

Laporan Tugas Kecil 3

IF4073 Interpretasi dan Pengolahan Citra



Oleh

Ruhiyah Faradishi Widiaputri 13519034

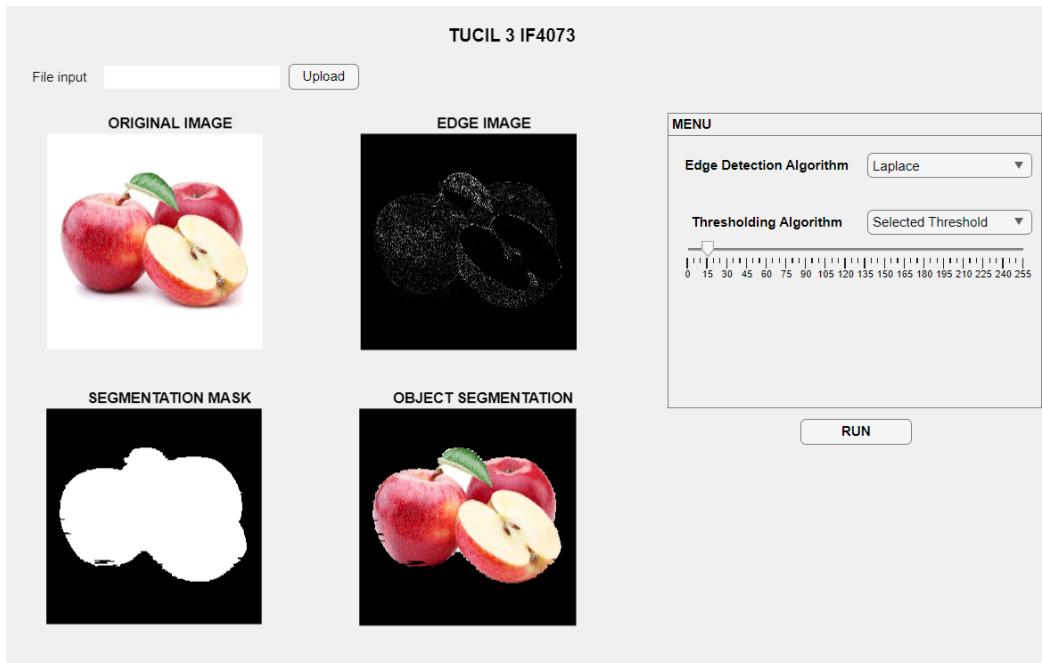
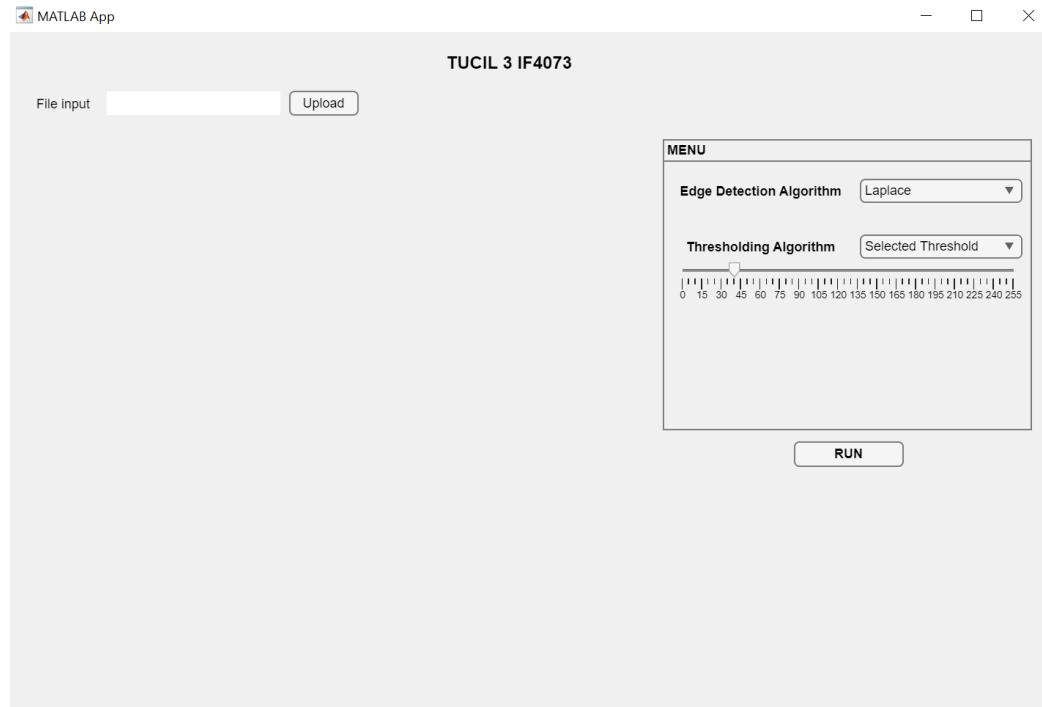
Sharon Bernadetha Marbun 13519092

INSTITUT TEKNOLOGI BANDUNG

2022

1. Screenshot GUI

Berikut adalah hasil screenshot GUI yang diimplementasi menggunakan App Designer Matlab.



2. Program

a. Pendeksiian Tepi

- Kode program:

```
classdef EdgeDetection
    methods (Static)
        function laplaceEdge = edgeDetectionLaplace(image)
            % Mask konvolusi laplace
            % H = [0 1 0; 1 -4 1; 0 1 0];
            H = [1 1 1; 1 -8 1; 1 1 1];

            % Konvolusi untuk mendapatkan tepi
            laplaceEdge = uint8(convn(double(image),
double(H), "same"));
        end

        function logEdge = edgeDetectionLog(image)
            % mask konvolusi LoG
            H = [0 0 -1 0 0;
                  0 -1 -2 -1 0;
                  -1 -2 16 -2 -1;
                  0 -1 -2 -1 0;
                  0 0 -1 0 0];

            % Konvolusi untuk mendapatkan tepi
            logEdge = uint8(convn(double(image), double(H), "same"));
        end

        function sobelEdge = edgeDetectionSobel(image)
            % Mask konvolusi Sobel;
            Sx = [-1 0 1;
                  -2 0 2;
                  -1 0 1];
            Sy = [1 2 1;
                  0 0 0;
                  -1 -2 -1];

            % Konvolusi
            Jx = conv2(double(image), double(Sx), "same");
            Jy = conv2(double(image), double(Sy), "same");

            % dapatkan edge
            sobelEdge = uint8(sqrt(Jx.^2 + Jy.^2));
        end

        function prewittEdge = edgeDetectionPrewitt(image)
            % Mask konvolusi Prewitt
            Px = [-1 0 1; -1 0 1; -1 0 1];
            Py = [-1 -1 -1; 0 0 0; 1 1 1];
```

```

Jx = convn(double(image), double(Px), 'same');
Jy = convn(double(image), double(Py), 'same');
% Menghitung kekuatan tepi
Jedge = sqrt(Jx.^2 + Jy.^2);

prewittEdge = uint8(Jedge);
end

function robertsEdge = edgeDetectionRoberts(image)
% Mask konvolusi Roberts
Rx = [1 0; 0 -1];
Ry = [0 1; -1 0];
Jx = convn(double(image), double(Rx), 'same');
Jy = convn(double(image), double(Ry), 'same');
% Menghitung kekuatan tepi
Jedge = sqrt(Jx.^2 + Jy.^2);

robertsEdge = uint8(Jedge);
end

function cannyEdge = edgeDetectionCanny(image)
% Mengonversi image berwarna ke grayscale
if (~Helper.isGrayscale(image))
    image = rgb2gray(image);
end

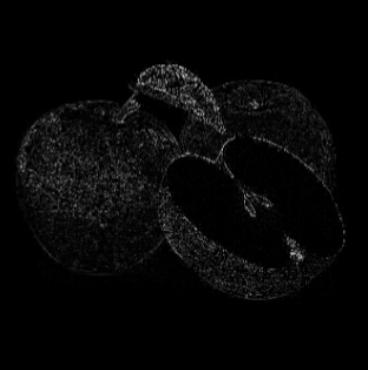
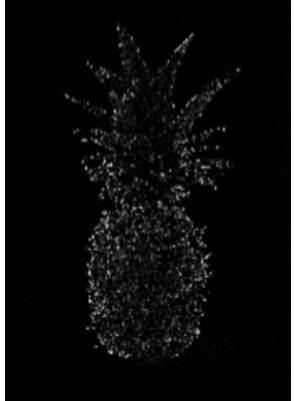
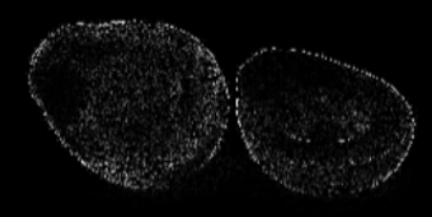
% Menggunakan fungsi built-in Canny
cannyEdge = im2uint8(edge(image, 'Canny'));
end

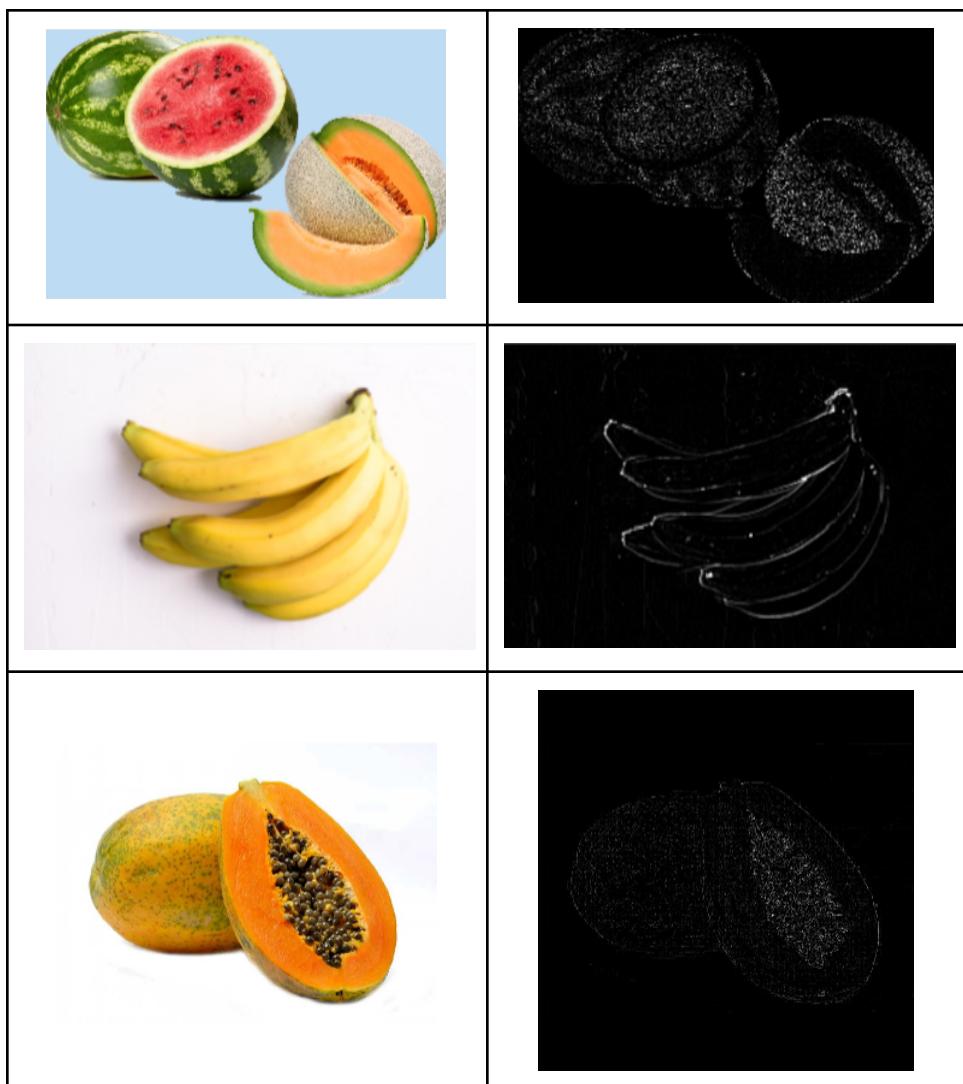
function res = edgeDetection(image, algorithm)
switch algorithm
    case "Laplace"
        res = EdgeDetection.edgeDetectionLaplace(image);
    case "LoG"
        res = EdgeDetection.edgeDetectionLoG(image);
    case "Sobel"
        res = EdgeDetection.edgeDetectionSobel(image);
    case "Prewitt"
        res = EdgeDetection.edgeDetectionPrewitt(image);
    case "Roberts"
        res = EdgeDetection.edgeDetectionRoberts(image);
    case "Canny"
        res = EdgeDetection.edgeDetectionCanny(image);
    end
end
end
end

```

- Contoh hasil eksekusi:
 - Deteksi tepi dengan menggunakan operator *laplace*

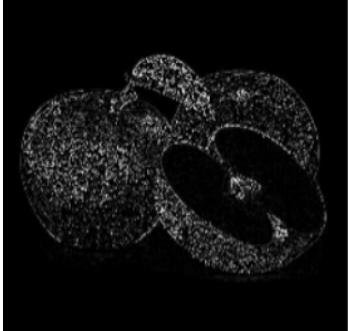
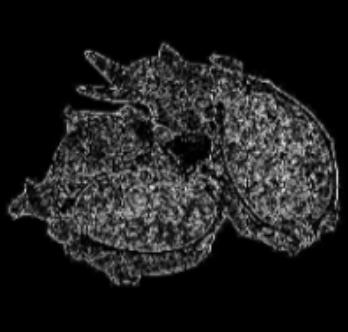
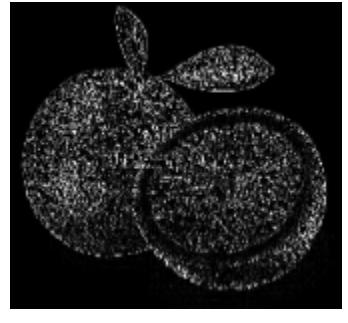
Deteksi tepi dengan menggunakan operator *laplace* dilakukan dengan menggunakan fungsi `edgeDetectionLaplace` di atas. Hasilnya adalah sebagai berikut.

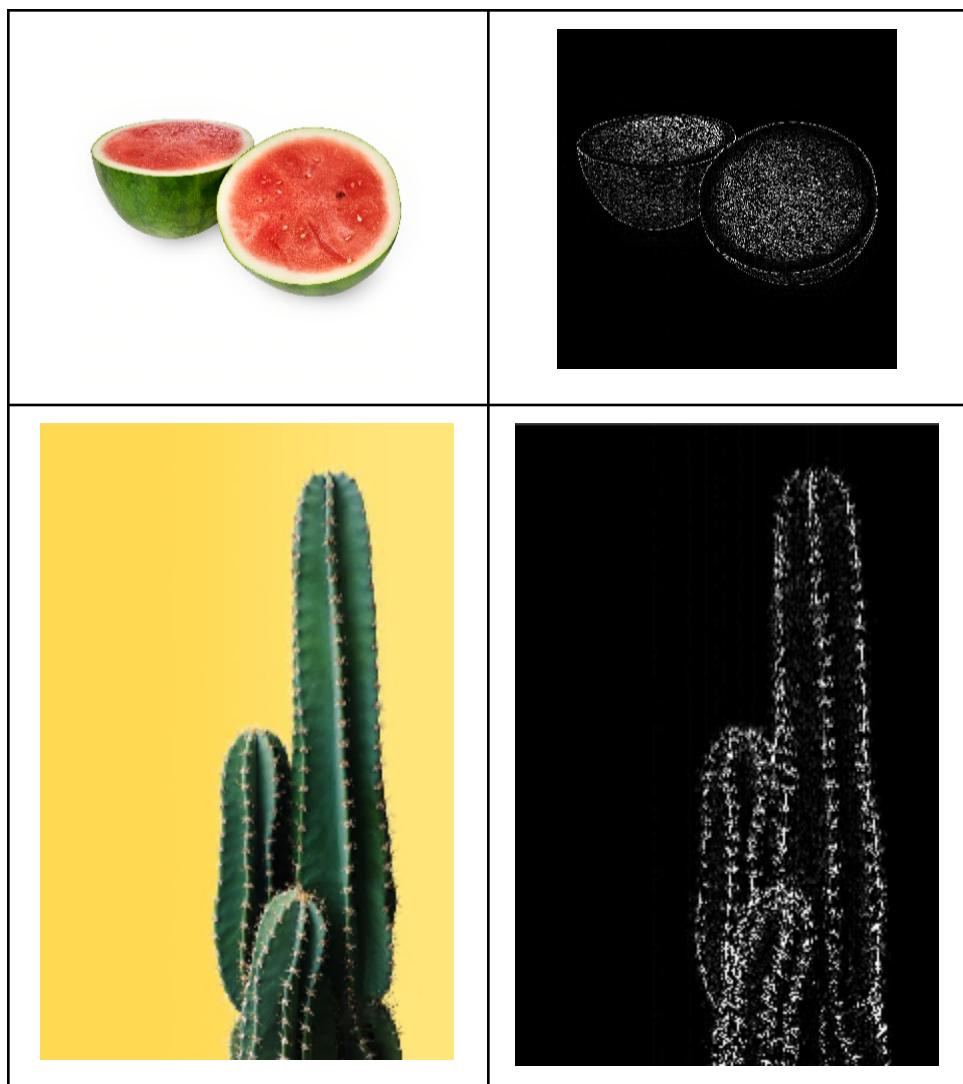
Input	Output
	
	
	



- Deteksi tepi dengan menggunakan operator LoG

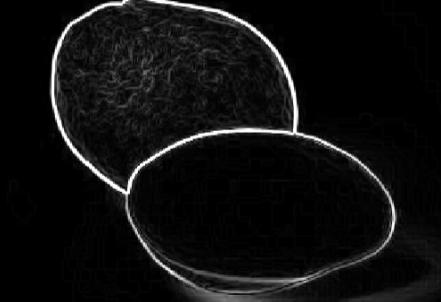
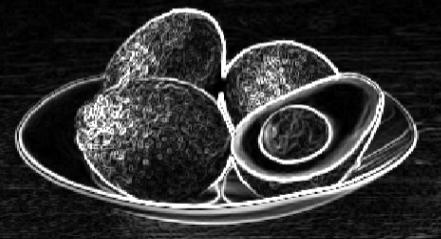
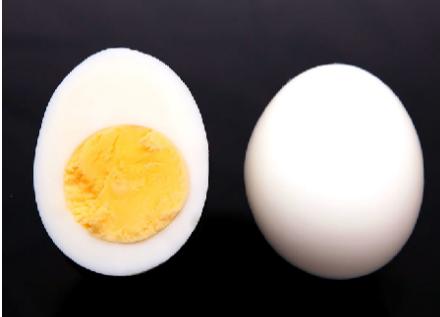
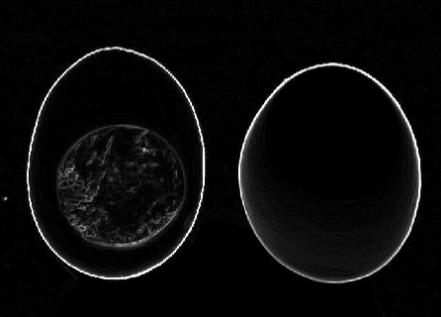
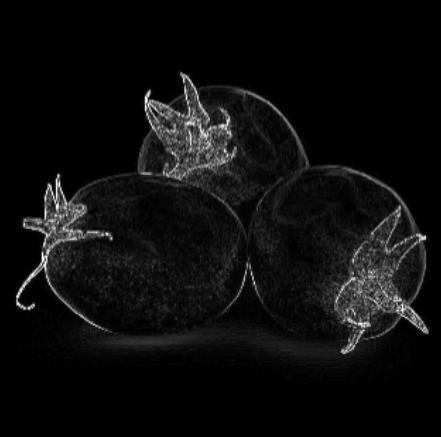
Deteksi tepi dengan menggunakan operator LoG (*laplace of gaussian*) dilakukan dengan menggunakan fungsi `edgeDetectionLoG` di atas. Hasilnya adalah sebagai berikut.

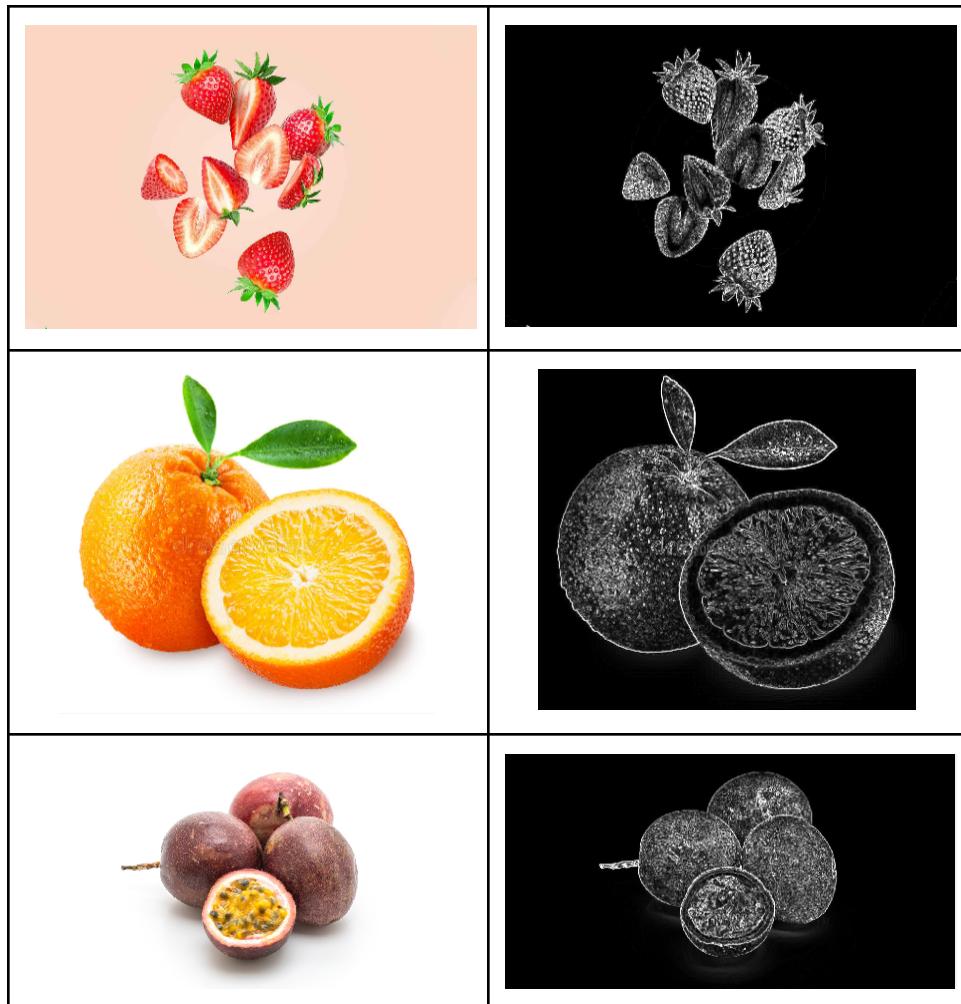
Input	Output
	
	
	
	



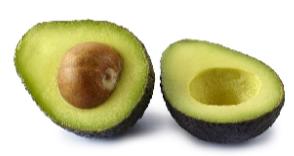
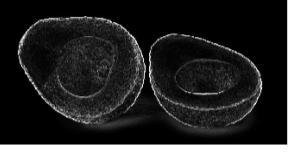
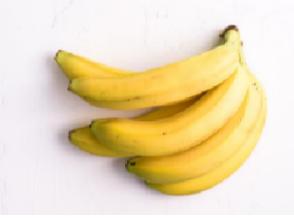
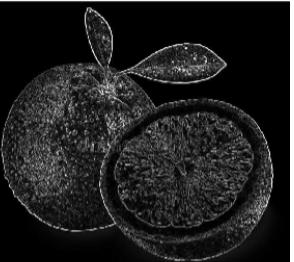
- Deteksi tepi dengan menggunakan operator Sobel

Deteksi tepi dengan menggunakan operator Sobel dilakukan dengan menggunakan fungsi `edgeDetectionSobel` di atas. Hasilnya adalah sebagai berikut.

Input	Output
	
	
	
	

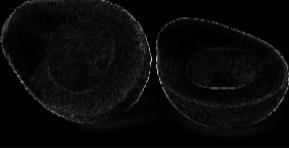
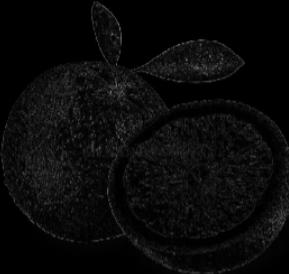


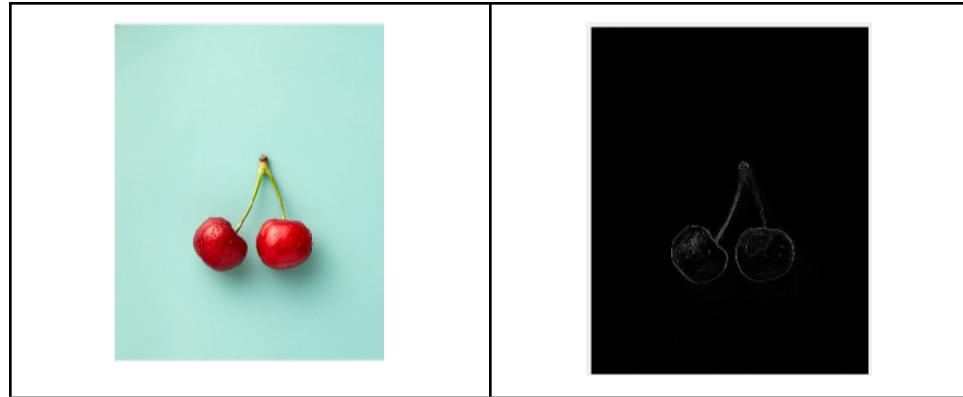
- Deteksi tepi dengan menggunakan operator Prewitt
- Deteksi tepi dengan menggunakan operator Prewitt dilakukan dengan menggunakan fungsi `edgeDetectionPrewitt` di atas. Hasilnya adalah sebagai berikut.

Input	Output
	
	
	
	
	

- Deteksi tepi dengan menggunakan operator Roberts

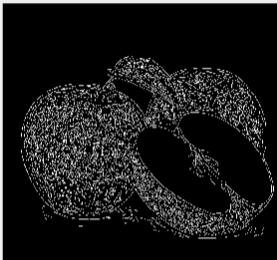
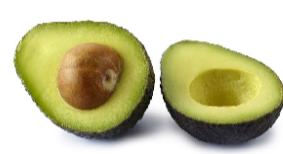
Deteksi tepi dengan menggunakan operator Roberts dilakukan dengan menggunakan fungsi `edgeDetectionRoberts` di atas. Hasilnya adalah sebagai berikut.

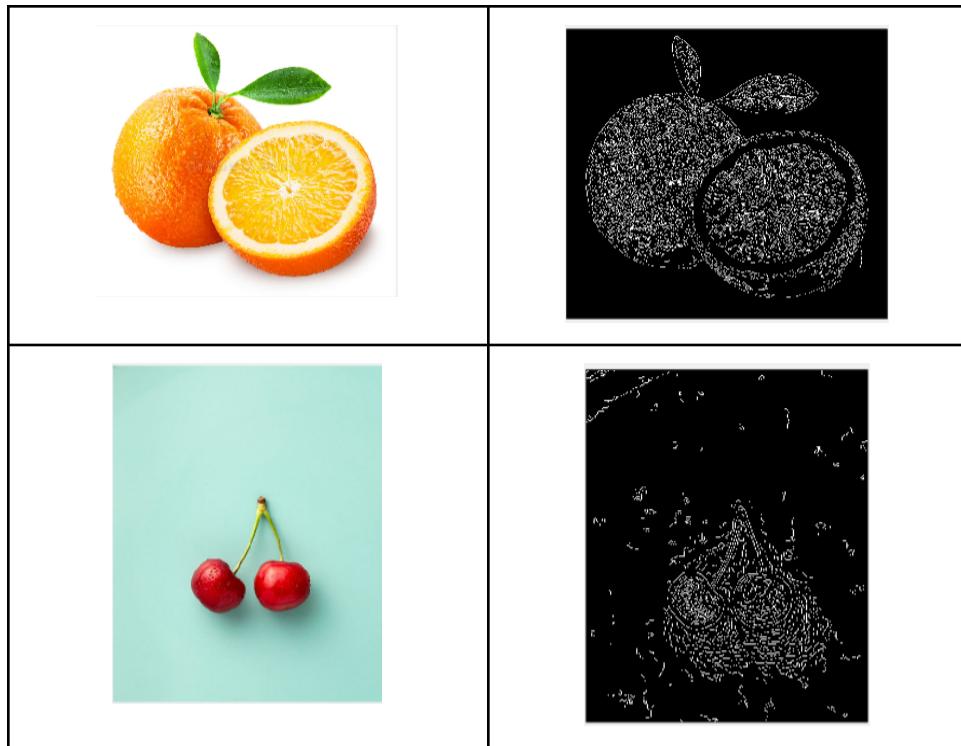
Input	Output
	
	
	
	



- Deteksi tepi dengan menggunakan operator Canny

Deteksi tepi dengan menggunakan operator Canny dilakukan dengan menggunakan fungsi `edgeDetectionCanny` di atas. Hasilnya adalah sebagai berikut.

Input	Output
	
	
	



- Analisis cara kerja dan hasilnya:

- Deteksi tepi dengan menggunakan operator *laplace*

Deteksi tepi dengan operator *laplace* (turunan kedua) mendeteksi lokasi tepi lebih akurat, khususnya pada tepi yang curam. Turunan kedua diperoleh dengan persamaan:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = G_1\left(\frac{\partial f(x,y)}{\partial x}\right) + G_1\left(\frac{\partial f(x,y)}{\partial y}\right)$$

Dengan penurunan rumus persamaan di atas pencarian citra tepi dapat disederhanakan menjadi konvolusi citra masukan dengan *mask* konvolusi:

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

Selain *mask* konvolusi di atas terdapat banyak operator *laplace* lainnya. Operator *laplace* yang digunakan di tugas ini adalah operator:

1	1	1
1	-8	1
1	1	1

Operator ini dipilih karena memberikan tepi yang lebih jelas dibanding operator sebelumnya.

- Deteksi tepi dengan menggunakan operator LoG

Operator *laplace of gaussian* (LoG) adalah operator yang dapat mengurangi kemunculan tepi palsu dengan menapis citra terlebih dahulu dengan fungsi Gaussian $G(x, y)$ sebelum dikenakan operator $\text{laplace } \nabla^2 f$.

Dengan demikian citra tepi $k(x, y)$ dapat dinyatakan sebagai:

$$k(x, y) = \nabla^2(f(x, y) * G(x, y))$$

$$k(x, y) = f(x, y) * \nabla^2 G(x, y)$$

Dengan $\nabla^2 G(x, y)$ adalah turunan kedua dari fungsi Gauss. Dengan penurunan rumus di atas pencarian citra tepi dapat disederhanakan menjadi konvolusi citra dengan penapis LoG. Penapis LoG yang digunakan pada tugas ini adalah penapis LoG yang berukuran 5×5 yaitu

$$\begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

- Deteksi tepi dengan menggunakan operator Sobel

Operator Sobel mencoba menghitung magnitudo dari gradien dengan persamaan

$$M = \sqrt{s_x^2 + s_y^2}$$

dengan

$$S_x = (a_2 + ca_3 + a_4) - (a_0 + ca_7 + a_6) \text{ dan} \quad \begin{bmatrix} a_0 & a_1 & a_2 \\ a_7 & (x, y) & a_3 \\ a_6 & a_5 & a_4 \end{bmatrix}$$

$$S_y = (a_0 + ca_1 + a_2) - (a_6 + ca_5 + a_4)$$

Dengan nilai $c = 2$, maka S_x dan S_y dapat dinyatakan dalam bentuk *mask* konvolusi

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

Citra tepi diperoleh dengan mendapatkan 2 matriks hasil konvolusi citra masukan dengan S_x dan S_y , kemudian dilakukan perhitungan besaran gradien dengan mengakar kuadratkan kedua hasil konvolusi tersebut. Besaran gradien tersebut menunjukkan kekuatan dari tepi gambar, sehingga hasilnya adalah citra output berupa gambar tepi.

- Deteksi tepi dengan menggunakan operator Prewitt
Persamaan gradien pada operator Prewitt sama seperti operator Sobel, tetapi menggunakan nilai $c = 1$. Dengan demikian, diperoleh mask konvolusi P_x dan P_y sebagai berikut.

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

Besarnya gradien dihitung dengan rumus:

$$G = \sqrt{P_x^2 + P_y^2}$$

Pertama-tama, citra dikonvolusikan dengan kernel horizontal (P_x), kemudian citra juga dikonvolusikan dengan kernel vertikal (P_y). Kemudian, dilakukan perhitungan besaran gradien dengan mengakar kuadratkan kedua hasil konvolusi tersebut. Besaran gradien tersebut menunjukkan kekuatan dari tepi gambar, sehingga hasilnya adalah citra output berupa gambar tepi.

- Deteksi tepi dengan menggunakan operator Roberts

Operator Roberts disebut juga sebagai operator silang karena dalam operasinya dilakukan perhitungan gradien dalam dua arah yang bersilangan, yaitu arah horizontal (x) dan arah vertikal (y). Gradien tersebut dihitung dengan rumus:

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

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

Sehingga, dalam bentuk mask konvolusi menjadi sebagai berikut.

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

Besarnya gradien dihitung dengan rumus:

$$G = \sqrt{R_+^2 + R_-^2}$$

Pertama-tama, citra dikonvolusikan dengan kernel horizontal (R_+), kemudian citra juga dikonvolusikan dengan kernel vertikal (R_-). Kemudian, dilakukan perhitungan besaran gradien dengan mengakar kuadratkan kedua hasil konvolusi tersebut. Besaran gradien tersebut menunjukkan kekuatan dari tepi gambar, sehingga hasilnya adalah citra output berupa gambar tepi.

- Deteksi tepi dengan menggunakan operator Canny

Pada tugas ini, pendekripsi tepi menggunakan operator Canny diimplementasikan menggunakan fungsi *built-in* Matlab. Namun, cara kerja operator Canny adalah sebagai berikut.

Pertama-tama, citra dihaluskan menggunakan penapis Gaussian dengan standar deviasi yang dispesifikasikan. Kemudian, dihitung gradien dan arah gradien setiap pixel dari citra masukan menggunakan salah satu dari operator sebelumnya (mis. operator Sobel). Jika nilai mutlak (magnitudo) gradien suatu pixel melebihi nilai ambang T, maka pixel tersebut termasuk pixel tepi.

Operator Canny menggunakan dua nilai ambang, T1 dan T2 ($T_1 < T_2$), sehingga memungkinkan deteksi dua jenis tepi: tepi kuat dan tepi lemah. Jika magnitudo pixel di dalam citra melebihi T2, maka pixel tersebut merupakan tepi kuat. Jika pixel yang merupakan tepi kuat memiliki nilai lebih besar dari T1, maka pixel tersebut merupakan tepi lemah.

b. Segmentasi Objek

- Kode program:

```

classdef ObjectSegmentation
methods (Static)
    function res = threshold(image, threshold_algorithm, T_input)
        % function to converting image to binary
        % based on threshold algorithm selected
        if (threshold_algorithm == "Selected Threshold")
            T = T_input/255;
            res = imbinarize(image, T);
        elseif (threshold_algorithm == "Otsu")
            T = graythresh(image);
            res = imbinarize(image, T);
        elseif (threshold_algorithm == "Adaptive")
            res = imbinarize(image, 'adaptive');
        end
    end

    function res = getSegmentationMask(image, threshold_algorithm,
T_input)
        % converting image to binary
        bwimg = ObjectSegmentation.threshold(image,
threshold_algorithm, T_input);

```

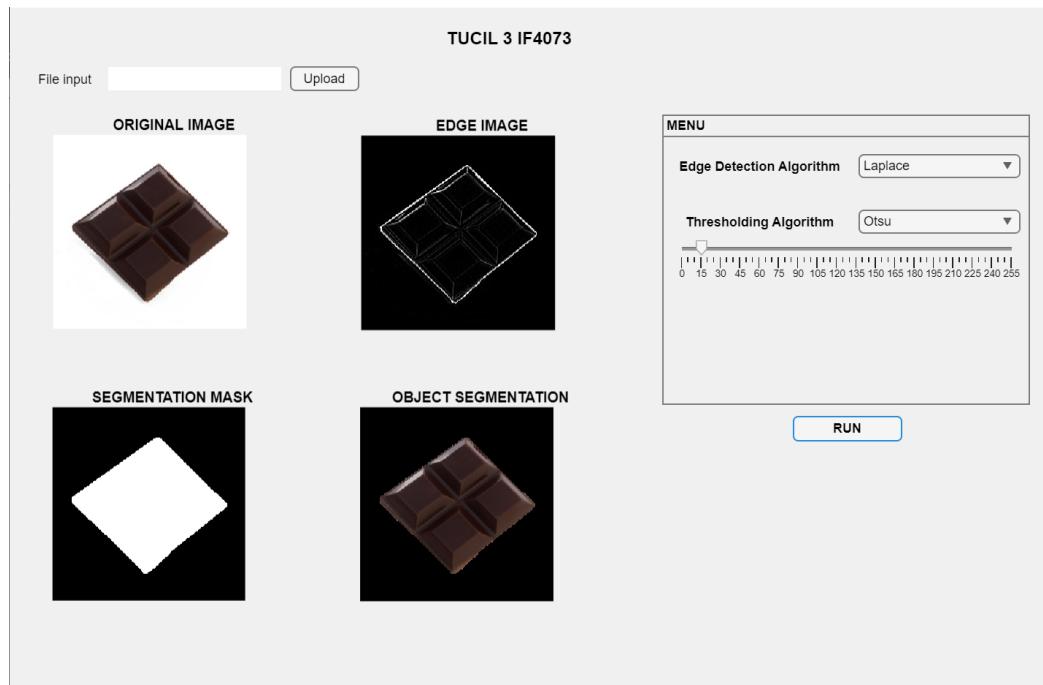
```

% get segmentation mask
% clean up the pixels on the edges of the image
res = imclearborder(bwimg);
% morphologically closing image so that adjacent edges can
be connected
res = imclose(res, strel('line', 10, 0));
% fill in the areas surrounded by connected edges
res = imfill(res, 'holes');
% morphologically open image
res = imopen(res, strel(ones(3,3)));
% remove under threshold filled area
res = bwareaopen(res, 1500);
end

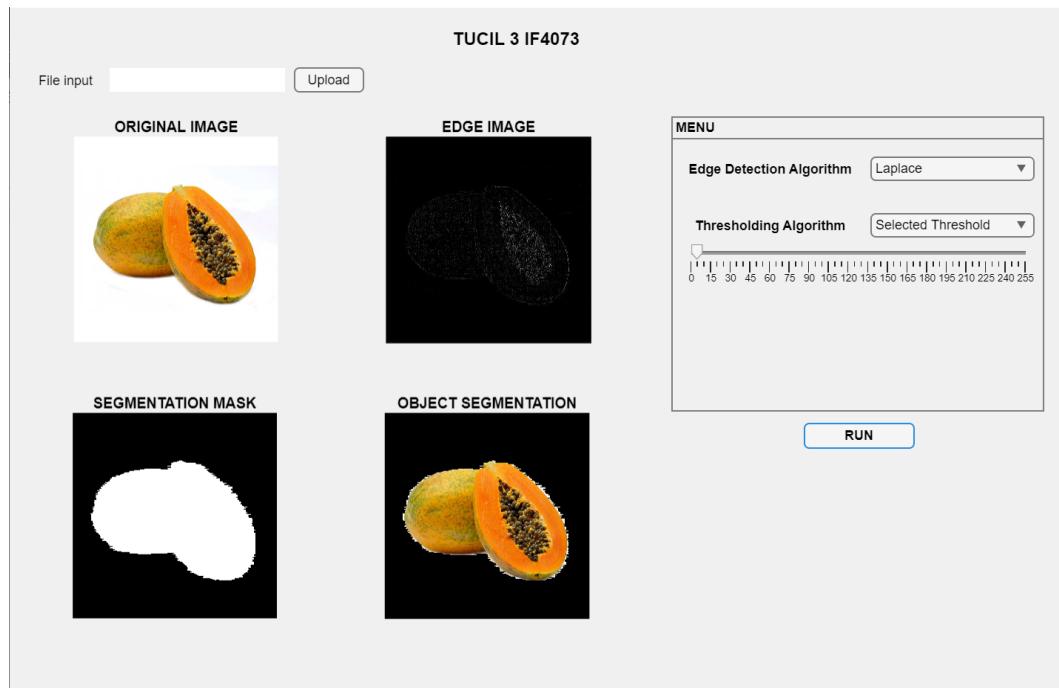
function res = segmentObject(image, segmask, isGrayscale)
if (isGrayscale)
    % if image is grayscale
    res(:,:) = image(:,:).*uint8(segmask);
else
    % if image is RGB (has 3 channels)
    res(:,:,1) = image(:,:,1).*uint8(segmask);
    res(:,:,2) = image(:,:,2).*uint8(segmask);
    res(:,:,3) = image(:,:,3).*uint8(segmask);
end
end
end

```

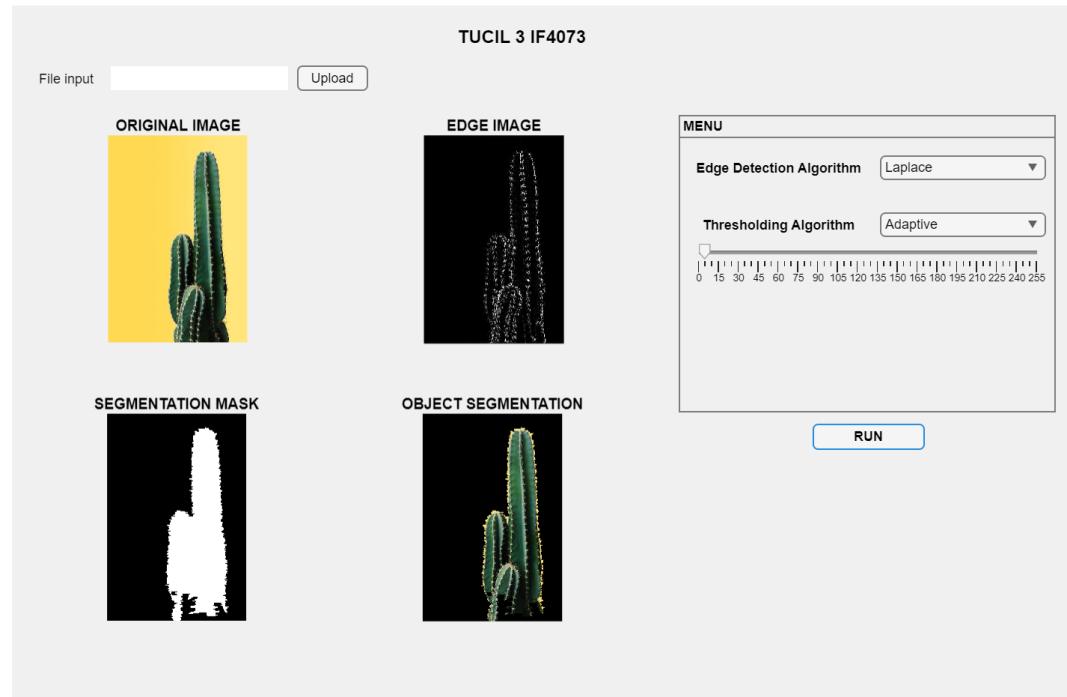
- Contoh hasil eksekusi



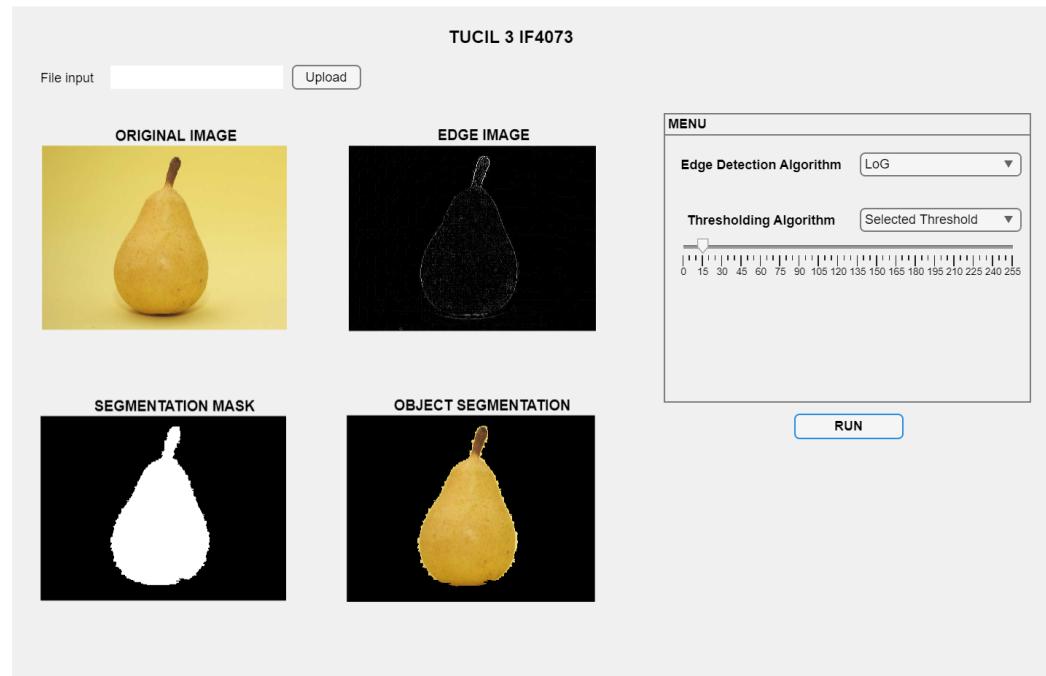
Menggunakan operator Laplace dan metode pengambangan Otsu



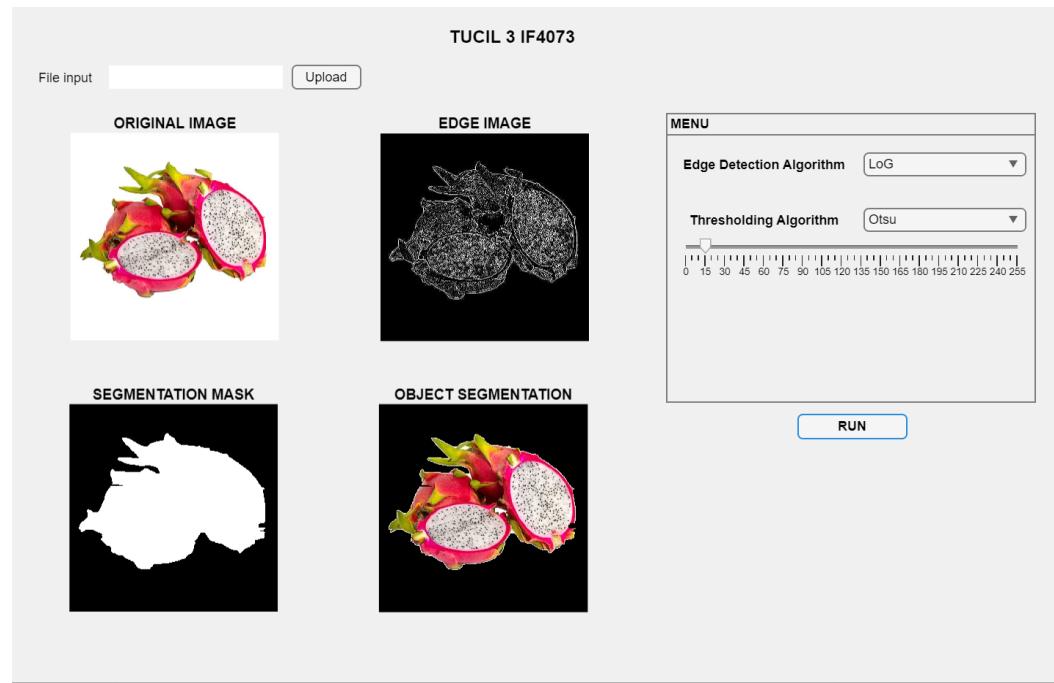
Menggunakan operator Laplace dan nilai ambang = 5



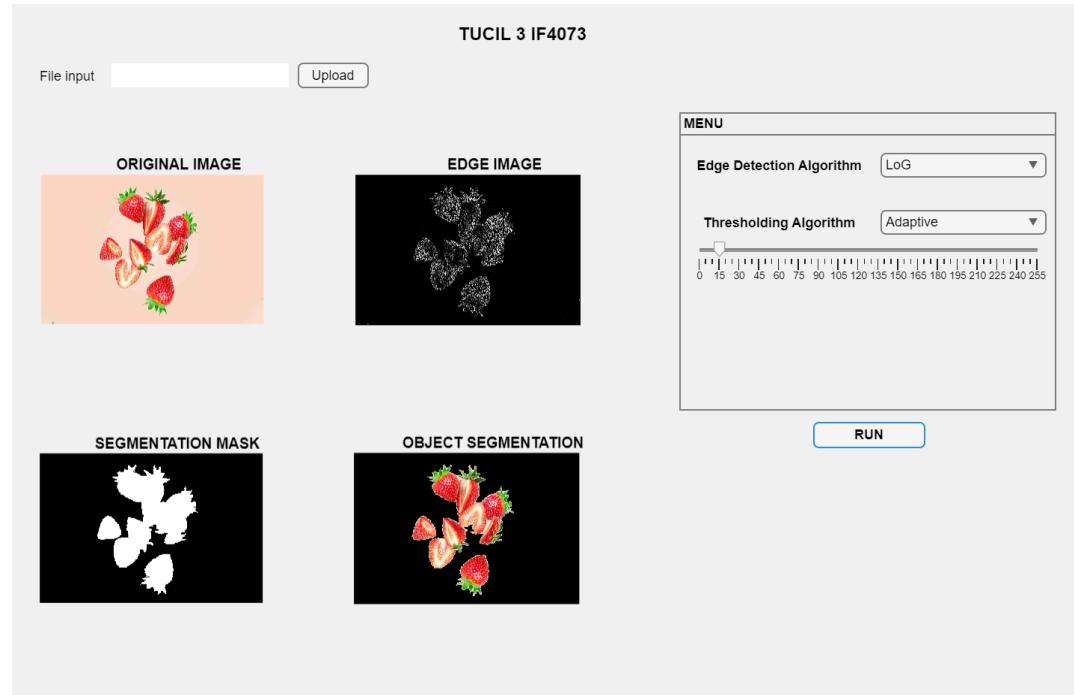
Menggunakan operator Laplace dan metode pengambangan adaptif



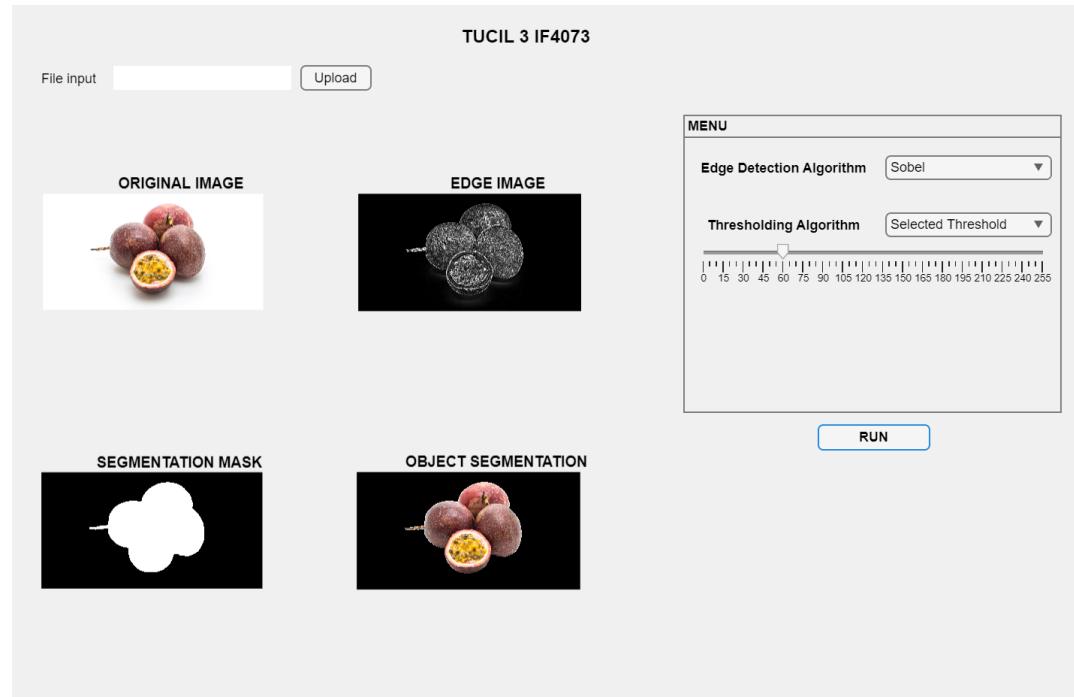
Menggunakan operator LoG dan nilai ambang = 15



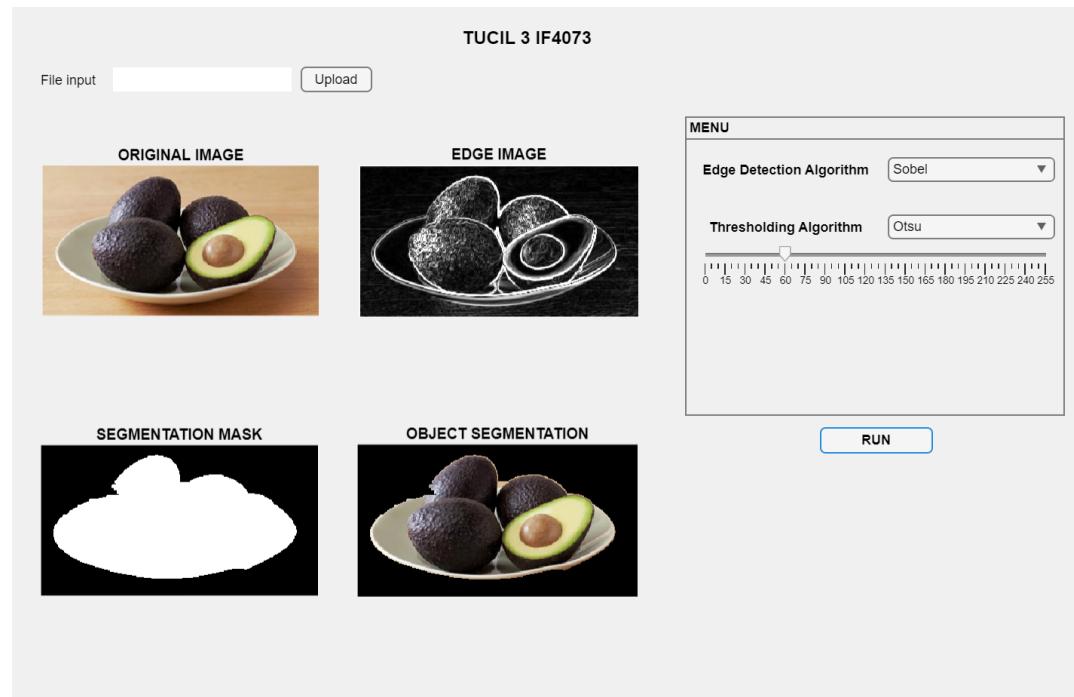
Menggunakan operator LoG dan metode pengambangan Otsu



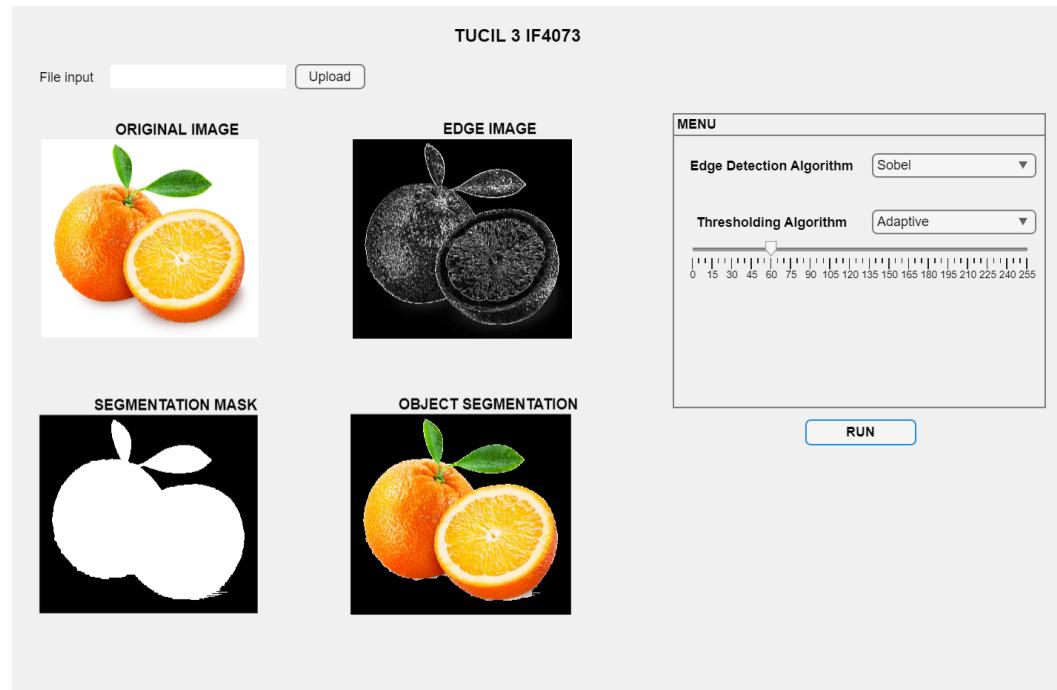
Menggunakan operator LoG dan metode pengambangan adaptif



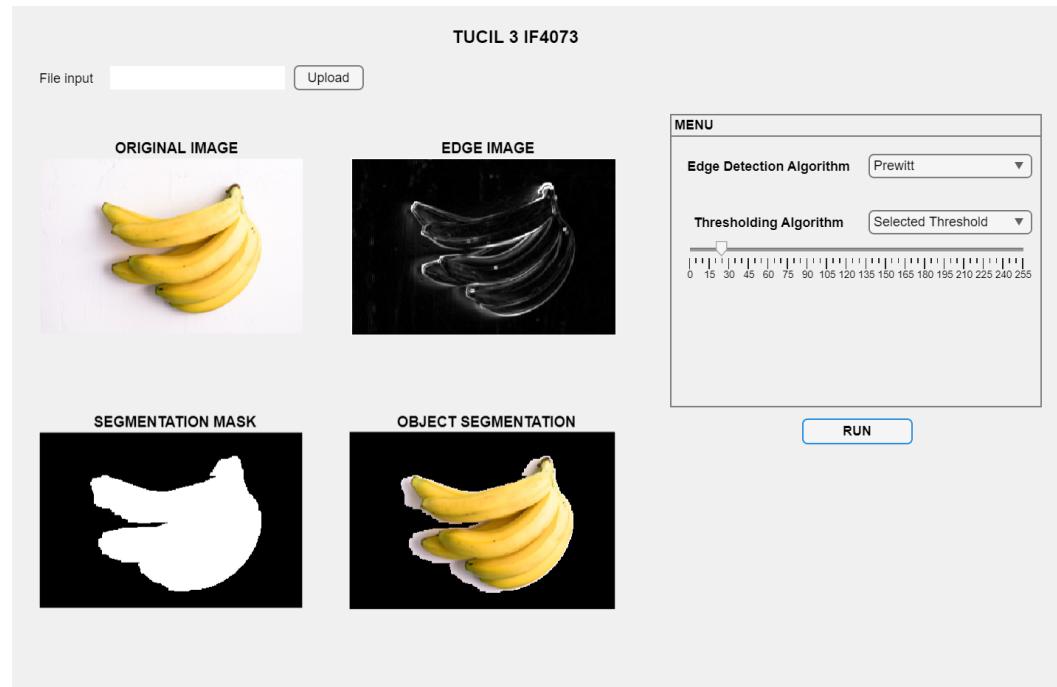
Menggunakan operator Sobel dan nilai ambang = 60



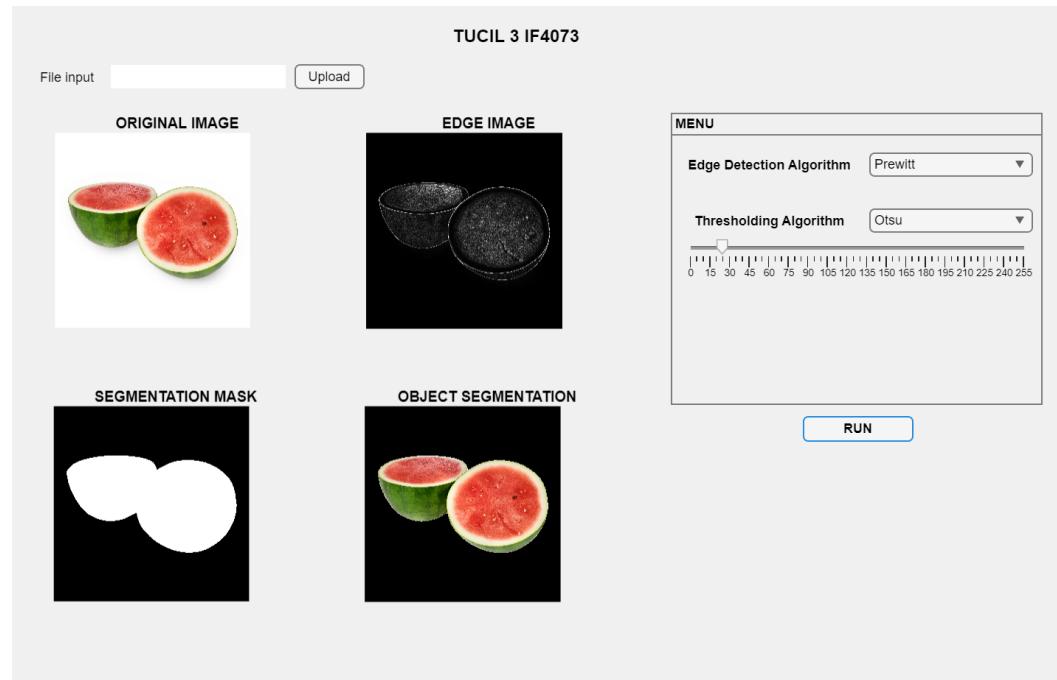
Menggunakan operator Sobel dan metode pengambangan Otsu



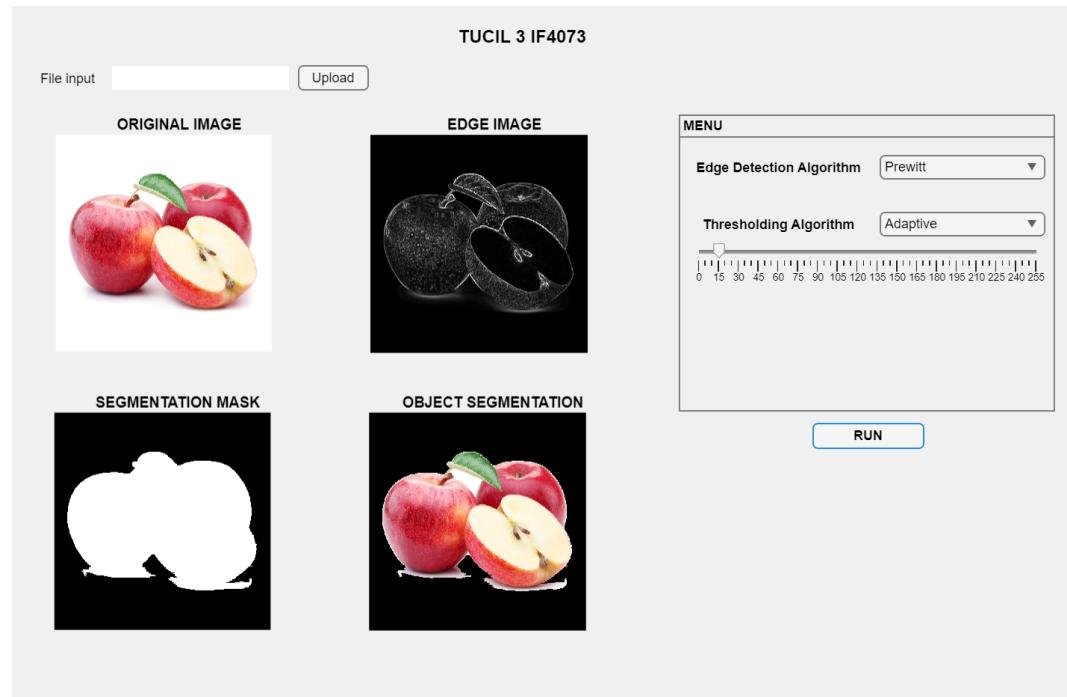
Menggunakan operator Sobel dan metode pengambangan adaptif



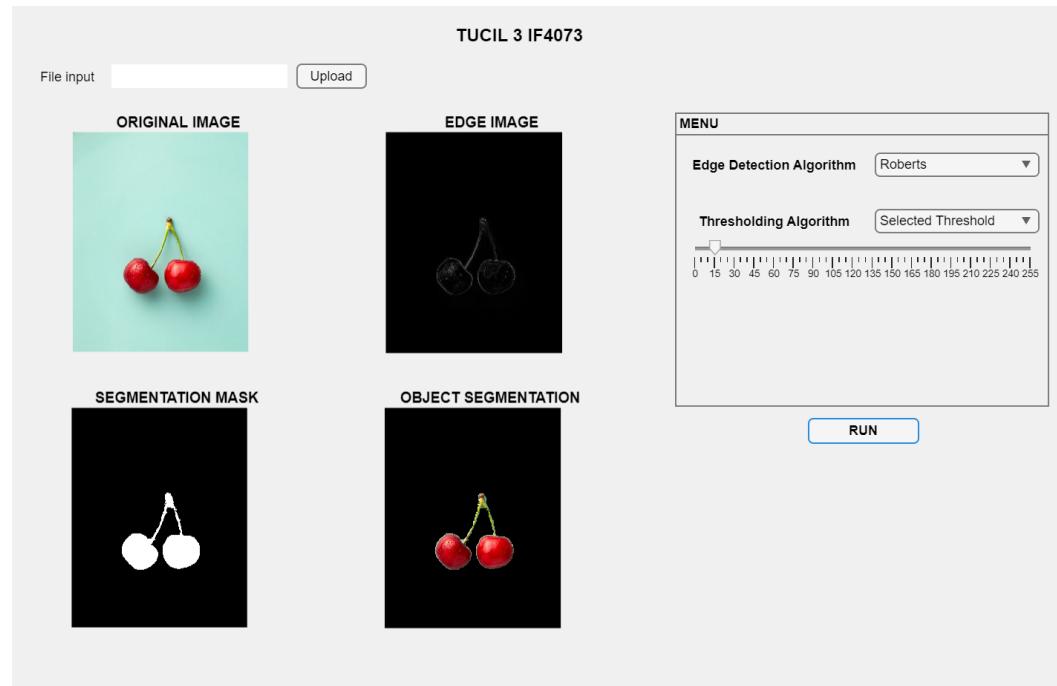
Menggunakan operator Prewitt dan nilai ambang = 25



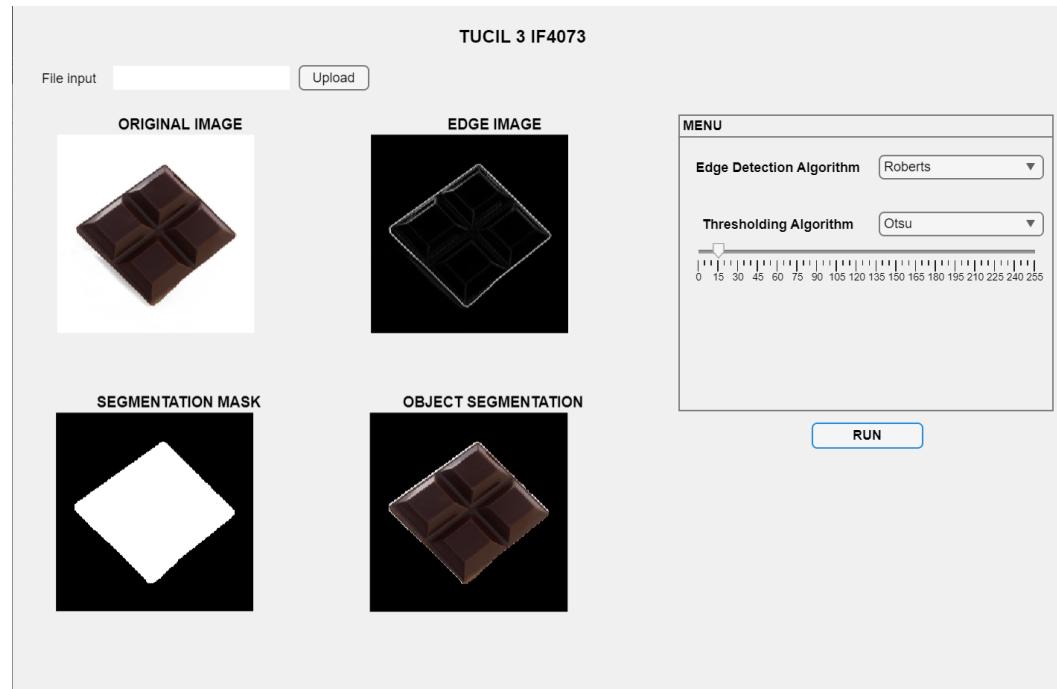
Menggunakan operator Prewitt dan metode pengambangan Otsu



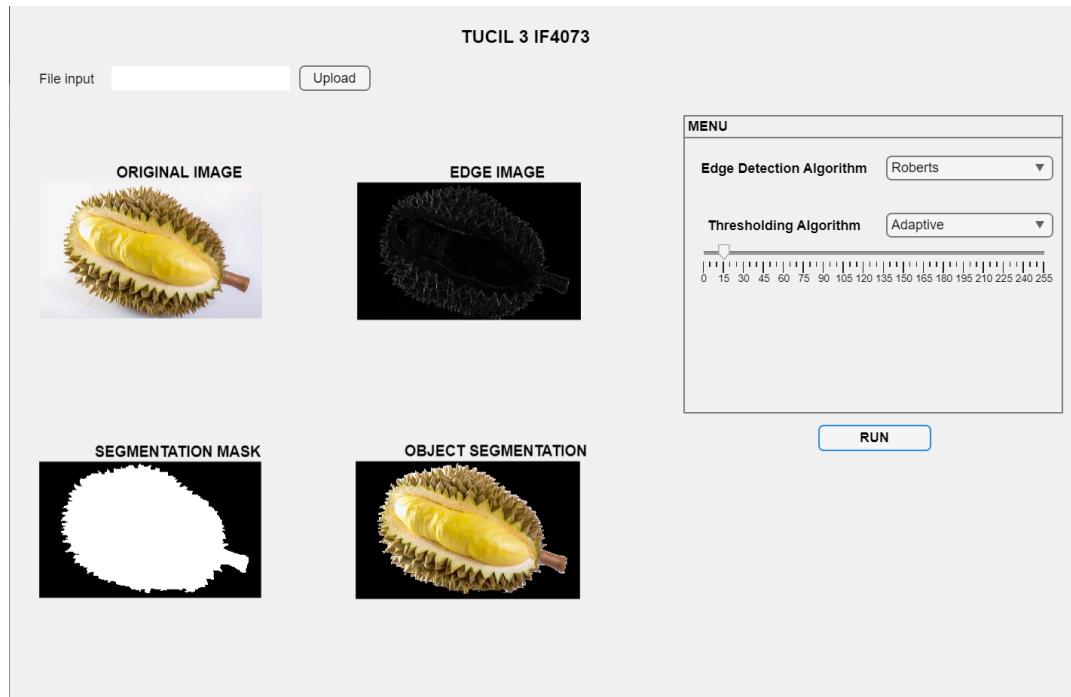
Menggunakan operator Prewitt dan metode pengambangan adaptif



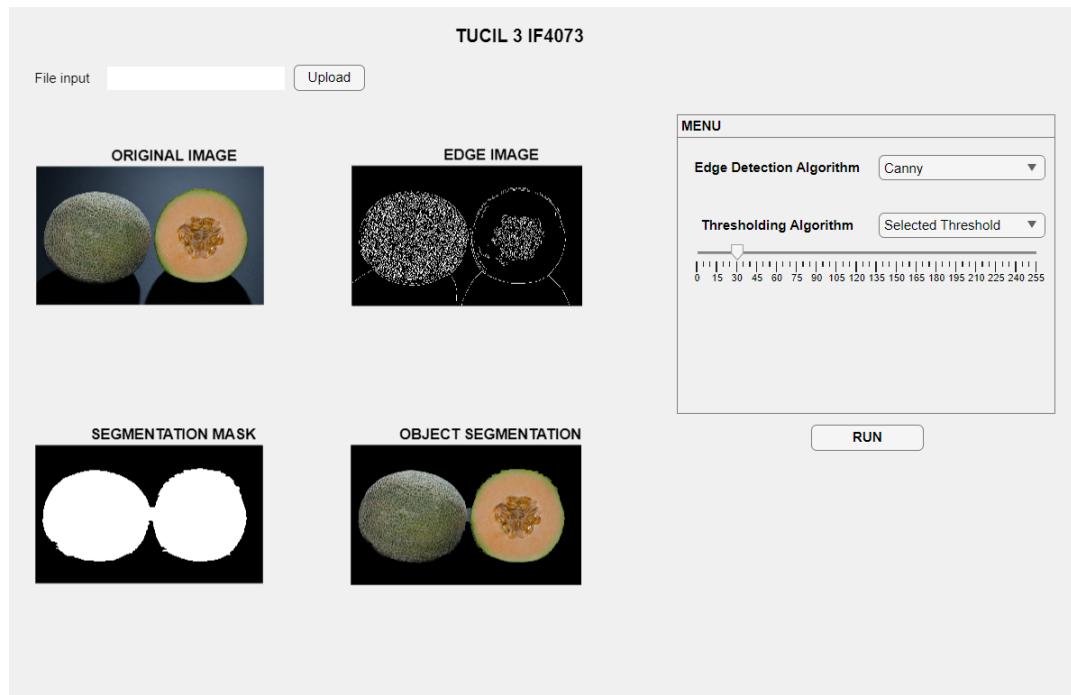
Menggunakan operator Roberts dengan nilai ambang = 15



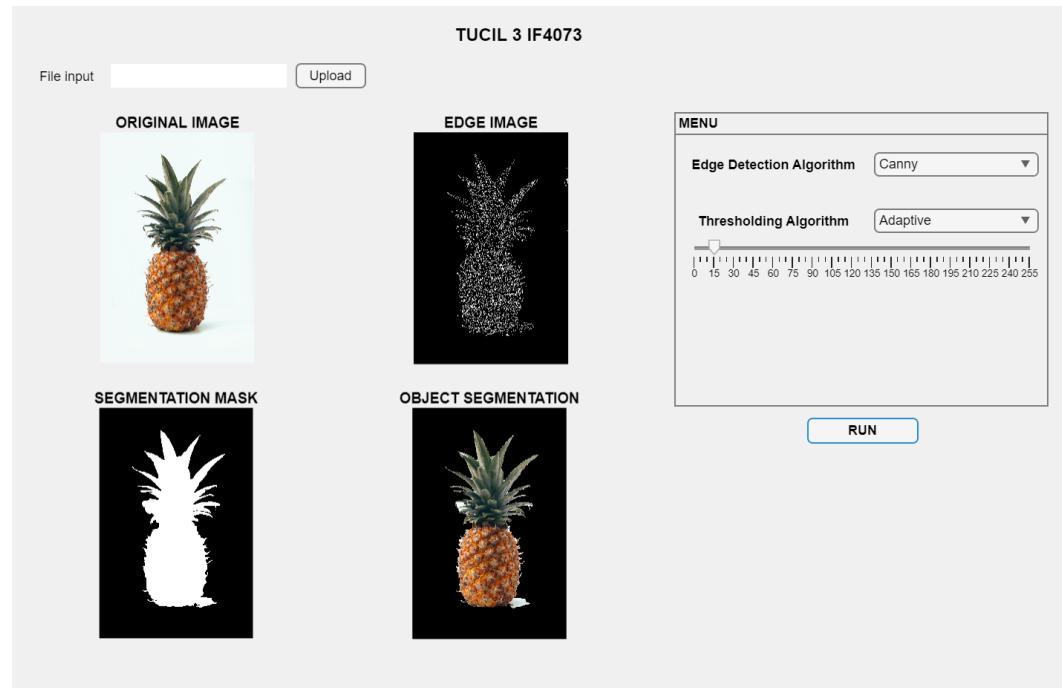
Menggunakan operator Roberts dengan metode pengambangan Otsu



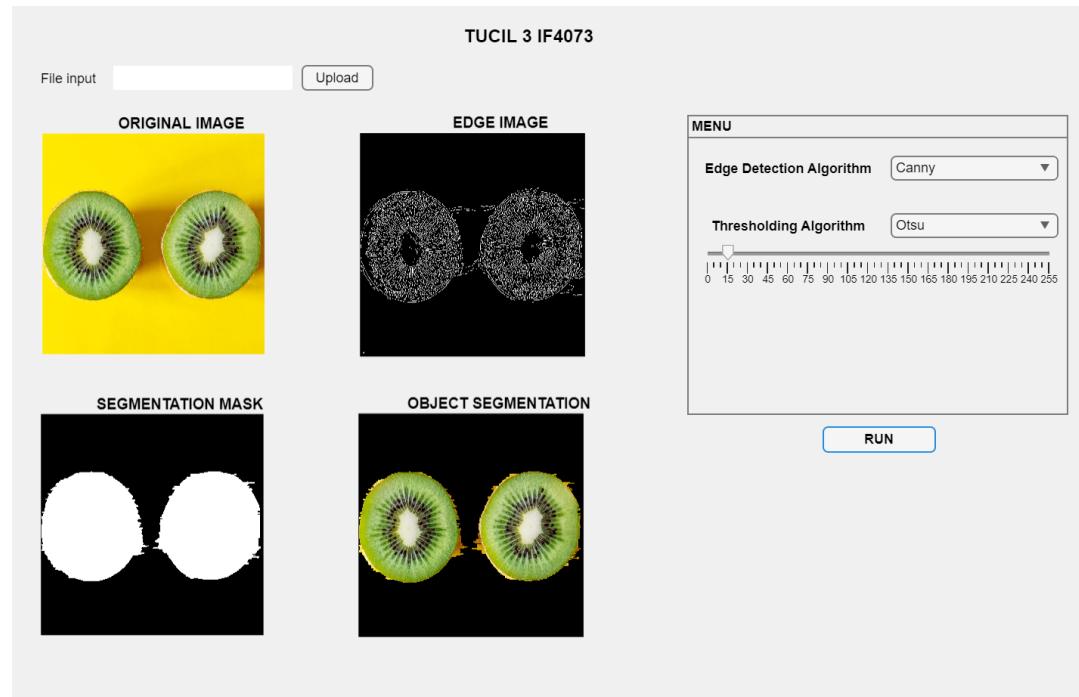
Menggunakan operator Roberts dengan metode pengembangan adaptif



Menggunakan operator Canny dan nilai ambang = 30



Menggunakan operator Canny dan metode pengambangan adaptif



Menggunakan operator Canny dan metode pengambangan Otsu

- Analisis cara kerja dan hasilnya:

Untuk mensegmentasi objek dilakukan 2 langkah berikut ini:

1. Membuat citra segmentasi

Citra segmentasi dibuat dengan memanfaatkan citra tepi yang diperoleh dengan salah satu dari enam operator *laplace*, LoG, Sobel, Prewitt, Roberts, dan Canny yang telah dijelaskan sebelumnya.

Citra tepi ini kemudian diubah menjadi citra biner dengan menggunakan salah satu metode *thresholding* seperti Otsu atau Adaptif, atau dengan menggunakan besaran *threshold* yang dipilih oleh pengguna secara manual melalui *slider* yang disediakan pada GUI.

Citra tepi yang sudah diubah menjadi citra biner kemudian diolah menjadi citra segmentasi dengan tahapan sebagai berikut.

- a. Bersihkan *pixel-pixel* pada tepi citra
- b. Lakukan operasi *morphological closing* pada citra agar tepi-tepi yang berdekatan bisa terhubung
- c. Isi daerah yang dikelilingi tepi terhubung. Dengan kata lain berikan nilai 1 untuk *pixel-pixel* yang berada di daerah yang dikelilingi tepi terhubung dan nilai 0 untuk *pixel-pixel* lainnya
- d. Lakukan operasi *morphological open* pada citra
- e. Hilangkan daerah-daerah bernilai 1 dengan luas di bawah *threshold* (ganti nilai *pixel-pixel* di daerah tersebut menjadi 0).

Hasilnya adalah citra segmentasi, yaitu citra biner yang nilai *pixel-pixel* yang bersesuaian dengan objek akan bernilai 1 sedangkan *pixel-pixel* lainnya bernilai 0. Citra segmentasi ini dapat dilihat pada GUI di axes *segmentation mask*.

2. Setiap elemen citra asli dikalikan dengan citra segmentasi sehingga menghasilkan citra objek yang sudah disegmentasi dengan *pixel-pixel* yang bersesuaian dengan objek akan memiliki nilai yang sama dengan *pixel-pixel* pada citra asli sedangkan *pixel-pixel* yang tidak bersesuaian dengan objek akan bernilai 0 (menjadi warna hitam).

Kesimpulannya, hasil segmentasi citra sudah cukup baik, terutama untuk citra dengan latar belakang polos. Namun, kami perhatikan bahwa segmentasi citra buruk untuk citra dengan latar belakang yang tidak polos. Hal ini dikarenakan pada citra dengan latar belakang yang tidak polos, latar belakang tersebut memiliki perubahan frekuensi yang besar sehingga bagian dari latar belakang tersebut terdeteksi juga sebagai tepi. Hal ini mengakibatkan citra segmentasi yang dihasilkan menjadi kurang berhasil dalam mensegmentasi objek yang diinginkan pada citra. Mungkin saja citra segmentasi memiliki nilai pixel 1 yang lebih banyak sehingga area yang terambil lebih luas. Mungkin saja citra segmentasi memiliki nilai pixel 0 yang lebih banyak (terutama jika menggunakan metode pengambangan Otsu) sehingga hanya sedikit area yang terambil. Selain itu, terdapat juga kemungkinan citra segmentasi memiliki nilai-nilai pixel yang salah karena sulit membedakan bagian dalam atau luar dari objek saat operasi *morphological closing* pada citra.

Alamat Github: https://github.com/ruhiyahfw/IF4073_tucil3