

Tutorial de procesar imágenes en Python (usando OpenCV) - Like Geeks

Mokhtar Ebrahim

19-25 minutos

En este tutorial, aprenderás cómo puedes procesar imágenes en Python usando la biblioteca OpenCV.

OpenCV es una librería gratuita de código abierto utilizada en el procesamiento de imágenes en tiempo real. Se utiliza para procesar imágenes, vídeos e incluso secuencias en directo, pero en este tutorial, procesaremos las imágenes sólo como un primer paso. Antes de empezar, vamos a instalar OpenCV.

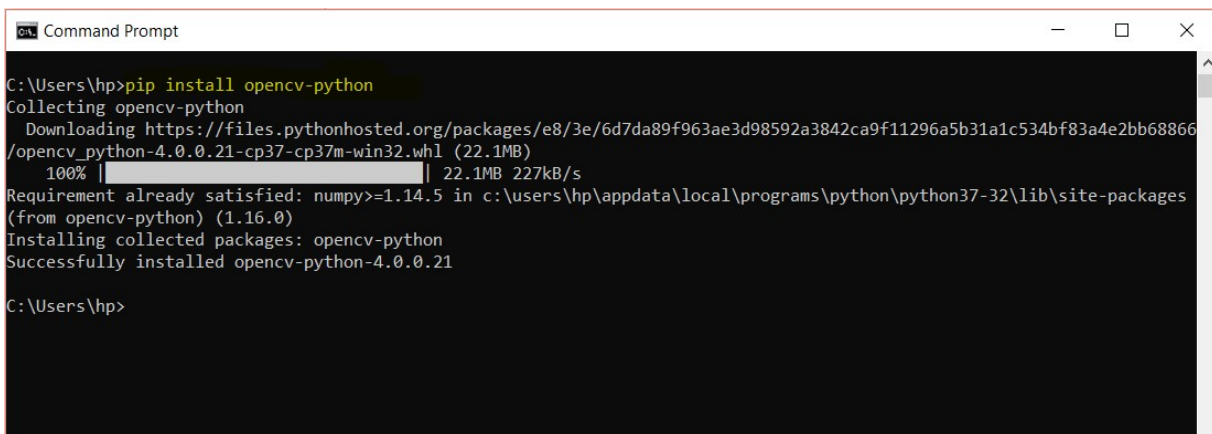
- [Instalar OpenCV](#)
- [Rotar una imagen](#)
- [Recortar una imagen](#)
- [Cambiar el tamaño de una imagen](#)
- [Ajustar el contraste de la imagen](#)
- [Hacer una imagen borrosa](#)
- [Borrón Gaussiano](#)
- [Median Blur](#)

- [Detectar los bordes](#)
- [Convertir la imagen a escala de grises \(Blanco y Negro\)](#)
- [Detección del centroide \(Centro de la mancha\)](#)
- [Aplicar una máscara para una imagen coloreada](#)
- [Extracción de texto de la imagen \(OCR\)](#)
- [Detecte y corrija la inclinación del texto](#)
- [Detección de color](#)
- [Reducir el ruido](#)
- [Obtener el contorno de la imagen](#)
- [Eliminar el fondo de una imagen](#)

Instalar OpenCV

Para instalar OpenCV en su sistema, ejecute el siguiente [comando pip](#):

```
pip install opencv-python
```



```
Command Prompt
C:\Users\hp>pip install opencv-python
Collecting opencv-python
  Downloading https://files.pythonhosted.org/packages/e8/3e/6d7da89f963ae3d98592a3842ca9f11296a5b31a1c534bf83a4e2bb68866/opencv_python-4.0.0.21-cp37-cp37m-win32.whl (22.1MB)
    100% |#####| 22.1MB 227kB/s
Requirement already satisfied: numpy>=1.14.5 in c:\users\hp\appdata\local\programs\python\python37-32\lib\site-packages (from opencv-python) (1.16.0)
Installing collected packages: opencv-python
Successfully installed opencv-python-4.0.0.21
C:\Users\hp>
```

Ahora OpenCV se ha instalado con éxito y estamos listos.

¡Vamos a divertirnos con algunas imágenes!

Rotar una imagen

En primer lugar, importa el módulo cv2

```
import cv2
```

Ahora para leer la imagen, utilice el método `imread()` del módulo `cv2`, especifique la ruta de la imagen en los argumentos y guarde la imagen en una variable como se indica a continuación:

```
img = cv2.imread("pyimg.jpg")
```

La imagen se trata ahora como una matriz con valores de filas y columnas almacenados en `img`.

En realidad, si comprueba el tipo de la `img`, le dará el siguiente resultado:

```
>>>print(type(img))
```

```
<class 'numpy.ndarray'>
```

¡Es un [NumPy array](#)! Por eso el procesamiento de imágenes usando OpenCV es tan fácil. Todo el tiempo estás trabajando con un arreglo de NumPy.

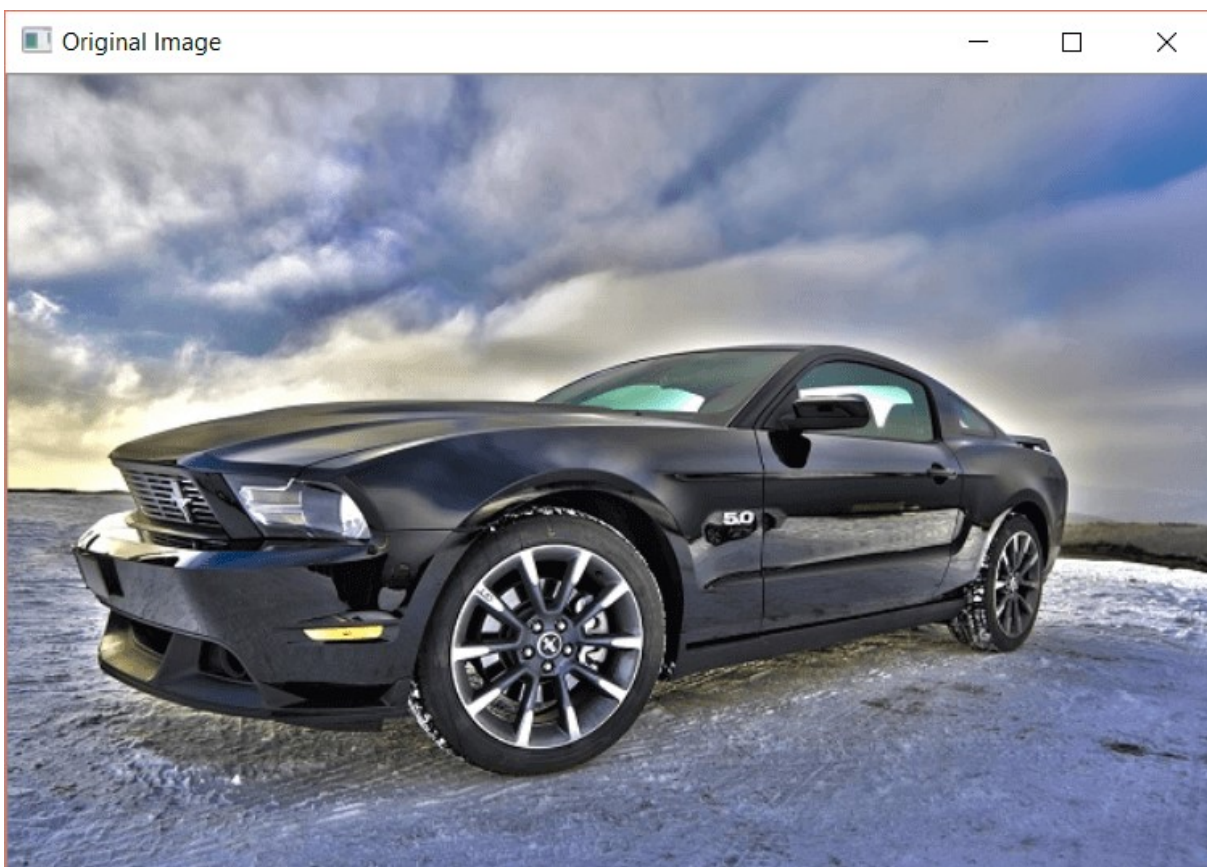
Para mostrar la imagen, puedes usar el método `imshow()` de `cv2`.

```
cv2.imshow('Original Image', img)
```

```
cv2.waitKey(0)
```

Las funciones de la tecla de espera toman el tiempo como argumento en milisegundos como un retardo para que la

ventana se cierre. Aquí ponemos el tiempo a cero para mostrar la ventana para siempre hasta que la cerremos manualmente.



Para rotar esta imagen, necesitas el ancho y el alto de la imagen porque los usarás en el proceso de rotación como verás más adelante.

```
height, width = img.shape[0:2]
```

El atributo `shape` devuelve la altura y el ancho de la matriz de la imagen. Si imprime, ,tendrá la siguiente salida:

```
>>> import cv2
>>> img = cv2.imread("pyimg.jpg")
>>> print(img.shape[0:2])
(432, 768)
```

Bien, ahora tenemos nuestra matriz de imagen y queremos obtener la matriz de rotación. Para obtener la matriz de rotación, usamos el método `getRotationMatrix2D()` de `cv2`. La sintaxis de

getRotationMatrix2D() es:

```
cv2.getRotationMatrix2D(center, angle, scale)
```

Aquí el centro es el punto central de rotación, el ángulo es el ángulo en grados y la escala es la propiedad de escala que hace que la imagen se ajuste a la pantalla.

Para obtener la matriz de rotación de nuestra imagen, el código será:

```
rotationMatrix =  
cv2.getRotationMatrix2D((width/2, height/2),  
90, .5)
```

El siguiente paso es rotar nuestra imagen con la ayuda de la matriz de rotación.

Para rotar la imagen, tenemos un método cv2 llamado `warpAffine` que toma como argumentos la imagen original, la matriz de rotación de la imagen y el ancho y alto de la imagen.

```
rotatedImage = cv2.warpAffine(img,  
rotationMatrix, (width, height))
```

La imagen rotada se almacena en la matriz `rotatedImage`. Para mostrar la imagen, usa `imshow()` como se indica a continuación:

```
cv2.imshow('Rotated Image', rotatedImage)
```

```
cv2.waitKey(0)
```

Después de ejecutar las líneas de código anteriores, tendrás la siguiente salida:





Recortar una imagen

Primero, necesitamos importar el módulo cv2 y leer la imagen y extraer el ancho y el alto de la misma:

```
import cv2
```

```
img = cv2.imread("pyimg.jpg")
```

```
height, width = img.shape[0:2]
```

Ahora obtenga el índice inicial y final de la fila y la columna. Esto definirá el tamaño de la imagen recién creada. Por ejemplo, desde la fila número 10 hasta la fila número 15, se mostrará la altura de la imagen.

De manera similar, comience desde la columna número 10 hasta la columna número 15 dará el ancho de la imagen.

Puede obtener el punto de partida especificando el valor porcentual de la altura total y el ancho total. De manera similar, para obtener el punto final de la imagen recortada, especifique los valores porcentuales como se indica a continuación:

```
startRow = int(height*.15)
```

```
startCol = int(width*.15)
```

```
endRow = int(height*.85)
```

```
endCol = int(width*.85)
```

Ahora mapee estos valores a la imagen original. Tenga en cuenta que tiene que asignar los valores iniciales y finales a números enteros porque cuando se mapea, los índices son siempre números enteros.

```
croppedImage = img[startRow:endRow,  
startCol:endCol]
```

Aquí especificamos el rango de principio a fin de las filas y columnas

Ahora muestre la imagen original y recortada en la salida:

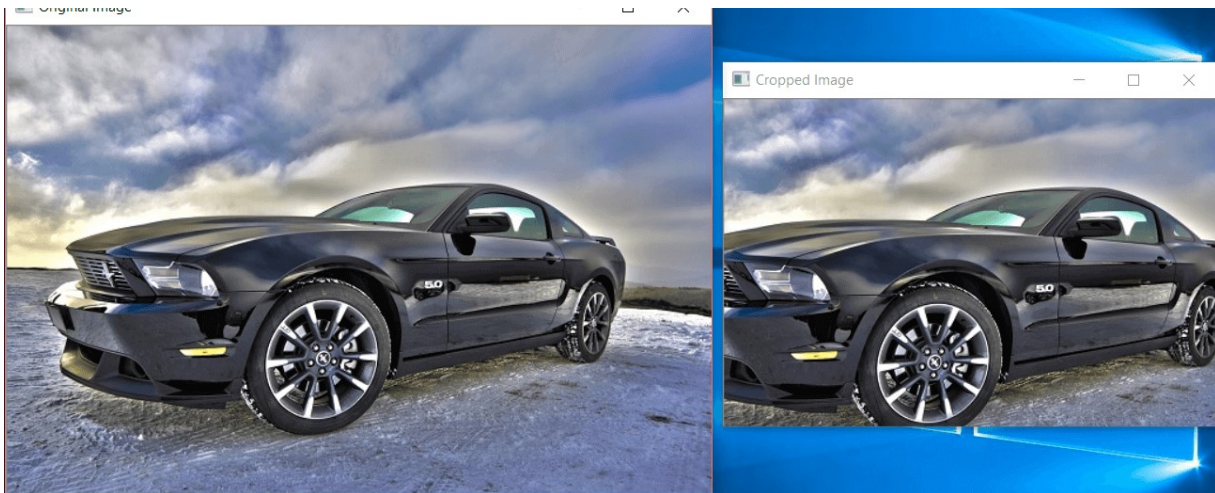
```
cv2.imshow('Original Image', img)
```

```
cv2.imshow('Cropped Image', croppedImage)
```

```
cv2.waitKey(0)
```

El resultado será el siguiente:





Cambiar el tamaño de una imagen

Para redimensionar una imagen, puede usar el método `resize()` de `openCV`. En el método `resize`, puede especificar los valores de los ejes `x` y `y` o el número de filas y columnas que indica el tamaño de la imagen.

Importar y leer la imagen:

```
import cv2
```

```
img = cv2.imread("pyimg.jpg")
```

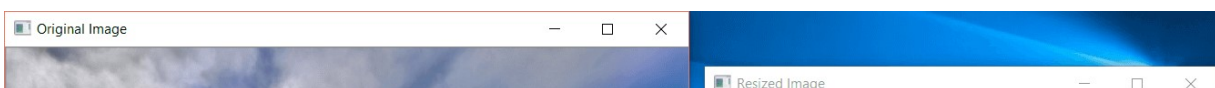
Ahora usando el método `resize` con los valores de los ejes:

```
newImg = cv2.resize(img, (0,0), fx=0.75,  
fy=0.75)
```

```
cv2.imshow('Resized Image', newImg)
```

```
cv2.waitKey(0)
```

El resultado será el siguiente:





Ahora usando los valores de filas y columnas para redimensionar la imagen:

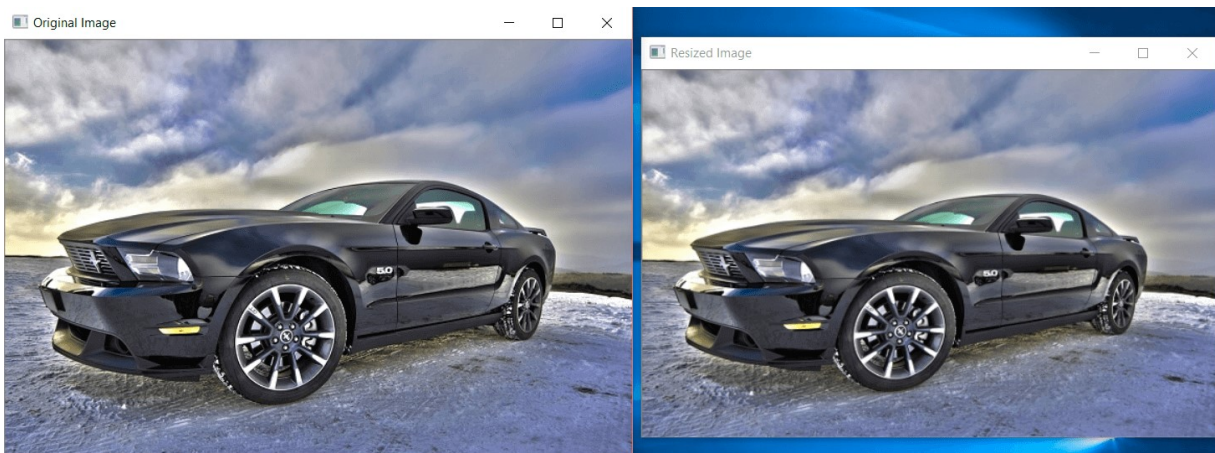
```
newImg = cv2.resize(img, (550, 350))
```

```
cv2.imshow('Resized Image', newImg)
```

```
cv2.waitKey(0)
```

Decimos que queremos 550 columnas (el ancho) y 350 filas (la altura).

El resultado será:



Ajustar el contraste de la imagen

En el módulo OpenCV de Python no hay una función particular para ajustar el contraste de la imagen, pero la documentación

oficial de OpenCV sugiere una ecuación que puede realizar el brillo y el contraste de la imagen al mismo tiempo.

```
new_img = a * original_img + b
```

Aquí a es alfa que define el contraste de la imagen. Si a es mayor que 1, habrá un mayor contraste.

Si el valor de a está entre 0 y 1 (menor que 1 pero mayor que 0), habrá menor contraste. Si a es 1, no habrá efecto de contraste en la imagen.

b significa beta. Los valores de b varían de -127 a +127.

Para implementar esta ecuación en Python OpenCV, puedes usar el método `addWeighted()`. Usamos el método `addWeighted()` ya que genera la salida en el rango de 0 y 255 para una imagen en color de 24 bits.

La sintaxis del método `addWeighted()` es la siguiente:

```
cv2.addWeighted(source_img1, alpha1,  
source_img2, alpha2, beta)
```

Esta sintaxis mezclará dos imágenes, la primera imagen fuente (`source_img1`) con un peso de α_1 y la segunda imagen fuente (`source_img2`).

Si sólo quiere aplicar contraste en una imagen, puede añadir una segunda imagen fuente como ceros usando NumPy.

Trabajemos en un ejemplo sencillo. Importe los siguientes módulos:

```
import cv2
```

```
import numpy as np
```

Lea la imagen original:

```
img = cv2.imread("pyimg.jpg")
```

Ahora aplique el contraste. Como no hay otra imagen, usaremos el `np.zeros` que creará una matriz de la misma forma y tipo de datos que la imagen original pero la matriz se llenará de ceros.

```
contrast_img = cv2.addWeighted(img, 2.5,  
np.zeros(img.shape, img.dtype), 0, 0)
```

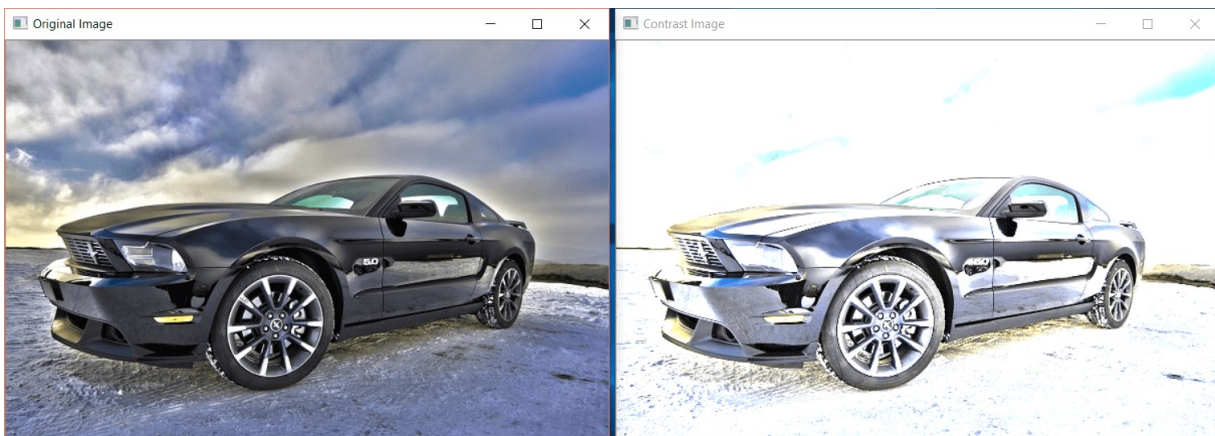
```
cv2.imshow('Original Image', img)
```

```
cv2.imshow('Contrast Image', contrast_img)
```

```
cv2.waitKey(0)
```

En el código anterior, el brillo se establece en 0 ya que sólo queremos aplicar el contraste.

La comparación de la imagen original y el contraste es la siguiente:



Hacer una imagen borrosa

Borrón Gaussiano

Para hacer una imagen borrosa, puede usar el método `GaussianBlur()` de OpenCV.

El `GaussianBlur()` utiliza el núcleo Gaussiano. La altura y el ancho del kernel deben ser un número positivo e impar.

Luego hay que especificar la dirección X e Y que es `sigmaX` y `sigmaY` respectivamente. Si sólo se especifica una, ambas se consideran iguales.

Considere el siguiente ejemplo:

```
import cv2

img = cv2.imread("pyimg.jpg")

blur_image = cv2.GaussianBlur(img, (7,7), 0)

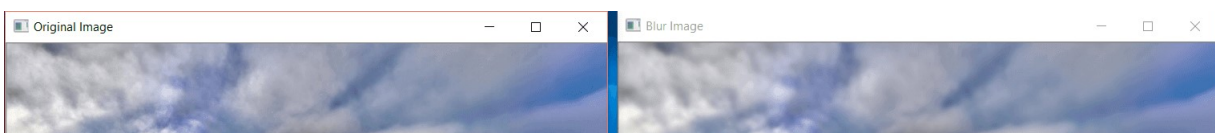
cv2.imshow('Original Image', img)

cv2.imshow('Blur Image', blur_image)

cv2.waitKey(0)
```

En el recorte anterior, la imagen real se pasa a `GaussianBlur()` junto con la altura y el ancho del núcleo y las direcciones X e Y.

La comparación de la imagen original y la borrosa es la siguiente:





Median Blur

En el desenfoque mediano, la mediana de todos los píxeles de la imagen se calcula dentro del área del núcleo. El valor central es entonces reemplazado por el valor mediano resultante. El desenfoque mediano se utiliza cuando hay ruido de sal y pimienta en la imagen.

Para aplicar el difuminado medio, puede usar el método `medianBlur()` de OpenCV.

Considere el siguiente ejemplo donde tenemos un ruido de sal y pimienta en la imagen:

```
import cv2

img = cv2.imread("pynoise.png")

blur_image = cv2.medianBlur(img, 5)

Esto aplicará un 50% de ruido en la imagen junto con el
desenfoque medio. Ahora muestre las imágenes:

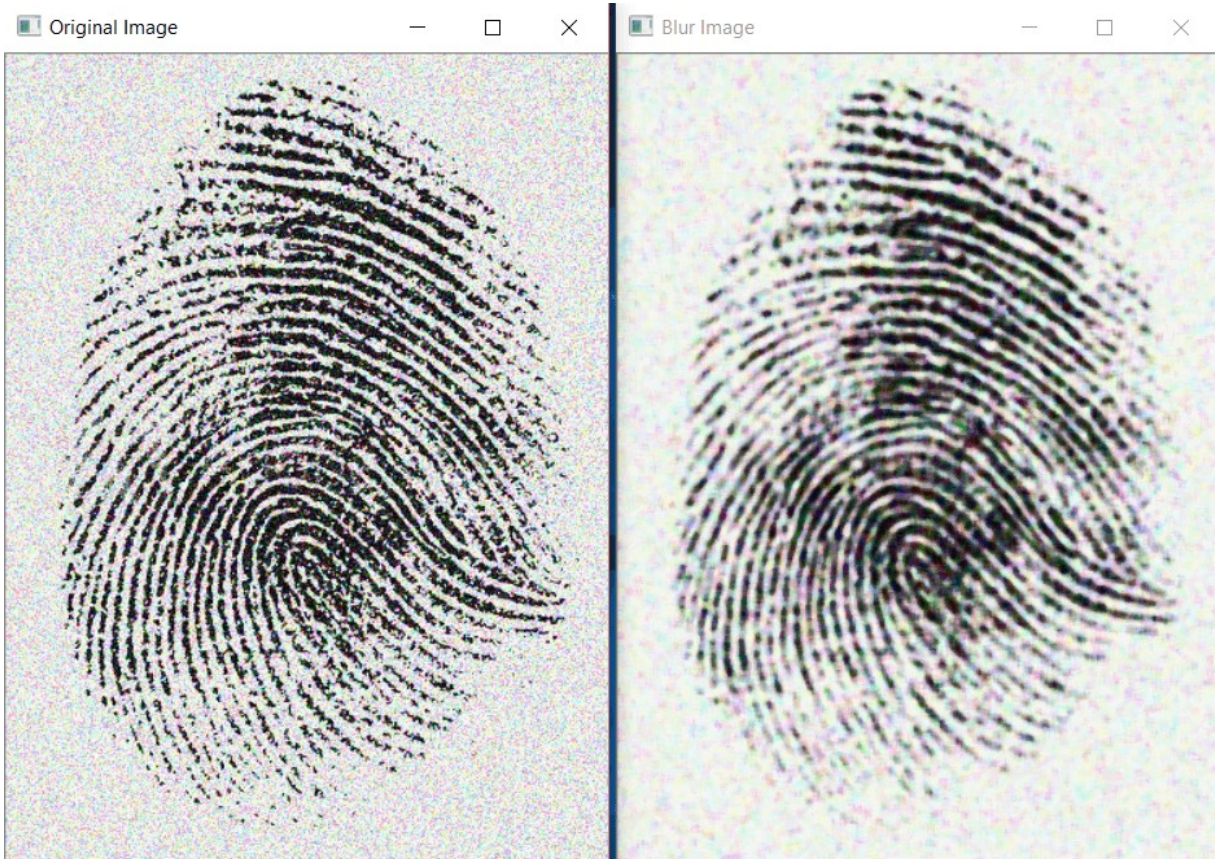
cv2.imshow('Original Image', img)

cv2.imshow('Blur Image', blur_image)
```

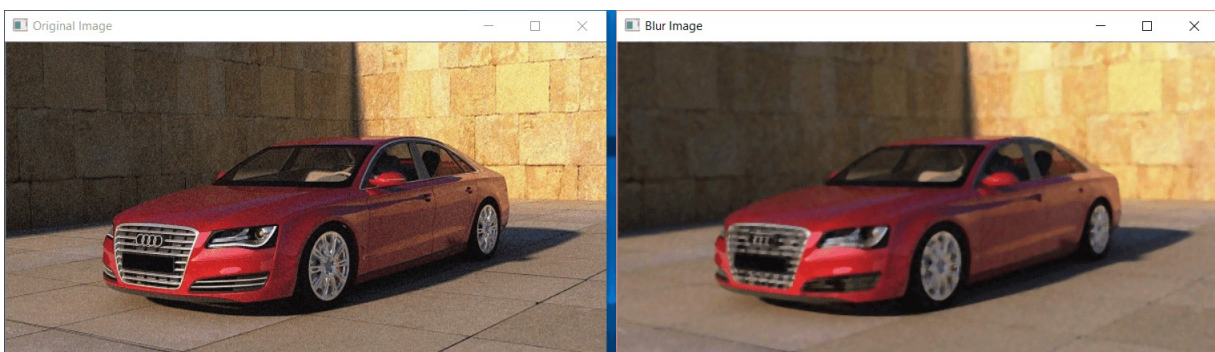


```
cv2.waitKey(0)
```

El resultado será como el siguiente:



Otra comparación de la imagen original y después del desenfoque:



Detectar los bordes

Para detectar los bordes en una imagen, puede usar el método `Canny()` de `cv2` que implementa el detector de bordes de Canny. El detector de bordes de Canny es también conocido como el

detector óptimo.

La sintaxis de Canny() es la siguiente:

```
cv2.Canny(image, minVal, maxVal)
```

Aquí minVal y maxVal son los valores de gradiente de intensidad mínimo y máximo respectivamente.

Considere el siguiente código:

```
import cv2

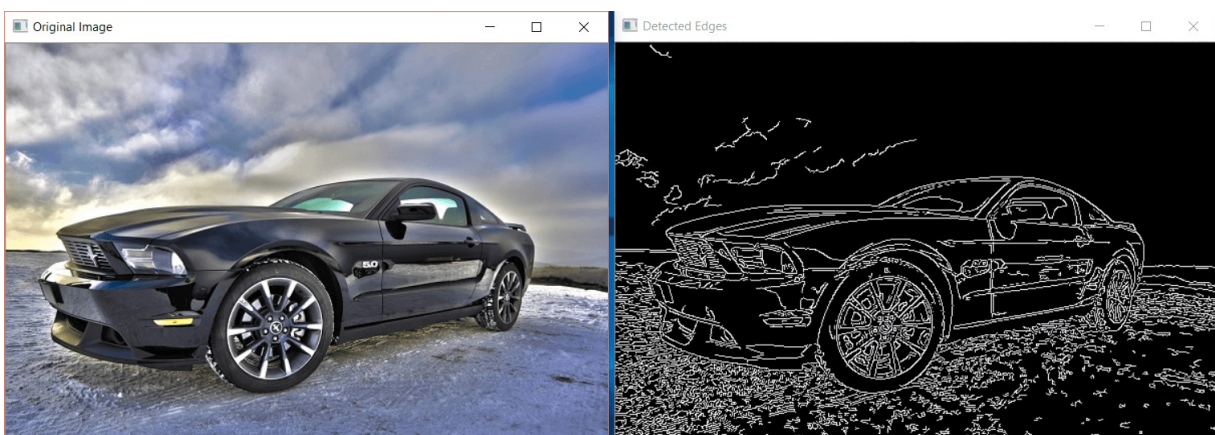
img = cv2.imread("pyimg.jpg")

edge_img = cv2.Canny(img,100,200)

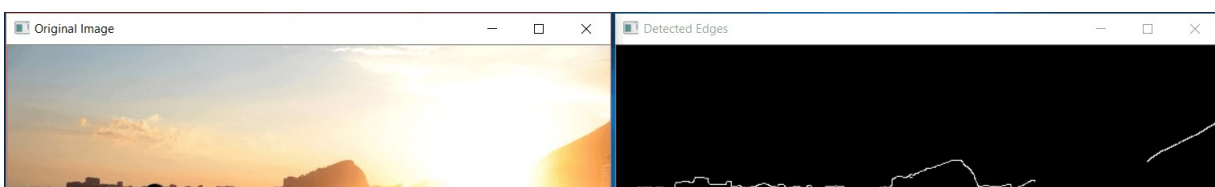
cv2.imshow("Detected Edges", edge_img)

cv2.waitKey(0)
```

La salida será la siguiente:



Aquí está el resultado del código anterior en otra imagen:





Convertir la imagen a escala de grises (Blanco y Negro)

La forma más fácil de convertir una imagen en escala de grises es cargarla así:

```
img = cv2.imread("pyimg.jpg", 0)
```

Hay otro método que usa BGR2GRAY.

Para convertir una imagen en color en una imagen en escala de grises, utilice el atributo BGR2GRAY del módulo cv2. Esto se demuestra en el siguiente ejemplo:

Importar el módulo cv2:

```
import cv2
```

Lea la imagen:

```
img = cv2.imread("pyimg.jpg")
```

Utilice el método `cvtColor()` del módulo cv2 que toma como argumento la imagen original y el atributo `COLOR_BGR2GRAY`.

Guarda la imagen resultante en una variable:

```
gray_img = cv2.cvtColor(img,  
cv2.COLOR_BGR2GRAY)
```

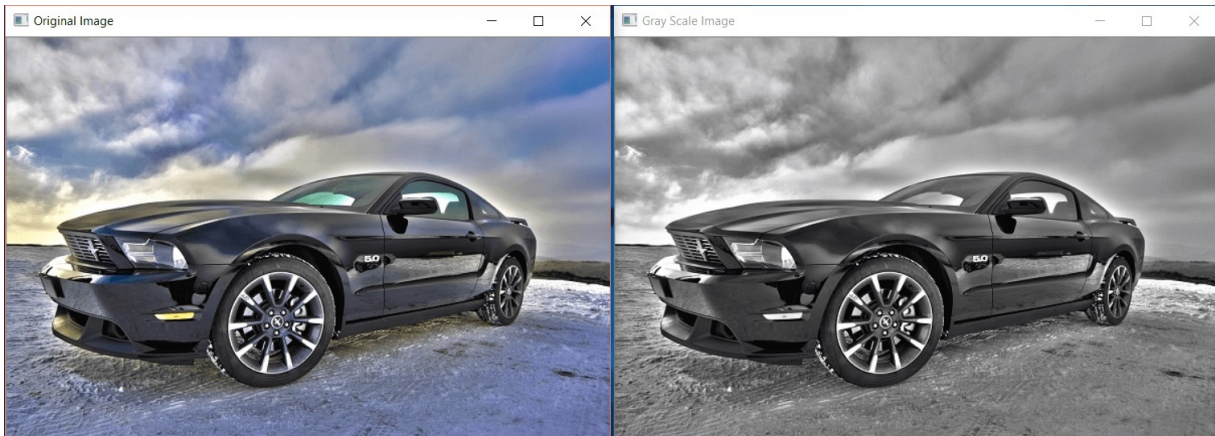
Mostrar las imágenes originales y en escala de grises:

```
cv2.imshow("Original Image", img)
```

```
cv2.imshow("Gray Scale Image", gray_img)
```

```
cv2.waitKey(0)
```

La salida será la siguiente:



Detección del centroide (Centro de la mancha)

Para encontrar el centro de una imagen, el primer paso es convertir la imagen original en escala de grises. Podemos usar el método `cvtColor` la imagen original es: `cv2.cvtColor()` de `cv2` como hicimos antes.

Esto se demuestra en el siguiente código:

```
import cv2
```

```
img = cv2.imread("py.jpg")
```

```
gray_img = cv2.cvtColor(img,  
cv2.COLOR_BGR2GRAY)
```

Leemos la imagen y la convertimos en una imagen en escala de

grises. La nueva imagen se almacena en `gray_img`.

Ahora tenemos que calcular los momentos de la imagen. Utiliza el método `moments()` de `cv2`. En el método `moments()`, la imagen en escala de grises se pasará como se indica a continuación:

```
moment = cv2.moments(gray_img)
```

Luego tenemos que calcular las coordenadas `x` y `y` del centro de la imagen usando los momentos que obtuvimos arriba:

```
X = int(moment ["m10"] / moment ["m00"])
```

```
Y = int(moment ["m01"] / moment ["m00"])
```

Finalmente, tenemos el centro de la imagen. Para resaltar esta posición del centro, podemos usar el método del círculo que creará un círculo en las coordenadas dadas del radio dado.

El método `circle()` toma la imagen, las coordenadas `x` e `y` donde se creará el círculo, el tamaño, el color que queremos que tenga el círculo y el grosor.

```
cv2.circle(img, (X, Y), 15, (205, 114, 101), 1)
```

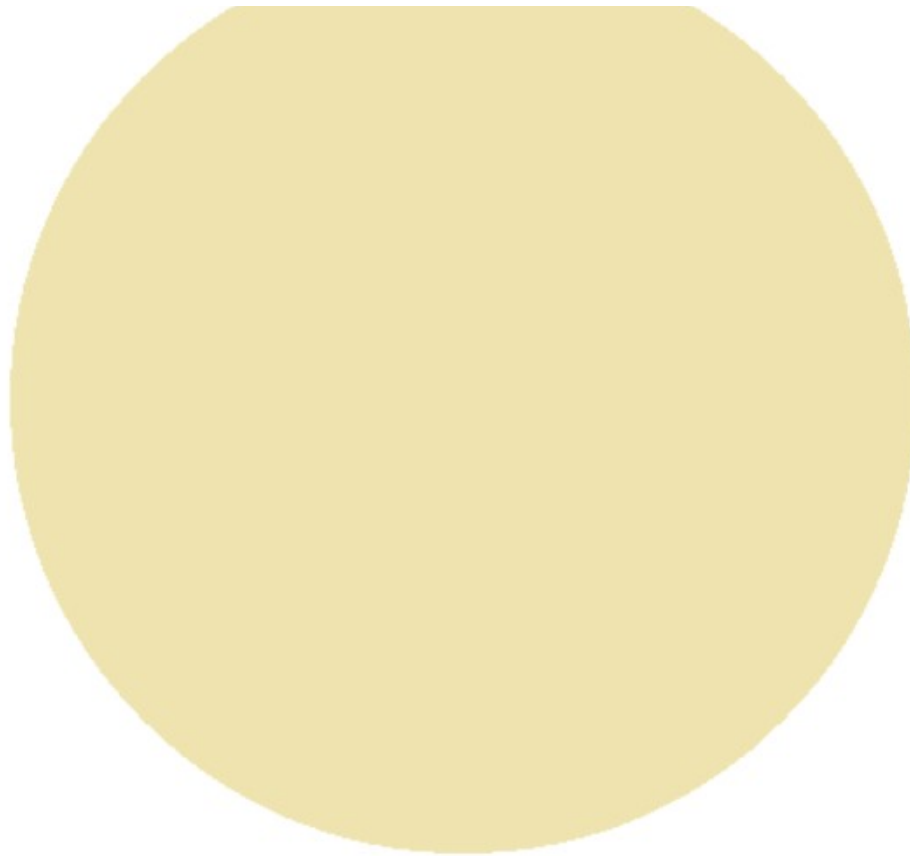
El círculo se crea sobre la imagen.

```
cv2.imshow("Center of the Image", img)
```


```
cv2.waitKey(0)
```

La imagen original es:

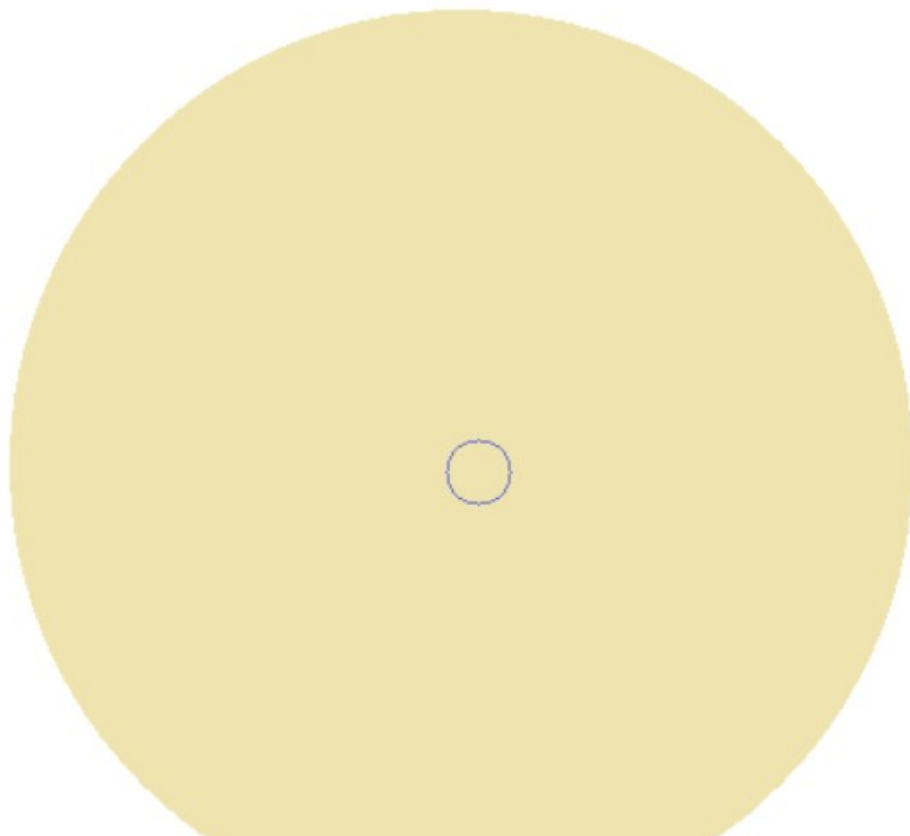




Después de detectar el centro, nuestra imagen será la siguiente:

 Center of the Image

— □ ×



Aplicar una máscara para una imagen coloreada

Enmascarar la imagen significa aplicar alguna otra imagen como una máscara en la imagen original o cambiar los valores de los píxeles en la imagen.

Para aplicar una máscara a la imagen, usaremos el método `HoughCircles()` del módulo `OpenCV`. El método `HoughCircles()` detecta los círculos en una imagen. Después de detectar los círculos, podemos simplemente aplicar una máscara sobre estos círculos.

El método `HoughCircles()` toma la imagen original, el `Hough Gradient` (que detecta la información de gradiente en los bordes del círculo), y la información de la siguiente ecuación del círculo:

$$(x - x_{center})^2 + (y - y_{center})^2 = r^2$$

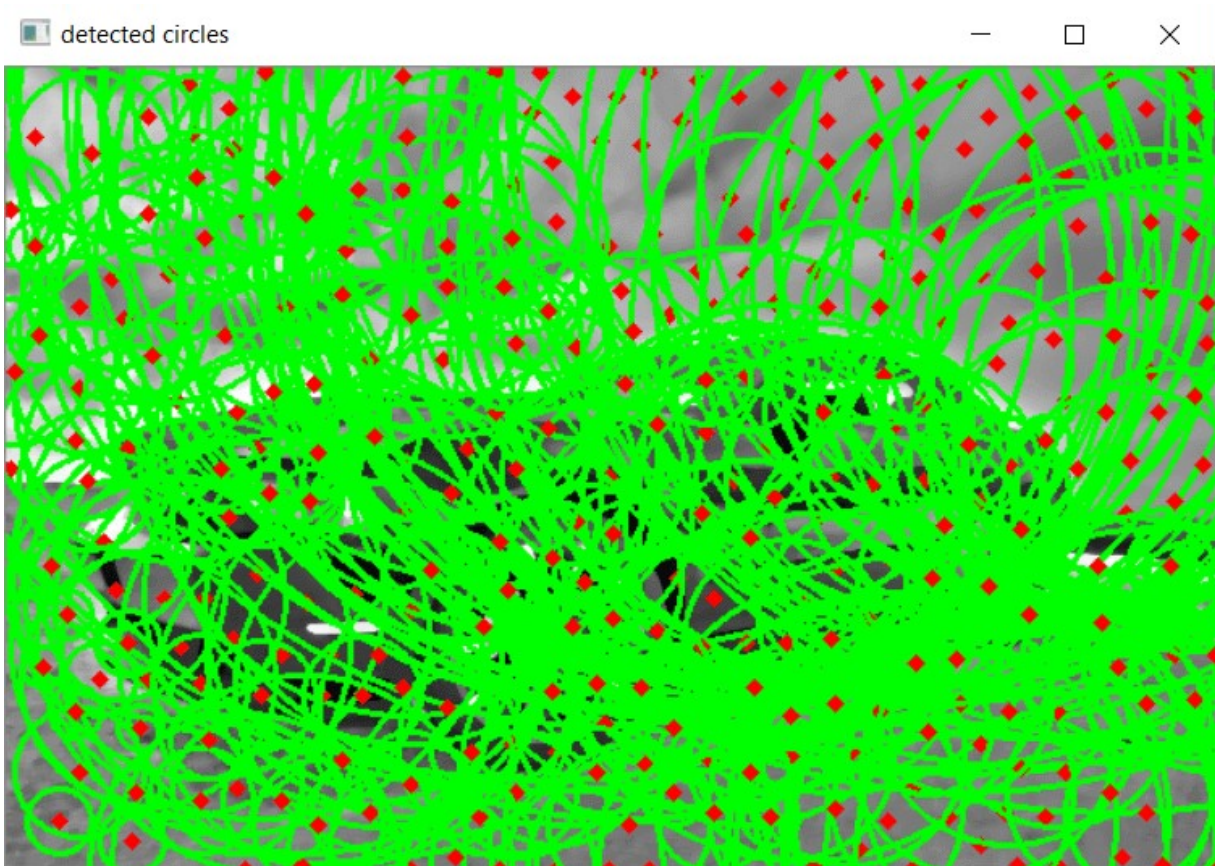
En esta ecuación (x_{centro}, y_{centro}) es el centro del círculo y r es el radio del círculo.

Nuestra imagen original es:





Después de detectar los círculos en la imagen, el resultado será:



Bien, entonces tenemos los círculos en la imagen y podemos aplicar la máscara. Considere el siguiente código:

```
import cv2

import numpy as np

img1 = cv2.imread('pyimg.jpg')
```

```
img1 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

Detección de los círculos de la imagen mediante el código `HoughCircles()` de [OpenCV: Hough Circle Transform](#):

```
gray_img = cv2.medianBlur(cv2.cvtColor(img,  
cv2.COLOR_RGB2GRAY), 3)
```

```
circles = cv2.HoughCircles(gray_img,  
cv2.HOUGH_GRADIENT, 1, 20, param1=50,  
param2=50, minRadius=0, maxRadius=0)
```

```
circles = np.uint16(np.around(circles))
```

Para crear la máscara, use `np.full` que devolverá un array NumPy de una forma dada:

```
masking=np.full((img1.shape[0],  
img1.shape[1]),0,dtype=np.uint8)
```

```
for j in circles[0, :]:
```

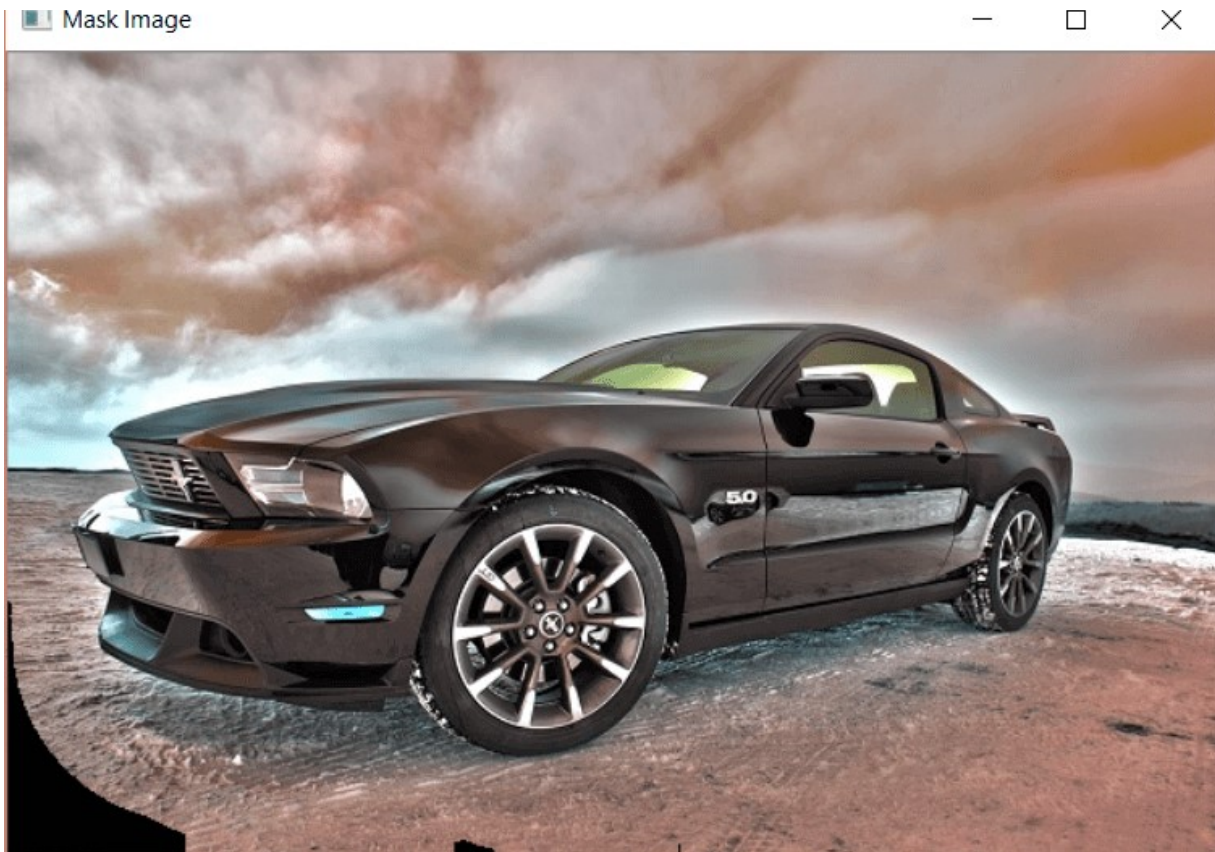
```
    cv2.circle(masking, (j[0], j[1]), j[2],  
(255, 255, 255), -1)
```

El siguiente paso es combinar la imagen y la matriz de enmascaramiento que hemos creado usando el operador *bitwise_or* de la siguiente manera:

```
final_img = cv2.bitwise_or(img1, img1,  
masking=masking)
```

se muestra la imagen resultante:





Extracción de texto de la imagen (OCR)

Para extraer el texto de una imagen, puedes usar Google Tesseract-OCR. Puedes descargarlo desde este [enlace](#)

Entonces deberías instalar el módulo pytesseract que es un wrapper de Python para Tesseract-OCR.

```
pip install pytesseract
```

```
Command Prompt
Microsoft Windows [Version 10.0.16299.19]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\hp>pip install pytesseract
Collecting pytesseract
  Downloading https://files.pythonhosted.org/packages/71/5a/d7600cad26276d991feecb27f3627ae2d0ee89aa1e3065fa4f9f1f2defbc
/pytesseract-0.2.6.tar.gz (169kB)
    100% |#####| 174kB 481kB/s
Collecting Pillow (from pytesseract)
  Downloading https://files.pythonhosted.org/packages/d7/ea/46fd5bd57c5df5a2e79e508294acec4be0fcc2fb3ce95c2cf1038ebaa533
/Pillow-5.4.1-cp37-cp37m-win32.whl (1.7MB)
    100% |#####| 1.7MB 638kB/s
Building wheels for collected packages: pytesseract
  Building wheel for pytesseract (setup.py) ... done
  Stored in directory: C:\Users\hp\AppData\Local\pip\Cache\wheels\d5\90\56\ab7b652592da86821293f7cad1c554aa376a0d57ce41
4d0a0
Successfully built pytesseract
Installing collected packages: Pillow, pytesseract
Successfully installed Pillow-5.4.1 pytesseract-0.2.6
You are using pip version 19.0.2, however version 19.0.3 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

La imagen de la que extraeremos el texto es la siguiente:



Ahora vamos a convertir el texto de esta imagen en una cadena de caracteres y mostrar el texto como una cadena en la salida:

Importar el módulo pytesseract:

```
import pytesseract
```

Establece la ruta del archivo ejecutable de Tesseract-OCR:

```
pytesseract.pytesseract.tesseract_cmd =  
r'C:\Program Files (x86)\Tesseract-  
OCR\tesseract'
```

Ahora use el *método image_to_string* para convertir la imagen en una cadena:

```
print(pytesseract.image_to_string('pytext.png'))
```

La salida será como sigue:

```
>>> import pytesseract  
>>> pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files (x86)\Tesseract-OCR\tesseract'  
>>> print(pytesseract.image_to_string('pytext.png'))  
LikeGeeks.com is a website  
which contains articles about  
Linux, programming, web  
development, iOS, Python and  
other tech tips.
```

¡Funciona como un encanto!

Detecte y corrija la inclinación del texto

En esta sección, corregiremos el sesgo del texto.

La imagen original es la siguiente:

LikeGeeks.com is a website
which contains articles about
Linux, programming, web
development, iOS, Python and
other tech tips.

Importar los módulos cv2, NumPy y leer la imagen:

```
import cv2
```

```
import numpy as np
```

```
img = cv2.imread("pytext1.png")
```

Convierte la imagen en una imagen en escala de grises:

```
gray_img=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Invertir la imagen en escala de grises usando *bitwise_not*:

```
gray_img=cv2.bitwise_not(gray_img)
```

Seleccione las coordenadas x y y de los píxeles mayores que cero usando el método `column_stack` de NumPy:

```
coordinates = np.column_stack(np.where(gray_img  
> 0))
```

Ahora tenemos que calcular el ángulo de inclinación. Usaremos el método `minAreaRect()` de `cv2` que devuelve un rango de ángulos de -90 a 0 grados (donde 0 no está incluido).

```
ang=cv2.minAreaRect(coordinates)[-1]
```

El ángulo de rotación de la región del texto será almacenado en la variable `ang`. Ahora añadimos una condición para el ángulo; si el ángulo de la región del texto es menor que -45, añadiremos 90 grados, o bien multiplicaremos el ángulo por un menos para hacer que el ángulo sea positivo.

```
if ang<-45:
```

```
    ang=- (90+ang)
```

```
else:
```

```
    ang=-ang
```

Calcule el centro de la región de texto:

```
height, width = img.shape[:2]
```

```
center_img = (width / 2, height / 2)
```

Ahora tenemos el ángulo de la inclinación del texto, aplicaremos el método `getRotationMatrix2D()` para obtener la matriz de rotación y luego usaremos el método `warpAffine()` para rotar el ángulo (explicado anteriormente).

```
rotationMatrix =
```

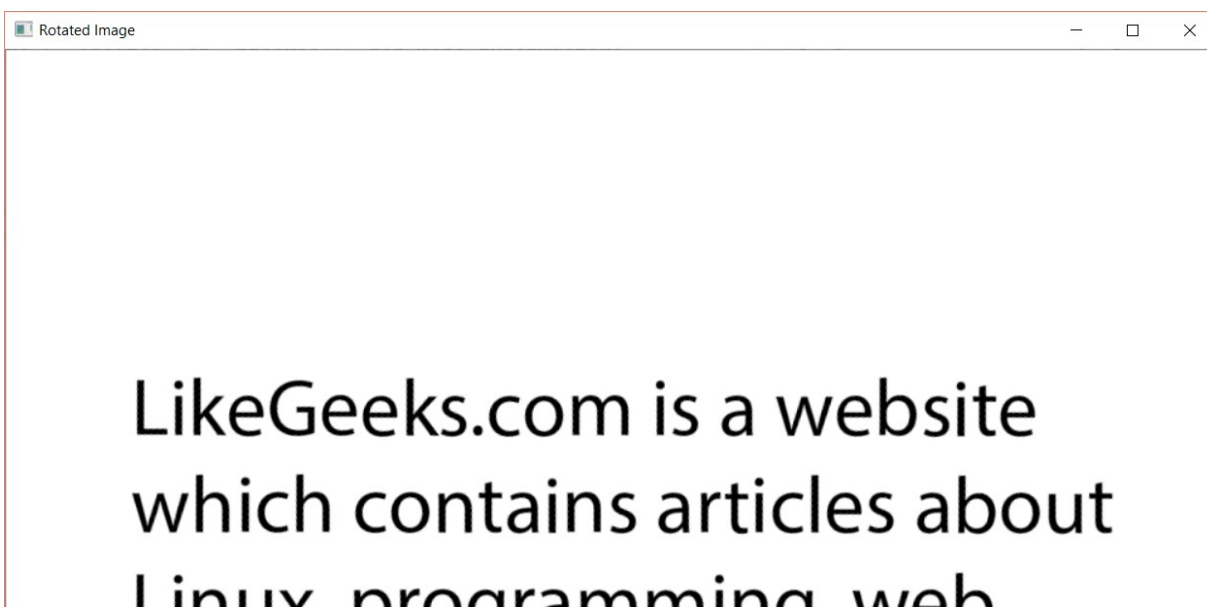
```
cv2.getRotationMatrix2D(center, angle, 1.0)
```

```
rotated_img = cv2.warpAffine(img,  
rotationMatrix, (width, height), borderMode =  
cv2.BORDER_REFLECT)
```

Mostrar la imagen rotada:

```
cv2.imshow("Rotated Image", rotated_img)
```

```
cv2.waitKey(0)
```



Linux, programming, web development, iOS, Python and other tech tips.

Detección de color

Detectemos el color verde de una imagen:

Importar los módulos cv2 para imágenes y NumPy para matrices de imágenes:

```
import cv2
```

```
import numpy as np
```

Lea la imagen y conviértala en HSV usando cvtColor():

```
img = cv2.imread("pydetect.png")
```

```
hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
```

Mostrar la imagen:

```
cv2.imshow("HSV Image", hsv_img)
```





Ahora cree una matriz NumPy para los valores verdes inferiores y los valores verdes superiores:

```
lower_green = np.array([34, 177, 76])
```

```
upper_green = np.array([255, 255, 255])
```

Utilice el método `inRange()` de `cv2` para comprobar si los elementos de la matriz de imágenes dados se encuentran entre los valores de la matriz de los límites superiores e inferiores:

```
masking = cv2.inRange(hsv_img, lower_green,  
upper_green)
```

Esto detectará el color verde.

Por último, muestre las imágenes originales y las resultantes:

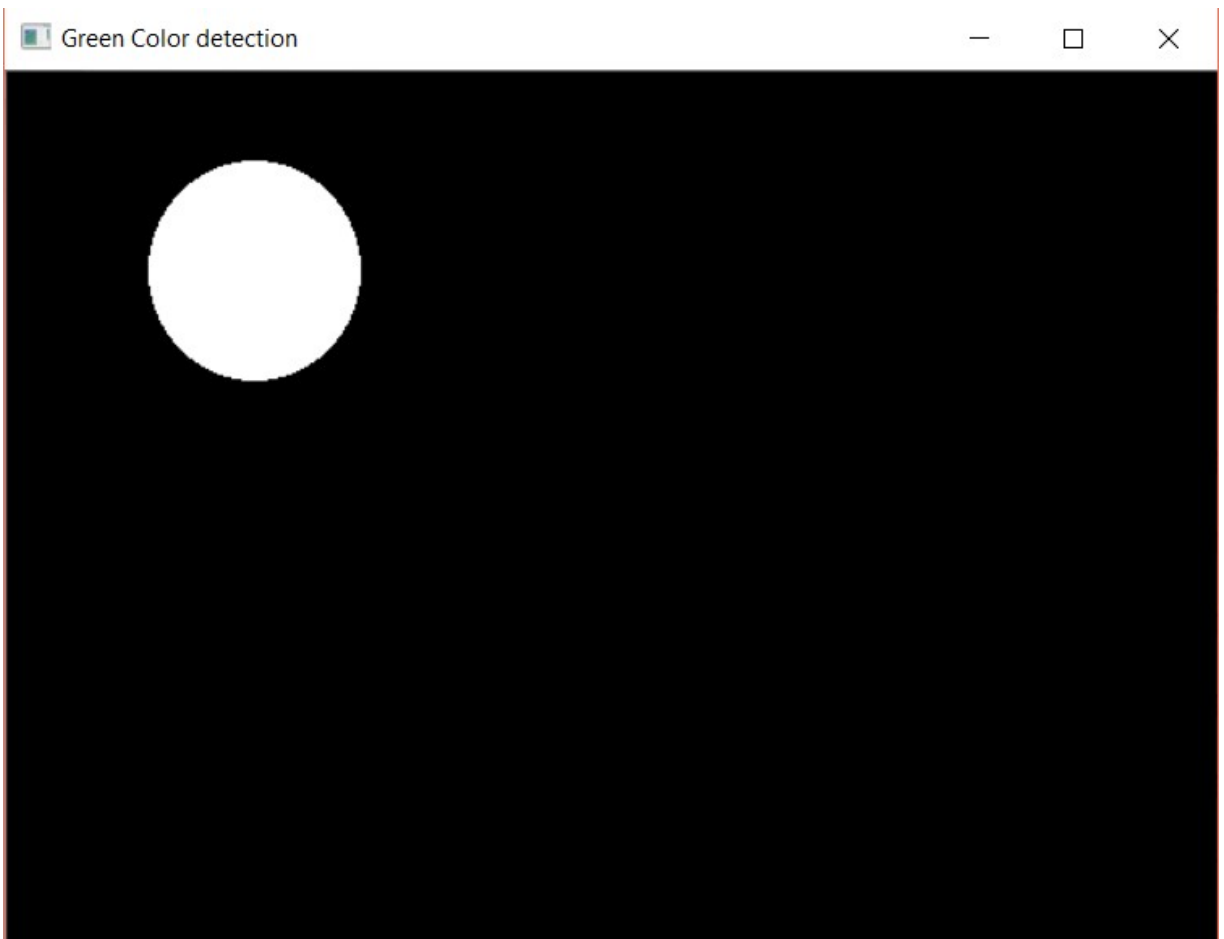
```
cv2.imshow("Original Image", img)
```





```
cv2.imshow("Green Color detection", masking)
```

```
cv2.waitKey(0)
```



Reducir el ruido

Para reducir el ruido de una imagen, OpenCV proporciona los siguientes métodos:

1. `fastNlMeansDenoising()`: Elimina el ruido de una imagen en escala de grises
2. `fastNlMeansDenoisingColored()`: Elimina el ruido de una imagen coloreada
3. `fastNlMeansDenoisingMulti()`: Elimina el ruido de los marcos de imagen en escala de grises (un vídeo en escala de grises)
4. `fastNlMeansDenoisingColoredMulti()`: Igual que el 3 pero funciona con marcos de color

Usemos `fastNlMeansDenoisingColored()` en nuestro ejemplo:

Importar el módulo `cv2` y leer la imagen:

```
import cv2
```

```
img = cv2.imread("pyn1.png")
```

Aplicamos la función de eliminación de ruido que toma respectivamente la imagen original (`src`), el destino (que no hemos guardado ninguno ya que estamos almacenando la resultante), la fuerza del filtro, el valor de la imagen para eliminar el ruido de color (normalmente igual a la fuerza del filtro o 10), el tamaño del parche de la plantilla en píxeles para calcular los pesos que siempre deberían ser impares (el tamaño

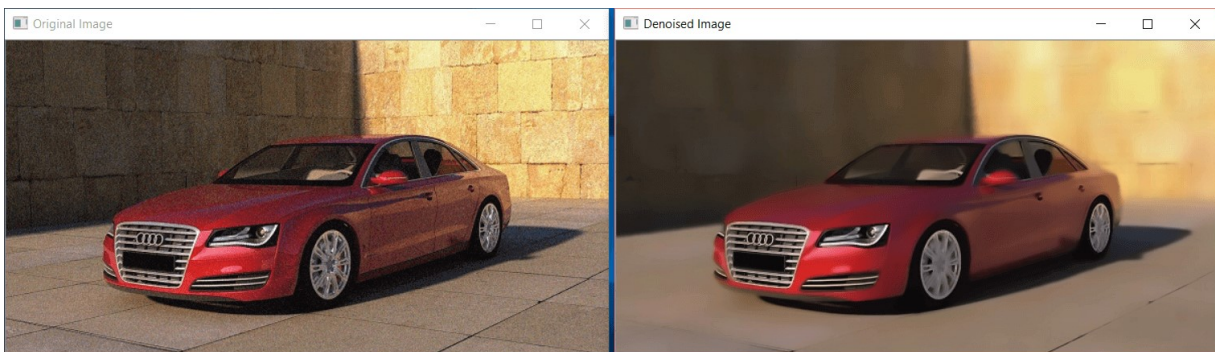
recomendado es igual a 7) y el tamaño de la ventana en píxeles para calcular el promedio del píxel dado.

```
result =  
cv2.fastNlMeansDenoisingColored(img, None, 20, 10, 7, 21)
```

Mostrar la imagen original y la imagen sin ruido:

```
cv2.imshow("Original Image", img)  
  
cv2.imshow("Denoised Image", result)  
  
cv2.waitKey(0)
```

La salida será:



Obtener el contorno de la imagen

Los contornos son las curvas de una imagen que se unen entre sí. Las curvas unen los puntos continuos de una imagen. La finalidad de los contornos es la de detectar los objetos.

La imagen original de la que obtenemos los contornos se muestra a continuación:



OpenCV

Consideremos el siguiente código donde usamos el método `findContours()` para encontrar los contornos en la imagen:

Importar el módulo `cv2`:

```
import cv2
```

Lee la imagen y conviértela en una imagen en escala de grises:

```
img = cv2.imread('py1.jpg')
```

```
gray_img = cv2.cvtColor(img,  
cv2.COLOR_BGR2GRAY)
```

Encuentra el umbral:

```
retval, thresh = cv2.threshold(gray_img, 127,  
255, 0)
```

Usa el `findContours()` que toma la imagen (pasamos el umbral aquí) y algunos atributos. Ver [findContours\(\) Oficial](#).

```
img_contours, _ = cv2.findContours(thresh,  
cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

Dibuje los contornos en la imagen usando el método `drawContours()`:

```
cv2.drawContours(img, img_contours, -1, (0,  
255, 0))
```

Mostrar la imagen:

```
cv2.imshow('Image Contours', img)
```

```
cv2.waitKey(0)
```

El resultado será:



Eliminar el fondo de una imagen

Para eliminar el fondo de una imagen, encontraremos los contornos para detectar los bordes del objeto principal y crearemos una máscara con `np.ceros` para el fondo y luego combinaremos la máscara y la imagen usando el operador `bitwise_and`.

Consideremos el ejemplo siguiente:

Importar los módulos (NumPy y cv2):

```
import cv2
```

```
import numpy as np
```

Lea la imagen y convierta la imagen en una imagen en escala de grises:

```
img = cv2.imread("py.jpg")
```

```
gray_img = cv2.cvtColor(img,  
cv2.COLOR_BGR2GRAY)
```

Encuentra el umbral:

```
_, thresh = cv2.threshold(gray_img, 127, 255,
cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
```

En el método `threshold()`, el último argumento define el estilo del umbral. Vea la documentación oficial de [OpenCV threshold](#).

Encuentra los contornos de la imagen:

```
img_contours = cv2.findContours(thresh,
cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)[-2]
```

Ordena los contornos:

```
img_contours = sorted(img_contours,
key=cv2.contourArea)
```

```
for i in img_contours:
```

```
    if cv2.contourArea(i) > 100:
```

```
        break
```

Generate the mask using `np.zeros`:

```
mask = np.zeros(img.shape[:2], np.uint8)
```

Dibujar los contornos:

```
cv2.drawContours(mask, [i], -1, 255, -1)
```

Aplicar el operador `bitwise_and`:

```
new_img = cv2.bitwise_and(img, img, mask=mask)
```

Mostrar la imagen original:

```
cv2.imshow("Original Image", img)
```

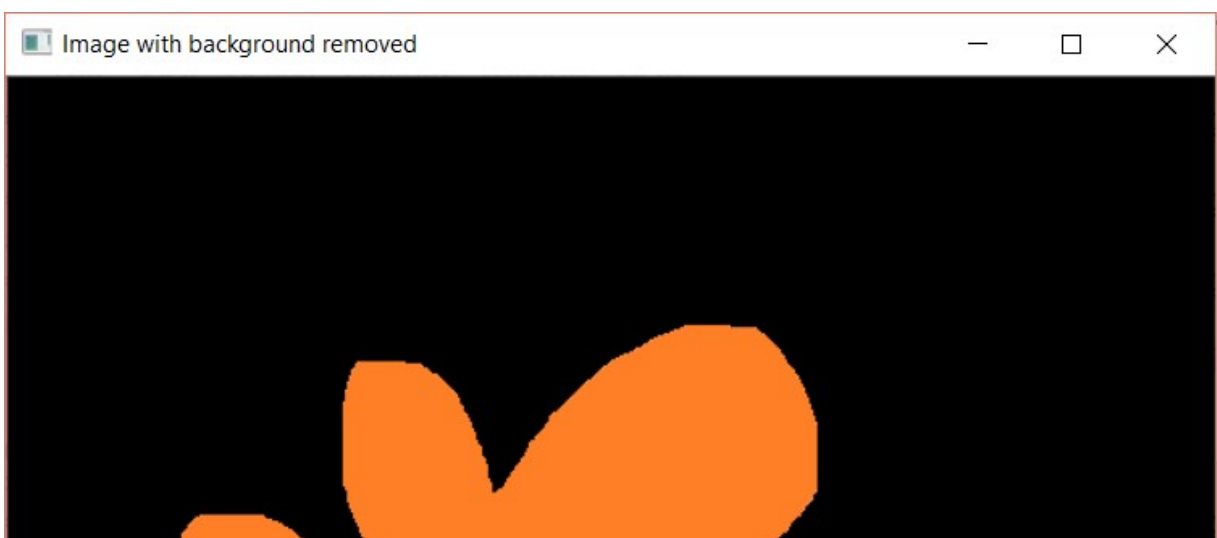




Mostrar la imagen resultante:

```
cv2.imshow("Image with background removed",  
new_img)
```

```
cv2.waitKey(0)
```





El procesamiento de imágenes es divertido cuando se utiliza OpenCV como se ha visto.

Espero que el tutorial te resulte útil. Sigue Volviendo.

Gracias



Fundadora de LikeGeeks. Estoy trabajando como administrador de sistemas Linux desde 2010. Soy responsable de mantener, proteger y solucionar problemas de servidores Linux para múltiples clientes de todo el mundo. Me encanta escribir guiones de shell y Python para automatizar mi trabajo.