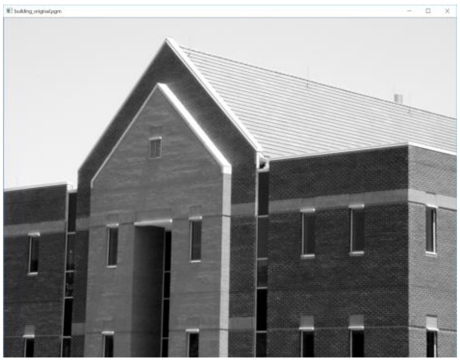
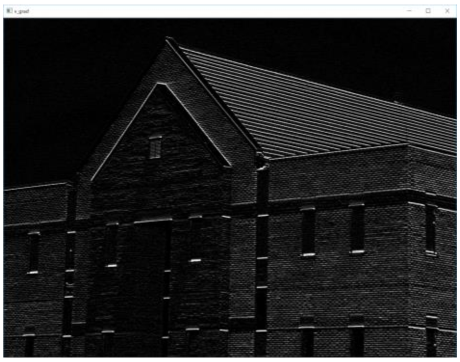
谭树杰 11849060

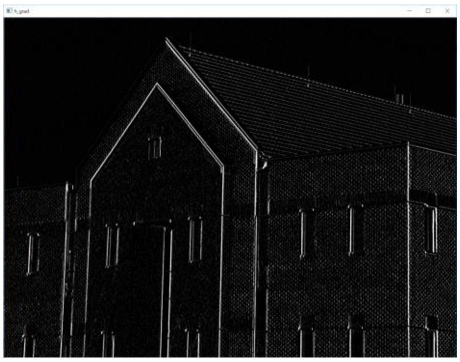
Lab10 Report

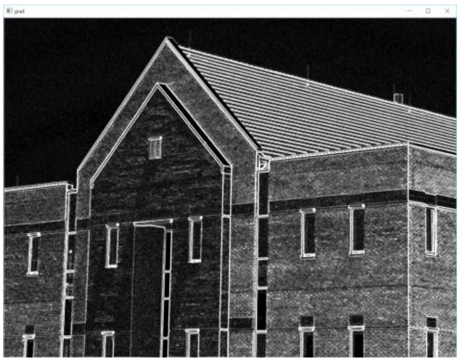
1. **Use Roberts, Prewitt, Sobel gradient operators to obtain gradient images, then threshold the images to compare the results among different operators. The images are headCT_Vandy.pgm, building_original.pgm, noisy_fingerprint.pgm**

(1) The building_original.pgm,its gradient images(gx,gy,|gx|+|gy|) using different masks and thresholded images(using threshold = 0.24 * max(pixel values)) are shown below







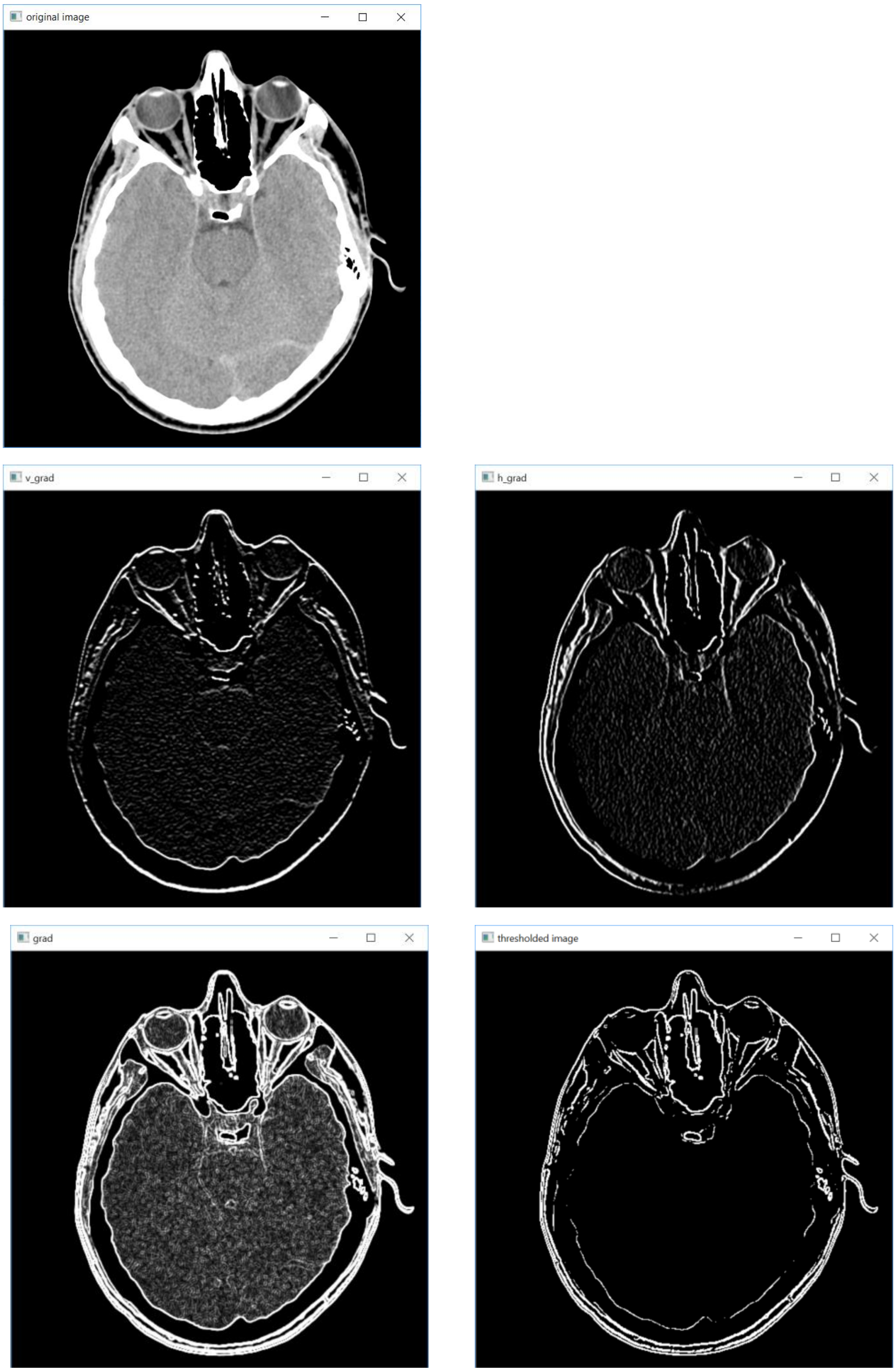


Sobel

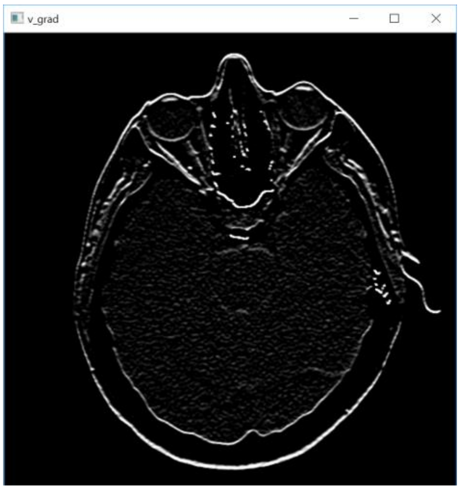
Prewitt

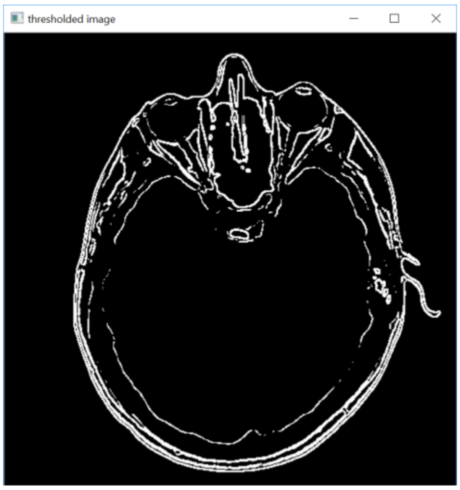
Roberts

(2) The headCT_Vandy.pgm, its gradient images(gx,gy,|gx|+|gy|) using different masks and thresholded images(using threshold = 0.24 ∗ max(pixel values)) are shown below

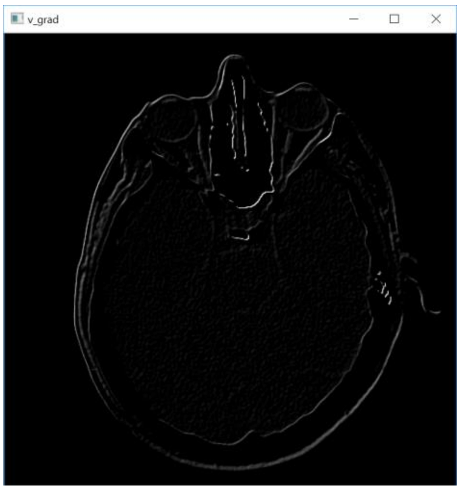


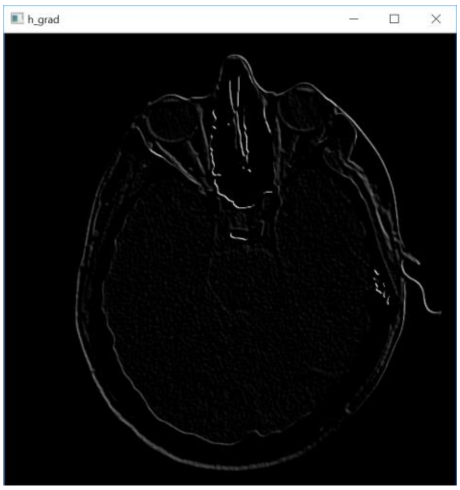
Sobel

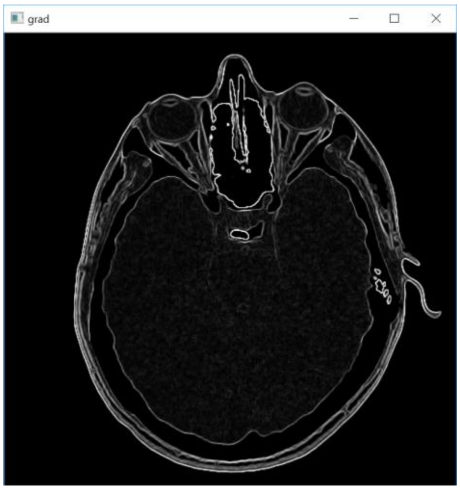


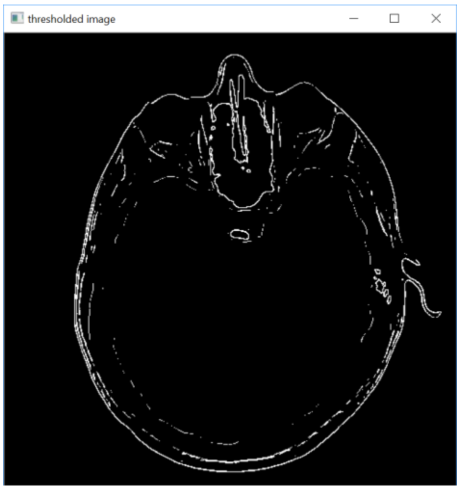


Prewitt



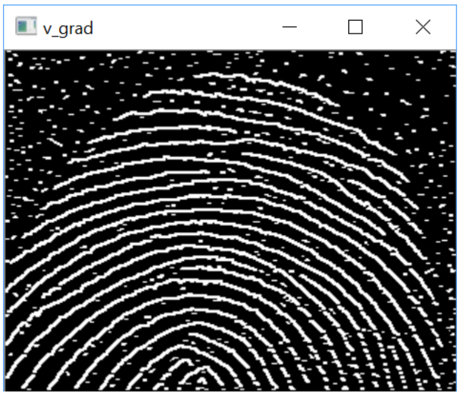


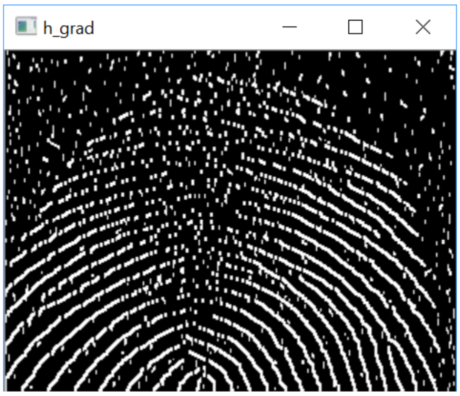


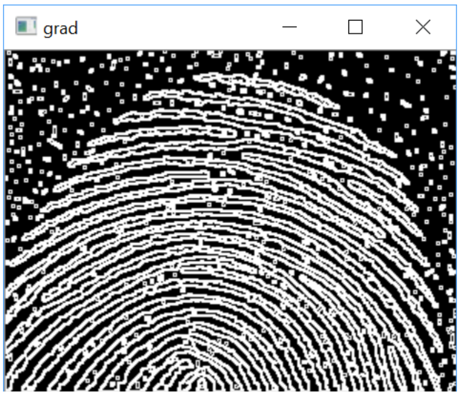


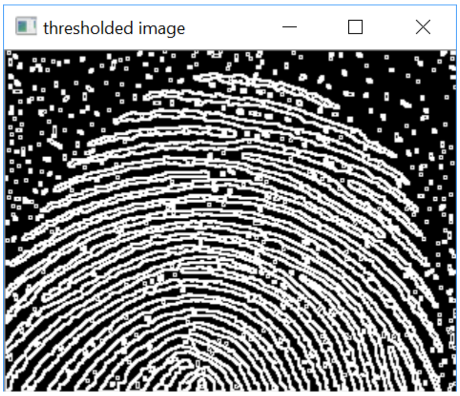
Roberts

(3) The noisy_fingerprint.pgm, their gradient images(gx,gy,|gx|+|gy|) using different masks and thresholded images(using threshold = 0.24 * max(pixel values)) are shown below

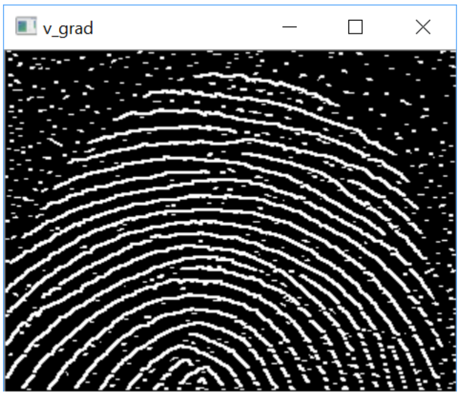


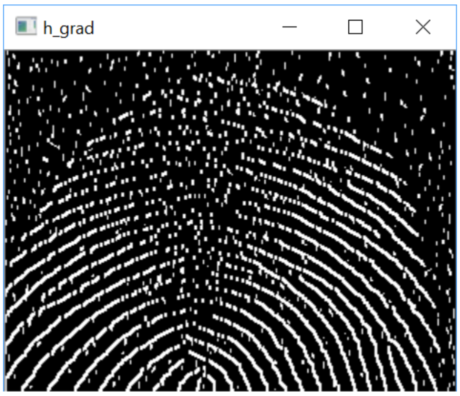


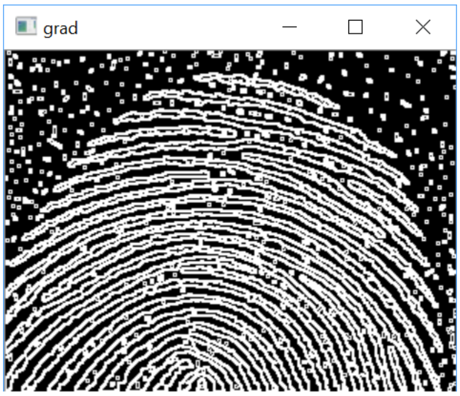


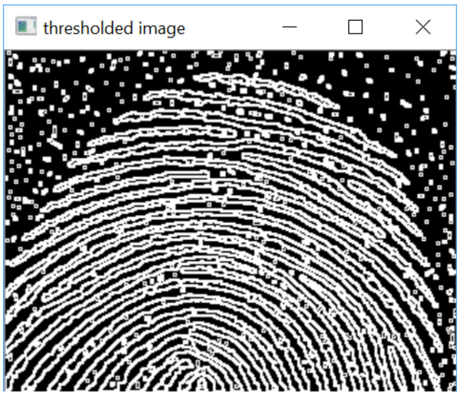


Sobel

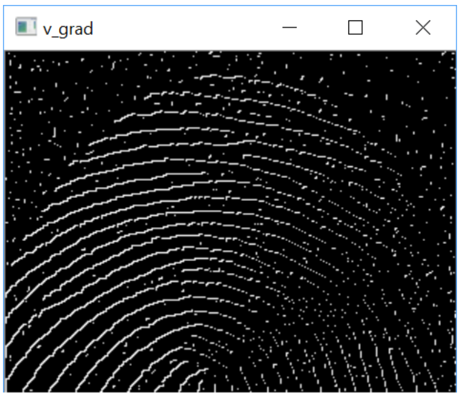


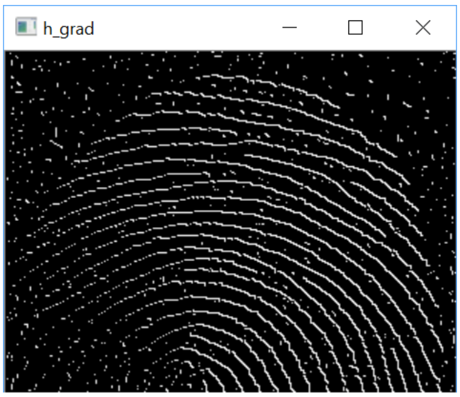


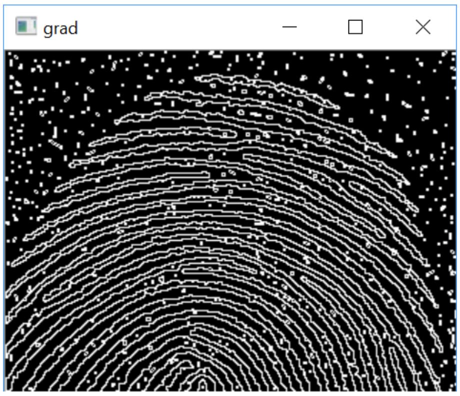


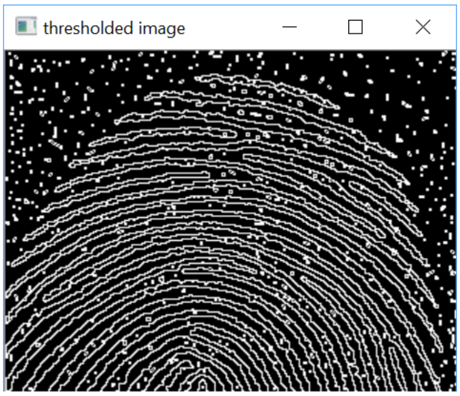


Prewitt









Roberts

Analysis:

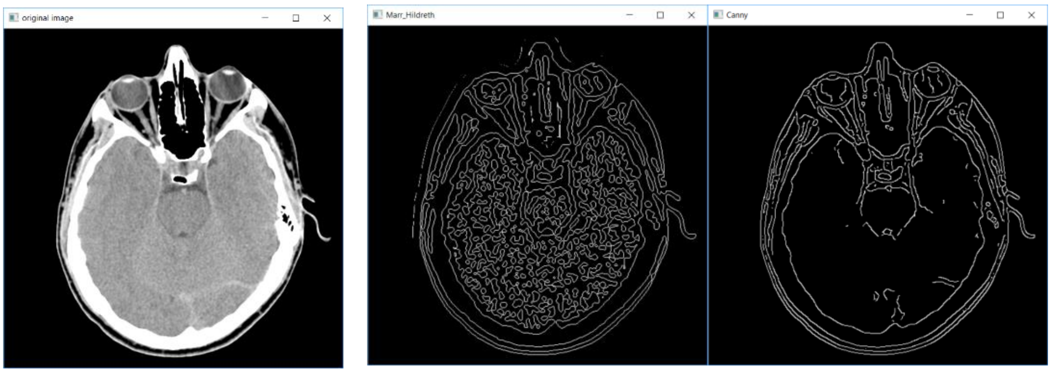
The directionality of the horizontal and vertical components of the gradient is evident in building_original.pgm. The horizontal and vertical Sobel masks do not differentiate between edges oriented in the directions. We see that there are fewer edges in the thresholded image,and that the edges in this image are much sharper (see, for example, the edges in the roof tile). On the other hand, numerous edges, such as the 45° line defining the far edge of the roof, are broken in the thresholded image.

The implementation code is shown below:

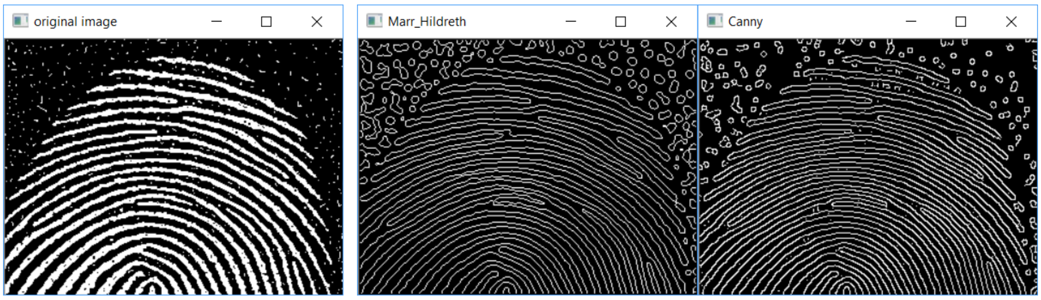
```python
import numpy as np
import cv2
from matplotlib import pyplot as plt
from copy import deepcopy
from collections import Counter

def myfilter(src, kernel):
    margin = int(kernel.shape[0] / 2)
    dst = np.zeros_like(src, dtype=float)
    img_padded = np.pad(src, ((margin, margin), (margin, margin)), 'constant')
    for r in range(margin, img_padded.shape[0] - margin):
        for c in range(margin, img_padded.shape[1] - margin):
            if kernel.shape[0] % 2 == 1:
                filter_window = np.copy(img_padded[r - margin:r + margin + 1, c - margin:c + margin + 1])
            else:
                filter_window = np.copy(img_padded[r - margin:r + margin, c - margin:c + margin])
            dst[r - margin, c - margin] = np.sum(filter_window * kernel)
    return dst
def threshold(src, th):
    dst = np.zeros_like(src, dtype=float)
    dst[src <= th] = 0
    dst[src > th] = 1
    return dst
# path_in = './in/building_original.pgm'
# path_in = './in/headCT-Vandy.pgm'
path_in = './in/noisy_fingerprint.pgm'
img = cv2.imread(path_in, 0)
img = img.astype(float)
cv2.normalize(img, img, 0, 1, cv2.NORM_MINMAX)
cv2.imshow("original image", img)
# v_kernel = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]]) # Sobel
# v_kernel = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])   # Prewitt
v_kernel = np.array([[-1, 0], [0, 1]])   # Roberts
# h_kernel = np.transpose(v_kernel)
h_kernel = np.array([[0, -1], [1, 0]])   # Roberts
v_grad = myfilter(img, v_kernel)
cv2.imshow("v_grad", v_grad)
h_grad = myfilter(img, h_kernel)
cv2.imshow("h_grad", h_grad)
grad = np.abs(v_grad) + np.abs(h_grad)
cv2.imshow("grad", grad)
thres = threshold(grad, 0.24 * np.amax(grad))
cv2.imshow("thresholded image", thres)
```

2. Implement Canny edge detection and LoG detection algorithms on headCT_Vandy.pgm and noisy_fingerprint.pgm

(1) The headCT_Vandy.pgm, its edge images by applying Marr-Hildreth and Canny algorithms are shown below.



(2) The noisy_fingerprint.pgm, its edge images by applying Marr-Hildreth and Canny algorithms are shown below.



Analysis:

The Canny algorithm using the parameters t = 0.05, T = 0.15 (3 times the value of the low threshold), sigma = 2. The Marr-Hildreth edge-detection algorithm with a threshold of 0.002, sigma = 3.

The Marr-Hildreth algorithm detected major edges of the boundary of the headCT-Vandy.pgm but can't eliminate the edges associated with the gray matter.

The Canny algorithm was the only procedure capable of yielding a totally unbroken edge for the posterior boundary of the brain. It was also the only procedure capable of finding the best contours while eliminating all the edges associated with the gray matter in the original image.

The implementation code is shown below:

```python
# 2.

from scipy import misc

import numpy as np

from scipy.ndimage.filters import gaussian_filter

from scipy import ndimage

from scipy.ndimage import sobel, generic_gradient_magnitude,generic_filter



def round_angle(angle):

    """ Input angle must be \in [0,180) """

    angle = np.rad2deg(angle) % 180

    if (0 <= angle < 22.5) or (157.5 <= angle < 180):

        angle = 0

    elif 22.5 <= angle < 67.5:

        angle = 45

    elif 67.5 <= angle < 112.5:

        angle = 90

    elif 112.5 <= angle < 157.5:

        angle = 135

    return angle



def gradient_intensity(img):

    # Kernel for Gradient in x-direction

    Kx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], np.float)

    # Kernel for Gradient in y-direction

    Ky = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]], np.float)

    # Apply kernels to the image

    Ix = ndimage.filters.convolve(img, Kx)

    Iy = ndimage.filters.convolve(img, Ky)

    G = np.hypot(Ix, Iy)    # gradient-intensed image

    D = np.arctan2(Iy, Ix)  # gradient directions

    return G, D



def suppression(src, D):

    rows, cols = src.shape

    dst = np.zeros_like(src)
```

```python
    for i in range(rows):
        for j in range(cols):
            # find neighbour pixels to visit from the gradient directions
            where = round_angle(D[i, j])
            try:
                if where == 0:
                    if (src[i, j] >= src[i, j - 1]) and (src[i, j] >= src[i, j + 1]):
                        dst[i, j] = src[i, j]
                elif where == 90:
                    if (src[i, j] >= src[i - 1, j]) and (src[i, j] >= src[i + 1, j]):
                        dst[i, j] = src[i, j]
                elif where == 135:
                    if (src[i, j] >= src[i - 1, j - 1]) and (src[i, j] >= src[i + 1, j + 1]):
                        dst[i, j] = src[i, j]
                elif where == 45:
                    if (src[i, j] >= src[i - 1, j + 1]) and (src[i, j] >= src[i + 1, j - 1]):
                        dst[i, j] = src[i, j]
            except IndexError as e:
                """ Todo: Deal with pixels at the image boundaries. """
                pass
    return dst


def threshold(img, t, T):
    # define gray value of a WEAK and a STRONG pixel
    cf = {
        'WEAK': np.float(0.2),
        'STRONG': np.float(1),
    }
    # get strong pixel indices
    strong_i, strong_j = np.where(img > T)
    # get weak pixel indices
    weak_i, weak_j = np.where((img >= t) & (img <= T))
    # get pixel indices set to be zero
    zero_i, zero_j = np.where(img < t)

    # set values
    img[strong_i, strong_j] = cf.get('STRONG')
    img[weak_i, weak_j] = cf.get('WEAK')
    img[zero_i, zero_j] = np.float(0)
    return img, cf.get('WEAK')
```

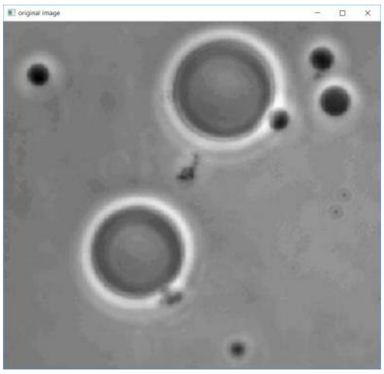
```python
def tracking(img, weak, strong=1):
    hight, width = img.shape
    for i in range(hight):
        for j in range(width):
            if img[i, j] == weak:
                # check if one of the neighbours is strong (=1 by default)
                try:
                    window = img[i - 1:i + 2, j - 1:j + 2]
                    if np.any(window == strong):
                        img[i, j] = strong
                    else:
                        img[i, j] = 0
                except IndexError as e:
                    pass
    return img


def canny(src, sigma, t, T):
    dst = np.copy(src)
    dst = gaussian_filter(dst, sigma)
    dst, D = gradient_intensity(dst)
    dst = suppression(dst, D)
    dst, weak = threshold(dst, t, T)
    dst = tracking(dst, weak)
    return dst


from mpl_toolkits.mplot3d import Axes3D
def get_gaussian(sigma)


def MarrHildreth(img, sigma, t=0.002):
    size = int(2*(np.ceil(3*sigma))+1)
    r = int(np.ceil(size / 2))
    x, y = np.meshgrid(np.arange(-r, r+1), np.arange(-r, r+1))
    normal = 1 / (2.0 * np.pi * sigma**2)
    kernel = ((x**2 + y**2 - (2.0*sigma**2)) / sigma**4) * np.exp(-(x**2+y**2) /
     (2.0*sigma**2)) / normal # LoG filter

    kern_size = kernel.shape[0]
    # applying filter
    log = ndimage.filters.convolve(img, kernel)
      zero_crossing = np.zeros_like(log)
```

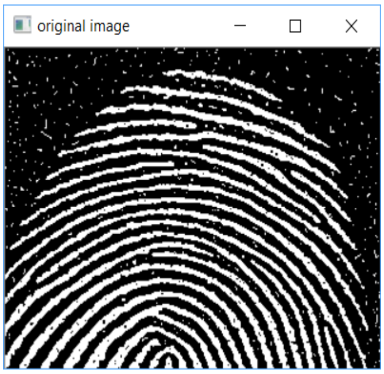
```python
# computing zero crossing
    index = np.array([[-1,-1],[-1,0],[-1,1],[0,-1]])
    for i in range(1, log.shape[0]-1):
        for j in range(1, log.shape[1]-1):
            if log[i][j] < 0:
                if (log[i][j - 1] > 0) or (log[i][j + 1] > 0) \
                        or (log[i - 1][j] > 0) or (log[i + 1][j] > 0):
                    zero_crossing[i][j] = 1
                    continue
            for k in index:
                ax = [i,j] + k
                p1, p2 = log[tuple(ax)], log[tuple(-ax)]
                diff = abs(p1-p2)
                if log[i,j] == 0:
                    if p1 * p2 < 0 and diff > t:    #  or  log[i,j]==0
                        zero_crossing[i][j] = 1
                        break

    return zero_crossing


# path_in = './in/headCT-Vandy.pgm'
path_in = './in/noisy_fingerprint.pgm'
img = cv2.imread(path_in, 0)
img = img.astype(float)
cv2.normalize(img, img, 0, 1, cv2.NORM_MINMAX)
cv2.imshow("original image", img)

dst = canny(img, 2, 0.05, 0.15)      # Canny, headCT-Vandy.pgm
cv2.imshow("Canny", dst)

dst = MarrHildreth(img, sigma=3, t=0.0002)
cv2.imshow("Marr_Hildreth", dst)
```

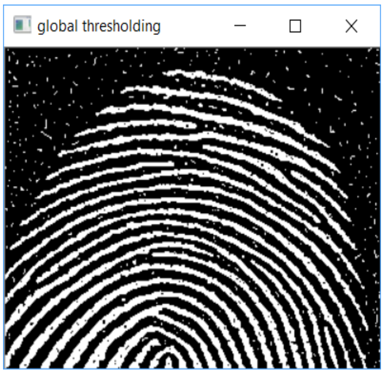
3. Use global thresholding to perform segmentation separately on polymersomes.pgm and noisy_fingerprint.pgm

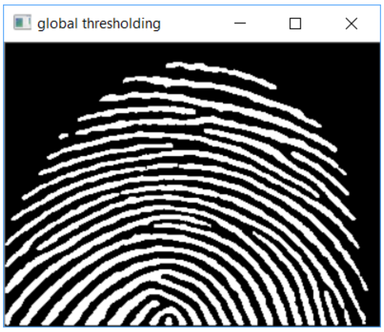
Result:

The polymersomes.pgm and noisy_fingerprint.pgm and their thresholded images are shown below.









Analysis:

Because the histogram of polymersomes.pgm has no distinct valleys and the intensity difference between the background and objects is small, the algorithm failed to achieve the desired segmentation. The final image is achieved by applying smoothing to noisy_fingerprint.pgm first, then using global thresholding. We can observe the improvement of denoising of the image.

The implementation code is shown below:

```python
# 3. Use global thresholding to perform segmentation separately on
# polymersomes.pgm and noisy_fingerprint.pgm
from myfilters import arithmetic_mean_filter
def global_thresholding(src, t):
    if np.amin(src) == np.amax(src):
        dst = np.copy(src)
        return dst
    oldT = 0.5 * (np.amax(src) + np.amin(src))
    G1 = np.copy(src[src > oldT])
    G2 = np.copy(src[src <= oldT])
    m1 = np.mean(G1)
    m2 = np.mean(G2)
    T = 0.5 * (m1 + m2)
    while T - oldT >= t:
        oldT = T
        G1 = src[src > T]
        G2 = src[src <= T]
        m1 = np.mean(G1)
        m2 = np.mean(G2)
        T = 0.5 * (m1 + m2)
    dst = np.zeros_like(src)
    dst[src > T] = 1
    return dst


# path_in = './in/polymersomes.pgm'
path_in = './in/noisy_fingerprint.pgm'
img = cv2.imread(path_in, 0)
img = img.astype(float)
cv2.normalize(img, img, 0, 1, cv2.NORM_MINMAX)
cv2.imshow("original image", img)

dst = global_thresholding(img, 0.002)
cv2.imshow("global thresholding", dst)

dst = arithmetic_mean_filter(img, size=5)
cv2.imshow("average", dst)
dst = global_thresholding(dst, 0.002)
cv2.imshow("global thresholding", dst)
```