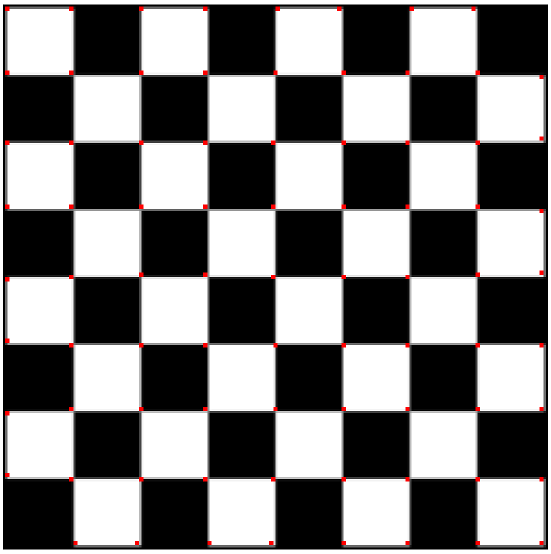
Homework #1

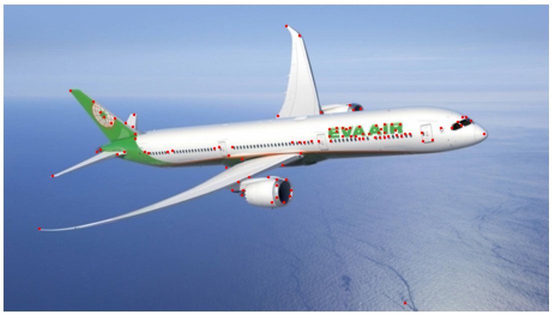
April 1, 2021

Part1: Harris corner detector

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



(a) 1.png

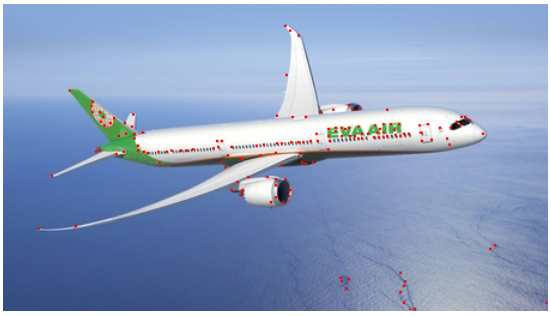


(b) 2.png

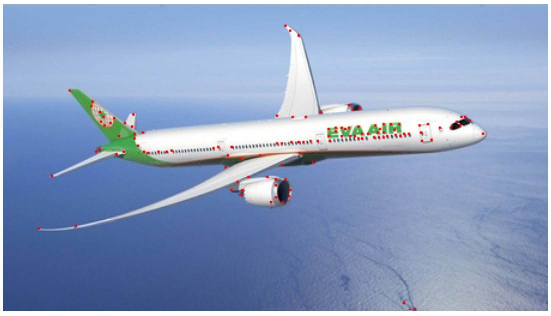
(c) 3.png

Figure 1

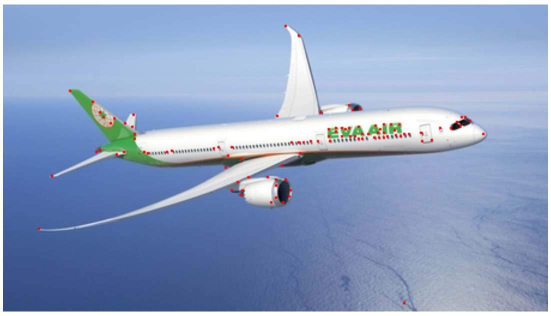
- Use three thresholds (25, 50, 100) on 2.png and describe the difference



(a) threshold=25



(b) threshold=50



(c) threshold=100

Figure 2

threshold 越小取的點越多，可以看到 Figure 2(a) 在海上也偵測到許多多餘的點，但對比 threshold 取 50 和 100 並沒有太大的區別，另外 detection point 的數量分別為 151, 118, 99 (threshold = 25, 50, 100)。

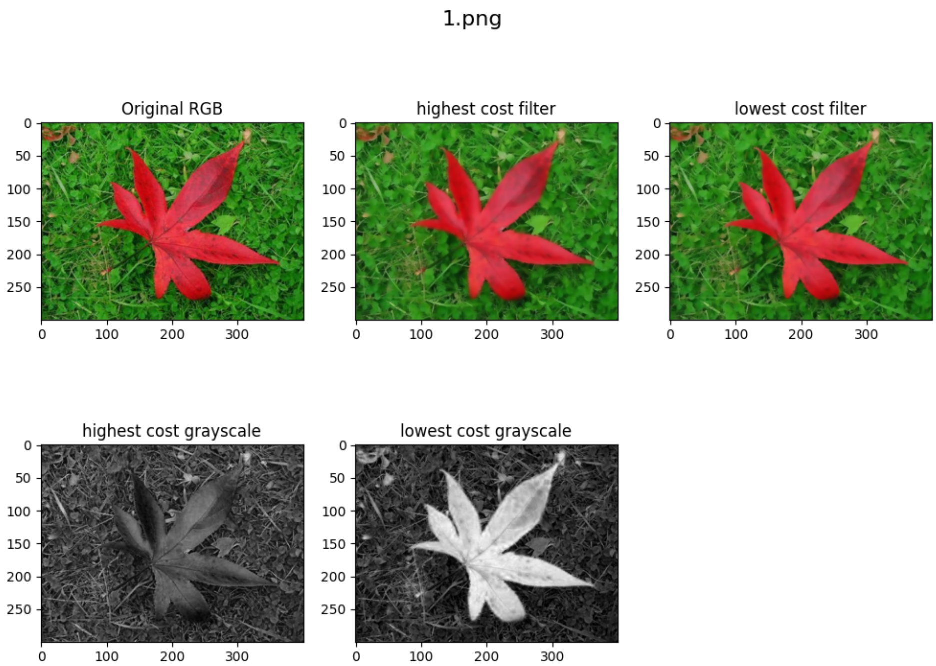
Part2: Joint bilateral filter

● 1.png

Cost

BGR2GRAY	(R, G, B)				
	(0.0, 0.0, 1.0)	(0.0, 1.0, 0.0)	(0.1,0.0,0.9)	(0.1,0.4,0.5)	(0.8,0.2,0.0)
1207799	1151211	1305961	1161246	1160306	1365893

Original RGB image / two filtered RGB images and two grayscale images



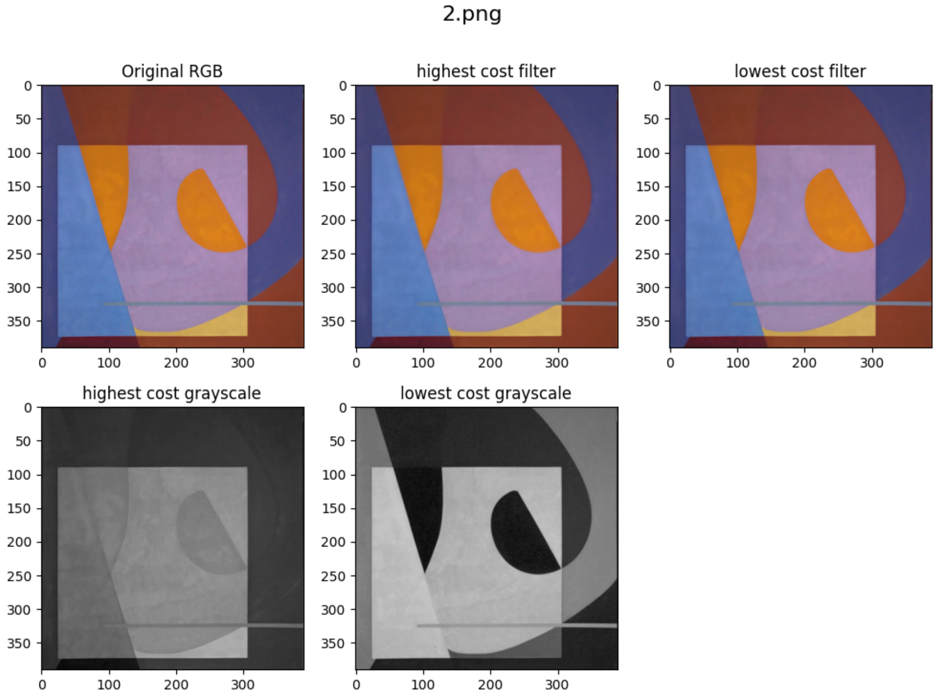
highest cost image 可以看到葉子的地方和地面的顏色接近一樣，因此在做 joint bilateral filter 時葉片周圍會叫模糊，而 lowest cost image 顏色差異較大，因此與用原圖做 bilateral filter 較相近。

● 2.png

Cost

BGR2GRAY	(R, G, B)				
	(0.1,0.0,0.9)	(0.2,0.0,0.8)	(0.2,0.8,0.0)	(0.4,0.0,0.6)	(1.0,0.0,0.0)
183851	129350	158077	137949	163917	71001

Original RGB image / two filtered RGB images and two grayscale images



highest cost image 在小正方形內顏色差距不明顯，lowest cost image 則反之。

● how I speed up the implementation of bilateral filter

- 因為 spatial kernel 對於每個點的值都一樣因此先計算 Gs
- 使用 unrolled inner product 做 convolution 的運算
- 計算 range kernel 時，把要使用的 guidance 區域找用索引區段 (begin: end) 找出來去計算，會比起用 for loop 一個點一個點算要快
- 矩陣乘法先算 Gs * Gr
- O(window size * window size) for compute Gs, O(img size) for compute Gr and U(img)

Algorithm 1: joint bilateral filter

Input: img, guidance, wndw_size, pad_w, sigma_s, sigma_r

Output: joint bilateral filter image

Calculate Gs(i, j) matrix and reshape to (wndw_size × wndw_size);

Gr is (img_size[0] × img_size[1], wndw_size × wndw_size) matrix;

U_img is (img_size[0] × img_size[1], wndw_size × wndw_size, 3) matrix;

i := 0;

for *i++ < img_size[0]* × *img_size[1]* **do**

  x = i // img_size[1];

  y = i % img_size[1];

  Gr[i] = exp(-sum((guidance[x - pad_w: x + pad_w, y - pad_w: y + pad_w] - guidance[x, y])) ** 2 / 2 / (sigma_r ** 2));

  U_img[i] = img[x - pad_w: x + pad_w, y - pad_w: y + pad_w].reshape(-1);

end

output = U_img * (Gr * Gs) / |Gr * Gs| and reshape to image size;

return output;