TUGAS 2

PENINGKATAN KUALITAS CITRA

Susaf Noor Azhar

18/437646/PTK/12679

VISI KOMPUTER

PROGRAM STUDI MAGISTER INFORMASI

DEPARTEMENT TEKNIK ELEKTRO DAN TEKNOLOGI INFORMASI

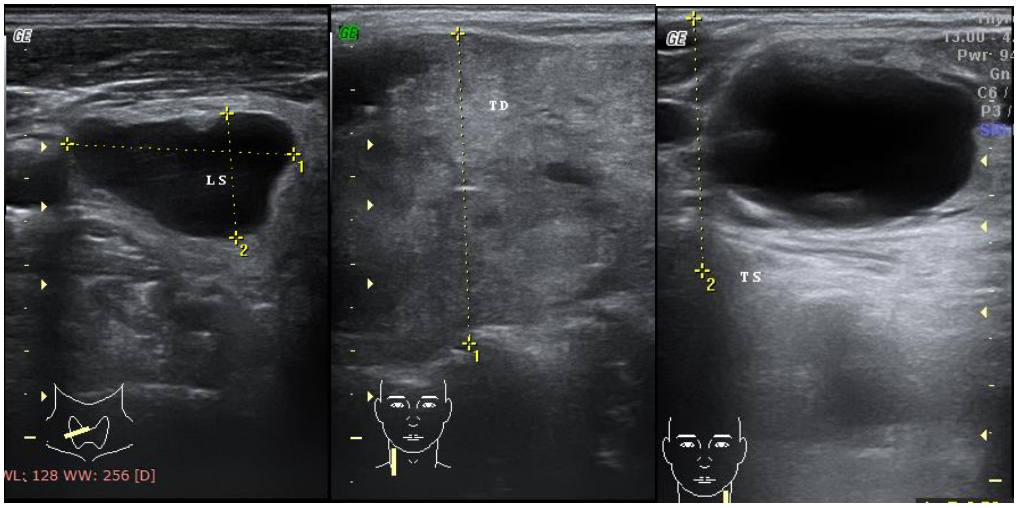
FAKULTAS TEKNIK

UNIVERSITAS GADJAH MADA

FEBRUARI 2020

1. Tujuan Eskperim

Diberikan tiga gambar seperti yang telihat pada Gambar 1 yang merupakan gambar medis berupa hasil scan X-Ray dengan beberapa macam artifak. Adapun tujuan dari percobaan ini adalah mendapatkan gambar X-Ray tanpa artifak.



Gambar 1. Tiga gambar x-ray dengan artifak berupa penanda (marking), keterangan dan label. (dari kiri image1, image2 dan image3)

2. Hipotesis

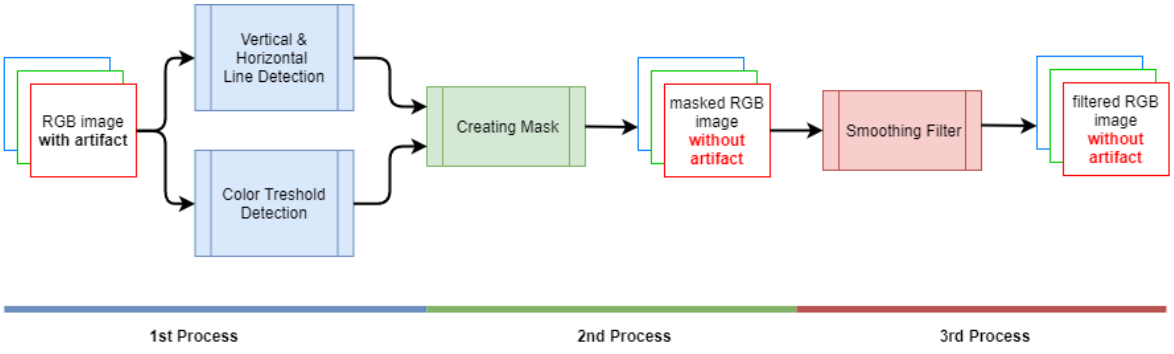
Dari gambar, artifak yang ada dapat dikelompokkan menjadi:

1. Objek bewarna,
2. Kombinasi garis vertikal dan horisontal (penanda dan tulisan)

Sehingga sistem yang dibuat harus mampu menghilangkan kedua tipe artifak tersebut.

3. Eksperimen

Pada eksperimen ini, gambar akan diolah menggunakan MATLAB. Adapun kode program yang dibuat dapat ditinjau pada bab lampiran. Adapun program yang dibuat terdiri dari dua file, yaitu file utama (A2_main.m) dan file fungsi (imageRestoration.m),



Gambar 2. Proses diagram dari sistem peningkatan citra yang dibuat, terdiri dari 3 proses utama yaitu: deteksi, masking dan smoothing.

Alur proses dari pengolahan citra dapat dilihat pada Gambar 2, dalam gambar terlihat bahwa dalam keseluruhan proses, data yang diolah berupa data RGB (*red green blue*). Adapun proses pengolahan citra dapat dibagi menjadi tiga tahap, yaitu:

3.1. Tahap 1: deteksi artifak

Tahap pertama dimulai dengan deteksi artifak berupa garis dan warna. Deteksi garis vertikal dan horisontal dilakukan menggunakan filter berupa matriks sebagai berikut:

$$vf = \begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}$$

$$hf = \begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix}$$

dimana *vf* merupakan matriks vertikal dan *hf* adalah matrik horisontal yang digunakan untuk melakukan proses filtering. Namun, menggunakan metode ini saja tidak cukup. Seperti yang digambarkan pada Gambar 3, masih terlihat artifak berupa pixel bewarna. Hal yang sama juga didapat pada Image2 dan Image3.

Gambar 3. Hasil deteksi artifak menggunakan vertical and horizontal line detection pada image1 (terlihat masih terdapat beberapa artifak bewarna yang tersisa)

Untuk itu, digunakan deteksi pixel bewarna. Proses deteksi pixel bewarna dapat dilakukan dengan melihat perbedaan nilai pixel antara setiap kanal warna merah dengan hijau dan biru.

$$color = \begin{cases} \exists\ \Delta(R,G,B) > 20 & , color = colorful \\ \forall\ \Delta(R,G,B) \leq 20 & , color = grayscale \end{cases}$$

dimana sebuah pixel dianggap bewarna apabila salah satu perbedaan antar channel lebih dari 20, dan pixel dianggar hitam putih apabila semua perbedaan antar channel memiliki nilai kurang dari 20. Selain warna, juga dilakukan deteksi pixel dengan warna yang terang (diatas 200). Adapun hasilnya terlihat pada Gambar 4 yang menunjukkan sisa artifak berhasil terdeteksi.

Dalam matlab, kedua fungsi tersebut ditulis dalam kode sebagai berikut:

Line Detection:

```
%% Image Filtering
vf = [-1 2 -1; -1 2 -1; -1 2 -1]; %vertical filter
hf = [-1 -1 -1; 2 2 2; -1 -1 -1]; %horizontal filter
% vertical lines detection
dRC_vline = filter2(vf, dRC_Ori); %double Red Channel - vertical line
dGC_vline = filter2(vf, dGC_Ori);
dBC_vline = filter2(vf, dBC_Ori);
% horizontal lines detection
dRC_hline = filter2(hf, dRC_Ori); %double Red Channel - horizontal line
dGC_hline = filter2(hf, dGC_Ori);
dBC_hline = filter2(hf, dBC_Ori);
```

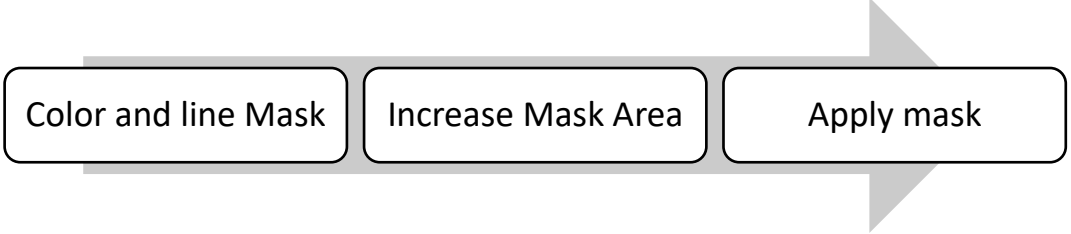
Color Detection:

```
% color treshold mask by calculate the value difference between channel
iRGC_diff = abs(int16(iRGB_Ori(:,:,1)) - int16(iRGB_Ori(:,:,2)));
%integer Red-Green Channel - difference value
iRBC_diff = abs(int16(iRGB_Ori(:,:,1)) - int16(iRGB_Ori(:,:,3)));
%integer Red-Blue Channel - difference value
iGBC_diff = abs(int16(iRGB_Ori(:,:,2)) - int16(iRGB_Ori(:,:,3)));
%integer Green-Blue Channel - difference value
% detecting coloured (>30) and grayish pixel (<30)
mask_color = (iRGC_diff>20) | (iRBC_diff>30) | (iGBC_diff>20);
% detecting bright pixel (>200)
mask_bright = iRC_Ori>=200 & iGC_Ori>=200 & iBC_Ori>= 200;
```

Gambar 4. Hasil deteksi artifak menggunakan vertical and horizontal line detection pada image1 (tidak terlihat artifak pixel bewarna)

3.2. Tahap 2: penghilangan artifak menggunakan *masking*

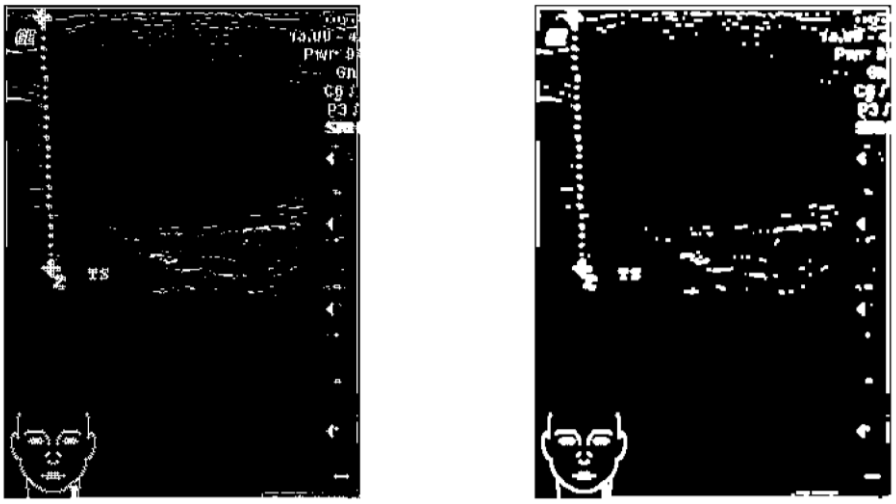
Tahap kedua adalah pembuatan mask/filter biner yang digunakan untuk menghapus artifak yang terdeteksi. Dalam proses ini terdapat tiga subsproces yaitu:



Gambar 5. Proses pembuatan dan penerapan mask pada gambar yang diolah.

Pembuatan Color and Line Mask dilakukan dengan metode sederhana, yaitu menggunakan treshold dengan hasil berupa matriks bilangan biner. Dalam proses ini didapat 4 mask yang berbeda, yaitu: `mask_color`, `mask_bright`, `mask_line`, `mask_hline`. Dimana keempatnya digabung menjadi `mask_all`.

Karena gambar merupakan file hasil kompresi PNG, maka terdapat atifak dari proses anti-aliasing yang ada. Untuk menghilangkanya, maka area mask yang ada perlu diperluas. Proses ini dapat dilakukan dengan menerapkan mean filter pada matrik mask.



Gambar 6. Perbedaan mask tanpa anti aliasing (AA) dengan mask dengan AA. Terlihat mask dengan AA memiliki area yang lebih luas.

Dalam MATLAB, kedua proses diatas ditulis sebagai berikut:

Color and Line Mask:

```
mask_color = (iRGC_diff>20) | (iRBC_diff>30) | (iGBC_diff>20);
% detecting bright pixel (>200)
mask_bright = iRC_Ori>=200 & iGC_Ori>=200 & iBC_Ori>= 200;
% merge two mask
mask_color = mask_color | mask_bright;

% line treshold mask
mask_line = (dRC_vline>0.4) | (dGC_vline>0.4) | (dBC_vline>0.4);
mask_hline = (dRC_hline>0.5) | (dGC_hline>0.5) | (dBC_hline>0.5);
mask_line = mask_line | mask_hline;
mask_all = mask_line | mask_color; %merge all mask
```

Increase Mask Area:

```
% increase masked area
m = [1 1 1; 1 1 1; 1 1 1]/9; %mean value
nmask = filter2(m, mask_all);
mask_all = nmask>0.2;
```

Setelah didapat mask yang diinginkan, proses selanjutnya adalah penerapan mask yang ada pada gambar yang diolah. Implementasinya dalam program dapat dilihat pada kode dibawah.

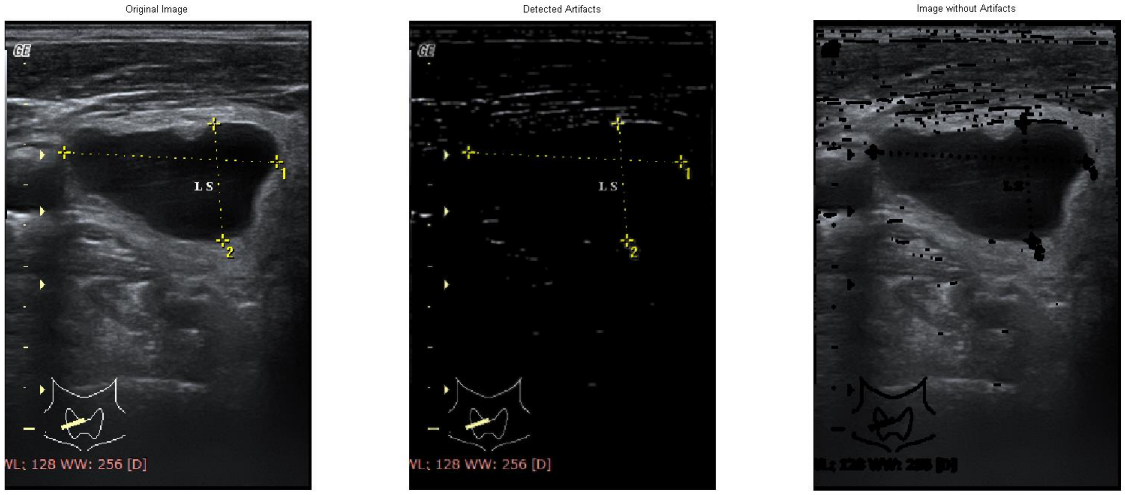
Artifact Mask:

```
%% Image Masking
imgRC_masked = dRC_Ori.*nmask;
imgGC_masked = dGC_Ori.*nmask;
imgBC_masked = dBC_Ori.*nmask;
imgMasked = cat(3, imgRC_masked, imgGC_masked, imgBC_masked); %masked
image
```

Inverse Artifact Mask (image without artifact):

```
invfmask = ones(size(mask_all)) - mask_all; %inverse masked area
imgRC_masked_inv = dRC_Ori.*invfmask;
imgGC_masked_inv = dGC_Ori.*invfmask;
imgBC_masked_inv = dBC_Ori.*invfmask;
imgMasked_inv = cat(3, imgRC_masked_inv, imgGC_masked_inv,
imgBC_masked_inv); %inverse of masked area
```

Hasil dari penerapan masking ini dapat dilihat pada Gambar 7-Gambar 9. Gambar kiri merupakan gambar asli yang mengandung artifak, gambar tengah merupakan artifak yang terdeteksi dan gambar kanan merupakan gambar tanpa artifak sebelumnya.



Gambar 7. Deteksi dan penghapusan artifak pada image1 (kiri: gambar asli, tengah: artifak yang terdeksi, kanan: gambar tanpa artifak)

Gambar 8. Deteksi dan penghapusan artifak pada image2 (kiri: gambar asli, tengah: artifak yang terdeksi, kanan: gambar tanpa artifak)

Gambar 9. Deteksi dan penghapusan artifak pada image3 (kiri: gambar asli, tengah: artifak yang terdeksi, kanan: gambar tanpa artifak)

Terlihat dari hasil, bahwa semua artifak mampu dideteksi, namun beberapa bagian yang bukan artifak ikut terdeteksi dan proses penghilangan artifak ini menimbulkan artifak baru berupa warna hitam. Untuk mengatasi masalah ini, perlu dilanjutkan dengan proses selanjutnya.

3.3. Tahap 3: peningkatan kualitas citra menggunakan *smooth filter*

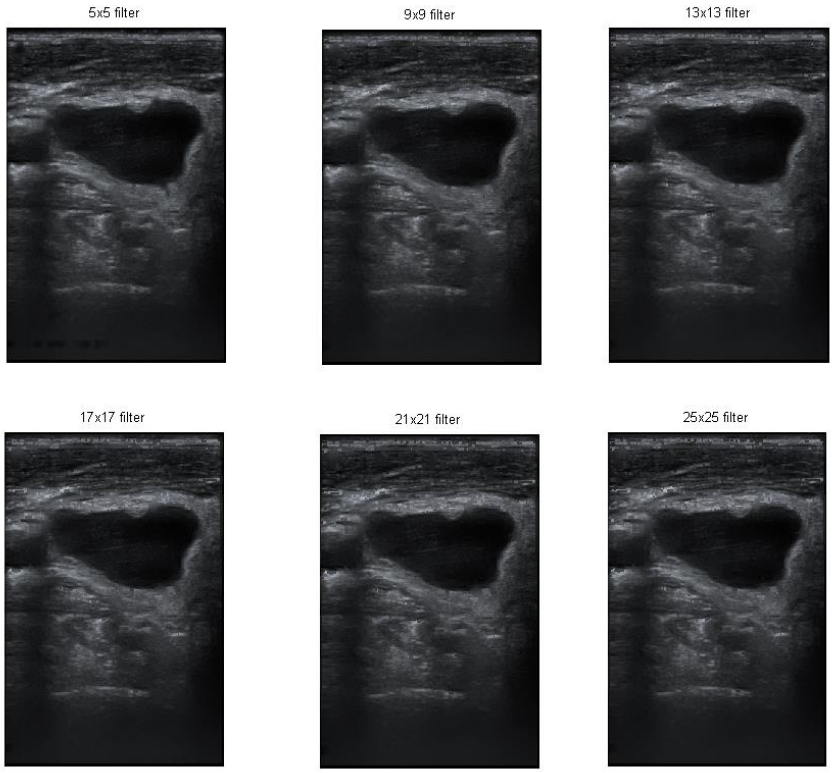
Pada proses ini, dilakukan perbaikan gambar dengan smooth filter, yang berupa matriks of one (berisi 1).

$$mf_{(n,n)} = \begin{bmatrix} 1 & \cdots & 1_{(1,n)} \\ \vdots & \ddots & \vdots \\ 1_{(n,1)} & \cdots & 1_{(n,n)} \end{bmatrix}$$

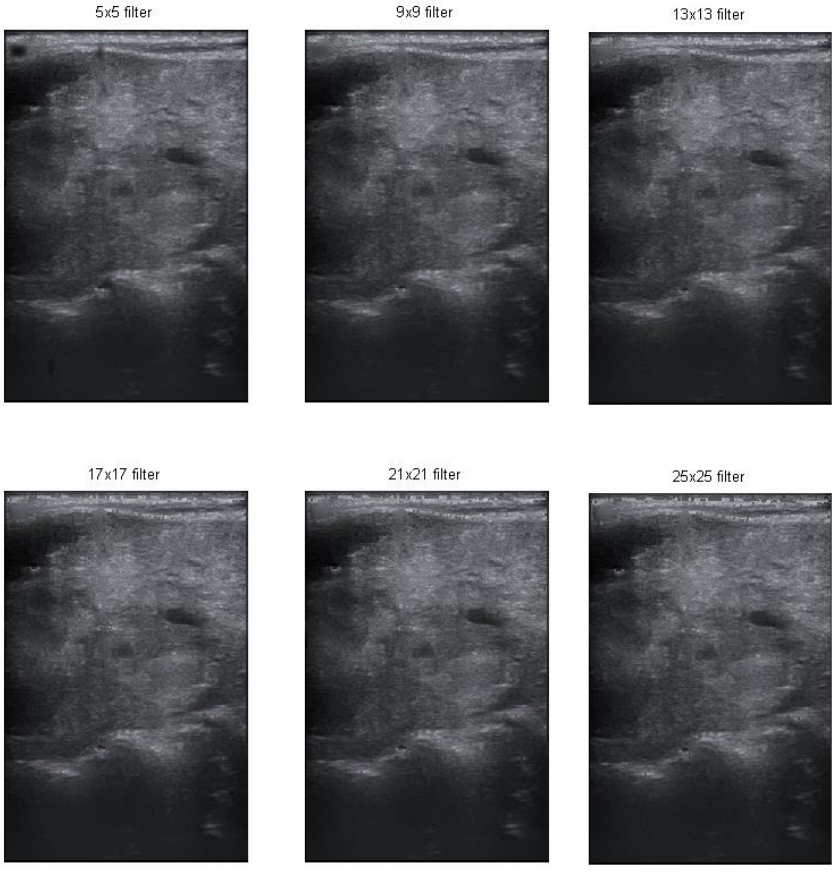
dimana n merupakan ukuran matriks. Pada proses filtering pertama, nilai elemen tengah matriks dibuat bernilai 0 agar menghilangkan berapapun nilai aslinya sedangkan pada proses filtering selanjutnya, elemen tengah tetap bernilai 1.

Sebagai percobaan, dipilih beberapa ukuran filter yang berbeda, yaitu: 5x5, 9x9, 13x13, 17x17, 21x21 dan 25x25. Dalam percobaan ini setiap filter diulang sebanyak 10x. Hasil dari percobaan ini terlihat pada Gambar 10-Gambar 12.

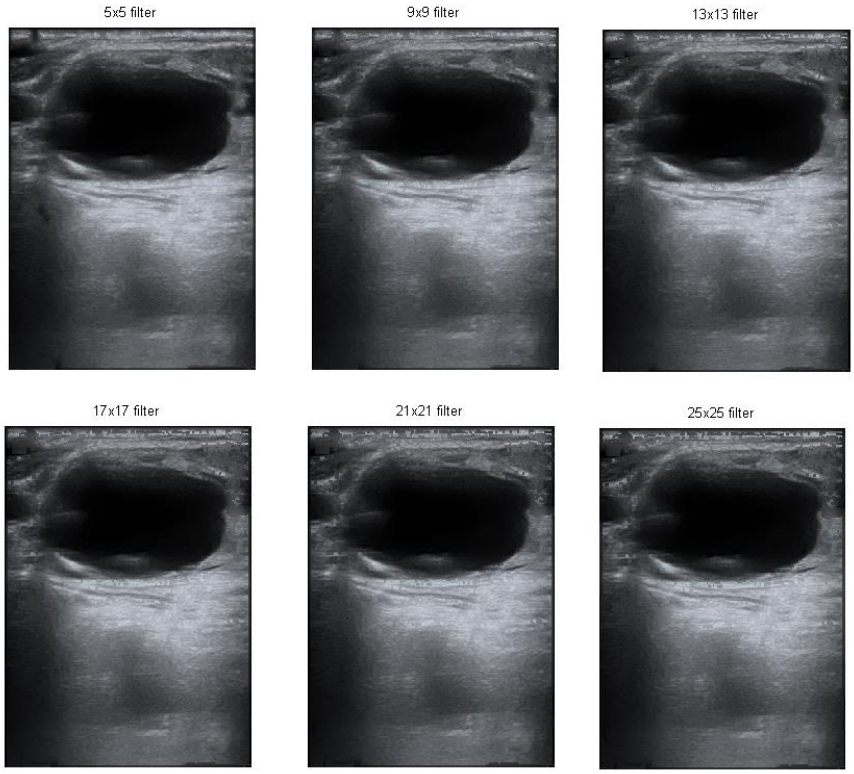
Dari hasil nampak bahwa bagian yang sebelumnya kosong dapat diisi dengan nilai pixel baru. Namun, masing-masing ukuran filter memberikan hasil yang sedikit berbeda.



Gambar 10. Percobaan menggunakan ukuran filter yang berbeda (5-25) untuk image1

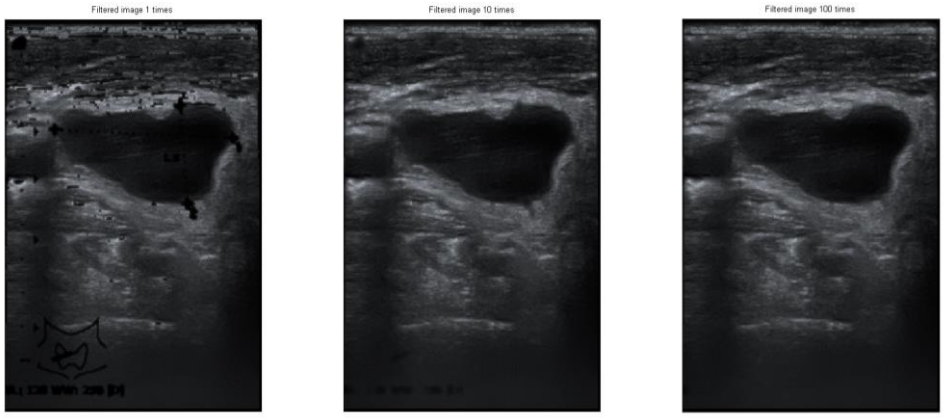


Gambar 11. Percobaan menggunakan ukuran filter yang berbeda (5-25) untuk image2



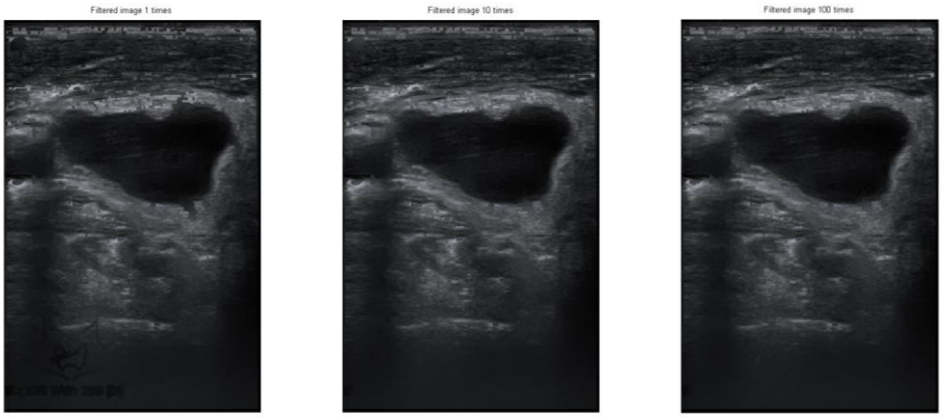
Gambar 12. Percobaan menggunakan ukuran filter yang berbeda (5-25) untuk image3

Dari percobaan, dipilih ukuran matrix 5x5 dan 25x25 sebagai nilai ekstrim atas dan bawa untuk dibandingkan lebih lanjut. Gambar 13-21 merupakan hasil dari eksperimen yang dilakukan.



Gambar 13. Implementasi filter 5x5 - 1,10 dan 100 kali untuk image1

Pada filter 5x5, bagian yang hilang dapat secara signifikan tersamarkan setelah dilakukan 100x proses.



Gambar 14. Implementasi filter 25x25 - 1,10 dan 100 kali untuk image1

Sedangkan pada filter 25x25, pixel yang hilang mampu diperbaiki secara cepat pada loop 1 dan 10 dibandingkan menggunakan 5x5 filter, namun tidak mampu mendapatkan detail yang lebih baik bahkan setelah 100x proses.

Gambar 15. Implementasi filter 5x5 - 1,10 dan 100 kali untuk image1 (diperbesar)

Gambar 16. Implementasi filter 25x25 - 1,10 dan 100 kali untuk image1 (diperbesar)

Hal ini diperjelas lagi pada Gambar 15 dan Gambar 16, yang merupakan perbesaran (zoom) pada daerah tertentu. Terlihat filter 5x5 mampu memberikan detail yang lebih baik.

Gambar 17. Implementasi filter 5x5 - 1,10 dan 100 kali untuk image2

Pada image2, filter 5x5 dan 25x25 tidak menunjukkan perbedaan signifikan. Sehingga cukup ditampilkan satu gambar saja.

Gambar 18. Implementasi filter 5x5 - 1,10 dan 100 kali untuk image3

Gambar 19. Implementasi filter 25x25 - 1,10 dan 100 kali untuk image3

Gambar 20. Implementasi filter 5x5 - 1,10 dan 100 kali untuk image3 (diperbesar)

Gambar 21. Implementasi filter 25x25 - 1,10 dan 100 kali untuk image3 (diperbesar)

Sama seperti pengolahan pada image1, pada image3 terlihat bahwa filter 5x5 mampu menangani detail lebih baik dibandingkan filter 25x25.

3.4. Improvisasi: Metode dynamic filter size

Dengan menggabungkan kekurangan dan kelebihan dari dua ukuran filter yang berbeda, yaitu filter ukuran besar mampu menyamarkan dengan cepat namun kurang akurat, sedangkan filter kecil mampu memberikan akurasi namun lebih lambat.

Dibuat sebuah filter baru yang memiliki ukuran filter yang dinamis. Pada metode ini, ukuran filter berubah mengikuti nilai berikut: [21 19 17 15 13 11 9 7 5 3].

Adapun kode untuk metode ini adalah sebagai berikut:

```
% dynamic size filter
figure('name',['Filtered image using dynamic dimensional filter']);
j = 10;
k = j+1;
for i = 1:j
    fsize = (k-i)*2+1;

    if i == 1
        imgOut = imageRestoration(imgMasked_inv,mask_all,fsize,0,1);
    else
        imgOut = imageRestoration(imgOut,mask_all,fsize,1,1);
    end

    if i == 1
        subplot(131);imshow(imgOut);
```

```
        title(['Filtered image ', num2str(i) ,' times']);
    elseif i==floor(j/2)
        subplot(132);imshow(imgOut);
        title(['Filtered image ', num2str(i) ,' times']);
    elseif i==j
        subplot(133);imshow(imgOut);
        title(['Filtered image ', num2str(i) ,' times']);
    end
end
```

Adapun hasilnya memiliki kehalusan yang sama dengan 5x5 filter namun memiliki kecepatan proses yang lebih cepat (hanya butuh 10x proses).

Gambar 22. Hasil menggunakan dynamic size filter

4. Kesimpulan

Penggunaan metode edge detection dan treshold mampu mendeteksi artifak yang ada namun terdapat beberapa bagian dari gambar yang bukan artifak turut terdeteksi. Sehingga ditambahkan metode deteksi warna.

Proses segmentasi menggunakan masking mampu menghapus artifak yang terdeteksi, pada proses ini perlu diperhatikan adanya artifak akibat proses Anti-Aliasing pada gambar yang terlah mengalami kompresi. Pada proses ini didapat gambar tanpa artifak lama, digantikan dengan pixel yang bernilai kosong (0 / hitam).

Pemulihan bagian yang kosong dapat dilakukan dengan mean filter. Proses ini mengambil nilai dari rata-rata pixel disekitar pixel yang diinginkan. Walaupun tidak dapat memulihkan informasi yang hilang secara 100%, sistem ini mampu mengurangi bekas dari artifak.

Filter dengan matriks ukuran besar mampu menghaluskan pixel dengan cepat namun memiliki akurasi yang kurang dibandingkan filter dengan ukuran matriks kecil. Kombinasi keduanya dengan menggunakan filter berukuran dinamis (dari ukuran besar ke ke kecil) dapat memperoleh hasil yang cukup akurat dengan cepat.

5. Lampiran

```
% Image Enhancement
% By: Susaf N.A
% Computer Vision Class
% February 2020 MTI UGM

%% Inizialization and data acquisition
clc; %clearing command window
clear all; close all;  %clear all variable and close all window
fontSize = 30; %label font size

% load image name with the extention type
%filename = 'img_1.png';
%filename = 'img_2.png';
filename = 'img_3.png';

% Read in indexed image.
[imgOri, storedColorMap] = imread(filename);
[rows, columns, numberOfColorChannels] = size(imgOri);

% Checking if RGB indexed or not
if numberOfColorChannels <3
    % Convert to RGB image by appying the colormap
    iRGB_Ori = ind2rgb(imgOri, storedColorMap);
else
    % Use original if already indexed
    iRGB_Ori = imgOri;
end

%% Data preprocessing

% separating by channel
% Extract the individual red, green, and blue color channels.
% int type
iRC_Ori = iRGB_Ori(:, :, 1); %integer Red Channel - Original value
iGC_Ori = iRGB_Ori(:, :, 2); %integer Green Channel - Original value
iBC_Ori = iRGB_Ori(:, :, 3); %integer Blue Channel - Original value

% double type
dRGB_Ori = im2double(iRGB_Ori);
dRC_Ori = dRGB_Ori(:,:,1); %double Red Channel - Original value
dGC_Ori = dRGB_Ori(:,:,2); %double Green Channel - Original value
dBC_Ori = dRGB_Ori(:,:,3); %double Blue Channel - Original value

%% Image Filtering
vf = [-1 2 -1; -1 2 -1; -1 2 -1]; %vertical filter
hf = [-1 -1 -1; 2 2 2; -1 -1 -1]; %horizontal filter

% vertical lines detection
dRC_vline = filter2(vf, dRC_Ori); %double Red Channel - vertical line
dGC_vline = filter2(vf, dGC_Ori);
dBC_vline = filter2(vf, dBC_Ori);
% horizontal lines detection
dRC_hline = filter2(hf, dRC_Ori); %double Red Channel - horizontal line
dGC_hline = filter2(hf, dGC_Ori);
dBC_hline = filter2(hf, dBC_Ori);

%% Creating Mask
```

```
% color treshold mask by calculate the value difference between channel
iRGC_diff = abs(int16(iRGB_Ori(:,:,1)) - int16(iRGB_Ori(:,:,2)));
%integer Red-Green Channel - difference value
iRBC_diff = abs(int16(iRGB_Ori(:,:,1)) - int16(iRGB_Ori(:,:,3)));
%integer Red-Blue Channel - difference value
iGBC_diff = abs(int16(iRGB_Ori(:,:,2)) - int16(iRGB_Ori(:,:,3)));
%integer Green-Blue Channel - difference value
% detecting coloured (>30) and grayish pixel (<30)
mask_color = (iRGC_diff>20) | (iRBC_diff>30) | (iGBC_diff>20);
% detecting bright pixel (>200)
mask_bright = iRC_Ori>=200 & iGC_Ori>=200 & iBC_Ori>= 200;
% merge two mask
mask_color = mask_color | mask_bright;

% line treshold mask
mask_line = (dRC_vline>0.4) | (dGC_vline>0.4) | (dBC_vline>0.4);
mask_hline = (dRC_hline>0.5) | (dGC_hline>0.5) | (dBC_hline>0.5);
mask_line = mask_line | mask_hline;
mask_all = mask_line | mask_color; %merge all mask

% increase masked area
m = [1 1 1; 1 1 1; 1 1 1]/9; %median value
nmask = filter2(m, mask_all);
mask_all = nmask>0.2;

%% Image Masking
imgRC_masked = dRC_Ori.*mask_all;
imgGC_masked = dGC_Ori.*mask_all;
imgBC_masked = dBC_Ori.*mask_all;
imgMasked = cat(3, imgRC_masked, imgGC_masked, imgBC_masked); %masked
image

invfmask = ones(size(mask_all)) - mask_all; %inverse masked area
imgRC_masked_inv = dRC_Ori.*invfmask;
imgGC_masked_inv = dGC_Ori.*invfmask;
imgBC_masked_inv = dBC_Ori.*invfmask;
imgMasked_inv = cat(3, imgRC_masked_inv, imgGC_masked_inv,
imgBC_masked_inv); %inverse of masked area

%% Show Masked Image
subplot(131);imshow(imgOri);
title('Original Image');
subplot(132);imshow(imgMasked);
title('Detected Artifacts');
subplot(133);imshow(imgMasked_inv);
title('Image without Artifacts');

%% Image Restoration

j = 1;

%testing different size of filter
%imgOut_comp = zeros(rows, columns, numberOfColorChannels, 6);
%{
for i = 5%:4:25
    fsize = i;
    n = 100;
    imgOut = imageRestoration(imgMasked_inv,mask_all,fsize,0,1);
    imgOut2 = imageRestoration(imgOut,mask_all,fsize,1,10);
```

```
    imgOutN = imageRestoration(imgOut2,mask_all,fsize,1,n-10);
    %imgOut_comp(:,:,:,j) = imgOutN;

    figure('name',['Filtered image using ', num2str(fsize),' dimensional
filter']);
    subplot(131);imshow(imgOut);
    title('Filtered image 1 times');
    subplot(132);imshow(imgOut2);
    title('Filtered image 10 times');
    subplot(133);imshow(imgOutN);
    title(['Filtered image ', num2str(n) ,' times']);
    j = j+1;
end
%}
%{
figure();
subplot(231); imshow(imgOut_comp(:,:,:,1)); title('5x5 filter');
subplot(232); imshow(imgOut_comp(:,:,:,2)); title('9x9 filter');
subplot(233); imshow(imgOut_comp(:,:,:,3)); title('13x13 filter');
subplot(234); imshow(imgOut_comp(:,:,:,4)); title('17x17 filter');
subplot(235); imshow(imgOut_comp(:,:,:,5)); title('21x21 filter');
subplot(236); imshow(imgOut_comp(:,:,:,6)); title('25x25 filter');
%}

% dynamic size filter
figure('name',['Filtered image using dynamic dimensional filter']);
j = 10;
k = j+1;
for i = 1:j
    fsize = (k-i)*2+1;

    if i == 1
        imgOut = imageRestoration(imgMasked_inv,mask_all,fsize,0,1);
    else
        imgOut = imageRestoration(imgOut,mask_all,fsize,1,1);
    end

    if i == 1
        subplot(131);imshow(imgOut);
        title(['Filtered image ', num2str(i) ,' times']);
    elseif i==floor(j/2)
        subplot(132);imshow(imgOut);
        title(['Filtered image ', num2str(i) ,' times']);
    elseif i==j
        subplot(133);imshow(imgOut);
        title(['Filtered image ', num2str(i) ,' times']);
    end
end
```

```
% Image Enhancement
% By: Susaf N.A
% Computer Vision Class
% February 2020 MTI UGM

function [out_img] = imageRestoration(in_img,mask,fsize,type,n)
%imageRestoration function: generating restored image using median filter
%  -Usage-
%   [out_img]: imageRestoration(in_img,mask,fsize,type,n)
%
%  -Inputs-
%    in_img: original masked image (double RGB channel)
```

```
%    mask: masked area/region of interest (binary)
%    fsize: size of median filter (odd)
%    n: number of filter loop (int>0)
%  -Outputs-
%    out_Wiener:
%Author: Susaf Noor Azhar, University of Gadjah Mada

% error check
if n<1
    n = 1;
end
if ~mod(fsize,2)
    fsize = fsize+1;
end

% separating RGB channel
RC = in_img(:, :, 1);
GC = in_img(:, :, 2);
BC = in_img(:, :, 3);

% creating median filter
mid = (fsize+1)/2;
mf = ones(fsize,fsize);

if type == 0
    mf(mid,mid) = 0; %zero at center value
    mf = mf/(fsize*fsize-1);
else
    mf = mf/(fsize*fsize); %one at center value
end

% filtering loop
for i=1:n
    % filtering image at Region of Interest (ROI) using roifit
    RC = roifilt2(mf, RC, mask);
    GC = roifilt2(mf, GC, mask);
    BC = roifilt2(mf, BC, mask);
end

% output
out_img = cat(3, RC, GC, BC);

end
```