



UNIVERSIDAD DE GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERIAS

Proyecto Integrador Unidad 4

VISIÓN ROBÓTICA

INGENIERIA EN ROBOTICA

Alumno: Carlos Omar Rodriguez Vazquez

Introducción

En el presente informe se detalla el desarrollo de una actividad enfocada en la aplicación de filtros de imagen de manera dinámica y sincronizada con un video musical. Se implementaron diversas funciones para la aplicación de filtros, cada una diseñada para generar efectos visuales específicos en momentos clave del video. El objetivo principal fue crear una experiencia visual inmersiva que complementara la narrativa de la canción mediante la manipulación creativa de la imagen.

Funciones Implementadas

Aplicacion de Filtro a Imagnes

La principal idea detrás del uso de funciones fue implementar métodos que aplicaran cada uno de los filtros utilizados, con el fin de facilitar la elaboración del código y mejorar su legibilidad. Todas las funciones que aplican un filtro a la imagen reciben como parámetro la imagen a la cual se desea aplicar el filtro, la cual debe estar en formato 'RGB', y devuelven una imagen 'RGB' con el filtro aplicado.



Figura 1: Imagen Original

Filtro Borde

```
def filtroBorde(im, t=None):
    Ig = cv.cvtColor(im, cv.COLOR_BGR2GRAY)
    n = 3
    F = (1/n**2)*np.ones((n,n))

    sobel_x = np.array([[ -1,  0,  1],
                       [-2,  0,  2],
                       [-1,  0,  1]])

    sobel_y = np.array([[ -1, -2, -1],
                       [ 0,  0,  0],
                       [ 1,  2,  1]])
```

```

Ib = cv.filter2D(Ig, -1, F)

Gx = cv.filter2D(Ib, cv.CV_64F, sobel_x)
Gy = cv.filter2D(Ib, cv.CV_64F, sobel_y)

Mag2 = Gx**2 + Gy**2
Mag = np.sqrt(Mag2)

Mag = Mag.astype(np.uint8)

In = 255 - Mag
In = cv.cvtColor(In, cv.COLOR_GRAY2BGR)

return In

```

La función filtroBorde implementa los filtros Sobel para detectar bordes en la imagen. Primero, convierte la imagen de entrada a escala de grises utilizando la función cv.cvtColor. Luego, define matrices de convolución para los filtros Sobel en las direcciones horizontal (sobel_x) y vertical (sobel_y).

Se aplica un filtro de suavizado a la imagen en escala de grises utilizando la función cv.filter2D, con el fin de reducir el ruido en la imagen. Después, se aplican los filtros Sobel horizontal y vertical a la imagen suavizada para obtener las componentes de gradiente en ambas direcciones.

Finalmente, se calcula el negativo de la magnitud del gradiente, para que los bordes detectados aparezcan en negro sobre un fondo blanco. La imagen resultante se convierte de nuevo a formato BGR utilizando cv.cvtColor.



Figura 2: Imagen de la Figura 1 con filtroBorde

Filtro Dibujo

```

def filtroDibujo(im, t=None):
    Ig = cv.cvtColor(im, cv.COLOR_BGR2GRAY)
    n = 9
    F = (1/n**2)*np.ones((n,n))
    Ig2 = cv.filter2D(Ig, -1, F)
    Ig_2 = cv.divide(Ig, Ig2, scale=255)
    Ig_2 = cv.cvtColor(Ig_2, cv.COLOR_GRAY2BGR)

    return Ig_2

```

La función filtroDibujo comienza convirtiendo la imagen de entrada a escala de grises utilizando la función cv.cvtColor de OpenCV. Luego, define un kernel de convolución de tamaño 9x9 y lo utiliza para aplicar un filtro de promedio a la imagen en escala de grises mediante cv.filter2D.

Después, se realiza una división de la imagen original entre la imagen suavizada para resaltar los detalles y dar un efecto de dibujo. Este proceso se lleva a cabo utilizando cv.divide y ajustando la escala para obtener una imagen resultante en el rango de 0 a 255.



Figura 3: Imagen de la Figura 1 con filtroDibujo

Filtro Calido y Frio

```

def filtroTono(im, t):
    if t == 0:
        xc = [0, 70, 128, 255]
        yc = [0, 100, 170, 255]

        xd = [0, 70, 128, 255]
        yd = [0, 20, 70, 255]
    else:
        xc = [0, 70, 128, 255]
        yc = [0, 20, 70, 255]

        xd = [0, 70, 128, 255]
        yd = [0, 100, 170, 255]

    fc = UnivariateSpline(xc, yc)
    fd = UnivariateSpline(xd, yd)

    B,G,R = cv.split(im)

    x = np.linspace(0, 255, 256)
    yc = fc(x)

    R = cv.LUT(R, yc).astype(np.uint8)
    yd = fd(x)

    B = cv.LUT(B, yd).astype(np.uint8)

    Im2 = cv.merge((B,G,R))

    HSV = cv.cvtColor(Im2, cv.COLOR_BGR2HSV)
    H,S,V = cv.split(HSV)

    S = cv.LUT(S, yc).astype(np.uint8)

    HSV = cv.merge((H,S,V))

    Im3 = cv.cvtColor(HSV, cv.COLOR_HSV2BGR)

    return Im3
  
```

La función filtroTono acepta una imagen y un parámetro t que indica si se aplicará un filtro cálido (t=0) o frío (t=1). La lógica detrás de estos filtros radica en ajustar los tonos de los canales de color para crear una sensación de calidez o frialdad en la imagen.

En el caso del filtro cálido, se aumentan los tonos de rojo, mientras que en el filtro frío se disminuyen estos tonos y se aumentan los de azul.

Después de ajustar los tonos de los canales de color, se recomponen los componentes de la imagen y se convierte de BGR a HSV. En el espacio de color HSV, se realiza un ajuste adicional en la saturación. Esto ayuda a mejorar el efecto de temperatura deseado, ya sea cálido o frío.



(a) Tono Calido

(b) Tono Frio

Figura 4: Imagen de la Figura 1 con filtroTono

Filtro Suave

```
def filtroSuave(im, n):
    F = (1/n**2)*np.ones((n,n))
    im2 = cv.filter2D(im, -1, F)

    return im2
```

La función filtroSuave aplica un filtro de promedio a la imagen, lo que resulta en una apariencia borrosa o suavizada. El parámetro n controla la intensidad del filtro, determinando el tamaño del kernel de convolución utilizado para el promedio.

Al aumentar el valor de n, se incrementa el tamaño del kernel, lo que resulta en un mayor suavizado de la imagen. Por el contrario, al disminuir n, se reduce el tamaño del kernel y, por lo tanto, el suavizado aplicado a la imagen.

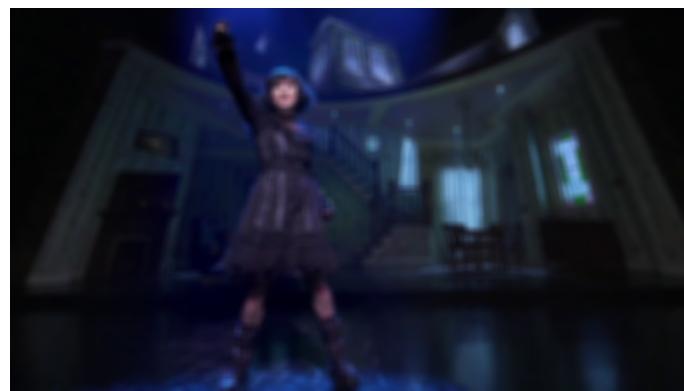


Figura 5: Imagen de la Figura 1 con filtroSuave.

Funciones Especiales

Filtro Figura

```
def filtroFigure(im, filtro, t):
    ig = cv.cvtColor(im, cv.COLOR_BGR2GRAY)
    mask = np.zeros_like(ig)

    cv.circle(mask, (round(w/2), round(h/2)), r, (255,255,255), -1)

    maskn = 255 - mask

    im2 = cv.bitwise_and(im, im, mask=maskn)

    im3 = cv.bitwise_and(im, im, mask=mask)

    im4 = filtro(im3, t)

    if len(im4.shape) < 3:
        im4 = cv.bitwise_and(im4, im4, mask=mask)
        im4 = cv.cvtColor(im4, cv.COLOR_GRAY2BGR)

    im5 = cv.bitwise_or(im2, im4)

    return im5
```

La función filtroFigure toma una imagen, un filtro y un parámetro t como entrada. Su propósito es aplicar el filtro especificado dentro de una región circular centrada en el medio de la imagen y con un radio dado.

Primero, la imagen se convierte a escala de grises utilizando cv.cvtColor. Luego, se crea una máscara de imagen del mismo tamaño que la imagen de entrada, inicialmente llena de ceros. Se dibuja un círculo blanco en esta máscara con el centro en el punto medio de la imagen y un radio definido.

Después, se crea una máscara inversa maskn. Se aplican estas máscaras a la imagen original utilizando operaciones de bitwise cv.bitwise.and, donde im2 contiene la parte de la imagen fuera del círculo y im3 contiene la parte dentro del círculo.

El filtro especificado se aplica a la región dentro del círculo utilizando la función filtro. Si la imagen resultante del filtro no es en color (es decir, si tiene una sola dimensión), se convierte a color utilizando cv.cvtColor.

Finalmente, se combina la parte dentro y fuera del círculo utilizando la operación bitwise cv.bitwise.or, y se devuelve la imagen resultante. Esta función permite aplicar cualquier filtro dentro de una región circular en una imagen.

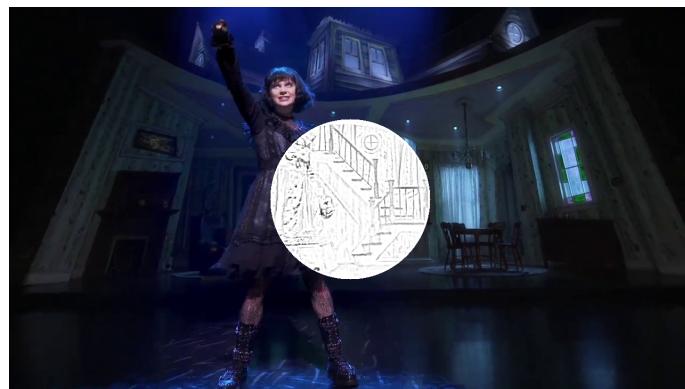


Figura 6: Imagen de la Figura 1 con filtroFigure.

Centro Cara

```

def centerFace(im):
    global cx, cy
    im2 = cv.cvtColor(im, cv.COLOR_BGR2RGB)
    results = face_detection.process(im2)

    if results.detections:
        for detection in results.detections:
            relative_bounding_box = detection.location_data.relative_bounding_box

            height, width, c = im.shape

            cx = int((relative_bounding_box.xmin +
                      relative_bounding_box.width/2)*width)
            cy = int((relative_bounding_box.ymin +
                      relative_bounding_box.height/2)*height)

        return cx, cy
    else:
        return cx, cy

```

La función centerFace devuelve las coordenadas del centro de la cara detectada en una imagen. Utiliza un modelo de detección de rostros para identificar la ubicación de la cara en la imagen de entrada. Si se detecta al menos una cara, la función devuelve las coordenadas del centro de esa cara. En caso de que no se detecte ninguna cara, devuelve las coordenadas del centro de la cara detectada previamente, o valores predeterminados si no se ha detectado ninguna cara antes.

Código

Para gestionar el momento adecuado para aplicar cada filtro en el video, se empleó una variable denominada 'b', la cual almacenaba el número de fotograma actual del video. De esta manera, si se deseaba aplicar un filtro desde el segundo 2 hasta el segundo 3.5, simplemente se realizaba la conversión a fotogramas y se aplicaba el filtro si se cumplía la condición $b > 60$ y $b < 105$.

Primera aplicación

En cierto punto del video, la canción menciona 'a lightning strike', por lo que se decidió aplicar el filtro de Bordes para simular lo mencionado anteriormente.

```

if b>73 and b<84:
    im = filtroBorde2(im)

```

Segunda aplicación

Después del momento en que ocurre el rayo, se aplica un filtro Suave que gradualmente difumina la imagen, creando una apariencia cada vez más borrosa.

```

if b>83 and b<214:
    im = filtroSuave(im, n)
    if b%25 ==0:
        n += 2

```

Tercera aplicación

En una parte específica de la canción, se menciona 'or drop a nuclear bomb', acompañado de un gesto de los brazos que empiezan desde abajo y van subiendo poco a poco. Se decidió reflejar este movimiento

aplicando el filtroTono Cálido de manera gradual. Este filtro va aumentando progresivamente, creando la ilusión de que las manos están aplicando el filtro mientras se levantan hacia arriba, para reforzar la conexión entre la letra de la canción y la acción visual en la imagen.

```
if b>171 and b<205:
    y = h-a

    Im2 = im[y:h-1,:,:]
    Im2 = filtroTono(Im2, 0)

    im[y:h-1] = Im2
    a += 12
```

Cuarta aplicación

Justo después de que termina la mímica de levantar los brazos, se representa una explosión. Para reflejar esta acción visualmente, se aplicó el filtroFigure junto con el filtroTono Cálido. El filtroFigure se utilizó para crear un círculo que simula la expansión de la explosión, y el filtroTono Cálido se aplicó para mantener la coherencia visual con la temática de la explosión. El radio del círculo se incrementó rápidamente para dar la sensación de una explosión en rápido crecimiento.

```
if b>204 and b<214:
    im = filtroFigure(im, filtroTono, 0)
    r += 250
```

Quinta aplicación

Posteriormente, durante un período de tiempo en el video, se aplicó el filtroDibujo.

```
f b>213 and b<347:
    im = filtroDibujo(im)
```

Sexta aplicación

Inmediatamente después de la aplicación del filtro anterior, se aplicó otro filtroDibujo, pero esta vez solo a una sección específica del video. Esta sección se va reduciendo gradualmente hasta alcanzar aproximadamente la mitad de la imagen.

```
if b>346 and b<470:
    im2 = im[:,0:u,:]
    im2 = filtroDibujo(im2)
    im[:,0:u,:] = im2

    u -= 5
```

Septima aplicación

Continuando con la aplicación anterior, el filtroDibujo sigue aplicándose solo a una parte específica del video. Sin embargo, en esta ocasión, se utiliza la función centerFace para seleccionar la sección de la imagen que corresponde al centro de la cara detectada en cada fotograma del video.

```
if b>469 and b<613:
    cx, cy = centerFace(im)
    im2 = im[:,0:cx,:]
    im2 = filtroDibujo(im2)
    im[:,0:cx,:] = im2
```

Resultado

El video final se puede ver aquí.

Conclusión

La actividad proporcionó una oportunidad para explorar las capacidades de procesamiento de imágenes y video, así como para desarrollar habilidades en la implementación de algoritmos de filtrado y manipulación de imágenes en tiempo real. La sincronización de los filtros con los eventos del video permitió crear una narrativa visual coherente y cautivadora que realzó la experiencia del espectador. Este ejercicio demostró el potencial creativo de la tecnología digital para expresar ideas y emociones de manera innovadora y estimulante.