Transformaciones

Araguás, Gastón   Redolfi, Javier

10 de abril del 2019

Traslación

- Usamos la función
  `dst = cv2.warpAffine(src, M, dsize[, dst])`
- Es una función genérica que nos permite aplicar una matriz de transformación **`M`** de tamaño 2×3
  - **`src`** es la imagen a transformar
  - **`M`** es la matriz de transformación, ver más abajo
  - **`dsize`** es el tamaño de la imagen de salida
  - **`dst`** imagen de salida que le podemos pasar en forma opcional, caso contrario es el valor de retorno de la función

Traslación

- Usamos la función
  `dst = cv2.warpAffine(src, M, dsize[, dst])`
- Es una función genérica que nos permite aplicar una matriz de transformación **`M`** de tamaño 2×3
  - **`src`** es la imagen a transformar
  - **`M`** es la matriz de transformación, ver más abajo
  - **`dsize`** es el tamaño de la imagen de salida
  - **`dst`** imagen de salida que le podemos pasar en forma opcional, caso contrario es el valor de retorno de la función

En el caso de una **`traslación`** la matriz **`M`** nos queda:

$$\begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \end{bmatrix}$$

Código para realizar una traslación

```python
#! /usr/bin/env python
# -*- coding: utf-8 -*-


import numpy as np
import cv2


def translate(image, x, y):
    (h, w) = (image.shape[0], image.shape[1])

    M = np.float32([[1, 0, x],
                    [0, 1, y]])

    shifted = cv2.warpAffine(image, M, (w, h))

    return shifted
```

Rotación

- Primero usamos el método `M = cv2.getRotationMatrix2D(center, angle, scale)` para calcular la matriz de rotación
  - `center` es el centro de rotación en la imagen de entrada, `(x, y)`
  - `angle` es el ángulo de rotación, entendido en sentido antihorario y suponiendo como origen la coordenada superior izquierda
  - `scale` factor de escala isotrópico
  - `M` es la matriz de rotación que nos devuelve el método
- Y luego usamos esa matriz obtenida con el método `cv2.warpAffine`

Rotación

- Primero usamos el método **`M = cv2.getRotationMatrix2D(center, angle, scale)`** para calcular la matriz de rotación
  - **`center`** es el centro de rotación en la imagen de entrada, **`(x, y)`**
  - **`angle`** es el ángulo de rotación, entendido en sentido antihorario y suponiendo como origen la coordenada superior izquierda
  - **`scale`** factor de escala isotrópico
  - **`M`** es la matriz de rotación que nos devuelve el método
- Y luego usamos esa matriz obtenida con el método **`cv2.warpAffine`**

En el caso de la rotación la matriz **`M`** nos queda:

$$\begin{bmatrix} \text{s}\cdot\cos(\text{angle}) & \text{s}\cdot\sin(\text{angle}) & [1\text{-s}\cdot\cos(\text{angle})]\text{x-}\sin(\text{angle})\cdot\text{y} \\ \text{-s}\cdot\sin(\text{angle}) & \text{s}\cdot\cos(\text{angle}) & \text{s}\cdot\sin(\text{angle})\cdot\text{x-}(1\text{-s}\cdot\cos(\text{angle}))\cdot\text{y} \end{bmatrix}$$

donde s es el factor de escala y (x, y) son las coordenadas del centro de rotación.

Código para realizar una rotación

```python
#! /usr/bin/env python
# -*- coding: utf-8 -*-


import cv2


def rotate(image, angle, center=None, scale=1.0):
    (h, w) = image.shape[:2]

    if center is None:
        center = (w/2, h/2)

    M = cv2.getRotationMatrix2D(center, angle, scale)

    rotated = cv2.warpAffine(image, M, (w, h))

    return rotated
```

Práctica 0 - Rotación + Traslación (o Transformación Euclidiana)

Práctica 0 - Rotación + Traslación (o Transformación Euclidiana)

Combinar las dos transformaciones anteriores para aplicar una transformación euclidiana (traslación + rotación) a una imagen. Crear un método nuevo que haga esto y que reciba los siguientes parámetros:

- Parámetros
  - **`angle`**: Ángulo
  - **`tx`**: traslación en x
  - **`ty`**: traslación en y

Práctica 0 - Rotación + Traslación (o Transformación Euclidiana)

Combinar las dos transformaciones anteriores para aplicar una transformación euclidiana (traslación + rotación) a una imagen. Crear un método nuevo que haga esto y que reciba los siguientes parámetros:

- Parámetros
  - **`angle`**: Ángulo
  - **`tx`**: traslación en x
  - **`ty`**: traslación en y

Recordemos que la transformación euclidiana tiene la siguiente forma:

$$\begin{bmatrix} \cos(\text{angle}) & \sin(\text{angle}) & \text{tx} \\ \text{-}\sin(\text{angle}) & \cos(\text{angle}) & \text{ty} \end{bmatrix}$$

Práctica 1 - Transformación de similaridad

Práctica 1 - Transformación de similaridad

Agregarle un parámetro al método anterior para permitir también el escalado de la imagen.

- Parámetros
  - **`angle`**: Ángulo
  - **`tx`**: traslación en x
  - **`ty`**: traslación en y
  - **`s`**: escala

Práctica 1 - Transformación de similaridad

Agregarle un parámetro al método anterior para permitir también el escalado de la imagen.

- Parámetros
  - **`angle`**: Ángulo
  - **`tx`**: traslación en x
  - **`ty`**: traslación en y
  - **`s`**: escala

Recordemos que la transformación de similaridad tiene la siguiente forma:

$$\begin{bmatrix} s\cdot \cos(\text{angle}) & s\cdot \sin(\text{angle}) & tx \\ -s\cdot \sin(\text{angle}) & s\cdot \cos(\text{angle}) & ty \end{bmatrix}$$

Espejado

Espejado

- Usamos la función
  `dst = cv2.flip(src, flipCode[, dst])`
- **`src`** es la imagen a transformar
- **`flipCode`** es un entero que indica la forma del espejado:
  - **`0`** indica espejado sobre el eje **`x`**
  - **`1`** indica espejado sobre el eje **`y`**
  - **`-1`** indica espejado sobre ambos ejes
- **`dst`** imagen de salida que le podemos pasar en forma opcional, caso contrario es el valor de retorno de la función

Código para realizar un espejado

```python
#! /usr/bin/env python
# -*- coding: utf-8 -*-


import cv2


modes = {'x': 0, 'y': 1, 'b': -1}


def flip(img, mode):
    if(mode not in modes.keys()):
        return img

    flipped = cv2.flip(img, modes[mode])

    return flipped
```

Práctica 2 - Transformación afín

Práctica 2 - Transformación afín

Recordemos

- Una transformación afín se representa con una matriz de 2×3
- Tiene **6** grados de libertad y puede ser recuperada con **3** puntos

Práctica 2 - Transformación afín

Recordemos

- Una transformación afín se representa con una matriz de 2×3
- Tiene **6** grados de libertad y puede ser recuperada con **3** puntos

Práctica 2 - Incrustando imágenes

- Crear un programa que permita seleccionar con el mouse 3 puntos de una primera imagen.
- Luego crear un método que compute una transformación afín entre las esquinas de una segunda imagen y los 3 puntos seleccionados.
- Por último aplicar esta transformación sobre la segunda imagen, e incrustarla en la primera imagen.

Práctica 2 - Transformación afín

Recordemos

- Una transformación afín se representa con una matriz de 2×3
- Tiene **6** grados de libertad y puede ser recuperada con **3** puntos

Práctica 2 - Incrustando imágenes

- Crear un programa que permita seleccionar con el mouse 3 puntos de una primera imagen.
- Luego crear un método que compute una transformación afín entre las esquinas de una segunda imagen y los 3 puntos seleccionados.
- Por último aplicar esta transformación sobre la segunda imagen, e incrustarla en la primera imagen.

Ayuda

- **`cv2.getAffineTransform`**
- **`cv2.warpAffine`**
- Generar una máscara para insertar una imagen en otra

Práctica 3 - Homografía o Transformación Perspectiva

Práctica 3 - Homografía o Transformación Perspectiva

Recordemos

- Una homografía se representa con una matriz de 3×3
- Tiene **8** grados de libertad y puede ser recuperada con **4** puntos

Práctica 3 - Homografía o Transformación Perspectiva

Recordemos

- Una homografía se representa con una matriz de 3×3
- Tiene **8** grados de libertad y puede ser recuperada con **4** puntos

Práctica 3 - Rectificando imágenes

- Crear un programa que permita seleccionar 4 puntos de una primera imagen.
- Luego crear un método que compute una homografía entre dichos 4 puntos y una segunda imagen estándar de $m \times n$ pixeles.
- Por último aplicar esta transformación para llevar (rectificar) la primera imagen a la segunda de $m \times n$.

Práctica 3 - Homografía o Transformación Perspectiva

Recordemos

- Una homografía se representa con una matriz de 3×3
- Tiene **8** grados de libertad y puede ser recuperada con **4** puntos

Práctica 3 - Rectificando imágenes

- Crear un programa que permita seleccionar 4 puntos de una primera imagen.
- Luego crear un método que compute una homografía entre dichos 4 puntos y una segunda imagen estándar de $m \times n$ pixeles.
- Por último aplicar esta transformación para llevar (rectificar) la primera imagen a la segunda de $m \times n$.

Ayuda

- **`cv2.getPerspectiveTransform`**
- **`cv2.warpPerspective`**