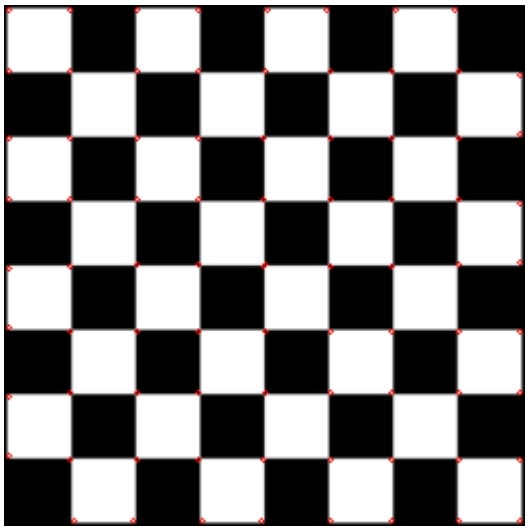
Part1: Harris corner detector

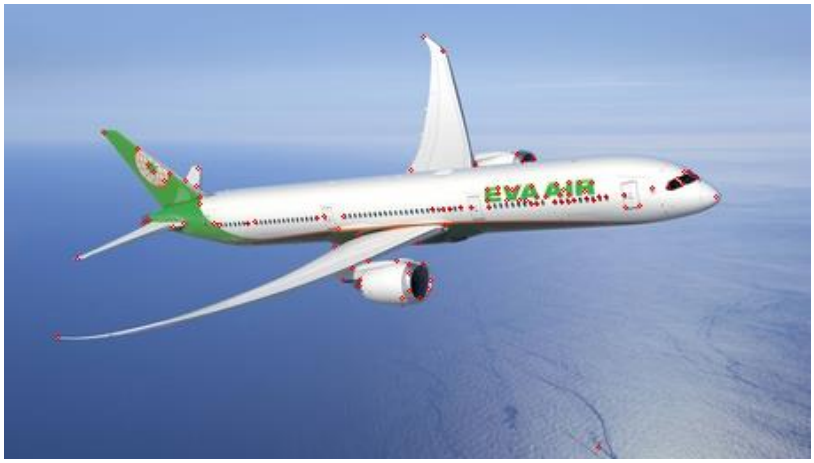
- **Visualize the detected corner for 1.png, 2.png, 3.png:**

threshold use default value (100.)

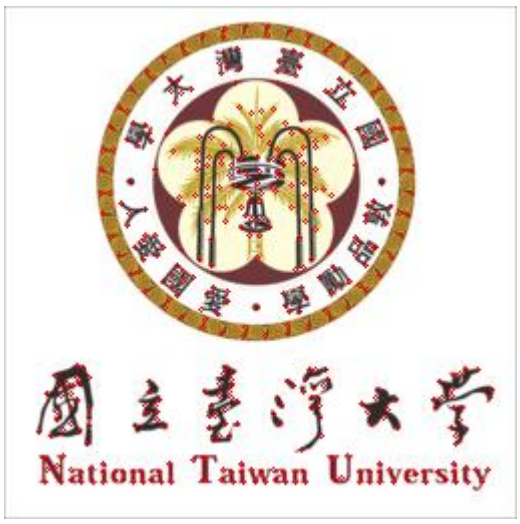
1.png



2.png



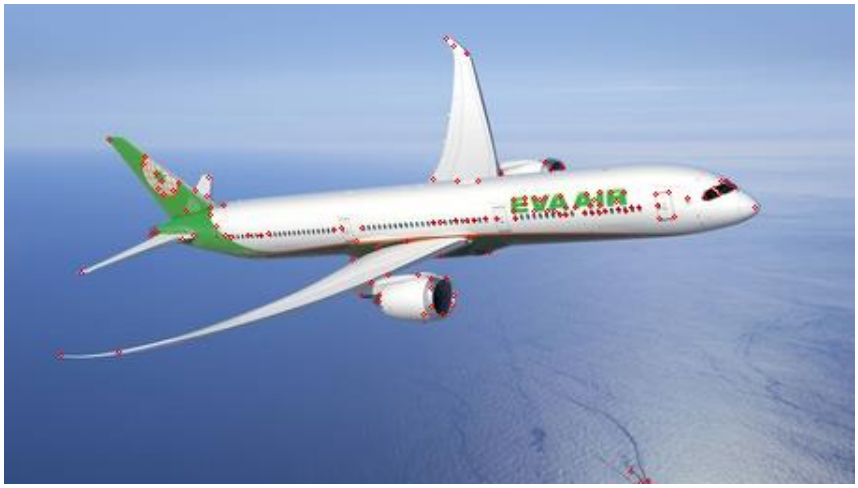
3.png



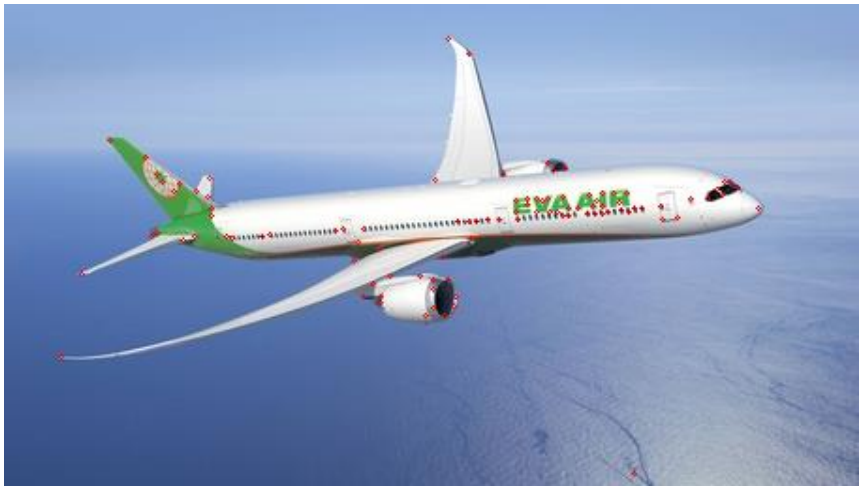
• Use three thresholds (25, 50, 100) on 2.png and describe the difference:

thresholds 25

thresholds 50



thresholds 100



Threshold 設越大時會有越少的點被選為 candidates，因此得出來的 corner 數量會較少。

Part2: Joint bilateral filter

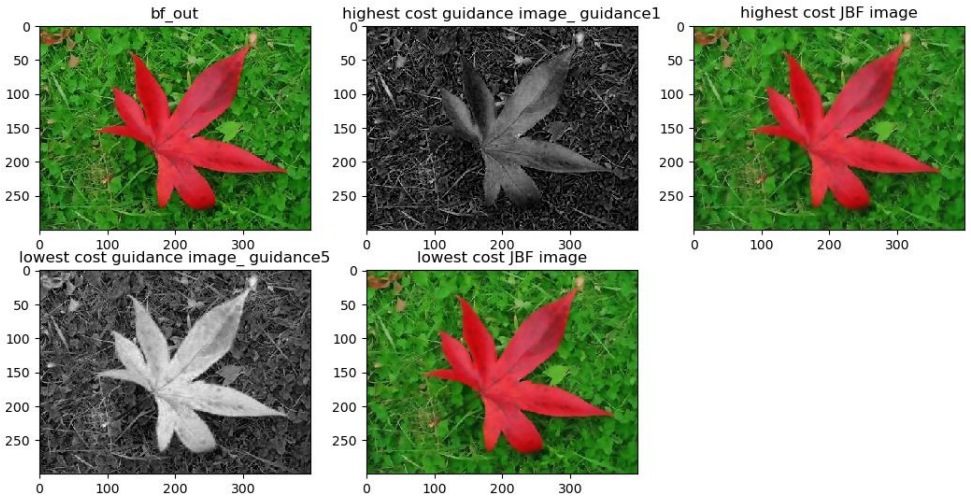
• For 1.png :

• Report the cost for each filtered image (by using 6 grayscale images as guidance)

由左至右分別為 guidance1~guidance5 及 img_gray(original cv2 gray conversions) :

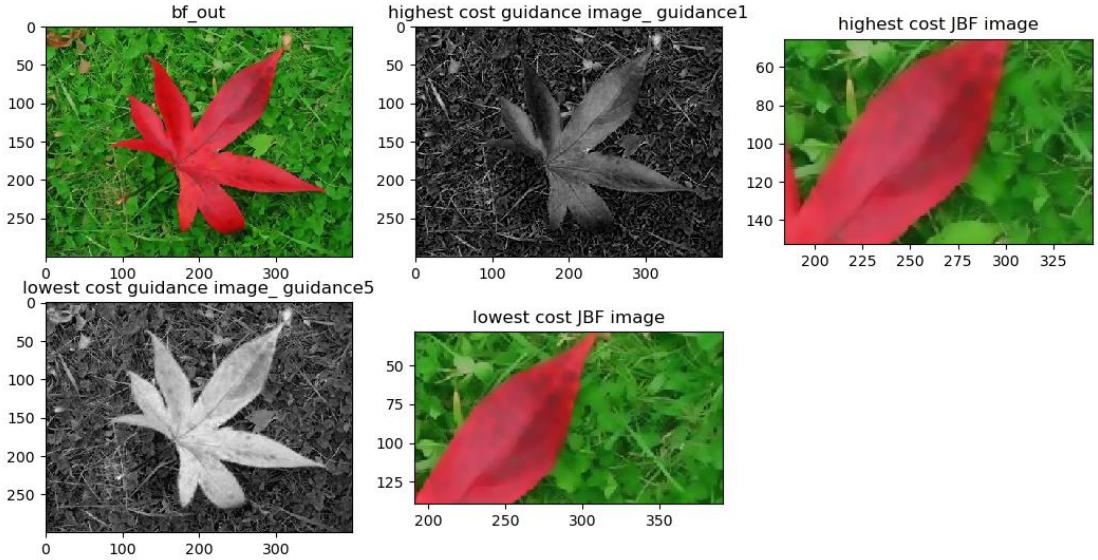
L1_norm = [1439568, 1305961, 1393620, 1279697, 1127913, 1207799]

• Show original RGB image / two filtered RGB images and two grayscale images with highest and lowest cost (five images in total for each input image)

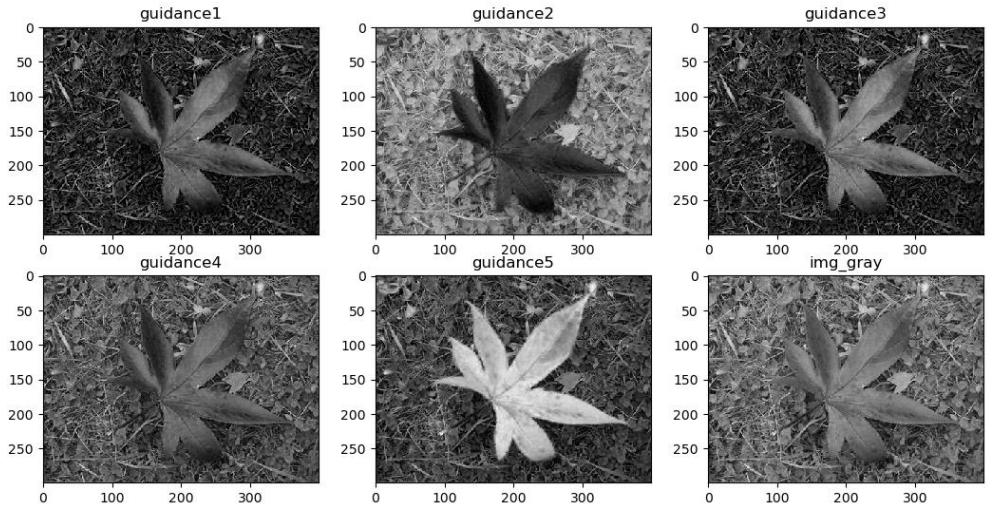


• Describe the difference between those two grayscale images

放大上圖 JBF 結果圖，可看到 lost 值最小的 guidance5 邊緣較明顯，因此做為 guidance 最能保有邊緣的部分。由 grayscale images 也可看到 guidance1 圖片



的主要物件與背景顏色差不多，無法有效區分主物件與背景(邊緣位置)。



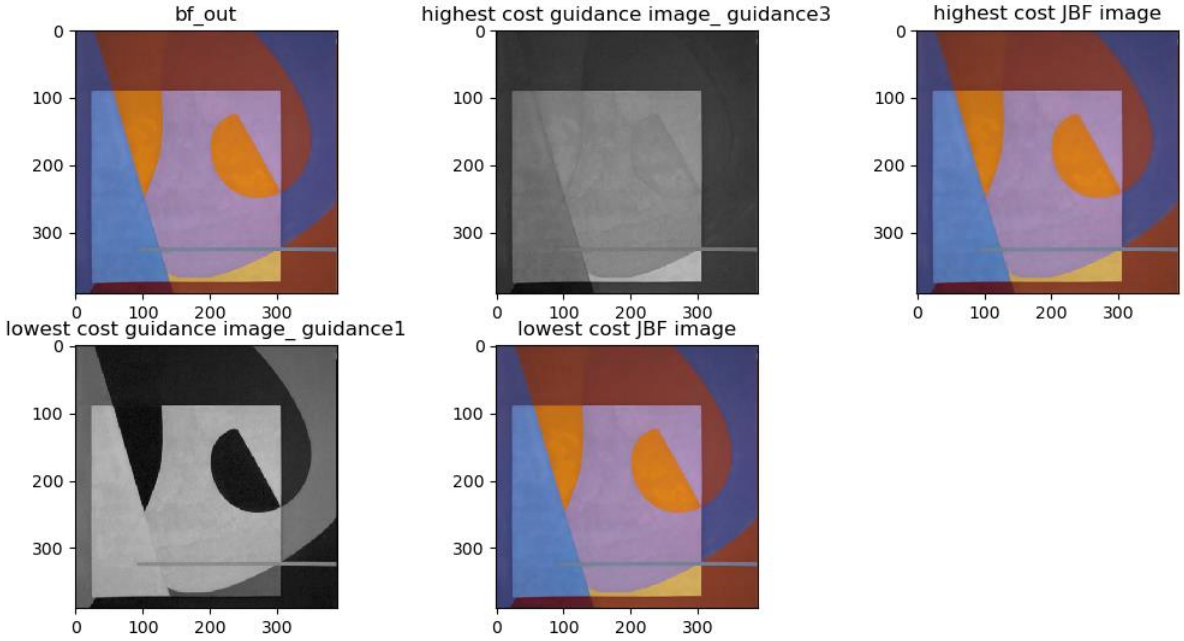
• For 2.png :

• Report the cost for each filtered image (by using 6 grayscale images as guidance)

由左至右分別為 guidance1~guidance5 及 img_gray :

L1_norm = [77882, 86023, 188019, 128341, 110862, 183850]

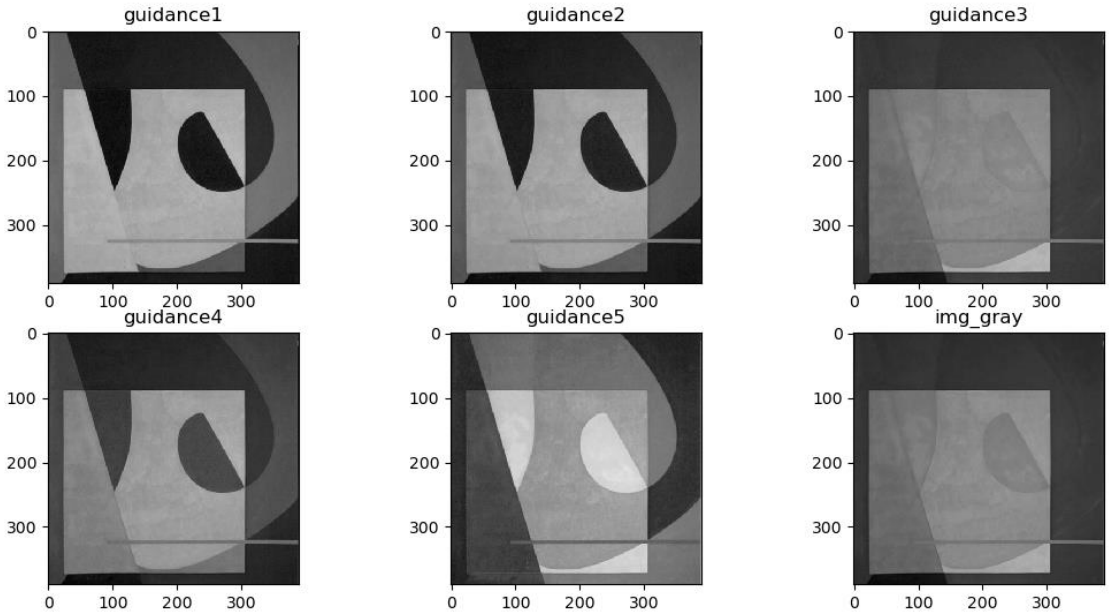
• Show original RGB image / two filtered RGB images and two grayscale images with highest and lowest cost (five images in total for each input image)



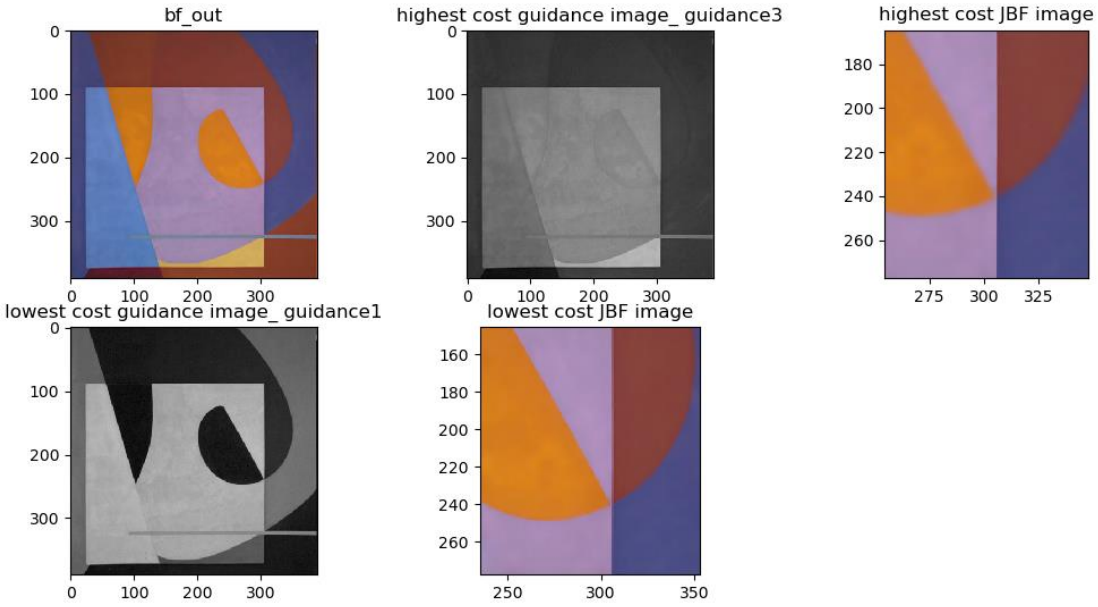
• Describe the difference between those two grayscale images

程式跑出來的結果如上圖，可見 guidance3 是 L1 cost 值最高，guidance1 是 L1 cost 值最低的，印出所有 1~6(img_gray)來比較，也可看到 guidance3 是邊緣最不明顯的 grayscale images，而相對的 guidance1 是所有邊緣最明顯者，所以 cost 值合理最小。

此外還可以觀察到的 guidance1 與 guidance2 的 cost 值相近(77882, 86023)，對照原圖兩者確實相近且邊緣都較明顯，同樣的 guidance3 與 img_gray 最像，也可由 L1 cost 值看到此情形。(188019, 183850)。



由下圖也可看到 lost 值大的 guidance 形成的 JBF image 邊緣較銳利。(尤其橘色半圓處)



• **Describe how you speed up the implementation of bilateral filter**

1.由於 exponential 運算較花時間，因此在運算 range kernel 時，以 look up table 查對應的值，同樣的因 spatial kernel 不受 pixel value 影響，僅與距 center 的距離有關，因此也可先算好，避免在計算每個 output pixel value 時在 for-loop 裡運算。

2.使用 numpy 平行運算的特點，讓矩陣運算更快，減少 for-loop 的使用。

```
(如: padded_guidance[(row - r):(row + r + 1), (col - r):(col + r + 1)] - padded_guidance[row, col]))一次框出一塊範圍做運算。
```

3.原本還有嘗試 cython 的方式，使用 cython 語法，將 JBF.py 編譯成.pyx，藉由變數及函式的事先宣告，減少會拖慢程式執行的靜態變數型別，不過實驗完發現效果沒有很顯著，因此後來就沒放了。