Computer Vision

Los Del DGIIM, `losdeldgiim.github.io`

Doble Grado en Ingeniería Informática y Matemáticas
Universidad de Granada

Computer Vision

Los Del DGIIM, `losdeldgiim.github.io`

Joaquín Avilés de la Fuente

Granada, 2025-2026

Índice general

1. History

No relevante

2. Image processing

La forma más sencilla de entender una imagen digital es como una matriz de valores de intensidad. Cada celda de la matriz se llama píxel (picture element). En una imagen de escala de grises estándar de 8 bits, estos valores oscilan entre 0 (negro total) y 255 (blanco total). Lo que nosotros percibimos como formas y texturas, el ordenador lo interpreta como una tabla de números donde los valores altos representan zonas claras y los bajos zonas oscuras.

Para poder aplicar herramientas matemáticas (como las derivadas que usaremos para detectar bordes), definimos la imagen como una función bidimensional:

$$f(x, y)$$

donde

- (x, y) son las coordenadas espaciales (la posición del píxel en la columna x y fila y).
- f: Es la amplitud o intensidad en ese punto.

Si estuviéramos trabajando con una imagen continua (analógica), f sería una función definida sobre un plano real. Sin embargo, en el mundo digital, trabajamos con una función discreta, donde x e y son números enteros (índices de la matriz).

2.1. Point Processing

Entender concepto solo, no es necesario memorizar fórmulas (no es pregunta de examen).

2.1.1. Gamma Correction

La **Gamma Correction** es una operación de punto (point operation) no lineal que ajusta la relación entre el valor del píxel y la intensidad real de luz. La fórmula fundamental que rige este proceso es la ley de potencia

$$V_{out} = A \cdot V_{in}^{\gamma}$$

donde

- V_{in}: Es el valor de intensidad de entrada (normalizado entre 0 y 1).
- V_{out}: Es el valor de intensidad de salida.

- γ (Gamma): Es el exponente que define la forma de la curva de transformación.
- A: Es una constante (normalmente 1 en cálculos normalizados).

2.1.2. Histogram Equalization

A veces, una imagen tiene todos sus píxeles concentrados en un rango muy estrecho de valores (por ejemplo, una imagen muy oscura donde todos los píxeles están entre 0 y 50), por lo que tiene poca variación de intensidad, es decir, poco contraste. El **histograma** de esta imagen mostraría una gran "montaña" en la izquierda y nada en el resto.

La **ecualización del histograma** busca una función de transformación T tal que el histograma de la imagen de salida sea lo más plano (uniforme) posible. Esto significa que queremos que cada nivel de gris tenga, idealmente, el mismo número de píxeles. Esto maximiza el contraste.
Para lograr esto, utilizamos una herramienta estadística llamada **Función de Distribución Acumulada** (en inglés, CDF - Cumulative Distribution Function). La receta matemática es la siguiente:

1. Calcular el histograma de la imagen original $p(r_k)$, que nos dice cuántos píxeles hay de cada valor r_k

$$p(r_k) = \frac{\text{Number of pixels with intensity } r_k}{\text{Total number of pixels } n}$$

2. Normalizar el histograma, que viene a ser calcular la función CDF del histograma

$$CDF(r_k) = \sum_{j=0}^{k} p(r_j)$$

3. Aplicar la transformación a cada píxel de la imagen original para obtener la imagen ecualizada. La CFD se normaliza de tal manera que los valores mínimo y máximo del CDF corresponden a los valores mínimos y máximos posibles de intensidad (0 y 255 para una imagen de 8 bits)

$$r'_k = round\left(\frac{(CDF(r_k) - CDF_{min}) \cdot (L-1)}{N - CDF_{min}}\right)$$

donde

- r'_k: Es el nuevo valor de intensidad para el píxel con valor original r_k.
- CDF_{min}: Es el valor mínimo de la CDF (correspondiente al primer nivel de gris que tiene píxeles).
- L: Es el número total de niveles de gris (por ejemplo, 256 para imágenes de 8 bits).
- N: Es el número total de píxeles en la imagen.

En la Figura 2.1 se muestra un ejemplo de cómo la ecualización del histograma mejora el contraste de una imagen al distribuir los valores de intensidad de manera más uniforme a lo largo del rango disponible. La imagen original tiene un histograma concentrado en un rango estrecho, mientras que la imagen ecualizada tiene un histograma más uniforme, lo que mejora el contraste visual.

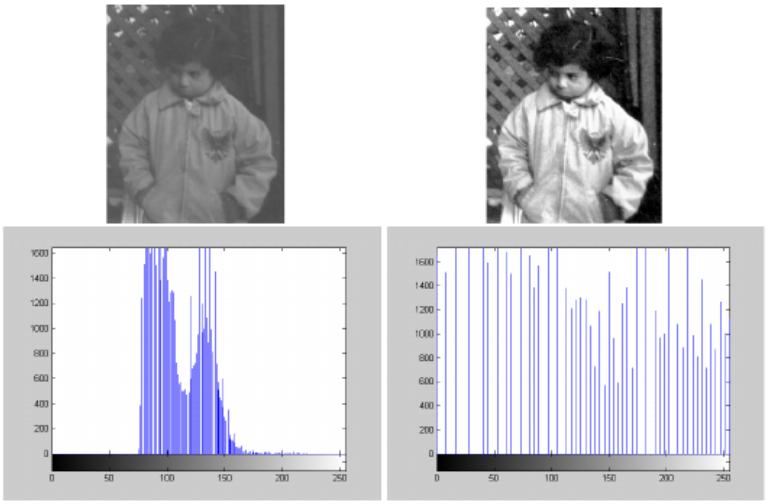

Figura 2.1: Ejemplo de ecualización de histograma. La imagen original tiene un histograma concentrado en un rango estrecho, mientras que la imagen ecualizada tiene un histograma más uniforme, lo que mejora el contraste.

2.2. Linear filtering

El filtrado lineal es una técnica donde cada píxel de una imagen es reemplazado por una combinación lineal (una suma ponderada) de sus píxeles vecinos. La "receta" que define los pesos de esta combinación se conoce como **kernel, máscara o filtro**.

Existen dos operaciones matemáticas fundamentales para aplicar estos filtros: la **correlación cruzada** y la **convolución**.

2.2.1. Correlación Cruzada (Cross-correlation)

Matemáticamente, puede entenderse como un "producto punto" entre el vecindario local de la imagen y el kernel. Si F es la imagen original, H es el kernel (de tamaño $(2k+1) \times (2k+1)$) y G es la imagen resultante, la operación se define como:

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u, j+v]$$

Propiedad matemática: Esta operación no es asociativa ni conmutativa, lo que significa que el orden de aplicación de los filtros importa, y no se pueden reordenar sin cambiar el resultado. Esto es importante a tener en cuenta al diseñar cadenas de procesamiento de imágenes, ya que aplicar los filtros en un orden diferente puede producir resultados significativamente distintos.

2.2.2. Convolución (Convolution)

Es la operación estándar en el procesamiento de imágenes. Es idéntica a la correlación cruzada, con una única diferencia crítica: el kernel H se invierte (se voltea horizontal y verticalmente) antes de realizar la suma ponderada. Se denota con el símbolo $*$ y su fórmula es:

$$G[i,j] = (F * H)(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v]$$

Propiedades matemáticas clave de la convolución: Al voltear el kernel, la convolución adquiere propiedades algebraicas que la correlación cruzada no tiene:

- Conmutativa: $a * b = b * a$
- Asociativa: $a * (b * c) = (a * b) * c$
- Distributiva sobre la suma: $a * (b + c) = (a * b) + (a * c)$
- Escalares: $\alpha a * b = a * \alpha b = \alpha(a * b)$
- Identidad: Existe un impulso unitario $e = [\ldots, 0, 0, 1, 0, 0, \ldots]$ tal que $a * e = a$.

La **importancia de la asociatividad**: en la práctica, es común aplicar múltiples filtros secuencialmente (por ejemplo, suavizar y luego detectar bordes). Matemáticamente, esto sería

$$(((a * b_1) * b_2) * b_3)$$

Gracias a la asociatividad, podemos pre-calcular un único filtro maestro

$$b_{master} = (b_1 * b_2 * b_3)$$

y aplicarlo a la imagen a una sola vez, ahorrando gran cantidad de cómputo.

2.2.3. Filtros Separables y Eficiencia Computacional

Definición 2.2.1. Un filtro $2D$ se considera **separable** si su matriz tiene un rango de 1, lo que significa que todas sus filas o columnas son múltiplos escalares entre sí. Matemáticamente, esto permite que la matriz del kernel se factorice como el producto externo de un vector columna por un vector fila. Podemos ver su representación matemática de la siguiente manera:

$$H = h_{col} \cdot h_{row}^T$$

donde h_{col} es un vector columna de tamaño $k \times 1$ y h_{row}^T es un vector fila de tamaño $1 \times k$. Esta propiedad de separabilidad es crucial porque permite que la convolución $2D$ con el filtro H se realice en dos pasos sucesivos de convolución $1D$:

1. Primero, se realiza una convolución $1D$ de la imagen con el filtro de columna h_{col}.
2. Luego, se realiza una convolución $1D$ del resultado con el filtro de fila h_{row}^T.

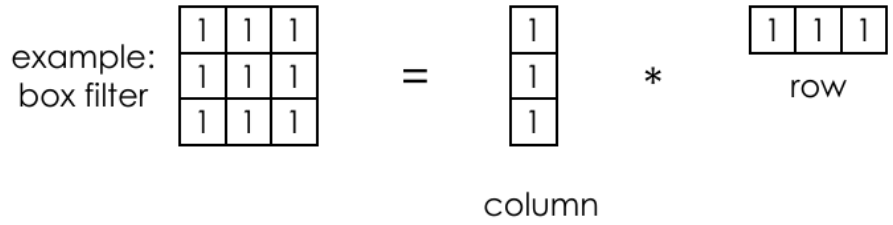

Figura 2.2: Ejemplo de un filtro $2D$ separable que se descompone en dos filtros $1D$.

Podemos ver una representación visual de esta propiedad en la Figura 2.2, donde se muestra cómo un filtro $2D$ se puede descomponer en dos filtros $1D$.
La convolución $2D$ con un filtro separable es matemáticamente equivalente a realizar dos convoluciones $1D$ consecutivas (una con el filtro de columna y otra con el de fila). Esto tiene un impacto significativo en la eficiencia computacional. Si aplicamos un filtro de tamaño tamaño $K \times K$ (por ejemplo, un filtro de 3×3, donde $K = 3$), para calcular un solo píxel de la imagen de salida, el ordenador tiene que:

1. Poner el kernel sobre el píxel correspondiente.
2. Multiplicar cada uno de los valores del kernel por el píxel que tiene debajo.
    - En un kernel de 3×3, hay 9 valores. Por tanto, hacemos 9 multiplicaciones ($3^2 = 9$).
3. Sumar esos 9 resultados para obtener el valor final.

Si generalizamos a un kernel de tamaño $K \times K$, para cada píxel de la imagen realizamos K^2 multiplicaciones. Si tu imagen tiene $N \times N$ píxeles, el coste total es $N^2 \cdot K^2$.

Sin embargo, si el filtro es separable, hemos dicho que $H = h_{col} \cdot h_{row}^T$. Esto significa que podemos descomponer la operación en dos pasos:

1. Filtrado Horizontal: Pasamos un kernel de $1 \times K$ (una fila). Para cada píxel, solo hacemos K multiplicaciones. Resultado: Una imagen intermedia.
2. Filtrado Vertical: Sobre la imagen intermedia, pasamos un kernel de $K \times 1$ (una columna). Para cada píxel, hacemos otras K multiplicaciones. Resultado: La imagen final.

En total, para cada píxel tenemos que

$$K \text{ (del horizontal)} + K \text{ (del vertical)} = 2K \text{ multiplicaciones}$$

esto representa una mejora significativa en términos de rendimiento, especialmente para filtros grandes. En lugar de K^2 multiplicaciones por píxel, solo necesitamos $2K$, lo que es una reducción drástica en la cantidad de cálculos necesarios para procesar la imagen.

2.2.4. Padding

Cuando aplicas un filtro (convolución) usando un kernel (por ejemplo, una ventana de 3×3 o 5×5), necesitas posicionar el centro del kernel sobre cada píxel de la imagen. El problema surge en los píxeles de los bordes y esquinas: el kernel "sobresale" de la imagen, encontrándose con una zona vacía donde no existen píxeles. El **padding** es la estrategia matemática que usamos para rellenar ese vacío artificialmente.

Las 4 estrategias usadas son las siguientes:

- **Zero-padding (Recorte o Relleno con Ceros)**: asume que todos los píxeles fuera de los límites de la imagen original tienen un valor de cero (color negro).
- **Wrap / Periodic (Envoltura o Periódico)**: trata la imagen como si fuera una estructura cilíndrica o toroidal repetitiva. sta estrategia consiste en pegar los bordes opuestos de la imagen para darle continuidad:
  - Pegas el borde derecho con el borde izquierdo, formando un cilindro.
  - Pegas el borde superior con el borde inferior.

  al hacer esto si el filtro (kernel) se sale por el lado derecho de la imagen, los píxeles que utiliza para rellenar ese vacío son los que están en el extremo izquierdo de esa misma fila.
- **Clamp / Replicate (Fijar o Replicar)**: extiende la imagen repitiendo el valor del último píxel disponible del borde de manera infinita hacia afuera.
- **Reflect / Mirror (Reflexión o Espejo)**: refleja los píxeles del interior de la imagen hacia el exterior, como si el borde fuera un espejo plano.

2.2.5. Key filters

Box filter (Mean filter)

El **filtro de caja** (box filter) o **filtro de media** (mean filter) es un ejemplo clásico de filtro lineal que se utiliza para suavizar imágenes. Su kernel es una matriz donde todos los valores son iguales y suman 1. Por ejemplo, para un filtro de 3×3, el kernel sería

$$H = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Este filtro es separable, ya que se puede descomponer en dos filtros $1D$:

$$h_{col} = \frac{1}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad h_{row}^T = \frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

Al aplicar este filtro a una imagen, cada píxel de la imagen de salida se calcula como el promedio de los píxeles vecinos en la imagen original, lo que resulta en una imagen suavizada. Sin embargo, el filtro de caja puede introducir artefactos no

deseados, como bordes borrosos y pérdida de detalles finos, especialmente cuando se utilizan kernels grandes. Además, debido a que todos los píxeles vecinos tienen el mismo peso, el filtro de caja no es tan efectivo para reducir el ruido sin afectar significativamente los bordes, lo que puede ser problemático en aplicaciones donde la preservación de los detalles es importante.

Gaussian filter

El **filtro Gaussiano** se basa en la función de distribución normal (Gaussiana). En 2D, la función del kernel $G(x, y)$ viene dada por

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

donde

- (x, y): son las coordenadas relativas al centro del kernel.
- σ (Sigma): es la desviación estándar. Es el parámetro más importante, ya que define el ancho de la campana y, por tanto, el "grado" de desenfoque.
- e: es la base del logaritmo natural.

A diferencia del filtro de caja (**mean/box filter**) donde todos los valores son iguales (ej. todos 1/9), en el filtro Gaussiano el valor máximo está en el centro del kernel y va disminuyendo suavemente a medida que nos alejamos del centro, siguiendo la forma de una campana de Gauss. Esto significa que los píxeles más cercanos al centro del kernel tienen un mayor peso en la suma ponderada, mientras que los píxeles más alejados contribuyen menos al resultado final.

Sharpen filter

El **filtro de nitidez** (sharpen filter) es un tipo de filtro que se utiliza para realzar los bordes y detalles finos en una imagen. A menudo, se construye a partir de la combinación de un filtro de suavizado (como el filtro de caja o el filtro Gaussiano) con la imagen original. La idea es restar la versión suavizada de la imagen original para obtener una máscara de alta frecuencia que resalte los detalles, y luego sumar esta máscara a la imagen original para obtener una imagen más nítida.

El proceso matemático es el siguiente

$$\text{Detalles} = \text{Original} - \text{Suavizada}$$

Si restamos a la imagen original su versión borrosa, nos quedamos solo con los bordes y cambios bruscos. Si luego le sumamos eso a la original, los bordes se acentúan, es decir, tenemos que

$$\text{Sharpened} = \text{Original} + \alpha \cdot \text{Detalles}$$

donde α es un factor de control que determina cuánto queremos realzar los bordes, de manera que

- Si $\alpha = 0$, no hay realce y la imagen de salida es igual a la original.
- Si $\alpha > 0$, los bordes se acentúan.
- Si $\alpha < 0$, se produce un efecto de suavizado adicional.

De forma más matemática queda como

$$f_{sharp}(x, y) = f(x, y) + \alpha[f(x, y) - f_{smooth}(x, y)]$$

donde

- $f(x, y)$: es la imagen original.
- $f_{smooth}(x, y)$: es la versión suavizada de la imagen (obtenida mediante un filtro de caja o Gaussiano).
- α: es un factor que controla cuánto queremos realzar los bordes.

Si operamos la fórmula, tenemos

$$f_{sharp} = (1 + \alpha)f - \alpha f_{smooth}$$

2.3. Non-linear filtering

Entender concepto solo, no es necesario memorizar fórmulas (no es pregunta de examen).

A diferencia de los lineales (como el promedio o el Gaussiano), donde el resultado es una suma ponderada de los vecinos, en los filtros no lineales la operación no puede expresarse como una matriz de pesos fija (kernel). El valor de salida depende de una regla lógica o estadística aplicada a la vecindad.

Bilateral filter

El filtro **bilateral** es un filtro no lineal que se utiliza para suavizar imágenes mientras preserva los bordes, por lo que se presenta como una evolución avanzada del filtro Gaussiano, diseñada para resolver su principal problema: el desenfoque de los bordes. Mientras que el filtro Gaussiano es un filtro çiego"que suaviza todo por igual, el filtro bilateral es un filtro preservador de bordes (edge-preserving).

Si aplicamos un filtro Gaussiano estándar en una zona donde hay un borde (un cambio brusco de intensidad), el filtro mezclará los píxeles de ambos lados del borde, emborronándolo. El filtro bilateral, en cambio, solo promedia aquellos píxeles que son cercanos en el espacio Y similares en intensidad.

A diferencia del filtro Gaussiano que solo tiene un componente, el filtro bilateral combina dos funciones Gaussianas. El valor del píxel de salida $g(x, y)$ se calcula como:

$$g(x,y) = \frac{1}{W_p} \sum_{u,v} f(x+u, y+v) \cdot G_{\sigma_s}(u,v) \cdot G_{\sigma_r}(|f(x,y) - f(x+u, y+v)|)$$

donde

- G_{σ_s} (Domain Kernel): es la Gaussiana espacial que ya conoces. Da más peso a los píxeles que están cerca físicamente. Depende de la distancia en coordenadas.
- G_{σ_r} (Range Kernel): es la "magia" del filtro. Da más peso a los píxeles que tienen un valor de intensidad (color) similar al píxel central. Depende de la diferencia de brillo.
- W_p: es un factor de normaliçzación para asegurar que la suma de los pesos sea 1.

Filtro de la mediana (Median Filter)

Es el filtro no lineal más famoso y utilizado. En lugar de promediar los píxeles de la ventana, los ordena de menor a mayor y selecciona el valor que queda justo en la posición central (la mediana). Es extremadamente eficaz para eliminar el ruido de "sal y pimienta" (puntos blancos y negros aleatorios). Se puede ver un ejemplo de este ruido en la Figura 2.3.

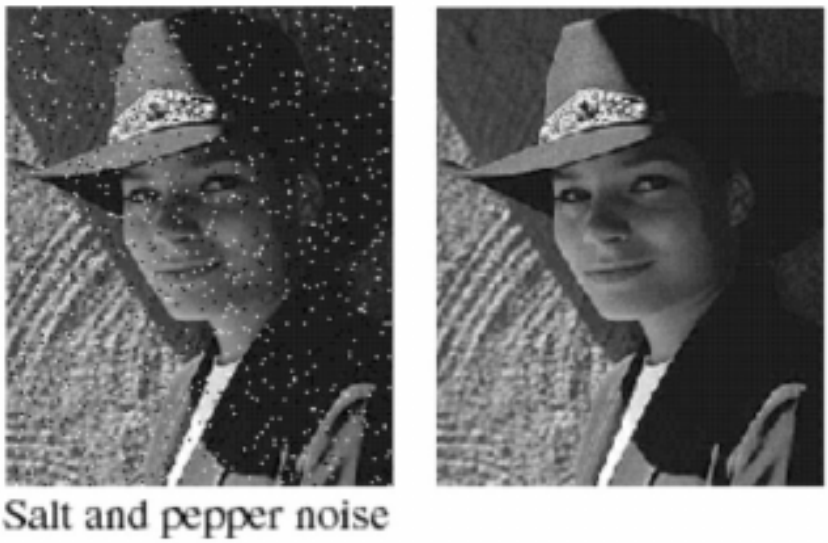

Figura 2.3: Ejemplo de filtro de la mediana.

Threshold filter

El threshold filter es una de las operaciones de punto más sencillas pero más potentes en el procesado de imágenes. Se utiliza principalmente para la segmentación, es decir, para separar los objetos del fondo.

Consiste en comparar cada píxel de la imagen con un valor constante llamado umbral (T). Dependiendo de esa comparación, el píxel se transforma en uno de dos valores posibles (normalmente blanco o negro). Viene dado de esta manera

$$g(x,y) = \begin{cases} 255 & \text{si } f(x,y) > T \\ 0 & \text{si } f(x,y) \leqslant T \end{cases}$$

donde

- $f(x,y)$: intensidad del píxel original.
- T: el umbral (un valor entre 0 y 255).
- $g(x,y)$: imagen binaria resultante.

2.4. Sampling y Aliasing

Entender concepto solo, no es necesario memorizar fórmulas (no es pregunta de examen).

El **aliasing** es un defecto visual que ocurre cuando la **tasa de muestreo** (sampling rate) que utilizamos no es lo suficientemente alta para capturar toda la cantidad de detalle fino presente en la imagen.

Definición 2.4.2. La **tasa de muestreo** es la densidad de píxeles (píxeles por milímetro o la resolución total de la cuadrícula de tu sensor de cámara). Si una imagen tiene $N \times N$ pixeles, entonces la tasa de muestreo es N por cada eje.

Visualmente, esto se traduce en artefactos engañosos, como patrones extraños, anillos o bordes aserrados que no existían en la imagen original. Al perderse información clave, obtenemos una imagen que representa una señal incorrecta, lo que se conoce como un "alias".

Para evitar el aliasing y realizar el muestreo correctamente, la teoría del procesamiento de señales se apoya en el **Teorema de muestreo de Nyquist-Shannon**, que indica que para no perder información y no generar aliasing al reducir una imagen, la tasa de muestreo debe ser al menos el doble de la frecuencia máxima presente en la imagen, e decir

$$f_s \geqslant 2 \cdot f_{max}$$

A esta tasa de muestreo mínima requerida se le llama la **Tasa de Nyquist**.

Subsampling y Upsampling

El proceso para conseguir el **subsampling** de forma correcta es el siguiente, si queremos reducir el tamaño de una imagen (por ejemplo, a la mitad), estamos reduciendo la cantidad de píxeles disponibles para representar la información. Esto significa que la frecuencia máxima que podemos representar también se reduce. Si

no hacemos nada para eliminar las frecuencias altas antes de reducir el tamaño, esas frecuencias se convertirán en alias (artefactos) en la imagen reducida, por lo que hacemos es lo siguiente

1. **Analizamos la nueva tasa**: Si vamos a reducir la imagen a la mitad, nuestra nueva frecuencia máxima permitida (Límite de Nyquist) es ahora mucho más baja.

2. **Aplicamos el filtro**: Pasamos un filtro Gaussiano que actúa como un filtro de paso bajo. Su función es eliminar (emborronar) todos los detalles cuya frecuencia sea mayor que el nuevo límite de Nyquist.

3. **Muestreamos**: Ahora que hemos "limpiado" las frecuencias prohibidas, podemos quitar píxeles de forma segura (por ejemplo, nos quedamos con uno de cada dos píxeles).

Esto se formaliza al construir la **Pirámide Gaussiana**, donde cada nivel l de la pirámide se obtiene del nivel anterior $l-1$ mediante: convolución con un kernel Gaussiano (normalmente con un σ específico para esa escala), y después eliminación de cada segunda fila y columna.

$$G_l = \text{subsample}(G_{l-1} * \text{Gaussian})$$

Si omitiéramos el paso del filtro, la imagen resultante en el nivel superior estaría llena de ruido y errores de aliasing. Al filtrar, nos aseguramos de que la información que queda es una representación fiel, aunque menos detallada, de la original.

Ahora nos centraremos en ver cómo podemos hacer el proceso contrario, es decir, el **upsampling** o aumento de tamaño. En este caso, el proceso es el siguiente

1. **Muestreamos**: si tenemos una imagen de $N \times N$, creamos una nueva matriz de $2N \times 2N$, donde colocamos los píxeles originales en las posiciones pares y rellenamos los nuevos huecos con ceros.

2. **Aplicamos el filtro**: Pasamos un filtro Gaussiano para suavizar la imagen resultante. Esto es necesario porque al insertar nuevos píxeles sin información real, obtenemos una imagen con bordes muy duros y artefactos. El filtro ayuda a interpolar esos nuevos píxeles de manera más natural.

Matemáticamente, el proceso de upsampling se puede expresar como:

$$G_{l-1} = \text{upsample}(G_l) * \text{Gaussian}$$

donde G_l es la imagen en el nivel actual de la pirámide, y G_{l-1} es la imagen resultante después de el upsampling y el filtrado. El resultado final es una imagen más grande y suave.

2.5. Image derivatives

Entender concepto solo, no es necesario memorizar fórmulas (no es pregunta de examen).

Un **borde** es, por definición, una zona donde la intensidad de los píxeles cambia de forma brusca (por ejemplo, pasamos de un fondo negro a un objeto blanco). Matemáticamente, un cambio brusco genera una derivada con un valor alto. Al derivar la imagen, convertimos las zonas planas en negro (valor 0, ya que no hay cambio) y resaltamos los bordes en blanco.

Como tenemos dos dimensiones (x, y), calculamos las variaciones en ambas direcciones:

- **Derivada en X** ($\frac{\partial f}{\partial x}$): detecta cambios de intensidad horizontales (bordes verticales), se mira el píxel que está a la derecha del píxel actual y el que está a la izquierda, y resta sus valores de brillo.
- **Derivada en Y** ($\frac{\partial f}{\partial y}$): detecta cambios de intensidad verticales (bordes horizontales), se mira el píxel que está justo abajo del píxel actual y el que está justo arriba, y resta sus valores de brillo.

Matemáticamente, en el mundo discreto de los píxeles, usamos la aproximación de diferencia finita

$$\frac{\partial f}{\partial x} \approx f(x+1, y) - f(x, y)$$

Podemos ver un ejemplo en la Figura 2.4.

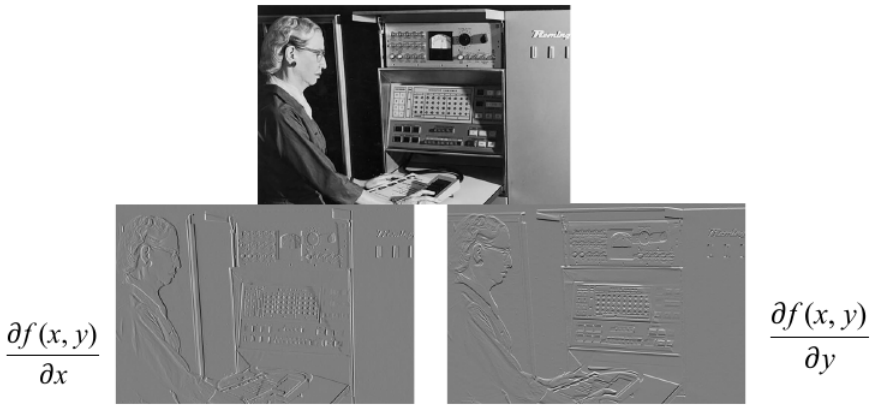

Figura 2.4: Ejemplo de derivadas de la imagen.

El **gradiente** es un vector que agrupa ambas derivadas parciales

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

De este vector extraemos dos piezas de información críticas para cualquier algoritmo de visión:

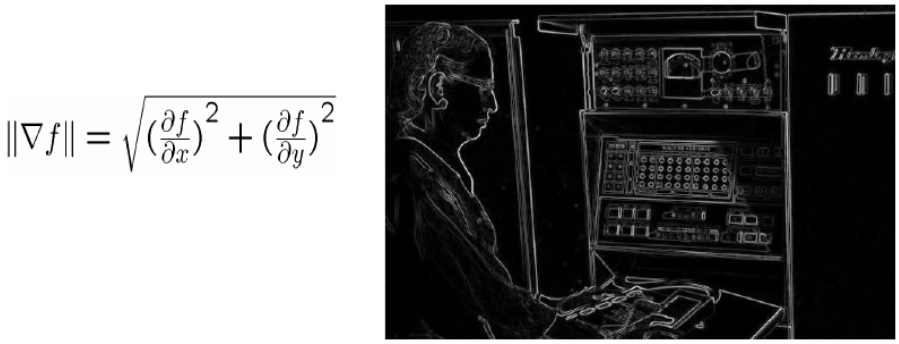

Figura 2.5: Ejemplo de magnitud y orientación del gradiente.

- **Magnitud del Gradiente**: Indica la "fuerza"del borde. Se calcula como

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

- **Orientación del Gradiente**: Indica la dirección del cambio de intensidad más rápido. Se calcula como

$$\theta = \tan^{-1}\left(\frac{\partial f/\partial y}{\partial f/\partial x}\right)$$

Podemos ver un ejemplo en la Figura 2.5, donde se muestra la magnitud y orientación del gradiente para una imagen dada.

Hay que tener en cuenta que las derivadas son muy sensibles al ruido, por lo que es común aplicar un filtro de suavizado (como el Gaussiano) antes de calcular las derivadas para obtener resultados más estables y precisos. En este proceso solemos usar el **Teorema de la Convolución**, que nos dice que la convolución de una imagen con la derivada de un filtro es equivalente a calcular la derivada de la imagen suavizada con ese filtro. Esto se expresa matemáticamente como:

$$\frac{\partial}{\partial x}(f * G) = f * \frac{\partial G}{\partial x}$$

Esto significa que podemos pre-calcular la derivada del filtro Gaussiano (que es un filtro de Sobel, veremos más adelante qué es eso) y luego convolucionarlo con la imagen original, en lugar de tener que suavizar la imagen primero y luego calcular la derivada. Esto es mucho más eficiente computacionalmente y nos da el mismo resultado.

2.6. Edge detection

Otra forma de encontrar bordes es utilizando filtros de segunda derivada, como el **filtro Laplaciano**. En la segunda derivada, un borde ya no es un "pico", sino el

punto exacto donde la función cruza el valor de cero. A esto se le llama cruce por cero (**zero-crossing**), y tiene la ventaja de que es computacionalmente más fácil de localizar que buscar un valor máximo. Matemáticamente, el filtro Laplaciano se define como la suma de las segundas derivadas parciales:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \implies LoG = \nabla^2(G_\sigma * f) = \nabla^2(G_\sigma) * f$$

que se va a aproximar de esta manera

$$\frac{\partial^2 f}{\partial x^2} \approx f(x+1) - 2f(x) + f(x-1)$$

donde LoG es el filtro de Laplaciano de Gaussiano, que se obtiene aplicando el filtro de Laplaciano a la imagen suavizada con un filtro Gaussiano. El proceso de detección de bordes con el filtro Laplaciano implica buscar los puntos donde el resultado del filtro cambia de signo (de positivo a negativo o viceversa), lo que indica la presencia de un borde.

2.6.1. Derivatives filters

Sobel filter

El **filtro de Sobel** se utiliza para aproximar la primera derivada de la imagen. Su objetivo es calcular el gradiente de la intensidad en cada píxel, lo que nos indica la fuerza de un borde y su dirección. La clave del Sobel es que combina dos operaciones en una sola matriz:

- **Diferenciación**: cacula la diferencia de intensidad (derivada).
- **Suavizado**: aplica un pequeño desenfoque (promedio pesado) en la dirección perpendicular a la derivada para reducir el ruido.

Para obtener el gradiente completo, aplicamos dos filtros independientes, uno para la dirección horizontal y otro para la vertical. Típicamente son matrices de 3×3:

- **Gradiente Horizontal (G_x)**: Detecta bordes verticales. Su kernel es:

$$S_x = \frac{1}{8}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \implies G_x = S_x * f$$

- **Gradiente Vertical (G_y)**: Detecta bordes horizontales. Su kernel es:

$$S_y = \frac{1}{8}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \implies G_y = S_y * f$$

aunque la versión estándar no tiene la normalización de $1/8$, se suele incluir para mantener los valores dentro del rango de intensidad. Al aplicar estos filtros a la imagen, obtenemos dos imágenes de gradiente (G_x y G_y) que representan la fuerza de

Figura 2.6: Ejemplo de los gradientes G_x y G_y obtenidos al aplicar el filtro de Sobel a una imagen.

los bordes en las direcciones horizontal y vertical, respectivamente. Luego, podemos combinar estas dos imágenes para obtener la magnitud del gradiente total, que nos da una medida de la fuerza del borde en cada píxel. De hecho, el filtro de Sobel es separable, veámoslo

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

donde tenemos

- Filtro Vertical:

  $$\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}^T$$

  es un componente de suavizado (smoothing). Al tener valores positivos, promedia los píxeles de las filas adyacentes, lo que ayuda a reducir el ruido antes de calcular la diferencia.

- Filtro Horizontal:

  $$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

  es el componente de diferenciación (derivada). Es el que realmente calcula la diferencia de intensidad entre el lado izquierdo y el derecho para detectar el borde.

Lo que podemos ver aqui es que el operador de Sobel no es más que una derivada suavizada, pues si solo usáramos $[-1, 0, 1]$, el ruido afectaría mucho al resultado, pero al multiplicarlo por $[1, 2, 1]^T$, estamos diciendo: "calcula la derivada, pero dale más importancia a la fila central y promedia un poco con las filas de arriba y abajo para que el resultado sea más estable". Podemos ver un ejemplo del resultado de ambos gradientes en la Figura 2.6.

El resultado final de aplicar Sobel, es unir ambos gradientes, G_x y G_y, para obtener la magnitud del gradiente total, que nos da una medida de la fuerza del borde en cada píxel. Esto se calcula como:

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

Figura 2.7: Comparación bordes entre filtro de Sobel (izquierda) y algoritmo de Canny (derecha).

y la orientación del borde se calcula como

$$\theta(x,y) = \tan^{-1}\left(\frac{G_y(x,y)}{G_x(x,y)}\right) \quad \equiv \quad \theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Observación. La orientación del borde no es estrictamente necesaria si solo aplicamos el filtro de Sobel, pero veremos más adelante su uso en otros casos.

El filtro de Sobel es ampliamente utilizado en aplicaciones de visión por computadora para la detección de bordes, ya que proporciona una buena aproximación de la derivada de la imagen mientras reduce el impacto del ruido, lo que lo hace ideal para detectar bordes en imágenes reales que a menudo contienen ruido.

El **resultado final** de aplicar el filtro de Sobel a una imagen es una nueva imagen donde los bordes se resaltan como líneas brillantes (blancas) sobre un fondo oscuro. La intensidad de estas líneas blancas corresponde a la fuerza del borde detectado, es decir, a la magnitud del gradiente en esa región.

Observación. Es normal que al aplicar Sobel, el resultado nos de líneas blancas gruesas que marquen los bordes, puesto que el filtro de Sobel no solo detecta el borde, sino que también lo suaviza, lo que hace que el borde resultante tenga un grosor de varios píxeles. Para obtener bordes más delgados, se pueden aplicar técnicas adicionales como la supresión de no-máximos (non-maximum suppression) o utilizar filtros de derivada más precisos como el filtro de Canny, que veremos a continuación.

Podemos ver un ejemplo en la Figura 2.7 de la diferencia de aplicar el filtro de Sobel y Canny, para ver como en el primer caso las líneas de los bordes son más gruesas.

2.6.2. Canny Edge Detector

No es simplemente un filtro, sino nuestro primer pipeline completo de Computer Vision. Fue diseñado por John Canny en 1986 con el objetivo de ser el “detector

óptimo" basándose en tres criterios: baja tasa de error, buena localización y respuesta única. El algoritmo se divide en cuatro pasos fundamentales, veámoslo

1. **Suavizado (Smoothing)**: El primer paso es aplicar un filtro Gaussiano para eliminar el ruido de alta frecuencia que podría generar bordes falsos, es decir, aplicar
$$fs = G_\sigma * f \quad \text{(eliminar ruido)}$$

2. **Cálculo del Gradiente**: Se calcula la magnitud y la orientación del gradiente utilizando operadores de derivada (como Sobel). La magnitud (M) indica la fuerza del cambio de intensidad, mientras que la orientación (θ) indica la dirección perpendicular al borde, véamolos
$$M = \|\nabla fs\| \quad \text{y} \quad \theta = \tan^{-1}\left(\frac{\partial fs/\partial y}{\partial fs/\partial x}\right)$$

3. **Supresión de No-Máximos (Non-maximum Suppression)**: este paso es el que hace que Canny sea especial. El objetivo es "adelgazar" los bordes para que tengan solo 1 píxel de grosor.
Cómo funciona: el algoritmo recorre cada píxel y comprueba si su magnitud es un máximo local en la dirección de su gradiente. Lo que hacemos es aislar un vector de 3 píxeles, de manera que solo comparamos el píxel central con sus dos vecinos en la dirección del gradiente, de la siguiente manera
   - Si el píxel central tiene una magnitud mayor que ambos vecinos, se mantiene su valor.
   - Si no es un máximo local, se pone a cero.

   Esto convierte las "rampas" de intensidad en líneas finas y precisas.

4. **Umbralización por Histéresis (Hysteresis Thresholding)**: en lugar de usar un solo umbral (que podría romper un borde si su intensidad varía ligeramente), Canny utiliza dos umbrales: uno Alto (T_h) y uno Bajo (T_l), de la siguiente manera
   - Píxeles Fuertes: Si la magnitud $> T_h$, se acepta como borde definitivo.
   - Píxeles Débiles: Si $T_l <$ magnitud $< T_h$, solo se acepta como borde si está conectado a un píxel fuerte.
   - Ruido: Si la magnitud $< T_l$, se descarta.

   El objetivo de este proceso, es asegurar que las líneas que marquen los bordes tengan una intensidad mínima, y que en los casos en los que una píxel de esa linea que define un borde tenga menos intensidad, antes de eliminarlo completamente, comprobamos si está conectado a un borde fuerte, lo que nos permite mantener bordes completos incluso si su intensidad varía a lo largo de su longitud. Podemos ver un ejemplo de esta ultima fase en la Figura 2.8.

   Observación. Para ver si está conectado a un pixel fuerte, como bien se hace en el ejemplo de la figura, cogeremos los píxeles de su entorno (los 8 de su alrededor).

Podemos ver un resumen de este tema en la Figura 2.9.

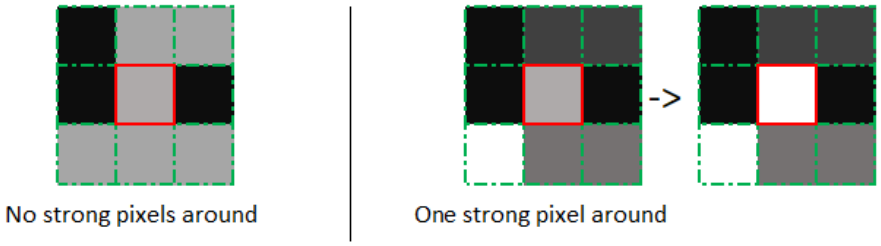

Figura 2.8: Ejemplo de umbralización por histéresis en el algoritmo de Canny.

Key Takeaways – Lecture 2

1. Image = function $f(x, y)$; digital = discrete + quantized. Color pipeline: Bayer CFA → demosaic → sRGB.
2. **Point processing**: per-pixel transform ($g = T[f]$): gamma, histogram equalization, CLAHE.
3. **Cross-correlation vs convolution**: convolution flips the kernel; both slide a weighted sum across the image. Convolution is associative and commutative.
4. **Separable filters**: $\mathcal{O}(L^2MN) \to \mathcal{O}(2LMN)$; rank-1 matrix test.
5. **Gaussian** = canonical low-pass filter. **Bilateral** = edge-preserving (spatial + intensity weights).
6. **Nyquist**: $f_s \geq 2f_{\max}$. Fix aliasing with Gaussian pre-filtering before subsampling.
7. Gradient: $\|\nabla f\|$ = edge strength; $\theta \perp$ edge. Use **Derivative of Gaussian** (smooth+derive in one step).
8. **Canny**: Gaussian → gradient → NMS → hysteresis. Still gold standard.

Figura 2.9: Resumen de Tema 2.

3. Gaussian image pyramid

En el procesamiento de imágenes analógicas o digitales tradicionales, las características de la imagen están fuertemente ligadas a su resolución original. El **enfoque multiescala** (Multi-Resolution Representation) surge de una idea fundamental:

> "Dado que una gran cantidad de suavizado limita la frecuencia de las características de una imagen, ¡no necesitamos mantener todos los píxeles a nuestro alrededor!"

La estrategia consiste en **reducir progresivamente el número de píxeles a medida que suavizamos más y más la imagen**, lo que da lugar a una estructura jerárquica conocida como **pirámide de imágenes**.

3.1. Gaussian Pyramid

La pirámide gaussiana es una secuencia de imágenes submuestreadas donde cada nivel representa una versión cada vez más borrosa y pequeña del nivel anterior.

En la construcción de la pirámide gaussiana, el uso del filtro gaussiano no es una elección arbitraria; se debe principalmente a una propiedad matemática y física crucial llamada **auto-reproducibilidad** (o propiedad de cascada del filtro gaussiano) y a la prevención del fenómeno de aliasing.

La propiedad de **self-reproducing property**, matemáticamente indica que si realizas la convolución de una señal con una función gaussiana de desviación estándar σ_1, y el resultado lo vuelves a convolucionar con otra gaussiana de desviación estándar σ_2, el resultado final es exactamente equivalente a haber filtrado la imagen original una sola vez con una única función gaussiana cuya varianza es la suma de las varianzas

$$\sigma_{\text{final}} = \sqrt{\sigma_1^2 + \sigma_2^2}$$

esto permite un suavizado incremental y constante en cada nivel.

3.1.1. Algoritmo de Construcción

El proceso es iterativo (síntesis recursiva). Para cada nivel i, la imagen se construye aplicando dos operaciones consecutivas:

1. Start with the original image G_0.

2. Repeat until minimum resolution is reached:

   a) **Filtrado (Suavizado):** apply Gaussian blur to the current level G_k. Se utiliza el filtro Gaussiano porque es *auto-reproducible*, lo que permite un suavizado incremental estable.

   b) **Submuestreo (Downsampling):** reducción del tamaño de la imagen (típicamente eliminando filas y columnas alternas, reduciendo cada dimensión a la mitad, es decir, un 25 % del área por paso), obteniendo G_{k+1}.

Este bucle se repite:

$$\text{Filtrar} \longrightarrow \text{Submuestrear}$$

hasta alcanzar una resolución mínima predefinida (por ejemplo, plantillas de 8×8 píxeles).

3.1.2. Tamaño Total en Memoria

Una pregunta matemática clave planteada en clase es: *¿Cuánto más grande es la pirámide completa en comparación con la imagen original?*

Si la imagen original tiene un tamaño (área) de 1, los niveles sucesivos ocuparán $\frac{1}{4}$, $\frac{1}{16}$, $\frac{1}{64}$, etc. La suma geométrica infinita es:

$$\text{Tamaño Total} = 1 + \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \cdots = \sum_{k=0}^{\infty} \left(\frac{1}{4}\right)^k = \frac{1}{1 - \frac{1}{4}} = \frac{4}{3}$$

Por lo tanto, **la pirámide completa ocupa solo $\frac{4}{3}$ (aproximadamente 1,33) del tamaño de la imagen original**. Es una representación muy compacta y eficiente.

3.1.3. Propiedades de la Pirámide Gaussiana

- **Pérdida de detalles:** a medida que ascendemos a niveles superiores (resolución más baja), los detalles finos y las altas frecuencias se difuminan (pérdida de información).

- **Preservación de estructuras:** en los niveles más altos se preservan únicamente las **grandes regiones uniformes** de la imagen original.

- **Irreversibilidad:** debido al submuestreo y al filtrado (operación "lossy" o con pérdida), **no es posible** reconstruir la imagen original exacta a partir de una capa superior borrosa de la pirámide gaussiana.

3.2. Laplacian Pyramid

Para solucionar el problema de la pérdida de información y poder lograr una representación **lossless** (sin pérdida), Burt y Adelson (1983) propusieron la pirámide laplaciana.

En lugar de almacenar las imágenes borrosas, en cada nivel de la pirámide laplaciana retenemos únicamente el **residuo** (la diferencia o detalle perdido).

Notación. Let G_k be the $k-$th level of the Gaussian pyramid. Define:

$$L_k = G_k - \text{upsample}(G_{k+1})$$

where "upsample" means expanding G_{k+1} back to the size of G_k (typically by inserting zeros and blurring).

3.2.1. Algoritmo de Construcción

El algoritmo de construcción de cada nivel funciona de la siguiente manera:

1. **Filter**: se toma la imagen del nivel actual (G_i) y se suaviza (Blur).
2. **Compute residual**: se calcula el **residuo** (imagen residual L_i) mediante la resta:
   $$L_i = G_i - \text{Blur}(G_i)$$
   Este residuo captura la información de los bordes y los detalles finos que el filtro Gaussiano elimina.
3. **Subsample**: se submuestrea la imagen borrosa para generar el siguiente nivel inferior de resolución (G_{i+1}).

3.2.2. Reconstrucción de la Imagen Original

A diferencia de la Gaussiana, la pirámide Laplaciana sí permite reconstruir la imagen original de forma exacta. Para reconstruir, necesitamos almacenar únicamente:

1. Todos los **residuos** de cada nivel ($L_0, L_1, \dots$).
2. La **imagen más pequeña** del último nivel de la pirámide (la aproximación más baja), G_N.

El **proceso de reconstrucción** (colapsar la pirámide) se realiza de forma inversa de arriba hacia abajo (Coarse-to-Fine):

Nivel Superior Up-sampled & Blurred+Residuo del nivel actual = Imagen Reconstruida del Nivel A

donde el **algoritmo de reconstrucción** se define recursivamente como:

1. Start from G_N (the coarsest level).
2. Repeat until original resolution:

a) **Upsample & Blur**: se toma la imagen del nivel superior (G_{k+1}), se expande (upsample) y se suaviza (blur) para obtener una versión borrosa del nivel actual.

b) **Add the residual**: se suma el residuo almacenado en el nivel actual (L_k) a la imagen borrosa obtenida en el paso anterior para recuperar la imagen del nivel actual:

$$G_k = \text{upsample}(G_{k+1}) + L_k$$

Nos podemos preguntar entonces, ¿por qué se llama Laplaciana?

Se denomina así porque el residuo obtenido al restar una imagen borrosa de su versión original equivale matemáticamente a aplicar un filtro **Laplaciano de Gaussiano (LoG)**.
Una **Diferencia de Gaussianas (DoG)** aproxima de manera excelente el operador diferencial Laplaciano, sirviendo como un excelente detector de bordes debido a los "zero crossings" (cruces por cero) en las transiciones de intensidad.

Observación. El término **coarse-to-fine** (que se traduce conceptualmente como "de lo grueso a lo fino" o "de menor a mayor resolución") describe una estrategia para analizar, procesar o buscar estructuras dentro de una imagen a través de diferentes escalas. Imaginemoslo de la siguiente manera:

- **Coarse (Lo grueso / Alta jerarquía):** corresponde a los niveles superiores de la pirámide, aquí la imagen ha sufrido mucho suavizado (blurring). Las texturas y los detalles finos han desaparecido , dejando únicamente las estructuras globales y las regiones uniformes grandes de la imagen original.

- **Fine (Lo fino / Base de la pirámide):** corresponde a los niveles inferiores o a la imagen original, aquí es donde residen los detalles de alta frecuencia, bordes definidos y texturas precisas.

3.3. Aplicaciones Esenciales en Computer Vision

Las pirámides de imágenes se utilizan de forma masiva en la disciplina debido a cinco aplicaciones principales:

1. **Compresión de imágenes:** el almacenamiento de residuos laplacianos tiende a tener menos entropía.

2. **Denoising (Eliminación de ruido):** separación de frecuencias para filtrar ruidos de alta frecuencia.

3. **Detección Multiescala (Multi-scale Detection):** permite buscar objetos usando plantillas fijas (Templates). Si una plantilla de búsqueda tiene un tamaño fijo en píxeles (ej. caja roja para detectar aves), solo detectará objetos que encajen perfectamente. Al pasar la misma plantilla fija a través de todos los niveles de la pirámide, podemos detectar instancias del objeto a diferentes escalas en la escena (aves pequeñas, medianas y grandes).

4. **Mezcla de imágenes (Image Blending):** permite realizar fusiones suaves (como la fusión de una manzana y una naranja) combinando las pirámides laplacianas de ambas imágenes ponderadas por una pirámide gaussiana de una máscara de transición.

5. **Registro de imágenes y mejora del Flujo Óptico (Optical Flow Improvement).**

3.4. Pirámides aplicadas al Flujo Óptico

El cálculo del flujo óptico asume pequeños desplazamientos de píxeles entre fotogramas consecutivos. Cuando hay movimientos grandes, los algoritmos estándar fallan por completo. Las pirámides de imágenes resuelven este problema mediante una estimación de **grueso a fino (Coarse-to-fine estimation)**.

Definición 3.4.1 (Flujo óptico). El **flujo óptico** es el patrón del movimiento aparente de los objetos, superficies y bordes en una escena visual, causado por el movimiento relativo entre el observador (la cámara) y la escena. En términos sencillos: es el vector de desplazamiento que nos dice hacia dónde y cuántos píxeles se ha movido cada punto de la imagen de un fotograma al siguiente (del tiempo t al tiempo $t+1$).

3.4.1. ¿Por qué se usan pirámides en el Flujo Óptico?

- **Gestión de Grandes Movimientos (Handle large motion):** si un objeto se desplaza 32 píxeles en la imagen original, un algoritmo local no lo encontrará. Sin embargo, en el nivel superior de la pirámide (por ejemplo, tras reducir el tamaño 4 veces), ese desplazamiento de 32 píxeles se convierte en un movimiento de solo $32/2^4 = 2$ píxeles. ¡Un movimiento grande a escala completa se transforma en un movimiento pequeño a escala reducida!

- **Estimación Coarse-to-Fine:** se calcula el flujo óptico primero en el nivel más pequeño y borroso (coarse). Esa estimación se escala (upsample) y se utiliza como una suposición inicial para corregir y refinar el flujo óptico en el nivel de resolución inmediatamente inferior, repitiendo el proceso hasta llegar a la resolución completa (fine).

- **Mejora de la Robustez y Velocidad (Improve robustness & Faster tracking):** al guiar la búsqueda con suposiciones iniciales macroscópicas muy sólidas, el algoritmo evita caer en mínimos locales causados por texturas repetitivas y reduce el área de búsqueda local, logrando un rastreo mucho más rápido y computacionalmente eficiente.

4. Frequency domain and Fourier Transform

Hasta ahora, hemos operado en el **dominio espacial**, donde una imagen se representa como una matriz de intensidades indexadas por coordenadas espaciales (x, y). Sin embargo, Jean Baptiste Joseph Fourier demostró que cualquier función periódica puede descomponerse en una suma de senos y cosenos de diferentes frecuencias.

El bloque básico de construcción es un sinusoidal:

$$A\sin(\omega x + \phi)$$

donde:

- A = amplitud (fuerza/intensidad)
- ω = frecuencia angular (qué tan rápido oscila)
- ϕ = fase (posición inicial del sinusoidal)

Agregando suficientes sinusoides a diferentes frecuencias ω con las amplitudes A y fases ϕ correctas, se puede obtener cualquier señal periódica deseada.

En visión por computadora, pasamos al **dominio de la frecuencia** para analizar no "dónde" ocurren los cambios de brillo, sino con qué **rapidez o tasa de variación** se producen en la imagen.

4.1. Fourier Transform

Definición 4.1.1 (Continuous Fourier Transform). La **Transformada de Fourier** descompone una señal continua $f(x)$ en su contenido de frecuencia al calcular, para cada frecuencia s, cuánto de esa frecuencia está presente:

$$F(s) = \int_{-\infty}^{+\infty} f(x)e^{-i2\pi sx}dx$$

Usando la fórmula de Euler $e^{i\theta} = \cos\theta + i\sin\theta$, esto esencialmente proyectar $f(x)$ sobre sinusoides de cada frecuencia s.

Cada $F(s)$ es un número complejo en cada frecuencia s. Codifica tanto:

- Amplitud (magnitud): $|F(s)| = \sqrt{\text{Re}(F)^2 + \text{Im}(F)^2}$, que indica qué tan fuerte es cada frecuencia.
- Fase: $\angle F(s) = \arctan\left(\frac{\text{Im}(F)}{\text{Re}(F)}\right)$, que indica el desplazamiento temporal de cada sinusoidal.

Observación. Definimos la Transformada de Fourier Continua para establecer la base teórica perfecta del análisis de señales. Nos proporciona el marco ideal para modelar filtros teóricos y entender cómo la luz interactúa físicamente con los sistemas ópticos de las cámaras.

Definición 4.1.2 (Inverse Fourier Transform). La **Transformada Inversa de Fourier** permite regresar al dominio espacial:

$$f(x) = \int_{-\infty}^{+\infty} F(s)e^{i2\pi sx}ds$$

La transformada de Fourier es invertible: $f(x) \overset{\mathcal{F}}{\leftrightarrow} F(s)$, lo que significa que no se pierde información al pasar al dominio de la frecuencia.

La **Transformada de Fourier Discreta (DFT)** es la herramienta matemática que mapea una imagen desde el dominio espacial al dominio de la frecuencia.

Definición 4.1.3. Dada una imagen digital $f(x, y)$ de tamaño $M \times N$, su **discreat Fourier transform** bidimensional, denotada como $F(u, v)$, se define formalmente como:

$$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

donde:

- x, y son las coordenadas en el espacio.
- u, v son las variables de frecuencia espacial.
- $j = \sqrt{-1}$ representa la unidad imaginaria.

Observación. La DFT es la herramienta que permite aplicar toda la magia teórica de Fourier dentro de un chip o un procesador. Sin la DFT (y su implementación algorítmica ultrarrápida, la FFT o Transformada Rápida de Fourier), sería imposible que un software procesara imágenes digitalmente en tiempo real.

Definición 4.1.4. Para regresar al dominio espacial sin perder información (operación reversible), aplicamos la **Transformada Inversa Discreta de Fourier (IDFT)**:

$$f(x,y) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

Dado que $F(u, v)$ es un número complejo, comúnmente lo visualizamos a través de dos componentes:

1. **Espectro de Magnitud ($|F(u,v)|$):** Representa la cantidad (energía) de cada frecuencia presente en la imagen.

$$\text{Magnitud} = |F(u,v)| = \sqrt{\text{Real}(F)^2 + \text{Imaginario}(F)^2}$$

2. **Ángulo de Fase ($\phi(u,v)$):** Contiene la información sobre las posiciones de las ondas.

$$\phi(u,v) = \tan^{-1}\left(\frac{\text{Imaginario}(F)}{\text{Real}(F)}\right)$$

4.2. Significado Físico de las Frecuencias en una Imagen

Cuando pensamos en frecuencias en la vida cotidiana, pensamos en el tiempo (por ejemplo, el sonido: una onda que vibra muchas veces por segundo es un sonido agudo). Pero en las imágenes no tenemos tiempo, tenemos espacio (píxeles distribuidos en un plano X e Y). Por lo tanto, hablamos de frecuencias espaciales.

Regla de oro: la frecuencia espacial mide con qué rapidez cambia el brillo (la intensidad del color) a medida que nos desplazamos caminando de un píxel al siguiente en una dirección de la imagen.

¿Qué significa realmente que una imagen tenga altas o bajas frecuencias?

4.2.1. Bajas Frecuencias (Low Frequencies)

- Corresponden a variaciones de intensidad muy lentas y suaves a lo largo del espacio.
- **¿Qué representan?** La iluminación general de la escena, los fondos homogéneos, las transiciones suaves de sombras y las formas macroscópicas estables.
- **Ejemplo**: imagina que caminas horizontalmente a lo largo de una foto que muestra un cielo despejado durante el atardecer. Empiezas en un azul profundo y, tras caminar 500 píxeles, el tono ha cambiado suavemente a un color naranja. El brillo ha cambiado, sí, pero le ha tomado mucho espacio hacerlo. Eso es una baja frecuencia.

4.2.2. Altas Frecuencias (High Frequencies)

- Corresponden a cambios abruptos y rápidos de intensidad entre píxeles adyacentes.
- **¿Qué representan?** Los **bordes**, las texturas finas, los contornos nítidos de los objetos y, muy importantemente, el **ruido de alta frecuencia** (como el ruido gaussiano o de grano).

- **Ejemplo**: imagina que ahora caminas por una foto de una cebra. Estás sobre un píxel negro como el carbón e, inmediatamente en el píxel de al lado (a un solo píxel de distancia), la intensidad salta a un blanco deslumbrante. El cambio ha sido instantáneo y radical. Eso es una alta frecuencia.

4.2.3. Información que proporciona la Transformada de Fourier

Al inspeccionar el espectro de magnitud centrado de una imagen, podemos extraer tres propiedades estructurales fundamentales:

1. **Frequency content:** nos dice si la imagen es predominantemente suave (poca energía fuera del centro) o muy detallada/ruidosa (energía dispersa hacia los bordes del espectro).

2. **Orientation information:** los patrones o líneas en el dominio espacial aparecen como ráfagas de energía perpendiculares en el dominio de la frecuencia, donde

   - Si una imagen tiene líneas verticales fuertes (como los barrotes de una celda), la intensidad de la imagen cambia bruscamente cuando te mueves de izquierda a derecha (en el eje horizontal). Por lo tanto, en el espectro de Fourier verás una ráfaga de luz brillante alineada de forma horizontal saliendo del centro.
   - Si la imagen tiene un horizonte marino muy marcado (horizontal), el cambio brusco ocurre al movernos de arriba a abajo. En el espectro verás una línea brillante vertical.

   Podemos ver en la Figura 4.3 (que usa un filtro de paso-alto), un ejemplo de cómo la orientación de las líneas en el dominio espacial se refleja en la orientación de las ráfagas de energía en el dominio de la frecuencia.

3. **Shape information:** las geometrías repetitivas o estructuras periódicas generan picos o patrones simétricos específicos en la distribución espacial de frecuencias.

Observación. Una cosa muy importante que nos da la transformada de Fourier al aplicarla a la imagen original es que se guarda la siguiente información:

- En el **centro del espectro** están almacenadas las bajas frecuencias. Las bajas frecuencias representan las formas gigantes, los fondos lisos y los cambios de luz muy lentos (por ejemplo, el color uniforme de una pared o el degradado suave del cielo).

- En la **periferia** (los bordes de la matriz) están almacenadas las altas frecuencias. Las altas frecuencias representan los cambios bruscos, instantáneos y afilados de color de la imagen (es decir: los bordes de los objetos, los detalles pequeños, las texturas rugosas y el ruido).

4.3. Filtrado Digital en el Dominio de la Frecuencia

Una de las mayores ventajas de la DFT proviene del **Teorema de Convolución**, el cual establece que la convolución en el dominio espacial equivale a una multiplicación punto a punto en el dominio de la frecuencia:

$$f(x,y) * g(x,y) \iff F(u,v) \cdot G(u,v)$$

Esto abarata enormemente el costo computacional de aplicar filtros grandes.

Tenemos estas dos igualdades también:

- **Convolución en el espacio ↔ Multiplicación en la frecuencia**

$$\mathcal{F}[g * h] = \mathcal{F}[g] \cdot \mathcal{F}[h]$$

- **Multiplicación en la frecuencia ↔ Convolución en el espacio**

$$\mathcal{F}^{-1}[g \cdot h] = \mathcal{F}^{-1}[g] * \mathcal{F}^{-1}[h]$$

4.3.1. Flujo de Trabajo Básico (Basic Workflow)

El proceso estándar para filtrar cualquier imagen en frecuencias consta de 4 pasos estrictos:

$$\boxed{\text{Imagen } f(x,y)} \longrightarrow 1.\ \text{Calcular DFT} \longrightarrow \boxed{F(u,v)} \longrightarrow 2.\ \text{Multiplicar por Máscara } H(u,v) \longrightarrow$$
$$\longrightarrow \boxed{G(u,v)} \longrightarrow 3.\ \text{Calcular IDFT} \longrightarrow \boxed{\text{Imagen Filtrada } g(x,y)}$$

donde cada caja representa una matriz de números complejos (en el caso de $F(u,v)$ y $G(u,v)$) o una matriz de intensidades reales (en el caso de $f(x,y)$ y $g(x,y)$), que representan la imagen en el dominio espacial o de la frecuencia respectivamente.

Tenemos los siguientes detalles del flujo:

- $f(x,y)$: la imagen original que deseas filtrar, representada como una matriz de intensidades indexada por coordenadas espaciales.
- $F(u,v)$: obtienes una nueva matriz del mismo tamaño, pero cuyos valores ahora son números complejos que representan frecuencias espaciales (u para la frecuencia horizontal y v para la vertical). Las bajas frecuencias se concentran típicamente en el centro y las altas en la periferia.
- $H(u,v)$: es la matriz filtro llamada Máscara o Función de Transferencia. Esta máscara actúa como un .ªtenuador.º un ecualizador.
- $G(u,v)$: obtienes un nuevo espectro modificado, donde las frecuencias que querías eliminar se han convertido en 0 y las que querías conservar se han quedado intactas.

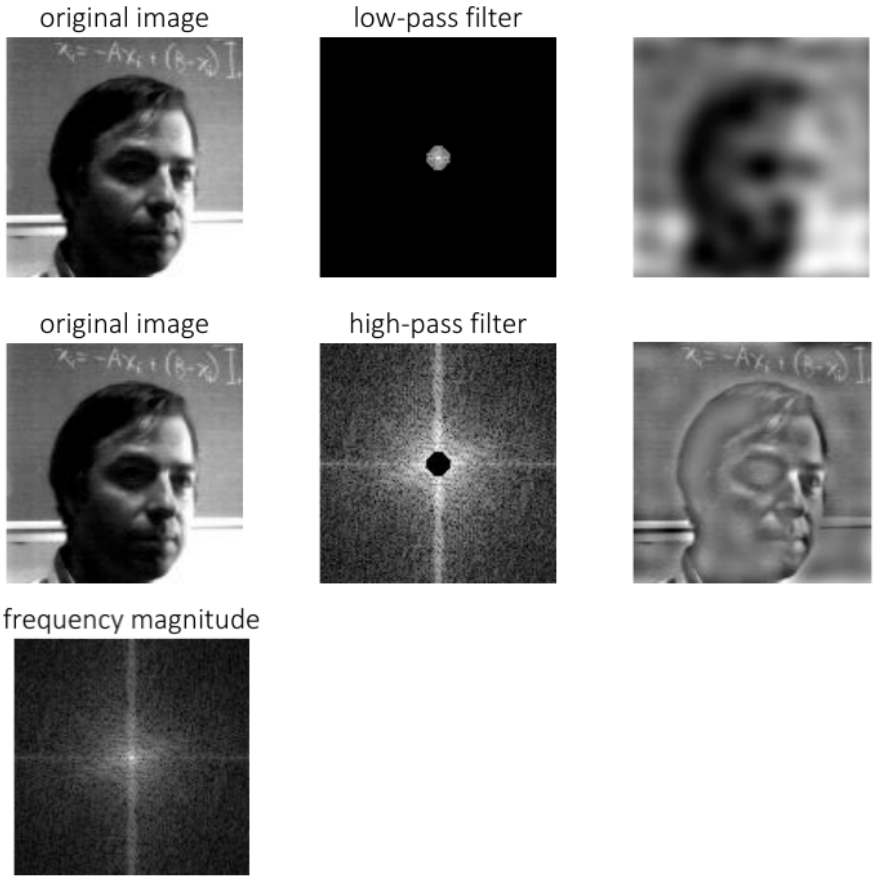

Figura 4.1: Ejemplo de frequency magnitude, and LPF and HPF aplicados a la misma imagen.

- $g(x, y)$: obtienes la imagen filtrada de vuelta en el dominio espacial. Si usaste un pasa-bajas, verás la imagen suavizada; si usaste un pasa-altas, verás solo sus bordes marcados.

Matemáticamente, el proceso de filtrado se expresa como:

$$g(x, y) = \mathcal{F}^{-1}\{F(u, v) \cdot H(u, v)\}$$

donde $H(u, v)$ es la **función de transferencia del filtro** (filtro atenuador o máscara).

4.4. Tipos de Filtros

En esta sección, nos centraremos en dos tipos de filtros fundamentales: el **filtro pasa-bajas** y el **filtro pasa-altas**. Ambos son herramientas esenciales para la mejora de imágenes (**image enhancement**), cada uno con aplicaciones específicas en la reducción de ruido y el realce de detalles respectivamente.

Podemos ver un ejemplo de ambos filtros aplicados a la misma imagen en la Figura 4.1.

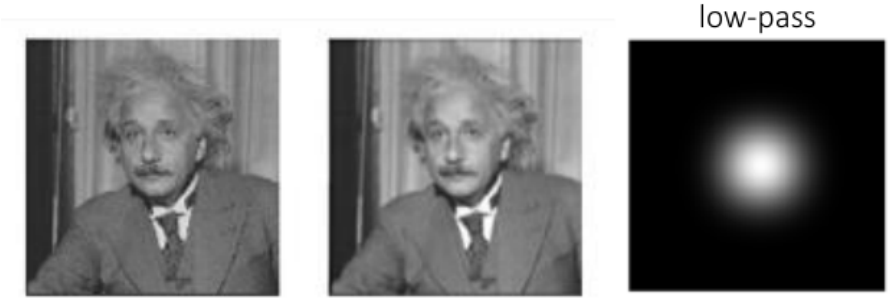

Figura 4.2: Ejemplo filtro de paso bajo (LPF)

4.4.1. Low-pass Filter (LPF)

Un filtro pasa-bajas permite el paso de las frecuencias cercanas al centro (bajas) y atenúa o elimina las frecuencias lejanas (altas), de manera que la máscara del paso-bajas es una matriz que tiene un **círculo blanco en el centro y negro en la periferia**.

- **Efecto en la imagen:** elimina los cambios bruscos de intensidad.
- **Resultado práctico:** produce un **suavizado o desenfoque (Blur/Smoothing)** y es sumamente útil para **eliminar el ruido** de alta frecuencia.
- **Tipos comunes:** filtro pasa-bajas ideal (recorte abrupto que genera artefactos de *ringing*), Butterworth y Gaussiano (suave y sin oscilaciones parásitas).

A low-pass filter would be a perfect rectangle in the frequency domain, such as

$$H_{ideal}(u,v) = \begin{cases} 1 & \text{if } |s| \leqslant s_c \\ 0 & \text{if } |s| > s_c \end{cases}$$

Podemos ver un ejemplo en la Figura 4.2.

4.4.2. High-pass Filter (HPF)

Un filtro pasa-altas bloquea las frecuencias bajas centrales y permite el paso de las frecuencias altas periféricas, de manera que la máscara del paso-alto es una matriz que tiene un **círculo negro (ceros) en el centro exacto y todo lo demás a su alrededor hasta las esquinas es completamente blanco (unos)**.

- **Efecto en la imagen:** conserva únicamente los lugares donde la intensidad cambia rápidamente y elimina las áreas de iluminación uniforme.
- **Resultado práctico:** realza los bordes y detalles finos, utilizándose de forma masiva para el afilado de imágenes (**Sharpening**) y realce de contornos.
- **Matriz de relación:** un filtro pasa-altas puede entenderse conceptualmente como restar la versión suavizada a la imagen original

$$H_{\text{high}}(u,v) = 1 - H_{\text{low}}(u,v)$$

Podemos ver un ejemplo en la Figura 4.3.

Figura 4.3: Ejemplo filtro de paso alto (HPF)

4.5. Nyquist-Shannon sampling theorem

Definición 4.5.5. El **aliasing** es un artefacto, distorsión o error de reconstrucción que ocurre cuando una señal continua (como la luz del mundo real o los detalles finos de una escena) es digitalizada (muestreada) de forma insuficiente. Cuando esto ocurre, las altas frecuencias (los detalles muy rápidos, pequeños o bordes afilados) pierden su identidad real y se camuflan adoptando una identidad falsa (un "alias") en forma de una frecuencia más baja que no existía originalmente en la escena real.

Este teorema dice que una señal continua $f(x)$ cuya transformada de Fourier $F(s)$ es cero para $|s| > s_{\max}$ (es decir, es limitada en banda) puede ser perfectamente reconstruida a partir de sus muestras tomadas a una tasa:

$$f_s \geqslant 2s_{\max}$$

cuando $f_s < 2s_{\max}$, los componentes de alta frecuencia se pliegan (alias) sobre frecuencias más bajas, corrompiendo la reconstrucción de manera irreversible.

Notación. Conceptualmente, en una cámara fotográfica, f_s es la densidad física de los píxeles en el sensor de silicio:

- Representa cuántos sensores fotosensibles individuales (fotodiodos) hay colocados por cada milímetro o pulgada cuadrada de sensor.
- A mayor f_s: Los píxeles están más juntos y apretados, lo que significa que tomas muestras de la luz de la escena real con mucha frecuencia espacial (puedes capturar detalles más pequeños).
- A menor f_s: Los píxeles están más separados entre sí, por lo que tomas muestras de la realidad de forma más espaciada y perezosa, reduciendo tu resolución espacial.

Observación. Cuando hablamos de limitada en banda, significa que la señal no contiene frecuencias por encima de un cierto umbral $s_{\max}$. En el contexto de las imágenes, esto se traduce en que no hay detalles más finos que un cierto nivel de detalle. Si intentamos muestrear una imagen con detalles más finos que el límite impuesto por la tasa de muestreo, esos detalles se distorsionarán y aparecerán como patrones falsos o artefactos en la imagen digitalizada.

Explicación física: Para poder registrar fielmente una oscilación periódica (un ciclo de línea blanca y línea negra), tu sensor necesita, como mínimo absoluto, colocar dos píxeles por cada ciclo: un píxel encargado de capturar la muestra del punto máximo (el blanco) y otro píxel encargado de capturar el punto mínimo (el negro). Si colocas menos de dos píxeles por ciclo, la matemática del sumatorio de la transformada discreta de Fourier (DFT) pierde la capacidad de distinguir esa frecuencia analógica y colapsa.

La conexión con la pirámide gaussiana es la siguiente:

- Cuando realizas un submuestreo de reducción a la mitad (2× downsampling), tu frecuencia de muestreo f_s se reduce automáticamente a la mitad. Por lo tanto, tu nueva Tasa de Nyquist máxima permitida cae a la mitad. Si dejaras la imagen tal cual, el s_{max} original violaría la nueva regla de Nyquist. Al aplicar el filtro Gaussiano justo antes de achicar la matriz, fuerzas matemáticamente a que el s_{max} de la imagen disminuya, recortando las frecuencias periféricas para que queden por debajo del nuevo límite de velocidad de la imagen pequeña.

5. Local Features - Harris Corner Detection

En visión por computadora, para resolver problemas como el emparejamiento de imágenes o el rastreo de objetos, no analizamos la imagen global de golpe. En su lugar, buscamos **características locales** (puntos de interés) que sean distintivos, repetibles y robustos a transformaciones.

El objetivo de todo este ecosistema (Harris + Feature Matching + RANSAC) no es tomar dos fotos globales y decir: "A ver, ¿se parecen en un 80 %?". No estamos haciendo una comparación global de imágenes. Su verdadero propósito es localizar elementos idénticos dentro de ellas para poder alinearlas, transformarlas y unirlas geométricamente (Image Alignment & Stitching).

El flujo de trabajo se divide en dos componentes críticos que debemos diferenciar perfectamente:

1. **Detector de Características (Feature Detector):** algoritmo encargado de buscar e identificar de forma precisa *dónde* están los puntos de interés en la imagen (coordenadas (x, y)), como esquinas o manchas.
2. **Descriptor de Características (Feature Descriptor):** un vector matemático que codifica la apariencia visual de la vecindad o parche que rodea al punto detectado. Convierte la información visual en una firma matemática única para poder compararla con descriptores de otras imágenes.

Definición 1. Un **feature descriptor** es un vector matemático de números (una lista ordenada de valores continuos) que codifica de forma compacta y única cómo se ve visualmente la región o parche de la imagen que rodea a esa esquina. En lugar de guardar solo el píxel central de la esquina, el descriptor analiza el vecindario (por ejemplo, una ventana de 16×16 píxeles alrededor de la esquina). A esa ventana le extrae información matemática clave, como por ejemplo:

- **Dirección**: dirección hacia la que apuntan los gradientes de brillo (si hay cambios hacia arriba, abajo, diagonales).
- **Textura**: la distribución de las texturas locales.
- **Contraste**: el contraste o las intensidades relativas.

Toda esa estructura visual se "aplana" y se transforma en un vector de números (un vector de características).

5.1. Harris Corner Detector

Definición 5.1.2. Una **esquina** (o keypoint) se define idealmente como una región de la imagen donde la intensidad del brillo cambia drásticamente en todas las direcciones. Cuando hablamos de todas las direcciones, nos referimos a dos fuerzas de cambio ortogonales (perpendiculares) e independientes que rellenan todo el espacio bidimensional.

Observación. Uno de primeras puede pensar que para detectar una esquina, basta con analizar los cambios de intensidad en horizontal y vertical. Sin embargo, esto no es suficiente. Para entenderlo, imaginemos una línea diagonal perfectamente inclinada a 45°. Si nos movemos un píxel hacia la derecha (horizontal), el brillo cambia. Si nos movemos un píxel hacia abajo (vertical), el brillo también cambia. Alguien podría pensar: "¡Ostras! Cambia en horizontal y cambia en vertical, ¡entonces es una esquina!". **Falso**. Si en esa misma línea diagonal intentamos moverte en la dirección de la propia diagonal (combinando avanzar a la derecha y avanzar hacia abajo a la vez), verás que el brillo no cambia en absoluto, porque te estás deslizando a lo largo de la línea. Por lo tanto, no es una esquina, es un borde inclinado (Edge). Sus cambios horizontales y verticales estaban "atados" por la misma pendiente.

Observación. Cuando hablamos de esquina, no tiene porque ser una esquina geométrica perfecta, puede ser cualquier punto que tenga un cambio de intensidad fuerte en todas las direcciones, por lo que muchas veces se nombrará como **punto de interés** o **keypoint** para evitar confusiones. En la práctica, lo que el detector de Harris busca es precisamente eso: puntos de interés que tengan una estructura local rica en información visual, lo que los hace ideales para tareas de emparejamiento y reconocimiento.

Para entonces poder analizar con exactitud una esquina, seguiremos el siguiente proceso y nos centraremos en el **tensor de estructura** (structure tensor) que es la base matemática del detector de Harris.

Primero, analizamos cómo cambia la intensidad al desplazar una ventana local (u, v) una pequeña distancia (x, y). La función de suma de diferencias al cuadrado (E) se define como:

$$E(x, y) = \sum_{u,v} w(u, v) \left[I(u + x, v + y) - I(u, v) \right]^2$$

donde $w(u, v)$ es una función de ventana (uniforme o Gaussiana). Aplicando una aproximación por **Serie de Taylor** de primer orden para pequeñas traslaciones:

$$I(u + x, v + y) \approx I(u, v) + I_x x + I_y y$$

y sustituyendo esto en la ecuación original, llegamos a la forma cuadrática fundamental

$$E(x, y) \approx \begin{bmatrix} x & y \end{bmatrix} H \begin{bmatrix} x \\ y \end{bmatrix}$$

Notación. Aquí tenemos dos términos importante tener claros:

- I_x (**Gradiente Horizontal**): Mide cuánto cambia el brillo de la imagen cuando te mueves de izquierda a derecha (horizontalmente, a lo largo del eje X).
- I_y (**Gradiente Vertical**): Mide cuánto cambia el brillo de la imagen cuando te mueves de arriba a abajo (verticalmente, a lo largo del eje Y).

5.1.1. Structure Tensor

Definición 5.1.3. La matriz H se conoce como el **Tensor de Estructura** (o matriz de covarianza de gradientes) y condensa la distribución de los gradientes locales en la ventana:

$$H = \sum_{u,v} w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix}$$

Las propiedades geométricas de la vecindad quedan codificadas por los **autovalores** (λ_1, λ_2) de la matriz H:

- **Región Plana (Flat):** $\lambda_1 \approx 0$ y $\lambda_2 \approx 0$ (no hay cambios de intensidad en ninguna dirección).
- **Edge:** un autovalor es grande y el otro es pequeño ($\lambda_1 \gg 0, \lambda_2 \approx 0$). Solo hay cambio de intensidad perpendicular al borde.
- **Corner:** ambos autovalores son grandes ($\lambda_1 \gg 0$ y $\lambda_2 \gg 0$). Hay cambios fuertes en todas las direcciones.

5.1.2. Función de Respuesta de Esquina (R)

Calcular los autovalores exactos para cada píxel es computacionalmente costoso. Harris y Stephens propusieron una alternativa elegante utilizando el determinante y la traza de la matriz H:

$$R = \det(H) - k \cdot (\text{traza}(H))^2$$

Donde:

- $\det(H) = \lambda_1 \lambda_2 = (I_x^2)(I_y^2) - (I_x I_y)^2$
- $\text{traza}(H) = \lambda_1 + \lambda_2 = I_x^2 + I_y^2$
- k es una constante empírica (típicamente $k \in [0{,}04, 0{,}06]$).

Clasificación según la respuesta R

- Si $R > 0 \longrightarrow$ Es una **esquina** (Ambos autovalores dominan).
- Si $R < 0 \longrightarrow$ Es un **borde** ($\det(H)$ es pequeño frente a la traza al cuadrado).
- Si $|R|$ es pequeño $\longrightarrow$ Es una **región plana**.

5.1.3. Algoritmo de Harris

Para programar o evaluar el detector de Harris, se siguen estos pasos ordenados:

1. **Compute gradients:** calcular las derivadas espaciales de la imagen I_x e I_y (por ejemplo, usando filtros de Sobel).

2. **Build structure tensor:** calcular los productos de gradientes píxel a píxel: I_x^2, I_y^2, e I_xI_y.

3. **Apply Window Smoothing (suavizado de la ventana):** filtrar las imágenes de los componentes anteriores con una Gaussiana $w(u, v)$ para obtener los promedios locales $\langle I_x^2 \rangle$, $\langle I_y^2 \rangle$ y $\langle I_xI_y \rangle$.

4. **Compute corner response (R):** evaluar la fórmula de Harris para cada píxel, de manera que se tiene una matriz de valores R de igual tamaño que la imágen original que indica la fuerza de la esquina en cada ubicación.

5. **Threshold corners** :filtrar los píxeles manteniendo solo aquellos donde R supere un umbral predefinido T_{h_harris} lo que provocará que una zona (por ejemplo, alrededor del pico de una mesa) se detecte como una esquina fuerte.

6. **Non-Maximum Suppression**: se realizará una búsqueda local para conservar únicamente los picos locales (máximos locales), evitando que una sola esquina genere múltiples detecciones agrupadas, y podamos obtener un conjunto limpio de puntos de interés sin redundancias. Lo que se hace es lo siguiente:

   a) Se define un vecindario local, es decir, una ventana de tamaño fijo centrada en el píxel actual.

   b) Se compara el valor de R del píxel central con los valores de R de todos los píxeles vecinos dentro de esa ventana.

   c) Si el píxel central tiene el valor de R más alto de todo su vecindario, se conserva como una esquina válida. Si cualquier vecino tiene un valor de R estrictamente mayor que el píxel central, entonces el píxel central se descarta (se le asigna un valor de 0), ya que no es un pico local.

5.1.4. Robustez y Limitaciones de Harris

- **Robusto a la Rotación (Rotation invariant):** la matriz M rota de forma solidaria con la imagen, pero sus autovalores permanecen matemáticamente **invariantes**. Por ende, una esquina sigue detectándose como esquina aunque giremos la foto.

- **Robusto a cambios de intensidad (Intensity shifts):** al basarse estrictamente en derivadas (gradientes), añadir una constante de brillo a la imagen no altera el resultado (I_x sigue siendo igual). Es parcialmente robusto a cambios afines de iluminación.

- **NO es invariante a la Escala (Not scale invariant):** *¡Pregunta clave de examen!* Si haces un zoom gigante a una esquina, lo que a escala pequeña parecía un punto afilado pasa a verse a escala macroscópica como un borde suave y plano dentro de la pequeña ventana del filtro. El detector de Harris estándar falla si hay cambios drásticos de escala.

5.2. Feature Matching

Una vez que hemos detectado los puntos e indexado sus descriptores vectoriales de apariencia entre dos imágenes distintas, debemos cruzarlos para encontrar correspondencias. La idea aqui, es poder encontrar puntos que se vean igual en ambas fotos, aunque estén tomadas desde ángulos o condiciones de iluminación diferentes. Para esto, existen varias estrategias de matching que se basan en comparar la similitud entre los descriptores de las esquinas detectadas en ambas imágenes.

5.2.1. Estrategias de Matching

- **Descriptor distance comparison:** medimos la similitud calculando la distancia euclídea (L_2) o la distancia de Manhattan (L_1) entre los vectores de características.
- **Nearest Neighbor Matching (NN):** para un punto de interés en la Imagen A, buscamos el punto en la Imagen B cuyo descriptor tenga la distancia mínima absoluta.
- **Ratio Test (Lowe idea):** el vecino más cercano puro puede fallar si hay texturas repetitivas. Para asegurar que el match es ambiguo o sólido, calculamos la razón entre la distancia del primer vecino más cercano (d_1) y la distancia del segundo vecino más cercano (d_2):

$$\text{Ratio} = \frac{d_1}{d_2}$$

  Si esta razón es menor que un umbral estricto (ej. $\leq 0{,}8$), aceptamos el match. Si se acerca a 1, significa que el punto se parece a múltiples zonas de la imagen (ambigüedad), por lo que se descarta.

5.3. Algoritmo RANSAC

A pesar de aplicar el ratio test, en el mundo real siempre aparecen **outliers**, por lo que para estimar un modelo geométrico válido (como una homografía o matriz fundamental) ignorando los outliers, se utiliza el algoritmo **RANSAC (RANdom Sample Consensus)**.

Definición 5.3.4. Un **outlier** es un punto de correspondencia que no sigue el modelo geométrico dominante que estamos tratando de estimar. En otras palabras, es un match erróneo causado por oclusiones, ruido o cambios de perspectiva.

5.3.1. Algoritmo RANSAC

RANSAC es un enfoque iterativo basado en la siguiente filosofía de muestreo aleatorio:

1. **Randomly sample:** seleccionar de forma completamente aleatoria un subconjunto mínimo de correspondencias necesarias para calcular el modelo matemático (por ejemplo, 4 pares de puntos para una Homografía). Ese subconjunto se toma de nuestro conjunto de **esquinas** (o keypoints) emparejadas entre las dos imágenes, usando feature matching.

2. **Estimate model:** calcular los parámetros del modelo geométrico provisional basándose únicamente en ese subconjunto mínimo.

3. **Count inliers:** evaluar el modelo calculado con el resto de todas las correspondencias de la imagen. Contar cuántos puntos se adaptan al modelo dentro de un umbral de tolerancia de error (estos puntos válidos se llaman **Inliers**), de la siguiente manera:

   - Tomamos un punto p_i de la primera imagen y lo proyectamos usando el modelo provisional calculado (Hp_i).
   - Medimos la distancia euclídea en píxeles entre dónde cayó esa proyección teórica y dónde está el punto real emparejado en la segunda imagen (p'_i).
   - Si esa distancia es menor que un umbral de tolerancia ε (un margen de error pequeño, por ejemplo, 1 o 2 píxeles), el punto se considera un Inlier (un voto a favor del modelo). Si la distancia es mayor, se ignora por ser un outlier.

4. **Repeat:** repetir este proceso de muestreo (los 3 pasos anteriores), estimación y conteo de inliers un número máximo de iteraciones predefinido.

   Nosotros para cada modelo tomamos $s = 4$ parejas de puntos, con un ratio de outliers e (por ejemplo, $e = 0{,}5$ si la mitad de los matches son erróneos) y queremos que el modelo correcto salga con una probabilidad de éxito p (por ejemplo, $p = 0{,}99$), entonces el número de iteraciones necesarias para garantizar ese nivel de confianza se calcula con la fórmula:

   $$N \geqslant \frac{\log(1-p)}{\log(1-(1-e)^s)}$$

   Observación. El outlier ratio e es una estimación de cuántos matches erróneos hay en nuestro conjunto de correspondencias, como esto no se sabe con certeza en el laboratorio, se suele asumir un valor conservador (ej. $e = 0{,}5$) para garantizar que el número de iteraciones sea suficiente para encontrar un modelo correcto. Podemos ver en la Figura 5.1 una tabla que indica el número de iteraciones necesarias para diferentes valores de s y e para alcanzar una probabilidad de éxito del 99 %.

5. **Keep best model:** conservar el mejor modelo matemático, que corresponde a aquel que logró reclutar el **mayor número de inliers**.

	proportion of outliers e						
s	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177

$p = 0.99$

Figura 5.1: Número de iteraciones necesarias para probabilidad de éxito del 99 % en RANSAC, dependiendo del número de puntos mínimos s y el ratio de outliers e.

6. **Refinement:** opcionalmente, reajustar el modelo final mediante mínimos cuadrados utilizando el conjunto completo de todos los inliers descubiertos. Esto se refiere a recalcular un modelo nuevo usando todos los inliers (del modelo ganador) y sus errores para obtener una estimación más precisa, usando técnicas de optimización global, como **Mínimos Cuadrados (Least Squares)**.

   En este nuevo modelo, se obvian las 4 primeras parejas aletorias (el modelo se genera gracias a todos los inliers) y se eliminan los outliers que no se consiguieron juntar, para que no afecten a la precisión del modelo final.

Observación. Puede ocurrir que en las parejas asociadas en el feature matching, haya outliers, de manera que en el algoritmo RANSAC, el modelo usado siempre este comparando la proyección de un keypoint con un matching erróneo, pero eso no es un gran problema, porque el modelo correcto (que se genera a partir de un subconjunto de inliers) seguirá reclutando más inliers que cualquier modelo erróneo generado a partir de outliers, por lo que al final el modelo correcto será el que tenga más votos (inliers) y ganará la competición.

5.4. Aplicaciones de las Características Locales

Dominar este flujo de descriptores, detectores y RANSAC nos permite habilitar las siguientes tecnologías fundamentales expuestas en las diapositivas:

1. **Image Stitching / Panorama Creation:** unir múltiples fotografías de un paisaje buscando esquinas comunes, emparejando sus descriptores, calculando la homografía con RANSAC y proyectándolas en un lienzo común sin costuras visibles.

2. **Reconocimiento de Objetos (Object Recognition):** identificar un objeto en una escena desordenada emparejando sus puntos de interés locales con los de una base de datos del objeto plantilla.

3. **Structure from Motion (SfM):** reconstruir la geometría tridimensional 3D de una escena y las posiciones de las cámaras en movimiento rastreando las

trayectorias de los puntos de interés emparejados a lo largo de una secuencia de video o colección de imágenes.

6. Local Features - Scale Invariant Feature Transform

El detector de esquinas de Harris presenta una limitación crítica: **no es invariante a los cambios de escala**. Un punto que se comporta como una esquina a escala fina se convierte en un borde o una región plana cuando se examina a una escala macroscópica (zoom).

Para resolver esto, necesitamos detectar estructuras que sean estables en un espacio continuo de posiciones y escalas. El algoritmo **SIFT (Scale Invariant Feature Transform)** aborda este problema mediante la construcción de un *Scale-Space* (Espacio de Escala).

6.1. El Pipeline Completo de SIFT

El algoritmo SIFT funciona tanto como un **detector** como un **descriptor** de características locales a través de 4 pasos estrictos y secuenciales que debes memorizar.

El objetivo absoluto de SIFT (Scale Invariant Feature Transform) es extraer puntos clave de una imagen que sean tan únicos e invariantes que el ordenador pueda volver a encontrarlos y emparejarlos en otra foto, incluso si esa otra foto ha cambiado de escala (zoom), ha rotado, o tiene una iluminación diferente.

Básicamente, busca crear un "DNI matemático" para partes de la imagen que sea inmune a los cambios del mundo físico.

6.1.1. Paso 1: Construcción del Espacio de Escala y Diferencia de Gaussianas (DoG)

Para buscar características en múltiples escalas simultáneamente, la imagen se filtra repetidamente con núcleos Gaussianos utilizando diferentes desviaciones estándar (σ).

- **Espacio de Escala $L(x, y, \sigma)$:** se obtiene mediante la convolución de la imagen de entrada $I(x, y)$ con un filtro Gaussiano variable $G(x, y, \sigma)$:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

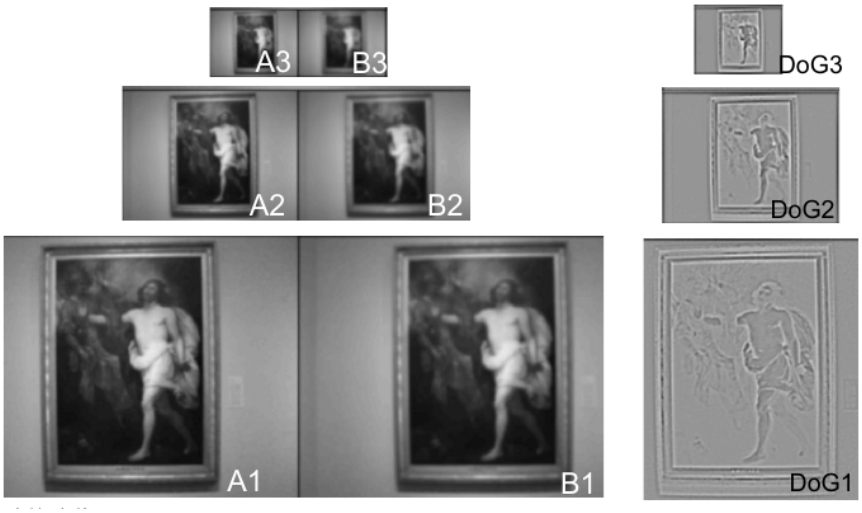

Figura 6.1: Ejemplo de pirámide de escala y las imágenes DoG resultantes.

Las imágenes se organizan en **Octavas**. Al pasar a la siguiente octava, la imagen se submuestrea a la mitad de su tamaño físico y el valor de σ se duplica.

Observación. Una octava no es una sola imagen, sino un grupo o conjunto de imágenes que tienen el mismo tamaño de píxeles, pero diferentes niveles de blur (distinto valor de σ), por lo que en lo que llamabamos pirámide de escala, cada octava es un bloque vertical de imágenes con misma cantidad de píxeles pero diferentes σ.
Como todas las imágenes de esta octava mantienen el mismo tamaño (mismos píxeles de ancho y alto), el ordenador puede restarlas directamente entre sí para obtener las matrices DoG (Diferencia de Gaussianas).

- **Difference of Gaussians (DoG):** calcular el Laplaciano de Gaussiano (LoG) de forma directa es computacionalmente costoso. SIFT solventa esto aproximando el LoG mediante la resta aritmética de dos imágenes adyacentes en la escala, separadas por un factor constante k:

$$DoG(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$$

Observación. La relación de DoG con el LoG es la siguiente

$$DoG(x, y, \sigma) \approx (k-1)\sigma^2\nabla^2 L \quad \text{donde LoG} = \sigma\nabla^2 L$$

lo que significa que la matriz DoG es matemáticamente igual al LoG multiplicado por una constante constante ($(k-1)\sigma^2$). Como en SIFT solo nos interesa buscar los puntos donde la respuesta es máxima (los extremos locales), multiplicar toda la matriz por una constante no cambia la posición de los máximos ni de los mínimos.

Podemos ver un ejemplo de la pirámide de escala y las imágenes DoG resultantes en la Figura 6.1.

Definición 6.1.1 (Laplaciano de Gaussiano (LoG)). El **Laplaciano de Gaussiano** es un operador matemático diseñado para detectar estructuras circulares o manchas (llamadas blobs) en una imagen a una escala determinada (σ). Se compone de dos pasos lógicos combinados en un solo filtro:

- **Suavizado Gaussiano:** primero se aplica un filtro Gaussiano a la imagen para eliminar el ruido y seleccionar la escala del detalle que queremos buscar (controlada por σ).
- **Operador Laplaciano (∇^2):** después se calcula la segunda derivada espacial ($L_{xx}+L_{yy}$). Físicamente, la derivada segunda busca dónde terminan las rampas de brillo. Cuando se aplica sobre una estructura circular que coincide con el tamaño del filtro, la respuesta matemática alcanza un pico máximo (un extremo).

6.1.2. Paso 2: Keypoint Localization

Una vez obtenidas las matrices DoG, el algoritmo busca candidatos a puntos clave identificando los **extremos locales** (máximos o mínimos) en un entorno tridimensional (posición y escala):

- Cada píxel en una imagen DoG se compara con sus 8 vecinos espaciales en la misma capa, más los 9 vecinos de la capa superior y los 9 vecinos de la capa inferior (un total de 26 comparaciones). Si el valor del píxel central es estrictamente mayor que los 26 vecinos, o estrictamente menor que los 26 vecinos, se le corona como **extremo local** (Máximo o Mínimo), y se almacena como sigue
$$\mathbf{x} = \{x, y, \sigma\}$$
- **Refinamiento y Eliminación de puntos inestables:** muchos candidatos iniciales son inestables. Se aplica una aproximación por serie de Taylor para localizar el extremo con precisión sub-píxel. Se descartan de forma inmediata:
  1. Los puntos con bajo contraste (sensibles al ruido).
  2. Los puntos localizados a lo largo de bordes rectos (falsos positivos). Tomamos H la matriz Hessiana de la DoG en el punto clave, y tenemos que si se cumple
  $$\frac{(\lambda_1 + \lambda_2)^2}{\lambda_1 \lambda_2} > \frac{(r+1)^2}{r}$$
  con $r = 10$, entonces el punto se descarta por ser demasiado parecido a un borde (un eigenvalor es mucho mayor que el otro), porque a la hora de intentar emparejarlo con otra imagen, va a coincidir con cualquier borde que tenga la misma orientación, lo que genera muchos falsos positivos.

6.1.3. Paso 3: Keypoint Orientation Assignment

Para lograr la **invariancia a la rotación**, se calcula una dirección dominante para cada punto clave superviviente (dada la ubicación (x, y) y la capa σ) basada en las propiedades del gradiente local:

- Se calcula la magnitud $m(x, y)$ y la orientación $\theta(x, y)$ del gradiente en una vecindad Gaussiana alrededor del punto.

- Se crea un **histograma de orientaciones** con 36 contenedores (*bins*), donde cada contenedor cubre un rango de 10°. Cada píxel vota en el histograma ponderado por la magnitud de su gradiente y por una ventana Gaussiana central.

    - El voto de un píxel no vale 1, vale la magnitud de su propio gradiente (si el brillo cambia mucho, su voto tiene mucha fuerza) multiplicado por una ventana Gaussiana central. Esto significa que los píxeles que están pegados al punto clave tienen votos sumamente importantes, mientras que los píxeles que están más alejados en los bordes de la ventana apenas influyen en el resultado.

- El pico más alto del histograma se define como la **orientación dominante** del punto clave. Si existe otro pico que supere el 80 % del máximo, se crea un segundo punto clave independiente en la misma posición pero con esta nueva orientación.

Observación. Es perfectamente posible que coincidan puntos clave en el mismo píxel de la misma octava. Si tienen distinto blur, significa que ese mismo punto del espacio contiene información física de diferentes tamaños (un detalle pequeño y una estructura grande a la vez). Si tienen el mismo blur, puede deberse a que el punto tiene múltiples orientaciones dominantes que superaron el umbral del 80 %.

6.1.4. Paso 4: Descriptor Generation

Con la posición, escala y orientación fija, el algoritmo construye la firma matemática (descriptor):

1. Tomar la imagen suavizada por Gauss a la escala σ del punto clave.

2. Se toma una ventana de 16×16 píxeles alrededor del punto clave.

3. La ventana se **rota alineándola con la orientación dominante** calculada en el paso anterior (garantizando inmunidad al giro).
   Esto sirve para que podamos alinear el descriptor con la dirección de la característica, de modo que si la imagen se gira, el descriptor se ajustará automáticamente a esa rotación, manteniendo su consistencia, con la imagen original.

4. Esta ventana de 16×16 se subdivide en 16 sub-bloques más pequeños de 4×4 píxeles cada uno.

5. Para cada sub-bloque de 4×4, se calcula un histograma de orientaciones de gradiente abreviado con 8 contenedores (direcciones cardinales principales), donde cada píxel vota con su magnitud de gradiente. Esto da un vector de 8 dimensiones por sub-bloque.

6. El descriptor final es la concatenación de todos los histogramas

$$16 \text{ sub-bloques} \times 8 \text{ direcciones} = \mathbf{128} \text{ dimensiones}$$

un vector de 128 números.

7. Al final se aplica **normalización** (inmunidad lumínica): ese vector de 128 números se divide por su norma euclídea (L_2). Al normalizar el vector a longitud unidad, si multiplicas el brillo de la imagen entera por dos (un cambio de iluminación global), las magnitudes de los gradientes se duplican, pero al normalizar, el vector final de 128 números vuelve a dar exactamente el mismo resultado. El descriptor se vuelve invariante a la iluminación.

6.2. ¿Por qué SIFT es Invariante? (Justificación Teórica)

Memorizar las razones exactas de su robustez frente al mundo físico:

- **Invariancia a la Escala:** gracias a la estructura de **pirámide multi-escala** y la búsqueda tridimensional de extremos en el espacio DoG (lo que se hace en Paso 2 al trabajar con la posición (x, y) y la escala σ). La característica se detectará en la capa que mejor se adapte a su tamaño real.

- **Invariancia a la Rotación:** gracias a la **asignación de la orientación dominante**. Como el descriptor rota sus ejes adaptándose a este ángulo antes de medir los gradientes, la firma de 128 dimensiones no cambia si la imagen se gira.

- **Invariancia a la Iluminación:** al basarse puramente en gradientes de intensidad (derivadas métricas), se eliminan los desplazamientos constantes de brillo. Además, el vector final de 128 dimensiones se **normaliza a longitud unidad** (L_2), lo que cancela los cambios multiplicativos de contraste.

6.2.1. Limitaciones de SIFT

- **Lento y costoso computacionalmente**: la construcción de decenas de niveles Gaussianos, restas DoG e histogramas densos de 128 dimensiones consume demasiados recursos, imposibilitando su uso nativo en sistemas de tiempo real embebidos o robótica ligera antigua.

6.3. Evolución y Alternativas: SURF, BRIEF y ORB

Debido a la lentitud de SIFT, surgieron variantes optimizadas en la literatura:

6.3.1. SURF (Speeded Up Robust Features)

SURF nace con una sola misión: replicar la robustez de SIFT a la escala y la rotación, pero reduciendo drásticamente el tiempo de cálculo.

- **Filosofía:** es una versión acelerada de SIFT basada en aproximaciones drásticas, usando **filtros de caja cuadrados** (Box Filters) ($\mathcal{O}(1)$) en lugar de Gaussianos (no hacemos tantas multiplicaciones de flotantes), y aprovechando las **imágenes integrales** para calcular respuestas de escala en tiempo constante ($\mathcal{O}(1)$).

- **Detection**: utiliza un detector basado en la matriz Hessiana, donde el Laplaciano se aproxima con filtros de caja de 9×9 aplicados sobre la imagen integral.

- **Scale-space**: en lugar de reducir la imagen, SURF aumenta el tamaño del filtro para simular la búsqueda en escala, evitando el submuestreo (empezamos con un filtro de 9×9, luego pasamos a uno de 15×15, 21×21, etc.). Como la imagen integral tarda lo mismo en procesar un filtro pequeño que uno gigante, escalar el filtro es gratis a nivel computacional.

- **Resultado:** mucho más rápido que SIFT, aunque a veces sacrifica precisión en transformaciones de perspectiva extremas. Su descriptor estándar se reduce a 64 dimensiones.

Observación. $I_{\sum}(x, y) = \sum_{i=0}^{x} \sum_{j=0}^{y} I(i, j)$ es la imagen integral, que permite calcular la suma de cualquier región rectangular en tiempo constante con solo 4 accesos a memoria.

6.3.2. BRIEF vs ORB: Descriptores Binarios

Para entornos de tiempo real estricto, la distancia Euclídea entre vectores flotantes (como las 128 dimensiones de SIFT) es un cuello de botella. Nacen los descriptores binarios:

- **BRIEF (Binary Robust Independent Elementary Features):**

  - **Concepto:** no calcula histogramas ni gradientes. Toma un parche suavizado y realiza una serie de pruebas de comparación binarias de intensidad (brillo) entre parejas de píxeles aleatorias predefinidas.
  - **El código:** Si el píxel $A > B \to 1$; si no $\to 0$, es decir

$$\tau(p; A, B) = \begin{cases} 1 & \text{si } I(A) > I(B) \\ 0 & \text{si } I(A) \leq I(B) \end{cases}$$

    y se tiene que el descriptor es una cadena de bits (ej. 256 bits). El matching es ultra veloz porque se mide con la **distancia de Hamming** (operaciones XOR a nivel de procesador).

- **Defecto crítico:** *¡Pregunta típica de examen!* BRIEF **NO es invariante a la rotación**. Si la imagen gira, las parejas fijas de píxeles apuntan a zonas erróneas.

- **ORB (Oriented FAST and Rotated BRIEF):**

  - **Concepto:** es la alternativa moderna, libre de patentes y ultra eficiente para tiempo real.
  - **Mejora sobre BRIEF:** añade invariancia a la rotación calculando el **centroide de intensidad** del parche para deducir su orientación. Luego, rota de forma solidaria el patrón de pruebas binarias de BRIEF antes de generar la cadena de bits.

    El algoritmo calcula el centroide de intensidad (el "centro de masa" del brillo) del parche. Imagina que el parche es una superficie y el brillo es el peso. Si un lado del parche es muy blanco y brillante, y el otro es oscuro, el centro de masa se desplazará hacia la zona brillante. ORB traza un vector que va desde el centro geométrico del parche $(0, 0)$ hasta ese centroide de intensidad (C_x, C_y). El ángulo de este vector se convierte en la orientación del punto clave (θ).
  - **Ventajas:** es tan rápido como BRIEF, inmune a la rotación, y órdenes de magnitud más veloz que SIFT, ideal para SLAM visual en tiempo real.

7. 2D Transformation and Alignment

Es fundamental diferenciar las dos operaciones principales de modificación de imágenes en visión por computadora:

- **Image Filtering (Filtrado):** modifica el *rango* (los valores de intensidad/brillo de los píxeles), pero mantiene las posiciones espaciales intactas. Su expresión matemática es
$$g(x) = h(f(x))$$
como por ejemplo Convolución Gaussiana, Sobel.

- **Image Warping (Deformación):** modifica el *dominio* (las posiciones geométricas de los píxeles), mapeando las coordenadas de una imagen a nuevas posiciones espaciales. Su expresión matemática es
$$g(x) = f(h(x))$$
como por ejemplo rotación, distorsión de lente, homografías.

7.1. Transformaciones 2D

Para poder representar traslaciones y proyecciones complejas mediante multiplicaciones de matrices compactas, convertimos las coordenadas cartesianas 2D (x, y) en **Coordenadas Homogéneas** añadiendo un factor de escala w al vector:

$$\begin{bmatrix} x \\ y \end{bmatrix} \Longrightarrow \begin{bmatrix} x_h \\ y_h \\ w \end{bmatrix} = \begin{bmatrix} w \cdot x \\ w \cdot y \\ w \end{bmatrix}$$

Para regresar al plano cartesiano físico, simplemente dividimos las coordenadas escaladas entre el último elemento: $(x, y) = (x_h/w, y_h/w)$. Si $w = 1$, las coordenadas corresponden directamente a los píxeles físicos.

7.1.1. Tipos de Transformaciones Geométricas 2D

A continuación se detallan las transformaciones geométricas del tema organizadas por sus grados de libertad (DoF) y sus propiedades de invarianza. Todas se expresan como $p' = M \cdot p$:

1. Traslación (Translation) – 2 DoF

Desplaza los puntos una distancia fija en el eje X (t_x) y en el eje Y (t_y):

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2. Rotación (Rotation) – 1 DoF (3 DoF combinada con traslación)

Gira los puntos un ángulo θ con respecto al origen de coordenadas de la imagen:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

3. Escalado (Scaling) – 2 DoF

Modifica el tamaño de la estructura visual aplicando factores de escala independientes s_x y s_y:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

4. Transformación Afín (Affine Transform) – 6 DoF

Es una transformación lineal general que combina rotación, escalado, traslación y cizalladura (*shear*). Su propiedad fundamental es que **preserva las líneas paralelas**.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & t_x \\ a_3 & a_4 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Definición 7.1.1. Una **cizalladura (shear)** es una deformación que desplaza los puntos de la imagen en una dirección de forma proporcional a la distancia que tengan respecto a un eje fijo. Los puntos que están en la base (distancia cero) no se mueven nada, y cuanto más subes en altura (coordenada Y más alta), más se desplazan los píxeles hacia un lado (coordenada X).

Tenemos dos tipos de cizalladura dependiendo del eje fijo que elijamos:

- **Cizalladura Horizontal (Shear en X):** el eje fijo es el horizontal (X), y los píxeles se desplazan horizontalmente según su altura. Su matriz pura es:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \implies \begin{cases} x' = x + sh_x \cdot y \\ y' = y \end{cases}$$

  la nueva altura y' se queda exactamente igual. Sin embargo, la nueva posición horizontal x' es la posición original x más un extra que depende de la altura (y) multiplicada por el factor de cizalladura sh_x.

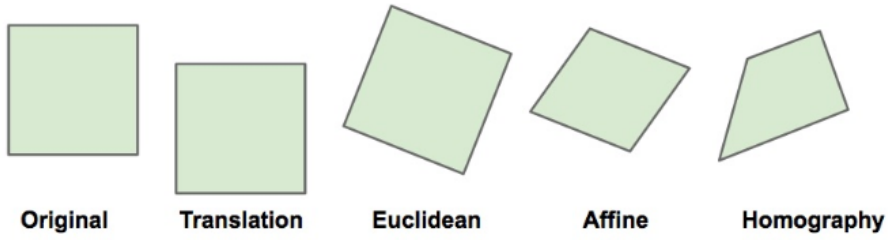

Figura 7.1: Example 2D transformations.

- **Cizalladura Vertical (Shear en Y)**: el eje fijo es el vertical (Y), y los píxeles se desplazan verticalmente según su distancia horizontal. Su matriz pura es:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \implies \begin{cases} x' = x \\ y' = y + sh_y \cdot x \end{cases}$$

  la nueva posición horizontal x' se queda exactamente igual. Sin embargo, la nueva altura y' es la posición original y más un extra que depende de la distancia horizontal (x) multiplicada por el factor de cizalladura sh_y.

5. Homografía (Homography / Projective) – 8 DoF

Es la transformación más general en el plano geométrico. Representa una transformación proyectiva que mapea puntos de un plano físico a otro plano bajo una perspectiva de cámara distinta. **No preserva el paralelismo**, pero sí preserva la colinealidad (las líneas rectas siguen siendo líneas rectas).

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \sim \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Dado que opera bajo un factor de escala global en coordenadas homogéneas, la matriz contiene 9 elementos pero solo **8 grados de libertad independientes** (se suele normalizar fijando $h_{33} = 1$).

Podemos ver en la Figura 7.1 ejemplos de algunas de estas transformaciones.

7.2. El Rol Crítico de la Homografía

La homografía es la herramienta matemática fundamental para dos aplicaciones estrella en computer vision:

1. **Mapeo de Puntos Planares (Planar Mapping):** si los puntos del mundo real yacen sobre una superficie completamente plana (como una pared, el suelo o la portada de un libro), la relación geométrica exacta entre sus posiciones en dos fotos tomadas desde cualquier ángulo de cámara se describe perfectamente mediante una matriz de homografía H, por lo que se usa la homografía para

enderezar y alinear las imágenes.

Aplicaciones como CamScanner (cuando le haces una foto de lado a un folio o a una pizarra y la app la endereza mágicamente como un documento PDF perfecto) usan exactamente este principio de Planar Mapping.

2. **Coser Panorámicas (Panorama Stitching):** Cuando creamos una foto panorámica rotando la cámara sobre su propio centro óptico, las imágenes capturadas están relacionadas de forma estricta por una homografía, lo que permite proyectar y "alinear" todas las tomas sobre un lienzo virtual común.

7.3. Pipeline Completo de Alineamiento de Imágenes

Para fusionar dos imágenes solapadas y alinearlas de forma automática, el ordenador ejecuta secuencialmente los siguientes pasos:

1. **Detectar Características (Detect Features):** Extraer los puntos clave estables en ambas imágenes independientes utilizando algoritmos como Harris o SIFT, obteniendo las coordenadas (x, y, σ).

2. **Emparejar Características (Match Features):** Calcular los descriptores vectoriales de apariencia y buscar las correspondencias más sólidas mediante el vecino más cercano (Nearest Neighbor) y filtros de seguridad como el Ratio Test de Lowe.

3. **Estimar la Homografía (Estimate Homography):** Filtrar los emparejamientos erróneos (*outliers*) e identificar el modelo matemático mayoritario mediante el algoritmo iterativo **RANSAC**. Como una homografía tiene 8 DoF y cada punto da 2 ecuaciones, RANSAC toma muestras mínimas de $s = 4$ **parejas de puntos** para calcular matrices candidatas H. Al final del consenso, refina la mejor matriz usando todos los *inliers* por mínimos cuadrados.

4. **Deformar la Imagen (Warp Image):** Aplicar la matriz de homografía calculada sobre los píxeles de la imagen de origen para proyectarla geométricamente sobre el espacio de coordenadas de la imagen de destino.

7.3.1. Mecanismos de Warping: Forward vs. Inverse Mapping

Al ejecutar el último paso de deformación (*warping*), nos enfrentamos a un problema práctico de discretización en la red de píxeles:

- **Forward Mapping:** Toma cada píxel (x, y) de la imagen original y calcula su nueva posición destino $(x', y') = H(x, y)$. *Defecto:* Como las nuevas coordenadas dan números decimales flotantes, al redondearlos a enteros se generan

  - **Huecos vacíos:** habrá píxeles en la imagen de destino donde ningún píxel de origen cayó al redondear. Esos píxeles se quedarán negros, creando una desagradable "red de puntos negros" o agujeros en tu imagen deformada.
  - **Solapamientos**: dos píxeles de origen distintos podrían terminar redondeando a la misma coordenada de destino, peleándose por ver quién pinta encima.

- **Inverse Mapping (Mapeo Inverso):** Soluciona el problema recorriendo cada píxel (x', y') de la pantalla de destino vacía. Aplica la homografía inversa para calcular de qué píxel exacto de la imagen original debió venir: $(x, y) = H^{-1}(x', y')$. Luego, lee el color en esa coordenada de origen.

  Dado que $H^{-1}(x', y')$ devolverá valores decimales flotantes (por ejemplo, venir del píxel origen $x = 45{,}3, y = 20{,}1$), se aplican técnicas de **interpolación** para estimar el color real combinando los píxeles vecinos:

  - **Nearest Neighbor Interpolation:** toma el color del píxel entero más cercano. Es rápido pero genera bordes pixelados ("serruchado").
  - **Bilinear Interpolation:** realiza un promedio ponderado lineal combinando los 4 píxeles que rodean la coordenada decimal. Genera transiciones suaves y texturas continuas de alta calidad.

7.4. Pipeline de Transformación de Coordenadas de Cámara

Para entender de dónde surgen estas transformaciones espaciales en las fotos, debemos trazar el camino físico que recorre un punto del universo tridimensional hasta plasmarse en la pantalla digital del ordenador:

$$\mathbf{P_{world}} \xrightarrow{[R|t]} \mathbf{P_{camera}} \xrightarrow{K} \mathbf{p_{image}}$$

1. **Punto en el Mundo ($\mathbf{P_{world}}$):** coordenadas 3D reales del objeto medidas respecto a un origen global arbitrario del universo (X_w, Y_w, Z_w).

2. **Coordenadas de Cámara ($\mathbf{P_{camera}}$):** se transforma la posición del punto del mundo al sistema de coordenadas local del ojo de la cámara aplicando una rotación R y una traslación t (parámetros extrínsecos de la cámara). El punto pasa a medirse con respecto al centro óptico de la lente.

3. **Coordenadas de Imagen ($\mathbf{p_{image}}$):** el punto 3D de la cámara se proyecta sobre el plano del sensor fotosensible mediante la matriz de calibración intrínseca K, la cual tiene en cuenta la distancia focal (f) y el centro del plano del sensor (c_x, c_y). El resultado es la coordenada discreta de píxeles (x, y) que procesamos en nuestro código.

Todos estos conceptos de distancia focal, centro del plano del sensor y parámetros extrísicos e intrínsicos de la cámara se estudian en profundidad en el tema de calibración de cámaras (Tema 14), pero es importante entender que las transformaciones

2D que aplicamos a las imágenes son una consecuencia directa de cómo la geometría del mundo real se proyecta a través de la lente y se captura en el sensor digital.

8. Recognition

El reconocimiento en visión por computador consiste en dotar a un sistema informático de la capacidad de interpretar el contenido semántico de una imagen digital. A diferencia de las etapas de bajo nivel (como el filtrado de imágenes o la detección de bordes), el reconocimiento opera en el nivel más alto de la jerarquía de abstracción visual.

Históricamente, los primeros intentos en las décadas pasadas se basaban en la ingeniería de características manuales (*hand-crafted features*) tratando de buscar patrones geométricos rígidos, bordes o esquinas. Sin embargo, este enfoque fracasó ante la inmensa variabilidad del mundo real. El paradigma moderno se fundamenta en el **Aprendizaje Supervisado (Supervised Learning)**, donde el sistema aprende las reglas de reconocimiento de forma automática exponiéndose a grandes volúmenes de datos etiquetados.

8.0.1. Taxonomía de las Tareas de Reconocimiento Visual

Dependiendo del nivel de granularidad espacial y del objetivo semántico, el reconocimiento se divide en cuatro tareas principales:

1. **Clasificación de Imágenes (Image Classification):** El objetivo es asignar una única etiqueta categórica global a toda la imagen. No discrimina la ubicación ni la cantidad de objetos presentes.

2. **Detección de Objetos (Object Detection):** Consiste en localizar de forma espacial la presencia de uno o múltiples objetos dentro de la imagen, dibujando una caja delimitadora (*Bounding Box*) alrededor de cada uno y asignándoles una clase.

3. **Segmentación Semántica (Semantic Segmentation):** Implica clasificar cada uno de los píxeles de la imagen en una categoría semántica específica (ej. píxel de carretera, píxel de cielo, píxel de coche). Agrupa las instancias de una misma categoría bajo una única máscara común.

4. **Segmentación de Instancias (Instance Segmentation):** Es una combinación híbrida de alta precisión. No solo clasifica los píxeles semánticamente, sino que discrimina de forma exacta e individualizada cada instancia u objeto independiente (ej. diferencia el coche A del coche B aunque estén adyacentes).

8.1. Classification Concepts

La clasificación de imágenes actúa como el pilar fundamental sobre el cual se construyen los algoritmos de reconocimiento más complejos.

8.1.1. Data-Driven Approach

Dado que no existe un algoritmo matemático explícito para definir visualmente qué es un objeto (debido a las variaciones de iluminación, deformaciones, oclusión y puntos de vista), el diseño se basa en tres etapas:

$$\text{Dataset Labeling} \longrightarrow \text{Training a Classifier} \longrightarrow \text{Evaluation on Test Set}$$

Un clasificador se formaliza matemáticamente mediante una función de predicción $f(\mathbf{x}, \mathbf{W})$ que toma un vector de características de imagen $\mathbf{x}$ y un conjunto de parámetros o pesos entrenables $\mathbf{W}$, y devuelve una puntuación para cada clase.

8.1.2. Linear Classifier

Es el modelo fundamental base. Asumiendo una imagen de entrada aplanada en un vector columna $\mathbf{x} \in \mathbb{R}^{D\times 1}$ (donde D es el número total de píxeles) y un problema con K clases posibles, el clasificador lineal se define analíticamente como:

$$\mathbf{f}(\mathbf{x}, \mathbf{W}) = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{8.1}$$

donde:

- $\mathbf{W} \in \mathbb{R}^{K\times D}$ es la matriz de pesos (*weights*). Cada fila de esta matriz actúa matemáticamente como una plantilla o *template* para una clase específica.
- $\mathbf{b} \in \mathbb{R}^{K\times 1}$ es el vector de sesgos (*bias*), que parametriza las preferencias independientes de la escala de los datos para ciertas clases.

Interpretación Geométrica del Clasificador Lineal

Desde una perspectiva geométrica, el clasificador lineal establece hiperplanos de decisión de alta dimensionalidad en el espacio de características. Cada clase está delimitada por una frontera lineal. Si los datos correspondientes a una clase no son linealmente separables (debido a distribuciones complejas o texturas entrelazadas), el clasificador lineal sufrirá una degradación severa en su rendimiento, lo que justifica la necesidad histórica de transicionar hacia modelos no lineales profundos (como las Redes Neuronales Convocucionales).

8.1.3. Loss Functions y Optimización

Para entrenar los pesos $\mathbf{W}$, se necesita cuantificar el error del modelo mediante una función de pérdida:

- **Pérdida SVM Multiclase (Hinge Loss):** Evalúa que la puntuación de la clase correcta s_{y_i} supere a las puntuaciones de las clases incorrectas s_j por un margen de seguridad fijo Δ:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta) \tag{8.2}$$

- **Clasificador Softmax (Pérdida de Entropía Cruzada):** Convierte las puntuaciones brutas (*logits*) en una distribución de probabilidad normalizada mediante la función Softmax:

$$P(Y = k \mid X = \mathbf{x}_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \tag{8.3}$$

  La pérdida asociada para la muestra busca maximizar el logaritmo de la probabilidad de la clase correcta:

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right) \tag{8.4}$$

El ajuste óptimo de los parámetros se realiza mediante la minimización de la pérdida total balanceada con un término de Regularización ($R(\mathbf{W})$) para evitar el sobreajuste (*overfitting*):

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \lambda R(\mathbf{W}) \tag{8.5}$$

La optimización se ejecuta de forma iterativa calculando el gradiente analítico mediante el algoritmo de **Descenso de Gradiente Estocástico (SGD)**: $\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} L$.

8.2. Segmentation Idea

La segmentación avanza un paso más allá de la clasificación global al intentar comprender la escena a nivel de píxel individual. Conceptualmente, busca responder a la pregunta: *¿A qué categoría pertenece geométrica y semánticamente este píxel exacto de la coordenada* (u, v)*?*

8.2.1. Segmentación Semántica Tradicional frente a Profunda

- **Aproximaciones Primitivas:** Se basaban fuertemente en algoritmos de clustering de color (como K-Means en el espacio de color CIE Lab) o en cortes de grafos (*Graph Cuts*) buscando discontinuidades abruptas en los gradientes de brillo de la imagen. No tenían noción semántica real; solo agrupaban regiones homogéneas de color o textura.
- **Aproximaciones Modernas (Fully Convolutional Networks - FCN):** Diseñadas por Long et al., reemplazan las capas densas terminales de un clasificador por capas puramente convolucionales. Esto permite que la red reciba una imagen de cualquier resolución y devuelva un mapa de puntuaciones con correspondencia espacial directa píxel a píxel.

8.2.2. El Mecanismo de Downsampling y Upsampling

Para procesar la información de segmentación de forma eficiente, las arquitecturas estándar emplean un diseño simétrico:

1. **Ruta de Contracción (Encoder / Downsampling):** Utiliza circunvoluciones y capas de Pooling para reducir progresivamente el tamaño espacial de la imagen mientras incrementa la profundidad de los canales de características. Esto permite capturar el contexto semántico global (*"qué"* hay en la escena), sacrificando la resolución espacial exacta (*"dónde"* está).

2. **Ruta de Expansión (Decoder / Upsampling):** Utiliza operaciones de convolución transpuesta (*Transposed Convolution*) o interpolaciones bilineales para proyectar los mapas de características abstractos de baja resolución de vuelta a las dimensiones espaciales originales de la imagen de entrada. Esto recupera la localización geométrica precisa de los bordes de los objetos.

9. Optical flow

El **Flujo Óptico** es el patrón del movimiento aparente de los objetos, superficies y bordes en una escena visual, causado por el movimiento relativo entre el observador (la cámara) y la escena misma, es decir, se busca rastrear a dónde se ha ido cada píxel en el siguiente fotograma.

Observación. Es importante destacar la parte de "movimiento **aparente**" en la definición de flujo óptico. El flujo óptico no es una representación directa del movimiento físico real en el espacio 3D, sino una estimación basada únicamente en las variaciones de brillo captadas por la cámara, lo que puede hacer que haya situaciones en las que el flujo óptico no refleje fielmente la dinámica real de la escena (por ejemplo, en presencia de sombras, cambios de iluminación o texturas homogéneas).

Mientras que el *Motion Field* (Campo de Movimiento) representa la proyección geométrica 3D real de las velocidades de los puntos del espacio sobre el plano de la imagen, el Flujo Óptico es una aproximación basada puramente en las variaciones espacio-temporales del brillo captadas por el sensor. El objetivo del flujo óptico es estimar un vector de desplazamiento bidimensional $\mathbf{v} = [u, v]^T$ para cada píxel entre fotogramas consecutivos en el tiempo t y $t + 1$.

Lo que vamos a obtener con todo este despliegue matemático es pasar de ver imágenes estáticas inconexas a entender la dinámica cinemática del espacio 3D a través de los cambios de luz en 2D. Estamos convirtiendo un sensor pasivo (una cámara que solo captura color) en un sensor activo capaz de medir velocidades, predecir impactos y separar objetos en movimiento.

9.1. Brightness constancy

Para poder rastrear un píxel de un fotograma a otro, el algoritmo requiere imponer restricciones físicas en el mundo visual. La hipótesis central y más importante es la **Asunción de Constancia de Brillo (Brightness Constancy Assumption)**:

El brillo de un punto en movimiento permanece constante a lo largo del tiempo.

Matemáticamente, si un píxel en la coordenada espacial (x, y) en el instante de tiempo t se desplaza una distancia $(\Delta x, \Delta y)$ en el siguiente instante $t + \Delta t$, su intensidad de iluminación I debe cumplir la igualdad:

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t)$$

9.2. Optical flow equation

A partir de la asunción de constancia de brillo, si asumimos que el movimiento $(\Delta x, \Delta y, \Delta t)$ es infinitesimalmente **pequeño**, podemos aplicar una aproximación por **Serie de Taylor de primer orden** al lado derecho de la ecuación:

$$I(x + \Delta x, y + \Delta y, t + \Delta t) \approx I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t$$

Sustituyendo esta aproximación en la igualdad original de constancia de brillo:

$$I(x, y, t) = I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t$$

Cancelando el término $I(x, y, t)$ en ambos lados obtendremos:

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0$$

Si dividimos toda la expresión entre el diferencial de tiempo Δt para obtener tasas de cambio continuas:

$$\frac{\partial I}{\partial x}\frac{\Delta x}{\Delta t} + \frac{\partial I}{\partial y}\frac{\Delta y}{\Delta t} + \frac{\partial I}{\partial t} = 0$$

Definiendo los componentes de velocidad del flujo como $u = \frac{\Delta x}{\Delta t}$ (velocidad horizontal) y $v = \frac{\Delta y}{\Delta t}$ (velocidad vertical), y reescribiendo las derivadas parciales como subíndices (I_x, I_y, I_t), llegamos a la **Ecuación del Flujo Óptico (Optical Flow Constraint Equation)**:

$$\mathbf{I_x u + I_y v + I_t = 0} \quad \Longleftrightarrow \quad \nabla I \cdot \mathbf{v} + I_t = 0$$

donde $\nabla I = [I_x, I_y]^T$ representa el gradiente espacial de la imagen y $\mathbf{v} = [u, v]^T$ representa el vector de velocidad buscado.

9.3. Aperture Problem

La ecuación fundamental del flujo óptico presenta una limitación matemática crítica conocida como el **Problema de la Apertura**:

- Para cada píxel individual disponemos de una sola ecuación

  $$I_x u + I_y v + I_t = 0$$

  pero tenemos dos incógnitas por resolver (u y v). El sistema está subdeterminado (infinitas soluciones válidas), lo que significa que el conjunto de soluciones es una línea recta en el espacio de velocidades (u, v), y no un punto único.

- **Implicación Física:** Cuando observamos una estructura a través de una apertura local pequeña, **el movimiento a lo largo de la dirección de un borde recto no se puede estimar de forma fiable**. Solo somos capaces de medir la componente de la velocidad perpendicular al borde (normal al gradiente ∇I). El movimiento paralelo al borde es completamente invisible para las derivadas locales de brillo.

Ejemplo. Imagina que tienes una pared blanca y una línea negra diagonal (en este sentido "/") pintada en ella. Ahora, coges un trozo de cartulina negra, le haces un agujero pequeño y circular en el centro (una apertura), y lo pones sobre la pantalla de modo que solo puedas ver un trozo de la línea diagonal a través del agujero.

Si yo muevo la línea real físicamente hacia la derecha, ¿qué verás tú a través del agujero? Verás que la línea se mueve. Pero, ¿qué pasa si muevo la línea diagonal hacia abajo? ¡Verás exactamente el mismo movimiento! De hecho, si muevo la línea en una combinación diagonal de derecha y abajo, la percepción visual dentro de tu pequeño agujero será idéntica.

¿Por qué ocurre esto? Porque cuando miras un borde liso a través de una apertura pequeña, cualquier movimiento que ocurra de forma paralela a la dirección del propio borde es completamente invisible. Los píxeles que entran por un lado del borde tienen exactamente el mismo color y brillo que los que salen por el otro. Solo eres capaz de percibir el movimiento que ocurre en dirección perpendicular (normal) al borde de la línea diagonal.

9.4. La Solución de Lucas-Kanade

Como bien aprendimos en el paso anterior, si analizas un píxel individual flotando en mitad de un borde recto, estás ciego ante el movimiento paralelo: tienes una sola ecuación para dos incógnitas (u y v).

La **genialidad** de Bruce Lucas y Takeo Kanade en 1981, y en la que se basa el **algoritmo de Lucas-Kanade** fue decir: «Es verdad que un píxel por sí solo no tiene información suficiente. Pero los píxeles en el mundo real no se mueven de forma caótica e independiente; pertenecen a objetos sólidos».

Viendo esto, las tres suposiciones clave que hicieron fueron:

1. **Brightness Constancy:** el brillo de un punto se mantiene constante a lo largo del tiempo.
2. **Small motion:** el desplazamiento entre fotogramas es lo suficientemente pequeño como para justificar la aproximación de Taylor de primer orden.
3. **Spatial coherence:** los píxeles vecinos dentro de una pequeña ventana cuadrada se mueven de forma similar, compartiendo el mismo vector de velocidad, es decir, **asume que el movimiento es pequeño y prácticamente idéntico para todos los píxeles vecinos dentro de una pequeña ventana cuadrada Ω (por ejemplo, de tamaño 5×5 o 3×3) centrada en el píxel de interés.**

donde (1) y (2) ya los habíamos visto antes, pero la verdadera innovación de Lucas-Kanade fue introducir la tercera suposición de coherencia espacial local para resolver el problema de la apertura.

Veremos ahora el desarrollo matemático que concluye de haber asumido estas tres premisas. Si una ventana de trabajo contiene n píxeles (ej. para una ventana de 5×5, $n = 25$), y todos comparten la misma velocidad (u, v), podemos plantear un sistema de n ecuaciones con 2 incógnitas:

$$\begin{aligned} I_{x1}u + I_{y1}v &= -I_{t1} \\ I_{x2}u + I_{y2}v &= -I_{t2} \\ &\vdots \\ I_{xn}u + I_{yn}v &= -I_{tn} \end{aligned}$$

Este sistema se puede unificar de forma matricial compacta como $A\mathbf{v} = b$:

$$\begin{bmatrix} I_{x1} & I_{y1} \\ I_{x2} & I_{y2} \\ \vdots & \vdots \\ I_{xn} & I_{yn} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_{t1} \\ I_{t2} \\ \vdots \\ I_{tn} \end{bmatrix}$$

Tenemos que el sistema está sobredeterminado ($n > 2$) con 25 líneas rectas en el espacio de velocidades y, debido al ruido de la cámara y a las imperfecciones del mundo real, no se van a cortar exactamente todas en el mismo punto infinitesimal. No existe una solución perfecta que cumpla las 25 ecuaciones a la vez. Por lo tanto, aplicamos el método de **Mínimos Cuadrados** multiplicando por la transpuesta A^T en ambos lados ($A^T A\mathbf{v} = A^T b$):

$$\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

La matriz cuadrada de 2×2 resultante ($A^T A$) es exactamente la **matriz de estructura o tensor de estructura**, idéntica a la utilizada en el detector de esquinas de Harris. Para poder despejar de forma única el vector de velocidad $\mathbf{v} = (A^T A)^{-1} A^T b$, la matriz debe ser invertible, lo que exige que ambos autovalores sean grandes ($\lambda_1, \lambda_2 \gg 0$). Esto significa físicamente que Lucas-Kanade solo funciona con precisión si la ventana se posiciona sobre **esquinas o regiones con texturas ricas en dos direcciones**.

9.5. Limitaciones de Lucas-Kanade y la Solución por Pirámides

El algoritmo estándar de Lucas-Kanade falla de forma inmediata si se rompen sus hipótesis de partida. Su debilidad principal es que **asume movimientos muy pequeños** (debido al truncamiento de la serie de Taylor de primer orden). Si un objeto se desplaza a gran velocidad, saltándose el vecindario inmediato de los píxeles adyacentes, las derivadas locales dejan de tener sentido físico y el método diverge.

9.5.1. ¿Por qué ayudan las Pirámides Gaussianas? (Coarse-to-Fine)

Para poder gestionar **grandes desplazamientos (large displacements)** y garantizar la convergencia matemática, el flujo óptico se procesa de forma jerárquica me-

diante una estructura de **Pirámide Gaussiana (Coarse-to-Fine computation)**:

1. **Construcción:** se reduce el tamaño de la imagen original sucesivamente dividiendo su resolución a la mitad en cada nivel superior de la pirámide.

2. **Procesamiento en Escala Macro (Cúspide de la pirámide):** en los niveles más altos y pequeños, un desplazamiento que originalmente medía 30 píxeles de distancia en la imagen original pasa a medir, por ejemplo, menos de 1 píxel. Al volverse un movimiento microscópico, la asunción de Lucas-Kanade se cumple a la perfección y el algoritmo estima un flujo óptico inicial aproximado
$$\mathbf{v}_{\text{cúspide}} = [u_{\text{cúspide}}, v_{\text{cúspide}}]$$

3. **Propagación y Refinamiento (Warping):** ese flujo estimado en la escala superior
$$\mathbf{v}_{\text{cúspide}} = [u_{\text{cúspide}}, v_{\text{cúspide}}]$$
se proyecta al nivel inferior duplicando su magnitud
$$\mathbf{v}_{\text{cúspide}} = 2 \cdot \mathbf{v}_{\text{cúspide}}$$
y se hereda como una estimación inicial del movimiento a ese nivel intermedio. Se utiliza este valor para deformar geométricamente (*warp*) la imagen intermedia, acercándola ópticamente a la posición esperada, como sigue

   - Tomas la Foto 1 intermedia (I_{inter}) y, usando ese movimiento de $\mathbf{v}_{\text{cúspide}}$ píxeles que acabas de heredar de la cúspide, la deformas (la empujas $u_{\text{cúspide}}$ píxeles hacia la derecha y $v_{\text{cúspide}}$ píxeles hacia abajo).
   - Creas una imagen nueva artificial: la Foto 1 Deformada (I_{warp1}).

   A continuación, Lucas-Kanade calcula únicamente el "residuo de corrección" del movimiento restante de grano fino, es decir, comparamos la Foto 2 intermedia (I_{inter2}) con la Foto 1 Deformada (I_{warp1}), y como el desplazamiento del objeto va a ser mínimo (ya lo hemos acercado con la deformación), entonces Lucas-Kanade se puede aplicar (cumple los requisitos) y calcula con éxito ese residuo milimétrico de movimiento que falta para alcanzar la posición exacta.

   Tenemos por tanto que el movimiento real exacto para este nivel intermedio es
$$\mathbf{v}_{\text{inter}} = [u_{\text{inter}}, v_{\text{inter}}] \quad \text{donde} \begin{cases} u_{\text{inter}} & = u_{\text{cúspide}} + u_{\text{residuo}} \\ v_{\text{inter}} & = v_{\text{cúspide}} + v_{\text{residuo}} \end{cases}$$
   y este proceso se realiza iterativamente hasta llegar a la escala original, refinando el movimiento a cada nivel de la pirámide.

4. **Resultado:** este enfoque iterativo mejora drásticamente la convergencia del algoritmo, permitiendo capturar dinámicas de movimiento de objetos veloces manteniendo la precisión matemática local del gradiente.

10. Monocular Depth Estimation and Efficiency

No relevante.

La pérdida de la tercera dimensión es la primera consecuencia directa e irreversible del proceso de captura de una cámara fotográfica. Al proyectar una escena tridimensional 3D sobre un plano bidimensional 2D (el sensor), la información de la profundidad (Z) desaparece.

Para poder estimar la distancia a partir de imágenes, la visión artificial y los sistemas biológicos se apoyan en diferentes pistas visuales o estímulos conocidos como **Depth Cues**:

- **Pistas Binoculares (Estéreo):** Dependen de la disparidad y la triangulación entre dos puntos de vista ligeramente desplazados (visión estéreo, como vimos en temas anteriores).
- **Pistas Monoculares:** Estímulos visuales contenidos dentro de una única imagen plana que permiten inferir relaciones espaciales relativas. Se dividen en:
  - *Perspectiva lineal:* Convergencia de líneas paralelas en la distancia hacia un punto de fuga.
  - *Tamaño relativo / familiar:* Si conocemos el tamaño real de un objeto (ej. un coche o un peatón), su escala en la imagen nos indica si está cerca o lejos.
  - *Oclusión:* Un objeto que tapa o cubre a otro está, inequívocamente, más cerca del observador.
  - *Gradiente de textura:* Los detalles se vuelven más densos y difusos a medida que se alejan en el fondo.
  - *Perspectiva atmosférica:* Los objetos lejanos pierden contraste y adquieren un tono azulado o neblinoso debido a la dispersión de la luz en el aire.

10.1. Monocular Depth Estimation

La estimación monocular de profundidad (*Monocular Depth Estimation - MDE*) consiste en predecir un mapa de distancias continuas utilizando como única entrada una sola imagen estática RGB.

¿Por qué MDE es un problema matemáticamente mal propuesto (ill-posed)?

Bajo el modelo de proyección de una cámara pinhole, un punto del espacio tridimensional $\mathbf{P} = [X, Y, Z]^\top$ se mapea en un píxel $\mathbf{x} = [u, v]^\top$ mediante las ecuaciones de proyección:

$$u = f_x \frac{X}{Z} + c_x, \quad v = f_y \frac{Y}{Z} + c_y \tag{10.1}$$

Si disponemos únicamente de la imagen 2D, conocemos los píxeles (u, v) y los parámetros de calibración interna de la cámara, pero las coordenadas reales (X, Y, Z) en metros son incógnitas. Este sistema matemático posee **infinitas soluciones correctas**. Un punto pequeño situado muy cerca de la lente proyectará exactamente el mismo píxel en la pantalla que un objeto idéntico pero gigante situado en el fondo de la escena (un fenómeno conocido en el arte como *ilusión de perspectiva de Ames*).

Al existir infinitas escenas tridimensionales distintas que generan la misma imagen exacta de dos dimensiones, el problema es matemáticamente imposible de resolver de forma determinista mediante geometría pura. Por este motivo, el software moderno recurre al **Deep Learning**, utilizando redes neuronales para que aprendan el *prior* de la escena (es decir, que memoricen el contexto semántico y las pistas monoculares del entorno) para deducir cuál de las infinitas soluciones es la correcta.

10.2. Paradigmas de Entrenamiento de Redes Neuronales para MDE

Para resolver este problema ambiguo, una Red Neuronal Convolucional (CNN) o un Vision Transformer (ViT) toma una imagen RGB y produce una matriz del mismo tamaño denominada mapa de profundidad predicho $\hat{D}$. Existen dos formas principales de entrenar este modelo:

10.2.1. Supervised Learning con LiDAR

Es el enfoque más directo y tradicional. Consiste en alimentar la red neuronal con una imagen y contrastar su predicción píxel a píxel contra una medición real, física y exacta del terreno (*Ground Truth*).

1. **Recolección de Datos:** Se utilizan vehículos instrumentados equipados con sensores **LiDAR** (dispositivos láser de barrido activo). Mientras la cámara toma la fotografía RGB, el LiDAR dispara haces de luz y mide el tiempo de retorno para esculpir una nube de puntos 3D milimétrica de la escena.

2. **Proyección de Nube a Matriz:** Los puntos 3D del LiDAR se proyectan matemáticamente sobre el plano de la cámara usando matrices de transformación, generando una matriz dispersa de profundidad real (D).

3. **Cálculo de la Función de Pérdida:** La red neuronal realiza una predicción $\hat{D}$. El error se calcula penalizando las desviaciones numéricas frente al LiDAR

mediante funciones de pérdida regresiva, tales como el Error Cuadrático Medio (L_2) o el Error Absoluto Medio (L_1):

$$\mathcal{L}_{\text{supervisada}} = \frac{1}{N}\sum_{i=1}^{N} \|\hat{D}(s_i) - D(s_i)\|^2 \tag{10.2}$$

Desventaja Principal: Conseguir datos de LiDAR es extremadamente costoso, los sensores son caros, requieren una calibración de hardware muy compleja y el mapa resultante es *disperso* (tiene huecos vacíos donde el láser no rebotó).

10.2.2. Self-Supervised Learning

Este enfoque elimina la necesidad de contar con sensores LiDAR caros. El modelo aprende a estimar la profundidad analizando únicamente **secuencias de vídeo normales**. Su pilar fundamental es el principio de la **Consistencia Fotométrica (Photometric Consistency)**.

Veámos cómo funciona este mecanismo de aprendizaje paso a paso:

1. **Configuración de Entrada:** Tomamos dos fotogramas consecutivos de un vídeo: el fotograma actual (I_t) y el fotograma siguiente (I_{t+1}). Entre ambas capturas, la cámara se ha movido (el coche ha avanzado).

2. **Redes Neuronales en Paralelo:**

   - La red *Depth Network* analiza el fotograma I_t y estima su mapa de profundidad monocular $\hat{D}_t$.
   - Una segunda red auxiliar, llamada *Pose Network*, analiza ambos fotogramas en paralelo (I_t, I_{t+1}) y estima el movimiento rígido relativo en 3D que sufrió la cámara (la rotación R y traslación $\mathbf{t}$).

3. **Deformación Geométrica (Inverse Warping):** Usando la profundidad estimada $\hat{D}_t$ y el movimiento relativo $[R \mid \mathbf{t}]$, la geometría proyectiva nos permite tomar los píxeles del fotograma siguiente (I_{t+1}) y "re-proyectarlos" hacia atrás en el tiempo para reconstruir sintéticamente cómo debería verse el fotograma actual. A esta imagen reconstruida la llamamos **Imagen Deformada** ($\hat{I}_t$).

4. **La Pérdida Fotométrica:** Si la profundidad predicha $\hat{D}_t$ y la pose estimada son correctas, la imagen sintética inventada $\hat{I}_t$ debería ser visualmente idéntica en colores y texturas a la imagen real capturada I_t. La función de pérdida evalúa el error de brillo píxel a píxel:

$$\mathcal{L}_{\text{fotométrica}} = \|I_t - \hat{I}_t\| \tag{10.3}$$

El sistema se optimiza minimizando este error de reconstrucción de imágenes. La red se entrena de forma completamente gratuita (*self-supervised*) porque la propia consistencia de la secuencia de vídeo actúa como el supervisor supervisor.

10.3. Efficiency y Trade-offs en Hardware Limitado

Una vez que el modelo es capaz de estimar la profundidad, se enfrenta al desafío del despliegue en entornos reales (como los chips embebidos de drones, robots autónomos o smartphones), donde los recursos energéticos y de memoria son severamente escasos.

Analicemos el **Trade-off entre Tiempo de Inferencia y Rendimiento** (Métricas de error frente a velocidad):

- **Métricas de Rendimiento:** Se miden usando el error RMSE (Root Mean Squared Error) o el error relativo absoluto (*AbsRel*). Cuanto más bajo sea el valor, más precisa es la estimación del relieve.
- **Métricas de Eficiencia de Hardware:**
  - *Número de Parámetros:* El tamaño que ocupa el modelo en la memoria RAM/VRAM.
  - *FLOPs (Floating Point Operations):* La cantidad de operaciones de coma flotante que necesita ejecutar el procesador por segundo para dar una respuesta.
  - *Latencia / Tiempo de Inferencia:* Los milisegundos que tarda la red en procesar un fotograma.

Conclusión de Optimización (Métrica EER): Para equilibrar estas métricas y poder comparar modelos de forma objetiva, usaremos la métrica **EER (Efficient Evaluation Ratio)**, que promedia el coste relativo de los parámetros y las operaciones aritméticas frente a un modelo de referencia:

$$EER = \frac{1}{\|i\|} \cdot \sum_{i} \left(\frac{M_i}{R_i} \right) \tag{10.4}$$

El diseño de ingeniería moderno busca reducir drásticamente los FLOPs (mediante convoluciones separables en profundidad o mecanismos de atención eficientes en Transformers) para lograr que el error RMSE se mantenga bajo pero permitiendo que el tiempo de ejecución caiga de segundos a milisegundos, garantizando que el sistema opere a tiempo real crítico en sistemas embebidos de automoción.

11. Transformers in Computer Vision

No relevante.

Las **redes recurrentes** (RNN) están diseñadas para procesar datos secuenciales (como texto o series temporales) donde el orden de los elementos importa. Formalmente se definen como

- **Generación de estados ocultos** (h_t): los modelos recurrentes generan de forma iterativa una secuencia de estados ocultos o vectores de memoria, representados como h_t
- **Función de transición**: cada estado oculto en el instante de tiempo actual t se calcula como una función matemática dependiente de dos variables

$$h_t = f(h_{t-1}, x_t)$$

  donde h_{t-1} es el estado oculto del paso inmediatamente anterior (que idealmente condensa la información del pasado), y x_t es el token o vector de entrada en la posición actual t (por ejemplo, la palabra actual en una frase).

Los problemas que tienen las RNN son:

- **Pérdida de memoria a largo plazo**: a medida que la secuencia se hace más larga, las RNN tienen dificultades para mantener información relevante de los primeros elementos, lo que se conoce como el problema del **desvanecimiento del gradiente** (Vanishing Gradient). Esto significa que la influencia de los primeros tokens en la secuencia disminuye exponencialmente a medida que avanzamos en la secuencia, lo que dificulta el aprendizaje de dependencias a largo plazo.
- **Dificultad para paralelizar**: las RNN procesan los datos de forma secuencial, lo que significa que cada paso depende del resultado del paso anterior, es decir, el estado oculto h_t depende de h_{t-1}.

La solución a los problemas es añadir atención, de estas dos maneras

- Reemplazar la recurrencia con **auto-atención** (self-attention), donde cada token puede atender a todos los demás tokens de la secuencia, lo que permite capturar dependencias a largo plazo sin importar la distancia entre los tokens.

- Eliminar las conexiones recurrentes, lo que permite un procesamiento completamente paralelo de la secuencia, mejorando significativamente la eficiencia computacional.

En esta sección nos basaremos en los **transformers**, que son un tipo de modelo de atención que ha revolucionado el campo del procesamiento del lenguaje natural (NLP) y se ha extendido a otras áreas como la visión por computadora. El transformer es una arquitectura de red sencilla basada en el mecanismo de atención que aumenta la velocidad con la que estos modelos pueden entrenar.

11.1. Self Attention

El mecanismo de **self-attention** (auto-atención) es una técnica que permite a un modelo de aprendizaje automático asignar diferentes pesos a diferentes partes de la entrada al procesar una secuencia. En el contexto de los transformers, la auto-atención se utiliza para capturar las relaciones entre los tokens de una secuencia, independientemente de su posición relativa. Esto es especialmente útil para tareas como la traducción automática, el análisis de sentimientos y la generación de texto, donde la comprensión de las relaciones entre palabras es crucial.

Para entender la matemática, se utiliza la analogía de una búsqueda en una base de datos. Cada elemento (token o patch de imagen) se proyecta dinámicamente en tres vectores distintos mediante matrices de pesos entrenables

- **Query** (Q): representa lo que el elemento actual está "buscando" en los demás.
- **Key** (K): representa el "índice" o descripción de lo que ese elemento ofrece.
- **Value** (V): representa la información real que contiene el elemento y que se transmitirá si hay coincidencia.

estos vectores se calculan mediante multiplicaciones matriciales con matrices de pesos entrenables W_Q, W_K y W_V respectivamente. En la arquitectura del Transformer, antes de calcular la atención, la secuencia de entrada pasa por tres capas lineales independientes (capas de proyección o capas Dense/Linear sin función de activación). Cada una de estas tres capas lineales posee su propia matriz de pesos asociados. Esas tres matrices de pesos son, precisamente, W_Q, W_K y W_V.

Antes de que el modelo vea una sola imagen o palabra (al inicio del entrenamiento), las matrices no saben nada. Sus valores se generan mediante métodos de inicialización aleatoria (como la inicialización de Xavier/Glorot o He/Kaiming, que son estándares en Deep Learning para mantener la varianza de las activaciones controlada).

Veámos los pasos para calcular la atención

1. Dado un token de entrada x_i, se proyecta a tres vectores:

$$q_i = W_Q x_i,\ k_i = W_K x_i \ \text{ y } \ v_i = W_V x_i$$

2. Se calcula la puntuación de atención entre el token i y cada token j de la secuencia utilizando el producto escalar entre sus vectores de consulta (q_i) y clave (k_j):

$$s_{ij} = q_i^T k_j$$

3. Escalar la puntuación de atención por $\sqrt{d_k}$ para estabilizar los gradientes durante el entrenamiento:

$$\hat{s}_{ij} = s_{ij}/\sqrt{d_k}$$

donde d_k es la dimensión de los vectores de clave.

4. Aplicar la función softmax sobre las puntuaciones escaladas para obtener los pesos de atención:

$$a_{ij} = \text{softmax}_j(\hat{s}_{ij})$$

lo que normaliza los pesos para que sumen 1.

5. Agregar los valores ponderados para obtener el vector de contexto para el token i

$$z_i = \sum_j a_{ij} v_j$$

donde z_i es el vector de contexto que representa la información relevante para el token i basada en su atención a los demás tokens de la secuencia.

Resumiendo los pasos anteriores, la fórmula de la atención se puede expresar de manera compacta en forma matricial como

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

y se llama **Scaled Dot-Product Attention** (Atención por producto escalar escalado). En esta fórmula, Q, K y V son matrices que contienen los vectores de consulta, clave y valor para toda la secuencia, respectivamente. El producto QK^T calcula las puntuaciones de atención para todos los pares de tokens, que luego se escalan y se normalizan con softmax antes de multiplicar por V para obtener los vectores de contexto finales.

Observación. En cuanto a la **complejidad**, el cálculo de la atención tiene una complejidad de $O(n^2 d)$ en tiempo y $O(n^2)$ en memoria, donde n es la longitud de la secuencia (número de tokens o parches) y d es la dimensión de los vectores de consulta, clave y valor. Esto se debe a que se necesita calcular las puntuaciones de atención para cada par de tokens, lo que resulta en una matriz de atención de tamaño $n \times n$. Esta complejidad cuadrática es el principal cuello de botella para secuencias largas, como imágenes de alta resolución, donde el número de tokens puede ser muy grande.

11.1.1. Multi-Head Attention

La intuición de este modelo es resolver una limitación crítica del bloque de auto-atención simple (**Single-Head Attention**). En un mecanismo de atención único, el cálculo suele verse dominado por el propio token (es decir, una palabra o un píxel tiende a prestarse la mayor cantidad de atención a sí mismo) o se ve obligado a enfocarse en un único tipo de relación geométrica o semántica, ignorando otras que ocurren en paralelo.

En lugar de realizar una sola operación de atención gigante, el modelo divide el espacio y ejecuta h cabezas de atención en paralelo. Cada cabeza (i) opera en su propio subespacio de representación porque posee su propio conjunto exclusivo de matrices de pesos entrenables:

$$W_Q^{(i)},\ W_K^{(i)}\ \text{ y }\ W_V^{(i)}$$

El proceso técnico consta de 3 pasos:

1. **Computación en paralelo**: cada una de las h cabezas calcula su propia matriz de atención utilizando la fórmula estándar de Scaled Dot-Product Attention que ya conocemos, generando un output independiente denominado $Z^{(i)}$

   $$Z^{(i)} = \text{Attention}(QW_Q^{(i)}, KW_K^{(i)}, VW_V^{(i)})$$

   Al final de este paso, tenemos h vectores de salida diferentes $(Z^{(1)}, Z^{(2)}, \ldots, Z^{(h)})$ para el mismo token.

2. **Concatenación**: para poder continuar con el flujo de la red, no podemos tener los resultados fragmentados. El modelo une (concatena) espacialmente todos los vectores resultantes uno al lado del otro en una única matriz

   $$\text{Output Concatenado} = [Z^{(1)}, Z^{(2)}, \ldots, Z^{(h)}]$$

3. **Proyección Lineal Final**: la matriz concatenada es muy "ancha" computacionalmente. Para devolver el vector a su dimensión original de diseño (d_{model}) y permitir que la información de todas las cabezas se mezcle e interactúe de forma efectiva, se multiplica por una nueva matriz de proyección final entrenable, llamada W^O

   $$\text{MHA} = [Z^{(1)}, Z^{(2)}, \ldots, Z^{(h)}]W^O$$

Los beneficios de la atención multi-cabeza son que cada cabeza puede enfocarse en diferentes tipos de relaciones (sintácticas, de coreferencia, de co-ocurrencia); efectivamente creando múltiples "subespacios de representación". Esto permite al modelo capturar una variedad más rica de patrones y relaciones en los datos.

Una cosa que no hemos tenido en cuenta en el modelo descrito hasta ahora es una forma de tener en cuenta el orden de las palabras en la secuencia de entrada. Para

abordar esto, el transformer añade un vector a cada embedding de entrada que le ayuda a determinar la posición de cada palabra. Este vector se llama **positional coding** y se suma al embedding de cada token antes de ser procesado por las capas de atención, lo podemos ver en la Figura 11.1.

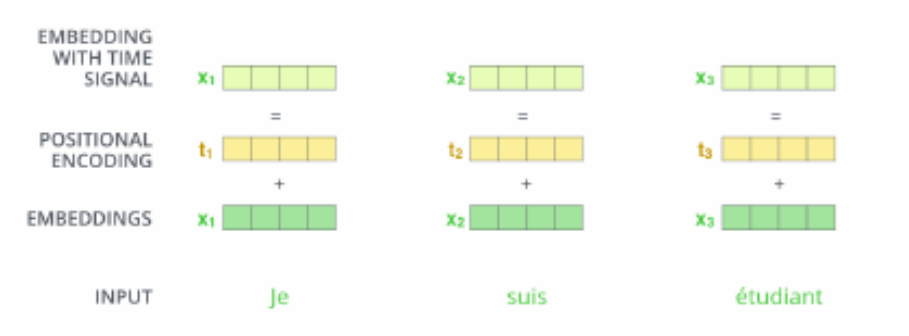

Figura 11.1: Vector de control del orden de cada palabra.

Cada sub-layer (self-attention, feed forward neural net) en cada encoder tiene una **conexión residual** alrededor de él, y es seguido por un paso de normalización de capa (layer-normalization). Esto ayuda a estabilizar el entrenamiento y permite que la información fluya más fácilmente a través de las capas del modelo, mitigando problemas como el desvanecimiento del gradiente.

En lugar de que una capa intente aprender directamente la representación ideal (llamémosla $H(x)$), la conexión residual obliga a la capa a aprender únicamente una función de residuo o modificación: $F(x) = H(x) - x$. Por lo tanto, la operación matemática exacta que ocurre en cada subcapa del bloque se define así:

$$\text{Salida} = x + F(x)$$

donde:x es la identidad o información original que entra a la subcapa (viaja a través de la flecha que "salta" la capa por el lateral), y $F(x)$ es la transformación compleja computada por la subcapa (ya sea la atención o el MLP).

Observación. Una de las claves de usar los residuales es evitar el **desvanecimiento del gradiente**, que se basa en que si los valores de los pesos de las capas son menores que 1, al multiplicarlos tantas veces el gradiente se reduce exponencialmente hasta hacerse cero (0). Al llegar a las primeras capas del modelo, el gradiente ha desaparecido y la red deja de aprender. La solución matemática, viene al aplicar la derivada sobre la suma residual $\frac{\partial}{\partial x}(x + F(x))$, donde obtenemos

$$\frac{\partial \text{Salida}}{\partial x} = 1 + \frac{\partial F(x)}{\partial x}$$

ese número 1 es mágico, pues garantiza que, incluso si la derivada de la capa compleja $\frac{\partial F(x)}{\partial x}$ se desvanece y se vuelve cero, el gradiente total siempre tendrá al menos un valor de 1. El gradiente puede fluir directamente hacia atrás a través del "atajo" sin alterarse, permitiendo que todas las capas del modelo se entrenen correctamente.

11.2. Tranformer Encoder

El bloque de encoder del transformer es fundamental porque representa la "caja negra" que recibe una secuencia de datos y la transforma en una representación matemática profundamente contextualizada. Vamos a explicar su arquitectura y funcionamiento.

Cada transformer encoder, tiene varios bloques, que se ejecutan varias veces, pero todos tienen la misma estructura que viene dada en la Figura **??** y explicaremos a continuación

1. **Multi-Head Attention (MHA)**: es el motor que ya estudiamos. Toma la secuencia y calcula las relaciones globales y simultáneas entre todos los tokens utilizando múltiples cabezas en paralelo.

2. **Add & Norm**: en esta capa se aplica normalización a la salida anterior y su suma con los residuales, lo que queda como

$$x' = \text{LayerNorm}(x + MHA(x))$$

3. **Feed Forward Network** (FFN): una vez que la atención ha permitido que los tokens "hablen entre sí" e intercambien información, cada token pasa de forma independiente por una red neuronal densa. Dicha red neuronal suele ser dos capas con ReLu como función de activación, aplicada de forma posicional (es decir, cada token se procesa de forma independiente sin compartir información entre ellos en esta etapa).

11.3. Vision Transformer (ViT)

Si aqui intentasemos aplicar el transforemer de forma nativa a una imagen tratando cada píxel como si fuera una palabra (un token), la complejidad computacional colapsaría el hardware. Para una imagen estándar y pequeña de 256×256 píxeles, tenemos un total de 65,536 píxeles, lo que resulta inviable.

La solución a este problema es dividir la imagen en parches (patches) de tamaño fijo, por ejemplo, 16×16 píxeles. Cada parche se trata como un token individual, lo que reduce drásticamente el número de tokens a procesar.

Observación. Un **pipeline** (que se puede traducir como tubería o canalización de datos) se refiere a una secuencia ordenada y encadenada de pasos o bloques operativos donde la salida (output) de un componente se convierte directamente en la entrada (input) del siguiente.

Veremos a continuación el pipeline completo de un Vision Transformer (ViT):

1. **Patch extraction**: la imagen se divide en N parches. El número total de parches se calcula con la fórmula

$$N = \frac{H \times W}{P^2}$$

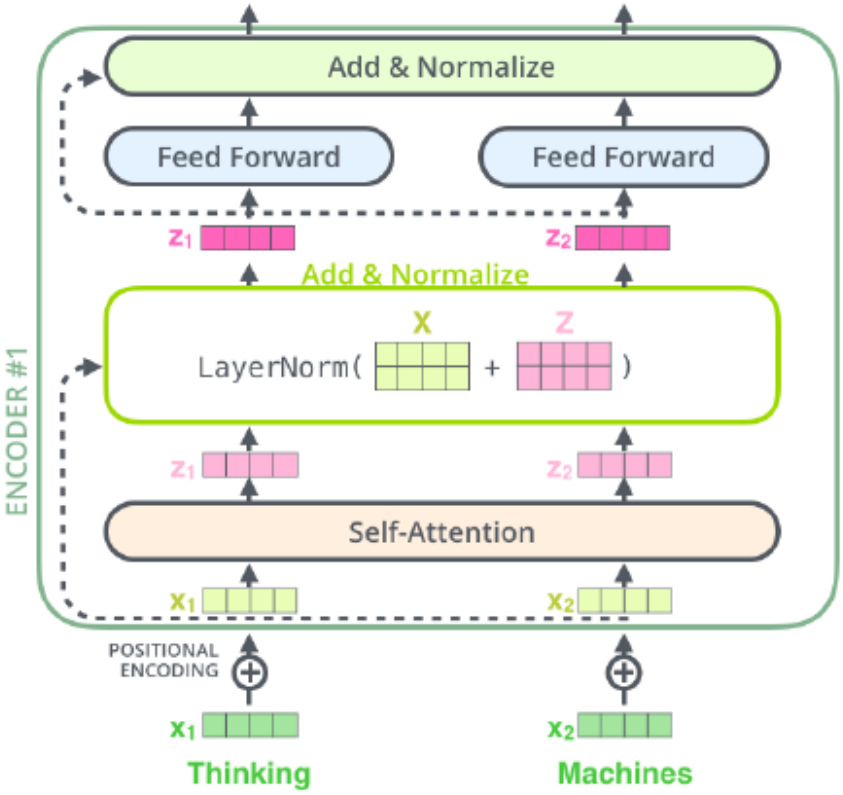

Figura 11.2: Arquitectura de un bloque de encoder del transformer.

el predeterminado es $P = 16$.

2. **Linear Projection**: cada parche bidimensional tiene un tamaño de $P \times P \times C$ (donde C son los canales de color, ej. 3 para RGB). Este parche se aplana (flatten) en un vector largo y se multiplica por una matriz de proyección lineal para transformarlo en un vector de características de dimensión D (el embedding).

3. **Prepend class token**: se añade un token especial inventado y "aprendible" al principio de la secuencia. Este token no corresponde a ningún trozo de la imagen; su función es absorber la información global de todos los demás parches a lo largo de la red. Al final, la clasificación de la imagen se hará únicamente mirando la salida de este token.

4. **Add positional embeddings**: los transformers procesan todos los tokens en paralelo, por lo que el modelo no sabe qué parche está arriba a la izquierda y cuál abajo a la derecha. Para darles noción espacial, se les suma un vector de posición unidimensional (1D) que el modelo aprende durante el entrenamiento.

5. **Transformer encoder**: la secuencia completa (el token de clase + los tokens de los parches con sus posiciones) se introduce en un codificador estándar como el que estudiamos antes

$$\text{Transformer Encoder} = \underbrace{\text{MHA} + \text{FFN} + \text{LayerNorm}}_{L \text{ capas}}$$

6. **MLP Head**: la salida del [class] token se pasa por un Multilayer Perceptron (MLP) final que realiza la clasificación (por ejemplo, predecir si la imagen es un "perro" o un "gato").

Algunas de las propiedades clave de ViT son

- No tiene sesgos inductivos específicos de las imágenes (no es invariante a traslaciones, no tiene localización implícita).
- Tiene un campo receptivo global desde la primera capa (cada parche puede atender a todos los demás).
- Requiere pre-entrenamiento a gran escala para igualar el rendimiento de las CNNs.
- A gran escala, supera el rendimiento de las CNNs y es altamente paralelizable.

Observación. Podemos hablar de las **Hybrid ViTs**, que son modelos que combinan la arquitectura de los transformers con las CNNs. En lugar de alimentar directamente los parches de imagen al transformer, se utiliza una CNN para extraer características locales de la imagen (como ResNet), y luego estas características se alimentan al transformer para capturar relaciones globales.

11.4. Swin Transformer

El Swin Transformer es una variante del Vision Transformer que introduce la idea de ventanas deslizantes (sliding windows) para reducir la complejidad computacional. En lugar de calcular la atención global entre todos los parches, el Swin Transformer divide la imagen en ventanas no superpuestas y calcula la atención solo dentro de cada ventana. Las dos claves de la innovación de los swin transformers son la **jerarquía** y la **localidad**.

Veámos la explicación de los puntos más innovadores de esta arquitectura:

- **Hierarchical feature maps (mapas de características jerárquicos)**: en lugar de mantener los parches fijos, swin comienza construyendo parches muy pequeños (submuestreo de $4\times$) y, a medida que la información avanza hacia las capas más profundas de la red, va fusionando parches vecinos (pasando de $4\times \to 8\times \to 16\times$). Esto construye una pirámide de características (Feature Pyramid), exactamente igual que hacían las CNNs. Las capas iniciales procesan detalles finos y locales, mientras que las capas profundas capturan estructuras globales de alto nivel.
- **Shifted Window (SW) attention (Atención en ventanas locales)**: para destruir la complejidad cuadrática $O(n^2)$, swin decide que los parches no pueden mirar a toda la imagen libremente. La imagen se divide en un conjunto de "ventanas" locales de tamaño fijo $M \times M$ parches. La regla matemática es que el operador de autoatención se calcula únicamente entre los parches que están dentro de la misma ventana. Como el tamaño de la ventana M es constante, la complejidad del modelo pasa de ser cuadrática ($O(n^2)$) a ser lineal ($O(n)$) respecto al tamaño de la imagen.

- **Shifted Windows (Ventanas desplazadas)**: limitar la atención a ventanas aisladas tiene un peligro: los parches de la Ventana A jamás podrían "hablar" con los parches de la Ventana B, perdiendo la capacidad de entender el contexto global. Para solucionar esto sin perder la eficiencia lineal, swin introduce su innovación estrella: alternar las particiones. En una capa de la red, las ventanas se organizan de forma regular. En la siguiente capa, la cuadrícula de ventanas se desplaza físicamente (un "shift") una fracción de su tamaño hacia abajo y hacia la derecha. Al desplazar la cuadrícula, las nuevas ventanas ahora contienen parches que antes pertenecían a ventanas vecinas distintas, forzando la comunicación y la transferencia de contexto entre ventanas contiguas en el paso anterior.

Observación. Un resumen nemotécnico puede ser el siguiente

- ViT: Escala única ($16\times$) + Atención globalizada $\rightarrow$ Complejidad $O(n^2)$
- Swin: Multiescala (pirámide) + Atención confinada a ventanas locales desplazadas $\rightarrow$ Complejidad lineal $O(n)$.
- Ventana desplazada (Shifted Window): El truco de diseño que permite conectar la información de ventanas vecinas sin disparar el coste computacional.

11.4.1. Transformers for Specific Vision Tasks

DETR. El **Detection Transformer** (DETR) es un modelo de detección de objetos que utiliza la arquitectura de transformers para abordar la tarea de detectar y clasificar objetos en imágenes. DETR es un modelo completamente End-to-End, lo que significa que puede aprender a detectar objetos directamente a partir de los datos sin necesidad de pasos intermedios específicos.
Su pipeline estructural se compone de una CNN (como ResNet) que extrae un mapa de características de la imagen, seguido de un Transformer Encoder que procesa estas características. Luego, un Transformer Decoder toma un número fijo de vectores aprendibles llamados Object Queries, que interrogan a las características del encoder para predecir directamente las cajas delimitadoras y las clases de los objetos presentes en la imagen.

11.4.2. Self-Supervised and Multimodal Transformers

DINO. El **Distillation with No Labels** es un modelo de aprendizaje autosupervisado (self-supervised) que se basa en la arquitectura de Vision Transformer (ViT). DINO se entrena sin utilizar etiquetas (labels) y logra aprender representaciones visuales muy potentes.
El mecanismo clave detrás de DINO es la auto-destilación, donde se utilizan dos redes con la misma arquitectura ViT: un Estudiante y un Profesor. Al estudiante se le muestran recortes pequeños y modificados de la imagen (augmented crops), mientras que al profesor se le muestran vistas globales más grandes. El estudiante es optimizado matemáticamente para replicar las

predicciones que realiza el profesor, lo que permite que el modelo aprenda representaciones visuales ricas y útiles sin necesidad de supervisión explícita.

12. Vision Language models

No relevante.

13. Denoising probabilistc models

No relevante.

14. Camera Calibration and Stereo Vision

Las cámaras fotográficas operan esencialmente como máquinas de reducción de dimensionalidad, proyectando el mundo tridimensional real $\mathbb{R}^3$ sobre un plano bidimensional discreto $\mathbb{R}^2$. El modelo matemático estándar que describe este fenómeno es la **Cámara Pinhole**.

La relación geométrica exacta entre un punto del espacio tridimensional expresado en coordenadas homogéneas $\mathbf{X} = [X, Y, Z, 1]^T$ y su correspondiente proyección en el plano de la imagen en píxeles homogéneos $\mathbf{x} = [x_h, y_h, w]^T$ se define mediante la **Ecuación Proyectiva de la Cámara**:

$$\mathbf{x} \sim P\mathbf{X}$$

donde P es la **Matriz de la Cámara (Camera Matrix)**, una matriz de dimensiones 3×4 y con 11 grados de libertad independientes que concentra tanto la geometría interna como la posición espacial del dispositivo.

Para entender las matrices, primero debemos saber que el ordenador maneja tres "mundos" o sistemas de referencia:

- **Mundo Real ($\mathbf{X}_w$):** donde las coordenadas se miden en metros respecto a un origen arbitrario (ej. la esquina de la habitación).
- **Cámara ($\mathbf{X}_c$):** un sistema donde el origen $(0, 0, 0)$ es exactamente el centro óptico (el .ºjo") de la cámara. El eje Z apunta hacia adelante (eje óptico), el eje X a la derecha y el eje Y hacia abajo.
- **Imagen (x):** el plano 2D digital medido puramente en píxeles.

14.1. Descomposición de la Matriz de la Cámara P

La matriz global de proyección P se puede descomponer algebraicamente en el producto de dos bloques de matrices diferenciados, mapeando el punto desde el sistema global del universo, pasando por el ojo de la cámara, hasta el sensor digital:

$$P = K\,[R \mid \mathbf{t}]$$

donde K contiene los parámetros intrínsecos de la cámara (de tamaño 3×3) y $[R \mid \mathbf{t}]$ representa los parámetros extrínsecos (de tamaño 3×4).

14.1.1. Parámetros Intrínsecos (Matriz K)

Los **parámetros intrínsecos** describen las propiedades geométricas y ópticas internas de la cámara. Su calibración es fija e independiente de dónde se coloque el dispositivo en el espacio. La matriz K se estructura de la siguiente forma:

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

- **Distancia Focal (f_x, f_y):** Representa la distancia desde el centro óptico (pinhole) hasta el plano del sensor, expresada en unidades de píxeles en lugar de metros o milímetros para absorber el tamaño físico de los fotosentidos en los ejes X e Y.

  En una cámara cualquier, el sensor del foco de una camara es donde se recoge la información de luz, mientras que el pinhole es el punto de proyección, que nosotros lo pensaremos como la lente de cristal por el que entra toda la información (lo que viene a ser el cristal que podemos limpiar). La distancia focal es la distancia entre el pinhole y el sensor, y es un parámetro fundamental que determina el campo de visión de la cámara, y es propio de esta. Aunque de primeras pensemos que tiene sentido que se mida dicha distancia en milímetros, se mide en pixeles porque el tamaño físico del sensor (y por tanto el tamaño de cada píxel) varía entre cámaras.

- **Punto Principal (c_x, c_y):** son las coordenadas del centro óptico de proyección, **pinhole** (el eje óptico que intersecta perpendicularmente al sensor) medidas en el sistema de píxeles de la imagen (típicamente cercano al centro de la resolución de la foto). Es el punto exacto donde la luz se concentra y se cruza.

- **Factor de Cizalladura / Skew (s):** mide la falta de ortogonalidad o la distorsión angular entre los ejes de píxeles del sensor. En las cámaras modernas, este valor es prácticamente cero ($s = 0$).

14.1.2. Parámetros Extrínsecos ($[R \mid \mathbf{t}]$)

Los **parámetros extrínsecos** definen la transformación de cuerpo rígido que mapea las coordenadas desde el sistema de referencia global del mundo real al sistema de coordenadas local de la cámara:

- **Matriz de Rotación (R):** una matriz ortogonal de 3×3 ($R^T R = I$ que implica que no deforma el espacio) con 3 grados de libertad (que corresponden a los tres giros posibles en el espacio tridimensional) que orienta los ejes de la cámara respecto al universo. El objetivo es alinear los ejes de coordenadas del mundo con los ejes de la cámara, es decir, se encarga de girar los ejes del

espacio para que el eje X de la habitación mire hacia el mismo lado que el eje X de la cámara, y lo mismo con el Y y el Z. Los tres movimientos básicos son

- **Pan (o Yaw)**: girar la cámara hacia la izquierda o la derecha.
- **Tilt (o Pitch)**: cabecear la cámara hacia arriba o hacia abajo.
- **Roll**: inclinar la cámara de lado (como cuando ladeas la cabeza para poner una foto en vertical).

- **Vector de Traslación (t):** un vector de 3×1 que desplaza el origen de coordenadas global hasta la posición física del centro óptico de la cámara, dado como sigue

$$\mathbf{t} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

El vector $\mathbf{t}$ mide simplemente el desplazamiento físico en el espacio entre los dos orígenes de coordenadas. Te dice cuántos metros tienes que caminar desde el origen de coordenadas del mundo hasta llegar al centro del objetivo de la cámara (origen de la cámara).

14.2. Algoritmo de Calibración de la Cámara

El objetivo del **Algoritmo de Calibración** es estimar numéricamente de forma precisa los elementos de la matriz intrínseca K y los parámetros extrínsecos $[R \mid \mathbf{t}]$. El algoritmo es un método de estimación pasivo: requiere que le demos datos de entrada conocidos.

1. **Inputs de Calibración (Calibración Pasiva):** se colocan imágenes de un patrón de geometría perfectamente conocida de distintos ángulos, comúnmente una plantilla de tablero de ajedrez (checkerboard images), donde se conocen exactamente las coordenadas 3D de cada esquina del tablero en el sistema de referencia del mundo.

2. **Correspondencias de Puntos:** para cada una de las imágenes capturadas, el algoritmo debe identificar un conjunto de n puntos de control.

   - Definimos un sistema de coordenadas del mundo virtual fijando el origen $(0, 0, 0)_w$ en la esquina superior izquierda del tablero físico. Cada esquina del tablero tiene coordenadas métricas 3D conocidas en este sistema, por ejemplo: $\mathbf{X}_w^{(1)} = (0, 0, 0)^T$, $\mathbf{X}_w^{(2)} = (30, 0, 0)^T$ (en milímetros). Podemos calcular de forma exacta y perfecta la posición tridimensional de cada esquina interna en el plano del tablero ($Z_w = 0$)

$$\mathbf{X}_w^{(1)} = (0, 0, 0)^T, \quad \mathbf{X}_w^{(2)} = (30, 0, 0)^T, \quad \mathbf{X}_w^{(3)} = (60, 0, 0)^T \ldots$$

   - El ordenador procesa la fotografía digital utilizando detectores de esquinas basados en gradientes de brillo (como el detector de Harris o algoritmos de análisis de sillas de montar). Localiza en qué coordenadas de

píxeles exactas de la pantalla ha salido retratada cada una de esas esquinas

$$\mathbf{u}^{(1)} = (u^{(1)}, v^{(1)})^T, \quad \mathbf{u}^{(2)} = (u^{(2)}, v^{(2)})^T \dots$$

- Al final de este paso, tenemos una lista de parejas $\left\{\left(\mathbf{X}_w^{(i)}, \mathbf{u}^{(i)}\right)\right\}_{i=1}^{n}$ que correlacionan el espacio real con la foto.

3. **Estimación Matemática (DLT):** sabemos que la cámara proyecta los puntos siguiendo el modelo pinhole homogéneo $\mathbf{x} \sim P\mathbf{X}$, donde P es la matriz de la cámara de tamaño 3×4 compuesta por 12 incógnitas (p_{11} hasta p_{34}). Si expandimos la multiplicación de la matriz por el vector del mundo para un punto i, el sistema nos dicta

$$\begin{bmatrix} w \cdot u^{(i)} \\ w \cdot v^{(i)} \\ w \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} X_w^{(i)} \\ Y_w^{(i)} \\ Z_w^{(i)} \\ 1 \end{bmatrix}$$

Para deshacernos del factor de escala "fantasma" w, sustituimos la tercera fila ($w = p_{31}X_w^{(i)} + p_{32}Y_w^{(i)} + p_{33}Z_w^{(i)} + p_{34}$) en las dos primeras filas. Al hacer esto, obtenemos las ecuaciones no lineales de proyección real en píxeles

$$u^{(i)} = \frac{p_{11}X_w^{(i)} + p_{12}Y_w^{(i)} + p_{13}Z_w^{(i)} + p_{14}}{p_{31}X_w^{(i)} + p_{32}Y_w^{(i)} + p_{33}Z_w^{(i)} + p_{34}} \quad , \quad v^{(i)} = \frac{p_{21}X_w^{(i)} + p_{22}Y_w^{(i)} + p_{23}Z_w^{(i)} + p_{24}}{p_{31}X_w^{(i)} + p_{32}Y_w^{(i)} + p_{33}Z_w^{(i)} + p_{34}}$$

4. **Rearrange into a linear system:** para procesar todos los puntos a la vez, el ordenador vectoriza los 12 elementos desconocidos de la matriz en un único vector columna largo $\mathbf{p} = (p_{11}, p_{12}, \dots, p_{34})^T$ de tamaño 12×1. Al apilar las filas de coeficientes conocidos (compuestos por las coordenadas físicas y los píxeles medidos), construimos una matriz gigante A de tamaño $2n \times 12$, es decir, para cada punto tenemos estas dos ecuaciones

$$p_{11}X_w^{(i)} + p_{12}Y_w^{(i)} + p_{13}Z_w^{(i)} + p_{14} - u^{(i)}X_w^{(i)}p_{31} - u^{(i)}Y_w^{(i)}p_{32} - u^{(i)}Z_w^{(i)}p_{33} - u^{(i)}p_{34} = 0$$
$$p_{21}X_w^{(i)} + p_{22}Y_w^{(i)} + p_{23}Z_w^{(i)} + p_{24} - v^{(i)}X_w^{(i)}p_{31} - v^{(i)}Y_w^{(i)}p_{32} - v^{(i)}Z_w^{(i)}p_{33} - v^{(i)}p_{34} = 0$$

que equivale en forma matricial a

$$\begin{bmatrix} X_w^{(i)} & Y_w^{(i)} & Z_w^{(i)} & 1 & 0 & 0 & 0 & 0 & -u^{(i)}X_w^{(i)} & -u^{(i)}Y_w^{(i)} & -u^{(i)}Z_w^{(i)} & -u^{(i)} \\ 0 & 0 & 0 & 0 & X_w^{(i)} & Y_w^{(i)} & Z_w^{(i)} & 1 & -v^{(i)}X_w^{(i)} & -v^{(i)}Y_w^{(i)} & -v^{(i)}Z_w^{(i)} & -v^{(i)} \end{bmatrix} \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \\ p_{31} \\ p_{32} \\ p_{33} \\ p_{34} \end{bmatrix} = \begin{bmatrix}$$

de manera que el problema se reduce a resolver el siguiente sistema homogéneo

$$\mathbf{Ap} = \mathbf{0}$$

de dimensión $2n \times 12$.

5. **Solve via SVD**: el sistema homogéneo $A\mathbf{p} = \mathbf{0}$ tiene una solución trivial $\mathbf{p} = \mathbf{0}$, que no es útil. Para encontrar la solución no trivial se busca minimizar el error de proyección (debido al ruido no conseguiremos esa igualdad exacta de 0), es decir

   $$\|A\mathbf{p}\|^2$$

   donde restringiremos $\|p\| = 1$ pues existen muchas matrices escalares de P que representan la misma cámara. Para ellos se aplica la **Descomposición en Valores Singulares (SVD)** a la matriz A.

   La solución óptima es el vector singular de

   $$A^T A$$

   correspondiente al valor singular más pequeño $\lambda_{\text{mín}}$ de A, que es equivalente al último vector de la matriz V de la SVD de A, donde tenemos

   $$A = U\Sigma V^T$$

Observación. Dado $n \geqslant 6$ correspondencias 3D-2D, construimos la matriz A de tamaño $2n \times 12$, calculamos la SVD de A y tomamos la última columna de V como el vectorizado de la matriz de proyección p. Reshapeamos p en la matriz de proyección P de tamaño 3×4. Este procedimiento se llama el **Direct Linear Transform (DLT)**.

14.2.1. Extracción de parámetros

Dada la matriz P obtenida por el algoritmo de calibración, el siguiente paso es extraer los parámetros intrínsecos K y extrínsecos $[R \mid \mathbf{t}]$ de forma separada, donde tenemos

$$P = K\,[R \mid \mathbf{t}] = \begin{bmatrix} KR & | & K\mathbf{t} \end{bmatrix}$$

Separar K y R. The left 3×3 sub-matrix of P is the product KR

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} = KR$$

since K is upper-triangular and R is orthonormal, their product can be uniquely decomposed using RQ factorisation (a variant of QR factorisation). This gives K and R separately.

Recovering the Translation t. The last column of the matrix P is by definition the block $K\mathbf{t}$:

$$\begin{bmatrix} p_{14} \\ p_{24} \\ p_{34} \end{bmatrix} = K\mathbf{t} \implies \mathbf{t} = K^{-1} \begin{bmatrix} p_{14} \\ p_{24} \\ p_{34} \end{bmatrix}$$

the camera centre in world coordinates can then be recovered as $\mathbf{c}_w = -R^T\mathbf{t}$.

14.3. Stereo Vision Basics

La visión estéreo emula la percepción tridimensional humana utilizando **dos cámaras o vistas separadas** horizontalmente por una distancia conocida para recuperar la tercera dimensión (la profundidad Z) ausente en una sola proyección plana.

14.3.1. La Relación Disparidad-Profundidad

Definición 14.3.1. Se define la **disparidad** (d) como la diferencia en la posición horizontal en píxeles que sufre la proyección de un mismo punto del espacio entre la imagen izquierda (x_l) y la imagen derecha (x_r)

$$d = x_l - x_r$$

A partir de la semejanza de triángulos en una geometría de cámaras perfectamente idealizada y alineada, se deduce la **Ecuación Fundamental de la Visión Estéreo**:

$$\mathbf{Z = \frac{f \cdot B}{d}}$$

donde:

- Z: distancia de profundidad del punto al plano de las cámaras.
- f: distancia focal del sistema óptico.
- B: **línea de base (Baseline)**, la distancia física de separación horizontal entre los centros ópticos de ambas cámaras.
- d: disparidad calculada.

14.3.2. Disparidad Map

El resultado de procesar un par estéreo es un **mapa de disparidad (Disparidad Map)**, que se puede interpretar de forma directa como un mapa de relieve geométrico inverso gracias a su ley de proporcionalidad:

- **Mayor Disparidad $\Longrightarrow$ Objeto Cercano:** Si un objeto está muy próximo a las lentes, su salto de píxeles horizontal entre la imagen izquierda y derecha será gigante.
- **Menor Disparidad $\Longrightarrow$ Objeto Lejano:** Si el objeto está situado en el fondo de la escena o cerca del infinito, sus proyecciones impactarán prácticamente en las mismas coordenadas en ambas pantallas ($d \to 0$), lo que implica matemáticamente que la profundidad tiende a infinito ($Z \to \infty$).

14.4. Image Rectification

Definición 14.4.2. Si tú ves un punto en la imagen izquierda, en el mundo real ese píxel podría ser un objeto cercano, o un objeto mediano, o una estrella en el infinito. Ese píxel representa un rayo de luz infinito que sale de la cámara. Visto desde la segunda cámara, ese rayo de luz se ve como una línea recta inclinada que cruza su pantalla. Esa es la **línea epipolar**. Buscar el punto homólogo significa buscar a lo largo de esa línea inclinada.

En una configuración física real, es mecánicamente imposible colocar dos cámaras de forma que queden perfectamente paralelas y alineadas a nivel de micras de píxel (situación que suponíamos antes).
Debido a este desalineamiento físico, si intentas buscar el mismo punto en ambas imágenes, la línea epipolar (el camino donde se esconde el punto hermano) saldrá inclinada y torcida en diagonal a través de la pantalla de la segunda cámara. Para encontrar el punto homólogo, tu algoritmo tendría que buscar haciendo un barrido en dos dimensiones (2D): recorriendo píxel por píxel en horizontal y en vertical a lo largo de esa diagonal. Esto es computacionalmente lentísimo ($O(N^2)$) y propenso a errores.

Esto obliga a realizar una fase de software previa llamada **rectificación**, lo que viene a ser "enderezar la perspectiva de las fotos", no tocamos las cámaras físicas, lo que hacemos es corregir las fotografías digitalmente por software después de haber sido tomadas.

- **En qué consiste conceptualmente:** el algoritmo toma los parámetros de calibración que calculamos previamente (con el tablero de ajedrez) y calcula dos homografías (transformaciones proyectivas planas), una para la imagen izquierda (H_L) y otra para la derecha (H_R).

  Conceptualmente, es como si el ordenador tomara las dos fotos reales, las estirara y las retorciera matemáticamente en el aire, y las proyectara sobre un lienzo virtual común que es perfectamente plano (coplanar) y paralelo a la línea que une las dos cámaras. Es una "cirugía de perspectiva".

- **Alineamiento de Líneas Epipolares:** su efecto inmediato es que las líneas epipolares se vuelven **paralelas** y se **alinean horizontalmente** con las filas de la imagen.

- **Simplificación del Emparejamiento (Matching):** sin rectificación, buscar un punto homólogo requiere explorar toda la imagen en dos dimensiones (un coste de búsqueda 2D de $\mathcal{O}(N^2)$). Tras la rectificación, el punto homólogo de un píxel situado en la fila $Y = 45$ de la imagen izquierda está garantizado que vive estrictamente en **la misma fila $Y = 45$ de la imagen derecha**. La búsqueda se colapsa a un problema lineal unidimensional (1D) a lo largo de la fila, acelerando drásticamente el cálculo en tiempo real.

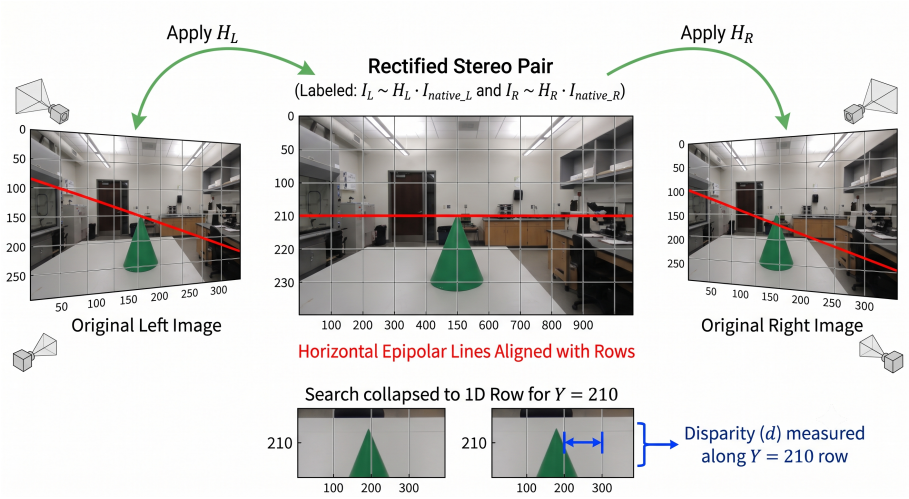

Figura 14.1: Ejemplo de rectificación de la imagen.

Podemos ver una representación visual de la rectificación en la Figura 14.1. Destacar que aunque en la figura se vea que solo queda una imagen para las dos homologías, esto no es del todo cierto, realmetne se crean dos imágenes, una para cada homología y por tanto para cada cámara, pero en la figura se ha representado solo una de ellas para simplificar la visualización. De manera, que se pasa de una imagen a otra manteniendo el eje Y constante, es decir, la rectificación no cambia la coordenada vertical de los píxeles, solo la horizontal. Esto es lo que permite que las líneas epipolares se alineen horizontalmente y se vuelvan paralelas.

15. Epipolar Geometry and Fundamental Matrix

En la geometría de una única vista, la proyección de un punto tridimensional del espacio sobre el plano de la imagen introduce una pérdida irreversible de dimensionalidad. Dado un único punto proyectado $\mathbf{x}$, solo es posible realizar una retroproyección en el espacio en forma de rayo infinito sobre el cual descansa el punto 3D real.

La **Geometría de Dos Vistas (Two-View Geometry)** resuelve esta ambigüedad introduciendo una segunda cámara posicionada en una ubicación arbitraria del espacio. El objetivo central de esta rama es modelar la relación geométrica intrínseca entre un punto 3D del espacio, su proyección en la primera imagen y su proyección homóloga en la segunda captura.

15.1. Epipolar Geometry

La geometría epipolar describe la geometría proyectiva intrínseca entre dos vistas independientes de una misma escena estática. Depende exclusivamente de los parámetros internos de las cámaras y de su movimiento relativo (coplanares rígidos), siendo totalmente independiente de la estructura de la escena tridimensional vista.

15.1.1. Elementos Fundamentales de la Anatomía Epipolar

Consideremos dos cámaras con centros ópticos (o centros de proyección) localizados en $\mathbf{O}$ y $\mathbf{O}'$, respectivamente:

- **Línea de Base (Baseline):** Es el segmento de recta física en el espacio tridimensional que conecta de forma directa los dos centros de proyección de las cámaras ($\mathbf{O}$ y $\mathbf{O}'$).
- **Epipoles – $\mathbf{e}, \mathbf{e}'$:** el epípolo es matemáticamente la proyección del centro óptico de una cámara sobre el plano de la imagen de la otra cámara.
    - El epípolo izquierdo $\mathbf{e}$ es la proyección de $\mathbf{O}'$ en la imagen izquierda.
    - El epípolo derecho $\mathbf{e}'$ es la proyección de $\mathbf{O}$ en la imagen derecha.
    - Alternativamente, se definen como los puntos de intersección de la línea de base con ambos planos de imagen.

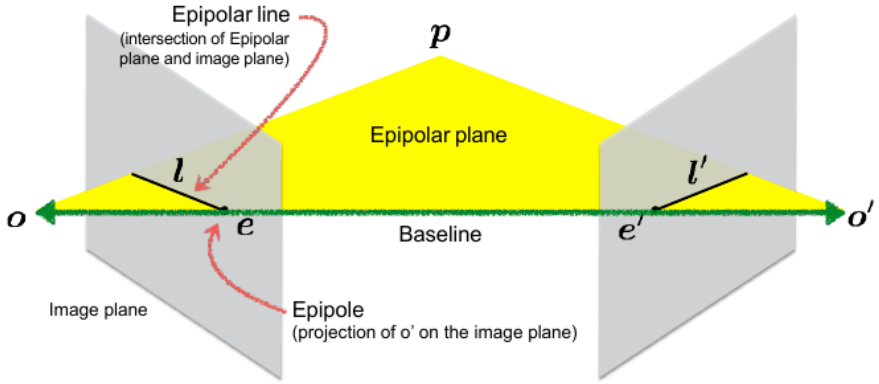

Figura 15.1: Anatomía de la Geometría Epipolar: Línea de Base, Epípolos, Plano Epipolar y Líneas Epipolares

- **Epipolar Plane:** es el plano tridimensional único delimitado y definido por el punto del espacio real $\mathbf{P}$ y los dos centros ópticos de las cámaras $(\mathbf{O}, \mathbf{O}', \mathbf{P})$. Al rotar el punto $\mathbf{P}$ por el espacio, este plano pivota utilizando la línea de base como su eje central de bisagra.
- **Epipolar Lines – $\mathbf{l}, \mathbf{l}'$:** son las líneas rectas resultantes de la intersección del plano epipolar con los planos de imagen de cada cámara. Toda línea epipolar en una imagen contiene obligatoriamente al epípolo de dicha vista.

Podemo ver una representación de estos elementos en la Figura 15.1.

15.1.2. Epipolar Constraint

El principio fundamental de la restricción epipolar dicta que:

> *"Un punto geométrico medido en una primera imagen restringe la búsqueda de su correspondencia homóloga en la segunda vista a yacer estrictamente sobre la línea epipolar correspondiente."*

Intuición Geométrica Crucial: sin el uso de la restricción epipolar, encontrar el emparejamiento de un píxel requiere realizar una búsqueda bidimensional ($\mathcal{O}(N^2)$) por toda la superficie de la segunda imagen. Al aplicar este principio, el espacio de búsqueda **se colapsa de forma directa a una dimensión (1D)** a lo largo de la línea epipolar recta.

15.1.3. Casos Especiales de Configuración de Cámaras

1. **Converging Cameras:** los epípolos son puntos finitos que se localizan frecuentemente dentro o cerca de los límites de la imagen, y las líneas epipolares convergen radialmente hacia ellos.
2. **Parallel Cameras:** ocurre cuando los ejes de visión de ambas cámaras son estrictamente paralelos y perpendiculares a la línea de base. En este escenario

geométrico, los centros ópticos proyectados se encuentran en el infinito proyectivo, lo que implica que **los epípolos divergen al infinito** ($\mathbf{e}, \mathbf{e}' \to \infty$). Consecuentemente, las líneas epipolares se vuelven paralelas entre sí y se alinean perfectamente de forma horizontal con las filas de los sensores píxel a píxel.

3. **Forward Motion:** ocurre en trayectorias de traslación rectilínea a lo largo del eje óptico (ej. cámaras a bordo de vehículos). El epípolo coincide exactamente con el *Foco de Expansión* (punto principal de la imagen), y las líneas epipolares divergen de forma radial desde dicho centro hacia afuera.

   Ejemplo. Imaginemos que vamos conduciendo por una autopista recta de noche. Si mirams al fondo, hay un punto exacto en el horizonte (el final de la carretera) del cual parecen nacer todas las líneas de los carriles y las farolas.

   - **Físicamente**: ese punto del horizonte es el lugar hacia el que nos dirigimos.
   - **En la imagen**: a ese punto de fuga se le llama *Foco de Expansión*. Coincide de forma aproximada con el centro de tu imagen (el punto principal (c_x, c_y) que estudiamos en la calibración).

15.2. Matriz Esencial → Matriz Fundamental

15.2.1. La Matriz Esencial (E)

Propuesta por **Longuet-Higgins** en 1981, asume una configuración estéreo calibrada donde las matrices de parámetros intrínsecos de ambas cámaras actúan como matrices identidad ($K = K' = I$). En este espacio analítico, las coordenadas 2D de la imagen están expresadas directamente en el sistema de referencia métrico de la cámara.

Imaginemos que estamos en un espacio tridimensional. Tenemos un punto físico en el mundo (por ejemplo, la esquina de una mesa) al que llamaremos $\mathbf{X}$ (en mayúscula para denotar que está en 3D). Para la cámara 1, este punto se proyecta en la imagen como $\mathbf{x}$, y para la cámara 2 se proyecta como $\mathbf{x}'$, dado por

$$\mathbf{x}' = R(\mathbf{x} - \mathbf{t})$$

donde para poder saber donde esta la cámara 2 con respecto a la cámara 1, se introduce una transformación rígida compuesta por una rotación R y una traslación $\mathbf{t}$ que conecta ambos sistemas de referencia de las cámaras.

Si nos paramos en el centro de la Cámara 1, podemos definir tres vectores que viven (están atrapados) dentro de ese mismo plano:

- El vector $\mathbf{x}$ (que va del centro 1 al punto del objeto).
- El vector $\mathbf{t}$ (que va del centro 1 al centro 2).

- El vector $(\mathbf{x}-\mathbf{t})$ (que es el vector que va desde el centro 2 al punto del objeto, pero medido desde los "ojos" de la Cámara 1).

por lo que si tres vectores están en el mismo plano (son coplanares), significa que no tienen volumen en 3D. Podemos ver una represanticón de estos vectores en la Figura 15.2. El **scalar triple product** calcula el volumen del paralelepípedo que forman. Como están aplastados en un plano, ese volumen es cero:

$$(\mathbf{x}-\mathbf{t})^\top(\mathbf{t}\times\mathbf{x})=0$$

puesto que $x'=R(x-t)$, tenemos $(x-t)=R^\top x'$ (donde $R^{-1}=R^\top$ por ser matriz ortogonal), entonces

$$x'^\top R(t\times x)=0$$

El truco maravilloso de Longuet-Higgins, fue usar que the cross product $t\times x$ can be written as a matrix-vector multiplication using the skew-symmetric matrix $[t]_\times$:

$$t\times x=[t]_\times x \quad \text{donde } [t]_\times=\begin{bmatrix}0 & -t_3 & t_2\\ t_3 & 0 & -t_1\\ -t_2 & t_1 & 0\end{bmatrix}$$

sabiendo que

$$t\times x=\det\begin{pmatrix}\hat{i} & \hat{j} & \hat{k}\\ t_1 & t_2 & t_3\\ x_1 & x_2 & x_3\end{pmatrix}=\big((t_2x_3-t_3x_2),\ (t_3x_1-t_1x_3),\ (t_1x_2-t_2x_1)\big)$$

y sustituyendo esto en la ecuación anterior

$$x'^\top R[t]_\times x=0 \implies E=R[t]_\times$$

donde tenemos la **Longest-Higgins equation**

$$\boxed{x'^\top Ex=0} \quad \text{where } E=R[t]_\times$$

donde E es la **Matriz Esencial** que encapsula la relación epipolar entre las dos vistas calibradas.

Conclusión Conceptual de E: la matriz esencial $E=R[\mathbf{t}]_\times$ codifica únicamente las propiedades extrínsecas de la escena (cómo está rotada una cámara respecto a otra y hacia dónde se ha desplazado). Su gran utilidad es que si un coche autónomo o un robot detecta un punto $\mathbf{x}$ en la primera cámara, al multiplicarlo por E obtiene instantáneamente la línea exacta donde está obligado a buscar ese mismo punto en la segunda cámara.

15.2.2. Fundamental Matrix (F)

Para modelar imágenes reales sin necesidad de calibración previa, eliminamos la suposición de matriz identidad parametrizando la transición desde las coordenadas

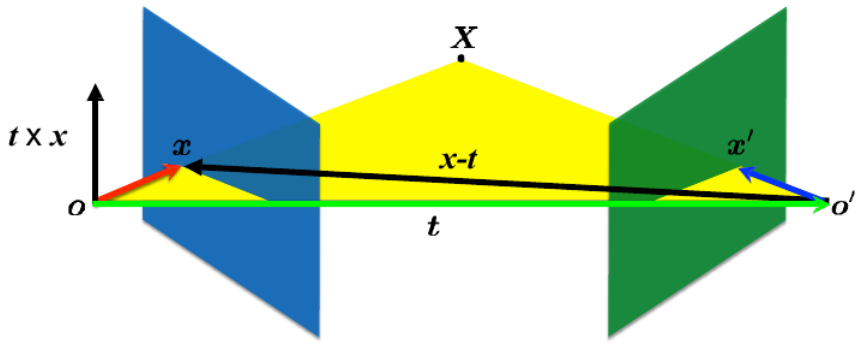

Figura 15.2: Vectores $\mathbf{x}$, $\mathbf{t}$ y $(\mathbf{x} - \mathbf{t})$ coplanares.

métricas puras de la cámara ($\hat{\mathbf{x}}$) hacia las coordenadas reales de la pantalla medidas en píxeles normales ($\mathbf{x}$) mediante la inversa de los intrínsecos, es decir

$$\hat{\mathbf{x}} = K^{-1}\mathbf{x} \quad \text{y} \quad \hat{\mathbf{x}}' = K'^{-1}\mathbf{x}' \tag{15.1}$$

recordemos que

- $\hat{\mathbf{x}}$ es un vector en metros que te dice exactamente con qué inclinación entró el rayo de luz a la cámara, lo que viene a ser el rayo de luz de la cámara apuntando al espacio (2D homogéneo). Su tercera coordenada siempre vale 1 por diseño. No sabe a cuántos metros reales estaba el farol (perdió la profundidad), solo sabe en qué dirección venía el rayo.

- $\mathbf{x}$ es el punto proyectado en coordenadas de píxeles de la imagen (en píxeles).

Sustituyendo estas expresiones en la matriz esencial, tenemos

$$(K'^{-1}\mathbf{x}')^\top E(K^{-1}\mathbf{x}) = 0 \implies \mathbf{x}'^\top(K'^{-\top}EK^{-1})\mathbf{x} = 0 \tag{15.2}$$

y definimos de este modo de forma única la **Matriz Fundamental** (F) con su ecuación correspondiente

$$\boxed{x'^\top Fx = 0} \quad \text{donde} \quad F = K'^{-\top}EK^{-1} \tag{15.3}$$

Para diferenciar ambas matrices tenemos estas ideas cruciales:

- La Matriz Esencial (E) solo entiende de geometría física en el mundo real (metros, ángulos, rotación, traslación). Requiere cámaras perfectamente calibradas.

- La Matriz Fundamental (F) entiende el lenguaje de la computadora (píxeles). Absorbe los parámetros intrínsecos de las cámaras y te permite relacionar dos imágenes directamente a partir de sus correspondencias de puntos sin saber qué cámara tomó la foto.

15.2.3. Propiedades Matemáticas y Relaciones Algebraicas de F

- **Mapeo de Punto a Línea:** Multiplicar la matriz F por las coordenadas homogéneas de un punto de la primera imagen computa de forma directa los coeficientes de la línea epipolar recta en la vista opuesta:

$$\mathbf{l}' = F\mathbf{x} \quad \text{(Línea epipolar en la imagen derecha)}$$
$$\mathbf{l} = F^\top \mathbf{x}' \quad \text{(Línea epipolar en la imagen izquierda)}$$

- **Grados de Libertad (DoF):** F es una matriz cuadrada de tamaño 3×3. Al operar sobre un espacio homogéneo, está definida únicamente salvo un factor de escala multiplicativo global arbitrario (reduciendo sus parámetros libres de 9 a 8). Adicionalmente, al estar forzada analíticamente por la restricción de rango, pierde un grado de libertad extra, resultando en un total de **7 Grados de Libertad independientes**.

- **Restricción de Rango (Rank Constraint):** Debido a que la matriz antisimétrica $[\mathbf{t}]_\times$ posee intrínsecamente rango igual a 2 y las matrices de calibración e intrínsecos K poseen rango completo (3), la multiplicación matemática dicta de forma estricta que la Matriz Fundamental posee una deficiencia de rango obligatoria:

$$\text{Rank}(F) = 2 \implies \det(F) = 0$$

- **Relaciones Matemáticas con los Epípolos (e, e'):** Dado que todos los rayos de proyección convergen en los epípolos correspondientes, estos constituyen los vectores del espacio nulo analítico (*Null Space*) derecho e izquierdo de la matriz F, respectivamente:

$$\boxed{F\mathbf{e} = 0} \implies \mathbf{e} \text{ vive en el espacio nulo derecho de } F$$
$$\boxed{F^\top \mathbf{e}' = 0} \implies \mathbf{e}\text{' vive en el espacio nulo izquierdo de } F$$

  Podemos ver más claramente porque los epípolos pertenecen siempre a las líneas epipolares en la imagen para cada punto $\mathbf{x}$ en la Figura 15.3.
  El resultado de la propiedad viene porque dada la primera propiedad se tiene $F\mathbf{x} = \mathbf{l}'$, y como $\mathbf{e}'$ pertenece a la línea epipolar $\mathbf{l}'$, entonces

$$\mathbf{e}'^\top \mathbf{l}' = 0 \implies \mathbf{e}'^\top (F\mathbf{x}) = 0 \implies (\mathbf{e}'^\top F)\mathbf{x} = 0$$

  como vale para cualquier $\mathbf{x}$, entonces

$$\mathbf{e}'^\top F = 0 \implies F^\top \mathbf{e}' = 0$$

15.3. Estimación de la Matriz Fundamental (F)

15.3.1. Eight-Point Algorithm

Cada pareja de puntos correspondientes conocidos $\{\mathbf{x}_m, \mathbf{x}'_m\}$ en coordenadas homogéneas de píxeles $\mathbf{x}_m = [x_m, y_m, 1]^\top$ y $\mathbf{x}'_m = [x'_m, y'_m, 1]^\top$ debe satisfacer de forma estricta la ecuación epipolar:

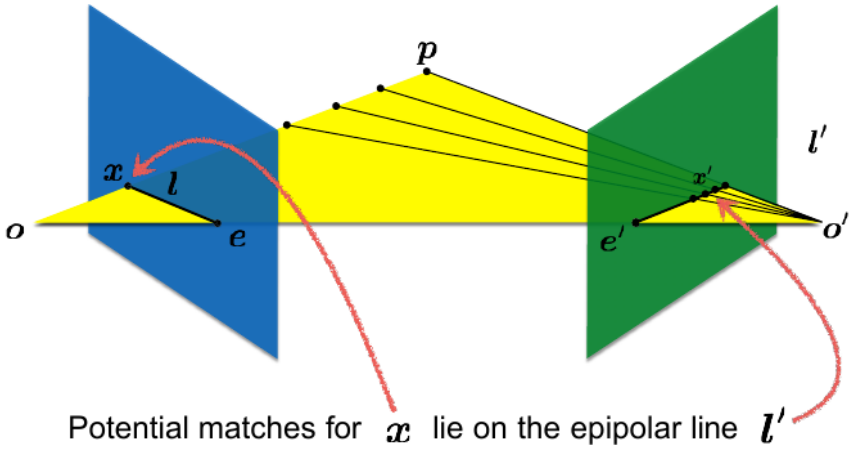

Figura 15.3: El mapeo de un punto $\mathbf{x}$ a su línea epipolar $\mathbf{l}'$ en la imagen derecha mediante la multiplicación por F.

$$\begin{bmatrix} x'_m & y'_m & 1 \end{bmatrix} \begin{bmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{bmatrix} \begin{bmatrix} x_m \\ y_m \\ 1 \end{bmatrix} = 0$$

Al expandir algebraicamente la multiplicación término a término, se demuestra de forma analítica que **un único emparejamiento de puntos aporta exactamente una única ecuación lineal limpia** con 9 incógnitas:

$$x_m x'_m f_1 + y_m x'_m f_2 + x'_m f_3 + x_m y'_m f_4 + y_m y'_m f_5 + y'_m f_6 + x_m f_7 + y_m f_8 + f_9 = 0$$

Al recolectar e indexar un conjunto mínimo de $M \geq 8$ correspondencias puntuales discretas, podemos apilar las filas de coeficientes numéricos medidos para construir un sistema homogéneo lineal matricial de la forma:

$$A\mathbf{f} = \mathbf{0}$$

donde A es una matriz gigante de tamaño $M \times 9$, y $\mathbf{f} = [f_1, f_2, \ldots, f_9]^\top$ corresponde al vector largo vectorizado que almacena las incógnitas de la cámara.

Resolución Matemática mediante Descomposición SVD

Para evitar el colapso del sistema matemático hacia la solución trivial obvia ($\mathbf{f} = \mathbf{0}$), se formula el problema bajo la teoría de Mínimos Cuadrados Totales (*Total Least Squares*) imponiendo una Restricción de Norma Unidad para la escala:

$$\min_{\mathbf{f}} \|A\mathbf{f}\|^2 \quad \text{sujeto a} \quad \|\mathbf{f}\|^2 = 1$$

El teorema del álgebra lineal demuestra que la solución óptima a esta minimización estructurada se obtiene de forma directa calculando la **Descomposición en Valores Singulares (SVD)** de la matriz de coeficientes del sistema A:

$$A = U\Sigma V^\top$$

El vector de incógnitas óptimo **f** se extrae tomando de forma directa **la última columna de la matriz** V (el autovector correspondiente al menor valor singular de la matriz). Posteriormente, se realiza una operación de cambio de forma (*reshape*) de 9×1 a una rejilla de 3×3 para moldear la matriz estimadora inicial F.

15.3.2. El Defecto del Algoritmo de 8 Puntos Estándar y Necesidad de Normalización

En imágenes digitales reales (ej. resolución Full HD 1920×1080), las coordenadas en píxeles de los extremos superiores de la matriz A son del orden de 10^3, lo que provoca que términos cuadráticos cruzados como $x_m x'_m$ alcancen magnitudes del orden de 10^6. En contraste, las últimas columnas de la matriz A contienen coordenadas constantes iguales a 1 (10^0), que se ve tan fácilmente sabiendo que

$$A_m = \begin{bmatrix} xx' & xy' & x & yx' & yy' & y & x' & y' & 1 \end{bmatrix}$$

donde $x_m = [x, y, 1]^\top$ (en la primera imagen) y $x'_m = [x', y', 1]^\top$ (en la segunda imagen).

Esta enorme **disparidad de órdenes de magnitud** causa que la matriz de datos A posea un número de condición pésimo y esté extremadamente mal condicionada numéricamente. Cualquier leve ruido de medición de SIFT en los píxeles desviará catastróficamente el cálculo de mínimos cuadrados.

Observación. Aqui se nombra el algoritmo SIFT, puesto que las parejas que usaremos para construir la matriz A las extraemos de los descriptores SIFT, que aunque son robustos, no son perfectos y contienen un porcentaje de correspondencias erróneas (outliers) que también afectan a la calidad de la estimación de F. Esto último lo solucionaremos usando RANSAC (se explica más adelante).

15.3.3. El Algoritmo de los 8 Puntos Normalizado de Hartley

Para solventar el mal condicionamiento numérico, Richard Hartley propuso en 1995 un pipeline de precondicionamiento obligatorio estructurado en los siguientes pasos secuenciales:

1. **Normalización de Coordenadas:** Antes de construir la matriz A, se calculan de forma independiente dos matrices de transformación afín T y T' para la imagen izquierda y derecha, respectivamente. Estas transformaciones aplican un cambio de escala y traslación sobre los puntos de modo que cumplan dos condiciones estadísticas:
    - El centroide de masa (media geométrica) de todos los puntos de la imagen se desplaza de forma exacta hacia el origen de coordenadas cartesianas $(0, 0)$.
    - Los vectores de píxeles se escalan de forma homogénea para que la distancia media cuadrática desde el origen de coordenadas recalculado hacia los puntos normalizados sea exactamente igual a $\sqrt{2}$ píxeles (logrando que el rango promedio de los datos oscile estrictamente en el intervalo $[-1, 1]$).

Los nuevos puntos normalizados se computan como $\tilde{\mathbf{x}}_m = T\mathbf{x}_m$ y $\tilde{\mathbf{x}}'_m = T'\mathbf{x}'_m$.

2. **Estimación Lineal:** Se construye la matriz del sistema utilizando los puntos normalizados $\{\tilde{\mathbf{x}}_m, \tilde{\mathbf{x}}'_m\}$, se calcula su SVD y se extrae la matriz fundamental normalizada intermedia, denotada como $\tilde{F}$.

3. **Imposición de la Restricción de Rango 2 (Rank Enforcing):** La matriz numérica calculada $\tilde{F}$ en el paso anterior no poseerá rango igual a 2 de forma perfecta debido al ruido analítico presente en las mediciones ($\det(\tilde{F}) \neq 0$). Para forzar la consistencia epipolar, calculamos su SVD interna:

$$\tilde{F} = U\Sigma V^\top = U \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix} V^\top$$

   Donde los valores singulares están ordenados en orden descendente ($\sigma_1 \geq \sigma_2 \geq \sigma_3$). Se aplica el teorema de Eckart-Young eliminando el menor valor singular de la diagonal (forzando $\sigma_3 = 0$) para construir una nueva matriz diagonal corregida Σ':

$$\Sigma' = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

   Se reconstruye la matriz fundamental normalizada de rango corregido mediante la multiplicación:

$$\tilde{F}_{\text{coaccionada}} = U\Sigma' V^\top$$

4. **Des-normalización Final al Espacio de Unidades Original:** Para recuperar la matriz aplicable sobre las coordenadas de píxeles reales de la foto inicial, se deshace algebraicamente el precondicionamiento aplicando las transformaciones transpuestas inversas de Hartley:

$$\boxed{F = T'^\top \tilde{F}_{\text{coaccionada}} T}$$

15.3.4. Uso de RANSAC en la Estimación Práctica

En entornos prácticos y reales de ingeniería, las correspondencias extraídas mediante descriptores como SIFT contienen un porcentaje elevado de emparejamientos erróneos (llamados analíticamente *outliers*) debido a texturas repetitivas o cambios bruscos de iluminación.

Dado que el criterio de mínimos cuadrados del algoritmo SVD es extremadamente sensible a datos erróneos, se integra de forma mandatoria el algoritmo robusto **RANSAC (Random Sample Consensus)**. RANSAC toma repetidamente subconjuntos aleatorios de solo 8 parejas de puntos, estima una matriz candidata F, y cuenta cuántos puntos del conjunto global la validan actuando como *inliers*. Al finalizar las iteraciones, se descartan los *outliers* y se calcula la matriz fundamental definitiva mediante el algoritmo normalizado utilizando únicamente el conjunto purificado de correspondencias correctas.