LAPORAN TUGAS 3

IF4073 Interpretasi dan Pengolahan Citra

Implementasi Deteksi Tepi dan Segmentasi Objek Menggunakan MATLAB

Disusun oleh

Annisa Rahim	13518089
Stefanus Gusega Gunawan	13518149

Tanggal Pengumpulan

Jumat, 1 April 2022

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2022

Daftar Isi

Kode Program MATLAB

Pada tugas ini, setiap kelompok diminta untuk mengimplementasikan berbagai jenis operator pendeteksi tepi untuk melakukan segmentasi objek. Operator yang harus diimplementasikan antara lain: Laplacian, Laplacian of Gaussian (LoG), Sobel, Prewitt, Roberts, dan Canny. Penggunaan fungsi *built-in* edge di MATLAB hanya diperbolehkan untuk operator Canny. Setelah deteksi tepi, dilakukan segmentasi objek menggunakan tepi yang terdeteksi sebelumnya. Citra masukan berisi minimal satu buah-buahan.

Berikut ini akan dijelaskan implementasi masing-masing algoritma pendeteksi tepi, segmentasi objek, dan fungsi tambahan lain yang diimplementasikan dengan menggunakan bahasa pemrograman MATLAB.

1. *Edge Detection*

Pendeteksian tepi dilakukan dalam fungsi utama **`detect_edge`**. Berikut rincian implementasinya.

`detect_edge.m`

```
function [imgOut] = detect_edge(imgIn, operator, T, sigma,
laplacian_version, T1, T2, mask_dim)
    % Check the dimension
    [height, width, dim] = size(imgIn);

    % Check the dimension
    if ~(dim == 1)
        throw(MException('ImageError:sizeNotOne', 'The input image should
be 1D array. Current: %dD array.', dim))
    end

    % Check if threshold defined. If not, then set automatically
    if (nargin == 2 | isempty(T))
        T = 0.09 * max(imgIn, [], 'all');
    end

    if (~(T > 0 && T < 255))
        throw(MException('RangeError:outOfRange', 'The T should be in
range (0, 255). Current: T = %d', T));
    end

    % Check if mask_dim defined. If not, then set automatically based on
sigma
    if (nargin == 7 | isempty(mask_dim))
        mask_dim = ceil(sigma * 3) * 2 + 1;
    end

    % Generate mask based on operator
```

```
    switch (operator)
        case 'sobel'
            [filterX, filterY] = sobel();
        case 'prewitt'
            [filterX, filterY] = prewitt();
        case 'roberts'
            [filterX, filterY] = roberts();
        case 'laplacian'
            mask = laplacian(laplacian_version);
        case 'log'
            mask = laplacian_of_gaussian(sigma, mask_dim);
    end

    % Do convolution between input image and the mask
    if ismember(operator, {'sobel' 'prewitt' 'roberts'})
        resX = conv2(double(imgIn), double(filterX), 'same');
        resY = conv2(double(imgIn), double(filterY), 'same');

        result = sqrt(resX.^2 + resY.^2);

    elseif (strcmp(operator, 'canny'))
        % If canny, directly output the resulting image
        imgOut = canny(imgIn, T1, T2, sigma);
    else
        result = conv2(double(imgIn), double(mask), 'same');
    end

    % Typecasting to unsigned integer except Canny and then apply
thresholding
    if (~(strcmp(operator, 'canny')))
        imgOut = thresholding(uint8(result), T);
    end

end
```

Fungsi **`detect_edge`** menerima 8 parameter, sebagai berikut:

- `imgIn`: citra masukan yang harus diambil dari satu *channel* saja.
- `operator`: operator yang digunakan untuk melakukan deteksi tepi. Nilai-nilai yang mungkin dari parameter ini adalah `laplacian`, `log`, `sobel`, `prewitt`, `roberts`, dan `canny`.
- `T`: nilai *threshold* yang digunakan untuk operasi pengambangan pada citra. Nilai yang mungkin adalah mulai dari 1 hingga 254. Jika tidak ditentukan, maka nilai *threshold* akan ditentukan menjadi 0,09 kali nilai intensitas maksimum pada citra.
- `sigma`: standar deviasi.
- `laplacian_version`: versi dari operasi laplacian. Parameter ini wajib di-*specify*, jika menggunakan operator laplacian untuk melakukan

deteksi tepi. Nilai-nilai yang mungkin dari parameter ini adalah `original` dan `diagonal`.

- `T1`: *threshold* pertama yang digunakan untuk menunjukkan *weak edge* pada citra. Jika nilai intensitas lebih kecil dari *threshold* pertama, maka akan diabaikan. Nilai `T1` harus lebih kecil dari `T2`. Nilai ini harus di-*specify* jika menggunakan operator Canny.
- `T2`: *threshold* kedua yang digunakan untuk menunjukkan *strong edge* pada citra. Jika nilai intensitas lebih besar dari `T1` dan lebih kecil dari `T2`, maka akan dikategorikan *weak edge*. Sedangkan, untuk nilai intensitas lebih besar dari `T2`, maka akan dikategorikan *strong edge*. Nilai ini harus di-*specify* jika menggunakan operator Canny. Jika `T1` dan `T2` tidak di-*specify*, maka akan dipilih secara otomatis dengan `T2` adalah 0,09 kali nilai maksimum intensitas pada citra, sedangkan `T1` adalah 0,05 kali `T2`.
- `mask_dim`: dimensi dari *mask* konvolusi yang digunakan. Nilai ini digunakan untuk operasi LoG (*Laplacian of Gaussian*), karena ukuran *mask* bisa bermacam-macam. Jika nilai ini tidak di-*specify*, maka nilainya akan ditentukan otomatis dengan formula $dim = \lceil (\sigma \times 3) \times 2 + 1) \rceil$. Dengan σ adalah standar deviasi yang digunakan pada distribusi Gaussian yang digunakan.

Pertama, akan dilakukan pengecekan parameter terlebih dahulu, sedemikian rupa sehingga tidak terjadi *unexpected behavior*. Kemudian akan ditentukan *mask*-nya sesuai dengan operator yang diinginkan. Lalu, akan dikelompokkan masing-masing operasi konvolusinya. Pertama, jika operator yang digunakan adalah Sobel, Prewitt, atau Roberts, maka akan dilakukan konvolusi citra masukan terhadap *mask* untuk arah X dan juga *mask* untuk arah Y. Lalu, hasil dari masing-masing konvolusi dari arah X dan arah Y akan dihitung menggunakan formula $result = \sqrt{res_X^{\,2} + res_Y^{\,2}}$ secara *element-wise*. Lalu, yang kedua jika operator yang digunakan merupakan Canny, maka langsung hasilkan citra keluaran, karena langsung menggunakan fungsi *built-in* dari MATLAB. Terakhir, jika operator yang digunakan merupakan Laplacian atau LoG, maka akan dilakukan konvolusi langsung antara citra masukan dengan *mask*. Tahap terakhir adalah melakukan *thresholding* jika operator yang digunakan bukan Canny. Maka, hasil *thresholding* itulah yang merupakan citra keluaran (*edge image*).

Operator pendeteksi akan dipanggil di dalam fungsi `detect_edge` untuk memanggil mask yang menjadi bahan konvolusi citra masukan. Operator dibagi ke dalam dua kelompok, yaitu berbasis *gradient* (turunan pertama) dan berbasis *laplacian* (turunan kedua). Sobel, Prewitt, Roberts, dan Canny dikelompokkan ke operator berbasis gradien, sedangkan Laplacian dan LoG dikelompokkan ke operator berbasis *laplacian.*

Gradient Based

Berikut rincian implementasi untuk operator berbasis gradien.

sobel.m

```
function [filterX,filterY] = sobel()
    filterX = [-1  0  1;
               -2  0  2;
               -1  0  1];

    filterY = [1  2  1;
               0  0  0;
              -1 -2 -1];
end
```

Fungsi ini akan mengembalikan mask `filterX` dan `filterY` yang merupakan gradien atau turunan parsial dari pixel di sekitar pixel tengah. Untuk posisi pixel sebagai berikut:

$$\begin{bmatrix} a_0 & a_1 & a_2 \\ a_7 & (x,y) & a_3 \\ a_6 & a_5 & a_4 \end{bmatrix}$$

Berikut persamaan turunan parsial yang digunakan:

$$s_x = (a_2 + ca_3 + a_4) - (a_0 + ca_7 + a_6)$$
$$s_y = (a_0 + ca_1 + a_2) - (a_6 + ca_5 + a_4)$$

Dengan konstanta c = 2, persamaannya adalah sebagai berikut:

$$s_x = (a_2 + 2a_3 + a_4) - (a_0 + 2a_7 + a_6)$$
$$s_y = (a_0 + 2a_1 + a_2) - (a_6 + 2a_5 + a_4)$$

Sehingga matriks yang dihasilkan sesuai dengan koefisien persamaan. Berikut matriks yang dihasilkan dari `filterX` dari persamaan s_x dan `filterY` dari persamaan s_y:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

prewitt.m

```
function [filterX,filterY] = prewitt()
    filterX = [-1  0  1;
               -1  0  1;
               -1  0  1];
```

```
    filterY = [1  1  1;
               0  0  0;
              -1 -1 -1];
end
```

Fungsi ini mengembalikan mask yang dihasilkan dengan cara yang sama seperti operator sobel, namun dengan konstanta c = 1. Berikut persamaan turunan parsial yang digunakan:

$$p_x = (a_2 + a_3 + a_4) - (a_0 + a_7 + a_6)$$
$$p_y = (a_0 + a_1 + a_2) - (a_6 + a_5 + a_4)$$

Sehingga matriks yang dihasilkan sesuai dengan koefisien persamaan. Berikut matriks yang dihasilkan dari filterX dari persamaan p_x dan filterY dari persamaan p_y:

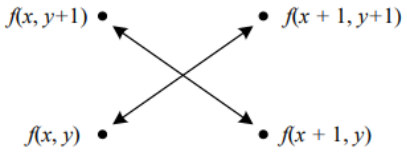
$$P_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad P_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

roberts.m

```
function [filterX,filterY] = roberts()
    filterX = [1  0;
               0 -1];

    filterY = [0  1;
              -1  0];
end
```

Mask yang dihasilkan pada fungsi ini berasal dari gradien roberts. Bila posisi elemen matriks sebagai berikut:



Gradien roberts yang merupakan operator silang memiliki persamaan sebagai berikut:

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

FilterX dan FilterY diambil dari persamaan tersebut dalam bentuk matriks mask. Hasilnya adalah sebagai berikut:

$$R_+ = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad R_- = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

canny.m

```
function imgOut = canny(imgIn, T1, T2, sigma)
    %canny - Edge detection for Canny method
    %
    % Syntax: imgOut = canny(imgIn, T1, T2, sigma)
    %
    % This is the shortcut to edge function of MATLAB function

    % If T1 and T2 undefined, then pick the automatically
    if (isempty(T1) & isempty(T2))
        T2 = max(imgIn, [], 'all') * 0.09;
        T1 = T2 * 0.05;
    end

    % Check the T1 and T2 within the range
    if (~(T1 > 0 & T2 > 0 & T1 < 255 & T2 < 255))
        throw(MException('RangeError:outOfRange', 'The T1 and T2 should
be in range (0, 255). Current: T1 = %d, T2 = %d', T1, T2))
    end

    % Check if T1 <= T2
    if (~(T1 < T2))
        throw(MException('OrderError:wrongOrder', 'The T1 should be less
than T2.'))
    end

    % Normalize to [0, 1]
    T1 = double(T1) / 255
    T2 = double(T2) / 255

    % Do edge detection
    imgOut = edge(imgIn, 'Canny', [T1 T2], sigma);
end
```

Selanjutnya adalah fungsi `canny`. Fungsi ini berguna untuk melakukan *edge detection* dengan operator Canny, dengan menggunakan fungsi *built-ini* dari MATLAB, yaitu `edge`. Fungsi ini menerima 4 parameter, sebagai berikut.

- `imgIn`: citra masukan yang akan dilakukan *edge detection* dan harus memiliki dimensi 2D, sehingga hanya dari satu *channel* warna saja.
- `T1`: *threshold* pertama yang menandakan *weak edge threshold*.
- `T2`: *threshold* kedua yang menandakan *strong edge threshold*.
- `sigma`: standar deviasi yang digunakan untuk menentukan distribusi Gaussian pada citra.

Pertama, akan dilakukan pengecekan parameter dan *auto-assign* pada `T1` dan `T2`, jika keduanya tidak di-*specify*. Selanjutnya, nilai *threshold* akan dinormalisasi menjadi rentang $[0, 1]$. Lalu, akan dilakukan *edge detection* dengan menggunakan fungsi *built-in* MATLAB yaitu `edge`. Fungsi ini langsung

mengembalikan citra *thresholded* sehingga tidak perlu di-*thresholding* lagi kedepannya.

Langkah-langkah operator Canny adalah sebagai berikut:

1. Penghalusan citra menggunakan *filter* Gaussian ($G * I$), dengan standar deviasi atau sigma yang dispesifikasikan.
2. Menggunakan salah satu dari operator lain (sobel, prewitt, dll) untuk menghitung gradien dan arah gradien setiap pixel.
3. Mengelompokkan pixel tepi dengan melihat apakah magnitude dari gradien melebihi nilai ambang T. Terdapat dua nilai ambang (T1<T2) dan dua jenis tepi yang dihasilkan:
    a. Tepi kuat: magnitudo > `T2`
    b. Tepi lemah: terhubung ke tepi kuat dan magnitudo > `T1`

Laplacian Based

Operator yang dikelompokkan berbasis *laplacian* adalah laplacian dan LoG. Berikut rincian implementasinya.

`laplacian.m`

```
function mask = laplacian(version)
    %laplacian - Generate Laplacian mask
    %
    % Syntax: mask = laplacian(version)
    %

    % Check the version
    if ~ismember(version, {'original' 'diagonal'})
        throw(MException('VersionError:wrongVersion', "The Laplacian
version should be one of these: 'original' or 'diagonal'. Current: %s.",
version))
    end

    if strcmp(version, 'original')
        mask = [0 1 0;
                1 -4 1;
                0 1 0];
    elseif strcmp(version, 'diagonal')
        mask = [1 1 1;
                1 -8 1;
                1 1 1];
    end

end
```

Pertama, akan dibahas mengenai operator Laplacian yang proses penghasilan *mask*-nya terdapat pada fungsi `laplacian`. Fungsi ini hanya menerima satu parameter, yaitu versi dari operator Laplacian. Pertama, akan dilakukan pengecekan versi dari Laplacian yang di-*specify* apakah sudah benar. Versi ini

terbagi menjadi *original* dan *diagonal*. Untuk versi *original* akan dihasilkan *mask* sebagai berikut.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Sedangkan, untuk versi *diagonal* akan dihasilkan *mask* sebagai berikut.

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Laplacian_of_gaussian.m (LoG)

```
function value = laplacian_of_gaussian(sigma, mask_dim)
    %laplacian_of_gaussian - Generate the Laplacian of Gaussian mask
    %
    % Syntax: value = laplacian_of_gaussian(sigma, mask_dim)

    left_boundary = -floor(mask_dim / 2);
    right_boundary = floor(mask_dim / 2);
    lin = round(linspace(left_boundary, right_boundary, mask_dim));

    [meshgrid_x, meshgrid_y] = meshgrid(lin, lin);

    D = meshgrid_x.^2 + meshgrid_y.^2;
    value = ((D - 2 * (sigma^2)) ./ (sigma^4)) .* exp(-D ./ (2 *
(sigma^2)));

end
```

Operator yang kedua adalah Laplacian of Gaussian atau yang biasa disingkat LoG. Fungsi ini menerima parameter sebagai berikut.

- `sigma`: standar deviasi yang digunakan pada distribusi Gaussian.
- `mask_dim`: dimensi yang diinginkan dari *mask* yang nanti dihasilkan.

Penentuan nilai-nilai pada *mask* mengikuti formula Laplacian of Gaussian berikut.

$$\nabla^2 G(x,y) = (\frac{x^2+y^2-2\sigma^2}{\sigma^4})e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Pertama, harus ditentukan terlebih dahulu batas-batas dari *plane* pada *mask* yang dimulai dari $-\lfloor\frac{dim}{2}\rfloor$ hingga $\lfloor\frac{dim}{2}\rfloor$ pada tiap sumbu. Lalu, akan dibuat *meshgrid* 2D dengan rentang tersebut. Kemudian, dilakukan penghitungan nilai *mask* sesuai formula di atas.

Menangani Kasus RGB

Pada citra masukan berwarna yang memiliki channel RGB, dilakukan pendeteksian tepi dengan memproses channel satu persatu terlebih dahulu. Setelah menghasilkan citra tepi masing-masing yang sudah dilakukan *thresholding*, penggabungan citra tepi dari setiap *channel* dilakukan dengan menggunakan operator *bitwise* OR. Implementasi dilakukan pada kode GUI dari app.mlapp sebagai berikut.

```
switch size(im,3)
    case 1
        % Display the grayscale image
        edge_only = getOutput(app, im, tab_num);
        segmented = segment(edge_only, im);
        imagesc(currEdgeImageAxes, edge_only);
        imagesc(currSegmentImageAxes, segmented);

    case 3
        edge_1 = getOutput(app, im(:,:,1), tab_num);
        edge_2 = getOutput(app, im(:,:,2), tab_num);
        edge_3 = getOutput(app, im(:,:,3), tab_num);

        % Do bitwise OR to merge all channels
        edge_only = edge_1 | edge_2 | edge_3;
        segmented = segment(edge_only, im);

        imagesc(currEdgeImageAxes, edge_only);
        imagesc(currSegmentImageAxes, segmented);
end
```

Operator *bitwise* OR dipilih untuk menggabungkan tepi yang terdeteksi pada channel Red, Green, dan Blue sehingga semua jenis tepi tersimpan dan tergabung dalam citra akhir tepi. Selanjutnya, segmentasi dilakukan berdasarkan *edge* yang dihasilkan dari operasi OR tadi.

2. *Object Segmentation*

Segmentasi objek diimplementasikan pada fungsi `segment` pada `segment.m`. Berikut ini adalah implementasi dari fungsi tersebut.

```
function imgOut = segment(edgeIm, oriIm)
    % segment - function for object segmentation
    %
    % Syntax: imgOut = segment(edgeIm, oriIm)
    %
    % Object Segmentation based on edge image.

    [h,w,d] = size(edgeIm);
```

```
    % clean image border
    clear = edgeIm;
    clear(1,:) = 0;
    clear(h,:) = 0;
    clear(:,1) = 0;
    clear(:,w) = 0;

    % connecting edge lines
    mask = imdilate(clear, strel('line', 3, 0));
    mask = imdilate(mask, strel('line', 3, 45));
    mask = imdilate(mask, strel('line', 3, 90));
    mask = imdilate(mask, strel('line', 3, 135));

    mask = imdilate(mask, strel('disk', 5));

    % fill in object gaps
    mask = imfill(mask, 8, 'holes');

    % apply to original image
    imgOut = oriIm .* uint8(mask);

end
```

Fungsi segment ini menerima 3 parameter, yaitu:

- `edgeIm`: citra *binary* yang berisikan *edge* dari citra asli, bisa berukuran 3D ataupun 2D.
- `oriIm`: citra asli, bisa berukuran 3D ataupun 2D
- `technique`: teknik yang digunakan pada *edge segmentation* sebelumnya.

Dengan menerima masukan citra tepi yang dihasilkan pada langkah *edge detection,* fungsi ini akan menyempurnakan tepi yang terbentuk agar tersambung secara utuh membentuk objek. Proses penyempurnaan tepi dilakukan dengan langkah-langkah berikut:

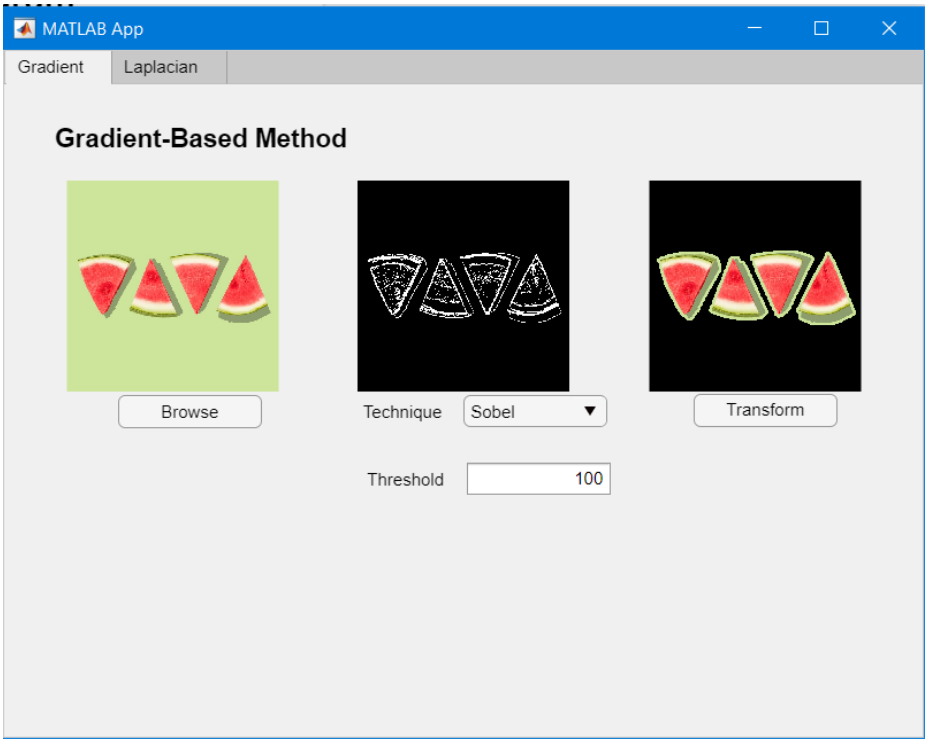
1. membersihkan *border* citra dari tepi yang terdeteksi,
2. menggunakan `imdilate` untuk memperlebar tepi-tepi ke segala arah, dan
3. mengisi tempat yang masih berlubang dengan menggunakan `imfill.`

Setelah itu, *mask* dikalikan dengan citra asli secara *element-wise* sehingga bisa melakukan *filter* menggunakan *mask* tersebut.

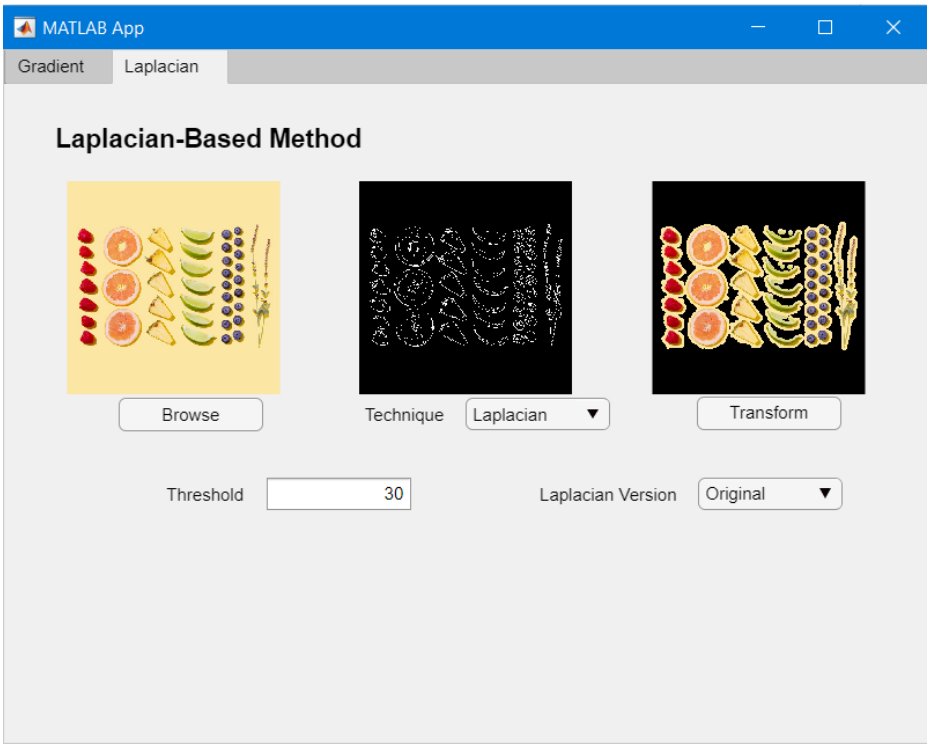
Screenshot Antarmuka Program

Berikut *screenshot* yang menunjukan tampilan GUI dari aplikasi yang dibuat.

Menu *Gradient Based Object Segmentation*



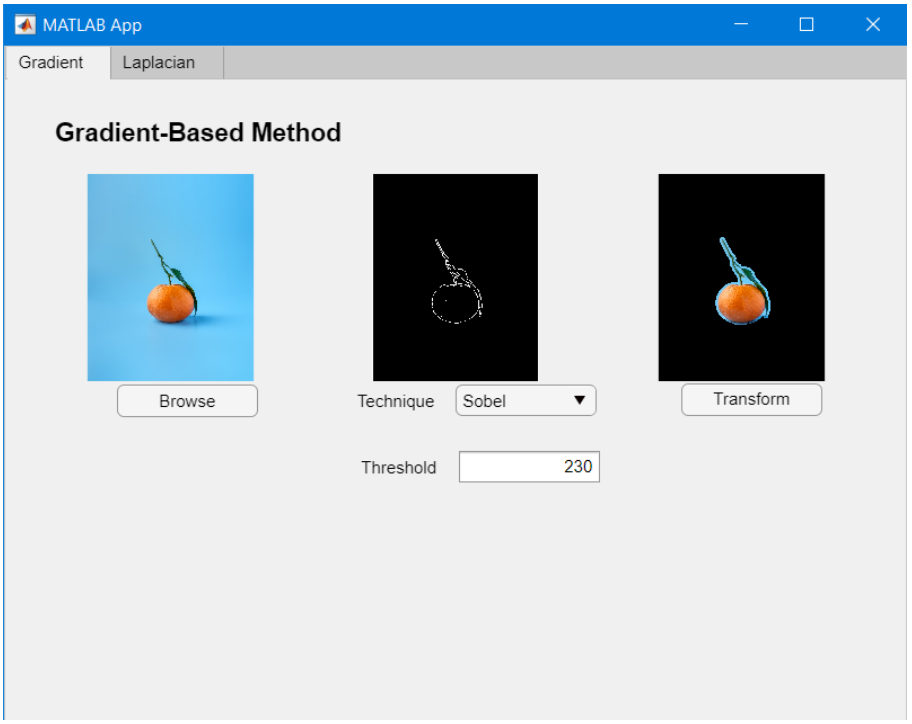
Menu *Laplacian Based Segmentation*

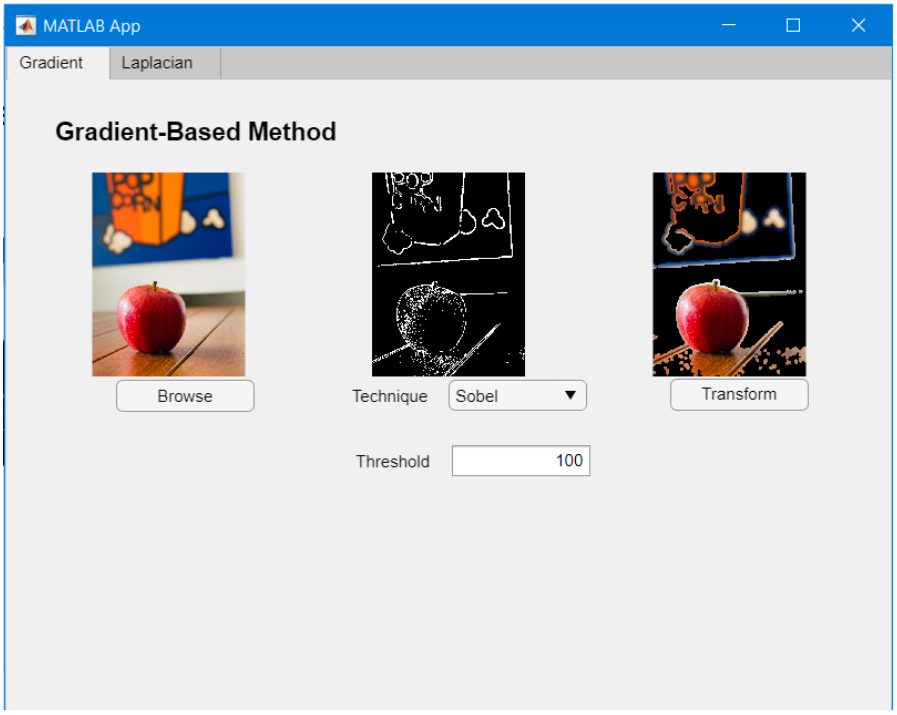


Contoh Hasil Eksekusi Program

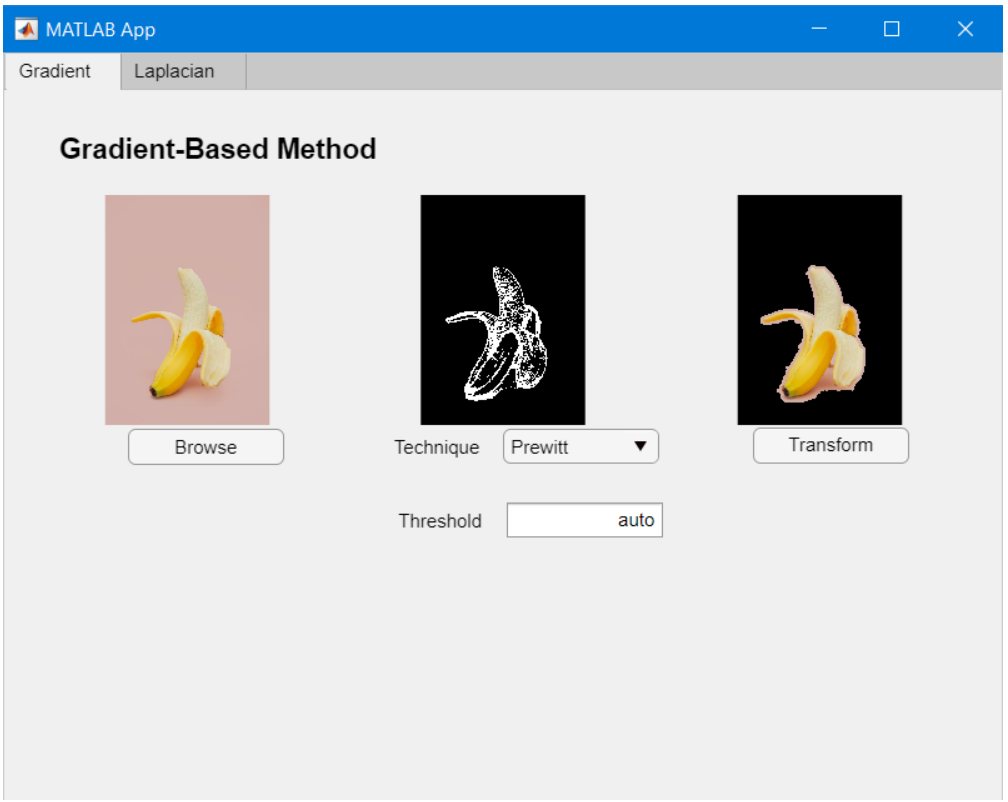
Gradient

- Sobel

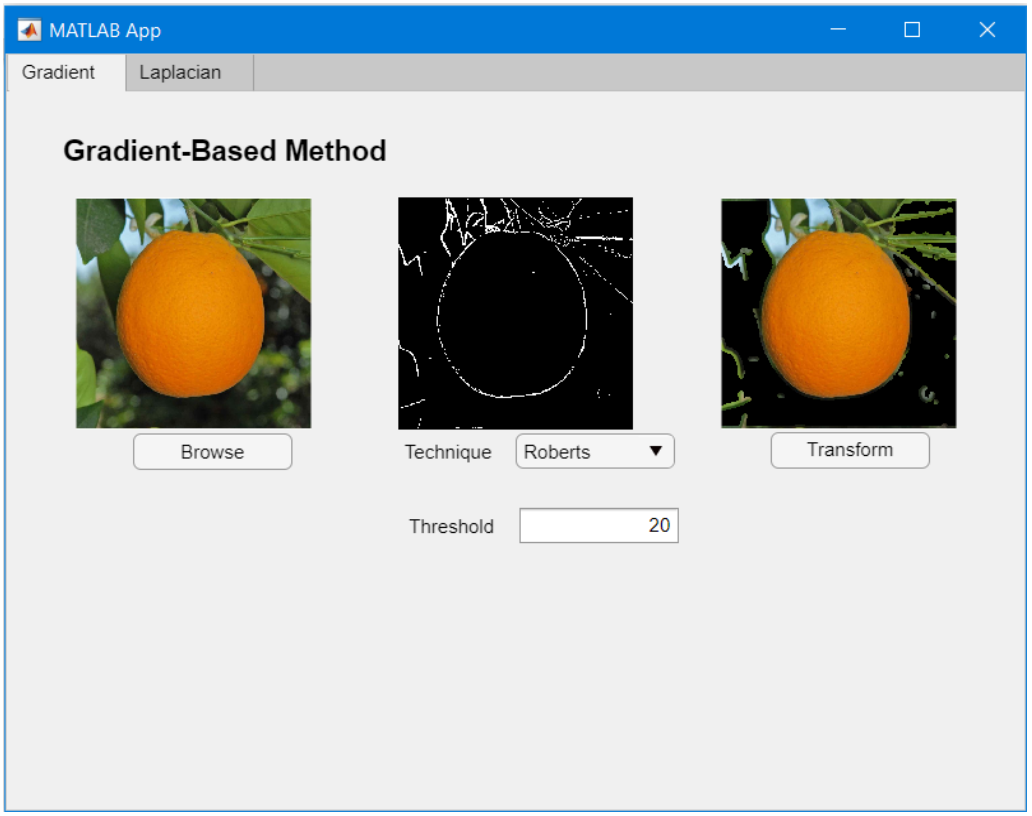




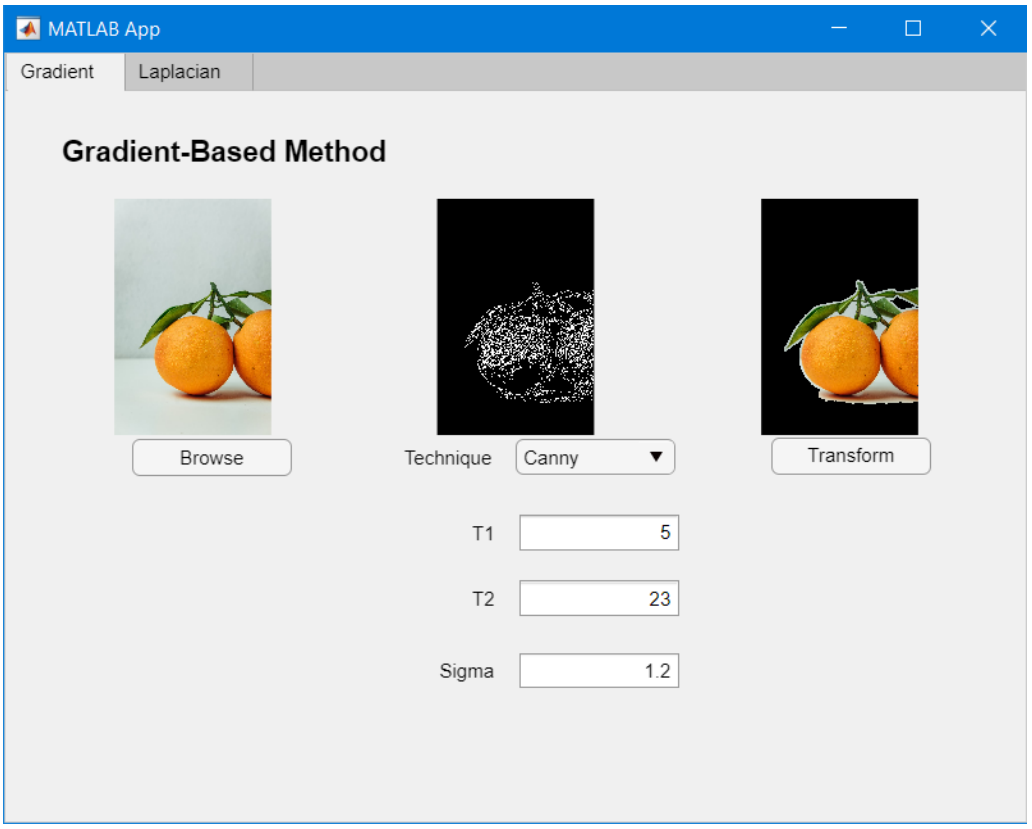
- Prewitt



- Roberts

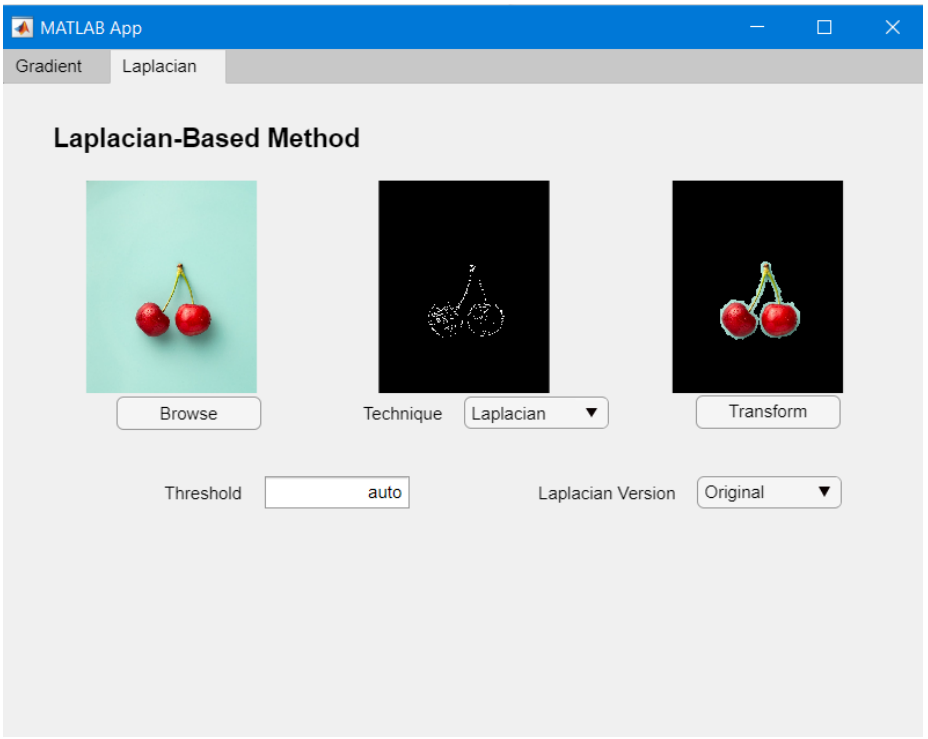


- Canny

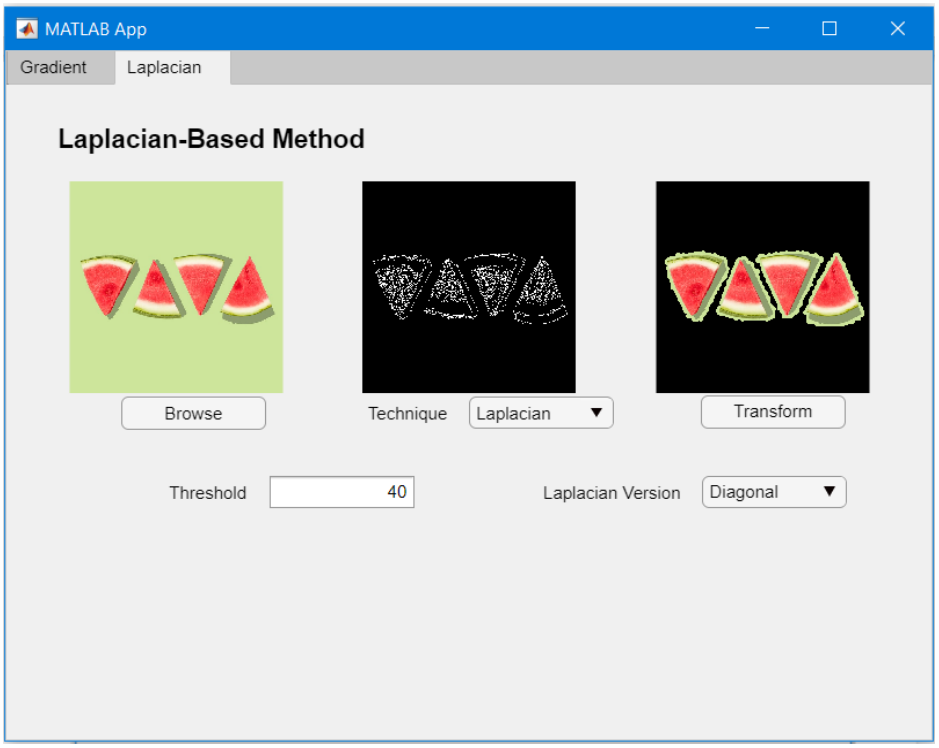

MATLAB App
Gradient
Laplacian
Gradient-Based Method
Browse
Technique
Canny
Transform
T1
5
T2
23
Sigma
1.2

Laplacian

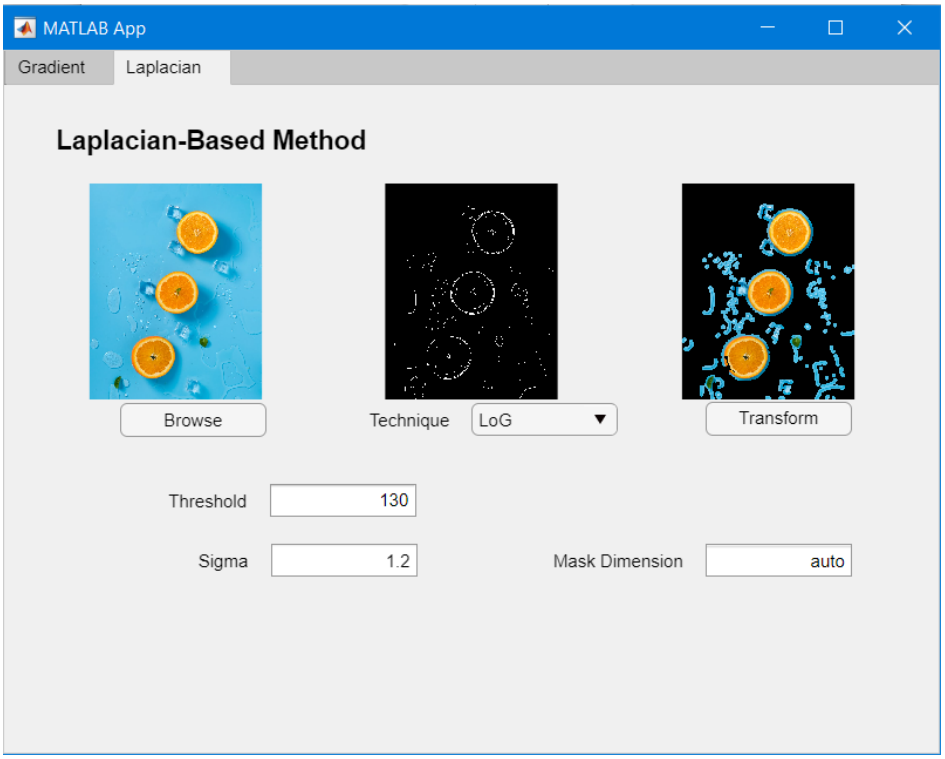
- Laplacian original

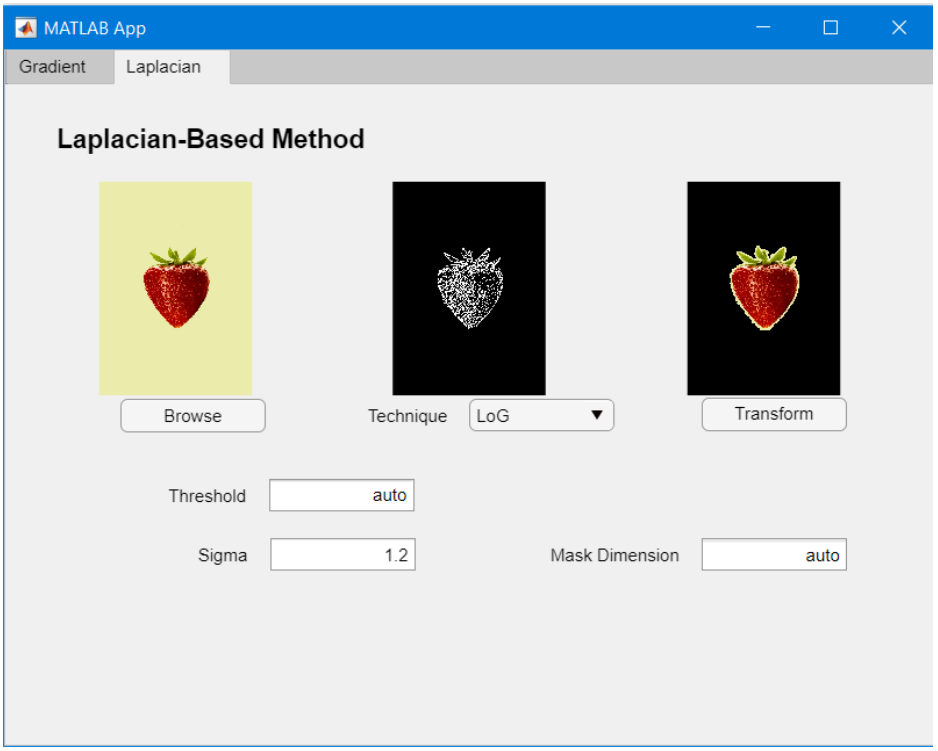


- Laplacian diagonal



- LoG


MATLAB App
Gradient
Laplacian
Laplacian-Based Method
Browse
Technique
LoG
Transform
Threshold
130
Sigma
1.2
Mask Dimension
auto


MATLAB App
Gradient
Laplacian
Laplacian-Based Method
Browse
Technique
LoG
Transform
Threshold
auto
Sigma
1.2
Mask Dimension
auto

Pranala Kode Program pada GitHub

Kode program dapat diakses pada pranala berikut ini.

Pembagian tugas adalah sebagai berikut.

13518089	- Mengimplementasikan fungsi `sobel` - Mengimplementasikan fungsi `prewitt` - Mengimplementasikan fungsi `roberts` - Mengimplementasikan fungsi `detect_edge` - Mengimplementasikan fungsi `segment` - Mengimplementasikan GUI *Gradient-based*
13518149	- Mengimplementasikan fungsi `laplacian` - Mengimplementasikan fungsi `laplacian_of_gaussian` - Mengimplementasikan fungsi `canny` - Refactor fungsi `detect_edge` - Mengimplementasikan GUI *Laplacian-based* - Mengimplementasikan fungsi `segment`

Referensi

- Citra buah-buahan diambil dari
    - https://unsplash.com
    - https://www.tomorrowsharvest.com/store/frost-valencia-orange.html