Universidad Autónoma de Madrid
Escuela Politécnica Superior

– Trabajo fin de Grado –

DETECCIÓN DE PERSONAS EN TIEMPO REAL

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Patricia Marín Belinchón

Mayo 2014

DETECCIÓN DE PERSONAS EN TIEMPO REAL

AUTORA: Patricia Marín Belinchón

TUTOR: Álvaro García Martín
PONENTE: José M. Martínez Sánchez

Video Processing and Understanding Lab - UAM
Dpto. de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Mayo 2014

Trabajo parcialmente financiado por el gobierno español bajo el proyecto
TEC2011-25995 (EventVideo) (2012-2014)

Resumen y Palabras Clave

Resumen

Este proyecto tiene como objetivo principal la implementación de varios algoritmos de detección de personas sobre una plataforma de análisis de vídeo propietaria denominada DiVA, adaptando su funcionalidad de tal manera que además de ejecutarse sobre vídeos *off-line*, puedan funcionar también sobre videocámaras en tiempo real.

Adicionalmente se han desarrollado demostradores para cada uno de estos algoritmos en forma de interfaces gráficas. De esta forma se permite la interactividad entre el usuario y el algoritmo y la comprobación del funcionamiento del mismo de una manera más sencilla e intuitiva.

Para llevar a cabo este trabajo y durante el desarrollo del mismo, se han estudiado, implementado y evaluado cada uno de los detectores de personas con los que se ha trabajado. Estos detectores han sido cuatro: Por un lado, HOG y Latent SVM, que realizan una búsqueda exhaustiva para sus detecciones y, por el otro, Fusion y Edge, que utilizan segmentación previa a la clasificación, gracias a lo cual funcionan en tiempo real.

Durante el desarrollo de este trabajo se ha estudiado en profundidad el funcionamiento de todos estos algoritmos de detección de personas para, finalmente, llevar a cabo un análisis de resultados en el que se evalúa el coste computacional de cada uno de los detectores desarrollados.

Palabras Clave

Detección de personas, vídeo-vigilancia, modelo, *"cell"*, histograma, distribución de gradientes, segmentación, extracción de objetos, *"blob"*, detector de borde, clasificador y rendimiento.

Abstract and Keywords

Abstract

The main goal of this project is the implementation of several people detection algorithms on a propietary video analysis platform called DiVA, adapting its functionality so that it can run with textitoff-line videos, as well as in real time.

Furthermore, demonstrators have been developed for each of these algorithms as graphic interfaces. In this way, interactivity between the user and the algorithm is allowed and testing its performance becomes a simpler and more intuitive task.

To carry out this work and during its development, each of the selected people detectors have been studied, implemented and evaluated. These are four, namely: First, HOG and Latent SVM, which perform an exhaustive search for detection and, secondly, Fusion and Edge, which use pre-segmentation classification, thus allowing real-time detection.

During this work the performance of all these algorithms for people detection have been studied in depth to finally analyse the results, evaluating the computational cost of each implemented detector.

Keywords

People detection, video surveillance, model, cell, histogram, gradient distribution, segmentation, object extraction, blob, edge detector, classifier and performance.

Agradecimientos

Para empezar me gustaría agradecer a mi tutor, Álvaro García, por haberme dado la oportunidad de realizar este trabajo y por su ayuda y asesoramiento. Quiero dar las gracias también a los jefes del grupo, Chema y Jesús, cuya implicación y profesionalidad son dignas de admiración.

A todas las personas del laboratorio, por el buen ambiente que crean y por la comodidad que supone trabajar con ellos día a día. Sobre todo me gustaría hacer especial mención a una de ellas, Carlos Sánchez, por haberme ayudado de forma inmediata durante todo el curso cada vez que lo he necesitado.

Quiero agradecer de igual forma a todos mis compañeros el haber compartido conmigo estos años de carrera, por haber formado una pequeña familia que se ha convertido en una gran promoción. Sin embargo, mención aparte se merecen Alberto, por su grata compañía diaria que ha hecho de mi lugar de trabajo un sitio más agradable; Marta y Ana, por estar en todos los buenos y malos momentos que me llevo de estos años y por regalarme su valiosa amistad; y sin duda alguna Sara, por estar a mi lado desde la niñez, por pertenecer a todos y cada uno de los momentos importantes que recuerdo y sobre todo por enseñarme que existe la amistad verdadera para toda la vida.

También quiero mencionar al resto de mis amigos, todos los que de una u otra forma son parte responsable de mis alegrías desde hace mucho tiempo.

Y por supuesto no puedo olvidarme de mi novio Pedro, por darme su apoyo incondicional, las fuerzas y la energía que he necesitado cada día, por brindarme su ayuda directa con este trabajo, por estar siempre presente a pesar de la distancia y, en definitiva, por demostrarme que no pude elegir mejor.

Pero si hay personas a las que tenga que dar las gracias por encima de todo, ésas son mis padres, mi hermana y mis yayos, por confiar en mí ciegamente, por ser mi pilar básico y por haber hecho de mí la persona que soy. Ellos son la principal razón de que hoy esté escribiendo estas líneas.

A todos, os dedico este trabajo. Gracias.

Patricia Marín Belinchón
Mayo 2014

Índice general

Índice de figuras

Índice de tablas

Glosario de acrónimos

- **BLOB**: Binary Large Object
- **CPU**: Central Processing Unit
- **CVSG**: Chroma Video Segmentation Ground-Truth
- **DIVA**: Distributed Video Analysis Framework
- **FPS**: Frames Per Second
- **HOG**: Histogram of Oriented Gradients
- **OpenCV**: Open Source Computer Vision
- **PDds**: Person Detection Dataset
- **RAM**: Random Access Memory
- **SVM**: Support Vector Machine
- **VPU**: Video Processing and Understanding Lab

1 Introducción

1.1. Motivación

En los últimos años el procesado de señal ha estado en constante evolución. De manera más concreta, se han llevado a cabo enormes esfuerzos y progresos en el procesado de imagen y vídeo digital debido a su gran utilidad dentro de la sociedad en la que vivimos actualmente.

Considerando la gran demanda que existe en el área de los sistemas de seguridad, una de las mayores líneas de investigación es la de vídeo-vigilancia. La necesidad de brindar a las personas y a sus pertenencias la seguridad que necesitan en el día a día es lo que explica el enorme desarrollo y expansión de los sistemas de vídeo-vigilancia actuales.

Dentro del área de investigación en el procesado de imagen y vídeo digital, existe una gran cantidad de algoritmos utilizados en seguridad para detección de movimiento, objetos, sucesos, etc [3, 4]. La detección automática de personas en secuencias de vídeo pertenece a este grupo de algoritmos y actualmente es un problema complejo con múltiples aplicaciones [5], no sólo en vídeo-vigilancia, sino también en diferentes áreas como inteligencia artificial, videojuegos, etc.

La complejidad del problema de detección de personas está basada principalmente en la dificultad que plantea el modelado de las mismas. Esto es debido a su gran variabilidad en la apariencia física, poses, movimientos, puntos de vista e interacciones entre diferentes personas y objetos.

Como resultado de estas reflexiones, la motivación principal de este trabajo es lograr una solución práctica y efectiva para diferentes detectores de personas, adaptándolos y evaluándolos sobre secuencias de vídeo captadas con cámaras que funcionen en tiempo real, ya que éste es el factor esencial en las aplicaciones de vídeo-vigilancia tan demandadas en áreas de seguridad.

1.2. Objetivos y enfoque

Establecida la motivación de este trabajo, establecemos los objetivos partiendo de propósitos parciales que finalmente permitirán alcanzar el objetivo global de una manera progresiva y adecuada. El desglose de los objetivos es el siguiente:

1. Estudio detallado del estado del arte.
    - Conocer en qué consiste la detección de personas, analizando los posibles enfoques

que toman los detectores para llevar a cabo su propósito, así como las ventajas y los inconvenientes de cada uno de ellos.

- Profundizar en el conocimiento de los detectores Fusion [6], Edge [7], HOG [1] y Latent SVM [2], estudiando todas sus particularidades, arquitectura y modo de funcionamiento.

2. Aprendizaje de herramientas y bibliotecas.

   Utilizar las herramientas y bibliotecas necesarias para el diseño y la implementación que permiten llevar a cabo este trabajo: OpenCV [8], DiVA [9] y Biblioteca QT [10].

3. Adaptación de algoritmos de detección de personas.

   Transformación e integración de los códigos de los algoritmos estudiados en el estado del arte utilizando el lenguaje C++ para que funcionen sobre una nueva plataforma de análisis de vídeo denominada DiVA (*Distributed Video Analysis Framework*), de tal manera que puedan ser ejecutados no sólo sobre vídeos de prueba sino también sobre videocámaras en tiempo real.

4. Creación de interfaces gráficas.

   Adquirir los conocimientos necesarios para programar una interfaz gráfica para cada uno de los algoritmos implementados, utilizando la biblioteca QT y programación en C++.

5. Analizar los resultados obtenidos.

   Comparar el rendimiento y la eficacia de cada uno de los detectores de personas en diferentes ámbitos, para conocer sus ventajas y sus limitaciones.

Con todo ello la finalidad que se persigue no es otra que la de conseguir un entorno de trabajo común, práctico y efectivo de los algoritmos de detección de personas **Fusion** [6], **Edge** [7], **HOG** [1] y **Latent SVM** [2].

1.3. Organización de la memoria

La memoria de este trabajo se compone de los siguientes capítulos:

- **Capítulo 1**: Introducción y motivación del proyecto.
- **Capítulo 2**: Estado del arte de la detección de personas. Estudio de las características y peculiaridades de cuatro algoritmos concretos: Fusion, Edge, HOG y Latent SVM. Introducción del uso del sistema DiVA y de la biblioteca QT.
- **Capítulo 3**: Diseño y Desarrollo. Descripción de la adaptación y programación de cada uno de los detectores estudiados en el estado del arte para que funcionen en el entorno de DiVA (*Distributed Video Analysis Framework*).
- **Capítulo 4**: Demostradores. Creación de las interfaces gráficas para cada uno de los detectores mediante la biblioteca QT.
- **Capítulo 5**: Análisis de resultados. Análisis comparativo de la eficiencia en términos de coste computacional de los detectores Fusion, Edge, HOG y Latent SVM para evaluar su funcionamiento en tiempo real y comparar los resultados con el estado del arte.
- **Capítulo 6**: Conclusiones obtenidas tras el análisis de resultados del trabajo presentado. Problemas pendientes y trabajo futuro.
- **Referencias y Glosario.**

2 Estado del arte

2.1. Introducción a la detección de personas

La detección de personas consiste principalmente en, en primer lugar, el diseño y el entrenamiento de un modelo de persona basado en parámetros característicos de la misma (movimiento, dimensiones, silueta, etc), y en segundo lugar, el ajuste de este modelo a los posibles objetos de una escena y determinar si son personas o no. Sólo los objetos candidatos que se ajusten al modelo serán detectados o clasificados como una persona.

La mayor parte de los enfoques están basados en **información acerca de la apariencia**, aunque algunos de ellos añaden robustez a la detección incorporando información sobre el movimiento mediante algoritmos de seguimiento [1, 6, 11, 12, 13, 14, 15, 16, 17].

Dentro de esta gran cantidad de posibilidades existe una pequeña parte de enfoques basados únicamente en **información de movimiento** [18, 19], cuyas principales ventajas son la independencia en la variabilidad en la apariencia y la baja complejidad que presentan habitualmente. Sin embargo y como contrapartida, suelen obtener peores resultados.

Los métodos basados en la apariencia [1, 6, 13, 14, 16, 17] pueden ser clasificados de acuerdo al modelo de persona utilizado. Los métodos basados en modelos simplificados de persona (sólo una región o la forma) normalmente presentan menor dificultad pero no soportan variaciones de postura. Sin embargo, los métodos basados en modelos de persona más complejos, los cuales normalmente tienen una mayor dificultad, sí son capaces de lidiar con estas variaciones [11, 12, 15]. Otra ventaja añadida es que realizan la decisión final combinando múltiples evidencias, de tal manera que suelen ser más fiables que los métodos basados en modelos más simples.

2.2. Clasificación de algoritmos de detección de personas

Basándonos en la idea de un sistema real de vídeo-vigilancia, los algoritmos de detección de personas se pueden clasificar en dos grupos principales dependiendo de si trabajan en tiempo real o no, separando los enfoques de cada caso en dos sistemas claramente diferenciados:

- Los **sistemas que operan en tiempo real** normalmente seleccionan regiones candidatas a ser detectadas mediante segmentación on-line de la imagen. Algunos utilizan la sustracción del fondo [17, 20] y otros la visión estéreo o la información 3D [21, 22]. Además, debido a las restricciones computacionales, estos sistemas normalmente emplean modelos de personas simplificados.

- Los **sistemas que no operan en tiempo real** [1, 11, 14, 15, 23, 24, 25] seleccionan estas regiones candidatas a la detección escaneando la imagen completa a varias escalas y rotaciones. En estos casos los modelos de persona deben ser complejos para poder obtener una mejor clasificación, ya que, debido a la búsqueda exhaustiva que realizan, se enfrentan a una gran cantidad de objetos candidatos a ser personas. El escaneo y el uso de estos modelos mejoran la tasa de detección pero los costes computacionales son demasiado altos para permitir un procesado en tiempo real.

Algunos sistemas, como por ejemplo [26], intentan acelerar los métodos utilizados en sistemas que no trabajan en tiempo real pero manteniendo un nivel de precisión similar, consiguiendo así una mejor aproximación al trabajo en tiempo real.

2.3. Principales Detectores

2.3.1. Detectores VPU

Estos sistemas (Fusion [6] y Edge [7]) incluyen las etapas de procesado de un sistema canónico de análisis de vídeo automatizado para detección de personas [4, 27], las cuales podemos ver en la Figura 2.1. Ambos detectores usan la segmentación como método de extracción de objetos para obtener de forma rápida los objetos candidatos a ser personas.

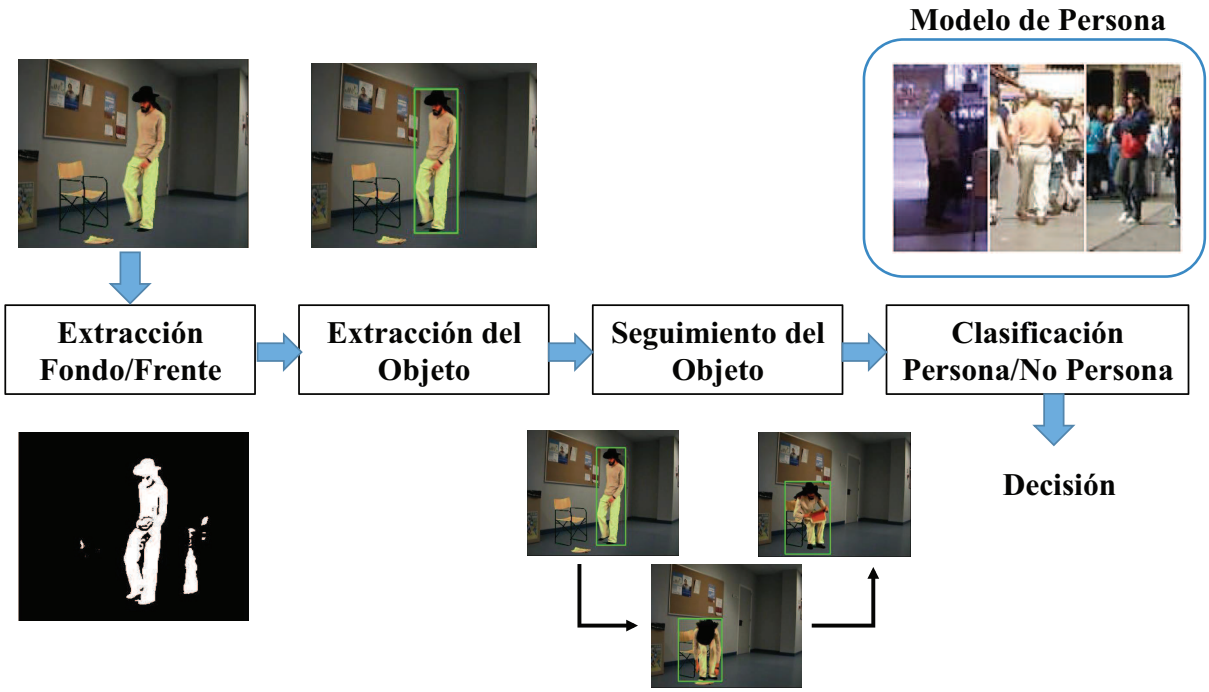


Figura 2.1: Arquitectura global de los sistemas Fusion y Edge.

- **Extracción frente/fondo.** Es una técnica usada habitualmente para detección de movimiento y segmentación. Tiene como objetivo segmentar las regiones que se corresponden con objetos móviles de la imagen (frente), basándose para ello en [28]. Las siguientes etapas son muy dependientes de los resultados obtenidos en este proceso.
- **Extracción de objetos.** Después de la segmentación se aplican operaciones morfológicas para reducir el ruido de la máscara de la imagen resultante y mejorar la extracción de

objetos [4]. Una vez extraído el objeto, se aplica un análisis de componentes conexos [29]. Únicamente los objetos extraídos en esta etapa son analizados en las próximas. Cada objeto se define con un blob, conjunto de píxeles similares entre ellos y a su vez contrastados con el resto de los píxeles presentes en la imagen. Se define por un rectángulo con coordenada (x,y) en uno de sus vértices, ancho y alto.

- **Seguimiento de objetos.** El objetivo es generar la trayectoria de un objeto en el tiempo localizando su posición en cada imagen de la secuencia de vídeo en la que aparece. Se usa un algoritmo simple de seguimiento basado en el filtro de Kalman [30] y que genera las trayectorias de los blobs entre imágenes consecutivas usando la información de color, dimensiones de los blobs y posiciones de los mismos.

- **Clasificación de objetos.** Consiste en un reconocimiento estándar de patrones. Este proceso compara los modelos de objetos entrenados previamente con los modelos generados a partir de la imagen o la secuencia y realiza la decisión final basándose en su similitud.

2.3.1.1. Fusion

Fusion [6] es un detector que fusiona evidencias derivadas de tres detectores de personas rápidos e independientes. Dos de ellos usan técnicas basadas en la silueta [12, 16] y el tercero una relación de aspecto sencilla. Los tres detectores se aplican por separado a cada blob detectado como frente mediante un esquema de extracción de fondo propuesto en [31].

De cada detector, después de haberlo aplicado a un blob específico, se extraen una serie de características relacionadas con personas. La presencia de una persona dentro de un blob se detecta según la medida en la que se evidencien cada una de estas características extraídas. Por último, las evidencias correspondientes a los tres detectores con respecto al mismo blob se fusionan en una evidencia combinada, la cual será umbralizada para poder tomar decisiones acerca de si el blob corresponde a una persona o no.

- El **primer detector** en fusionarse está basado en una única característica, la relación de aspecto del blob, E_a, definida como el cociente entre la anchura w y la altura h.

$$E_a = E_{\mu_a,\sigma_a}(\frac{w}{h}) \tag{2.1}$$

  Donde μ y σ son la media y la desviación estándar de las relaciones de aspecto del conjunto de entrenamiento. En este trabajo se ha utilizado μ=0.3 y σ=0.2.

- El **segundo detector** aplicado a cada blob se basa en el algoritmo propuesto en [16], el cual calcula iterativamente la mayor elipse contenida en la región de frente asociada al blob. El proceso iterativo finaliza cuando la elipse está completamente contenida en la región de frente del blob o cuando el número de iteraciones alcanza un máximo predefinido. En este detector se evalúan tres características:

  - E_{e1}: relación entre el número de iteraciones y el máximo posible de ellas.
  - E_{e2}: porcentaje de puntos de la elipse que se quedan fuera del blob, en caso de que los hubiese.
  - E_{e3}: relación de aspecto de la elipse.

  La evidencia final E_e asociada a este detector resulta ser la media de estas tres evidencias parciales.

$$E_e = \frac{E_{e1} + E_{e2} + E_{e3}}{3} \tag{2.2}$$

- El **tercer detector** se basa en el algoritmo Ghost propuesto en [12], que aproxima el contorno del blob al de un polígono cerrado. Es el más complejo y robusto de los tres detectores. También se evalúan tres características para este detector basadas en ese polígono:

  - E_{g1}: número de puntos del polígono que captura la complejidad de la estructura del contorno.
  - E_{g2}: relación entre la cantidad de vértices convexos y no convexos que normalmente es balanceada para una persona.
  - E_{g3}: inverso del número de vértices que pertenecen a la parte superior del polígono según el procedimiento descrito en [12].

  La evidencia final (E_g) asociada a este detector es definida como la media de las tres evidencias anteriores, siempre y cuando las dos últimas superen un umbral ρ=0.6.

$$E_g = \frac{E_{g1} + H(E_{g2} - \rho)E_{g2} + H(E_{g3} - \rho)E_{g3}}{1 + H(E_{g2} - \rho) + H(E_{g3} - \rho)} \tag{2.3}$$

  Donde H(x) es la función escalón de Heaviside.

La evidencia global total (E) sobre un blob determinado para decidir si corresponde o no a una persona se obtiene promediando las evidencias proporcionadas por los tres detectores, teniendo en cuenta que las pertenecientes al primer y segundo detector sólo se toman en cuenta si están por encima del umbral de relevancia predefinido ρ.

$$E = \frac{E_g + H(E_a - \rho)E_a + H(E_e - \rho)E_e}{1 + H(E_a - \rho) + H(E_e - \rho)} \tag{2.4}$$

Por último, el blob analizado será clasificado como una persona si la evidencia combinada final es superior a un umbral de decisión predefinido τ (en este caso τ=0.75).

Fusion es un algoritmo basado en modelos simples y rápidos, pero por ello sus resultados no siempre son acertados. Además, depende en gran medida de la segmentación previa, lo cual se puede convertir en un problema.

2.3.1.2. Edge

Este detector de personas [7] se basa en el algoritmo propuesto en [15], el cual propone un método de detección de personas en escenas llenas de gente, pero trabajando únicamente con imágenes estáticas. La idea principal consiste en identificar las características de bordes para cada parte del cuerpo y generar, por tanto, cuatro modelos de bordes: cuerpo, cabeza, torso y piernas. En particular, se propone caracterizar los bordes mediante Edgelets, definidos como un conjunto de desplazamientos y orientaciones que describan dicho borde. La imagen se escanea con cuatro detectores de borde independientes previamente entrenados, como se puede apreciar en la Figura 2.2 .

La fase de entrenamiento es realizada utilizando el algoritmo Real Adaboost [32] y una estructura en cascada anidada [33]. Las respuestas de cada detector por partes se combinan para obtener un modelo de probabilidad conjunta. Este algoritmo soporta cambios de pose y cambios de puntos de vista de la cámara.

El algoritmo del que se parte [15] está destinado a imágenes estáticas y búsqueda exhaustiva, por esta razón los modelos de personas deben ser complejos para ser capaces de efectuar una correcta clasificación. Además, el entrenamiento está orientado a la reducción de la tasa de

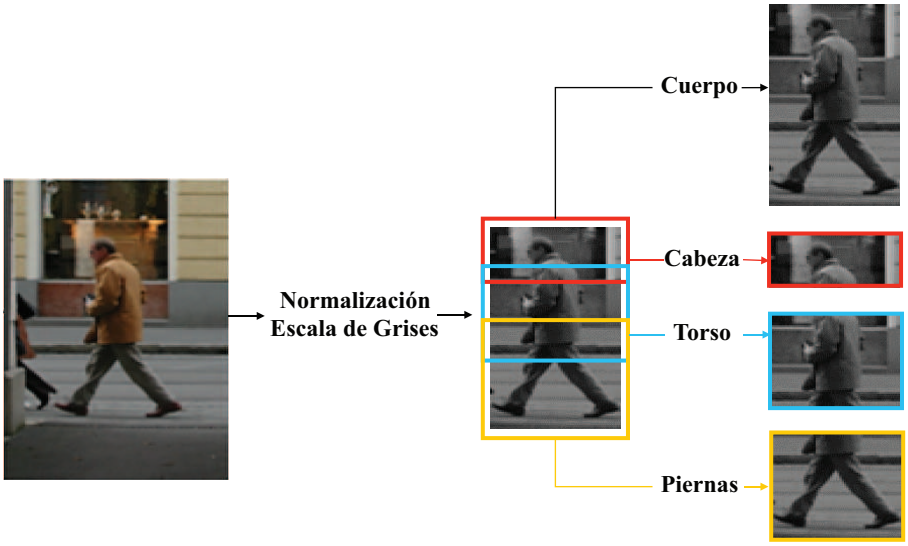


Figura 2.2: Segmentación de las partes del cuerpo en el detector Edge.

falsos positivos, lo cual incrementa sustancialmente el tiempo de procesado. Para conseguir un algoritmo más rápido, el detector Edge [7] únicamente procesa objetos en movimiento detectados en etapas previas en lugar de realizar un escaneado exhaustivo (Figura 2.1). Además simplifica el modelo de cada parte del cuerpo y, como consecuencia, también se simplifica el modelo final de persona, reduciendo el tiempo de detección.

Se construye un clasificador débil [15] para cada borde característico y después se utiliza el algoritmo de AdaBoost [32] para convertirlos en clasificadores fuertes. Para reducir el coste computacional e identificar los bordes más característicos de cada sección del cuerpo, se entrena de forma iterativa el mejor clasificador correspondiente a cada parte del cuerpo y se seleccionan los mejores 100 bordes asociados. Finalmente el algoritmo AdaBoost en cascada [34] es usado para el aprendizaje de cada detector. Esta fase de entrenamiento no sólo está enfocada a la reducción de la tasa de falsos positivos sino también a la obtención de resultados precisos.

Las mejoras son, por lo tanto, el uso de un ranking sobre los mejores bordes para cada parte del cuerpo y, por otro lado, lograr que la fase de entrenamiento no sólo esté centrada en la reducción de los falsos positivos, sino también en conseguir una buena precisión en los resultados.

Tal y como observamos en la Figura 2.1, sólo se clasifican los objetos que han sido detectados después de tres etapas previas. Se normaliza cada blob de la imagen y luego, mediante clasificadores en cascada, se generan los cuatro modelos correspondientes a las distintas partes del cuerpo.

La evidencia final de que el blob analizado es una persona se obtiene promediando las cuatro evidencias independientes proporcionadas por los cuatro modelos de las distintas partes del cuerpo. La detección final de personas, teniendo en cuenta que utiliza un modelo simplificado y un menor número de clasificadores, es menos compleja y el sistema completo es más rápido, sin perder la buena precisión de los resultados.

Este detector, al igual que Fusion, también es completamente dependiente de la segmentación. Sin embargo, al añadir complejidad en sus modelos, obtiene buenos resultados sin dejar de ser un algoritmo rápido.

2.3.2. HOG

Este método [1, 35] se basa en la evaluación de histogramas locales y normalizados de las orientaciones de los gradientes de una imagen en una cuadrícula densa. La idea básica es que la apariencia y la forma de los objetos se pueden caracterizar tanto mediante la distribución de los gradientes de intensidad local como por las direcciones de bordes, incluso cuando se desconoce la ubicación de cualquiera de ellos. La Figura 2.3 nos muestra un ejemplo de ello.

Figura 2.3: (a) Imagen promedio del gradiente en los ejemplos de entrenamiento. (b) Imagen de prueba. (c) Cálculo de su descriptor HOG. Imágenes extraídas de [1].

En la práctica se implementa dividiendo cada ventana de la imagen en pequeñas regiones espaciales (cells), acumulando en cada una de ellas un histograma de las direcciones de gradiente o de las orientaciones de borde sobre los píxeles que se encuentran en cada cell.

Para una mejor invariancia a la iluminación y sombras, es útil normalizar el contraste de zonas locales antes de operar sobre ellas. Esto se realiza acumulando la energía resultante de los histogramas locales sobre cualquier región de bloques mayor y usando los resultados para normalizar todos los cells en el bloque. Nos referiremos a los bloques de descriptores normalizados como descriptores HOG (Histogram of Oriented Gradient). La ventana de detección es dividida en una cuadrícula con un grado de solapamiento de bloques de los que se extraen los vectores de características HOG, que serán clasificados mediante un SVM (*Support Vector Machine*) lineal.

En definitiva, la secuencia de pasos que se siguen en el algoritmo para la detección de personas se muestran en la Figura 2.4.

HOG captura los bordes o estructuras de gradientes características de forma local y lo hace con un grado de invariancia a transformaciones fácilmente controlable: traslaciones o rotaciones apenas varían el resultado. La ventana de detección, además, se escanea de forma exhaustiva en cualquier posición y escala.

La principal ventaja de este detector es su independencia de la segmentación. Aunque además añade modelos de personas con una complejidad mayor y por lo tanto obtiene muy buenos resultados, su coste computacional es alto y por lo tanto se convierte en un algoritmo más lento.

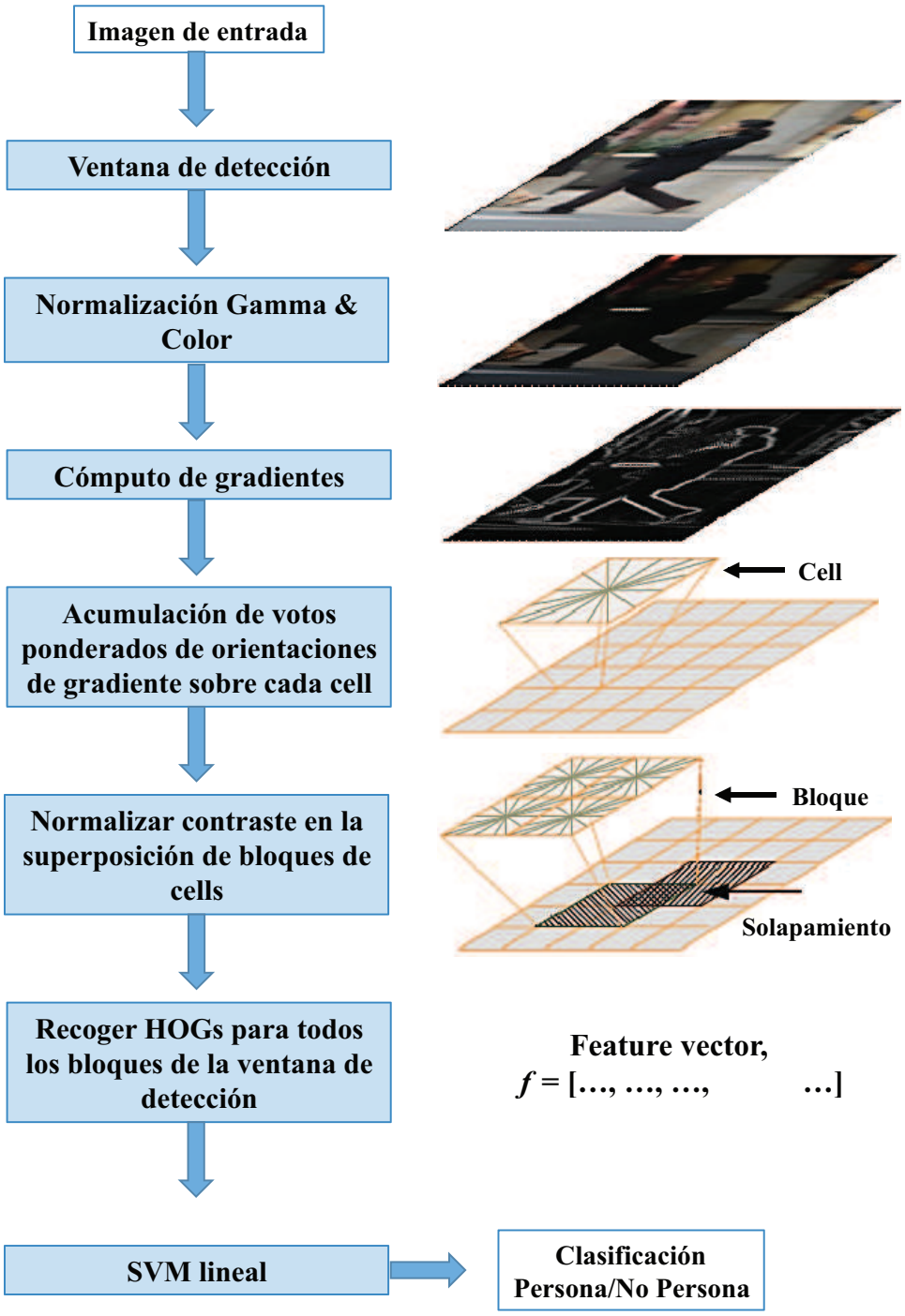


Figura 2.4: Visión global de la estructura del detector HOG.

2.3.3. Latent SVM

Latent SVM [2] es un detector que usa un modelo basado en un filtro de raíz similar al HOG [1] más un conjunto de filtros por partes y modelos de deformación asociados a ellos. La Figura 2.5 muestra este modelo para la categoría de personas. Tanto las puntuaciones del filtro de raíz como las de los filtros por partes están definidas por el producto escalar entre un filtro que sea el resultado de un conjunto de pesos y una subventana de la pirámide de características calculada a partir de la imagen de entrada.

Los filtros por partes capturan las características al doble de la resolución espacial a la que

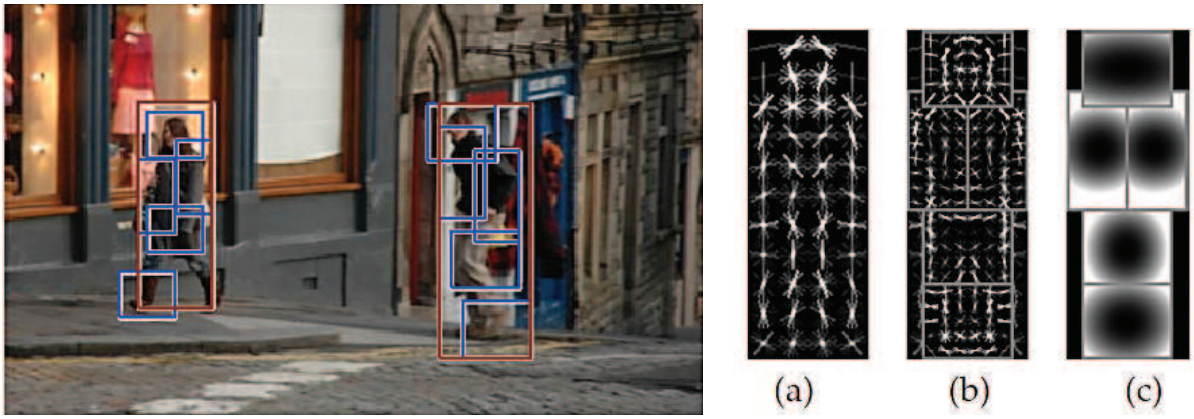


Figura 2.5: Detecciones obtenidas con un único modelo de persona. El modelo está definido por un filtro de raíz grueso(a), varios filtros por partes de mayor resolución(b) y un modelo espacial para la ubicación de cada parte relativa a la raíz(c). Imágenes extraídas de [2].

se capturan las características con el filtro de raíz. De esta manera se modela la apariencia visual a múltiples escalas, tal y como vemos en la Figura 2.6 .

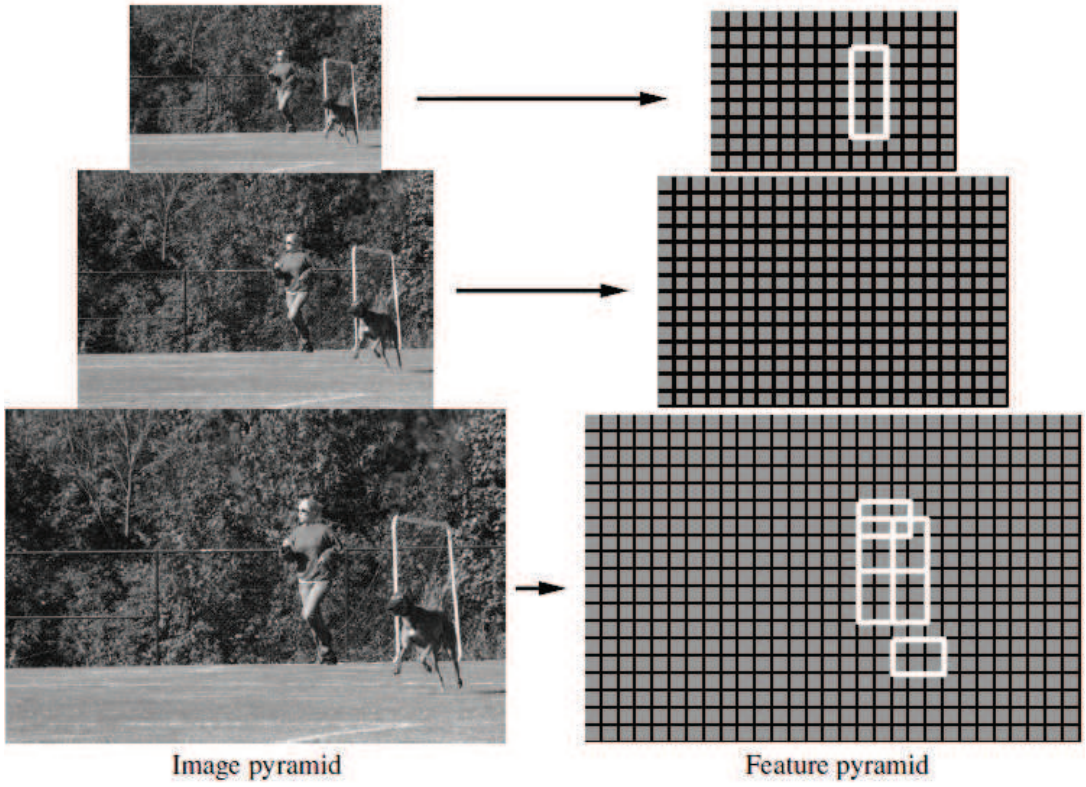


Figura 2.6: Pirámide de imágenes y pirámide de características de un modelo de persona. Los filtros por partes se sitúan en el doble de la resolución espacial en la que se ubica la raíz. Imagen extraída de [2].

Para la detección de objetos, los modelos pueden ser fácilmente entrenados utilizando métodos discriminatorios, como es el caso de las máquinas de soporte vectorial (SVM). Los modelos más enriquecidos son más difíciles de entrenar debido a que normalmente usan información latente.

Al entrenar un modelo basado en partes sobre imágenes etiquetadas únicamente con bounding boxes sobre los objetos de interés, las partes no etiquetadas se deben tratar como variables ocultas (latentes). Un etiquetado sobre un mayor número de partes soporta un mejor entrenamiento siempre que estas partes que se hayan usado para las etiquetas sean óptimas. Es por ello por lo que un etiquetado por partes automático puede lograr un mejor rendimiento encontrando las partes efectivas instantáneamente.

El detector Latent SVM obtiene unos resultados muy buenos debidos al uso de la búsqueda exhaustiva y el uso de modelos por partes complejos, además de su completa independencia de la segmentación de la imagen. Como contrapartida, es un algoritmo muy lento.

2.4. DiVA

La plataforma DiVA (*Distributed Video Analysis Framework*) [9] establece un **entorno distribuido** para la intercomunicación simultánea de múltiples fuentes de vídeo con algoritmos de proceso, la conexión en cascada de estos algoritmos de proceso, la visualización de resultados parciales, y la inclusión formalizada de información contextual en el proceso de análisis, todo ello posibilitando que los flujos de datos se procesen en tiempo real.

Tanto la plataforma como las aplicaciones que hacen uso de ella han sido desarrolladas e implementadas por el *Video Processing and Understanding Lab* (VPU) de la Escuela Politécnica Superior de la Universidad Autónoma de Madrid.

Este sistema es escalable (puede añadir módulos de procesado adicionales), eficiente (opera sin apenas incrementar el coste computacional total), generalizable (sus funcionalidades se desarrollan usando herramientas y protocolos genéricos) y con tolerancia a fallos.

Se propone un desarrollo modular en distintos subsistemas. Cada subsistema está relacionado con una función específica dentro de la plataforma. Estas funciones contemplan desde la captura de datos desde distintos dispositivos físicos hasta el almacenaje y presentación de los resultados obtenidos por los algoritmos, todo ello de una forma distribuida. Los subsistemas que componen la plataforma se muestran en la Figura 2.7.

- El **subsistema de captura de datos** reserva los recursos necesarios para su funcionamiento y comienza a capturar datos de las fuentes de vídeo que han sido seleccionadas para capturar la señal de vídeo.
- El **subsistema de bases de datos** proporciona un contexto de aplicación a los algoritmos de análisis y almacena los resultados del procesado de otros módulos de análisis.
- El **subsistema de procesamiento** realiza el propio procesado de la señal de vídeo y proporciona una interfaz de trabajo para cualquier algoritmo de procesado de vídeo.
- El **subsistema de presentación de datos** presenta en pantalla los datos de los análisis resultantes del subsistema de procesamiento.

La plataforma está desarrollada bajo un modelo cliente/servidor en el que se separa la parte servidora de contenido (subsistemas de captura y almacenamiento de datos) de la parte que lo consume (subsistemas de procesamiento y presentación de datos).

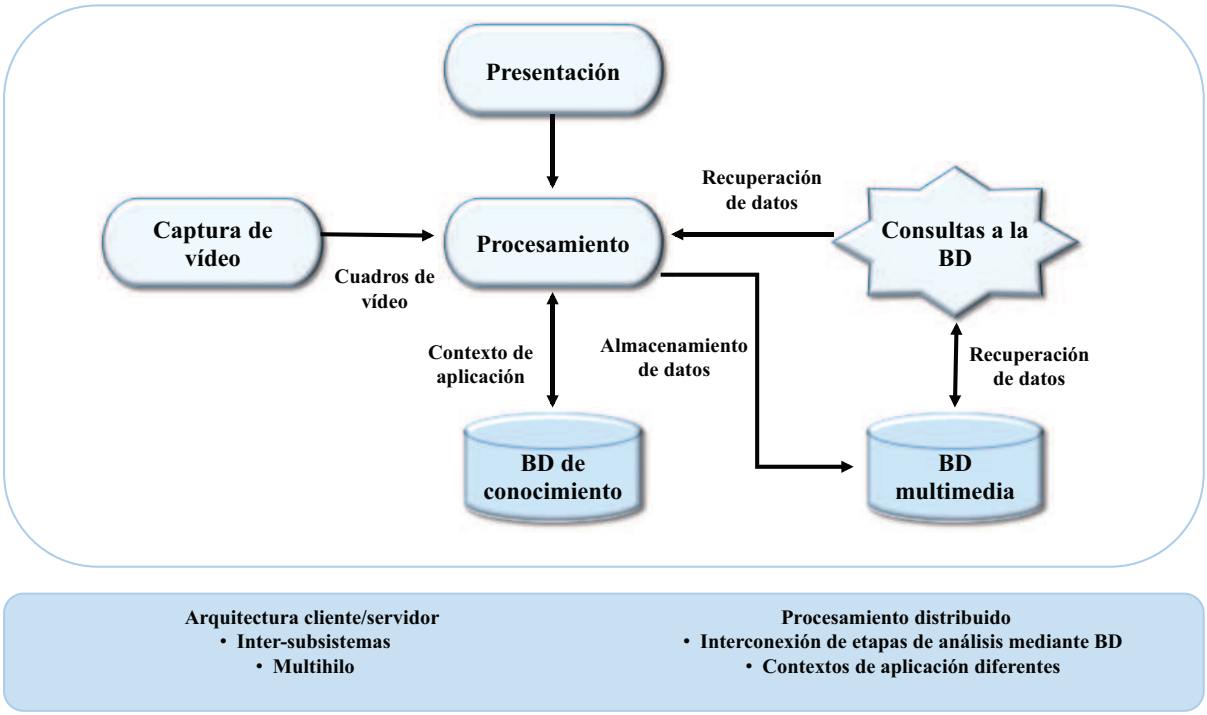


Figura 2.7: Arquitectura Global de la plataforma DiVA.

2.5. Biblioteca QT

Qt es una biblioteca multiplataforma [10] usada para desarrollar aplicaciones con interfaz gráfica de usuario, aunque también puede ser utilizada para el desarrollo de programas sin interfaz gráfica, como herramientas para línea de comandos y consolas para servidores. Esta biblioteca se desarrolla como un software libre y de código abierto a través de Qt Project, donde participa tanto la comunidad, como desarrolladores de Nokia, Digia y otras empresas.

Qt utiliza el lenguaje de programación C++ de forma natural, pero puede ser empleada en varios otros lenguajes de programación a través de bindings, es decir, adaptaciones de la biblioteca que permiten poder usarla en lenguajes distintos al original.

En este trabajo se utiliza la biblioteca Qt para la creación de las interfaces gráficas de los algoritmos de detección de personas presentados. Esta parte se encuentra desarrollada en detalle en el capítulo 4.

3 Diseño y desarrollo

La integración de los algoritmos para la detección de personas en la plataforma DiVA se ha realizado en tres etapas, aunque en en este capítulo sólo se desarrollan las dos primeras:

1. **Preparación del algoritmo:** Se ha creado una clase C++ siguiendo unas pautas de desarrollo determinadas y tomando como punto de partida cada uno de los algoritmos estudiados en el estado del arte. Para ello ha sido necesario el estudio de su funcionamiento para diseñar su adaptación de la mejor forma.
2. **Encapsulación del algoritmo** utilizando el encapsulador DiVAAlgorithm presente en la plataforma.
3. **Desarrollo de la aplicación** y de la interfaz gráfica de usuario. Esta etapa se explica en detalle en el capítulo 4.

3.1. Implementación de los detectores

Para crear una nueva clase C++ y facilitar la integración se han construido los siguientes módulos:

- Un **constructor** que inicializa todos los parámetros y que reserva los recursos necesarios para la ejecución del algoritmo.
- Un **destructor** que libera los recursos necesarios.
- Un método que actúe como **procesador**, recibiendo cada imagen en formato OpenCV y procesándolas.
- Un **mostrador de resultados** del algoritmo.
- Métodos *set/get* para **configurar parámetros**.

DiVAAlgorithm es una clase cuya función es encapsular los algoritmos en la plataforma DiVA [9]. Para ello incorpora petición de datos por parte de un cliente, un método de presentación de datos y otro que añade el procesado de análisis de imagen. Se ha creado para cada algoritmo, por tanto, una clase derivada de DiVAAlgorithm.

En esta sección se detallan las partes más importantes en el desarrollo de cada uno de ellos.

3.1.1. Detectores VPU

3.1.1.1. Fusion

Para la integración de este algoritmo se ha usado código disponible en el VPULab.

Este detector [6] sigue el esquema mostrado en la Figura 2.1. Teniendo esto en cuenta, los pasos seguidos en el procesado de cada frame en este algoritmo son los siguientes:

1. **Extracción de los píxeles de fondo** de la imagen. Para ello se crea una clase *GammaBkgExtractor* y se procesa con ella cada frame recibido.

2. **Se extraen todos los blobs** en cada imagen de frente aplicando la clase *BlobExtractor*. Una vez creada, se extraen regiones conexas sobre el frente obtenido en el punto anterior y se sacan los blobs candidatos a ser personas.

   Después los representamos sobre la imagen utilizando la función *cvRectangle*.

3. Se define el **seguimiento del objeto detectado** tomando como base el filtro de Kalman [30]. Es por ello por lo que se crea una clase *Basic_ KalmanBlobTracker* que procesa la imagen de entrada y la lista de blobs obtenida en el segundo punto. De esta manera se permite no perder la localización de los objetos.

4. **Se implementa el algoritmo de detección Fusion** combinando los detectores **Ellipse**[16], **Ghost**[12] y **Ratio** (relación de aspecto del blob definida en la ecuación 2.1 de la sección 2.3.1.1). Los dos primeros se muestran en la Figura 3.1. Todos ellos procesan la lista de blobs ya obtenida y, por cada uno de estos blobs, se toma la decisión conjunta acerca de si pertenece o no a una persona. La función que se utiliza para ello es *PeopleLikelihoodOfBlobs*.

5. El siguiente paso es **dibujar el rectángulo de detección sobre la imagen** analizada en el instante actual.

   La función utilizada para este cometido es *rectangle*. Para dibujarlo únicamente se necesita la imagen a procesar, los dos vértices opuestos del rectángulo extraídos de la estructura tipo Rect ya creada, el color y el grosor del rectángulo de detección. Estos dos últimos parámetros forman parte de los configurables en este trabajo, para así facilitar la visión de la detección cuando el fondo de la imagen sea muy heterogéneo.

Es importante aclarar que la decisión de detección de este algoritmo únicamente se toma a partir de la máscara extraída, de los blobs resultantes de ella y del modelo de persona entrenado previamente. Este modelo de persona, una vez entrenado, no permite variación de su configuración, salvo en el umbral de detección. Este es el motivo por el que los parámetros que se han configurado para la posible modificación del algoritmo durante la ejecución del algoritmo sean mayoritariamente aquellos que influyen en la extracción de los píxeles de fondo. Estos parámetros son:

i) **El umbral de detección**, (el único perteneciente a la detección en sí).

ii) **La ventana de segmentación.**

iii **La sensibilidad al ruido.**

iv) **El aprendizaje de fondo.**

Todos estos parámetros son explicados en detalle en el capítulo 4 (sección 4.1.2), ya que la modificación de sus valores adquiere más importancia en el demostrador del detector que en el nivel de desarrollo del algoritmo.

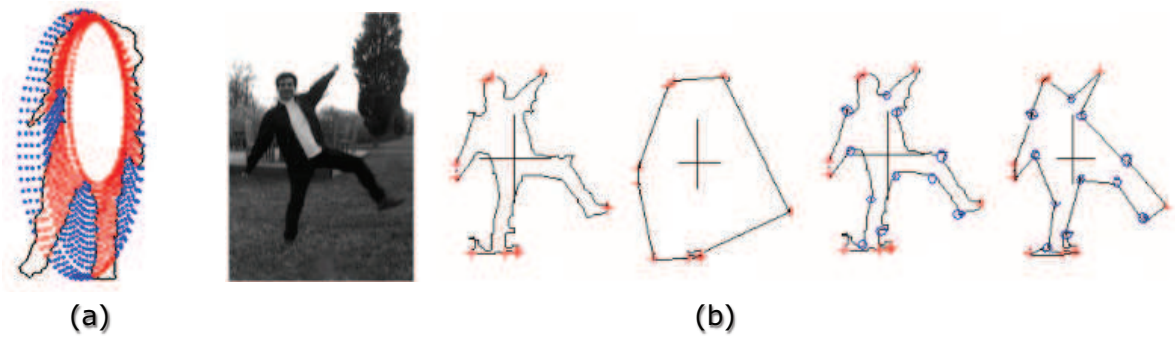


Figura 3.1: Detectores Ellipse(a) y Ghost(b). Imágenes extraídas de [16] y [12].

3.1.1.2. Edge

Al igual que para el algoritmo anterior, para la integración de Edge se ha usado código disponible en el VPULab.

El detector de personas Edge [7], al igual que el detector Fusion [6], funciona siguiendo el esquema de la Figura 2.1.

La forma de procesar cada frame de entrada es, por tanto, prácticamente la misma que la que se ha detallado en la sección 3.1.1.1 y cuyo esquema se ilustra en la Figura 3.2.

Es decir, este detector comparte los 3 primeros pasos de Fusion y dibuja los resultados sobre la imagen exactamente de la misma forma. Sin embargo, el algoritmo de detección Edge se construye **inicializando algoritmos de detección de partes del cuerpo independientes**. Estos detectores se basan en la cabeza, cuerpo, torso y piernas, tal y como se detalla en la Figura 2.2. De esta forma se consigue una detección más robusta y fiable.

Los modelos previamente entrenados pertenecientes a los detectores por partes se localizan en un fichero externo, del cual se extraen los archivos necesarios para cada tipo de detector. Siendo así, podemos modificar el contenido de estos modelos en función de lo que se persiga obtener en los resultados de la detección.

Al igual que en la sección anterior, la detección de personas se lleva a cabo con la función *PeopleLikelihoodOfBlobs*, partiendo de la máscara resultante de la extracción del fondo y los blobs obtenidos durante el procesado de la imagen. Después se dibuja el rectángulo de detección sobre la imagen analizada en el instante actual utilizando para ello la función.

Nuevamente la decisión de detección únicamente se toma a partir de la máscara extraída, de los blobs resultantes de ella y del modelo de persona previamente entrenado que no permite variación de su configuración, salvo en el umbral de detección. Es justamente por esto por lo que los parámetros que se modifican en este algoritmo son los mismos que se han seleccionado para Fusion (Para más detalle acerca de ellos, recurrir a la sección 4.1.2.):

i) **El umbral de detección.**

ii) **La ventana de segmentación.**

iii) **La sensibilidad al ruido.**

iv) **El aprendizaje de fondo.**

Sin embargo, a pesar de compartir el proceso de segmentación, en el caso del detector Edge se obtienen unos resultados más fiables, ya que sobre los blobs extraídos inicialmente se realiza búsqueda exhaustiva y el resultado es la combinación de los cuatro modelos de partes del cuerpo.

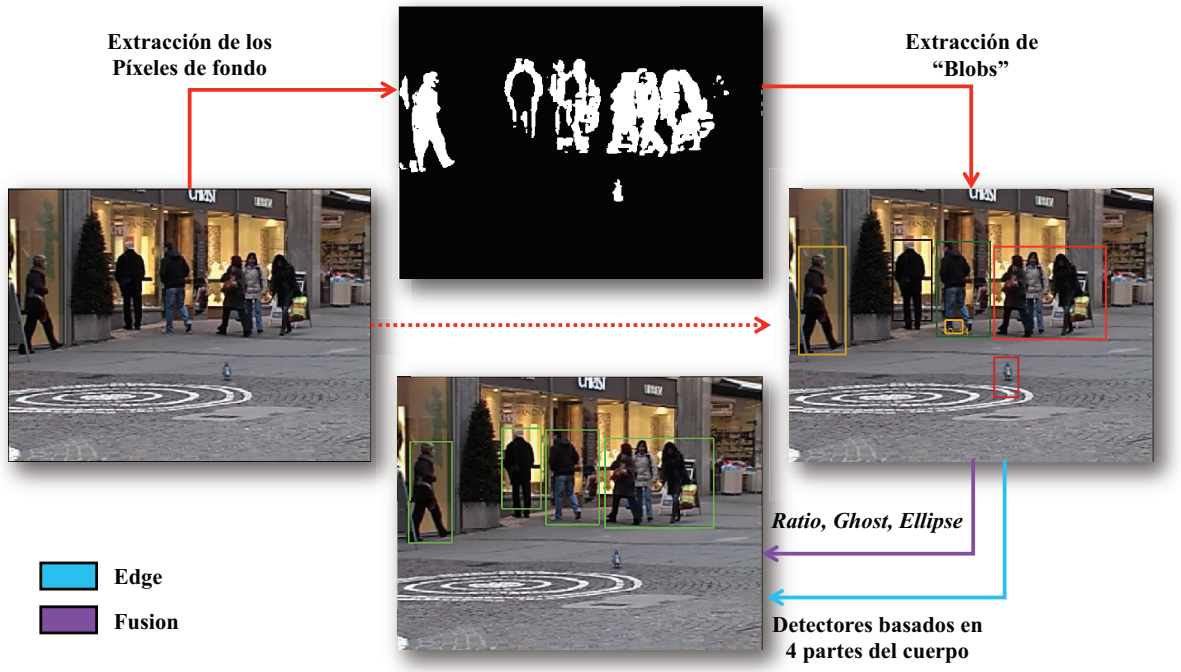


Figura 3.2: Procesado de frames en los detectores Fusion y Edge.

3.1.2. HOG

Para la integración de este algoritmo se ha utilizado el código disponible en OpenCV.

HOG [1] utiliza un método de búsqueda exhaustiva. Por esta razón, los pasos que se siguen en el procesador para la detección de personas son más complejos.

Para este detector se define una estructura de tipo *HOGDescriptor*. Esta estructura se crea con unos parámetros por defecto establecidos por el autor durante la etapa de entrenamiento y que son los siguientes:

- Tamaño 64 × 128 de la ventana de detección.
- Tamaño de bloque 16 × 16 píxeles y tamaño del cell 8× 8 píxeles.
- Número de bins. Tiene una precisión de únicamente 9 bins por cell. Es decir, para un cell dado, sólo se busca la probabilidad de encontrar una persona en 9 puntos del mismo, tal y como se muestra en la Figura 3.3.
- Parámetro gaussiano de la ventana de suavizado.
- Umbral de normalización situado en 0.2.
- Corrección gamma preprocesamiento.
- Número máximo de aumentos de ventanas de detección. Este número es 64.

Los parámetros que, por el contrario, se seleccionan para su posible modificación, son **el coeficiente de aumento de la ventana de detección**, que define el grado de detalle que se permite entre escalas y **el coeficiente que regula el umbral de solapamiento**. Este último define el grado de solapamiento permitido entre varias detecciones y elimina posibles detecciones múltiples sobre un mismo objeto.

permite que, al realizar alguna detección, pueda cubrir los objetos encontrados con varios rectángulos. Es decir, cuanto mayor sea el valor de este coeficiente, mayor será el solapamiento permitido entre personas detectadas.

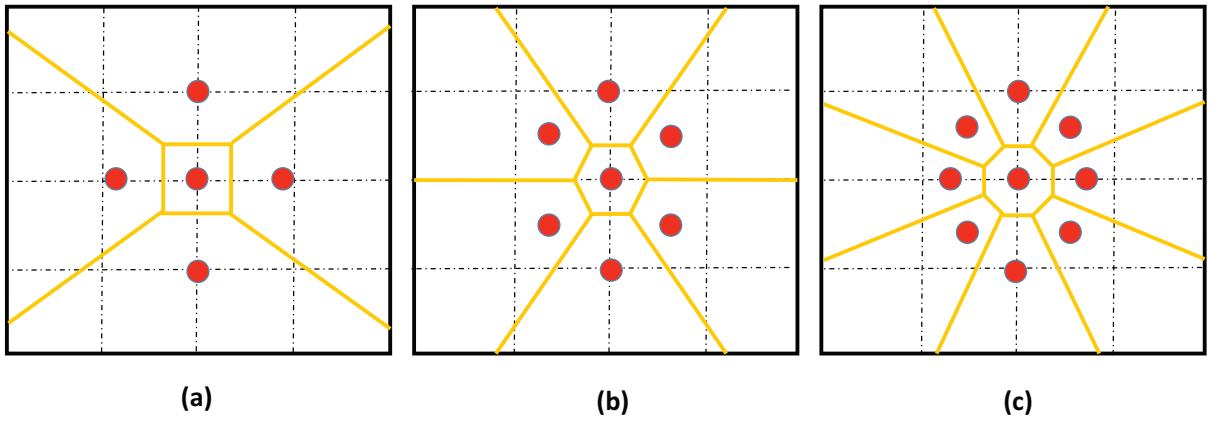


Figura 3.3: Precisión de búsqueda dada por el número de bins. Ejemplos para (a) 5 bins, (b) 7 bins y (c) 9 bins, utilizado en HOG.

Para empezar la implementación del algoritmo, en el constructor **se establecen los coeficientes para el clasificador SVM lineal**. Para ello, primero obtenemos los coeficientes del clasificador entrenado para la detección de personas:

gpu :: HOGDescriptor :: getDefaultPeopleDetector ()

Y después, sobre el resultado, aplicamos la función:

gpu :: HOGDescriptor :: setSVMDetector (HOGDescriptor::getDefaultPeopleDetector())

Así habremos obtenido los coeficientes e inicializamos el clasificador SVM.

Después, para cada frame o imagen de entrada, realizaremos los siguientes pasos (Figura 3.4):

1. **Se detectan los objetos** presentes en la imagen con una ventana multi-escala. La función que nos permite esta detección es:

   gpu::HOGDescriptor::detectMultiScale(const GpuMat & img, vector<Rect>& found_ locations, double hit_ threshold=0, Size win_ stride=Size(), Size padding=Size(), double scale = 1.05, int group_ threshold =2)

   Donde

   - *img* es la imagen que estemos procesando en ese momento.
   - *found_ locations* es un vector que contendrá los objetos detectados.
   - *hit_ threshold* es el umbral para la distancia entre las características y el plano de clasificación SVM.
   - *win_ stride* será el tamaño del bloque de zancada, lo definimos en 8x8.

- *padding* es un parámetro que permite trabajar con bloques de dimensión mínima 8x8 píxeles, añadiendo 0 en aquellos bloques que no alcancen este tamaño.
- *scale* es el coeficiente de aumento de la ventana de detección. Como se ha mencionado antes, forma parte de uno de los parámetros configurables en este trabajo.
- *group_ threshold* es el coeficiente para regular el umbral de solapamiento entre detecciones. Se trata de otro de los parámetros seleccionados para su posible modificación.

2. Para cada objeto detectado **se almacena su localización** en el vector *found_ locations*, por lo tanto, para cada uno de ellos:

   – Se almacenan las coordenadas del rectángulo de detección guardando el contenido del vector *found_ locations* en una estructura de tipo Rect, la cual almacena las coordenadas (x,y) de la esquina superior izquierda, el alto y el ancho del rectángulo.

   – Comprobamos que no existen objetos repetidos en *found_ locations*, es decir, no se obtienen las mismas coordenadas para los rectángulos.

   – Una vez comprobado que este rectángulo de detección es único, guardamos sus coordenadas en un nuevo vector definitivo *found_ filtered*.

3. Una vez creado este nuevo **vector** filtrado del anterior **cuyo contenido son rectángulos de detección únicos**, guardamos sus coordenadas en una variable de tipo Rect.

   Sin embargo, es importante aclarar que el detector HOG crea rectángulos ligeramente más grandes que los objetos reales. Es por ello por lo que los reducimos en una pequeña proporción, para obtener un resultado más agradable y ajustado.

4. El siguiente paso es **dibujar el rectángulo de detección sobre la imagen** analizada en el instante actual.

   La función utilizada para este cometido es *rectangle*. Para dibujarlo únicamente se necesita la imagen a procesar, los dos vértices opuestos del rectángulo extraídos de la estructura tipo Rect ya creada, el color y el grosor del rectángulo de detección. Estos dos últimos parámetros forman parte de los configurables en este trabajo, para así facilitar la visión de la detección cuando el fondo de la imagen sea muy heterogéneo.

5. Por último, y para que los rectángulos de detección resultantes para un frame no permanezcan durante el análisis de los frames siguientes, **reseteamos el vector** *found_ filtered* creado en el punto 2 y que contiene los rectángulos de detección únicos calculados para cada frame.

En la Figura 3.4 aparecen los pasos llevados a cabo por el algoritmo de manera sintética.

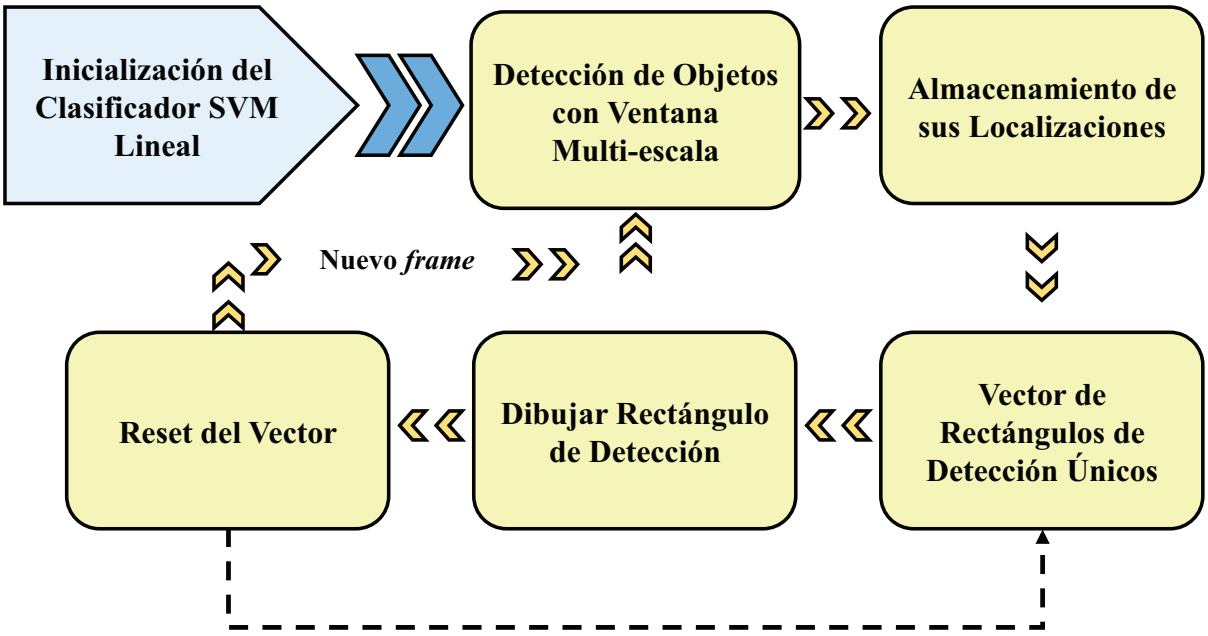


Figura 3.4: Esquema de la implementación del algoritmo HOG.

3.1.3. Latent SVM

Al igual que para HOG, este detector se ha integrado a partir del código proporcionado por las librerías de OpenCV.

El detector Latent SVM [2], en su versión actual implementada, procesa las imágenes de tal forma que no le permite funcionar en tiempo real. No sólo es exhaustivo, como ocurre con HOG, sino que también utiliza un modelo de persona más complejo basado en partes.

Latent SVM se implementa de la siguiente manera:

- En el constructor se carga el **modelo de persona** a partir del cual se van a realizar las detecciones de las mismas. Para ello se crea una variable perteneciente a la clase *CvLatentSvmDetector* y sobre ella se ejecuta la función *cvLoadLatentSvmDetector*,que cargará el detector entrenado desde un archivo.
- Una vez establecido el modelo, el procesador se encarga de analizar cada uno de los frames recibidos para detectar personas en ellos.

  La forma de **obtener las detecciones** en Latent SVM es utilizando la función *cvLatentSvmDetectObjects*. Esta función encuentra regiones rectangulares de la imagen con unos niveles de confianza adecuados y que, por lo tanto, les hacen susceptibles de contener los objetos buscados, es decir, personas. Los parámetros que se necesitan para la localización de estas regiones son:

  - La **imagen** que se está procesando en el instante actual.
  - El **detector** definido por el modelo de personas entrenado introducido en el constructor del algoritmo.
  - Un **almacenamiento** en memoria de la secuencia resultante de rectángulos de los objetos detectados.
  - Un **umbral de detección** definido en 0.5 y configurable. Cualquier objeto que, después de compararse con el detector entrenado, supere este umbral, será detectado como persona.

Por último, para cada elemento detectado se dibuja un rectángulo de detección con la función *cvRectangle* del mismo modo en el que se ha utilizado para los tres algoritmos anteriores.

Debido a la complejidad y exhaustividad del procesado de imagen, los resultados de la detección de Latent SVM son muy buenos, pero como ya se ha comentado previamente, el precio a cambio de la precisión de este algoritmo es un alto coste computacional.

4 Demostradores

Una vez explicadas las pautas de desarrollo de cada uno de los algoritmos estudiados en este trabajo y la encapsulación de los mismos en la plataforma DiVA, sólo queda desarrollar la aplicación de los detectores para conseguir la integración completa.

Para llevar a cabo este propósito se han creado **demostradores** en los que se pueda comprobar el funcionamiento de estos algoritmos de detección de personas. En lugar de construir demostradores que necesiten hacer uso de funciones para inicializar o cerrar las conexiones con los servidores y que permitan la configuración de los parámetros por consola, se ha decidido crear demostradores más avanzados y funcionales: **interfaces gráficas**.

4.1. Funcionalidad de la Interfaz Gráfica

Para desarrollar la interfaz haremos uso del programa *Qt Designer*, que utiliza la biblioteca Qt introducida en el capítulo 2. Es en el *Qt Designer* en el que se diseña la apariencia de la aplicación, incluyendo títulos, iconos, logos, botones, barras de herramientas y ventanas adicionales.

Para que este diseño construido en *Qt Designer* sea utilizado como demostrador de los algoritmos de detección, se deben realizar varios cambios sobre el código del programa. Se han añadido nuevos archivos en C++ para la funcionalidad de cada ventana utilizada en la interfaz gráfica, además de los archivos *ui* generados automáticamente por *Qt Designer* y que permiten la interactividad de la interfaz con el código programado del algoritmo. En este trabajo las ventanas necesarias para los detectores son: la principal, la de configuración de parámetros, la de cámaras, la de ayuda y la de información.

Antes de explicar las interfaces gráficas específicas para cada algoritmo, se van a describir los aspectos más característicos del desarrollo de las dos ventanas más importantes:

- **La ventana principal**. En ella se realizan las conexiones necesarias entre las funciones del código y los botones y las acciones disponibles en la barra de herramientas que se han diseñado. De esta manera se permite obtener un resultado al pulsar cualquier elemento de la interfaz.

  Las funciones posibles, son, entre otras: la conexión a cámaras insertando su dirección IP y su puerto, inicialización del algoritmo, muestra de resultados en la pantalla, detenención, acceso a las ventanas de propiedades, ayuda, información, etc.

- **La ventana de propiedades**. En esta ventana se configuran los parámetros modificables de cada detector. Para ello conecta todos los elementos de transformación de datos numé-

ricos como *SpinBoxes, Scrollbars, Sliders...* con funciones que permiten la extracción y la asignación de nuevos valores a los parámetros seleccionados.

En este trabajo, los algoritmos para los que se ha creado una interfaz gráfica a modo de demostrador son **HOG, Fusion y Edge**.

Es importante destacar que no se ha construido un demostrador con estas características para Latent SVM debido al hecho de que es un detector enormemente lento e incapaz de trabajar en un tiempo razonable (en el capítulo 5 se profundiza en esta cuestión). Por lo tanto, para él se han desarrollado únicamente los dos primeros niveles de integración definidos al comienzo del capítulo 3.

4.1.1. HOG

En esta sección se introduce la interfaz del detector HOG, creada siguiendo los pasos comentados. El resultado se puede ver en la Figura 4.1, donde se muestran las diferentes partes de la misma. Atendiendo a la numeración de la figura, la funcionalidad de cada una es la siguiente:

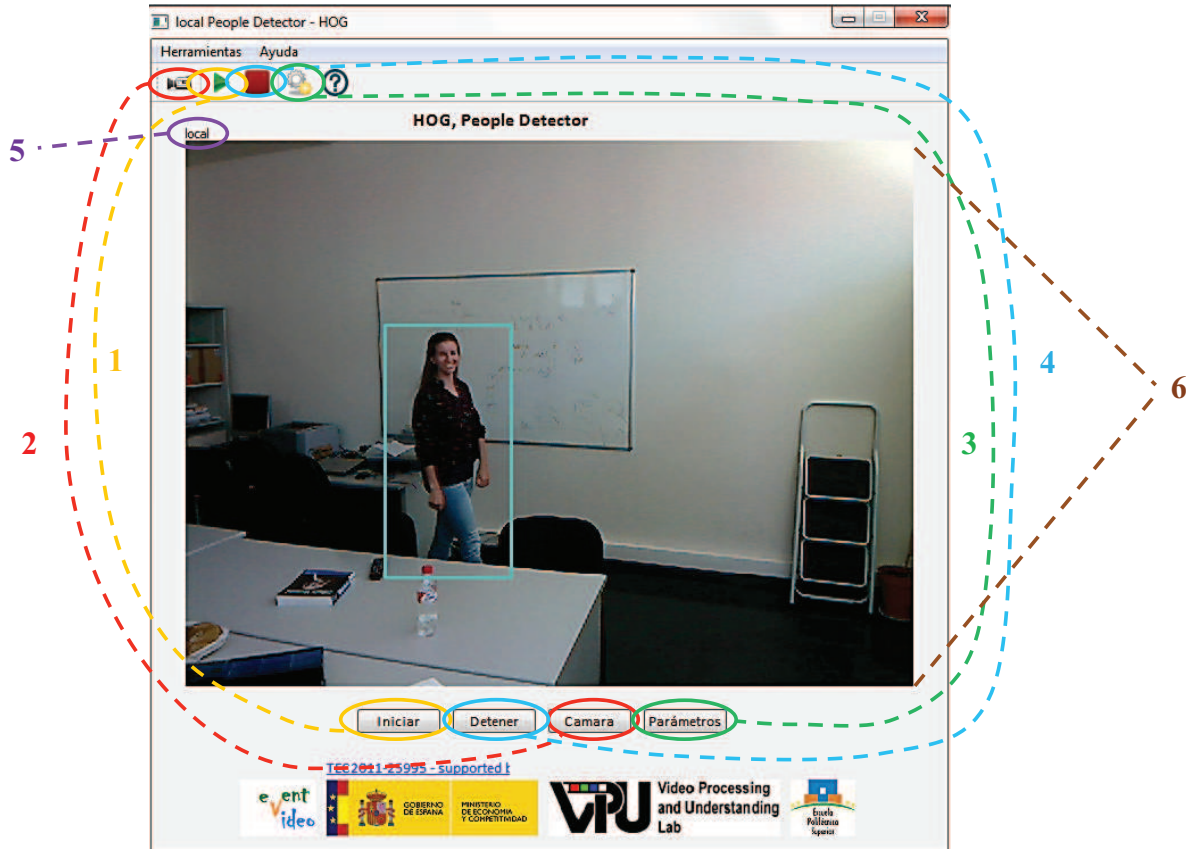


Figura 4.1: Interfaz del detector HOG. Significado de 1-6 detallado en el texto.

- **1.** Inicio del algoritmo.

  No comenzaremos a visualizar los resultados de la detección hasta que no sea pulsado, momento en el cual se conectará con la cámara seleccionada y comenzará a procesar los frames que reciba de ella.

- **2.** Selección de la cámara.

  Estos botones permiten seleccionar una dirección IP y un puerto para la cámara a la que queramos conectarnos. Además, se permite asignarle un nombre con el que se guarda o incluso editar alguno de sus campos en caso de que hiciese falta.

- **3.** Configuración de parámetros.

  Permite acceder a otra ventana mostrada en la Figura 4.2 en la cual se pueden modificar los parámetros del algoritmo que sean posibles o que se hayan permitido. La funcionalidad de esta ventana se detalla más adelante.

- **4.** Detención del algoritmo.

  Al pulsar, se cierra la conexión con la cámara a la que estuviésemos conectados y dejamos de recibir frames hasta que no se decida reiniciar la aplicación. Es posible realizar un cambio de cámara durante esta detención y reanudar procesando frames de una videocámara distinta.

- **5.** Etiqueta de la cámara seleccionada.

  Aporta información acerca de la cámara a la que está conectado el algoritmo en ese momento. Este nombre es el que haya asignado el usuario en la conexión inicial a ella.

- **6.** *Display.*

  Pantalla en la que se muestran los frames procesados por el detector HOG.

La interfaz de HOG es bastante sencilla por el hecho de que sólo necesita mostrar la ventana de detección final. Sin embargo, como se verá en la próxima sección, los algoritmos Fusion y Edge realizan diferentes etapas de proceso, por lo que para ellos es posible la visualización de resultados intermedios.

Tal y como se detalla en la sección 3.1.2, muchos de los parámetros de HOG que nos ofrecerían resultados interesantes al cambiar sus valores no permiten su modificación por el hecho de que se definen automáticamente al crear la estructura HOGDescriptor de OpenCV con la que se procesan los frames en el código. Ejemplos de estos parámetros son el número de bins (Figura 3.3), el tamaño de la ventana de detección, el umbral de detección, etc.

Sin embargo, en la Figura 4.2 se muestran los parámetros con posibilidad de modificación del algoritmo HOG con sus valores inicializados por defecto. Dos de ellos, ***Scale*** y ***Group_ Threshold***, se relacionan directamente con la forma en la que se detectan las personas. *Scale* es el coeficiente de aumento de la ventana de detección, que dará lugar a una búsqueda más refinada cuanto menor sea su valor (pero siempre por encima de 1, ya que de lo contrario no aumentaríamos el tamaño de la misma) y *Group_ Threshold* es el coeficiente para regular el umbral de solapamiento entre múltiples detecciones. Éste último permitirá un mayor solape de rectángulos de detección cuanto mayor sea su valor, eliminando cualquier solapamiento posible si su valor es 0. Por otro lado se han añadido dos parámetros que tienen que ver con la visualización de los resultados: ***Thickness*** y RGB, los cuales definen el grosor y las componentes de color rojo, verde y azul (0–255) para el rectángulo de detección, permitiendo una mayor comodidad de su visión en caso de tratar con escenarios que camuflen los resultados.

Figura 4.2: Ventana de configuración de parámetros de HOG.

En el momento en el que se pulsa el botón 'OK', se aplican todos los cambios producidos en los valores de los parámetros a las variables correspondientes dentro del procesador del algoritmo. De igual manera, la próxima vez que se recurra a esta ventana, los datos numéricos que aparecerán serán los pertenecientes a los parámetros en ese justo instante.

4.1.2. Fusion y Edge

Como se ha explicado en el capítulo 3, el procesado de frames que realizan Fusion y Edge es muy parecido, ya que ambos siguen el esquema mostrado en la Figura 2.1.

La única diferencia a nivel de programación entre ellos está en los modelos a partir de los cuales extraen las evidencias para la clasificación *persona/no persona*. Por lo tanto **la funcionalidad de la interfaz de ambos será idéntica**, con incluso los mismos parámetros de configuración, ya que estos únicamente influyen en la segmentación, etapa común en los dos detectores. Es por ello por lo que en esta sección se mostrarán los detalles de la interfaz gráfica de uno solo de ellos (Edge), suponiendo una equivalente para Fusion.

La ventana principal de esta interfaz, que se muestra en la Figura 4.3, es más compleja que la que se ha descrito para HOG en la sección anterior. En esta aplicación y según la figura, se han diseñado 3 partes distintas y necesarias:

- **1.** Acciones de control de la aplicación.

  En esta parte se encuentran las siguientes opciones: inicio del algoritmo, detención de su ejecución, selección de cámara a partir de la cual recibir frames a procesar y configuración de los parámetros, cuya funcionalidad se detalla más adelante y se muestra en la Figura 4.4. Todas estas acciones son comunes a las descritas para la interfaz de HOG, ya que son básicas para la creación de cualquier demostrador de algoritmos. Además también están disponibles en la barra de herramientas junto con opciones de ayuda e información.

- **2.** Selección de visualización de resultados.

  Se han diseñado cuatro *checkboxes* que, en caso de ser seleccionados, permiten mostrar las ventanas de resultados correspondientes a:

  - *Input*. Muestra cada frame recibido directamente de la cámara seleccionada, sin procesar.
  - *Output*. Presenta el frame procesado por el algoritmo, detectando las personas presentes en la escena con rectángulos de detección. Es, por tanto, la ventana de resultados final.
  - *Background Extractor*. Segmentación del fondo. Muestra una imagen binaria en la que se han extraído los píxeles del fondo y sólo destacan los píxeles correspondientes al frente.
  - *Blob Tracking*. Enseña los blobs extraídos a partir de la segmentación de fondo y su seguimiento, para su posterior procesado y clasificación.

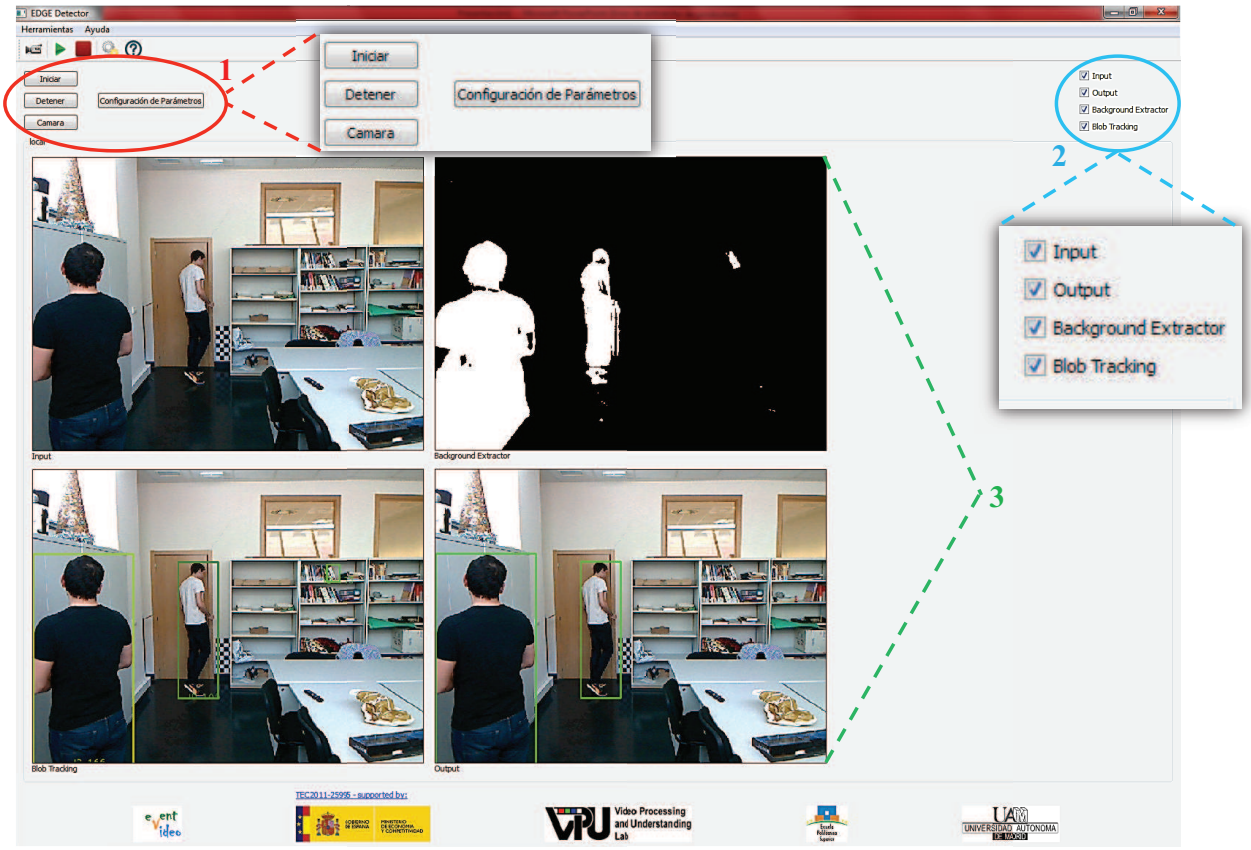


Figura 4.3: Interfaz válida para los detectores Edge y Fusion. Significado de 1-3 detallado en el texto.

- **3.** *Display* múltiple.

  Muestra todas las ventanas de resultados seleccionadas por los *checkboxes* explicados en el punto anterior.

  Esta relación se consigue creando funciones de conexión con el algoritmo. Cada vez que el estado de un *checkbox* cambie, se ejecuta una función que añade o elimina del *display* la opción de visualización de resultados seleccionada.

Además se incluyen etiquetas bajo cada una de las ventanas para facilitar la información de lo que se está visualizando y que se corresponden con el nombre de cada *checkbox*. Este etiquetado es útil porque en función del orden en el que seleccionemos las ventanas, los resultados se colocarán en el *display* de una forma u otra.

En cuanto a la ventana de configuración (Figura 4.4) de estos algoritmos de detección, los únicos parámetros que tiene sentido modificar son los relacionados con la forma en la que se realiza la segmentación de frente/fondo. Esto es así porque, tal y como se ha descrito en el capítulo 3 al explicar su funcionamiento interno y queda reflejado en la Figura 2.1, ambos detectores clasifican personas en función de los blobs extraídos de la segmentación. Es decir, únicamente se analizan las figuras consideradas como frente en la escena. Por lo tanto, los parámetros que ajustan la extracción del fondo son los que definirán la calidad de la detección: cuanto mejor sea la segmentación, mejores serán los resultados obtenidos.

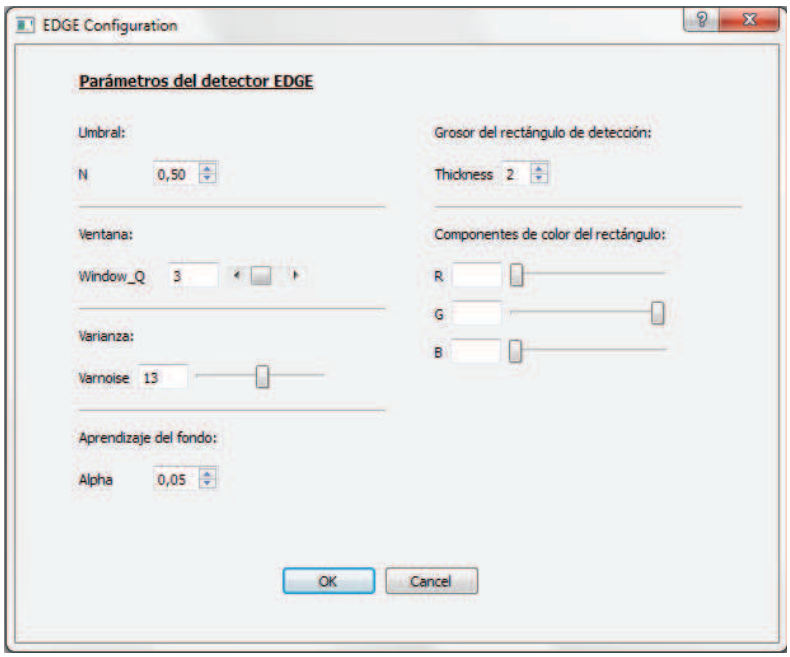


Figura 4.4: Ventana de configuración de parámetros válida para los detectores Edge y Fusion.

Estos parámetros son:

- **Umbral de detección**, N. Fija un valor de probabilidad o confianza a partir del cual, todo blob que obtenga un resultado mayor en el proceso de clasificación, será detectado como persona.

- **Ventana de segmentación**, *Window_ Q*. Dimensión de la ventana ($Window\_Q \times Window\_Q$) que realiza la segmentación sobre cada píxel comparándolo con los de su alrededor.

  En consecuencia, cuanto mayor sea este parámetro, mayor será el número de píxeles que abarcará la ventana de segmentación, es decir, para clasificar un píxel como frente hace uso de la información de un mayor número de píxeles vecinos, dando como resultado una segmentación más gruesa y viceversa. Un ejemplo de lo que ocasiona la variación de este parámetro se encuentra en la Figura 4.5.

- **Sensibilidad al ruido**, *Varnoise*. Su valor define la varianza del ruido de fondo. En la Figura 4.5 también se muestra de una forma aclaratoria las consecuencias de la modificación de este parámetro.

  Cuanto mayor sea su valor, mayor será el aislamiento frente al ruido en el proceso de segmentación, es decir, será más robusto a él. Sin embargo, si el valor es muy grande, esta opción puede tener consecuencias no deseadas como puede ser la pérdida de información de algunas partes del cuerpo, resultando en una detección por partes en lugar de una detección de la persona entera. Por otro lado, si *Varnoise* disminuye más de lo debido, el algoritmo de segmentación será muy sensible al ruido y lo incluirá como frente en el resultado, obteniendo una detección menos ajustada al contorno de los cuerpos o incluso falsas detecciones.

- **Aprendizaje del fondo**, *Alpha*. Coeficiente que describe la capacidad que tiene el segmentador para actualizar el estado del fondo y así facilitar la separación del frente. Si el algoritmo se inicializa con la presencia única del fondo, el valor de este parámetro puede ser mínimo, ya que no necesita actualizar la información del mismo para segmentar correctamente a toda persona que se cruce en la escena. Sin embargo, si la inicialización no fuese correcta, los problemas que aparecerían en la segmentación se podrían solucionar aumentando el valor del aprendizaje del fondo.

Por último y para mejorar la visualización de los resultados en caso de ser necesario, se añaden los parámetros de grosor y componentes de color del rectángulo de detección, tal y como se hizo con la ventana de parámetros del algoritmo HOG en la sección anterior.

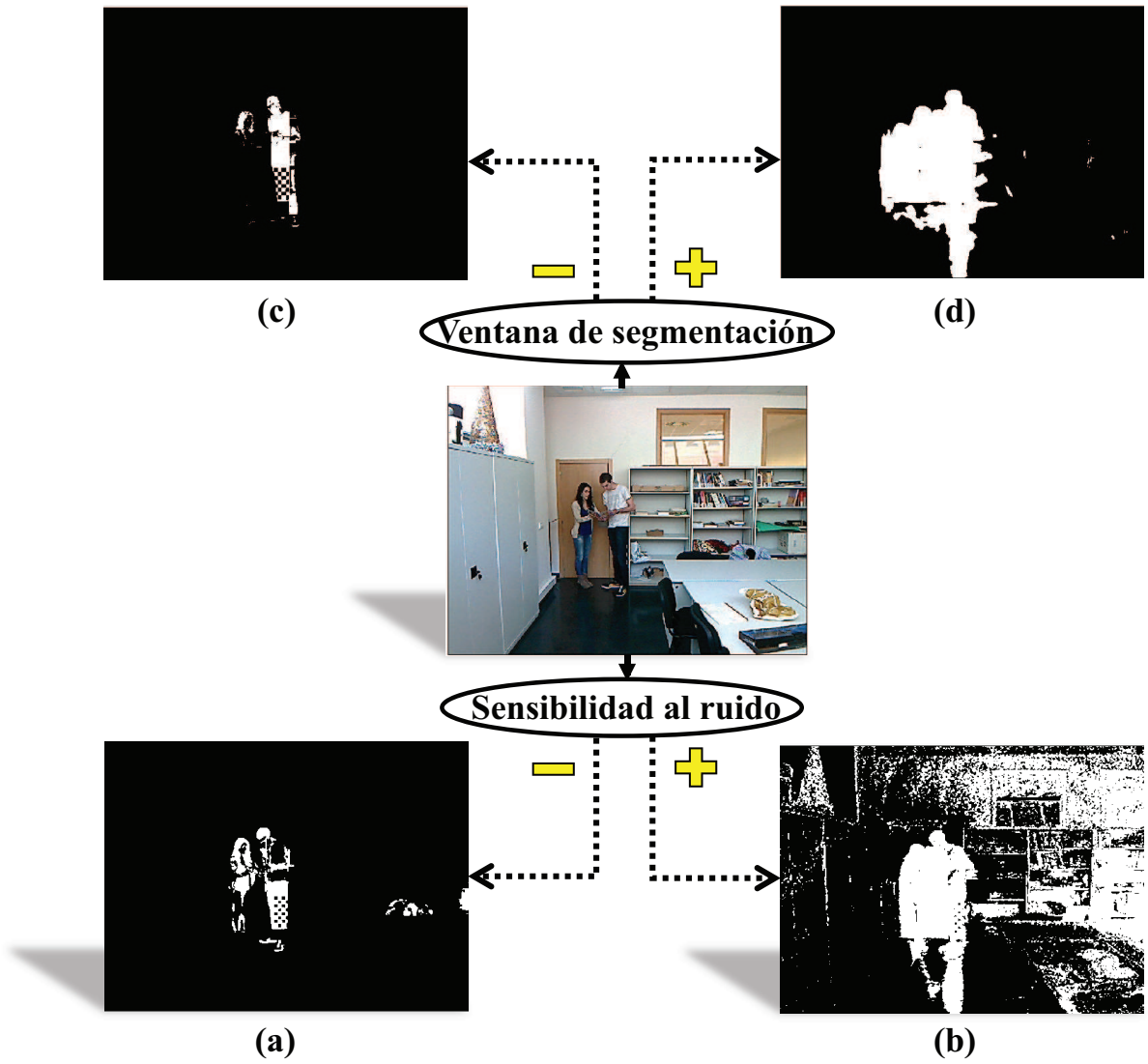


Figura 4.5: Resultados de la variación de la sensibilidad al ruido y de la dimensión de la ventana de segmentación para Edge y Fusion. Ejemplos para (a) $Varnoise$=25, (b) $Varnoise$=3, (c) $Window\_Q$=1 y (d) $Window\_Q$=7.

5 Análisis de Resultados

5.1. Características de Evaluación

En este capítulo no se evalúan los resultados de la detección de los algoritmos estudiados, debido a que dicha evaluación se remite a los resultados de los propietarios.

El objetivo en este trabajo es facilitar la ejecución de dichos algoritmos en tiempo real, permitiendo al usuario evaluar fácilmente el resultado visual sobre cualquier escenario. Por ese motivo nos centraremos en evaluar el **rendimiento** o coste computacional de los detectores de personas desarrollados: **Fusion** [6], **Edge** [7], **HOG** [1] y **Latent SVM** [2].

Todos los algoritmos han sido implementados utilizando el lenguaje C++, las librerías de OpenCV [8] y la plataforma DiVA [9].

Los resultados del rendimiento han sido obtenidos utilizando:

- Un equipo Windows 8.1, Procesador Intel(R) Core(TM) i5-3330, CPU 3.00GHz, Memoria RAM 8.00 GB y Sistema operativo de 64 bits.
- Un conjunto de vídeos extraído del **dataset PDds** [36], compuesto por varias secuencias de vídeo de diferente complejidad.

  En particular se han extraído 18 secuencias de un dataset público relacionado con el propósito de clasificar personas u objetos. Estas secuencias utilizan algunas partes de las secuencias pertenecientes al dataset CVSG (*Chroma Video Segmentation Ground-Truth*) [37], centrado en la evaluación de máscaras de segmentación basadas en movimiento.

  A pesar de ello, este dataset se ha modificado para ser útil en la evaluación los algoritmos de detección de personas con los que se ha trabajado. En la Figura 5.1 se muestran algunos ejemplos de varias secuencias incluyendo blobs de detección. Cada objeto o persona presente en la escena ha sido etiquetado en frames separados, lo cual permite obtener la detección de personas de forma automática para cada secuencia. Además, la variación de complejidad en el dataset permite observar los resultados ante factores críticos que afectan al comportamiento de los algoritmos de detección.

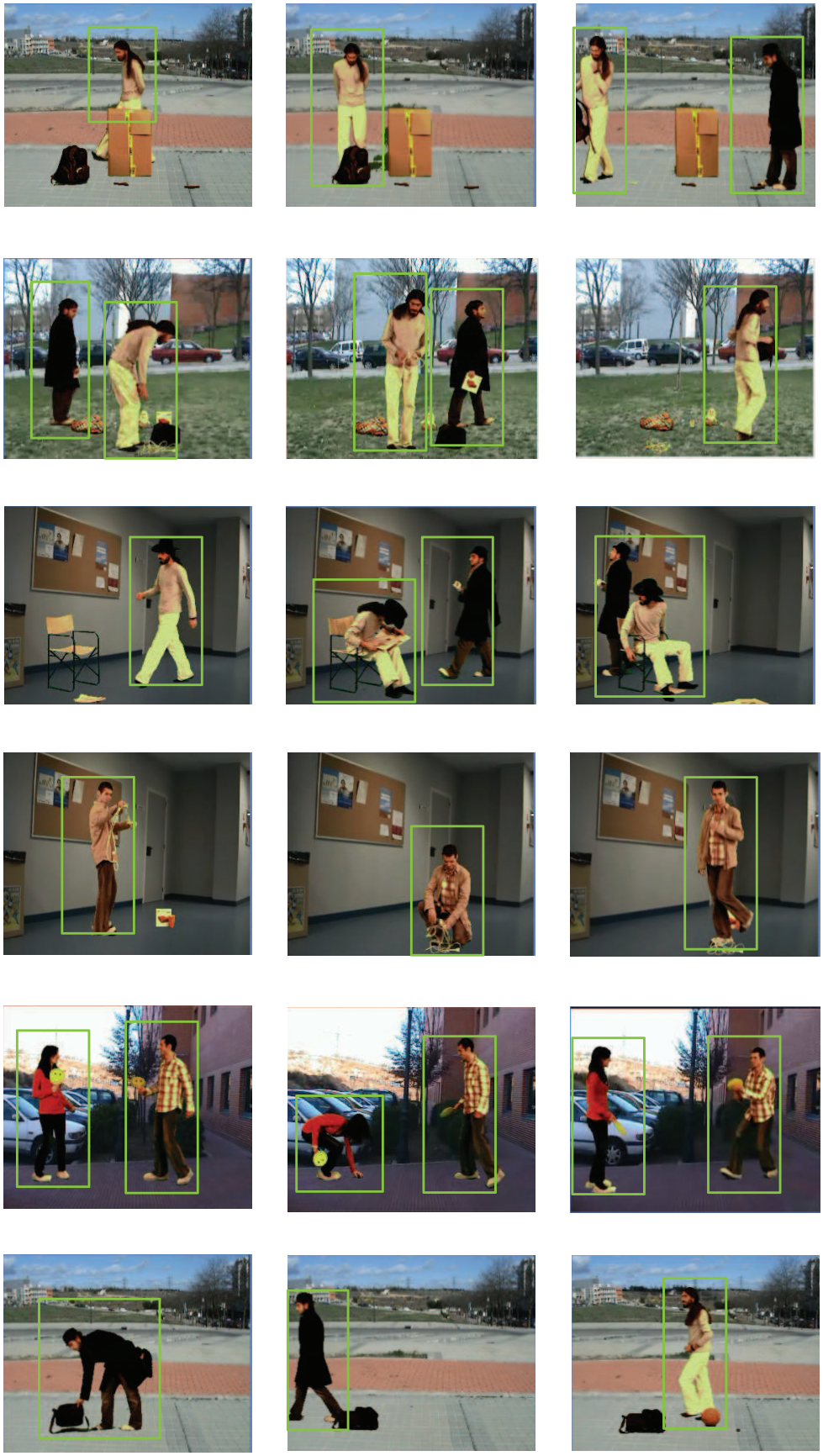


Figura 5.1: Ejemplos de secuencias del dataset PDds. Cada secuencia muestra tres frames aleatorios.

5.2. Comparativa entre Detectores

En esta sección se analizan los resultados obtenidos en términos del coste computacional de los algoritmos Edge, Fusion, HOG y Latent SVM.

Para ello se han ejecutado los cuatro detectores para los 18 vídeos seleccionados del dataset y cuyo tamaño de frame es de 352×288 píxeles. Teniendo en cuenta el número de frames contenidos en cada uno de ellos, se han extraído los resultados presentes en la Tabla 5.1.

	Latent SVM	HOG	Fusion	Edge
t (seg.)/frame	1,7795	0,0641	0,0135	0,0064
t (seg.)/25 frames	44,4800	1,6025	0,3375	0,1600
n° frames/seg. (fps)	0,5610	15,6000	74,0740	156,2500

Tabla 5.1: Tiempos de detección y número de frames procesados. Tiempo medio en segundos dedicado por cada detector al procesado de 1 o 25 frames y fps.

Estos valores son el promedio del tiempo que cada uno de los algoritmos tarda en procesar cada frame, clasificando los objetos contenidos en la escena como persona o no persona.

A la vista de los resultados podemos evaluar la eficacia de estos detectores para su funcionamiento en tiempo real. Teniendo en cuenta que para que el sistema visual humano perciba un movimiento de forma fluida y no una mera sucesión de imágenes necesita que las secuencias de vídeo reproduzcan 25 frames por segundo, se concluye, a la vista de los datos de la Tabla 5.1, que el algoritmo **Latent SVM no es capaz de funcionar cumpliendo con las exigencias del tiempo real**, ya que necesita casi 2 segundos para procesar cada uno de los frames.

HOG, por su parte, dedica menos tiempo al procesamiento pero **no llega a funcionar estrictamente en tiempo real**. Este detector tiene una frecuencia próxima a 15 fps, es decir, puede tratar 15 o 16 frames por segundo, lo que se traduce en una baja eficacia cuando se trabaja con secuencias de vídeo de 25 frames por segundo. Esto ocurre por el método de búsqueda exhaustiva que lleva a cabo HOG antes de la clasificación, el cual, al igual que ocurre con Latent SVM, aumenta sustancialmente el tiempo de procesado. La evaluación del coste individual de ambas etapas, búsqueda exhaustiva y clasificación, no es posible.

Por último, los algoritmos para la detección de personas **Fusion y Edge** tienen un tiempo de procesamiento por frame tan bajo que no dejan duda de su **capacidad para funcionar en tiempo real**. De hecho, de acuerdo con los resultados obtenidos en la Tabla 5.1, en un solo segundo el detector Fusion podría llegar a tratar 74 frames, mientras que Edge incluso alcanzaría los 156 frames. Este hecho es debido a la diferencia de etapas que siguen estos algoritmos frente a HOG y Latent SVM. En lugar de utilizar una búsqueda exhaustiva, Fusion y Edge realizan una segmentación seguida de la clasificación de los objetos segmentados en la primera etapa. Esto agiliza el proceso de detección permitiendo estos resultados tan apropiados para el funcionamiento en tiempo real.

Al contrario de lo que ocurre con Latent SVM y HOG, es posible evaluar el rendimiento de las diferentes etapas en las que se dividen la segmentación y la clasificación presentes en los detectores Fusion y Edge. El conjunto de todas ellas compone el procesado total de los mismos. Los resultados de esta evaluación se muestran en la Tabla 5.2 como porcentaje de tiempo dedicado a cada una de ellas. El tiempo de procesado total se divide en las siguientes tareas:

- *Background*. Extracción de los píxeles de frente.
- *Extraction*. Extracción de los blobs en cada imagen de frente.
- *Tracking*. Seguimiento de los blobs.
- *People*. Clasificación persona/no persona para cada blob.
- *Others*. Cualquier proceso interno adicional o transición entre los anteriores.

Algoritmo	Background	Extraction	Tracking	People	Others
Fusión	25,10 %	4,97 %	1,36 %	**67.80 %**	0,74 %
Edge	60,39 %	10,79 %	4,49 %	**22.89 %**	1,42 %

Tabla 5.2: Distribución del Tiempo de Procesado. Porcentaje de tiempo dedicado a cada etapa en %.

El algoritmo Fusion dedica la mayor parte del tiempo de procesado de cada frame a la clasificación de personas realizada sobre cada blob. Esto es así por la combinación de evidencias que toma este detector antes de tomar la decisión final, las cuales provienen de la relación de aspecto del blob, del algoritmo de Ellipse [16] y Ghost [12], tal y como se explica en la sección 2.3.1.1.

Por otro lado, el detector Edge inicializa los modelos por partes del torso, cabeza, piernas y cuerpo y, una vez hecho esto, la detección de las personas a partir de la extracción del fondo es casi automática. Esta es la razón por la que, aunque este algoritmo necesite más tiempo para su arranque, una vez iniciada su ejecución apenas consuma tiempo para la clasificación persona/no persona, disminuyendo por tanto el tiempo de detección conseguido en el algoritmo Fusion.

Los porcentajes de la Tabla 5.2 dan información de la distribución del tiempo de procesado de frames para Fusion y Edge. Sin embargo, se debe tener en cuenta que Edge consume menos tiempo que Fusion en el procesamiento de un cada frame. Por lo tanto, aunque algunos datos como el porcentaje de la extracción de los píxeles de frente o la extracción de blobs sea mayor en el detector Edge, la realidad es que ambos gastan aproximadamente el mismo tiempo en todas las tareas menos en una: la clasificación de personas. Ésta es, por tanto, la principal causa de que Fusion sea un algoritmo más lento que Edge.

6 Conclusiones y Trabajo futuro

6.1. Conclusiones

Este trabajo tenía como propósito la adaptación y la encapsulación de varios algoritmos de detectores de personas en una plataforma que permitiese ejecutarlos sobre videocámaras que transmiten en tiempo real, ampliando así su funcionalidad. Esta plataforma en cuestión es DiVA.

Para cumplir con este objetivo se ha realizado un estudio del estado del arte de los detectores con los que se ha trabajado: Latent SVM, HOG, Fusion y Edge. En este estudio se han aprendido, a partir de los artículos de referencia, los aspectos más importantes en el procesado de cada uno de ellos, incluyendo sus ventajas y también sus limitaciones.

Por un lado, HOG y Latent SVM seleccionan las regiones de la imagen candidatas a ser detectadas como personas escaneando la imagen completa a varias escalas y rotaciones, es decir, realizando una búsqueda exhaustiva. Los modelos de persona que utilizan son complejos y permiten altas tasas de detección, pero el coste computacional es demasiado alto para permitir que funcionen en tiempo real.

Todo lo contrario ocurre con los sistemas de Fusion y Edge. Éstos realizan el procesado de las imágenes mediante técnicas de segmentación, en concreto la sustracción de fondo. Emplean modelos más simplificados y reducen su coste computacional, razón por la cual funcionan en tiempo real.

El rendimiento de estos detectores ha sido evaluado durante el desarrollo de este trabajo, consiguiendo evidencias que prueban las características extraídas de las referencias para cada algoritmo de detección. En concreto, se aportan datos que verifican el funcionamiento en tiempo real de los detectores Fusion y Edge, la aproximación a este funcionamiento por parte de HOG y la imposibilidad ofrecida por Latent SVM.

Para la integración de estos algoritmos de detección se ha implementado una nueva clase C++ capaz de funcionar en la plataforma DiVA, para lo cual ha sido necesario profundizar en el conocimiento del procesado de frames llevado a cabo por cada uno de ellos. De manera resumida:

- **Latent SVM** parte de un detector de persona entrenado y, basándose de él, busca regiones en las imágenes con un nivel de confianza lo suficientemente alto como para detectar que se encuentra una persona.
- **HOG** define una estructura con un tamaño determinado de la ventana de detección, un tamaño del cell y del bloque, un número de bins fijado en 9, un umbral de detección

y un número máximo de ventanas de detección. Después establece coeficientes para un clasificador SVM lineal y utiliza una ventana multi-escala para detectar objetos.

- **Fusion y Edge** comparten prácticamente todo el desarrollo del procesado: Primero extraen los píxeles de fondo de la imagen y extraen los blobs de las imágenes de frente resultantes, siendo capaces de realizar un seguimiento sobre ellos con el filtro de Kalman. El único paso en el que se diferencian es en la toma de evidencias para clasificar cada blob como persona/no persona:

  - **Fusion** combina los detectores Ellipse, Ghost y una relación de aspecto del blob.
  - **Edge** inicializa algoritmos de detección de cuatro partes del cuerpo independientes (piernas, cabeza, torso y cuerpo) y los toma como referencia.

  Para ambos algoritmos, la detección de personas depende por completo de la segmentación realizada, ya que únicamente se analizan los blobs obtenidos a partir de la máscara calculada.

Una vez implementados los algoritmos y conseguido el objetivo de que funcionen tanto sobre vídeo como sobre imágenes tomadas por cámaras que transmiten en directo, se ha creado y desarrollado una interfaz gráfica a modo de demostrador para cada uno de los algoritmos Edge, Fusion y HOG, descartando Latent SVM por su insuficiencia computacional. Esta interfaz de ha desarrollado con *Qt Designer*, programa que utiliza la biblioteca Qt y que permite el diseño de la aplicación. Gracias a ella se consigue la interacción usuario-algoritmo y una mejor visualización de los resultados del mismo. Además, permite modificar parámetros durante la ejecución de los algoritmos para ajustar la detección de personas en función de la escala, el umbral, la sensibilidad al ruido, la ventana de detección, etc.

A pesar de que el rendimiento de todos los algoritmos depende de la tasa de aciertos o el nivel de confianza para la detección, los detectores Edge y Fusion sólo clasifican objetos (persona/no persona) detectados en etapas previas. En consecuencia, la precisión del sistema estará limitada por los pasos anteriores.

Sin embargo, para HOG y Latent SVM esta limitación depende del escaneado de la imagen. No funciona en tiempo real pero justamente debido a su búsqueda exhaustiva se trata de un algoritmo robusto y con muy buenos resultados.

Parte de este trabajo ha sido presentado en el workshop "Strategies for Object Segmentation, Detection and Tracking in Complex Environments for Event Detection in Video Surveillance and Monitoring", 23-05-2014, y forma parte del proyecto TEC2011-25995 EventVideo (2012-2014).

6.2. Trabajo futuro

Partiendo del trabajo realizado y expuesto en este documento, es evidente la necesidad que existe de seguir trabajando en algoritmos útiles en aplicaciones de vídeo-vigilancia.

El detector Edge es bastante eficiente y trabaja muy bien en tiempo real; de hecho es el más rápido de los que se han estudiado en este trabajo. Sin embargo, sigue presentando una serie de limitaciones respecto a la segmentación de la que depende que pueden mejorarse con trabajo futuro.

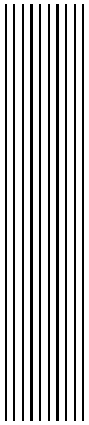
Se propone el estudio, por tanto, de técnicas de modelado de fondo multimodal, disminución del ruido y detección de sombras para refinar la extracción de fondo, pilar básico para el funcionamiento de los algoritmos que funcionan en tiempo real. De hecho, una mejora significativa sería sustituir el segmentador que se ha utilizado en este trabajo por otro que diese mejores resultados, o incluso trabajar en conseguir un ajuste automático de los parámetros del mismo, opción que, sin ninguna duda, facilitaría la segmentación.

Además, es posible añadir otros detectores de personas al desarrollo de este trabajo, opción que ha sido descartada por limitaciones de tiempo.

Como mejora también se plantea optimizar algoritmos que no funcionan en tiempo real, para lo cual se sugiere reducir el espacio de búsqueda espacial en los algoritmos de búsqueda exhaustiva o limitar automáticamente el rango de escalas a analizar.

Aunque en este trabajo no se ha analizado el rendimiento de los algoritmos en términos de precisión y *recall*, gracias a su integración en un marco común (DiVA), como trabajo futuro se podría generar un formato común de resultados de salida y su evaluación con *ground truth*.

Otras propuestas importantes son el estudio de algoritmos de seguimiento o *tracking*, que son más robustos a oclusiones y permiten seguir múltiples objetivos. Aunque en este trabajo no se ha hecho uso de información de movimiento, añadirla a los detectores de personas sería probablemente una gran mejora que añadiría robustez a la detección.



Bibliografía

[1] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 886–893. IEEE, 2005.

[2] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

[3] KN Plataniotis and CS Regazzoni. Visual-centric surveillance networks and services. *IEEE Signal Processing Magazine*, 22(2):12–15, 2005.

[4] M Valera and SA Velastin. Intelligent distributed surveillance systems: a review. *IEE Proccedings on Vision, Image and Signal Processing*, 152(2):192–204, 2005.

[5] Markus Enzweiler and Dariu M Gavrila. Monocular pedestrian detection: Survey and experiments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2179–2195, 2009.

[6] Víctor Fernández-Carbajales, Miguel Ángel García, and José M Martínez. Robust people detection by fusion of evidence from multiple methods. In *Proceedings of the Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS)*, volume 8, pages 55–58, 2008.

[7] Alvaro Garcia-Martin and Jose M Martinez. Robust real time moving people detection in surveillance scenarios. In *Proceedings of Seventh IEEE International Conference Advanced Video and Signal Based Surveillance (AVSS)*, pages 241–247. IEEE, 2010.

[8] Opencv. `http://opencv.org/`.

[9] J. C. SanMiguel and J. M. Martinez. Strategies for object segmentation, detection and tracking in complex environments for event detection in video surveillance and monitoring. *DiVA Documentation D1.2v1. TEC2011-25995 EventVideo*, 2012-2014.

[10] Qt project. `http://qt-project.org/`.

[11] Mykhaylo Andriluka, Stefan Roth, and Bernt Schiele. Pictorial structures revisited: People detection and articulated pose estimation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1014–1021. IEEE, 2009.

[12] Ismail Haritaoglu, David Harwood, and Larry S Davis. Ghost: A human body part labeling system using silhouettes. In *Proceedings of Fourteenth International Conference on Pattern Recognition (ICPR)*, volume 1, pages 77–82. IEEE, 1998.

[13] Prahlad Kilambi, Evan Ribnick, Ajay J Joshi, Osama Masoud, and Nikolaos Papanikolopoulos. Estimating pedestrian counts in groups. *Computer Vision and Image Understanding*, 110(1):43–59, 2008.

[14] Bastian Leibe and Bernt Schiele. Scale-invariant object categorization using a scale-adaptive mean-shift search. In *Pattern Recognition*, pages 145–153. Springer, 2004.

[15] Bo Wu and Ramakant Nevatia. Detection of multiple, partially occluded humans in a single image by bayesian combination of edgelet part detectors. In *Proceedings of Tenth IEEE International Conference on Computer Vision (ICCV)*, volume 1, pages 90–97. IEEE, 2005.

[16] Fengliang Xu and Kikuo Fujimura. Human detection using depth and gray images. In *Proceedings of IEEE Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 115–121. IEEE, 2003.

[17] Jianpeng Zhou and Jack Hoang. Real time robust human detection and tracking system. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Workshops (CVPR Workshops)*, pages 149–149. IEEE, 2005.

[18] Ross Cutler and Larry S. Davis. Robust real-time periodic motion detection, analysis, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):781–796, 2000.

[19] Hedvig Sidenbladh. Detecting human motion with support vector machines. In *Proceedings of the 17th International Conference on Pattern Recognition (ICPR)*, volume 2, pages 188–191. IEEE, 2004.

[20] Tao Zhao and Ramakant Nevatia. Tracking multiple humans in complex situations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1208–1221, 2004.

[21] Ignacio Parra Alonso, David Fernández Llorca, Miguel Ángel Sotelo, Luis Miguel Bergasa, Pedro Revenga de Toro, Jesús Nuevo, Manuel Ocaña, and MA Garcia Garrido. Combination of feature extraction methods for svm pedestrian detection. *IEEE Transactions on Intelligent Transportation Systems*, 8(2):292–307, 2007.

[22] Dariu M Gavrila and Stefan Munder. Multi-cue pedestrian detection and tracking from a moving vehicle. *International journal of computer vision*, 73(1):41–59, 2007.

[23] Edgar Seemann and Bernt Schiele. Cross-articulation learning for robust detection of pedestrians. In *Pattern Recognition*, pages 242–252. Springer, 2006.

[24] Paul Viola, Michael J Jones, and Daniel Snow. Detecting pedestrians using patterns of motion and appearance. In *Proceedings of Ninth IEEE International Conference on Computer Vision (ICCV)*, pages 734–741. IEEE, 2003.

[25] Bo Wu and Ram Nevatia. Detection and tracking of multiple, partially occluded humans by bayesian combination of edgelet based part detectors. *International Journal of Computer Vision*, 75(2):247–266, 2007.

[26] Qiang Zhu, M-C Yeh, Kwang-Ting Cheng, and Shai Avidan. Fast human detection using a cascade of histograms of oriented gradients. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 1491–1498. IEEE, 2006.

[27] Weiming Hu, Tieniu Tan, Liang Wang, and Steve Maybank. A survey on visual surveillance of object motion and behaviors. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 34(3):334–352, 2004.

[28] Andrea Cavallaro and Touradj Ebrahimi. Video object extraction based on adaptive background and statistical change detection. In *Proceedings of Photonics West 2001-Electronic Imaging*, pages 465–475. International Society for Optics and Photonics, 2000.

[29] Michael B Dillencourt, Hanan Samet, and Markku Tamminen. A general approach to connected-component labeling for arbitrary image representations. *Journal of the ACM (JACM)*, 39(2):253–280, 1992.

[30] Ted J Broida and Rama Chellappa. Estimation of object motion parameters from noisy images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (1):90–99, 1986.

[31] Andrea Cavallaro, Olivier Steiger, and Touradj Ebrahimi. Semantic video analysis for adaptive content delivery and automatic description. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(10):1200–1209, 2005.

[32] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[33] Chang Huang, Haizhou Ai, Bo Wu, and Shihong Lao. Boosting nested cascade detector for multi-view face detection. In *Proceedings of International Conference on Pattern Recognitio (ICPR)*, pages 415–418, 2004.

[34] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages I–511. IEEE, 2001.

[35] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proceedings of International Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 886–893, 2005.

[36] Álvaro García-Martín, José M Martínez, and Jesús Bescós. A corpus for benchmarking of people detection algorithms. *Pattern Recognition Letters*, 33(2):152–156, 2012.

[37] Fabricio Tiburzi, Marcos Escudero, Jesús Bescós, and José M Martínez. A ground truth for motion-based video-object segmentation. In *Proceedings of 15th IEEE International Conference on Image Processing (ICIP)*, pages 17–20. IEEE, 2008.