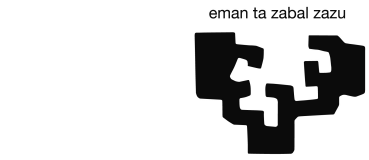


Ingeniería en Informática

Proyecto de Fin de Carrera

Detección de personas

Autor

Jon Intxaurbe Txarterina

2013

Resumen

Hoy en día, todavía son muchas las personas que pierden la vida en accidentes de trabajo por culpa de atropellos. A pesar de que los vehículos que se desarrollan ahora son mucho mas seguros que antes, también son más rápidos y más numerosos . Los nuevos sistemas de seguridad van más allá de la implantación de materiales que absorban mejor los impactos o componentes que reduzcan las lesiones de los ocupantes del vehículo producidas durante el accidente. En la actualidad, los ingenieros de la industria tratan de evitar que se produzcan estos accidentes.

El presente proyecto se centra en proteger a los peatones de las vías urbanas o de una fábrica donde conviven con robots móviles, pues son los mayores afectados en los accidentes producidos en estos entornos. El objetivo es diseñar un algoritmo basado en visión mono capaz de detectar a los usuarios de forma rápida y precisa de tal forma que se tenga constancia en todo momento de los peatones que se encuentran delante del vehículo.

La técnica que se va a utilizar para la localización de los peatones se basa en los histogramas de gradientes orientados (HOG). Se trata de un método que ofrece resultados robustos gracias a su invariación ante cambios en la iluminación, en el fondo o en las posturas de los peatones. Mediante una serie de operaciones previas se busca conseguir que esta detección se realice en tiempo real. Es necesario realizar un estudio de los distintos parámetros del sistema para alcanzar buenos resultados tanto en el tiempo de cómputo del algoritmo como en la eficacia de la detección. Para el entrenamiento de imágenes se ha utilizado SVM. Para el desarrollo del algoritmo se recurre a las librerías OpenCV, muy útiles para el procesamiento de imágenes y la visión artificial.

Palabras clave: visión artificial, detección de humanos, HOG, SVM.

Índice general

Índice de figuras

Indice de tablas

PARTE I: GESTIÓN DEL PROYECTO

CAPÍTULO 1

Documento de objetivos del proyecto

1.1. Introducción

En este documento se recoge la descripción del proyecto, los objetivos buscados, el alcance del proyecto y la estructura de descomposición, la planificación de trabajos a realizar, así como el análisis de riesgos y las políticas de contención y seguridad para el correcto desarrollo y finalización del proyecto.

1.2. Descripción

Detección de personas es el nombre dado al proyecto propuesto por Gorka Azkune de *Tecnalia Fatronic* como Proyecto Fin de Carrera (II). Este trabajo se enmarca dentro del proyecto Europeo AUTORECON. El objetivo final de este proyecto AUTORECON es la implementacion de un nuevo procedimiento de navegación basado en la visión artificial capaz de detectar a personas. El sistema cuenta con una cámara mono.

La concepción inicial del proyecto era la elaboración de un modelo 3D capaz de integrar los diversos dispositivos fuente y de realizar una detección de objetos en tiempo real.

El problema de la detección de objetos en secuencias de vídeo es un tema de actualidad y muy estudiado por la comunidad científica. La combinación de técnicas de visión por computador utilizadas en este proyecto no son las únicas y no necesariamente las óptimas,

por este motivo vamos a comentar técnicas del estado del arte parecidas a cada una de las que hemos utilizado y así explicar los motivos que nos han llevado a usarlas. Conviene aclarar que las explicaciones dadas a continuación por cada técnica serán algo breves ya que más adelante se explicarán las elegidas con más detalle.

Para realizar la tarea de detección y seguimiento de personas, así como para muchas otras, es necesario definir las especificaciones y condiciones sobre las cuales se va a aplicar. Es decir, con qué tipo de sensores se cuenta, cuáles son las condiciones ambientales, así como si funcionará sobre una plataforma móvil, o si el sensor se encontrará en una posición estática. Una vez establecidas estas condiciones, se pueden seleccionar las técnicas más adecuadas.

1.3. Contenido de la memoria

En este primer capítulo, prescindiendo de muchos detalles en aras a la claridad, se han introducido varios conceptos sobre el proyecto desarrollado. Muchos de los temas cubiertos se plantearán con más detalle en los capítulos posteriores. El resto del documento presenta la siguiente estructura:

- En el primer capítulo se describirán los objetivos del proyecto con los recursos utilizados en el. También estará detallado la planificación del proyecto, tanto la inicial como la real, explicando por que ha habido tal desvío temporal.
- En el segundo capítulo, se analizara el estado del arte, y se incluirá una introducción esencial sobre la visión por computador, fundamental para las ideas básicas del proyecto.
- En el tercer capítulo, se analizarán en profundidad los conceptos técnicos sobre los que se sustenta nuestro objeto de estudio, así como el desarrollo del algoritmo.
- En el cuarto capítulo, la implementación del software, paso a paso.
- En el quinto capítulo, se describirán los experimentos realizados y se evaluarán los resultados obtenidos en los distintos experimentos, así como las posibles líneas de investigación que se dejan abiertas.

1.4. Objetivos iniciales del proyecto

Inicialmente, objetivo era la creación de un modelo capaz de detectar una persona en tiempo real con una camara en 3D, es decir, una cámara con doble lente para poder medir las distancias donde estaría la persona y hacer su seguimiento para la navegación de un robot en un entorno *indoor*. Sin embargo, el elevado coste de una estéreo cámara y su necesidad de recalibración periódica han hecho que se busque un metodo diferente para la detección de personas.

Existen diversas técnicas para el análisis de formas en imágenes. Algunos autores utilizan técnicas basadas en extracción de características y patrones de formas [1], [2], [3], [4], técnicas de Boosting [5], detección de objetos basada en formas mediante métodos Chamfer [6], correlación con patrones humanos probabilísticos [7], máquinas de soporte vectorial (SVM) [8], graph kernels [9], análisis de movimiento [10], análisis de componentes principales [11] y clasificadores basados en redes neuronales [12].

En la detección de personas es especialmente difícil buscar una solución, ya que las personas pueden encontrarse en diferentes posiciones, con diferentes prendas, diferentes fondos y diferentes condiciones de iluminación. A todo ello, debemos añadir el hecho de que el dispositivo de captura va a introducir un temblor en las imágenes debido a que el sistema va embarcado sobre el robot, además de tener que ser un sistema capaz de funcionar en tiempo real y con elevada tasa de acierto.

Nuestro método se basa en evaluar histogramas locales normalizados de una imagen de gradientes orientados. La imagen se divide en pequeñas celdas cada una de las cuales acumula direcciones del histograma de gradiente u orientaciones de los bordes de los píxeles de las celdas.Se recomienda para una mejor respuesta normalizar el contraste en unas zonas más grandes (denominadas bloques) y utilizar dicho resultado para normalizar las celdas del bloque. Estos bloques de descriptores normalizados son lo que los autores denominan descriptores HOG. Por último, se utilizan los descriptores HOG de la ventana de detección como entrada a un clasificador SVM^{light} [13], que es una implementación de un SVM (support vector machine) adecuada para trabajar con grandes conjuntos de datos.

Mientras por otra parte, se trata de un análisis cognitivo de la escena mediante técnicas de procesado de imagen en tiempo real para la empresa Fatronic Tecnalia.

El algoritmo ha sido desarrollado en los lenguajes de programación C/C++, basándose en las librerías OpenCV en el entorno de desarrollo integrado en ROS.

1.5. Planificación del proyecto

En la planificación del proyecto, en primer lugar se realizó una estimación temporal de proyecto. Para ello, se elaboró una lista con todas las actividades a desarrollar, se estimó un tiempo para cada una de ellas y se establecieron las dependencias entre las mismas. Por otra parte, también se realizó una estimación de los recursos necesarios para la elaboración del proyecto y se asociaron a las distintas tareas que se habían planificado. Durante el desarrollo del proyecto, se ha realizado un seguimiento de la planificación realizada y se han ido corrigiendo las desviaciones que se iban produciendo.

1.5.1. Planificación Inicial Temporal

Tras detallar el proyecto y sus requisitos se procedió a descomponer todo el trabajo en distintas tareas. Posteriormente, se realizó una descomposición temporal de las actividades. En el diagrama de Gantt (figura 1.1), se muestra la planificación Inicial Temporal, dividida por dias para su mejor visualización (un día equivalente a 8 horas), en dónde se pueden observar las tareas que se planificaron realizar para la elaboración de este proyecto, así como su duración y dependencias.

Resumiendo esta tabla, primero vemos el nombre de tarea, la fecha inicial, la fecha final y la duración en días. Cuando se realizó la planificación inicial, la idea era la realizaciar la formación necesaria en los meses de noviembre, octubre y diciembre, sin contar los días libres y durante los siguientes meses empezar con la programación.

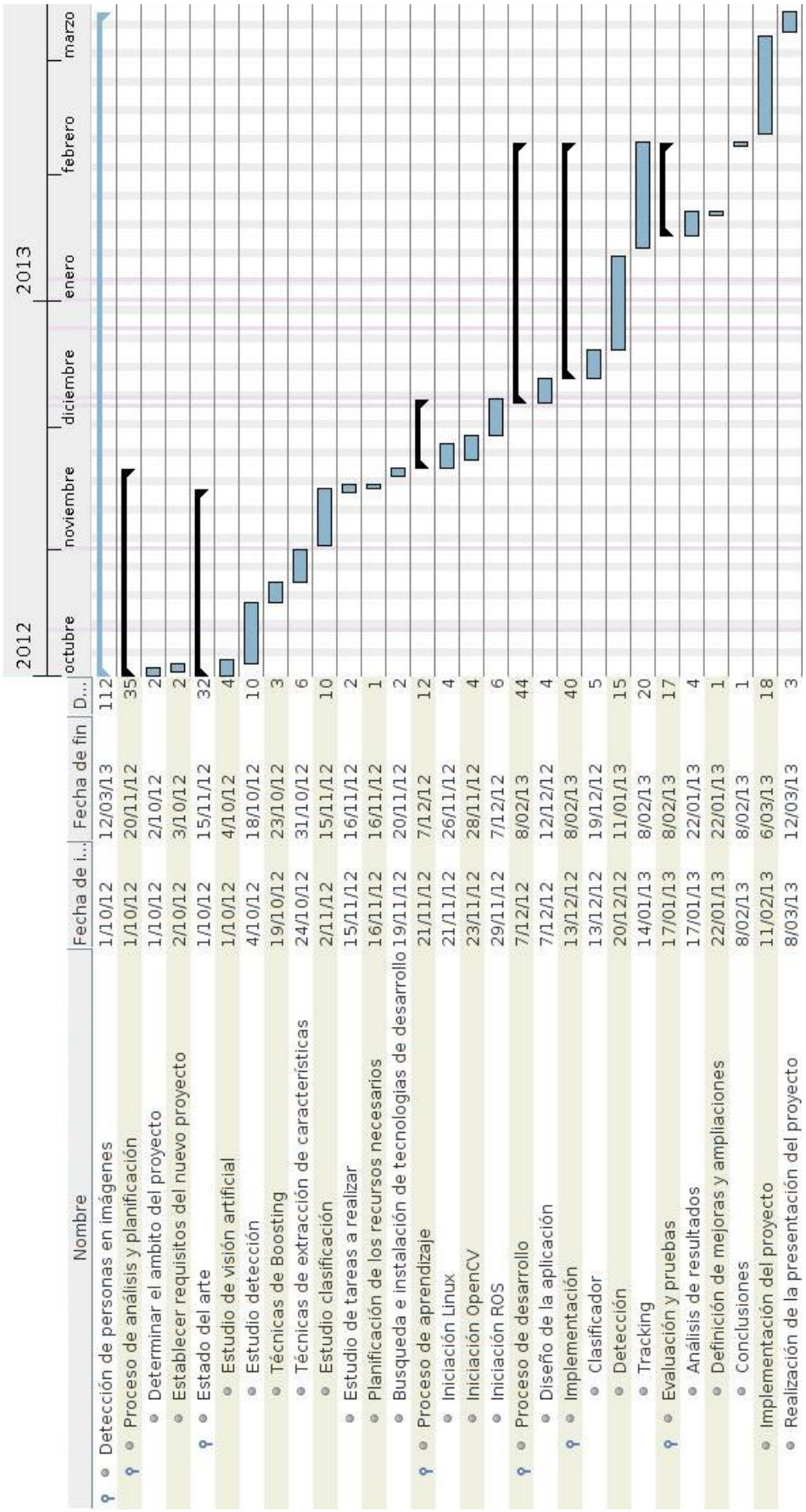


Figura 1.1: Planificación Inicial Temporal

1.5.2. Planificación Real Temporal

Además de la planificación temporal, se estudiaron también los recursos necesarios para llevar a cabo el proyecto. La planificación realizada fue una provechosa guía durante el desarrollo del proyecto. Sin embargo, surgieron dificultades que derivaron retrasos temporales, ya fuera por falta de experiencia en el lenguaje de desarrollo o por decisiones erroneas tomadas en la implementación. Todas las modificaciones sufridas por la planificación inicial se recogen en la siguiente sección 1.3.3, Desvío Temporal.

Los cambios se han acentuado sobre todo en la etapa de implementación, debido en gran parte a la necesidad de ampliar los conocimientos de la tecnología. Durante esta etapa, la tarea de implementación se ha visto alargada, debido a dificultades surgidas durante la programación de la aplicación (figura 1.2). Algunas de estas dificultades son, por ejemplo, la correcta utilización de la máquina de entrenamiento SVM^{light}.

Como se puede observar en el diagrama de Gantt (figura 1.2), se planificó que se le dedicaría gran parte del tiempo a la tarea de estado del arte y a la definición de mejoras y ampliaciones. Ello se debe a la importancia que supone partir de unas hipótesis coherentes para el posterior perfeccionamiento de la primera versión del algoritmo.

1.5.3. Desvío Temporal

Se empieza a trabajar en este PFC en Octubre de 2012 estableciendo los objetivos iniciales mencionados en la Sección 1.4 de esta memoria, así como las etapas necesarias para cumplir dichos objetivos las cuales se pueden apreciar en la figura 1.1.

Sin embargo esa planificación no ha podido cumplirse completamente, principalmente los tiempos pero tambien la fase de *Tracking* ha sido suprimida. Son varios los motivos que han llevado a que se haga este cambio, por un lado está el hecho de que los conocimientos previos en cuanto a programación en C++, pero además el uso de herramientas nunca antes utilizadas, y el desconocimiento de problemas para conseguir la detección, han contribuido a que la estimación preliminar del proyecto fuera muy complicada de cumplir en tiempo.

De esta forma el PFC planificado para realizarse en ocho meses, se ha convertido en un proyecto de una duración de un año. El desvío temporal es de 4 meses, el estudio de las tecnologías y los problemas personales que tuve en el primer cuatrimestre de este

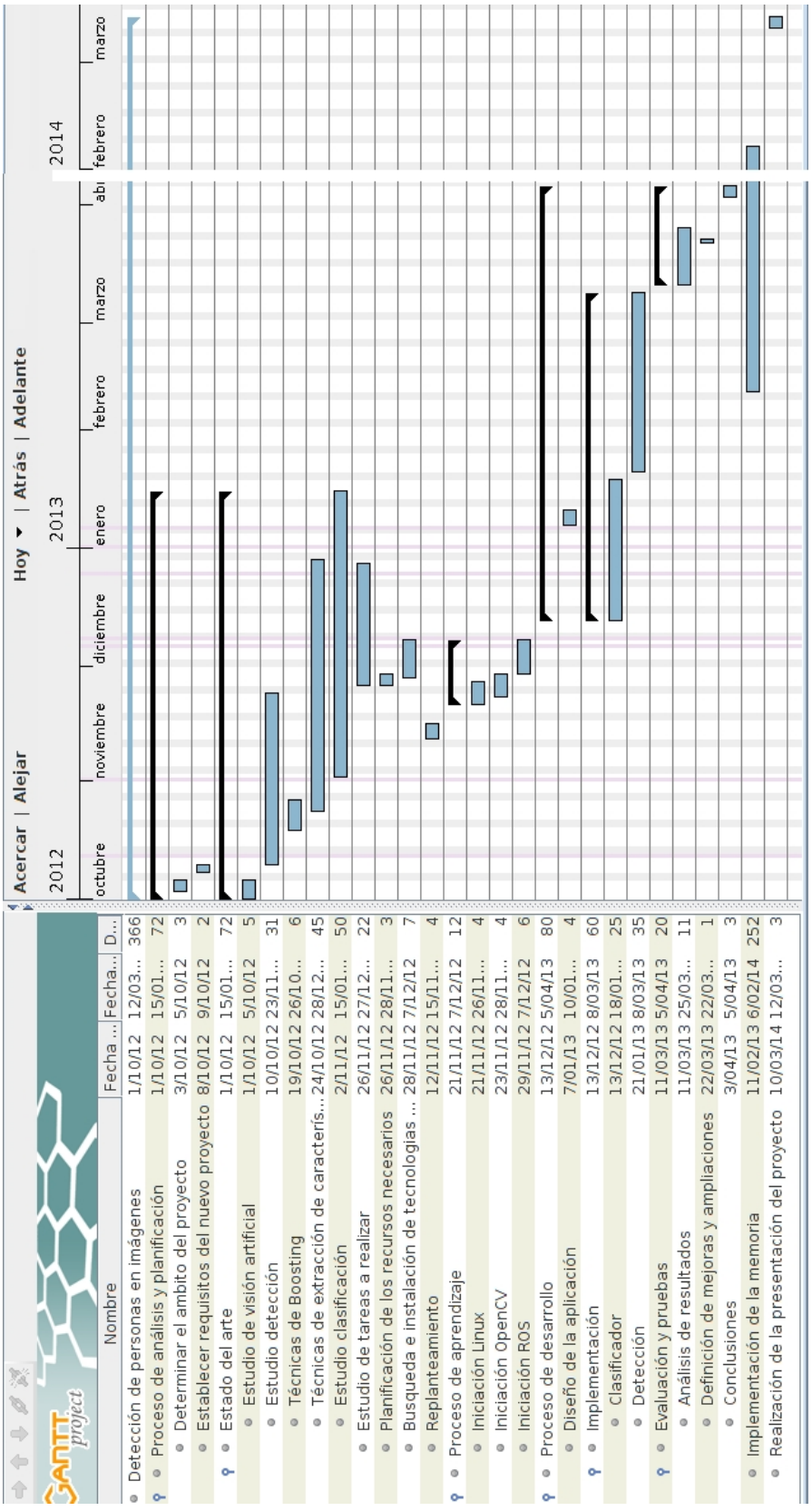


Figura 1.2: Planificación Real Temporal

curso cuando solo me quedaba terminar la memoria fue uno de los culpables de este contratiempo.

1.6. Recursos

Este capítulo nace con la finalidad de proporcionar al lector toda la información que necesita sobre las distintas herramientas y dispositivos necesarios para ejecutar la aplicación de detección que aquí se trata.

1.6.1. Tecnología

El proyecto ha sido desarrollado en Fatronic Tecnalia, en el departamento de Robótica, en la sección de Visión artificial. Se ha construido sobre el sistema operativo Ubuntu versión 12.04 LTS sobre un Intel® Core™ i5 CPU M 450 y con la tarjeta grafica Quadro FX570. Para adquirir las imágenes en el laboratorio se a utilizado la cámara Point Grey Chameleon CMLN-13S2C-CS. Las imágenes son de gran resolución, ocupan 3MB cada una. Los equipos del departamento de robótica de Fatronic tienen a día de hoy el sistema ROS instalado, para facilitar el tratamiento de datos en el ámbito de la robótica y la inteligencia artificial. La implementación se ha realizado en C++ haciendo uso de la librería OpenCV. Esta librería de Intel implementa un conjunto de algoritmos y utilidades orientadas al procesamiento de imagen, vídeo y visión por ordenador. Con ayuda de esta librería se ha implementado un algoritmo que hace uso de métodos basados en las técnicas planteadas. La programación, tanto del método de HOG como el de SVM, ha sido facilitada gracias al IDE Eclipse.

1.6.2. Software

Sistema operativo

La aplicación se desarrolla sobre el sistema operativo Ubuntu. Es un sistema operativo basado en Linux y que se distribuye como software libre y gratuito, el cual incluye su propio entorno de escritorio denominado Unity. Está orientado al usuario novel y medio, con un fuerte enfoque en la facilidad de uso y en mejorar la experiencia de usuario. Al igual que otras distribuciones se compone de múltiples paquetes de software normalmente distribuidos bajo una licencia libre o de código abierto.

En cuanto a las desventajas, instalar controladores de Hardware y programas resulta ser mas complicado que en Windows. Esto es debido a que las empresas creadoras de con-

troladores crean sus productos en base a Windows, el sistema operativo mas usado a nivel mundial. Aun así no tuvimos muchos problemas a la hora de utilizar la cámara *Point Grey*.

1. *Requisitos:*

   Los requisitos mínimos recomendados para Ubuntu son los siguientes:

   - Procesador: 1GHz x86
   - Memoria RAM: 512MB
   - Disco Duro: 5GB (para una instalación completa con swap incluida).
   - Tarjeta gráfica VGA y monitor capaz de soportar una resolución de 1024x768.
   - Lector de CD-ROM o tarjeta de red-7
   - Tarjeta de sonido.
   - Conexión a Internet.

   Cabe destacar que por lo general se puede ejecutar Ubuntu en hardware mas antiguo de lo especificado, aunque el rendimiento necesariamente va a ser menor.

2. *Instalación:*

   En caso de no tener instalado el Ubuntu es necesario seguir una serie de sencillos pasos, que están detallados en el anexo A.

Eclipse

La plataforma Eclipse, ofrece muchas de las características que cabría esperar de un IDE de calidad comercial: editor con sintaxis coloreada, compilación incremental, un depurador que tiene en cuenta los hilos a nivel fuente, un navegador de clases, un controlador de ficheros/proyectos, e interfaces para control estándar de código fuente.

Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado *Java Development Toolkit* (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como *BitTorrent* o *Azureus*.

Eclipse es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas.

En caso de no tener instalado el Eclipse es necesario seguir una serie de sencillos pasos, que estan detallados en el anexo A.

OpenCV

OpenCV es una biblioteca de software desarrollada en 1999 por la compañía Intel para el procesado de imágenes y el desarrollo de la visión por computador. Fue publicada bajo licencia *Berkeley Software Distribution* (BSD) y es multiplataforma, de forma que se puede utilizar en aplicaciones para GNU/Linux, MacOS X o Windows.

En la actualidad OpenCV es desarrollado siguiendo el modelo del software libre, pero está coordinado por el *Willow Garage* un grupo de desarrolladores y expertos en robótica.

Como meta pretende proporcionar un entorno de desarrollo fácil de utilizar y altamente eficiente. Esto se ha logrado, realizando su programación en código C y C++ optimizados, aprovechando además las capacidades que proveen los procesadores multi núcleo. OpenCV puede además utilizar el sistema de primitivas de rendimiento integradas de Intel, que es un conjunto de rutinas de bajo nivel específicas para procesadores Intel.

La versión utilizada en este PFC es la 2.4.3 publicada en Noviembre de 2012. En caso de no tener instalado el OpenCV es necesario seguir una serie de sencillos pasos, que están detallados en el anexo A.

SVM^{light}

En el proyecto se utiliza el software de SVM^{light} creado por Thorsten Joachims, versión 6.02 [13], con variantes para aprendizaje supervisado, y para semisupervisado transductivo. Es una herramienta de software libre para la ciencia.

Las características principales del software son los siguientes:

- El algoritmo de optimización rápida.
- Resuelve problemas de clasificación (por ejemplo, funciones de recuperación de aprendizaje en el motor de búsqueda Striver) y regresión.
- Calcula *XiAlpha* de las estimaciones de la tasa de error, la precisión, y la exhaustividad (*recall*).

- Ausencia de manera eficiente calcula-One-Out estimaciones de la tasa de error, la precisión, y la exhaustividad.
- Incluye algoritmo para la formación de unos grandes SVMs transductivos (TSVMs).
- Se puede entrenar a SVMs con los modelos de costos y gastos que dependen del ejemplo.
- Etc...

SVM^{light} consiste en un módulo de aprendizaje (*svm learn*) y un módulo de clasificación (*svm classify*).

Una vez descargado el archivo desde su pagina web [13], la instalación es sencilla, solo necesitamos seguir los pasos que se indican en el Anexo A.

CAPÍTULO 2

Estado del arte

2.1. Conceptos esenciales

En este capítulo se presenta al lector el área de investigación donde se enmarca este proyecto, dentro del amplio espectro de posibles áreas que engloban las ciencias de la computación. A continuación se lleva a cabo un breve recorrido por la historia de la visión por computador, para profundizar posteriormente en el análisis y procesamiento de imágenes y en el reconocimiento de objetos. Especialidad donde se enmarca exactamente el proyecto realizado.

2.1.1. Historia de la visión por computador

La visión artificial o visión por computador es una disciplina compleja que involucra otras ciencias e incluye estudios de física, matemáticas, ingeniería electrónica, ingeniería informática... El continuo desarrollo de algoritmos, funciones y aplicaciones hace que sea una disciplina en continua evolución. La visión artificial es un subcampo de la inteligencia artificial y su propósito es programar un computador para que "entienda" una escena o las características de una imagen.

Como es evidente, el diseño de un sistema de visión artificial por computador intenta simular lo que una persona humana capta por su sentido de la vista. Es decir, recono-

cimiento de figuras, objetos, distancia hasta ellos, textura que lo conforma y todas las características que un humano deduce de un objeto con solo verlo.

Los primeros conocimientos que se tienen de esta materia se remontan a los años veinte del siglo pasado, cuando se mejora la calidad de las imágenes digitalizadas de los periódicos, enviadas por cable submarino entre Londres y Nueva York. Sin embargo, no es hasta la década de los 50 cuando empiezan a aparecer los primeros trabajos relacionados con la visión artificial. Al principio se piensa que es una tarea sencilla y alcanzable en pocos años, esto se debe a los importantes trabajos realizados por Roberts en 1963 [14] y Wichman en 1968 [15]. El primero demuestra la posibilidad de procesar una imagen digitalizada para obtener una descripcion matemática de los objetos que aparecían y, el segundo presenta por primera vez una cámara de televisión conectada a un ordenador.

En la década de los ochenta vuelven a aparecer las investigaciones relacionadas con la visión por computador, en este caso encaminadas a la extracción de características. Así se tiene la detección de texturas [16] y la obtención de la forma a través de ellas [17]; y ese mismo año, 1981, se publican artículos sobre: visión estéreo (Mayhew y Frisby [18]), deteccion de movimiento (Horn [19]) y interpretación de formas (Steven); o los detectores de esquina (Kitechen y Rosendfekd, en 1982 [20]). A pesar de la importancia de las investigaciones y artículos recién comentados, el trabajo mas importante de la década es el libro de David Marr [21], donde se abordaba por primera vez una metodología completa del análisis de imágenes a través de ordenador.

2.1.2. Visión por computador

La visión, de una manera simple y resumida, consiste en capturar imágenes y procesar el contenido que hay en ellas para obtener información. Para un ordenador, la parte de captación de imágenes ya está hecha. Tan solo debemos utilizar el hardware adecuado para capturar imágenes (cámaras web, cámaras digitales, videocámaras,...) y, una vez obtenidas estas imágenes, debemos realizar la parte de procesamiento de imágenes, aunque esta fase es una ardua tarea. Con el procesamiento de imágenes, se puede establecer la relación entre el mundo tridimensional y las vistas bidimensionales tomadas de él. Se puede hacer por una parte, una reconstrucción del espacio tridimensional a partir de sus vistas y, por otra parte, llevar a cabo una simulación de una proyección de una escena tridimensional en la posición deseada a un plano bidimensional.

Para que el ordenador pueda procesar la información contenida en la imagen se debe

facilitar el entendimiento de la misma. Esto se realiza mediante transformaciones en la imagen de manera que la información que interese predomine en la imagen por encima de la información que no es relevante para la tarea.

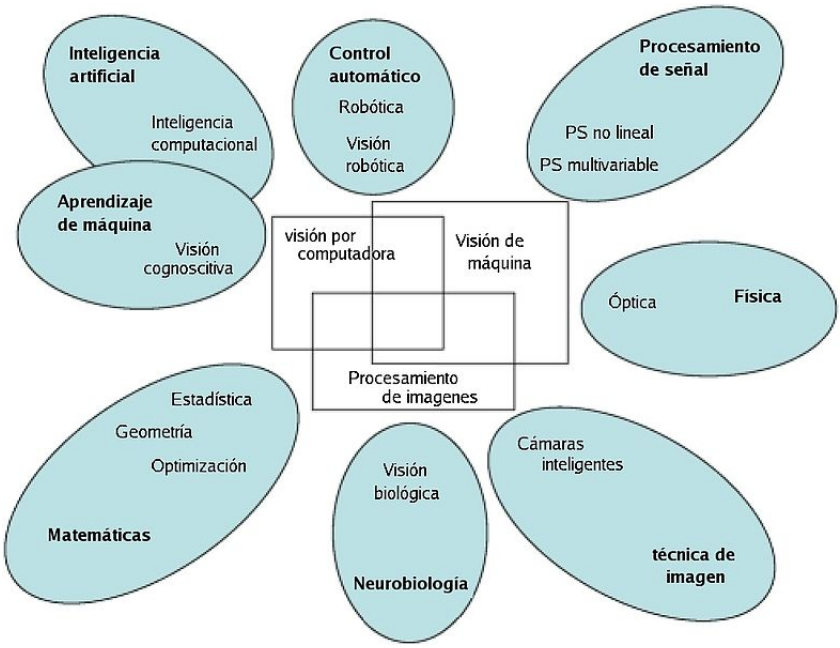


Figura 2.1: Relación de la visión por computador y otras áreas afines

El problema de la elección de un modelo como patrón, resulta en consecuencias frustrantes para cualquier línea de investigación. Hoy por hoy, comportamientos triviales para el humano no están todavía científicamente tipificados, y entre ellos el proceso de la visión humana. Por esta razón, la construcción de un sistema que emule el sistema visual humano es imposible. Aunque existe una enorme cantidad de publicaciones en neurofisiología, psicología y psicofísica, lo que se conoce del sistema de visión humano más allá del propio ojo es principalmente disjunto, especulativo y escaso.

Como podemos ver en la figura 2.1 la visión por computador involucra entre otras ciencias la inteligencia artificial, aprendizaje de máquina o la técnica de imagen. Los sensores son la fuente de datos del mundo exterior para cualquier sistema informático sofisticado, y entre éstos, las cámaras digitales son los sensores que mas información son capaces de proveer al sistema, dotándolo de lo que se conoce como visión artificial.

Este mundo es percibido a través de cámaras digitales, las cuales modelan el mundo real utilizando los principios de la geometría. Una característica de los principios fundamentales de los sistemas presentes y futuros se basa en la comprensión espacial del mundo que

percibe el robot. El dominio de la relación espacial, medición del espacio tridimensional, la propagación de la luz a través de un lente y el modelo matemático de formas y tamaños de objetos son los verdaderos fundamentos de esta disciplina.

La visión por computador actualmente comprende tanto la obtención como la caracterización e interpretación de las imágenes. Esto supone algoritmos de muy diversos tipos y complejidades.Como podemos ver en la figura 2.2, en un sistema de visión por computador actual se pueden distinguir seis etapas o fases:

- Captación: Es el proceso a través del cual se obtiene una imagen visual desde una cámara.
- Preprocesamiento: Conjunto de técnicas que facilitan el procesado pos terior. Incluye técnicas tales como la reducción de ruido y realce de detalles.
- Segmentación: Es el proceso que divide a una imagen en objetos que sean de nuestro interés.
- Descripción: Es el proceso mediante el cual se obtienen características convenientes para diferenciar un tipo de objeto de otro, por ejemplo tamaño y forma.
- Reconocimiento: Es el proceso que identifica a los objetos de una escena. Por ejemplo, las diferentes tipos de piezas en un tablero de ajedrez.
- Interpretación: Es el proceso que asocia un significado a un conjunto de objetos reconocidos.

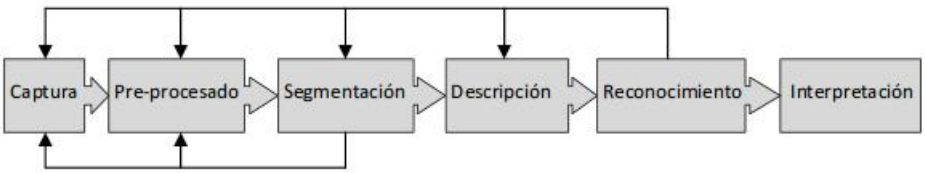


Figura 2.2: Fases de un sistema de visión por computador.

Esto supone distintos tipos de procesamiento en función del nivel en el que nos movamos:

- Visión de bajo nivel: Comprende la captación y el preprocesamiento. Ejecuta algoritmos típicamente de filtrado, restauración de la imagen, realce, extracción de contornos, etc.
- Visión de nivel intermedio: Comprende la segmentación, descripción y reconocimiento, con algoritmos típicamente de extracción de características, reconocimiento de formas y etiquetado de éstas.
- Visión de alto nivel: Comprende la fase de interpretación, normalmente es tos algoritmos se refieren a la interpretación de los datos generalmente mediante procedimientos típicos de la Inteligencia Artificial para acceso a bases de datos, búsquedas, razonamientos aproximados, etc.

2.2. Métodos de adquisición de información visual

La adquisición de imágenes puede hacerse por medio de cámaras convencionales, estéreo-cámaras, cámaras infrarrojos, estéreo-cámaras infrarrojos... Todas ellas existen en el mercado, y para cada una de ellas se puede plantear un tipo de detección diferente. Las cámaras infrarrojo son especialmente útiles para la detección en entornos poco iluminados y con condiciones meteorológicas adversas. Sin embargo, su elevado coste para su comercialización y su necesidad de recalibración periódica (normalmente cada año), especialmente cuando nos referimos a estéreo-cámaras, hacen difícil su inserción en el mercado. Por otro lado, las cámaras de estéreo-visión no presentan tales ventajas frente a condiciones de iluminación adversas, pero suponen una ventaja en cuanto a precisión de los mapas de profundidad y a la menor calibración requerida.

2.2.1. Cámaras 2D

La cámara monocular es el sensor más común, el más económico, para la solución de los problemas mencionados. Con este tipo de sensores se procesa la información en un espacio bidimensional, espacio imagen (proyección 2D de una escena 3D). Una de las ventajas que presentan este tipo de sensores, es que la información se puede trabajar en color o niveles de gris, éste último reduciendo la cantidad de información a procesar.

A continuación se muestran algunos trabajos relacionados con la detección y seguimiento de personas usando una cámara monocular.

En 1998, Schlegel et al. [22] en el Instituto de Investigación para el Procesamiento del Conocimiento Aplicado, de la Universidad de Ulm, en Alemania, publicaron un trabajo donde presentaban un sistema para detectar y seguir a una persona con el objetivo de mejorar la interfaz persona-máquina de un robot autónomo móvil. La etapa de inicialización para que el robot realice el seguimiento de una persona, consiste en situarse frente al robot, para que posteriormente el robot cree un modelo de la persona, el cual es una combinación de un enfoque basado en color y contornos.

En el año 2000, Mckenna et al. [23] presentaron un trabajo titulado "*Tracking Groups of People*", el cual consiste en un sistema de visión por computador para el seguimiento de múltiples personas. Para la detección de personas, se realiza un modelo del fondo, el cual combina los valores RGB de la imagen con los valores de cromaticidad con gradientes

de imagen local. Una vez creado este modelo, detectan las variaciones del fondo con la escena actual, y de esta forma segmentan las regiones de primer plano.

En el año 2005, B. Leibe et al. en el articulo "*Pedestrian detection in crowed scenes*" [24], se abordaba el problema de la detección de peatones en las escenas de la vida real llena de superposiciones entre las personas. Su método era la combinación de las señales locales y globales a través de una segmentación probabilística de *top-down*.

En el año 2006, Dalal et al. [25] desarrollaron un detector para peatones, probado sobre secuencias de vídeos. Usando la técnica de histogramas orientados de flujo óptico diferencial combinado con los descriptores de apariencia de Histogramas Orientados a Gradientes, detectan las regiones en movimiento y las segmentan del fondo de la escena. Con base a las proporciones se clasifican y segmentan humanos de otros objetos móviles.

En 2007, Ess et al. [26] combinaron las imágenes de una cámara con la información de profundidad, es decir, como entradas tienen un fotograma y su correspondiente mapa de profundidad. Su razonamiento es obtener una estimación inicial de la geometría de la escena, para así facilitar el análisis. Las estimaciones del mapa obtenido se pasan a una etapa de optimización global que controla las interacciones en un pixel.

En 2008, Spinello et al. [27], presentaron un nuevo método de detección y seguimiento de personas basado en un enfoque de fusión entre un láser 2D y una cámara digital (figura 2.3). Los puntos del láser se agruparon con un método gráfico utilizando AdaBoost en un clasificador de SVM. En la fase de detección los datos del láser se proyectan en las imágenes de la cámara para definir una región de interés. Para el tracking utilizan filtros de Kalman (*Extended Kalman Filters*) con tres modelos diferentes de movimiento.

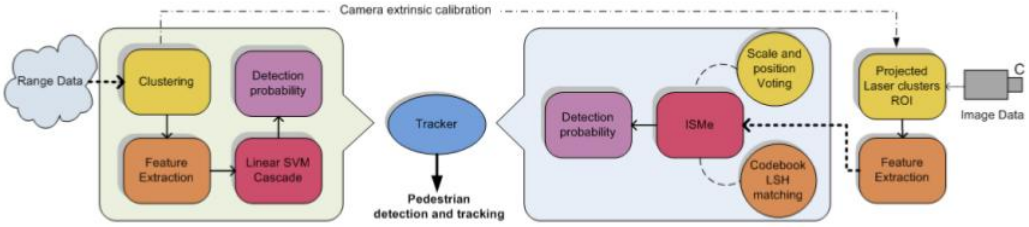


Figura 2.3: Método de detección presentado por Spinello.

2.2.2. Cámaras 3D

Aun que las cámaras estéreo son la forma más fiable de adquirir información 3D de una escena, existen otras alternativas mas económicas y versátiles, aun que menos precisas, como por ejemplo el dispositivo Microsoft Kinect.

El sensor Kinect es capaz de capturar el mundo que lo rodea en 3D mediante la combinación de la información obtenida de la cámara RGB y del sensor de profundidad (figura 2.4). El resultado de esta combinación es una imagen RGB-D (color + profundidad) con una resolución de 320x240, donde a cada uno de los píxeles se le asigna una información de color y de profundidad (figura 2.5).

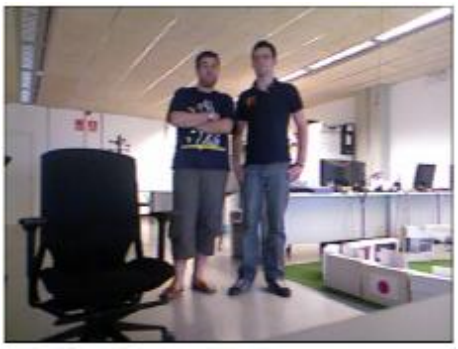


Figura 2.4: Sensor RGB (Kinect)

Figura 2.5: Sensor Depth (Kinect)

La cámara RGB que monta el dispositivo Kinect es una cámara digital estándar. La luz atraviesa una lente que la dirige a un filtro encargado de separarla en los colores primarios, los cuales son proyectados sobre un sensor fotosensible. Este sensor genera una señal eléctrica en función de la intensidad de la señal que incide sobre el. Posteriormente, esta señal es convertida a digital mediante un ADC (*Analog Digital Convert*), que mas tarde es analizada y reconstruida para su almacenamiento. Esto se consigue gracias a la interpolación, que permite rellenar aquellos espacios en los que falta información.

La aparición en el mercado de un sensor con estas prestaciones a un precio muy asequible ha disparado el número de implementaciones y aplicaciones que utilizan este sistema [28]. De entre todas ellas, es de destacar el middleware *PrimeSense NITE* (*Natural interaction*) [29], el cual permite comunicarse con los sensores de audio, vídeo y sensor de profundidad de kinect. A su vez, este middleware proporciona una API que facilita el desarrollo de aplicaciones que funcionen con interacción natural, por gestos y/o movimientos corporales. Además, haciendo uso del framework de OpenNI [30], se obtiene una primera descripción semántica del entorno a un coste computacional muy bajo.

2.3. Técnicas para la detección y clasificación

La detección de personas en secuencias reales resulta ser un reto debido a las variadas formas en las que se pueden encontrar las personas. Ahora veremos diferentes técnicas para la detección y clasificación.

2.3.1. Técnicas de extracción de características

Es el proceso de extraer características que puedan ser usadas en el proceso de clasificación de los datos. En ocasiones viene precedido por un preprocesado de la señal, necesario para corregir posibles deficiencias en los datos debido a errores del sensor, o bien para preparar los datos de cara a posteriores procesos en las etapas de extracción de características o clasificación.

Las características elementales están explícitamente presentes en los datos adquiridos y pueden ser pasados directamente a la etapa de clasificación. Las características de alto orden son derivadas de las elementales y son generadas por manipulaciones o transformaciones en los datos.

A continuación detallamos brevemente el funcionamiento de los mas importantes:

- **Descriptores Wavelet de Haar**

  Este método fue propuesto por Viola y Jones en 2004 [31]. Los descriptores Wavelet de Haar permiten definir de manera robusta clases de objetos complejos, siendo invariantes a cambios de color y de textura. Se emplean habitualmente para la descripción de personas. Presentan la capacidad de codificar rasgos tales como cambios de intensidades a diferentes escalas. La base de wavelet más sencilla es la de Haar que consiste en que se recorre la imagen con una ventana a la que se le aplican varios clasificadores en serie, cada uno más complejo que el anterior, los cuales usan las características para confirmar o descartar la hipótesis de que se trata del objeto buscado. Si la hipótesis se rechaza en cualquier nivel, el proceso no continúa pero si se confirma todos los filtros significará que se ha detectado el objeto deseado. Los patrones se consideran girados en varios posibles ángulos. Además, el algoritmo puede ejecutarse a varias escalas para obtener objetos de diferentes tamaños o de tamaño desconocido.

- **SIFT (Scale-invariant feature transform)**

SIFT es un método propuesto por David Lowe en 1999 [32], que se centra en buscar puntos característicos que cumplen criterios espacio-escalares. Los descriptores se calculan a través de la orientación de los gradientes de cada punto. Así se extraen puntos característicos invariantes y distintivos de una imagen que pueden ser usados para mejorar la correspondencia entre dos vistas diferentes de un objeto o una escena.

- **SURF (Speeded Up Robust Feature)**

  Es uno de los sucesores más importantes de SIFT, ha sido el algoritmo Speeded-Up Robust Features (SURF)[33]. SURF fue presentado en 2006 en el ECCV en Graz (Austria). Está parcialmente inspirado en SIFT y se ha demostrado que en la práctica totalidad de los casos consigue mejorar el rendimiento de este algoritmo [34]. Se basa en el cálculo del determinante de la matriz Hessiana (DoH: Determinant of Hessian) para la detección de puntos interesantes y en las wavelets de Haar para la descripción de dichos puntos. Esta aproximación es aún más rápida que DoG (la utilizada por SIFT) y ofrece una respuesta superior en cuanto a calidad de descripción de las imágenes.

- **Detectores de bordes Canny**

  Fue desarrollado por John F. Canny en 1986 que utiliza un algoritmo de múltiples etapas para detectar una amplia gama de bordes en imágenes [35]. Extraen los bordes de los objetos en las imágenes mediante la selección de aquellas regiones con altas derivadas espaciales. El hecho de tener en cuenta sólo los bordes de los elementos en la imagen reduce significativamente el tamaño de los datos a tratar, y filtra la información no útil de la imagen, conservando las formas, que es lo que proporciona la información relevante.

- **HOG**

  Este método fue presentado por Navneed Dalal y Bill Triggs en el Instituto Nacional de Investigación en Informática y Automática (INRIA), en 2005 [2]. Consiste en la división de la imagen en subbloques distribuidos a lo largo y ancho de la misma y con cierto solape entre ellos. Cada bloque se subdivide en subloques (o celdas) y sobre estos últimos se calcula la magnitud y orientación de los gradientes en cada píxel. Sobre cada uno de estos bloques se calcula el histograma de los gradientes orientados promediado por un peso gausiano, y luego se almacena en el vector de características de la imagen.

En este trabajo nos basaremos en los métodos HOG, dado su robustez frente a diferentes condiciones de iluminación, pequeños cambios en el contorno de la imagen, diferentes fondos y escalas, y dado que este metodo presenta buenas prestaciones según resultados previos de otros autores [2] [24] [4].

2.3.2. Clasificadores

Los clasificadores son algoritmos capaces de, aprender una cierta distribución de datos a partir de una serie de ejemplos de entrenamiento, para posteriormente poder predecir la clase a la que pertenecen nuevos ejemplos no utilizados en el entrenamiento.

Podemos encontrar varias ramas de clasificadores según su aprendizaje:

- **Aprendizaje supervisado**

  Es una técnica de aprendizaje artificial que elabora una función matemática a partir de datos de entrenamiento previamente etiquetados como por ejemplo SVM [36] o Adaboost [37].

- **Aprendizaje no supervisado**

  En este caso, el conjunto de entrenamiento no dispone de etiquetas conocidas, así que requiere de técnicas de agrupamiento que intenten construir estas etiquetas. Por ejemplo, Hidden Markov Models (HMM) [38], K-means [39] o self-organizing maps (SOM) [40].

- **Aprendizaje semi-supervisado**

  Es una combinación del aprendizaje supervisado y del no supervisado [41]. Surge de la dificultad que conlleva obtener los datos etiquetados requeridos en el aprendizaje supervisado. Por esa razón este método recurre al uso de una parte de datos etiquetados y un conjunto más extenso sin etiquetar mejorando, de este modo, la construcción de los modelos.

  Se asume que los datos no etiquetados siguen la misma distribución que los etiquetados puesto que, de no ser así, estos datos serían de poca utilidad. Por ejemplo, Co-training o re-weighting.

- **Aprendizaje por refuerzo**

  Este método no trata de aprender a partir de un conjunto de ejemplos si no a través de la experiencia. Por ejemplo Q-Learning [42].

El abanico de aplicaciones de los clasificadores es muy amplio, pero como existen bases de datos etiquetadas de personas como por ejemplo la de *INRIA* [43], hemos elegido el aprendizaje supervisado.

Aprendizaje supervisado

La detección se puede realizar mediante el aprendizaje automático de las diferentes vistas de un objeto extraídas de un conjunto de ejemplos empleados como plantillas, es decir, a partir de un conjunto de ejemplos de aprendizaje se genera una función que relaciona las entradas con las salidas de interés. Se considera muy importante la correcta selección de las características o descriptores de los objetos que se van a tomar para la clasificación. Uno de los inconvenientes de este método es que para obtener resultados precisos, se requiere de un gran conjunto de ejemplos para la realización del aprendizaje. Cabe destacar las siguientes técnicas dentro de este ámbito:

- **Adaptive Boosting**

  Las técnicas de Boosting son métodos iterativos que obtienen clasificadores muy precisos mediante la combinación de muchos clasificadores base no tan precisos. Estos clasificadores base se distribuyen en grupos y, a su vez, estos grupos denominados "etapas" se enlazan formando una cascada y, actuando cada uno sobre las predicciones del anterior, dando lugar al clasificador final (Paul Viola y Michael Jones) [44].

- **Algoritmo KNN**

  El algoritmo KNN (K nearest neighbors) [45] es un método de clasificación supervisada que sirve para estimar la probabilidad de que un elemento x pertenezca a la clase $C(j)$ a partir de la información proporcionada por el conjunto de prototipos, donde k determina el numero de vecinos que son contemplados para realizar la clasificación.

  En el reconocimiento de patrones, el algoritmo KNN es usado como método de clasificación de objetos basado en un entrenamiento mediante ejemplos cercanos en el espacio de los elementos.

- **Support Vector Machines (SVM)**

  Las Máquinas de Soporte Vectorial o Máquinas de Vectores de Soporte (SVM) son un conjunto de algoritmos de aprendizaje supervisado empleados para la clasifica-

ción y la regresión desarrollados por Vladimir Vapnik [46]. Dado un conjunto de ejemplos de entrenamiento (muestras) podemos etiquetar las clases y entrenar una SVM para construir un modelo que prediga la clase de una nueva muestra.

Tomando los datos de entrada como conjuntos de vectores en un espacio n- dimensional, una máquina de vectores soporte construirá un hiperplano de separación en ese espacio. Se considera que es mejor clasificador de datos aquel hiperplano que maximice la distancia (o margen) con los puntos que estén más cerca de él. Siendo los vectores de soporte los puntos que tocan el límite del margen. En el contexto que se está tratando en este proyecto de detección de personas, las clases de datos corresponderán al objeto (muestras positivas), mientras que el resto de la imagen será tachada como muestras negativas.

PARTE II: DESARROLLO TÉCNICO

CAPÍTULO 3

Desarrollo del proyecto

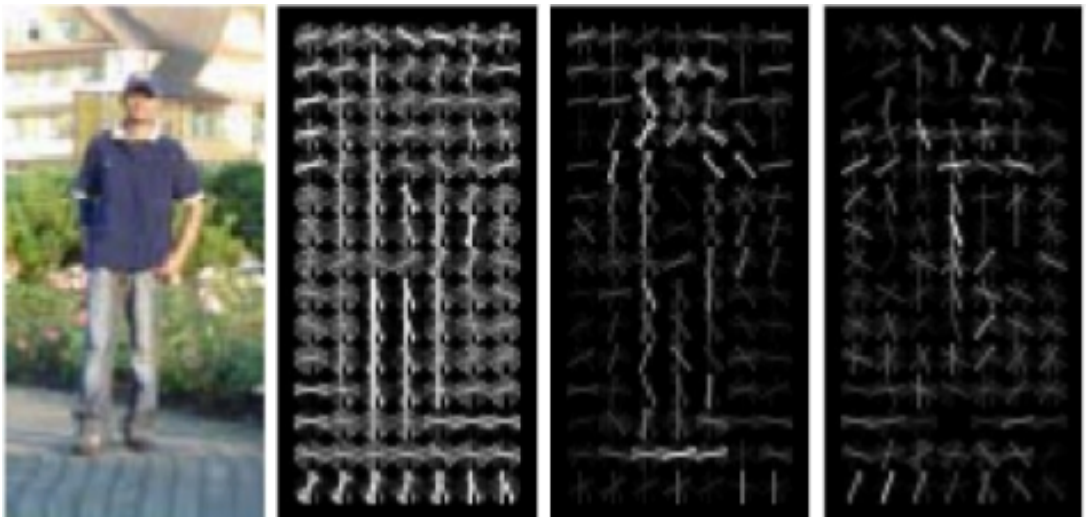
En el presente capítulo se van a introducir los conceptos teóricos a los que se ha recurrido durante el desarrollo del algoritmo y se explica cómo se pretende dar solución a todas las necesidades que se han planteado en el apartado 1.4.

En primer lugar se muestran unas bases teóricas para facilitar la comprensíon del lector, para finalizar analizando en detalle cada fase de nuestra aplicación, estudiando tanto la teoría de las técnicas existentes como su aplicación concreta.

3.1. Fundamentos teóricos

3.1.1. Descriptores basados en Histogramas de Gradientes Orientados

Los descriptores HOG (del inglés Histogram of Oriented Gradients-HOG) se basan en la orientación del gradiente en áreas locales de una imagen. La imagen se divide en pequeñas celdas cada una de las cuales acumula direcciones del histograma de gradiente u orientaciones de los bordes de los píxeles de las celdas. Se recomienda para una mejor respuesta normalizar el contraste en unas zonas más grandes (denominadas bloques) y utilizar dicho resultado para normalizar las celdas del bloque. Estos bloques de descriptores normalizados son lo que los autores denominan descriptores HOG (figura 3.1). Por último, se utilizan los descriptores HOG de la ventana de detección como entrada a un clasificador SVM^{light}.



(a) Imagen de prueba (b) Descriptores HOG (c) Descriptores HOG en positivo (d) Descriptores HOG en negativo

Figura 3.1: Ejemplo de la extracción de descriptores HOG

Los descriptores HOG nos proporciona información tal como los cambios de intensidad debido a los contornos o bordes de una imagen. Al tener en cuenta la relación con sus zonas vecinas y colindantes, es posible reconocer cuándo existe una frontera entre un objeto y otro. De esta manera, podremos identificar objetos de siluetas más suaves o más pronunciadas. El descriptor de HOG es, por lo tanto, especialmente adecuado para la

detección de personas, independientemente de su tamaño y sus colores, y fijándonos más en su relación con el entorno, distinguiendo los cambios más pronunciados.

Idea básica

Las bases teóricas de los métodos HOG residen en trabajos previos tales como Histogramas de bordes orientados (Freeman and Roth 1995, [47]), descriptores SIFT (Lowe 1999, [32]) y reconocimiento de formas (Belongie et al. 2001 [48]), entre otros. Sin embargo, la diferencia añadida que presentan los métodos HOG consiste en que los gradientes no se calculan uniformemente sobre un mallado denso, sino que se divide la imagen en bloques y, a su vez, cada bloque en diversos sub-bloques, y se calcula en cada uno de ellos los gradientes y el histograma.

El algoritmo HOG es capaz detectar la presencia de peatones presentes en una escena. Una vez detectados aquellos peatones, que por su cercanía al robot corren peligro de ser atropellados, el robot puede ser alertado de la presencia de éstos con la suficiente antelación como para poder reaccionar en el caso de que exista un riesgo. Es por ello que se insta a que el programa procese las imágenes tomadas con la mayor brevedad posible. El cálculo de los descriptores HOG no presenta un coste de tiempo de computación bastante elevado por el hecho de calcular el HOG en cada una de las celdas.

La elección de este método para llevar a cabo la detección de los peatones en la escena se basa en que destaca por su robustez frente a diferentes condiciones de iluminación, pequeños cambios en el contorno de la imagen y diferencias de fondos y de escalas. Los descriptores propuestos se basan en trabajos previos, tales como histogramas de bordes orientados, descriptores SIFT y reconocimiento de formas.

Dada una imagen en color, lo primero que se hace es transformar a escala de grises. A continuación se calculan los gradientes espaciales sobre toda la imagen. Posteriormente, se divide la imagen en bloques, solapados cierta área. El avance de bloques se realiza eliminando la columna de las celdas de la izquierda y añadiendo la columna de la derecha para el desplazamiento horizontal, mientras que para el vertical se elimina la fila de las celdas de arriba, añadiendo la fila de celdas de abajo. A su vez, cada bloque se divide en subregiones o celdas, calculándose en cada uno de ellos el histograma de los gradientes orientados, de tal forma que se logra mejorar el rendimiento.

Finalmente se aplica una ventana gausiana sobre cada bloque, almacenándose dicha información en el vector de características de la imagen.

El procedimiento para el cálculo de los descriptores de una imagen cualquiera de tamaño 64x128 puede verse en la figura 3.2.

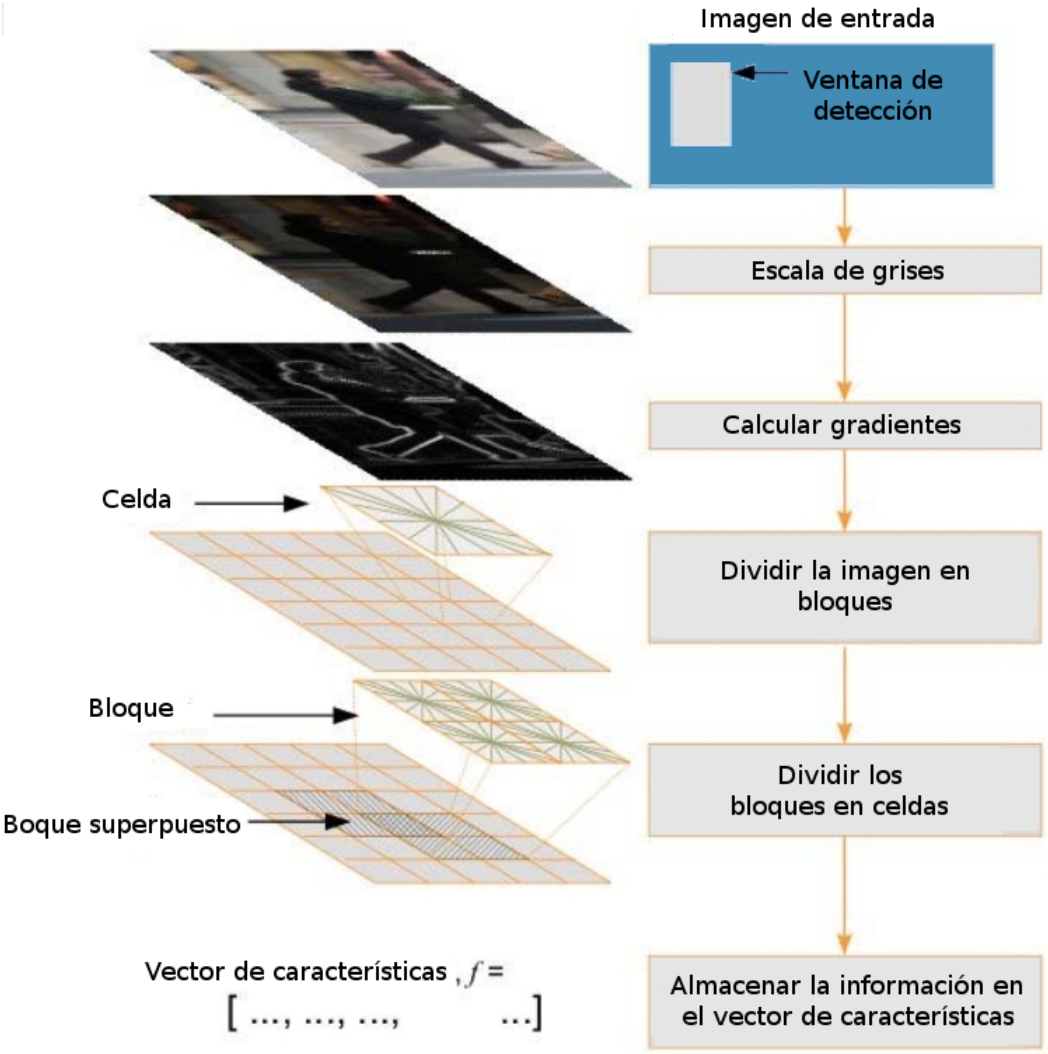


Figura 3.2: Proceso de extracción de características pàra una ventana de detección

Descripción matemática

El histograma de una imagen divide el rango de valores posibles de los píxeles de la imagen en una serie de sub-rangos o clases – de mismo o distinto tamaño entre ellos–(e.g. Dado el rango de valores de píxel [0,255], se realiza una división en ocho clases del mismo tamaño: [0,32), [32, 64),[64, 96)... [224, 255]), y almacena en cada clase la frecuencia de

píxeles con un valor comprendido entre ese sub-rango, es decir, el número de píxeles en la imagen cuyo valor está entre los valores de inicio y fin de cada sub-rango.

El Histograma de Gradientes Orientados de una imagen tiene como rango de valores posibles las distintas orientaciones que pueden tomar los gradientes de los píxeles, i.e. los distintos grados que pueden tomar sus ángulos de gradiente (e.g. [-90°, 90°], [0°, 180°], [0°, 360°]...). Este rango se divideen sub-clases –del mismo tamaño o distintos –(e.g. para el rango [0°, 180°], dividiendo en nueve sub-rangos: [0°, 20°), [20°, 40°)... [160°, 180°]), y se almacena en cada uno de ellas la suma de las magnitudes de gradiente de los píxeles cuyo ángulo de gradiente se encuentra comprendido entre esos valores.

Partiendo del concepto de los HOG, se puede obtener más información de una imagen por medio de un Descriptor de HOG (del inglés HOG Descriptor). En éste, la imagen se divide en un cierto número de sub-imágenes del mismo tamaño, denominadas celdas, y éstas se agrupan en bloques con un mismo número de celdas de ancho y alto todos ellos. Además, estos bloques se encuentran solapados de forma que el avance de bloques horizontalmente se realiza eliminando la columna de celdas de la izquierda y añadiendo la columna de la derecha y, verticalmente, eliminando la fila de celdas de arriba y añadiendo la fila de celdas de abajo.

De este modo, dados una imagen A de tamaño $W \times H$; un tamaño de celda $C_W \times C_H$ con $W mod C_W = 0$ y $H mod C_H = 0$; y un tamaño de bloque en celdas $B_W \times B_H$; el el ancho y alto de la imagen en celdas, W_C y H_C, y el número de bloques distribuidos horizontalmente y verticalmente, N_{BW} y N_{BH}, se calculan de la siguiente manera:

$$W_C = \frac{W}{C_W}, \qquad H_C = \frac{H}{C_H} \tag{3.1}$$

$$N_{BW} = 1 + W_C - B_W, \qquad N_{BH} = 1 + H_C - B_H \tag{3.2}$$

Y por tanto, el número total de celdas N_C y el número total de bloques N_B resultantes de la imagen A será igual a:

$$N_C = W_C * H_C \tag{3.3}$$

$$N_B = N_{BW} * N_{BH} \tag{3.4}$$

Y la distribución de celdas (C) y bloques (B) es la siguiente:

$$A = \begin{pmatrix} a_{00} & a_{10} & \dots & a_{(W-1)0} \\ a_{01} & a_{11} & \dots & a_{(W-1)1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{0(H-1)} & a_{1(H-1)} & \dots & a_{(W-1)(H-1)} \end{pmatrix}$$

$$C = \begin{pmatrix} c_{00} & c_{10} & \dots & c_{(W_C-1)0} \\ c_{01} & c_{11} & \dots & c_{(W_C-1)1} \\ \vdots & \vdots & \ddots & \vdots \\ c_{0(H_C-1)} & c_{1(H_C-1)} & \dots & c_{(W_C-1)(H_C-1)} \end{pmatrix}$$

$$B = \begin{pmatrix} b_{00} & b_{10} & \dots & b_{(N_BW-1)0} \\ b_{01} & b_{11} & \dots & b_{(N_BW-1)1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{0(B_H-1)} & b_{1(B_H-1)} & \dots & b_{(N_BW-1)(N_BH-1)} \end{pmatrix}$$

A partir de esta estructuración de la imagen, el descriptor de HOG calcula de forma independiente el HOG de cada celda y cada bloque agrupa los HOGs de sus celdas correspondientes. Entonces, el número de HOGs que contiene un descriptor será:

$$N_{HOG} = B_W * B_H * N_B \tag{3.5}$$

Y si se divide el HOG en n clases, dado que cada bloque contiene $B_W * B_H$ descriptores HOG, entonces el número total de valores N_V que se tomará de la imagen A será:

$$N_V = n * N_{HOG} \tag{3.6}$$

Para el cálculo del modelo de detección, sobre la colección de imágenes se debe calcular el descriptor de HOG de cada imagen, etiquetando cada descriptor como positivo si es una imagen de persona (“+1”) o negativo si no lo es (“-1”).

El cálculo de los descriptores de HOG puede tener un coste en tiempo de computación bastante grande pues requiere del cálculo de un HOG por cada celda. Para agilizar esto se puede usar una técnica denominada Integral de HOG por el cual se mejora el coste de

computación a cambio de aumentar el coste en memoria. Para calcular la Integral de HOG se necesitan primero n matrices auxiliares de tamaño igual a la imagen de entrada $A_{W\times H}$, y donde n es el número de clases en el HOG. Cada una de estas matrices estará asociada a una única clase, es decir, a un único rango de ángulos de gradiente, y donde cada uno de sus píxeles cumple:

$$M(b,i,j) = \begin{cases} |G(i,j)|, & \theta(i,j) \in R(b) \\ 0, & eoc \end{cases}$$

donde $b \in \{0,1,...,n-1\}$ indica la clase $i \in \{0,1,...,W-1\}$ indica la columna de la matriz, $j \in \{0,1,...,H-1\}$ indica la fila de la matriz y $R(b)$ es el rango de valores de ángulo asociado a la clase b. En la figura 3.3 se puede ver una representación gráfica de estas matrices.

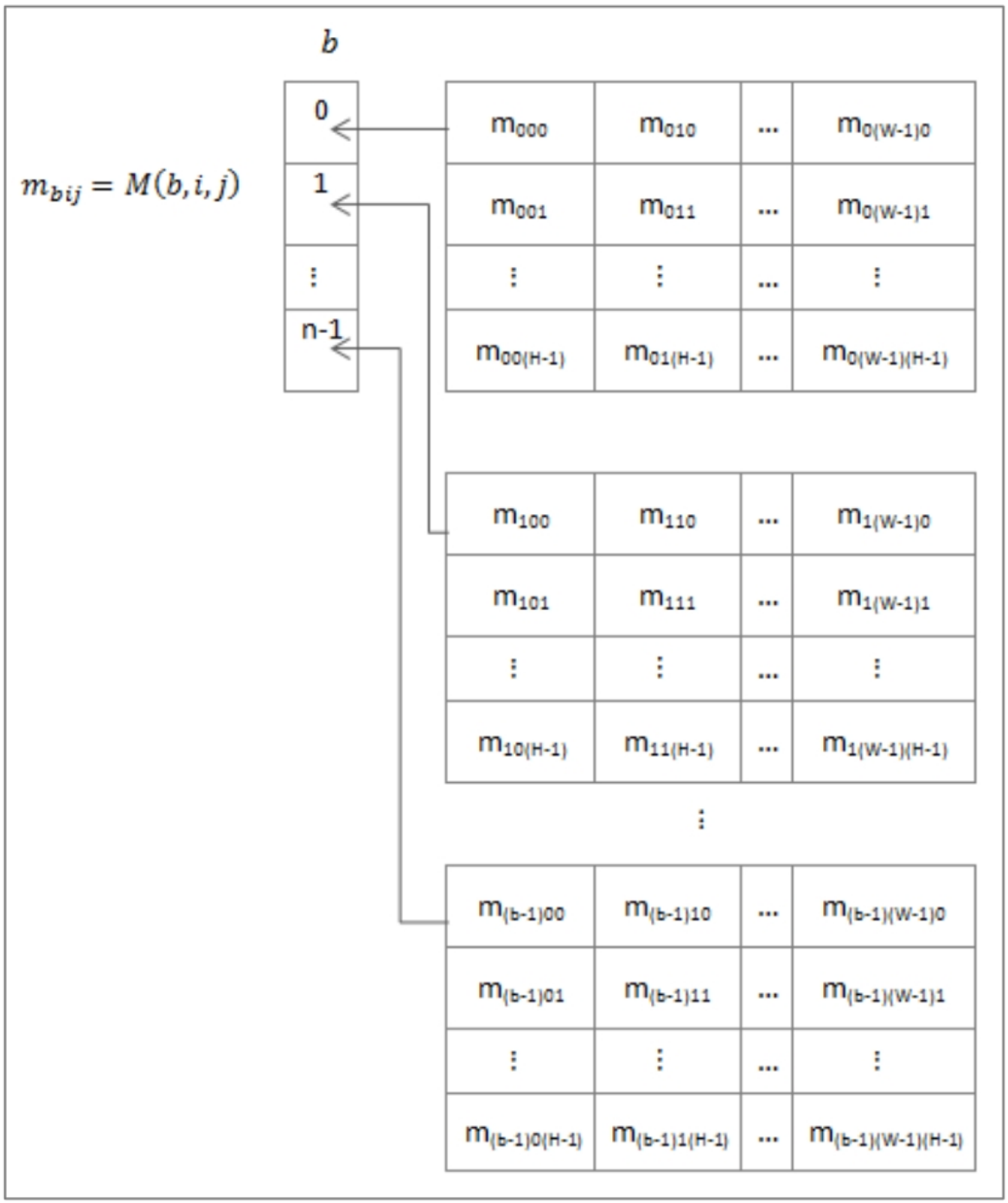


Figura 3.3: Matrices auxiliares para el cálculo de una Integral de HOG

La Integral de HOG (figura 3.4) está compuesta de matrices de n tamaño $(W+1)\times(H+1)$, cada una de ellas asociadas a una clase y donde cada píxel (i,j) indica la suma de las magnitudes del rectángulo de la imagen A con esquina superior izquierda (0,0) y de tamaño $i\times j$. Éstas matrices se calculan a partir de las matrices M anteriores y se puede definir una Integral de HOG (IH), como la siguiente función recursiva:

$$IH(b,i,j) = \begin{cases} 0, & i=0, \quad j=0 \\ IH(b,i,j-1) + \sum_{k=0}^{i} B(b,k,j), & eoc \end{cases}$$

De esta forma, para el cálculo del HOG de cada celda de un Descriptor de HOG se deberá realizar una simple resta por cada clase por medio de la siguiente ecuación:

$$C(b,i,j,W_C,H_C) = IH(b,i+W_C,j+H_C) - IH(b,i+W_C,j) - IH(b,i,j+H_C) + IH(b,i,j)$$

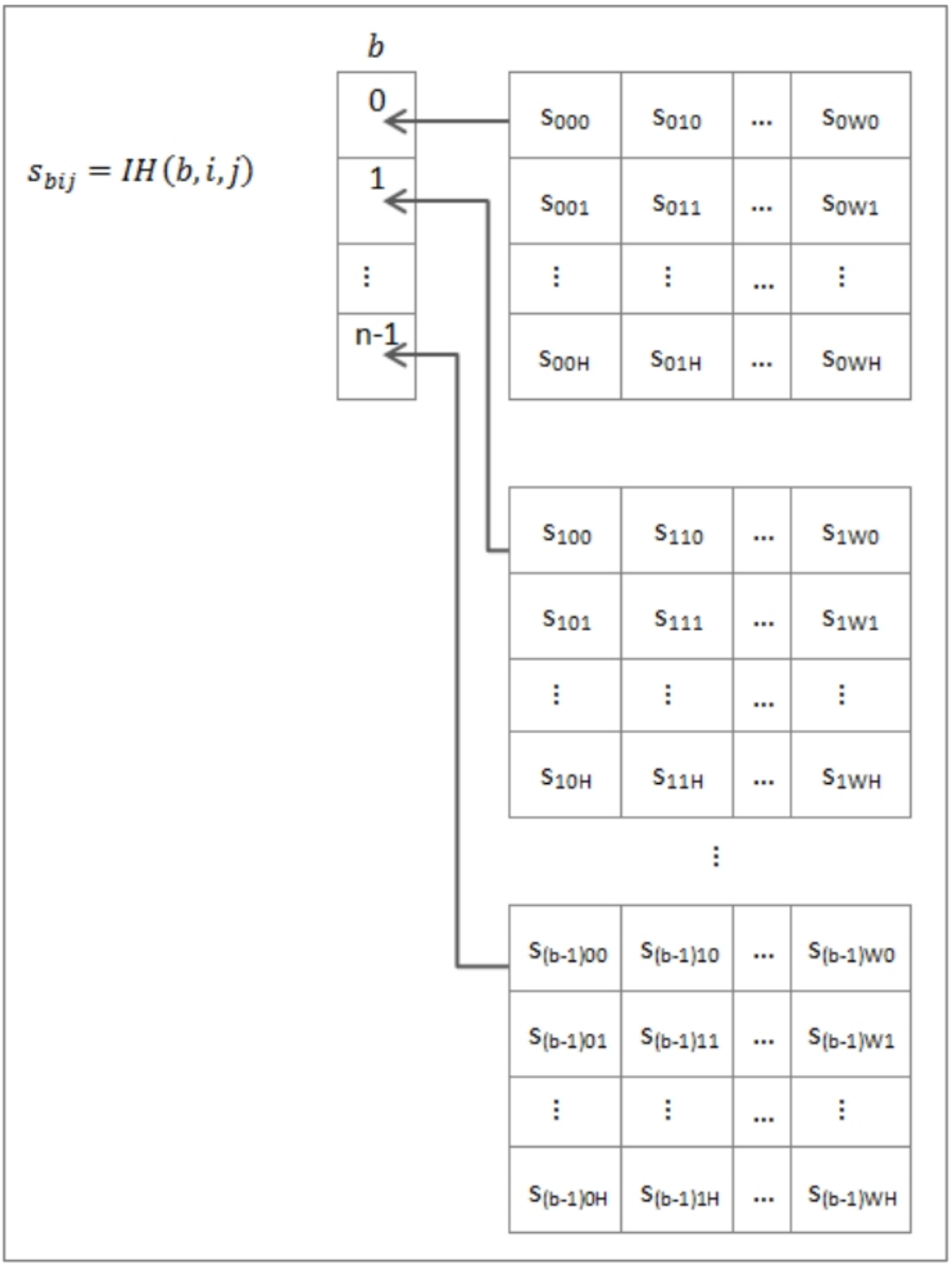


Figura 3.4: Integral de HOG

3.1.2. Máquinas de soporte vectorial (SVM)

Las Máquinas de Vectores Soporte son estructuras de aprendizaje basadas en la teoría estadística del aprendizaje. Se basan en transformar el espacio de entrada en otro de dimensión superior (infinita) en el que el problema puede ser resuelto mediante un hiperplano óptimo (de máximo margen).

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

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

Idea básica

Las Máquinas de Soporte Vectorial o Máquinas de Vectores de Soporte (SVM) son un conjunto de algoritmos de aprendizaje supervisado empleados para la clasificación y la regresión. Dado un conjunto de ejemplos de entrenamiento (muestras) podemos etiquetar las clases y entrenar una SVM para construir un modelo que prediga la clase de una nueva muestra.

Tomando los datos de entrada como conjuntos de vectores en un espacio n-dimensional, una máquina de vectores soporte construirá un hiperplano de separación en ese espacio (figura 3.5). Se considera que es mejor clasificador de datos aquel hiperplano que maximice la distancia (o margen) con los puntos que estén más cerca de él. Siendo los vectores de soporte los puntos que tocan el límite del margen. En el contexto que se está tratando en este proyecto de detección de personas, las clases de datos corresponderán al humano (muestras positivas), mientras que el resto de la imagen será tachada como muestras negativas. La SVM busca un hiperplano que separe de forma óptima a los puntos de una clase de la de otra, que eventualmente han podido ser previamente proyectados a un espacio de dimensionalidad superior.

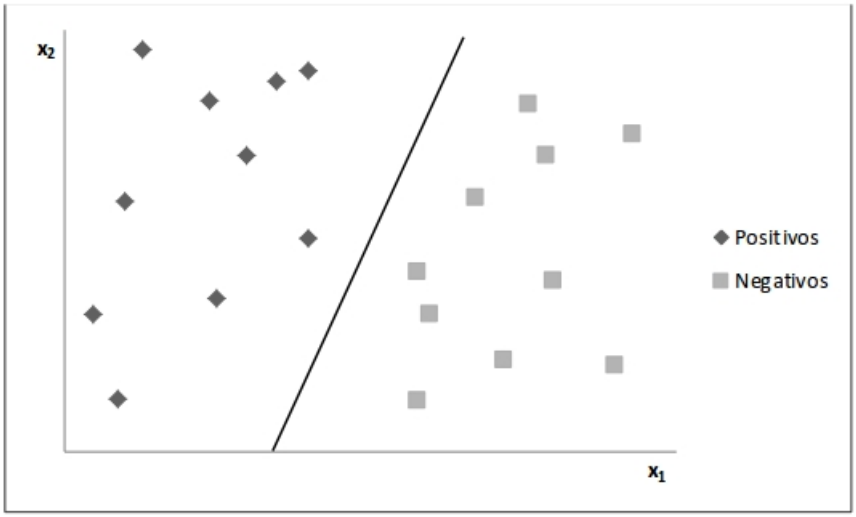


Figura 3.5: Hiperplano de separación de dos clases

En ese concepto de "separación óptima" es donde reside la característica fundamental de las SVM: este tipo de algoritmos buscan el hiperplano que tenga la máxima distancia (margen) con los puntos que estén más cerca de él mismo. Por eso, también a veces se les conoce a las SVM como clasificadores de margen máximo. De esta forma, los puntos del vector que son etiquetados con una categoría estarán a un lado del hiperplano y los casos que se encuentren en la otra categoría estarán al otro lado.

Para separar linealmente los datos se procede a realizar un cambio de espacio mediante una función que transforme los datos de manera que se puedan separar linealmente. Esta función recibe el nombre de Kernel.

En este caso, los conjuntos son "personas" y "no personas". Para ello, se necesita un entrenamiento previo de la máquina, facilitándole ejemplos de personas o "positivos" y ejemplos de no-personas o "negativos". Con todos los ejemplos de entrenamiento, el algoritmo de clasificación SVM elabora una curva M-dimensional que divide ambos conjuntos, obteniendo de esta forma el kernel de la máquina. Las dimensiones del espacio dependen del número de componentes de cada vector a clasificar.

Descripción matemática

La formulación de las máquinas de vectores se basa en el principio de minimización estructural del riesgo, que ha demostrado ser superior al principio de minimización del riesgo empírico. Las maquinas de vectores soporte presentan un buen rendimiento al generalizar en problemas de clasificación, pese a no incorporar conocimiento específico sobre el dominio. La solución no depende de la estructura del planteamiento del problema.

La idea es construir una función clasificadora que:

- Minimice el error en la separación de los objetos dados. Error en clasificación.
- Maximice el margen de separación (mejora la generalización del clasificador).

Consideremos el conjunto de entrenamiento:

$$(\bar{x}_i, z_i) : \bar{x}_i \in \Re^m, i = 1, .., N \tag{3.7}$$

Supongamos que existe un hiperplano que separa los puntos de ambas clases. Los puntos **x** sobre el hiperplano satisfacen $w^T x + b = 0$ donde el vector **w** es normal al hiperplano, |b|/||w|| es la distancia perpendicular del hiperplano al origen y ||w|| es la norma euclídea de w.

Sea d_+ (d-) la distancia más corta del hiperplano de separación a la muestra positiva (negativa, respectivamente) más cercana. Definamos el margen del hiperplano (figura 3.6) como la suma $d_+ + d$. En el caso linealmente separable, el algoritmo buscará simplemente el hiperplano con mayor margen. A continuación formularemos esta idea.

Supongamos que todos los datos de entrenamiento satisfacen las siguientes desigualdades:

$$w^T x_i + b > +1 \qquad para \quad z_i = +1 \tag{3.8a}$$

$$w^T x_i + b < -1 para \quad z_i = +1 \tag{3.8b}$$

esto es,

$$z_i(w^T x_i + b) - 1 \geq 0 \qquad \forall i \tag{3.8c}$$

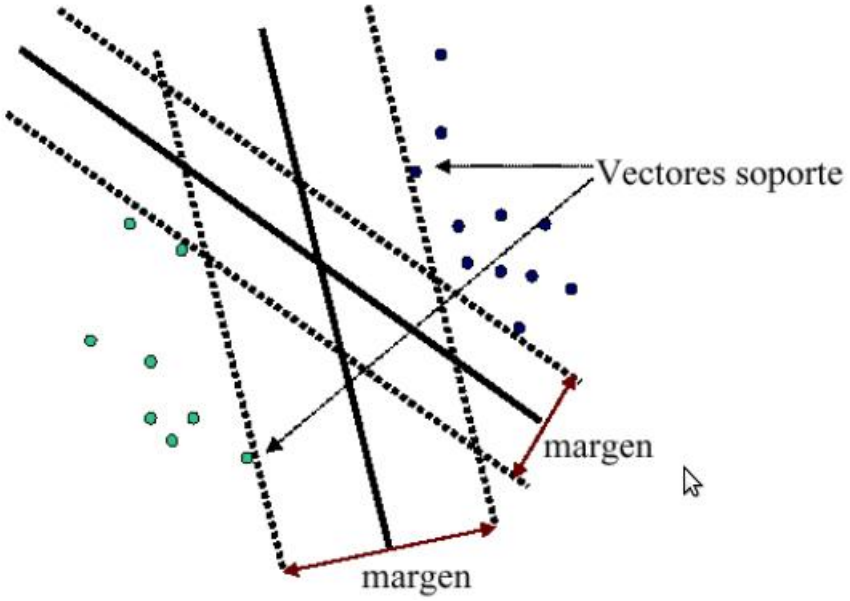


Figura 3.6: Distintas posibilidades para obtener el margen del hiperplano

Ahora consideramos los puntos para los que se cumple la igualdad en (3.8a) (que este punto exista es equivalente a elegir una escala adecuada para **w** y **b**). Estos puntos están sobre el hiperplano $H_1 : w^T x_i + b = 1$ con normal **w** y distancia al origen |1-b|/||**w**||.

De forma similar, para el hiperplano H_2 la distancia al origen es |-1-b|/||w||. Por lo tanto, $d_+ = d = 1/||W||$ y el margen es simplemente 2/||w||. Nótese que H_1 y H_2 son paralelos (tienen la misma normal) y que no hay puntos de entrenamiento entre ellos (figura 3.7). Podemos, en definitiva, encontrar el par de hiperplanos que dan el máximo margen minimizando la función de coste $\frac{1}{1}||w||^2$ con las restricciones (3.8c).

Ahora pasaremos a una formulación lagrangiana del problema. Hay dos razones importantes para hacer esto:

La primera es que las restricciones de la ecuación (3.8c) se sustituirán por restricciones sobre multiplicadores de Lagrange, que serán más fáciles de manejar.

La segunda es que con esta reformulación del problema, los datos del entrenamiento solo aparecen en forma de productos escalares entre vectores. Esta propiedad es crucial para generalizar el procedimiento al caso no lineal como veremos.

Por lo tanto, introduzcamos N multiplicadores de Lagrange que denotaremos por α_1, α_2, ..., α_n uno para cada una de las restricciones de (3.8c). La regla en aplicaciones lagrangianas es que para restricciones de la forma $\varsigma_i \geq 0$ las ecuaciones que las definen se mul-

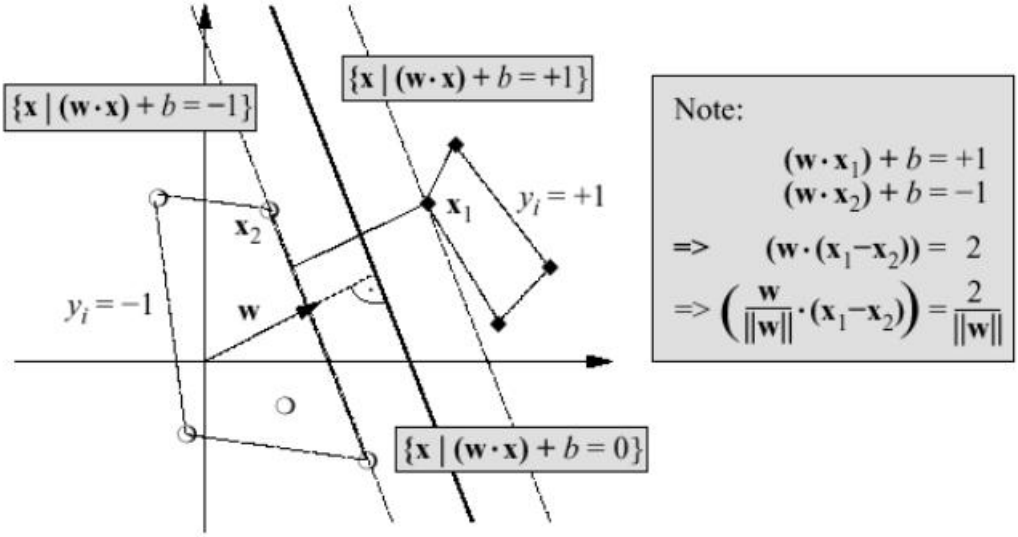


Figura 3.7: Ilustración de la idea de hiperplano de separación óptimo para el caso de patrones linealmente separables. Los vectores soporte(aquellos que yacen sobre H_1, H_2 y cuya eliminación cambiaría la solución encontrada) se muestran rodeados por un circulo.

tiplican por multiplicadores de Lagrange positivos y se restan de la función objetivo para formar el lagrangiano. En el caso de restricciones de la forma $\varsigma_i = 0$, los multiplicadores de Lagrange no tienen restricciones. Lo anterior da el lagrangiano:

$$L_p = \frac{1}{2}||\bar{w}||^2 - \sum_{n=1}^{N} \alpha_i z_i (\bar{w}^T \bar{x}_i + b) + \sum_{n=1}^{N} \alpha_i \tag{3.9}$$

Ahora debemos minimizar L_p con respecto a **w**, **b** y a la vez exigir que las derivadas de L_p con respecto a todos los α_i se anulen, todo sujeto a las restricciones $\alpha_i \geq 0$ (restricciones ς_1). Esto quiere decir que podemos resolver de forma equivalente el siguiente problema *dual*: maximizar L_p sujeto a la restricción de que el gradiente de L_p con respecto a **w** y **b** se anule, y sujeto también a la restricción de que $\alpha_i \geq 0$ (restricciones ς_2). Esta formulación particular del problema se conoce como *dual de Wolfe* y presenta la probabilidad de que el máximo de L_p con las restricciones ς_2 ocurre en los mismos valores de **w**, **b** y α que el mínimo L_p de con las restricciones ς_1.

Al requerir que se anule el gradiente de L_p con respecto a **w** y **b**, obtenemos las condiciones:

$$\frac{\partial L_p}{\partial \bar{w}} = 0 \Longrightarrow \bar{w} = \sum_{i=1}^{N} \alpha_i z_i \bar{x}_i \tag{3.10a}$$

$$\frac{\partial L_p}{\partial b} = 0 \Longrightarrow \sum_{i=1}^{N} \alpha_i z_i = 0 \tag{3.10b}$$

Ya que estas restricciones aparecen igualdades, podemos sustituirlas en la ecuación

$$p(x) = \frac{1}{m} \sum_{y_i=1+1} K(x, x_i) \tag{3.11}$$

para obtener:

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j z_i z_j \bar{x_i} \bar{x_j} \tag{3.12}$$

La solución se obtiene minimizando L_P o maximizando L_D. Se trata, en definitiva, de un problema de programación cuadrática (QP).

Nótese que hay un multiplicador de Lagrange α_i para cada muestra de entrenamiento. Tras obtener una solución, aquellos puntos para los que $\alpha_i > 0$ se denominan *vectores soporte* y yacen sobre los hiperplanos H_1, H_2. El resto de las muestras tienen $\alpha_i = 0$, por ello, el vector **w** se escribirá como combinación de los vectores soporte.

Con estas máquinas, los vectores soporte son los elementos críticos del conjunto entrenamiento: son los más cercanos a la frontera de decisión y si el resto de puntos no se consideran en un nuevo entrenamiento, el algoritmo encontraría el mismo hiperplano de separación. Los patrones que conformarán el clasificador son los vectores soporte, el resto de patrones de entrenamiento son irrelevantes a los efectos de clasificación.

Podemos observar cómo **w** está determinado explícitamente por el algoritmo de entrenamiento, pero no es este el caso del umbral **b**, aunque su obtención es inmediata.

Una vez que hayamos entrenado una máquina de vectores soporte (SVM), para clasificar un patrón de evaluación **x** basta determinar en qué parte de la frontera de decisión se encuentra y asignarle la etiqueta de la clase correspondiente, es decir, asignamos a **x** la clase $sgnw^T x + b$ donde *sgn* es la función signo.

SVM^{light}

El proceso de clasificación mediante una Máquina de Soporte Vectorial consta de dos pasos: entrenamiento y clasificación, donde en el primero se reconocen los patrones del conjunto de datos de entrenamiento con el fin de crear un modelo que luego será empleado en la clasificación de nuevos datos. Este proceso presenta complejidad de orden cuadrático respecto a las dimensiones de los datos de entrenamiento por lo que los problemas que se pueden solucionar con esta técnica se ven limitados. Actualmente existen tres algoritmos fundamentales [49] para el entrenamiento de SVM en software: Chunking [50], Sequential Minimum Optimization (SMO) [51] y SVM^{light} [52]. Este último es una mejora propuesta al algoritmo planteado en el trabajo de Osuna, *"Improved Training Algorithm for Support Vector Machines"* [53]. El algoritmo tiene requisitos de memoria escalable, resuelve problemas de clasificación y regresión y por lo tanto, es el mas adecuado para la detección de personas.

SVM^{light} [13], es una implementación de SVM en C, con variantes para aprendizaje supervisado, y para semisupervisado transductivo. Hemos utilizado esta implementación ya que se puede utilizar su código para investigación.

Las principales características del programa son los siguientes:

- Algoritmo de optimización rápida.
- Resuelve problemas de clasificación y regresión.
- Calcula *XiAlpha* de las estimaciones de la tasa de error, *precision* y *recall*.
- Incluye algoritmo para la formación de unos grandes SVMs transductivos (TSVMs).
- Se puede entrenar a SVMs con los modelos de costos y gastos que dependen del ejemplo.
- Maneja hasta diez mil ejemplos de entrenamiento.
- Maneja varios miles de vectores de soporte.
- Soporta funciones de núcleo estándar.
- Usa representación por vector disperso.
- ...

Este algoritmo es usado para el problema de reconocimiento de patrones. El algoritmo tiene requerimientos de memoria escalable y puede manejar problemas con muchos miles de vectores soporte eficientemente. El código ha sido usado sobre un largo rango de problemas, incluyendo categorización de texto, reconocimiento de tareas y aplicaciones medicas [54], [55], [56].

3.2. Desarrollo del algoritmo

En esta sección explicaremos la metodología que hemos utilizado para nuestra aplicación. Empezaremos por explicar el programa esquemáticamente para luego profundizar más, dando detalles de cada apartado.

3.2.1. Metodología empleada

En esta primera fase se lleva a cabo la extracción de aquellos obstáculos que se encuentran en la vía urbana a estudio y que, por su cercanía al vehículo, resultan de interés por poder tratarse de peatones.

Para la técnica de detección de los elementos localizados a una determinada distancia podría bastar el empleo de sensores con capacidad para medir estas distancias y que alerten al conductor en el caso de que esta distancia sea reducida. Sin embargo, la utilización de un sistema formado por una cámara proporciona una descripción del entorno muy completa.

Ello da lugar a que se pueda distinguir el tipo de elemento del que se trata y, sobre todo, la principal ventaja que presenta es que permite anticiparse a los movimientos, en este caso del peatón. No obstante, este método también presenta inconvenientes, ya que el procesamiento de toda la información resulta complejo, dando lugar a un alto coste computacional.

La detección de los peatones en las distintas imágenes se va a realizar, mediante la implementación del algoritmo de Dalal y Triggs (HOG+SVM) [2]. Sin embargo, dicho algoritmo presenta un alto coste computacional, por lo que se busca reducirlo, implementándolo únicamente en aquellas regiones de interés que puedan contener a uno o varios peatones. Se van a llevar a cabo una serie de descartes y clasificaciones, con el objetivo de que el número de píxeles procesados sea el menor posible, sin que ello afecte a la eficacia de la detección de los peatones.

En primer lugar, se van a descartar aquellos objetos localizados demasiado lejos del robot, pues no suponen un riesgo de accidente. Por tanto, no se van a tener en cuenta aquellos píxeles que presenten una disparidad inferior a una prefijada. Los obstáculos que se localicen en el mapa de obstáculos dentro del margen de búsqueda van a ser consideradas objetos de la escena. No obstante, el tamaño de algunas de ellas va a ser tan reducido que

va a ser imposible que se trate de un peatón. Esto significa que se podrá hacer un nuevo descarte según el tamaño de dichos obstáculos.

El resto de las posibles personas localizadas van a ser englobadas por rectángulos, dando lugar a las conocidas Regiones de Interés (ROIs), con las que se va a trabajar durante el resto del proyecto.

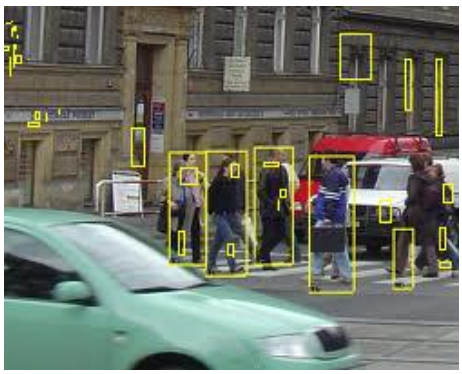


Figura 3.8: ROIs obtenidas, sin descartar aquéllas de areas mas pequéñas.

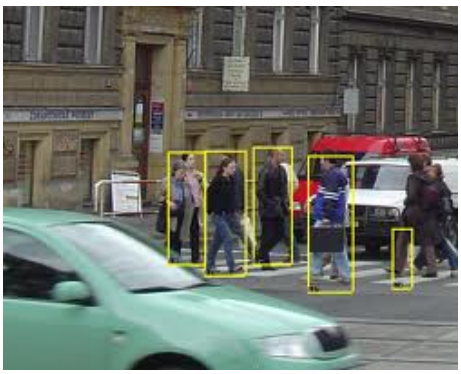


Figura 3.9: Descartando las ROIs que no pueden ser peatones.

En la figura (3.9) se muestra este descarte, pues en la figura (3.8) se observa como muchas de las regiones representadas son demasiado pequeñas como para englobar a un peatón, por lo que se decide dejar de tenerlas en consideración.

A continuación se detallan los pasos seguidos para la detección.

El detector HOG es bastante simple de entender. Una de las principales razones de esto es que utiliza una función de "global"para describir a una persona en lugar de un conjunto de características "locales". En pocas palabras, esto significa que la totalidad de la persona está representada por un único vector de características, a diferencia de muchos vectores de características que representan partes más pequeñas de la persona.

Debido a la arbitrariedad que existe en la distribución de la luminosidad en las imágenes, hemos considerado que en el cálculo de los gradientes podríamos prescindir de la información acerca del sentido del gradiente. Esto es así porque en detección de peatones lo que nos interesa saber no es el sentido sino su dirección, debido a la arbitrariedad en las prendas que puedan llevar los peatones y en los diferentes objetos e iluminaciones que puedan darse en el fondo de la imagen. Nos interesa conocer la existencia de un cambio de luminosidad y su distribución, pero no en qué dirección. El detector utiliza una ventana de detección de deslizamiento que se mueve alrededor de la imagen. En cada posición de la ventana del detector, se calcula un descriptor de HOG para la ventana de detección.

Este descriptor se muestra a continuación, a la SVM entrenado, lo que lo clasifica como “persona“ o “no es una persona“.

Por último, haremos un estudio de la posición óptima de estos puntos, partiendo de ideas previas basadas y la propia intuición sobre el problema. Nos basaremos en solapes entre bloques de valor 0, ½ y ¾. Asimismo, también analizaremos la viabilidad de eliminar los puntos más cercanos al borde de la imagen, ya que a parte de no aportar información relevante, suponen un mayor coste temporal en su procesado.

3.2.2. Obtención de características

El objetivo de esta etapa es diferenciar los píxeles de la imagen que pertenecen al contorno del objeto de estudio (en nuestro caso, personas) del resto de píxeles de la imagen. Posteriormente, a partir de él se obtiene una estructura que describen puntos y características relevantes de la persona.

Es necesario antes de calcular las características una fase previa de adaptación de la imagen de entrada:

- La imagen de la base de datos codificada en tres canales RGB, es convertida en una imagen de un solo canal de escala de grises para mejorar el rendimiento y simplificar los cálculos sucesivos.
- Cambiar el tamaño de la imagen a 64x128.
- Sobre esta imagen se realiza una ecualización del histograma con lo cual se maximiza el contraste. Esta técnica aumenta la probabilidad de encontrar el contorno correctamente.

Tras estos pasos previos se comienza con la fase de obtención de características.

Adaptación de las imagenes de las Bases de datos

La imagen a color de tres canales con codificación RGB capturada por la cámara (figura 3.10) es transformada en una imagen en escala de grises de un solo canal (figura 3.11). Para conseguirlo se calcula la media ponderada de los tres canales BGR y el resultado es el valor del canal en escala de grises. Es decir, para cada píxel se aplica:

$$RGB[A]toGray : Y \leftarrow 0{,}299 * R + 0{,}587 * G + 0{,}114 * B$$

El beneficio de realizar esta conversión consiste en reducir el número de píxeles que se tienen que computar, ya que se ha pasado de una imagen con tres canales a una imagen de un solo canal. Esta simplificación se puede realizar gracias al hecho de que, para realizar el cálculo de bordes, no son relevantes los valores concretos de intensidad de cada canal de color primario (Red, Green,Blue), sino más bien la variación de intensidad global que se produce.

Figura 3.10: Imagen de la base de datos *INRIA* antes de la conversión.

Figura 3.11: Conversión de la imagen RGB a escala de grises.

Histogramas de gradientes

Una vez adaptada la imagen, hay que meter las características en un vector de vectores para luego poder entrenar la SVM (de cada imagen se calcula un vector de características). Para calcular el descriptor HOG, se utilizan celdas de 8×8 píxeles dentro de la ventana de detección. Estas celdas se organizarán en bloques superpuestos.

En la figura (3.12) se pude ver una versión ampliada de una de las imágenes, con una celda de 8×8 dibujado en rojo, para que hacernos una idea del tamaño de la celda y la resolución de la imagen en la que se trabaja.

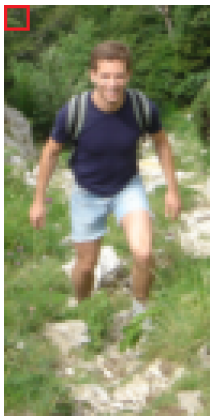


Figura 3.12: Ilustración de las celdas 8x8.

Supongamos que queremos calcular el vector gradiente en el píxel resaltado en rojo en la figura (3.13).

Ésta es una imagen en escala de grises, por lo que los valores de los píxeles sólo están entre 0 y 255 (0 es negro, 255 es blanco). Los valores de los píxeles a la izquierda y

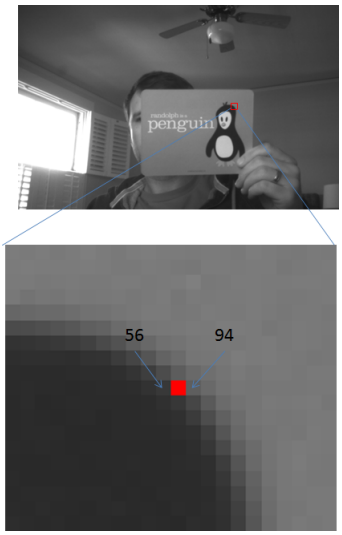


Figura 3.13: Ejemplo vector de gradiente.

derecha de nuestro pixel están marcados en la imagen: 56 y 94 (figura 3.13). Si se cogen los valores de derecha a izquierda, el ritmo de cambio en la dirección X es de 38 (94-56 = 38). Se puede calcular el gradiente sustrayendo de izquierda a derecha o al revés, sólo hay que ser coherente en toda la imagen.

Podemos hacer lo mismo para los píxeles de encima y de debajo (figura 3.14) para conseguir el cambio en la dirección Y (93 – 55 = 38):

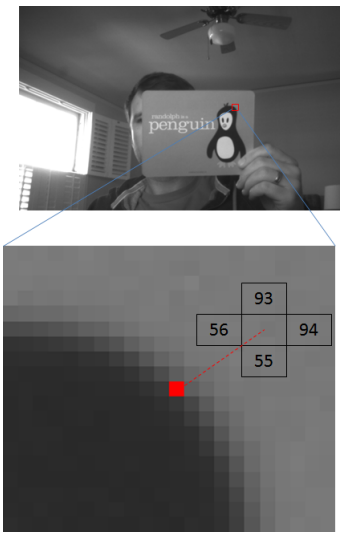


Figura 3.14: Vector de gradiente X Y.

Poniendo estos dos valores juntos, tenemos nuestro vector gradiente.

$$\begin{pmatrix} 38 \\ 38 \end{pmatrix} \tag{3.13}$$

Asi, se hallan tanto la magnitud (3.14) como la dirección de los valores de gradiente (3.15) en cada uno de los píxeles de la región. El cálculo de la derivadas espaciales de la imagen a lo largo de los ejes X e Y $(I_x I_Y)$

$$|G| = \sqrt{I_x^2} + \sqrt{I_y^2} \tag{3.14}$$

$$\Theta = arctan(\frac{I_x}{I_y}) \tag{3.15}$$

Por lo tanto, cada pixel de la celda tiene un cierto peso en el histograma de orientación, basado en el valor calculado de la magnitud de su gradiente. Cada imagen se representa a través del histograma de cada una de las celdas. Estos histogramas quedan ordenados en un vector según su peso, dando lugar al vector de características de la imagen.

3.2.3. Entrenamiento de la SVM

Para poder entrenar este clasificador se requiere un conjunto de ejemplos positivos y otro negativo. Con ello, como se ha explicado previamente, la máquina sitúa estos puntos en un espacio N-dimensional y traza una curva que separa ambos conjuntos. En función de la precisión con la que la curva separa ambos conjuntos, y de la calidad de las muestras de los mismos (es decir, si las muestras tienen características relevantes), obtendremos unos resultados u otros. Para evaluar las prestaciones de la máquina, necesitamos un conjunto muestral de test, sobre el que calcular unos parámetros de comparación, que se detallan en el siguiente punto.

Adecuación de la base de datos

El Data-set es una base de datos desarrollada específicamente en este proyecto. Consta de diferentes tipos imágenes y archivos de diferentes objetos, con las cuales se enseña y prueba nuestro sistema.

A partir del conjunto muestral obtenido de las bases de datos *INRIA* [22] y *MIT Pedestrian* [21], donde agrupamos los ejemplos positivos por un lado y los negativos por otro, vamos a definir una metodología para proceder al entrenamiento y testeo de los diferentes kernels. Para ello, decidimos utilizar un 90 % de las muestras de cada conjunto para realizar el entrenamiento, y reservar un 10 % para el testeo del clasificador. Así, eliminamos dependencias de cualquier tipo que pudiera ocasionar la selección de un único conjunto de test.

Para cada escenario se entrena el clasificador y posteriormente se testea introduciéndole los ejemplos de test, los cuales conocemos a priori. Una vez construido el data-set y elegidos los dos subconjuntos para cada fase, se procede al entrenamiento del sistema que, dependiendo del método que se utilice en el módulo, se realizará de una forma u otra. Una característica común a todos los métodos es que adquieren la información de entrenamiento siempre de la base de datos. La máquina clasifica estos ejemplos y predice unas etiquetas para cada muestra. De estos resultados se obtienen unos parámetros de comparación que evaluan las prestaciones de la máquina. Finalmente, para cada parámetro se hará la media de cada escenario, obteniéndose al final unos valores promediados y, por tanto, menos dependientes del conjunto seleccionado para el test.

Adecuación del Kernel

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

Hay que tener en cuenta que son muchas las aplicaciones en las que este tipo de máquinas toman parte, con lo que no hay a priori una inclinación por un método u otro.

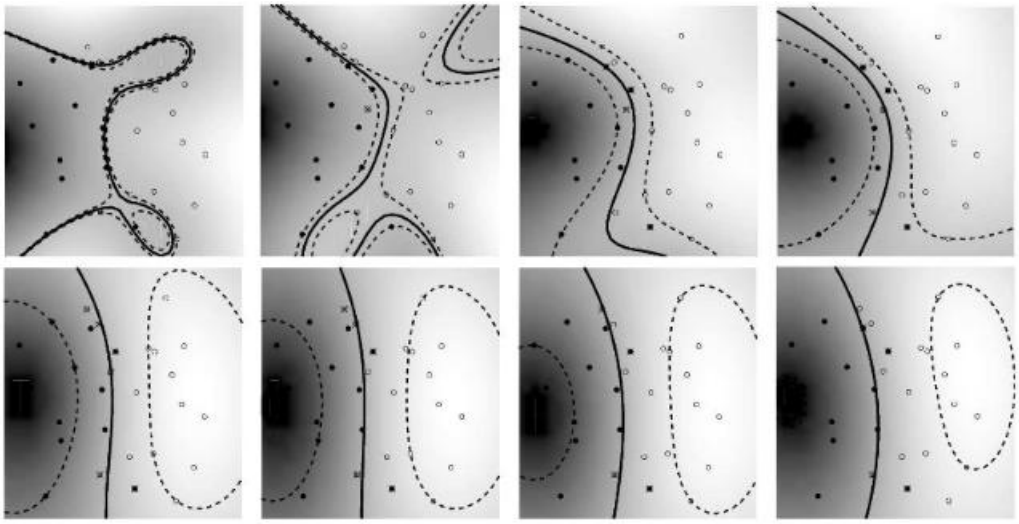


Figura 3.15: Kernel gausiano para varios valores del parámetro gamma.Los valores de este parámetro van de gamma=0.1 (arriba izquierda) a gamma=0.8 (abajo derecha).

En la figura 3.15 se muestra un ejemplo de funcionamiento de un kernel gausiano. Cuanto más alto sea el valor gamma, más puntos permitimos que se encuentren en la zona situada entre ambos conjuntos.

Una vez trazada la línea que divide ambos espacios M-dimensionales, cuando queramos clasificar una nueva muestra no identificada deberemos introducirla en la máquina y representarla como un punto en este espacio. La distancia euclídea de la muestra a la curva trazada en la etapa de entrenamiento de la máquina resultará ser la medida que nos da la predicción sobre la muestra. El signo de esta magnitud nos informa que la muestra pertenece a "personas" o "no personas". Un signo positivo nos indicará que la muestra clasificada pertenece al primer grupo. Análogamente, un signo negativo nos indicará que la muestra pertenece al segundo grupo. Así mismo, el módulo de la misma nos indica la

probabilidad de que la muestra esté en un conjunto u otro. Así por ejemplo, valores de 3.2 indicarán que la muestra pertenece al espacio de "personas" con mayor probabilidad que si la predicción hubiera sido de 0.2. En el segundo caso, la máquina no predice con tanta certeza la pertenencia de la muestra a un conjunto u otro.

Proceso de realimentación

Una vez que se ha obtenido el primer modelo de detección se debe testear con un conjunto de imágenes de prueba para comprobar su eficiencia. Si una imagen positiva no logró ser detectada, es decir un falso negativo, ésta deberá ser añadida como imagen positiva a la colección de imágenes de muestra a partir de la cual se calculó el detector. En cambio, si una imagen negativa fue detectada, es decir un falso negativo, ésta deberá ser añadida a la colección como imagen negativa. Cuando una imagen positiva es detectada o una negativa no es detectada, entonces el funcionamiento será correcto y no se deberá hacer nada con esas imágenes.

Tras realizar esto con todas las imágenes de testeo, se volverá a usar el SVM con nuevas imágenes añadidas para calcular un nuevo modelo y se repetirá el test.

Parámetros de SVM*light*

- **Opciones de aprendizaje**:
  - z {c,r,p} - clasificación (c), de regresión (r), y orden de preferencia (p)
  - c float - C: trade-off entre el error de entrenamiento y el margen.
  - w [0..] - anchura épsilon para la regresión.
  - j float - Costo: el costo de factores, por lo que los errores de capacitación.
  - b [0,1] - hiperplano sesgado o hiperplano imparcial.
  - i [0,1] - eliminar ejemplos inconsistentes de entrenamiento y reciclar.
- **Opciones de estimación de rendimiento**:
  - x [0,1] - calcular las estimaciones de la leave-one-out.
  - o [0..2] - valor de rho para estimar XiAlpha y para la poda leave-one-out.
  - k [0..100] - búsqueda de profundidad extendida a XiAlpha-estimator.

- **Opciones de transducción**:
  - p [0..1] - fracción de ejemplos no etiquetados para ser clasificados en la clase positiva (por defecto es la relación de ejemplos positivos y negativos en los datos de entrenamiento)
- **Opciones de kernel**:
  - t int - tipo de función kernel.
  - d int - parámetro d para kernel polinomial.
  - g float - parámetro gamma.

Parámetros de comparación

Atendiendo a otros trabajos, los parámetros más comúnmente utilizados para analizar las prestaciones de cada kernel son los siguientes: Accuracy, Precision, Recall, FP, FPPW, MR, y se definen como sigue:

- **Precision** (Proporción de True Positives para las detecciones.): $Precision(\%) = \frac{TP}{TP+FP}$
- **Accuracy** (Proporción de detecciones correctas, tanto True Positives como True Negatives): $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
- **Recall** (Relacionado con el grado de pérdida de candidatos):$Recall(\%) = 100 - MR * 100$
- **FPPW** (Falsas Positivas por Ventana): $FPPW = \frac{FP}{Ntestneg}$
- **MR** (Miss Rate). Indica el grado de pérdida de candidatos: $MR = \frac{FN}{Ntestpos}$
-
  - **TP**: 'True Positives' Indica el número de imágenes clasificadas como positivas, siendo positivas.
  - **FP**: 'False Positives'. Indica el número de imágenes clasificadas erróneamente como positivas.
  - **TN**: 'True Negatives'. Indica el número de imágenes clasificadas erróneamente como negativas, siendo negativas.

- **FN**: 'False Negatives'. Indica el número de imágenes clasificadas erróneamente como negativas.
- **Ntestpos**: Número de ejemplos de test positivos.
- **Ntestneg**: Número de ejemplos de test negativos .

Presentaremos los resultados en forma de tablas, gráficas y resultados visuales donde el análisis subjetivo también juega un papel importante, en el capítulo 5.

CAPÍTULO 4

Implementación

Este capítulo está dedicado a explicar el funcionamiento general del programa, exponiendo cada una de las fases de las que está formado, desde el momento en que se obtiene una imagen de la cámara hasta que conseguimos la detección de las personas. Vamos a describir aquellas líneas de código cuya interpretación puede resultar más compleja, así como algunas de las funciones correspondientes a las librerías OpenCV consideradas más importantes para el correcto desarrollo del algoritmo.

Como ya hemos mencionado anteriormente en el capítulo 3 (3.2.1), el programa está dividido en dos partes. La primera parte es el train o entrenamiento, donde se coge una base de datos de imágenes y de allí se sacan una serie de características, para luego poder detectar a las personas en la segunda parte, el test.

Utilizamos la base de datos *MIT Pedestrian* que representa una base de datos estándar en la detección de personas y contiene un total de 924 imágenes positivas de tamaño 64x128. Dichas personas se encuentran en determinadas posiciones, con diferentes iluminaciones y diferentes fondos. Por otra parte, tambien incluimos la base de datos de *INRIA*, donde originalmente se presenta un conjunto de datos positivos y negativos, organizadas en dos carpetas: test y train. Cada una de estas carpetas contiene a su vez dos más, una con ejemplos positivos y otra con negativos.

Primero veremos un breve resumen de la implementación y acontinuación veremos la adecuación de la base de datos, para seguir con la parte del train y finalmente terminar con la parte del test.

4.1. Resumen

Para empezar a explicar el procedimiento de nuestra aplicación, explicaremos el proceso esquemáticamente para poder entender mejor lo que estamos haciendo, dividido en dos partes: entrenamiento y el test.

1. Modificar la base de datos de *INRIA*, para mejorar el entrenamiento.
2. Leer archivos de imagen de la muestra de entrenamiento positivos y negativos de los directorios especificados.
3. Calcular sus características HOG y realizar un seguimiento de sus clases (pos, neg).
4. Guardar mapa de características (vector de vectores / matrices) al sistema de archivos.
5. Leer y aprobar las características y sus clases a un algoritmo de aprendizaje de máquina, SVM^{light}.
6. Capacitar al algoritmo de aprendizaje automático utilizando los parámetros especificados.
7. Utilizar los vectores de soporte calculados y modelo de SVM para calcular un solo descriptor de detección de vectores.
8. Establecer el vector descriptor y utilizar la entrada de cámara para probar el detector HOG recién generado.
9. Establecer la parte del test, para probar los resultados.

4.2. Experimentación con las bases de datos

Para la creación de la base de datos de entrenamiento se ha tomado como referencia la base de datos de personas *INRIA* [43]. La base de datos *INRIA* se compone de 3302 imágenes humanas. Como se muestra en la figura 4.1, el conjunto de imágenes disponible cuenta con varias vistas de una misma figura, aumentando así el número de posturas reconocibles. Por otro lado, las imágenes negativas son variadas para así poder entrenar mejor, desde imágenes de un entorno de una calle peatonal, como partes de una bicicleta o de un árbol (figura 4.3).

Figura 4.1: Ejemplos positivos de la base de datos *INRIA*.

Figura 4.2: Ejemplos negativos de la base de datos *INRIA*.

Las carpetas 'train_64x128_H96' y 'test_64x128_H96' se corresponden con el conjunto de datos normalizado. Ambas carpetas tienen dos subcarpetas: (a) 'pos' (contienen imágenes positivas), (b) 'neg' (que contiene el entrenamiento negativo original o imágenes de prueba).

En cambio, las imágenes en la carpeta *train/pos* son de 96x160 píxeles (un margen de 16 píxeles alrededor de cada lado), y las imágenes en la carpeta *test/pos* son de 70x134

píxeles (un margen de 3 píxeles alrededor de cada lado). Esto se ha hecho para evitar condiciones de contorno (por lo tanto, para evitar cualquier sesgo particular en el clasificador). En ambas carpetas, se utiliza una ventana de 64x128 píxeles centrada en la tarea de detección inicial.

Por último, las imágenes negativas para el entrenamiento y para el test son de diferentes dimensiones desde los 70x134 píxeles hasta los 420x380 píxeles.

Las imágenes de *INRIA* se han modificado, ya que en muchas imágenes positivas para el entrenamiento no esta claro dónde esta la persona, es decir, la base de datos no es del todo adecuada para el objetivo del proyecto, ya que aparecen personas ocultas por el entorno que pueden confundir en el entrenamiento (figura 4.3).

Figura 4.3: Ejemplos positivos de la base de datos *INRIA* que causan confusion en el entrenar.

Por otra parte, se a utilizado la base de datos *MIT Pedestrian* [57] que representa una base de datos estándar en la detección de personas y contiene un total de 924 imágenes positivas de tamaño 64x128 (figura 4.4). Dichas personas se encuentran en determinadas posiciones, con diferentes iluminaciones y diferentes fondos.

Figura 4.4: Ejemplos positivos de la base de datos *MIT pedestrian*.

Sin embargo, esta base de datos es muy limitada ya que las situaciones reales están llenas

de una mayor variabilidad tanto en la posición de las personas como en el fondo. *MIT Pedestrian* no tiene imágenes negativas en su base de datos, por lo tanto, hay que meter otras imágenes aparte. Por ello incluimos la base de datos de *INRIA*, donde originalmente se presenta un conjunto de datos positivos y negativos, organizadas en dos carpetas: *test* y *train*. En esta base de datos encontramos un total de 3302 imágenes de tamaño 64x128 positivas para entrenamiento, 1132 imágenes de tamaño 70x134 positivas para test, 1218 imágenes negativas de tamaño 320x240 para entrenamiento y 453 imágenes negativas de tamaño 320x240 para test.

En total, vamos a juntar todas las imágenes en dos grupos: positivas y negativas. Aparte de la base de datos de *INRIA* y *MIT*, hemos metido unas cuantas imágenes que hicimos nosotros en el taller, bien positivas, bien negativas para mejorar el rendimiento. Estos conjuntos de imágenes contienen personas de diversos tamaños, en movimiento, con fondos variantes, diferentes condiciones de iluminación, ligeras rotaciones de la cámara sobre un eje perpendicular al plano de la imagen, así como muchos otros factores. Respecto a los ejemplos negativos, decidimos generar un programa que seleccione subimágenes de tamaño 64x128 a partir de cada una de 320x240. Dichas subimágenes son generadas en posiciones totalmente aleatorias y con tamaños aleatorios, siendo luego normalizados a 64x128. Con esto aumentamos el conjunto de imágenes negativas. Además, decidimos calcular las reflexiones sobre un eje vertical de todas las imágenes positivas y negativas, y añadir estas muestras a la base de datos. En definitiva obtenemos un total de: 8024 imágenes positivas y 16043 imágenes negativas, sin distinción alguna referida a su uso para entrenamiento o test. Más adelante se verá cómo estructurar este conjunto de ejemplos para procederá la experimentación y veremos como ampliamos el conjunto muestral mediante el aprendizaje de la máquina clasificadora.

4.3. Modificación la base de datos de *INRIA*

Las imágenes de *INRIA* no vienen en las dimensiones que necesitamos. Las positivas están en 96x160 píxeles. Por ello, hemos hecho un programa para que recorte siempre desde el centro a 64x128 píxeles ya que las personas para el entrenamiento se encuentran en el centro de la imagen, y en los bordes hay otros objetos que podrían entorpecer nuestro entrenamiento.

Las imágenes negativas de *INRIA*, por el contrario, no tienen las mismas dimensiones, varían entre 248x373 píxeles hasta 420x380 píxeles, por lo que hemos optado por recortar cada imagen negativa en seis sub-imágenes de 64x128 píxeles.

4.3.1. Modificación de las imágenes positivas

Como podemos ver en el siguiente pseudocódigo (listing 4.1), cogemos cada imagen una por una y seleccionamos en ella una región de interés, que empezara desde las coordenadas 13x31 así logrando recortar la imagen desde el centro con las dimensiones deseadas.

Listing 4.1: Modificación de las imagenes positivas

```
Entorno:X,Y son las coordenadas donde empieza la imagen que queremos recortar.
Algoritmo:

    proceso modificacion_imagenes_positivas

    obtener las imágenes del directorio
    para i=0 hasta el total de imagenes con incremento +1{
        leer la imagen y meterla a una matriz
        calcular X dividiendo la columnas en 2 y restando 32
        calcular Y dividiendo las filas en 2 y restando 64
        coger el rectangulo deseado de la imagen
        guardar el rectangulo
    }
```

4.3.2. Modificación de las imágenes negativas

Como hemos comentado antes, las imágenes negativas de *INRIA* no tienen un tamaño especifico. Por ello, dividimos las columnas y las filas de cada imagen en tres, siempre teniendo en cuenta el ancho máximo de la imagen que queremos tomar. Se pierden unos

cuantos píxeles de la margen derecha de la imagen original, pero no es relevante. Por lo tanto hacemos dos bucles para así poder sacar de una imagen 6 sub-imágenes y así poder mejorar el entrenamiento (listing 4.2).

Listing 4.2: Modificación de las imagenes negativas

```
Entorno:X1,Y1,X,Y son las coordenadas donde empieza la imagen que queremos recortar.
Algoritmo:

    proceso modificacion_imagenes_negativas

    obtener las imágenes del directorio
    para i=0 hasta el total de imagenes con incremento +1{
        leer la imagen y meterla a una matriz;
        calcular X1 dividiendo la columnas-65 en 3;
        calcular Y1 dividiendo las filas-129 en 3;
        X,Y 0
        para z=0 hasta z<3 con incremento +1{
            para j=0 hasta j<3 con incremento +1{
                crear rectangulo(X,Y,64,128);
                guardar el rectangulo como imagen;
                establecer la nueva X, X + X1;
            }
            establecer la nueva Y;
            establecer X a 0
        }
    }
```

4.4. Descripción del código de entrenamiento

La función de entrenamiento, es muy importante dentro de la aplicación debido a que su funcionamiento puede determinar un elevado grado de aciertos a la hora de comparar imágenes.

La función de entrenamiento debe encargarse de encontrar y almacenar una serie de características de cada imagen para que éstas puedan ser comparadas.

Durante el desarrollo del programa, el análisis de las imágenes se lleva acabo con imágenes que en su momento fueron tomadas por una cámara, pero que se encuentran almacenadas en una carpeta del ordenador. Dichas imágenes para el entrenamiento están dentro de la carpeta *data*. Allí se encuentran diferentes carpetas para el entrenamiento, pero para explicar el programa solo utilizaremos la carpeta *pos1* y *neg_inria*.

En el archivo *features.dat* guardaremos las características extraídas y en *descriptorvector.dat* el vector descriptor, que se explica detalladamente en esta sección.

Listing 4.3: Dirección de las carpetas y archivos

```
1 Directorio ejemplos positivos = "../data/pos1/";
2 Directorio ejemplos negativos = "../data/neg_inria";
3 Archivo para guardar las características = "../genfiles/features.dat";
4 Archivo para guardar el modelo SVM = "../genfiles/svmlightmodel.dat";
5 Archivo para guardar el vector descriptor = "../genfiles/descriptorvector.dat"
```

Utilizaremos los parámetros por defecto que no están incluidos en la clase HOG. La separación entre bloques es de 8x8. Los bloques son de 16x16 píxeles (1 bloque = 2x2 celdas, 1 celda = 8x8 píxeles). El algoritmo utiliza un 50 % de superposición entre los bloques, en el apartado de resultados (Capítulo 5) analizaremos por qué utilizamos estos parámetros y otros que no están incluidos en la clase HOG(listing 4.4).

Listing 4.4: Parámetros no incluidos en la clase HOG.

```
1 Relleno trainingPadding = Tamaño(0,0);
2 Separación entre bloques winStride = Tamaño(8,8);
```

4.4.1. Parámetros del HOG

- **win_size**: Tamaño de la ventana de detección.
- **block_size**: El tamaño de bloques en píxeles.
- **block_stride**: Paso Block. Tiene que ser un múltiplo del tamaño de la celda.
- **cell_size**: Tamaño de la celda.
- **nbins**: Número de contenedores.
- **win_sigma**: Parámetro Gaussiano para suavizar la ventana.
- **threshold_L2hys**: Umbral para el método de normalización.
- **gamma_correction**: Bandera para especificar si se requiere o no el procesamiento previo de corrección gamma.
- **nlevels**: Número máximo de incrementos de detección.

4.4.2. Train

En el Main del entrenamiento lo primero que se hace es abrir las imágenes una por una y llamar al método $calculateFeaturesFromInput(currentImageFile, featureVector, hog)$ para poder calcular el vector de gradientes de cada imagen. Como argumentos recibe el nombre de la imagen y el hog.

Cálculo de características

Como se ve en el Listing 4.5, empezamos leyendo la imagen, para poder escalarla. Para ello utilizamos *cv::Mat*.

Convertimos la imagen a un solo canal de escala de grises y escalamos a la imagen a (64, 128) para mejorar el rendimiento y simplificar los cálculos. Las imágenes de *INRIA* las hemos recortado previamente a este tamaño, centrandonos solo en la persona ya que tenian un margen demasiado grande para que fuese un clasificador fiable. En cambio, las de *MIT* están en diferentes tamaños y no utilizan el mismo tipo de marco. Con esta conversión se reduce el número de píxeles que se tienen que computar, ya que se ha pasado de una imagen con tres canales a una imagen de un solo canal. Para realizar el

cálculo de bordes, no son relevantes los valores concretos de intensidad de cada canal de color primario (RGB), sino más bien la variación de intensidad global que se produce. De este modo se facilita el siguiente paso de búsqueda de contornos.

Listing 4.5: Paso para escalar a (64,128) y un solo canal.

```
leer la imagen
SI la imagen esta vacía
    limpiar el vector de características
    imprimir mensaje de error
SI NO
    cambiar el tamaño a (64,128)
    convertir la imagen a un solo canal
```

Debido a que la función *hog.detectMultiScale* empleada trabaja con imágenes de tipo *cv::Mat*, se realiza la conversión de la imagen original para de ella extraer las regiones de interés. El tamaño idóneo de las regiones se ha obtenido tras realizar una serie de pruebas como ya se verá en el Capítulo 5.

Para extraer las características (Listing 4.6) se utilizan los parámetros por defecto del HOG *winStride=Size(8,8)* y *padding=Size(0,0)*. Para ello se utliza la funcion $hog :: compute$. Una vez que se extraen las características se libera la imagen, asi conseguimos que no se llene la memoria.

Listing 4.6: Paso para extraer las características HOG.

```
vector locations;
hog.compute(imageData, featureVector, winStride,
            trainingPadding, locations);
liberar la imagen;
```

Al sacar las características *calculateFeaturesFromInput* pasa al programa *main* el vector *featureVector*, con las características de la imagen.

Estas características se guardan en la carpeta *data* en $features.dat$ donde el tamaño del archivo ronda los 110 Mb para luego poder pasar a SVM^{light}.

SVM^{light}

Dentro del programa principal, lo último que se hace es pasar las características sacadas en el anterior apartado para que lo entrene. A partir de este punto, el algoritmo se centra en la detección de los objetos de la imagen y sus clasificaciones posteriores. Por defecto se utiliza un suave entrenador SVM lineal con SVM^{light} ligeramente modificado para reducir el uso de la memoria, ya que genera vectores descriptores muy grandes.

Antes de aplicar la función *hog.detectMultiScale*, tiene que haber sido entrenado el HOG mediante el SVM (listing 4.7), empleando la función $SVMlight :: getInstance() -> train()$; y luego $SVMlight :: getInstance() -> saveModelToFile(svmModelFile)$; para guardar el modelo en el archivo *genfiles/features.dat*. Este entrenamiento basta con realizarlo una única vez, por lo que se añade al *main*.

Listing 4.7: Fase del entrenamiento con SVM.

```
1 Imprimir ("Llamando a SVMlight");
2 SVMlight::getInstance()->read_problem(featuresFile);
3 SVMlight::getInstance()->train(); //Llamar a libsvm para entrenar
4 SVMlight::getInstance()-> Guardar modelo (svmModelFile);
```

A continuación vemos una captura de pantalla del entrenamiento, este proceso dura aproximadamente seis minutos:

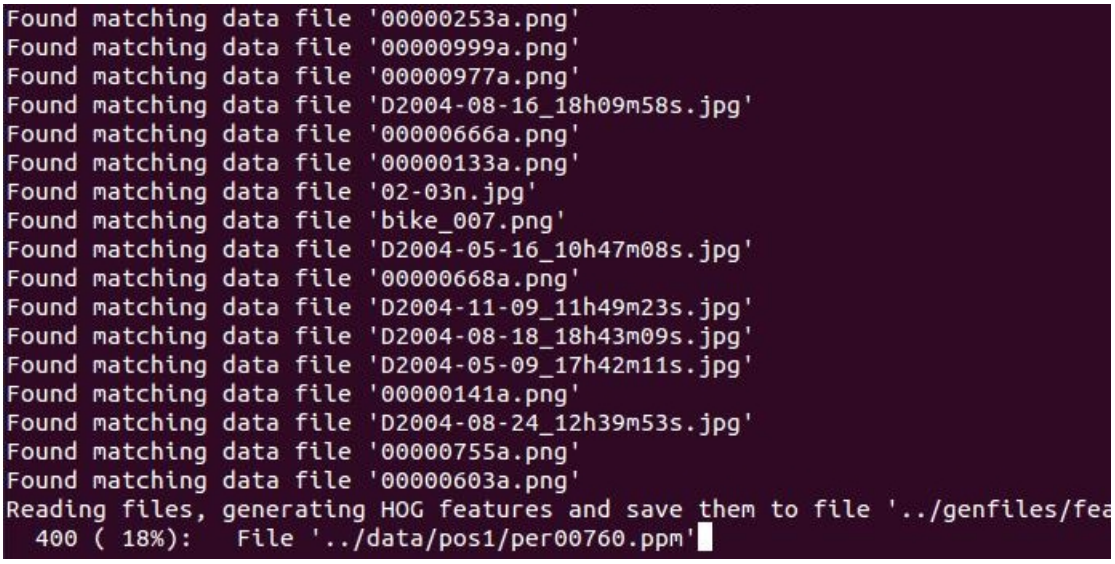


Figura 4.5: Captura de pantalla.

Como podemos ver en la figura 4.5, se visualizan primero las imágenes de la base de

datos, se generan las características de HOG de cada imagen y se guardan en *features.dat* (figura 4.6), para luego entrenar con SVM.

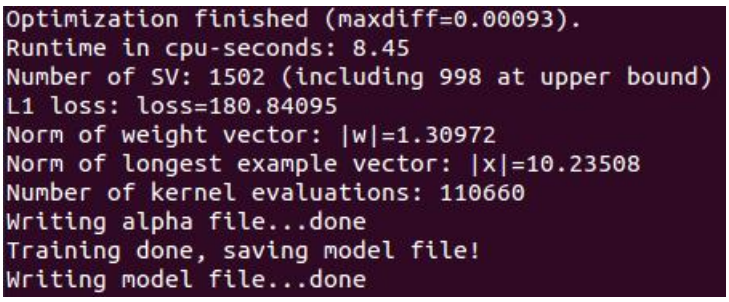


Figura 4.6: Captura de pantalla una vez terminado el entrenamiento.

4.4.3. Test

Una vez guardado el entrenamiento de las imágenes, se procede a la parte del test. En este apartado se analizarán las imágenes en las que se desean detectar personas y asi poder ver el grado de fiabilidad del entrenamiento generado.

Parámetros:

- **Sin parámetro**: Detección con la Webcam por defecto.
- **Primer parámetro**: Dirección donde esta el fichero con las direcciones de las imágenes positivas.
- **Segundo parámetro**:Dirección donde esta el fichero con las direcciones de las imágenes negativas.

Lo primero que hacemos es cargar el fichero del entrenamiento (listing 4.8):

Listing 4.8: Cargar el SVM.

```
1 SVMlight::getInstance() ->loadModelFromFile(svmModelFile);
```

Antes de aplicar la función *hog.detectMultiScale*, tiene que haber sido entrenado el HOG mediante el SVM, empleando la función *hog.setSVMDetector*. Basta con realizarlo una única vez, por lo que se añade al *main*.

Listing 4.9: hog.setSVMDetector.

```
1 hog.setSVMDetector(descriptorVector);
```

Al llamar a la función *hog.detectMultiScale* (listing 4.10), se obtiene como resultado un vector dinámico al cual se accede en el caso de que se haya localizado en dicha región un peatón.

Listing 4.10: detectMultiScale.

```
1 vector found;
2 groupThreshold = 2;
3 padding(32,32);
4 winStride(8,8);
5 hitThreshold = 0.;
6 hog.detectMultiScale(testImage, found, hitThreshold, winStride, padding, 1.05,
      groupthreshold);
```

Parámetros de *HOGDescriptor* :: *detectMultiScale*:

- **testImage**: La imagen en la que queremos que detecte.
- **found**: Límites de lo objetos detectados.
- **hitThreshold**: Umbral para la distancia entre las características y el plano de clasificación de la SVM.
- **winStride**: El paso de la ventana.Debe ser un múltiplo de *blockStride*.
- **padding**: Parámetro Mock para mantener la compatibilidad de interfaz de la CPU (0,0).
- **scale**: Coeficiente de la ventana de detección de aumento.
- **group_threshold**: Coeficiente para regular el umbral de similitud. Cuando se detecta, algunos objetos pueden ser cubiertos por muchos rectángulos. 0 significa no realizar el agrupamiento.

Una vez hecho *detectMultiScale* se dibujan los rectangulos donde están las personas. Para ello, hemos hecho la función *showDetections* (listing 4.11):

Listing 4.11: showDetections.

```
imprimir numero de personas detectadas;
showDetections(found, testImage, image_file);
```

Como parámetros pasamos el vector $found$, la imagen y el nombre de la imagen.

Por el contrario, si queremos hacer la parte del test por la Webcam (listing 4.12), no le pasamos ningún parámetro al programa. Lo primero que hacemos es abrir la cámara y comprobar si lo hemos conseguido. En el bucle obtenemos un frame, pasamos a escala de grises y hacemos la parte del test. Por último, se muestra la imagen con las personas detectadas.

Listing 4.12: Test obteniendo imágenes por la Webcam.

```
SI NO esta la camara abierta{
    imprimir error;
    return fallo al abrir;
}
MIENTRAS NO se pulse la tecla Esc{
```

```
6       capturar la imagen de la webcam;
7       cambiar la imagen a escala de grises;
8       detectTest(hog, imagen);
9       enseñar la imagen detectada;
10  }
```

PARTE III: CONCLUSIONES Y TRABAJO FUTURO

CAPÍTULO 5

Resultados

En este capítulo se analizan los resultados obtenidos por el algoritmo desarrollado en el presente proyecto. Para concretar los parámetros, se han hecho las pruebas con la base de datos de *INRIA* y *MIT Pedestrian* para sacar las conclusiones de qué SVM y con qué parámetros utilizar en nuestro algoritmo. Para realizar el estudio de los resultados se han seleccionado un par de secuencias de imágenes en las que se pueden observar dos situaciones diferentes que tienen en común la presencia de varios peatones en una vía urbana.

El objetivo principal de este estudio consiste en crear un sistema robusto capaz de conseguir unas predicciones lo más exactas posibles. Cuando hablamos de exactitud nos referimos principalmente a dos parámetros: a) el número de falsos positivos que da el sistema y b) el número de falsos negativos, es decir, personas no detectadas.

5.1. Resultados

Los resultados obtenidos en los siguientes apartados han sido obtenidos bajo el sistema operativo Ubuntu 12.04.4 LTS. Las máquinas utilizadas han sido de tipo INTEL 1.73GHz con 2GB de RAM. Por otra parte, la version de OpenCV es la 2.4.3.

Se han hecho dos tipos diferentes de pruebas. Para la primera prueba, se han utilizado las imágenes originales que están en la carpeta Test de *INRIA*, con el que han servido para

concluir qué Kernel utilizar, además de las conclusiones de la implementación. En total son 2332 imágenes, divididas en 614 positivas y 1718 negativas.

En la segunda prueba se ha seleccionado una secuencia de imágenes tomadas en el taller de Fatronic con la presencia de varios trabajadores. La secuencia está compuesta por 853 imágenes.

En primer lugar, se van a analizar los resultados obtenidos con las imágenes del test de *INRIA* para luego terminar con la secuencia del taller.

5.2. Pruebas con el test de *INRIA*

5.2.1. Efecto de realimentación de la máquina

La ventaja de este tipo de máquinas clasificadoras es la posibilidad que permiten de aprendizaje. Esto es, la posibilidad de ser re-entrenadas con todas aquellas muestras que en otras ocasiones ha clasificado mal. Este hecho hace a la máquina más robusta con cada entrenamiento.

Para probar el efecto que tiene el entrenamiento de la máquina, partimos de un kernel previo considerado como definitivo, en este caso un kernel paramétrico con d=3, entrenado con 614 ejemplos positivos y 1718 ejemplos negativos. Por otro lado, disponemos para el testeo de otro conjunto muestral extenso con ejemplos negativos. Clasificamos estos ejemplos negativos con la máquina, y obtenemos unas predicciones. Seleccionamos todas aquellas muestras mal predichas o cuyo valor se encuentre cercano a cero, y las incluimos en el conjunto muestral negativo con el que hemos entrenado inicialmente la máquina, aumentando el número de ejemplos.

Para ver con mayor claridad el efecto del re-entrenamiento, repetimos el proceso para un número de muestras de test mayor, considerando el umbral: th=0;

En la figura 5.1 se muestra un pequeño conjunto de muestras que son realimentadas. Se pueden apreciar ciertas formas que pueden asemejarse a las siluetas de una persona.

En la tabla 5.1 se aprecia cómo, tras la realimentación del kernel con 500 muestras negativas, obtenemos considerables mejoras en cuando al parámetro *Precision*. Sin embargo, disminuye ligeramente el *Recall*. Esto es debido a que el parámetro *Precision* está ligado con los falsos positivos, y debido a que hemos realimentado con muchos más ejemplos negativos, hemos reducido este problema.

Figura 5.1: Ejemplos de imágenes que han sido clasificados como positivos por las primeras versiones del clasificador.

	Kernel sin realimentar	**Realimentando con 500 muestras neg**	**Realimetando con 1500 muestras neg**
Precision(%)	82.1	90.36	92.76
Accuracy(%)	80.61	84.19	83.37
MR(%)	0,1891	0,2069	0.2602
Recall(%)	81.09	79.31	73.98

Tabla 5.1: Resultados del efecto de realimentación de muestras en un kernel.

Para el caso de realimentación con 1000 muestras, se mejora mucho más el parámetro *Precision* del clasificador. Sin embargo, notamos un importante descenso en el *Recall*. Esto se debe a que el conjunto muestral de ejemplos negativos para el entrenamiento es mucho mayor que el conjunto de ejemplos positivos (3218 frente a 614). Por ello, el clasificador tiende a perder candidatos.

Sin embargo, esto no tiene por qué ser un problema. Si modificamos el umbral de decisión, podemos obtener resultados de *Recall* suficientemente buenos, sin llegar a perder excesiva *Precision*.

Las siguientes gráficas (figura 5.2 y figura 5.3) muestran la relación entre los parámetros FPPW y MR, variando el umbral de decisión:

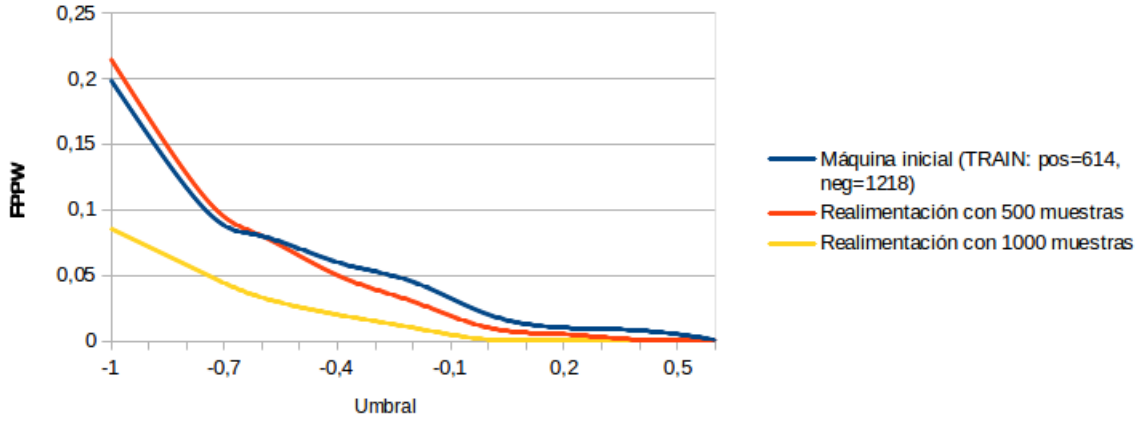


Figura 5.2: Gráfica FPPW-Umbral.

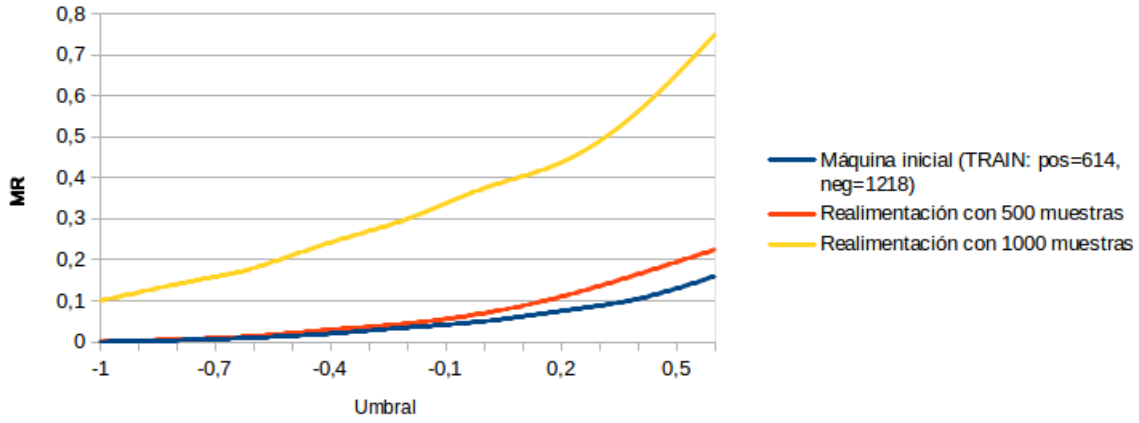


Figura 5.3: Gráfica MR-Umbral.

5.3. Kernel para la SVM

El conjunto de imágenes de entrenamiento es el citado en este capitulo (614 ejemplos positivos y 1718 ejemplos negativos). Sin embargo, por motivos que acabamos de explicar, hemos aumentado el conjunto muestral de ejemplos negativos, utilizando la realimentación. Para ello, se ha generado previamente otro conjunto de imágenes negativas (aproximadamente 3200 muestras). Aquellas muestras mal clasificadas o etiquetadas con valores negativos cercanos a cero han sido incorporadas a la base de datos definitiva.

En este apartado vamos a definir cuál es la mejor máquina clasificadora para nuestro caso. Asumiremos durante este estudio que el umbral de decisión para determinar si una etiqueta es positiva o negativa es igual a 0 (th = 0).

En las tablas siguientes contrastamos los parámetros más significativos, es decir los valores de *Accuracy*, *Precision* y *Recall*. Ha esto le añadimos el *Space in disk (MB)*, que representa el espacio en MB que ocupa en disco cada kernel, y el tiempo medio que la máquina necesita para clasificar una imagen de test.

Kernel polinomial

	d=2	**d=3**	**d=4**	**d=5**	**d=6**	**d=7**	**d=8**	**d=9**
Precision (%)	89,78	**90,11**	90,43	90,43	90,52	90,63	90,72	90,83
Accuracy (%)	84,14	**84,24**	84,14	84,12	84,12	84,05	83,82	83,6
Recall(%)	78,56	**79,76**	78,43	78,35	78,24	77,76	77,54	77,25
Number of Support Vector	3842	**4163**	4506	4853	5202	5581	5973	6380
Space in disk (MB)	28.6	**31.2**	33.4	35.9	38.5	41.2	44.6	47.5
Runtime per image (s)	0.0171	**0.0183**	0.0192	0.0203	0.0205	0.0227	0.0238	0.0251

Tabla 5.2: Estudio comparativo para un kernel lineal variando el parámetro gamma d.

Como hemos comentado en la introducción a la sección de Resultados (5), nos interesa disponer de un valor de *Precision* elevado. En la tabla 5.2 se observa cómo para valores del parámetro 'd' tenemos un valor alto de precision. Sin embargo, observamos un mayor incremento en el coste temporal al compararlo con valores de 'd' ligeramnente inferiores. Además, para valores elevados de este parámetro obtenemos muy bajos porcentajes de

Recall, lo cual no es bueno. Sin embargo, para valores de 'd' iguales a 3 o 4, tenemos valores de *Precision* aceptables, y nos aproximamos ligeramente al óptimo valor de *Recall*. Además, los costes temporales para estos kernels son más bajos.

Por todo lo anterior, encontramos el óptimo valor de 'd' para d=3, ya que existe un buen compromiso entre los valores de *Precision*, *Recall* y *Runtime*.

Radial basis kernels

	g = 0.05	**g = 0.1**	**g=0.125**	**g=0.15**	**g= 0.175**	**g=0.2**
Precision (%)	84,11	85,45	86,12	87,21	88,3	89,02
Accuracy (%)	84,2	84,2	84,2	84,37	84,69	84,77
Recall(%)	79,2	79,15	79,01	78,15	79,88	79,91
Number of Support Vector	3967	4337	4337	4457	4589	4717
Space in disk (MB)	29,9	31,3	32	33	34	35
Runtime per image (s)	0.018	0.0191	0.02	0.0209	0.0192	0.0193

	g = 0.25	**g =0.3**	**g = 0.5**	**g=1**	**g=2**	**g=4**
Precision (%)	91,06	90,72	90,38	86,23	83,01	81,48
Accuracy (%)	85,21	85,01	84,27	83,95	81,3	77,3
Recall(%)	78,4	77,9	75,78	71,01	54,63	21,63
Number of Support Vector	4978	5233	6371	10563	25622	31965
Space in disk (MB)	37	39	47	80	130	235
Runtime per image (s)	0.0195	0.0198	0.02	0.0312	0.0641	0.082

Tabla 5.3: Estudio comparativo para un kernel gausiano variando el parámetro gamma g.

Una vez realizado el estudio de los diferentes kernels, seleccionaremos los que tengan mejores prestaciones. Sin embargo, llegados a este punto la máquina no puede considerarse definitiva todavía. Debemos notar que para que un resultado sea considerado como aceptable debe alcanzar valores de precisión altos.

Comparando los mejores kernels polinómicos con los mejores gausianos, vemos como todos reunen características similares. El parámetro *Recall* es ligeramente superior en el caso de los gausianos preseleccionados. Asimismo, el coste temporal en la clasificación de una muestra es ligeramente mejor para el caso gausiano. Aunque el parámetro *Precision*

es más elevado en el caso paramétrico seleccionado, vemos que si hubiéramos seleccionado un kernel gausiano con g=0.25 obtendríamos mejores prestaciones que en el kernel polinómico propuesto, en todos los parámetros. Es por ello por lo que nos inclinamos por la selección de los kernels gausianos con valores de g=0.175 y g=0.2 como los definitivos. Sobre ellos se basarán los apartados siguientes.

5.4. Rendimiento del método HOG

A continuación se va a comentar cómo los parámetros introducidos en el método HOG afectan al rendimiento. Para cuantificar el rendimiento del detector se trazan las curvas de compensación del error de detección (DET - *Detection Error Trade-off*). Esta curva surge para solucionar uno de los inconvenientes que presenta la curva ROC, que es que no tiene en cuenta el error de clasificar mal las muestras positivas.

A menudo se utiliza el ratio 10^{-4} como punto de referencia para los resultados. Esto corresponde a una tasa de error de alrededor de 0.8 falsos positivos por imagen.

En las curvas DET, pequeñas mejoras en el ratio de pérdida dan lugar a importantes aumentos de FPPW en la tasa de fallos.

5.4.1. Escala de gradiente

El rendimiento del detector es sensible al modo en el que se obtienen los gradientes, pero es el sistema más sencillo el que resulta ser el mejor al proporcionar mejores resultados (figura 5.4).

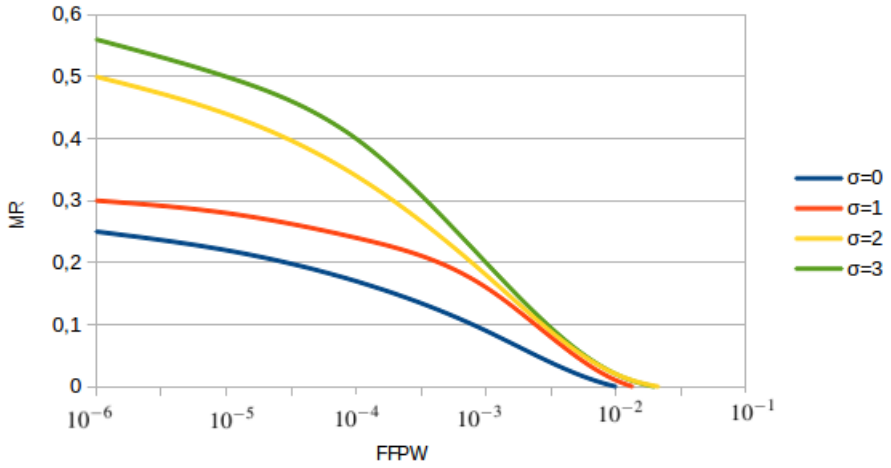


Figura 5.4: Estudio del efecto que tiene σ sobre la escala del gradiente.

Las máscaras centradas [-1, 0, 1] con $\sigma = 0$ son las que proporcionan mejor rendimiento.

5.4.2. Subrangos de orientación

A la hora de calcular el histograma de gradientes orientados de una imagen, lo que se hace, es dividir el rango que contiene todos los posibles valores de orientación de los gradientes en varios subrangos. De esta forma, en el histograma se tendrán los distintos gradientes de los píxeles agrupados según pertenezcan a un subrango o a otro.

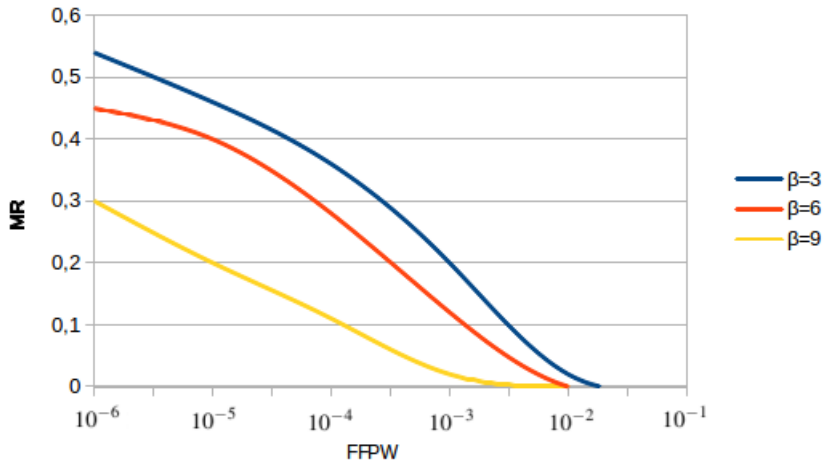


Figura 5.5: Estudio del efecto que tiene sobre el rendimiento la modificación del número de subrangos de orientación.

Debido a la arbitrariedad que existe en la distribución de la luminosidad en las imágenes, se considera que en el cálculo de los gradientes se podría prescindir de la información acerca del sentido del gradiente. Esto es así porque en la detección de personas lo que interesa saber no es el sentido sino su dirección, debido a la arbitrariedad en las prendas que puedan llevar los peatones y en los diferentes objetos e iluminaciones que puedan darse en el fondo de la imagen. Interesa conocer la existencia de un cambio de luminosidad y su distribución, pero no en qué dirección.

En la figura 5.5 se muestra cómo incrementar el número de subrangos de orientación mejora el rendimiento significativamente. Después de una serie de pruebas, representadas en el grafico, se llega a la conclusión de que el número más apropiado de subregiones es 9, pues a partir de este número la diferencia experimentada es nula.

5.4.3. Número de celdas superpuestas

El rendimiento mejora cerca de un 3 % en 10^{-4} FPPW a medida que se aumenta la superposición. En la figura 5.6 se analiza el efecto de solape entre bloques para valores de solape 0 (stride=16), ½ (stride=8) y ¾ (stride=4).

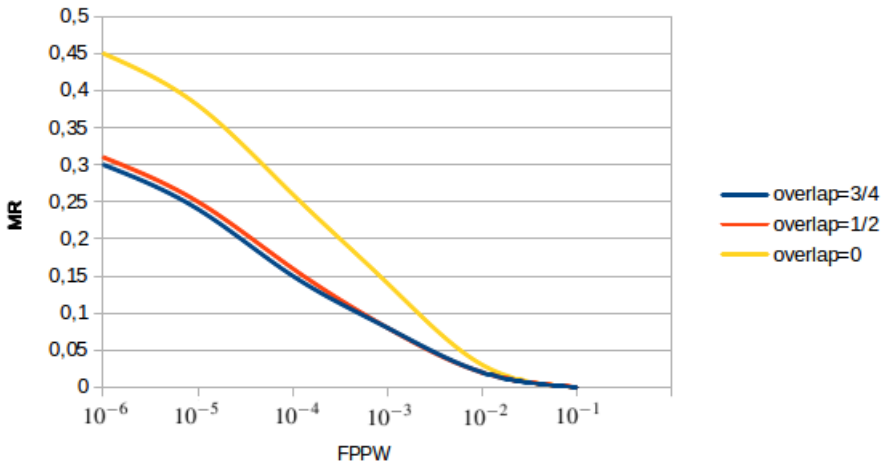


Figura 5.6: Estudio del que tiene la superposición sobre la escala del gradiente.

Se llega a la conclusión de que los mejores resultados dentro de que exista un solape entre bloques se obtienen para un solape de ½ . Esto es debido a que al estar más espaciados los bloques, que en solape=0 es necesario un menor número de ellos para cubrir toda la imagen, lo que hace que el rendimiento mejore. No hay mucha diferencia entre el solape ¼ y solape ¾ y cogiendo el solape ½ es necesario un menor número de bloques para cubrir toda la imagen, lo que hace que el rendimiento mejore.

Los parámetros establecidos inicialmente para llevar a cabo este estudio son: HOG: GMA = 8píxeles (SIGMA corresponde con el tamaño de una celda). SVM: kernel Gausiano g=0.2, th=0.

	Solape ¾	Solape ½	Solape 0
Precision(%)	93.80	94.01	94.36
Accuracy(%)	83.24	85.34	86.73
Recall(%)	93.43	93.52	93.62

Tabla 5.4: Estudio del efecto de los diferentes solapes.

Los resultados de la tabla 5.4 muestran cómo para solapes entre bloques menores se

reduce considerablemente el número de características del descriptor. Esto es debido a que al estar más espaciados los bloques, es necesario un menor número de ellos para cubrir toda la imagen.

5.4.4. Dimensiones de la ventana

Una ventana de detección de 64x128 incluye alrededor de 16 píxeles de margen alrededor de las personas en los 4 lados de la imagen. Esta frontera muestra una cantidad significante de información que permite la detección.

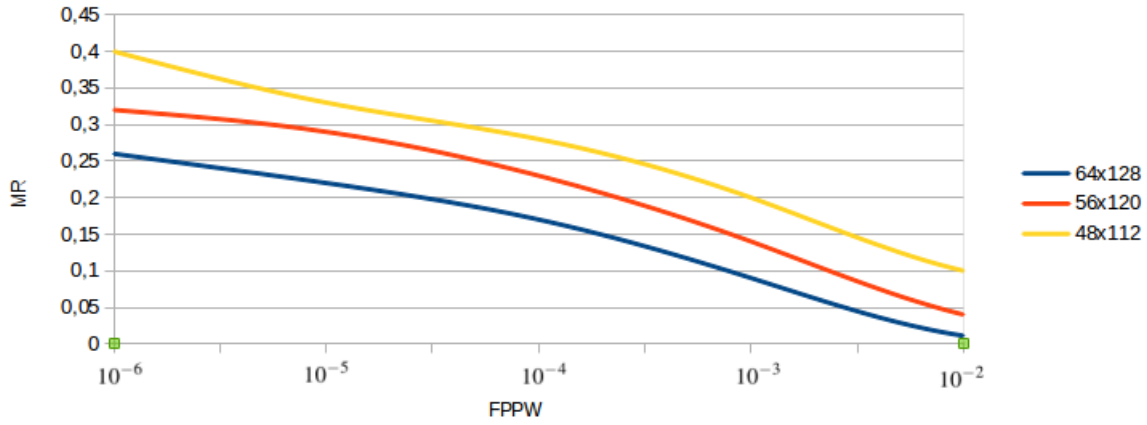


Figura 5.7: Estudio del efecto que tiene sobre el rendimiento la modificación de las dimensiones de la ventana de detección.

En la figura 5.7, se observa que si se reduce este margen de 16 a 8 píxeles, es decir, se pasa a una ventana de detección de dimensiones 48x112 el rendimiento se reduce un 8 % en 10^{-4} FPPW. Si manteniendo una ventana de 64x128 lo que se hace es aumentar el tamaño de la persona (en este caso también se está reduciendo la frontera) la pérdida de rendimiento es similar a la que se producía reduciendo el tamaño de la ventana, a pesar de que la resolución de la persona haya sido incrementada.

5.5. Secuencia en el taller

Esta secuencia de imágenes está tomada desde dos ángulos distintos, emulando un robot que está paseando de un lado al otro, transcurre en el taller de Fatronic. Hemos realizado esta prueba para ver los resultados en un taller real, para poder ver los resultados que da la aplicación en un entorno idoor. El objetivo del proyecto desde el principio era realizar una implementación para detectar personas en este tipo de entornos. En dicho taller se pueden observar varios peatones, uno caminando a paso normal, otro trabajando tal y como se muestra en la figura 5.8. Durante el trascurso de esta secuencia los peatones se acercan y se alejan de la cámara desapareciendo de la escena y volviendo a entrar. La escena completa se grabó un día con mucha gente trabajando en el taller, para así poder ver mejor los resultados obtenidos.

Figura 5.8: Imágenes de la secuencia tomada en el taller.

5.5.1. Resultados

Unos datos útiles para determinar el grado de eficacia del algoritmo desarrollado son los falsos positivos y negativos que se obtienen en cada una de las imágenes que componen las secuencias. La primera secuencia (figura 5.8) tiene 316 imágenes y la segunda (figura 5.9) 648 imágenes.

Figura 5.9: Imágenes de la secuencia tomada desde el segundo ángulo.

Solo con el entrenamiento original de *INRIA* y *MIT pedestrian*, va a existir un falso positivo cuando el algoritmo identifique como peatón al brazo robótico amarillo que hay en la izquierda de la imagen del segundo plano. Eso lo arreglamos con una realimentación en el entrenamiento.

Con todo ello, con un entrenamiento definitivo de 3000 imágenes negativas y 800 positivas conseguimos los siguientes resultados (tabla 5.5):

	Primera secuencia	**Segunda secuencia**
Precission(%)	88,56	88,48
Accuracy(%)	92,76	90,76
MR(%)	0,04	0,076
Recall(%)	96	92,39

Tabla 5.5: Resultados de las secuencias del taller.

Aun así, sin haber grandes cambios en los resultados, podemos ver que en un entorno real, con el adecuado entrenamiento se mejoran los resultados del test de *INRIA*, ya que el entorno no varia tanto como en las imágenes de test.

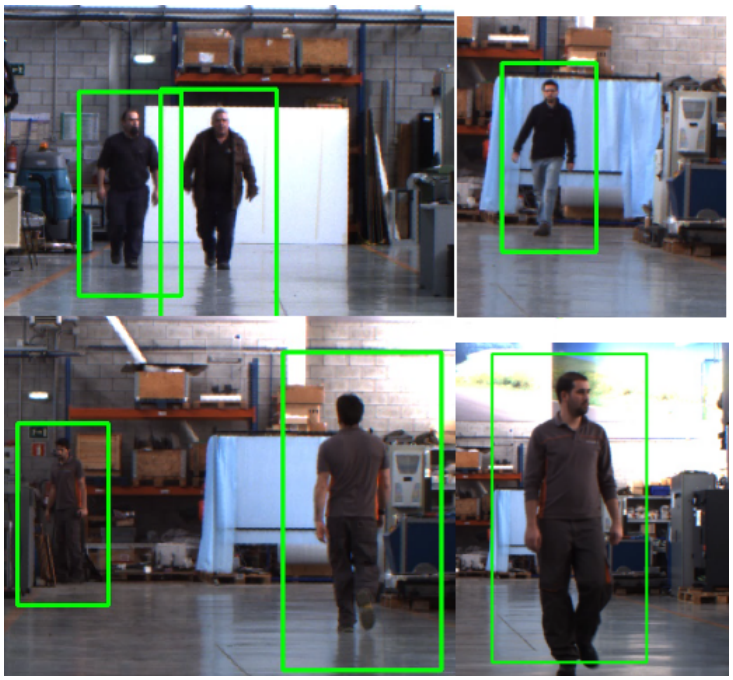


Figura 5.10: Ejemplos de las detecciones realizadas en el taller.

Para concluir, se puede decir que con los valores definidos finalmente se ha logrado alcanzar un equilibrio, de manera que los resultados ofrecidos por este algoritmo cumplen con el objetivo que se persigue en este proyecto.

CAPÍTULO 6

Conclusiones y trabajos futuros

Una vez analizados los distintos parámetros del algoritmo y obtenido los resultados correspondientes se puede proceder a exponer las conclusiones del proyecto. Además se mencionan posibles trabajos que podrían derivar del actual.

6.0.2. Conclusiones

Este trabajo se enmarca dentro del proyecto Europeo AUTORECON. El objetivo del proyecto AUTORECON es la implementación de un nuevo procedimiento de navegación basado en visión artificial en lugar de la utilización de los láseres. El sistema solo cuenta con una cámara mono.

La concepción inicial de esta tarea era la elaboración de un modelo 3D capaz de integrar los diversos dispositivos fuente y de realizar una detección de objetos en tiempo real. Tras unos meses de estudio, se comprobó que la mejor forma de abordar el problema de reconocimiento de objetos para hacerla efectiva en tiempo real era el procesado de imagen en 2D, descartando la creación de modelos en 3D por su elevado coste computacional. Esta opción sigue permitiendo la integración de este subsistema de detección de objetos con el resto de dispositivos de entrada.

El primer paso realizado fue centrarse en la detección de personas y estudio de la viabilidad de aplicar métodos ya existentes. Se vio que los métodos actuales, a pesar de dar muy buenos resultados de predicción, no sirven para nuestro trabajo, en primer lugar porque

pocos sistemas de detección de objetos van montados sobre un sujeto móvil, donde el fondo es cambiante y las imágenes empeoran su calidad, y en segundo lugar por el hecho de ser un sistema en tiempo real.

Por ello, decidimos recurrir a la combinación de HOG-SVM. Por un lado, se ha comprobado que la técnica de HOG en detección de personas proporciona resultados rápidos, aunque con algun que otro falso positivo. Para reducir estos falsos positivos decidimos realimentar el entrenamiento consiguiendo así mejores resultados. Conocimientos previos sobre estas técnicas nos mostraban una gran precisión en el funcionamiento de las máquinas clasificadoras, aunque conocíamos su elevado coste computacional.

El primer paso fue seleccionar un algoritmo de extracción de características robusto, como es el HOG. En segundo lugar, tratamos de mejorar calidad de las características extraídas así como su coste temporal. Para ello eliminamos la dependencia del gradiente con el sentido del mismo, tomando sólo en consideración su dirección. En segundo lugar establecemos los puntos óptimos donde extraer las características. Ello implica seleccionar un solape entre bloques óptimo, no sólo en robustez de las características, sino en eficiencia temporal. Para ello se realizó un estudio de las dimensiones del vector de características frente a los resultados estadísticos y el coste computacional, que reveló que el tamaño óptimo de bloque es de 4x4 celdas, cada celda con 8 píxeles, distribuidos con un solape entre bloques de la cuarta parte, y no calculando el descriptor en aquellos puntos de la imagen situados en los bordes, por el hecho de contener información no característica de las muestras, además de aumentar inútilmente las dimensiones del vector. Una vez optimizado el algoritmo de extracción de características nos centramos en la elaboración de una máquina de clasificación de características robusta. Para ello hemos probado diferentes tipos de kernels, obteniendo un el óptimo para un kernel con función gausiana con valor de gamma = 0.175. Hemos procedido a la realimentación de la máquina para hacerla más robusta, y así obtener la máquina final.

Asimismo, se ha buscado el umbral óptimo en las predicciones de SVM. Se ha tratado de optimizar los parámetros de Precision y Recall, que hacen referencia a la robustez del subsistema frente falsas alarmas y frente a la probabilidad de pérdida de candidatos, respectivamente.

El estudio de las prestaciones de este subsistema nos ha revelado que la combinación de estos dos métodos es satisfactoria, aunque no es óptima. Por una parte, el subsistema cumple con las especificaciones temporales de funcionamiento (10 Hz en el procesado de cada frame). Sin embargo, en cuanto a la fiabilidad del método no se ha llegado a

cumplir con lo esperado ya que los candidatos sufren ligeras desviaciones en su cuadro delimitador, con lo que la clasificación de estas muestras con SVM produce un elevado número de pérdidas en el sistema.

Aunque el sistema da bastantes buenos resultados, está lejos de cumplir con los óptimos, y por tanto de ser un sistema definitivo. Para mejorarlo se proponen diversas opciones, tanto metódicas como de fusión con nueva información sobre la escena, como son el uso de la información proporcionada por los mapas de profundidad generados por las estéreo-cámaras.

6.0.3. Trabajos futuros

Los cambios constantes en la iluminación y fallos en la detección de determinadas siluetas dan lugar a que se produzcan detecciones de falsos positivos y falsos negativos. No obstante, la tasa de error del algoritmo en este aspecto es lo suficientemente reducida como para no impedir lograr los objetivos finales. Buscar disminuir estos falsos resultados mediante el tratamiento de la imagen, implicaría un aumento del coste computacional.

Las técnicas de boosting consisten en métodos que combinan un conjunto de clasificadores débiles que por sí solos proporcionan resultados poco robustos, pero que al combinarlos mejoran considerablemente sus prestaciones. Estos clasificadores débiles se distribuyen en grupos, y estos grupos se enlazan formando una cascada, y actuando cada uno sobre las predicciones del anterior, constituyendo de esa forma el clasificador final [31]. Cada uno de estos grupos se denomina una “etapa”, y el clasificador final estará constituido por varias etapas. Las primeras etapas filtran directamente aquellas zonas que son fácilmente identificables como “no personas”, siendo cada etapa sucesiva más restrictiva que la anterior. En función del número de etapas obtendremos unas prestaciones u otras, es decir, a mayor número de etapas filtraremos más las falsas alarmas pero tendremos perderemos también imágenes positivas que no interesaba eliminar. Así lograremos mejorar nuestro algoritmo.

Los clasificadores con un mayor número de reglas débiles a priori consiguen mejores resultados, pero a costa de un elevado coste computacional. Por ello, y dado el contexto de este trabajo, se tendría que hacer un estudio de la viabilidad de utilizar un número determinado de etapas para la clasificación.

Anexos

ANEXO A

Instalar Software

A.1. Instalación de Ubuntu

En caso de no tener instalado Ubuntu es necesario seguir una serie de sencillos pasos:

1. Descargar el CD de instalacion de Ubuntu, el Desktop Cd de su pagina web:

   www.ubuntu.com/download/desktop

2. El archivo descargado es una imagen ISO que se debe grabar en un disco para proceder con la instalación.

3. Arrancar el ordenador desde el CD, para ello reiniciar el equipo con el disco grabado en el lector.

4. Por ultimo se deben ir siguiendo los pasos de la instalación. Como ayuda existe la siguiente página web:

   http://www.guia-ubuntu.org/index.php?title=Instalaci%C3%B3nest%C3%A1ndar

   En ella, aparte de ver los pasos a seguir durante la instalación, también se puede encontrar el Desktop Cd y tutoriales.

A.2. Instalación de Eclipse

1. En primer lugar, debes instalar la plataforma Eclipse – núcleo del IDE – entonces puedes elegir / instalar las extensiones del lenguaje y herramientas necesarias de acuerdo a tu necesidad.

2. Abre un terminal y ejecuta

   -*sudo apt-get install eclipse-platform*

3. Ahora, que está instalada la plataforma central , ya se pueden instalar los plugins de desarrollo de acuerdo a tu necesidad.

   Instalar CDT para C / C + + Desarrollo

4. Para utilizar Eclipse IDE para C / C + + desarrollo , a continuación la manera de instalar CDT (c / c + + de las herramientas de desarrollo) para los paquetes de Eclipse

   -*sudo apt-get install eclipse-cdt*

A.3. Instalación OpenCV

La instalación es muy sencilla, sin embargo puede durar bastante debido a la gran cantidad de paquetes que se tienen que instalar.

1. Actualiza los repositorios:

   Abre el **terminal** y ejecuta el comando:

   -*sudo apt-get update*

   Y a continuación, el siguiente comando:

   -*sudo apt-get upgrade*

2. Instala las dependencias:

   El siguiente paso es instalar las dependencias de OpenCV. Para ello, ejecuta el siguiente comando en tu terminal:

   -*sudo apt-get install build-essential libgtk2.0-dev libjpeg-dev libtiff4-dev libjasper-dev libopenexr-dev cmake python-dev python-numpy python-tk*

libtbb-dev libeigen2-dev yasm libfaac-dev libopencore-amrnb-dev libopencore-amrwb-dev libtheora-dev libvorbis-dev libxvidcore-dev libx264-dev libqt4-dev libqt4-opengl-dev sphinx-common texlive-latex-extra libv4l-dev libdc1394-22-dev libavcodec-dev libavformat-dev libswscale-dev

3. Descarga y descomprime OpenCV

   Hay dos opciones:

   La primera opción: entrar en la página web oficial de descargas de OpenCV, descargar OpenCV 2.4.2 for Linux/Mac en un principio también funciona con OpenCV 2.4.3), se descomprime y entrar a dicho directorio creado desde la terminal.

   La segunda opción (recomendada): ejecuta los siguientes comandos en tu terminal:

   -cd wget http://downloads.sourceforge.net/project/opencvlibrary/opencv-unix/2.4.2/OpenCV-2.4.2.tar.bz2 tar -xvf OpenCV-2.4.2.tar.bz2 cd OpenCV-2.4.2

4. Compila e instala OpenCV

   Ahora hay que asegurarse de estar dentro del directorio de OpenCV y ejecutar los siguientes comandos:

   -mkdir build

   -cd build

   -mkdir build

   -cmake -D WITH_TBB=ON -D BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_V4L=ON -D INSTALL_C_EXAMPLES=ON -D INSTALL_PYTHON_EXAMPLES=ON -D BUILD_EXAMPLES=ON -D WITH_QT=ON -D WITH_OPENGL=ON ..

   -make

   -sudo make install

5. Preparar OpenCV

   Ahora hay que editar el archivo de configuración de OpenCV con *-sudo gedit /etc/ld.so.conf.d/opencv.conf*. Seguramente el archivo esté vacío, escribe /usr/local/lib en él, guárdalo y sal del editor.

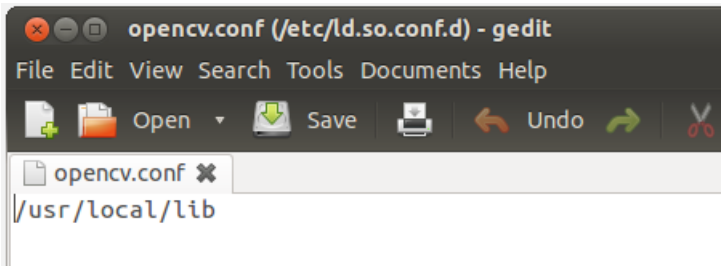


Figura A.1: Ilustración editor.

6. Ahora hay que ejecutar en la terminal:

   -sudo ldconfig

7. Añade el PATH editando el arranque de Bash. Para ello ejecuta:

   -sudo gedit /etc/bash.bashrc

   Al final del archivo añade:

   -PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/local/lib/pkgconfig export PKG_CONFIG_PATH

   Guarda el archivo y sal del editor. Por ultimo, REINICIA EL ORDENADOR y tendrás OpenCV finalmente instalado y preparado para desarrollar en él.

A.4. Instalación de SVM^{light}

Para instalar SVM^{light} hay que seguir los siguientes pasos:

1. Crear un directorio llamado svm_light:

   -mkdir svm_light

2. Mover el archivo descargado svm_light.tar.gz al directorio anterior y descomprimir:

   -gunzip -c svm_light.tar.gz | tar xvf

3. Ahora ejecutar:

   -make or make all

4. El sistema crea dos archivos ejecutables:

   -svm_learn(learning module)

-svm_classify (classification module

Bibliografía

[1] N. Dalal, *Finding people in images and videos*. PhD thesis, Institut National Polytechnique de Grenoble-INPG, 2006.

[2] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, pp. 886–893, IEEE, 2005.

[3] J. Wu, C. Geyer, and J. M. Rehg, "Real-time human detection using contour cues," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 860–867, IEEE, 2011.

[4] Y. Pang, Y. Yuan, X. Li, and J. Pan, "Efficient hog human detection," *Signal Processing*, vol. 91, no. 4, pp. 773–781, 2011.

[5] Y.-T. Chen and C.-S. Chen, "Fast human detection using a novel boosted cascading structure with meta stages," *Image Processing, IEEE Transactions on*, vol. 17, no. 8, pp. 1452–1464, 2008.

[6] D. Schreiber, C. Beleznai, and M. Rauter, "Gpu-accelerated human detection using fast directional chamfer matching," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2013 IEEE Conference on*, pp. 614–621, IEEE, 2013.

[7] H. Nanda and L. Davis, "Probabilistic template based pedestrian detection in infrared videos," in *IEEE Intelligent Vehicle Symposium*, vol. 1, pp. 15–20, 2002.

[8] H. Cheng, N. Zheng, and J. Qin, "Pedestrian detection using sparse gabor filter and support vector machine," in *Intelligent Vehicles Symposium, 2005. Proceedings. IEEE*, pp. 583–587, IEEE, 2005.

[9] F. Suard, V. Guigue, A. Rakotomamonjy, and A. Benshrair, "Pedestrian detection using stereo-vision and graph kernels," in *Intelligent Vehicles Symposium, 2005. Proceedings. IEEE*, pp. 267–272, IEEE, 2005.

[10] C. Curio, J. Edelbrunner, T. Kalinke, C. Tzomakas, and W. Von Seelen, "Walking pedestrian recognition," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 1, no. 3, pp. 155–163, 2000.

[11] W. R. Schwartz, A. Kembhavi, D. Harwood, and L. S. Davis, "Human detection using partial least squares analysis," in *Computer vision, 2009 IEEE 12th international conference on*, pp. 24–31, IEEE, 2009.

[12] L. Zhao and C. E. Thorpe, "Stereo-and neural network-based pedestrian detection," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 1, no. 3, pp. 148–154, 2000.

[13] T. J. <thorsten@joachims.org>Cornell University Department of Computer Science, "`http://svmlight.joachims.org/`," 14.08.2008.

[14] L. G. Roberts, *MACHINE PERCEPTION OF THREE-DIMENSIONAL soups*. PhD thesis, Massachusetts Institute of Technology, 1963.

[15] K. K. Pingle, J. A. Singer, and W. M. Wichman, "Computer control of a mechanical arm through visual input.," in *IFIP Congress (2)*, pp. 1563–1569, 1968.

[16] R. M. Haralick, K. Shanmugam, and I. H. Dinstein, "Textural features for image classification," *Systems, Man and Cybernetics, IEEE Transactions on*, no. 6, pp. 610–621, 1973.

[17] A. P. Witkin, "Recovering surface shape and orientation from texture," *Artificial intelligence*, vol. 17, no. 1, pp. 17–45, 1981.

[18] J. E. Mayhew and J. P. Frisby, "Psychophysical and computational studies towards a theory of human stereopsis," *Artificial Intelligence*, vol. 17, no. 1, pp. 349–385, 1981.

[19] B. K. Horn and B. G. Schunck, "Determining optical flow," in *1981 Technical Symposium East*, pp. 319–331, International Society for Optics and Photonics, 1981.

[20] L. Kitchen and A. Rosenfeld, "Gray-level corner detection," *Pattern recognition letters*, vol. 1, no. 2, pp. 95–102, 1982.

[21] D. Marr and A. Vision, "A computational investigation into the human representation and processing of visual information," *WH San Francisco: Freeman and Company*, 1982.

[22] C. Schlegel, J. Illmann, H. Jaberg, M. Schuster, and R. Wörz, "Vision based person tracking with a mobile robot.," in *BMVC*, pp. 1–10, 1998.

[23] S. J. McKenna, S. Jabri, Z. Duric, and H. Wechsler, "Tracking interacting people," in *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*, pp. 348–353, IEEE, 2000.

[24] B. Leibe, E. Seemann, and B. Schiele, "Pedestrian detection in crowded scenes," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, pp. 878–885, IEEE, 2005.

[25] N. Dalal, B. Triggs, and C. Schmid, "Human detection using oriented histograms of flow and appearance," in *Computer Vision–ECCV 2006*, pp. 428–441, Springer, 2006.

[26] A. Ess, B. Leibe, and L. Van Gool, "Depth and appearance for mobile scene analysis," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pp. 1–8, IEEE, 2007.

[27] L. Spinello and R. Siegwart, "Human detection using multimodal and multidimensional features," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 3264–3269, IEEE, 2008.

[28] T. Leyvand, C. Meekhof, Y.-C. Wei, J. Sun, B. Guo, *et al.*, "Kinect identity: Technology and experience," *Computer*, vol. 44, no. 4, pp. 94–96, 2011.

[29] NITE, "`http://www.primesense.com/nite`."

[30] OpenNI, "`https://github.com/OpenNI/OpenNI`."

[31] P. Viola and M. J. Jones, "Robust real-time face detection," *International journal of computer vision*, vol. 57, no. 2, pp. 137–154, 2004.

[32] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2, pp. 1150–1157, Ieee, 1999.

[33] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision–ECCV 2006*, pp. 404–417, Springer, 2006.

[34] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 10, pp. 1615–1630, 2005.

[35] J. Canny, "A computational approach to edge detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, no. 6, pp. 679–698, 1986.

[36] M. A. Hearst, S. Dumais, E. Osman, J. Platt, and B. Scholkopf, "Support vector machines," *Intelligent Systems and their Applications, IEEE*, vol. 13, no. 4, pp. 18–28, 1998.

[37] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, "An efficient boosting algorithm for combining preferences," *The Journal of machine learning research*, vol. 4, pp. 933–969, 2003.

[38] R. J. Elliott, L. Aggoun, and J. B. Moore, *Hidden Markov Models*. Springer, 1995.

[39] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, p. 14, California, USA, 1967.

[40] T. Kohonen, *Self-organizing maps*, vol. 30. Springer, 2001.

[41] S. Abney, "Semisupervised learning for computational linguistics."

[42] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[43] INRIA, "`pascal.inrialpes.fr/data/human/`."

[44] P. Viola and M. Jones, "Fast and robust classification using asymmetric adaboost and a detector cascade," *Advances in Neural Information Processing Systems*, vol. 2, pp. 1311–1318, 2002.

[45] B. W. Silverman and M. C. Jones, "E. fix and jl hodges (1951): an important contribution to nonparametric discriminant analysis and density estimation: commentary on fix and hodges (1951)," *International Statistical Review/Revue Internationale de Statistique*, pp. 233–238, 1989.

[46] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[47] W. T. Freeman and M. Roth, "Orientation histograms for hand gesture recognition," in *International Workshop on Automatic Face and Gesture Recognition*, vol. 12, pp. 296–301, 1995.

[48] J. Malik, S. Belongie, T. Leung, and J. Shi, "Contour and texture analysis for image segmentation," *International journal of computer vision*, vol. 43, no. 1, pp. 7–27, 2001.

[49] G. Wang, "A survey on training algorithms for support vector machine classifiers," in *Networked Computing and Advanced Information Management, 2008. NCM'08. Fourth International Conference on*, vol. 1, pp. 123–128, IEEE, 2008.

[50] V. Vapnik and S. Kotz, *Estimation of dependences based on empirical data*. Springer, 2006.

[51] J. Platt *et al.*, "Sequential minimal optimization: A fast algorithm for training support vector machines," 1998.

[52] T. Joachims, "Making large scale svm learning practical," 1999.

[53] E. Osuna, R. Freund, and F. Girosi, "An improved training algorithm for support vector machines," in *Neural Networks for Signal Processing [1997] VII. Proceedings of the 1997 IEEE Workshop*, pp. 276–285, IEEE, 1997.

[54] H. Ocak, "A medical decision support system based on support vector machines and the genetic algorithm for the evaluation of fetal well-being," *Journal of medical systems*, vol. 37, no. 2, pp. 1–9, 2013.

[55] P.-c. Li and S.-h. Xu, "Support vector machine and kernel function characteristic analysis in pattern recognition," *Computer Engineering and Design*, vol. 26, no. 2, pp. 302–304, 2005.

[56] H.-L. Chen, B. Yang, J. Liu, and D.-Y. Liu, "A support vector machine classifier with rough set-based feature selection for breast cancer diagnosis," *Expert Systems with Applications*, vol. 38, no. 7, pp. 9014–9022, 2011.

[57] MIT, "`http://cbcl.mit.edu/software-datasets/PedestrianData.html`."