عبدالرءوف أحمد عبدالرؤف حجازى

Section: 2

1700124

Write a program (code) in any programming language to implement the following algorithms and show your input image and your output image and your code.

1. Sobel Edge Detector

2. Prewitt Gradient Operator

3. Laplacian of Gaussian (LoG)

4. The Canny operator as a multi-step algorithm

• Smooth the image f(m, n) with a Gaussian filter to reduce noise and unwanted details and textures.

• Compute gradient of g(m, n)

• Threshold M: try different thresholds till you get better results

• Link edge segments in to form continuous edges Using your non-maxima suppression Algorithm

import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread(tom and jerry.jpg',)

a = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

plt.imshow(cv2.cvtColor(img, cv2.COLOR\_BGR2RGB))

plt.title('Original')

plt.show()

#Prewitt

def Prewitt(img):

kernelx = np.array([[1,1,1],[0,0,0,],[-1,-1,-1]])

kernely = np.array([[-1,0,1],[-1,0,1],[-1,0,1]])

PrewittX = cv2.filter2D(img, -1, kernelx)

PrewittY = cv2.filter2D(img, -1, kernely)

plt.imshow(PrewittX,cmap = 'gray')

plt.title('PrewittX')

plt.show()

plt.imshow(PrewittY,cmap = 'gray')

plt.title('PrewittY')

plt.show()

plt.imshow(PrewittX + PrewittY,cmap = 'gray')

plt.title('Prewitt')

plt.show()

cv2.imwrite("New after Prewitt.jpg", img)

#Sobel

def Sobel(img):

SobelX = cv2.Sobel(img,cv2.CV\_64F,1,0)

SobelY = cv2.Sobel(img,cv2.CV\_64F,0,1)

plt.imshow(SobelX,cmap = 'gray')

plt.title('SobelX')

plt.show()

plt.imshow(SobelY,cmap = 'gray')

plt.title('SobelY')

plt.show()

plt.imshow(SobelX + SobelY,cmap = 'gray')

plt.title('Sobel')

plt.show()

cv2.imwrite("New after Sobel.jpg", img)

#Laplacian

def Laplacian(img):

Laplacian = cv2.Laplacian(img,cv2.CV\_64F)

plt.imshow(Laplacian,cmap = 'gray')

plt.title('Laplacian')

plt.show()

cv2.imwrite("New after Laplacian.jpg", img)

#Canny

def Canny(img, weak\_th = None, strong\_th = None):

# Original

plt.imshow(img,cmap = 'gray')

plt.title('Original')

plt.show()

# Noise reduction

img = cv2.GaussianBlur(img, (5, 5), 1.4)

plt.imshow(img,cmap = 'gray')

plt.title('After Noise reduction')

plt.show()

cv2.imwrite("New after Noise reduction.jpg", img)

# Compute gradient

gx = cv2.Sobel(img, cv2.CV\_64F, 1, 0)

gy = cv2.Sobel(img, cv2.CV\_64F, 0, 1)

mag, ang = cv2.cartToPolar(gx, gy, angleInDegrees = True)

mag\_max = np.max(mag)

if not weak\_th:weak\_th = mag\_max \* 0.1

if not strong\_th:strong\_th = mag\_max \* 0.5

height, width = img.shape

plt.imshow(mag,cmap = 'gray')

plt.title('After Compute gradient')

plt.show()

cv2.imwrite("New after Compute gradient.jpg", img)

# selecting the neighbours

for i\_x in range(width):

for i\_y in range(height):

grad\_ang = ang[i\_y, i\_x]

grad\_ang = abs(grad\_ang-180) if abs(grad\_ang)>180 else abs(grad\_ang)

# In the x axis direction

if grad\_ang<= 22.5:

neighb\_1\_x, neighb\_1\_y = i\_x-1, i\_y

neighb\_2\_x, neighb\_2\_y = i\_x + 1, i\_y

# top right direction

elif grad\_ang>22.5 and grad\_ang<=(22.5 + 45):

neighb\_1\_x, neighb\_1\_y = i\_x-1, i\_y-1

neighb\_2\_x, neighb\_2\_y = i\_x + 1, i\_y + 1

# In y-axis direction

elif grad\_ang>(22.5 + 45) and grad\_ang<=(22.5 + 90):

neighb\_1\_x, neighb\_1\_y = i\_x, i\_y-1

neighb\_2\_x, neighb\_2\_y = i\_x, i\_y + 1

# top left direction

elif grad\_ang>(22.5 + 90) and grad\_ang<=(22.5 + 135):

neighb\_1\_x, neighb\_1\_y = i\_x-1, i\_y + 1

neighb\_2\_x, neighb\_2\_y = i\_x + 1, i\_y-1

# restarts the cycle

elif grad\_ang>(22.5 + 135) and grad\_ang<=(22.5 + 180):

neighb\_1\_x, neighb\_1\_y = i\_x-1, i\_y

neighb\_2\_x, neighb\_2\_y = i\_x + 1, i\_y

# Non-maximum suppression step

if width>neighb\_1\_x>= 0 and height>neighb\_1\_y>= 0:

if mag[i\_y, i\_x]<mag[neighb\_1\_y, neighb\_1\_x]:

mag[i\_y, i\_x]= 0

continue

if width>neighb\_2\_x>= 0 and height>neighb\_2\_y>= 0:

if mag[i\_y, i\_x]<mag[neighb\_2\_y, neighb\_2\_x]:

mag[i\_y, i\_x]= 0

plt.imshow(mag,cmap = 'gray')

plt.title('After non-maxima suppression')

plt.show()

cv2.imwrite("New non-maxima suppression.jpg", img)

# double thresholding step

ids = np.zeros\_like(img)

for i\_x in range(width):

for i\_y in range(height):

grad\_mag = mag[i\_y, i\_x]

if grad\_mag<weak\_th:

mag[i\_y, i\_x]= 0

elif strong\_th>grad\_mag>= weak\_th:

ids[i\_y, i\_x]= 1

else:

ids[i\_y, i\_x]= 2

plt.imshow(mag,cmap = 'gray')

plt.title('After double thresholding')

plt.show()

cv2.imwrite("New double thresholding.jpg", img)

f1 = Prewitt(a)

f2 = Sobel(a)

f3 = Laplacian(a)

f4 = Canny(a)



Original

Prewitt:



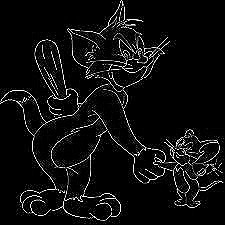
After prewitt

Sobel:



After sobel

Laplacian:



After laplacian

Canny:



After noise reduction



After computer gradient



After non-maxima suppression



After double thresholding