AA 274A: Principles of Robot Autonomy I
Problem Set 3

Name: Abhyudit Singh Manhas
SUID: 06645995

Problem 2: Line Extraction

(i) Implemented the functions `SplitLinesRecursive`, `FindSplit`, `FitLine` and `MergeColinearNeigbors` in `ExtractLines.py`.

(ii) Extracted lines for `rangeData_5_5_180.csv` with parameters

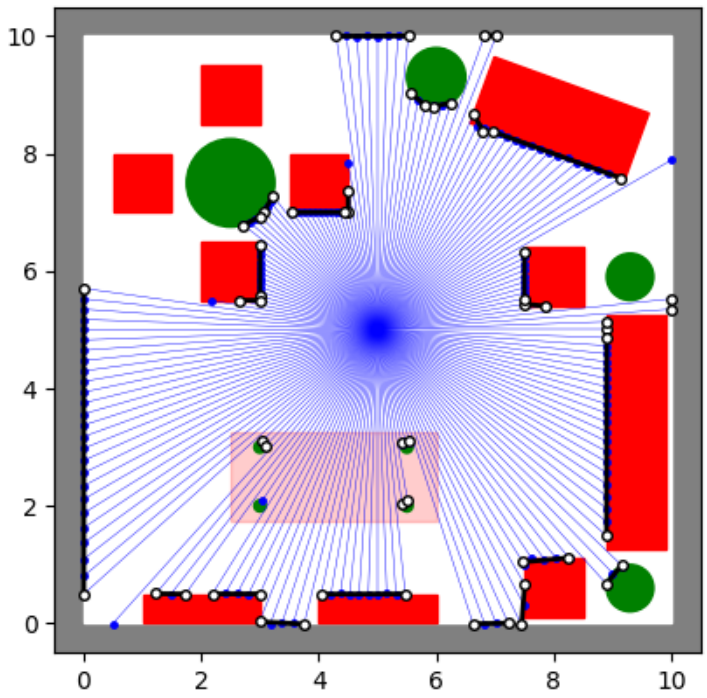
$$\text{MIN\_SEG\_LENGTH} = 0.05$$

$$\text{LINE\_POINT\_DIST\_THRESHOLD} = 0.02$$

$$\text{MIN\_POINTS\_PER\_SEGMENT} = 2$$

$$\text{MAX\_P2P\_DIST} = 0.35$$

is shown below:



Extracted lines for rangeData_4_9_360.csv with parameters

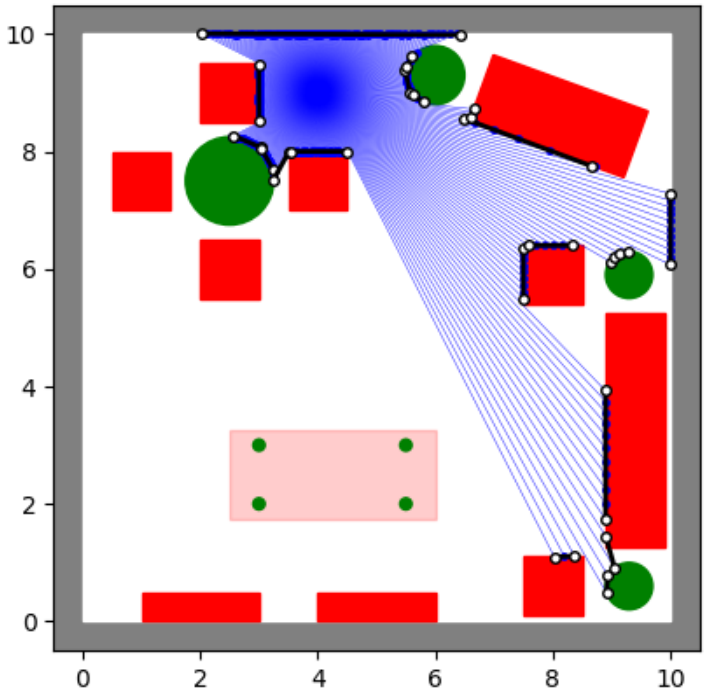
$$\text{MIN\_SEG\_LENGTH} = 0.05$$

$$\text{LINE\_POINT\_DIST\_THRESHOLD} = 0.02$$

$$\text{MIN\_POINTS\_PER\_SEGMENT} = 2$$

$$\text{MAX\_P2P\_DIST} = 0.8$$

is shown below:



Extracted lines for rangeData_7_2_90.csv with parameters

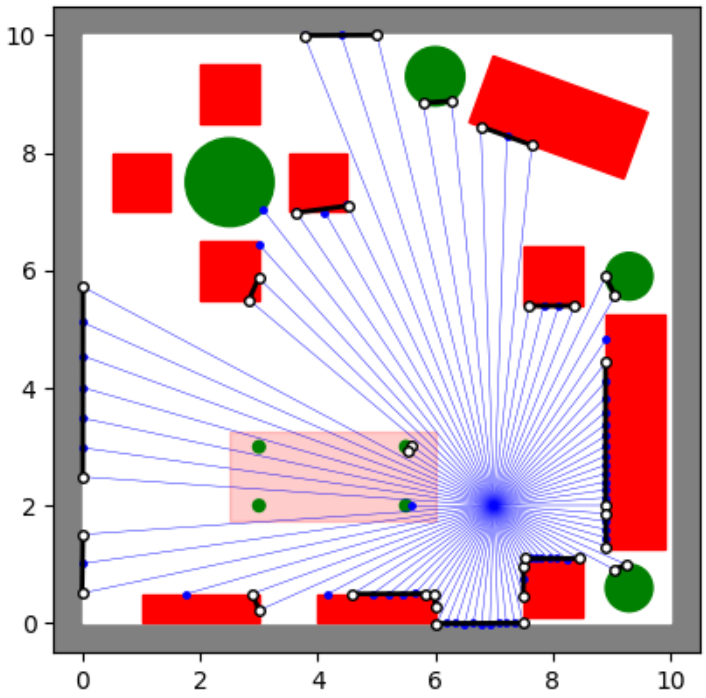
$$\text{MIN\_SEG\_LENGTH} = 0.05$$

$$\text{LINE\_POINT\_DIST\_THRESHOLD} = 0.02$$

$$\text{MIN\_POINTS\_PER\_SEGMENT} = 2$$

$$\text{MAX\_P2P\_DIST} = 0.3$$

is shown below:


10
8
6
4
2
0
0
2
4
6
8
10

Problem 3: Linear Filtering

(i) We have the original image $I = \begin{bmatrix} 7 & 4 & 1 \\ 8 & 5 & 2 \\ 9 & 6 & 3 \end{bmatrix}$ and the zero-padded image $\bar{I} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 7 & 4 & 1 & 0 \\ 0 & 8 & 5 & 2 & 0 \\ 0 & 9 & 6 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

(a) $F = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$. For this, $k = l = 3$. Hence, we have

$$G(i,j) = \sum_{u=0}^{k-1} \sum_{v=0}^{l-1} F(u,v) \cdot \bar{I}(i+u, j+v)$$

$$\implies G(i,j) = \sum_{u=0}^{2} \sum_{v=0}^{2} F(u,v) \cdot \bar{I}(i+u, j+v)$$

$$\implies G(i,j) = F(u=1, v=1) \cdot \bar{I}(i+1, j+1)$$

$$\implies G(i,j) = \bar{I}(i+1, j+1)$$

For example, $G(0,0) = \bar{I}(1,1) = 7$. Hence the output image G is

$$G = \begin{bmatrix} 7 & 4 & 1 \\ 8 & 5 & 2 \\ 9 & 6 & 3 \end{bmatrix}$$

(b) $F = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$. For this, $k = l = 3$. Hence, we have

$$G(i,j) = \sum_{u=0}^{k-1} \sum_{v=0}^{l-1} F(u,v) \cdot \bar{I}(i+u, j+v)$$

$$\implies G(i,j) = \sum_{u=0}^{2} \sum_{v=0}^{2} F(u,v) \cdot \bar{I}(i+u, j+v)$$

$$\implies G(i,j) = F(u=0, v=0) \cdot \bar{I}(i,j)$$

$$\implies G(i,j) = \bar{I}(i,j)$$

For example, $G(0,0) = \bar{I}(0,0) = 0$. Hence the output image G is

$$G = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 7 & 4 \\ 0 & 8 & 5 \end{bmatrix}$$

(c) $F = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$. For this, $k = l = 3$. Hence, we have

$$G(i,j) = \sum_{u=0}^{k-1} \sum_{v=0}^{l-1} F(u,v) \cdot \bar{I}(i+u, j+v)$$

$$\implies G(i,j) = \sum_{u=0}^{2} \sum_{v=0}^{2} F(u,v) \cdot \bar{I}(i+u, j+v)$$

$$\implies G(i,j) = F(0,0) \cdot \bar{I}(i,j) + F(0,1) \cdot \bar{I}(i,j+1) + F(0,2) \cdot \bar{I}(i,j+2) +$$

$$F(2,0) \cdot \bar{I}(i+2,j) + F(2,1) \cdot \bar{I}(i+2,j+1) + F(2,2) \cdot \bar{I}(i+2,j+2)$$

$$\implies G(i,j) = \bar{I}(i,j) + \bar{I}(i,j+1) + \bar{I}(i,j+2) - \bar{I}(i+2,j) - \bar{I}(i+2,j+1) - \bar{I}(i+2,j+2)$$

For example

$$G(0,0) = \bar{I}(0,0) + \bar{I}(0,1) + \bar{I}(0,2) - \bar{I}(2,0) - \bar{I}(2,1) - \bar{I}(2,2) = 0 + 0 + 0 - 0 - 8 - 5 = -13$$

Hence the output image G is

$$G = \begin{bmatrix} -13 & -15 & -7 \\ -4 & -6 & -4 \\ 13 & 15 & 7 \end{bmatrix}$$

The filter F takes the derivative of the image along the y-axis (vertical direction), which could be used to perform edge detection tasks.

For the filter $F' = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$, the output image G is

$$G = \begin{bmatrix} 9 & -12 & -9 \\ 15 & -18 & -15 \\ 11 & -12 & -11 \end{bmatrix}$$

The filter F' is different from F in that it takes the derivative of the image along the x-axis (horizontal direction). Even this can be used to perform edge detection tasks.

(d) $F = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$. For this, $k = l = 3$. Hence, we have

$$G(i,j) = \sum_{u=0}^{k-1} \sum_{v=0}^{l-1} F(u,v) \cdot \bar{I}(i+u, j+v)$$

$$\implies G(i,j) = \sum_{u=0}^{2} \sum_{v=0}^{2} F(u,v) \cdot \bar{I}(i+u, j+v)$$

$$\implies G(i,j) = \frac{1}{16}(\bar{I}(i,j) + 2\bar{I}(i,j+1) + \bar{I}(i,j+2) + 2\bar{I}(i+1,j) + 4\bar{I}(i+1,j+1) + 2\bar{I}(i+1,j+2) +$$

$$\bar{I}(i+2,j) + 2\bar{I}(i+2,j+1) + \bar{I}(i+2,j+2))$$

For example

$$G(0,0) = \frac{1}{16}(4 \cdot 7 + 2 \cdot 4 + 2 \cdot 8 + 5) = \frac{57}{16} = 3.5625$$

Hence the output image G is

$$G = \frac{1}{16}\begin{bmatrix}57 & 52 & 21\\ 84 & 80 & 36\\ 69 & 68 & 33\end{bmatrix} = \begin{bmatrix}3.5625 & 3.25 & 1.3125\\ 5.25 & 5 & 2.25\\ 4.3125 & 4.25 & 2.0625\end{bmatrix}$$

The filter F is a normalized Gaussian filter which blurrs the regions of the image. It could be used to filter out noise/high frequency components on either axis.

For the filter $F' = \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$, the output image G is

$$G = \frac{1}{9}\begin{bmatrix}24 & 27 & 12\\ 39 & 45 & 21\\ 28 & 33 & 16\end{bmatrix} = \begin{bmatrix}2.67 & 3 & 1.33\\ 4.33 & 5 & 2.33\\ 3.11 & 3.67 & 1.78\end{bmatrix}$$

The filter F' is different from F in that it is a moving average filter which returns the average of the pixels in the image. It achieves a smoothing effect and removes sharp features.

(ii) We have

$$G(i,j) = \sum_{u=0}^{k-1}\sum_{v=0}^{l-1}\sum_{w=0}^{c-1} F(u,v,w)\cdot \bar{I}(i+u, j+v, w) \tag{1}$$

Changing the order of the sum, we get

$$G(i,j) = \sum_{w=0}^{c-1}\sum_{u=0}^{k-1}\sum_{v=0}^{l-1} F(u,v,w)\cdot \bar{I}(i+u, j+v, w) \tag{2}$$

Through a slight abuse of notation, let $F_w(u,v)$ and $\bar{I}_w(i+u, j+v)$ denote column vectors of size $c\times 1$, for a specific value of u and v. We can thus drop all the sums in equation 2 and write out the vectors explicitly. We get

$$G(i,j) = \begin{bmatrix}F_w^{\mathrm{T}}(0,0) & F_w^{\mathrm{T}}(0,1) & \dots & F_w^{\mathrm{T}}(k-1,l-1)\end{bmatrix}\begin{bmatrix}\bar{I}_w(i,j)\\ \bar{I}_w(i,j+1)\\ \vdots\\ \bar{I}_w(i+k-1,j+l-1)\end{bmatrix} \tag{3}$$

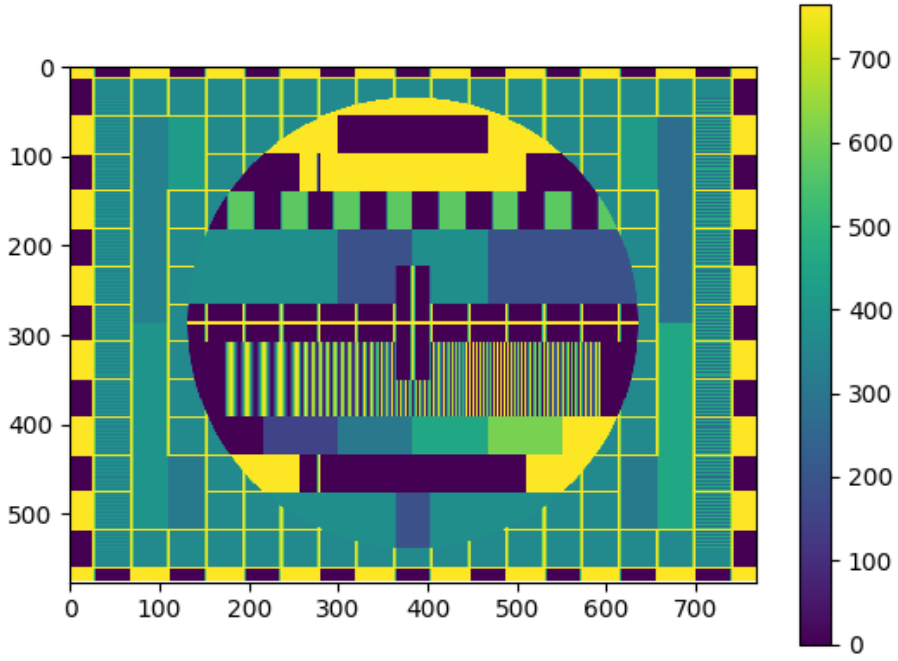
$$\implies G(i,j) = \begin{bmatrix}F_w(0,0)\\ F_w(0,1)\\ \vdots\\ F_w(k-1,l-1)\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}\bar{I}_w(i,j)\\ \bar{I}_w(i,j+1)\\ \vdots\\ \bar{I}_w(i+k-1,j+l-1)\end{bmatrix} \tag{4}$$

$$\implies G(i,j) = f^{\mathrm{T}} t_{ij} \tag{5}$$

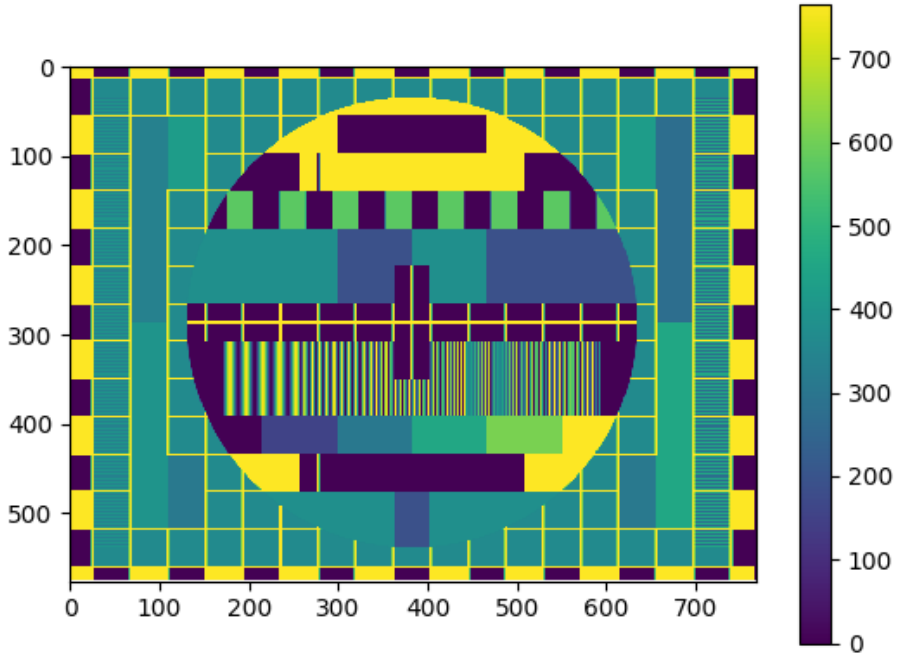
Hence, we have proved that correlation can be written as a dot product of two specially-formed vectors, f and t_{ij}. Both f and t_{ij} are column vectors of size $k\cdot l\cdot c\times 1$, and are given by

$$f = \begin{bmatrix}F_w(0,0)\\ F_w(0,1)\\ \vdots\\ F_w(k-1,l-1)\end{bmatrix}, \quad t_{ij} = \begin{bmatrix}\bar{I}_w(i,j)\\ \bar{I}_w(i,j+1)\\ \vdots\\ \bar{I}_w(i+k-1,j+l-1)\end{bmatrix} \tag{6}$$

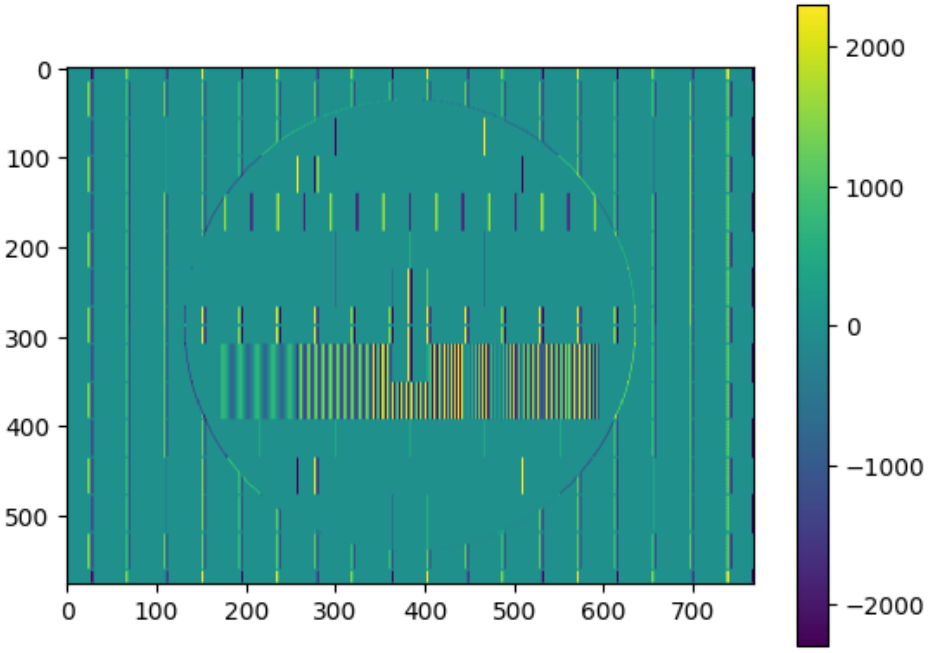
(iii) Implemented the correlation operation within the `corr` function in `linear_filter.py`.
The output image when `test_card.png` is correlated with filter 1 is shown below:



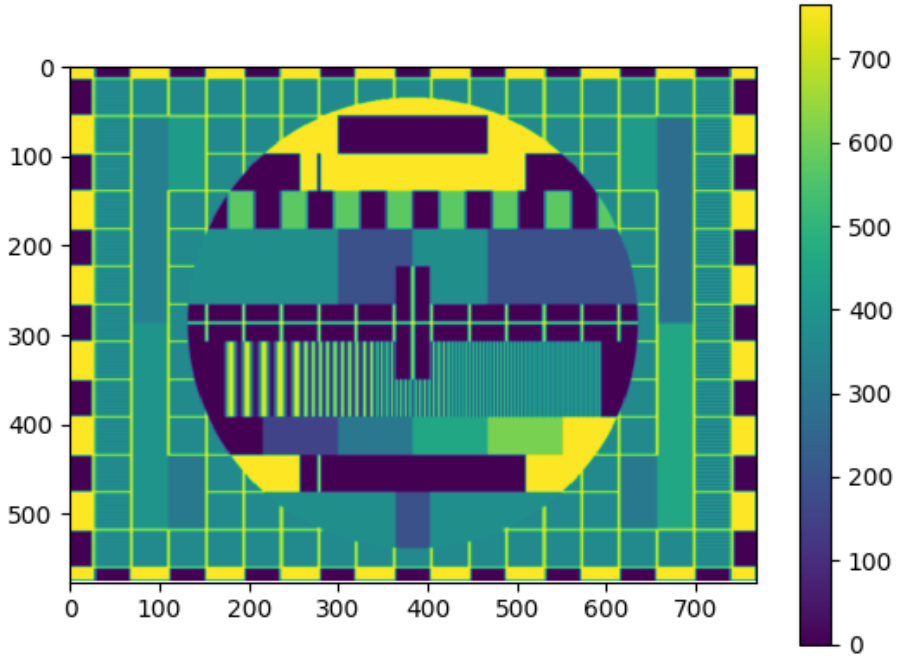
The output image when `test_card.png` is correlated with filter 2 is shown below:



The output image when `test_card.png` is correlated with filter 3 is shown below:



The output image when `test_card.png` is correlated with filter 4 is shown below:



(iv) The runtimes of the correlation function are:

1.8045427799224854 seconds for filter 1
2.1053833961486816 seconds for filter 2
2.02475905418396 seconds for filter 3
2.028918743133545 seconds for filter 4

These runtimes are slow and the filtering needs to be sped up. Two ways through which we could speed up the filtering are:

(a) We really do not need to compute each pixel in G sequentially, and so one way by which the filtering could be sped up is through parallel computing. For example, we can use GPUs for this very reason because of their parallel processing architecture. Most commercial GPUs calculate around 3k $\sim$ 6k pixels concurrently, which means they are 3k $\sim$ 6k times faster than sequential calculation.

(b) A second way to speed up the filtering is if the filter F can be written as an outer product, i.e. $F = ff^{\mathrm{T}}$. If we apply a filter of size $k \times k$ over an input image of size $w \times h$ (single channel), the total number of addition and multiplication operations is $\mathcal{O}(k^2wh)$. If this filter can be expressed as an outer product, the total cost becomes $\mathcal{O}(2kwh)$. Hence the filtering can be sped up if the filter F has this property.

(v) The task is to obtain the vector f if the $k \times k$ filter can be expressed as $F = ff^{\mathrm{T}}$. Let $f = \begin{bmatrix} f_1 & f_2 & \dots & f_k \end{bmatrix}^{\mathrm{T}}$. We thus have

$$F = ff^{\mathrm{T}} \tag{7}$$

$$\implies F = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_k \end{bmatrix} \begin{bmatrix} f_1 & f_2 & \dots & f_k \end{bmatrix} \tag{8}$$

$$\implies F = \begin{bmatrix} f_1^2 & f_1f_2 & \dots & f_1f_k \\ f_2f_1 & f_2^2 & \dots & f_2f_k \\ \vdots & \vdots & \ddots & \vdots \\ f_kf_1 & f_kf_2 & \dots & f_k^2 \end{bmatrix} \tag{9}$$

The LU decomposition of F, which is computed by Gauss elimination, is given by

$$F = LU = \begin{bmatrix} 1 & 0 & \dots & 0 \\ \frac{f_2}{f_1} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \frac{f_k}{f_1} & 0 & \dots & 1 \end{bmatrix} \begin{bmatrix} f_1^2 & f_1f_2 & \dots & f_1f_k \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix} \tag{10}$$

Let the first column of L be denoted by x, and the first row of U be denoted by y. That is,

$$x = \begin{bmatrix} 1 \\ \frac{f_2}{f_1} \\ \vdots \\ \frac{f_k}{f_1} \end{bmatrix}, \qquad y = \begin{bmatrix} f_1^2 & f_1f_2 & \dots & f_1f_k \end{bmatrix} \tag{11}$$

We can clearly observe that

$$x_iy_i = \left(\frac{f_i}{f_1}\right) \cdot f_1f_i = f_i^2 \tag{12}$$

$$\implies f_i = \sqrt{x_iy_i} \tag{13}$$

Equation 13 gives the elements of vector f, and so we can obtain vector f from the LU decomposition

of filter F. Lets work this out for the unnormalized Gaussian filter given by $F = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$. We have

$$F = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\implies x = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}, \quad y = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

$$\implies f_1 = \sqrt{1 \cdot 1} = 1, \quad f_2 = \sqrt{2 \cdot 2} = 2, \quad f_3 = \sqrt{1 \cdot 1} = 1$$

$$\implies f = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

We are able to express the unnormalized Gaussian filter as an outer product

$$F = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} = ff^{\mathrm{T}}$$

Note: Since the filter F has rank 1, the LU decomposition will not be unique. When run in Python and MATLAB, they first do partial pivoting (through a permutation matrix) and then find the LU decomposition (in which case L is the permuted L matrix). However the method discussed above still works. For example, `scipy.linalg.lu(F, permute_l=True)` gives

$$F = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 0.5 & 1 & 0 \\ 1 & 0 & 0 \\ 0.5 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\implies x = \begin{bmatrix} 0.5 \\ 1 \\ 0.5 \end{bmatrix}, \quad y = \begin{bmatrix} 2 & 4 & 2 \end{bmatrix}$$

$$\implies f_1 = \sqrt{0.5 \cdot 2} = 1, \quad f_2 = \sqrt{1 \cdot 4} = 2, \quad f_3 = \sqrt{0.5 \cdot 2} = 1$$

$$\implies f = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

For F to be expressed as an outer product, it has to be symmetric and positive semi-definite.

(vi) Implemented normalized cross-correlation within the `norm_cross_corr` function in `linear_filter.py`.

(vii) For convolution, the output image G is given by

$$G(i,j) = \sum_{u=0}^{k-1} \sum_{v=0}^{l-1} F(u,v) \cdot \bar{I}(i-u+k-1, j-v+l-1) \tag{14}$$

Through a change of variables: $u \to k-u-1$, $v \to l-v-1$, equation 14 can be rewritten as

$$G(i,j) = \sum_{u=0}^{k-1} \sum_{v=0}^{l-1} F(k-u-1, l-v-1) \cdot \bar{I}(i+u, j+v) \tag{15}$$

We observe that convolution with a filter F is equivalent to correlation with a filter $\tilde{F}$ that is filter F flipped in all its dimensions. For example, convolution with a filter $F = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ is equivalent to correlation with filter $\tilde{F} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. Hence convolution with a filter F can be implemented by using correlation with filter F flipped in all its dimensions.