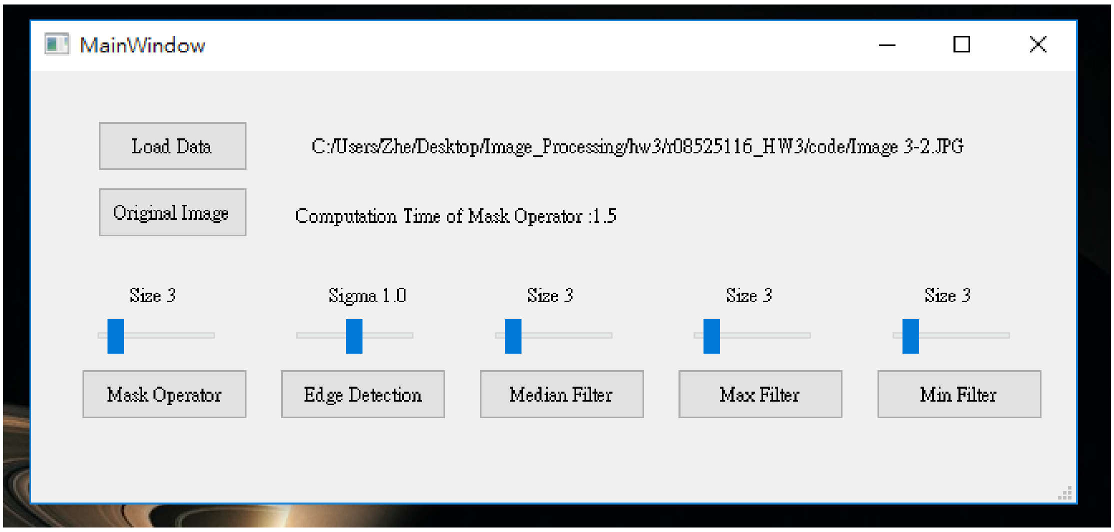
Homework3

工科所 **R08525116** 吳承哲

GUI



Original Image 為顯示原圖。

其餘選項的顯示中，左為原圖，右為處理過後的圖片。

Part 2

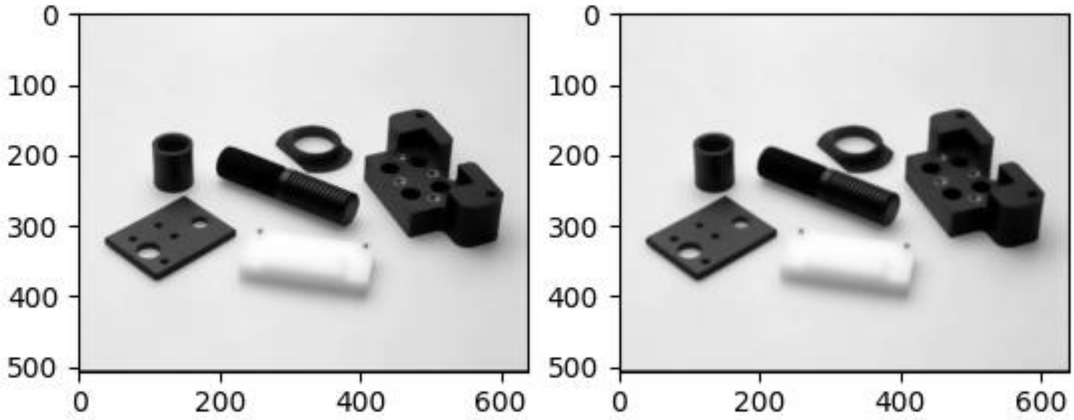
Code

```python
def mask_operator(img, size=3):
    kernel = np.ones((size, size))
    kernel_sum = np.sum(kernel)
    filter_kernel = kernel / kernel_sum

    m, n = filter_kernel.shape

    padding_y = (m - 1)//2
    padding_x = (n - 1)//2

    # zero padding
    padding_img = cv2.copyMakeBorder(
        img, padding_y, padding_y, padding_x, padding_x, cv2.BORDER_CONSTANT, value=0)

    y, x = padding_img.shape

    y_out = y - m + 1
    x_out = x - n + 1

    # convolution
    new_img = np.zeros((y_out, x_out))
    for i in range(y_out):
        for j in range(x_out):
            new_img[i][j] = np.sum(padding_img[i:i+m, j:j+n]*filter_kernel)

    new_img = new_img.astype(np.uint8)

    computation_time = (m*n)/(m + n)

    return new_img, computation_time
```

因為不知道如何使用 GUI 去調整係數，因此先寫好 kernel 的規則，將係數全部設為 1，再除以係數和，kernel size 可以做調整。

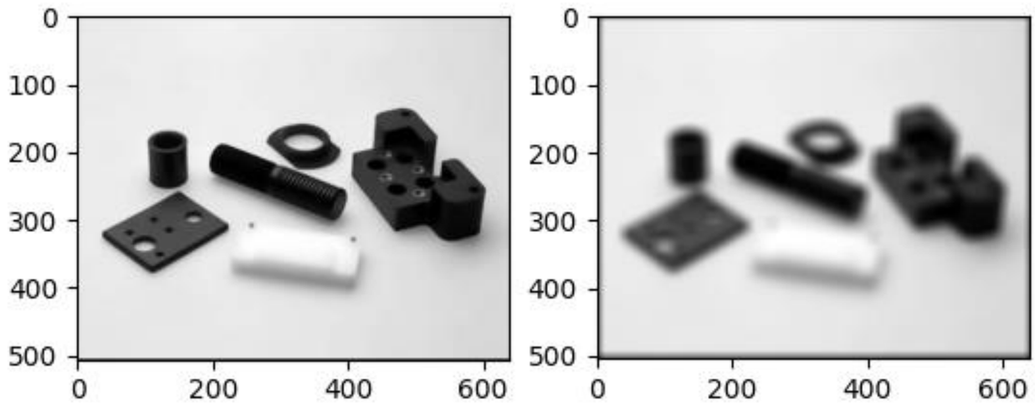
Discussion

Kernel Size = 3*3 (左圖為原圖)



Computation Time of Mask Operator :1.5

Kernel Size = 20*20(左圖為原圖)



Computation Time of Mask Operator :10.0

由此可知，Kernel size 越大，模糊的效果越好。

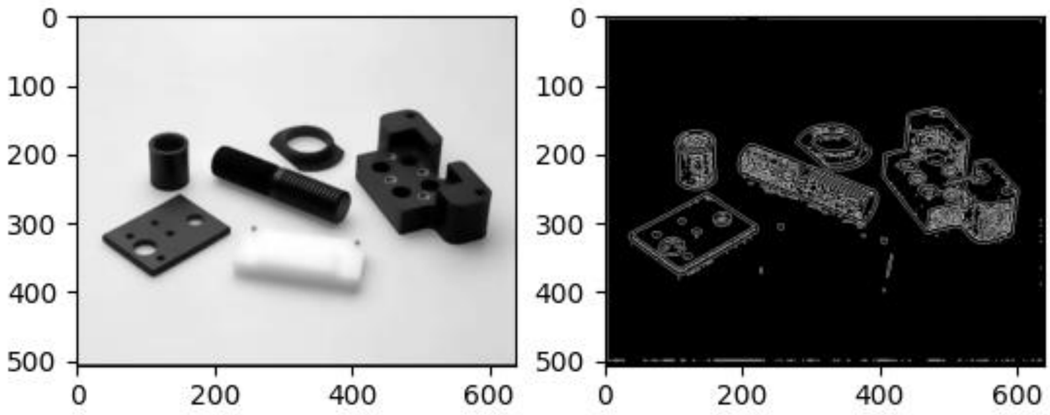
Part 3

```python
def Marr_Hildreth(img, sigma=1):
    size = int(2*(np.ceil(3*sigma))+1)

    x, y = np.meshgrid(np.arange(-size/2+1, size/2+1),
                       np.arange(-size/2+1, size/2+1))

    normal = 1 / (2.0 * np.pi * sigma**2)

    # LoG filter
    kernel = ((x**2 + y**2 - (2.0*sigma**2)) / sigma**4) * \
        np.exp(-(x**2+y**2) / (2.0*sigma**2)) / normal

    kern_size = kernel.shape[0]
    img_LoG = np.zeros_like(img, dtype=float)

    # filtering
    for i in range(img.shape[0]-(kern_size-1)):
        for j in range(img.shape[1]-(kern_size-1)):
            window = img[i:i+kern_size, j:j+kern_size] * kernel
            img_LoG[i, j] = np.sum(window)

    img_LoG = img_LoG.astype(np.int64, copy=False)

    zero_crossing = np.zeros_like(img_LoG)
```

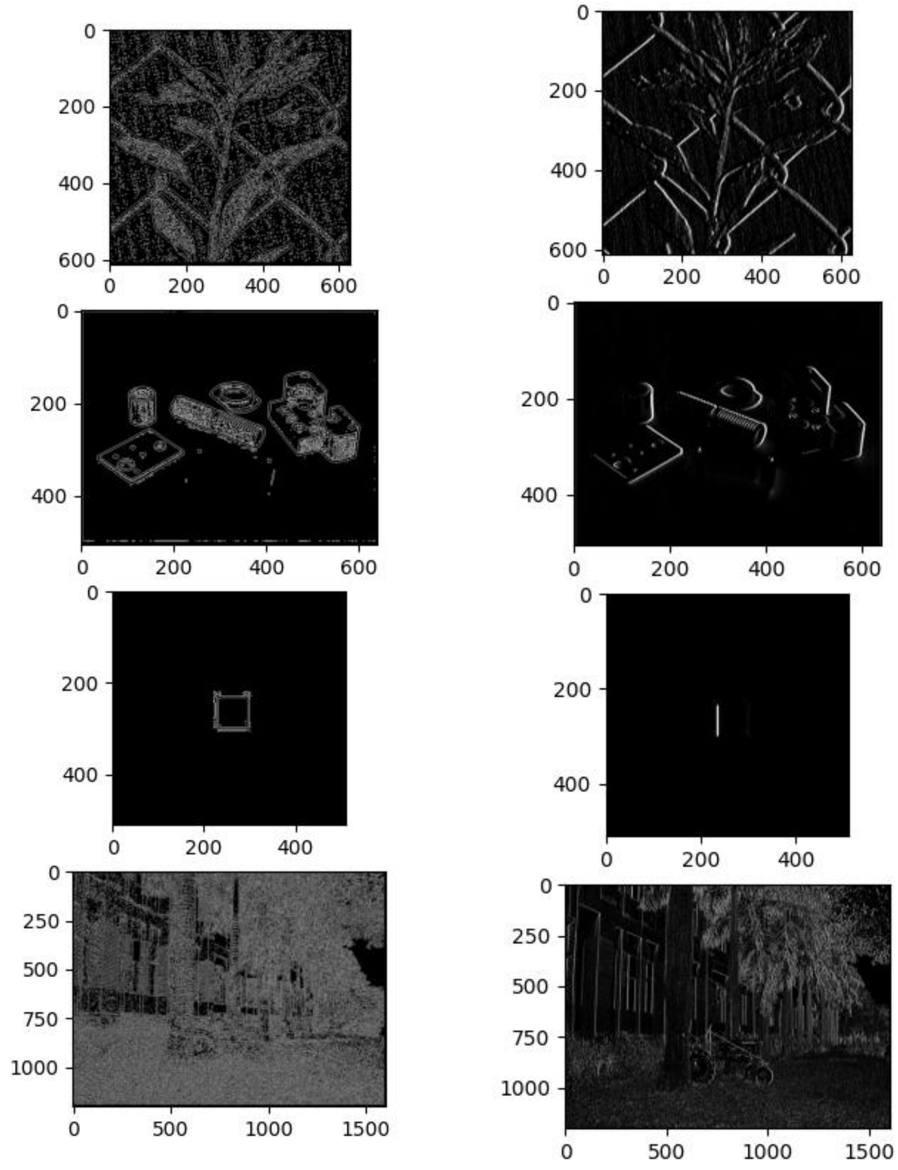
```python
    # computing zero crossing
    for i in range(img_LoG.shape[0]-(kern_size-1)):
        for j in range(img_LoG.shape[1]-(kern_size-1)):
            if img_LoG[i][j] == 0:
                if (img_LoG[i][j-1] < 0 and img_LoG[i][j+1] > 0) or \
                    (img_LoG[i][j-1] < 0 and img_LoG[i][j+1] < 0) or \
                    (img_LoG[i-1][j] < 0 and img_LoG[i+1][j] > 0) or \
                        (img_LoG[i-1][j] > 0 and img_LoG[i+1][j] < 0):
                    zero_crossing[i][j] = 255
            if img_LoG[i][j] < 0:
                if (img_LoG[i][j-1] > 0) or (img_LoG[i][j+1] > 0) or \
                (img_LoG[i-1][j] > 0) or (img_LoG[i+1][j] > 0):
                    zero_crossing[i][j] = 255

    return zero_crossing
```

Discussion

GUI 在此部分只會顯示原圖跟 Marr Hildreth Edge Detection 的結果。



將 Marr Hildreth(左)跟 Sobel(右)做比較，此處的 Sobel 為對 X 方向求一階導數。
以下為 test.py 中 part3 的 function。



Part4

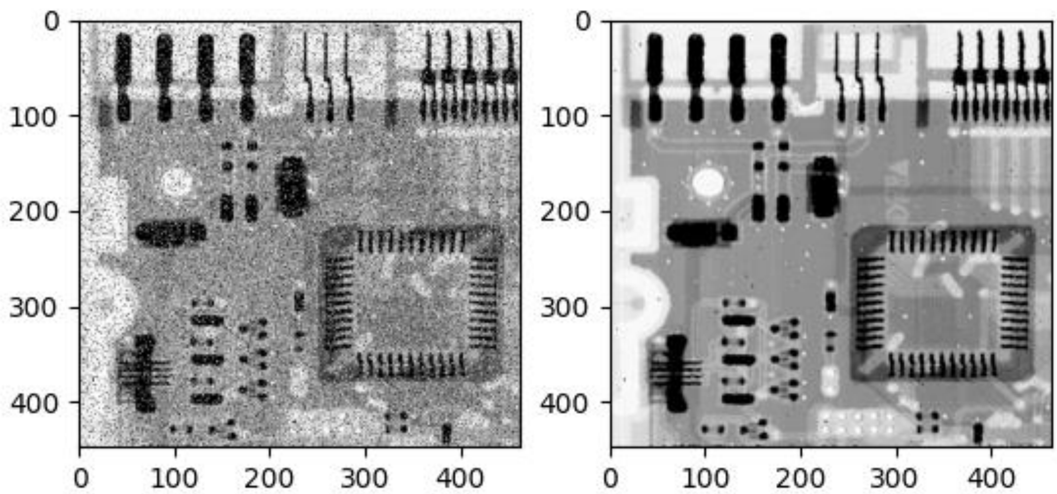
Code

```
def median_filter(img, filter_size=3):
    padding = (filter_size - 1)//2

    # zero padding
    padding_img = cv2.copyMakeBorder(
        img, padding, padding, padding, padding, cv2.BORDER_CONSTANT, value=0)

    y, x = padding_img.shape

    y_out = y - filter_size + 1
    x_out = x - filter_size + 1

    # convolution
    new_img = np.zeros((y_out, x_out))
    for i in range(y_out):
        for j in range(x_out):
            sort_array = sorted(
                padding_img[i:i+filter_size, j:j+filter_size].flatten())
            new_img[i][j] = np.median(sort_array)

    new_img = new_img.astype(np.uint8)

    return new_img
```

```
def max_filter(img, filter_size=3):
    padding = (filter_size - 1)//2

    # zero padding
    padding_img = cv2.copyMakeBorder(
        img, padding, padding, padding, padding, cv2.BORDER_CONSTANT, value=0)

    y, x = padding_img.shape

    y_out = y - filter_size + 1
    x_out = x - filter_size + 1

    # convolution
    new_img = np.zeros((y_out, x_out))
    for i in range(y_out):
        for j in range(x_out):
            sort_array = sorted(
                padding_img[i:i+filter_size, j:j+filter_size].flatten())
            new_img[i][j] = np.max(sort_array)

    new_img = new_img.astype(np.uint8)

    return new_img
```

```python
def min_filter(img, filter_size=3):
    padding = (filter_size - 1)//2

    # zero padding
    padding_img = cv2.copyMakeBorder(
        img, padding, padding, padding, padding, cv2.BORDER_CONSTANT, value=0)

    y, x = padding_img.shape

    y_out = y - filter_size + 1
    x_out = x - filter_size + 1

    # convolution
    new_img = np.zeros((y_out, x_out))
    for i in range(y_out):
        for j in range(x_out):
            sort_array = sorted(
                padding_img[i:i+filter_size, j:j+filter_size].flatten())
            new_img[i][j] = np.min(sort_array)

    new_img = new_img.astype(np.uint8)

    return new_img
```

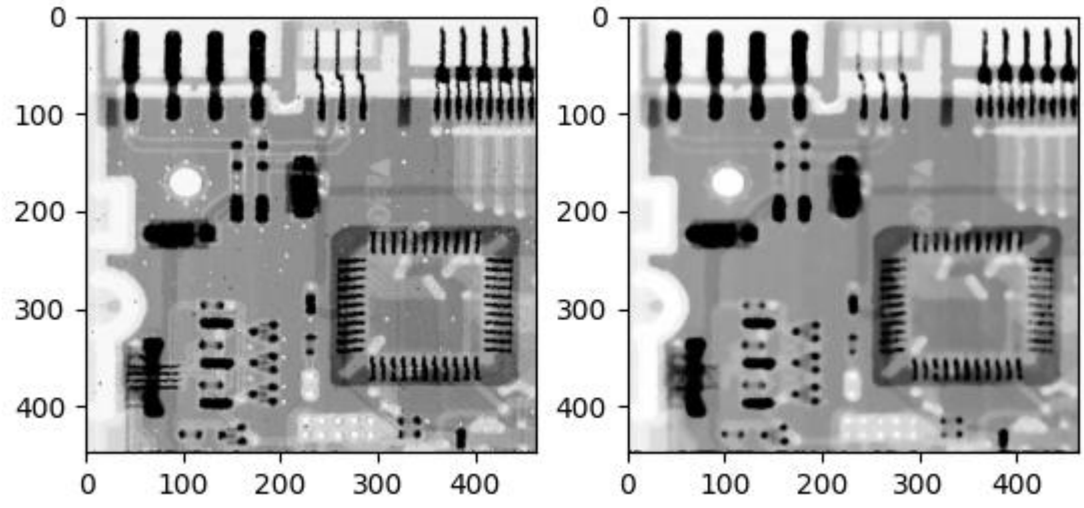
GUI，左邊為原圖，右邊為經過 filter 處理過後的圖片。



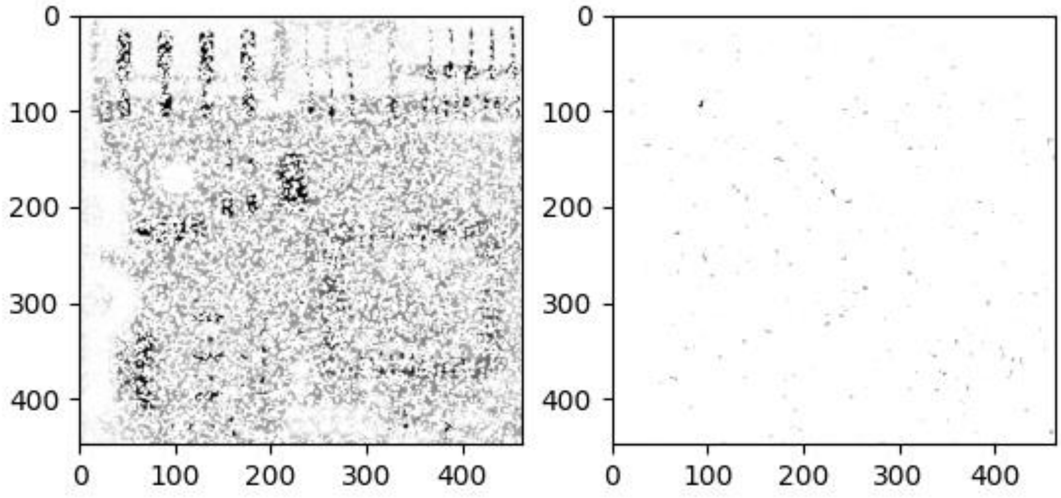
以下為 test.py 中 part4 的 function。

Kernel size = 3 (左) Kernel size = 20(右)

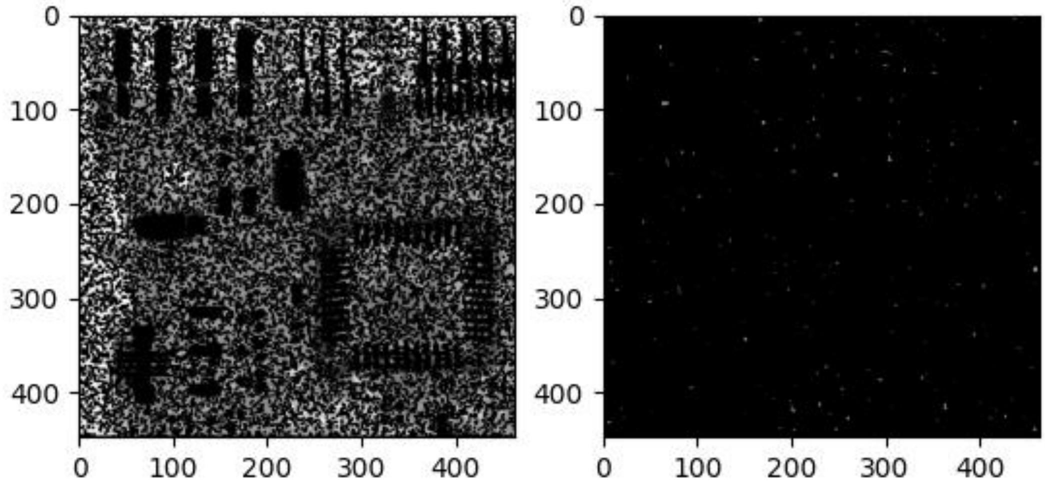
Median filter



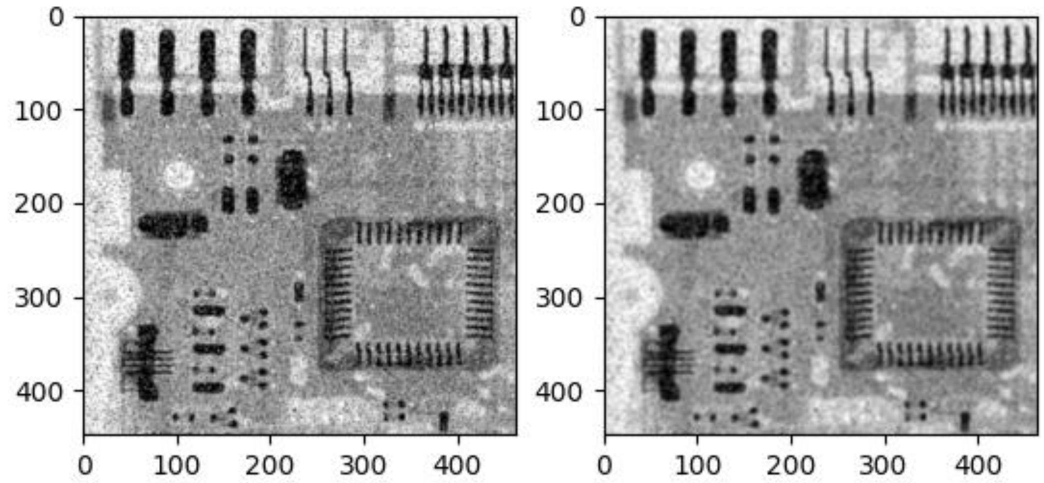
Max filter



Min filter



Gaussian 標準差 = 0



從這些結果可以發現，filter size 越大，效果越劇烈。