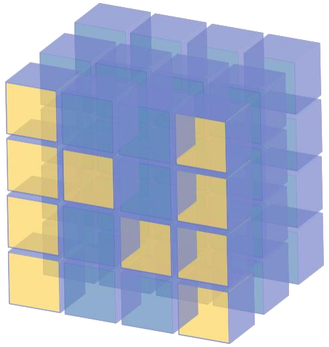
PROCESAMIENTO DIGITAL DE IMAGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

Módulos principales



NumPy

matplotlib

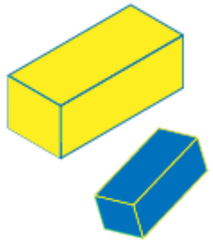
OpenCV

pip install opencv-python

PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

TEMAS QUE VEREMOS HOY



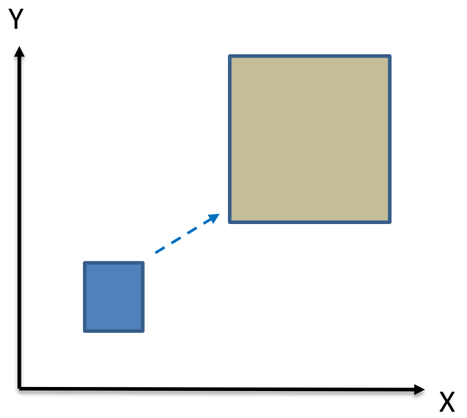
- Transformaciones geométricas dé imágenes
- Redimensionalización
- Traslación
- Rotación
- Transformación afín
- Transformación de perspectiva

PROCESAMIENTO DIGITAL DE IMÁGENES

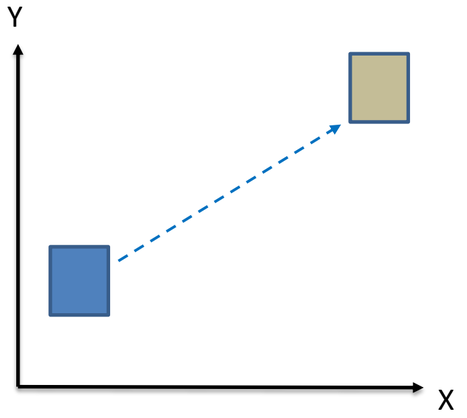
UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

TRANSFORMACIONES GEÓMETRICAS DE IMÁGENES

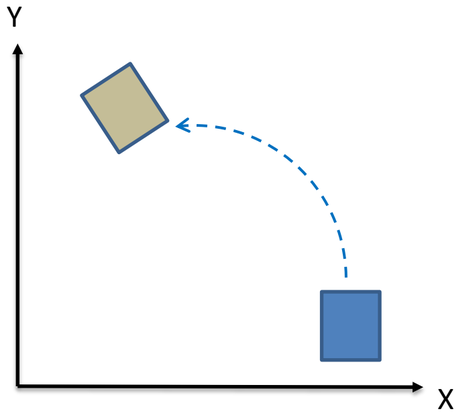
- Se busca representar las operaciones como matrices
- Tener todo como matrices permite realizar las operaciones mediante la multiplicación de matrices
- Un punto en un espacio 2D es representado como un vector de 3 componentes



Redimensionalización



Traslación

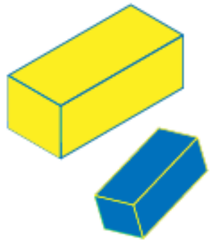


Rotación

PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

TRANSFORMACIONES GEÓMETRICAS DE IMÁGENES



- En un espacio de 2 dimensiones, los puntos son representados como una matriz 3x1
- La variable “x” indica la posición del punto en el eje X
- La variable “y” indica la posición del punto en el eje Y
- La representación de la posición es mediante coordenadas homogeneas

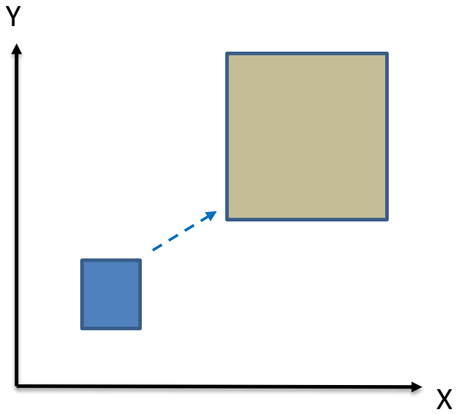
$$P = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

REDIMENSIONALIZACIÓN

- Consiste en modificar el tamaño de un objeto mediante un factor de escalamiento
- El factor de escalamiento está definido como (Sx, Sy)
- Se escala un punto (x, y) hacia (x’, y’)



Matricialmente:

x’ = Sx * x

y’ = Sy * y

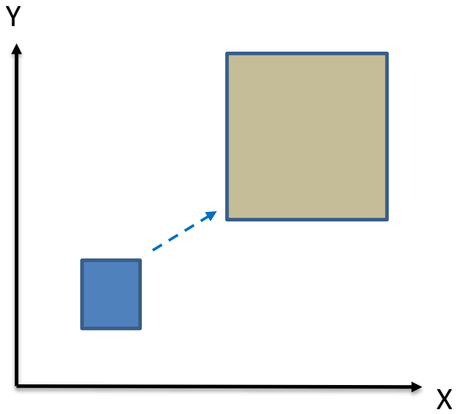
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix}$$

PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

REDIMENSIONALIZACIÓN

- Al aplicar el factor de escalamiento se modifica el tamaño de la imagen
- La imagen se moverá a otra posición si el factor de escalamiento no se aplica desde las coordenadas (0, 0)
- La imagen se mueve debido que los valores de la posición son modificados



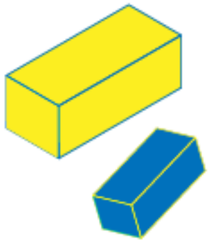
Matricialmente:

x’ = Sx * x

y’ = Sy * y

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix}$$

REDIMENSIONALIZACIÓN



- Por estandarización con el resto de transformaciones, en lugar de trabajar con una matriz 2x2 se usará una matriz 3x3

Matricialmente:

x’ = Sx * x

y’ = Sy * y

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix}$$

Matricialmente:

x’ = Sx * x

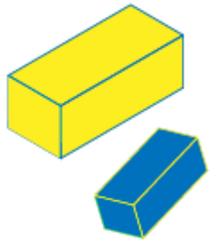
y’ = Sy * y

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

Mostrar una imagen



resize : Cambia el tamaño de una imagen

Sintaxis: cv2.resize(imagen, tamaño[, fx[, fy]])

Parametros:
imagen : Imagen a la cual modificar su tamaño
tamaño : Nuevo tamaño de la imagen como una tupla (ancho, alto)
fx : Factor de escalamiento en el eje X
fy : Factor de escalamiento en el eje Y

Nota Si el valor del parámetro “tamaño” es cero o nulo entonces se usará los parámetros “fx” y “fy” para calcular el nuevo tamaño de la imagen

PROCESAMIENTO DIGITAL DE IMÁGENES

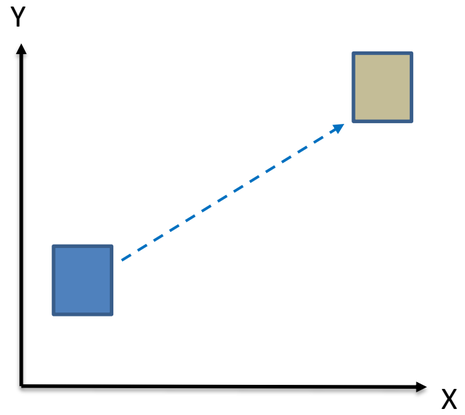
UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

Ejemplo:

```python
import cv2

image = cv2.imread("random_scene.jpg")

Sx = 1.2
Sy = 1.5

image_transformed = cv2.resize(image, None, fx=Sx, fy=Sy)

cv2.imshow("Imagen", image)
cv2.imshow("Imagen Transformada", image_transformed)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

TRASLACIÓN

- Consiste en modificar la posición de un punto a través de una línea recta
- El desplazamiento efectuado está definido como (Tx, Ty)
- Se traslada un punto (x, y) hacia (x’, y’)



Matricialmente:

x’ = x + Tx

y’ = y + Ty

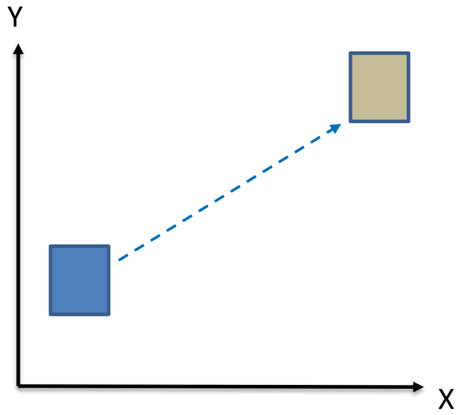
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} Tx \\ Ty \end{bmatrix}$$

PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

TRASLACIÓN

- Al aplicar el desplazamiento se modifica la posición del punto
- Para desplazar una imagen se debe desplazar del mismo modo todos sus puntos



Matricialmente:

$x' = x + Tx$

$y' = y + Ty$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} Tx \\ Ty \end{bmatrix}$$

PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

TRASLACIÓN

- Por estandarización con el resto de transformaciones, en lugar de trabajar con una matriz 2x2 se usará una matriz 3x3
- Esto permite que se aplique una multiplicación de matrices en lugar de una suma

Matricialmente:

$x' = x + Tx$

$y' = y + Ty$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} Tx \\ Ty \end{bmatrix}$$

Matricialmente:

$x' = x + Tx$

$y' = y + Ty$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & Tx \\ 0 & 1 & Ty \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

Mostrar una imagen

warpAffine : Aplica una transformación a una imagen

Sintaxis: cv2.warpAffine(imagen, matriz, tamaño)

Parametros:
image : Imagen a la cual aplicar la transformación
matriz : Matriz de la transformación
tamaño : Tamaño de la imagen como una tupla (ancho x alto)

Nota Los valores de la matriz de transformación se deben especificar con “numpy”. En opencv, la matriz de transformación tiene el siguiente formato:

$$\begin{bmatrix} 1 & 0 & Tx \\ 0 & 1 & Ty \end{bmatrix}$$

No es necesario especificar los valores de la última fila de la matriz 3x3

PROCESAMIENTO DIGITAL DE IMÁGENES

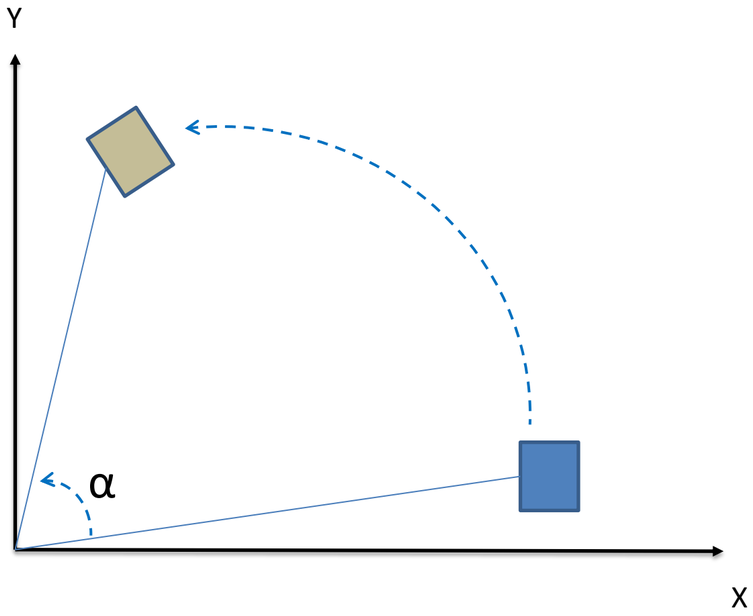
UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

Ejemplo:

```python
import cv2
import numpy as np

image = cv2.imread("random_scene.jpg")
height, width, channels = image.shape

Tx = 50
Ty = 70

matrix_transformation = np.float32([[1, 0, Tx],
                                    [0, 1, Ty]])

image_transformed = cv2.warpAffine(image, matrix_transformation, (width, height))

cv2.imshow("Imagen", image)
cv2.imshow("Imagen Transformada", image_transformed)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

ROTACIÓN

- Consiste en modificar la posición de un punto a través de una rotación
- Por defecto, la rotación se realiza en el eje de coordenadas (0, 0)
- La rotación se define mediante un ángulo
- Si el ángulo es positivo se aplica una rotación en sentido anti-horario
- Si el ángulo es negativo se aplica una rotación en sentido horario

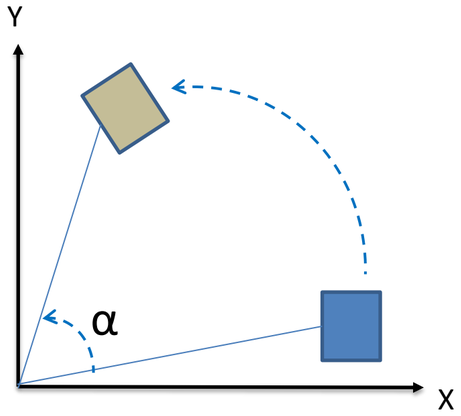


PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

ROTACIÓN

- Se rota un punto (x, y) hacia (x’, y’)
- Al aplicar la rotación se modifica la posición del punto
- Para rotar una imagen se debe rotar del mismo modo todos sus puntos



Matricialmente:

x’ = x*cos(α) – y*sen(α)

y’ = x*sen(α) + y*cos(α)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} * \begin{bmatrix} \cos(\alpha) & -sen(\alpha) \\ sen(\alpha) & \cos(\alpha) \end{bmatrix}$$

PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

ROTACIÓN

- Por estandarización con el resto de transformaciones, en lugar de trabajar con una matriz 2x2 se usará una matriz 3x3

Matricialmente:

x’ = x*cos(α) – y*sen(α)

y’ = x*sen(α) + y*cos(α)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} * \begin{bmatrix} \cos(\alpha) & -sen(\alpha) \\ sen(\alpha) & \cos(\alpha) \end{bmatrix}$$

Matricialmente:

x’ = x*cos(α) – y*sen(α)

y’ = x*sen(α) + y*cos(α)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & -sen(\alpha) & 0 \\ sen(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

Mostrar una imagen

getRotationMatrix2D : Obtiene la matriz de rotación

Sintaxis: cv2.getRotationMatrix2D (centro, ángulo, escala)

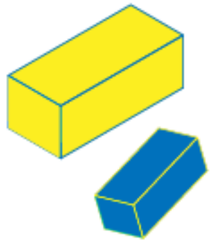
Parametros:
centro : Centro de rotación en la imagen
ángulo : Ángulo de rotación en grados
escala : Factor de escala isotrópico a aplicar en la imagen. Se puede entender como el acercamiento realizado en la imagen

Nota Si el ángulo es positivo, la rotación será en sentido anti-horario, siendo que el origen de coordenadas se encuentra en la esquina superior-izquierda. El método “getRotationMatrix2D” permite obtener una matriz de tal forma que la rotación a aplicar se realice respecto al centro de la imagen

Ejemplo:

```python
import cv2

image = cv2.imread("random_scene.jpg")
height, width, channels = image.shape

center = (width/2, height/2)

angle = 30

matrix_transformation = cv2.getRotationMatrix2D(center, angle, 1)

image_transformed = cv2.warpAffine(image, matrix_transformation, (width, height))

cv2.imshow("Imagen", image)
cv2.imshow("Imagen Transformada", image_transformed)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

COMPOSICIÓN DE TRANSFORMACIONES



- Consiste en aplicar varias transformaciones sucesivamente, para lograr un efecto deseado
- Dado que las transformaciones son matrices 3x3, se puede formar una transformación general multiplicando todas las transformaciones a aplicar

P = Punto

T1 = Rotación

T2 = Traslación

T3 = Redimensionalización

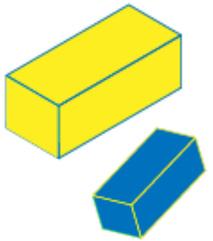
T4 = Traslación

Composición: T4 * T3 * T2 * T1

T4 * (T3 * (T2 * (T1 * P))) = Composición * P

PROCESAMIENTO DIGITAL DE IMÁGENES

COMPOSICIÓN DE TRANSFORMACIONES



Las composiciones pueden crearse multiplicando las transformaciones a realizar, las reglas de composición son:

1) La multiplicación de transformaciones es asociativa
- T1 * T2 * T3 = T1 * (T2 * T3) = (T1 * T2) * T3

2) Normalmente, la multiplicación de transformaciones no es conmutativa
- T1 * T2 != T2 * T1

3) La multiplicación de transformaciones de traslación es conmutativa

Nota: No es lo mismo aplicar primero una rotación y luego una traslación que hacerlo a la inversa

PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

Ejemplo:

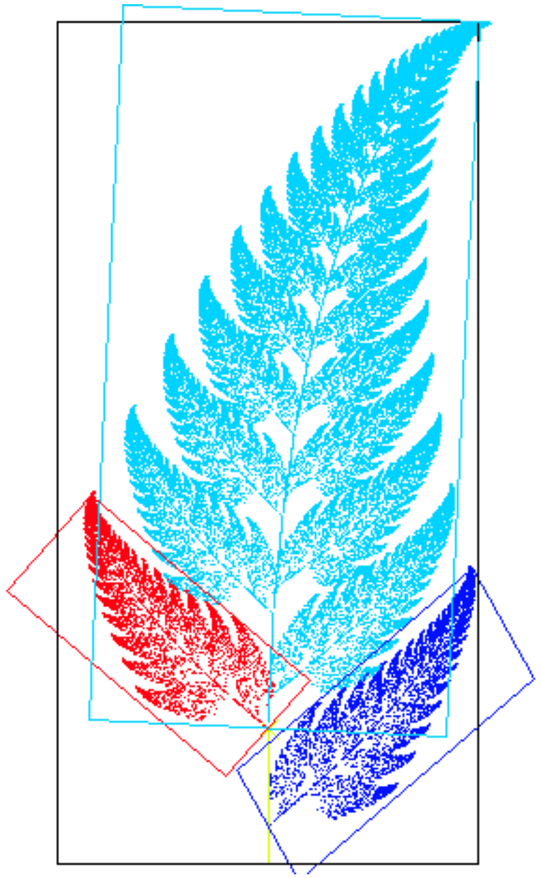
```python
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

image = Image.open("random_scene.jpg")
width, height = image.size

Sx = 1.2
Sy = 1.5

matrix_transformation_1 = np.float32([[Sx, 0, 0],
                                      [0, Sy, 0],
                                      [0,  0, 1]])

Tx = 50
Ty = 70

matrix_transformation_2 = np.float32([[1, 0, Tx],
                                      [0, 1, Ty],
                                      [0, 0, 1]])

angle = 30
angle = np.radians(angle)

matrix_transformation_3 = np.float32([[np.cos(angle), -np.sin(angle), 0],
                                      [np.sin(angle),  np.cos(angle), 0],
                                      [0,                          0, 1]])

composition = matrix_transformation_3 @ matrix_transformation_2 @ matrix_transformation_1

composition_inverse = np.linalg.inv(composition)
composition_inverse = composition_inverse[:-1]

output_size = (int(Sx*width), int(Sy*height))
image_transformed = image.transform(output_size, Image.AFFINE, data=composition_inverse.flatten())

plt.title("Imagen Transformada")
plt.imshow(np.array(image_transformed))
plt.show()
```

PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

TRANSFORMACIÓN AFÍN

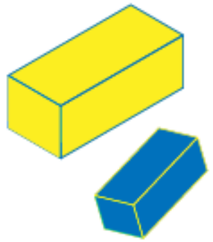
- Consiste en una composición de transformaciones aplicada a un punto
- Cualquier transformación afín puede ser descompuesta en las transformaciones de redimensionalización, traslación y rotación



PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

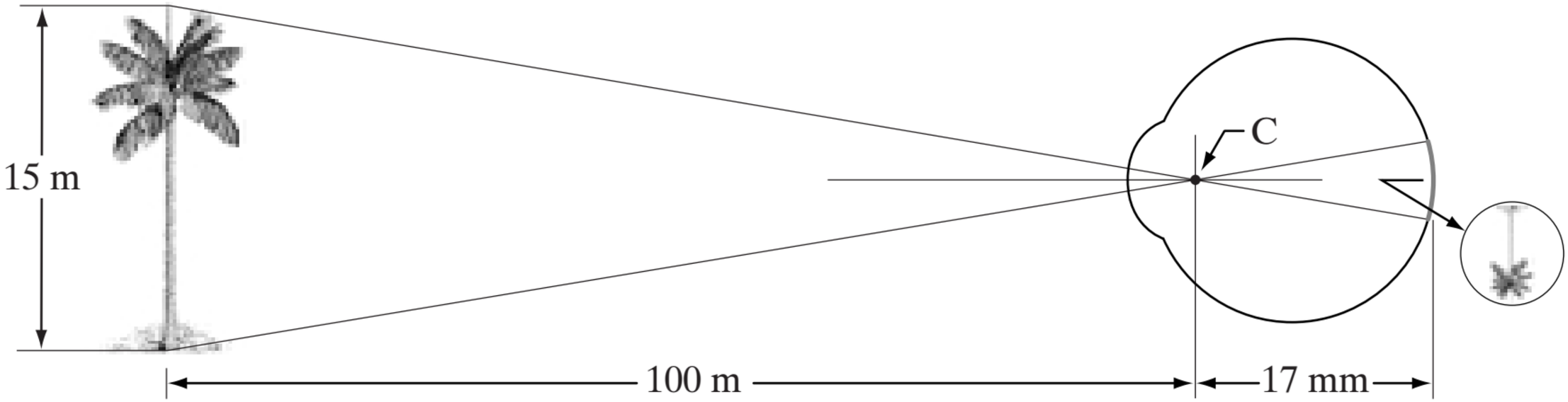
Ejemplo:



```python
import numpy as np

image = cv2.imread("random_scene.jpg")
height, width, channels = image.shape

points_1 = np.float32([[50, 50],
                       [200, 50],
                       [50, 200]])

points_2 = np.float32([[10, 100],
                       [200, 50],
                       [100, 250]])

matrix_transformation = cv2.getAffineTransform(points_1, points_2)

image_transformed = cv2.warpAffine(image, matrix_transformation, (width, height))

for point in points_1:
    x, y = point
    point = tuple([int(x), int(y)])

    cv2.circle(image, point, 3, (0, 255, 0), -1)

for point in points_2:
    x, y = point
    point = tuple([int(x), int(y)])

    cv2.circle(image_transformed, point, 3, (0, 255, 0), -1)

cv2.imshow("Imagen", image)
cv2.imshow("Imagen Transformada", image_transformed)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

TRANSFORMACIÓN DE PERSPECTIVA

- Los humanos tenemos una proyección perspectiva

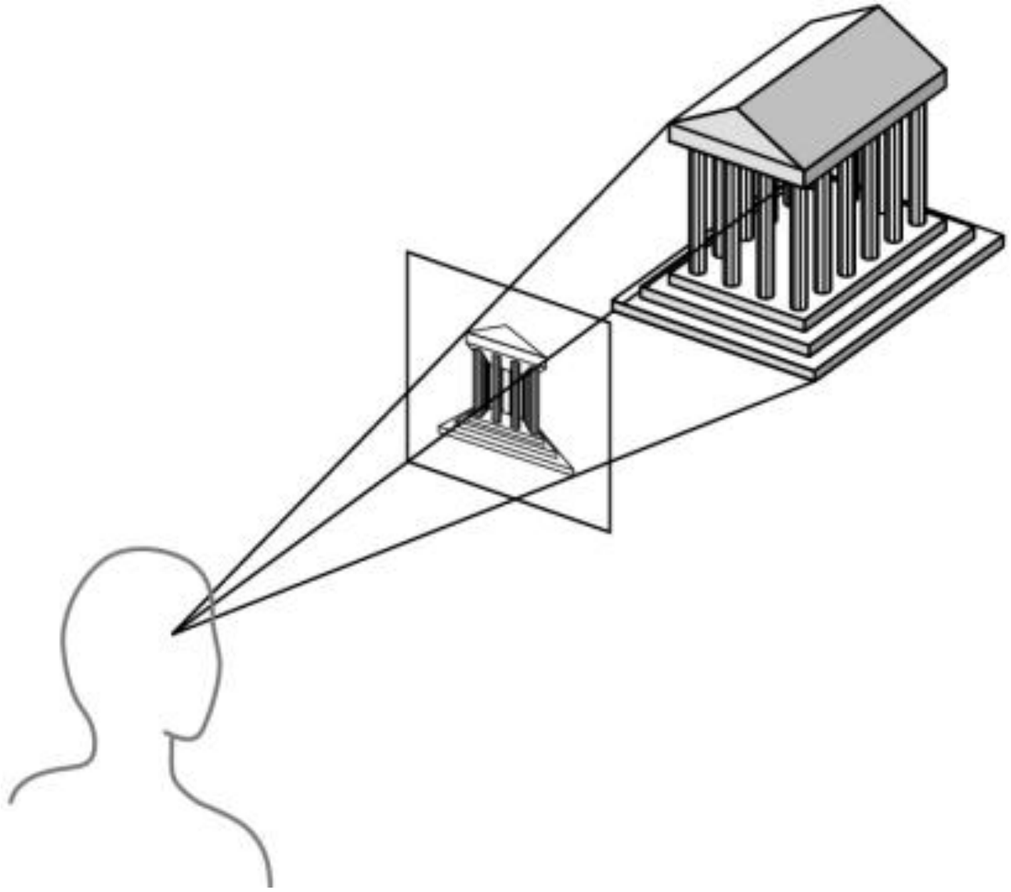


PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

TRANSFORMACIÓN DE PERSPECTIVA

- Mediante la proyección perspectiva se obtiene una plano paralelo de la imagen observada respecto del punto de vista

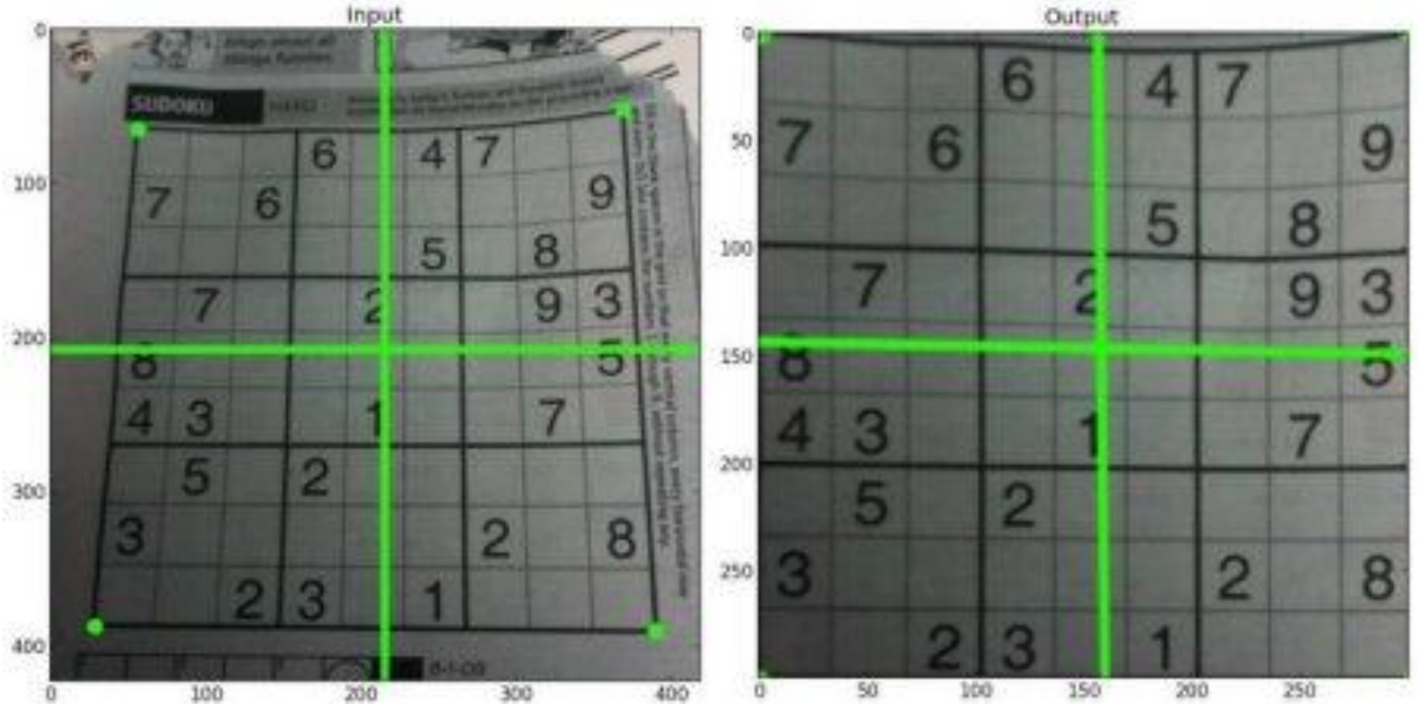


PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

TRANSFORMACIÓN DE PERSPECTIVA

- La transformación de perspectiva consiste en observar una determinada región en una imagen
- La región es observada de arriba hacia abajo
- De la imagen se obtiene solo la región a observar

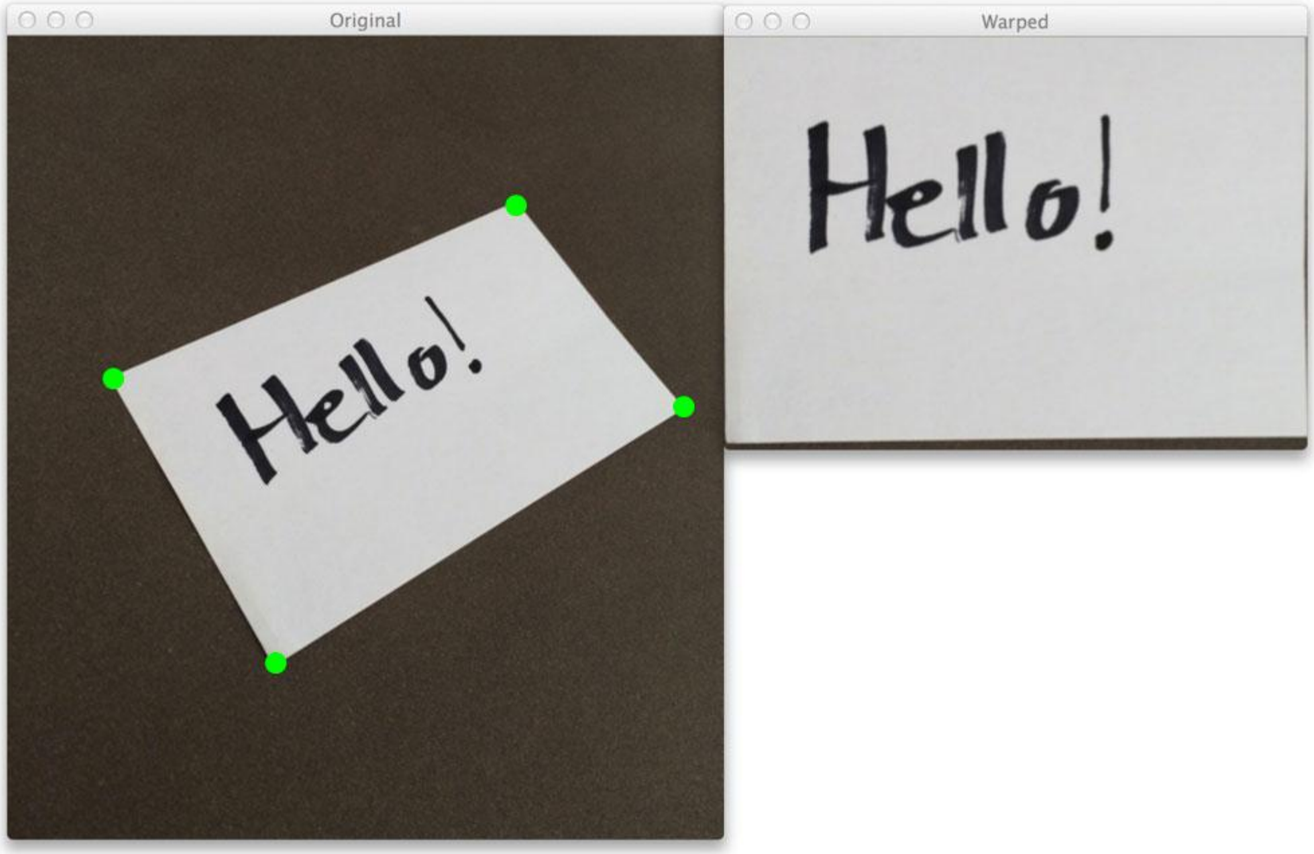


PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

TRANSFORMACIÓN DE PERSPECTIVA

- No importa si la región no está en modo horizontal o vertical en la imagen



PROCESAMIENTO DIGITAL DE IMÁGENES

Ejemplo:

```python
import cv2
import numpy as np

image = cv2.imread("sudoku.jpg")
height, width, channels = image.shape

points_1 = np.float32([[56, 65],
                       [368, 52],
                       [28, 387],
                       [389, 390]])

points_2 = np.float32([[0, 0],
                       [300, 0],
                       [0, 300],
                       [300, 300]])

matrix_transformation = cv2.getPerspectiveTransform(points_1, points_2)

image_transformed = cv2.warpPerspective(image, matrix_transformation, (300, 300))

for point in points_1:
    x, y = point
    point = tuple([int(x), int(y)])

    cv2.circle(image, point, 3, (0, 255, 0), -1)

for point in points_2:
    x, y = point
    point = tuple([int(x), int(y)])

    cv2.circle(image_transformed, point, 3, (0, 255, 0), -1)

cv2.imshow("Imagen", image)
cv2.imshow("Imagen Transformada", image_transformed)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO