Universidad Nacional de San Antonio Abad del Cusco
Departamento Académico de Ing. Informática
VISION COMPUTACIONAL
Práctica Nº 05

TRANSFORMACIONES GEOMETRICAS

Iván C. Medrano Valencia

1. OBJETIVO.

- Conocer y aplicar las transformaciones geométricas en imágenes.
- Utilizar las funciones de OpenCV para realizar transformaciones geométricas

2. INTRODUCCIÓN.

A menudo, durante el procesamiento de imágenes, es necesario transformar la geometría de la imagen (por ejemplo, el ancho y el alto de la imagen). Una transformación geométrica de una imagen es una transformación en su sistema de coordenadas. Es decir, no se altera el valor de los pixeles sino la posición de los mismos.

Como sabemos, las imágenes no son más que matrices, por lo que podemos utilizar un enfoque más matemático para comprender estos temas.

Dado que las imágenes son matrices, si aplicamos una operación a las imágenes (matrices) y terminamos con otra matriz, llamamos a esto una transformación. Esta idea básica se utilizará ampliamente para comprender y aplicar varios tipos de transformaciones geométricas, como traslación, rotación, escalamiento, etc.

3. FUNCIONES DE OpenCV PARA HACER TRANSFORMACIONES GEOMETRICAS

OpenCV proporciona dos funciones de transformación, **cv2.warpAffine** y **cv2.warpPerspective**, con las que se pueden realizar todo tipo de transformaciones.

La función **cv2.warpAffine** toma una matriz de transformación 2×3 mientras **cv2.warpPerspective** toma una matriz de transformación 3×3 como entrada

Los parámetros de **cv2.warpAffine** son los siguientes:

- Imagen de entrada.
- M, matriz de transformación de 2 filas x 3 columnas (2×3).
- Tamaño de la imagen de salida.

4. ACTIVIDAD I.

4.1. TRASLACIÓN

Una traslación es el desplazamiento de la posición de un objeto. Si se conoce la magnitud del desplazamiento (t_x,t_y) en las direcciones x e y, respectivamente, se puede escribir la matriz de transformación M como:

$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

Esta matriz luego se pasa como argumento a la función **cv2.warpAffine()**.

Ejercicio 5.1

El siguiente ejercicio traslada una imagen haciendo un desplazamiento (210, 150)

```
import cv2
import numpy as np
#--leer la imagen
img = cv2.imread('images/ave.jpg',0)
#--obtenemos ancho y alto de la imagen
rows,cols = img.shape
#--desplazamiento
tx = 210
ty = 150
#--crea la matriz de traslación
M = np.float32([[1,0,tx],
                [0,1,ty]])
#--realiza la traslación
dst = cv2.warpAffine(img,M,(cols,rows))
#--muestra la imagen
cv2.imshow('traslación',dst)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

4.2. ROTACION

La rotación de una imagen, en un cierto ángulo θ, se logra aplicando la siguiente matriz de transformación:

$$M = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

Sin embargo, OpenCV permite además personalizar más la rotación multiplicando por un factor de escala. Por otra parte, también permite cambiar el centro de rotación. La matriz de transformación modificada, con estas dos nuevas opciones, tiene la forma:

$$\begin{bmatrix} \alpha & \beta & (1-\alpha)\cdot center.x - \beta \cdot center.y \\ -\beta & \alpha & \beta \cdot center.x + (1-\alpha)\cdot center.y \end{bmatrix}$$

donde

$$\alpha = scale \cdot \cos\theta, \\ \beta = scale \cdot \sin\theta$$

Para encontrar esta matriz de transformación, OpenCV proporciona la función **cv2.getRotationMatrix2D** cuyos parámetros son:

- Centro de rotación de la imagen de entrada.
- Ángulo de rotación en grados (rotación en sentido antihorario).
- Escala, valor de escala isotrópica.

Ejercicio 5.2. Compruebe a continuación un ejemplo en el cual se gira la imagen 45 grados con respecto al centro sin aplicar ningún factor de escala.

```
import cv2
import numpy as np
#--leer imagen
image = cv2.imread('images/ave.jpg')
#--obtener dimensiones de la imagen
ancho = image.shape[1] #columnas
alto = image.shape[0] # filas
#--define la matriz de rotación
M = cv2.getRotationMatrix2D((ancho//2,alto//2),15,1)
#--ejecuta la rotación
imageOut = cv2.warpAffine(image,M,(ancho,alto))
#--muestra las imágenes
cv2.imshow('Imagen de entrada',image)
cv2.imshow('Imagen de salida',imageOut)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

4.3. ESCALADO

El escalado se implementa mediante una función específica llamada **cv2.resize**, que permite indicar unas dimensiones concretas o una proporción entre la imagen origen y destino.

Esta función tiene los siguientes parámetros:

- Imagen de entrada.
- (ancho, alto) que se le quiere dar a la nueva imagen.
- Método de interpolación. (Banderas de interpolación que se puede usar)

Enumerator
INTER_NEAREST Python: cv.INTER_NEAREST
INTER_LINEAR Python: cv.INTER_LINEAR
INTER_CUBIC Python: cv.INTER_CUBIC
INTER_AREA Python: cv.INTER_AREA
INTER_LANCZOS4 Python: cv.INTER_LANCZOS4
INTER_LINEAR_EXACT Python: cv.INTER_LINEAR_EXACT
INTER_MAX Python: cv.INTER_MAX
WARP_FILL_OUTLIERS Python: cv.WARP_FILL_OUTLIERS
WARP_INVERSE_MAP Python: cv.WARP_INVERSE_MAP

Ejercicio 5.3. El siguiente ejercicio escala una imagen a 600 x 300 pixeles utilizando el método de interpolación bicúbica.

```
import cv2
#--leer imagen
image = cv2.imread('images/ave.jpg')
#--obtener dimensiones
ancho = image.shape[1] #columnas
alto = image.shape[0] # filas
print(alto,ancho)
#--escalando la imagen
imageOut = cv2.resize(image,(600,300), interpolation=cv2.INTER_CUBIC)
#--mostrar las imágenes
cv2.imshow('Imagen de entrada',image)
cv2.imshow('Imagen de salida',imageOut)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

4.4. RECORTE

Para recortar una imagen vamos a tratar a esta como una matriz, para ello emplearemos indexación de Numpy, de donde escogeremos las filas y columnas que deseemos recortar de la imagen.

Ejercicio 5.4. El siguiente ejercicio recorta una imagen teniendo como punto de inicio el pixel (60, 220) y el punto final el pixel (280,480).

```
import cv2
#--leer la imagen
image = cv2.imread('images/ave.jpg')
#--recortar una imagen
imageOut = image[60:220,280:480]
#--mostra las imágenes
cv2.imshow('Imagen de entrada',image)
cv2.imshow('Imagen de salida',imageOut)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

4.5. INCLINACION

La inclinación se puede aplicar sobre el eje X, Y o ambos si lo deseamos así, al igual que otras transformaciones debemos definir la siguiente matriz:

$$I = \begin{bmatrix} 1 & \theta_x & 0 \\ \theta_y & 1 & 0 \end{bmatrix}$$

Ejercicio 5.5. El siguiente ejemplo agrega una inclinación de 20 grados en el eje X y de 15 grados en el eje Y, si deseamos inclinar solamente en X establecemos Y a cero o viceversa.

```
import cv2
import numpy as np
import math
#--leer la imagen
src = cv2.imread('images/lena.jpg')
#--obtener dimensiones
rows, cols = src.shape[:2]
#--obtener las tangentes en ambos ejes
ix = math.tan(20 * math.pi / 180)
iy = math.tan(15 * math.pi / 180)
#--obtener la matriz de transformación
M = np.float32([[1, ix, 0], [iy, 1, 0]])
#--ejecutar la trasnformación
dst = cv2.warpAffine(src, M, (cols + 256, rows + 256))
#--mostrar las imágenes
cv2.imshow('lena.jpg', src)
cv2.imshow('Inclinar', dst)
cv2.waitKey()
```

4.6. TRANSFORMACIÓN AFÍN

En una transformación afín todas las líneas paralelas en la imagen original seguirán siendo paralelas en la imagen de salida. Para encontrar la matriz de transformación, necesitamos tres puntos de la imagen de entrada y sus ubicaciones correspondientes en la imagen de salida. Luego **cv2.getAffineTransform** creará una matriz 2×3 que se pasará a **cv2.warpAffine**.

Ejercicio 5.6. A continuación en el ejemplo, se realiza una transformación afin.

```
import numpy as np
import matplotlib.pyplot as plt #carga la librería para graficar
import cv2
#--leer la imagen
img = cv2.imread('images/cuadricula.jpg')
#--obtener dimensiones
rows,cols,ch = img.shape

#--definir 3 puntos de la imagen original
pts1 = np.float32([[100,400],[400,100],[100,100]])
#--definir 3 puntos en la imagen de salida
pts2 = np.float32([[50,300],[400,200],[80,150]])

#--obtener la matriz de transformación
M = cv2.getAffineTransform(pts1,pts2)
#--aplicar la transofrmación
dst = cv2.warpAffine(img,M,(cols,rows))
#--mostrar las imágenes
plt.subplot(121),plt.imshow(img),plt.title('Input')
plt.subplot(122),plt.imshow(dst),plt.title('Output')
plt.show()
```

4.7. TRANSFORMACIÓN DE PERSPECTIVA

Para realizar una transformación de perspectiva es necesario especificar una matriz 3×3. Luego de aplicar este tipo de transformación, las líneas rectas permanecerán rectas. Para generar la matriz de 3×3 es necesario indicar cuatro puntos sobre la imagen

inicial y los correspondientes puntos sobre la imagen resultante. Tres de los cuatro puntos, tienen que ser no-colineales. De esta manera la matriz de transformación puede ser generada utilizando la función **cv2.getPerspectiveTransform**. Luego, para aplicar la transformación, se utiliza **cv2.warpPerspective** utilizando la matriz 3×3 generada con la función anterior.

Ejercicio 5.7. El siguiente ejemplo realiza la transformación de perspectiva.

```
import math
import numpy as np
import cv2
#--leer la imagen
src = cv2.imread('images/left.jpg')
#--obtener dimensiones
rows, cols = src.shape[:2]

#--definir los 4 puntos en la imagen original
pts1 = np.float32([[113, 137], [256, 136], [270, 337], [140, 377]])
#--definir los 4 puntos en la imagen de salida
pts2 = np.float32([[0, 0], [165, 0], [165, 223], [0, 223]])

#--obtener la matriz de transformación
M = cv2.getPerspectiveTransform(pts1, pts2)

#--ejecutar la transformación
dst = cv2.warpPerspective(src, M, (165, 223))
#--mostrar las imágenes
cv2.imshow('Imagen con perspectiva', src)
cv2.imshow('Transform.', dst)
cv2.waitKey()
```

Ejercicio 5.8. El siguiente ejemplo realiza la transformación de perspectiva seleccionando los puntos desde el mouse.

```
import cv2
import numpy as np

def clics(event,x,y,flags,param):
    global puntos
    if event == cv2.EVENT_LBUTTONDOWN:
        cv2.circle(imagen,(x,y),5,(0,255,0),2)
        puntos.append([x,y])

def uniendo4puntos(puntos):
    cv2.line(imagen,tuple(puntos[0]),tuple(puntos[1]),(255,0,0),1)
    cv2.line(imagen,tuple(puntos[0]),tuple(puntos[2]),(255,0,0),1)
    cv2.line(imagen,tuple(puntos[2]),tuple(puntos[3]),(255,0,0),1)
    cv2.line(imagen,tuple(puntos[1]),tuple(puntos[3]),(255,0,0),1)

puntos = []
imagen = cv2.imread('images/licenseplate1.jpg')
aux = imagen.copy()
cv2.namedWindow('Imagen')
cv2.setMouseCallback('Imagen',clics)

while True:

    if len(puntos) == 4:
        uniendo4puntos(puntos)
```

```
            pts1 = np.float32([puntos])
            pts2 = np.float32([[0,0], [480,0], [0,300], [480,300]])

            M = cv2.getPerspectiveTransform(pts1,pts2)
            dst = cv2.warpPerspective(imagen, M, (480,300))

            cv2.imshow('dst', dst)
        cv2.imshow('Imagen',imagen)

        k = cv2.waitKey(1) & 0xFF
        if k == ord('n'):
            imagen = aux.copy()
            puntos = []

        elif k == 27:
            break

    cv2.destroyAllWindows()
```

5. TAREA.

Investigar la librería **imutils** para realizar transformaciones geométricas ilustrando con ejemplos la utilización de esta librería.

6. REFERENCIAS BIBLIOGRÁFICAS

- Dawson-Howe, K. (2014). A Practical Introduction to Computer Vision With OpenCV, Wiley & Sons Ltd.
- Hafsa Asad, W. R., Nikhil Singh (2020). The Computer Vision Workshop. Birmingham U.K., Pack Publishing.
- Szeliski, R. (2011). Computer Vision Algorithms and Applications. London, Springer.