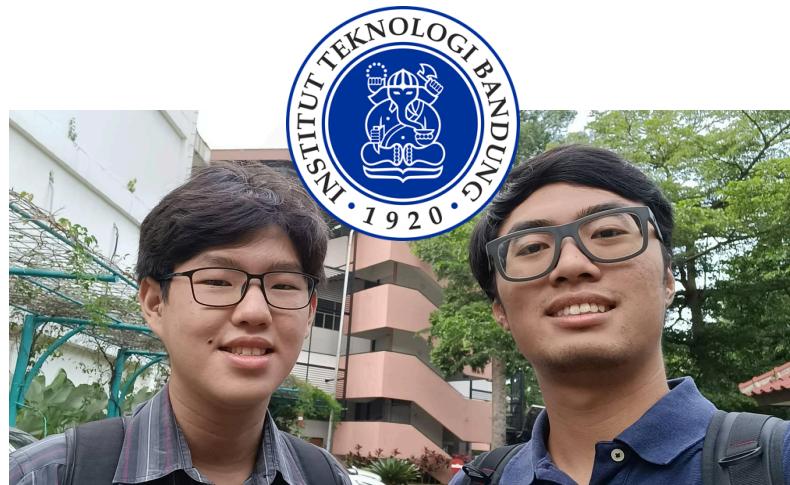


# **PEMANFAATAN KAKAS MATLAB DALAM MELAKUKAN PEMROSESAN SEDERHANA TERHADAP CITRA DALAM RANAH SPASIAL**

## **LAPORAN TUGAS 1**

Disusun untuk memenuhi salah satu tugas mata kuliah IF4073  
Pemrosesan Citra Digital Semester 7 di Institut Teknologi Bandung



Disusun oleh:

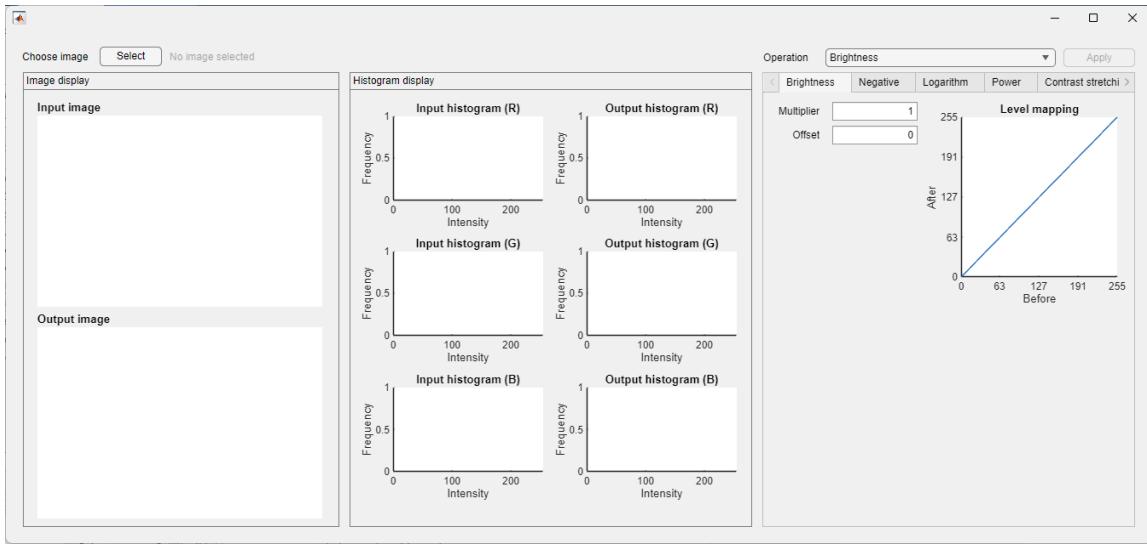
Jericho Russel Sebastian	13521107
Ryan Samuel Chandra	13521140

## **PROGRAM STUDI TEKNIK INFORMATIKA**

Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung  
Jl. Ganesa No.10, Lb. Siliwangi, Kecamatan Coblong,  
Kota Bandung, Jawa Barat 40132

**2024**

## 1. Tutorial GUI Program



Antarmuka dari program terdiri atas tiga panel yang tersusun secara horizontal. Panel pertama berisi tampilan dari citra masukan dan luaran yang dihasilkan oleh program. Pada bagian atas panel, terdapat tombol *Select* untuk memilih file citra masukan yang akan digunakan.

Panel kedua memuat hingga 3 buah histogram yang merepresentasikan sebaran nilai piksel citra masukan untuk masing-masing kanal warna, serta hingga 3 buah histogram yang bersesuaian untuk citra luaran. Histogram citra masukan akan terisi ketika sebuah citra baru dipilih sebagai citra masukan, sedangkan histogram citra luaran akan terisi ketika sebuah operasi pemrosesan citra dilakukan.

Panel ketiga terdiri dari 7 buah *tab* yang berisi parameter-parameter untuk ketujuh operasi pemrosesan citra yang didukung oleh program ini, yaitu: pencerahan, negasi, transformasi logaritmik, transformasi pangkat (*gamma*), peregangan kontras (*contrast stretching*), perataan histogram (*histogram equalization*), dan spesifikasi histogram (*histogram specification*). Operasi dapat dipilih menggunakan menu *drop-down* yang terletak di atas panel. Panel akan menampilkan tab parameter untuk operasi yang dipilih. Sebagian besar operasi memiliki parameter-parameter yang dapat diubah oleh pengguna. Mengklik tombol *Apply* akan memerintahkan program untuk melaksanakan operasi yang dipilih menggunakan nilai parameter yang disediakan. Citra hasil dari operasi tersebut akan ditampilkan pada panel pertama.

Berikut adalah alur dasar pengoperasian program:

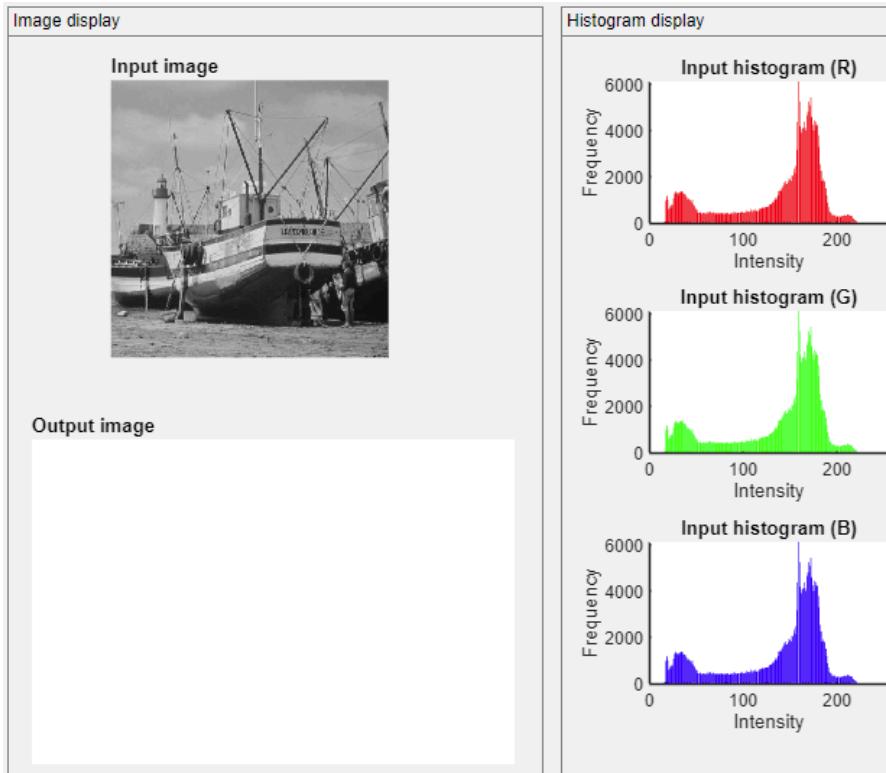
1. Memilih citra masukan dengan mengklik tombol *Select*.
2. Memilih operasi menggunakan menu *drop-down*.
3. Memasukkan nilai-nilai parameter operasi pada panel di sisi kanan.
4. Mengklik tombol *Apply* untuk menjalankan operasi.

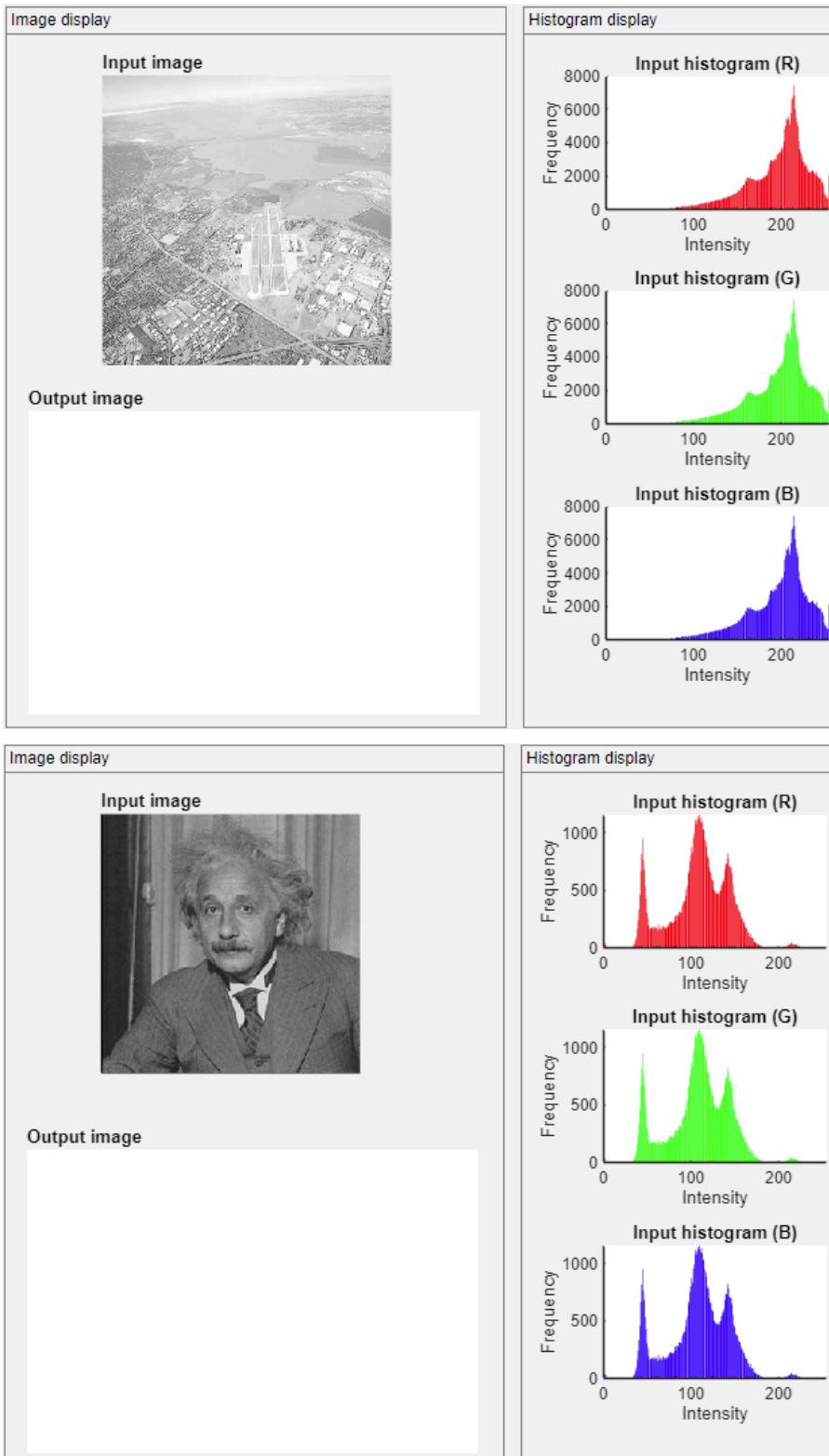
## 2. Penjelasan Kode Program

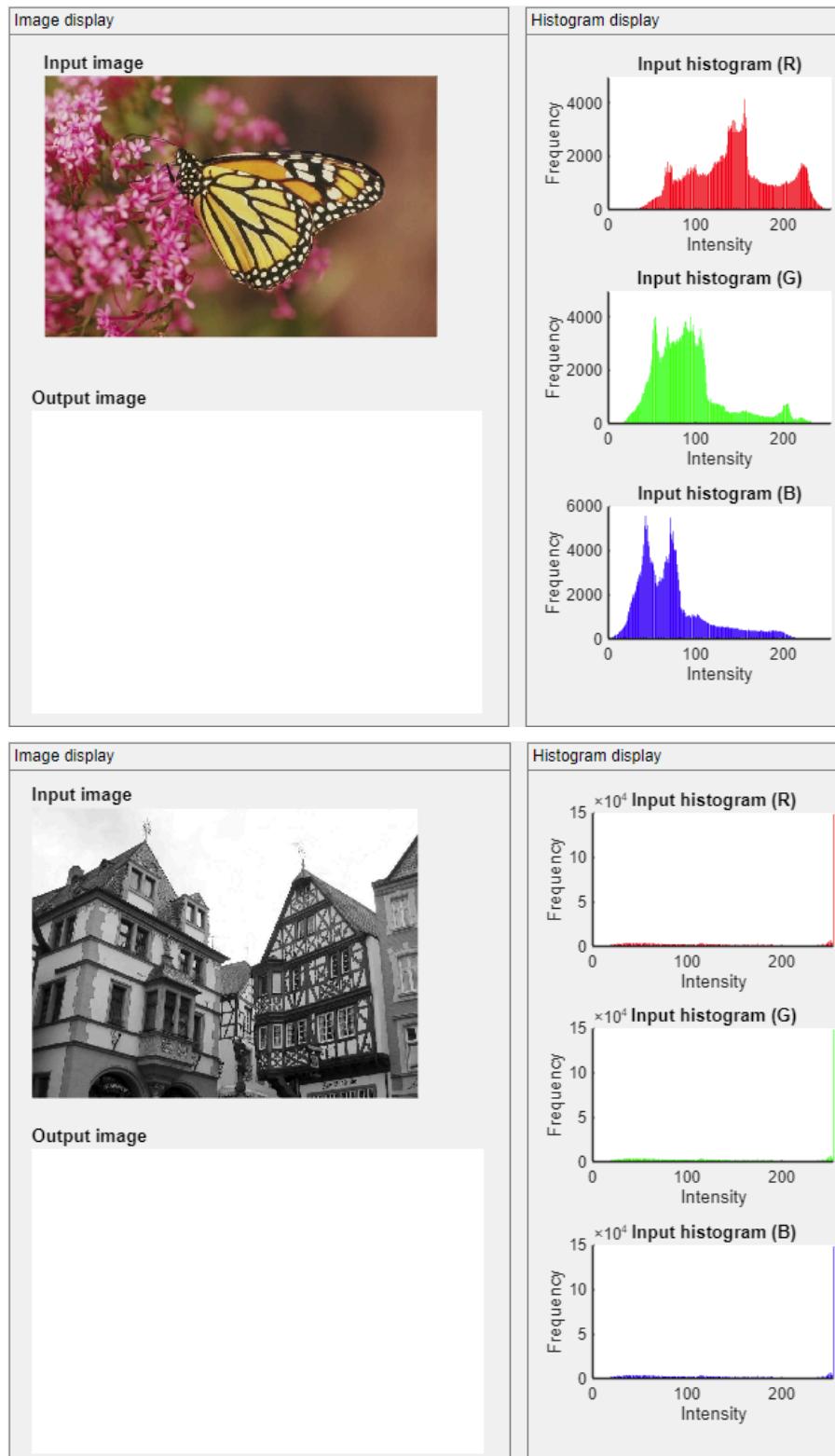
### 2.1 make\_histogram.m

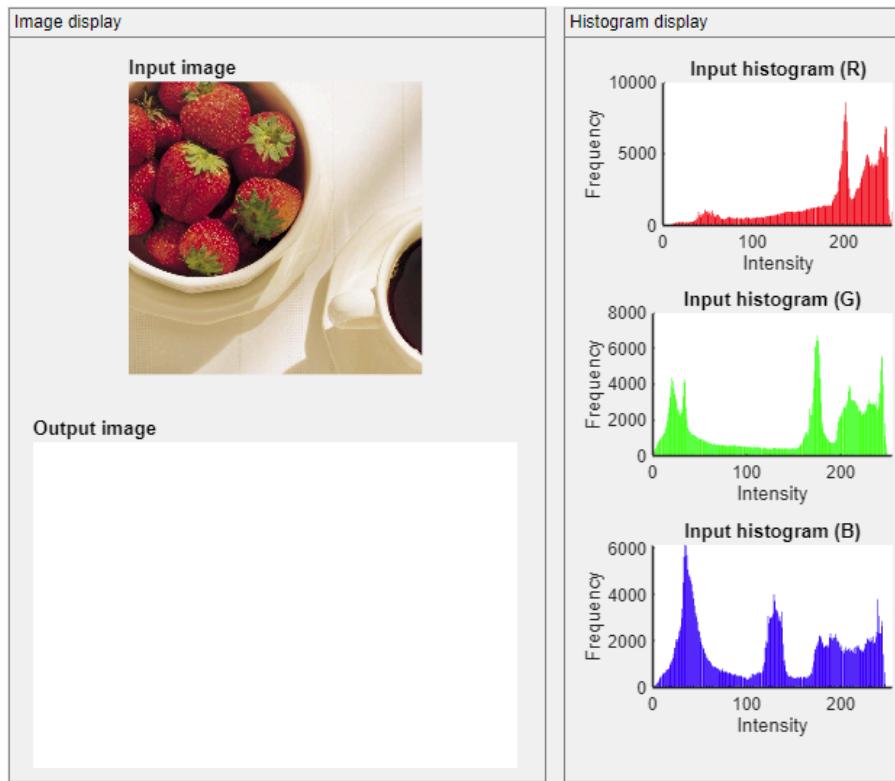
```
make_histogram.m * +  
1 function [hist] = make_histogram(I)  
2 [M, N, C] = size(I);  
3  
4 % Menyiapkan matriks seukuran channel x #pixel_level  
5 hist = zeros(C, 256);  
6  
7 % Iterasi setiap nilai pixel pada I kemudian nilai pada hist pada  
8 % channel yang sesuai, pada indeks sesuai dengan nilai pixel I yang  
9 % sedang disorot, ditambahkan dengan 1  
10 for k = 1 : C  
11     for i = 1 : M  
12         for j = 1 : N  
13             hist(k, (I(i,j,k) + 1)) = hist(k, (I(i,j,k) + 1)) + 1;  
14         end  
15     end  
16 end  
17
```

#### ➤ Contoh Eksekusi

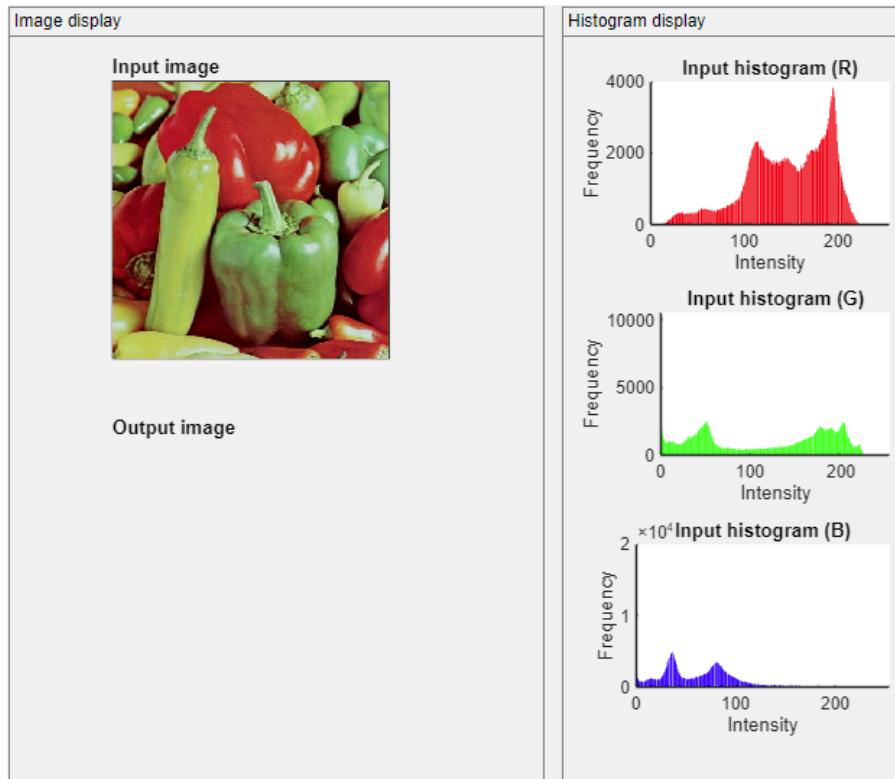


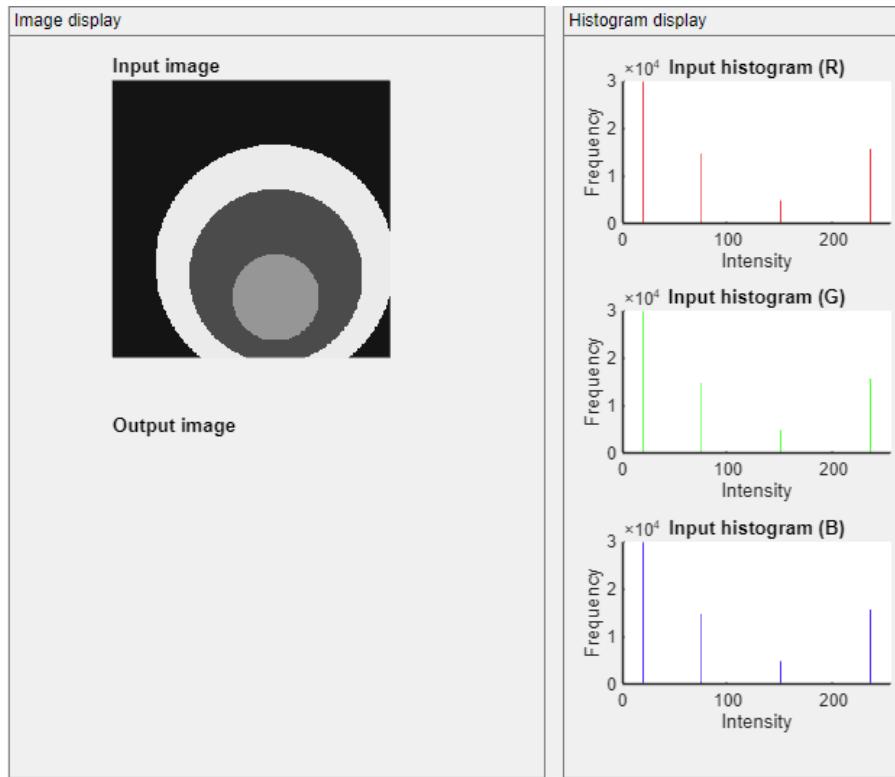






### Tambahan:





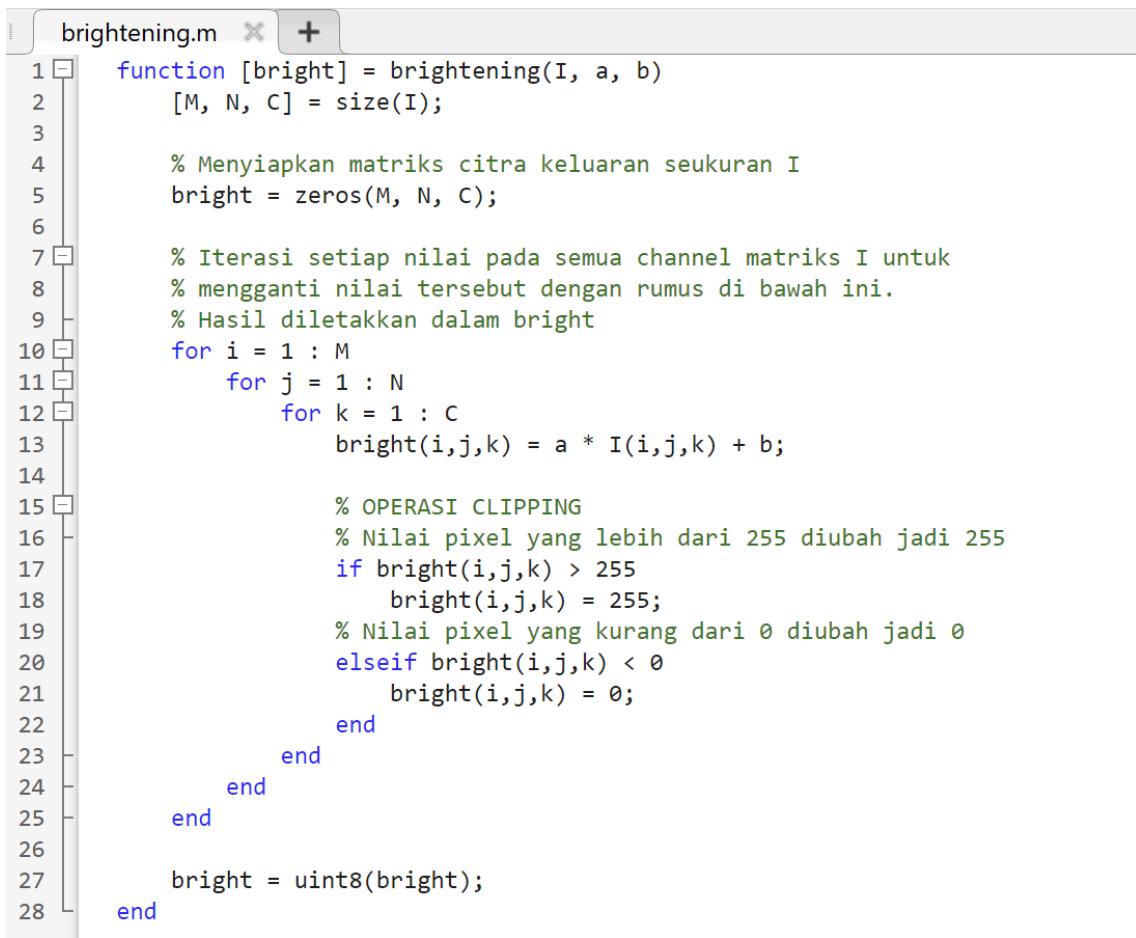
➤ Analisis Algoritma dan Hasilnya

Algoritma pembuatan histogram secara manual bekerja dengan kompleksitas  $O(3n^2) = O(n^2)$ , dengan  $n^2$  adalah jumlah baris  $\times$  jumlah kolom, dan “3” adalah jumlah *channel* terbanyak yang mungkin. Komputer perlu mengiterasi seluruh baris, kolom, dan *channel* untuk membaca satu per satu nilai piksel dalam matriks citra masukan.

Nilai yang dibaca kemudian dijadikan indeks kolom yang nilainya harus ditambahkan satu, pada matriks representasi histogram (*hist*), pada *channel* (*k*) yang sesuai. Terkait *hist(k, I(i, j, k) + 1)*, “+ 1” harus dilakukan karena nilai piksel bervariasi antara 0 sampai dengan 255, sedangkan penggunaan indeks dalam MATLAB dimulai dari 1.

Ternyata, hasil pembuatan histogram manual ini sangat sama dengan histogram yang dibuat menggunakan fungsi `imhist(I)` dari *library*. Ada cara tersendiri untuk menampilkan *hist* sebagai kembalian dari fungsi `make_histogram`, dan langsung dilakukan pada kode GUI.

## 2.2 brightening.m



```
function [bright] = brightening(I, a, b)
    [M, N, C] = size(I);

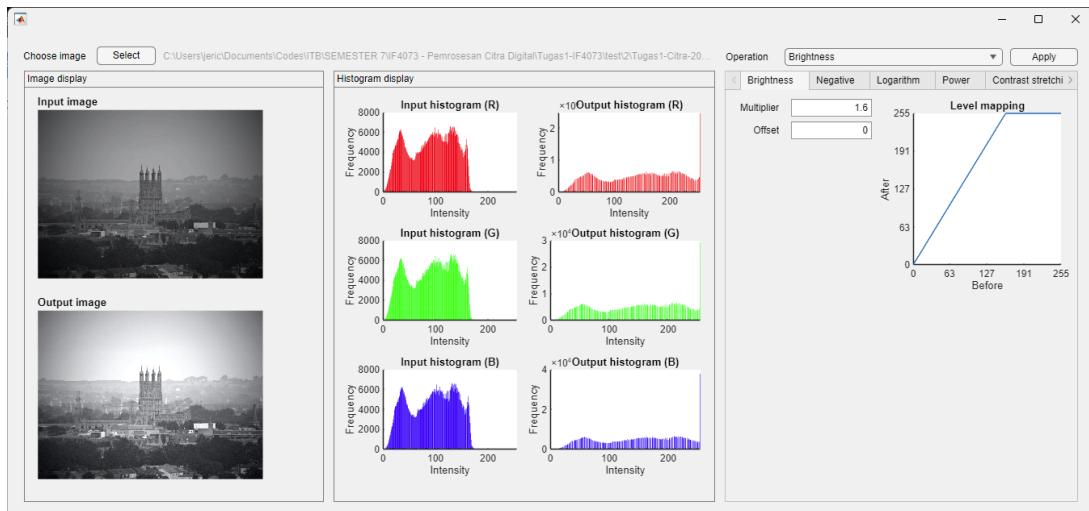
    % Menyiapkan matriks citra keluaran seukuran I
    bright = zeros(M, N, C);

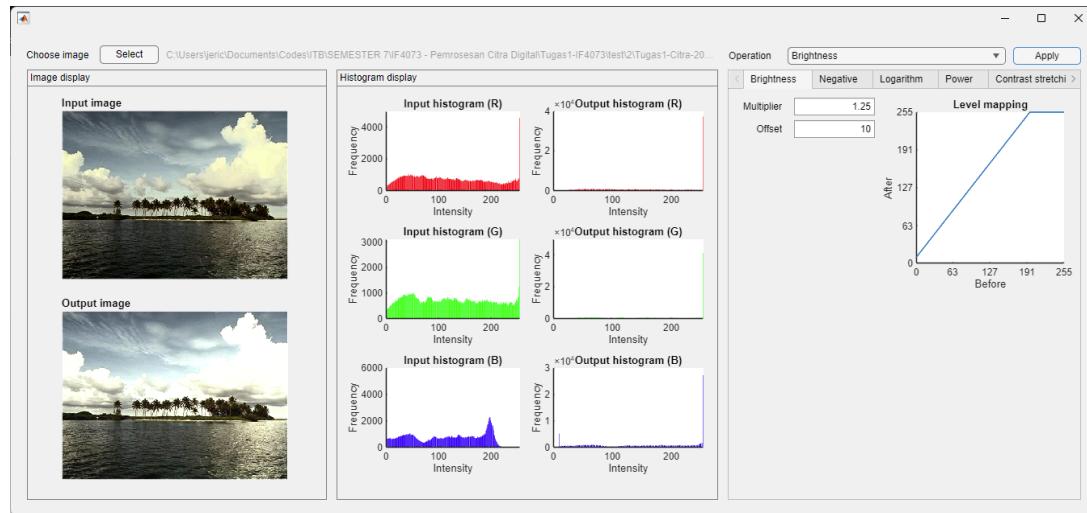
    % Iterasi setiap nilai pada semua channel matriks I untuk
    % mengganti nilai tersebut dengan rumus di bawah ini.
    % Hasil diletakkan dalam bright
    for i = 1 : M
        for j = 1 : N
            for k = 1 : C
                bright(i,j,k) = a * I(i,j,k) + b;

                % OPERASI CLIPPING
                % Nilai pixel yang lebih dari 255 diubah jadi 255
                if bright(i,j,k) > 255
                    bright(i,j,k) = 255;
                % Nilai pixel yang kurang dari 0 diubah jadi 0
                elseif bright(i,j,k) < 0
                    bright(i,j,k) = 0;
                end
            end
        end
    end

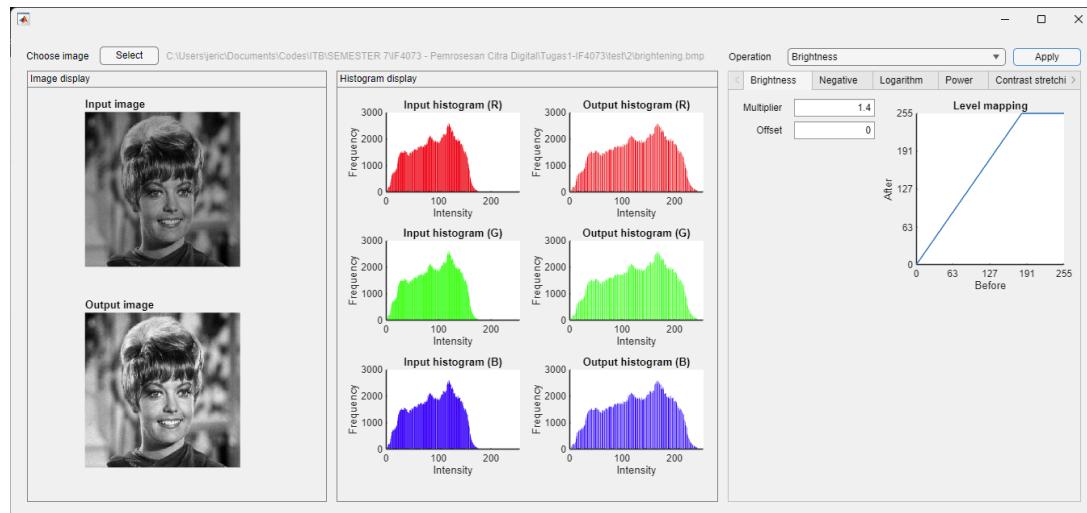
    bright = uint8(bright);
end
```

### ➤ Contoh Eksekusi





## Tambahan:



### ➤ Analisis Algoritma dan Hasilnya

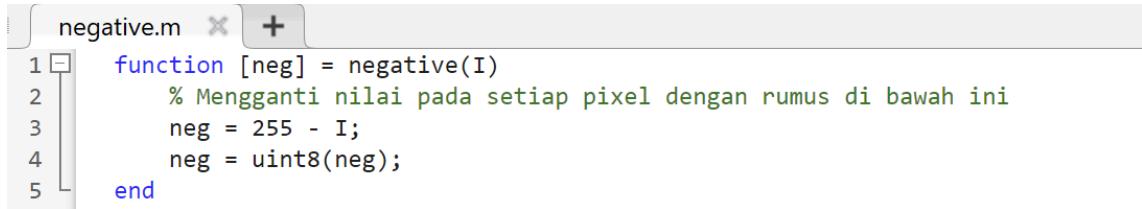
Dengan parameter fungsi berupa  $I$ ,  $a$ , dan  $b$ , komputer membaca satu per satu nilai  $x$  pada matriks citra masukan, kemudian mengubahnya menjadi  $new\_x = a * x + b$  sesuai dengan keinginan pengguna (batasan:  $a \geq 0$ ). Dengan demikian, kompleksitas algoritma ini juga adalah  $O(n^2)$ .

Setelah pengubahan setiap nilai, diperiksa pula apakah nilai akhirnya lebih besar dari 255 atau lebih kecil dari 0. Nilai yang lebih dari 255 dijadikan 255 dan yang lebih kecil dari 0 dijadikan 0. Operasi ini dinamakan *clipping*. *Clipping* perlu dilakukan karena nilai piksel hanya bisa bervariasi antara 0 dan 255.

Citra yang sudah diproses dengan algoritma ini bisa menghasilkan citra yang lebih terang atau lebih gelap, tergantung dengan nilai  $a$  dan  $b$  yang dimasukkan oleh pengguna. Jika nilai-nilai tersebut membuat  $new\_x$  lebih kecil daripada  $x$ , citra

menjadi lebih gelap. Sedangkan jika sebaliknya, citra menjadi lebih terang. Histogram citra akhir mirip dengan histogram citra masukan, hanya saja seperti bergeser ke kanan (jika citra akhir lebih terang), atau ke kiri (citra akhir lebih gelap).

### 2.3 negative.m

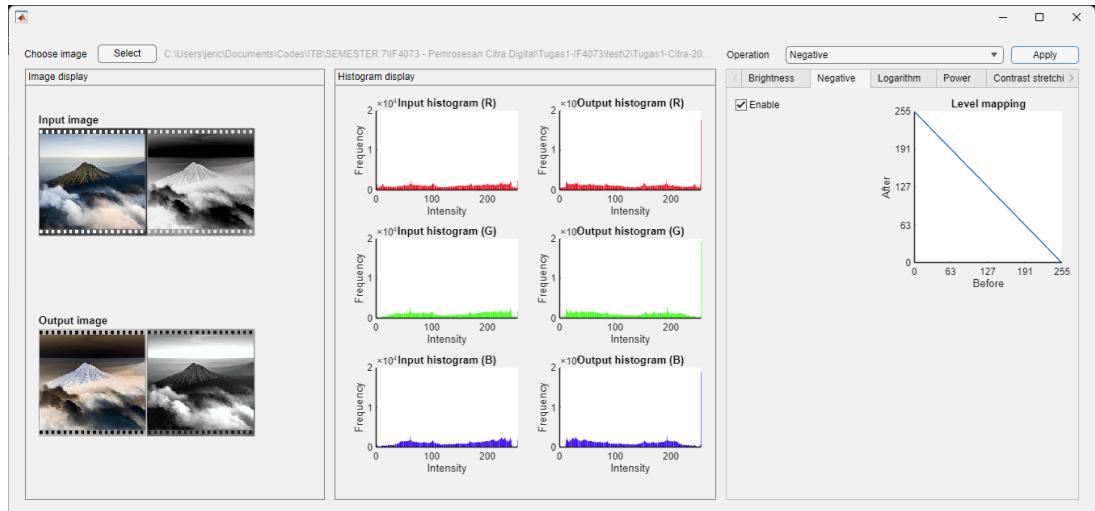


```

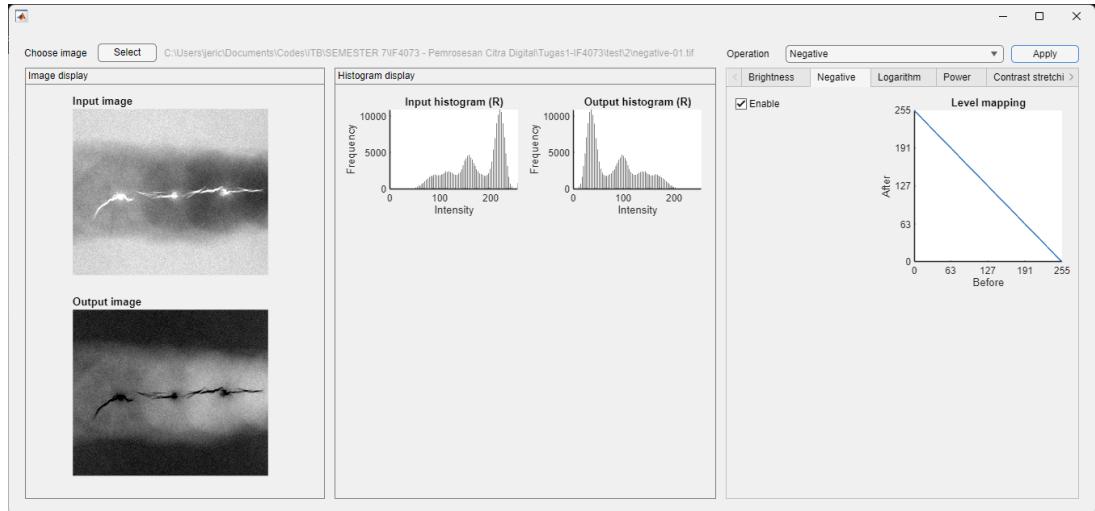
1 function [neg] = negative(I)
2 % Mengganti nilai pada setiap pixel dengan rumus di bawah ini
3 neg = 255 - I;
4 neg = uint8(neg);
5 end

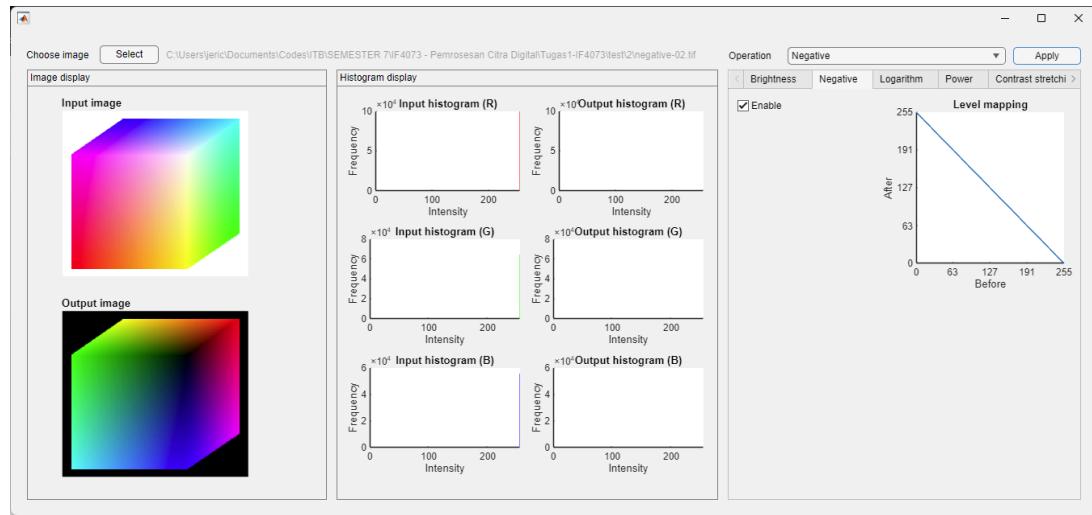
```

#### ➤ Contoh Eksekusi



#### Tambahan:





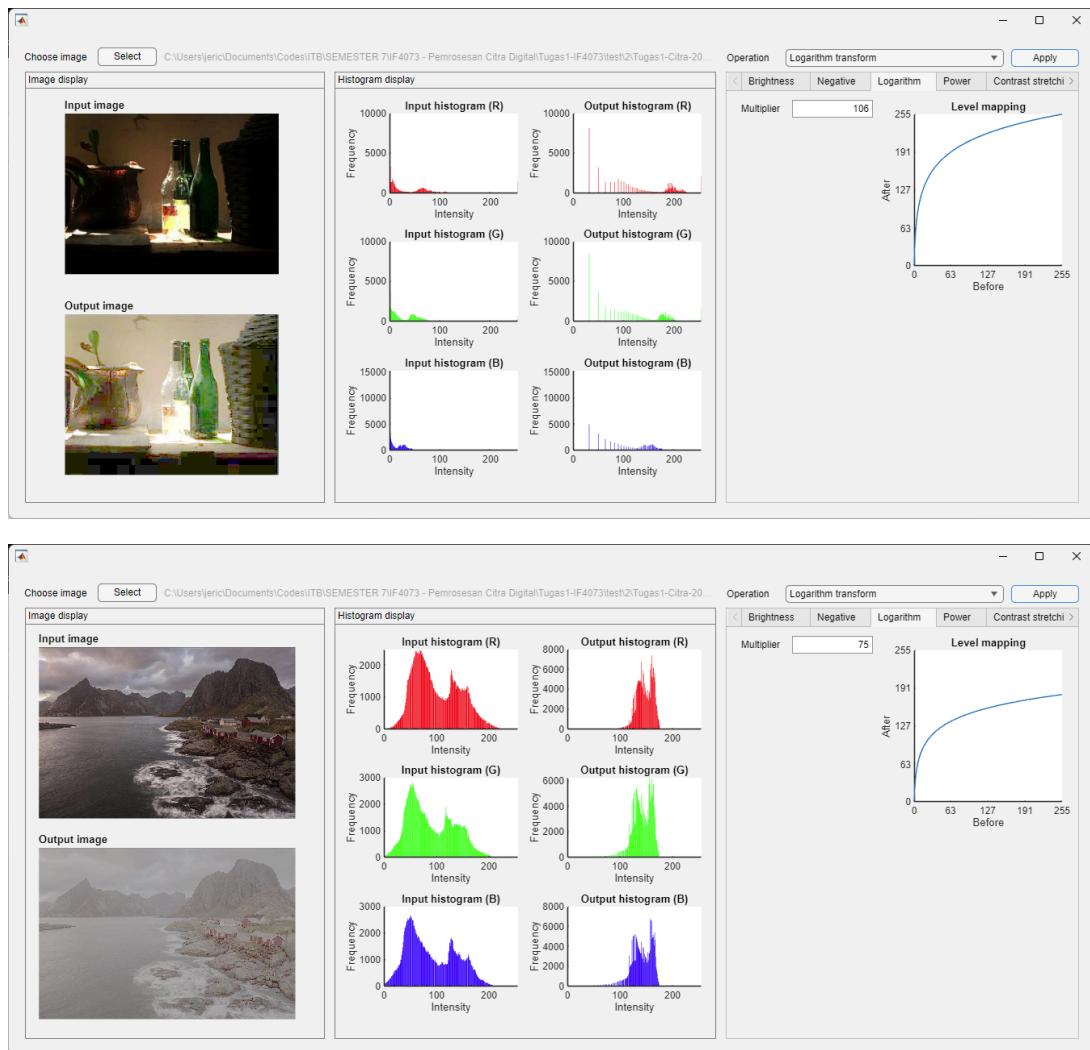
## ➤ Analisis Algoritma dan Hasilnya

Algoritma yang sangat sederhana dengan kompleksitas tetap  $O(n^2)$ . Seluruh nilai  $x$  dalam matriks citra masukan langsung diubah menjadi  $255 - x$ . Citra normal yang diproses dengan algoritma ini menghasilkan citra yang memiliki efek “seram”, sebab seluruh nilai dibalik sehingga yang tinggi menjadi rendah dan yang rendah menjadi tinggi. Histogram citra akhir seperti histogram citra semula yang dicerminkan (direfleksikan terhadap cermin vertikal).

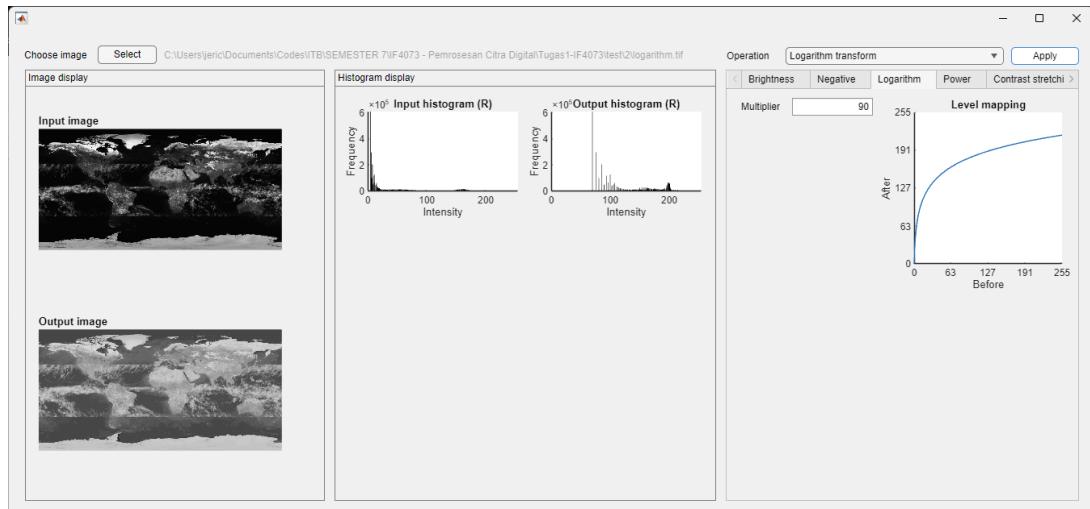
## 2.4 logarithm.m

```
logarithm.m x +
1 function [log_transformed] = logarithm(I, c)
2 % Mengganti nilai setiap pixel dengan rumus di bawah ini
3 % Hasil diletakkan dalam log_transformed
4 log_transformed = c * log10(double(1 + I));
5 log_transformed = uint8(log_transformed);
6 end
```

## ➤ Contoh Eksekusi



## Tambahan:

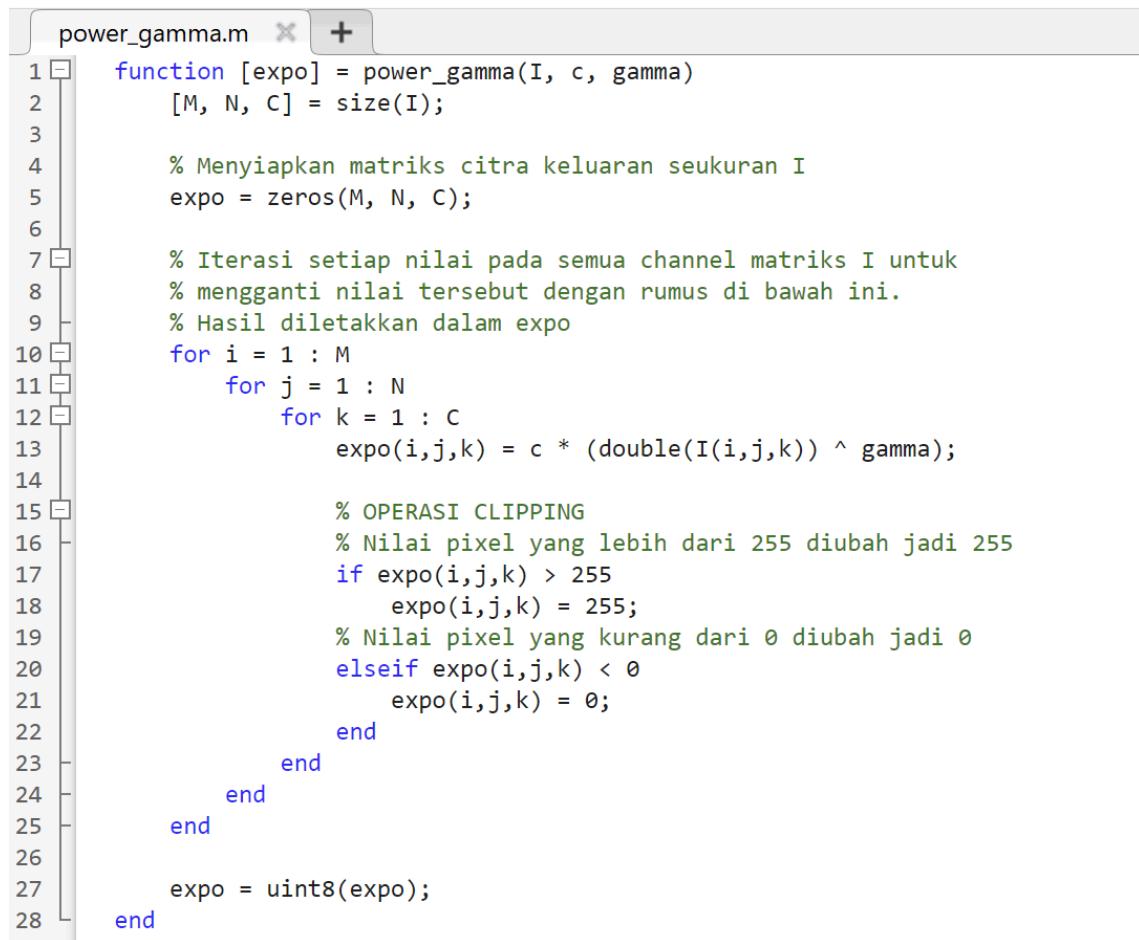


## ➤ Analisis Algoritma dan Hasilnya

Algoritma transformasi logaritma juga bekerja dengan kompleksitas  $O(n^2)$ . Mirip seperti *brightening* maupun *negative*, nilai piksel  $x$  pada matriks citra masukan langsung diubah semua menjadi  $new\_x = c * 10\log(1 + x)$ , dengan  $c$  masukan bebas dari pengguna (batasan  $c \geq 0$ ). Setelah melalui berbagai percobaan, nilai *default*  $c$  adalah 106, karena masih menghasilkan citra *output* yang nilai pikselnya bervariasi antara 0 dan 255 (tidak rata, tetapi cenderung ada perwakilan dari setiap nilai).

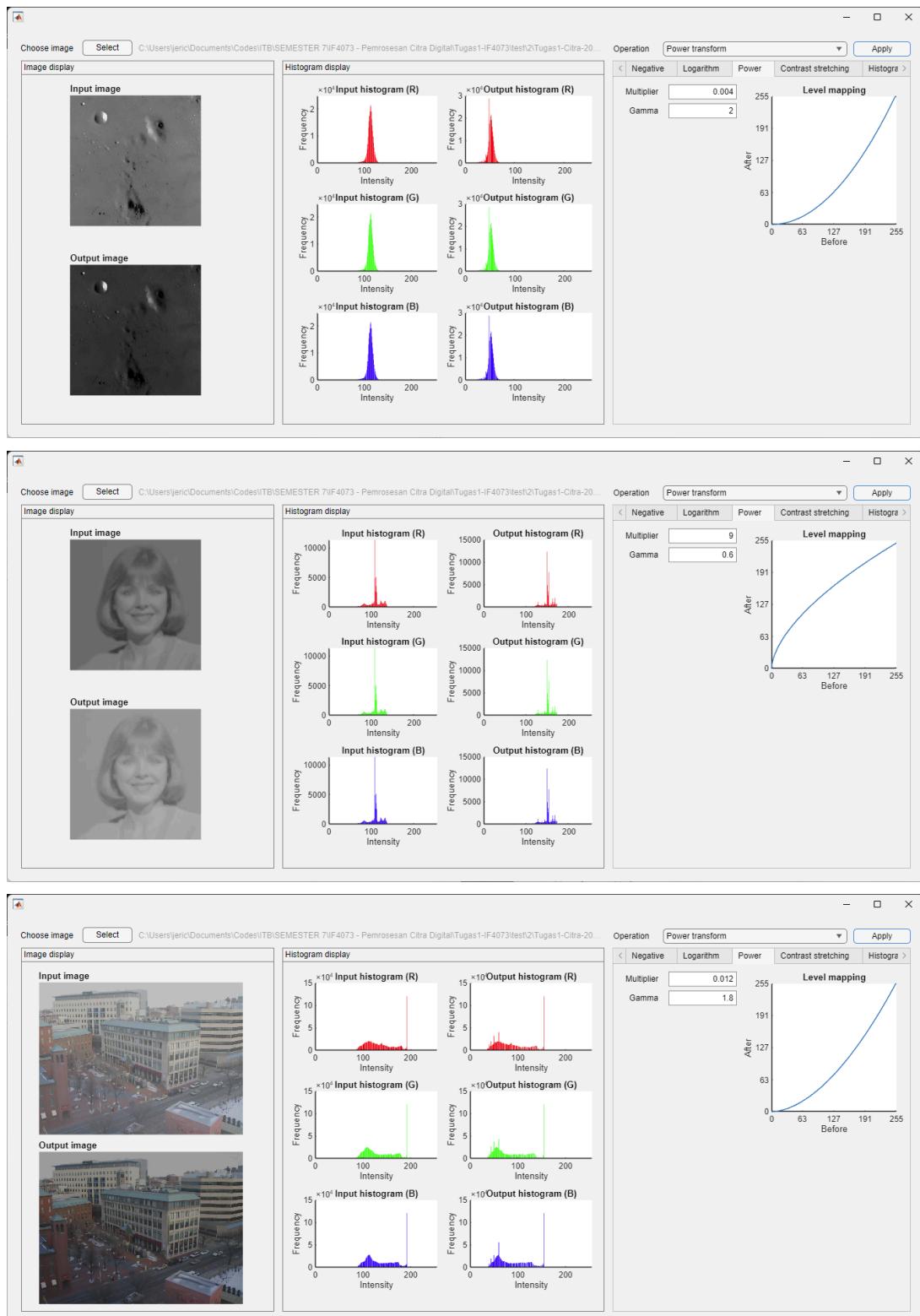
Citra yang diproses dengan algoritma ini menghasilkan citra yang histogramnya lebih sempit, apabila  $c$  yang digunakan lebih kecil dari 106. Ini disebabkan karena nilai piksel rendah akan dipetakan menjadi jauh lebih tinggi, sedangkan yang sudah tinggi hanya dipetakan menjadi sedikit lebih tinggi. Akibatnya, citra akhir tampak memiliki kontras yang jauh lebih rendah.

## 2.5 power\_gamma.m

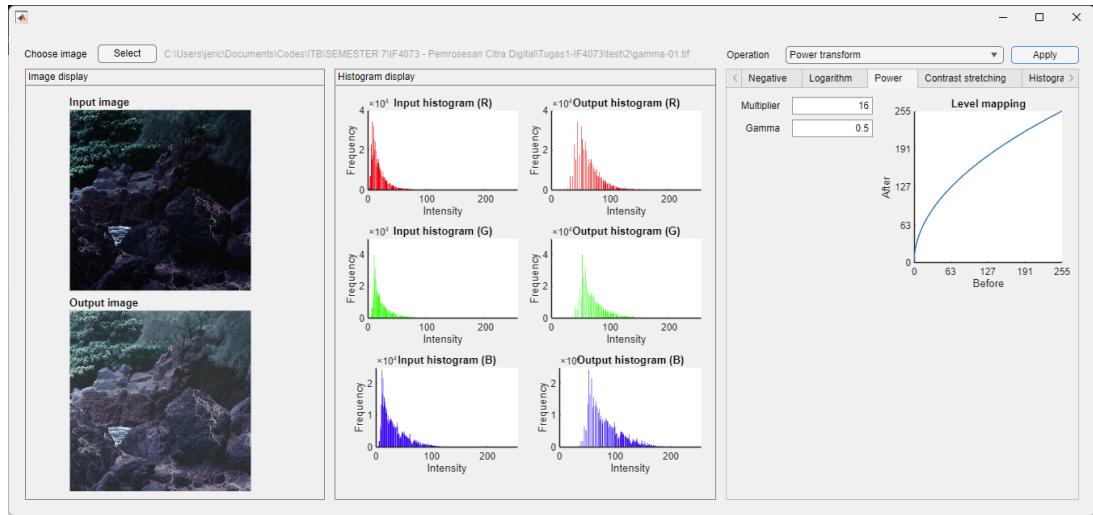


```
power_gamma.m +  
1 function [expo] = power_gamma(I, c, gamma)  
2 [M, N, C] = size(I);  
3  
4 % Menyiapkan matriks citra keluaran seukuran I  
5 expo = zeros(M, N, C);  
6  
7 % Iterasi setiap nilai pada semua channel matriks I untuk  
8 % mengganti nilai tersebut dengan rumus di bawah ini.  
9 % Hasil diletakkan dalam expo  
10 for i = 1 : M  
11     for j = 1 : N  
12         for k = 1 : C  
13             expo(i,j,k) = c * (double(I(i,j,k)) ^ gamma);  
14  
15             % OPERASI CLIPPING  
16             % Nilai pixel yang lebih dari 255 diubah jadi 255  
17             if expo(i,j,k) > 255  
18                 expo(i,j,k) = 255;  
19             % Nilai pixel yang kurang dari 0 diubah jadi 0  
20             elseif expo(i,j,k) < 0  
21                 expo(i,j,k) = 0;  
22             end  
23         end  
24     end  
25 end  
26  
27 expo = uint8(expo);  
28 end
```

## ➤ Contoh Eksekusi



## Tambahan:



### ➤ Analisis Algoritma dan Hasilnya

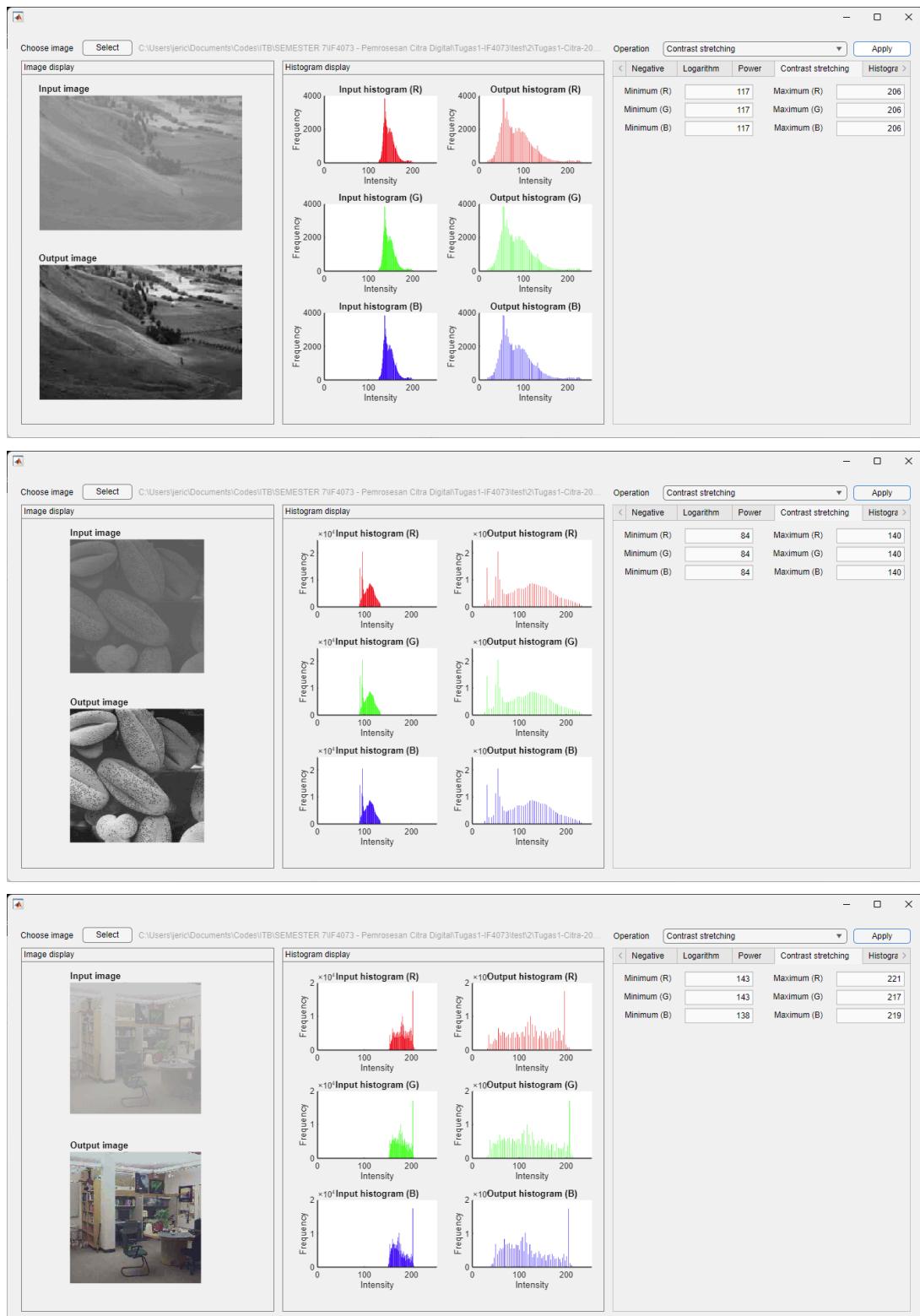
Berbeda hanya pada satu baris dengan *brightening.m*, algoritma transformasi pangkat ini pun memiliki kompleksitas  $O(n^2)$ . Bedanya, setiap nilai  $x$  pada matriks citra masukan diubah menjadi  $new\_x = c * (x^{gamma})$ . Operasi *clipping* perlu dilakukan agar seluruh nilai piksel citra akhir masuk dalam batas nilai yang mungkin.

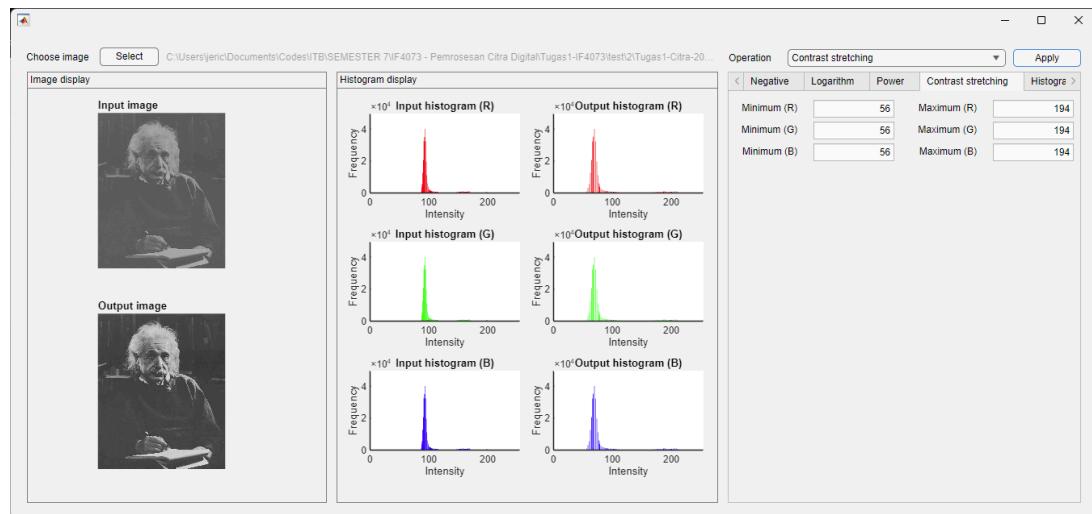
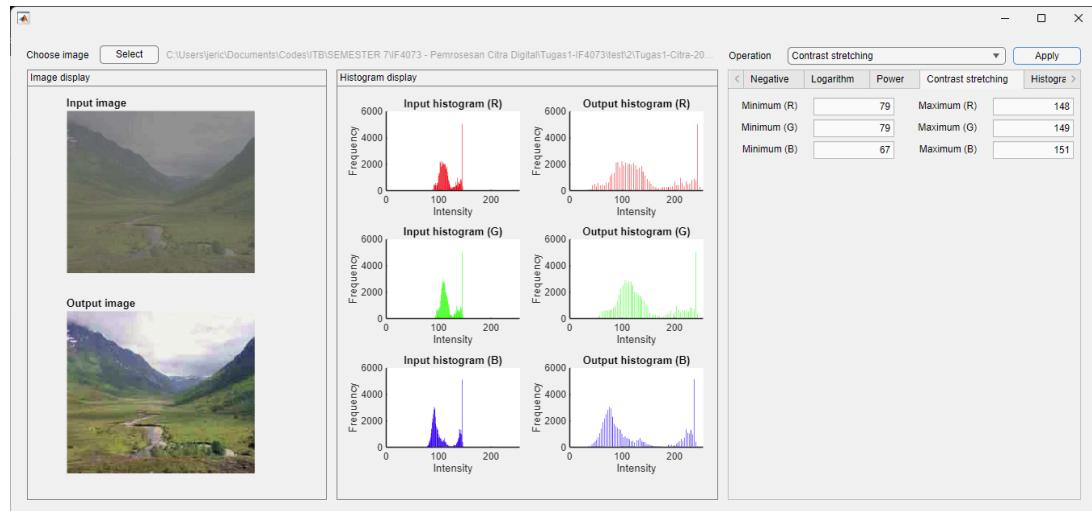
Transformasi pangkat mengemulasikan fenomena hukum pangkat (*power law*) yang menghasilkan respons *gamma* yang ditampilkan oleh banyak perangkat yang menampilkan citra. Secara khusus, transformasi pangkat dengan nilai *gamma* di bawah 1 dikenal sebagai pemampatan *gamma* (*gamma compression*), yang menguatkan nilai piksel rendah sesuai dengan hukum pangkat. Pemampatan *gamma* digunakan untuk menganulir respons *gamma* dari monitor CRT. Sedangkan, transformasi pangkat dengan nilai *gamma* di atas 1 dikenal sebagai perluasan *gamma* (*gamma expansion*), yang menekan piksel-piksel bernilai tinggi.

## 2.6 contrast\_stretching.m

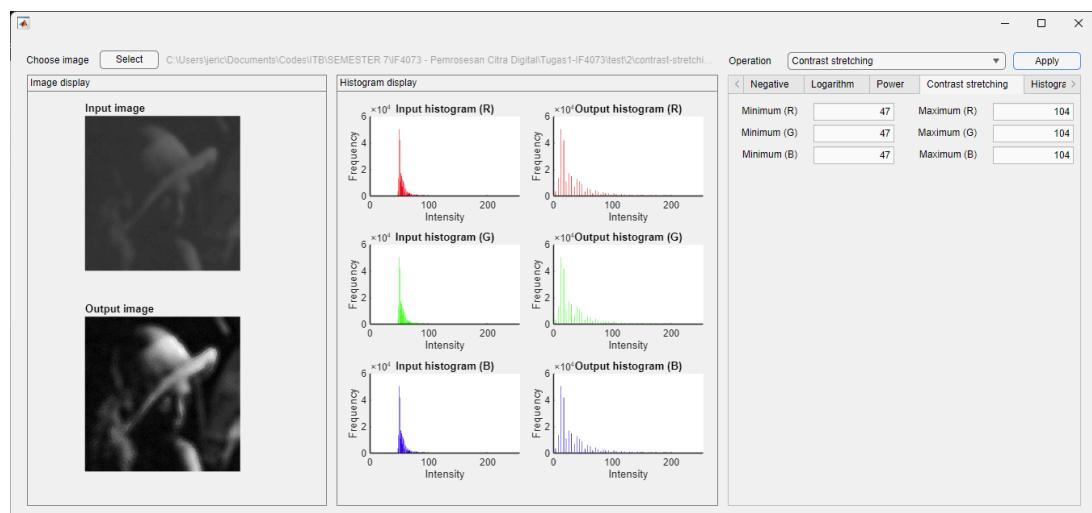
```
contrast_stretching.m +  
1 function [new_I, minValue, maxValue] = contrast_stretching(I)  
2 % Membuat histogram dari I  
3 hist = make_histogram(I);  
4  
5 % Mempersiapkan matriks citra keluaran seukuran I  
6 [M, N, C] = size(I);  
7 new_I = zeros(M, N, C);  
8 minValue = zeros(C);  
9 maxValue = zeros(C);  
10  
11 % ITERASI CHANNEL  
12 for k = 1 : C  
13     % Menemukan nilai pixel maksimum dan minimum  
14     % di setiap channel, yang jumlahnya != 0  
15     minValue(k) = 0;  
16     maxValue(k) = 0;  
17  
18     for j = 1 : size(hist,2)  
19         if hist(k,j) > 0  
20             minValue(k) = j-1;  
21             break;  
22         end  
23     end  
24  
25     for j = size(hist,2) : -1 : 1  
26         if hist(k,j) > 0  
27             maxValue(k) = j-1;  
28             break;  
29         end  
30     end  
31  
32     % Iterasi baris dan kolom untuk mengganti nilai pada I  
33     % dengan nilai yang didapat dari rumus di bawah ini  
34     % Hasil disimpan di new_I  
35     for i = 1 : M  
36         for j = 1 : N  
37             new_I(i,j,k) = 255 * (double(I(i,j,k)) - minValue(k)) / (maxValue(k) - minValue(k));  
38         end  
39     end  
40  
41     new_I = uint8(new_I);  
42  
43 end
```

## ➤ Contoh Eksekusi





## Tambahan:



➤ Analisis Algoritma dan Hasilnya

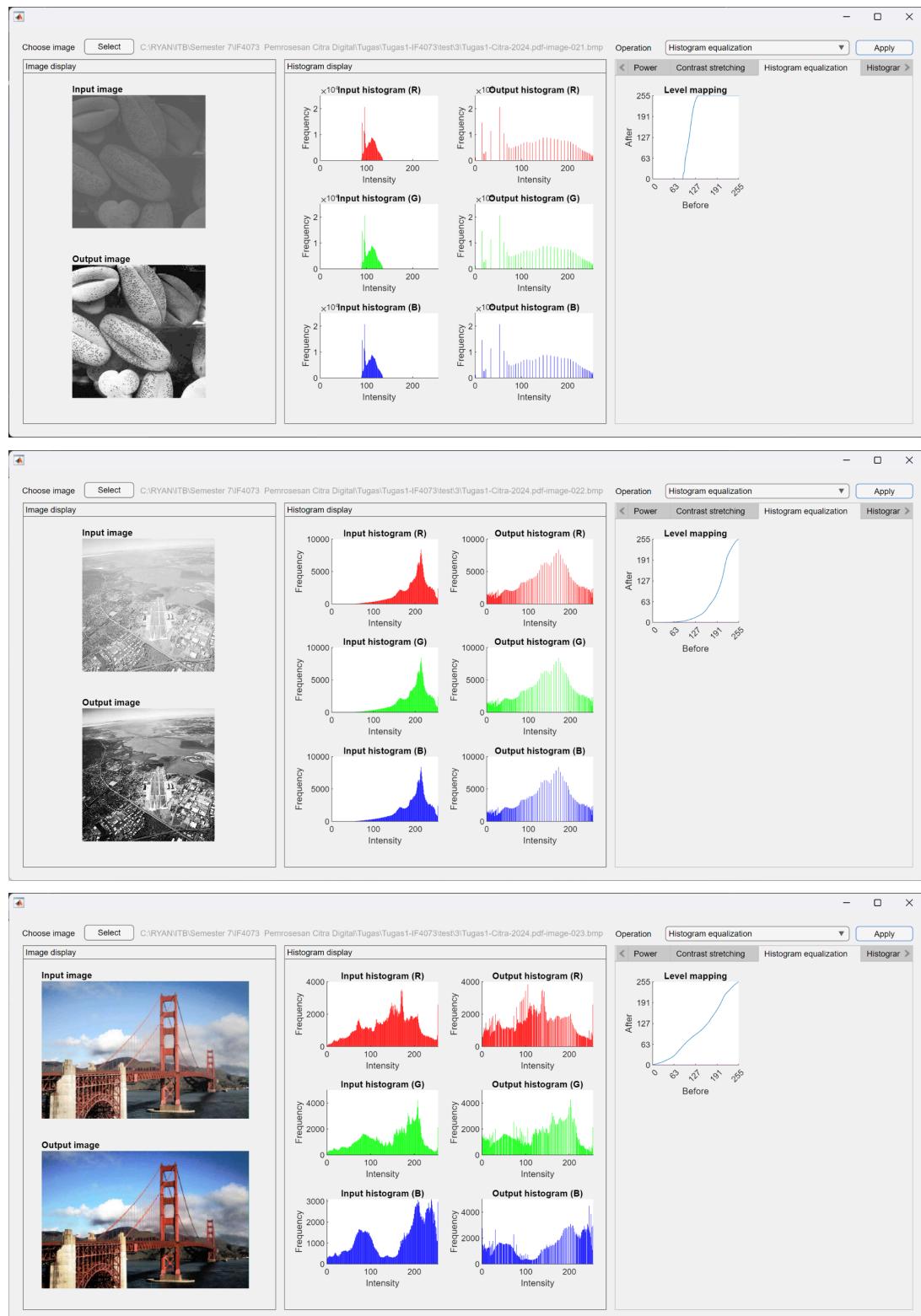
Algoritma ini dimulai dengan mencari nilai piksel terbesar ( $r_{max}$ ) dan terkecil ( $r_{min}$ ) yang jumlahnya tidak 0 pada setiap *channel* citra masukan. Setelah itu, dibuat persamaan garis lurus antara  $[r_{min}, 0]$  dan  $[r_{max}, 255]$ , dengan maksud memetakan  $r_{min}$  menjadi 0 dan  $r_{max}$  menjadi 255. Lalu, nilai lain yang ada di antara 0 dan 255 dapat dimasukkan ke dalam persamaan garis yang terbentuk, sehingga terpetakan menjadi sebuah nilai lain di antara 0 dan 255 juga.

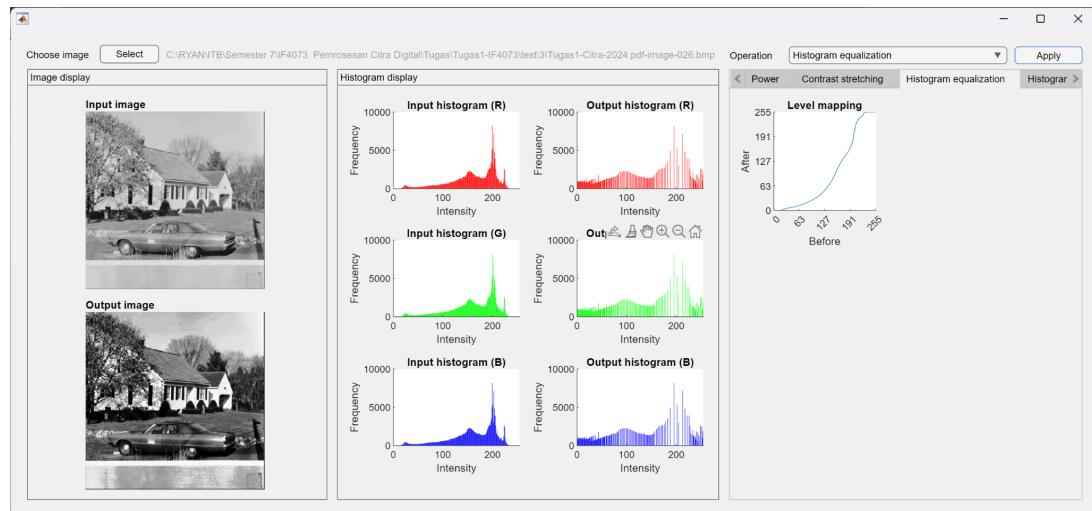
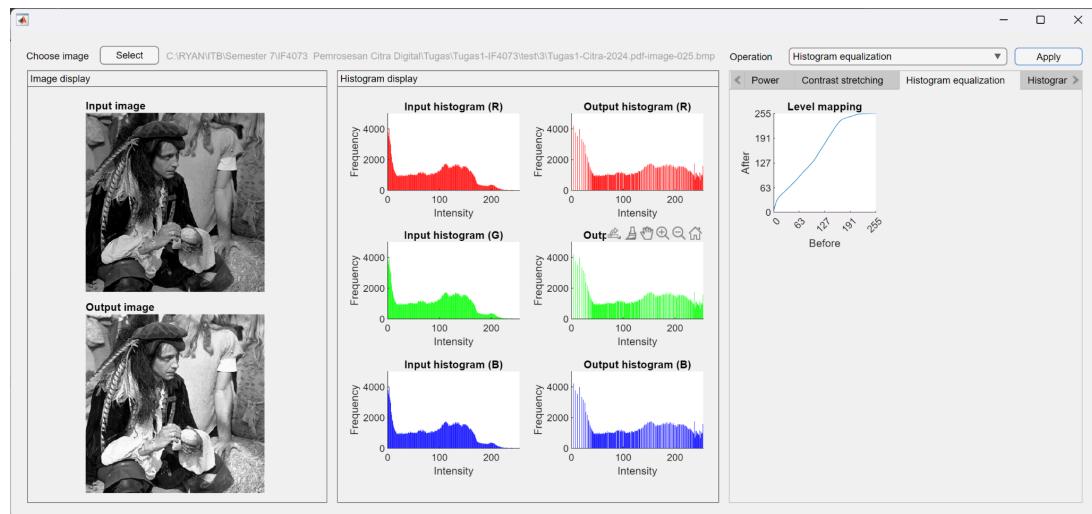
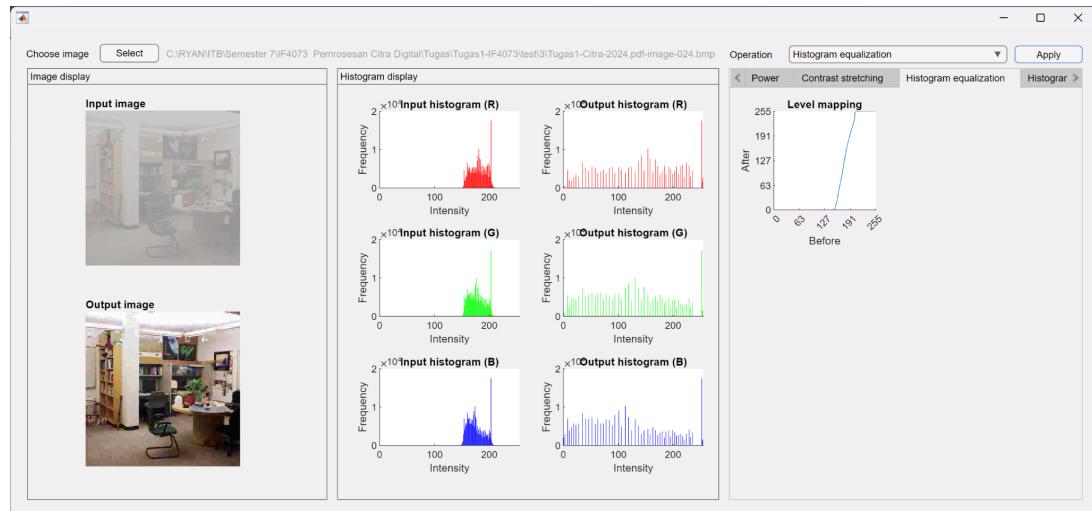
Dengan penjelasan demikian, dapat disimpulkan bahwa kompleksitas algoritma *contrast stretching* ini adalah  $O(2n^2 + n^2) = O(n^2)$ . Citra yang dihasilkan bukan hanya memiliki puncak histogram yang lebih lebar, tetapi juga pewarnaan yang lebih bagus karena nilai piksel membentang antara 0 sampai dengan 255.

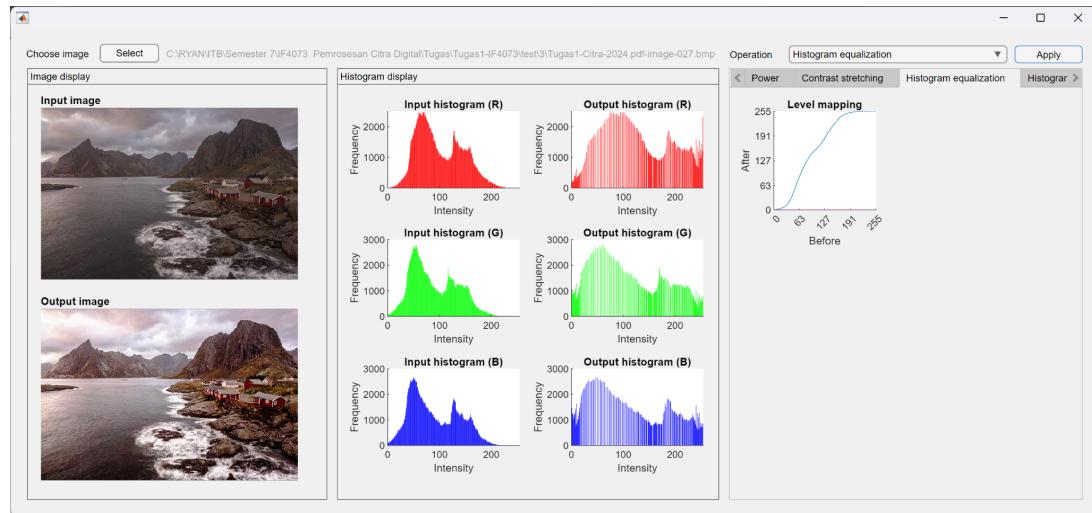
## 2.7 hist\_equalization.m

```
hist_equalization.m +  
1 function [new_I, change_list] = hist_equalization(I, round)  
2     hist = make_histogram(I);  
3  
4     [M, N, C] = size(I);  
5  
6     % Menyiapkan change_list dengan 1 baris dan 256 kolom  
7     change_list = zeros(size(hist,2));  
8  
9     % ITERASI UNTUK MENGISI CHANGE_LIST  
10    for j = 1 : size(hist, 2)  
11        if j == 1  
12            % Untuk indeks 1, bagi {rerata semua kolom pada indeks yang  
13            % bersesuaian (j) pada histogram} dengan {ukuran citra}  
14            change_list(j) = mean(hist(:, j)) / (M*N);  
15        else  
16            % Indeks lainnya, tambahkan nilai pada indeks sebelumnya dengan {rerata  
17            % semua kolom dengan indeks j pada histogram} dibagi {ukuran citra}  
18            change_list(j) = change_list(j - 1) + mean(hist(:, j)) / (M*N);  
19  
20            % Mengalikan nilai pada indeks sebelumnya dengan  
21            % nilai piksel maksimum yang mungkin, yaitu 255  
22            change_list(j - 1) = change_list(j - 1) * 255;  
23  
24            % MEMBEDAKAN ANTARA HASIL PERKALIAN YANG DI-ROUND DAN TIDAK  
25            % Jika perlu di-round (citra semula), diterapkan fungsi floor  
26            % ke nilai pada indeks sebelumnya di change_list.  
27            if round == 1  
28                change_list(j - 1) = floor(change_list(j - 1));  
29            end  
30  
31            % Menerapkan perkalian dengan 255 untuk nilai pada indeks terakhir  
32            if j == size(hist, 2)  
33                change_list(j) = change_list(j) * 255;  
34  
35            % Menerapkan fungsi floor (jika round==1) untuk nilai pada indeks terakhir  
36            if round == 1  
37                change_list(j) = floor(change_list(j));  
38            end  
39            end  
40        end  
41    end  
42  
43    % MEMBUAT CITRA YANG HISTOGRAMNYA DIRATAKAN  
44    % Menyiapkan matriks kosong sesuai dengan ukuran I  
45    new_I = zeros(M, N, C);  
46  
47    % Mengganti setiap nilai pixel pada I dengan di change_list di channel  
48    % yang sesuai, pada indeks sesuai dengan nilai pixel tersebut  
49    for k = 1 : C  
50        for i = 1 : M  
51            for j = 1 : N  
52                new_I(i,j,k) = change_list(I(i,j,k)+1);  
53  
54                % OPERASI CLIPPING  
55                % Nilai pixel yang lebih dari 255 diubah jadi 255  
56                if new_I(i,j,k) > 255  
57                    new_I(i,j,k) = 255;  
58                % Nilai pixel yang kurang dari 0 diubah jadi 0  
59                elseif new_I(i,j,k) < 0  
60                    new_I(i,j,k) = 0;  
61                end  
62            end  
63        end  
64    end  
65  
66    new_I = uint8(new_I);  
67
```

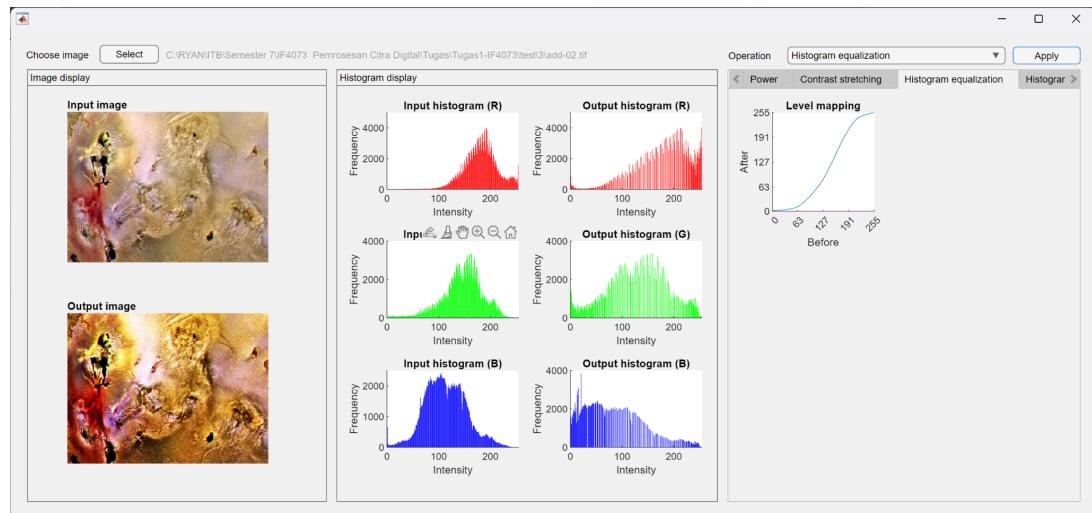
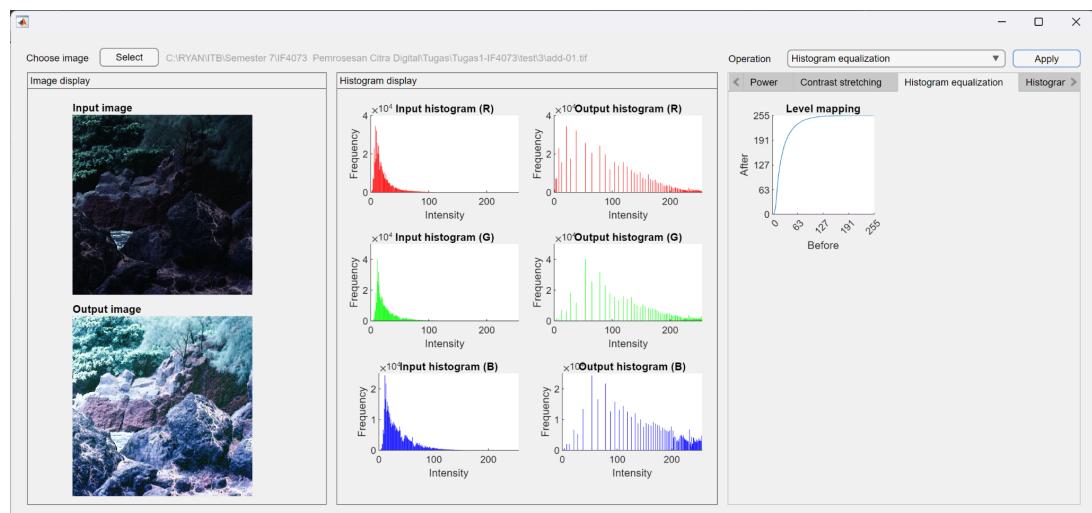
## ➤ Contoh Eksekusi







## Tambahan:



➤ Analisis Algoritma dan Hasilnya

Perataan histogram dapat dilakukan terhadap citra *grayscale* (1 kanal) maupun berwarna (3 kanal). Mulanya, disiapkan sebuah larik bernama `change_list` dengan ukuran  $1 \times 256$ , tidak peduli jumlah kanal dari citra masukan. Larik ini digunakan untuk menampung nilai perubahan; misalnya pada indeks 1 nilainya adalah 100, maka semua nilai 1 pada posisi  $[u,v]$  pada matriks citra masukan diubah menjadi 100 dan disimpan pada posisi yang sama di matriks citra hasil.

Algoritma untuk mengisi `change_list` dapat dilihat pada kode program di atas. Secara umum, `change_list(1)` diisi dengan  $hist(1) / (M \times N)$ . Kemudian untuk indeks lainnya ( $j$ ),

```
change_list(j) = change_list(j-1) + (hist(j) / (M*N)).
```

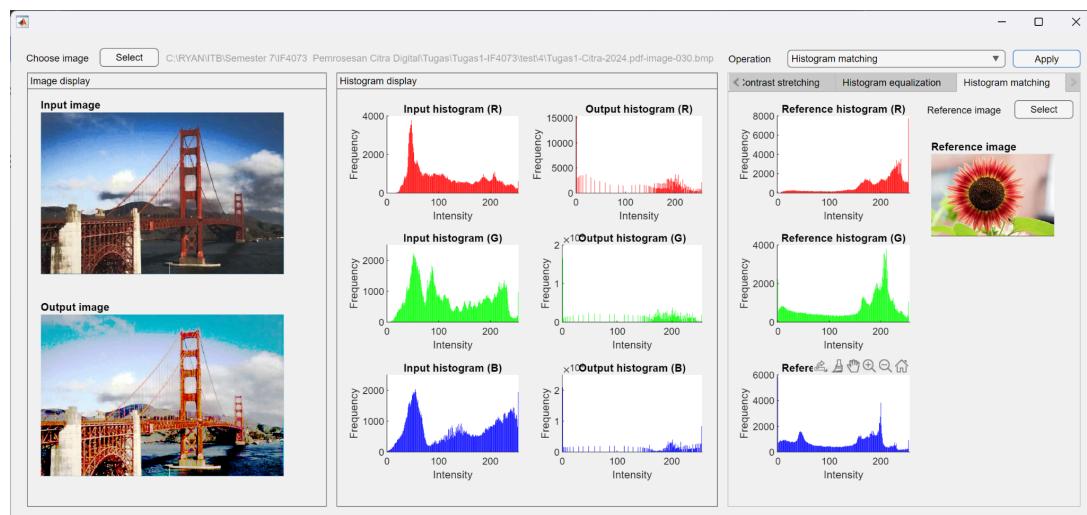
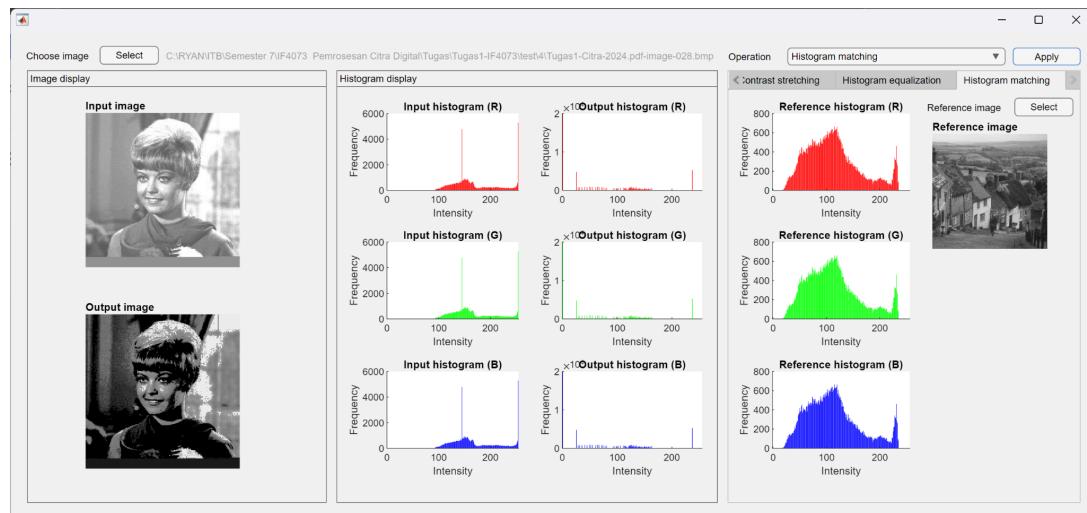
Khusus untuk citra berwarna, `change_list` tetap hanya satu baris, dan nilai `hist(j)` yang digunakan adalah rata-rata nilai `hist(j)` dari ketiga kanal. Alasannya sederhana, yaitu perataan histogram tidak dapat dilakukan terpisah terhadap masing-masing kanal, karena variasai nilainya berbeda-beda. Sebaliknya, perataan harus dilakukan terhadap keseluruhan citra sekaligus.

Citra yang dihasilkan dari algoritma ini lebih kaya akan warna daripada citra semula. Hal ini disebabkan karena citra keluaran memiliki rentang piksel yang lebih lebar, dan frekuensi setiap piksel relatif sama.

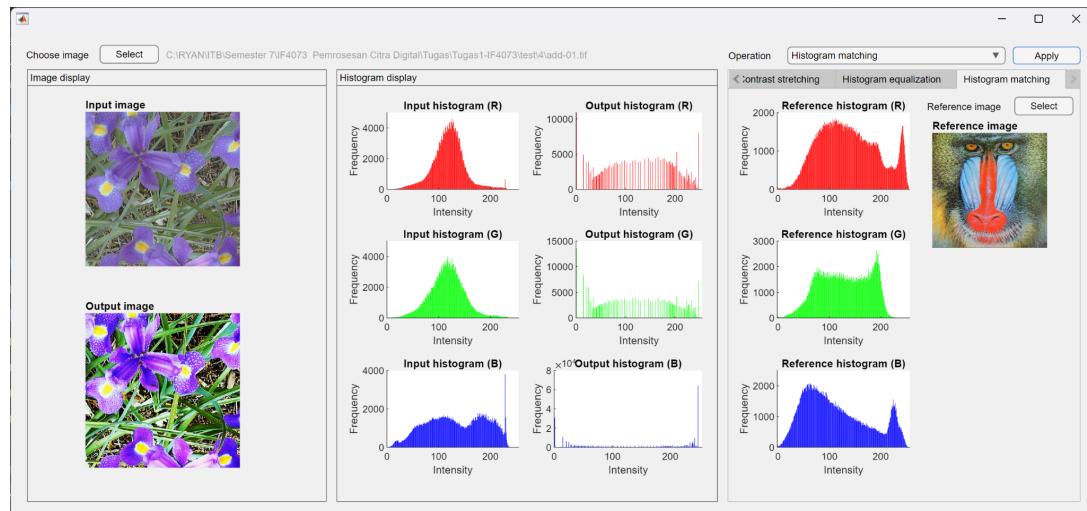
## 2.8 hist\_specification.m

```
hist_specification.m  make_histogram.m  brightening.m  +
1 function [final_I] = hist_specification(I1, I2)
2 % final_I = imhistmatch(I1, I2);
3
4 % PERATAAN HISTOGRAM UNTUK KEDUA CITRA
5 % change1 adalah sebuah larik yang menunjukkan nilai pixel pada I1 harus
6 % diubah menjadi berapa agar menghasilkan citra yang histogramnya merata
7 [new_I1, change1] = hist_equalization(I1, 1);
8 [~, change2] = hist_equalization(I2, 0);
9
10 % MEMBUAT MATRIKS PERUBAHAN AKHIR (INVERS)
11 % Menyiapkan matriks kosong seukuran change1
12 final_change = zeros(size(change1, 1));
13
14 % Iterasi setiap nilai pada setiap channel di change1 untuk melihat
15 % nilai pada indeks mana di change2, channel yang sama, yang paling
16 % mendekati nilai di change1 tersebut. Taruh hasil di final_change
17 for j = 1 : size(change1, 1)
18     minimum = 255;
19     idx = 0;
20     for l = 1 : size(change2, 1)
21         currDist = abs(change1(j) - change2(l));
22         if currDist < minimum
23             minimum = currDist;
24             idx = l;
25         end
26     end
27
28     final_change(j) = idx;
29 end
30
31 % Iterasi new_I1 untuk mengganti setiap nilai pixel dengan nilai
32 % final_change(channel, pixel). Hasil diletakkan di final_I
33 [M, N, C] = size(new_I1);
34 final_I = zeros(M, N, C);
35
36 for k = 1 : C
37     for i = 1 : M
38         for j = 1 : N
39             final_I(i,j,k) = final_change(new_I1(i,j,k)+1);
40         end
41     end
42 end
43
44 final_I = uint8(final_I);
45 end
46
```

## ➤ Contoh Eksekusi



## Tambahan:



➤ Analisis Algoritma dan Hasilnya

Spesifikasi histogram dilakukan untuk menapis citra menggunakan sebuah citra acuan sedemikian hingga histogram dari citra hasil tapisan mendekati histogram dari citra acuan tersebut. Metode yang digunakan memanfaatkan algoritma perataan histogram di atas. Pertama, citra semula diratakan sehingga memiliki sebaran nilai piksel seragam. Kemudian, citra acuan juga diratakan, dan larik perubahan dari penapisan tersebut disimpan. Larik ini digunakan untuk melakukan pemetaan balik nilai piksel citra dari sebaran seragam menjadi sebaran yang dimiliki oleh citra acuan.

Algoritma yang digunakan sebagai implementasi spesifikasi histogram dalam program ini memiliki kompleksitas waktu  $O(n^2)$ . Hasil spesifikasi dari citra semula merupakan sebuah citra baru yang sebaran nilai pikselnya mendekati sebaran nilai piksel dari citra acuan.

### 3. Pranala Penting

- Github *repository*: <https://github.com/RyanSC06/Tugas1-IF4073.git>