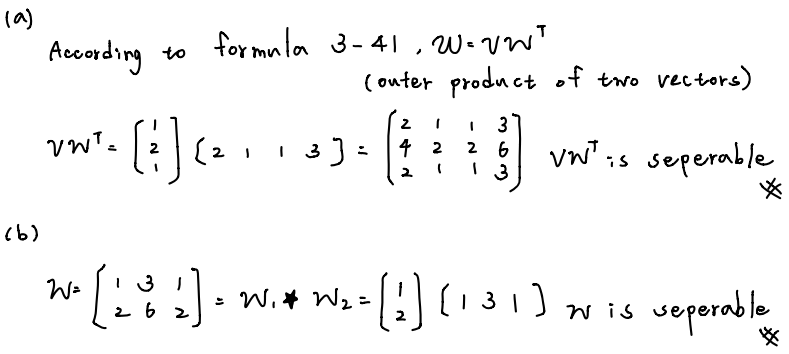
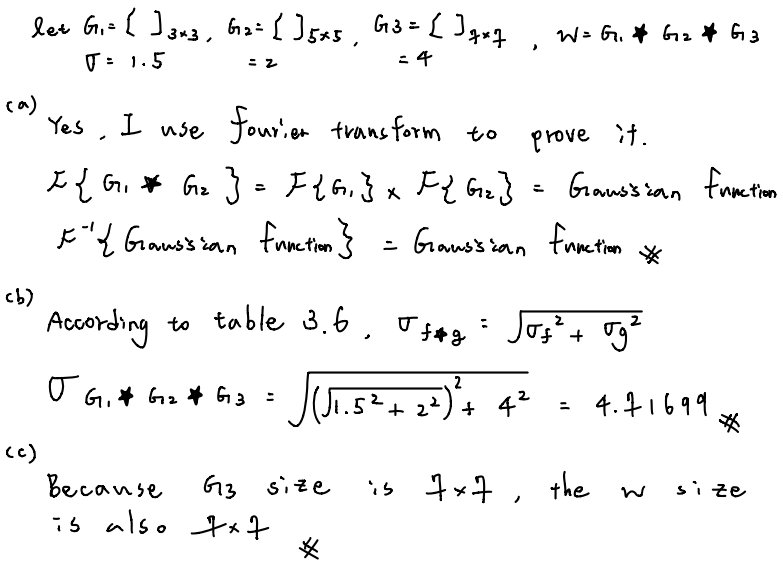
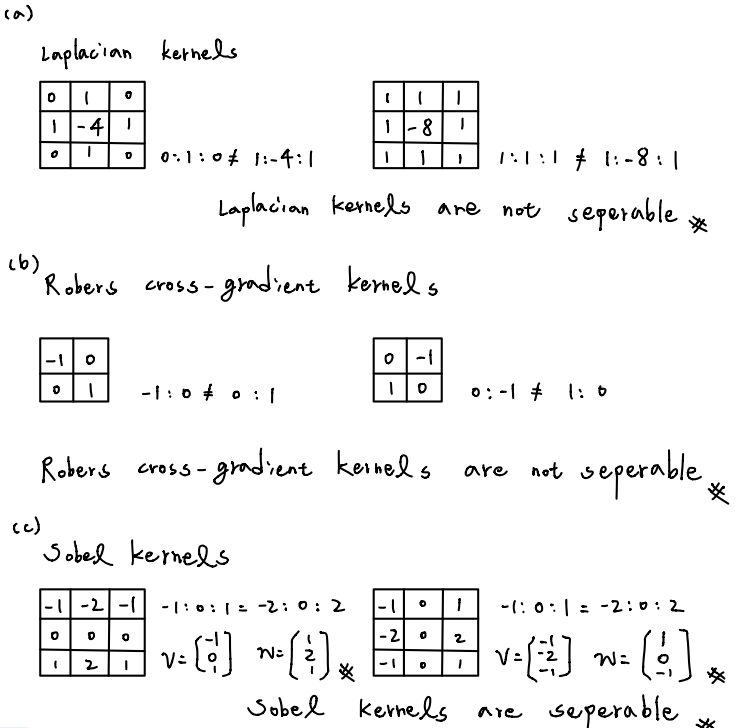
Digital Image Processing HW03 Report

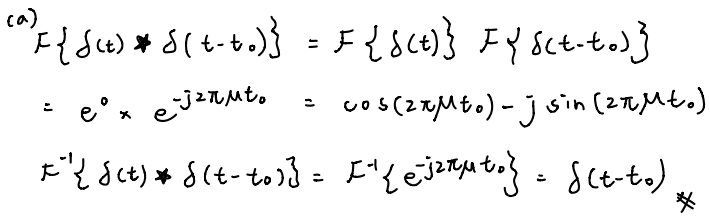
生機碩一 R09631007 吳乙澤

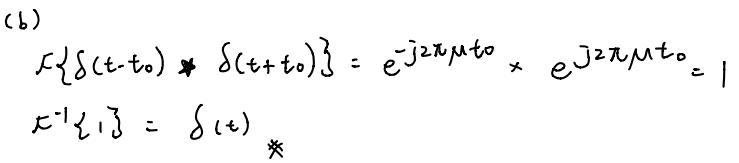
**一、課本習題**

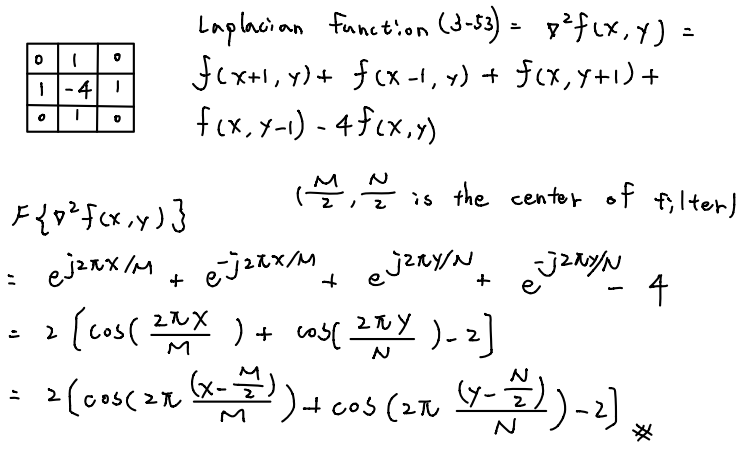
**3.22**

**3.28**

**3.44** 

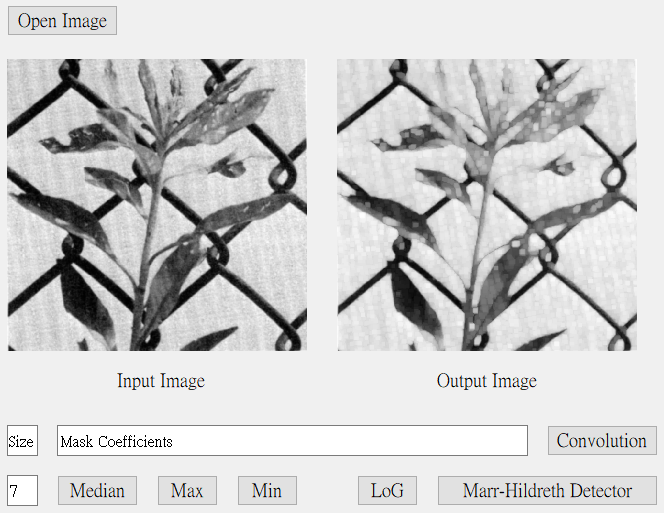
**4.3**



**4.51**

**二、軟體簡介**

以下介紹本次作業所寫出的軟體及使用方法。開啟圖片後，需於 Size 處輸入濾波器大小，大小為奇數。 3 代表 3 \* 3 、 5 代表 5 \* 5，依此類推。若濾波器太大會導致程式當掉。Mask Coefficients 能讓使用者自訂濾波器係數，可為小數、負數，但不可以分數形式表示。注意，輸入時需在係數間插入空白，並從左至右、從上到下的順序輸入。舉例，Size 處輸入 3，Mask Coefficients 處輸入 -1 0 1 -2 0 2 -1 0 1，該係數便是一個 sobel filter。另外，下方還有個輸入框可輸入最大、最小、中值濾波的濾波器大小，輸入規則同 Convolution。右下角有 LoG 和 Marr-Hildreth Detector 按鈕可以做邊緣檢測。邊緣檢測的濾波器大小與係數是固定的，不可自訂。最後，本軟體的濾波效果不可疊加。例如，做完中值濾波後，做 LoG。則此時是 LoG 對原圖作用，而不是對中值濾波後的結果做 LoG。



**三、演算法說明與結果討論**

本作業有三個功能需完成。以下依序說明這些功能的演算法並討論結果。

* 1. **Adjusting mask size and setting coefficients in the mask**

該段我分成兩個部分說明。第一部分說明我如何達成讓使用者能自行調整濾波器係數；第二部分則說明卷積的程式碼。

第一部分，見下方程式碼，我以字串型的變數coefficientsString，儲存使用者輸入的濾波器係數，如「0 0 0 -4 8 -4 0 0 0」。接下來將coefficientsString以空格為單位，分割成各個獨立字串。最後，以atof函數，將字串型的變數轉為雙精度浮點數，並放到coefficientsDouble中。

coefficientsString = ui->maskCoefficients->text().toStdString();

stringstream input(coefficientsString);

coefficientsDouble.clear();

while(input>>splitResult)

coefficientsDouble.push\_back(atof(splitResult.c\_str()));

見下方程式碼，為節省空間我把部分程式刪除，只留下較重要的片段。我利用四個for迴圈來完成卷積。前面的兩個迴圈，用來鎖定卷積時濾波器的中心位置。之後的兩個迴圈根據濾波器的中心位置，找到圖片要卷積的像素值。最後，從coefficientsDouble取出濾波器係數，再與像素值相乘後相加即可完成卷積。

for (int i = (maskSize+1)/2-1 ; i < imgCols-(maskSize-1)/2 ; i++)

for (int j = (maskSize+1)/2-1 ; j < imgRows-(maskSize-1)/2 ; j++)

for (int x = i-(maskSize-1)/2 ; x < i-(maskSize-1)/2+maskSize ; x++)

for (int y = j-(maskSize-1)/2 ; y < j-(maskSize-1)/2+maskSize ; y++)

{

double B = MatImgIn.at<Vec3b>(y,x)[0];  
 double G = MatImgIn.at<Vec3b>(y,x)[1];  
 double R = MatImgIn.at<Vec3b>(y,x)[2];

BGRCoefficient = coefficientsDouble.at(index);

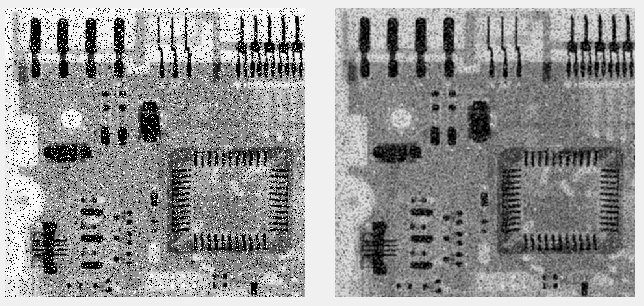
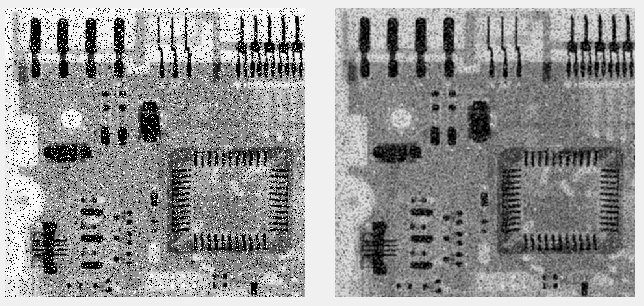
pixelB += (B \* BGRCoefficient);

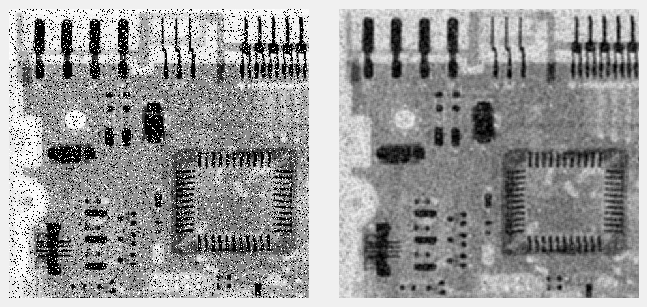
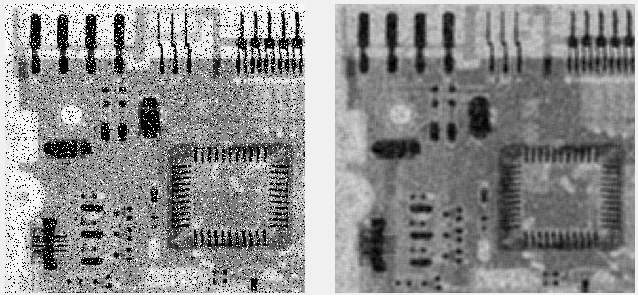
pixelG += (G \* BGRCoefficient);

pixelR += (R \* BGRCoefficient);

}

以下討論濾波的結果。首先，比較不同大小的低通濾波器，對圖片所造成的不同影響及運算時間。見下圖，可以很容易地發現，越大的低通濾波器，對圖片的模糊效果越顯著。這是因為越大的濾波器，所涵蓋的運算範圍也越大所導致的。另外，運算時間的部分，在該例子中，3\*3跟7\*7的運算時間有差異但相差不多。如果選擇更大的圖片做卷積，想必運算時間也會增加許多。

左上為原圖、右上經過3\*3大小的低通濾波器、  
左下經過5\*5大小的低通濾波器、右下經過7\*7大小的低通濾波器

   
蔬果。銳利化

* 1. **Zero-crossing threshold on the Marr-Hildreth edge detection method**

Marr-Hildreth Detector 先對圖片使用 LoG (Laplacian of Gaussian) 運算後，再找過零點做二值化。見下方程式碼，我將 LoG 濾波器的係數寫死，並用 3-1 部分的程式碼做卷積運算，之後把卷積結果放入 MatForZero 中。最後，再使用兩個 for 迴圈，配合 if 找出過零點並將之設為 255，若非過零點則將像素值設為 0。因此，運算後的圖片，白色處為物體邊緣，黑色處為非邊緣。

double log3DMask[9][9] = {{0, 0, 3, 2, 2, 2, 3, 0, 0},

{0, 2, 3, 5, 5, 5, 3, 2, 0},

{3, 3, 5, 3, 0, 3, 5, 3, 3},

{2, 5, 3, -12, -23, -12, 3, 5, 2},

{2, 5, 0, -23, -40, -23, 0, 5, 2},

{2, 5, 3, -12, -23, -12, 3, 5, 2},

{3, 3, 5, 3, 0, 3, 5, 3, 3},

{0, 2, 3, 5, 5, 5, 3, 2, 0},

{0, 0, 3, 2, 2, 2, 3, 0, 0}};

MatForZero.at<double>(i-(maskSize+1)/2+1, j-(maskSize+1)/2+1) = pixel;

// Find zero-crossing

for (int i = 1 ; i < imgCols-maskSize ; i++)

{

for (int j = 1 ; j < imgRows-maskSize ; j++)

{

if ((MatForZero.at<double>(i - 1, j) \* MatForZero.at<double>(i + 1, j) <= 0) || (MatForZero.at<double>(i, j + 1) \* MatForZero.at<double>(i, j - 1) <= 0) || (MatForZero.at<double>(i + 1, j - 1) \* MatForZero.at<double>(i - 1, j + 1) <= 0) || (MatForZero.at<double>(i - 1, j - 1) \* MatForZero.at <double> (i + 1, j + 1) <= 0))

QImgOut.setPixel(i, j, qRgb(255, 255, 255));

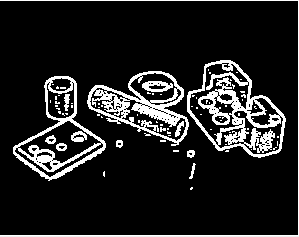
else

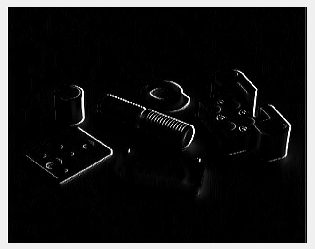
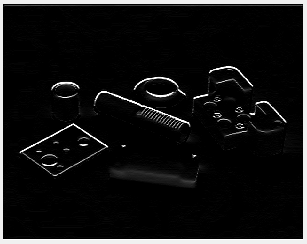
QImgOut.setPixel(i, j, qRgb(0, 0, 0));

}

}

下圖為Marr-Hildreth Detector跟Sobel Filter的比較。可以發現，Marr-Hildreth Detector能同時偵測直線方向跟橫線方向的邊緣，Sobel Filter則只能一次偵測一個方向的邊緣。並且，由於Marr-Hildreth Detector有做二值化，因此圖片只有黑或白兩種顏色，整體看起來比Sobel的濾波效果乾淨許多。

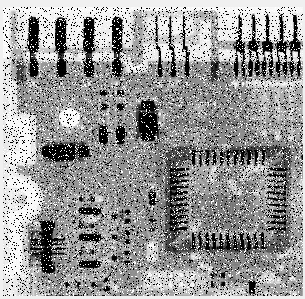
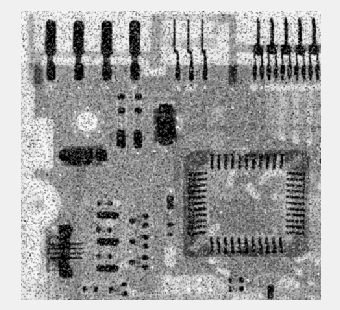
 

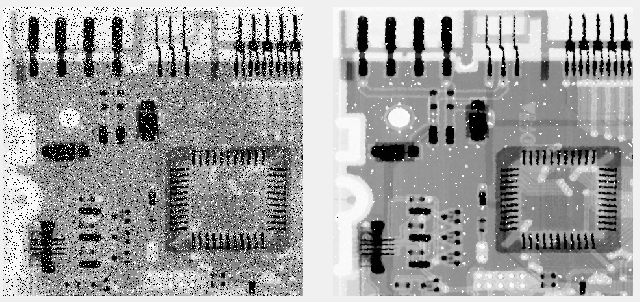
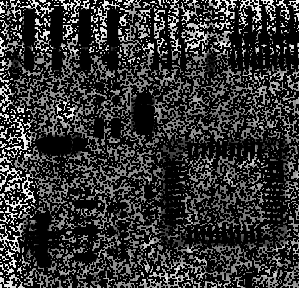
從左上到右下依序為原圖、Marr-Hildreth Detector、  
3\*3 直線方向的 Sobel Filter、3\*3橫線方向的 Sobel Filter

* 1. **Compare order-statistic filter function and gaussian filter**

最大、最小、中值等三個濾波器的程式碼和 3-1 的部分雷同，一樣是使用四個 for 迴圈。唯一的差別在於，不需要拿濾波器係數跟圖片像素相乘，只要找出濾波器遮罩內的最大值、最小值、中值即可。見下方程式碼，我使用 Vector 的 sort 方法進行排序，再以適當的索引值從 Vector 中取出最大值、最小值、中值後，放入 QImgOut 即可完成該題。  
  
sort(pixelSeriesR.begin(), pixelSeriesR.end());sort(pixelSeriesG.begin(), pixelSeriesG.end());sort(pixelSeriesB.begin(), pixelSeriesB.end());// Set pixelpixelB = pixelSeriesB.at((pixelSeriesB.size()+1)/2);pixelG = pixelSeriesG.at((pixelSeriesG.size()+1)/2);pixelR = pixelSeriesR.at((pixelSeriesR.size()+1)/2);QImgOut.setPixel(i-(orderMaskSize+1)/2+1, j-(orderMaskSize+1)/2+1, qRgb(pixelR, pixelG, pixelB));

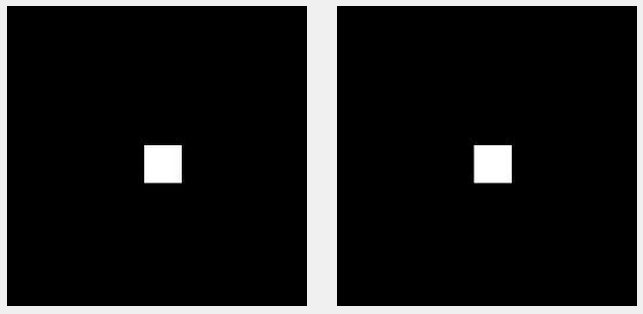
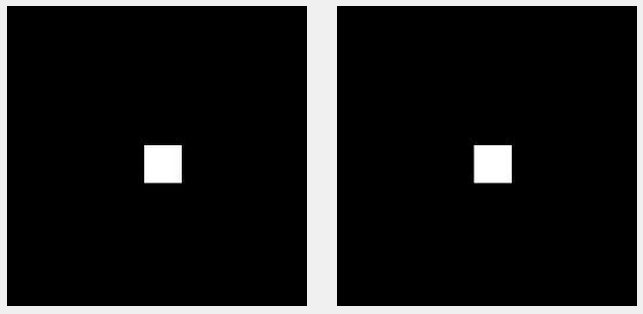
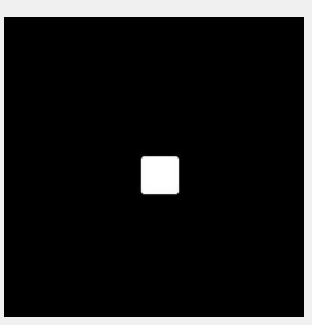
見下圖，這些圖片都是經過3\*3大小的濾波器濾波而成的。比較高斯跟中值濾波，可以看出高斯濾波後的圖片，仍有許多黑白點存留，中值濾波後的黑白點卻少得多，表示中值濾波去椒鹽雜訊的能力，比高斯濾波強。

由左上到右下依序為原圖、高斯濾波、中值濾波、最大濾波、最小濾波

見下圖，比較中值濾波器的大小對圖片造成的影響。該圖的中間為白色的正方形，背景為黑色。仔細觀察圖片，可以從3\*3中值濾波到31\*31中值濾波的圖形變化，發現越大的濾波器越會將正方形的邊角吃掉，但是正方形的中心卻始終保持白色。這個現象的原因，可以從濾波的運算和該張圖片的特性來看。由於方形中心部分的像素值單一，因此濾波後的結果和原來圖片的像素值沒有差異。有差異的部分只出現在像素值不一致的地方，也就是方形的邊緣。

由左到右依序為原圖、3\*3中值濾波、11\*11中值濾波、31\*31中值濾波