**COMPUTER VISION AND IMAGE PROCESSING**

**Percobaan 5: Filter**



**Oleh:**

**Luthfi Aminulloh 1020181013**

**TEKNIK ELEKTRO**

**PROGRAM PASCASARJANA S2 TERAPAN**

**POLITEKNIK ELEKTRONIKA NEGERI SURABAYA**

**2019**

# Tujuan Percobaan

1. Mahasiswa mengetahui cara dan dapat merekayasa nilai setiap piksel dari gambar untuk proses penambahan nilai brightness dan nilai contrast
2. Mahasiswa mengetahui cara dan dapat mentransformasikan warna dari RGB ke grayscale
3. Mahasiswa mengetahui cara dan dapat menggabungkan dua buah gambar menjadi sebuah gambar lain
4. Mahasiswa dapat merekayasa tingkat gelap-terang gambar menggunakan teknik histogram dan histogram equalization

# Persiapan

1. Praktikum ini dapat dikerjakan dengan pra-syarat bahwa mahasiswa:

* Telah mendapatkan matematika lanjut
* Memiliki konsep dasar sinyal dan sistem
* Memiliki konsep dasar pemrograman dan pengoperasian MS Visual C++

1. Software yang diperlukan:

* Microsoft Visual C++ 2010 express atau lebih tinggi.
* OpenCV Library 2.4.9 atau lebih tinggi.

1. Sarana penunjang praktikum:

* File gambar (.bmp, .jpg)
* File video (.avi)

# Pendahuluan

## Smoothing

Smoothing, juga disebut blurring, adalah operasi pemrosesan gambar sederhana dan sering digunakan. Ada banyak alasan untuk menghaluskan gambar. Dalam praktikum ini kita akan fokus pada smoothing untuk mengurangi noise. Untuk melakukan operasi penghalusan, sebuah filter akan diterapkan ke gambar. Jenis filter yang paling umum adalah linier, di mana nilai piksel output (yaitu g(i, j)) ditentukan sebagai jumlah tertimbang dari nilai piksel masukan (yaitu f(i + k, j + l)):

g(i, j) = ∑k, lf (i + k, j + l) h (k, l)

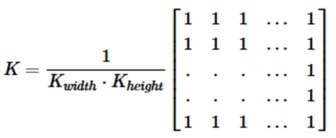
dimana h (k, l) disebut kernel, yang tidak lebih dari koefisien filter. Ini membantu untuk memvisualisasikan filter sebagai jendela koefisien meluncur di gambar.

|  |
| --- |
|  |
| Gambar . Berbagai macam filter smoothing dan hasil penerapannya |

Ada banyak jenis filter, di sini kita akan menyebutkan yang paling banyak digunakan:

* Normalized box filter

Filter ini paling sederhana dari semuanya. Setiap piksel output adalah rata-rata dari tetangga kernelnya (semuanya berkontribusi dengan bobot yang sama). Kernelnya dinyatakan dalam bentuk seperti di bawah ini:



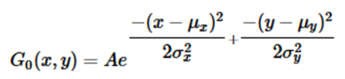
* Gaussian filter

Mungkin filter yang paling berguna (meskipun bukan yang tercepat). Penyaringan Gaussian dilakukan dengan mengaitkan setiap titik dalam array masukan dengan kernel Gaussian dan kemudian menjumlahkannya semua untuk menghasilkan larik keluaran. Hanya untuk membuat gambar lebih jelas, ingat bagaimana tampilan kernel Gaussian 1D?

|  |
| --- |
|  |
| **Gambar 2.** Distribusi fungsi gaussian |

Dengan asumsi bahwa gambar adalah 1D, Anda dapat melihat bahwa piksel yang terletak di tengah akan memiliki bobot terbesar. Berat tetangganya menurun karena jarak spasial di antara mereka dan pusat piksel meningkat.

Ingat bahwa 2D Gaussian dapat direpresentasikan sebagai:



dimana μ adalah mean (puncak) dan σ mewakili varians (masing-masing variabel x dan y).

* Median filter

Filter median berjalan melalui setiap elemen sinyal (dalam hal ini gambar) dan mengganti setiap piksel dengan median piksel tetangganya (terletak di lingkungan persegi di sekitar piksel yang dievaluasi).

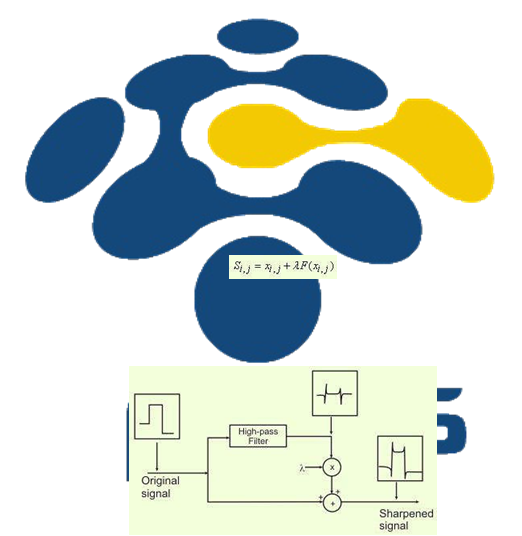
* Bilateral filter

Sejauh ini, kami telah menjelaskan beberapa filter yang tujuan utamanya adalah untuk memperhalus gambar masukan. Namun, terkadang filter tidak hanya melenyapkan kebisingan, tetapi juga menghaluskan ujungnya. Untuk menghindari hal ini (setidaknya sejauh tertentu), kita dapat menggunakan filter bilateral.

Dalam cara yang analog seperti filter Gaussian, filter bilateral juga mempertimbangkan piksel tetangga dengan bobot yang ditetapkan untuk masing- masing. Bobot ini memiliki dua komponen, yang pertama adalah bobot yang sama yang digunakan oleh filter Gaussian. Komponen kedua memperhitungkan perbedaan intensitas antara piksel tetangga dan yang dievaluasi.

## Sharpening

Persepsi manusia sangat sensitif terhadap tepi dan detail dari citra. Tepi dan detail citra terdiri dari komponen frekuensi tinggi. Kualitas visual dari sebuah citra akan berkurang jika frekuensi tinggi melemah atau hilang. Memperbaiki komponen frekuensi tinggi dari sebuah citra akan menaikkan kualitas visual. Sharpening (penajaman) citra banyak digunakan dalam percetakan dan industri fotografi untuk meningkatkan kontras lokal dan menajamkan citra. Secara prinsip, penajaman citra terdiri dari penambahan citra asli yang proporsional terhadap versi high pass filter dari citra asli.



Dimana : xi,j adalah nilai intensitas piksel asli pada koordinat (i,j), F() adalah high-pass filter, ß adalah parameter yang bernilai lebih besar dan sama dengan nol, dan Si,j adalah piksel yang ditajamkan pada koordinat (i,j). Nilai ß tergantung pada tingkat ketajaman yang diinginkan.

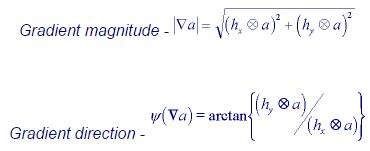
|  |
| --- |
|  |
| **Gambar 3.** Proses penajaman gambar menggunakan filter sharpening |

|  |
| --- |
|  |
| **Gambar 4.** Contoh hasil sharpening filter |

## Edge Gradient

Citra merupakan sebuah sinyal berkategorisasi 2-D, dimana citra dapat memiliki dua variabel (atau lebih) sehingga perlu didefinisikan arah dari turunan yang akan diambil (vertikal dan horisontal). Vektor turunan dari gradient ❑a(m, n) dari sebuah citra dinyatakan dengan:



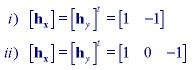


## Edge basic derivative

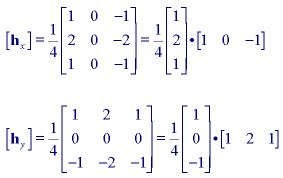
Filter ini mengacu pada materi sinyal dan sistem tentang konsep filter digital, dimana untuk mendapatkan beda/derivasi diantara dua buah data dapat menggunakan persamaan two term atau central difference filter sebagai berikut:

yn = yn - yn-1 atau yn = yn - yn-1

Oleh karena itu, dalam bentuk matriks kernel filter basic derivative dapat dituliskan sebagai berikut:

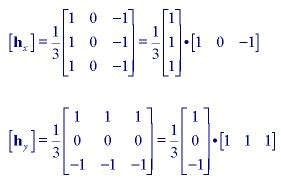


### Edge-Sobel:



|  |
| --- |
|  |
| **Gambar 5.** Hasil penerapan filter edge menggunakan algoritma Sobel |

### Edge-Prewit:



|  |
| --- |
|  |
| **Gambar 6.** Hasil penerapan filter edge menggunakan algoritma Prewitt |

# Prosedur percobaan

## Smoothing

|  |
| --- |
| List Program: |
| #include <math.h>  #include <string.h>  #include <opencv2\core\core.hpp>  #include <opencv2\highgui\highgui.hpp>  #include <opencv2\imgproc\imgproc.hpp>  using namespace cv;  //Percobaan 1 B  void main()  {  Mat img;  Mat blur;  img = imread("../data/OrangeCat.jpg", 1); // “1” >> gambar berwarna  cv::blur(img, blur, Size(3, 3), Point(-1, -1), 4);  imshow("ORIGINAL", img);  imshow("BLURRING", blur);  imwrite("blurred.jpg", blur);  waitKey(100000);  } |

Hasil Percobaan:

|  |
| --- |
|  |
|  |

Langkah Percobaan:

* Pada contoh program diatas, ukuran kernel filter yang digunakan adalah 3x3. Ubah- ubahlah ukuran kernel filter tersebut berturut-turut 7x7, 11x11, dan 15x15! Amati apakah yang terjadi dengan gambar hasilnya? Jelaskan!
* Contoh program diatas menggunakan fungsi jadi dari OpenCV, yaitu cv::blur(). Ubahlah fungsi tersebut dengan menggunakan fungsi filter2D()!
* Ubahlah program diatas dengan membuat filter buatan anda sendiri melalui proses dasar perkalian matriks! Dalam hal ini, gunakan kernel filter berukuran 3x3!
* Ubahlah program diatas untuk membuat filter gaussianblur (anda boleh menggunakan fungsi jadi dari OpenCV)! Ubah-ubahlah ukuran kernel filter tersebut berturut-turut 3x3, 7x7, 11x11, dan 15x15! Amati apakah yang terjadi dengan gambar hasilnya? Jelaskan!
* Ubahlah program diatas untuk membuat filter medianblur (anda boleh menggunakan fungsi jadi dari OpenCV)! Ubah-ubahlah ukuran kernel filter tersebut berturut-turut 3x3, 7x7, 11x11, dan 15x15! Amati apakah yang terjadi dengan gambar hasilnya? Jelaskan!
* Buatlah program untuk menggambarkan cara kerja Kuwahara filter!

## Sharpening

|  |
| --- |
| Listing Program: |
| void main()  {  Mat img; Mat sharp;  img = imread("../data/OrangeCat.jpg", 1);  GaussianBlur(img, sharp, Size(1, 1), 0, 0);  addWeighted(img, 1.5, sharp, -0.5, 0, sharp);  imshow("ORIGINAL", img);  imshow("SHARP", sharp);  waitKey(100000);  } |

Hasil Percobaan:

|  |
| --- |
|  |
|  |

* Ubah-ubahlah nilai α dan ß pada fungsi “cv::addWeighted()” diatas! Amati apa yang terjadi? Jelaskan! Catatan: nilai α dan ß jika dijumlahkan harus bernilai total “1”.
* Ubahlah program diatas dengan menggunakan program dasar perkalian dan penjumlahan matriks untuk kedua fungsi diatas: “cv::GaussianBlur() dan cv::addWeighted()”!

## Edge

|  |
| --- |
| Listing Program: |
| void main()  {  Mat img;  Mat edge, edge\_abs;  img = imread("../data/Lenna.jpg", 0);  imshow("ORIGINAL", img);  Mat output1(img.size().width, img.size().height,CV\_8UC1);  Mat output2(img.size().width, img.size().height, CV\_8UC1);  Mat grading(img.size().width, img.size().height, CV\_8UC1);  Mat abs\_grad\_x;  Mat abs\_grad\_y;  Sobel(img, output1, CV\_16S, 1, 0, 3, 1.0, 0.0, 4); //grad\_x  Sobel(img, output2, CV\_16S, 0, 1, 3, 1.0, 0.0, 4); //grad\_y    convertScaleAbs(output1,abs\_grad\_x);  convertScaleAbs(output2, abs\_grad\_y);  addWeighted(abs\_grad\_x, 0.5, abs\_grad\_y, 0.5,0,grading);  imshow("edge1", output1);  imshow("edge2", output2);  imshow("Grad", grading);  waitKey(0);  } |

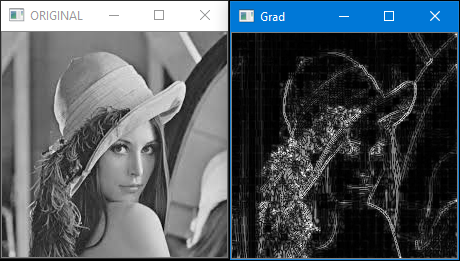
Hasil Percobaan:

|  |
| --- |
|  |
|  |

Langkah Percobaan:

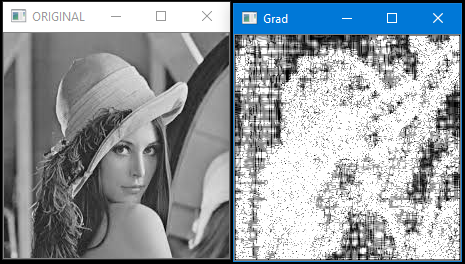
* Ubah-ubahlah parameter-parameter dalam fungsi “cv::Sobel()” berturut-turut sebagai berikut:
  + Sobel 1

|  |
| --- |
| Sobel(img, output1, CV\_16S, 2, 0, 3, 1.0, 0.0, 4);  Sobel(img, output2, CV\_16S, 0, 2, 3, 1.0, 0.0, 4); |



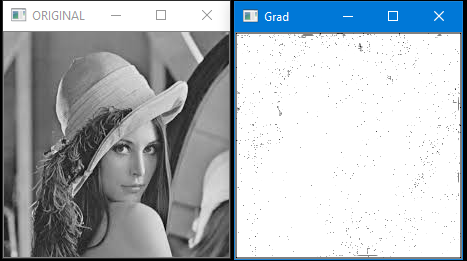
* + Sobel 2

|  |
| --- |
| Sobel(img, output1, CV\_16S, 3, 0, 7, 1.0, 0.0, 4);  Sobel(img, output2, CV\_16S, 0, 3, 7, 1.0, 0.0, 4); |



* + Sobel 3

|  |
| --- |
| Sobel(img, output1, CV\_16S, 5, 0, 11, 1.0, 0.0, 4);  Sobel(img, output2, CV\_16S, 0, 5, 11, 1.0, 0.0, 4); |



Amati apakah yang terjadi? Jelaskan!

* Contoh program diatas menggunakan fungsi jadi dari OpenCV, yaitu cv::Sobel(). Ubahlah fungsi tersebut dengan menggunakan fungsi filter2D()!
* Buatlah program untuk mendemonstrasikan hasil dari filter edge Prewitt!