# Práctica 5: Detección de características

**Integrantes: Alejandro Bolaños García y David García Díaz**

## Explicación del código en python

**Importación de bibliotecas:** Bibliotecas utilizadas, incluyen `cv2` (OpenCV), `numpy`, `tkinter`, `Image` de PIL y `pyplot` de Matplotlib.

**Variables globales:** Se definen varias variables globales que se utilizarán para controlar la imagen y las transformaciones. Estas variables incluyen `drawing` para el estado del dibujo de la selección de área, `ix` e `iy` para las coordenadas iniciales de la selección, `fx` y `fy` para las coordenadas finales de la selección, `img` para la imagen original, `cropped_image` para la imagen recortada seleccionada, `transformed_image` para la imagen transformada, y `w` y `h` para el ancho y alto de la imagen original.

**Función `update_transformation()`:** Esta función se encarga de actualizar todas las transformaciones a la vez y ajustar la ventana de visualización. Primero verifica si la imagen está cargada. Luego, escala la imagen utilizando la función `cv.resize()` de OpenCV. A continuación, crea una matriz de transformación para la rotación y traslación utilizando la función `cv.getRotationMatrix2D()`. Después, aplica la rotación y traslación a la imagen escalada utilizando la función `cv.warpAffine()`. Finalmente, ajusta el tamaño de la ventana de visualización y muestra la imagen transformada utilizando las funciones `cv.namedWindow()`, `cv.resizeWindow()` y `cv.imshow()` de OpenCV.

**Función `draw_figures()`:** Esta función es una función de callback para el ratón que permite seleccionar una región en la imagen. Cuando se presiona el botón izquierdo del ratón, se establecen las coordenadas iniciales de la selección. Mientras se mueve el ratón, se dibuja un rectángulo en la selección. Al soltar el botón izquierdo, se establecen las coordenadas finales de la selección y se llama a la función `crop_image()` para recortar la imagen en la región seleccionada.

**Función `crop_image()`:** Esta función recorta la región seleccionada de la imagen utilizando las coordenadas iniciales y finales de la selección.

**Función `sift_matches()`:** Esta función realiza el emparejamiento de características SIFT entre la región seleccionada y la imagen transformada. Primero verifica si la región seleccionada y la imagen transformada existen. Luego, convierte las imágenes a escala de grises utilizando la función `cv.cvtColor()` de OpenCV. A continuación, obtiene los valores de los parámetros de SIFT desde los sliders de la interfaz gráfica. Después, detecta y calcula las características SIFT en ambas imágenes utilizando la función `sift.detectAndCompute()` de OpenCV. Luego, realiza el emparejamiento de características utilizando el algoritmo BFMatcher y dibuja los mejores `matches` utilizando la función `cv.drawMatches()` de OpenCV. Finalmente, muestra los `matches` en una ventana de OpenCV y en Matplotlib utilizando las funciones `cv.imshow()` y `plt.imshow()` respectivamente.

**Función `open_file()`:** Esta función se encarga de abrir y preprocesar una imagen desde un archivo. Utiliza la función `filedialog.askopenfilename()` de Tkinter para abrir un cuadro de diálogo de selección de archivo. Luego, carga la imagen utilizando la función `Image.open()` de PIL y la convierte a un array NumPy utilizando `np.array()`. A continuación, convierte la imagen a escala de grises, aplica mejoras de contraste y suavizado utilizando las funciones de OpenCV, y muestra la imagen en una ventana de OpenCV utilizando `cv.imshow()`.

**Funciones de callback de los sliders:** Estas funciones se encargan de actualizar las transformaciones cuando se cambian los valores de los sliders de la interfaz gráfica. Cada función actualiza una variable global correspondiente a una transformación y luego llama a la función `update_transformation()` para aplicar todas las transformaciones.

**Configuración de la interfaz de Tkinter:** Se crea una ventana de Tkinter y se configuran los elementos de la interfaz gráfica, incluyendo etiquetas, sliders y botones.

## Conclusión

Durante el uso del algoritmo SIFT, observamos que al aplicar escalas grandes a las imágenes, a menudo no se lograban capturar todas las características de forma precisa. Esto parece estar relacionado tanto con la calidad de las imágenes como con las transformaciones aplicadas, que afectaban la precisión del detector. Además, intentamos comparar una región de interés seleccionada en una imagen con otra imagen distinta. En este caso, seleccionamos imágenes de distintos casos de derrames cerebrales, extrayendo la región donde se había producido el derrame y comparándola con otras imágenes para verificar si el algoritmo podría detectar un derrame en ellas. Sin embargo, este enfoque no resultó exitoso, ya que no se lograba detectar el derrame de manera efectiva en las imágenes comparadas, por lo que decidimos descartar esta parte de la implementación.