SURF (Speeded Up Robust Features)
KEY POINT DETECTOR

--Implement with Python

Env :
python=3.10.9
opencv-python=4.9.0.80

410978049 統計四(STAT) 江泓志

Processing steps

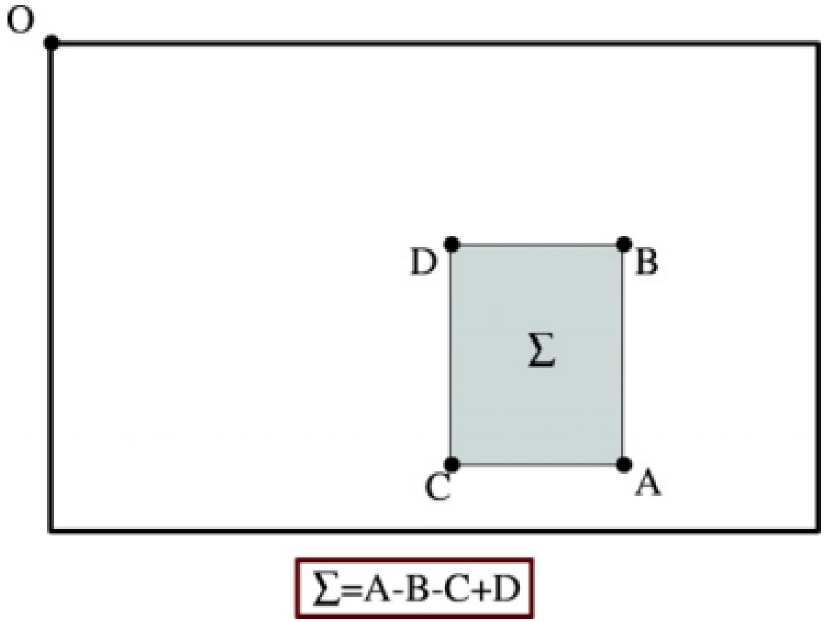
1. Integral Image
2. Hessian Matrix
3. Scale Space
4. Interest Point
5. Orientation
6. Descriptor
7. Matching

1. Integral Image

We can construct an integral image using the definition provided in the Ch3 PPT.

$$I(x,y)=f(x,y)+I(x,y-1)+I(x-1,y)-I(x-1,y-1)$$

```
def construct_image_integral(self):
    # I(x, y) = sum_i^x sum_j^y f(i, j)
    self.image_integral = self.image_broadcast.copy()

    for i in range(self.image_broadcast.shape[0]) :
        for j in range(self.image_broadcast.shape[1]) :
            self.image_integral[i, j] = self.image_broadcast[i, j] \
            + (self.image_integral[i, j-1] if j-1 > -1 else 0) \
            + (self.image_integral[i-1, j] if i-1 > -1 else 0) \
            - (self.image_integral[i-1, j-1] if i-1 > -1 and j-1 > -1 else 0)
```

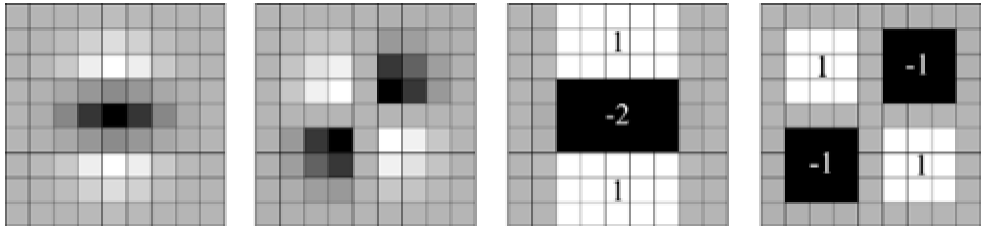


We can also use this formula. $I(x,y)=\sum_{i=0}^{x}\sum_{j=0}^{y} f(i,j)$

```
self.image_integral = np.cumsum(np.cumsum(self.image_broadcast, axis=0), axis=1)
```

2. Hessian Matrix

One of the main contribution of the paper



At first, we have the formula for the Hessian

$$\mathcal{H}(\mathbf{x},\sigma)=\begin{bmatrix} L_{xx}(\mathbf{x},\sigma) & L_{xy}(\mathbf{x},\sigma) \\ L_{xy}(\mathbf{x},\sigma) & L_{yy}(\mathbf{x},\sigma) \end{bmatrix}$$

After approximating with a box filter, we can obtain this formula

$$\begin{aligned} Det(H_{approx}) &= L_{xx}L_{yy} - L_{xx}Lxy \\ &= D_{xx}\frac{L_{xx}}{D_{xx}}D_{yy}\frac{L_{yy}}{D_{yy}} - D_{xy}\frac{L_{xy}}{D_{xy}}D_{xy}\frac{L_{xy}}{D_{xy}} \\ &= D_{xx}D_{xy} - (wD_{xy})^2 \end{aligned}$$

where $w=\frac{|L_{xy}(1.2)|_F\,|D_{yy}(9)|_F}{|L_{yy}(1.2)|_F\,|D_{xy}(9)|_F}=0.912...\simeq 0.9$

finally $\det(\mathcal{H}_{\text{approx}}) = D_{xx}D_{yy} - (0.9D_{xy})^2$

```
# 算高斯 (以Box Filter來逼近)
Dxx = self.compute_Dxx(L, x, y)
Dyy = self.compute_Dyy(L, x, y)
Dxy = self.compute_Dxy(L, x, y)

# 其approximate的DoH (Determinant of Hessian)
DoH = normalizer * (Dxx * Dyy - ((w * Dxy) ** 2))
DoH_L[y][x] = DoH

lapSign_L[y][x] = 1 if (Dxx + Dyy) > 0 else -1
```

Laplacian Sign for matching

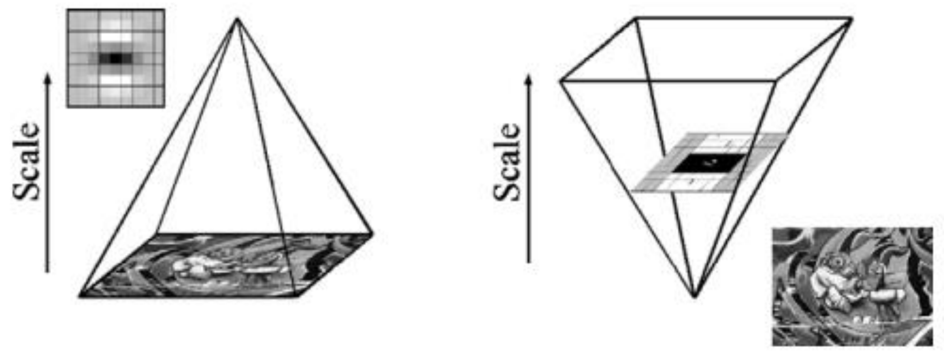
3. Scale Space

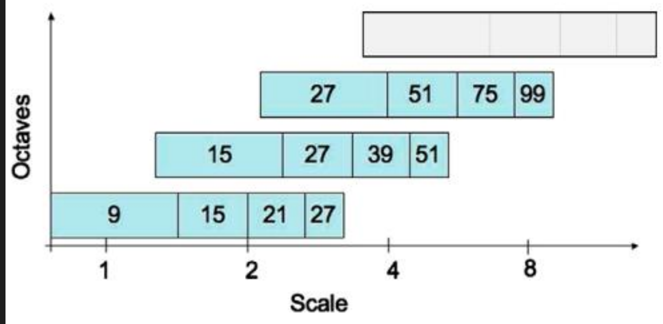
I compute all the parameters about L in the class constructor

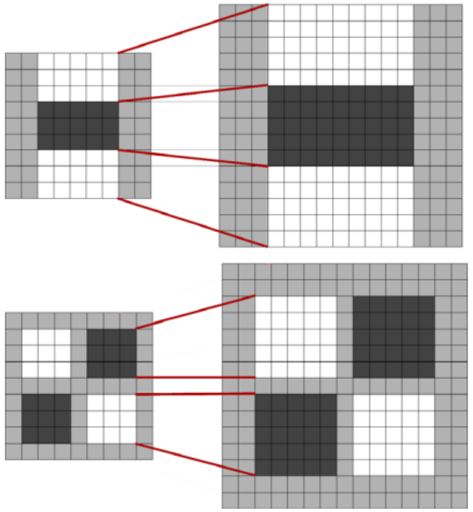
```python
def __init__(self, octaves=3, layers=4, threshold=1000):
    self.octaves = octaves  # 幾組
    self.layers = layers # 金字塔高度
    self.threshold = threshold # for Hessian
    self.params_related_to_L = {}
    self.octaves_layers_L = {}

    # 這裡的L其實都是論文的l0 (L/3) 以下論文的L用L_P代替
    for o in range(1, self.octaves + 1):
        self.octaves_layers_L[o] = {}
        for i in range(1, self.layers + 1):
            L = (2 ** o) * i + 1 # 2, 4, 8 ... 來算每一組的尺寸(l0)
            self.octaves_layers_L[o][i] = L
            # 每個L所對應的一些參數
            # w就不算了直接當0.9
            self.params_related_to_L[L] = {
                'sigma_L': round(0.4 * L, 2), # sigma = 1.2 * L_P/9 = 0.4 * l0
                'L': L, # l0
                'l': np.int(0.8 * L) # descip sigma
            }
```

Compute Hessian(with Box-Filter) for every L

Another the main contribution of the paper







Compare the number with the paper

```
>>> testSURF.octaves_layers_L # 每層的l0長度
{1: {1: 3, 2: 5, 3: 7, 4: 9}, 2: {1: 5, 2: 9, 3: 13, 4: 17}, 3: {1: 9, 2: 17, 3: 25, 4: 33}}
```

4. Interest Point

First, we determine whether it is higher than the threshold

```
if DoH > self.threshold:
    if self.local_extrema_DoH_bool(L_index, x, y, DoH):
```

Then we compute whether it is a local extrema(max) in the neighborhood scale

```
max_value = np.amax(self.DoH[lL:L + 2, ly:y + 2, lx:x + 2])
```

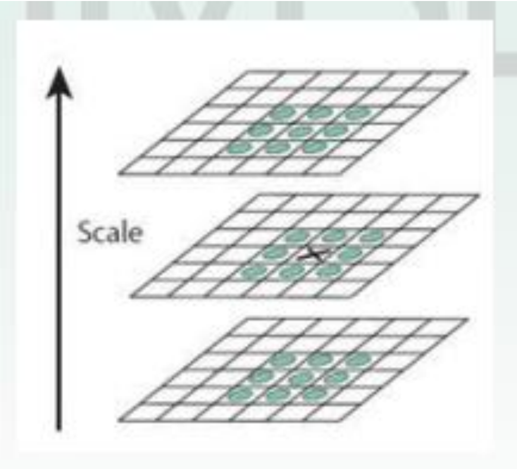


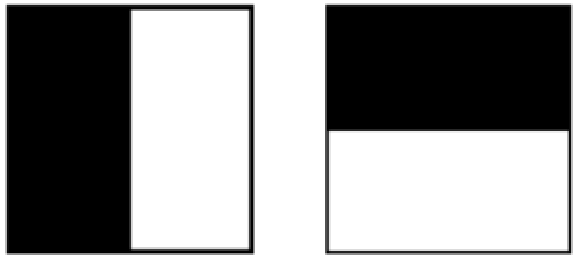
Figure in SIFT

5. Orientation

$$m_w = \sum_w dx + \sum_w dy$$
$$\theta = arctan(\sum_w dy / \sum_w dx)$$

We can compute the Haar wavelet responses around the key point

```
DxL = self.compute_Dx_Haar(l, x, y)
DyL = self.compute_Dy_Haar(l, x, y)
G = G_list.get(10 * i + j) / G_sum
image_value = self.get_image_value(x, y)

fi = np.array([[
    np.arctan2(DyL, DxL),
    DxL * image_value * G,
    DyL * image_value * G,
]])
```



Haar Wavelet Filters
x(left) y(right)

Then, we slide the orientation window to find the orientation of interest point (with longest vector)

```
orientation = np.arctan2(list_theta[max_theta_norm_index][3], list_theta[max_theta_norm_index][2])
```

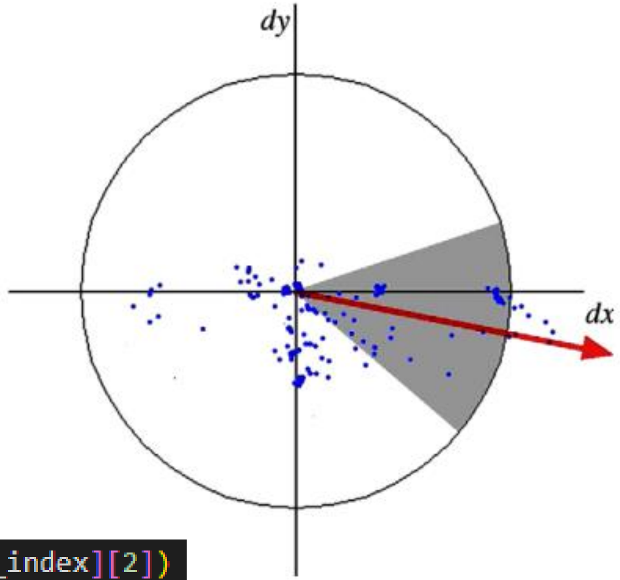


Fig. 10. Orientation assignment: a sliding orientation window of size $\frac{\pi}{3}$ detects the dominant orientation of the Gaussian weighted Haar wavelet responses at every sample point within a circular neighbourhood around the interest point.

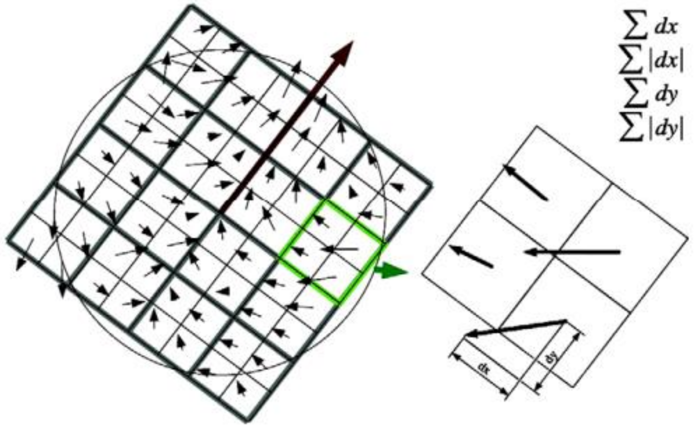
6. Descriptor

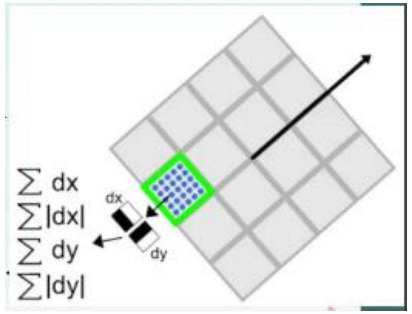
The descriptor's form is as follows, as described in the paper

$$\mathbf{v} = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|)$$

First, we rotate the axis to the orientation and compute the Haar responses

```
x, y = sigma * (R @ np.array([u, v])) + np.array([x0, y0])
x = np.int(x)
y = np.int(y)

try:
    DxL = self.compute_Dx_Haar(Dl, x, y)
    DyL = self.compute_Dy_Haar(Dl, x, y)
```





Then, We can calculate the descriptors

```
dx_uv, dy_uv = (R_minus @ np.array([DxL, DyL])) * G
```

```
        sum_dx_uv += dx_uv
        sum_dy_uv += dy_uv
        sum_dx_uv_abs += np.abs(dx_uv)
        sum_dy_uv_abs += np.abs(dy_uv)

mk[i, j, 0] += sum_dx_uv
mk[i, j, 1] += sum_dy_uv
mk[i, j, 2] += sum_dx_uv_abs
mk[i, j, 3] += sum_dy_uv_abs
```

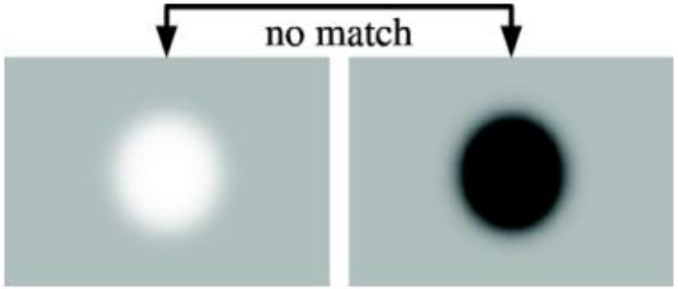
There are 4*4 = 16 sample blocks, each containing 4 sample blocks.
Thus, the descriptor has a dimension of 64 (16 blocks * 4)

7. Matching

We compute all the descriptors of keypoints in the two images and record the distances

```python
for desc_vector1, theta1, SignLaplassian1 in descriptors1:
    lengths_to_point = np.ndarray((len(descriptors2),))
    v2_index = 0

    for desc_vector2, theta2, SignLaplassian2 in descriptors2:
        dist = np.inf

        if (SignLaplassian1 == SignLaplassian2):
            # 兩vector的距離
            dist = np.linalg.norm(desc_vector1 - desc_vector2)

        lengths_to_point[v2_index] = dist
```

Note that we don't perform matching when the sign of the Laplacian is different



Finally, we can collect the keypoints that are most likely to match in the images

```python
indxs_sorted = np.argpartition(lengths_to_point, 2) # 得到距離最短兩個的index
dist1 = lengths_to_point[indxs_sorted[0]] # min
dist2 = lengths_to_point[indxs_sorted[1]] # second min

if dist2 == 0 or dist1 / dist2 <= threshold: # 第一個足夠的小
    match = cv2.DMatch(
        _distance = dist1, # 距離
        _queryIdx = v1_index, # 在原始的idx
        _trainIdx = indxs_sorted[0] # 在新的比較圖的idx
    )

    matches.append(match)
```

RESULT

SURF

Original Image

SURF keypoint detect
(threshold = 1000)

```
========== SURF START ==========

Image name : lenna.png
Input image size : (330, 330, 3)
Gray scale image size : (330, 330)

==========    INIT    ==========

SURF detector create successfully...
SURF detector initializing...
init running...
Time : 7.68 seconds

==========  KEYPOINT  ==========

SURF detector detecting...
detectAndCompute running...
Time : 16.85 seconds
Finish detecting...
Keypoints len: 1185

==========  DRAWING   ==========

Drawing keypoint plot...

==========   FINISH   ==========
```

RESULT

SURF

Original Image

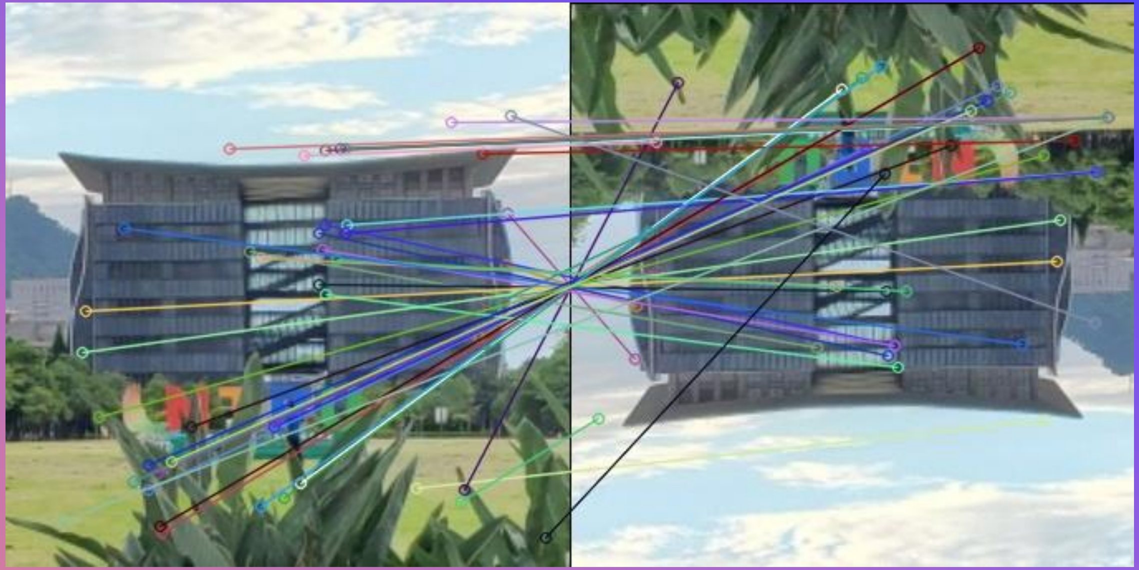
SURF keypoint detect
(threshold = 1000)

```
========== SURF START ==========

Image name : ntpu.jpg
Input image size : (341, 341, 3)
Gray scale image size : (341, 341)

==========    INIT    ==========

SURF detector create successfully...
SURF detector initializing...
init running...
Time : 8.46 seconds

==========  KEYPOINT  ==========

SURF detector detecting...
detectAndCompute running...
Time : 13.32 seconds
Finish detecting...
Keypoints len: 946

==========  DRAWING   ==========

Drawing keypoint plot...

==========   FINISH   ==========
```

MATCH(INVARIANT)

SURF

Scale Invariant

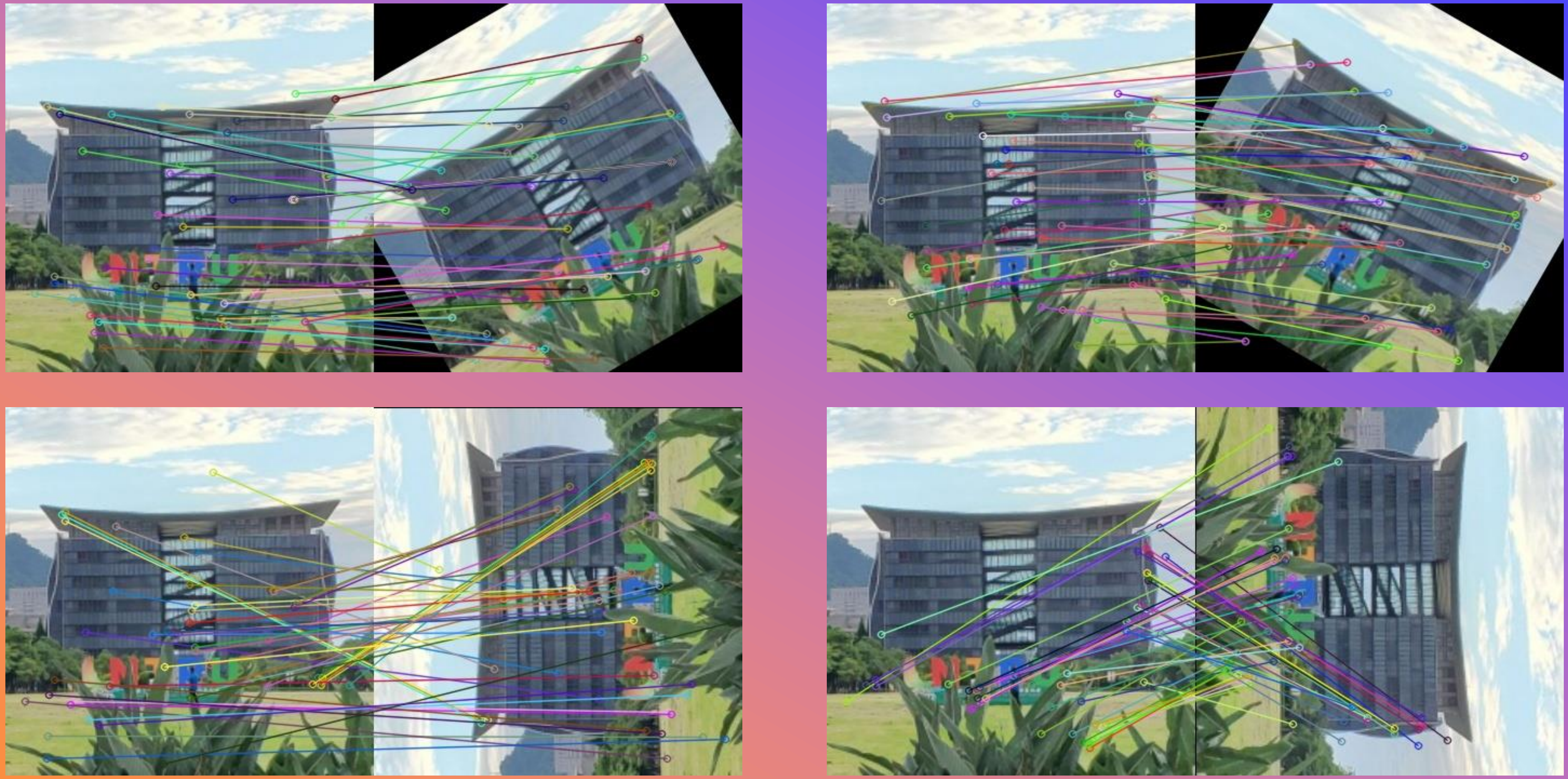


Rotation Invariant

We choose the top 50 closest keypoints to show in the comparison image

MATCH(INVARIANT)

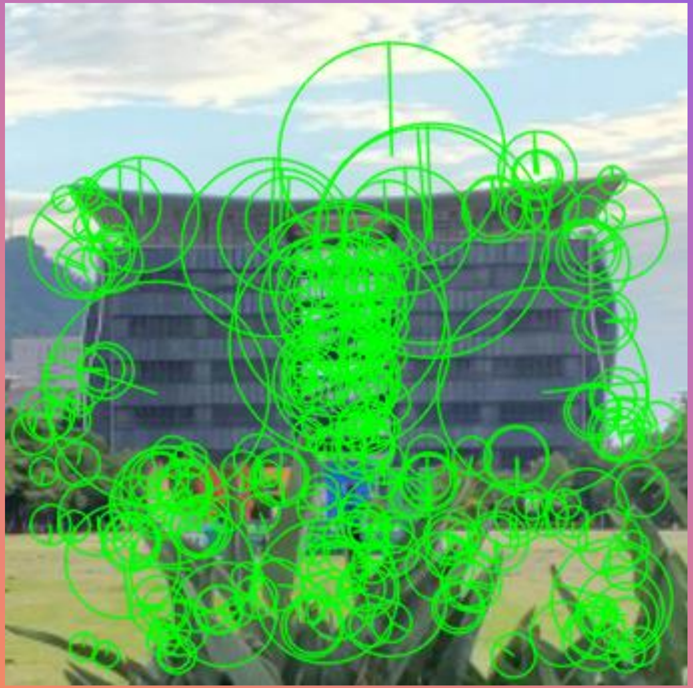
SURF



Rotation Invariant

Compare to opencv xfeatures2d.SURF

Env :
python=3.6.13
opencv-python=3.4.2.16
opencv-contrib-python=3.4.2.16



OPENCV SURF

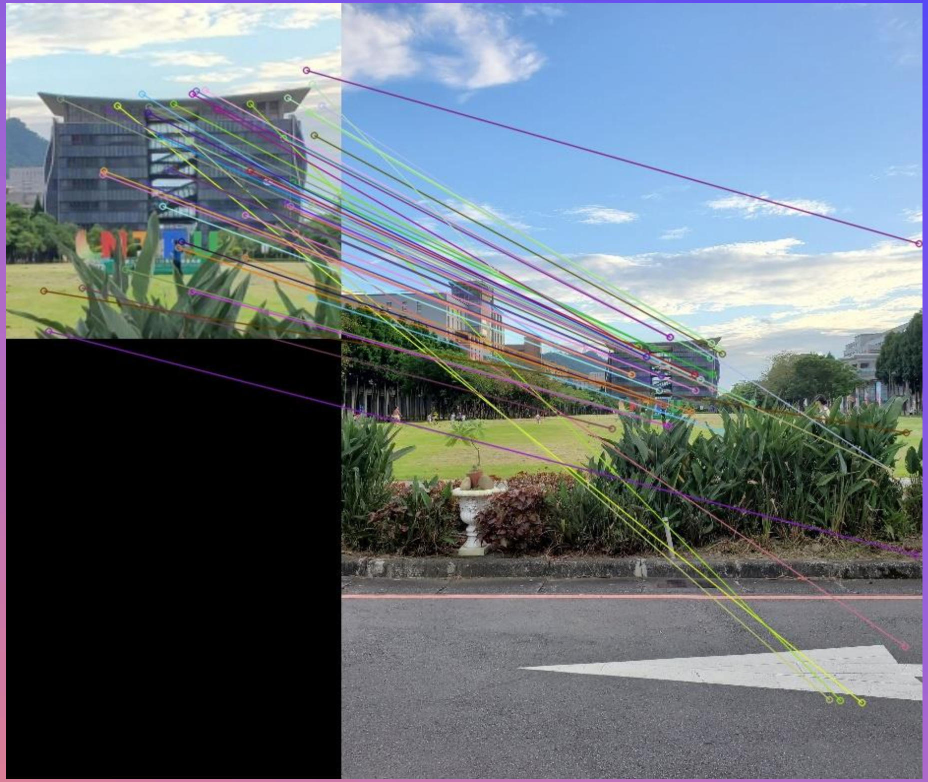
OUR SURF

Both octave=3, layers=4, threshold=1000

Compare to opencv xfeatures2d.SURF

Env :
python=3.6.13
opencv-python=3.4.2.16
opencv-contrib-python=3.4.2.16

OPENCV SURF

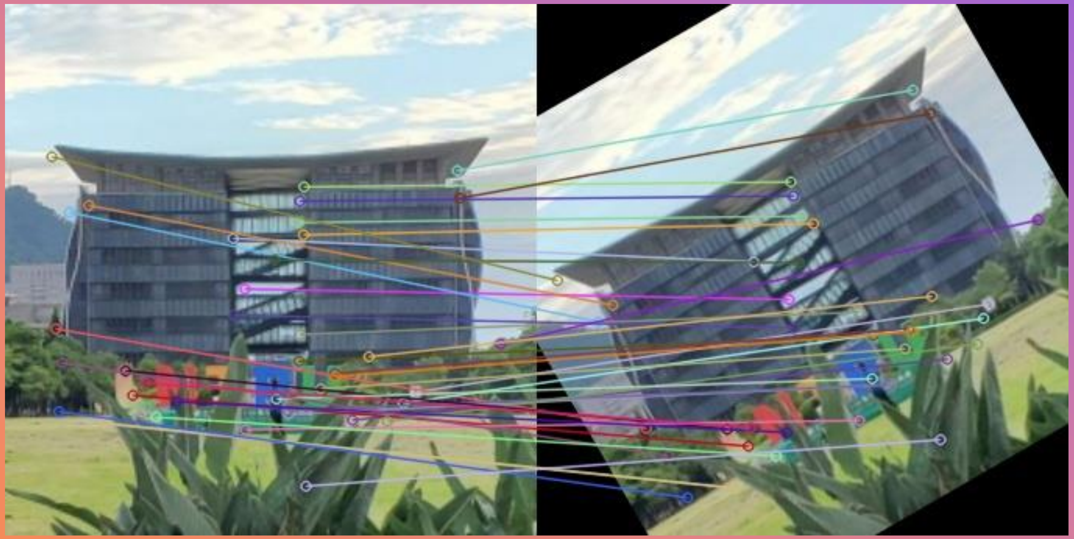


OUR SURF

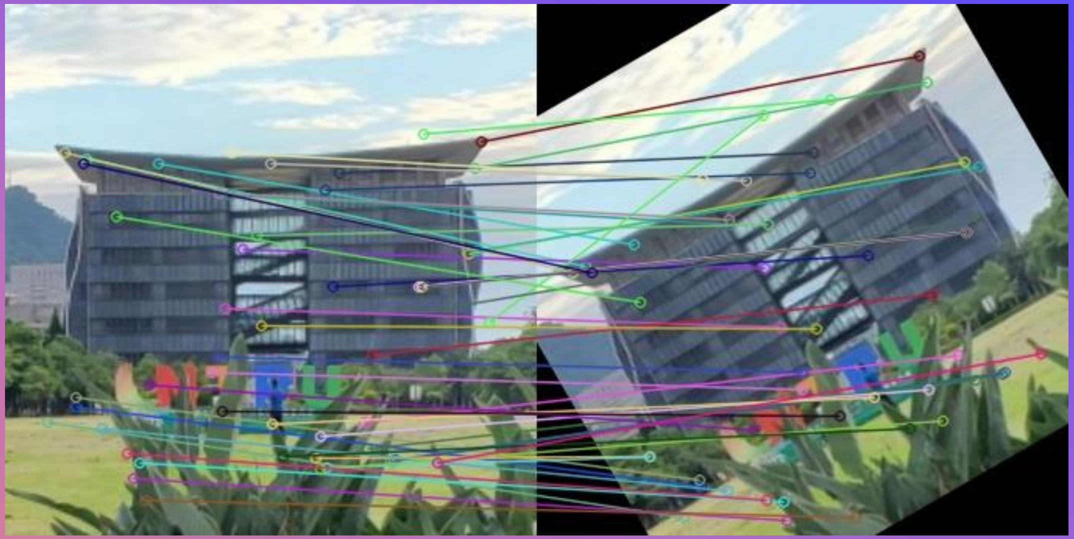
Both octave=3, layers=4, threshold=1000

Compare to opencv xfeatures2d.SURF

Env :
python=3.6.13
opencv-python=3.4.2.16
opencv-contrib-python=3.4.2.16



OPENCV SURF



OUR SURF

Both octave=3, layers=4, threshold=1000

References

[1] **Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool, Speeded-Up Robust Features (SURF), Computer Vision and Image Understanding, Volume 110, Issue 3, 2008.**

[2] **Introduction to SURF (Speeded-Up Robust Features)** (2019) https://medium.com/@deepanshut041/introduction-to-surf-speeded-up-robust-features-c7396d6e7c4e

[3] **[基础知识] Speeded Up Robust Features （SURF特征）** (2021) https://zhuanlan.zhihu.com/p/365403867

[4] **SURF(Speeded Up Robust Features)算法原理** (2017) https://blog.csdn.net/shenziheng1/article/details/72579635

[5] **Linear Algebra Course project** (2023) https://github.com/germes/ds-surf/tree/main

SURF

THANK YOU!!