Analisis Deteksi Tepi Pada Citra Digital Berbasis JPG Dengan Operator Canny Menggunakan Matrix Laboratory

Desi Herawati dan Aqwam Rosadi Kardian

Program Studi Sistem Informasi, STMIK Jakarta STI&K
Jl. BRI No.17 Radio Dalam, Kebayoran Baru, Jakarta Selatan
E-mail : desiherra1212@gmail.com, aqwam@jak-stik.ac.id

Abstrak

Citra (*image*) sebagai salah satu komponen multimedia memegang peranan penting sebagai bentuk informasi visual. Suatu citra memiliki karakteristik tersendiri, yang tidak dimiliki oleh data teks, yaitu citra kaya akan informasi. Dalam arti umum citra (*image*) merupakan sesuatu yang mereproduksi kembali kemiripan dari bentuk fisik suatu benda. Sedangkan dalam pengertiannya, citra merupakan fungsi menerus (*continue*) dari intensitas cahaya pada bidang dwimatra, yaitu di bidang sumbu x dan y pada koordinat cartesian. Dalam sebuah citra terdapat pertemuan antara bagian obyek dan latar belakang dinamakan sebagai tepi obyek. Oleh karenanya tepi sebuah obyek berguna untuk memisahkan obyek-obyek yang saling bersinggungan sehingga mereka tidak dianggap sebagai satu obyek yang besar. Deteksi tepi merupakan suatu proses yang menghasilkan tepi dari objek citra, yang bertujuan untuk menandai bagian yang menjadi detail citra dan untuk memperbaiki detail citra yang kabur.

Kata Kunci : JPG *Image*s, Canny Algorithm, *Edge Detection*, Matrix Laboratory

Pendahuluan

Citra (*image*) sebagai salah satu komponen multimedia memegang peranan penting sebagai bentuk informasi visual. Dalam arti umum citra kaya (*image*) merupakan sesuatu yang mereproduksi kembali kemiripan dari bentuk fisik suatu benda. Sedangkan dalam pengertiannya, citra merupakan fungsi menerus (continue) dari intensitas cahaya pada bidang dwimatra, yaitu di bidang sumbu x dan y pada koordinat cartesian.

Suatu obyek yang berada dalam bidang citra lain dan tidak bersinggungan dengan batas bidang citra tersebut, menyatakan bahwa obyek tersebut dikelilingi daerah yang bukan obyek atau biasa disebut sebagai latar belakang. Pertemuan antara bagian obyek dan latar belakang dinamakan sebagai tepi obyek. Oleh karenanya tepi sebuah obyek berguna untuk memisahkan obyek yang saling bersinggungan sehingga mereka tidak dianggap sebagai satu obyek yang besar dan tetap dapat dilacak atau dianalisa secara individu [10].

Secara harfiah, citra (*image*) merupakan gambar pada bidang dua dimensi. Ditinjau dari sudut pandang matematis, citra merupakan fungsi menerus (continue) dari intensitas cahaya pada bidang dua dimensi. Sumber cahaya menerangi objek, objek memantulkan kembali sebagian dari berkas cahaya tersebut. Pantulan cahaya ini ditangkap oleh alat-alat optik, misalnya mata pada manusia, kamera, penindai (*scanner)*, sehingga bayangan objek yang disebut citra tersebut terekam [2].

Citra digital merupakan sebuah larik (array) yang berisi nilai real maupun komplek yag direpresentasikan dengan deretan Suatu citra dapat didefinisikan sebagai fungsi f(x,y) berukuran M baris dan N kolom, dengan x dan y adalah koordinat spasial, dan amplitudo f di titik koordinat (x,y) dinamakan intensitas atau tingkat keabuan dari citra pada titik tersebut. Apabila nilai x,y dan nilai amplitudo f secara keseluruhan berhingga (finite) dan bernilai diskrit maka dapat dikatakan bahwa citra tersebut adalah citra digital.

Perumusan Masalah

Ditemukan beberapa kondisi berdasarkan latar belakang yang telah dijelaskan sebelumnya, muncul beberapa masalah mengenai penurunan kualitas suatu citra, diantaranya adalah :

a) Apakah suatu citra yang mengalami penurunan kualitas dapat diperbaiki?

b) Apakah dengan menggunakan operator canny dapat memperbaiki kualitas citra?

c) Apakah dengan menggunakan aplikasi Matlab dapat memperbaiki kualitas citra tersebut?

Tujuan Penelitian

Penelitian ini adalah yaitu :

a) Membuat aplikasi yang dapat membantu pengguna dalam menghasilkan tepi dari objek citra, yang menjadi detail citra.

b) Memperbaiki detail citra yang kabur, yang disebabkan karena error atau adanya efek dari proses akuisisi citra.

c) Membuktikan bahwa suatu citra dapat dilakukan pendeteksian tepi dengan menggunakan algoritma canny.

Metode Penelitian

Dalam penelitian ini dilakukan tahapan proses analisis pendeteksian tepi pada suatu citra, yaitu analisa terhadap proses pendeteksian tepi suatu citra untuk menghasilkan tepi-tepi dari objek citra, yang bertujuan untuk menandai bagian yang menjadi detail citra dan untuk memperbaiki detail citra yang kabur, yang disebabkan karena error atau adanya efek dari proses akuisisi citra. Pada penelitian ini mengambil dua jenis citra berformat JPG, yaitu citra skala keabuan (*grayscale*) dan warna (*true color*). Analisis pendeteksian tepi tersebut dilakukan dengan memanfaatkan fungsi aplikasi Matlab untuk citra skala keabuan (*grayscale*) dan warna (*true color*).

Tahapan dalam penelitian, diantaranya :

a) Studi Literatur dan Pemahaman, melalui mengumpulkan bahan-bahan referensi yang membahas tentang pengolahan citra, aplikasi Matlab, dan analisis pendeteksian tepi dengan operator canny.

b) Perancangan dan Implementasi. Perancangan dan implementasi deteksi tepi citra didasarkan pada operator canny dengan menggunakan alat bantu aplikasi Matlab.

c) Pengujian dan Analisis. Pada tahap ini dilakukan uji coba terhadap objek yang dibahas yaitu dengan membuat program untuk dapat melakukan proses pendeteksian tepi. Bagian akhir merupakan proses analisa terhadap perubahan yang terjadi pada citra yang diteliti.

Deteksi Tepi

Deteksi tepi adalah operasi pengolahan citra yang termasuk dalam bidang analisa citra. Deteksi tepi (*Edge Detection*) merupakan suatu proses yang menghasilkan tepi-tepi dari objek-objek citra, yang bertujuan untuk menandai bagian yang menjadi detail citra dan untuk memperbaiki detail citra yang kabur, yang disebabkan karena error atau adanya efek dari proses akuisisi citra.

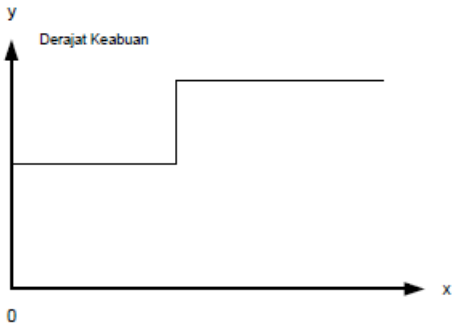
Dalam hal proses pendeteksian tepi terdapat banyak variasi teknik yang dapat digunakan, di mana tiap teknik tersebut dapat dikembangkan menjadi operator yang memiliki karakteristik tersendiri. Karakteristik sebuah operator deteksi tepi terkadang tidak dapat memberikan informasi mengenai karakteristik citra yang memadai, sehingga hasil akhir dari proses pengolahan citra tersebut tidak sesuai seperti apa yang diinginkan.

Deteksi tepi (*Edge Detection*) pada suatu citra merupakan suatu proses yang menghasilkan tepi-tepi dari objek-objek citra, yang bertujuan untuk menandai bagian yang menjadi detail citra dan untuk memperbaiki detail citra yang kabur, yang disebabkan karena error atau adanya efek dari proses akuisisi citra. Suatu titik (x,y) dikatakan sebagai tepi (edge) dari suatu citra bila titik tersebut mempunyai perbedaan yang tinggi dengan tetangganya.

Tepi itu sendiri adalah perubahan nilai intensitas derajat keabuan yang besar dalam jarak yang singkat. Terdapat tiga macam tepi di dalam citra digital, yaitu [3]:

a) Tepi Curam

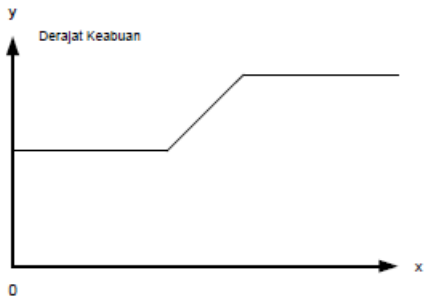
Tepi curam adalah tepi dengan intensitas yang tajam. Arah tepi berkisar 900. Gambar-1 menunjukan deteksi tepi tipe "tepi curam".



Gambar 1: Tepi Curam

b) Tepi Landai

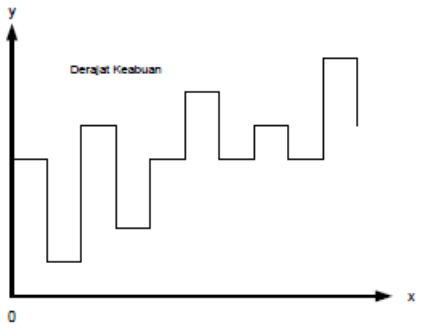
Tepi landai yaitu tepi dengan sudut arah yang kecil. Tepi landai dapat dianggap terdiri dari sejumlah tepi lokal yang lokasinya berdekatan. Pada gambar 2 menunjukan deteksi tepi tipe "tepi landai".



Gambar 2: Tepi Landai

c) Tepi yang mengandung derau

Umumnya tepi yang terdapat pada aplikasi visi komputer mengandung derau. Operasi peningkatan kualitas citra dapat dilakukan terlebih dahulu sebelum pendeteksian tepi. Gambar-3 menunjukan deteksi tepi tipe "tepi yang mengandung derau"



Gambar 3: Tepi curam dengan derau

Dalam pemrosesan deteksi tepi terdapat beberapa macam metode yang dapat digunakan, diantaranya adalah :

a) Metode Sobel

Metode ini menggunakan prinsip operasi spasial dengan ukuran mask 3x3. Tinjau pengaturan piksel di sekitar piksel (x,y) :

$$\begin{bmatrix} a_0 & a_1 & a_2 \\ a_7 & (x,y) & a_3 \\ a_6 & a_5 & a_4 \end{bmatrix} \quad (1)$$

Metode sobel adalah magnitudo dari gradien yang dihitung dengan :

$$M = \sqrt{(S_x^2+)S_y^2} \quad (2)$$

Metode Sobel dapat dinyatakan dalam bentuk mask :

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

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

b) Metode Prewitt

Persamaan gradien pada operator prewitt sama dengan operator sobel, tetapi menggunakan nilai c=1.

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

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

Metode Roberts dalam bentuk mask konvolusi

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

Untuk kekuatan tepi dihitung dengan rumus :

$$G[f(x,y)] = |R+| + |R-| \quad (3)$$

Metode turunan kedua disebut juga operator Laplace. Metode ini memperkirakan suatu Laplacian pada suatu fungsi kontinu yang didefinisikan sebagai diferensial parsial kedua dari fungsi intensitas piksel terhadap sumbu X dan Y. Metode Laplacian dalam deteksi tepi pada umumnya tidak dipergunakan secara langsung, namun dikombinasikan dengan suatu kernel Gaussian menjadi sebuah metode Laplacian of Gaussian (LoG).

Operator Canny

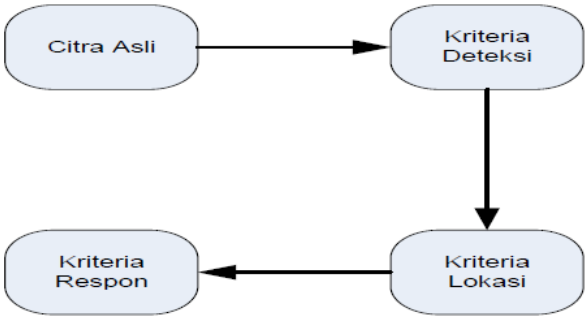
Salah satu operator deteksi tepi modern adalah deteksi tepi dengan menggunakan Operator Canny, deteksi tepi Canny ditemukan oleh Marr dan Hildreth yang meneliti permodelan persepsi visual manusia. Ada beberapa kriteria pendeteksi tepian paling optimum yang dapat dipenuhi oleh operator canny:

a) Mendeteksi dengan baik (kriteria deteksi)

b) Melokalisasi dengan baik (kriteria lokasi)

c) Respon yang jelas (kriteria respon)

Berdasarkan ketiga kriteria tersebut, langkah-langkah dalam deteksi tepi dengan operator canny terlihat pada gambar berikut ini.



Gambar 4: Deteksi Tepi dengan Operator Canny

1) Gaussian Smoothing

Langkah pertama dalam deteksi tepi dengan menggunakan operator Canny adalah melakukan smoothing terhadap citra asli, dengan tujuan untuk mengurangi respon sistem terhadap noise serta melakukan kontrol terhadap detail yang muncul pada tepi citra [7]. Smoothing dilakukan dengan mengkonvolusi citra menggunakan operator Gaussian g(x,y), dinyatakan sebagai [1]:

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

Dimana:

σ = standar deviasi dari distribusi Gaussian

x = sumbu horizontal

y = sumbu vertical

Persamaan (4) dapat digunakan untuk menghitung koefisien dari Gaussian template yang kemudian dikonvolusikan dengan citra.

Konvolusi Dengan Operator Turunan Pertama Konvolusi adalah perkalian total dari dua buah fungsi f dan h yang didefinisikan dengan :

$$f*h = \int_0^T f(t)h(T-t)dt \quad (5)$$

Untuk fungsi f dan h yang berdimensi 2, maka konvolusi dua dimensi didefinisikan dengan :

$$f*h = \int_0^{T_x}\int_0^{T_y} f(x,y)h(T_x - x, T_y - y)dxdy \quad (6)$$

Konvolusi dua dimensi inilah yang banyak digunakan dalam pengolahan citra digital, namun rumus diatas sangat sulit diimplementasikan menggunakan komputer, karena pada dasarnya komputer hanya bisa melakukan pada data yang diskrit sehingga tidak dapat digunakan untuk menghitung integral di atas. Konvolusi pada fungsi diskrit f(n,m) dan h(n,m) didefinisikan dengan :

$$y(k1,k2) = \sum_{n=1}^{Tn}\sum_{m=1}^{Tm} f(k1+n, k2, m)h(n,m) \quad (7)$$

Fungsi penapis g(x,y) disebut juga convolution filter, convolution mask, convolution kernel, atau template. Dalam bentuk diskrit convolution kernel dinyatakan dalam bentuk matriks (umumnya dalam bentuk matriks 3x3). Ukuran matriks ini biasanya lebih kecil dari ukuran citra.

Setiap elemen matriks disebut koefisien convolution.

Dalam operator turunan pertama ini akan menemukan gradient vertikal citra dan gradient horizontal citra, yang nantinya akan digabung untuk menghasilkan nilai gradient dari citra yang diteliti.

2) *Thresholding*

Meskipun Gaussian Smoothing sudah dilakukan pada tahap pertama deteksi tepi, namun masih ada kemungkinan citra hasil output dari proses pencarian gradient mengandung potongan-potongan garis tepi yang tidak sesuai dikarenakan adanya derau. Solusi yang biasa digunakan untuk mengatasi masalah ini adalah thresholding, dimana semua garis tepi yang bernilai dibawah nilai ambang (*threshold*) tertentu diabaikan. Tetapi dengan menggunakan metode ini, penentuan nilai ambang yang tepat sulit dilakukan. Akibatnya, masih akan ada garis tepi yang salah terdeteksi jika nilai ambang terlalu rendah, atau garis tepi yang hilang jika nilai ambang yang digunakan terlalu besar.

Untuk mengatasi masalah tersebut, Canny menggunakan dua nilai ambang yaitu ambang atas dan ambang bawah. Proses thresholding dimulai dengan titik yang nilainya lebih besar dari ambang atas. Titik ini dikenali sebagai garis tepi, kemudian digunakan sebagai seed untuk tahap berikutnya. Dari seed tersebut lalu dilakukan analisa keterhubungan/linking, dimana 8 piksel tetangga dari seed dicek apakah nilainya melebihi ambang bawah atau tidak. Jika melebihi ambang bawah, maka piksel tersebut dikenali sebagai garis tepi dan digunakan sebagai seed berikutnya. Proses ini dilakukan secara rekursif, sampai tidak ada lagi piksel tetangga dari seed yang memiliki nilai di atas nilai ambang bawah. Dengan demikian, *thresholding* akan mengurangi kemungkinan adanya noise pada garis tepi maupun hilangnya garis tepi.

3) Thinning

Thinning merupakan suatu proses untuk membuat garis pada sebuah gambar menjadi bentuk yang lebih sederhana. Dengan melakukan proses thinning, maka garis-garis yang terdapat pada citra akan memiliki ketebalan satu piksel. Tujuan dari proses thinning adalah menghilangkan piksel tertentu pada objek sehingga tebal objek tersebut menjadi hanya satu piksel [4].

Hasil dan Pembahasan

Deteksi tepi (*edge detection*) pada suatu citra merupakan suatu operasi pengolahan citra yang termasuk dalam bidang analisa citra. Deteksi tepi bertujuan untuk mencari perbedaan intensitas yang menyatakan batas suatu objek dalam suatu citra. Meskipun sebuah citra kaya akan informasi, namun terkadang citra tersebut tidak dapat mewakili informasi yang akan disampaikan. Hal tersebut dikarenakan kurangnya kualitas suatu citra.

Dalam proses deteksi tepi dengan operator canny, proses pertama yaitu penghalusan citra (smoothes) untuk menghilangkan noise. Selanjutnya proses pencarian tepian horizontal dan tepian vertikal untuk menentukan gradient dari citra yang akan menanandai daerah mana yang memiliki nilai spatial derivatives yang sangat tinggi. Operator tersebut kemudian memeriksa seluruh daerah dan memisahkan antara objek dan latar belakangnya dengan menggunakan *threshold*. Proses terakhir dalam operator ini yaitu adalah proses thinning, proses ini akan mempertipis garis tepi yang telah dihasilkan pada proses sebelumnya, sehingga bentuk objek dapat terlihat lebih jelas.

Objek citra yang akan diteliti ada dua macam, yaitu :

1) Citra Warna (*True Color*)

Citra warna (*true color*) bernama fruits.jpg berukuran 1600x1200 pixels dan kapasitasnya 493 KB. Tampilan citra tersebut adalah sebagai berikut

Gambar 5: Citra Warna

2) Citra Skala Keabuan (*Grayscale*)

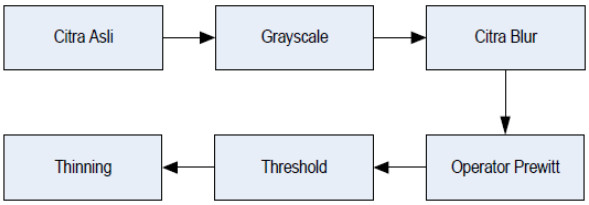
Citra *grayscale* bernama the beatles.jpg berukuran 520x642 pixels dan berkapasitas 45,9 KB. Tampilan citra tersebut adalah sebagai berikut :

Gambar 6: Citra *Grayscale*

Pembahasan Masalah

Citra digital merupakan sebuah larik (array) yang berisi nilai-nilai real maupun komplek yag direpresentasikan dengan deretan bit tertentu. Sebuah citra diubah ke dalam bentuk digital agar dapat disimpan dalam memori komputer atau media lainnya. Proses perubahan citra ke dalam bentuk digital dapat dilakukan dengan beberapa perangkat seperti, scanner, kamera digital, dan handycam. Bermacam macam proses pengolahan citra dapat dilakukan ketika sebuah citra telah diubah dalam bentuk digital.

Berikut ini merupakan tampilan blok diagram pengolahan citra digital yang akan dibahas :



Gambar 7: Diagram Tahapan Deteksi Tepi

Blok diagram tersebut menunjukan alur kerja pendeteksi tepian dengan menggunakan operator canny. Diagram ini menjelaskan proses citra digital asli dengan format .jpg ditransformasikan menjadi citra kabur (blurring) dengan metode Gaussian Filter, proses ini bertujuan untuk menyaring dan membuang noise pada citra asli. Proses selanjutnya adalah penggunaan operator prewitt dalam menemukan tepi dengan menggunakan gradient dari gambar tersebut.

Setelah proses pencarian garis tepi dengan menggunakan operator prewitt selesai, tahap selanjutnya adalah *thresholding* untuk mengurangi noise serta menghasilkan garis tepi pada citra. Setelah melalui proses *threshold*, diperlukan proses thinning untuk menipiskan garis tepi pada tahap sebelumnya, serta mempertegas bentuk objek yang diteliti.

Teknik Konversi Citra RGB menjadi Citra *Grayscale*

Dalam pemrosesan deteksi tepi pada citra RGB fruits.jpg diperlukan tahap awal, yaitu pengkonversian citra RGB menjadi citra *grayscale*. Berikut tampilan awal citra RGB fruits.jpg sebelum dikonversikan menjadi citra *grayscale* :

Gambar 8: Citra Berwarna (RGB)

Pada gambar 8, merupakan citra RGB asli sebelum yang akan diproses untuk dirubah menjadi citra *grayscale*. Dengan menggunakan

aplikasi Matlab citra RGB fruits.jpg di atas diproses dengan menggunakan script berikut

```
%Baca citra 'fruits.jpg'
gambar = imread('fruits.jpg');
figure; imshow(gambar);
title('Original Picture');
%Konversi citra RGB menjadi Grayscale
gray = rgb2gray(gambar);
figure; imshow(gray); title('Grayscale');
```

Berikut tampilan citra fruits.jpg yang telah dikonversikan menjadi citra *grayscale* :

Gambar 9: Citra RGB setelah proses konversi menjadi *Grayscale*

Dengan menggunakan script diatas, citra RGB fruits.jpg dirubah menjadi citra *grayscale* fruits.jpg. Pengkonverisan citra RGB menjadi citra *grayscale* sangat dibutuhkan sebagai tahapan awal pada kebanyakan proses pengolahan citra termasuk deteksi tepi, hal ini bertujuan untuk menyederhanakan model citra. Citra yang ditampilkan dari proses ini adalah citra berwarna abu-abu (*grayscale*). Citra *grayscale* berbeda dengan citra hitam putih. Citra *grayscale* merupakan variasi dari warna hitam pada bagian yang berintensitas lemah dan warna putih pada bagian yang berintensitas tinggi.

Teknik Penghalusan Citra RGB dengan Filter Gaussian

Dalam pembahasan ini citra asli akan dihaluskan dengan menggunakan filter gaussian menggunakan aplikasi Matlab setelah melalui proses konversi citra RGB menjadi citra *grayscale*.

Filter gaussian merupakan salah satu metode untuk menimbulkan efek seperti low pass filter yang berfungsi untuk menghilangkan noise serta menghaluskan citra RGB fruits yang telah dikonversikan menjadi citra *grayscale*. Berikut tampilan citra fruits.jpg yang akan diterapkan metode smoothing dengan menggunakan filter gaussian :

Gambar 10: Citra sebelum proses penghalusan

Dengan menggunakan script dibawah ini, gambar 10 diproses untuk menghasilkan citra yang lebih halus tampilannya. Berikut script yang digunakan untuk penghalusan citra fruits.jpg dengan menggunakan bantuan dari aplikasi Matlab.

```
%Baca citra 'fruits.jpg'
gambar = imread('fruits.jpg');
figure; imshow(gambar);
title('Original Picture');
%Konversi citra RGB menjadi Grayscale
gray = rgb2gray(gambar);
figure; imshow(gray);
title('Grayscale');
%Gaussian filter
gaussian = fspecial('gaussian',[12 12], 5)
hasil=imfilter(gray,
gaussian,'symmetric','conv');
figure; imshow(hasil), title('Blurred');
```

Berikut tampilan hasil proses penghalusan citra fruits.jpg dengan menggunakan filter gaussian :

Gambar 11: Citra setelah proses penghalusan

Filter gaussian didapat dari operasi konvolusi. Operasi perkalian yang dilakukan ialah perkalian antara matriks kernel dengan matriks gambar asli. Pada program diatas, proses filterisasi secara gaussian dilakukan dengan menggunakan fungsi Matlab yaitu :

$$h = fspecial(`gaussian', hsize, sigma)$$

Dimana :

h = nama fungsi.

fspecial = jenis filter yang terdapat pada aplikasi matlab.

gaussian = filter gaussian yang terdapat pada aplikasi matlab.

hsize = vektor yang menyatakan jumlah baris dan kolom matriks.

sigma = nilai sigma yang digunakan.

Cara kerja dari fungsi filter gaussian yang terdapat dalam program adalah dengan menggunakan matriks dengan ukuran 12x12 dan sigma 5, citra akan diburamkan dengan intensitas yang ringan. Pengambilan nilai matriks dan sigma pada script tersebut dilakukan secara acak, dan dirasakan bahwa matriks 12x12 dan sigma = 5 sudah cukup untuk memenuhi tahapan penghalusan gambar dengan menggunakan filter gaussian.

Teknik Pencarian Gradient Horizontal pada Citra RGB

Setelah proses penghalusan (smoothing) pada citra RGB fruits.jpg, langkah selanjutnya yaitu pencarian gradient horizontal untuk menentukan garis tepi horizontal dengan menggunakan operator prewitt. Berikut ini merupakan tampilan citra sebelum melalui tahap pencarian gradient horizontal :

Gambar 12: Citra RGB yang telah di smoothing

Script berikut digunakan dalam proses pencarian gradient horizontal pada citra RGB fruits.jpg :

```
%Baca citra 'fruits.jpg'
gambar = imread('fruits.jpg');
figure;
imshow(gambar); title('Original Picture');
%Konversi citra RGB menjadi Grayscale
gray = rgb2gray(gambar);
figure; imshow(gray);
title('Grayscale');
%Gaussian filter
gaussian=fspecial('gaussian',[12 12], 5)
hasil=imfilter(gray,
gaussian,'symmetric','conv');
figure; imshow(hasil), title('Blurred');
%Deteksi Tepi Horizontal
px = [-1 0 1;-1 0 1;-1 0 1];
hasil_px = filter2 (px, hasil); figure;
imshow(hasil_px/255),
title('Tepi Horizontal');
```

Berikut tampilan citra RGB fruits.jpg setelah melalui proses pencarian gradient horizontal

Gambar 13: Gradient horizontal pada citra RGB

Pada gambar 13, sudah terlihat bahwa objek-objek yang terdapat pada citra RGB fruits.jpg tidak terlalu nampak, hanya beberapa garis tipis yang dapat terlihat. Hal tersebut dikarenakan garis tepi yang dihasilkan hanya berupa garis tepi secara horizontal. Proses pencarian gradient horizontal pada citra RGB fruits.jpg berdasarkan pada metode prewitt.

Dimana metode ini menggunakan peninjauan pengaturan piksel di sekitar piksel (x,y) dengan menggunakan mask 3x3 yaitu :

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

Berdasarkan mask 3x3 tersebut, gradient horizontal dihitung dengan menggunakan persamaan :

$$M = \sqrt{P_x^2 + P_y^2} \qquad (8)$$

Dalam hal ini turunan parsial dihitung dengan cara

$$P_x = (a_2 + ca_3 + a_4) - (a_0 + ca_7 + a_6)$$

Dengan menggunakan nilai konstanta c=1, maka Px dapat dinyatakan dalam bentuk mask yaitu

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

Berdasarkan mask Px yang telah didapat, maka proses pencarian gradient horizontal dapat dilakukan dengan menggunakan matriks

[-1 0 1] [-1 0 1] [-1 0 1]

Pada script proses pencarian gradient horizontal mask Px berguna untuk mencari gradient, filter2 digunakan sebagai filterisasi pada citra dua dimensi, dan nilai 255 dimaksudkan untuk menampilkan garis tepi berwarna putih.

Teknik Pencarian Gradient Vertikal pada Citra RGB

Setelah melakukan proses pencarian gradient horizontal, tahap selanjutnya yaitu proses pencarian gradient vertikal pada citra RGB fruits.jpg. Proses ini bertujuan untuk mencari garis tepi vertikal pada citra dengan menggunakan operator prewitt. Berikut ini tampilan citra RGB fruits.jpg sebelum dilakukan proses pencarian gradient vertikal.

Gambar 14: Citra RGB sebelum pencarian gradient vertikal

Script berikut digunakan dalam proses pencarian gradient vertikal pada citra RGB fruits.jpg.

```
%Baca citra 'fruits.jpg'
gambar = imread('fruits.jpg');
figure; imshow(gambar);
title('Original Picture');
%Konversi citra RGB menjadi Grayscale
gray = rgb2gray(gambar);
figure; imshow(gray); title('Grayscale');
%Gaussian filter
gaussian = fspecial('gaussian',[12 12], 5)
hasil=imfilter(gray,gaussian,
'symmetric','conv');
figure; imshow(hasil), title('Blurred');
%Deteksi Tepi Vertikal
py = [-1 -1 -1;0 0 0;1 1 1];
hasil_py = filter2 (py, hasil);
figure; imshow(hasil_py/255),
title('Tepi Vertikal');
```

Berikut tampilan citra RGB fruits.jpg setelah melalui proses pencarian gradient vertikal.

Gambar 15: Gradient vertikal pada citra RGB

Pada gambar 15, terlihat bahwa objek-objek yang terdapat pada citra RGB fruits.jpg tidak terlalu nampak, hanya beberapa garis tipis yang dapat terlihat. Hal tersebut dikarenakan garis tepi yang dihasilkan hanya berupa garis tepi secara vertikal. Proses pencarian gradient vertikal pada citra RGB fruits.jpg berdasarkan pada metode prewitt.

Dimana metode ini menggunakan peninjauan pengaturan piksel di sekitar piksel (x,y) dengan menggunakan mask 3x3 yaitu

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

Berdasarkan mask 3x3 tersebut, gradient vertikal dihitung dengan menggunakan persamaan

$$M = \sqrt{P_x^2 + P_y^2} \qquad (9)$$

Dalam hal ini turunan parsial dihitung dengan cara

$$P_y = (a_o + ca_1 + a_2) - (a_6 + ca_5 + a_4)$$

Dengan menggunakan nilai konstanta c=1, maka Py dapat dinyatakan dalam bentuk mask yaitu

$$P_Y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Berdasarkan mask Px yang telah didapat, maka proses pencarian gradient vertikal dapat dilakukan dengan menggunakan matriks

[-1 -1 -1] [0 0 0] [1 1 1]

Pada script proses pencarian gradient vertikal mask Py berguna untuk mencari gradient, filter2 digunakan sebagai filterisasi pada citra dua dimensi, dan nilai 255 dimaksudkan untuk menampilkan garis tepi vertikal berwarna putih.

Teknik Penggabungan Kedua Gradient pada Citra RGB

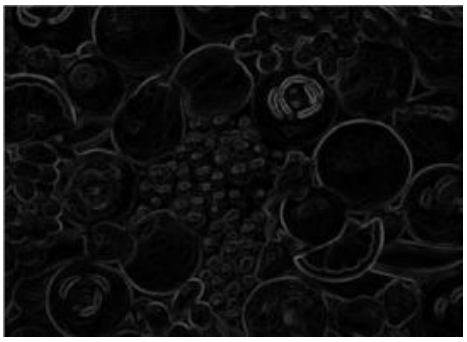
Untuk menemukan gradient pada citra RGB fruits.jpg diperlukannya penggabungan gradient vertikal dan gradient horizontal yang telah didapat dengan menggunakan operator prewitt. Berikut merupakan tampilan gradient horizontal yang akan digabungkan dengan gradient vertikal.

Gambar 16: Gradient horizontal yang akan digabungkan dengan gradient vertikal pada citra RGB

Dengan menggunakan kedua gradient yaitu gradient vertikal dan gradient horizontal yang telah didapat dari citra RGB fruits.jpg, maka proses selanjutnya adalah proses penggabungan kedua gradient dengan menggunakan script berikut ini.

```
%Baca citra 'fruits.jpg'
gambar = imread('fruits.jpg');
figure; imshow(gambar);
title('Original Picture');
%Konversi citra RGB menjadi Grayscale
gray = rgb2gray(gambar);
figure; imshow(gray); title('Grayscale');
%Gaussian filter
gaussian = fspecial('gaussian',[12 12], 5)
hasil = imfilter(gray,
gaussian,'symmetric','conv');
figure; imshow(hasil), title('Blurred');
%Deteksi Tepi Horizontal
px = [-1 0 1;-1 0 1;-1 0 1];
hasil_px = filter2 (px, hasil);
figure; imshow(hasil_px/255),
title('Tepi Horizontal');
%Deteksi Tepi Vertikal
py = [-1 -1 -1;0 0 0;1 1 1];
hasil_py = filter2 (py, hasil);
figure; imshow(hasil_py/255),
title('Tepi Vertikal');
%Penggabungan Kedua Gradient
gradient = sqrt(hasil_px.^2 + hasil_py.^2);
figure; imshow (gradient/255),
title('Gradient')
```

Berikut tampilan citra RGB fruits.jpg setelah kedua gradient vertikal dan gradient horizontal digabungkan



Gambar 17: Penggabungan Gradient pada Citra RGB

Dengan menggunakan script tersebut, proses penggabungan gradient dilakukan. Proses penggabungan gradient pada citra RGB fruits.jpg itu sendiri didasari pada persamaan

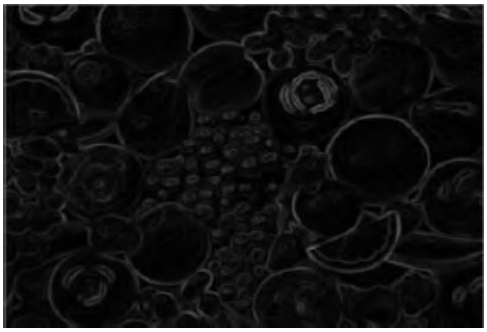
$$G = \sqrt{G_X^2 + G_Y^2} \qquad (10)$$

Dimana gradient pada citra RGB fruits.jpg didapatkan dari hasil akar penjumlahan gradient vertikal kuadrat dan gradient horizontal kuadrat. Dengan penggabungan kedua gradient vertikal dan gradient horizontal yang telah didapat dengan menggunakan operator prewitt, maka hasil yang didapatkan pada citra RGB fruits.jpg adalah mulai tampaknya garis-garis tepi pada objek, sehingga bentuk pada objek dapat terlihat.

Teknik Penyaringan Noise Citra RGB dengan Thresholding

Melalui proses penggabungan gradient vertikal dan gradient horizontal pada citra RGB

fruits.jpg, telah dihasilkan garis-garis tepi pada objek. Namun garis tepi yang dihasilkan pada proses penggabungan kedua gradient tersebut masih mengandung garis-garis halus dan juga noise pada citra RGB fruits.jpg, yang mengakibatkan objek-objek yang ada pada citra RGB fruits.jpg masih kurang terlihat dengan jelas. Berdasarkan permasalahan tersebut, maka diperlukan adanya proses *thresholding*. Berikut ini merupakan tampilan citra RGB fruits.jpg yang akan diproses dengan menggunakan teknik *thresholding*



Gambar 18: Citra RGB sebelum proses treshholding

Source code berikut digunakan dalam pengolahan citra dengan menggunakan teknik *thresholding.*

```
%Baca citra 'fruits.jpg'
gambar = imread('fruits.jpg');
figure; imshow(gambar);
title('Original Picture');
%Konversi citra RGB menjadi Grayscale
gray = rgb2gray(gambar);
figure; imshow(gray); title('Grayscale');
%Gaussian filter
gaussian = fspecial('gaussian',[12 12], 5)
hasil = imfilter(gray,
gaussian,'symmetric','conv');
figure; imshow(hasil), title('Blurred');
%Deteksi Tepi Horizontal
px = [-1 0 1;-1 0 1;-1 0 1];
hasil_px = filter2 (px, hasil);
figure; imshow(hasil_px/255),
title('Tepi Horizontal');
%Deteksi Tepi Vertikal
py = [-1 -1 -1;0 0 0;1 1 1];
hasil_py = filter2 (py, hasil);
figure; imshow(hasil_py/255),
title('Tepi Vertikal');
%Penggabungan Kedua Gradient
gradient = sqrt(hasil_px.^2+hasil_py.^2);
figure; imshow (gradient/255),
title('Gradient')
%Threshold
level = graythresh(gradient/255);
bw = im2bw(gradient/255,level);
bw = bwareaopen (bw, 50);
figure; imshow(bw),title('Threshold');
```

Dengan menggunakan script tersebut, maka tampilan citra yang dihasilkan adalah sebagai berikut.

Gambar 19: Citra RGB setelah proses *thresholding*

Berdasarkan hasil citra RGB fruits.jpg yang telah didapatkan setelah melalui proses *thresholding,* maka garis tepi yang dihasilkan sudah terlihat dengan jelas. Proses thrresholding itu sendiri bertujuan untuk menghilangkan noise, memisahkan antara objek dengan latar belakangnya, dan untuk mempertegas garis tepi yang telah didapat pada citra RGB fruits.jpg yang dihasilkan dari proses sebelumnya. Proses *thresholding* pada prgogram ini menggunakan fungsi pada aplikasi matlab yaitu

```
level = graythresh(gradient/255);
bw = im2bw(gradient/255,level);
bw = bwareaopen (bw, 50);
```

Dengan menggunakan fungsi "graythresh", dan menggunakan fungsi "im2bw" yang berguna untuk merubah citra RGB fruits.jpg menjadi citra biner. Setelah citra RGB fruits.jpg diubah menjadi citra biner, langkah selanjutnya adalah pemisahan objek dengan latar belakangnya dengan menggunakan fungsi "bwareaopen" dan dengan menggunakan nilai *thresholding*=50. Pengambilan nilai *threshold* dalam program ini dilakukan secara acak demi mempersingkat waktu.

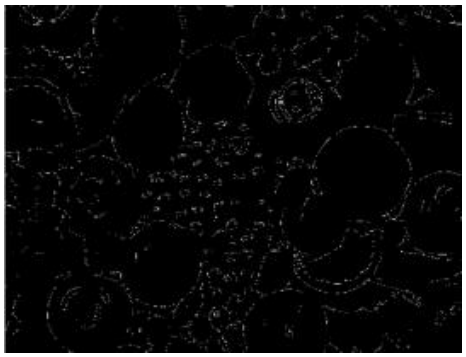
Teknik Penipisan Garis Tepi pada Citra RGB

Meskipun garis tepi pada citra RGB fruits.jpg telah terbentuk jelas dengan menggunakan teknik *thresholding*, namun garis tepi yang dihasilkan terlalu tebal sehingga beberapa objek yang berukuran kecil tidak telihat dengan jelas. Proses thinning dibutuhkan untuk menipiskan garis tepi yang terlalu tebal, sehingga semua objek dapat terlihat bentuknya. Dengan menggunakan script berikut, garis tepi yang tampak pada gambar 20, akan melalui proses penipisan.

Berikut ini script yang digunakan dalam proses thinning :

```
%Baca citra 'fruits.jpg'
gambar = imread('fruits.jpg');
figure; imshow(gambar);
title('Original Picture');
%Konversi citra RGB menjadi Grayscale
gray = rgb2gray(gambar);
figure; imshow(gray);
title('Grayscale');
%Gaussian filter
gaussian = fspecial('gaussian',[12 12], 5)
hasil = imfilter(gray,
gaussian,'symmetric','conv');
figure; imshow(hasil),
title('Blurred');
%Deteksi Tepi HOrizontal
px = [-1 0 1;-1 0 1;-1 0 1];
hasil_px = filter2 (px, hasil);
figure; imshow(hasil_px/255),
title('Tepi Horizontal');
%Deteksi Tepi Vertikal
py = [-1 -1 -1;0 0 0;1 1 1];
hasil_py = filter2 (py, hasil);
figure; imshow(hasil_py/255),
title('Tepi Vertikal');
%Penggabungan Kedua Gradient
gradient=sqrt(hasil_px.^2+hasil_py.^2);
figure; imshow (gradient/255),
title('Gradient')
%Threshold
level = graythresh(gradient/255);
bw = im2bw(gradient/255,level);
bw = bwareaopen (bw, 50);
figure; imshow(bw),title('Threshold');
%Thinning
bw2 = bwmorph (bw,'skel',Inf);
figure; imshow(bw2),title('Thinning');
```

Berikut ini merupakan tampilan citra RGB fruits.jpg setelah melalui proses thinning.



Gambar 20: Hasil akhir deteksi tepi pada citra RGB

Dengan menggunakan proses thinning, objek-objek pada citra RGB fruits.jpg menjadi terlihat lebih jelas hingga ke objeknya yang terkecil sekalipun. Proses thinning dalam program diatas digunakan untuk merubah ukuran garis tepi yang pada citra RGB fruits.jpg yang didapat dari proses *thresholding* menjadi satu piksel. Pada proses thinning pada citra RGB fruits.jpg, digunakan fungsi Matlab yaitu

```
bw2 = bwmorph (bw,'skel',Inf);
```

Dimana fungsi yang digunakan untuk pemrosesan thinning adalah "bwmorph", dengan menggunakan nilai "Inf" (tak terhingga) operasi "skel" menghilangkan piksel-piksel yang terdapat pada batasan objek tanpa membuat terputusnya garis tepi pada objek.

Teknik Penghalusan Citra *Grayscale* dengan Filter Gaussian

Pada citra *grayscale* beatles.jpg, proses pendeteksian tepi tidak jauh berbeda dengan proses pendeteksian tepi pada citra RGB fruits.jpg. Hanya saja pada langkah awal pendeteksian tepi pada citra *grayscale* beatles.jpg tidak melalui proses konversi citra menjadi citra *grayscale*. Hal tersebut dikarenakan citra itu sendiri sudah berupa citra *grayscale*, maka dari itu proses pendeteksian tepi pada citra *grayscale* beatles.jpg langsung dimulai dengan proses penghalusan gambar dengan menggunakan filter gaussian.

Berikut ini merupakan tampilan awal citra sebelum melalui proses penghalusan gambar dengan menggunakan filter gaussian untuk citra *grayscale*.

Gambar 21: Hasil akhir deteksi tepi pada citra RGB

Untuk proses penghalusan citra *grayscale* langkahnya sama seperti citra berwarna, source code sebagai berikut.

```
%Baca citra 'beatles.jpg'
gambar = imread('beatles.jpg');
gray = rgb2gray(gambar);
figure; imshow(gray); title('Grayscale');
%Gaussian filter
gaussian = fspecial('gaussian',[12 12], 5)
hasil = imfilter(gray,
gaussian,'symmetric','conv');
figure; imshow(hasil), title('Blurred');
```

Hasil dari script program diatas terlihat pada gambar 22, berikut

Gambar 22: Citra *grayscale* setelah proses penghalusan

Proses penghalusan citra *grayscale* beatles.jpg tidak jauh berbeda dengan proses penghalusan citra RGB fruits.jpg, dimana proses filterisasi secara gaussian dilakukan dengan menggunakan fungsi Matlab yaitu

$$h = fspecial(\text{`}gaussian\text{'}, hsize, sigma)$$

Dimana :

h = nama fungsi.

fspecial = jenis filter yang terdapat pada aplikasi matlab.

gaussian = filter gaussian yang terdapat pada aplikasi matlab.

hsize = vektor yang menyatakan jumlah baris dan kolom matriks.

sigma = nilai sigma yang digunakan

Ukuran matriks dan juga nilai sigma yang digunakan dalam proses penghalusan gambar pada citra *grayscale* beatles.jpg sama dengan yang digunakan pada citra RGB fruits.jpg yaitu ukuran matriks adalah 12x12 dan nilai sigma yang digunakan adalah 5. Penggunaan matriks dan sigma dalam program ini adalah untuk memburamkan citra dengan intensitas yang ringan, sehingga citra *grayscale* beatles.jpg dapat diproses untuk tahapan selanjutnya yaitu proses pencarian gradient citra.

Teknik Pencarian Gradient Horizontal pada Citra *Grayscale*

Sama halnya dengan proses penghalusan citra *grayscale* beatles.jpg, proses pencarian gradient horizontal pada citra *grayscale* beatles.jpg tidak jauh berbeda dengan proses pencarian gradient horizontal pada citra RGB fruits.jpg.

Berikut adalah script program untuk pencarian gradient horizontal pada citra *grayscale*.

```
%Baca citra 'beatles.jpg'
gambar = imread('beatles.jpg');
gray = rgb2gray(gambar);
figure; imshow(gray); title('Grayscale');
%Gaussian filter
gaussian=fspecial('gaussian',[12 12], 5)
hasil=imfilter(gray,
gaussian,'symmetric','conv');
figure; imshow(hasil), title('Blurred');
%Deteksi Tepi Horizontal
px = [-1 0 1;-1 0 1;-1 0 1];
hasil_px = filter2 (px, hasil);
figure; imshow(hasil_px/255),
title('Tepi Horizontal');
```

Berikut ini merupakan tampilan citra *grayscale* setelah proses pencarian gradient horizontal.

Gambar 23: Gradient Horizontal pada Citra *Grayscale*

Untuk mendapatkan gradient horizontal pada citra *grayscale* beatles.jpg, dibutuhkan operator turunan pertama. Dalam program ini penulis menggunakan operator prewitt yang juga digunakan pada proses pencarian gradient pada citra RGB fruits.jpg.

Dimana metode ini menggunakan peninjauan pengaturan piksel di sekitar piksel (x,y) dengan menggunakan mask 3x3 yaitu :

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

Berdasarkan mask 3x3 tersebut, gradient horizontal dihitung dengan menggunakan persamaan

$$M = \sqrt{P_x^2 + P_y^2} \qquad (11)$$

Dalam hal ini turunan parsial dihitung dengan cara

$$P_x = (a_2 + ca_3 + a_4) - (a_0 + ca_7 + a_6)$$

Dengan menggunakan nilai konstanta c=1, maka Px dapat dinyatakan dalam bentuk mask yaitu

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

Berdasarkan mask Px yang telah didapat, maka proses pencarian gradient horizontal dapat dilakukan dengan menggunakan matriks

[-1 0 1] [-1 0 1] [-1 0 1]

Pada script proses pencarian gradient horizontal mask Px berguna untuk mencari gradient, filter2 digunakan sebagai filterisasi pada citra dua dimensi, dan nilai 255 dimaksudkan untuk menampilkan garis tepi berwarna putih.

Teknik Pencarian Gradient Vertikal pada Citra *Grayscale*

Dalam melakukan proses pencarian gradient pada citra *grayscale* beatles.jpg, selain gradient horizontal diperlukan juga gradient vertikal untuk mendapatkan gradient yang tepat pada citra *grayscale* beatles.jpg.

Berikut ini merupakan tampilan citra *grayscale* beatles.jpg sebelum melalui proses pencarian gradient vertikal.

Gambar 24: Gradient Vertikal pada Citra *Grayscale*

Berikut ini adalah script program untuk pencarian gradient vertikal pada citra *grayscale*.

```
%Baca citra 'beatles.jpg'
gambar = imread('beatles.jpg');
gray = rgb2gray(gambar);
figure; imshow(gray); title('Grayscale');
%Gaussian filter
gaussian=fspecial('gaussian',[12 12], 5)
hasil = imfilter(gray,
gaussian,'symmetric','conv');
figure; imshow(hasil), title('Blurred');
%Deteksi Tepi Vertikal
py = [-1 -1 -1;0 0 0;1 1 1];
hasil_py = filter2 (py, hasil);
figure; imshow(hasil_py/255),
title('Tepi Vertikal');
```

Untuk mendapatkan gradient vertikal pada citra *grayscale* beatles.jpg, dibutuhkan operator turunan pertama. Dalam program ini penulis menggunakan operator prewitt yang juga digunakan pada proses pencarian gradient pada citra RGB fruits.jpg.

Dimana metode ini menggunakan peninjauan pengaturan piksel di sekitar piksel (x,y) dengan menggunakan mask 3x3 yaitu :

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

Berdasarkan mask 3x3 tersebut, gradient vertikal dihitung dengan menggunakan persamaan

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

Dalam hal ini turunan parsial dihitung dengan cara

$$P_y = (a_o + ca_1 + a_{2)} - (a_6 + ca_5 + a_4)$$

Dengan menggunakan nilai konstanta c=1, maka Py dapat dinyatakan dalam bentuk mask yaitu

$$P_Y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Berdasarkan mask Px yang telah didapat, maka proses pencarian gradient vertikal dapat dilakukan dengan menggunakan matriks :

[-1 -1 -1] [0 0 0] [1 1 1]

Pada script proses pencarian gradient vertikal mask Py berguna untuk mencari gradient, filter2 digunakan sebagai filterisasi pada citra dua dimensi, dan nilai 255 dimaksudkan untuk menampilkan garis tepi vertikal berwarna putih.

Teknik Penggabungan Kedua Gradient pada Citra *Grayscale*

Setelah pencarian gradient vertikal dan gradient horizontal pada citra *grayscale* beatles.jpg selesai, langkah selanjutnya yaitu proses penggabungan kedua gradient untuk mendapatkan gradient pada citra *grayscale* beatles.jpg.

Dengan menggunakan kedua gradient yaitu gradient vertikal dan gradient horizontal yang telah didapat dari citra *grayscale* beatles.jpg, maka proses selanjutnya adalah proses penggabungan kedua gradient dengan menggunakan script berikut ini :

```
%Baca citra 'beatles.jpg'
gambar = imread('beatles.jpg');
gray = rgb2gray(gambar);
figure; imshow(gray); title('Grayscale');
%Gaussian filter
gaussian = fspecial('gaussian',[12 12], 5)
hasil = imfilter(gray,
gaussian,'symmetric','conv');
figure; imshow(hasil), title('Blurred');
%Deteksi Tepi Horizontal
px = [-1 0 1;-1 0 1;-1 0 1];
hasil_px = filter2 (px, hasil);
figure; imshow(hasil_px/255),
title('Tepi Horizontal');
%Deteksi Tepi Vertikal
py = [-1 -1 -1;0 0 0;1 1 1];
hasil_py = filter2 (py, hasil);
figure;
imshow(hasil_py/255),title('Tepi Vertikal');
%Penggabungan Kedua Gradient
gradient=sqrt(hasil_px.^2 + hasil_py.^2);
figure;
imshow (gradient/255),title('Gradient');
```

Berikut tampilan citra *grayscale* beatles.jpg setelah kedua gradient vertikal dan gradient horizontal digabungkan.

Gambar 25: Penggabungan kedua gradient pada citra *grayscale*

Dengan menggunakan script tersebut, proses penggabungan gradient dilakukan. Proses penggabungan gradient pada citra *grayscale* beatles.jpg itu sendiri didasari pada persamaan

$$G = \sqrt{G_X^2 + G_Y^2} \qquad (12)$$

Dimana gradient pada citra *grayscale* beatles.jpg didapatkan dari hasil akar penjumlahan gradient vertikal kuadrat dan gradient horizontal kuadrat. Dengan penggabungan kedua gradient vertikal dan gradient horizontal yang telah didapat dengan menggunakan operator prewitt, maka hasil yang didapatkan pada citra *grayscale* beatles.jpg adalah mulai tampaknya garis-garis tepi pada objek, sehingga bentuk pada objek dapat terlihat.

Teknik Penyaringan Noise Citra *Grayscale* dengan *Thresholding*

Proses selanjutnya setelah gradient pada citra *grayscale* beatles.jpg telah ditemukan yaitu, proses penyaringan noise pada citra *grayscale* beatles.jpg dengan menggunakan *thresholding*.

Untuk melakukan proses pengurangan noise pada citra tersebut, maka berikut ini script proses threshold dengan menggunakan aplikasi matlab

```
%Baca citra 'beatles.jpg'
gambar = imread('beatles.jpg');
gray = rgb2gray(gambar);
figure; imshow(gray); title('Grayscale');
%Gaussian filter
gaussian = fspecial('gaussian',[12 12], 5)
hasil=imfilter(gray,
gaussian,'symmetric','conv');
figure; imshow(hasil), title('Blurred');
%Deteksi Tepi Horizontal
px = [-1 0 1;-1 0 1;-1 0 1];
hasil_px = filter2 (px, hasil);
figure; imshow(hasil_px/255),
title('Tepi Horizontal');
%Deteksi Tepi Vertikal
py = [-1 -1 -1;0 0 0;1 1 1];
hasil_py = filter2 (py, hasil);
figure;
imshow(hasil_py/255),
title('Tepi Vertikal');
%Penggabungan Kedua Gradient
gradient=sqrt(hasil_px.^2 + hasil_py.^2);
figure;
imshow (gradient/255),title('Gradient');
%Threshold
level = graythresh(gradient/255);
bw = im2bw(gradient/255,level);
bw = bwareaopen (bw, 50);
figure; imshow(bw),title('Threshold');
```

Berikut ini merupakan tampilan dari pengga- bungan kedua gradient pada citra *grayscale* beatles.jpg

Gambar 26: Citra *grayscale* setelah proses *thresholding*

Pada proses ini dapat terlihat bahwa hasil pencarian gradient pada citra RGB fruits.jpg maupun pada citra *grayscale* beatles.jpg masih menghasilkan derau (noise) yang cukup banyak. Hal tersebut mengakibatkan objek pada citra tersebut menjadi kurang terlihat jelas. Maka dari itu proses *thresholding* sangat dibutuhkan untuk menghasilkan garis tepi yang dapat mempertegas objek pada citra RGB fruits.jpg dan juga pada citra *grayscale* beatles.jpg.

Proses *thresholding* pada program menggunakan fungsi pada aplikasi matlab

```
level=graythresh(gradient/255);
bw = im2bw(gradient/255,level);
bw = bwareaopen (bw, 50);
```

Dengan menggunakan fungsi "graythresh", dan menggunakan fungsi "im2bw" yang berguna untuk merubah citra RGB fruits.jpg menjadi citra biner. Setelah citra RGB fruits.jpg diubah menjadi citra biner, langkah selanjutnya adalah pemisahan objek dengan latar belakangnya dengan menggunakan fungsi "bwareaopen" dan dengan menggunakan nilai *thresholding*=50. Pengambilan nilai threshold dalam program dilakukan secara acak untuk mempersingkat waktu.

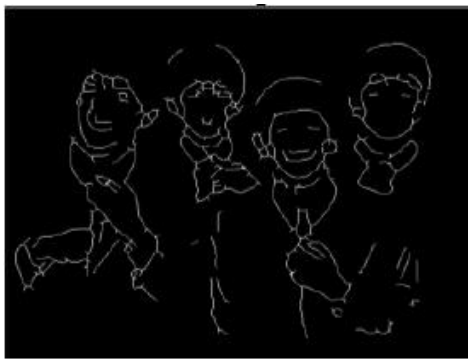
Teknik Penipisan Garis Tepi pada Citra *Grayscale*

Proses ini merupakan tahapan akhir dalam proses pendeteksian tepi pada citra *grayscale* beatles.jpg. Pada dasarnya proses threshold sudah menghasilkan garis tepi pada citra yang diteliti, hanya saja garis tepi yang dihasilkan pada proses threshold tersebut masih berupa garis tepi yang tebal sehingga beberapa objek tidak terlalu terlihat dengan jelas. Solusi dari permasalahan tersebut yaitu dengan menggunakan proses thinning yang berguna untuk menipiskan garis tepi yang ada, sehingga objek pada citra dapat terlihat dengan jelas.

Untuk melakukan proses penipisan garis tepi pada citra tersebut, maka berikut ini source code proses thinning dengan menggunakan aplikasi matlab

```
%Baca citra 'beatles.jpg'
gambar = imread('beatles.jpg');
gray = rgb2gray(gambar);
figure; imshow(gray);
title('Grayscale');
%Gaussian filter
gaussian = fspecial('gaussian',[12 12], 5)
hasil = imfilter(gray,
gaussian,'symmetric','conv');
figure; imshow(hasil), title('Blurred');
%Deteksi Tepi Horizontal
px = [-1 0 1;-1 0 1;-1 0 1];
hasil_px = filter2 (px, hasil);
figure;
imshow(hasil_px/255),
title('Tepi Horizontal');
%Deteksi Tepi Vertikal
py = [-1 -1 -1;0 0 0;1 1 1];
hasil_py = filter2 (py, hasil);
figure;
imshow(hasil_py/255),
title('Tepi Vertikal');
%Penggabungan Kedua Gradient
gradient=sqrt(hasil_px.^2+hasil_py.^2);
figure;
imshow (gradient/255),title('Gradient');
%Threshold
level = graythresh(gradient/255);
bw = im2bw(gradient/255,level);
bw = bwareaopen (bw, 50);
figure;
imshow(bw),title('Threshold');
%Thinning
bw2 = bwmorph (bw,'skel',Inf);
figure;
imshow(bw2),title('Thinning');
```

Berikut ini merupakan tampilan citra *grayscale* beatles.jpg setelah melalui proses thinning



Gambar 27: Hasil akhir deteksi tepi pada citra *grayscale*

Pada proses akhir, garis tepi objek pada citra *grayscale* beatles.jpg sudah tampak terlihat lebih jelas dibandingkan dengan proses

sebelumnya, hal tersebut dikarenakan adanya proses thinning.

Dengan menggunakan proses thinning, objek-objek pada citra *grayscale* beatles.jpg menjadi terlihat lebih jelas hingga ke objeknya yang terkecil sekalipun. Proses thinning dalam program diatas digunakan untuk merubah ukuran garis tepi yang pada citra *grayscale* beatles.jpg yang didapat dari proses *thresholding* menjadi satu piksel. Pada proses thinning pada citra *grayscale* beatles.jpg, digunakan fungsi Matlab yaitu

```
bw2=bwmorph(bw,'skel',Inf)
```

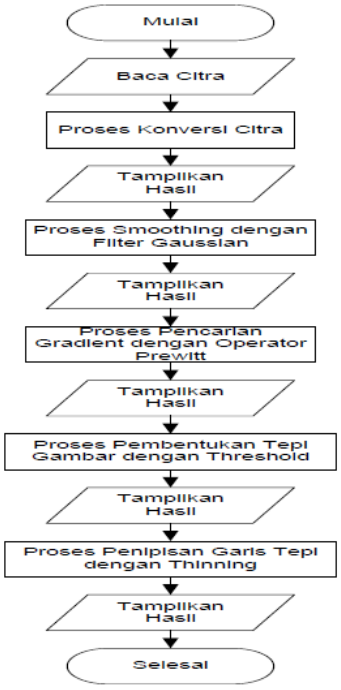
Dimana fungsi yang digunakan untuk pemrosesan thinning adalah "bwmorph", dengan menggunakan nilai "Inf" (tak terhingga) operasi "skel" menghilangkan piksel-piksel yang terdapat pada batasan objek tanpa membuat terputusnya garis tepi pada objek.

Flowchart Program

Dalam proses pendeteksian tepi pada citra RGB fruits.jpg dan citra *Grayscale* beatles.jpg, terdapat beberapa tahapan untuk mendapatkan tepi objek yang tepat. Berikut ini merupakan flowchart program untuk menghasilkan tepi objek pada citra RGB fruits.jpg dan citra *Grayscale* beatles.jpg

Deteksi Tepi pada Citra RGB

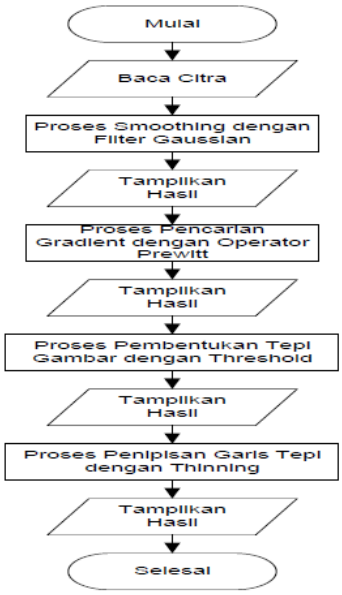
Untuk menghasilkan tepi objek pada citra RGB fruits.jpg dengan menggunakan operator canny diperlukan beberapa tahapan seperti tampak pada gambar 28



Gambar 28: Flowchart Deteksi Tepi pada Citra RGB

Deteksi Tepi pada Citra *Grayscale*

Pada proses deteksi tepi pada citra *grayscale* beatles.jpg tidak terlalu berbeda jauh dengan proses deteksi tepi pada citra RGB fruits.jpg, hanya saja pada citra *grayscale* tidak diperlukan proses konversi citra dikarenakan citra itu sendiri merupakan citra *grayscale*, berikut flowchart proses lengkap deteksi tepi pada citra *grayscale* beatles.jpg,



Gambar 29: Flowchart Deteksi Tepi pada Citra *Grayscale*

Perbandingan Citra

Pembahasan ini akan mengkaji mengenai perbandingan citra RGB dengan citra *grayscale* dari segi proses deteksi tepi, spesifikasi komputer yang dipakai hingga hasil gambar setelah dilakukan proses pengolahan citra dengan aplikasi Matlab. Dimana hasil dari pengolahan tersebut dapat dilihat pada tabel-1, berikut.

No	Indikator	Citra JPG		Keterangan
		RGB Citra fruits.jpg	Grayscale Citra beatles.jpg	
1.	Resolusi Citra	1600 x 1200	520 x 642	Resolusi pada RGB lebih besar.
2.	Kapasitas Citra	493 KB	45,9 KB	Kapasitas pada RGB lebih besar.
3.	Proses Deteksi Tepi	5 detik 16 second	2 detik 40 second	Tergantung kemampuan CPU.
4.	Hasil Deteksi Tepi	Terlihat Jelas	Terlihat Lebih Jelas	Tingkat keabuan pada citra grayscale lebih besar.
5.	Spesifikasi Perangkat	Butuh yang sepadan	Spesifikasi standar	Kapasitas dan resolusi citra RGB lebih besar.

Penutup

Pada analisa dan pembahasan masalah telah dibuktikan bahwa citra digital RGB dan *Grayscale* dapat berhasil mengalami perubahan kualitas (penurunan kualitas). Perubahan tersebut yang dimaksudkan adalah perubahan garis tepi pada citra yang terlihat jelas menjadi terlihat kurang jelas.

Metode algoritma Canny untuk proses deteksi tepi suatu citra dapat digunakan untuk meningkatkan kualitas suatu citra, untuk mendapatkan hasil yang dianggap optimal.

Aplikasi Matrix Laboratory (Matlab) 7.8.0 dapat digunakan untuk melakukan pendeteksian garis tepi pada citra digital, dimana citra yang memiliki kapasitas dan resolusi besar akan mempengaruhi kinerja aplikasi Matlab, sehingga aplikasi Matlab akan bekerja lambat saat program dijalankan dibandingkan dengan citra yang berkapasitas dan beresolusi lebih kecil.

Daftar Pustaka

[1] Y. Bambang, “Image Smoothing menggunakan Mean Filtering, Median Filtering, Modus Filtering dan Gaussian Filtering”, Universitas Pembangunan Nasional, Yogyakarta, 2010.

[2] Darma Putra, “Pengolahan Citra Digital”, Penerbit Andi, 2010.

[3] Edy Winarno, “Aplikasi Deteksi Tepi pada Realtime Video menggunakan Algoritma Canny Detection”, Universitas STIKUBANK, Semarang, 2011.

[4] Y. Ferdian dan W. Riyo, “Analisa Peningkatan Kualitas Citra Hasil Deteksi Tepi Menggunakan Dual Operator”, Lembaga Ilmu Pengetahuan Indonesia, Bandung, 2010

[5] Hendra, “Perancangan Aplikasi Peningkatan Kualitas Gambar dengan Metode Gaussian Blur”, Universitas Bina Nusantara Jakarta, 2010.

[6] Harry Ramza dan Yohannes Dewanto, “Teknik Pemrograman Menggunakan Matlab”, Grasindo, 2007.

[7] Siti Uswatun Hasanah, “Aplikasi Algoritma Hough Transform Untuk Ekstraksi Fitur-Fitur Penting pada Wajah”, Universitas Indonesia, 2004