Assignment 3

Tong Zhen 120090694

Part I Written Exercise

Q1:

Show that forming unweighted local averages, which yields an operation of the form

$$\mathcal{R}_{ij} = \frac{1}{(2k+1)^2} \sum_{u=i-k}^{u=i+k} \sum_{v=j-k}^{v=j+k} \mathcal{F}_{uv}$$

is a convolution. What is the kernel of this convolution?

A1:

$$R_{ij} = \sum_{u=i-k}^{u=i+k} \sum_{v=j-k}^{v=j+k} H_{i-u,j-v} \cdot F_{uv}.$$

H is a kernel. with size = $(2k+1) \times (2k+1)$.

each value in it $H_{ij} = \frac{1}{(2k+1)^2}$

Do linear filter for each pixal in input F. call F_{ij} calculate the average of a $(2k+1) \times (2k+1)$ centered in $[i,j]$ as output R_{ij}.

Q2:

Each pixel value in 500×500 pixel image I is an independent, normally distributed random variable with zero mean and standard deviation one. I is convolved with the (2k + 1) × (2k + 1) kernel G. What is the covariance of pixel values in the result? There are two ways to do this; on a case-by-case basis (e.g., at points that are greater than 2k + 1 apart in either the x or y direction, the values are clearly independent) or in one fell swoop. Don't worry about the pixel values at the boundary.

A2:

The pixel in the convoluted picture is a weighted sum of independent variables.

Two pixel share some random variable in their sum

Say one is $R_{ij} = \sum_{lm} G_{lm} I_{i-l, j-m}$.

The other is $R_{uv} = \sum_{st} G_{st} I_{u-s, v-t}$

They share pixel when $i-l = u-s$ and $j-m = v-t$

The covariance of them is the weights where these shared terms appear

$$\mathbb{E}(R_{ij}, R_{uv}) = \sum_{i-l=u-s, j-m=v-t} G_{lm} G_{st}$$

Part II Programming Exercises

Problem 1

Image Pyramid (20 points)

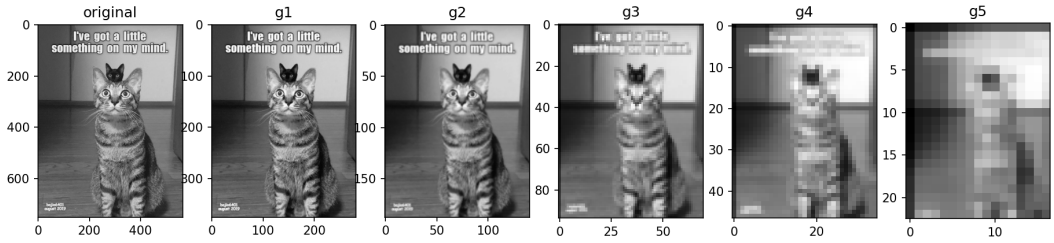
(a)

Display a Gaussian and Laplacian pyramid of level 5 (using your code). It should be formatted similar to the figure below.

Parameter	Value
Kernel size	11x11
Gaussian Sigma	0.8

First, do Gaussian Convolution to the image and down sample, here use nearest neighbor down sample. Iterate for 5 times.

Gaussian Pyramid

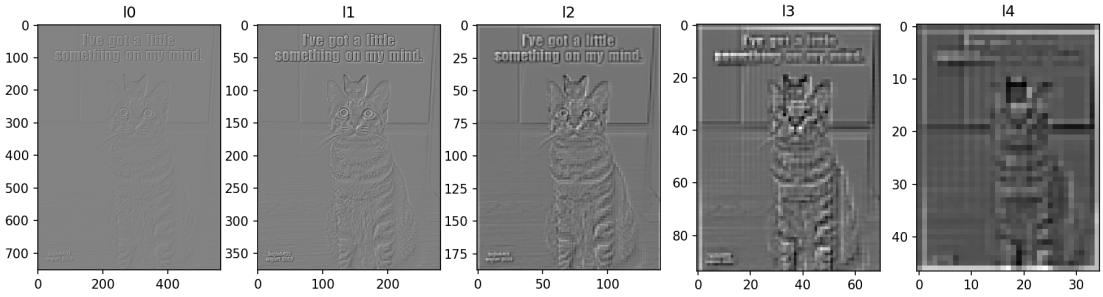


Do the up sample for g5, and get

$$e_4 = expand(g_5)$$
$$l_4 = g_4 - e_4$$

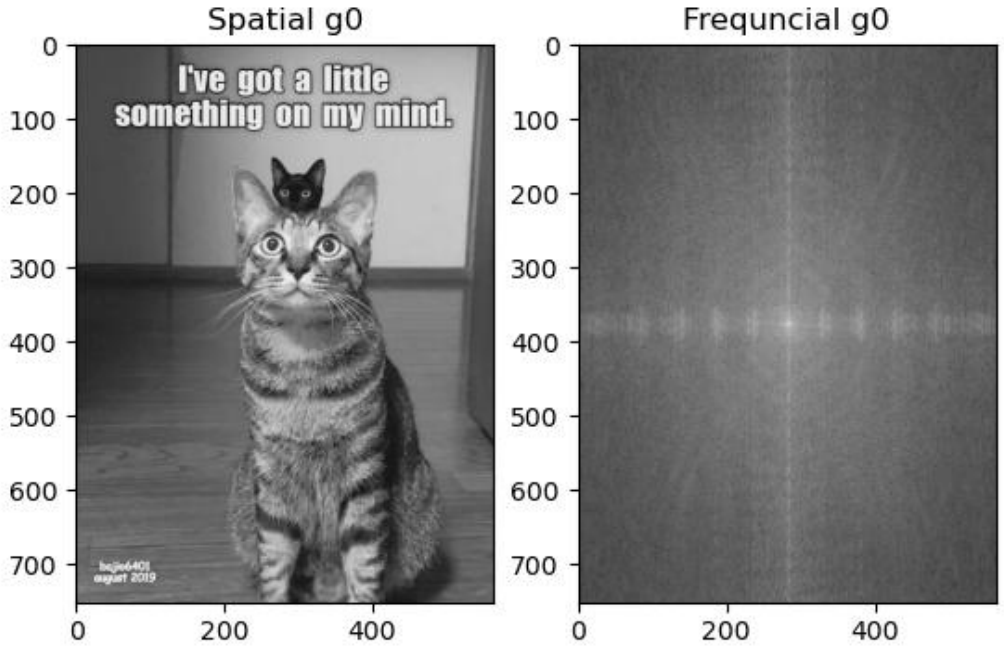
Iterate Laplacian until we get l0

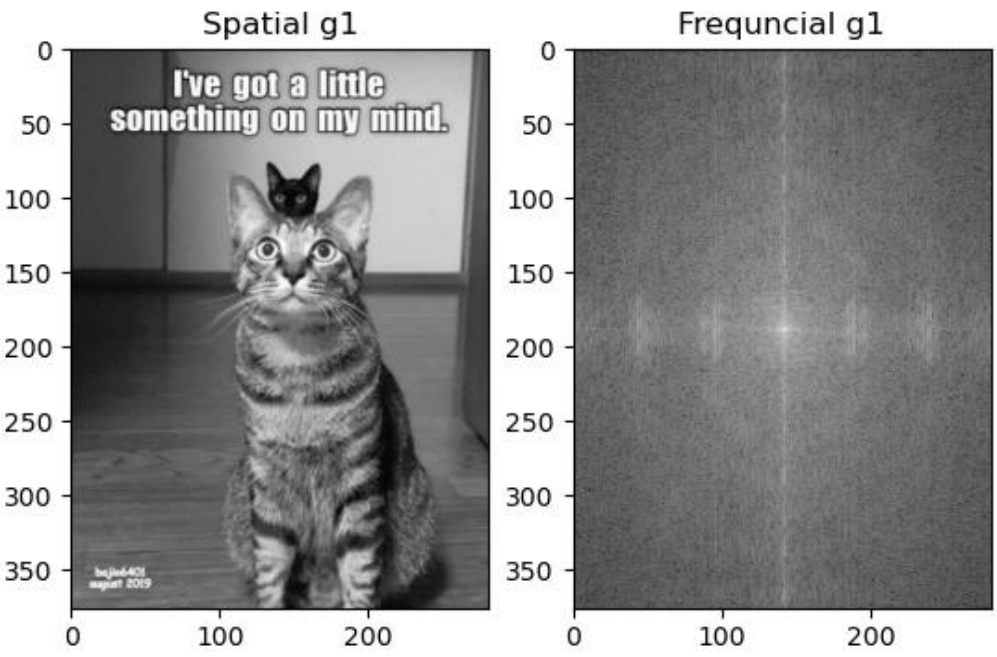
Laplacian Pyramid

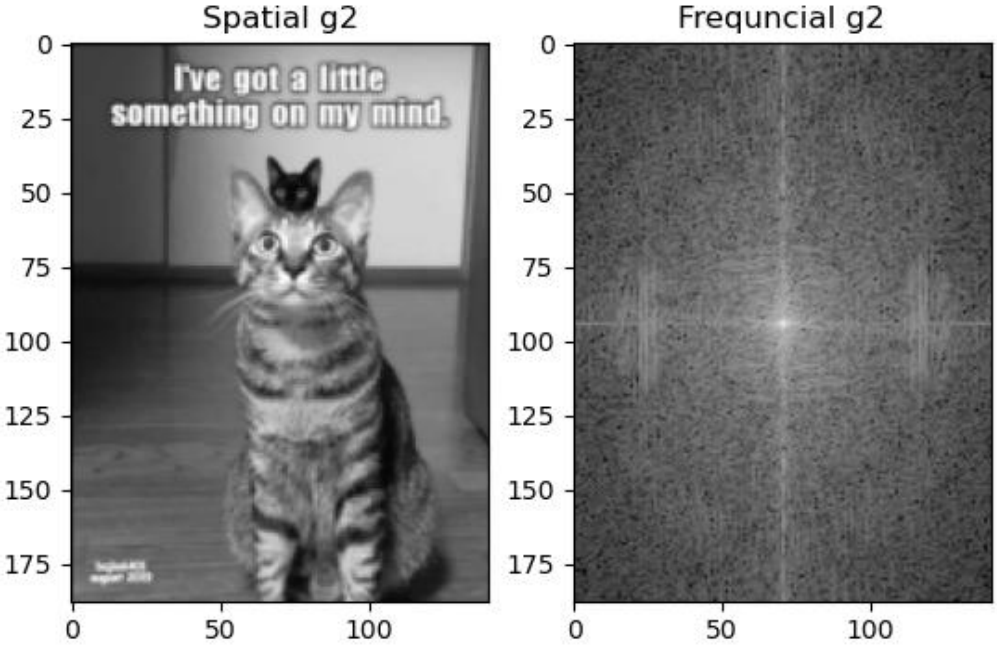


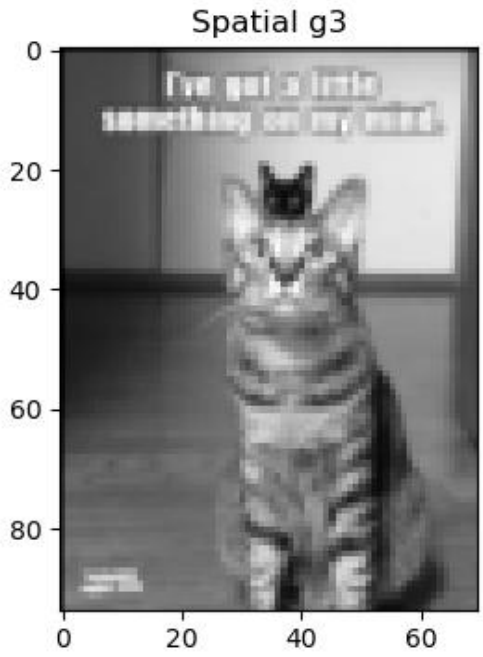
(b)

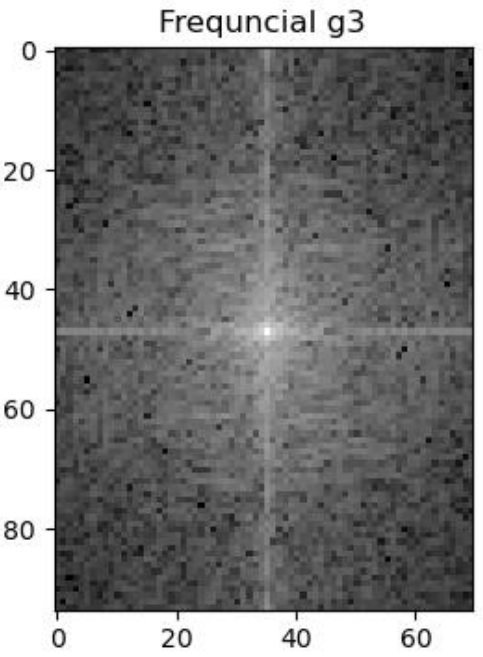
Display the FFT amplitudes of your Gaussian/Laplacian pyramids. Appropriate display ranges should be chosen so that the changes in frequency in different levels of the pyramid are clearly visible. Explain what the Laplacian and Gaussian pyramids are doing in terms of frequency.

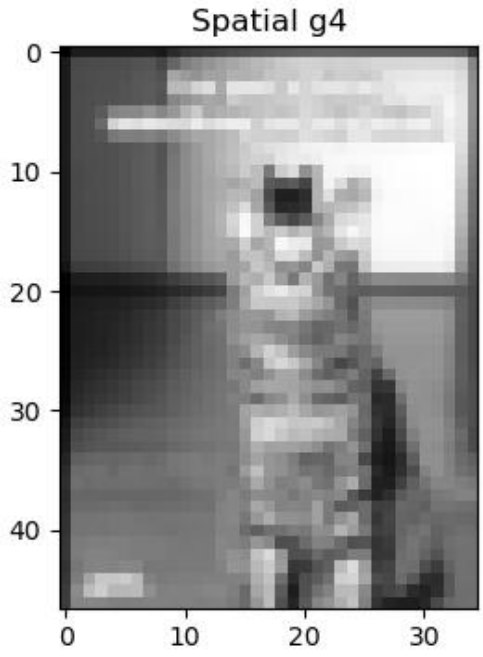


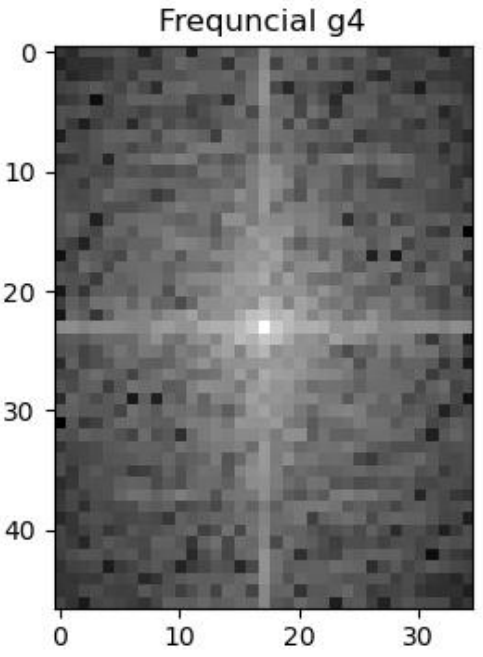

Spatial g1
Frequncial g1
I've got a little
something on my mind.
0
50
100
150
200
250
300
350
0
100
200

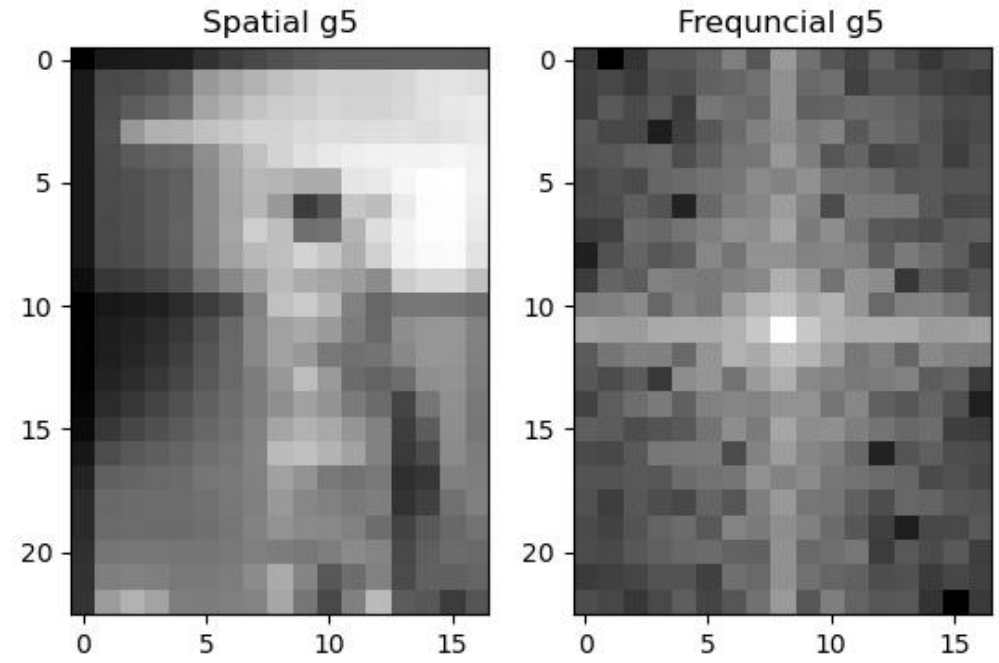

Spatial g2
Frequncial g2
I've got a little
something on my mind.
0
25
50
75
100
125
150
175
0
50
100

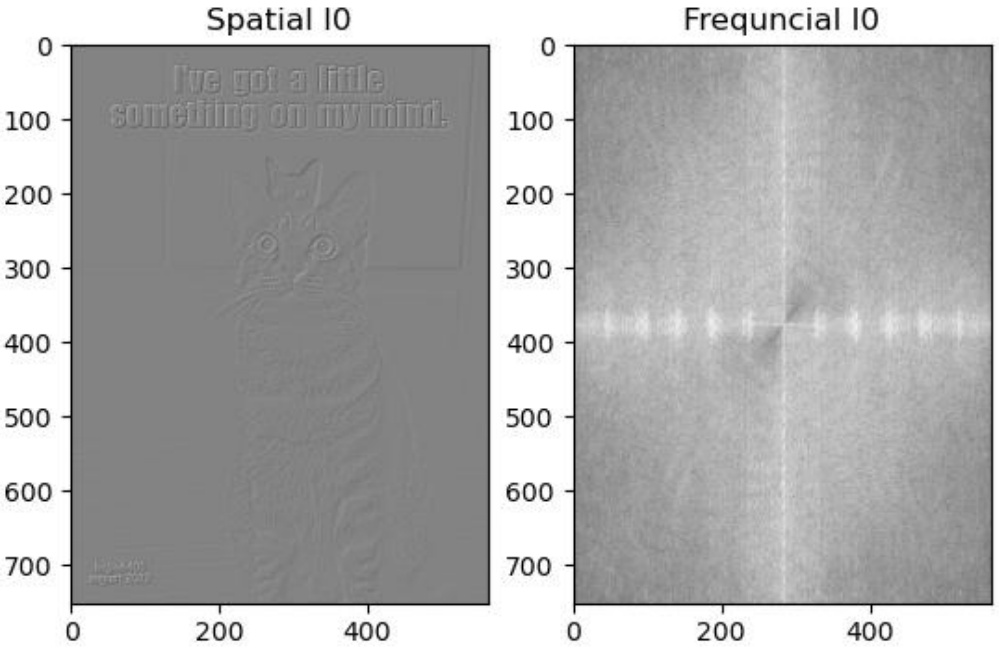

Spatial g3
0
20
40
60
80
0
20
40
60

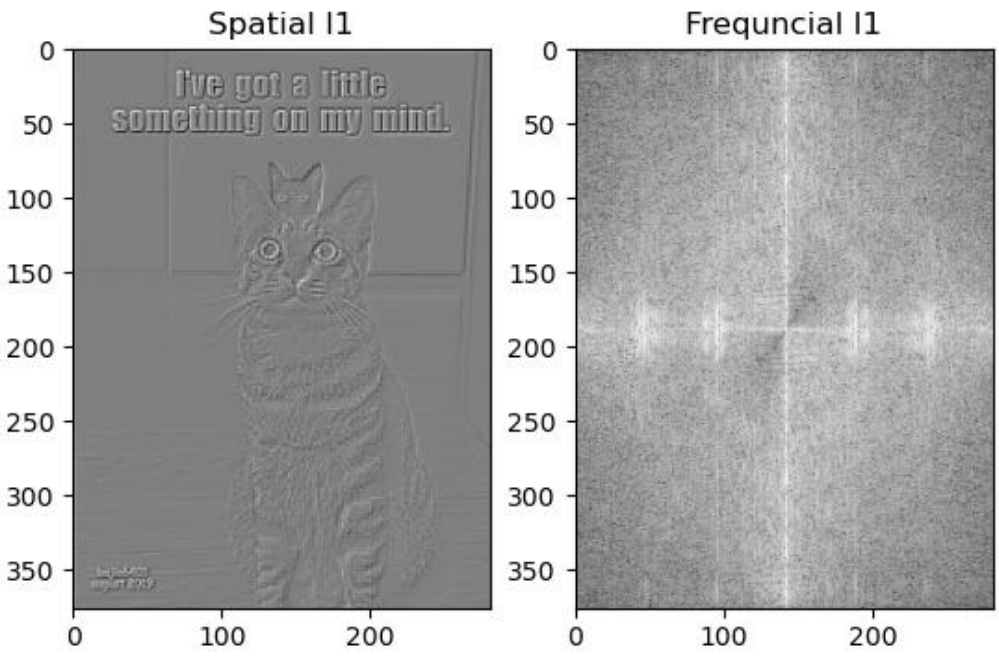

Frequncial g3
0
20
40
60
80
0
20
40
60

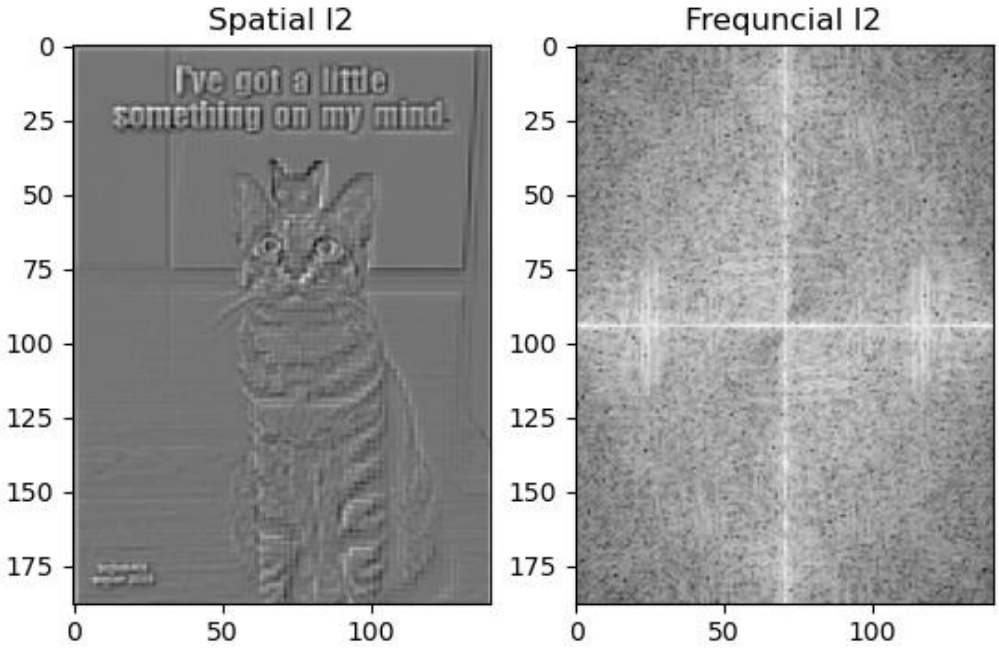

Spatial g4
0
10
20
30
40
0
10
20
30

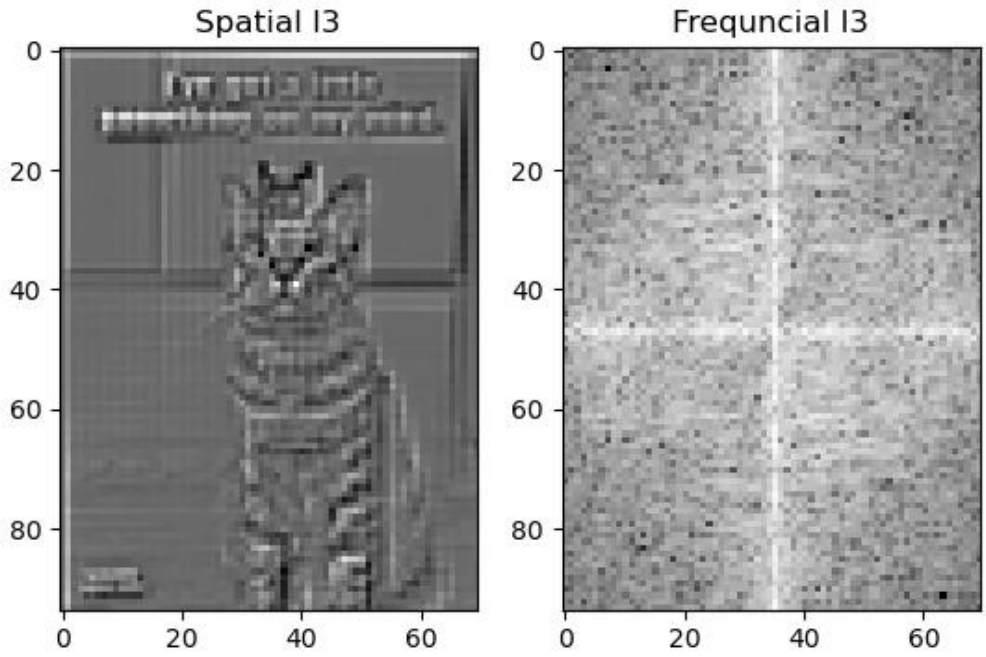

Frequncial g4
0
10
20
30
40
0
10
20
30

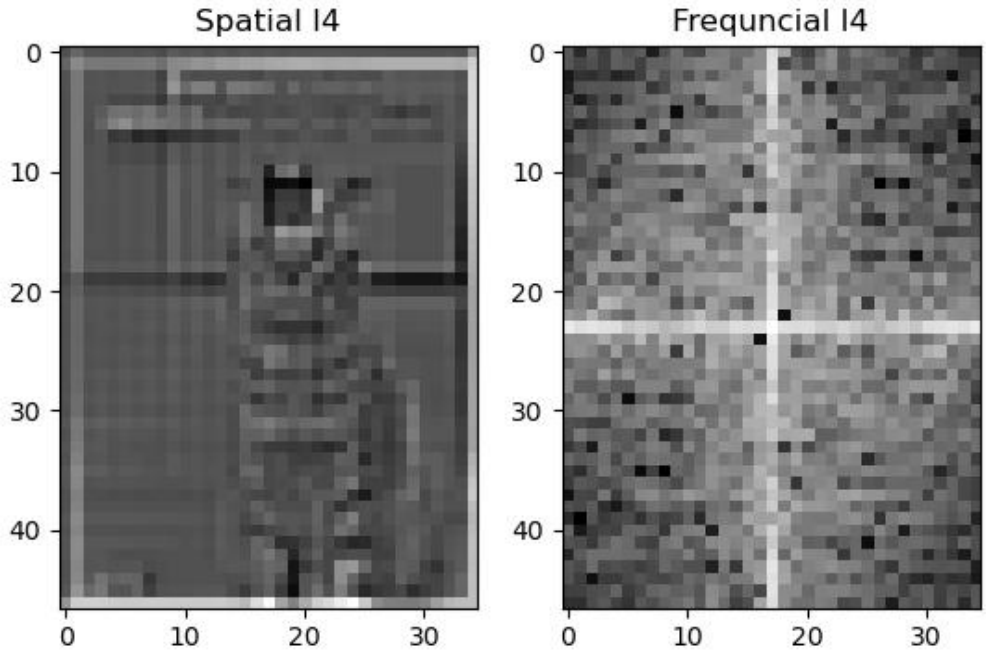

Spatial g5
Frequncial g5
0
5
10
15
20
0
5
10
15


Spatial I0
Frequncial I0
I've got a little
something on my mind.
0
100
200
300
400
500
600
700
0
200
400


Spatial I1
Frequncial I1
I've got a little
something on my mind.
0
50
100
150
200
250
300
350
0
100
200


Spatial I2
Frequncial I2
I've got a little
something on my mind.
0
25
50
75
100
125
150
175
0
50
100





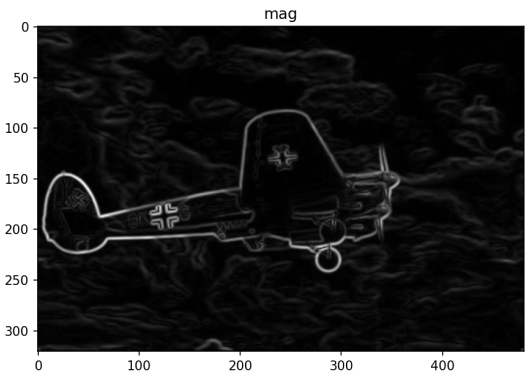
The Gaussian Pyramid can keep the low frequency information and the Laplacian Pyramid can keep the high frequency information.

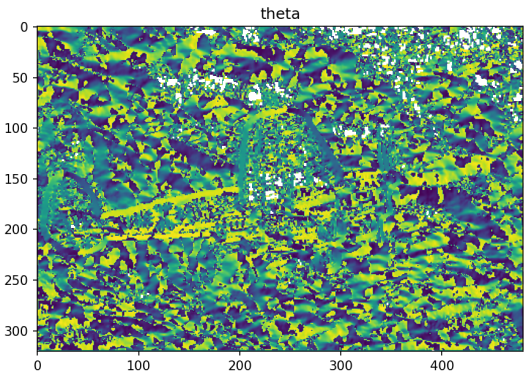
Problem 2

Edge Detection (20 points)

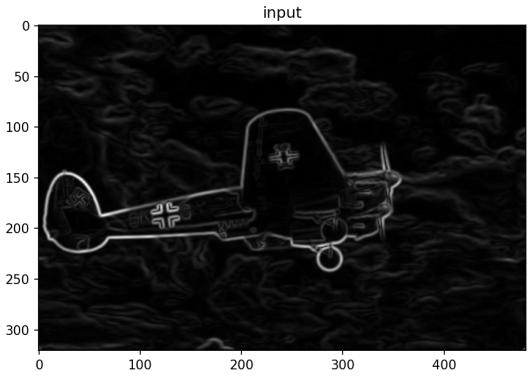
(a)

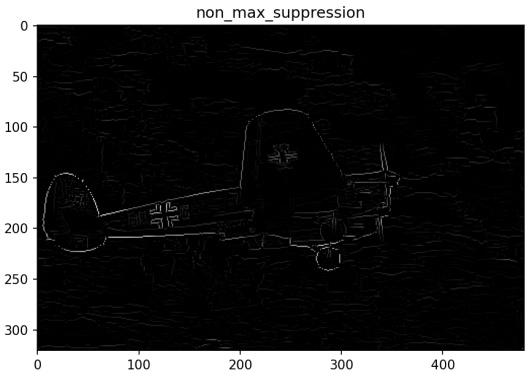
Build a simple gradient-based edge detector that includes the following functions `gradientMagnitude` & `edgeGradient`:





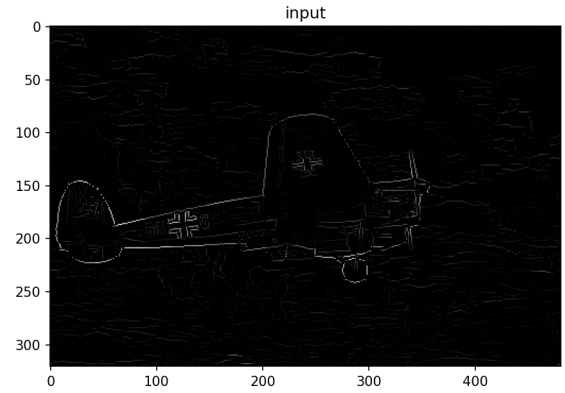
do the non-max-suppression



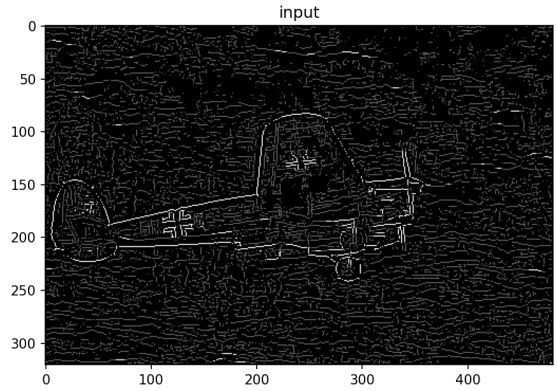


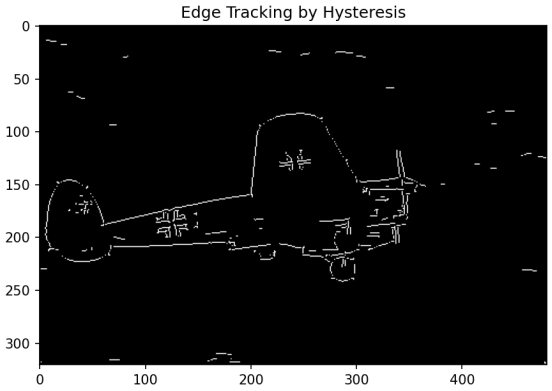
Double threshold

Parameter	Value
low Threshold Ratio	0.05
high Threshold Ratio	0.2
weak value	100
strong value	255



Edge Tracking by Hysteresis





Run

```
python edge_detector.py task_a
```

```
(ececv) tongzhen@yutianshu:~/projects/cv/prob_2$ python edge_detector.py task_a
data/images/test/226033.jpg
(321, 481, 3)
edge_detector.py:26: RuntimeWarning: invalid value encountered in double_scalars
△ 0   0
```

You will find the output under the **prob_2\data\png\test**

Run BSDS 500 boundary prediction evaluation suite

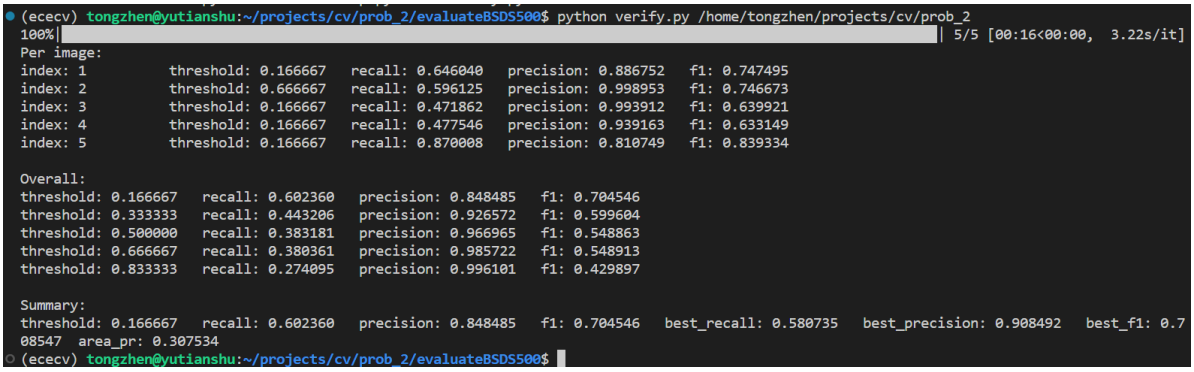
berkeley.edu/Research/Projects/CS/vision/bsds/

put the output of `data/images/test` pictures under `data/png/test`

Compile the extension module in the src directory.

```
python setup.py build_ext --inplace
python verify.py <path_to_bsds500_root_directory>
```

It gains 0.848485 precision.



Average	Value	Overall	Value
f1	0.747495	f1	0.704546
f2	0.746673	f2	0.599604
f3	0.639921	f3	0.548863
f4	0.633149	f4	0.548913
f5	0.839334	f5	0.429897

precision: 0.848485

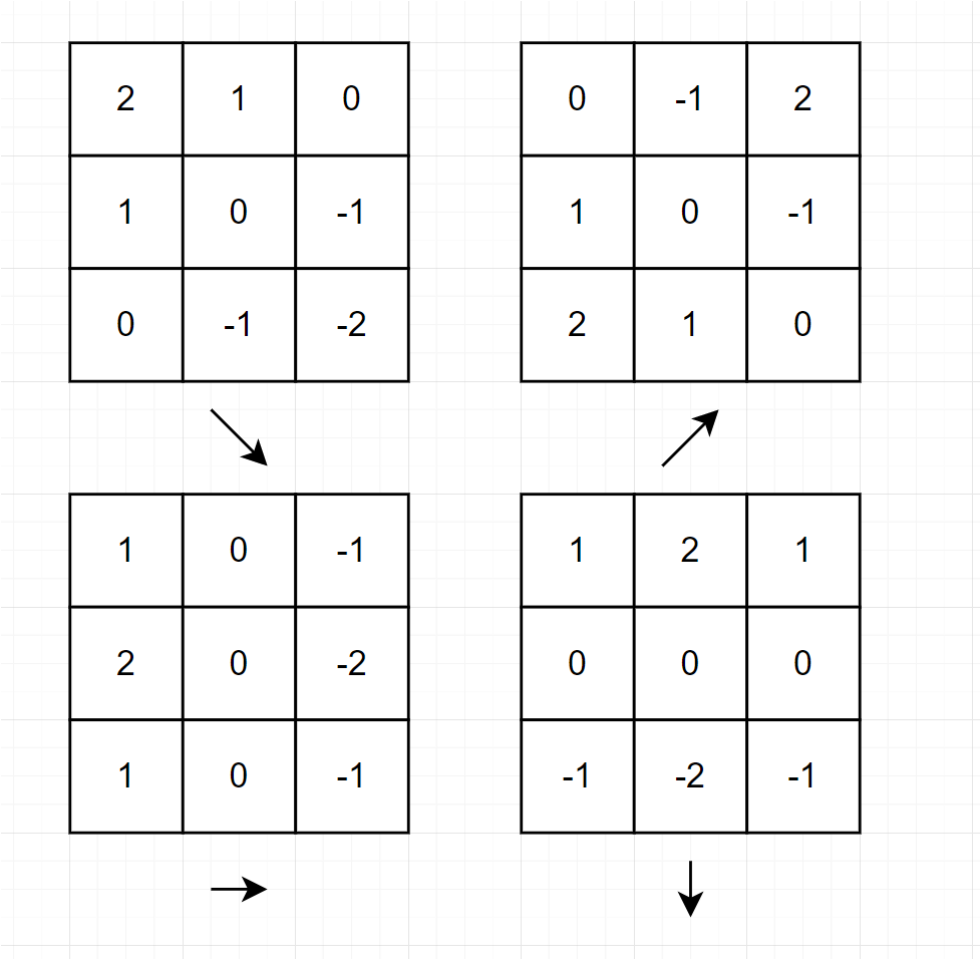
f1: 0.704546

best precision: .908492

best f1: 0.708547

(b)

Sobel-like kernels are used to calculate the derivative in other direction

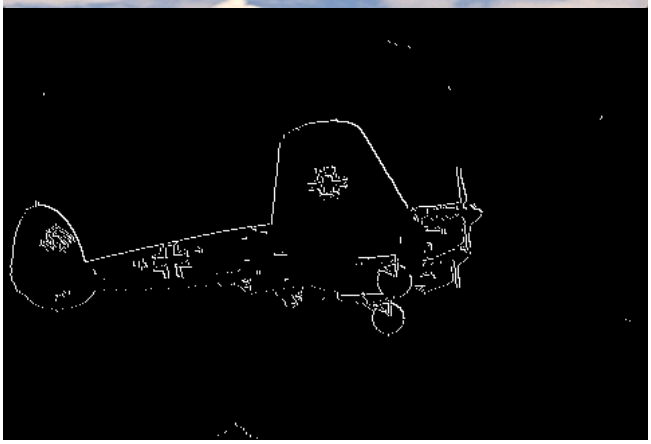


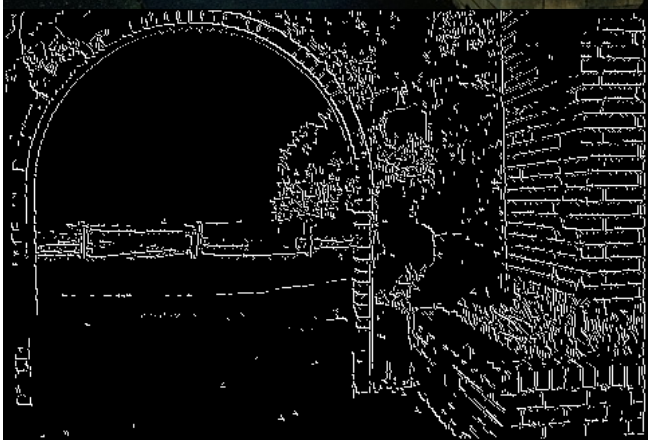
$$I_i = F_i * I, i \in [1, 2, 3, 4]$$
$$magnitude = max(I_1, I_2, I_3, I_4)$$

The theta is estimated by the largest value in the eight direction.

$$theta\_index = argmax(I_1, I_2, I_3, I_4, -I_1, -I_2, -I_3, -I_4)$$
$$theta = \frac{theta\_index}{8} \cdot \frac{\pi}{4}$$

Qualitative results:





Run

```
python edge_detector.py task_b
```

You will find the output under the prob_2\data\png\test

```
(ececv) tongzhen@yutianshu:~/projects/cv/prob_2$ python edge_detector.py task_b
data/images/test/226033.jpg
^CTraceback (most recent call last):
  File "edge_detector.py", line 260, in <module>
    task_b(read_dir, write_dir)
  File "edge_detector.py", line 237, in task_b
```

Run the verify.py as (a) above

```
(ececv) tongzhen@yutianshu:~/projects/cv/prob_2/evaluateBSDS500$ python verify.py /home/tongzhen/projects/cv/prob_2
100%|| 5/5 [00:14<00:00,  3.00s/it]
Per image:
index: 1         threshold: 0.166667   recall: 0.646110   precision: 0.887464   f1: 0.747795
index: 2         threshold: 0.666667   recall: 0.595475   precision: 1.000000   f1: 0.746455
index: 3         threshold: 0.166667   recall: 0.471655   precision: 0.994419   f1: 0.639835
index: 4         threshold: 0.166667   recall: 0.477546   precision: 0.939163   f1: 0.633149
index: 5         threshold: 0.166667   recall: 0.870008   precision: 0.811128   f1: 0.839537

Overall:
threshold: 0.166667   recall: 0.602316   precision: 0.848883   f1: 0.704653
threshold: 0.333333   recall: 0.443162   precision: 0.926713   f1: 0.599594
threshold: 0.500000   recall: 0.383296   precision: 0.966793   f1: 0.548953
threshold: 0.666667   recall: 0.380304   precision: 0.985903   f1: 0.548881
threshold: 0.833333   recall: 0.274081   precision: 0.995841   f1: 0.429855

Summary:
threshold: 0.166667   recall: 0.602316   precision: 0.848883   f1: 0.704653   best_recall: 0.580606   best_precision: 0.908969   best_f1: 0.7
08595  area_pr: 0.307572
(ececv) tongzhen@yutianshu:~/projects/cv/prob_2/evaluateBSDS500$
```

Overall	Value	Average	Value
f1	0.704653	f1	0.747795
f2	0.599594	f2	0.746455
f3	0.548953	f3	0.639835
f4	0.548881	f4	0.633149
f5	0.429855	f5	0.839537

precision: 0.848883

f1: 0.704653

best precision: .908969

best f1: 0.708595

Problem 3

Feature Tracker (40 points)

(a)

pseudocode

```
input image

get sobelx and sobelx from convolution image with Sobel filter
get Ixx, Ixy, Iyy from sobelx and sobelx
do the Gaussain Blur of Ixx, Ixy, Iyy
set a in [0.01, 0.06]
get determinant:detM = Ixx * Iyy - Ixy *Ixy
get trace: traceM = Ixx + Iyy
get response: response = detM - a * traceM *traceM
for element in response matrix{
    do threshold selection: element value > -tau && element value<tau
    do non-max-suppress: check element is the max in its 5x5 patch
}for end
```

```
for element in response matrix{
    if(element value > 0){
        add element position to keyXs, keyYs
    }
}for end

output keyXs, keyYs collection
```

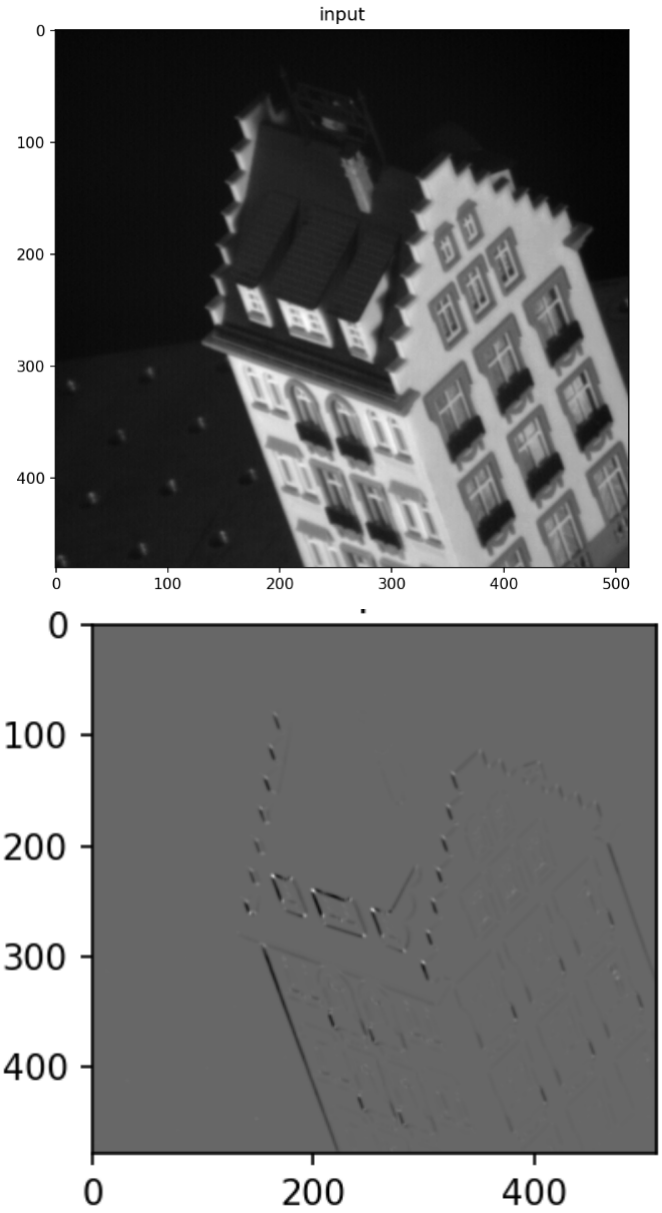
hyperparameter used:

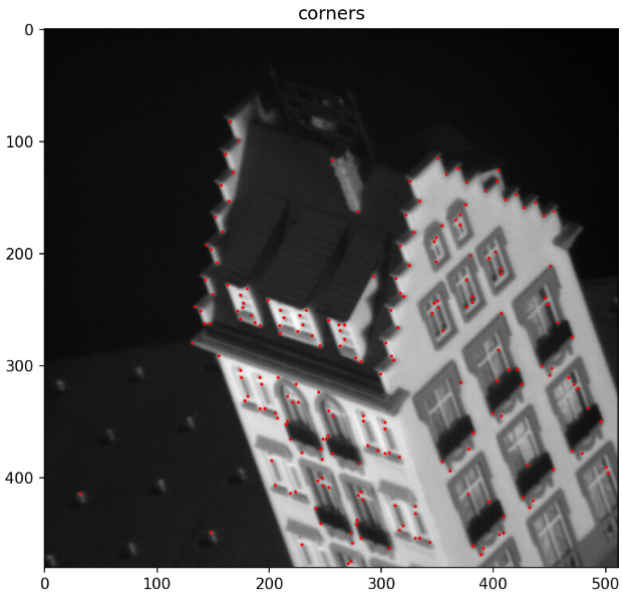
$$\tau = 10^7, a = 0.05$$

patch size used in non max suppression:

$$patch\_size = 5 \times 5$$

Harris key point:





(b)

Transition formula:

$$\begin{bmatrix} \sum_W I_x I_x & \sum_W I_x I_y \\ \sum_W I_x I_y & \sum_W I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum_W I_x I_t \\ \sum_W I_y I_t \end{bmatrix}$$

pseudocode

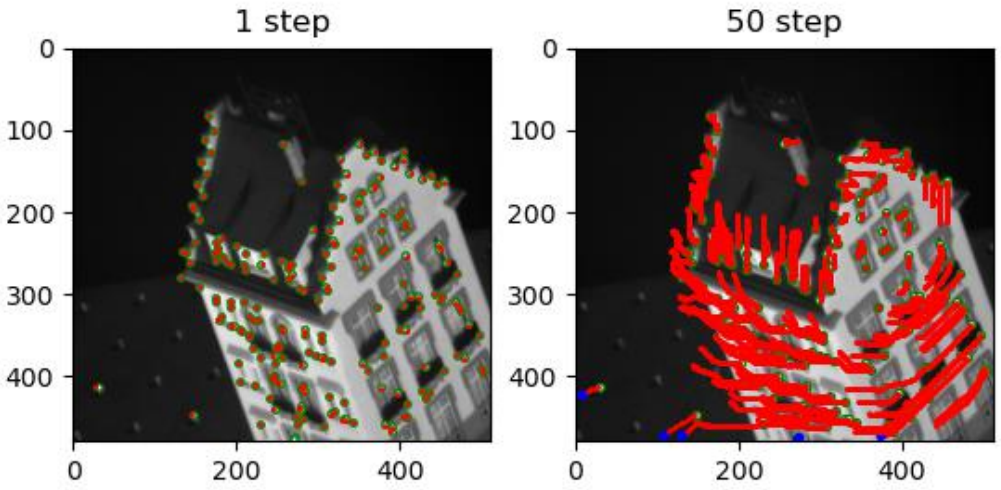
```
input Harris Corner points position x, y, cur_image, and next_image

do the Gaussain Blur for the cur_image, and next_image
get Ix and Iy by convolution the cur_image with sobel filter
initialize next_x, and next_y as x and y
get intensity patch I_xyt centered at x, y in image

while (find convergence less than maximen time){
    if the next_x or next_y out of image{
        output next_, next_y
    }
    get interpolation for image patch centered at next_x, and next_y
    do intensity difference: I_next_xy - I_xyt
    solve Ax=b in (Transition formula)
    update next_x += x[0]
    update next_y += x[1]
}
```

hyperparameter used:

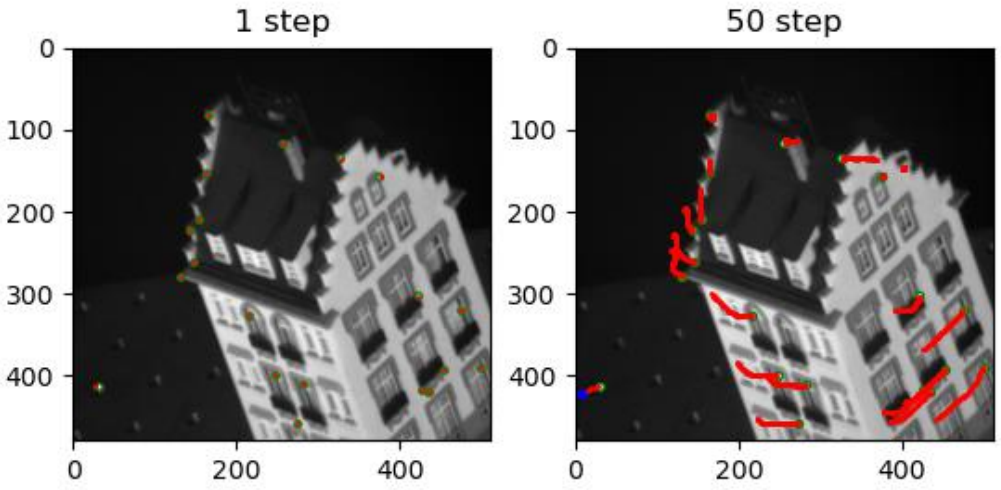
$$window\ size : W_{size} = 15$$
$$Iteration\ max = 30$$
$$stop\ threshold : |(u, v)|_2 <= 0.01$$

The green points are the corner points, the red points are the track trajectories for 50 steps, and the blue points are the points out of the boundary.

200 points to track



20 points to track



Run

input the numbers of points you want to track after the python file.

```
python FeatureTracker.py 20
```

```
(ececv) D:\Courses_2022_Fall\ECE4513\HM3\prob_3>python FeatureTracker.py 20
```

It will generate a file with track points number as name, like `20track.jpg`