por el soviético Ruslan Stratonovich. De hecho Stratonovich publicó alguna de las ecuaciones de su filtro poco después de conocer a Kalman en una conferencia en la ciudad de Moscú.

El filtro de Kalman puede ser usado para diversas aplicaciones tecnológicas, las más comunes son la navegación y control de vehículos principalmente en la industria aeroespacial. Además el filtro de Kalman es un filtro ampliamente aplicado en el procesamiento de señales y econometría. También se puede usar para la optimizando la trayectoria en la planificación y control del movimiento de los robot.

2.5.1.2.2 Filtro de partículas

Una de las desventajas del filtro de Kalman es que asume que las variables de estado siguen una distribución Gaussiana, por lo que el filtro de Kalma ofrece una mala estimación en el caso de que dichas variables de estado no sigan una distribución Gaussiana. El filtro de partículas es un método empleado para estimar el estado de un sistema que cambia con el tiempo.

Fue propuesto en 1993 por N. Gordon, D. Salmond y A. Smith para implementar filtros bayesianos recursivos. Básicamente el filtro de partículas se compone de un conjunto de muestras, llamadas partículas, y unos valores, o pesos, que se pueden representar como puntos en el espacio de estados.

2.5.2 Seguimiento del núcleo (kernel)

El seguimiento por núcleo se usa si la persona se representa por la forma, rectangular o elíptica, o por su apariencia. Para el seguimiento se usa el movimiento del núcleo, que suele ser una transformación parametrizada.

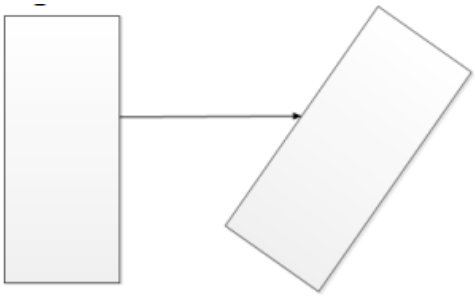

Figura 10. Seguimiento del núcleo

Se pueden distinguir dos categorías dentro de este tipo de seguimiento: basado en plantillas o en la apariencia y en modelos multi-vista.

2.5.2.1 Basado en plantillas

Los métodos basados en plantillas o en los modelos de apariencia se han usado a menudo debido a la simplicidad de su uso y el bajo coste computaciones que supone.

El enfoque más común es la comparación de plantillas, buscando en la nueva imagen una región que coincida con la imagen almacenada en la plantilla. Como la apariencia de los objetos puede cambiar en las imágenes en función de la luminosidad se suelen usar el gradiente de las imágenes para este tipo de metodología.

2.5.2.2 Basado en modelos multi-vista

Hasta ahora los algoritmos estudiados solo tienen un punto de vista del objeto, este método tiene en cuenta que un objeto se puede representar de diferentes formas en función del punto de vista. Por lo que para el seguimiento de un objeto "aprende" los diferentes puntos de vista del objeto.

Los métodos de entrenamiento de este tipo de algoritmo suele ser una máquina de vectores de soporte (SVM).

2.5.3 Seguimiento de la silueta

El seguimiento por silueta se usa para el seguimiento de objetos con forma compleja como puede ser las manos, la cabeza o la espalda. El objetivo de estos métodos es encontrar un objeto en un nuevo fotograma usando un modelo construido usando la información de fotogramas anteriores.

El seguimiento de la silueta se puede hacer mediante la comprobación de forma o evolución del contorno.

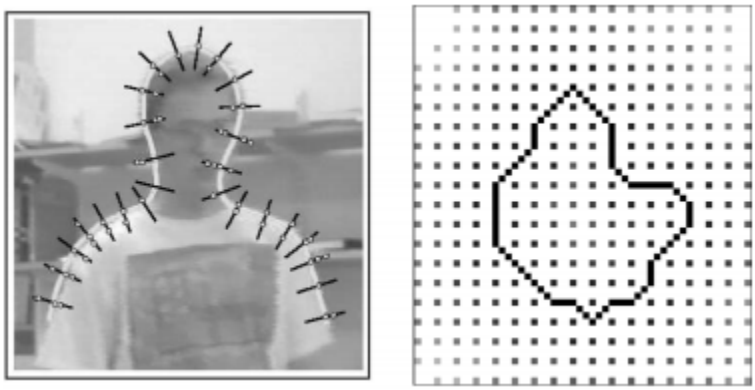

Figura 11. Seguimiento de la silueta

2.5.3.1 Comprobación de forma

El procedimiento general de este tipo de algoritmos es similar a los métodos basado en plantillas, es decir, se busca la silueta del objeto y su modelo asociado en el fotograma actual. Como el modelo se va generado a lo largo del seguimiento, tiene en cuenta la apariencia del objeto en momentos anteriores, permite un mejor seguimiento del objeto aunque cambie el punto de vista.

2.5.3.2 Seguimiento de contorno

Al contrario que el método de comprobación de la forma, en este método se sigue la evolución del contorno del objeto de un fotograma a otro. Para esto se requiere que parte del objeto en el fotograma actual esté superpuesto al objeto del fotograma anterior.

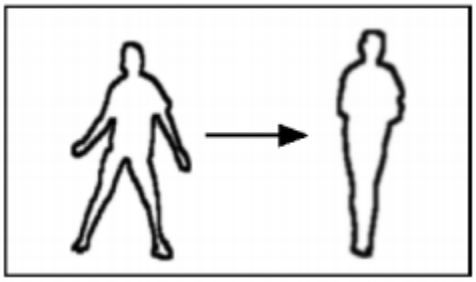

Figura 12. Seguimiento del contorno

2.6 Seguridad de los peatones en entornos viarios

Los accidentes en los que se ven envueltos peatones siguen siendo una gran preocupación para la seguridad vial. Los peatones, tanto en vías urbanas como interurbanas, son los usuarios más vulnerables ya que en caso de accidente son los que tienen mayor probabilidad de resultar heridos.

Peatón es el individuo, que sin ser conductor, transita a pie las vías públicas; también se considera peatón a quien empuja o arrastra otro vehículo sin motor, así como los minusválidos que circulan en silla de ruedas.

A pesar de que los accidentes mortales se han reducido en los últimos años, la cifra de muertes en los accidentes con peatones implicados sigue siendo alarmante, si en 2006 los peatones fallecidos eran un 15%, en el año 2010 la cifra asciende un 19%. %, mientras que en 2013 la cifra alcanza un preocupante 27%, según un estudio realizado por la OMS en el año 2013

En la Figura se muestra los porcentajes de fallecidos, en función del medio de movilidad elegido, en vías urbanas e interurbanas en algunas regiones del mundo.

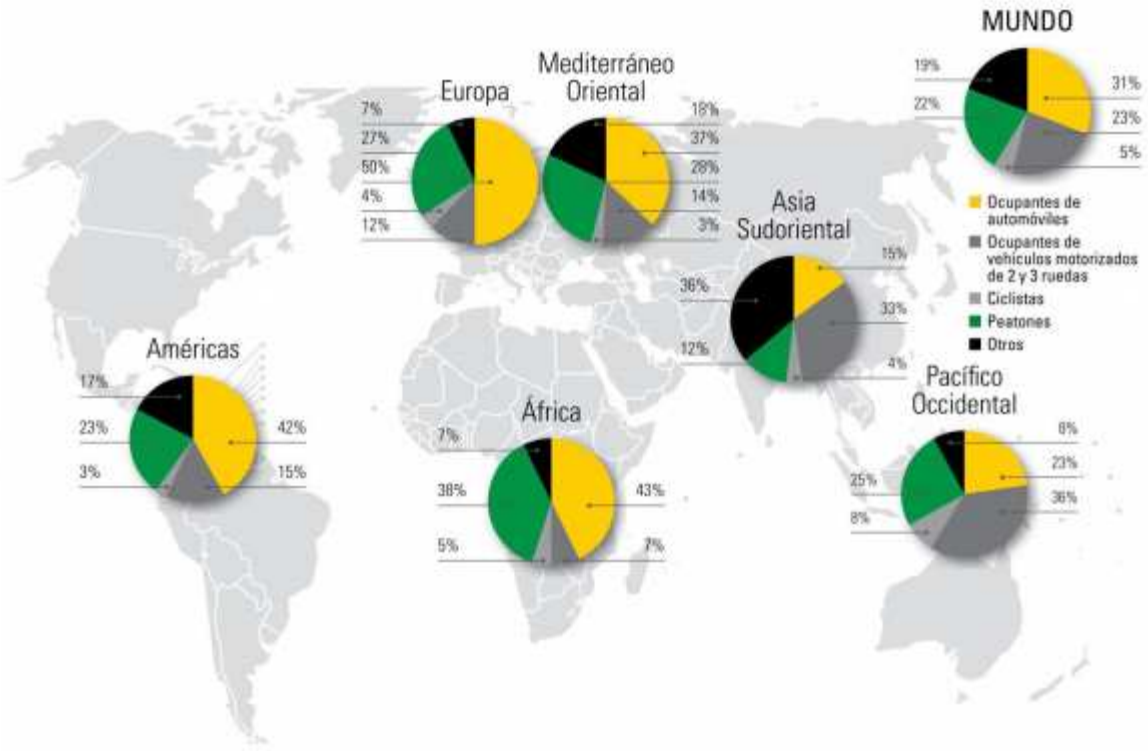

Figura 13. Accidentes de tráfico con resultado de muerte en el mundo

Los peatones atropellados en ciudad tiene una menor probabilidad de resultar gravemente heridos o muertos debido a que la velocidad en estas vías es menos que en vías interurbanas. Pero el porcentaje de atropellos es superior en ciudad que en vías interurbanas.

Como resultado de los accidentes con peatones implicados también se producen heridos resultantes, tanto graves como leves. Por ejemplo, en el año 2013 de las 89.519 víctimas de un accidente de tráfico, 378 de los muertos fueron peatones; además 2.053 personas resultaron heridas graves por atropello en vía urbana o ciudad. (RACE & GOODYEAR, 2015).

3 Plataforma de trabajo

3.1 Software

3.1.1 Versión de Android utilizada

Inicialmente, cuando se empezó a desarrollar el proyecto se uso un Smartphone cuya versión Android era la 2.2 conocida con el nombre "Froyo" y correspondiente a la API 8 lanzada el 20 de Mayo de 2010. Por esto fue la versión utilizada para desarrollar el proyecto, además es la primera versión con la que se asegura el correcto funcionamiento de las librerías de OpenCV.

Posteriormente se ha cambiado el modelo de Smartphone y por lo tanto la versión de Android utilizado, la versión del nuevo Smartphone es la 4.4, llamada "KitKat", cuya API es la 19 y fue lanzada el 31 de Octubre de 2013.

A pesar de que el actual Smartphone tenga una versión superior, para la realización del proyecto se ha mantenido la versión inicial. Ya que la versión Android en la que se desarrolla una aplicación determina a partir de que versión funciona, es decir, si una aplicación se desarrolla en la versión 2.2 funcionara en la versión 4.4 sin que haya problemas de incompatibilidades. Además como se ha visto en la Figura 2. la distribución actual de las versiones Android empiezan con la versión "Froyo" por lo que esta aplicación se puede usar en la mayoría de los Smartphone que se hallan actualmente en el mercado.

3.1.2 Versión OpenCV utilizada

La versión de OpenCV utilizada para desarrollar este proyecto es la 2.4.10, que se ha descargado de la página oficial de Android.

Dentro del fichero descargado, además de las funciones desarrolladas de OpenCV para Android, hay varios proyectos ya implementados en los que se muestra el uso de las librerías OpenCV como por ejemplo "15 puzzle", "camera-calibration", "face-detection", "image-manioulations", etc.

Para poder usar las funciones de OpenCV en el entorno de desarrollo elegido (Eclipse) hay que importar el fichero descargado dentro del espacio de trabajo donde se va a realizar el proyecto.

En caso de que se quiera desarrollar aplicaciones para Android en código nativo (C/C++ por ejemplo) también sería necesario instalar un "plugin" llamado Android NDK, que permite reutilizar código escrito en lenguajes nativos.

3.1.3 Eclipse + Android SDK

Eclipse es un entorno de desarrollo integrado (IDE) con varias herramienta de programación; como por ejemplo editor de texto, compilación en tiempo real, control de versiones, asistentes para creación de proyectos, clases, test, etc.

Eclipse fue desarrollado inicialmente por IBM y actualmente lo desarrolla la Fundación Eclipse, que fomenta una comunidad de código abierto. Tiene licencias de software libre pero no es compatible con la Licencia pública general GNU.

Eclipse se puede utilizar en diversas plataformas como pueden ser Windows, Linux, Mac. Además de permitir trabajar en diferentes plataformas también permite trabajar en diferentes leguajes de programación; Java, C/C++, Pyton, Android.

En este proyecto, al tratarse de un desarrollo en Android, se necesita una herramienta de desarrollo de Android (Android SDK). Android SDK posee las herramientas necesarias para desarrollar aplicaciones para esta plataforma: APIs, depurador, emulador, control de las funciones del dispositivo. No se recomienda usar el emulador ya que hay algunas funciones no se ejecutan de forma correcta.

Para poder usar esta herramienta de desarrollo en Eclipse se la vincula al entorno de desarrollo mediante un "plugin" llamado ADT (Android Development Tools).

3.2 Hardware

3.2.1 Modelo de Smartphone utilizado

Los Smartphones usados para el desarrollo del proyecto han sido dos muy diferentes.

El Smartphone con el que se empezó a desarrollar el proyecto fue un (Samsung Galaxy Mini S5570), de primera generación. Es un Smartphone de baja gama, lanzado en Enero de 2011 cuya versión de Android es la 2.2 ("Froyo"). Permite conectarse al ordenador en modo depuración, lo que es útil a la hora de probar la aplicación en desarrollo y posee cámara trasera de 3.5 megapíxeles.

Posteriormente, en el verano de 2015, se adquirió un nuevo terminal, un dispositivo (Samsung Galaxy J1). Al igual que el anterior es un Smartphone de gama baja pero tiene mayor memoria interna y RAM. Además dispone de una versión Android 4.4 ("Kit-Kat") . Este dispositivo fue lanzado al mercado en Enero del 2015. Permite conexión a ordenador en modo depuración y posee cámara trasera de 5 megapíxeles y delantera de 2 megapíxeles.

3.2.2 Cámara utilizada

Como se ha dicho en el apartado 3.2.1, el dispositivo que se ha usado posee dos cámaras, una delantera y otra trasera.

Para esta aplicación se usara la cámara trasera cuya resolución es de 5 megapíxeles (2592 x1944 píxeles); no posee una resolución muy alta, comparada con los actuales Smartphone del mercado, pero para esta aplicación la información que aporta es suficiente.

4 Desarrollo del algoritmo

En este proyecto se plantea desarrollar una aplicación, para un sistema operativo Android, que sea capaz de detectar peatones y hacer el seguimiento de los peatones detectados.

La elección del sistema operativo Android, en lugar de otros sistemas operativo para Smartphone, como pueden ser WindowsPhone o iOS, es debido a que su uso están muy extendido en el mercado actual. Además Android es una plataforma de código abierto que tiene múltiples colaboradores y desarrolladores, mejorando la plataforma. Además, es fácil de usar ya que se puede trabajar con él, con entornos de desarrollo intuitivos como Eclipse; en plataformas Android se pueden usar diferentes librerías de código abierto, como con las librerías OpenCV o OpenGL.

La idea de detectar peatones y su posterior seguimiento es evitar los accidentes de tráfico con peatones implicados que se producen cada año. Los peatones, junto con los ciclistas y conductores de ciclomotores, son los individuos más vulnerables en un entorno viario ya que en caso de cualquier incidente es más probable que resulte herido.

Esta aplicación se puede usar no solo en un coche autónomo sino también en los vehículos que necesitan conductor, ya que esta avisa de la presencia de peatones, en la zona hacia donde apunta la cámara, de modo que el conductor o el vehículo autónomo actúen en consecuencia y circulen con mayor precaución ante estas circunstancias, evitando de este modo los accidentes anteriormente mencionados.

Una vez se tienen claros los objetivos y las causas que dan lugar al proyecto su puede proceder al desarrollo del mismo.

4.1 Algoritmos utilizados

4.1.1 Detección de peatones

Como se ha visto en el apartado 2.4 hay diferentes métodos para detectar peatones, en este caso se ha usado un algoritmo holístico con un extractor de características Haar-like.

4.1.1.1 Características Haar-like

Las características Haar like son ciertas características de la imagen digital usadas en el reconocimiento de objetos.

La forma de obtener las características de un objeto, se aplica a una imagen las características Haar-like en determinadas zonas de la imagen. Posteriormente se suma la intensidad de los pixeles de cada región de la característica Haar-like, zona blanca y negra. Y para obtener el valor las características de las secciones de la imagen se restan los

valores de intensidad de la zona blanca con los de la zona negra, el cálculo de estos valores generalmente se hacen mediante una imagen integral.

La imagen integral es una estructura de datos cuyos píxeles son la suma de los valores de los píxeles más próximos. En una imagen integral sea cual sea la región elegida, siempre que tenga una forma rectangular, se puede resumir la suma del valor de sus píxeles sumando únicamente los píxeles de las esquinas de región elegida. El valor de los píxeles de la imagen integral se obtiene a partir de la siguiente fórmula; para la aplicación de esta fórmula, las posiciones que están fuera de la imagen integral tienen valor cero:

$$S(x,y) = i(x,y) + S(x-1,y) + S(x,y-1) - S(x-1,y-1)$$

Donde:
S (x,y): es el valor del píxel con las coordenadas x e y, en la imagen integral

i (x,y): es el valor del píxel con las coordenadas x e y, en la imagen original
S(x-1, y): es el valor del píxel de la izquierda, al que se está analizando, en la imagen integral
S (x, y-1): es el valor del píxel superior, al que se está analizando, en la imagen integral
S (x-1, y-1): es el valor del píxel superior-izquierdo al que se está analizando, en la imagen integral

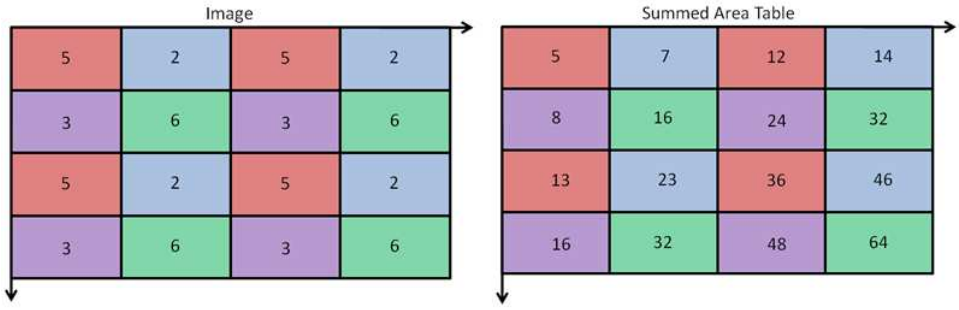

Figura 14. Izquierda: valores de los píxeles de la imagen original. Derecha: valores de una imagen integral (obtenida a partir de la imagen de la izquierda)

Las características obtenidas en cada imagen de muestra pueden variar y unas pueden caracterizar mejor el objeto que otras; por lo que se requiere un entrenamiento para extraer las características más interesantes del objeto a detectar. Para el entrenamiento de este tipo de algoritmo se usa un algoritmo AdaBoost. Usando este tipo de entrenamiento se consigue crear un clasificador fuerte a partir de clasificadores débiles mediante un sistema de votación basado en pesos. Al contrario que las SVM's y las redes neuronales, el proceso de entrenamiento AdaBoost solo selecciona las características conocidas que mejoran el poder predictivo del modelo.

4.1.2 Seguimiento de peatones

Como se vio en el apartado 2.5 hay diferentes alternativas para hacer el seguimiento de los peatones, a continuación se va a ver algunas de ellas.

Generalmente para hacer el seguimiento de una persona depende de la forma en se quiera definir, en función del esto se puede usar uno u otro algoritmo de seguimiento como se puede ver en la Figura 8.

En este proyecto se va a usar un algoritmo de seguimiento por puntos, ya que son los que requieren de un menor coste computacional. Dentro de los algoritmos de seguimiento por puntos hay dos subdivisiones:

- Métodos deterministas: consiste en una correspondencia de puntos uno a uno, asignando en cada fotograma un coste de asociación a cada objeto. Este coste se define usando alguna de las restricciones siguientes: asume que la posición del objeto no cambia significativamente de un fotograma a otro, tiene una velocidad máxima a la que puede moverse el objeto, los cambios de velocidad del objeto no cambia bruscamente, asume que el objeto es rígido (la distancia entre 2 puntos del objeto se mantienen constante). Un ejemplo de este tipo es el algoritmo húngaro (Kuhn, 1955).
- Métodos probabilísticos: usan el espacio de estados que modela las propiedades del objeto como por ejemplo la posición, la velocidad o la aceleración. Para hacer el seguimiento es necesario obtener mediciones de las propiedades del objeto, estas mediciones pueden contener ruido. Generalmente estos algoritmos estiman el siguiente estado del objeto del que están haciendo el seguimiento. Un ejemplo de estos algoritmos pueden ser el filtro de partículas (Gordon, Salmond, & Smith, 1993) o el filtro de Kalman (Kalman, 1960).

4.1.2.1 Filtro de Kalman

El filtro de Kalman es un algoritmo fue desarrollado por el húngaro Rudolf E. Kalman en 1960. Este filtro es un algoritmo de estimación cuadrática lineal, capaz de estimar un estado no medible de un sistema dinámico lineal teniendo en cuenta únicamente las medidas observadas en el momento actual, estas medidas pueden contener ruido, y el estado calculado previamente; este sistema está basado en la teoría probabilística que se conoce como cadena de Márkov.

La cadena de Márkov es un tipo especial de proceso estocástico, en el que la probabilidad de que ocurra un evento depende únicamente del evento anterior. Es decir, toda la información relevante de un sistema hasta su instante actual esta resumido en su estado presente, y esta información es usada para describir su probable estado futuro.

El filtro de Kalman considera que el sistema está modelado con un comportamiento dinámico (por ejemplo, sigue las leyes físicas del movimiento), del cual se conoce las entradas de control del sistema y múltiples medidas secuenciales con las que se puede estimar la variación del estado del sistema. Al tener múltiples medidas se obtiene una mejor estimación que si la estimación se realizará con una única medición.

Las mediciones del sistema pueden ser difíciles de obtener o tener un cierto grado de ruido, por lo que para determinar cómo afecta la medición, del momento actual, a la estimación del estado del sistema se define la ganancia 'K'. Una de las ventajas del filtro de Kalman con respecto a otros métodos estimadores de estado, como por ejemplo el observador de Luenberger, es que el cálculo de la ganancia 'K' se calcula teniendo en cuenta el error en las predicciones anteriores y el ruido que pueden tener las medidas.

Como se ha dicho en los párrafos anteriores, para determinar el estado futuro del sistema, además de la medición en el momento actual, también se necesita conocer el estado previo del sistema. Debido a esta necesidad el filtro de Kalman tiene un comportamiento recursivo; ya que usa el estado estimado en el momento actual, 'k', como entrada para estimar el estado en el siguiente momento, 'k+1'.

Para poder trabajar con varias dimensiones sin que ello implique un gran número de cálculos, el filtro de Kalman usa matrices para definir, por ejemplo, la transición de un estado a otro, las covariancias del error y vectores para definir tanto el estado del sistema como las entradas de control o las medidas del sistema en el momento actual. Las matrices también permiten representar la relación lineal entre las distintas variables de estado; posición, velocidad o aceleración.

A continuación se va a analizar las matemáticas que usa el filtro de Kalman para estimar el estado del sistema.

Para ello lo primero es definir el espacio de estados, vector que contiene las variables características del estado del sistema, en este caso:

$$X_k = \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix}$$

El filtro de Kalman asume que el estado en el momento 'k' evoluciona desde el estado en el momento anterior 'k-1', mediante la siguiente ecuación:

$$X_k = A_k X_{k-1} + B_k u_k + w_k$$

Donde:

X_k: vector de estado (posición, velocidad) en el momento 'k'

A_k: matriz de transición de estados, desde el estado en el momento 'k−1' al estado en el momento 'k'

X_{k-1}: vector de estado (posición, velocidad) en el momento 'k-1'

B_k: matriz de entradas de control, describe como afectan las entradas de control al vector de estados en el momento 'k'

u_k: vector de entradas de control, en el momento 'k'

w_k: vector que contiene el ruido del proceso para cada parámetro del vector de estados. Se asume una distribución Gaussiana de media 0 y covarianza dada por la matriz de covarianza Q_k.

Las medidas observadas en el momento 'k', z_k, son el verdadero estado del sistema en dicho momento y se pueden modelar según la siguiente ecuación:

$$z_k = H_k X_k + \nu_k$$

Donde:

z_k: vector de medidas en el momento 'k'

H_k: matriz que indica la relación entre las mediciones y el vector de estado en el momento 'k', suponiendo que no hubiera ruido en la medición

X_k: vector de estados (posición, velocidad) en el momento 'k'

ν_k: vector del ruido en la observación de la medida. Se asume una distribución Gaussiana de ruido blanco de media 0 y covarianza R_k

Se asume que el estado inicial y los vectores de ruido en cada paso, $\{x_0, w_1, ..., w_k, v_1, ..., v_k\}$, son independientes unos de otros

El filtro de Kalman se puede dividir conceptualmente en dos fases, "Predicción" y "Corrección". En la fase de predicción se usa el estado estimado en el momento anterior ('k-1') para estimar el estado del sistema en el momento actual ('k'). Este estado predicho, también llamado 'a priori', estima el estado del sistema en el momento actual ('k') pero no contiene información sobre las medidas observadas en el momento actual ('z_k'). En la fase de corrección, la estimación 'a priori' se combina con la medición observada en el momento actual para mejorar la estimación, también llamada estimación 'a posteriori'.

Las ecuaciones de la etapa de predicción y de corrección se muestran a continuación en la Figura 15 :

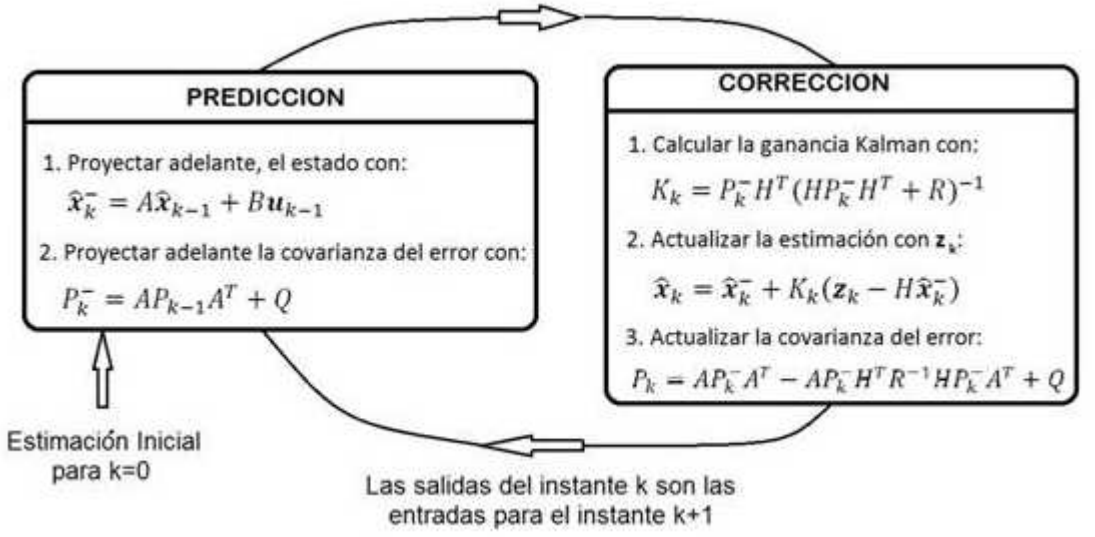

Figura 15. Ciclo del algoritmo de Kalman

Donde:

Fase de 'Predicción':

$\hat{x}_k^-$: vector de estado estimado, en el momento k, calculado 'a priori'

A: matriz de transición de estados, desde el estado en el momento 'k−1' al estado en el momento 'k'

$\hat{x}_{k-1}$: vector de estado en el momento 'k-1'

B: matriz de entradas de control, describe como afectan las entradas de control al vector de estados en el momento 'k'

u_{k-1}: vector de entradas de control en el momento 'k-1'

P_k^-: matriz de covarianzas del error, en el momento 'k', calculado *'a priori'*

P_{k-1}: matriz de covarianzas del error en el momento 'k-1'

Q: matriz de covarianzas del ruido, sigue una distribución Gaussiana

Fase de 'Corrección':

K_k: Ganancia de Kalman en el momento 'k'

H: matriz que indica la relación entre las mediciones y el vector de estado en el momento 'k', suponiendo que no hubiera ruido en la medición

R: matriz de covarianzas del ruido en la medida, sigue una distribución Gaussiana

$\hat{x}_k$: vector de estado estimado, en el momento k, calculado 'a posteriori' (después de incorporar la medición)

z_k: vector de medidas en el momento 'k'

P_k: matriz de la covarianza del error, en el momento k, calculado 'a posteriori' (después de incorporar la medición)

5 Implementación del algoritmo

Después de presentar los conocimiento teóricos aplicados en este proyecto, a continuación se va a explicar cómo se han usado e implementado. El algoritmo en general usado se puede dividir en dos partes; detección y posterior seguimiento del peatón y el funcionamiento de la actividad principal se puede ver en la Figura 16.

5.1 Detección de peatones

Acerca del funcionamiento de algoritmo de detección de peatones no se va a dar mucho detalle ya que fue proporcionado por el departamento de Sistemas Inteligentes pero si se va a explicar cómo se usa en este proyecto.

El algoritmo utilizado para la detección de peatones es un clasificador en cascada. Por lo que, en este caso, se va a usar la clase CascadeClassifier, ya implementada en OpenCV. La función concreta que se va a usar se llama detectMultiScale.

Esta clase dispone de varios constructores pero en este caso se hace uso del constructor que permite crear un nuevo objeto de este tipo a partir de un fichero con clasificadores ya entrenados. Por esto nada más iniciar la aplicación, en el método onResume( ), se crea un nuevo clasificador en cascada llamado mJavaDetector a partir del fichero, con un clasificador ya entrenado con muestras positivas y negativas de peatones, *"haarcascade_ped.xml"*.

Una vez se tiene ya creado el clasificador, éste se puede usar en las imágenes que provienen de la cámara para detectar peatones.

La detección de peatones se hace en tiempo real por lo que se aplica el clasificador en cascada, mJavaDetector, a cada fotograma. Por ello en la función onCameraFrame (CameraViewFrame inputFrame), después de obtener la imagen a analizar en escala de grises, se llama a la función detectMultiScale(Mat image, MatOfRect objects, double scaleFactor, int minNeighbors, int flags, Size minSize, Size maxSize) donde:

- image: es la imagen que se quiere analizar de tipo Mat
- objects: es un vector de rectángulos donde cada uno contiene un objeto detectado
- scaleFactor: es un parámetro que especifica la escala en que se reduce la imagen
- minNeighbors: es un parámetro que especifica cuantos vecinos debe tener cada candidato para que sea considerado valido
- flags: es el modo en que se quiere hacer la clasificación, por ejemplo detección de bordes de Canny o haar scale.
- minSize: es el tamaño mínimo del posible objeto. Objetos más pequeños son ignorados.
- maxSize: es el tamaño máximo del posible objeto. Objetos más grandes son ignorados.

Una vez se tiene la lista de rectángulos con los objetos detectados estos se muestran en la pantalla, rodeando el peatón detectado con un rectángulo verde. Y posteriormente se procede al seguimiento del peatón.

5.2 Seguimiento del peatón

El filtro de Kalman se ha implementado en el proyecto mediante una clase llamada MyKalmanFilter, dicha clase dispone de las siguientes funciones públicas:

- predict ( ): calcula la predicción del vector de estado. Siguiendo las ecuaciones de la fase de predicción, que se pueden ver en la Figura 9. Además maneja la posibilidad de que no haya una actualización, mediante las medidas reales tomadas, hasta la siguiente vez que se use esta función.
- predict (control): tiene el mismo funcionamiento que la función predict ( ), solo que en este caso se tiene en cuenta los parámetros de control del sistema.
- correct (measurement): actualiza el estado predicho teniendo en cuenta el estado actual medido.
- Así como getters y setters de sus atributos, como "mStatepre" (estado estimado 'a priori'), "mStatePost" (estado estimado 'a posteriori'), la matrices de covarianza de error y ruido, las matrices de transición o la ganancia del filtro.

Para contener la información acerca del peatón seguido se ha creado la clase MyTrackerInfo, que tiene los siguientes atributos:

- mTimeNotFound: en este atributo se almacena el tiempo que lleva sin detectarse al peatón.
- mPedestrianHeight: altura del rectángulo del peatón que se está siguiendo
- mPedestrianWidth: ancho del rectángulo del peatón que se está siguiendo
- mPedestrianX: última posición del peatón en el eje X
- mPedestrianY: última posición del peatón en el eje Y

Por último se ha creado una clase llamada KalmanTracker cuyos atributos principales son el filtro de Kalman y la información sobre el peatón seguido. Esta clase es la que contiene el algoritmo de seguimiento del peatón.

La función principal de seguimiento de peatón recibe el nombre de pedestrianTracker (pedestriansVector, dT, imageSize). En esta función si no existe un filtro de Kalman para el seguimiento del peatón se crea uno nuevo con el primer peatón almacenado en el vector "pedestriansVector", si existe, mediante la función createKalmanFilter (pedestrian) y posteriormente se procede a su seguimiento. En caso de que ya exista un filtro creado se actualiza su estado con la función updateKalmanFilter (pedestriansVector, dT, imageSize),

siempre que el tiempo sin que el peatón haya sido detectado sea inferior a dos segundos. Si el peatón lleva sin ser detectado más de dos segundos se elimina el seguimiento del peatón.

Cuando se crea el filtro de Kalman con la función createKalmanFilter (pedestrian) se configura los valores de las matrices de transición 'A', de medidas 'H', de covarianzas del ruido 'Q' y 'R', así como el estado inicial del peatón.

Para actualizar el estado del peatón se usa la función updateKalmanFilter (pedestriansVector, dT, imageSize). En esta función lo primero que se hace calcular la estimación 'a priori' del estado del peatón. Después se comprueba si se ha detectado algún peatón en el fotograma actual. Si es el caso, se comprueba si este peatón es el mismo que se ha estado siguiendo, para ello se define una Región de Interés (ROI). La Región de Interés se define alrededor del lugar donde se detecto al peatón con anterioridad, con un margen igual a su tamaño. Una vez se ha definido los límites del ROI, se analiza si las coordenadas de los peatones presentes en el vector pedestriansVector se encuentra en su interior, en caso de que sea así se usa estas medición para calcular la estimación del estado del peatón 'a posteriori'. Si, no se ha detectado ningún peatón en el fotograma actual o los peatones detectados no son el que se estaba siguiendo, se suma el intervalo de tiempo entre un fotograma y otro al tiempo que no se ha encontrado al peatón; el tiempo en que no se detecta el peatón debe ser consecutivo para ser considerado.

Esto funciona bien si se quisiera seguir a un solo peatón, pero en este caso se quiere controlar todos los posibles peatones que se encuentren alrededor del vehículo, por lo que en la actividad principal se ha definido un vector de KalmanTracker, uno para cada peatón detectado. En cada fotograma se actualiza el estado de los filtros existentes y en el caso de que haya peatones sin seguimiento se crea un nuevo filtro. Una vez se ha terminado de seguir a un peatón este se elimina del vector.

Por último una vez que se tiene la estimación del estado de todos los peatones esta se muestra por pantalla, marcando el estado estimado del sistema con un rectángulo rojo.

5.3 Modelo dinámico del peatón

Para modelar dinámicamente el movimiento del peatón se considera que tiene un movimiento rectilíneo uniforme que cumple la siguiente ecuación:

$$x_k = x_{k-1} + v_x \Delta t$$

Con lo que las matrices principales del filtro de Kalman quedarían:

$$A = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} y\ H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

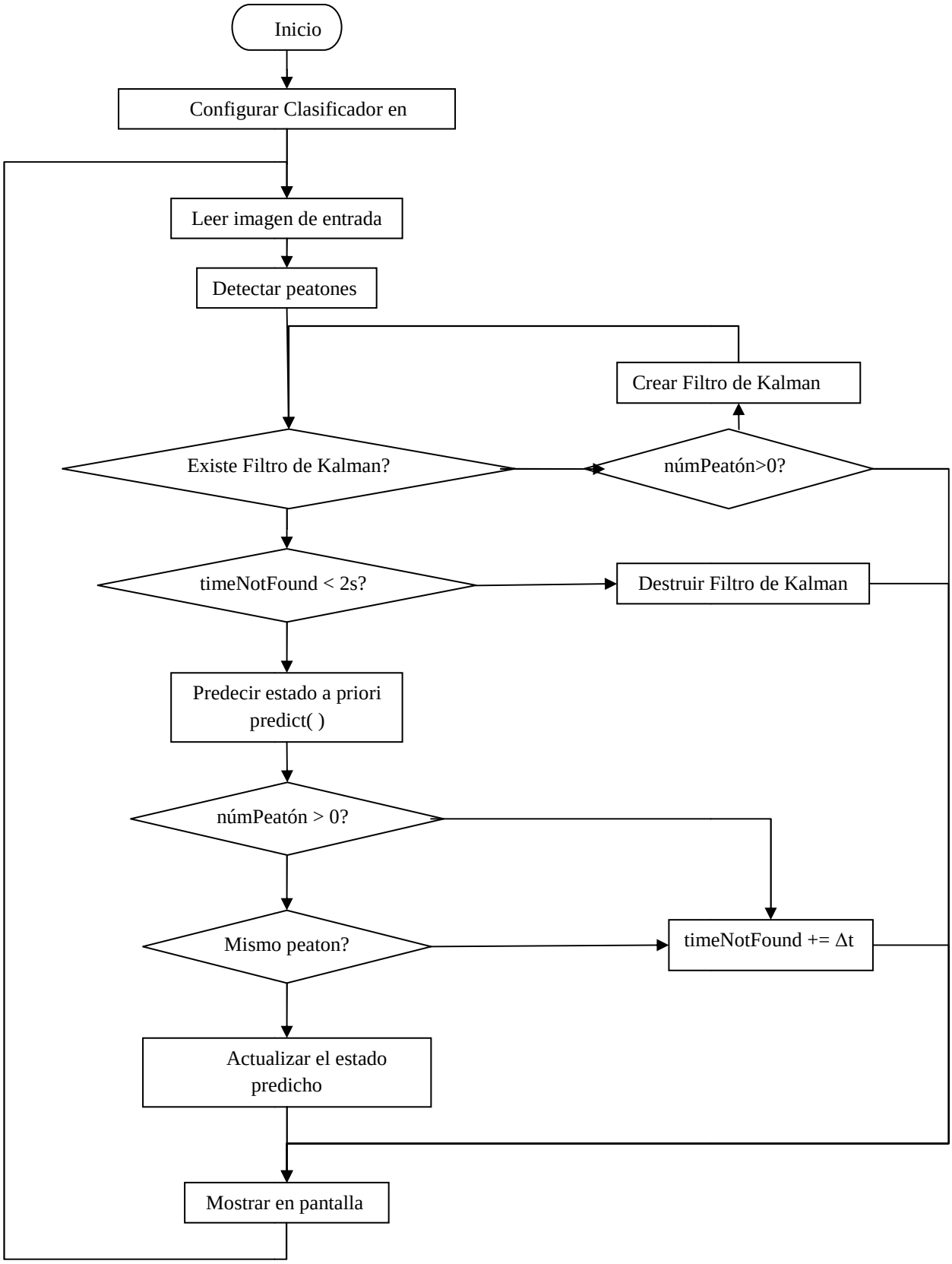

Figura 16. Diagrama de flujo de la aplicación

6 Pruebas

Las pruebas realizadas con la aplicación se presentan a continuación, las pruebas se realizaron con un dispositivo móvil.

6.1 Prueba de seguimiento de un solo peatón

Inicialmente se desarrollo la aplicación para que realizará el seguimiento de un solo peatón. y se realizó la siguiente prueba en un entorno real. En la Figura 17. Izquierda se puede ver el peatón detectado (rectángulo verde) y su estado estimado (rectángulo rojo). Y en la Figura 17. Derecha se puede ver cómo a pesar de la ausencia de detección se continua con el seguimiento del peatón.

Figura 17. Izquierda: Detección y seguimiento de un peatón. Derecha: seguimiento del peatón en ausencia de dectección

Como también se puede ver en la Figura 17. Derecha a pesar de que se detecta otro peatón solo se realiza el seguimiento de uno.

Cuando hay dos peatones se juntan o caminan juntos confunde las medición de la posición de uno con el otro, como se puede ver en la Figura 18.

Figura 18. Confusión al elegir el peatón que se esta siguiendo cuando hay dos peatones juntos

6.2 Prueba de seguimiento de varios peatones

Como el objetivo de este proyecto era detectar los peatones que haya cerca del vehículo en un entorno viario, se mejorado el código usado en las pruebas del apartado anterior. En la Figura 19 se puede ver como se está detectando y siguiendo a un peatón y al detectar un segundo también realiza el seguimiento del segundo, incluso cuando no hay detección de ninguno de los dos.

Figura 19. Detección y seguimiento de dos peatones

Al usar el seguimiento para varios peatones en algunas ocasiones se crean muchos más filtros de seguimiento que peatones hay detectados. Como se puede ver en la Figura 20. se crean tres filtros solo para uno de los peatones mientras que de los otros no se hace seguimiento.

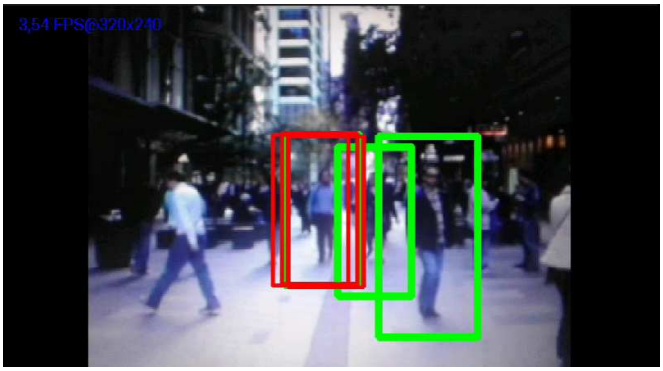

Figura 20. Fallo en el seguimiento

7 Conclusiones y trabajos futuros

En general se han conseguido los objetivos de este proyecto. Se ha desarrollado una aplicación para plataformas Android, que funciona de una manera fluida y permite detectar y seguir los peatones captados por la cámara del dispositivo móvil.

A pesar de que la aplicación es capaz de detectar varios peatones, tiene algunos fallos que ya se han descrito en el apartado anterior. Por ejemplo en cuando hay dos peatones juntos y uno de ellos se mueve, no es capaz de mantener el correcto seguimiento del peatón. Otro de los fallos es que se crean varios filtros de seguimiento para un mismo peatón, en presencia de otros peatones a los que no se hace seguimiento, cosa que es perjudicial para el procesamiento de la aplicación.

Cuando no hay suficientes mediciones en la etapa inicial del seguimiento, el filtro de Kalman no es capaz de realizar una estimación de estado correcta, lo cual es problemático por que en estos casos no se predice adecuadamente donde se encuentra el peatón.

El funcionamiento del seguimiento depende mucho de la detección del peatón, ya que al menos se necesita la medición inicial para empezar el seguimiento, es decir, la aplicación depende de que una correcta detección. En caso de que el algoritmo de detección no detecte bien el peatón o tenga falsos positivos, el seguimiento también fallará.

7.1 Trabajos futuros

En párrafos anteriores se ha dicho que el algoritmo tiene algunos problemas de funcionamiento, por lo que uno de los posibles trabajos futuros es mejorar el algoritmo de forman que se solucionen estos problemas. Es decir:

- Mejorar la detección en situaciones en las que haya peatones muy próximos unos de otros.
- Mejorar la separación del seguimiento cuando haya peatones que se junten o crucen.

También se ha dicho que el algoritmo de seguimiento depende mucho del algoritmo de detección, por lo que uno de los trabajos futuros puede ser realizar el seguimiento con otro de los métodos de seguimiento que existen. Una buena opción sería realizar el seguimiento mediante un algoritmo de seguimiento de contorno.

Esta aplicación se ha probado en situaciones con buena iluminación y visibilidad, por tanto otro de los trabajos futuros sería mejorar la aplicación para que funcionase por la noche o en circunstancias de poca visibilidad, con presencia de niebla o lluvia.

8 Bibliografía

COSMOS. (7 de Junio de 2016). *XAKATA ANDROID.* Obtenido de http://www.xatakandroid.com/mercado/1-de-cada-10-dispositivos-android-ya-estan-actualizados-a-marshmallow

download, O. (s.f.). *Open CV.* Obtenido de http://opencv.org/downloads.html

Freund, Y., & Schapire, R. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of computer and system sciences* , 119-139.

Gargenta, M. (2011). *Learning Android.* O'Reilly Media Inc.

Google. (s.f.). *Android*. Obtenido de http://www.android.com/intl/es_es/

Gordon, N., Salmond, D., & Smith, A. (1993). Novel approach to nonlinear/non-Gaussian Bayesian state estimation.

Goto, K., Kidono, K., Kimura, Y., & Naito, T. (2011). Pedestrian Detection and Direction Estimation by Cascade Detector with Multi-classifiers Utilizing Feature Interaction Descriptor. *Intelligent Vehicles Symposium*, (págs. 224-229). Baden-Baden.

Guenveur, L. (15 de 06 de 2016). *Kantar España Insights.* Obtenido de http://es.kantar.com/tech/m%C3%B3vil/2016/junio-2016-cuota-de-mercado-de-smartphones-en-espa%C3%B1a-abril-2016/

itseez. (s.f.). *OpenCV*. Obtenido de http://www.opencv.org

Kalman, R. E. (1960). A new Approach to Linear Filtering an Prediction Problems. *Journal of BasicEngineering* , 35-35.

Kuhn, H. (1955). The Hungarian Method for Assignment.

Messom, C., & Barezak, A. (2006). Fast and Efficient Rotated Haar-like Features using Rotated Integral Images. *Australian Conference on Robotics and Automation*, (págs. 1-6).

RACE, & GOODYEAR. (10 de 08 de 2015). *RACE.* Obtenido de http://www.race.es/race.es/documentos/seguridad_vial/biblioteca/factor_humano/Informe%20RACE-GOODYEAR%20Atropellos%20en%20zona%20urbana%202015.pdf

Samsung Galaxy J1. (s.f.). Obtenido de smartGSM: http://www.smart-gsm.com/moviles/samsung-galaxy-j1

Samsung Galaxy Mini S5570. (s.f.). Obtenido de smartGSM: http://www.smart-gsm.com/moviles/samsung-galaxy-mini-s5570

Suard, F., Rakotomamonjy, A., Bensrhair, A., & Broggi, A. (2006). Pedestrian Detection using Infrared images and Histograms of Oriented Gradients. *Intelligent Vehicles Symposium*, (págs. 206-212). Tokyo.

Triggs, B., & Dalal, N. (2005). Histograms of Oriented Gradients for Human Detection. *Computer Society Conference on Computer Vision and Pattern Recognition.*

Viola, P., & Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. *Computer Vision and Pattern Recognition*, (págs. 511-518).

Yilmaz, A., Javed, O., & Shah, M. (2006). Object Tracking: A Survey.