دانشکدگان فنی دانشگاه تهران
دانشکده مهندسی نقشه برداری و اطلاعات مکانی

تمرین شناسایی خطوط لبه

دانشجو:
علیرضا براهیمی

شماره دانشجویی:
810301017

استاد:
دکتر حسنلو

نیمسال اول سال تحصیلی 1401 – 1402

مقدمه :

تشخیص لبه شامل انواع روش های ریاضی است که هدف آنها شناسایی لبه ها و یا منحنی ها در یک تصویر دیجیتال است که در آن روشنایی تصویر به شدت تغییر می کند یا به طور رسمی تر، ناپیوستگی دارد. تشخیص لبه یک ابزار اساسی در پردازش تصویر، بینایی ماشین و بینایی کامپیوتری است، به ویژه در زمینه‌های تشخیص ویژگی و استخراج ویژگی. استفاده از این ابزار در کارهایی مانند شناسایی اجسام بسیار کاربرد دارد .در این پروژه به چهار روش robbert ، sobel ، prewitt و canny شناسایی لبه را انجام می‌دهیم .

Roberts :

در این روش یک گرادیان مکانی دو بعدی بر روی عکس اعمال می‌کنیم . در نتیجه مکانهایی با شیب مکانی زیاد که معمولا با خطوط لبه تطابق دارند برجسته می‌شوند . گرادیان در جهت x و y با استفاده از دو کرنل زیر به دست می‌آیند .

0	1
-1	0

1	0
0	-1

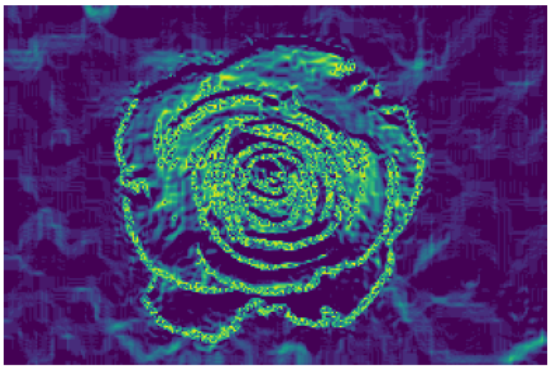
سپس با استفاده از رابطه زیر گرادیان کلی هر پیکسل به دست می‌آید .

$$G = \sqrt{G_x^2 + G_y^2}$$

سپس با اعمال کردن محدوده بر گرادیان می‌توان نقاط لبه و غیر لبه را از هم جدا کرد .

original image

edges detected by method robert



: Prewitt

این روش نیز مانند robert از دو کرنل در جهت x و y استفاده می‌کند و آن را روی عکس اعمال می‌کند و تفاوت آن با روش robert ، تعریف کرنل‌ها است که به صورت زیر است .

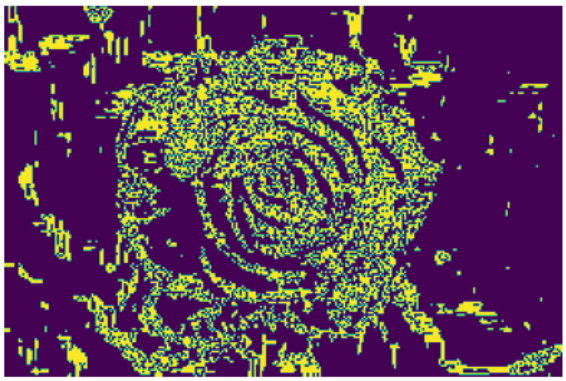
1	1	1
0	0	0
-1	-1	-1

-1	0	1
-1	0	1
-1	0	1

پس از اعمال دو کرنل و به دست آوردن گرادیان عکس بر آن نحدوده اعمال می‌کنیم و نقاط لبه را به دست می‌آوریم .

original image

edges detected by method prewitt



: Sobel

در این روش نیز همانند روش‌های قبلی دو کرنل در جهت x و y بر روی تصویر اعمال می‌شود با این تفاوت که در کرنل‌های آن پیکسل‌های مرکزی دارای ضریب هستند که معمولا این ضریب برابر ۲ می‌باشد .

-1	0	+1
-2	0	+2
-1	0	+1

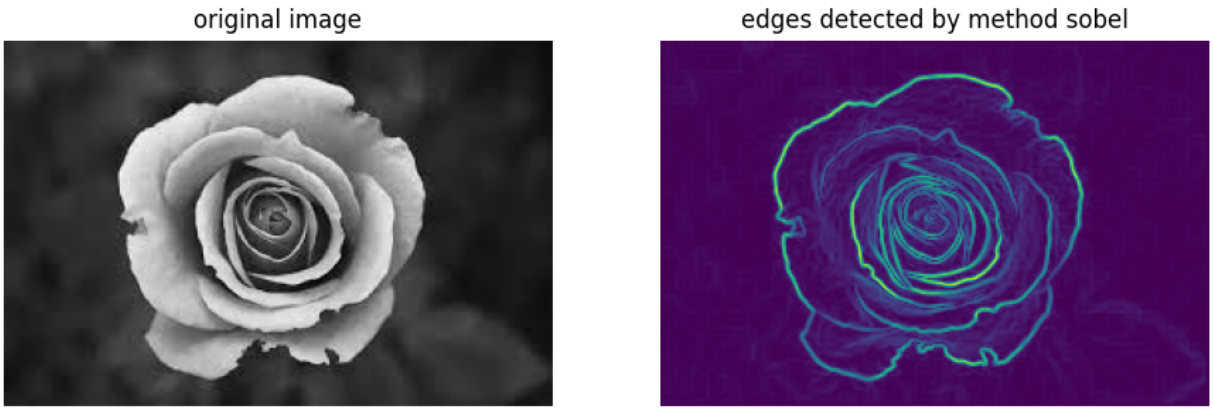
Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

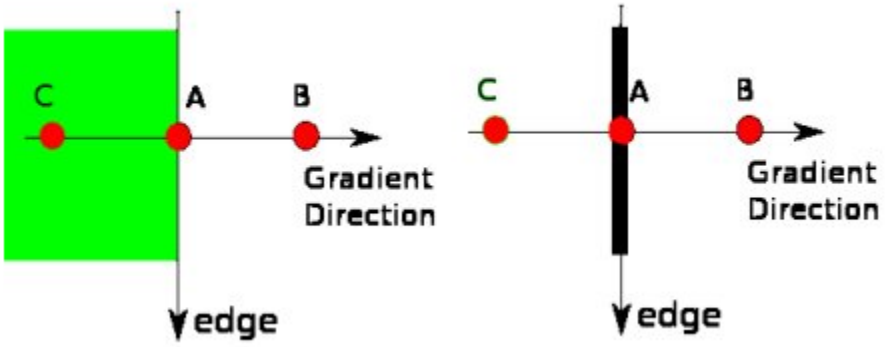
پس از بدست آوردن گرادیان در دو جهت و محاسبه گرادیان کلی تصویر ، بر روی آن محدوده اعمال می‌کنیم و ماسک نقاط لبه و غیر لبه را تشکیل می‌دهیم .

$$|G| = \sqrt{Gx^2 + Gy^2}$$

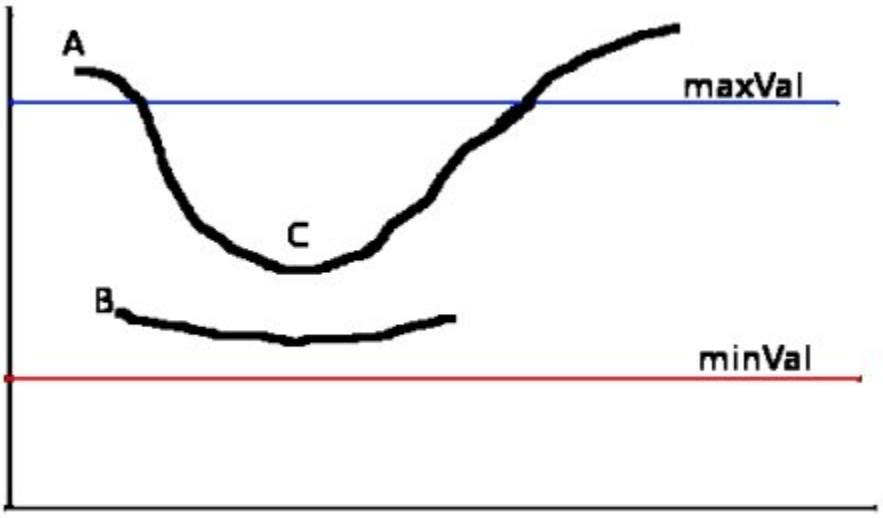


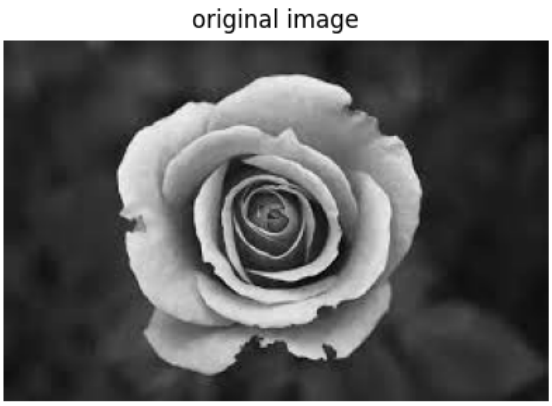
: Canny

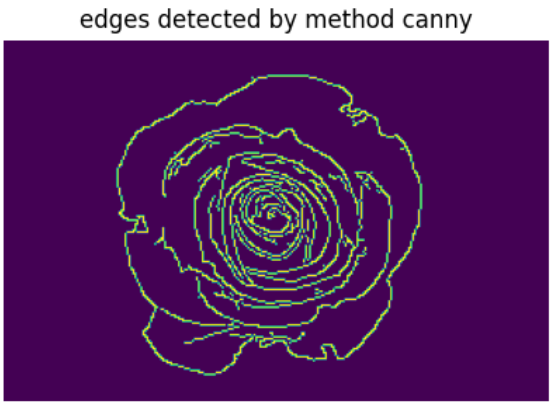
این الگوریتم یک روش چند مرحله‌ای است که آن هارا شرح می‌دهیم .
در مرحله اول ابتدا به منظور حذف نویز و جلوگیری از تشخیص آن‌ها به عنوان لبه بر روی عکس فیلتر گوسین اعمال می‌کنیم .
سپس مانند روش‌های قبلی گرادیان را برای عکس با استفاده از دو کرنل در راستای x و y به دست می‌آوریم و سپس گرادیان کلی .
برای این کار به عنوان مثال می‌توان از کرنل‌های sobel استفاده کرد .
در مرحله بعد برای لبه در جهت عمود بر لبه نقطه دارای ماکزیمم گرادیان به عنوان نقطه لبه در نظر گرفته می‌شود تا لبه‌ها نازک‌تر شوند .



در مرحله آخر برای هر خط که لبه تشخیص داده شده است تصمیم می‌گیریم که ابه است یا نه . برای این کار دو مقدار max و min در نظر می‌گیریم . اگر برای یک لبه شدت گرادیان از مقدار max بیشتر بود لبه و اگر از min کمتر بود غیر لبه است . برای لبه هایی که بین این دو مقدار قرار می‌گیرند لبه بودن آن ها بستگی به نسبت آن‌ها با لبه های دیگر دارد .







```python
import numpy as np
import cv2


def robert(img ,thre,use_thr):
   gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
   img_gaussian = cv2.GaussianBlur(gray, (3, 3), 0)
   roberts_cross_v = np.array([[1, 0],[0, -1]])
   roberts_cross_h = np.array([[0, 1],[-1, 0]])
   img_prewittx = cv2.filter2D(img_gaussian, -1, roberts_cross_v)
   img_prewitty = cv2.filter2D(img_gaussian, -1, roberts_cross_h)
   edge = np.sqrt(img_prewittx ** 2 + img_prewitty ** 2)
   if use_thr == 'on':
       edge = np.where(edge < thre , 0 ,255)
   return edge

def prewit(img , thre,use_thr):
   gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
   img_gaussian = cv2.GaussianBlur(gray, (3, 3), 0)
   kernelx = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]])
   kernely = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])
   img_prewittx = cv2.filter2D(img_gaussian, -1, kernelx)
   img_prewitty = cv2.filter2D(img_gaussian, -1, kernely)
   edge = np.sqrt(img_prewittx**2 + img_prewitty**2)
   if use_thr == 'on':
       edge = np.where(edge < thre , 0 ,255)
   return edge

def sobel(img , thre,use_thr):
   gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
   shape = gray.shape
   edge = np.zeros(shape)
   for i in range(1, shape[0]-1):
       for j in range(1, shape[1]-1):
           edge[i][j] = int(np.sqrt((gray[i-1][j+1] + 2*gray[i][j+1] + gray[i+1][j+1] -
gray[i-1][j-1] - 2*gray[i][j-1] - gray[i+1][j-1]) ** 2 + (gray[i+1][j-1] + 2*gray[i+1][j] +
gray[i+1][j+1] -gray[i-1][j-1] - 2*gray[i-1][j] - gray[i-1][j+1]) ** 2))
   if use_thr == 'on':
       edge = np.where(edge < thre , 0 ,255)
   return edge

def canny(img , upper_ther , lower_ther):
   gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
   edge = cv2.Canny(gray ,lower_ther, upper_ther)
   return edge
```