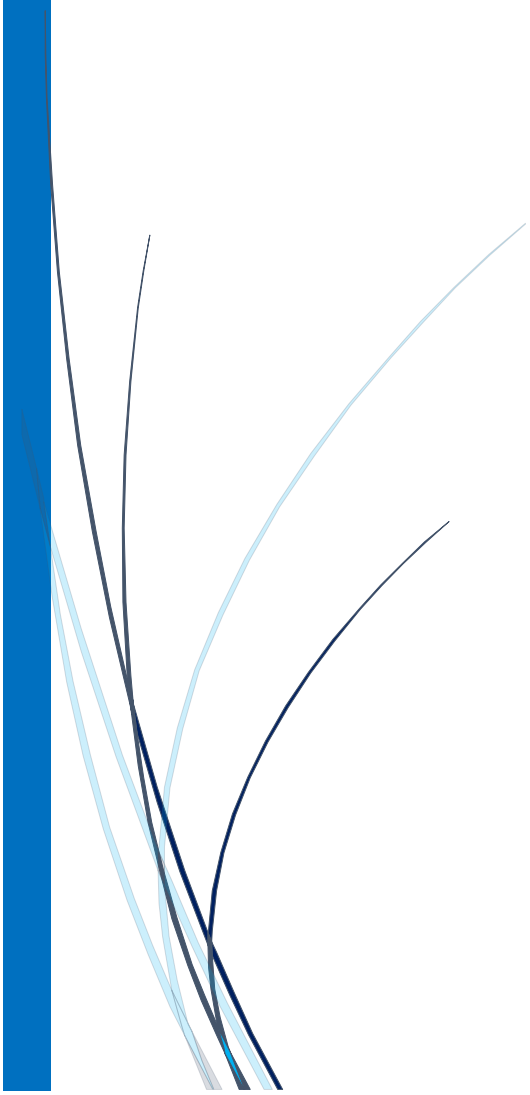
26-9-2022

Primer Examen Parcial

Visión Computacional

Prof. Dr. César Augusto Puente Montejano



DIEGO DOMINGO CHACON RIVERA - 307382

UASLP – FACULTAD DE INGENIERÍA

Planteamiento del problema a resolver

Aplicar una serie de instrucciones a una imagen, en el orden que al momento de aplicarlas se obtenga el mejor resultado posible a criterio propio.

Instrucciones:

- Aplicar el filtro de Sobel sobre la imagen original y guardarla en una nueva.
- Aplicar el filtro morfológico de Dilatación las veces necesarias con el objetivo de definir de la mejor manera el rostro de la persona que aparece en la imagen.
- Recortar la imagen de manera que obtenga una nueva imagen con sólo el rostro de la persona.
- Aplicar un filtro (el que usted decida) a la imagen de manera que el rostro de la persona aparezca lo más nítido posible
- Rotar la imagen de manera que el rostro no presente ningún grado de inclinación.

Descripción de la solución

Con la finalidad de trabajar de manera y teniendo en cuenta que cada función o proceso debe generar una imagen, **se definieron variables de ruta para cada imagen con la finalidad de utilizarlas en todas las funciones** para proporcionar un código más ordena y comprensible.

- Para **Guardar las imágenes** generadas requerimos únicamente de la función:

```
Imwrite(Nombre y/o ruta de la imagen, Imagen a guardar);
```

Orden de los pasos y descripción de cada uno de ellos

1. Aplicamos el **filtro de Sobel**, por lo cual creamos una función llamada:

   ```
   sobelImg(Nombre para guardar la imagen, Nombre de la imagen original);
   ```

   Dentro de la función, cargamos la imagen con la que vamos a trabajar. Para aplicar el **filtro de Sobel** utilizamos la función llamada:

   ```
   spatialGradient(Imagen para aplicar el filtro);
   ```

   Aplicamos esta función debido a que ya nos devuelve las derivadas del operador Sobel tanto para X y Y (dx, dy). Al devolvernos los bordes verticales y horizontales, daremos preferencia a los bordes horizontales ya que se resaltar con mayor detalle sobre la imagen.
   Imagen del codigo: anexos\f_sobelImg.png

2. Después aplicamos el **filtro morfológico de dilatación**, para esto creamos una función llamada:

   ```
   dilatarImg(Nombre para guardar la imagen, Nombre de la imagen a trabajar);
   ```

   Dentro de la función, cargamos la imagen con la que vamos a trabajar. Para aplicar el filtro de dilatación definimos un **tamaño m de la matriz**, después construimos **una matriz de tamaño m para el kernel** y la aplicamos en la siguiente función:

   ```
   dilate(Imagen a trabajar, Una matriz o kernel, Numero de iteraciones);
   ```

   Imagen del código: anexos\f_dilatarImg.png

3. Posteriormente aplicamos **el filtro opcional** creamos una función para hacer referencia a la llamada:

   filtroOPImg(Nombre para guardar la imagen, Nombre de la imagen a trabajar);

   Dentro de la función, cargamos la imagen con la que vamos a trabajar. El filtro con el cual vamos a trabajar con tal de mejorar la nitidez es:

   bilateralFilter(Imagen a trabajar, Tamaño del filtro (d), SigmaColor, SigmaSpace);

   - **d** nos indica que tan qué tan grande o fuerte será el efecto de suavizado de la imagen.
   - **SigmaColor** valor en el espacio de color
   - **SigmaSpace** valor en el espacio de ubicación.

   Este filtro nos va a ayudar a eliminar el ruido y suavizar la imagen mientras retiene la información de los **bordes en la imagen tanto como sea posible** (es decir, mantiene los bordes del objeto lo más nítidos posible).
   Imagen del código: anexos\f_filtroOPImg.png

4. Ahora bien, ya una vez aplicamos los filtros a la imagen, procedemos a aplicarle una **Rotación para dejar lo más centrada la imagen posible**, para lo cual creamos una función llamada:

   rotarImg(Nombre para guardar la imagen, Nombre de la imagen a trabajar);

   Dentro de la función, cargamos la imagen con la que vamos a trabajar. Para rotar la imagen requerimos de 2 funciones especiales que nos permitan rotar una cantidad exacta de grados, dichas funciones son:

   getRotationMatrix2D(Centro de la imagen, Angulo de rotación, Escala);

   Esta función nos va a retornar una matriz **M** que la utilizaremos en la siguiente función para aplicar la rotación indicada en la función.

   warpAffine(Imagen a trabajar, Matriz de rotación (M), Tamaño (w, h));

   El tamaño hace referencia a la imagen, para el largo **(w)** y para la altura **(h)**. Esto se puede obtener haciendo indexación Numpy:

   (h, w) = imagen.shape[:2];

   Imagen del código: anexos\f_rotarImg.png

5. Casi para concluir debemos **Recortar la cara de la imagen**, por lo cual creamos una función llamada:

   recortarImg(Nombre para guardar la imagen, Nombre de la imagen a trabajar);

   Dentro de la función, cargamos la imagen con la que vamos a trabajar. Para recortar la imagen tenemos que tratarla como una matriz, para ello aplicaremos indexación Numpy, aplicándolo de la siguiente manera:

   imagen[fila inicial : fila final, columna inicial : columna final]

   Imagen del código: anexos\f_recortarImg.png

6. Para concluir únicamente **Visualizamos el resultado final de la imagen** generada, con esto construimos una pequeña función llamada:

   visualizarImg(Nombre de la imagen);

Dentro de la función cargamos la imagen haciendo uso de:

```
Imread(Nombre y/o ruta de la imagen, 1);
```

Esta función nos regresa una imagen **(img)** en forma de matriz y la mostraremos con la siguiente función:

```
Imshow(Nombre y/o título de la ventana donde se mostrara la imagen, Imagen a mostrar (img));
```

Para hacer una pequeña pausa y que podamos ver la imagen con tranquilidad utilizamos la función:

```
waitKey(0);
```

Por último, debemos destruir todas las ventanas creadas, esto lo hacemos con la función:

```
destroyAllWindows();
```

Imagen del código: anexos\f_visualizarImg.png

Descripción de los resultados

1. Aplicamos la función **sobelImg()**, para aplicar el filtro de sobel sobre la imagen original. Esto debido a que nos creara una imagen en la cual se detectaron los bordes horizontales y verticales por separado. Además, que la imagen se convierte a RGB en escala de grises.

Imagen Original

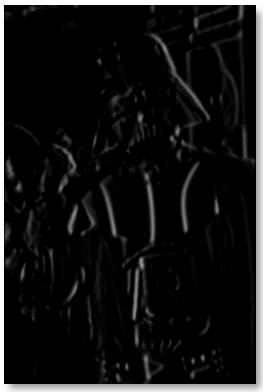


Imagen con el filtro Sobel aplicado

2. Una vez aplicado el filtro de Sobel procedemos a acentuar las características de la imagen, principalmente darle un resaltado a los bordes, por lo cual requerimos de un dilatado. En este paso aplicamos la función **dilatarImg();**

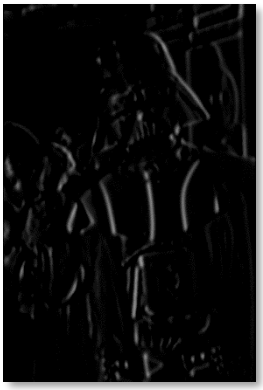


Imagen con el filtro Sobel aplicado

Imagen dilatada

Se ve una diferencia principalmente en el contraste de los bordes.

3. Ahora bien, damos a darle suavidad a la imagen para reducir el ruido (aplicar mayor nitidez). Aplicamos el filtro bilateral para que se conserven los bordes de la imagen y que se resalten de mejor manera todas las demás partes. En este paso aplicamos la función filtroOPImg().

Imagen dilatada

Imagen con filtro de nitidez

4. Una vez que ya aplicamos los 3 filtros, procedemos a rotar la imagen para que se quite esa pequeña inclinación hacia la izquierda. Para esto aplicamos la función rotarImg();

Imagen rotada

5. Por último, recortamos la cara del personaje (Darth Vader) de la imagen. Para esto aplicamos la función rocortarImg().

Imagen recortada

Discusión

La decisión de empezar con el filtro de Sobel se debe a que nos iba a cambiar toda la estructura de la imagen, por lo cual tenia un grado de primordialidad. Por otra parte, al aplicar los filtros de suavizado y dilatación los resultados variaban mucho del orden en el que se ejecutaban, por lo cual también era necesario hacer un reajuste en los valores de calibración de cada filtro, en ocasiones se sobrepasaba el dilatado y se veía muy blanco y/o brillante y, por otro lado, no se apreciaba nada de la imagen siendo su contraparte. Esto influyo a decidir establecer el orden de aplicación para el proyecto.

Ahora bien, las funciones de rotar y cortar se aplicaron hasta el ultimo debido a que la imagen ya había pasado por todos los filtros necesarios. Pero al momento de recortar justamente la cara del personaje, se pierde resolución. Gracias a que se resaltaron los bordes y se suavizo todo lo demás (sin dañar los bordes), logramos tener una buena apreciación del trazado de la imagen. Algo que también cabe mencionar es que la imagen inicial no tenía muy buena resolución lo cual influye bastante en el resultado final.

Conclusión

En mi opinión, fue muy interesante desarrollar el proyecto y aplicar algunas de las funciones que se le pueden hacer a una imagen con OpenCV. Al momento de aplicar los filtros te topas con una contradicción "El orden si importa", cada filtro aplica propiedades diferentes a las imágenes y al momento de mezclarlas se obtienen resultados diferentes.

Ahora bien, el tener que describir cada función nos ayuda a memorizar sus parámetros, para que se aplica y su resultado o bien "lo que devuelve". Esto nos ayuda a ya no tener que estar recurriendo a la documentación en todo momento. De igual manera la opción de dejar un filtro libre para aplique mayor nitidez te ayuda mucho a que veas el resultado y compares los resultados con las diferentes funciones que suavidad y reducción de ruido.

Por último, cabe señalar que el hacer el proyecto deja una buena enseñanza practica (comprender toda la teoría vista en clase). En el caso de OpenCV, como se utilizo bajo un sistema operativo base Linux/Unix y el lenguaje de programación Python, fue mas sencillo desde instalarlo, hasta usarlo en código que solo requiere de un import cv2, además que gracias al lenguaje no se necesitan definir tipos de variables.

En si todo el proyecto fue una buena aventura de desarrollo para el conocimiento de la visión computacional.