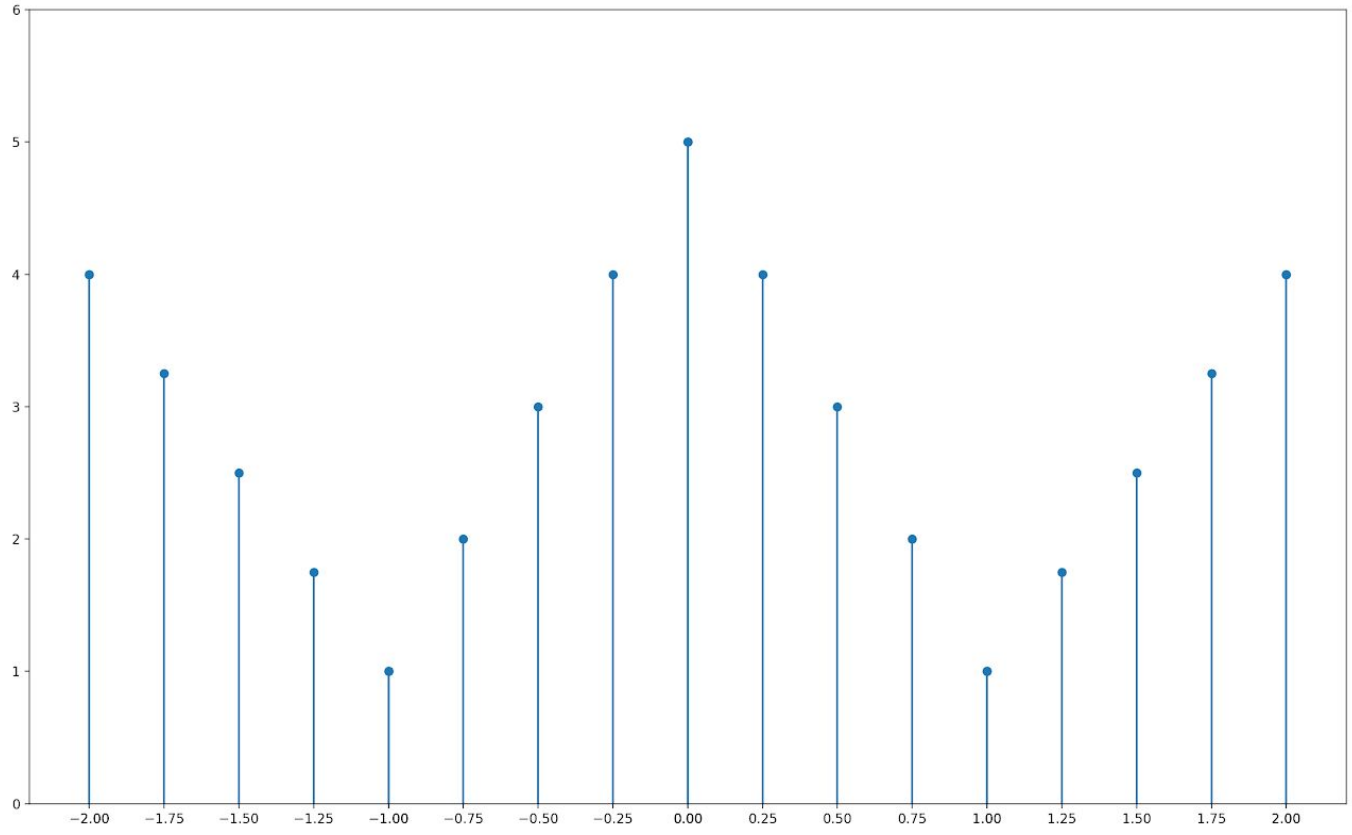
Question 1:

An additional 4 x coordinate is calculated between -2 and -1, -1 and 0, 0 and 1, 1 and 2.
Their corresponding F(x) value is calculated via the equation below:

$$F(x) = \frac{x - x_2}{x_1 - x_2} F(x_1) + \frac{x_1 - x}{x_1 - x_2} F(x_2)$$

Resulting graph and table:



x	f(x)
-2	4
-1.75	3.25
-1.5	2.5
-1.25	1.75
-1	1
-0.75	2

-0.5	3
-0.25	4
0	5
0.25	4
0.5	3
0.75	2
1.0	1
1.25	1.75
1.5	2.5
1.75	3.25
2	4

Question 2:

Given $x(t) = \sin^2(2F_a\pi t) + \frac{1}{7}\cos(F_b\pi t) + \frac{1}{5}\sin(5F_c\pi t)$

$= \frac{1}{2} - \frac{\cos(4F_a\pi t)}{2} + \frac{1}{7}\cos(F_b\pi t) + \frac{1}{5}\sin(5F_c\pi t)$ # by trigo identity $\sin^2\theta = 1 - \cos^2\theta$

Let 

Since each signal can be expressed as the form of $\sin(\omega t)$ or $\cos(\omega t)$, where $\omega = 2\pi f$ and f is the frequency of the individual component then, ① frequency of t_1, $f_1 = 0$, because the signal is a constant value

② $t_2 = \cos 4F_a\pi t$.
$\omega_2 = 4F_a\pi$
$2\pi f_2 = 4F_a\pi$
$f_2 = 2F_a$

③ $t_3 = \frac{1}{7}\cos(F_b\pi t)$
$\omega_3 = F_b\pi$
$2\pi f_3 = F_b\pi$
$f_3 = \frac{F_b}{2}$

④ $t_4 = \frac{1}{5}\sin(5F_c\pi t)$
$\omega_4 = 5F_c\pi$
$2\pi f_4 = 5F_c\pi$
$f_4 = \frac{5F_c}{2}$.

$\therefore$ sampling rate $= 2 \cdot \max\left(2F_a, \frac{F_b}{2}, \frac{5F_c}{2}\right)$ ※

Question 3a:

when $\lambda_1 = 0$, then λ_2 is ≥ 0
when λ_2 is > 0, it means that we are detecting edges.
when $\lambda_2 = 0$, then the patches we selected are textureless.

Question 3b:

Since we assume that N is positive semi-definite, then N can be expressed as the form of $e^T N e$, where e is a non-zero vector.
Let x' and y' be the upper bound at x and y respectively

Then, M can be written as

$$M = \sum_x \sum_y w(x,y)\, e^T \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix} e$$

$$= w(0,0)\, e^T \begin{pmatrix} I_0^2 & I_0 I_0 \\ I_0 I_0 & I_0^2 \end{pmatrix} e + w(0,1)\, e^T \begin{pmatrix} I_0^2 & I_0 I_1 \\ I_0 I_1 & I_1^2 \end{pmatrix} e + \cdots\cdots + w(x',y')\, e^T \begin{pmatrix} I_{x'} & I_{x'} I_{y'} \\ I_{x'} I_{y'} & I_{y'} \end{pmatrix} e$$

$$= w(0,0)\, e^T N_{00} e + w(0,1)\, e^T N_{01} e + \cdots\cdots + w(x',y')\, e^T N_{x'y'} e$$

$$= e^T \left(w(0,0) N_{00} + w(0,1) N_{01} + \cdots\cdots + w(x',y') N_{x'y'} \right) e$$

$$= e^T N' e, \text{ where } N' \text{ is the summation of all } N$$

$\therefore$ M is a positive semi-definite matrix

Part 2:

Function to find the path with minimum energy level

```
def find_min_path(img_grad):
    rows = img_grad.shape[0]
    cols = img_grad.shape[1]

    #create a table to keep track of the energy level and initilize them with infinity value
    energy_table = np.full(img_grad.shape, np.inf)
    #initialize the first row of the table with the calculated gradient value from the previous function
    energy_table[0] = img_grad[0]

    path_table = np.empty_like(img_grad, dtype=object)

    top_left_e = None
    top_center_e = None
    top_right_e = None

    for r in range(1, rows):
        for c in range(cols):
            if c-1 >= 0:
                top_left_e = energy_table[r -1 , c -1]
            else:
                top_left_e = np.inf

            if c + 1 < cols:
                top_right_e = energy_table[r-1, c + 1]
            else:
                top_right_e = np.inf

            top_center_e = energy_table[r-1, c]

            energy_table[r,c] = img_grad[r,c] + min(top_left_e, top_center_e, top_right_e)

            if top_left_e == min(top_left_e, top_center_e, top_right_e):
                path_table[r, c] = (r-1, c-1)

            elif top_center_e == min(top_left_e, top_center_e, top_right_e):
                path_table[r,c] = (r-1, c)

            else:
                path_table[r,c] = (r-1, c+1)

    min_c = np.inf
    min_c_index = 0
    for c in range(cols):
        if energy_table[rows-1, c] < min_c:
            min_c = energy_table[rows-1, c]
            min_c_index = c

    return path_table, (rows-1, min_c_index)
```

Removing the pixel

```
while True:
    i, j = seam_index[0], seam_index[1]

    carve_img[i,:,0] = np.delete(image[i,:,0], j)
    carve_img[i,:,1] = np.delete(image[i,:,1], j)
    carve_img[i,:,2] = np.delete(image[i,:,2], j)

    if i != 0:
        seam_index = path_table[i,j]
    else:
        break
```

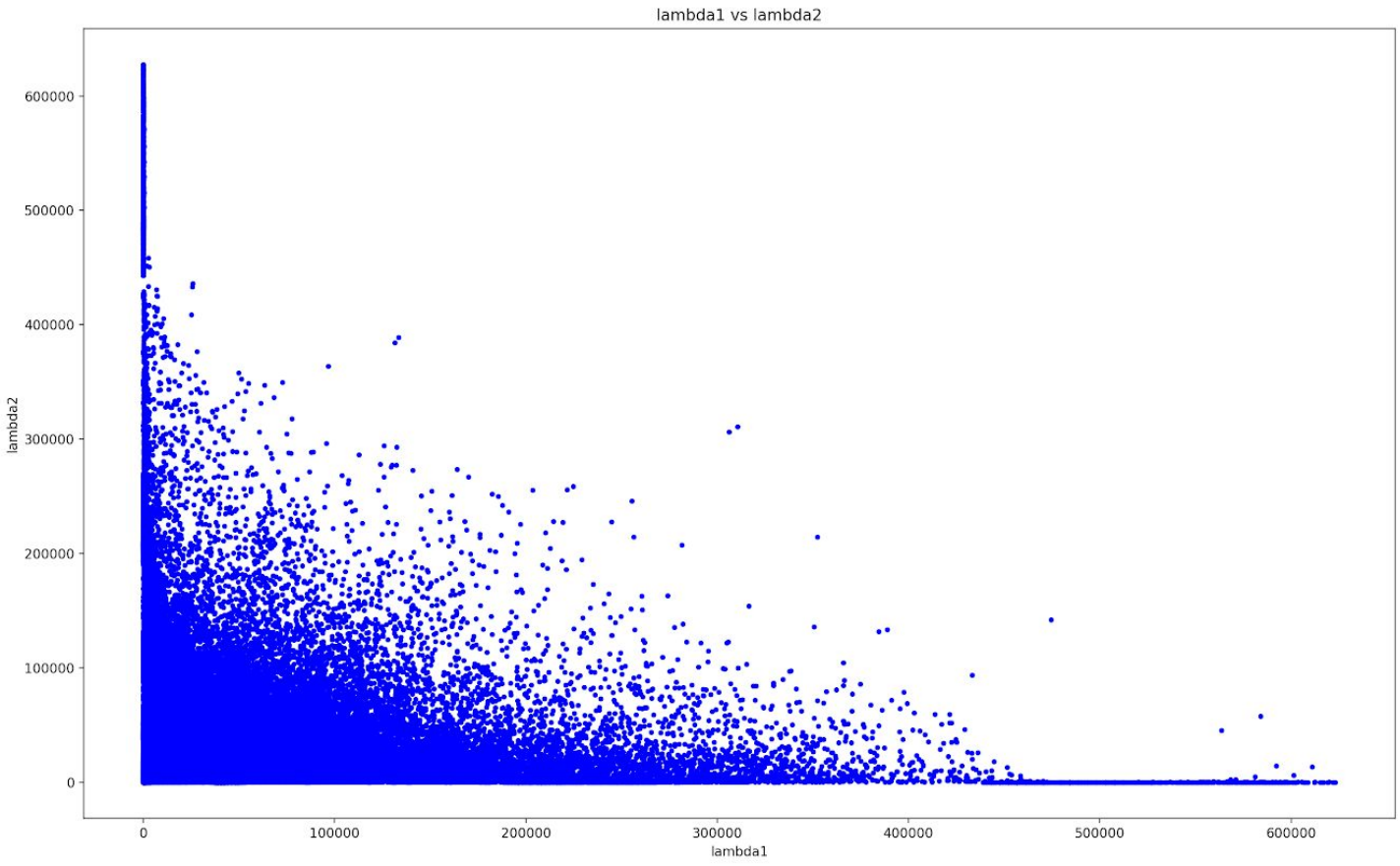
NOTE: result images aren't included in the extremely slow runtime to process the image even for one iteration, which makes it virtually impossible to finish processing an image in reasonable time. This is probably because the program has to delete the pixel one by one and then recompute its energy value and path. The program could work in reasonable time if we just need to remove just a few rows/ columns. However, the end result isn't noticeable, which is why I don’t even include that. The image size definitely shrinks and this could be checked manually on the image properties.

Part 3:

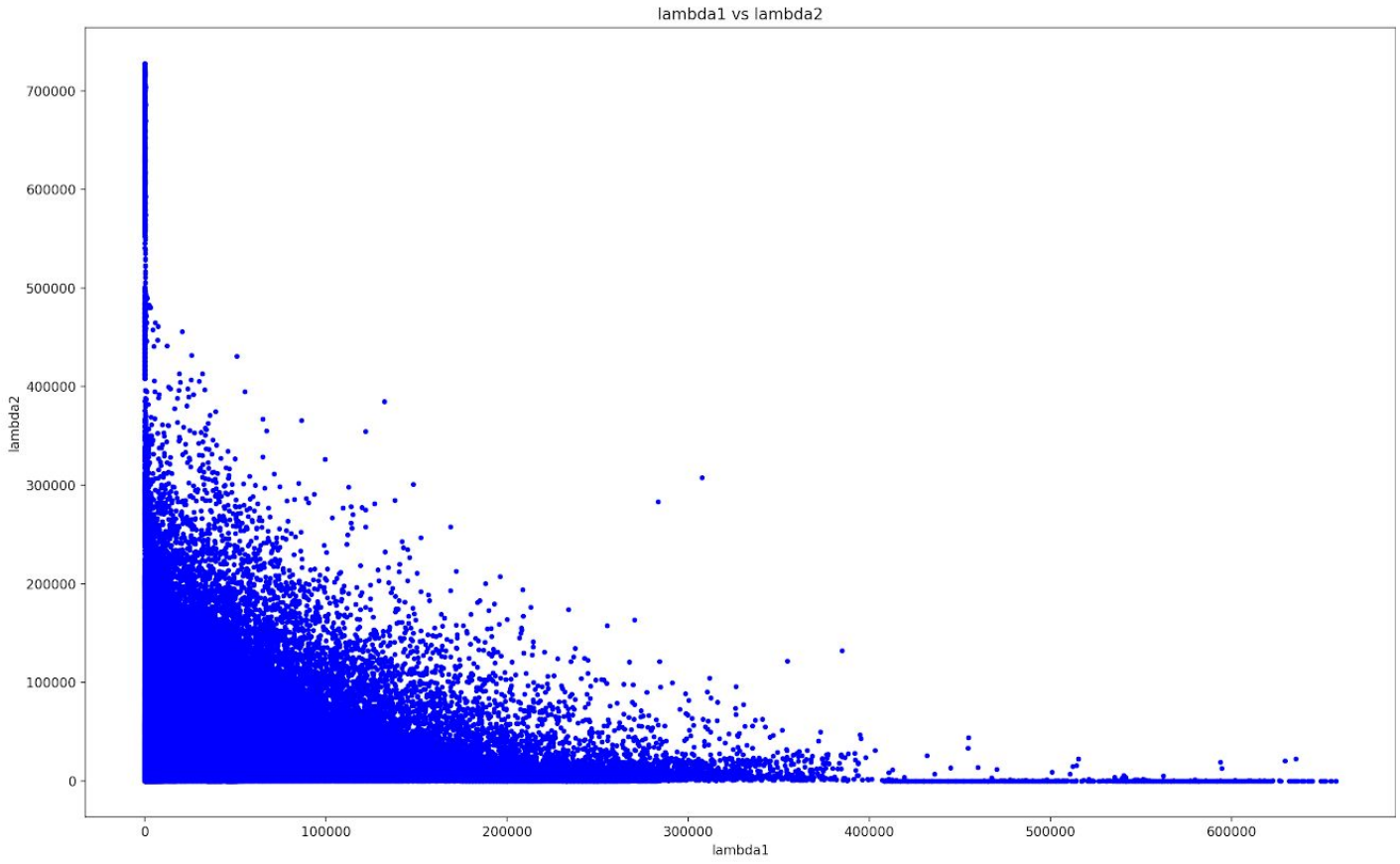
Calculate the eigenvalue

```
def corner_detector(gray_img, rgb_img, window_size, stddev, threshold):

    img_grad, IxIy, Ix_squared, Iy_squared = gradMagnitude(gray_img)

    if window_size %2 == 0:
        window_size = window_size +1

    k = int((window_size-1)/2)
    window = np.zeros((window_size, window_size))
    window[k,k] = 1
    window =ndimage.gaussian_filter(window, sigma= stddev)


    M_11 = ndimage.correlate(Ix_squared, window, mode = 'nearest')
    M_22 = ndimage.correlate(Iy_squared, window, mode = 'nearest')
    M_12 = M_21 = ndimage.correlate(IxIy, window, mode = 'nearest')
```

Scatterplot for University_College:

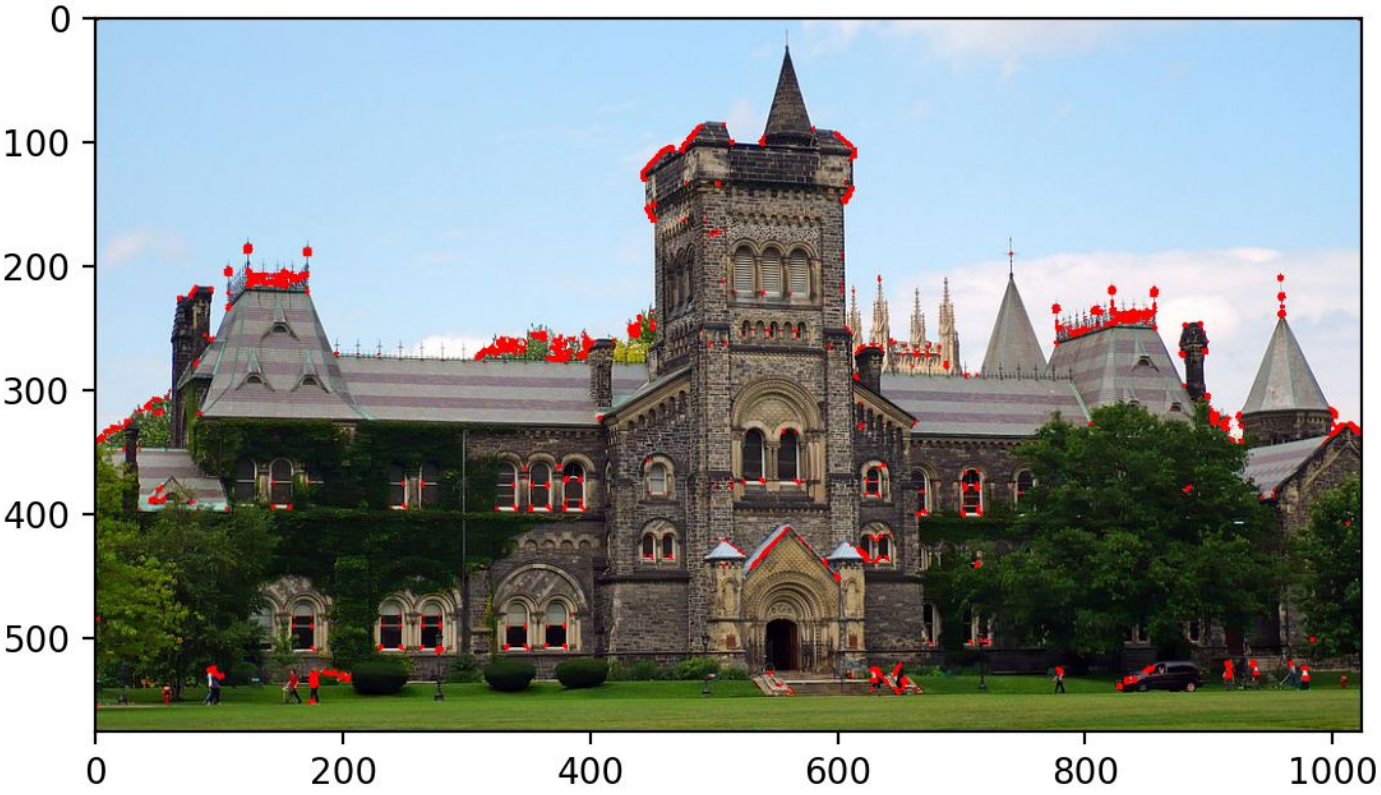


Scatterplot for University_College_Lawn:

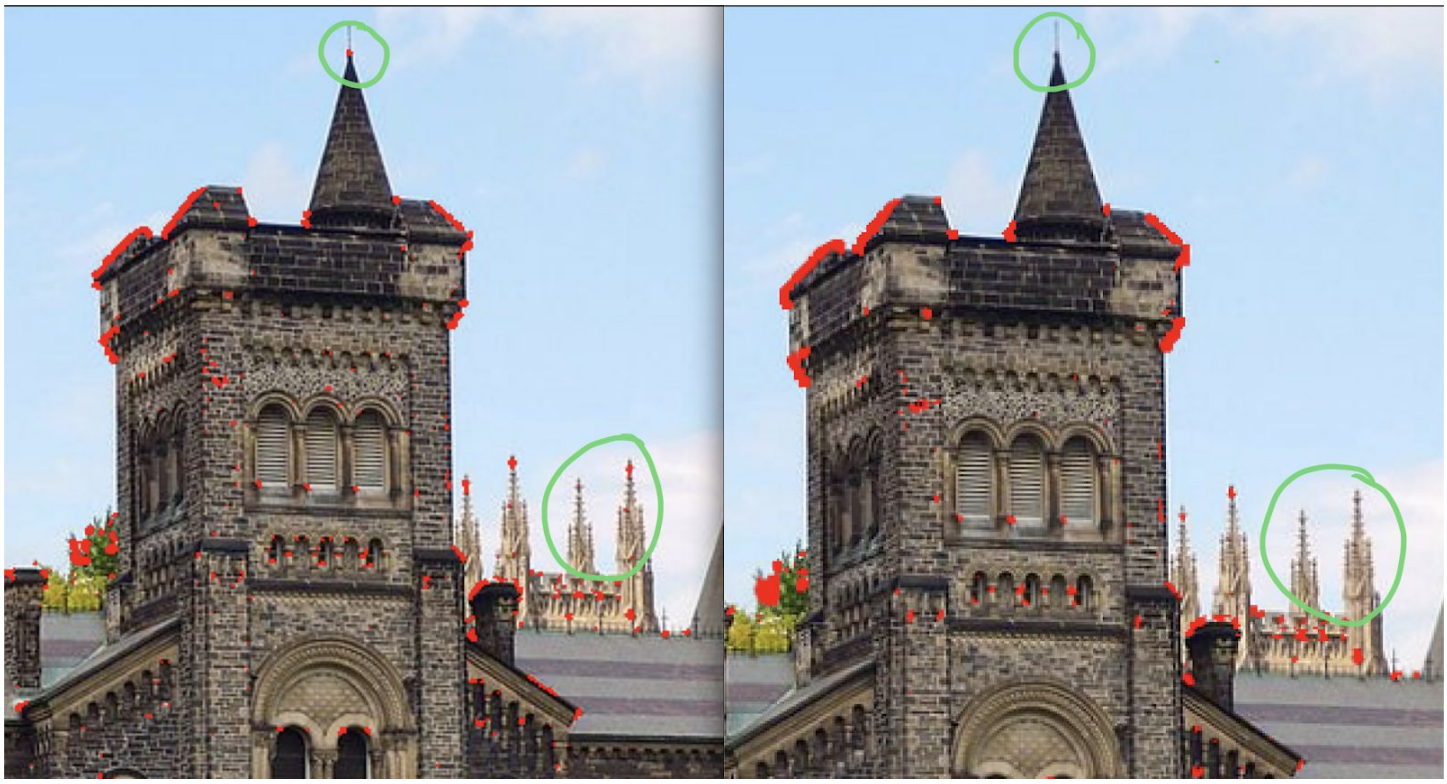


Threshold: 40 000 gaussian standard deviation: 1
*Corners are labelled red

Threshold: 40 000 gaussian standard deviation: 10



With a larger sigma value, when the gaussian filter is applied, the image will become less detailed and lose some of its texture. As a result, the gradient of a pixel with its neighbouring pixels decreases, which makes it harder to detect a corner. The comparison can be seen below:



small sigma (left) vs big sigma (right)

As shown in circles, the program is not able to detect sharp corners in images with larger sigma.