

Laboratorio

Procesamiento de Imágenes

Visión por Ordenador I
Ingeniería Matemática

Universidad Pontificia Comillas, ICAI



Laboratorio

Procesamiento de Imágenes

Profesor:

Erik Velasco	Email
Lionel Güitta	evelasco@icai.comillas.edu
Daniel Pinilla	lglopez@icai.comillas.edu
Luis Arias	dpinilla@icai.comillas.edu
Rodrigo Sánchez	learias@icai.comillas.edu
Ignacio de Rodrigo	rsmolina@icai.comillas.edu
	iderodrigo@comillas.edu

Cover: Erosion, tectonic uplift, and a human-built dam have all helped shape the Upper Lake Powell area in Utah. Image Credit: NASA

Contents

1 Sesión 2: Procesamiento de Imágenes	1
1.1 Materiales	1
1.2 Apartados de la práctica	1
1.3 Observaciones	1
1.4 Qué va a aprender	1
1.5 Evaluación	2
2 Apartado A: Segmentación por color	3
Tarea A.1: Carga de imágenes	3
Tarea A.2: Visualización y guardado de imágenes	3
Tarea A.3: Cambio de espacio de color	3
Tarea A.4: Segmentación de las partes naranjas	4
Tarea A.5: Segmentación de las partes blancas	4
Tarea A.6: Combinación de máscaras	4
Tarea A.7: Guardado de imágenes	5
Preguntas	5
3 Apartado B: Filtro Gaussiano y Detección de bordes: Sobel y Canny	6
Tarea B.1: Implementación de filtro Gausiano	6
Tarea B.2: Aplicación de filtro Gausiano	7
Tarea B.3: Implementación de filtro Sobel	7
Tarea B.4: Aplicación de filtro Sobel	7
Tarea B.5: Implementación de filtro Canny	8
Tarea B.6: Aplicación de filtro Canny	8
Preguntas	8
4 Apartado C: Operadores morfológicos	9

Sesión 2: Procesamiento de Imágenes

1.1. Materiales

En esta práctica se trabajará con los siguientes recursos (puede encontrarlos en la sección de Moodle *Laboratorio/Sesión 2*):

- **lab_2.ipynb**: notebook con el código que deberá completar.
- **data**: carpeta con imágenes para trabajar durante la práctica.

1.2. Apartados de la práctica

La Sesión 2 del laboratorio está dividida en los siguientes apartados:

- Instalaciones: Instalación de librerías necesarias.
- Librerías: Importación de las librerías que se utilizan en la sesión. Se recomienda realizar la importación en una celda inicial para mantener la organización del Notebook.
- Apartado A: Segmentación de imágenes por color.
- Apartado B: Implementación del filtro Gaussiano y detección de bordes con los métodos Sobel y Canny.
- Apartado C: Operadores Morfológicos.

1.3. Observaciones

Aunque el guion de la práctica y los comentarios en Markdown del Notebook estén escritos en español, observe que todo aquello que aparece en las celdas de código está escrito en inglés. Es una buena práctica que todo su código esté escrito en inglés.

Aquellas partes del código que deberá completar están marcadas con la etiqueta **TODO**.

Es muy importante que trabaje consultando la documentación de OpenCV¹ y *skimage*² para familiarizarse de cara al examen. Tenga en cuenta que en los exámenes no podrá utilizar herramientas de ayuda como Copilot.

1.4. Qué va a aprender

Al finalizar esta práctica, sabrá cómo trabajar con diferentes espacios de color para la segmentación de imágenes, implementará filtros Gaussianos, y aplicará métodos de detección de bordes como Sobel y Canny. Además, comparará sus implementaciones con las funciones predefinidas en *skimage*. También utilizará operadores morfológicos y comprenderá su utilidad.

¹Documentación de OpenCV: <https://docs.opencv.org/4.x/index.html>

²Documentación de skimage: <https://scikit-image.org/docs/stable/>

1.5. Evaluación

La nota que obtenga en esta sesión de laboratorio será la misma que obtenga su pareja. Los apartados de la práctica serán evaluados como refleja la Tabla 1.1. Tenga en cuenta que en esta práctica puede obtener puntuación extra si realiza el Apartado dedicado a *Operadores Morfológicos*

Tarea	Valor	Resultado
Pregunta A.1	3.0	
Pregunta B.1	3.5	
Pregunta B.2	3.5	
Pregunta C.1	2.0	
Total	12.0	

Table 1.1: Valoración de los apartados de la práctica.

2

Apartado A: Segmentación por color

En este apartado deberá segmentar los colores blancos y naranjas que aparecen en las imágenes disponibles en la carpeta data. Segmentar por colores es una técnica sencilla y útil en aplicaciones en las que se tiene un gran control sobre las condiciones de contorno: iluminación, tipo de objeto que se espera encontrar, color del fondo, etc. A continuación se añaden algunos comentarios para resolver paso a paso la tarea:

Tarea A.1: Carga de imágenes

Para trabajar de forma eficiente, antes de empezar a procesar imágenes deberá definir algunos métodos auxiliares. El primero de ellos es un método que debe cargar imágenes de la carpeta data.

Tarea A.2: Visualización y guardado de imágenes

De nuevo, estos métodos ya los utilizó en la 1^a sesión de laboratorio, así que aproveche este paso para afianzar conceptos. Defina los métodos para visualizar imágenes y guardarlas. Respete y utilice todos los argumentos que se proporcionan.

Tarea A.3: Cambio de espacio de color

Cambie el espacio de color de las imágenes (BGR) a uno donde la crominancia e intensidad estén separados (HSV). Puede interpretar este paso como un cambio de sistema de referencia (cartesiano a polar). De esta forma, le será más fácil discriminar los colores de interés. La Figura 2.1 ¹ muestra los espacios de colores con los que trabajará.

Para cambiar el espacio de color de sus imágenes, puede utilizar el método `cv2.cvtColor()` de OpenCV.

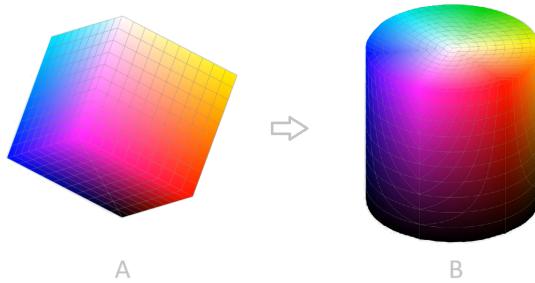


Figure 2.1: Espacios de color utilizados. Gráfico A: BGR (atención con el orden de canales en OpenCV) y Gráfico B: HSV.

¹Imagen modificada de VerbaGleb, CC BY-SA 4.0 https://commons.wikimedia.org/wiki/File:RGB_2_HSV_conversion_with_grid.ogg

Tarea A.4: Segmentación de las partes naranjas

Genere una máscara para cada imagen y realice una segmentación con ella. Una máscara es una imagen binaria (blanca y/o negra) donde los píxeles de interés aparecen en blanco, mientras que los píxeles que no son de interés aparecen en negro. Segmentar implica multiplicar la imagen original con la máscara. Así se obtienen los píxeles de interés de la imagen original.

Para obtener las máscaras y segmentar la imagen, puede utilizar respectivamente los métodos `cv2.inRange()` y `cv2.bitwise_and()` de OpenCV. Por otro lado, en este paso se facilitan los rangos de color naranja:

```
light_orange = (1, 190, 200)
dark_orange = (18, 255, 255)
```

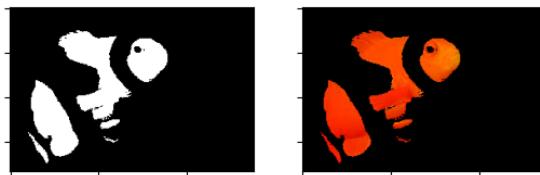


Figure 2.2: Máscara y segmentación de colores naranjas.

Tarea A.5: Segmentación de las partes blancas

En este paso deberá repetir un proceso similar al anterior, pero centrándose en los tonos blancos. Explore el rango de valores necesarios para segmentar los colores blancos de las imágenes.

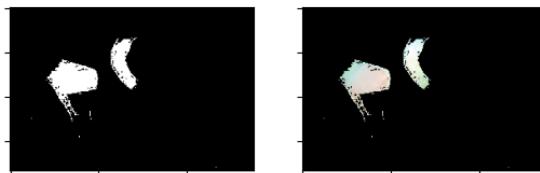


Figure 2.3: Máscara y segmentación de colores blancos.

Tarea A.6: Combinación de máscaras

Combine las máscaras de los colores naranja y blanco para cada imagen. Posteriormente, segmente cada imagen con la máscara que ha generado. Como resultado, conseguirá segmentar el cuerpo de los peces en el conjunto imágenes. Debería obtener resultados parecidos al ejemplo de la Figura 2.4.

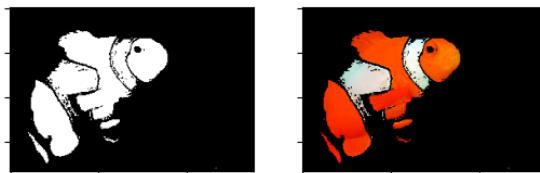


Figure 2.4: Máscara completa y segmentación del pez como combinación de colores blancos y naranjas.

Tarea A.7: Guardado de imágenes

Guarde las imágenes de interés como resultado de esta parte de la práctica.

Preguntas

Pregunta A.1

Segmenta por color el escudo de su equipo deportivo favorito: descompóngalo en al menos 2 colores. De igual forma que ha realizado en el ejercicio de clase, proporcione las máscaras binarias de cada color. Además, multiplique la máscara con la imagen original para visualizar solamente el color de interés. Por último indique el porcentaje que representa cada color sobre el total del escudo.

3

Apartado B: **Filtro Gaussiano y Detección de bordes: Sobel y Canny**

En este apartado, trabajará con filtros para procesar imágenes. Los conceptos clave de este apartado son:

- **Filtro o Kernel:** es una matriz $n \times n$ que se desliza de forma ordenada por la imagen. Segundo sus valores, un filtro puede servir para difuminar una imagen, extraer sus bordes, etc.
- **Convolución.** Es una operación matemática que en visión por ordenador sirve para modificar una imagen. Toma los valores del kernel y los de la imagen en la posición (i, j) para producir el valor del píxel (i, j) de la nueva imagen.¹
- **Imagen:** El elemento sobre el que se aplica la operación.

Trabajará en la definición de tres filtros: Gausiano, Sobel y Canny. Una vez definidos, deberá aplicar cada uno de los filtros sobre las imágenes de la carpeta data. A continuación se añaden algunos comentarios para resolver la tarea:

Tarea B.1: Implementación de filtro Gausiano

Para aplicar el filtro Gausiano, deberá tener cuenta los siguientes factores:

- **Tamaño del filtro.** Como podrá comprobar en los argumentos de la función, el tamaño del filtro puede (o no) venir dado. Cuando el tamaño del filtro no se pase como argumento, deberá obtener sus dimensiones a partir del valor de σ . Puede dar por supuesto que el efecto del filtro Gausiano se considera extinto más allá de $\pm 4\sigma$. Además, tenga en cuenta que se busca un filtro $n \times n$ donde n es impar. La Figura 3.1 ilustra cómo aumenta el tamaño del filtro (*kernel size*, K) cuando aumenta σ .

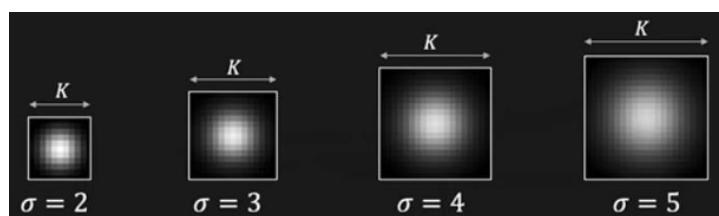


Figure 3.1: Tamaño del kernel en función de σ .

Con las restricciones y condiciones mencionadas, deberá definir el tamaño del filtro de tal forma que se cumpla lo indicado en la Ecuación 3.2:

¹Ejemplos animados de convoluciones: <https://gitlab.com/brohrer/convolution-2d-animation/-/tree/main>

$$\text{shape} = [L, L] \quad (3.1)$$

$$L = f(\sigma) \quad (3.2)$$

- **Coordenadas.** Con `np.mgrid()`, obtenga los valores de las coordenadas para x e y como dos arrays diferentes. Visite la documentación del método para visualizar un ejemplo de lo que se le pide.
- **Fórmula.** Defina la fórmula del filtro Gausiano para cada elemento (i, j) siguiendo la Ecuación 3.3.

$$G(i, j) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{i^2+j^2}{\sigma^2}\right)} \quad (3.3)$$

- **Convolución.** Aplique sobre el argumento `img` la operación de convolución con el filtro que acaba de definir. Utilice el método de OpenCV `cv2.Filter2D()`.

Tarea B.2: Aplicación de filtro Gausiano

Aplique el filtro que acaba de definir al conjunto de imágenes de la carpeta `data`. Defina las variables necesarias para llamar al método. Puede utilizar una *list comprehension* para trabajar de forma eficiente. Como resultado de aplicar el filtro, debería obtener imágenes difuminadas o desenfocadas.

Tarea B.3: Implementación de filtro Sobel

Para aplicar el filtro Sobel, deberá tener cuenta los siguientes factores:

- **Imagen en escala de grises.** Trabaje con la imagen en escala de grises. Para ello, puede usar el método `cv2.cvtColor()` (que ya utilizó en la primera sesión de laboratorio).
- **Filtro Gausiano.** Utilice el método que ha definido anteriormente. Sea cuidadoso proporcionando cada argumento para que sea heredable (es decir, utilice variables como argumentos).
- **Bordes verticales.** Obtenga los bordes verticales utilizando el filtro dado (`filter`). De nuevo, utilice el método `cv2.Filter2D()`.
- **Transformación del filtro.** Transforme el filtro para obtener los bordes en la dirección ortogonal.
- **Bordes horizontales.** Con el filtro transformado, utilice de nuevo `cv2.Filter2D()` para obtener los bordes ortogonales.
- **Composición de bordes.** Obtenga los bordes de la imagen como una composición de los bordes detectados en cada dirección. Puede utilizar para ello `np.hypot()`.
- **Normalización.** Normalice el resultado anterior para evitar valores fuera de rango y ruidos.
- **Orientación.** Obtenga la orientación de los bordes con `np.arctan2()`.

Tarea B.4: Aplicación de filtro Sobel

Aplique el filtro que acaba de definir al conjunto de imágenes de la carpeta `data`. Defina las variables necesarias para llamar al método. Puede utilizar una *list comprehension* para trabajar de forma eficiente. Como resultado de aplicar el filtro, debería obtener un resultado similar al de la Figura 3.2.



Figure 3.2: Ejemplo de bordes detectados usando el filtro Sobel.

Tarea B.5: Implementación de filtro Canny

El filtro Canny aporta un resultado más refinado para detectar bordes que el obtenido por Sobel. Para implementar el filtro Canny, deberá seguir los siguientes pasos:

- **Filtro Sobel.** Realice una llamada al método del filtro Sobel que ha definido en los pasos anteriores. De nuevo, sea cuidadoso proporcionando cada argumento para que sea heredable (es decir, utilice variables como argumentos).
- **Non Max Supresion.** Utilice el método `non_max_supression()` definido en el archivo `utils.py`. Inspeccione los argumentos que necesita utilizar este método para poder ejecutarlo de forma adecuada. Este método se encarga de refinar los bordes, devolviendo una imagen en escala de grises.

Tarea B.6: Aplicación de filtro Canny

Aplique el filtro que acaba de definir al conjunto de imágenes de la carpeta `data`. Defina las variables necesarias para llamar al método. Puede utilizar una *list comprehension* para trabajar de forma eficiente. Como resultado de aplicar el filtro, debería obtener los bordes de las imágenes de forma similar a cuando aplicó el filtro Sobel.

Preguntas

Pregunta B.1

Añada ruido a las imágenes de la carpeta `data`. Compare los resultados que obtiene al aplicar su filtro Sobel **con** y **sin filtro Gausiano**. Para añadir ruido, puede sumar a la imagen original una matriz de las dimensiones adecuadas. Puede utilizar `np.random.normal()`, pero tenga en cuenta que este proceso puede hacer que algunos valores de la imagen salgan del rango [0, 255] (tome las medidas necesarias para corregir esto).

Pregunta B.2

Utilice la librería `scikit-image` y compare el efecto de los filtros Sobel, Canny y Prewitt sobre las imágenes de la carpeta `data`. ¿Qué diferencias observa entre los filtros? ¿Puede obtener alguna conclusión y/o patrón?

4

Apartado C: Operadores morfológicos

Esta sección es completamente voluntaria y puede realizarla para mejorar su puntuación en la práctica.

Pregunta C.1

Implemente los operadores morfológicos dilatación y erosión. Utilice para ello un kernel de 3x3 con valores 255 (blanco) como structuring element. No olvide binarizar su imagen antes de aplicar los operadores morfológicos.