Escáner de Códigos QR

Un código QR (código de respuesta rápida) es un tipo de código de barras matricial bidimensional, inventado en 1994 por la empresa japonesa Denso Wave para etiquetar piezas de automóviles. Presenta cuadrados negros sobre un fondo blanco con marcadores fiduciales, que pueden ser leídos por dispositivos de imagen como cámaras.

Los datos necesarios se extraen de los patrones que están presentes tanto en los componentes horizontales como en los verticales de la imagen del código QR.

En Python, la librería más popular para leer códigos QR es `pyzbar`, pero en este proyecto utilizaremos un módulo añadido a partir de la versión 4.0 de OpenCV, llamado `cv2.QRCodeDetector`.

Proyecto

El proyecto consiste en lo siguiente:

1. Capturar el video de la cámara web.
2. Si se presiona la tecla `q`, se cierra la ventana de video.
3. Si se detecta algún código QR en el video, se muestra la información del código QR sobre el, y se dibuja un rectángulo que rodea al código QR.
4. Si se presiona la tecla `s`, se guarda una captura de pantalla de la imagen actual en la carpeta `output` (pueden guardarse múltiples capturas de pantalla).
5. Si se presiona la tecla `m`, se enmascaran los códigos QR detectados en la imagen.
6. Si se vuelve a presionar la tecla `m`, se desenmascaran los códigos QR detectados en la imagen.
7. Si no se detecta ningún código QR en el video, se muestra el mensaje "No se detectó ningún código QR" en rojo.

Ejemplo imagen de entrada

Imagen de salida

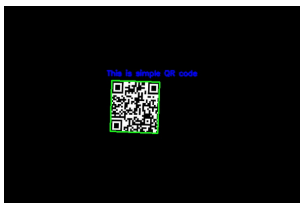

Imagen de salida enmascarada

Documentación:

1. Crear un objeto de la clase `cv2.QRCodeDetector`.

```
qrDecoder = cv2.QRCodeDetector()
```

2. Leer el código QR de una imagen.

```
data, bbox, rectifiedImage = qrDecoder.detectAndDecode(image)
```

- `data`: Contiene la información del código QR. Si no se detecta ningún código QR, `data` será una cadena vacía.
- `bbox`: Es una lista de las coordenadas de los vértices del rectángulo que rodea al código QR. (4 puntos)
- `rectifiedImage`: Es la imagen rectificada del código QR, es decir, la imagen del código QR enderezada y recortada.

3. También podemos leer múltiples códigos QR en una sola imagen utilizando el método `detectAndDecodeMulti`.

```
ret, decoded_info, points, straight_qrcode =
qrDecoder.detectAndDecodeMulti(image)
```

- `decoded_info`: Es una lista que contiene la información de todos los códigos QR detectados en la imagen.
- `points`: Es una lista de listas, donde cada sublista contiene las coordenadas de los vértices del rectángulo que rodea a un código QR.
- `straight_qrcode`: Es una lista de imágenes rectificadas de los códigos QR detectados.
- `ret`: Es un valor booleano que indica si se detectaron códigos QR en la imagen.

4. Para dibujar el rectángulo que rodea al código QR en la imagen, podemos utilizar la función `cv2.polylines`, la cual dibuja un polígono en una imagen.

```
cv2.polylines(image, [bbox], True, color, thickness)
```

- `image`: Es la imagen en la que se dibujará el rectángulo.
- `bbox`: Es una lista de las coordenadas de los vértices del rectángulo que rodea al código QR. **El tipo de dato de `bbox` debe ser `np.uint32`**.
- `True`: Indica que el polígono es cerrado.
- `color`: Es el color del polígono.
- `thickness`: Es el grosor del polígono.

5. Convertir las coordenadas de los vértices del rectángulo a un tipo de dato `np.uint32`.

```
bbox = np.array(bbox, dtype=np.uint32) //Método 1
bbox = bbox.astype(np.int32) //Método 2
```

6. Rellenar un polígono en una imagen.

```
cv2.fillPoly(image, [bbox], color)
```

- `image`: Es la imagen en la que se rellenará el polígono.
- `bbox`: Es una lista de las coordenadas de los vértices del rectángulo que rodea al código QR.
- `color`: Es el color del polígono.