Arestas e Contornos II

Conceito

Operadores de arestas baseados em primeiras derivadas geram bordas largas e difíceis de localizar com precisão. Operadores de borda que utilizam segundas derivadas da função imagem são uma alternativa, incluindo operadores modernos que resolvem o problema de bordas em diferentes níveis de escala.

Canny

O operador de Canny é um método popular de detecção de bordas em imagens que usa filtros orientados em várias resoluções de imagem. Ele tem como objetivo minimizar os falsos pontos de borda, obter boa localização das arestas e entregar apenas uma única marca em cada borda. É um método de gradiente baseado em primeiras derivadas e usa cruzamentos por zero de segundas derivadas para localização precisa de bordas. O operador de Canny é frequentemente preferido em relação a outros métodos de detecção de borda.

```python
import cv2
import matplotlib.pyplot as plt


image = cv2.imread('./images/rhino.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# canny edge detection
edges = cv2.Canny(gray, 100, 200)


fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))
ax1.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
ax1.set_title('Original Image')
ax1.axis('off')
ax2.imshow(edges, cmap='gray')
ax2.set_title('Canny Edges')
ax2.axis('off')
plt.show()
```

Edge Detection com Derivadas Segundas

A segunda derivada de uma função indica sua curvatura local e as arestas podem ser encontradas em cruzamentos zero desta derivada. Porém, para evitar a amplificação do ruído da imagem, é comum aplicar algum tipo de pré-suavização com filtros passa-baixa adequados.

O operador "Laplaciano-de-Gaussiano" (LoG) é um exemplo popular de operador de borda que usa derivadas segundas

O Laplaciano é um operador matemático que calcula a divergência do gradiente de uma função. Ele é usado em processamento de imagem para detectar bordas, pois é sensível a áreas de mudança rápida de intensidade. Devido a essa sensibilidade, ele trabalha mal em imagens ruidosas

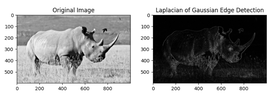
O Laplaciano da Gaussiana é um algoritmo de detecção de bordas em imagens. Ele combina o operador Laplaciano e um filtro Gaussiano para destacar as áreas de transição de intensidade. O filtro Gaussiano suaviza a imagem para reduzir o ruído antes de aplicar o Laplaciano. O resultado são valores positivos e negativos na imagem resultante, indicando bordas com transições de claro para escuro e de escuro para claro.

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt


img = cv2.imread('./images/rhino.jpg', 0)

# Laplacian of gaussian
img_log = cv2.GaussianBlur(img, (3, 3), 0)  # apply Gaussian filter
img_log = cv2.Laplacian(img_log, cv2.CV_64F)  # apply Laplacian filter
img_log = np.uint8(np.absolute(img_log))

plt.subplot(1, 2, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')
plt.subplot(1, 2, 2)
plt.imshow(img_log, cmap='gray')
plt.title('Laplacian of Gaussian Edge Detection')
plt.show()
```



Contornos

Ao detectar bordas em uma imagem, é gerado um valor contínuo de intensidade de borda para cada posição da imagem. Essas informações podem ser usadas para encontrar estruturas de imagem maiores e contornos de objetos específicos. O acompanhamento de contorno envolve traçar contornos sucessivos ao longo dos pontos de borda descobertos, mas isso pode ser difícil devido a problemas como bordas que terminam em regiões de gradiente de intensidade evanescente, arestas cruzadas e contornos que se ramificam em várias direções. Portanto, o acompanhamento de contorno geralmente não é aplicado a imagens originais ou imagens de borda de valor contínuo, exceto em situações simples com uma separação clara entre objetos e o plano de fundo.

Realce das Bodas

Amplificar os componentes de alta frequência da imagem é uma abordagem comum para tornar as imagens mais nítidas, isso pode ser feito com técnicas de detecção de borda

Laplace

O operador de Laplace é um método comum para realçar bordas em uma imagem. Ele funciona calculando a segunda derivada da intensidade da imagem em cada ponto e, em seguida, avaliando se o resultado é um máximo ou mínimo local. Essa operação destaca áreas onde a intensidade da imagem muda rapidamente, ou seja, onde as bordas estão presentes. O operador de Laplace pode ser aplicado diretamente à imagem, mas geralmente é suavizado primeiro com um filtro Gaussiano para reduzir o ruído antes de aplicar o operador. O resultado é uma imagem onde as bordas são realçadas e outras áreas são suprimidas, tornando as bordas mais distintas e fáceis de detectar.

É possível calcular esse operador com o seguinte filtro, e similares

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Em áreas de intensidade plana, o valor do pixel é próximo de zero, em áreas de intensidade variante, o operador tem números de maior grandeza

Unsharp Masking

O mascaramento de nitidez (USM) é uma técnica popular de processamento de imagens que é usada para realçar as bordas em uma imagem. Ele foi originalmente desenvolvido na fotografia clássica e é usado em muitas áreas de processamento de imagens, incluindo astronomia e impressão digital.

O processo envolve a criação de uma "máscara" subtraindo uma versão suavizada da imagem original. A suavização pode ser realizada por meio de um filtro de suavização, como um filtro de média ou gaussiano.

A máscara resultante é então adicionada de volta à imagem original, ponderada por um fator de nitidez, que controla a quantidade de nitidez aplicada.

A aplicação do filtro USM ajuda a realçar as bordas e aumentar a nitidez da imagem, sem afetar outras áreas da imagem que não são bordas. É uma técnica útil para melhorar a qualidade das imagens em muitas áreas, incluindo fotografia, astronomia e impressão digital.

Depois de aplicar o filtro, subtraímos os valores da imagem original, em certa escala.

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt


img = cv2.imread('./images/rhino.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Laplacian operator for edge enhancement
laplacian = cv2.Laplacian(gray, cv2.CV_64F)
# Rescale values to lie between 0 and 255 and convert to uint8
laplacian = np.uint8(np.absolute(laplacian))
enhanced = cv2.addWeighted(gray, 1.5, laplacian, -0.5, 0)


fig, axs = plt.subplots(1, 2)
axs[0].imshow(gray, cmap='gray')
axs[0].set_title('Gray')
axs[0].axis('off')
axs[1].imshow(enhanced, cmap='gray')
axs[1].set_title('Enhanced')
axs[1].axis('off')
plt.show()
```


```python
import cv2
import matplotlib.pyplot as plt
import numpy as np


img = cv2.imread('./images/rhino.jpg')

# defining the unsharp mask parameters
kernel_size = (5, 5)
sigma = 1.5
amount = 1.0
threshold = 0

# creating the unsharp mask kernel
kernel = np.zeros((kernel_size[0], kernel_size[1]), dtype=np.float32)
kernel[int(kernel_size[0] - 1) // 2, int((kernel_size[1] - 1) / 2)] = 2.0
kernel /= np.sum(kernel)

# applying the unsharp mask using cv2
img_blur = cv2.GaussianBlur(img, kernel_size, sigma)
img_usm = cv2.addWeighted(img, 1 + amount, img_blur, -amount, 0)

# plot
fig, ax = plt.subplots(1, 2)
ax[0].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
ax[0].set_title('Original')
ax[1].imshow(cv2.cvtColor(img_usm, cv2.COLOR_BGR2RGB))
ax[1].set_title('Unsharp masked')
plt.show()
```