

# COMPARATIVA DE TÉCNICAS DE PROCESAMIENTO DE IMÁGENES PARA LA DETECCIÓN DE PEATONES CON SVM

Álvaro Ruiz Gutiérrez, Lidia Jiménez Soriano, Víctor Flores González, Óscar Menéndez Márquez

29 de abril de 2025

## Resumen

En este trabajo se analiza y compara la eficiencia de cuatro métodos de extracción de características en imágenes: Harris Corner Detection, ORB, SIFT y HOG, aplicados a la detección de peatones mediante un modelo SVM preentrenado. Se estudia la teoría detrás de cada método, extrayendo y explicando sus principios fundamentales de manera didáctica. Posteriormente, se implementan detallando los parámetros utilizados y los efectos que tienen en el rendimiento del modelo. Para evaluar la eficiencia de cada método, se definen métricas estadísticas claves para analizarlas. Se presentan los resultados obtenidos mediante experimentación con un conjunto de datos de imágenes, analizando el impacto de cada técnica en la identificación de peatones. Finalmente, se expone de manera clara la comparativa de los métodos estudiados, destacando sus ventajas y desventajas en función de los resultados obtenidos.

**Palabras clave:** Harris Corner Detection, ORB, SIFT, HOG, procesamiento de imágenes, detección de peatones, SVM.

## 1. Introducción

El procesamiento de imágenes digitales es un campo clave dentro de la visión por computador, con aplicaciones en reconocimiento de patrones, inteligencia artificial y análisis de imágenes. Dentro de este ámbito, la detección de objetos y, en particular, la identificación de peatones, representa un desafío fundamental con implicaciones en seguridad, movilidad y automatización.

Este trabajo se centra en el estudio y comparación de cuatro técnicas de extracción de características utilizadas en la identificación de peatones: **Harris Corner Detection**, **ORB (Oriented FAST and Rotated BRIEF)**, **SIFT (Scale-Invariant Feature Transform)** y **HOG (Histogram of Oriented Gradients)**. Estas técnicas permiten identificar estructuras relevantes en una imagen, facilitando su uso en tareas de reconocimiento y clasificación. Para evaluar su rendimiento, se utilizará un **modelo SVM (Support Vector Machine) preentrenado**, el cual es ideal para tareas de clasificación binaria. Este procesará los descriptores generados por cada método para realizar la identificación de peatones en imágenes.

El **objetivo** principal de este trabajo es ofrecer una explicación didáctica de los métodos estudiados, tanto desde un punto de vista teórico como práctico. Para ello, el desarrollo del proyecto se estructura en cuatro fases clave:

- **Análisis teórico:** Se desarrollan los fundamentos teóricos y matemáticos de cada método, extrayendo las ideas principales de fuentes científicas y técnicas.
- **Implementación práctica:** Se realiza una prueba de concepto para comprender cómo aplicar estos algoritmos en código, detallando los pasos intermedios, el significado de los parámetros y la experimentación con distintas configuraciones.
- **Experimentación:** Con los algoritmos explicados e implementados, se lleva a cabo una experimentación utilizando un modelo SVM, con el objetivo de analizar la eficiencia de cada uno en la tarea de detección de peatones.

- **Presentación didáctica:** Se explican de forma estructurada todos los pasos necesarios para la detección de peatones utilizando cada método, complementando con ejemplos gráficos y análisis de resultados.

Dado que el desarrollo del trabajo está orientado a la investigación y aplicación de técnicas avanzadas no vistas en clase, se apoyará en artículos científicos recientes. La **planificación** y ejecución del proyecto se realizará en un equipo de trabajo compuesto por 4 alumnos, distribuyendo **70 horas por persona**. Además, se utilizarán herramientas de gestión de proyectos para organizar y documentar las tareas realizadas, asegurando un seguimiento adecuado de los avances. Concretamente, se utilizará GitHub como repositorio del código fuente y documentación, Clockify para el registro y seguimiento de horas y tareas, y Microsoft Project para realizar una planificación previa precisa de las fases, hitos y actividades previstas.

## 2. Planteamiento teórico

La detección automatizada de peatones a partir de imágenes digitales es una tarea clave en el campo del procesamiento de imágenes, con aplicaciones relevantes en ámbitos como la seguridad vial, la monitorización del tráfico o los sistemas autónomos.

En este contexto, se estudiarán en detalle cuatro técnicas ampliamente utilizadas para la extracción de características: **Harris**, **ORB**, **SIFT** y **HOG**. Cada uno de estos métodos permite detectar puntos clave o describir regiones relevantes de una imagen, lo que resulta esencial para identificar patrones asociados a la presencia de personas.

En esta sección se profundiza en los fundamentos teóricos y matemáticos de cada uno de estos algoritmos, explicando los principios que los sustentan, su formulación, y analizando sus principales ventajas y limitaciones frente a otros enfoques. Esta base teórica servirá como soporte para su posterior implementación y evaluación experimental.

### 2.1. Harris Corner Detection

El algoritmo **HARRIS** (Harris Corner Detection) es un método ampliamente utilizado en visión por computadora para la detección de esquinas y puntos de interés en imágenes. Fue desarrollado por *Chris Harris* y *Mike Stephens* en 1988 y se basa en la idea de que las esquinas son puntos donde la intensidad de la imagen cambia significativamente en múltiples direcciones. Este algoritmo es especialmente útil para tareas como el seguimiento de objetos, la reconstrucción 3D y la detección de características invariantes a escala y rotación.

#### Fundamentos matemáticos

El algoritmo Harris se basa en varios principios matemáticos que son los pilares de su funcionamiento. A continuación, se presentan algunos de los conceptos clave:

- **Extracción de los autovectores y autovalores:** El algoritmo Harris utiliza la matriz de autocorrelación para identificar esquinas en la imagen. El porque de utilizar la matriz de autocorrelación es que, al calcular los autovectores y autovalores de esta matriz, se pueden identificar puntos donde la intensidad de la imagen cambia significativamente en múltiples direcciones. La imagen que se muestra a continuación ilustra como varía los autovalores y autovectores en función de la intensidad de la imagen.

## Autovectores

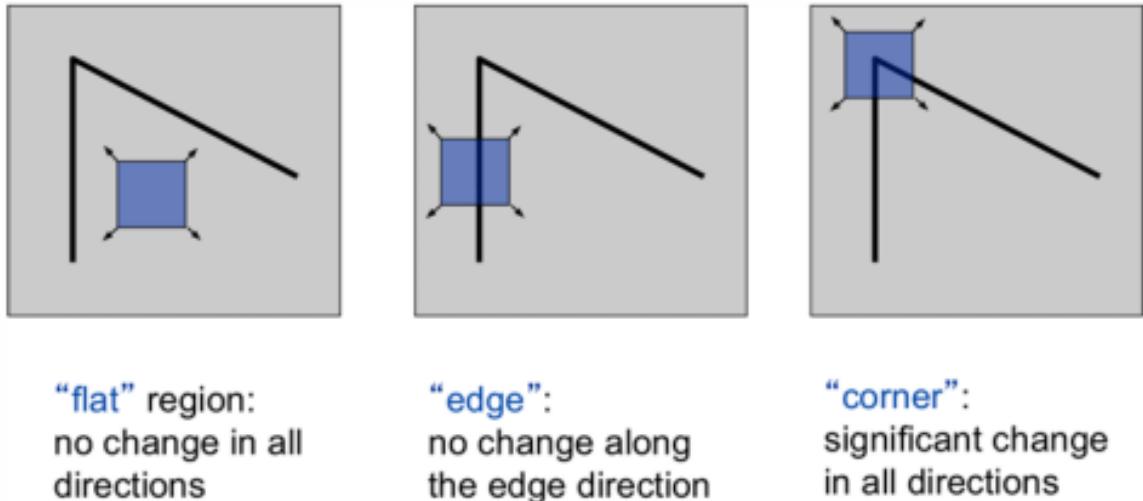


Figura 1: Ilustración sobre autovectores.

- Para el cálculo de dichos autovectores y autovalores, se utiliza la matriz de autocorrelación  $M$  que se define como:

$$M \cdot x = \lambda \cdot x$$

Donde:

- $M$  es la matriz de autocorrelación de la imagen.
- $x$  es nuestra matriz de autovectores.
- $\lambda$  es el llamado autovalor, es decir, cuánto aumenta en cada dirección el autovector.

Los autovectores y autovalores se obtienen resolviendo la siguiente igualdad.

$$\det(M - \lambda I) = 0$$

$$\det(M - \lambda I) = \det \left( \begin{bmatrix} a_{11} - \lambda & a_{12} \\ a_{21} & a_{22} - \lambda \end{bmatrix} \right) = 0$$

Los elementos  $a_n$  son los elementos de la matriz de autocorrelación  $M$  y  $\lambda$  es el autovalor.  $\lambda_{max}$  corresponde al autovalor máximo y  $\lambda_{min}$  al autovalor mínimo, es decir, cuanto aumenta la intensidad de la imagen en cada dirección.

$$(a_{11} - \lambda)(a_{22} - \lambda) - a_{12}a_{21} = 0$$

Como se trata de un polinomio de segundo grado, podemos obtener fácilmente las soluciones:

$$\lambda_{\max} = \frac{1}{2} \left( a_{11} + a_{22} + \sqrt{4a_{12}a_{21} + (a_{11} - a_{22})^2} \right)$$

$$\lambda_{\min} = \frac{1}{2} \left( a_{11} + a_{22} - \sqrt{4a_{12}a_{21} + (a_{11} - a_{22})^2} \right)$$

- **Calculo del gradiente de la imagen:** Debemos calcular el gradiente de la imagen para poder calcular la matriz de autocorrelación. Para cada pixel de la imagen, dicho gradiente se calcula como:

$$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f(x, y)}{\partial x} \\ \frac{\partial f(x, y)}{\partial y} \end{bmatrix}$$

Dado que este cálculo también es relativamente complejo y costoso, podemos aproximararlo de la siguiente manera:

$$\text{Filtro de Sobel} \approx \nabla f(x, y) = \begin{bmatrix} \frac{\partial f(x, y)}{\partial x} \\ \frac{\partial f(x, y)}{\partial y} \end{bmatrix} \approx \begin{bmatrix} I_x \\ I_y \end{bmatrix}$$

**Nota:** El resultado es en realidad una matriz con  $I_x$  y  $I_y$ , pero se ha representado como  $I_x$  e  $I_y$  por simplicidad.

- **Calculo de la matriz de autocorrelación:**

Debemos realizar el siguiente para obtener la matriz de autocorrelación  $M$  para cada pixel de la imagen:

$$A = \sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Para simplificar, evitamos el uso explícito de la función ventana  $w(x, y)$ , y aprovechamos que ya contamos con los valores de  $I_x$  e  $I_y$ , por lo que simplemente calculamos sus productos.

- **Calculo de las notas Harris para cada pixel:**

Finalmente, para cada pixel de la imagen, calculamos la nota Harris  $R$ . Esta nota representa el valor de "esquinidad" de cada pixel, es decir, que tan "esquina" es el pixel, a mayor valor, mayor esquinidad.

$$f(x, y) = \lambda_{\min} \lambda_{\max} - \kappa(\lambda_{\min} + \lambda_{\max})^2$$

**Nota:** El valor  $\kappa$  corresponde al parámetro  $k$  de la ecuación original de Harris, que se utiliza para ajustar la sensibilidad del detector. Un valor comúnmente utilizado es  $k = 0,04$ .

Afortunadamente, el valor de Harris se puede calcular de una manera equivalente pero más sencilla:

$$f(x, y) = \det(M) - \kappa \operatorname{tr}(M)^2 = I_x^2 I_y^2 - I_x I_y I_x I_y - \kappa(I_x^2 + I_y^2)^2$$

Esta formulación es mucho más simple y se puede obtener fácilmente mediante el uso de filtros de gradiente como el filtro de Sobel.

## Ventajas de HARRIS

El algoritmo HARRIS presenta varias ventajas en comparación con otros métodos de detección de características:

- **Robustez frente al ruido:** Gracias al uso de filtros de suavizado y la matriz de autocorrelación, el algoritmo es menos sensible al ruido en la imagen, lo que permite una detección más precisa de esquinas.
- **Alta precisión en la detección de esquinas:** Identifica puntos donde las variaciones de intensidad son altas en múltiples direcciones, lo que corresponde a esquinas reales con muy buena exactitud.

- **Invariancia a la rotación:** El detector es invariante a rotaciones: si rotas la imagen, las esquinas detectadas rotarán de manera consistente. Esto es útil para tareas como emparejamiento de imágenes o reconstrucción 3D.
- **Bajo costo computacional:** El detector usa cálculos de gradientes simples mediante filtros de Sobel, además de cálculos sencillos como las notas Harris, lo que lo hace eficiente en términos de tiempo de procesamiento.
- **Respuesta continua:** La salida del detector es continua, es decir, no se limita a un conjunto binario de píxeles, si no que ofrece una matriz de valores que representan la "esquinidad" de cada pixel. Esto permite ajustar el umbral de detección según las necesidades del problema, además de ofrecer ordenación de los puntos detectados.

## 2.2. ORB (Oriented FAST and Rotated BRIEF)

El algoritmo **ORB** (Oriented FAST and Rotated BRIEF) es una técnica eficiente y robusta para la detección y descripción de características en imágenes. Fue propuesto en 2011 por Ethan Rublee y su equipo como una alternativa rápida a SIFT y SURF, combinando dos técnicas previas: el detector **FAST** y el descriptor **BRIEF**, con mejoras para dotarlos de invariancia a la rotación.

ORB es ampliamente utilizado en aplicaciones en tiempo real, como visión en robótica y realidad aumentada, debido a su bajo coste computacional y buena precisión.

### Fundamentos teóricos

ORB consta de dos partes principales: el detector de puntos clave (keypoints) y el descriptor binario. A continuación se detallan los pasos fundamentales:

- **Detección de puntos clave con FAST:** ORB utiliza el algoritmo **FAST** (Features from Accelerated Segment Test) para detectar esquinas de forma eficiente. El criterio de FAST consiste en verificar si un píxel del centro de una ventana de 16 píxeles en un círculo presenta diferencias significativas de intensidad con respecto a un número fijo de píxeles del contorno. Es rápido pero no es invariante a la rotación.
- **Mejora de FAST con Harris Score:** Para ordenar y seleccionar los puntos más relevantes, ORB emplea la **Harris corner measure**, descartando puntos débiles y manteniendo los más robustos para su posterior descripción.
- **Asignación de orientación:** Para dotar a FAST de invariancia rotacional, ORB calcula la orientación dominante en la vecindad del punto clave usando momentos de orden 1. Dado un patch centrado en el keypoint, la orientación se calcula como:

$$\theta = \arctan \left( \frac{m_{01}}{m_{10}} \right)$$

donde los momentos se definen como:

$$m_{pq} = \sum_x \sum_y x^p y^q I(x, y)$$

- **Generación del descriptor BRIEF rotado:** ORB utiliza **BRIEF** (Binary Robust Independent Elementary Features) para describir el entorno de cada punto. BRIEF genera un vector binario comparando la intensidad de pares de píxeles dentro de una ventana. Para que sea invariante a la rotación, ORB rota la máscara de BRIEF según el ángulo calculado previamente.
- **Reducción de correlación en BRIEF:** ORB introduce un aprendizaje offline para seleccionar un subconjunto de pruebas de comparación que sean menos correlacionadas entre sí, mediante el análisis de varianza y correlación de las pruebas binarias.

## Ventajas de ORB

ORB combina velocidad, eficiencia y robustez, lo que lo convierte en una opción ideal para sistemas en tiempo real. Entre sus principales ventajas destacan:

- **Alta velocidad de cómputo:** Es mucho más rápido que SIFT y SURF, ya que utiliza técnicas binarizadas y eficientes computacionalmente.
- **Invariancia a la rotación:** Gracias a la estimación de orientación y rotación del descriptor BRIEF, ORB puede detectar características robustas a la rotación de la imagen.
- **Buena discriminación:** Aunque los descriptores binarios son más ligeros, ORB conserva una buena capacidad de discriminación, útil en tareas de emparejamiento y reconocimiento.
- **Libre de patentes:** A diferencia de SIFT o SURF, ORB es completamente libre de uso, lo que lo hace muy atractivo en aplicaciones comerciales o académicas sin restricciones de licencia.

## 2.3. SIFT (Scale-Invariant Feature Transform)

El algoritmo **SIFT** (Scale-Invariant Feature Transform) es una técnica ampliamente utilizada en visión por computadora para la detección y descripción de características en imágenes. Fue desarrollado por *David Lowe* en 1999 y destaca por su capacidad de identificar puntos clave en una imagen de manera robusta frente a transformaciones como cambios de escala, rotación e iluminación. Gracias a estas propiedades, SIFT se ha convertido en una herramienta fundamental para aplicaciones como el reconocimiento de objetos, la detección de patrones y el emparejamiento de imágenes.

### Fundamentos teóricos

El algoritmo SIFT se basa en la extracción de puntos característicos (keypoints) de una imagen y la generación de descriptores asociados a ellos. Este proceso se realiza en varias etapas, las cuales garantizan la invariancia a escala y a rotación.

- **Construcción de la Pirámide Gaussiana:** Se aplican sucesivas convoluciones con un filtro gaussiano para generar versiones suavizadas de la imagen en diferentes escalas. Este proceso permite analizar la imagen a distintos niveles de detalle, lo que hace que los puntos característicos sean robustos a cambios de tamaño y desenfoque.
- **Construcción de la Pirámide de Diferencia de Gaussianos (DoG):** Se calculan diferencias entre imágenes suavizadas consecutivas de la pirámide gaussiana. Esta operación permite resaltar bordes y estructuras significativas en la imagen, facilitando la detección de puntos clave.

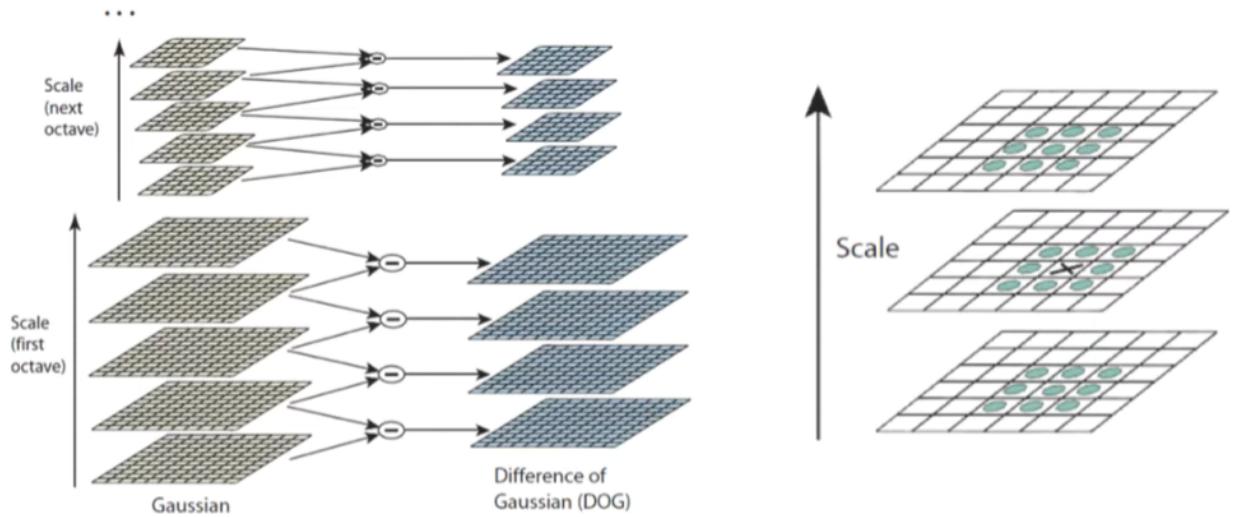


Figura 2: Proceso de la pirámide de diferencia gaussiana. Fuente: [https://docs.opencv.org/3.4/da/df5/tutorial\\_py\\_sift\\_intro.html](https://docs.opencv.org/3.4/da/df5/tutorial_py_sift_intro.html)

- **Detección de keypoints:** Se identifican los puntos extremos en la pirámide DoG, comparando cada píxel con sus vecinos en los niveles superior e inferior de la pirámide, así como en su entorno local. Los puntos seleccionados se someten a un proceso de refinamiento para descartar aquellos con bajo contraste o situados en bordes poco definidos.
- **Asignación de orientación:** A cada keypoint se le asigna una orientación dominante basada en la distribución de gradientes en su vecindad. Para ello, se calcula el gradiente horizontal y vertical, y se calcula el histograma de orientaciones del gradiente en una ventana centrada en el punto en la escala en la que se ha detectado. La orientación se discretiza en 36 intervalos y se van acumulando las amplitudes del gradiente en cada ángulo detectado. Esto permite que el algoritmo sea invarianta a rotaciones.

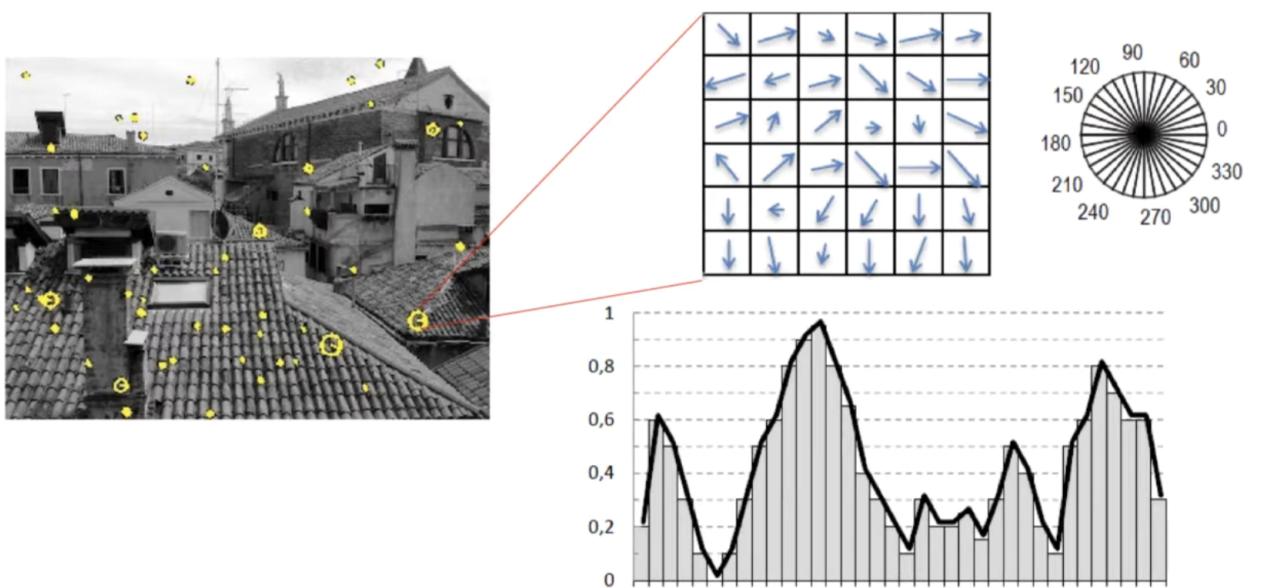


Figura 3: Proceso de orientación de un keypoint. Fuente: <https://tinyurl.com/wyv9vczp>

- **Generación del descriptor SIFT:** Para representar de manera robusta un keypoint, se genera un descriptor basado en la distribución de los gradientes dentro de una ventana

centrada en dicho punto. Esta región se divide en una cuadrícula de  $4 \times 4$  celdas, formando un total de 16 subregiones. En cada subregión se calcula un histograma de orientaciones de gradientes con 8 direcciones posibles. Finalmente, los 16 histogramas se concatenan en un único vector de características de 128 dimensiones, dado que proviene de la combinación de las 16 subregiones ( $4 \times 4$ ) y las 8 direcciones ( $16 \times 8 = 128$ ). Este descriptor permite identificar y comparar keypoints de manera robusta frente a variaciones en escala, rotación e iluminación.

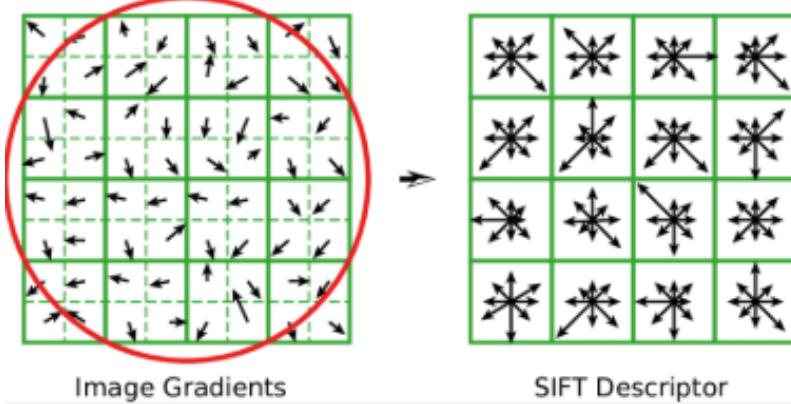


Figura 4: Keypoint Descriptor. Fuente: <https://tinyurl.com/3ufntmbn>

## Fundamentos matemáticos

El algoritmo SIFT se basa en varios principios matemáticos que garantizan su eficacia en la detección y descripción de características. A continuación, se presentan algunos de los conceptos clave:

- **Filtro Gaussiano:** La función gaussiana en dos dimensiones está definida por:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

donde  $\sigma$  controla el nivel de suavizado aplicado a la imagen.

- **Diferencia de Gaussianos (DoG):** Se define como la resta entre dos imágenes suavizadas con diferentes valores de  $\sigma$ :

$$DoG(x, y, \sigma) = G(x, y, \sigma_2) - G(x, y, \sigma_1)$$

Esta operación resalta los cambios de intensidad y permite la detección de keypoints.

- **Cálculo del gradiente:** Para determinar la orientación de un keypoint, se calcula el gradiente de la imagen en cada píxel utilizando:

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$$

$$\theta(x, y) = \tan^{-1}\left(\frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)}\right)$$

donde  $m(x, y)$  representa la magnitud del gradiente y  $\theta(x, y)$  su dirección.

- **Generación del descriptor SIFT:** El descriptor de cada keypoint como se explicó anteriormente, se forma a partir de histogramas de gradientes calculados en una ventana de  $16 \times 16$  píxeles alrededor del punto de interés. La ventana se subdivide en  $4 \times 4$  celdas y en cada celda se calcula un histograma de 8 orientaciones, generando un vector de:

$$4 \times 4 \times 8 = 128$$

dimensiones, que representa la información local de la imagen.

## Ventajas de SIFT

El algoritmo SIFT presenta varias ventajas en comparación con otros métodos de detección de características:

- **Invariancia a escala y rotación:** Gracias a la construcción de la pirámide gaussiana y la asignación de orientación, los keypoints detectados son robustos a cambios de tamaño y rotación en la imagen.
- **Robustez:** Los descriptores SIFT tienen una alta capacidad para diferenciar objetos en diferentes condiciones de iluminación y contraste.
- **Resistencia a ruido:** Debido a su método de detección basado en diferencias de gaussianos y análisis de gradientes, SIFT es menos sensible al ruido y sigue funcionando incluso cuando partes de la imagen están ocultas.
- **Aplicabilidad en reconocimiento de patrones y emparejamiento de imágenes:** El uso de descriptores de 128 dimensiones permite realizar comparaciones eficientes entre imágenes, facilitando tareas como el reconocimiento de objetos.

## 2.4. HOG (Histogram of Oriented Gradients)

El descriptor **HOG**, propuesto por *Dalal y Triggs* en 2005, es una técnica ampliamente utilizada para la detección de humanos en imágenes. Su éxito se basa en la idea de que la apariencia y la forma de los objetos, y en particular de las personas, pueden caracterizarse de manera efectiva mediante la distribución de gradientes o bordes. Esta representación es robusta frente a cambios en la iluminación y pequeñas variaciones en la geometría de la imagen, lo que la convierte en una herramienta idónea para alimentar clasificadores como el SVM.

### Fundamentos teóricos y matemáticos

El algoritmo HOG se fundamenta en los siguientes conceptos:

- **Cálculo de Gradientes:** La primera etapa del proceso consiste en convertir la imagen a escala de grises y calcular, para cada píxel, la magnitud y la orientación del gradiente. Utilizando operadores como el de Sobel, se obtienen dos imágenes: una con las derivadas en la dirección horizontal ( $G_x$ ) y otra en la vertical ( $G_y$ ). A partir de estas, se calcula la magnitud  $M$  y la orientación  $\theta$  según:

$$M(x, y) = \sqrt{G_x(x, y)^2 + G_y(x, y)^2}$$

$$\theta(x, y) = \arctan\left(\frac{G_y(x, y)}{G_x(x, y)}\right) \times \frac{180}{\pi}$$

Los ángulos se ajustan al rango **[0, 180°]** para mantener la invarianza ante la dirección (ya que un borde tiene la misma apariencia tanto en una dirección como en la opuesta).

- **División en Celdas:** La imagen se divide en pequeñas regiones (celdas), típicamente de  $8 \times 8$  o  $16 \times 16$  píxeles, en las que se construye un histograma de orientaciones. Cada celda produce un histograma donde se acumulan las magnitudes de los gradientes de los píxeles, distribuidas en un número fijo de intervalos (por ejemplo, 9 bins, donde cada bin abarca 20°).
- **Normalización de Bloques:** Para reducir los efectos de variaciones en iluminación y contraste, se agrupan las celdas en bloques solapados (por ejemplo,  $2 \times 2$  celdas). Cada bloque se aplana en un vector y se normaliza (usualmente mediante la normalización L2 o L2-Hys) según:

$$v_{\text{norm}} = \frac{v}{\|v\|_2 + \epsilon}$$

donde  $\mathbf{v}$  es el vector formado por la concatenación de los histogramas de las celdas del bloque y  $\epsilon$  es una pequeña constante para evitar la división por cero. La normalización asegura que el descriptor sea menos sensible a cambios en la iluminación.

- **Concatenación del Descriptor HOG:** Los vectores normalizados de todos los bloques se concatenan para formar el descriptor HOG final, que representa de forma global la distribución de gradientes en la imagen. Este descriptor es utilizado para alimentar el clasificador (SVM) que se encarga de diferenciar entre imágenes con y sin peatones.

### Ventajas de HOG

El descriptor HOG presenta varias ventajas en comparación con otros métodos de detección de características:

- **Robustez a Variaciones de Iluminación:** La normalización de bloques reduce el impacto de cambios en el brillo y contraste.
- **Invarianza a Pequeñas Variaciones Geométricas:** La estructura local capturada por las celdas y bloques permite que el descriptor sea relativamente invariante a pequeñas transformaciones, desplazamientos o deformaciones.
- **Simplicidad y Eficiencia:** HOG es relativamente sencillo de implementar y, a pesar de requerir cálculos intensivos de gradientes, puede optimizarse para aplicaciones en tiempo real.

### 3. Implementación

#### 3.1. Tecnologías

Para este proyecto se ha utilizado Python como lenguaje de programación junto con varias librerías especializadas en procesamiento de imágenes, aprendizaje automático y visualización de datos. A continuación, se detallan las principales tecnologías empleadas:

- **OpenCV** ([web oficial](#)): Biblioteca de código abierto para visión por computadora y procesamiento de imágenes. Se utilizó para la lectura, preprocesamiento y transformaciones de las imágenes.
- **Numpy** ([web oficial](#)): Biblioteca especializada en cálculo numérico y gestión de matrices multidimensionales. Se usó para la manipulación de datos y cálculos matemáticos en la transformación de imágenes.
- **Matplotlib** ([web oficial](#)): Biblioteca de visualización de datos. Se empleó principalmente para representar imágenes, mostrar los puntos de interés detectados y visualizar los resultados obtenidos.
- **Scikit-image** ([web oficial](#)): Biblioteca de procesamiento de imágenes que proporciona herramientas adicionales para realizar operaciones de filtrado, detección de bordes y transformaciones de imágenes.
- **Scikit-learn** ([web oficial](#)): Biblioteca de aprendizaje automático utilizada para entrenar y evaluar el modelo SVM que predice la presencia de peatones en las imágenes procesadas.

#### 3.2. Técnicas y Algoritmos Empleados

A lo largo del proyecto se implementaron varias técnicas de procesamiento de imágenes y detección de características, con el objetivo de extraer información relevante para alimentar un modelo de clasificación basado en **SVM (Support Vector Machine)**. A continuación, se detallan los algoritmos utilizados y su tratamiento sobre las imágenes:

##### 3.2.1. Harris Corner Detection

A continuación se detalla el procedimiento que realiza el algoritmo para la obtención de las esquinas de una imagen.

###### Paso 1: Cargar librerías y la imagen original

Ya que estamos trabajando con diversas librerías, debemos importarlas. Además se debe cargar la imagen con la que se va a trabajar.

Imagen Original



Figura 5: Imagen original cargada.

#### Paso 2: Cargar imagen en escala de grises

Es importante cargar la misma imagen en escala de grises. Aunque podríamos aplicar el algoritmo a cada una de los canales de color, esto no tendría sentido, ya que el algoritmo busca esquinas y bordes, y estos no dependen del color de la imagen, si no de la intensidad de la misma, la cual aparece reflejada en la escala de grises. Aunque otros motivos no tan evidentes son:

- **Reducción de ruido:** Al trabajar con una sola capa, se reduce el ruido y la complejidad de la imagen, lo que facilita la detección de características.
- **Invarianza al color:** La detección de esquinas y bordes no depende del color, sino de los cambios en la intensidad de la imagen. Al convertir a escala de grises, se eliminan las variaciones de color que no son relevantes para la detección.
- **Eficiencia computacional:** Al trabajar con un solo canal en lugar de tres, se reduce la carga computacional drásticamente.
- **Detección de características:** Los cambios en intensidad es la base de la detección de esquinas en este algoritmo y, el canal que mas información mantiene acerca de los cambios en intensidad, es el gris.

Esta es la imagen original convertida a escala de grises.



Figura 6: Imagen original convertida a escala de grises.

### Paso 3: Configuración de parámetros del algoritmo

El algoritmo Harris requiere la configuración de varios parámetros clave que afectan la detección de características en la imagen. Estos son definidos dentro de un constructor diseñado para su correcto funcionamiento. Dichos parámetros son:

- **window\_size = 5:** Tamaño de la ventana utilizada para aplicar supresión de no máximos.
- **$\kappa = 0.04$ :** Parámetro que controla la sensibilidad del detector a esquinas y bordes.
  - **Valores bajos de  $\kappa \approx 0,02$ :**
    - Favorece la detección de esquinas claramente definidas.
    - Menos sensible al ruido.
  - **Valores altos de  $\kappa \approx 0,15$ :**
    - Hace que el algoritmo sea más sensible a bordes y texturas.
    - Aumenta el riesgo de falsas detecciones debido al ruido.
- **gaussbox = 3:** Es el tamaño del filtro gaussiano usado para suavizar la imagen.
- **threshold = 0.1:** Umbral para filtrar las esquinas detectadas, eliminando aquellas con valores bajos.

Para la implementación, se ha utilizado la clase correspondiente en el módulo `algoritmos/harris.py`.

### Paso 4: Obtener el gradiente de cada pixel de la imagen

A continuación calculamos el gradiente de cada píxel de la imagen, utilizando el filtro Sobel. El filtro Sobel es un operador de detección de bordes que calcula la derivada de la imagen en las direcciones horizontal y vertical.

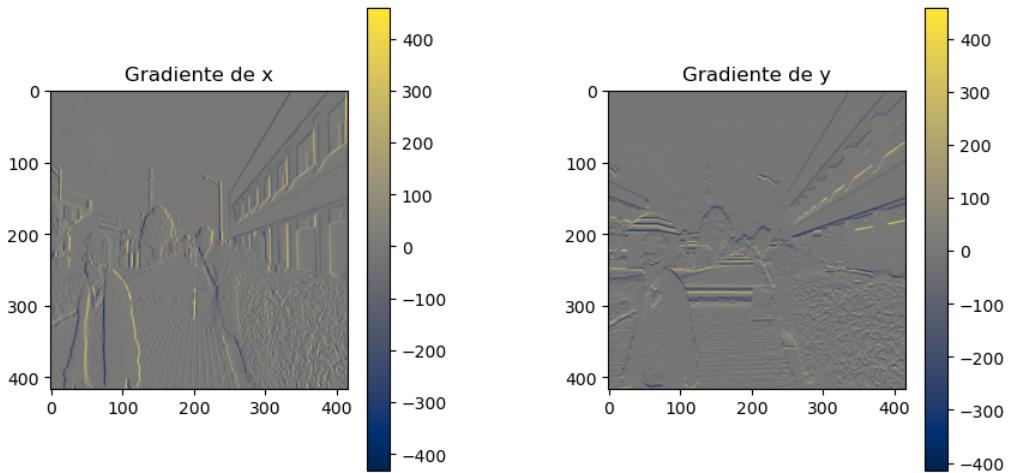


Figura 7: imagen tras aplicar un filtro Sobel.

El resultado nos ilustra claramente los bordes de la imagen, pero no nos da información acerca de las esquinas.

#### Paso 5: Calculo de la Matriz de autocorrelación

Debemos calcular la matriz de autocorrelación  $M$  para cada pixel de la imagen. Dado que este cálculo es relativamente complejo y costoso y, sabiendo que ambas imágenes obtenidas en el paso anterior corresponden con el gradiente de la imagen original en cada dirección, es decir,  $I_x$  e  $I_y$ , podemos calcular la matriz de autocorrelación de la siguiente manera:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Para simplificar y, dado que no tiene sentido su uso en nuestro en particular, evitamos el uso explícito de la función ventana  $w(x,y)$ , y aprovechamos que ya contamos con los valores de  $I_x$  e  $I_y$ , por lo que simplemente calculamos sus productos.

$$M = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Nota: El uso de la función ventana  $w(x,y)$  es imprescindible para el cálculo de la matriz de autocorrelación para cada píxel. Como en nuestro caso hemos calculado los gradientes de la imagen mediante un filtro Sobel, podemos prescindir de su uso, ya que el resultado es el mismo.

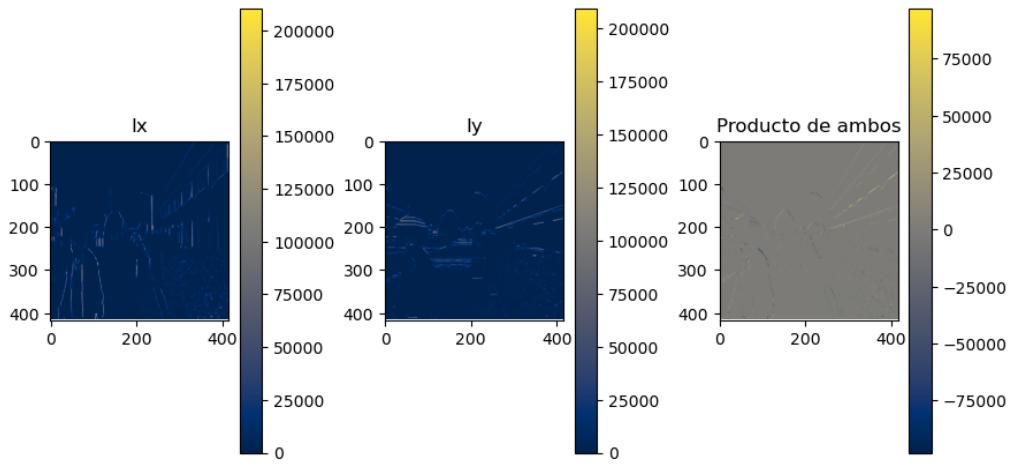


Figura 8: Matrices  $I_x^2$ ,  $I_y^2$  y  $I_x I_y$ .

#### Paso 6: Aplicar un filtro gausiano

Aplicamos un filtro gausiano cada una de las matrices para suavizar y eliminar el ruido.

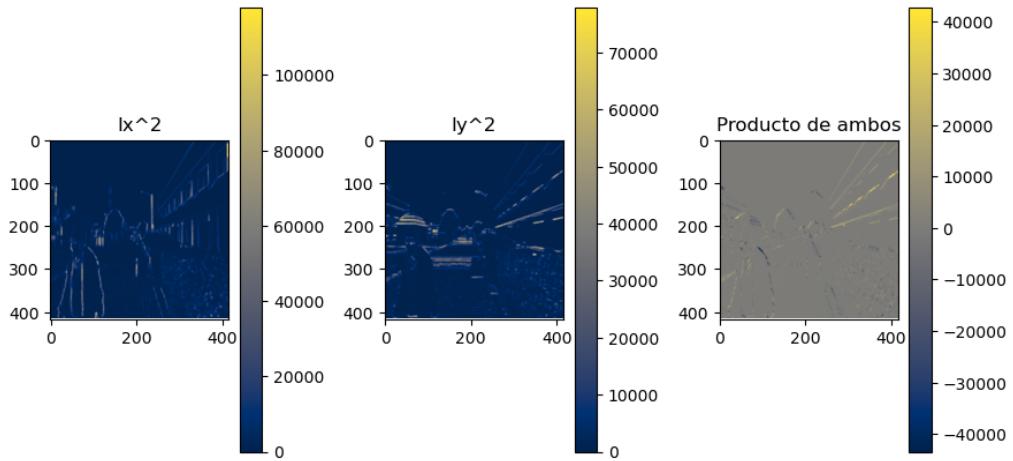


Figura 9: Matrices  $I_x^2$ ,  $I_y^2$  y  $I_x I_y$  tras aplicar un filtro gausiano.

#### Paso 7: Calculo de las notas Harris para cada pixel

Simplemente realizamos el siguiente cálculo para obtener la nota Harris  $R$  para cada pixel de la imagen:

$$f(x, y) = \det(M) - \kappa \operatorname{tr}(M)^2 = I_x^2 I_y^2 - I_x I_y \cdot I_x I_y - \kappa(I_x^2 + I_y^2)^2$$

El resultado de este cálculo es una matriz de la misma dimensión que la imagen original, donde cada pixel tiene un valor que representa su "esquinidad". A mayor valor, mayor esquinidad.

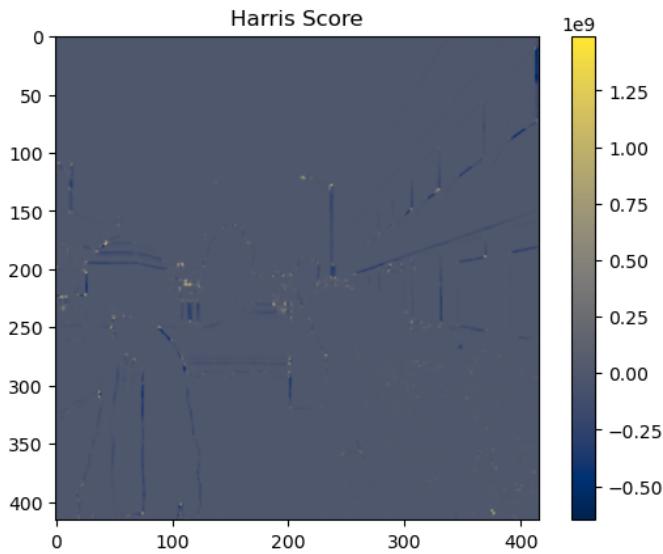


Figura 10: Matriz de notas Harris.

#### Paso 8: Normalizamos y aplicamos un umbral del 10 %

El objetivo es normalizar la matriz de las notas Harris para que se encuentren entre valores 0 y 255 y aplicar un umbral del 10 % para eliminar los valores irrelevantes. Este umbral, tal como se ha mencionado anteriormente, se puede ajustar según las necesidades del problema.

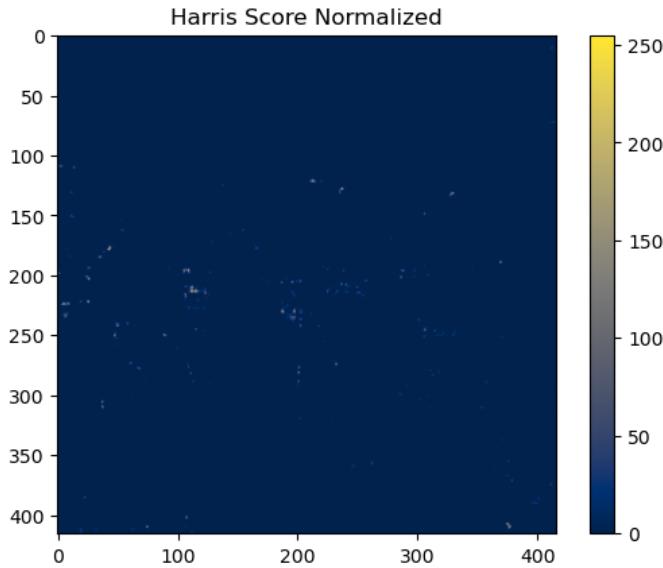


Figura 11: Matriz de notas Harris Normalizada.

#### Paso 9: Aplicar supresión de no máximos

Por último, debemos aplicar supresión de no máximos para sólo mantener a las esquinas con mayor nota. Supresión de no máximos consiste en aplicar a la matriz de notas, una función ventana que oscila en cada uno de los píxeles, y compara si el valor del píxel central es el mayor de entre sus vecinos.

Explíquemoslo con más detalle:

- Una ventana de tamaño `window_size` recorre cada píxel de la imagen.
- Esta comprueba si el píxel central es el mayor de sus vecinos.
- En el caso que sí lo sea, se añade dicho valor a una matriz resultado final. Esto se hace para comparar adecuadamente todos los píxeles con sus vecinos.
- El resultado final es una matriz que contiene solo los píxeles aceptados por la supresión de no máximos.

Este paso permite extraer sólo aquellos píxeles que son los que tiene mayor nota en su vecindario, y por lo tanto, lo que son más susceptibles a ser esquinas viables.

El resultado final es el siguiente:



Figura 12: Resultado Final.

Esta matriz final contiene los píxeles que han sido aceptados por la supresión de no máximos, y por lo tanto, son los que se consideran esquinas viables. Además, el valor de cada pixel es la nota Harris de dicho píxel, que indica su "esquinidad", lo cual es útil para determinar la calidad de la esquina y, por lo tanto, su viabilidad como punto de interés.

### 3.2.2. ORB (Oriented FAST and Rotated BRIEF)

#### Implementación práctica de ORB

La implementación práctica de ORB se ha realizado utilizando OpenCV y módulos desarrollados específicamente para este proyecto. A continuación, se detallan los pasos seguidos, ilustrados con los resultados intermedios obtenidos:

##### Paso 1: Carga y preprocessamiento de la imagen

Se carga una imagen de prueba y se convierte a escala de grises, ya que ORB trabaja únicamente sobre intensidades de píxeles:



Figura 13: Imagen original utilizada para la detección de características.

### Paso 2: Suavizado y generación de la pirámide de escalas

ORB utiliza una pirámide de escalas para lograr invariancia a tamaño. Antes de construir cada nivel de la pirámide, se aplica un suavizado Gaussiano para reducir ruido:



Figura 14: Aplicación de filtro Gaussiano con diferentes valores de  $\sigma$ .

Posteriormente, se genera la pirámide de imágenes:



Figura 15: Pirámide de escalas construida a partir de la imagen original.

### Paso 3: Detección inicial de puntos clave (FAST)

Se emplea el detector FAST para localizar esquinas de forma rápida:



Figura 16: Puntos clave detectados mediante el detector FAST.

#### Paso 4: Filtrado de keypoints usando respuesta de Harris

Para mejorar la calidad de los puntos clave detectados, se calcula la respuesta de Harris, descartando puntos poco significativos:



Figura 17: Keypoints tras aplicar filtrado basado en la respuesta de Harris.

#### Paso 5: Asignación de orientación usando momentos de imagen

Se asigna una orientación dominante a cada keypoint calculando los momentos del parche local:



Figura 18: Puntos clave con orientación asignada mediante cálculo de momentos.

#### Paso 6: Cálculo de descriptores rBRIEF

Una vez asignada la orientación, se calculan los descriptores rBRIEF:

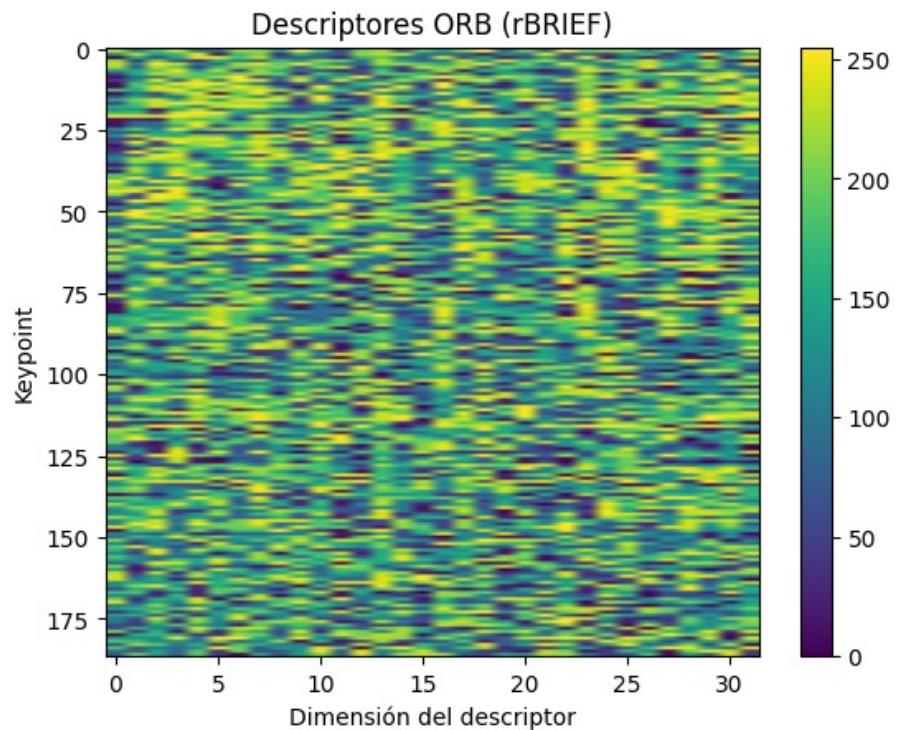


Figura 19: Visualización de la matriz de descriptores generados con rBRIEF.

## Paso 7: Entrenamiento de un clasificador SVM

Los descriptores generados se utilizan para entrenar un modelo SVM que distingue entre imágenes con presencia o ausencia de peatones:

- Se entrenó el clasificador utilizando conjuntos de imágenes etiquetadas.
- El modelo se guardó en el fichero `svm_orb.pkl`.

## Paso 8: Evaluación del modelo

Finalmente, el clasificador SVM se evaluó sobre conjuntos de prueba separados, calculando métricas de rendimiento como precisión y tasa de error.

ORB ha demostrado ser una técnica eficiente tanto en velocidad como en capacidad de detección robusta para el reconocimiento de peatones, logrando una buena relación entre coste computacional y rendimiento.

### 3.2.3. SIFT (Scale-Invariant Feature Transform)

El algoritmo **SIFT** es un método robusto para detectar características invariantes a escala y rotación en una imagen. Fue utilizado para extraer puntos de interés y construir descriptores característicos que permitieran la identificación de patrones en las imágenes.

### 3.2.4. Proceso de Aplicación de SIFT

A continuación, se detallan los pasos seguidos en la implementación del algoritmo SIFT:

#### Paso 1: Cargar y Mostrar la Imagen

Antes de aplicar el algoritmo, se importan las librerías necesarias y se carga la imagen con la que se va a trabajar.



Figura 20: Imagen original cargada. Fuente: Elaboracion propia.

#### Paso 2: Conversión de la Imagen a Escala de Grises

La imagen se convierte a escala de grises, ya que SIFT opera sobre la intensidad de los píxeles en lugar de los colores, lo que mejora la estabilidad de la detección de características.



Figura 21: Conversión a escala de grises. Fuente: Elaboracion propia.

### Paso 3: Configuración del Detector SIFT

El detector **SIFT** requiere la configuración de varios parámetros clave que afectan la detección de características en la imagen:

- **nfeatures = 5:** Número máximo de características a detectar. Si se establece en 0, se detectan todas las posibles características.
- **nOctaveLayers = 3:** Número de capas por octava en la pirámide Gaussiana.
- **contrastThreshold = 0.04:** Umbral para descartar características de bajo contraste.
- **edgeThreshold = 10:** Umbral para eliminar puntos en bordes poco definidos.
- **sigma = 1.6:** Desviación estándar inicial para el filtro Gaussiano.

Para la implementación, se ha utilizado la clase correspondiente en el módulo `algoritmos/sift.py`.

En las pruebas iniciales, se estableció `nfeatures` en **5** para reducir la cantidad de puntos clave detectados, facilitando la visualización y comprensión teórica del proceso.

### Paso 4: Construcción de la Pirámide Gaussiana

La pirámide Gaussiana se genera aplicando convoluciones sucesivas con filtros gausianos de distinta desviación estándar ( $\sigma$ ), reduciendo la resolución de la imagen en cada octava. Este proceso permite analizar la imagen a diferentes escalas para detectar características invariantes.

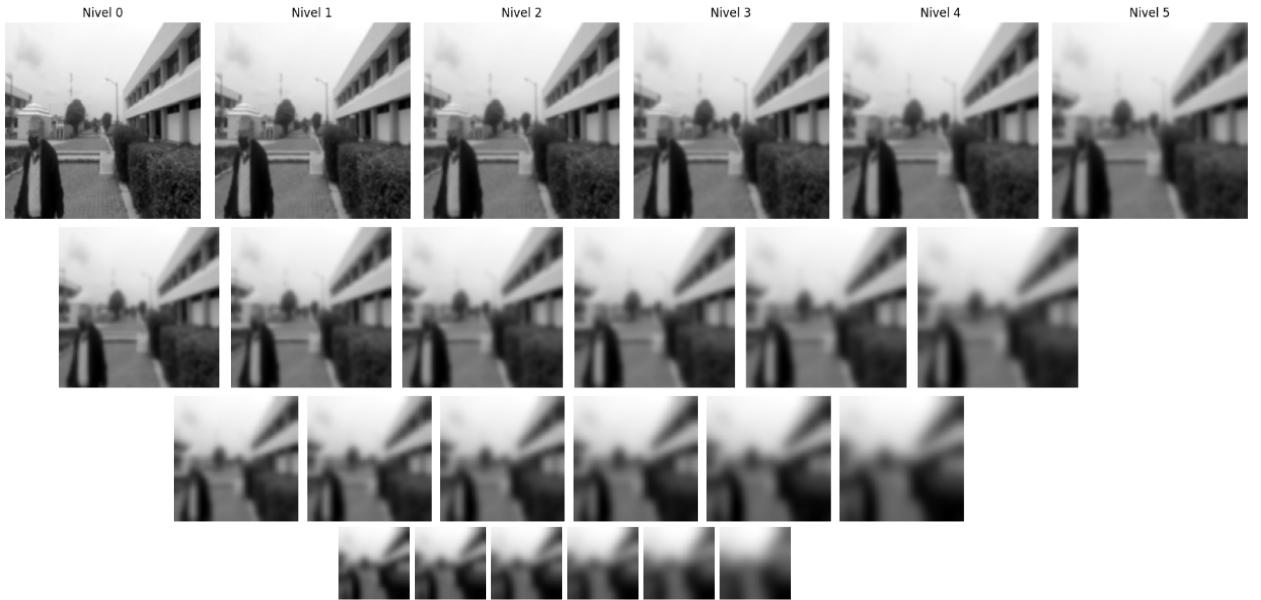


Figura 22: Construcción de la Pirámide Gaussiana. Fuente: Elaboracion propia.

#### **Paso 5: Construcción de la Pirámide de Diferencia de Gaussianos (DoG)**

Después de generar la pirámide Gaussiana, se construye la pirámide de Diferencia de Gaussianos (DoG) restando imágenes consecutivas de la pirámide. Esto resalta las variaciones de intensidad entre niveles de suavizado, facilitando la detección de puntos de interés.

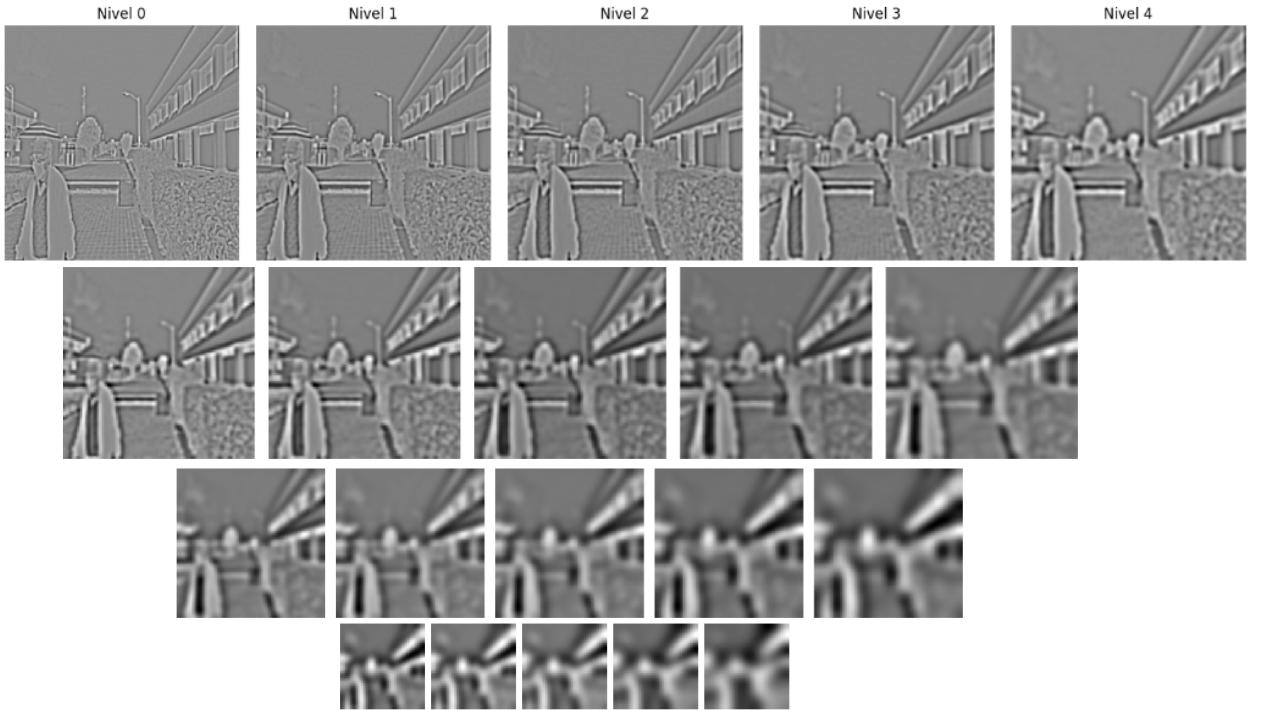


Figura 23: Construcción de la Pirámide de Diferencia de Gaussianos (DoG). Fuente: Elaboracion propia.

#### **Paso 6: Detección de Keypoints**

Los keypoints se detectan identificando los extremos locales en la pirámide DoG. Se comparan píxeles con sus vecinos en los niveles superiores e inferiores de la pirámide para encontrar puntos máximos y mínimos. Como resultado, obtenemos 5 keypoints debido a la configuración propuesta

anteriormente.



Figura 24: Detección de 5 keypoints en la imagen. Fuente: Elaboracion propia.

En el caso de aplicar el valor 0 al parámetro `nfeatures`, se detectarían todas las características posibles en la imagen, lo que aumentaría significativamente el número de keypoints detectados, como se observa en la siguiente imagen:



Figura 25: Detección de todos los keypoints con `nfeatures = 0`. Fuente: Elaboracion propia.

#### Paso 7: Asignación de Orientación a los Keypoints

Cada keypoint recibe una orientación basada en los gradientes locales de la imagen. Esto permite que los descriptores sean invariables a la rotación.

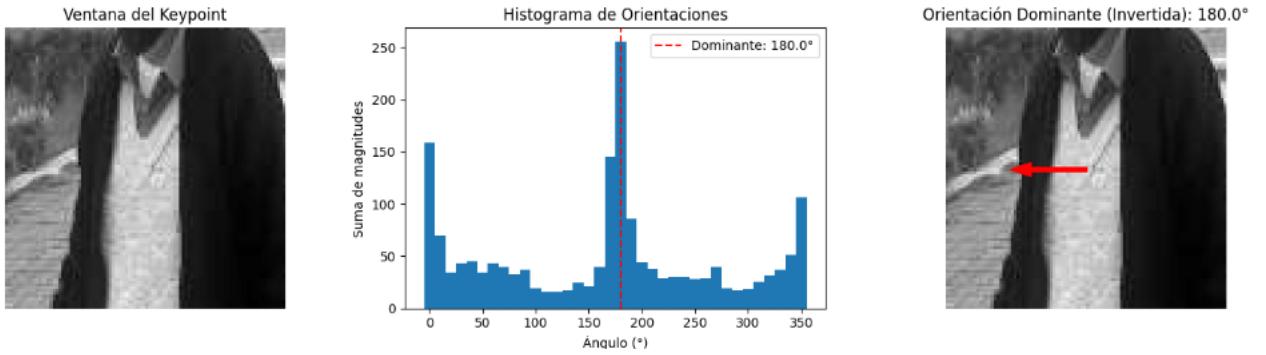


Figura 26: Orientación asignada a cada keypoint. Fuente: Elaboracion propia.

#### Paso 8: Visualización de los Keypoints con su Orientación

Los keypoints detectados y sus orientaciones asignadas se visualizan para evaluar la robustez del algoritmo.

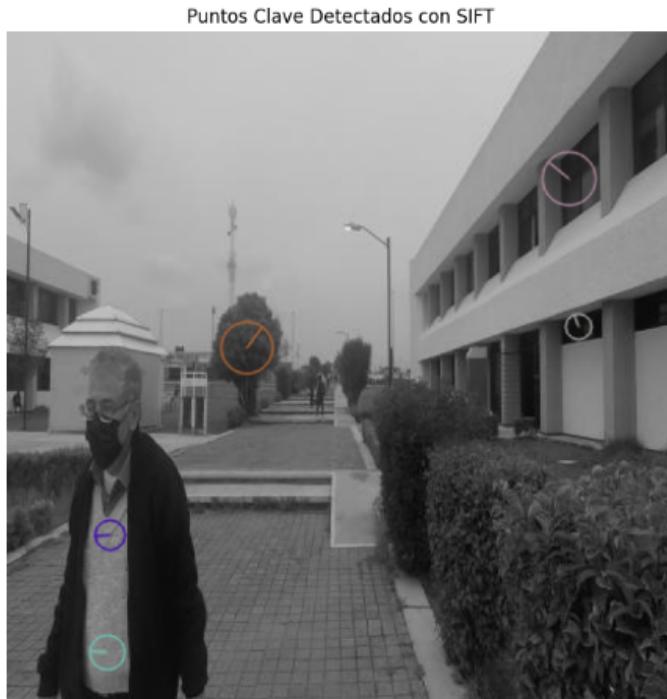


Figura 27: Visualización de keypoints con orientación. Fuente: Elaboracion propia.

#### Paso 9: Cálculo de Descriptores

Cada keypoint se describe mediante un vector de características basado en la distribución de gradientes en su vecindad.

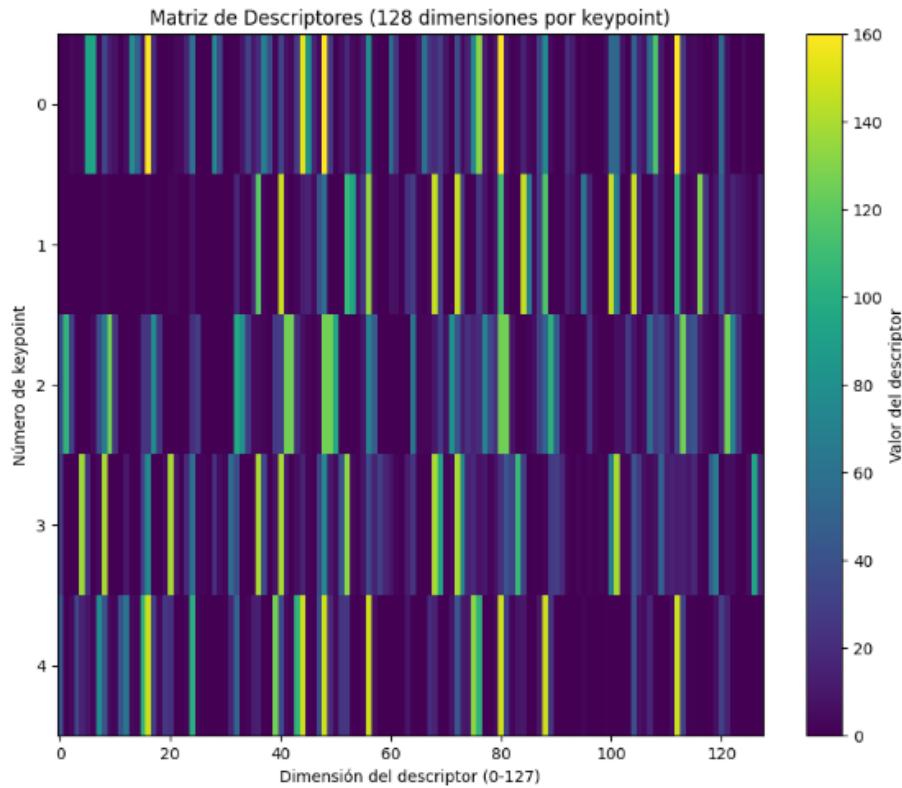


Figura 28: Ejemplo de descriptor SIFT calculado. Fuente: Elaboracion propia.

Estos descriptores serán los que permitan al modelo de aprendizaje automático emparejar características entre diferentes imágenes, facilitando tareas como el reconocimiento de objetos y la detección de peatones en escenas complejas.

Para comprobar la invarianza del algoritmo frente a cambios de escala, rotación y perspectiva, se ha implementado una función en el cuaderno de trabajo que aplica distintas transformaciones a una imagen, incluyendo modificaciones en la escala, rotación, iluminación y contraste. Esto permite observar que el algoritmo es capaz de detectar un número muy similar de puntos clave (keypoints) en las distintas versiones de la imagen, calculando descriptores prácticamente idénticos en cada caso.

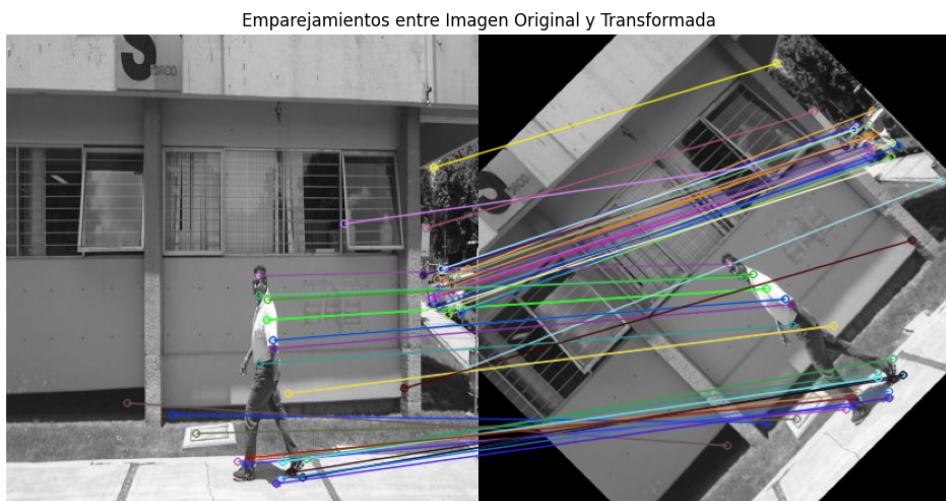


Figura 29: Visualización de keypoints en las dos imágenes. Fuente: Elaboracion propia.

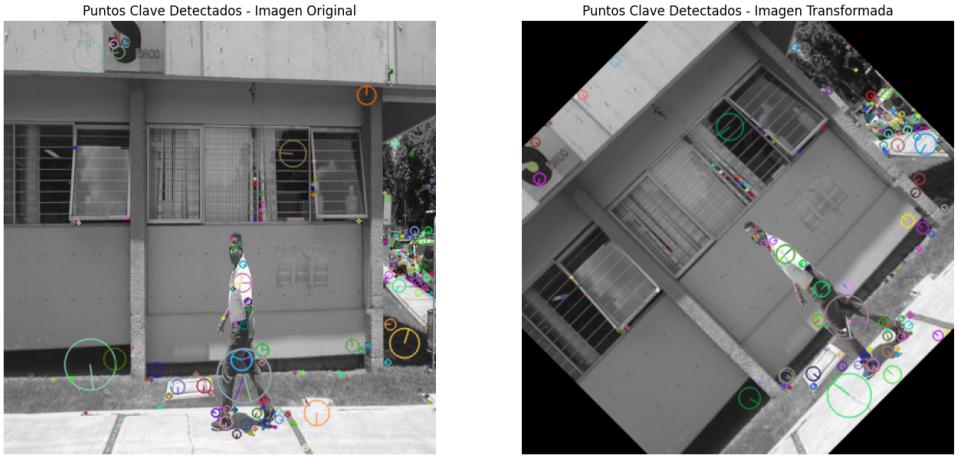


Figura 30: Visualización de descriptores en las dos imágenes. Fuente: Elaboracion propia.

### 3.2.5. HOG (Histogram of Oriented Gradients)

El algoritmo HOG es una técnica eficaz para extraer y representar características locales en imágenes mediante el cálculo de gradientes orientados en pequeñas celdas. HOG se ha consolidado como una herramienta robusta para la detección de objetos, en particular para la identificación de peatones, ya que al combinar la descripción detallada de la estructura visual con la normalización de bloques, logra ser invariante frente a variaciones de iluminación, contraste y posición.

### 3.2.6. Proceso de Aplicación de HOG

A continuación, se detallan los pasos seguidos en la implementación del algoritmo hog:

#### Paso 1: Cargar y Mostrar la Imagen

Antes de aplicar el algoritmo, se importan las librerías necesarias y se carga la imagen con la que se va a trabajar.

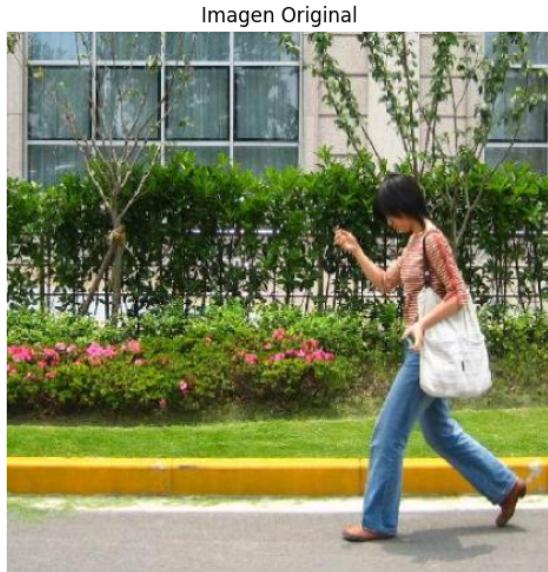


Figura 31: Imagen original cargada.

#### Paso2: Escalar la Imagen

La imagen se escala para ajustar la resolución a un tamaño óptimo, lo que permite que el descriptor HOG capture de manera consistente los patrones y detalles esenciales, minimizando la carga computacional sin perder información relevante.



Figura 32: Imagen original e imagen redimensionada.

### Paso 3: Configuración del detector HOG

El detector HOG requiere la configuración de varios parámetros clave que afectan la detección de características en la imagen:

- **cell\_size = (16,16)**: Tamaño de cada celda en la que se divide la imagen.
- **block\_size = (2,2)**: Número de celdas agrupadas para formar un bloque.
- **block\_stride = (8,8)**: Desplazamiento en píxeles entre bloques.
- **nbins = 9**: Número de intervalos del histograma de orientaciones (cada bin abarca 20°).
- **win\_stride = (8,8)**: Desplazamiento en píxeles de la ventana durante la detección.
- **padding = (8,8)**: Acolchamiento aplicado a la imagen en el proceso de detección.
- **scale = 1.05**: Factor de escala para construir la pirámide de escalas.
- **bin\_width = 20**: Ancho (en grados) de cada bin en el histograma.

Para la implementación se ha utilizado la clase correspondiente en el módulo `algoritmos/hog.py`.

### Paso 4: Conversión de la Imagen a Escala de Grises

La imagen se convierte a escala de grises para que HOG pueda centrarse exclusivamente en la estructura y forma de la imagen, lo que mejora la estabilidad de la extracción de gradientes y reducir la complejidad computacional.



Figura 33: Imagen en escala de grises.

#### Paso 5: Cálculo de Gradientes

El gradiente se calcula aplicando los operadores Sobel en las direcciones horizontal y vertical, obteniendo la magnitud y orientación de los cambios de intensidad en cada píxel. Esto permite resaltar los bordes y transiciones en la imagen, proporcionando información esencial sobre su estructura.

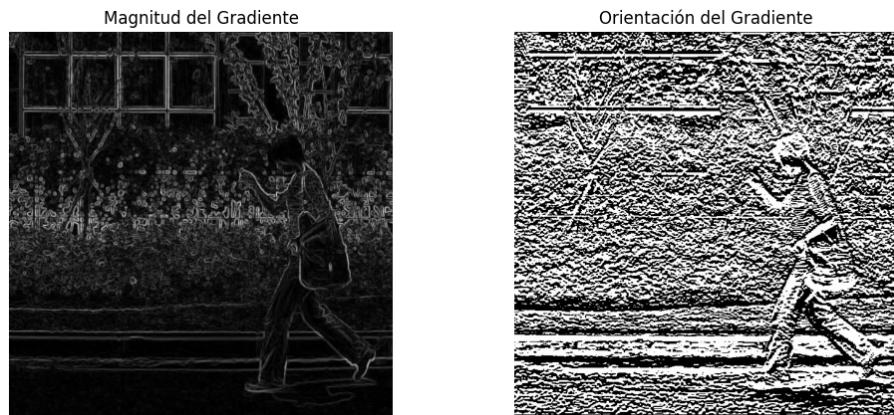


Figura 34: Magnitud del gradiente y Orientación del gradiente.

#### Paso 6: Cálculo del Histograma por celda

En cada celda se construye un histograma que acumula las magnitudes de los gradientes según la orientación de cada píxel, lo que permite capturar y resumir la distribución local de bordes y transiciones en esa región de la imagen.

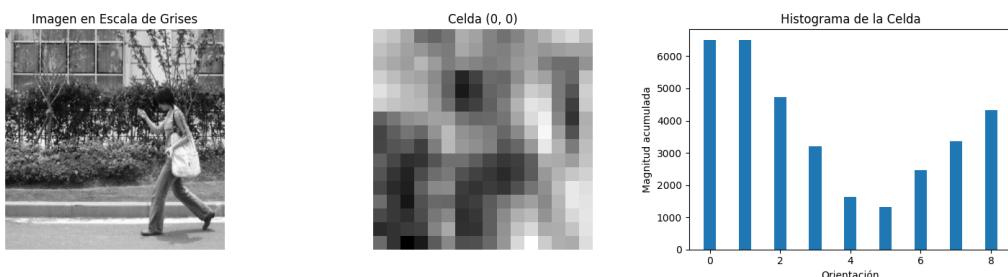


Figura 35: Ejemplo de cálculo del histograma para la celda (0,0).

## Normalización de Bloques y Concatenación de Descriptores

En este paso, los histogramas de las celdas se agrupan en bloques, y cada bloque se normaliza para reducir el efecto de variaciones en la iluminación y el contraste. Luego, los vectores normalizados de todos los bloques se concatenan para formar un descriptor global que resume la distribución de gradientes en toda la imagen.

### Paso 7: Visualización de los Descriptores HOG

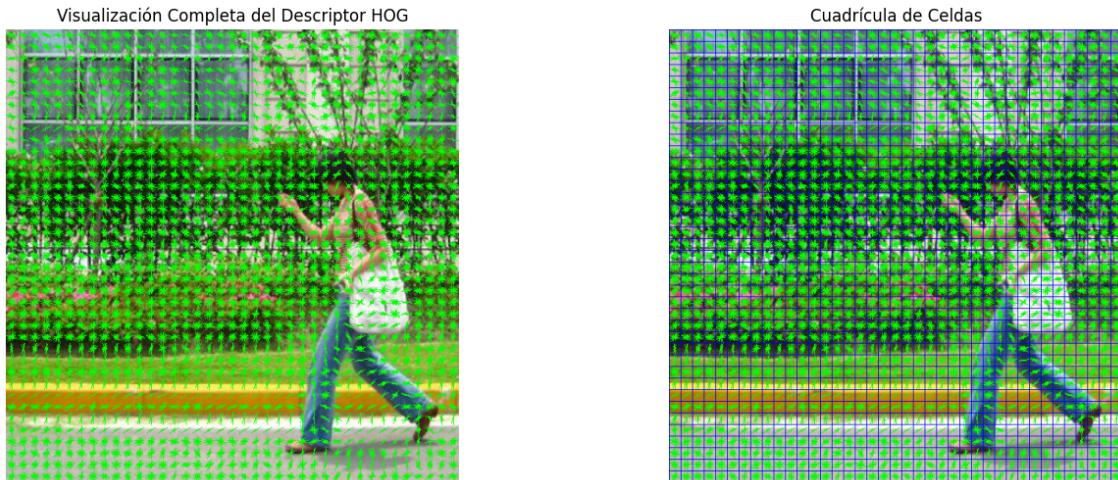


Figura 36: Visualización del descriptor y Visualización del descriptor con celdas.

### 3.3. Decisiones de Diseño

- **Separación de los algoritmos en archivos independientes:** Cada método de detección (Harris, ORB, SIFT y HOG) se implementó en archivos de Python independientes dentro de una carpeta central denominada **algoritmos**.
- **Presentación didáctica en Jupyter Notebooks:** Se desarrolló un `notebook.ipynb` por cada algoritmo donde se explican los algoritmos paso a paso.
- **Experimentación con un banco de imágenes de prueba:** Se realizó una prueba con un conjunto de imágenes para entrenar un modelo **SVM** y predecir si en una imagen hay o no un peatón.
- **Estructura organizada del proyecto:**
  - Se creó una carpeta `images` donde se almacenan los bancos de imágenes de entrenamiento del **SVM** y las pruebas para cada algoritmo. Las imágenes incluyen ejemplos con y sin personas, extraídas de `roboflow.com`.
  - Se añadió una carpeta `doc` para documentar el desarrollo del proyecto y actualizarlo progresivamente.
  - Se estableció una carpeta `machinelearning` dedicada específicamente al entrenamiento del modelo **SVM**.
  - Los Jupyter Notebooks están ubicados en la raíz del proyecto para facilitar su acceso y ejecución.

## 4. Experimentación

La experimentación es una parte fundamental de este trabajo, ya que permite evaluar el rendimiento de los algoritmos implementados en combinación con el modelo **SVM** (Support Vector Machine). Para obtener resultados significativos, hemos realizado pruebas sistemáticas con las configuraciones más eficientes con cada algoritmo y un conjunto variado de imágenes.

Una vez implementados y probados los algoritmos de detección de características (**Harris**, **ORB**, **SIFT** y **HOG**), decidimos entrenar un modelo **SVM** utilizando un banco de datos con más de 1000 imágenes, en las que se incluyen ejemplos con y sin peatones. El objetivo es evaluar la capacidad del modelo para predecir correctamente la presencia de un peatón en una imagen, siendo la dicha imagen preprocesada por cada uno de los algoritmos mostrados, es decir, el objetivo es experimentar y definir cual de los 4 algoritmos estudiados es el más conveniente para preprocesar imágenes para el modelo.

### 4.1. Preparación del Conjunto de Datos

Para garantizar un entrenamiento adecuado del **SVM**, cada imagen del banco de datos fue sometida a un tratamiento previo, el cual depende del algoritmo de detección de características a evaluar. Este preprocesamiento incluyó:

- Conversión a escala de grises (en los algoritmos que lo requieren).
- Aplicación del método de detección de características correspondiente (Harris, ORB, SIFT o HOG).
- Extracción de descriptores característicos de cada imagen.
- Normalización de los descriptores para mejorar la robustez del modelo.
- Extracción de los ROI (Region Of Interest) de las imágenes. Esto se dividió en dos categorías:
  - Imágenes con personas: Se utilizó un archivo .xml que indicaba la ubicación exacta del peatón mediante una caja delimitadora (ROI).
  - Imágenes sin personas: Se generó un ROI aleatorio de aproximadamente un 30% de la imagen. Este tamaño es el tamaño medio de los ROI del archivo .xml de las personas.

### 4.2. Entrenamiento del Modelo SVM

Tras procesar el conjunto de datos, se procedió al entrenamiento del **SVM** con los descriptores extraídos de cada imagen. Para ello, se utilizó un conjunto de entrenamiento compuesto por imágenes etiquetadas en las que tomaba el valor true en caso de haber una o más personas (señalando su ubicación), y false en caso contrario, para diferenciar aquellas que contienen peatones de aquellas que no.

## 5. Experimentación

La experimentación es una parte fundamental de este trabajo, ya que permite evaluar el rendimiento de los algoritmos implementados en combinación con un modelo **SVM** (Support Vector Machine). Con este fin, se llevaron a cabo pruebas sistemáticas utilizando las configuraciones más eficientes de cada algoritmo, sobre un conjunto variado de imágenes.

### 5.1. Preparación del Conjunto de Datos

Para entrenar el modelo **SVM**, se utilizó un conjunto de datos compuesto por imágenes divididas en dos categorías: con personas y sin personas. Las imágenes con personas venían acompañadas de un archivo .xml que indicaba la ubicación exacta del peatón mediante una caja delimitadora (ROI, Region of Interest). Para una mejor calidad en el entrenamiento, el procesamiento mediante los algoritmos de detección de características se aplicó exclusivamente sobre el ROI de cada imagen con persona. En cambio, las imágenes sin personas fueron procesadas en su totalidad.

Cada imagen fue sometida a un preprocesamiento específico, dependiendo del algoritmo utilizado (**Harris**, **ORB**, **SIFT** o **HOG**), siguiendo los pasos:

- Conversión a escala de grises.
- Aplicación del algoritmo de detección de características correspondiente.
- Extracción de los descriptores generados.

## 5.2. Adaptación de Descriptores para el SVM

Para entrenar un modelo **SVM**, es imprescindible que todos los vectores de entrada tengan la misma longitud. Sin embargo, los descriptores generados por los algoritmos de detección de características varían en tamaño según la imagen. Por ello, fue necesario diseñar un procedimiento específico para estandarizar los vectores generados por cada técnica. A continuación, se detalla el enfoque seguido en cada caso:

### **Harris**

El enfoque orginal del algoritmo Harris es generar una matriz de tamaño imagen, la cual contiene en sus pixeles el valor de la nota ofrecida por el algoritmo si este ha sido considerado como esquina o, 0s en el caso de no serlo. Para poder estandarizar dicha matriz y poder utilizarla como entrada al modelo SVM, se tomar en cuenta diferentes enfoques, dado que una simple matriz con puntos repartidos por el espacio difícilmente puede ser utilizada como entrada al modelo, pues no tiene un tamaño fijo y el modelo difícilmente puede aprender de puntos sin una estructura definida. Para ello se optó por cuatro enfoques diferentes:

- Se experimento para obtener la media de puntos que se calculaban por ROI, viendose un total de 28.5 puntos aproximadamente. Este enfoque consiste en pasar al modelo un vector fija de los 32 mejores puntos de la imagen, y llenar con ceros el resto de la matriz. El principal objetivo era que sabiendo que los ROIs contenian o no personas, si estas estaban presentes, y por tanto ser protagonistas de la imagen, el algoritmo obtendría muchos puntos y con notas altas de dicho ROI.
- El segundo enfoque consiste en calcular tres parámetros estadísticos (media, desviación estándar y valor máximo) de la matriz de notas Harris. Este enfoque permite obtener un vector de tamaño fijo, que puede ser utilizado como entrada al modelo SVM. Aunque este enfoque no introduce valores artificiales al modelo, este pierde mucha información como, posicional, relativa posisional o similitud de notas de puntos cercanos.
- El tercer enfoque consiste en obtener la densidad de puntos en la imagen. La base de este enfoque es que si existe una persona en la imagen ha de haber una mayor densidad de puntos en la región.
- El ultimo enfoque consiste en trabajar con la informacion posicional y relativa posisional de los puntos que nos ofrece la matriz de notas Harris. Consiste en obtener la dispersion de los puntos calculada con centroides, devuelve la distancia promedio, los centroides de cada eje, y la varianza de los ejes.

En general no se obtuvieron buenos resultados, pero no por culpa de los enfoques, sino por la naturaleza del algoritmo Harris, el cual arroja muy poca informacion sobre los puntos, privando de informacion al modelo. Aunque este es muy util para la obtención de esquinas de una manera rápida y sencilla.

La solución que finalmente se adoptó fue la de calcular los 32 primeros mejores puntos del ROI, y llenar con ceros el resto de la matriz. Esto ofreció resultados drásticamente mejores que el resto de enfoques, y aunque naturalmente no es el mejor enfoque, es el que mejor resultados ofreció.

## SIFT

El algoritmo **SIFT** genera descriptores de 128 dimensiones por keypoint, pero el número de keypoints detectados por imagen varía, lo que impide utilizar directamente sus descriptores como entrada al **SVM**.

Se probaron varios enfoques:

- Limitar el número de keypoints a 100 y llenar con ceros si no se alcanzaba dicha cantidad.
- Rellenar los huecos replicando descriptores existentes, aunque esto introducía redundancia.

Ambas opciones resultaron poco eficientes computacionalmente. Posteriormente, se intentó filtrar las imágenes que no tuvieran al menos 50 keypoints. Aunque esta estrategia permitía evitar el padding, y datos artificiales, eliminaba demasiadas imágenes del conjunto de test, reduciendo la muestra útil y por tanto dando información estadísticamente poco fiable.

La solución adoptada finalmente consistió en calcular un vector estadístico por imagen a partir de los descriptores SIFT. Este vector incluye medidas como la media, desviación estándar y valor máximo por dimensión de cada descriptor. Esta técnica permitió estandarizar los vectores de entrada al modelo sin añadir datos artificiales y manteniendo una buena eficiencia y computación.

## HOG

El descriptor **HOG** (Histogram of Oriented Gradients) genera vectores cuya longitud depende del tamaño de la imagen. Para poder usar estos vectores en un modelo **SVM**, es necesario que todos tengan la misma dimensión.

Para solucionar esto, se aplicaron dos enfoques:

- Redimensionado fijo: Todas las imágenes se escalan a un tamaño estándar ( $64 \times 128$  píxeles) antes de calcular el descriptor **HOG**, asegurando así una longitud uniforme del vector.
- Resumen estadístico: Alternativamente, se extraen tres estadísticas del descriptor original (media, desviación estándar y valor máximo), lo que da lugar a un vector corto pero fijo, útil para reducir la complejidad computacional.

## ORB

El algoritmo **ORB** genera descriptores binarios de tamaño fijo (usualmente de 256 bits) por cada keypoint detectado. Sin embargo, al igual que sucede con **SIFT**, el número de keypoints detectados en cada imagen puede variar considerablemente, lo que imposibilita el uso directo de los descriptores como entrada al modelo **SVM**.

Se analizaron diversas estrategias para adaptar los descriptores de **ORB** al modelo:

- **Límite de número de keypoints:** Se propuso fijar un número máximo de keypoints (por ejemplo, 100) y completar con vectores nulos (ceros) en caso de que no se alcanzara dicho número. Sin embargo, esta técnica introducía ruido cuando las imágenes no contenían suficientes keypoints de calidad.
- **Resumen estadístico:** Se calculó la **media**, la **desviación estándar** y el **valor máximo** de todos los descriptores **ORB** obtenidos en cada imagen. Este enfoque genera un vector compacto de tamaño fijo, que conserva información global sobre la distribución de los descriptores sin necesidad de padding artificial.

La solución adoptada finalmente fue la del **resumen estadístico** de los descriptores **ORB**. Concretamente, para cada dimensión del descriptor (256 bits), se calcularon los siguientes valores:

- Media de los valores de la dimensión en todos los keypoints.

- Desviación estándar de la dimensión.
- Valor máximo observado en la dimensión.

Este procedimiento permite transformar un conjunto variable de descriptores binarios en un vector de tamaño fijo, apto para su uso en el entrenamiento del modelo **SVM**. Además, al basarse en medidas estadísticas, mejora la robustez frente a variaciones menores en el número o calidad de los keypoints detectados.

Finalmente, esta estrategia demostró ser eficiente y efectiva para el uso de ORB en la clasificación de imágenes con o sin personas.

### 5.3. Entrenamiento del Modelo SVM

Con los descriptores estandarizados, se entrenó un modelo **SVM** cuyo objetivo es clasificar si una imagen contiene o no un peatón. El conjunto de entrenamiento se compuso de imágenes etiquetadas, asignando la clase **Persona** cuando existía al menos una anotación en el archivo `.xml`, y **No Persona** en caso contrario.

Cada modelo entrenado se guardó en un archivo independiente con el nombre `svm_algoritmo.pkl`, según el algoritmo utilizado para la extracción de características.

### 5.4. Pruebas de Evaluación

Una vez entrenado el modelo, se evaluó su rendimiento utilizando un banco de pruebas independiente, estructurado igual que el conjunto de entrenamiento: dos carpetas con imágenes con y sin personas, que no se habían utilizado durante la fase de entrenamiento.

Cada imagen de prueba fue procesada con el mismo algoritmo utilizado para el entrenamiento, extrayendo los descriptores correspondientes. Estos descriptores fueron posteriormente introducidos en el modelo **SVM** para realizar la predicción.

### 5.5. Resultados y Análisis

El rendimiento de los modelos entrenados con cada uno de los algoritmos se evaluó mediante métricas estándar en clasificación binaria, como la precisión, sensibilidad (recall), F1-Score y la matriz de confusión. A continuación, se explica brevemente en qué consiste cada una de estas métricas:

- **Precisión (Precision)**: mide la proporción de verdaderos positivos sobre el total de predicciones positivas realizadas por el modelo. Es decir, indica cuántas de las imágenes clasificadas como persona realmente contienen una persona. Se calcula como:

$$\text{Precisión} = \frac{TP}{TP + FP}$$

- **Sensibilidad (Recall)**: también conocida como *tasa de verdaderos positivos*, mide la capacidad del modelo para identificar correctamente todas las imágenes que realmente contienen personas. Se calcula como:

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1-Score**: es la media armónica entre la precisión y la sensibilidad. Esta métrica proporciona una visión equilibrada del rendimiento del modelo cuando existe un compromiso entre evitar falsos positivos y no perder verdaderos positivos. Se calcula como:

$$F1 = 2 \cdot \frac{\text{Precisión} \cdot \text{Recall}}{\text{Precisión} + \text{Recall}}$$

- **Matriz de confusión:** es una tabla que permite visualizar el rendimiento del modelo mostrando las cantidades de verdaderos positivos (TP), falsos positivos (FP), verdaderos negativos (TN) y falsos negativos (FN). Cada celda indica cuántas veces el modelo clasificó correctamente o se equivocó al predecir una clase frente a la clase real.

### Resultados con SIFT

- **Imágenes procesadas - Evaluación:** 747 con personas, 781 sin personas.
- **Precisión del modelo en el conjunto de prueba:** **79.25 %**
- **Precisión (Precision):** 76.41 %
- **Sensibilidad (Recall):** 83.27 %
- **F1-Score:** 79.69 %

#### Matriz de confusión:

$$\begin{bmatrix} 589 & 192 \\ 125 & 622 \end{bmatrix}$$

#### Distribución de casos:

- Verdaderos negativos (TN): 589
- Falsos positivos (FP): 192
- Falsos negativos (FN): 125
- Verdaderos positivos (TP): 622

#### Reporte de clasificación:

	precision	recall	f1-score	support
No Persona	0.82	0.75	0.79	781
Persona	0.76	0.83	0.80	747
accuracy			0.79	1528
macro avg	0.79	0.79	0.79	1528
weighted avg	0.80	0.79	0.79	1528

Los resultados obtenidos con SIFT muestran un rendimiento general equilibrado del modelo, destacando una buena sensibilidad (83.27 %), lo que indica una alta capacidad para identificar correctamente las imágenes que contienen personas. La precisión también es aceptable, aunque algo menor, lo que refleja la existencia de falsos positivos. En conjunto, estos datos evidencian que los descriptores generados a partir de SIFT son útiles para esta tarea, aunque podrían optimizarse con técnicas adicionales de selección o reducción de características.

### Resultados con HOG

#### Tipo 1: Redimensionado fijo:

- **Imágenes procesadas - Evaluación:** 747 con personas, 781 sin persona.
- **Precisión del modelo en el conjunto de prueba:** **95.09 %**
- **Precisión (Precision):** 95.90 %
- **Sensibilidad (Recall):** 93.98 %
- **F1-Score:** 94.93 %

**Matriz de confusión:**

$$\begin{bmatrix} 751 & 30 \\ 45 & 702 \end{bmatrix}$$

**Distribución de casos**

- Verdaderos negativos (TN): 751
- Falsos positivos (FP): 30
- Falsos negativos (FN): 45
- Verdaderos positivos (TP): 702

**Reporte de clasificación:**

	precision	recall	f1-score	support
No Persona	0.94	0.96	0.95	781
Persona	0.96	0.94	0.95	747
accuracy			0.95	1528
macro avg	0.95	0.95	0.95	1528
weighted avg	0.95	0.95	0.95	1528

**Tipo 2: Resumen estadístico:**

- Imágenes procesadas - Evaluación: 747 con personas, 781 sin persona.
- Precisión del modelo en el conjunto de prueba: **76.96 %**
- Precisión (Precision): 69.97 %
- Sensibilidad (Recall): 92.64 %
- F1-Score: 79.72 %

**Matriz de confusión:**

$$\begin{bmatrix} 484 & 297 \\ 55 & 692 \end{bmatrix}$$

**Distribución de casos:**

- Verdaderos negativos (TN): 484
- Falsos positivos (FP): 297
- Falsos negativos (FN): 55
- Verdaderos positivos (TP): 692

**Reporte de clasificación:**

	precision	recall	f1-score	support
No Persona	0.90	0.62	0.73	781
Persona	0.70	0.93	0.80	747
accuracy			0.77	1528
macro avg	0.80	0.77	0.77	1528
weighted avg	0.80	0.77	0.76	1528

Los resultados obtenidos con ambos tipos en HOG muestran que el método de redimensionado fijo (tipo 1) ofrece un rendimiento significativamente mejor. Al mantener la información completa del descriptor HOG, se logra mayor precisión, sensibilidad y F1-score, con menos falsos positivos y negativos. En cambio, el enfoque de resumen estadístico (tipo 2), al condensar la información en solo tres valores, pierde detalles cruciales, lo que afecta negativamente la capacidad del modelo SVM para distinguir correctamente las imágenes.

## 6. Resultados con ORB

### Clasificación mediante descriptores ORB

Imágenes procesadas - Evaluación: 747 con personas, 781 sin persona. Precisión del modelo en el conjunto de prueba: **75.39 %**

- **Precisión (Precision):** 66.67 %
- **Sensibilidad (Recall):** 99.33 %
- **F1-Score:** 79.78 %

#### Matriz de confusión:

$$\begin{bmatrix} 410 & 371 \\ 5 & 742 \end{bmatrix}$$

#### Distribución de casos:

- Verdaderos negativos (TN): 410
- Falsos positivos (FP): 371
- Falsos negativos (FN): 5
- Verdaderos positivos (TP): 742

#### Reporte de clasificación:

Clase	Precision	Recall	F1-Score	Soporte
No Persona	0.99	0.52	0.69	781
Persona	0.67	0.99	0.80	747
<b>Accuracy</b>	0.75 (1528 muestras)			
<b>Macro Avg</b>	0.83	0.76	0.74	1528
<b>Weighted Avg</b>	0.83	0.75	0.74	1528

Cuadro 1: Resultados de clasificación utilizando descriptores ORB

### Análisis de los resultados

Los resultados obtenidos con ORB muestran una alta capacidad de detección de personas, como demuestra su **recall del 99.33 %**, lo que implica que casi todos los individuos presentes fueron correctamente identificados. Sin embargo, su **precisión del 66.67 %** indica una elevada tasa de falsos positivos, es decir, en numerosas ocasiones el modelo predijo la existencia de una persona cuando no la había.

Este comportamiento es coherente con la naturaleza de los descriptores binarios de ORB: son rápidos y eficientes, pero menos detallados que los de SIFT o HOG. Aun así, el **F1-score global del 79.78 %** lo sitúa como un método viable en contextos donde se prioriza la detección (recall) por encima de la precisión.

El alto número de falsos positivos puede mitigarse ajustando el umbral de decisión del modelo o combinando ORB con técnicas de postprocesado espacial o de seguimiento en vídeo para reducir las detecciones erróneas aisladas.

### Resultados con Harris

#### Tipo 1: 32 mejores puntos de la imagen:

- Imágenes procesadas - Evaluación: 747 con personas, 781 sin persona.
- Precisión del modelo en el conjunto de prueba: **73.63 %**

- **Precisión (Precision):** 69.97 %
- **Sensibilidad (Recall):** 81.26 %
- **F1-Score:** 75.08 %

**Matriz de confusión:**

$$\begin{bmatrix} 518 & 263 \\ 140 & 607 \end{bmatrix}$$

- **Falsos positivos (TN):** 263
- **Falsos negativos (FP):** 140
- **Verdaderos positivos (FN):** 518
- **Verdaderos negativos (TP):** 607

**Reporte de clasificación:**

	precision	recall	f1-score	support
No Persona	0.79	0.66	0.72	781
Persona	0.70	0.81	0.75	747
accuracy			0.74	1528
macro avg	0.74	0.74	0.74	1528
weighted avg	0.74	0.74	0.74	1528

**Tipo 2: Tres parámetros estadísticos::**

- **Imágenes procesadas - Evaluación:** 747 con personas, 781 sin persona.
- **Precisión del modelo en el conjunto de prueba:** 57.98 %
- **Precisión (Precision):** 62.47 %
- **Sensibilidad (Recall):** 35.21 %
- **F1-Score:** 45.03 %

**Matriz de confusión:**

$$\begin{bmatrix} 623 & 158 \\ 484 & 263 \end{bmatrix}$$

- **Falsos positivos (TN):** 158
- **Falsos negativos (FP):** 484
- **Verdaderos positivos (FN):** 623
- **Verdaderos negativos (TP):** 263

**Reporte de clasificación:**

	precision	recall	f1-score	support
No Persona	0.56	0.80	0.66	781
Persona	0.62	0.35	0.45	747
accuracy			0.58	1528
macro avg	0.59	0.57	0.56	1528
weighted avg	0.59	0.58	0.56	1528

### **Tipo 3: Densidad de puntos:**

- **Imágenes procesadas - Evaluación:** 747 con personas, 781 sin persona.
- **Precisión del modelo en el conjunto de prueba:** **57.00 %**
- **Precisión (Precision):** 58.04 %
- **Sensibilidad (Recall):** 43.51 %
- **F1-Score:** 49.73 %

### **Matriz de confusión:**

$$\begin{bmatrix} 546 & 235 \\ 422 & 325 \end{bmatrix}$$

- **Falsos positivos (TN):** 235
- **Falsos negativos (FP):** 422
- **Verdaderos positivos (FN):** 546
- **Verdaderos negativos (TP):** 325

### **Reporte de clasificación:**

	precision	recall	f1-score	support
No Persona	0.56	0.70	0.62	781
Persona	0.58	0.44	0.50	747
accuracy			0.57	1528
macro avg	0.57	0.57	0.56	1528
weighted avg	0.57	0.57	0.56	1528

### **Tipo 4: Dispersión de los puntos:**

- **Imágenes procesadas - Evaluación:** 747 con personas, 781 sin persona.
- **Precisión del modelo en el conjunto de prueba:** **60.47 %**
- **Precisión (Precision):** 70.14 %
- **Sensibilidad (Recall):** 33.33 %
- **F1-Score:** 45.19 %

### **Matriz de confusión:**

$$\begin{bmatrix} 675 & 106 \\ 498 & 249 \end{bmatrix}$$

- **Falsos positivos (TN):** 106
- **Falsos negativos (FP):** 498
- **Verdaderos positivos (FN):** 675
- **Verdaderos negativos (TP):** 249

### **Reporte de clasificación:**

	precision	recall	f1-score	support
No Persona	0.58	0.86	0.69	781
Persona	0.70	0.33	0.45	747
accuracy			0.60	1528
macro avg	0.64	0.60	0.57	1528
weighted avg	0.64	0.60	0.57	1528

Una vez mostrados los resultados de cada enfoque podemos apreciar que el primero de ellos es el que mejores resultados ofrece.

## 7. Manual de usuario

El manual de usuario del proyecto se encuentra disponible en el archivo README.md dentro del repositorio del proyecto en GitHub. Dicho manual proporciona una guía detallada sobre el uso de la aplicación, incluyendo su instalación, configuración y funcionalidades principales.

Para acceder al manual de usuario, visite el siguiente enlace donde encontrará el repositorio del proyecto en GitHub:

<https://github.com/Lidiajim/AnalisisDeAlgoritmosPID>

## 8. Conclusiones

### 8.1. Síntesis de resultados

Como era de esperar, HOG obtuvo los mejores resultados, con una precisión del 95 %, lo cual tiene sentido porque este descriptor se creó precisamente para la detección de peatones. SIFT quedó segundo con un F1-score de 0,80: sus keypoints son muy informativos, pero al no poder usarlos directamente sobre el modelo svm, se ha perdido mucha información importante por lo que su rendimiento se ha visto muy afectado. Harris alcanzó un F1 de 0,75 incluso usando sólo sus 32 puntos más fuertes, y en cuanto a ORB, aunque es el algoritmo más rápido y ligero, su F1 ronda el 0.78, lo cual ofrece un equilibrio entre precisión y bajo coste computacional, lo que lo convierte en una buena opción en aplicaciones que antepongan la velocidad a la máxima exactitud.

### 8.2. Limitaciones y desviaciones respecto al plan inicial

- **Imposibilidad de usar SURF:** el algoritmo SURF está sujeto a patente y su implementación en cv2 no podía utilizarse libremente, por lo que hubo que descartarlo y usar otro algoritmo alternativo.
- **Vectores de longitud fija para el SVM:** para entrenar el clasificador fue imprescindible transformar los descriptores en vectores de dimensión fija. Esto obligó a diseñar procedimientos de *padding* o resúmenes estadísticos específicos para cada algoritmo.

### 8.3. Líneas futuras

1. Aumentar y balancear el dataset
2. Explorar otros modelos basados en aprendizaje automático

### 8.4. Lecciones aprendidas

- Hemos entendido el funcionamiento interno de un modelo de machine learning como es el **SVM**
- Aprendimos a transformar imágenes en vectores de características con distintos algoritmos (HOG, SIFT, ORB y Harris).
- Descubrimos la utilidad práctica de estos métodos en la detección de peatones y su aplicación en otros ámbitos de visión artificial, como la identificación de vehículos, el reconocimiento facial.

## 9. Autoevaluación

A continuación se presenta la autoevaluación individual realizada por cada miembro del equipo según la rúbrica proporcionada:

Rúbrica para evaluar los trabajos dirigidos				
Exposición	Álvaro	Lidia	Víctor	Óscar
Comprensión y dominio (puntuación triple)	Excelente	Excelente	Excelente	Excelente
Exposición didáctica	Excelente	Excelente	Excelente	Excelente
Integración del equipo	Excelente	Excelente	Excelente	Excelente
<b>Implementación</b>	<b>Álvaro</b>	<b>Lidia</b>	<b>Víctor</b>	<b>Óscar</b>
Objetivos (puntuación doble)	Excelente	Excelente	Excelente	Excelente
Aspectos didácticos	Excelente	Excelente	Excelente	Excelente
Experimentación y conclusiones	Bien	Excelente	Excelente	Excelente
<b>Documentación</b>	<b>Álvaro</b>	<b>Lidia</b>	<b>Víctor</b>	<b>Óscar</b>
Contenidos	Excelente	Excelente	Excelente	Excelente
Divulgación contenidos	Excelente	Excelente	Excelente	Excelente
Bibliografía y recursos científicos	Bien	Excelente	Excelente	Excelente

Cuadro 2: Rúbrica para evaluar trabajos dirigidos (ponderada sobre 7 puntos). Excelente (1 pto) Bien (0.65 ptos) Regular (0.35 ptos) Mal (0 ptos)

## 10. Tabla de tiempos

El seguimiento del trabajo de cada participante del proyecto se ha dividido en cuatro fases claramente definidas, en las cuales cada miembro del equipo se centró en tareas específicas:

Fase	Descripción y seguimiento asociado
Fase 1	Desarrollo de la idea inicial e investigación (seguimiento 0 y 1)
Fase 2	Implementación inicial mediante código (seguimiento 2)
Fase 3	Finalización de la implementación y documentación (seguimiento 3)
Fase 4	Preparación y presentación del proyecto (seguimiento 4)

### Seguimiento Individual

Fase	Álvaro Ruiz Gutiérrez	Lidia Jiménez Soriano	Víctor Flores González	Óscar Menéndez Márquez
Fase 1	23,44 horas	20,33 horas	24,37 horas	20,21 horas
Fase 2	30,12 horas	19,02 horas	16,27 horas	19,32 horas
Fase 3	18,44 horas	26,18 horas	24,44 horas	20,34 horas
Fase 4	5,55 horas	6,19 horas	5,50 horas	11,11 horas
<b>Total</b>	<b>78,36</b>	<b>71,45</b>	<b>76,19</b>	<b>71,39</b>

Para consultar con mayor detalle las fechas específicas, tareas asignadas, tiempo dedicado por cada participante y otra información relevante, puedes acceder al siguiente enlace a la ficha de datos de Clockify:

<https://github.com/Lidiajim/AnalisisDeAlgoritmosPID/blob/main/doc/clockify>

## Referencias

- [1] C. Harris y M. Stephens, *A Combined Corner and Edge Detector*, Proceedings of The Fourth Alvey Vision Conference, pp. 147–151, 1988. Disponible en: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=88cdfbeb78058e0eb2613e79d1818c567f0920e2>
- [2] H. Bay, T. Tuytelaars y L. Van Gool, *ORB: Oriented FAST and Rotated BRIEF*, en Computer Vision–ECCV 2006, Springer Berlin Heidelberg, pp. 404-417, 2006. Disponible en: [https://link.springer.com/chapter/10.1007/11744023\\_32](https://link.springer.com/chapter/10.1007/11744023_32)
- [3] I. Rey-Otero y M. Delbracio, *Anatomy of the SIFT Method*, Image Processing On Line, vol. 4, pp. 370–396, 2014. Disponible en: <https://doi.org/10.5201/ipol.2014.82>
- [4] N. Dalal y B. Triggs, *Histograms of oriented gradients for human detection*, en IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05), San Diego, CA, EE.UU., vol. 1, pp. 886-893, 2005. DOI: 10.1109/CVPR.2005.177. Disponible en: <https://ieeexplore.ieee.org/abstract/document/1467360>