# 影像處理作業

# Cosmic Ray Remove by Using Laplacian Operator and Self Made Mask

系級：資工碩一

學號：110522128

姓名：林彥宇

2021 年 1 月 14 日

# 目錄

1. 簡介
2. 相關方法
3. 實驗
4. 心得與討論
5. 參考文獻

# 一、簡介

宇宙射線 (Cosmic Ray)是一種帶電高能次原子粒子，在拍攝天文影像的時候經常會妨礙天文學家觀測發光天體，因此如何乾淨的去除宇宙射線，並且盡可能地還原圖片的原貌尤其是保住發光天體本來的樣貌在天文領域是一件重要的事情。第一眼看到圖片的時候，發現宇宙射線有個特徵是邊緣非常銳利，與周遭的像素色差很大，且很白；而發光天體雖然也很白，但可以注意到發光天體有明顯的光暈，因此我想到的方法是用先用**邊緣偵測**先找出宇宙射線的位置，並且標記座標之後，再到原本的圖片用自製的遮罩做**相鄰像素平均法**或使用**最小濾波器**將宇宙射線去除。

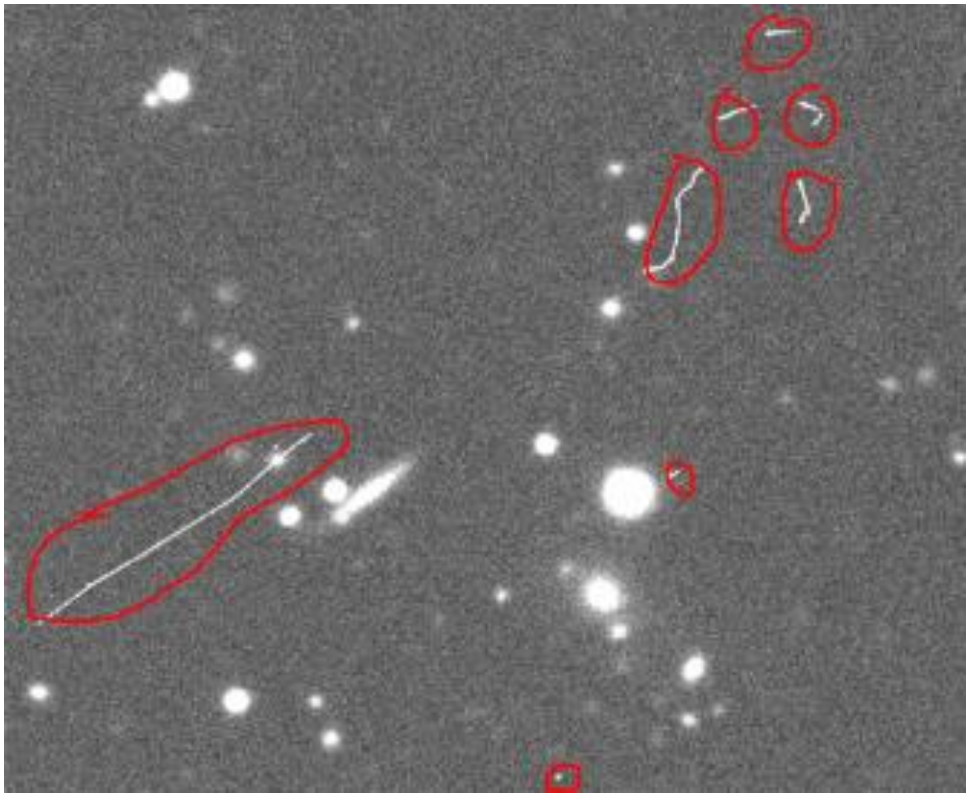

圖 1　紅筆圈起來的部分就是宇宙射線

# 二、相關方法

## 2-1　邊緣檢測

在這次的作業中，我嘗試了 3 種不同的邊緣檢測演算法，分別是 Sobel 算子、Canny 算子和拉普拉斯算子。測試各種算法時使用的圖片為圖 1 的原圖。

### Sobel 算子[1]：

Sobel 算子只用圖像的梯度最為判斷依據，對不同方向的邊界是用兩個不同的 Mask 來做偵測(圖 2)。

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 0 | -1 | | 1 | 2 | 1 |
| 2 | 0 | -2 | | 0 | 0 | 0 |
| 1 | 0 | -1 | | -1 | -2 | -1 |

圖 2　左邊的遮罩負責偵測水平方向的邊界，右邊的遮罩負責偵測垂直方向的邊界

當梯度變化超過一個閥值就會被判斷為邊界(圖 3)，將遮罩對圖片做捲積運算後即可得到整張圖片的邊界了。 Sobel 算子是一個離散的微分算子。它計算圖像強度函數的梯度的近似值，且 Sobel 算子結合了高斯平滑和微分。

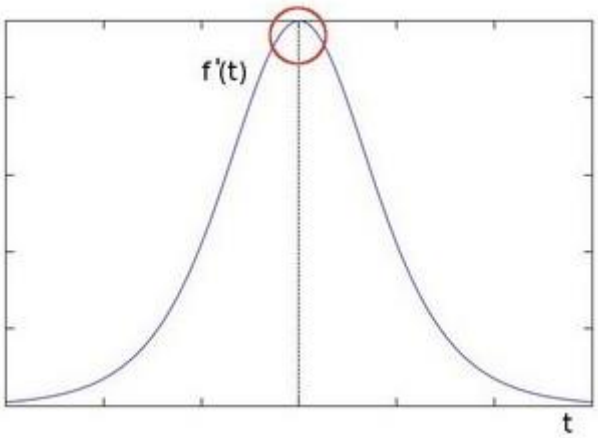


圖 3　通過定位高於高於其鄰點的像素位置或閥值，即能夠找到邊緣

具體的運算過程如下：
*Gx* 和 *Gy* 為水平方向及垂直方向邊緣檢測的影像，*I* 為原始影像，

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I \qquad \text{和} \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

每一個像素的水平及垂直方向的梯度近似值可以用一下的公式求得梯度大小：

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

然後再用下面的公式得到梯度方向：

$$\theta = arctan\left(\frac{G_y}{G_x}\right)$$

## 拉普拉斯算子[2]：

首先，拉普拉斯算子是一個二階微分算子，而二階導數為 0，我們也能用這個標準來檢測影像中的邊緣(圖 4)。

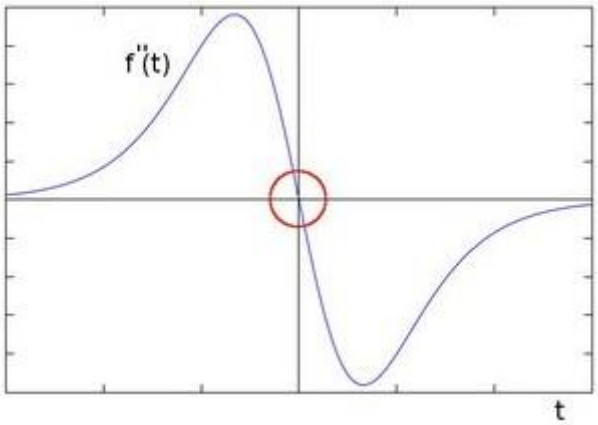


圖 4　0 也可能不會出現在邊緣，這會需要其他方式來進行過濾處理

由於影像資料是 2 維的，所以我們需要在兩個維度上取得導數，而拉普拉斯算子被定義如下：
*src* 是原始影像。

$$\mathtt{dst} = \Delta \mathtt{src} = \frac{\partial^2 \mathtt{src}}{\partial x^2} + \frac{\partial^2 \mathtt{src}}{\partial y^2}$$

拉普拉斯算子使用的遮罩為圖 5，透過這個遮罩作用到影像的每一個像素上可以得到二次微分的結果。

| | | |
|---|---|---|
| 0 | -1 | 0 |
| -1 | 4 | -1 |
| 0 | -1 | 0 |

圖 5 實作時通常以此遮罩對影像做捲積運算

## Canny 算子[3]：

Canny 有三大特徵:

1.低錯誤率ㄧ偵測出的亮點都會是邊緣

2.良好的定位ㄧ標示出的邊緣像素與實際邊緣像素之間的差距很小

3.解析度高ㄧ線條很細

Canny 是一個複合性的算法，第一步會先用圖 6 的高斯濾波器來去除噪點：

$$K = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

圖 6 透過此矩陣對影像做捲積運算就可以減少影像雜訊及降低細節層次

第二步會用 Sobel 算子來得到影像中，邊緣的強度與方向

第三步為，找出最大梯度的方向和最大值抑制

3-1 最大梯度的方向通常分為 4 種，分別為 0 度、45 度、90 度及 135 度

3-2 決定了梯度方向之後，就會將這方向中比較小的值歸零

第四步是最後一步，會決定兩個閥值（高的閥值與低的閥值，比例通常是 2:1 或 3:1），如果像素的梯度大於高的閥值，則該像素就是邊界，如果低於低的閥值，就不是邊界，如果像素的值介於兩個閥值之間，那只有他連接一個高於閥值的像素才會被接受為邊界。

## 2-2　雜訊去除

在這次的作業中，我嘗試了 2 種不同的平滑化演算法，分別是相鄰像素平均法和最小濾波器。測試時使用的圖片為圖 1 的原圖。

### 相鄰像素平均法：

若以 3x3 的遮罩為例，將指定像素及其 8 鄰點乘以權重遮罩後全部加起來除以權重遮罩的總和，再用得到的值取代指定像素。

### 相最小濾波器：

若以 3x3 的遮罩為例，取得指定像素的 8 鄰點像素中灰階最小的值，並取　代指定像素。

# 三、實驗

## 第一步：

由於宇宙射線的邊緣很銳利，因此我認為能夠用邊緣檢測直接找到宇宙射線，於是我先測試了各種邊緣檢測演算法來測試哪一種效果較佳。實驗使用的圖片都為圖 1 的原圖一圖 6。

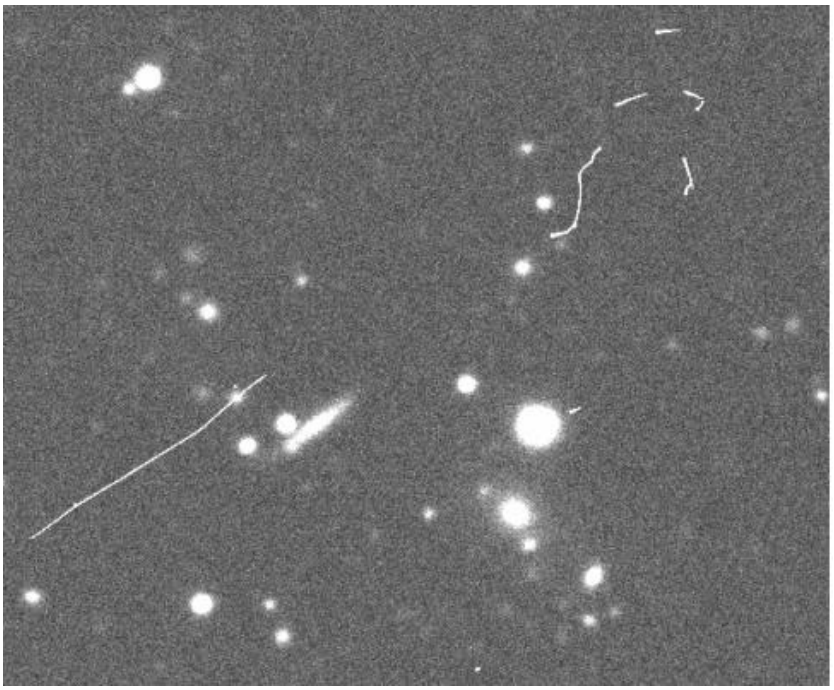

圖 6　實驗測試用的輸入影像

### 實驗 1-1：

首先是 **Sobel 算子**(圖 7)，輸入圖 6，得到的結果為圖 8。宇宙射線是能夠邊緣檢測出來的，但是其他發光天體的光暈也被偵測出來了，所以沒有採用 Sobel 算子。

```
sobelX = cv2.Sobel(img, cv2.CV_64F, 1, 0)
sobelY = cv2.Sobel(img, cv2.CV_64F, 0, 1)
sobelX = np.uint8(np.absolute(sobelX))
sobelY = np.uint8(np.absolute(sobelY))
sobelCombined = cv2.bitwise_or(sobelX, sobelY)
```

圖 7 Sobel 算子的程式碼

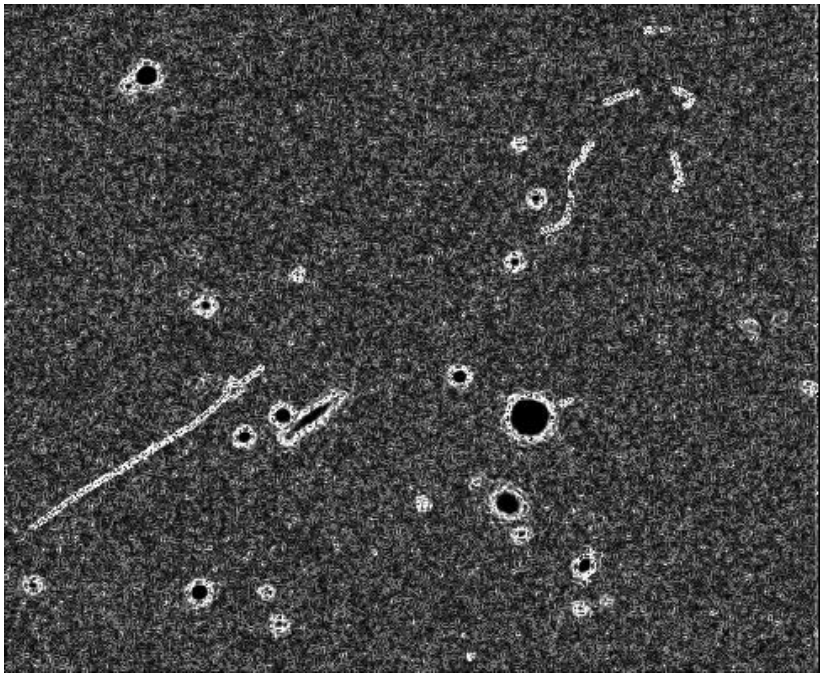

圖 8　用 sobel 算子，且 xy 軸都列入計算的結果。

**實驗 1-2：**

第二個測試的是用 **canny 算子**，得到的結果為圖 9。canny 算子雖然是非常優秀的邊緣檢測演算法，但由於其特性，解析度太高，太清晰與良好的定位，導致圖片難以辨識宇宙射線，反而使 canny 算子不適合這次的主題，因此也**沒有採用**。

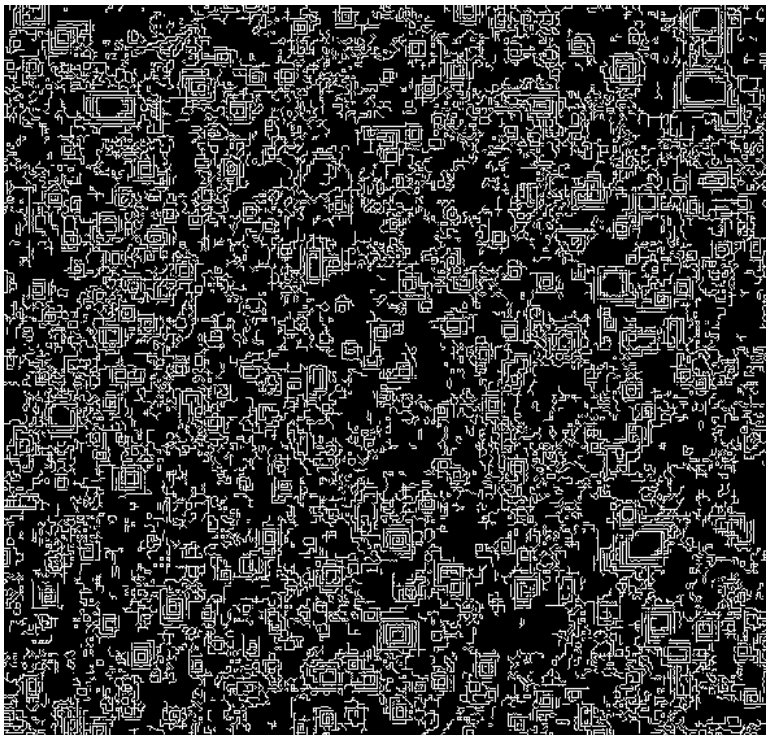

圖 9　canny 算法得出的結果已經無法辨識內容物

**實驗 1-3：**

最後測試的演算法是**拉普拉斯算子**，使用 openCV 的拉普拉斯算子得到的結果為圖 10。與其他邊檢測的演算法相比，拉普拉斯算子能夠明顯的取出宇宙射線，且會忽略掉發光天體，對於本次的專題來說是最合適的邊緣檢測演算法。因此決定使用拉普拉斯算子作為本專題的邊緣檢測演算法。

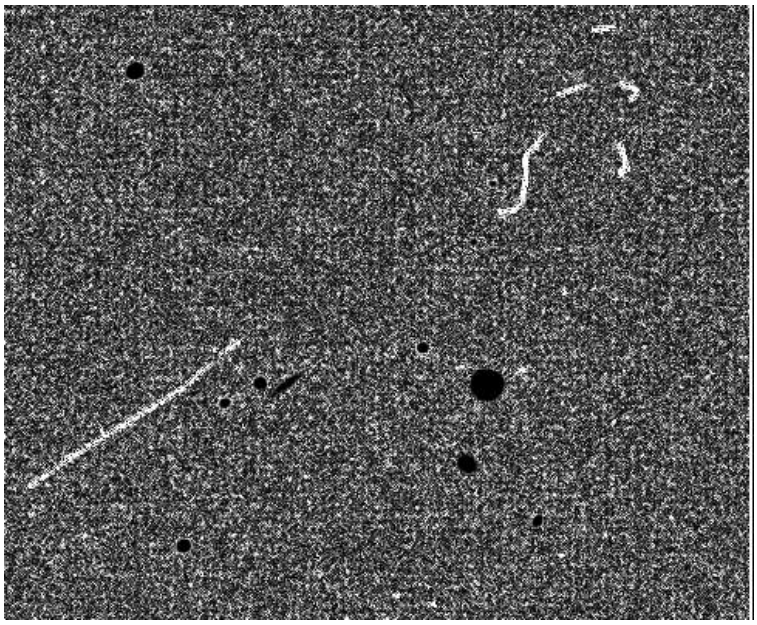

圖 10　拉普拉斯算子得到的結果能夠清楚辨識宇宙射線並忽略發光天體

由於決定使用拉普拉斯算子，因此稍微嘗試實做了一下拉普拉斯子，程式與結果為圖 11 及圖 12。雖然自製的拉普拉斯算子也能清楚地取出宇宙射線並忽略發光天體，但與 openCV 的拉普拉斯算子相比，解析度或是明度都比較低，推測還需要做影像增強才能達到理想的成果，因此後來實驗還是以使用 openCV 的拉普拉斯算子為主。

```python
def laplace(img, ksize=3):
    '''
    自製拉普拉斯邊緣檢測
    '''
    height = img.shape[0]
    width = img.shape[1]
    output_image = np.zeros(shape=(height, width))
    if ksize == 3:
        filter = np.array([[0, -1, 0],
                           [-1, 4, -1],
                           [0, -1, 0]])

    for i in range(1, height-1):
        for j in range(1, width-1):
            output_image[i, j] = abs(np.sum(img[i-1:i+2, j-1:j+2]*filter))
    return np.uint8(output_image)
```

圖 11　自製的拉普拉斯算子，使用的遮罩沒有判斷斜邊

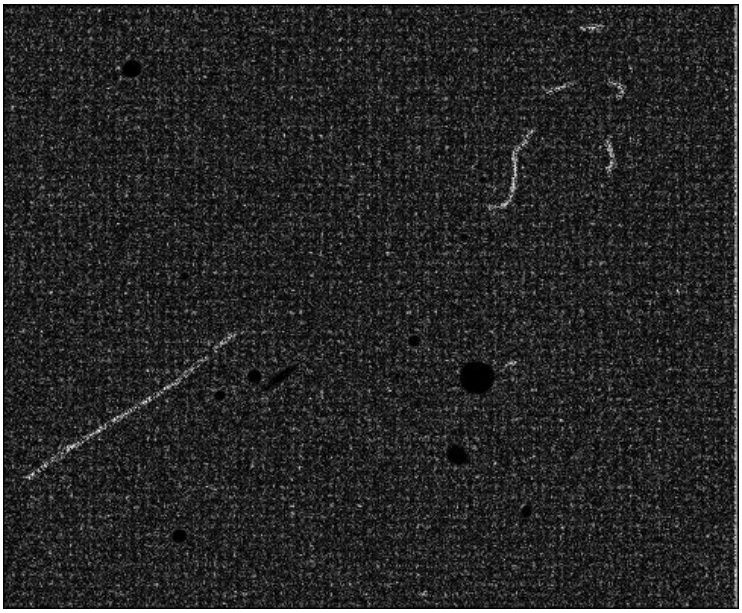

圖 12　自製拉普拉斯算子的出的結果

# 第二步：

接下來的就是標記影像中宇宙射線的位置，就實驗第一步的結果可以得到，檢測出是宇宙射線的部分，灰階值都很高，因此我手動設定一個閥值，只要指定像素的灰階值高於閥值，並且有兩個以上的相鄰像素的灰階值只有指定像素的一半，或是兩個以上的相鄰像素的值很接近就將指定像素標記為宇宙射線，存在另外一個陣列中。程式碼如圖 13 所示。

```
markedImgMap = np.zeros(shape=(img_rows, img_cols, 1), dtype=np.uint8)
for i in range(1, img_rows-1):
    for j in range(1, img_cols-1):
        markedImgMap[i, j] = MarkComsic(la_img, i, j, val=150)
```

```
def MarkComsic(img, row, col, val=230):

    center_value = img[row, col]
    same_color_point_count = 0
    diff_color_point_count = 0
    # 如果中心點是夠白的
    if center_value > val:
        # 判定周遭有多少顏色類似的點，且是否有落差極大的點
        for i in range(-1, 2):
            for j in range(-1, 2):
                # 正中間的點不計算
                if i == 0 and j == 0:
                    continue
                if img[row+i, col+j] > center_value-5 or img[row+i, col+j] < center_value + 5:
                    same_color_point_count += 1
                elif img[row+i, col+j] < center_value/2:
                    diff_color_point_count += 1
        # 是宇宙射線
        if same_color_point_count > 2:
            return 1
    # 不是宇宙射線
    return 0
```

圖 13　用兩個迴圈掃描影像，閥值設定為 150

把閥值設定為 230 的標記陣列做成圖的話，可以得到圖 14。

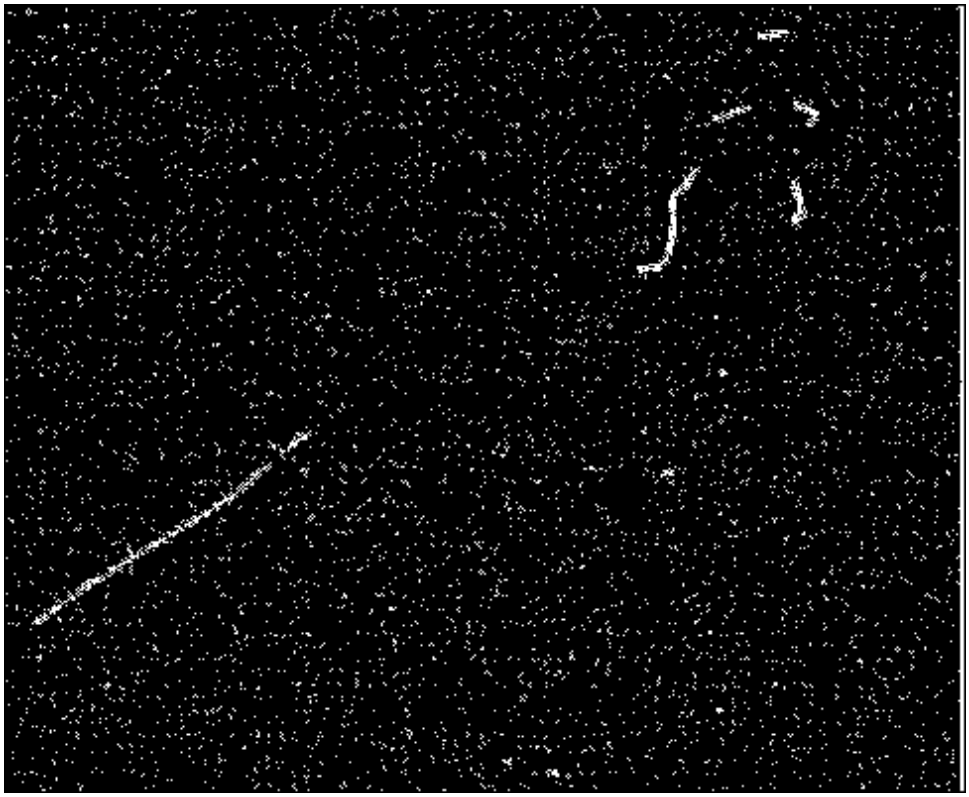

圖 14　被標記為宇宙射線的部分以白色呈現

# 第三步：

即最後一步，就是參考第二步標記完的陣列，回到原本的 Input image 來將宇宙射線抹除，這邊我測試了 5 種方式，分別為：

1. 靜態的遮罩做相鄰像素平均法
2. 動態生成的遮罩相鄰像素平均法
3. 靜態遮罩搭配重複實驗第 2-3 步，並逐步調整閥值
4. 動態生成的遮罩搭配重複實驗第 2-3 步，並逐步調整閥值
5. 最小濾波器直接消除宇宙射線

## 3-1：

**第一種方式**，我測試了一些遮罩，最後覺得效果最好的是圖 15 的遮罩，只取上面一列的 3 像素及左邊的 1 個像素。

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 0 | 0 |

圖 15　權重為 1 的像素必定不為宇宙射線，或是已經過處理

會這樣想是因為影像的掃描是由左上至右下，所以我只取相鄰且不為宇宙射線，或是已經過處理的像素來做指定處理。並且做矩陣的運算，即 2 個矩陣對應的值相乘，後相加得到的值來取代指定像素，如圖 16 所示。

```
def blurred(img, row, col, kernel=np.array([[1, 1, 1],
                                            [1, 0, 0],
                                            [0, 0, 0]])):

    value = np.array([[img[row-1, col-1], img[row-1, col], img[row-1, col+1]],
                      [img[row, col-1], img[row, col], img[row, col+1]],
                      [img[row+1, col-1], img[row+1, col], img[row+1, col+1]]])
    sum = np.sum(kernel)
    if sum == 0:
        return 125
    return np.sum(np.floor(kernel/sum * value))
```

圖 16　return 125 的部分在第一種方式不會使用到

用方法 1，並調整不同的閥值，得到的結果分別為圖 17。

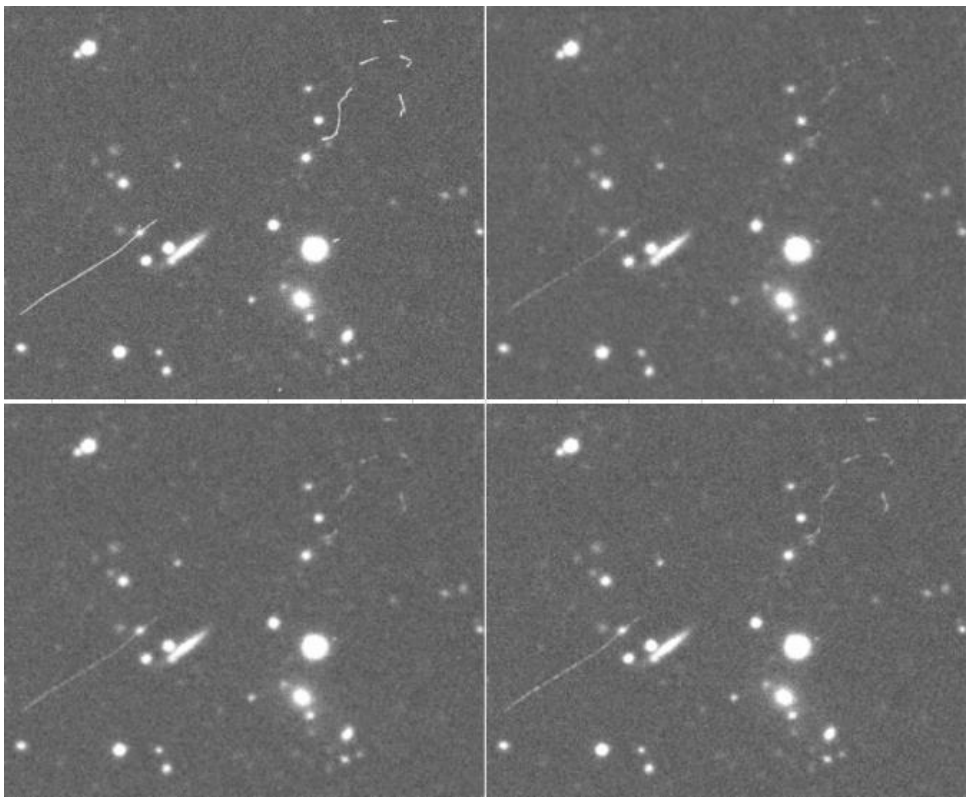

圖 17　左上為輸入的原圖。右上閥值為 100，宇宙射線，由其是圖片右上部分有特別明顯的削弱，但是將圖片放大來看的話，可以發現有些天體也被影響導致有缺角。

左下閥值為 150，宇宙射線的削除的效果雖然不及右上的圖，但其他天體受到的影響更小。右下閥值為 230，其他天體受到的影響又更小，但宇宙射線更加明顯一些。

圖 18 為受影響的發光天體的樣子。

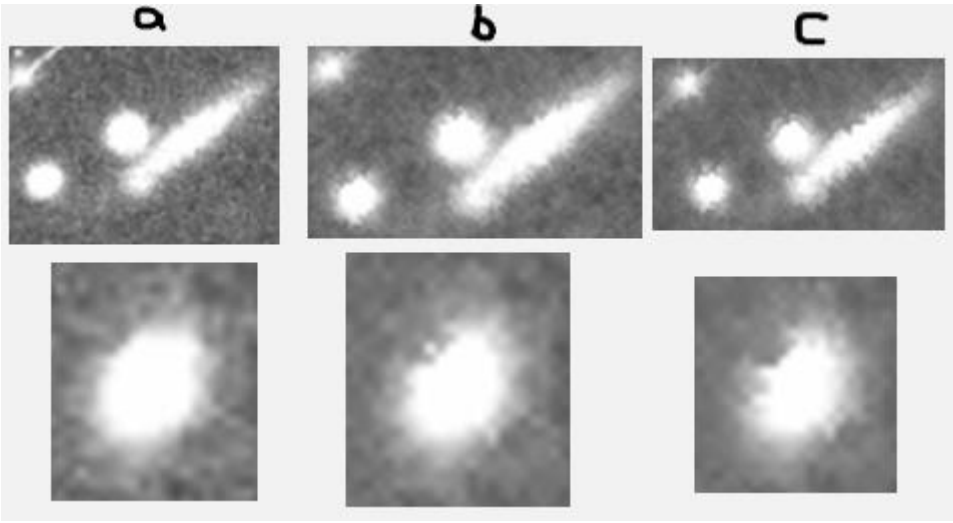


圖 18　a 行為原圖，b 行為閥值為 150 的圖，c 行為閥值為 100 的圖

## 3-2：

**方法 2** 中提到的動態生成遮罩，概念是，從指定要處理的像素的標記陣列的來看，鄰點是被標記的點權重就設 0，如果不是被標記的點權重就設 1，程式如圖 19。與原圖及方法 1 的比較為圖 20。

```
kernel = np.zeros(shape=(3, 3))
for x in range(-1, 2):
    for y in range(-1, 2):
        if markedImgMap[i+x, j+y] == 1:
            kernel[x+1, y+1] = 0
        else:
            kernel[x+1, y+1] = 1
img[i, j] = blurred(img, i, j, kernel)
markedImgMap[i, j] = 0
```

圖 19　kernel 的部分就是動態生成的遮罩

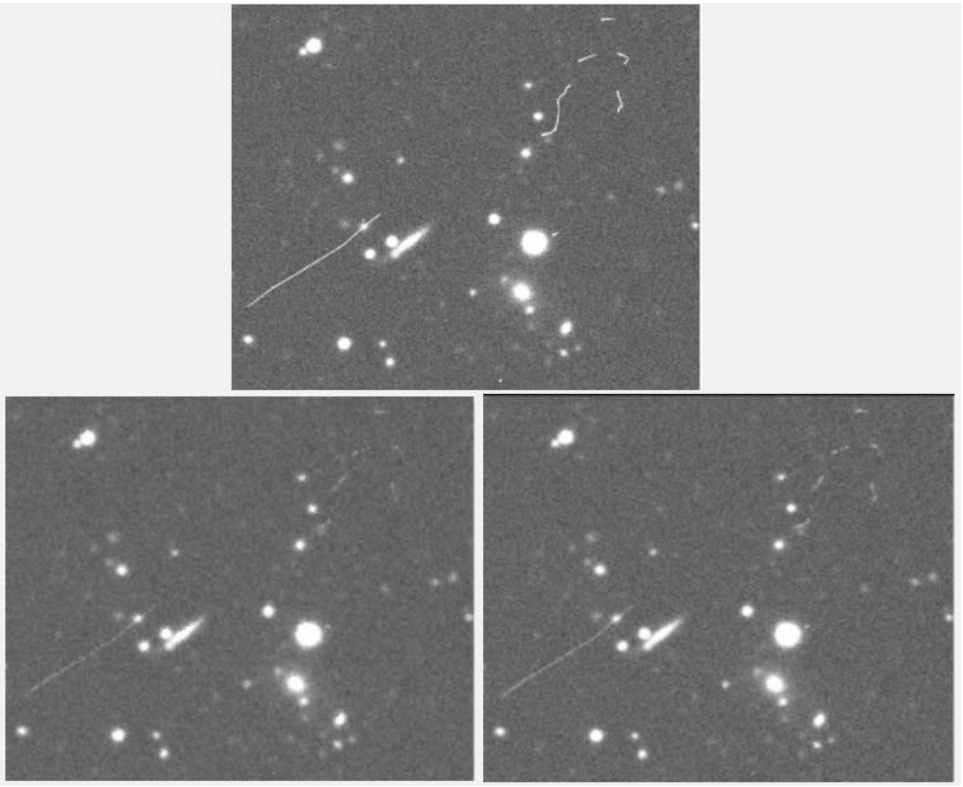

圖 20　**上圖**為原圖。**左圖**為閥值 150 的方法 1。**右圖**為閥值 150 的方法 2，與方法 1 相比，消除宇宙輻射方面，有顯著的效果，甚至可以說是稍微變差，但放大來看

的話，對天體的影響，也比較沒那麼大。

## 3-3：

基本上就是在方法 1 外面套一層 for 迴圈，並且每循環一次就就降低一次閥值，因
為經過第一次的處理後，宇宙射線就會變得沒那麼明顯了。得到的結果和方法 1 的結果比較如圖 21。

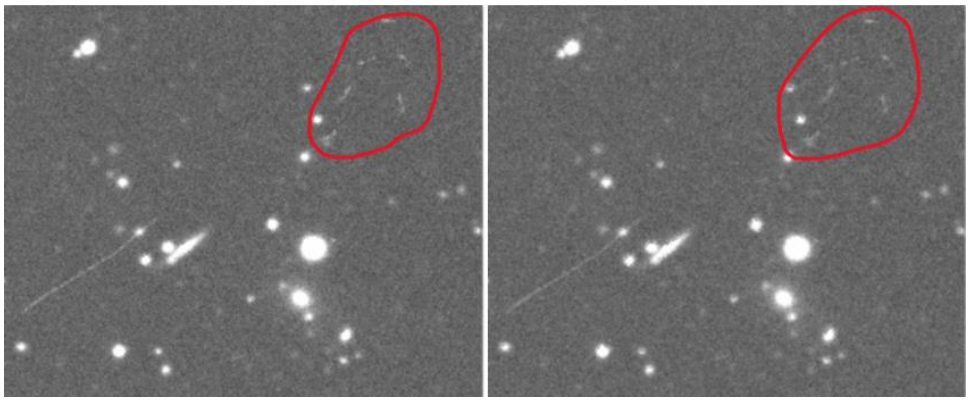

圖 21　左圖為方法 1，右圖為此方法。沒有顯著的差別，圈起來的部分變淡一些

## 3-4：

就是在方法 2 外面套一層 for 迴圈，並且每循環一次就就降低一次閥值，因為經過第一次的處理後，宇宙射線就會變得沒那麼明顯了。得到的結果和方法 2 的結果比較如圖 22。

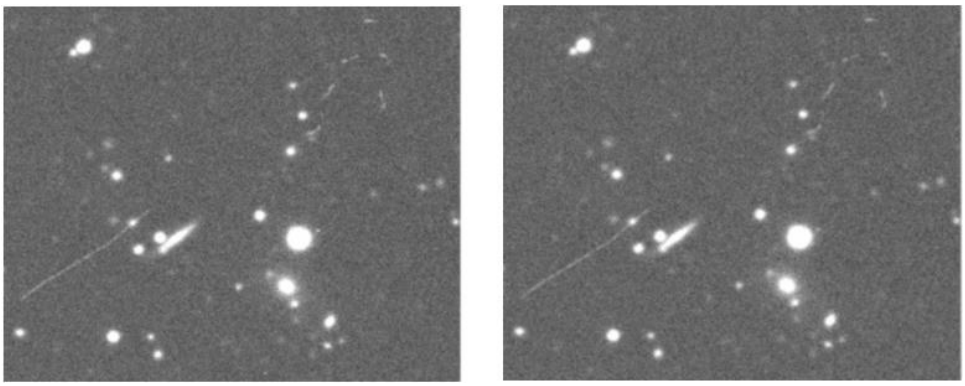

圖 22　左圖為方法 2，右圖為此方法。完全看不出有的差別。

## 3-5：

此部分的程式很單純就只是以宇宙射線為中心，用 3x3 範圍內最小的值來取代宇宙射線的像素。程式為圖 23，成果如圖 24 所示。

```
def min_filtering(img, row, col):
    '''
    最小濾波器
    value為原始圖片中指定像素與其8鄰點
    '''
    value = np.array([[img[row-1, col-1], img[row-1, col], img[row-1, col+1]],
                      [img[row, col-1], img[row, col], img[row, col+1]],
                      [img[row+1, col-1], img[row+1, col], img[row+1, col+1]]])

    return np.min(value)
```

圖 23　　value 為以宇宙射線為中心及其 8 鄰點組成的 3x3 陣列

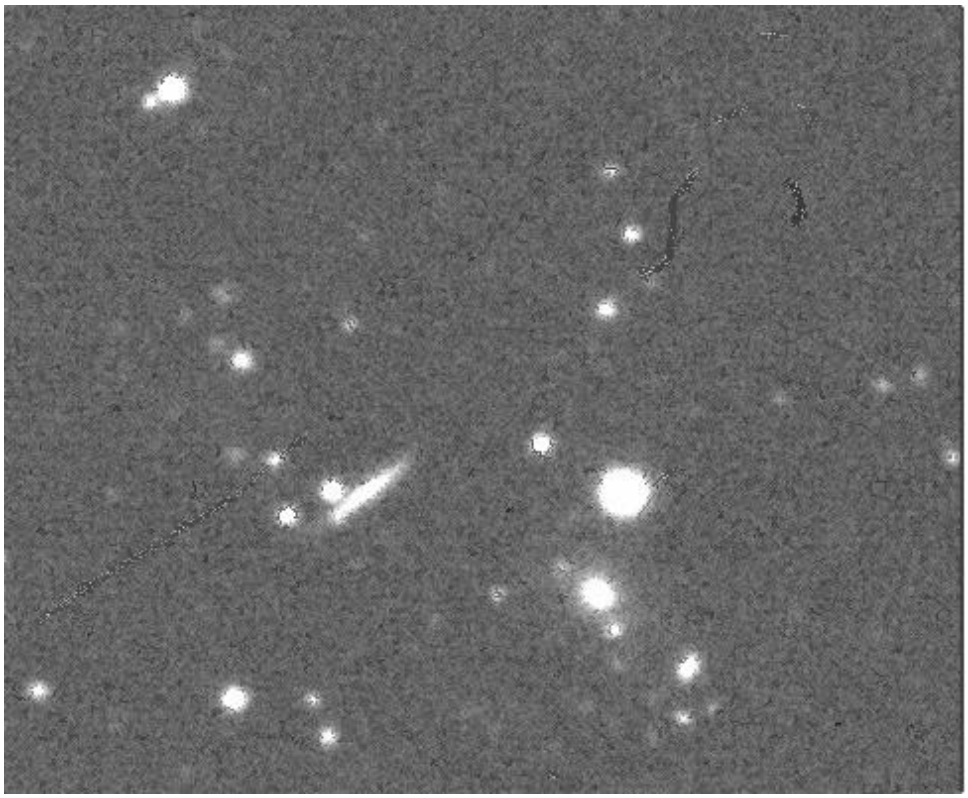

圖 24　　幾乎能夠完全去除宇宙射線原本的樣貌，但經過處裡的宇宙射線依然明顯

# 四、心得與討論

最後的結果不盡理想，雖然能夠移除部分的宇宙射線，但沒辦法全部去除，甚至大部分都只能做到淡化，想到的一個原因是，雖然拉普拉斯算子能夠取出大部分宇宙射線，但還是會有一些殘渣，而我想到處理的方式是，將原本標記為宇宙輻射的周遭的點也做另一種標記，再將那些部分也做處理，但最後發現會嚴重影響到發光星體，尤其是宇宙射線穿越發光星體的時候。另一方面是，消除宇宙射線用的方法本身可能需要重新構想，並不是單純一個遮罩能夠完全處理的，總的來說，還有很多細節的部份需要觀察與研究。

這次的影像處理課程讓我有了第一次實作影像處理契機，我認為是非常有趣且收穫良多的，也讓我知道要把一張影像處理到符合需求，是需要很多複雜技術的互相支持才能辦得到的。而這次選擇此題目是我跟一個中央大學天文所的朋友聊天而得到的靈感，我認為，如果是專業的天文人員，能夠得到更多數據的話，這個專題甚至能夠用深度學習來處理，也是一個很有趣的方向。

# 參考文獻


[1]
https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/sobel_derivatives/sobel_derivatives.html#sobel-derivatives (OpenCV- Sobel Derivatives)

[2]
https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/laplace_operator/laplace_operator.html#laplace-operator (OpenCV- Laplace Operator)

[3]
https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html#explanation (OpenCV – Canny Edge Detector)