Actividad_2_grupal

February 1, 2021

0.1 Caso grupal: Implementación de un filtro espacial o morfológico

Autores: Mikel Aldalur Corta, David Caviedes Velasco, Elena Murga Martínez y Rubén Rodríguez Hernández (equipo 18, grupo 1)

Esta práctica está inspirada en las herramientas de Inteligencia Artifical capaces de cambiar el fondo de un vídeo, especialmente utilizados en videollamadas, por ejemplo. En este Notebook se consigue, partiendo de una imagen con un objeto sobre un chroma de fondo seleccionar mediante combinación de filtros espaciales y morfológicos el área de la imagen correspondiente al objeto [**1**]. Posteriormente el objeto se sitúa sobre un fondo completamente diferente.

La línea de ejecución que sigue el software es la siguiente: 1. Después de leer la imagen con el objeto se aplica un filtro de realce de contraste. 2. Se aplica Gaussian Blur: consiste en un filtro Gaussiano que actúa como filtro paso bajo, eliminando las componentes de alta frecuencia de la imagen. 3. Utilizamos un filtro de detección de bordes. Aunque se han probado tanto el filtro Laplaciano como el de Sobel, se ha comprobado que el que mejores resultados da en este caso es el de Canny, que es bueno eliminando puntos aislados y uniendo bordes. 4. Se aplica el filtro morfológico de cierre para terminar de unir los bordes [**2**] y eliminar el ruido. Se utiliza un kernel de 7x7 en forma hexagonal (lo más similar al círculo) con el fin de redondear los bordes. 5. Se obtienen los contornos en la imagen y se dibuja el contorno sobre la imagen original con su área interior rellenada en color rosa. 6. Partiendo de esta imágen se crean las dos máscaras que se utilizan en la imagen del objeto y en la de fondo. En primer lugar se convierte la imagen a escala de grises y después se binariza, poniendo un umbral establecido de manera experimental. * La imagen binaria de fondo negro y objeto blanco sirve como máscara para extraer el objeto de la imagen original. * La imagen binaria de fondo blanco y objeto negro sirve para "borrar" en la imagen de fondo los pixels sobre los que irá el objeto. Antes de aplicar la máscara es necesario redimensionar la imagen de fondo para que coincida con las dimensiones de la imagen de objeto. 7. Finalmente se hace la suma de las dos imágenes con las máscaras aplicadas y se obtiene una imagen con el objeto y el fondo cambiado.

Referencias [**1**] https://likegeeks.com/es/procesar-de-imagenes-en-python/amp/

[**2**] https://docs.opencv.org/master/d9/d61/tutorial_py_morphological_ops.html

[2]:
```
import numpy as np
import cv2
```

[3]:
```
def convert_rgb_to_gray(rgb_image,show):
    '''
    Convertir imagen RGB a escala de grises
```

```
    Parametros
    ----------
    rgb_image: np.array (imagen en RGB)
    show: Boolean (indicador de si mostrar imagen en pantalla)

    Returns
    -------
    gray_image: np.array (imagen en escala de grises)
    '''
    gray_image = cv2.cvtColor(rgb_image, cv2.COLOR_BGR2GRAY)
    if show:
        cv2.imshow("Gray Image",gray_image)
    return gray_image
```

[4]:
```
def gausiana(gray_img,show):
    '''
    Aplicar filtro Gaussian Blur a imagen

    Parametros
    ----------
    gray_img: np.array (imagen a filtrar)
    show: Boolean (indicador de si mostrar imagen en pantalla)

    Returns
    -------
    gaus_image: np.array (imagen filtrada)
    '''
    gaus_image = cv2.GaussianBlur(gray_img, (5,5), 2)
    if show:
        cv2.imshow("Gaussian Image",gaus_image)
    return gaus_image
```

[5]:
```
def contraste(gray_img,show):
    '''
    Aumento de contraste en imagen

    Parametros
    ----------
    gray_img: np.array (imagen a tratar)
    show: Boolean (indicador de si mostrar imagen en pantalla)

    Returns
    -------
    contrast_img: np.array (imagen con mayor contraste)
    '''
```

```
    contrast_img = cv2.addWeighted(gray_img, 2.5, np.zeros(gray_img.shape,␣
↪gray_img.dtype), 0, 0)
    if show:
        cv2.imshow("Contrast Image",contrast_img)
    return contrast_img
```

[6]:
```
def convert_gray_to_binary(gray_image, adaptive, show):
    '''
    Convertir de gris a binario a partir de un valor umbral (global o␣
↪adaptativo)

    Parametros
    ----------
    gray_image: np.array (imagen a convertir)
    adaptive: Boolean (indicador de si aplicar el theshold de forma adaptada)
    show: Boolean (indicador de si mostrar imagen en pantalla)

    Returns
    -------
    binary_image: np.array (imagen binaria)
    '''
    if adaptive:
        binary_image = cv2.adaptiveThreshold(gray_image, 255, cv2.
↪ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 181, 10)
    else:
        _,binary_image = cv2.threshold(gray_image,151,255,cv2.THRESH_BINARY_INV)
    if show:
        cv2.imshow("Binary Image", binary_image)
    return binary_image
```

[7]:
```
def inverte(imagen):
    '''
    Invertir la imagen

    Parametros
    ----------
    imagen: np.array (imagen a invertir)

    Returns
    -------
    imagen: np.array (imagen invertida)
    '''
    imagen = (255-imagen)
    cv2.imshow('Binario inv', imagen)
    return imagen
```

[8]:
```python
def lap_grad(image):
    '''
    Aplicar filtro Laplaciano

    Parametros
    ----------
    image: np.array (imagen a filtrar)

    Returns
    -------
    laplace: np.array (imagen filtrada)
    '''
    laplace = cv2.Laplacian(image,cv2.CV_32F)
    return laplace
```

[9]:
```python
def sob_grad(image):
    '''
    Aplicar filtro Sobel

    Parametros
    ----------
    image: np.array (imagen a filtrar)

    Returns
    -------
    mag: np.array (magnitud del gradiente)
    dir: np.array (dirección del gradiente)
    '''
    gx = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=5)
    gy = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=5)
    # Python Calculate gradient magnitude and direction ( in degrees )
    mag, angle = cv2.cartToPolar(gx, gy, angleInDegrees=True)
    return (mag, angle)
```

[10]:
```python
def canny_grad(image):
    '''
    Aplicar filtro Canny

    Parametros
    ----------
    image: np.array (imagen a filtrar)

    Returns
    -------
    edges: np.array (imagen filtrada)
    '''
    edges = cv2.Canny(image,100,200)
```

```
    return edges
```

[11]:
```
def gradient(val,img_grad):
    '''
    Selección y llamada al filtro para detectar bordes

    Parametros
    ----------
    val: string (tipo de filtro a utilizar)
    img_grad: np.array (imagen a filtrar)

    Returns
    -------
    mod: string (nombre del gradiente utilizado)
    grad: np.array (imagen filtrada)
    '''
    #img = np.float32(img_grad) / 255.0
    #Kalkulatu gradientea bi modutako batera

    if val == 'sob':
        mod = 'Gradiente Sobel'
        grad, angle = sob_grad(img_grad)
    elif val == 'lap':
        mod = 'Gradiente Laplace'
        grad = lap_grad(img_grad)
    elif val == 'can':
        mod = 'Gradiente Canny'
        grad = canny_grad(img_grad)
    cv2.imshow(mod,grad)
    return(mod,grad)
```

[12]:
```
def close(imagen):
    '''
    Aplicar filtro morfológico cierre

    Parametros
    ----------
    imagen: np.array (imagen a filtrar)

    Returns
    -------
    closing: np.array (imagen filtrada)
    '''
    kernel = np.
 ↪array([[0,0,1,1,1,0,0],[0,1,1,1,1,1,0],[1,1,1,1,1,1,1],[1,1,1,1,1,1,1],[1,1,1,1,1,1,1],[0,1
 ↪uint8)
#     erosion = cv2.erode(imagen,kernel,iterations = 1)
```

```
#     dilation = cv2.dilate(imagen,kernel,iterations = 1)
#     opening = cv2.morphologyEx(imagen, cv2.MORPH_OPEN, kernel)
    closing = cv2.morphologyEx(imagen, cv2.MORPH_CLOSE, kernel)
    cv2.imshow('close2', closing)
    return closing
```

[13]:
```
def getContours(binary_image):
    '''
    Obtener contornos

    Parametros
    ----------
    binary_image: np.array (imagen de trabajo)

    Returns
    -------
    contours: list (lista de coordenadas de contorno)
    '''
    contours, hierarchy = cv2.findContours(binary_image,
                                           cv2.RETR_EXTERNAL,
                                          cv2.CHAIN_APPROX_SIMPLE)
    return contours
```

[14]:
```
def draw_contours(image, contours, image_name):
    '''
    Dibujar contorno y rellenar área interior en imagen

    Parametros
    ----------
    image: np.array (imagen a tratar)
    contours: list (lista de coordenadas de contorno)
    image_name: string (nombre de la imagen)
    '''
    index = -1 #inguraketa guztiak esan nahi du
    #thickness = 2 #inguraketa marraren lodiera
    thickness = -1 #area guztia betetzeko
    color = (255, 0, 255) #marraztuko den kolorea definitzeko
    cv2.drawContours(image, contours, index, color, thickness)
    cv2.imshow(image_name,image)
```

[15]:
```
def redimensionar(img_size, img_resize):
    '''
    Redimensionar la imagen de fondo para adaptarse a la principal (objeto)

    Parametros
    ----------
    img_size: np.array (imagen con objeto --> principal)
```

```
    img_resize: np.array (imagen de fondo --> imagen a ajustar)

    Returns
    -------
    resized: np.array (imagen ajustada)
    '''
    width = int(img_size.shape[1])
    height = int(img_size.shape[0])
    dim = (width, height)
    # resize image
    resized = cv2.resize(img_resize, dim, interpolation = cv2.INTER_AREA)
    return resized
```

[16]:
```
def main(imagen_a_procesar, imagen_fondo):
    '''
    Función principal

    Parametros
    ----------
    imagen_a_procesar: string (directorio de imagen objeto)
    imagen_fondo: string (directorio de imagen fondo)
    '''
    # Leer la imagen del croma
    img = cv2.imread(imagen_a_procesar)

    # Aumentar los contrastes
    contrastes = contraste(img,True)

    # Aplicar Gaussian Blur
    gaussiana = gausiana(contrastes,True)

    # Calcular el gradiente (Canny)
    modo, fil_img_1 = gradient('can', gaussiana)

    # Aplicar filtro morfológico cierre
    closing_im = close(fil_img_1)

    #fil_img_1 = closing_im
    # Obtener y dibujar contornos
    contours = getContours(closing_im)
    draw_contours(img, contours, 'RGB Contours2')

    # Obtener la máscara binarizando la imagen de contorno
    gray = convert_rgb_to_gray(img, True)
    binary = convert_gray_to_binary(gray,False, True)

    # Lectura de imagen de objeto original
```

```
    imagen_final = cv2.imread(imagen_a_procesar)

    # Aplicar máscara a imagen original (seleccionar objeto)
    res = cv2.bitwise_and(imagen_final, imagen_final, mask=binary)

    # Invertir máscara, redimensionar imagen de fondo y aplicar máscara
    binary_inv = inverte(binary)
    fondo = cv2.imread(imagen_fondo)
    fondo = redimensionar(imagen_final, fondo)
    res_2 = cv2.bitwise_and(fondo, fondo, mask=binary_inv)

    # Combinación de ambas imágenes y muestra de resultado
    resultado_final = res + res_2
    cv2.imshow("Result",resultado_final)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

[17]:
```
if __name__ == '__main__':
    imagenes='zebra.jpg'
    #imagenes='person.jpg'
    fondo = 'fondo_1.jpg'
    main(imagenes, fondo)
```

[ ]:
```

```