Alina Carías, Daniel Machic y Ariela Mishaan
Visión por Computadora

# Hoja de Trabajo 1

**Repositorio en GitHub: https://github.com/ArielaMishaanCohen/HT1.git**

**Task 1 – Análisis Teórico**

1. **Como director de un proyecto de conducción autónoma, debe dimensionar el hardware para un nuevo vehículo. El sistema utiliza 8 cámaras que capturan video a resolución 4K UHD (3840 x 2160). Debido a la necesidad de alto rango dinámico (HDR), los sensores operan a 12 bits por píxel (Raw Bayer Pattern) a 60 FPS. Métrica A: Enfocada puramente en el flujo vehicular.**

- Número de cámaras: 8
- Resolución: 4K UHD = 3840 x 2160 pixeles
- Bits por pixel: 12 bits
- Formato: Raw. Bayer (1 valor por pixel, no RGB completo)
- Frame rate: 60 FPS
- Sin compresión

**a. Calcule el tamaño exacto de una sola imagen (frame) cruda en Megabytes (MB).**

1) Número total de pixeles por imagen: $3840 \cdot 2160 = 8\,294\,400$ pixeles.
2) Bits totales por imagen (cada pixel usa 12 bits): $8\,294\,400 \cdot 12 = 99\,532\,800$ bits.
3) Bits → bytes: $\frac{99\,532\,800}{8} = 12\,441\,600$ bytes
4) MB binarios (1MB = $1024^2$ bytes) → $\frac{12\,441\,600}{1024^2} = \mathbf{11.87\ MB}$

→ El tamaño exacto de una imagen cruda es de 11.87 MB.

**b. Calcule el ancho de banda necesario (en Gbps) para transmitir el flujo de las 8 cámaras al procesador central sin compresión.**

1) Datos por segundo de una cámara: $\frac{11.87\ MB}{frame} \cdot 60\ FPS = 712.2\,\frac{MB}{s}$
2) Para las 8 cámaras: $712.2 \cdot 8 = 5697.6\,\frac{MB}{s}$
3) Conversión MB/s → Gbps
   - 1 byte = 8bits
   - 1 GB = 1024 MB

$$5697.6 \cdot 8 = 45\,580.8\,\frac{MB}{s}$$

$$\frac{45\ 580.8}{1024} = \mathbf{44.5\ Gbps}$$

→ Se requieren aproximadamente 44.5 Gbps de ancho de banda sin compresión.

**c. Si su procesador tiene una memoria RAM reservada de 16 GB exclusivamente para el buffer de video, ¿cuántos segundos de historia puede almacenar antes de empezar a sobrescribir datos?**

1) Memoria en MB: $16\ GB = 16 \cdot 1024 = 14\ 384\ MB$
2) Datos consumidos por segundo: $5697.6\ \frac{MB}{s}$ (inciso b, paso 2)
3) Tiempo máximo antes de sobreescritura $\frac{16384}{5697.6} = \mathbf{2.87\ segundos}$

→ El sistema solo puede almacenar aproximadamente 2.9 segundos de video antes de sobrescribir.

**d. Basado en su resultado, ¿es viable enviar estos datos "crudos" a la nube en tiempo real usando 5G? Justifique**

- 5G teórico máximo: ~20 Gbps (Flinders, Smalley, s.f.)[1]
- 5G realista sostenido: 1-5 Gbps
- Sistema requiere: ~44.5 Gbps

→ No es viable enviar datos crudos en tiempo real por 5G. El ancho de banda requerido supera ampliamente las capacidades reales del 5G.

**2. Considere un píxel con valor de intensidad $I_{in} = 50$ en una imagen estándar de 8 bits (0–255). Se aplican dos procesos de mejora secuenciales en el siguiente orden:**

a. Corrección Gamma con γ = 0.5 (para expandir sombras).
b. Ajuste Lineal con ganancia α = 1.2 y brillo β= -10 (para contrastar).

Realice los cálculos en el dominio de flotantes normalizados [0,1] como dicta la buena práctica y convierta a entero de 8 bits solo al final.

- Imagen 8 bits → rango [0, 255]
- Valor inicial del pixel: $I_{in} = 50$
- Ajuste lineal: $I_{out} = \alpha I + \beta$
- $\alpha = 1.2, \beta = -10$

[1] https://www.ibm.com/es-es/think/topics/5g

**a. Calcule el valor final del píxel I_out.**

1) Normalizar el pixel a [0,1]: $I_{norm} = \frac{50}{255} \sim 0.1961$
2) Corrección Gamma:

$$I_\gamma = I_{norm}^{\gamma} \rightarrow I_\gamma = (0.1961)^{\frac{1}{2}} = \sqrt{0.1961} \sim \mathbf{0.4429}$$

   - Interpretación: $\gamma < 1$ expande sombras. El valor aumenta significativamente.

3) Aplicación del ajuste lineal: $I_{in} = \alpha I_\gamma + \beta_{norm}$

$$\beta_{norm} = -\frac{10}{255} \sim -0.0392$$

$$I_{in} = 1.2 \cdot 0.4429 - 0.0392 = 0.5315 - 0.0392 = \mathbf{0.4923}$$

4) Verificar saturación (clipping): el rango válido es $0 \leq I \leq 1$. Como $I_{in} = 0.4923$, no hay clipping.

5) Convertir de vuelta a 8 bits

$$I_{out} = 0.4923 \cdot 255 \sim 125.53 \rightarrow \boldsymbol{I_{out} = 126}$$

**b. ¿Hubo saturación (clipping) en el proceso?**
El rango válido de $I_{in}$ es $0 \leq I \leq 1$. Como $I_{in} = 0.4923$ no es necesario el clipping.

**c. Si hubiéramos realizado las operaciones usando uint8 directamente sin convertir a float (truncando decimales en cada paso intermedio), ¿cuál habría sido el error numérico resultante?**

1) Gamma en uint8 (raíz del pixel normalizado): $\sqrt{0.1961} \cdot 255 = 112.93 \rightarrow 112$
2) Ajuste lineal en uint8: $I = 1.2 \cdot 112 - 10 = 134.4 - 10 = 124.4 \rightarrow \mathbf{124}$

→ el resultado correcto es 125, el resultado con uint8 es 124. El error numérico es de **2.**

**3. Usted está programando un robot clasificador de pelotas. Tiene dos objetos: una pelota roja brillante bajo el sol $\boldsymbol{R_{rgb} = (255, 0, 0)}$ y la misma pelota roja en una sombra profunda $\boldsymbol{S_{rgb} = (50, 0, 0)}$.**

**a. Calcule la distancia entre estos dos colores en el espacio RGB.**

Distancia euclidiana en RGB:

$$d = \sqrt{(R_1 - R_2)^2 + (G_1 - G_2)^2 + (B_1 - B_2)^2} = \sqrt{(255 - 50)^2 + (0 - 0)^2 + (0 - 0)^2}$$
$$= \sqrt{205^2} = \mathbf{205}$$

**b. Convierta ambos colores al espacio HSV (asuma rangos normalizados H ∈ [0,1], S ∈ [0,1], V ∈ [0,1] para simplificar, sabiendo que el Hue del rojo es 0).**

Color 1: Rojo brillante (255,0,0)

1) Normalización RGB: (1,0,0)
2) $V = \max = 1\,,\ S = \frac{V - \min}{V} = 1,$ Hue del rojo = 0

$$\rightarrow \boldsymbol{HSV_R} = (\mathbf{0}, \mathbf{1}, \mathbf{1})$$

Color 2: Rojo en sombra (50,0,0)

1) Normalización RGB: $\left(\frac{50}{255}, 0{,}0\right) = (0.196, 0{,}0)$
2) $V = 0.196, S = 1$ (rojo puro), Hue = 0.

$$\rightarrow \boldsymbol{HSV_s} = (\mathbf{0}, \mathbf{1}, \mathbf{0.196})$$

**c. Calcule la diferencia absoluta canal por canal en HSV**

$$\Delta H = |0 - 0| = 0$$
$$\Delta S = |1 - 1| = 0$$
$$\Delta|1 - 0.196| - 0.804$$

$$\rightarrow (\boldsymbol{\Delta H}, \boldsymbol{\Delta S}, \boldsymbol{\Delta V}) = (\mathbf{0}, \mathbf{0}, \mathbf{0.804})$$

**d. Argumente matemáticamente por qué un algoritmo de agrupación (clustering) simple fallaría en RGB pero funcionaría en HSV para determinar que ambos píxeles pertenecen al mismo objeto "pelota roja".**

En RGB, la iluminación afecta directamente los valores RGB y la diferencia entre ellos es drástica. La diferencia entre la pelota al sol y la pelota en la sombra causa que se vean como colores distintos:

$$\|R_{sol} - R_{sombra}\| = 205$$

Un algoritmo de clustering, como k-means, minimiza distancias. Entonces, si se utilizara RGB, el algoritmo estaría clasificando dos clusters separados, confundiendo la iluminación con color.

En HSV, el Hue (H) representa el color real. La iluminación solo afecta V, y la saturación permaneces alta. Por lo tanto, un clustering en (H,S) ignoraría las variaciones de brillo.

→ RGB falla porque la distancia está dominada por cambios de iluminación, HSV funciona porque el Hue permanece constante para el mismo objeto.

**Task 2 – Práctica**

Su objetivo es implementar un pipeline de pre-procesamiento manual, manipulando tensores y gestionando tipos de datos (uint8 vs float32) sin utilizar funciones de "caja negra" para la matemática. Puede utilizar el esqueleto de código lab_semana1.py que se adjuntó en el portal. Se permite el uso de numpy, opencv-python (solo para I/O y conversiones de espacio de color), matplotlib.

Ver código de python en repositorio.

**Task 3 – Preguntas Post-Práctica**

1. **En la diapositiva 15 se mencionó que "Iterar píxel a píxel en Python es un Pecado Capital". Explique en términos de gestión de memoria y CPU por qué una operación vectorizada en NumPy es órdenes de magnitud más rápida que un for loop.**

   Iterar píxel a píxel en Python es considerado un “pecado capital” debido al alto overhead computacional del intérprete, ya que cada iteración del for loop implica verificaciones dinámicas y múltiples llamadas internas, lo cual es muy ineficiente al procesar imágenes con millones de píxeles. En cambio, las operaciones vectorizadas de NumPy se ejecutan en código compilado en C/C++, trabajan sobre bloques contiguos de memoria y aprovechan instrucciones SIMD, optimizando el uso de la caché y reduciendo drásticamente el número de instrucciones ejecutadas. Por ello, la vectorización es órdenes de magnitud más rápida que la iteración explícita en Python.

2. **Al visualizar imágenes con matplotlib, ¿qué sucede si olvida que OpenCV carga las imágenes en formato BGR? ¿Cómo se ve visualmente el error?**

   Lo que ocurre es una interpretación incorrecta de los canales de color, Matplotlib asume que las imágenes están en formato RGB, por lo que al mostrar directamente una imagen cargada con OpenCV, los canales rojo y azul quedan intercambiados. Esto no altera la información espacial de la imagen, pero sí su representación cromática.

3. **Al visualizar imágenes con matplotlib, ¿qué sucede si olvida que OpenCV carga las imágenes en formato BGR? ¿Cómo se ve visualmente el error?**

   Visualmente este error se puede ver ya que los colores se muestran invertidos o poco realistas. Aunque la imagen mantiene su forma y estructura, la distorsión de color puede llevar a interpretaciones erróneas durante el análisis visual.