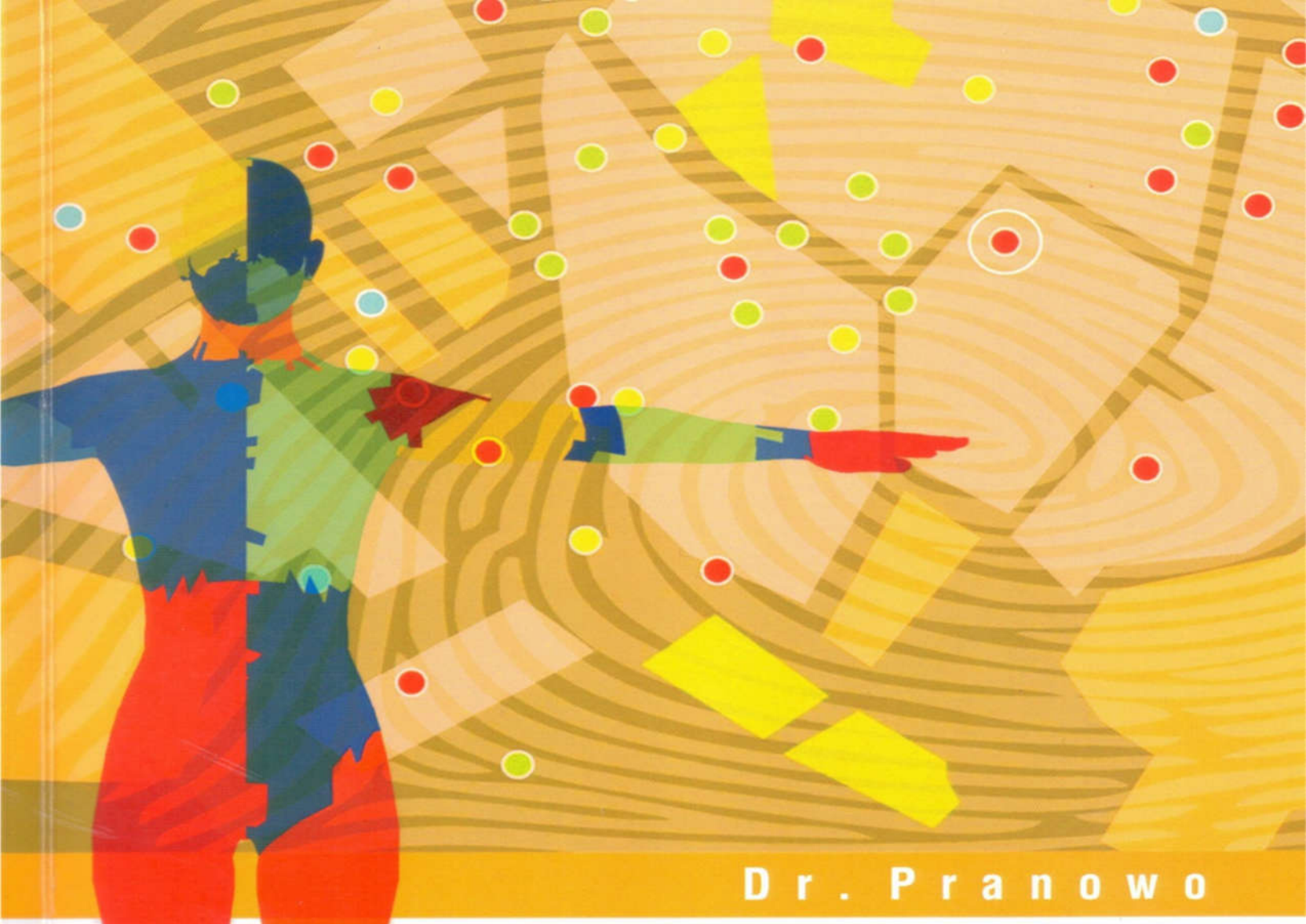


Dr. Pranowo

Pengolahan Citra Berbasis PDE dengan OpenCV

Dr. Pranowo

Pengolahan Citra BErbasis PDE dengan OpenCV

serviens in lumine veritatis

"Boys, be ambitious. Be ambitious not for money, not for selfish aggrandizement, not for the evanescent thing which men call fame. Be ambitious for the attainment of all that a man can be."

William Smith Clark *(July 31, 1826 – March 9, 1886)*

Sapporo Hitsujigaoka Observation Hill

Kupersembahkan kepada :
Istriku: Bernadetta Ayuk Martiana dan kedua anakku:
Tom & Kez

serviens in lumine veritatis

KATA PENGANTAR

Puji syukur penulis panjatkan pada Gusti Yesus "Guru Agung lan Pangon Kang Utama", atas berkat dan rahmatNya penulis dapat merampungkan penulisan buku ini. Materi dalam buku ini merupakan sebagian dari bahan kuliah Pengolahan Citra Magister Teknik Informatika UAJY. Buku ini membahas pengolahan citra dari sudut yang berbeda dari kebanyakan buku pengolahan citra yang sudah ada di toko-toko buku di Indonesia. Pembaca diharapkan sudah memahami teori dasar Pengolahan Citra, sebelum membaca buku ini. Pendekatan dalam buku ini adalah citra dianggap sebagai fungsi matematika multivariabel sehingga bisa dimodelkan dengan persamaan differensial parsial atau yang lebih dikenal dengan nama PDE. Karena dapat dimodelkan dalam PDE, pengembangan metodologi pengolahan citra mempunyai cakrawala yang luas. Metode numerik yang sudah "*well established*" dapat diadopsi dengan baik untuk pengolahan citra. Implementasi ke dalam program komputer menggunakan Bahasa C/C++ dan pustaka OpenCV. Sedangkan *compiler* yang digunakan adalah Microsoft Visual Studio 2010.

Buku ini terdiri dari 7 bab, bab pertama berisi penjelasan hubungan citra dengan PDE. Bab 2 berisi solusi numerik PDE dengan metode Beda Hingga. Bab 3 berisi uraian tentang OpenCV. Bagi pembaca yang sudah menguasai metode numerik untuk PDE dan OpenCV dapat langsung loncat ke Bab IV yang berisi tentang deteksi tepi. Bab V berisi materi tentang penapisan derau, bab VI memabahas proses segmentasi dan bab VII membahas tentang

inpainting. Materi bab IV-VII merupakan materi yang terpisah sehingga dalam mempelajarinya tidak harus urut.

Penulis berharap buku ini dapat memperkaya khasanah ilmu tentang pengolahan citra di tanah air. Penulis mengucapkan terima kasih kepada UAJY yang telah mendorong dan mengupayakan para dosen supaya bersedia menulis buku. Terima kasih juga penulis ucapkan kepada saudara Edwin Pratama, Agus Panca dan para mahasiswa peserta kuliah Pengolahan Citra Magister Teknik Informatika UAJY.

Badran, Awal Agustus 2015

Pranowo

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

serviens in lumine veritatis

BAB I
PENDAHULUAN

Citra

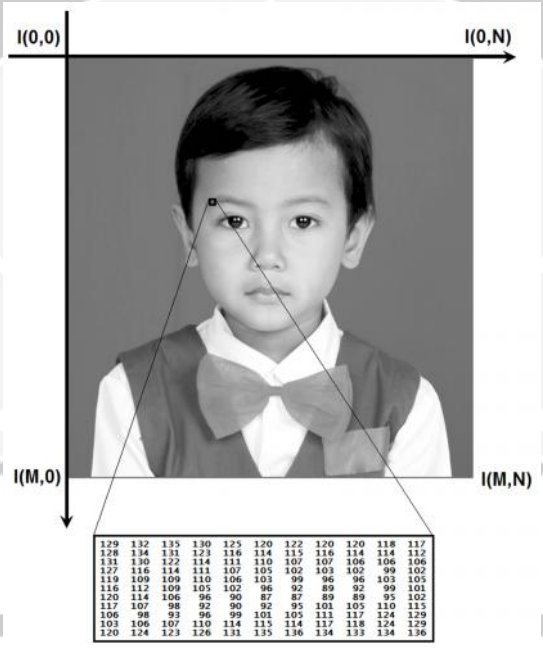
Interaksi kehidupan manusia dengan citra merupakan hal yang tidak terpisahkan. Jauh sebelum manusia menemukan huruf, manusia prasejarah sudah mampu mengekspresikan pikirannya dengan cara membuat guratan ataupun coretan yang membentuk citra di dinding gua tempat tinggalnya. Hal ini bisa dibuktikan dengan penemuan arkeologi berupa lukisan di sejumlah gua bekas tempat tinggal manusia prasejarah.

Seiring dengan perkembangan budaya dan teknologi umat manusia, peran citra sebagai media komunikasi semakin diakui keberadaannya. Hal ini karena citra mempunyai kandungan informasi yang padat, relatif mudah direplikasi dan dapat dengan mudah disebar-luaskan. Perkembangan teknologi digital terutama komputer mempunyai andil yang besar dalam meningkatkan peran dan manfaat citra dalam kehidupan manusia. Citra dalam bentuk digital lebih mudah disimpan dan diolah lebih lanjut menggunakan komputer.

Citra digital beserta cara pengolahannya dapat dimanfaatkan dalam banyak aspek kehidupan manusia. Disiplin ilmu Penginderaan Jarak Jauh mengolah citra digital permukaan bumi yang diambil dari jarak jauh oleh satelit. Sedangkan disiplin ilmu kedokteran memanfaatkan citra digital yang diambil dengan USG, *X-ray* maupun MRI untuk mendiagnosis suatu penyakit. Selain itu citra digital dapat dimanfaatkan di bidang pertahanan, klimatologi, industri mapun bidang hiburan.

Citra Digital

Citra digital adalah gambar 2-dimensi yang diperoleh dari gambar analog melalui proses pencuplikan dan kuantisasi (Aubert dan Kornprobts, 2002). Citra digital mengandung informasi intensitas atau warna dan data posisi. Citra digital abu-abu terdiri dari susunan pixel yang diatur menurut baris dan kolom, nilai pixel menunjukkan intensitas tingkat keabuan (gray level) yang berkisar antara 0 - 255. Sehingga citra tersebut dapat direpresentasikan dengan formula $I(m, n)$, I menyatakan intensitas pixel pada baris ke-m dan kolom ke-n . Urutan penomoran baris dan kolom dimulai dari pojok kiri atas, lihat Gambar 1.1



Gambar 1.1. Koordinat ruang citra digital berukuram M×N pixel

Partial Differential Equation (PDE)

Pengolahan citra mempunyai aneka ragam metode pendekatan. Contoh metode klasik yang biasa diajarkan di perguruan tinggi

adalah metode penapisan (*filtering*) dan Transformasi Fourier. Metode ini memandang citra sebagai sinyal multi-dimensi sehingga metode yang biasa digunakan untuk pengolahan sinyal juga dapat digunakan untuk pengolahan citra

Saat ini metode pengolahan citra yang lebih canggih banyak dikembangkan, menurut Aubert dan Kornprobts (2002) metode-metode tersebut dapat digolongkan menjadi 3 golongan, yaitu: metode yang berbasis stokastik, wavelet dan Persamaan Differensial Parsial (*Partial Differential Equations* disingkat menjadi PDE). Buku ini memfokuskan diri untuk membahas pengolahan citra berbasis PDE.

Pada mulanya PDE banyak digunakan untuk menyusun persamaan model matematika yang menggambarkan fenomena fisik, seperti: mekanika fluida, mekanika zat padat, perambatan gelombang elektromagnetik dan lain-lain. Para insinyur memecahkan persamaan model matematika tersebut secara numerik, salah satu metode yang termasuk tua dan sudah mapan pengembangannya adalah metode Beda Hingga (*finite difference*). Metode Beda Hingga dikembangkan berdasar teori Deret Taylor, dengan metode ini domain ruang didiskretisasi menjadi sejumlah titik nodal yang jika dihubungkan akan membentuk kotak-kotak kecil seperti jaring (mesh). Solusi numerik diperoleh di setiap titik nodal yang ada. Dengan demikian susunan pixel yang diatur menurut baris dan kolom dapat dianalogikan dengan grid diskretisasi metode Beda Hingga dan intensitas citra juga dapat dianalogikan dengan variabel pada persamaan model yang dicari solusinya. Berdasar analogi tersebut, dapat diketahui bahwa citra digital dapat dimodelkan dengan PDE dan dicari solusinya dengan metode Beda Hingga. Pengembangan pengolahan citra berbasis PDE banyak dikembangkan oleh departemen Matematika Terapan UCLA yang dipelopori oleh Profesor Osher (ftp://ftp.math.ucla.edu/pub/camreport/) dan Professor Tony Chan (Chan and Shen, 2005). Laporan riset dan naskah disertasi mahasiswa S3 dari jurusan tersebut dapat diakses di

OpenCV

OpenCV singkatan dari Open Source Computer Vision Library dan merupakan pustaka program untuk pengolahan citra maupun video. OpenCV mempunyai lebih dari 5000 algoritma yang ditulis menggunakan bahasa C/C++ dan sudah dioptimalkan. Pengembangan OpenCV pada awalnya dilakukan oleh perusahaan Intel, kemudian berkembang pesat karena siapapun boleh menambah dan memodifikasi algoritma di dalamnya. OpenCV dapat diunduh di alamat website : opencv.org. Pengenalan fungsi-fungsi utama OpenCV diberikan di Bab 3.

Sistematika Penulisan

Buku ini terdiri dari 7 bab dengan rincian seperti berikut:

BAB I PENDAHULUAN

Bab ini menguraikan pengertian citra digital, hubungan antara citra digital dan Persamaan Differensial Parsial serta pemaparan tentang Pustaka OpenCV secara singkat.

BAB II PDE DAN METODE BEDA HINGGA

Bab II menjelaskan tentang persamaan differensial parsial beserta solusi numeriknya menggunakan metode Beda Hingga (Finite Difference Method).

BAB III OPENCV

Bab ini akan membahas mengenai fungsi-fungsi utama yang terdapat dalam pustaka OpenCV.

BAB IV DETEKSI TEPI

Bab ini menjelaskan pengertian tepi citra dan kaitannya dengan turunan parsial.

BAB V PENAPISAN DERAU

Bab menjelaskan cara penapisan atau penghilangan derau pada citra digital, Persamaan yang digunakan adalah Persamaan Heat linier dan Perona Malik.

BAB VI SEGMENTASI

Bab menjelaskan segmentasi yang merupakan proses pembagian citra digital menjadi beberapa bagian atau objek.

BAB VII INPAINTING

Bab menjelaskan *Inpainting* yang merupakan proses perbaikan citra karena ada kerusakan pada bagian tertentu pada citra.

serviens in lumine veritatis

BAB II
PDE DAN METODE BEDA HINGGA

Bab II menjelaskan tentang persamaan differensial parsial beserta solusi numeriknya menggunakan metode Beda Hingga (*Finite Difference Method*) (Hoffman dan Chiang, 2000).

Klasifikasi PDE

Persamaan differensial parsial merupakan persamaan differensial yang mengandung lebih dari satu macam persamaan, untuk menyingkat penulisan dalam buku ini persamaan differensial parsial disebut dengan PDE yang sebetulnya merupakan singkatan dari partial differential equations. Karena citra digital merupakan fungsi matematika intensitas yang tergantung pada posisi pixel 2-dimensi, maka bentuk umum PDE untuk citra dapat ditulis seperti berikut:

$$A\frac{\partial^2 I}{\partial x_2} + B\frac{\partial^2 I}{\partial x \partial y} C\frac{\partial^2 I}{\partial y^2} + D\frac{\partial I}{\partial x} + E\frac{\partial I}{\partial y} + FI = G \qquad (2.1)$$

Dalam persamaan di atas $I(x, y)$ mewakili intensitas citra dan merupakan variabel tak-bebas, sedangkan variabel (x, y) adalah variabel bebas. Orde dari PDE adalah tingkat turunan tertinggi yang ada pada PDE tersebut, contohnya adalah:

$\frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} = 0$ merupakan PDE orde-2.

PDE juga digolongkan menjadi 2, yaitu: PDE linier dan non-linier. PDE disebut linier jika tidak ada suku yang mengandung perkalian variabel tak-bebas ataupun turunannya, jika ada suku yang mengandung suku demikina maka PDEnya disebut PDE non-linier. Pada umumnya PDE non-linier lebih sulit dicari solusinya dari pada PDE linier.

Karena PDE sebetulnya mewakili suatu fenomena fisis yang beraneka ragam dan mempunyai sifat yang khusus pula, maka solusinya juga berkaitan erat dengan fenomena yang dimodel-kannya. Olek karena itu PDE perlu diklasifikasi sehingga prosedur pencarian solusinya menyesuaikan dengan sifat PDE. Klasifikasi tersebut berdasar persamaan (2.1) adalah seperti berikut (Hoffman dan Chiang, 2000) :

Tabel 2.1. Klasifikasi PDE

Klasifikasi	Syarat	Contoh fenomena fisik
Hiperbolik	$B^2 > 4AC$	Perambatan gelombang
Parabolik	$B^2 = 4AC$	Perpindahan panas konduksi transien
Elliptik	$B^2 < 4AC$	Elektrostatik

Metode Beda Hingga

Solusi eksak dari PDE tersebut di atas adalah sangat sulit diperoleh secara analitik, bahkan pada banyak kasus di bidang rekayasa jawaban eksak tidak mungkin diperoleh. Untuk masa sekarang, meskipun solusi eksak tidak dapat diperoleh tetapi solusi pendekatan yang berakurasi tinggi dimungkinkan untuk diperoleh. Metode numerik menawarkan solusi pendekatan tersebut, dengan metode numerik PDE dibuat dalam bentuk diskret sehingga berbentuk sistem persamaan yang dapat diselesaikan dengan komputer. Metode numerik yang cocok untuk pemecahan PDE model citra digital adalag metode Beda Hingga karena metode ini membuat diskretisasi ruang berbentuk grid yang besesuaian dengan posisi piksel pada citra digital.

Metode Beda Hingga Satu Variabel

Metode Beda Hingga dikembangkan berdasar Deret Taylor yang mengekspansikan suatu fungsi menjadi deret polinomial. Ekspansi Deret Taylor untuk suatu fungsi *f(x)* yang mempunyai 1 variabel di sekitar nilai *x* adalah seperti berikut:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + + \frac{h}{3!}f'''(x) + O(h^4) \tag{2.2}$$

Notasi $f'(x) = \frac{df}{dx}(x)$ menyatakan turunan pertama, $O(h^1)$ menunjukkan bahwa deret dipotong sampai suku ketiga kesalahan perhitungan atau galat sebanding dengan h^4.

Pendekatan turunan pertama dari fungsi *f(x)* diperoleh memotong deret sampai suku kedua sehingga kesalahan perhitungan sebanding dengan h^2:

$$f'(x) = \frac{f(x) - f(x-h)}{h} + O(h^1) \tag{2.3}$$

Dalam perhitungan turunan tersebut kesalahan perhitungan $O(h^2)$ dibagi dengan h sehingga orde kesalahan perhitungan turunan pertama sebanding dengan $O(h^1)$. Karena pendekatan turunan pada titik x menggunakan nilai fungsi di titik x dan $x + h$, maka skema ini disebut Beda Maju (*forward difference*)

Penggunaan ekspansi arah mundur yaitu arah negatif juga dapat dilakukan, sehingga skema yang dihasilkan disebut skema Beda Mundur (*backward difference*).

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2!}f''(x) + O(h^3) \tag{2.4}$$

$$f'(x) = \frac{f(x) - f(x-h)}{h} + O(h^1) \tag{2.5}$$

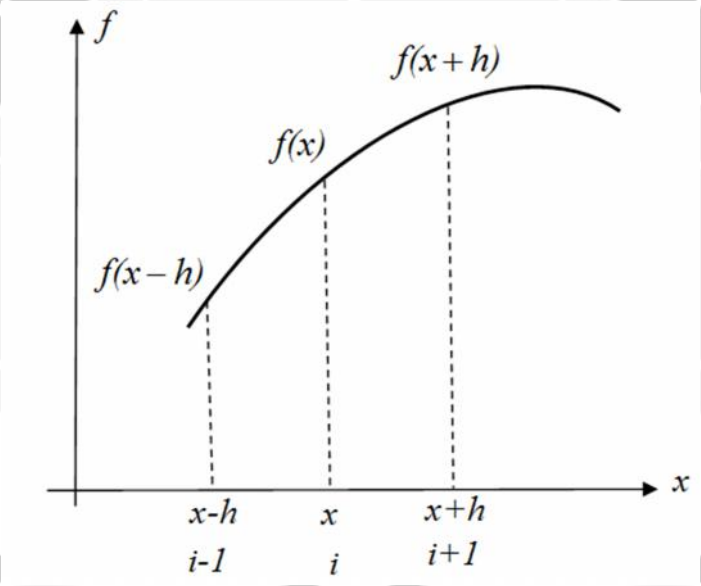
Dengan mengkombinasikan skema Beda Maju dan Beda Mundur diperoleh skema Beda Tengah (*central difference) yang mengandung kesalahan berorde O(h²)*.

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2!}f''(x) + O(h^3) \tag{2.6a}$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2!}f''(x) + O(h^3) \tag{2.6b}$$

$$f(x+h) - f(x-h) = 2hf'(x) + 2\frac{h^3}{3!}f''(x) + O(h^3) \tag{2.7}$$

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2) \tag{2.8}$$



Gambar 2.1. Skema Beda Hingga 1 variabel

Pendekatan perhitungan turunan kedua yang mempunyai kesalahan berorde h^2 diperoleh dengan mempertahankan suku orde ke- h^3.

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2!}f''(x) - \frac{h^3}{3!}f'''(x) + O(h^4) \quad (2.9a)$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2!}f''(x) - \frac{h^3}{3!}f'''(x) + O(h^4) \quad (2.9b)$$

Penjumlahan kedua persamaan di atas menghasilkan:

$$h^2f''(x) = f(x+h) - 2f(x) + f(x-h) + O(h^4) \quad (2.10)$$

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h^2) \quad (2.11)$$

Jika skema Beda Hingga dinyatakan dengan notasi berindeks hasilnya adalah:

Skema Beda Maju : $\frac{df}{dx} = \frac{f_i - f_{i-1}}{h}$

Skema Beda Mundur : $\frac{df}{dx} = \frac{f_i - f_{i-1}}{h}$

Skema Beda Tengah : $\frac{df}{dx} = \frac{f_{i+1} - f_{i-1}}{2h}$

Turunan kedua : $\frac{d^2f}{dx^2} = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2}$

Contoh perhitungan turunan suatu fungsi menggunakan pendekatan numerik metode Beda Hingga diberikan seperti berikut:

Hitung turunan pertama dan kedua fungsi $f(x) = e^x - x$ pada $x = 1$, jawaban eksaknya adalah $\frac{df(x)}{dx} = e^x - 1 = e^1 - 1 = 1{,}7183$. Jawaban numerik diperoleh dengan menentukan nilai h yang relatif kecil terlebih dahulu, misalkan : $h = 0{,}1$. Sehingga diperoleh nilai perhitungan:

$$h = 0{,}1\ ;\ x_{i-1} = x_i - h = 1 - 0{,}1 = 0{,}9\ ;\ x_i = 1\ ;\ x_{i+1} = x_i + h = 1 + 0{,}1 = 1{,}1$$

$$f(x_{i-1}) = f(0{,}9) = 1{,}5596\ ;\ f(x_i) = f(1) = 1{,}7183\ ;\ f(x_{i+1}) = f(1.1) = 1{,}9042$$

Nilai-nilai di atas kemudian dimasukkan ke dalam perhitungan metode Beda Hingga, maka diperoleh hasil perhitungan seperti berikut:

Skema Beda Maju :

$$f(x,y) + h\frac{\partial f}{\partial x}(x,y) + k\frac{\partial f}{\partial y}(x,y) + \frac{h^2}{2!}\frac{\partial^2 f}{\partial x^2}(x,y)$$

Skema Beda Tengah:

$$f'(x) = \frac{f(x_i) - f(x_{i-1})}{} = \frac{1{,}7183 - 1{,}5596}{}$$

Skema Beda Mundur:

$$f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{h} = \frac{1{,}7183 - 1{,}5596}{0{,}1}$$

Tabel 1 memperlihatkan pengaruh pemilihan nilai *h*, semakin kecil nilai h maka galat atau kesalahan perhitungan semakin kecil pula dan Skema Beda Tengah mempunyai akurasi yang lebih tinggi dibanding skema yang lain.

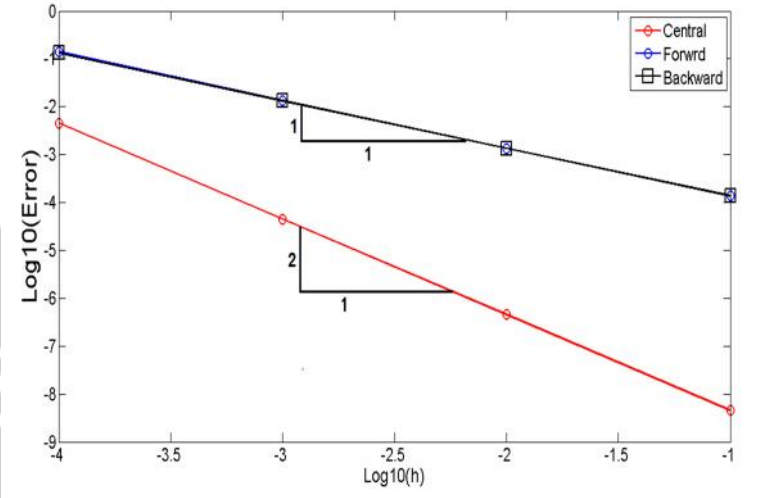
Tabel 2.2. Pengaruh Nilai *h*

h	*f'(x)* Eksak	Beda Maju		Beda Tengah		Beda Mundur	
		f'(x)	Galat	*f'(x)*	Galat	*f'(x)*	Galat
0,100	1,7183	1,8588	8.1803	1,7288	0.2638	1,5868	7,6527
0,010	1,7183	1,7319	0.7936	1,7183	0.0026	1,7047	0,7884
0,001	1,7183	1,7196	0.0791	1,7183	2.64×10^{-5}	1,7169	0.0791
0,001	1,7183	1,7184	0.0079	1,7183	2.64×10^{-7}	1,7181	0.0079

Galat perhitungan dihitung berdasar rumus:

$$\text{Galat} = \left|\frac{Numerik - Eksak}{Eksak}\right| \times 100\%$$

Gambar 2.2 menunjukkan bahwa skema Beda Tengah mempunyai akurasi orde-2 sedangkan skema Beda Maju dan Mundur hanya mempunyai akurasi orde-1, hal ini sudah sesuai dengan teori.



Gambar 2.2. Orde Akurasi

Metode Beda Hingga Dua Variabel

Pendekatan numerik Beda Hingga untuk turunan fungsi multivariabel akan dijelaskan pada bagian berikut ini. Fungsi multivariabel minimal mempunyai 2 variabel tak bebas, sehingga pendekatan turunannya juga minimal mempunyai dimensi-2. Seperti halnya fungsi dengan 1 variabel, maka pendekatan turunan fungsi multivariabel juga menggunakan Deret Taylor. Ekspansi Deret Taylor untuk fungsi f di dekat titik (x,y) adalah seperti berikut:

$$f(x+h,y+k) = f(x,y) + h\frac{\partial f}{\partial x}(x,y) + k\frac{\partial f}{\partial y}(x,y) + \frac{h^2}{2!}\frac{\partial^2 f}{\partial x^2}(x,y) + \frac{2hk}{2!}\frac{\partial^2 f}{\partial x \partial y}(x,y) + \frac{k^2}{2!}\frac{\partial^2 f}{\partial y^2}(x,y) + O(h^3) \qquad (2.12)$$

Konstanta *h* dan *k* adalah suatu konstanta yang bernilai kecil. Ekspansi di atas juga dapat ditulis dalam notasi vektor seperti ini:

$$f(\mathbf{x}+\mathbf{h}) = f(\mathbf{x}) + \mathbf{h}^{\mathbf{T}} \cdot \nabla f(\mathbf{x}) + \frac{1}{2!}\mathbf{h}^{\mathbf{T}} \cdot \nabla f(\mathbf{x}) \cdot \mathbf{h} + O(|\mathbf{h}|^3) \qquad (2.13)$$

$$\mathbf{h} = \begin{pmatrix} x \\ y \end{pmatrix}^{\mathbf{T}} = (h \quad k), \mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix}, \nabla = \frac{\partial^2}{\partial x \partial y}$$

Notasi vektor di atas berguna jika analisis numerik melibatkan suku sisa dengan turunan tinggi, tetapi karena pendekatan PDE untuk pengolahan citra tidak memerlukan turunan berorde tinggi maka notasi vektor tidak digunakan dalam buku ini. Ekspansi Deret Taylor fungsi *f(x,y)* pada arah sumbu-*x* direction adalah:

$$f(x-h,y) = f(x,y) - h\frac{\partial f}{\partial x}(x,y) + \frac{h^2\partial^2 f}{2!\,\partial x^2}(x,y) + O(h^3) \qquad (2.14a)$$

$$f(x-h,y) = f(x,y) - h\frac{\partial f}{\partial x}(x,y) + \frac{h^2\partial^2 f}{2!\,\partial x^2}(x,y) + O(h^3) \qquad (2.14b)$$

Skema Beda Maju diperoleh dari Persamaan (2.14a):

$$\frac{\partial f}{\partial x}(x,y) = f_x(x,y) = \frac{f(x,y) - f(x-h,y)}{2h} + O(h^1) \qquad (2.15a)$$

Sedangkan Skema Beda Mundur diperoleh dari Persamaan (2.14b):

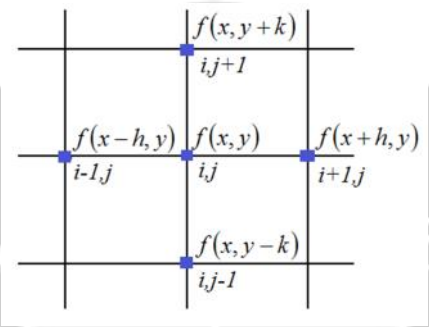
$$\frac{\partial f}{\partial x}(x,y) = f_x(x,y) = \frac{f(x,y) - f(x-h,y)}{2h} + O(h^1) \qquad (2.15b)$$

Jika Persamaan (2.14a) dikurangi Persamaan (2.14b), maka akan diperoleh pendekatan turunan skema Beda Tengah:

$$\frac{\partial f}{\partial x}(x,y) = f_x(x,y) = \frac{f(x+h,y) - f(x-h,y)}{2h} + O(h^2) \qquad (2.15c)$$

Turunan parsial kedua terhadap variabel x adalah seperti berikut:

$$\frac{\partial^2 f}{\partial x^2}(x,y) = f_{xx}(x,y) = \frac{f(x+h,y) - 2f(x+h,y) + f(x-h,y)}{h^2} + O(h^2) \qquad (2.15d)$$



Gambar 2.3. Skema Beda Hingga 2-variabel

Turunan parsial terhadap variabel y dicari dengan cara yang sama dengan di atas dan jika skema Beda Hingga 2-variabel dinyatakan dengan notasi berindeks hasilnya adalah:

Skema Beda Maju : $G_y = \frac{I}{\partial y} = \frac{i,j1 \quad i,j1}{2}$

Skema Beda Mundur : $G_y = \frac{I}{\partial y} = \frac{i,j1 \quad i,j1}{2}$

Skema Beda Tengah : $\frac{\partial f}{\partial x} = \frac{f_{i+1,j} - f_{i-1,j}}{2h}$ & $\frac{\partial f}{\partial y} = \frac{f_{i,j+1} - f_{i-1,j}}{2h}$

Sedangkan turunan kedua dapat ditulis menjadi:

$$\frac{\partial^2 f}{\partial^2 y} = \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{h^2}$$

$$\frac{\partial^2 f}{\partial^2 y} = \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{h^2}$$

Contoh perhitungan Beda Hingga untuk fungsi 2 variabel diberikan pada bagian berikut: hitung turunan pertama dan kedua dari fungsi :

$f(x,y) = 100e^{-(x^2+y^2)}$ **pada koordinat (*x*,*y*) = (1,2)**

Jawaban numerik diperoleh dengan menentukan nilai *h* yang relatif kecil misalkan : $h = k = 0{,}01$. Sehingga diperoleh nilai perhitungan:

$x_{i,j} = 1$; $x_{i-1,j} = x_{i,j} - h = 1 - 0{,}01 = 0{,}9$; $x_{i+1,j} = x_{i,j} + h = 1 + 0{,}01 = 1{,}01$

$y_{i,j} = 2$; $y_{i,j-1} = y_{i,j} - h = 2 - 0{,}01 = 1{,}9$; $y_{i,j+1} = y_{i,j} + h = 2 + 0{,}01 = 2{,}01$

$f(x_{i,j}, y_{i,j}) = 0{,}6738$; $f(x_{i-1,j}, y_{i-1,j}) = 0{,}6873$; $f(x_{i+1,j}, y_{i+1,j}) = 0{,}6604$

$f(x_{i,j-1}, y_{i,j-1}) = 0{,}7012$; $f(x_{i,j+1}, y_{i,j+1}) = 0{,}6473$

Jawaban eksak:

$$\frac{\partial f}{\partial x} = -200y\, e^{-(x^2+y^2)_{i,j}} = -1.3476$$

$$\frac{\partial f}{\partial x} = -200y\, e^{-(x^2+y^2)_{i,j}} = -2.6952$$

$$\frac{\partial^2 f}{\partial y^2} = 100e^{-2+2(x_{i,j}y_{i,j})}\left(-2+4y_{i,j}^2\right) = 1.3476 ;$$

$$\frac{\partial^2 f}{\partial y^2} = 100e^{-2+2(x_{i,j}y_{i,j})}\left(-2+4y_{i,j}^2\right) = 9.4331$$

Skema Beda Tengah untuk turunan pertama :

$$\frac{\partial f}{\partial y} = \frac{f_{i,j+1} - f_{i,j-1}}{h} = \frac{0.6473 - 0.7012}{2(0,01)} = -1.3475 ,$$

Galat = 0.0033%

$$\frac{\partial f}{\partial y} = \frac{f_{i,j+1} - f_{i,j-1}}{h} = \frac{0.6473 - 0.7012}{2(0,01)} = -2.6956 ,$$

Galat = 0.0167%

Perhitungan turunan kedua menghasilkan :

$$\frac{\partial^2 f}{\partial^2 y} = \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{h^2}$$

$$= \frac{0.6473 - 2(0,6738) + 0.7012}{(0,01)^2} = 1.3475 \text{ , Galat = } 0.0083\%$$

$$\frac{\partial^2 f}{\partial^2 y} = \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{h^2}$$

$$= \frac{0.6473 - 2(0,6738) + 0.7012}{(0,01)^2} = 9.4331 \text{ , Galat = } 0.0045\%$$

Penyelesaian PDE Dengan Metode Beda Hingga

PDE Elliptik

PDE bertipe elliptik dapat digunakan untuk pemodelan suatu peristiwa fisis yang tidak tergantung pada waktu, seperti perpindahan panas secara konduksi *steady*. Distribusi nilai besaran fisis yang ditinjau tergantung pada nilai besaran tersebut pada batas domain, nilai tersebut disebut syarat batas (*boundary condition*). Jawaban PDE elliptik merupakan jawaban PDE parabolik yang sudah mengalami keadaan steady.

PDE Elliptik Satu Dimensi

PDE Elliptik 1-dimensi mempunyai hanya 1 variabel bebas, sehingga PDE jenis ini tidak lain adalah setara dengan *Boundary Value Problem* (BVP) yang tergolong dalam Persamaan Differensial Biasa (Ordinary Differential Equations). Contoh perhitungan numerik persamaan ini disajikan seperti berikut:

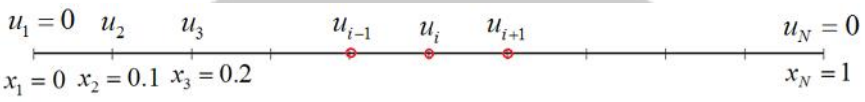
Diketahui Persamaan Elliptik satu dimensi :

$$\frac{\partial^2 u}{\partial x^2} + 2u = -1 + x^2$$

$$0 \le x \le 1 \qquad (2.16)$$

$$u(0) = 0,\ u(1) = 0$$

Selesaikan persamaan di atas dengan metode Beda Hingga, variabel u adalah variabel tak bebas, panjang domain = 1, syarat batas pada kedua ujung domain adalah $u = 0$. Panjang domain dibagi menjadi 10 bagian ruas garis, sehingga terdapat $N = 11$ titik sampel (nodal), sehingga $\Delta x = 0{,}1$. Perhitungan numerik semakin akurat jika titik nodal semakin banyak.



Gambar 2.4. Diskretisasi ruang 1-dimensi

Diskretisasi Persamaan (2.16) dengan metode Beda Hingga menghasilkan persamaan aljabar :

$$\frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2} + uf_i = 1 - (x_i)^2$$

$$\frac{1}{\Delta x^2} u_{i-1} + \left(-2 - \frac{2}{\Delta x^2}\right) u_i + \frac{1}{\Delta x^2} u_{i+1} = -1 + (x_i)^2 \qquad (2.17)$$

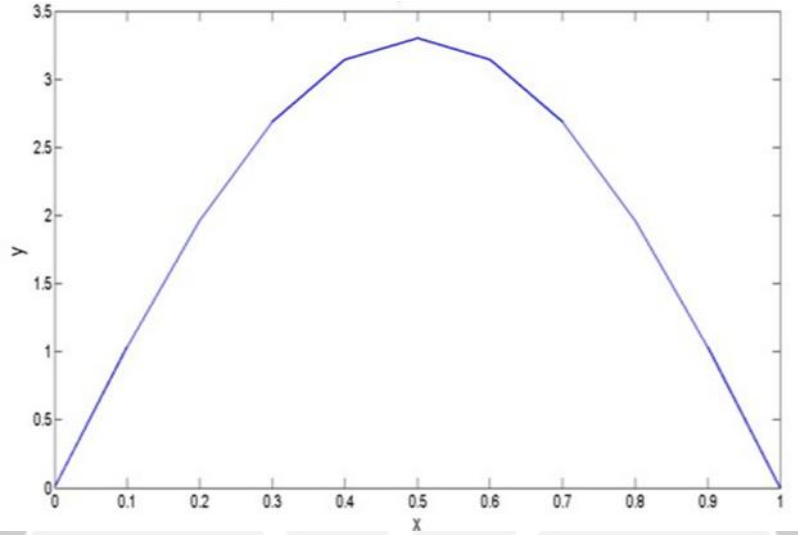
$$au_{i-1} + bu_{i-1} + cu_{i+1} = d_i$$

$$a = \frac{1}{\Delta x^2}; b = \left(-2 - \frac{2}{\Delta x^2}\right); c = \frac{1}{\Delta x^2}; d_i = -1 + (x_i)^2$$

Persamaan aljabar di atas disusun menjadi sistem persamaan linier dalam bentuk matriks :

$$\begin{bmatrix} b & c & 0 & 0 & 0 & 0 & 0 & 0 \\ a & b & c & 0 & 0 & 0 & 0 & 0 \\ 0 & a & b & c & 0 & 0 & 0 & 0 \\ 0 & 0 & a & b & c & 0 & 0 & 0 \\ 0 & 0 & 0 & a & b & c & 0 & 0 \\ 0 & 0 & 0 & 0 & a & b & c & 0 \\ 0 & 0 & 0 & 0 & 0 & a & b & c \\ 0 & 0 & 0 & 0 & 0 & 0 & a & b \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \end{bmatrix} = \begin{bmatrix} d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \\ d_8 \\ d_9 \end{bmatrix} \qquad (2.18)$$

Matriks di atas dapat dicari solusinya menngunakan metode Dekomposisi LU atau Eliminasi Gauss, dan hasilnya dapat dilihat pada Gambar 2.5:



Gambar 2.5. Solusi Numerik Persamaan Elliptik 1-dimensi

PDE Elliptik Dua Dimensi

Persamaan Elliptik 2-dimensi yang diambil sebagai contoh perhitungan adalah seperti di bawah ini.

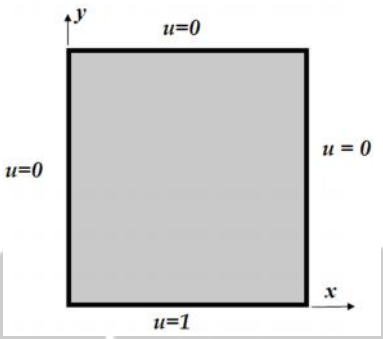
$$\frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} = 0$$

$0 < x < 1 \; ; \; 0 < y < 1$

$$u(0,y) = 0,\ u(1,y) = 0 \qquad (2.19)$$

$$u(x,0) = 1,\ u(x,1) = 0$$

Variabel *u* menyatakan suatu besaran fisis, seperti: temperatur, potensial tegangan, konsentrasi diffusi dan lain-lain. Domain ruang berukuran 1×1 , syarat batas pada sisi atas, kiri dan kanan adalah u = 0 dan pada sisi bawah adalah u = 1 seperti pada Gambar 2.6.



Gambar 2.6. Domain Ruang Persamaan Elliptik 2-D dan Syarat Batas

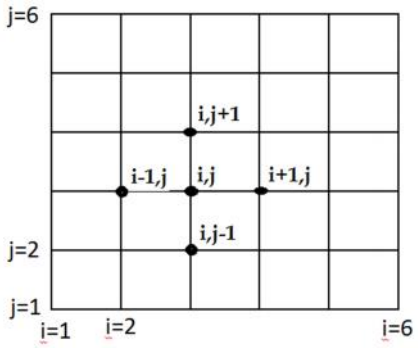
Untuk memudahkan contoh perhitungan, domain ruang dibagi menjadi 25 elemen kotak yang berukuran seragam ($\Delta x = \Delta y = 0{,}2$), lihat Gambar 2.7. Diskretisasi Beda Hingga untuk persamaan di atas dilakukan seperti berikut:

$$\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta x^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\Delta y^2} = 0$$

$$u_{i-1,j} - 2u_{i,j} + u_{i+1,j} + \left(\frac{\Delta x^2}{\Delta y^2}\right)(u_{i,j-1} - 2u_{i,j} + u_{i,j+1}) = 0 \qquad (2.20)$$

$$u_{i-1,j} + u_{i+1,j} + \beta^2 u_{i,j-1} + \beta^2 u_{i,j+1} - 2(1 + \beta^2) u_{i,j} = 0$$

$$\beta^2 = \left(\frac{\Delta x^2}{\Delta y^2}\right); \alpha = -2(1 + \beta^2)$$



Gambar 2.7. Diskretisasi Ruang (Grid) Persamaan Elliptik 2-D

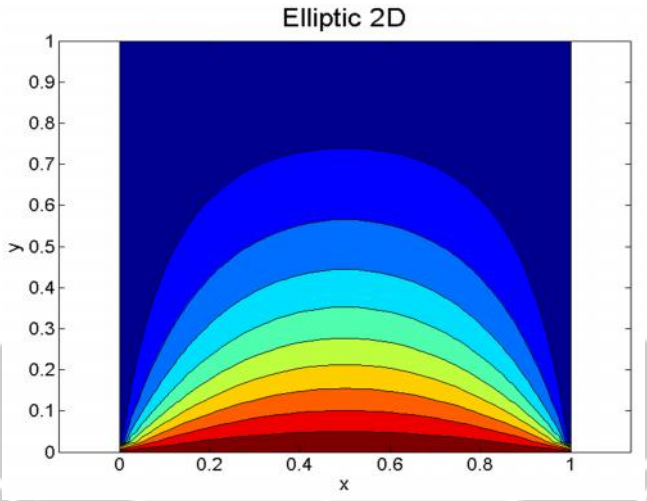
Diskretisasi persamaan dilakukan di setiap titik nodan bagian interior seperti di bawah ini:

$$
\begin{aligned}
&u_{3,2} + u_{1,2} + {}^{2}u_{2,3} + {}^{2}u_{2,1} - 2(1 + {}^{2})u_{2,2} = 0 \\
&u_{4,2} + u_{2,2} + {}^{2}u_{3,3} + {}^{2}u_{3,1} - 2(1 + {}^{2})u_{3,2} = 0 \\
&u_{5,2} + u_{3,2} + {}^{2}u_{4,3} + {}^{2}u_{4,1} - 2(1 + {}^{2})u_{4,2} = 0 \\
&u_{6,2} + u_{4,2} + {}^{2}u_{5,3} + {}^{2}u_{5,1} - 2(1 + {}^{2})u_{5,2} = 0 \\
&u_{3,3} + u_{1,3} + {}^{2}u_{2,4} + {}^{2}u_{2,2} - 2(1 + {}^{2})u_{2,3} = 0 \\
&u_{4,3} + u_{2,3} + {}^{2}u_{3,4} + {}^{2}u_{3,2} - 2(1 + {}^{2})u_{3,3} = 0 \\
&u_{5,3} + u_{3,3} + {}^{2}u_{4,4} + {}^{2}u_{4,2} - 2(1 + {}^{2})u_{4,3} = 0 \\
&u_{3,2} + u_{1,2} + {}^{2}u_{2,3} + {}^{2}u_{2,1} - 2(1 + {}^{2})u_{2,2} = 0 \\
&u_{3,2} + u_{1,2} + {}^{2}u_{2,3} + {}^{2}u_{2,1} - 2(1 + {}^{2})u_{2,2} = 0 \\
&u_{6,3} + u_{4,3} + {}^{2}u_{5,4} + {}^{2}u_{5,2} - 2(1 + {}^{2})u_{5,3} = 0 \\
&\ldots
\end{aligned}
\tag{2.21}
$$

Sistem persamaan linier akan membentuk matriks seperti berikut:

$$
\begin{bmatrix}
\alpha & 1 & 0 & 0 & \beta^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & \alpha & 1 & 0 & 0 & \beta^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & \alpha & 1 & 0 & 0 & \beta^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & \alpha & 1 & 0 & 0 & \beta^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\beta^2 & 0 & 0 & 1 & \alpha & 1 & 0 & 0 & \beta^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \beta^2 & 0 & 0 & 1 & \alpha & 1 & 0 & 0 & \beta^2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \beta^2 & 0 & 0 & 1 & \alpha & 1 & 0 & 0 & \beta^2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \beta^2 & 0 & 0 & 1 & \alpha & 1 & 0 & 0 & \beta^2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \beta^2 & 0 & 0 & 1 & \alpha & 1 & 0 & 0 & \beta^2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \beta^2 & 0 & 0 & 1 & \alpha & 1 & 0 & 0 & \beta^2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \beta^2 & 0 & 0 & 1 & \alpha & 1 & 0 & 0 & \beta^2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta^2 & 0 & 0 & 1 & \alpha & 1 & 0 & 0 & \beta^2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta^2 & 0 & 0 & 1 & \alpha & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta^2 & 0 & 0 & 1 & \alpha & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta^2 & 0 & 0 & 1 & \alpha & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta^2 & 0 & 0 & 1 & \alpha
\end{bmatrix}
\begin{bmatrix} u_{2,2} \\ u_{3,2} \\ u_{4,2} \\ u_{5,2} \\ u_{2,3} \\ u_{3,3} \\ u_{4,3} \\ u_{5,3} \\ u_{2,4} \\ u_{3,4} \\ u_{4,4} \\ u_{5,4} \\ u_{2,5} \\ u_{3,5} \\ u_{4,5} \\ u_{5,5} \end{bmatrix}
=
\begin{bmatrix} -u_{1,2} - \beta^2 u_{2,1} \\ -\beta^2 u_{3,1} \\ -\beta^2 u_{4,1} \\ -u_{6,2} - \beta^2 u_{5,1} \\ -u_{1,3} \\ 0 \\ 0 \\ -u_{6,3} \\ -u_{1,4} \\ 0 \\ 0 \\ -u_{6,4} \\ -u_{1,5} - \beta^2 u_{3,6} \\ -\beta^2 u_{3,6} \\ -\beta^2 u_{4,6} \\ -u_{6,5} - \beta^2 u_{5,6} \end{bmatrix}
$$

Setelah meyelesaikan matriks di atas dengan menggunakan metode Dekomposisi LU atau Eliminasi Gauss, maka hasil perhitungan numeriknya dapat dilihat pada Gambar 2.8:



Gambar 2.8. Hasil Perhitungan Persamaan Elliptik 2-D

PDE Parabolik

PDE ini pada umumnya mempunyai turunan kedua terhadap ruang dan turunan pertama terhadap waktu. Jawaban PDE tersebut mengambarkan perubahan suatu besaran fisis dalam domain ruang dan waktu. Sebagai contoh adalah peristiwa perpindahan panas secara konduksi secara transien dalam media logam, temperatur dalam logam mengalami perubahan terhadap waktu dan dimensi ruang. Perubahan ini tergantung pada properti material logam, temperatur awal pada keseluruhan logam dan temperatur pada batas logam. Temperatur pada keadaan mula-mula disebut keadaan awal (*initial condition*). Dalam waktu yang relatif lama temperatur berangsur-angsur menuju keadaan yang tetap dan tidak berubah lagi, keadaan ini disebut keadaan *steady*. Selain untuk pemodelan perpindahan panas, persamaan PDE tipe ini juga dapat memodelkan peristiwa pencampuran pewarna dalam air atau diffusi gas atau cairan.

PDE Parabolik Satu Dimensi

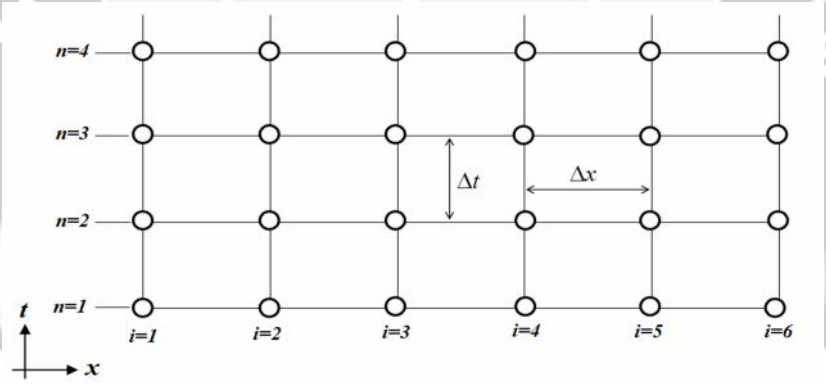
PDE Parabolik 1-dimensi merupakan PDE Parabolik yang paling sederhana, hanya mempunyai turunan tehadap variabel *x*

dan *t*. Model persamaan untuk konduksi panas 1-dimensi adalah seperti berikut:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$
$$0 \le x \le 1 \qquad (2.22)$$
$$u(x,0) = 50$$
$$u(0) = 10 \text{ , } u(1) = 20$$

Kontanta α adalah konstanta konduksi

Diskretisasi domain ruang dan waktu diperlihatkan pada Gambar 2.9, turunan kedua terhadap ruang menggunakan skema Beda Tengah, sedangkan Turunan pertama terhadap waktu dapat menggunakan skema Beda Maju atau skema Beda Mundur.



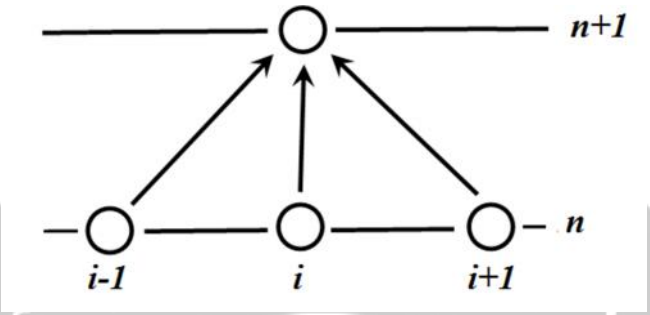
Gambar 2.9 Diskretisasi domain ruang dan waktu (*x*,*t*)

Keadaan awal adalah semua nilai variabel *u* pada level waktu *n*=1, sedangkan kondisi batas adalah nilai *u* pada *i*=1 dan *i*=6 untuk semua level waktu.

Jika turunan pertama terhadap waktu menggunakan skema Beda Maju, maka diperoleh formulasi eksplisit (Gambar 2.10):

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \alpha \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{\Delta x^2} \qquad (2.23)$$
$$u_i^{n+1} = u_i^n + \frac{\alpha \Delta t}{\Delta x^2}\left(u_{i-1}^n - 2u_i^n + u_{i+1}^n\right)$$

$CFL = \frac{\alpha \Delta t}{\Delta x^2}$ adalah Bilangan Courant Friedrich Lewy



Gambar 2.10. Formulasi Eksplisit PDE Parabolik 1-dimensi

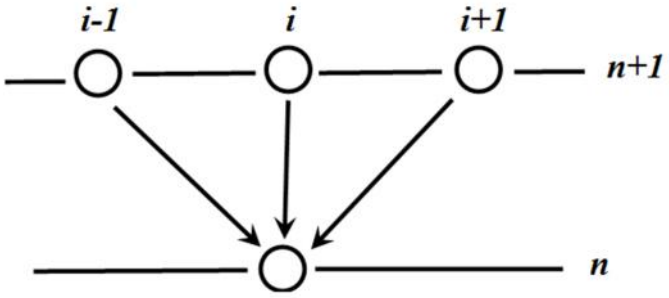
Berdasar Gambar 2.10 dapat diketahui bahwa hanya ada satu macam nilai, yaitu: u_i^{n+1} dan 3 macam nilau yang sudah diketahui $\left(u_{i-1}^n, u_i^n, u_{i+1}^n\right)$, sehingga nilai u^{n+1} dapat langsung dihitung.

Jika turunan pertama terhadap waktu menggunakan skema Beda Mundur, maka diperoleh formulasi implisit (Gambar 2.11):

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \alpha \frac{u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1}}{\Delta x^2}$$

$$\left(-\frac{\alpha \Delta t}{\Delta x^2}\right) u_{i-1}^{n+1} + \left(1 + 2\frac{\alpha \Delta t}{\Delta x^2}\right) u_i^{n+1} + \left(-\frac{\alpha \Delta t}{\Delta x^2}\right) u_{i+1}^{n+1} = u_i^n \quad (2.24)$$

$$-CFL u_{i-1}^{n+1} + (1 + 2CFL) u_i^{n+1} = CFL u_{i+1}^{n+1} = u_i^n$$



Gambar 2.11. Formulasi Implisit PDE Parabolik 1-dimensi

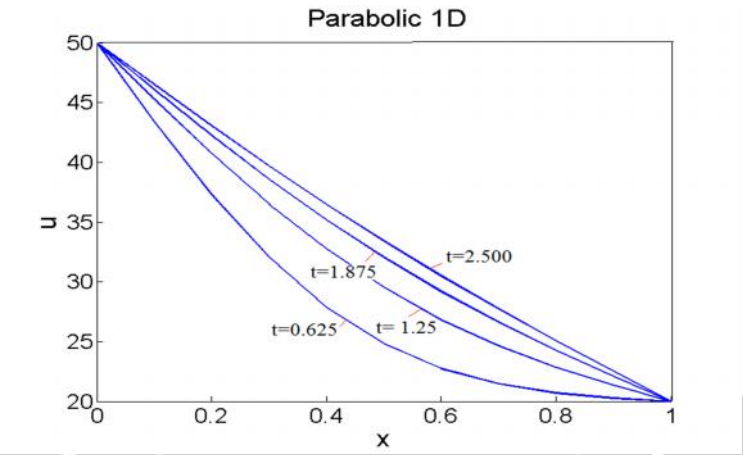
Berdasar Gambar 2.11 dan Persamaan 2.24 dapat diketahui bahwa hanya ada satu macam nilai yang diketahui, yaitu: u_4^n dan 3 macam nilai yang belum diketahui $\left(u_{i-1}^{n+1},\ u_i^{n+1},\ u_{i+1}^{n+1}\right)$. Sehingga untuk setiap level waktu n+1 harus disusun sistem persamaan linier berupa matriks terlebih dahulu baru kemudian diselesaikan untuk mendapatkan semua nilai u_i^{n+1}:

$$
\begin{aligned}
au_4^{n+1} + bu_5^{n+1} &= u^n{}_5 - cu^{n+1}{}_6 \\
au_3^{n+1} + bu_4^{n+1} + cu_5^{n+1} &= u_4^n \\
au_3^{n+1} + bu_4^{n+1} + cu_5^{n+1} &= u_4^n \\
au_4^{n+1} + bu_5^{n+1} &= u^n{}_5 - cu^{n+1}{}_6
\end{aligned}
$$

$a = -CFL$; $b = (1 + 2CFL)$; $c = CFL$

$$
\begin{bmatrix} b & c & 0 & 0 \\ a & b & c & 0 \\ 0 & a & b & c \\ 0 & 0 & a & b \end{bmatrix}
\begin{bmatrix} u_2^{n+1} \\ u_3^{n+1} \\ u_4^{n+1} \\ u^{n+1}{}_5 \end{bmatrix}
=
\begin{bmatrix} u_2^n - au_1^{n+1} \\ u_3^n \\ u_4^n \\ u^n{}_5 - cu^{n+1}{}_6 \end{bmatrix} \qquad (2.25)
$$

Keunggulan formulasi eksplisit adalah prosedur perhitungannya lebih mudah tetapi pemilihan nilai Δt dan Δx tidak bisa sembarang dan harus memenuhi persyaratan $CFL \leq 1$ atau untuk kasus 1-dimensi $\Delta t \leq \frac{\Delta x^2}{\alpha}$. Jika persyaratan ini dilanggar maka perhitungan tidak akan stabil. Sedangkan formulasi implisit pemilihan nilai Δt dan Δx bebas tidak bersyarat.



Gambar 2.12. Perubahan distribusi temperatur 1-dimensi menurut waktu

Hasil perhitungan PDE parabolik 1-dimensi dapat dilihat pada Gambar 1-dimensi untuk nodal N = 11, α = 0.1 dan Δt = 0.05, sedangkan Tabel 2.1 memperlihatkan perbandingan hasil perhitungan formulasi eksplisit dan implisit. Perbandingan menunjukkan hasil yang berdekatan.

Tabel 2.3. Hasil Perhitungan Parabolik1-D dengan formulasi eksplisit dan implisit

x	t=0.625		t=1.250		t=1.875		t=2.500	
	Eksplisit	Implisit	Eksplisit	Implisit	Eksplisit	Implisit	Eksplisit	Implisit
0.0	50.000	50.000	50.000	50.000	50.000	50.000	50.000	50.000
0.1	43.393	43.200	45.268	45.195	46.084	46.039	46.508	46.478
0.2	37.276	36.966	40.716	40.584	42.259	42.175	43.065	43.007
0.3	32.032	31.714	36.502	36.333	38.606	38.491	39.713	39.633
0.4	27.873	27.637	32.745	32.566	35.188	35.056	36.488	36.394
0.5	24.823	24.705	29.512	29.346	32.046	31.910	33.410	33.312
0.6	22.756	22.742	26.817	26.679	29.194	29.067	30.488	30.395
0.7	21.460	21.507	24.618	24.517	26.615	26.510	27.714	27.635
0.8	20.704	20.765	22.832	22.768	24.269	24.193	25.066	25.009
0.9	20.276	20.315	21.340	21.309	22.090	22.051	22.509	22.479
1.0	20.000	20.000	20.000	20.000	20.000	20.000	20.000	20.000

PDE Parabolik Dua Dimensi

Selanjutnya diskretisasi PDE parabolik-2 dimensi dijelaskan pada bagian ini. Contoh persamaan yang digunakan adalah:

$$\frac{\partial u}{\partial t} = \alpha\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{2.26}$$

$0 \le x \le 1 \; ; 0 \le y \le 1$

$u(x,y) = 0$

$u(0,y) = 0,\ u(1,y) = 0$

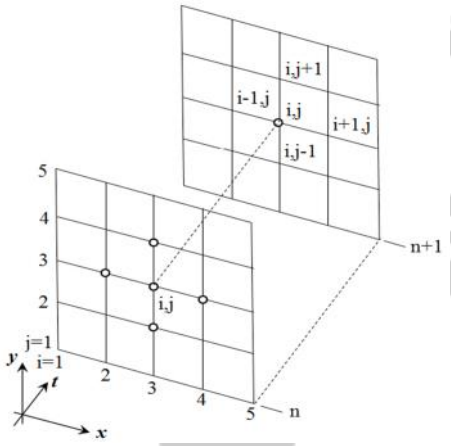
$u(x,0) = 1,\ u(x,1) = 0$

Sama dengan diskretisasi 1-dimensi, PDE parabolik 2-dimensi juga menggunakan formulasi eksplisit dan implisit. Diskretisasi Beda Hingga untuk persamaan di atas untuk formulasi eksplisit adalah:

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} = \alpha\left(\frac{u_{i,j}^{n+1} - 2u_{i,j}^{n} + u_{i-1,j}^{n}}{\Delta x^2} + \frac{u_{i,j+1}^{n} - 2u_{i,j}^{n} + u_{i,j-1}^{n}}{\Delta y^2}\right) \tag{2.27}$$

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \alpha\Delta t\left(\frac{u_{\pm 1,j}^{n} - 2u_{i,j}^{n} + u_{i-1,j}^{n}}{\Delta x^2} + \frac{u_{i,j+1}^{n} - 2u_{i,j}^{n} + u_{i,j-1}^{n}}{\Delta y^2}\right)$$

Titik nodal yang dilibatkan dalam formulasi eksplisit dapat dilihat pada Gambar 2.13.



Gambar 2.13. Formulasi Eksplisit PDE Parabolik 2-Dimensi

Sedangkan diskretisasi Beda Hingga untuk persamaan di atas untuk formulasi implisit adalah:

$$\frac{u_{i,j}^{n+1}-u_{i,j}^{n}}{\Delta t}=\alpha\left(\frac{u_{i,j}^{n+1}-2u_{i,j}^{n+1}+u_{i+1,j}^{n+1}}{\Delta x^2}+\frac{u_{i,j-1}^{n+1}-2u_{i,j}^{n+1}+u_{i,j+1}^{n+1}}{\Delta y^2}\right)$$

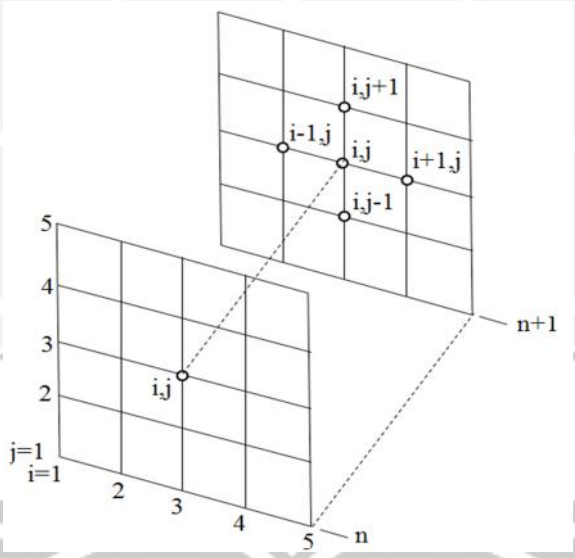
$$\frac{\alpha\Delta t}{\Delta x^2}u_{i-1,j}^{n+1}+\frac{\alpha\Delta t}{\Delta x^2}u_{i+1,j}^{n+1}-\left(1+\frac{2\alpha\Delta t}{\Delta x^2}+\frac{2\alpha\Delta t}{\Delta y^2}\right)u_{i,j}^{n+1}+\frac{\alpha\Delta t}{\Delta y^2}u_{i,j+1}^{n+1}+\frac{\alpha\Delta t}{\Delta y^2}u_{i,j-1}^{n+1}=-u_{i}^{n}$$

$$d_x=\frac{\alpha\Delta t}{\Delta y^2}\quad ;\ d_y=\frac{\alpha\Delta t}{\Delta y^2}$$

$$d_x u_{i-1,j}^{n+1}+d_x u_{i+1,j}^{n+1}-\left(1+2d_x+2d_y\right)u_{i,j}^{n+1}+d_y u_{i,j-1}^{n+1}+d_y u_{i,j+1}^{n+1}=-u_{i}^{n} \qquad (2.28)$$

$$au_{i-1,j}^{n+1}+bu_{i+1,j}^{n+1}+cu_{i,j}^{n+1}+du_{i,j-1}^{n+1}+eu_{i,j+1}^{n+1}=-u_{i}^{n}$$

Titik nodal yang dilibatkan dalam formulasi implisit dapat dilihat pada Gambar 2.14.

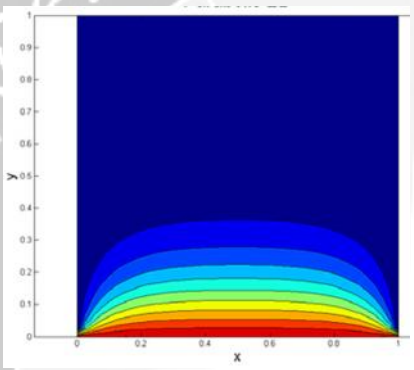


Gambar 2.14. Formulasi Implisit PDE Parabolik 2-Dimensi

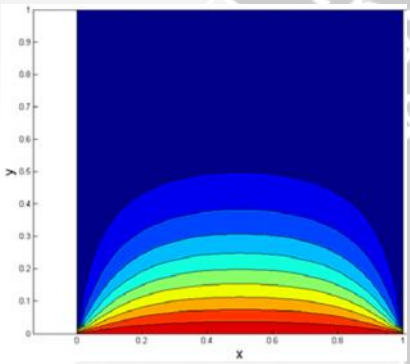
Sistem persamaan linier pada Persamaan 2.28 dapat ditulis dalam bentuk matriks:

$$\begin{bmatrix} c & e & 0 & a & 0 & 0 & 0 & 0 & 0 \\ d & c & e & 0 & a & 0 & 0 & 0 & 0 \\ 0 & d & c & e & 0 & a & 0 & 0 & 0 \\ b & 0 & d & c & e & 0 & a & 0 & 0 \\ 0 & b & 0 & d & c & e & 0 & a & 0 \\ 0 & 0 & b & 0 & d & c & e & 0 & a \\ 0 & 0 & 0 & b & 0 & d & c & e & 0 \\ 0 & 0 & 0 & 0 & b & 0 & d & c & e \\ 0 & 0 & 0 & 0 & 0 & b & 0 & d & c \end{bmatrix} \begin{bmatrix} u^{n+1}_{2,2} \\ u^{n+1}_{2,3} \\ u^{n+1}_{2,4} \\ u^{n+1}_{3,2} \\ u^{n+1}_{3,3} \\ u^{n+1}_{3,4} \\ u^{n+1}_{4,2} \\ u^{n+1}_{4,3} \\ u^{n+1}_{4,4} \end{bmatrix} = \begin{bmatrix} -u^n_{2,2} - bu^n_{1,2} - du^n_{2,1} \\ -u^n_{2,3} bu^n_{1,3} - eu^n_{2,4} \\ -u^n_{2,4} bu^n_{1,2} - eu^n_{2,5} \\ -u^n_{3,2} bu^n_{3,1} - au^n_{2,2} \\ -u^n_{3,3} - au^n_{2,3} - du^n_{4,1} \\ -u^n_{3,4} - au^n_{2,4} - eu^n_{4,5} \\ -u^n_{4,2} - au^n_{3,2} - cu^n_{4,1} \\ -u^n_{4,3} - au^n_{3,3} - cu^n_{4,2} - eu^n_{4,5} \\ -u^n_{4,4} - au^n_{3,4} - cu^n_{4,3} \end{bmatrix}$$

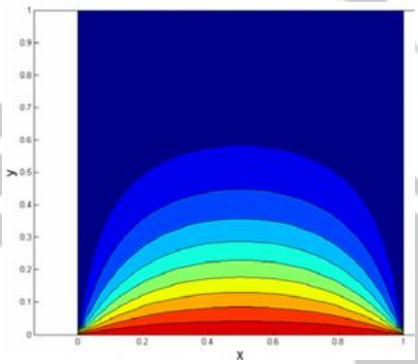
Contoh hasil perhitungan dapat dilihat pada Gambar 2.15a - 2.15f, dengan parameter $\alpha = 10$, $\Delta t = 0{,}00025$ dan $\Delta x = \Delta y = 0{,}025$.



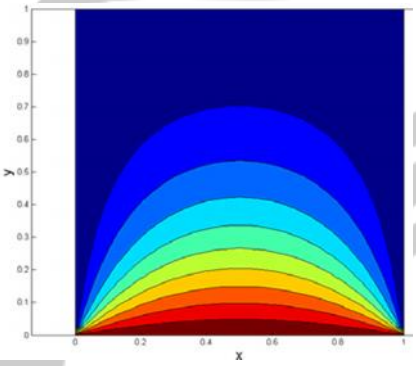
Gambar 2.15a. Temperatur saat t=0,0025



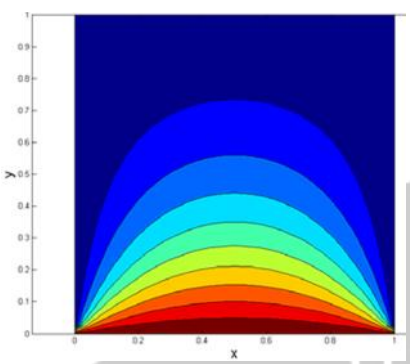
Gambar 2.15b. Temperatur saat t=0,005



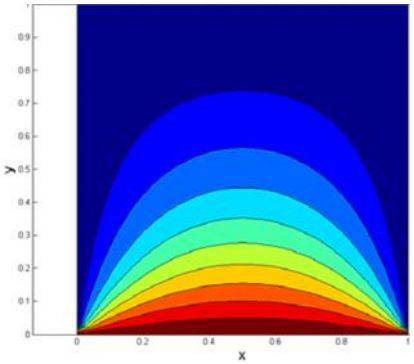
Gambar 2.14c. Temperatur saat t=0,0075



Gambar 2.15d. Temperatur saat t=0,0150



Gambar 2.15e. Temperatur saat t=0,0250



Gambar 2.15f. Temperatur saat t=0,0500

PDE Hiperbolik

PDE Hiperbolik biasanya mempunyai orde turunan terhadap waktu dan turunan terhadap ruang yang sama. PDE ini memodelkan fenomena suatu perambatan gelombang.

PDE Hiperbolik Satu Dimensi

Contoh PDE Hiperbolik 1-dimensi adalah seperti di bawah ini:

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0$$

$$0 \le x \le 1 \qquad (2.29)$$

$$IC : u(x,0) = e^{-300(x-0.15)^2}$$

$$BC : u(0) = 0,\ u(1) = 0$$

Persamaan ini memodelkan suatu perambatan sinyal dari kiri ke kanan, di mana konstana a menyatakan kecepatan perambatan. Diskretisasi waktu menggunakan formulasi eksplisit dan diskretisasi ruang dalam contoh ini diberikan 2 macam, yaitu diskretisasi dengan skema Beda Tengah dan Beda Mundur atau juga disebut skema Upwind.

Skema Beda Tengah :

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + a\frac{-u_{i-1}^n + u_{i+1}^n}{2\Delta x} = 0$$

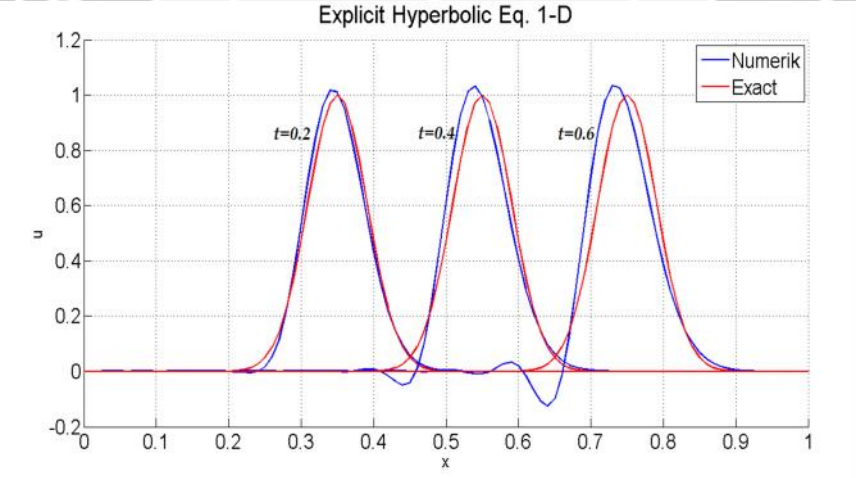
$$u_i^{n+1} = u_i^n + \frac{a\Delta t}{2\Delta x}\left(-u_{i-1}^n + u_{i+1}^n\right) \qquad (2.30a)$$

Skema Beda Mundur :

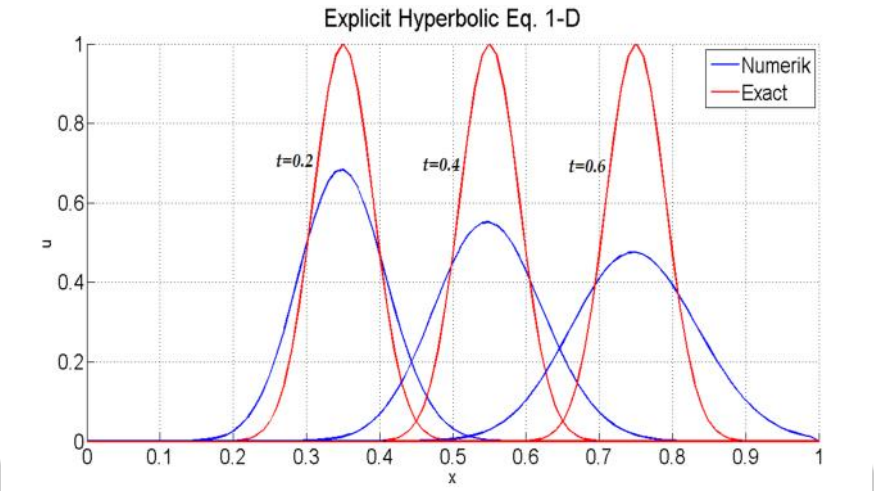
$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + a\frac{-u_{i-1}^n + u_i^n}{\Delta x} = 0$$

$$u_i^{n+1} = u_i^n + \frac{a\Delta t}{\Delta x}\left(-u_{i-1}^n + u_i^n\right) \qquad (2.30b)$$

Hasil perhitungan diperlihatkan pada Gambar 2.15a dan .15b.



Gambar 2.16.a. Hasil Perhitungan PDE Hiperbolik 1-D skema Beda Tengah



Gambar 2.16b. Hasil Perhitungan PDE Hiperbolik 1-D skema Beda Mundur

Dari hasil perhitungan tersebut dapat dilihat bahwa sinyal menjalar dari kiri ke kanan, amplitudo pada skema Beda Tengah tidak mengalami penurunan tetatp terjadi osilasi numerik yang akan bertambah kuat seiring dengan meningkatnya waktu. Sedangkan pada skema Beda Mundur, osilasi numerik tidak ada amplitudo mengalami penurunan terus.

PDE Hiperbolik Dua Dimensi

Cara siskretisasi PDE Hiperbolik 1-dimensi sama hal dengan PDE Hiperbolik 1-dimensi, persamaan beserta di:skretisasinya adalah seperti berikut

$$\frac{\partial u}{\partial t} + a\left|\left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}\right|\right) = 0$$

$$0 \le x \le 1 \; ; \; 0 \le y \le 1$$

$$IC \; : u(x,0) = e^{-300((x-0.15)^2 \; + (y-0.15)^2)} \qquad (2.31)$$

$$BC : u(0,y) = 0, \; u(1,y) = \; 0$$

$$u(x,0) = 1, \; u(x,1) = \; 0$$

Skema Beda Tengah :

$$\frac{u_{i,j}^{n+1}-u_{i,j}^{n}}{\Delta t}+a\left(\frac{-u_{i-1,j}^{n}+u_{i+1,j}^{n}}{2\Delta x}+\frac{-u_{i,j-1}^{n}-u_{i,j-1}^{n}}{2\Delta y}\right)=0$$

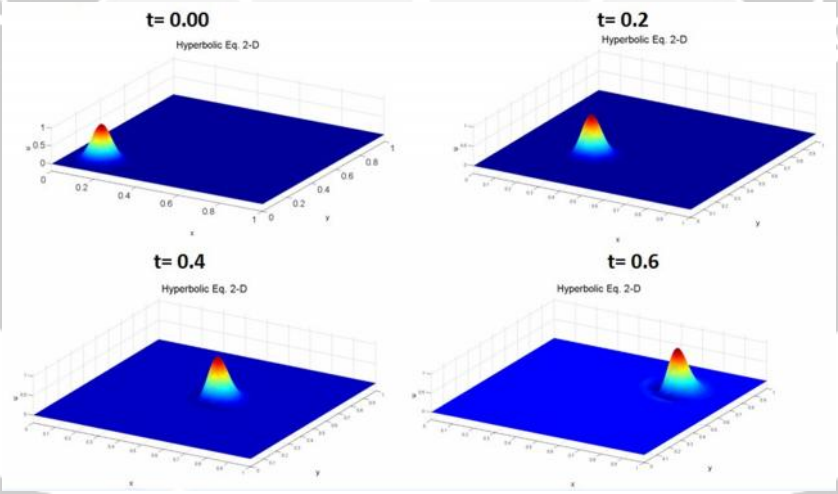
$$u_{i,j}^{n+1}=u_{i,j}^{n}-a\Delta t\left(\frac{-u_{i-1,j}^{n}+u_{i+1,j}^{n}}{2\Delta x}+\frac{-u_{i,j-1}^{n}+u_{i,j+1}^{n}}{2\Delta y}\right) \quad (2.32a)$$

Skema Beda Mundur :

$$A=\frac{u_{i,j}^{n+1}-u_{i,j}^{n}}{\Delta t}+a\left(\frac{-u_{i-1,j}^{n}+u_{i+1,j}^{n}}{\Delta x}+\frac{-u_{i,j-1}^{n}-u_{i,j-1}^{n}}{\Delta y}\right)=0$$

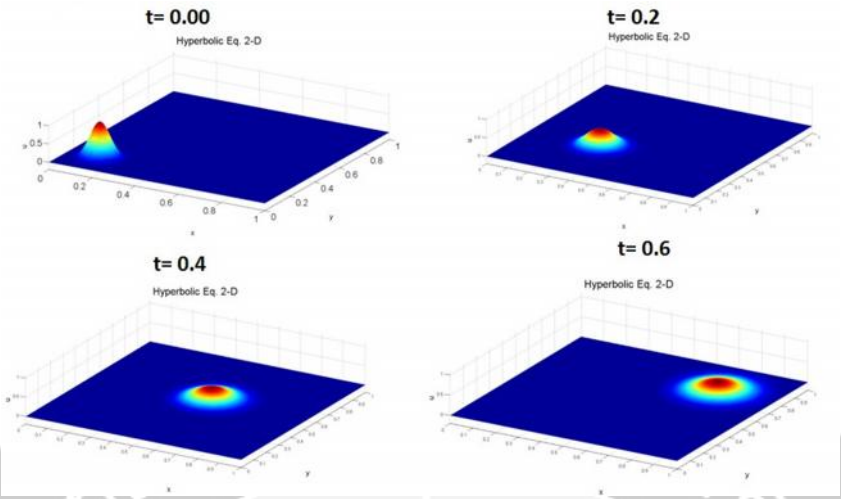
$$A=u_{i,j}^{n+1}=u_{i,j}^{n}-a\Delta t\left(\frac{-u_{i-1,j}^{n}+u_{i,j}^{n}-u_{i,j-1}^{n}+u_{i,j}^{n}}{\Delta x}+\frac{-u_{i,j-1}^{n}}{\Delta y}\right) \quad (2.32b)$$

Hasil perhitungan diperlihatkan pada Gambar 2.17 dan 2.18.



Gambar 2.17. Hasil Perhitungan PDE Hiperbolik 2-D skema Beda Tengah

Pada Gambar di atas, terlihat bahwa juga terjadi osilasi numerik yang menguat sengan meningkatnya waktu tetapi tidak terjadi penurunan amplitudo.



Gambar 2.18. Hasil Perhitungan PDE Hiperbolik 2-D skema Beda Mundur

Sama halnya kasus 1-dimensi, pada kasus 2-dimensi dengan skema Beda Mundur amplitudo tidak mengalami osilasi numerik tetapi mengalami penurunan yang tajam.

BAB III
OPENCV

Pengantar

Bab ini akan membahas mengenai perintah atau fungsi yang terdapat dalam pustaka OpenCV. Meskipun banyak terdapat fungsi pengolahan citra dalam OpenCV, hanya beberapa fungsi tertentu yang ditampilkan dalam buku ini. Fungsi. Hanya fungsi-fungsi yang bersifat mendasar yang akan digunakan, karena buku ini menekankan pada cara memahami algoritma pengolahan citra berbasis PDE dengan OpenCV bukan pada cara penggunaan fungsi-fungsi *built- in* OpenCV yang canggih. Sehingga diharapkan buku ini dapat memberikan perspektif yang berbeda dalam hal pengolahan citra.

Sejarah Singkat OpenCV

OpenCV merupakan pustaka untuk pengolahan citra yang bersifat *open source*, OpenCV berfokus pada pengembangan *computer vision*. *Computer vision* merupakan analisis citra dan video secara otomatis dengan komputer untuk mendapatkan pemahaman keadaan lingkungan sekitar (Howe, 2014). *Computer vision* berusaha mereplikasi cara sistem manusia melihat dan memahami lingkungan sekitarnya. Pustaka OpenCV ditulis dalam bahasa C dan C++ yang dapat berjalan di sistem operasi Linux, Windows dan OS X. OpenCV mempunyai lebih dari 500 fungsi yang sudah dioptimasi sehingga kehandalannya tidak diragukan. OpenCV juga mempunyai *interopability* yang tinggi, sehingga

dapat dioperasikan menggunakan *interface* bahasa Python, Ruby dan Matlab.

OpenCV pada awalnya dikembangkan oleh kelompok peneliti Perusahaan Intel di Rusia dan pertama kali diperkenalkan di publik pada saat acara IEEE Conference on Computer Vision and Pattern Recognition pada tahun 2000. Meskipun pada awalnya dikembangkan oleh perusahaan komersil, tetapi OpenCV merupakan tetap bersifat terbuka dan sekarang dimiliki oleh yayasan nirlaba OpenCV.org. Selaras dengan hal tersebut OpenCV dikembangkan dengan tujuan kurang lebih seperti berikut:

- Mendorong pengembangan riset *computer vision* dengan menyediakan *code* yang optimal dan bersifat terbuka.
- Penyebar-luasan pengetahuan dengan menyediakan infrastruktur yang dapat digunakan developer sehingga program menjadi lebih mudah dipahami dan disebar-luaskan.
- Mendorong pengembangan cde yang bersifat *portable*, optimal dan bersifat terbuka.

Fungsi-Fungsi Utama OpenCV

Kelas Input Output

- **Kelas imread**

Imread merupakan akronim dari "Image Read" yang berarti membaca citra. Sesuai dengan artinya perintah ini digunakan untuk mengambil nilai matrik pada citra digital kemudian akan disimpan dalam objek Mat. Sehingga nilai matrik dari citra digital dapat diolah sesuai dengan kebutuhan.

Class ini membutuhkan input lokasi citra dan variable *flag*. Berikut ini konstruksinya

```
Mat imread(const string& filename, int flags=1)
```

)

Flag yang ada adalah

- CV_LOAD_IMAGE_ANYDEPTH = mengambil nilai gambar sesuai dengan aslinya. Jika berwarna maka akan disimpan dalam 3 kanal, jika abu-abu maka akan disimpan dalam 1 kanal.
- CV_LOAD_IMAGE_COLOR = konfigurasi untuk mengambil warna dalam 3 kanal.
- CV_LOAD_IMAGE_GRAYSCALE = konfigurasi untuk mengambil warna untuk 1 kanal saja. Citra berwarna akan dikonversi menjadi abu-abu.

```
Mat img = imread("gambar.jpg", CV_LOAD_IMAGE_GRAYSCALE);
```

- **Kelas imwrite**

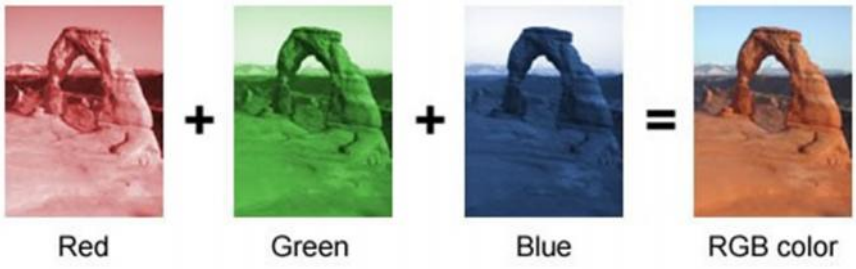
  Kebalikan dari `imread` perintah `imwrite` memiliki akronim "Image write" yang berarti menulis citra. Perintah ini biasa digunakan untuk membuat suatu citra digital yang berasal dari suatu objek Mat.

  Contoh pengguaan `imwrite` adalah:

```
Mat img = imread("alpha.jpg", CV_LOAD_IMAGE_GRAYSCALE);
imwrite("alphacopy.jpg", img);
```

Kelas Mat

Kelas Mat ini merupakan tipe data yang dapat digunakan untuk menyimpan *n* dimensi data (Bradski dan Kaehler, 2008),. Kelas ini biasa digunakan untuk menampung nilai vektor atau matriks. Kelas ini biasanya digunakan untuk menampung nilai matrik suatu gambar. Untuk gambar berwarna, matrik akan disimpan dalam 3 kanal secara otomatis, dimana tiap kanal akan mewakili matriks nilai dari warna merah, hijau dan biru. Sedangkan untuk gambar hitam putih atau grayscale dibutuhkan satu kanal saja.



Gambar 3.1. Ilustrasi Matrik RGB.

(Sumber Gambar : https://helpx.adobe.com/legacy/kb/using-image-alpha-channels-authorware/_jcr_content/main-pars/img_2.img.png/composite_rgb.jpg)

Setelah menjadi objek Mat, maka suatu matrik dapat diketahui dimensi matrik dengan menggunakan properti rows dan cols. Contoh:

```
Mat img = imread("gambar.jpg", CV_LOAD_IMAGE_GRAYSCALE);
cout<<"dimensi matrik adalah<<img.rows<<"x"<<img.cols;
```

Contoh implementasi perintah-perintah di atas ke dalam program komputer adalah seperti berikut:

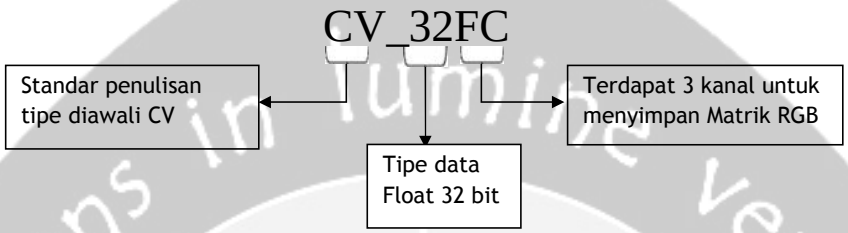
```
#include <iostream>

/* First OpenCV  program.
*/
#include <iostream>
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"


using namespace std;
using namespace cv;

int main(int argc, char*  argv[])
{
```

```
printf(“Hello world\n”);
// membaca file  citra
Mat image = imread(“Kezia.png  “);
// menampilan  citra
imshow(“Citra pertamaku”,  image);
cout<<”dimensi matrik =  “<<image.rows<<”x”<<image.cols;
// menulis file citra
imwrite(“Citra_pertamaku.png”, image);
// tunggu selama 2000 ms (jika disi 0 artinya menunggu sampai
ada kunci yang  ditekan)
waitKey(2000);
return EXIT_SUCCESS;
}
```

Setelah dijalankan, program tersebut menghasilkan tampilan berikut:

Gambar 3.2. Hasil program pertama

Untuk membuat objek *Mat* dibutuhkan parameter tipe data bentukan yang dibuat oleh OpenCV sebagai salah satu parameter dari kelas Mat. Tipe data bentukan ini yang akan digunakan sebagai referensi oleh tiap elemen matrik pada objek `Mat` sebagai tipe data. Tipe data ini juga menyertakan jumlah kanal yang akan digunakan. Tipe penulisan tipe data bentukan pada kelas Mat menggunakan format:



Gambar.3.3. Cara penulisan tipe data bentukan kelas Mat

Tabel 3.1 Tipe data bentukan kelas Mat

Tipe	Keterangan	Jangkauan Nilai
CV_8U	8 bit unsigned integers	0..255
CV_8S	8 bit signed integers	-128..127
CV_16U	16 bit unsigned integers	0..65535
CV_16S	16 bit signed integers	-32768..32767
CV_32S	32 bit signed integers	-2147483648..2147483647
CV_32F	32 bit floating	-FLT_MAX..FLT_MAX, INF, NAN
CV_64F	64 bit floating	-DBL_MAX..DBL_MAX, INF, NAN

Penulisan tipe data bentukan ini diawali dengan menuliskan tipe data, kemudian diikuti dengan jumlah kanal yang digunakan. Contoh: membutuhkan tipe data float untuk tiap element matrik citra grayskale, maka tipe data yang digunakan adalah CV_32FC1. Tipe data tersebut berarti menggunakan tipe data float 32 bit dan hanya menggunakan 1 kanal saja.

Tipe data merupakan hal yang penting dalam kelas `Mat`, oleh karena itu seringkali dibutuhkan konversi dari satu tipe data ke tipe data yang lain. Hal ini dapat dilakukan dengan cara menggunakan fungsi `convertTo`. Contoh penggunaan fungsi:

```
Mat img = imread("image.jpg"); // loading a 8UC3 image
Mat grey;
cvtColor(img, grey, CV_BGR2GRAY);
src.convertTo(dst, CV_32F);
```

Matriks dapat dibuat dengan cara dibaca dari suatu citra ataupun dibuat sendiri dengan code. Beberapa cara yang biasa digunakan untuk membuat objek `Mat`:

a. Fungsi `create`

Fungsi ini untuk mengubah matrik dari hasil inisialisasi.

```
Mat M(7,7,CV_32FC2,Scalar(1,3));
// mengubah M menjadi berukuran 100x60 15-channel 8-bit matrix. Data lama akan dihapus
M.create(100,60,CV_8UC(15));
```

b. Fungsi `zero`

Fungsi ini membuat matrik dengan isi 0 pada tiap elemennya

```
Mat A;
A = Mat::zeros(3, 3, CV_32F);
```

c. Fungsi `eye`

Fungsi ini membuat matrik identitias

```
// membuat matriks identitas 4x4
Mat A = Mat::eye(4, 4, CV_32F)*0.1;
```

d. Fungsi `ones`

Fungsi ini untuk membuat matrik dengan isi 1 pada tiap elemennya. Dapat dikombinasikan dengan perkalian scalar.

```
Mat A = Mat::ones(100, 100,  CV_8U)*3;
      //membuat matriks 100x100 berisi angka  3.
```

e. Fungsi `clone` dan `copyto`

Kedua fungsi ini mempunyai kegunaan yang sama yaitu digunakan untuk menduplikasi suatu matrik. Perbedaan yang ada hanyalah pemanggilan fungsi.

```
Mat A = Mat::ones(100, 100, CV_8U)*3;
Mat B= A.clone();
Mat C;
A.copyTo(C);
```

f. Inisialisasi yang dipisahkan dengan koma

Fungsi ini biasa digunakan untuk membuat matrik dengan nilai yang bervariasi.

```
// Membuat matriks identitas double-precision  3x3
Mat M = (Mat_<double>(3,3) << 1, 0, 0, 0, 1, 0, 0,
0, 1);
```

Pengaksesan matrik menggunakan fungsi `at`. Fungsi ini sangat berguna untuk pengaksesan per elemen dari matrik.

```
Mat H(100, 100,  CV_64F);
for(int i = 0; i < H.rows; i++)
 for(int j = 0; j < H.cols; j++)
 H.at<double>(i,j)=1./(i+j+1);
```

Dengan menggunakan class `Mat` maka akan sangat memudahkan dalam melakukan operasi matrik. Karena class ini banyak melakukan overriding operator dan penambahan fungsi manipulasi matrik, sehingga operasi matrik yang rumit menjadi terlihat lebih mudah.

Tabel 3.2. Operator pada kelas `Mat`

Operator	Penggunaan	Keterangan
+	A+B	Penjumlahan matrik A dan matrik B
	A+s atau s+A	Penjumlahan matrik A dan bilangan scalar
-	A-B	Pengurangan matrik A dan matrik B
	A-s atau s-A	Pengurangan matrik A dengan bilangan scalar
	-A	Negasi bilangan A
*	A * B	Perkalian matrik A dan B
/	A/B	Pembagian semua elemen dari matriks A dengan semua element matrik B

Tabel 3.3. Fungsi pada kelas `Mat`

Fungsi	Penggunaan	Keterangan
`mul()`	`A.mul(B)`	Perkalian element matrik A dengan elemen matrik B (element wise)
`t()`	`A.t()`	Transpose matrik
`inv()`	`A.inv()`	Mencari inverse matrik
`min()`	`A.min()`	Nilai minimum dari matrik
`max()`	`A.max()`	Nilai maximum dari matrik
`abs()`	`A.abs()`	Membuat matrik menjadi nilai absolute(positif).

serviens in lumine veritatis

BAB IV
DETEKSI TEPI

4.1 Definisi Deteksi Tepi

Tepi (*edge*) citra merupakan perubahan nilai intensitas piksel yang tajam (diskontinyuitas) terhadap jarak antar piksel yang berdekatan. Orientasi tepi tersebut dapat mendatar, tegak maupun miring. Dengan demikian deteksi tepi dapat diartikan sebagai proses pencarian tepi dan tujuannya adalah untuk pengenalan objek di dalam citra dan proses segmentasi. Secara matematis pengertian tepi citra tersebut di atas dapat didekati dengan pengertian gradien atau turunan parsial intensitas citra. Dengan menggunakan metode Beda Hingga turunan pertama tersebut dapat ditulis:

$$\nabla I = \begin{bmatrix} \frac{\partial I}{\partial x} & \frac{\partial I}{\partial x} \end{bmatrix}^T = \begin{bmatrix} G_x & G_y \end{bmatrix}^T \tag{4.1}$$

$$G_y = \frac{\partial I}{\partial y} = \frac{I_{i,j+1} - I_{i,j-1}}{2h}$$

$$G_y = \frac{\partial I}{\partial y} = \frac{I_{i,j+1} - I_{i,j-1}}{2h}$$

Untuk mempermudah perhitungan maka jarak mendatar dan tegak antara 2 nilai piksel berdekatan dibuat satu, sehingga persamaan di atas menjadi:

$$G_x = \frac{\partial I}{\partial y} = \frac{I_{i+1,j} - I_{i-1,j}}{2}$$
$$G_y = \frac{\partial I}{\partial y} = \frac{I_{i,j+1} - I_{i,j-1}}{2} \quad (4.2)$$

Proses perhitungan gradien dengan metode Beda Hingga setara dengan konvolusi citra dengan mask :

$$G_x = [-1 \quad 0 \quad 1)$$

$$G_y = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$

Magnitudo tepi dapat dihitung dengan rumus jarak euclidian:

$$G = \delta = \frac{1}{\pi} \frac{\varepsilon}{\phi^2 + \varepsilon^2} \quad (4.3)$$

Sedangkan sudut orientasi tepi dihitung dengan rumus :

$$\alpha = \tan^{-1} \frac{G_y}{G_x} \quad (4.4)$$

Berdasarkan ide di atas, Sobel (Solomon dan Breckon, 2010) mengusulkan *mask* untuk deteksi tepi seperti berikut:

$$S_x = \frac{1}{} \begin{bmatrix} 1 & 0 & -1 \\ 0 & -1 & \\ 1 & 0 & -1 \end{bmatrix} \text{ dan } S_y = \nabla\phi = \begin{bmatrix} \frac{\partial\phi}{\partial x} & \frac{\partial\phi}{\partial y} \end{bmatrix}^T \quad (4.5)$$

Sedangkan Prewitt (Solomon dan Breckon, 2010) mengusulkan masknya seperti berikut:

$$P_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \text{ dan } P_y = \nabla\phi = \begin{bmatrix} \frac{\partial\phi}{\partial x} & \frac{\partial\phi}{\partial y} \end{bmatrix}^T$$

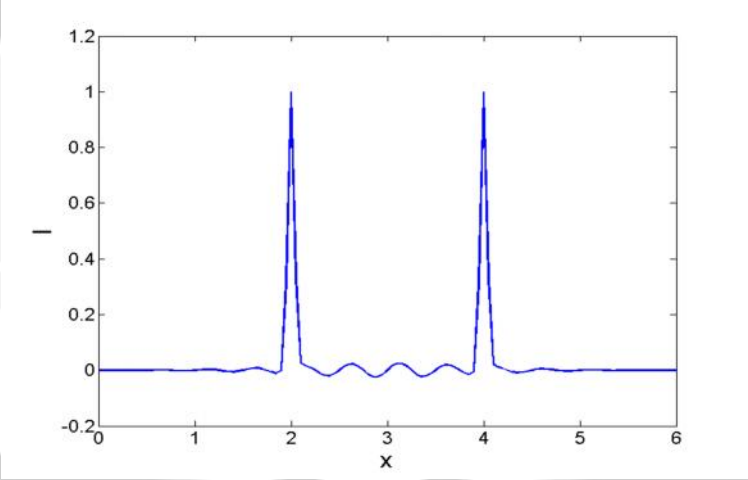
Selain menggunakan turunan parsial pertama, turunan kedua dab Laplacian juga dapat digunakan sebagai edge detector.

$$\frac{\partial^2 I}{\partial x^2} = I_{i+1,j} - 2I_{i,j} + I_{i-1,j}$$
$$\frac{\partial^2 I}{\partial x^2} = I_{i,j+1} - 2I_{i,j} + I_{i,j-1} \quad (4.6)$$

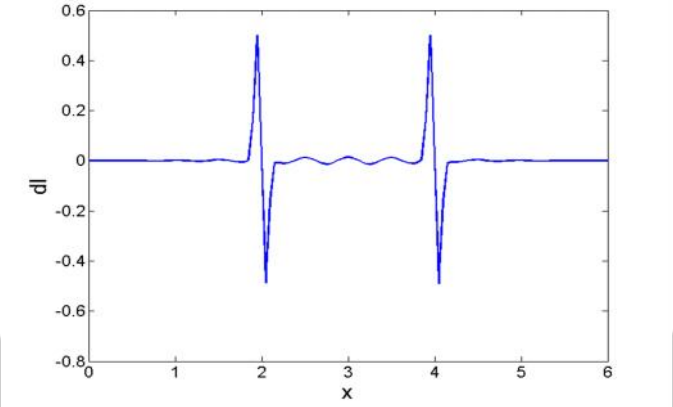
$$\frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial x^2} = I_{i+1,j} + I_{i-1,j} + I_{i-1,j} + I_{i,j+1} + I_{i,j-1} - 4I_{i,j} \quad (4.7)$$

Untuk memperlihatkan kemampuan turunan parsial pertama dan kedua dalam mendeteksi tepi, 3 macam data sinyal 1-dimensi yang mempunyai nilai 2 macam ekstrim pada posisi $x = 2$ dan $x = 4$ diambil sebagai contoh perhitungan:

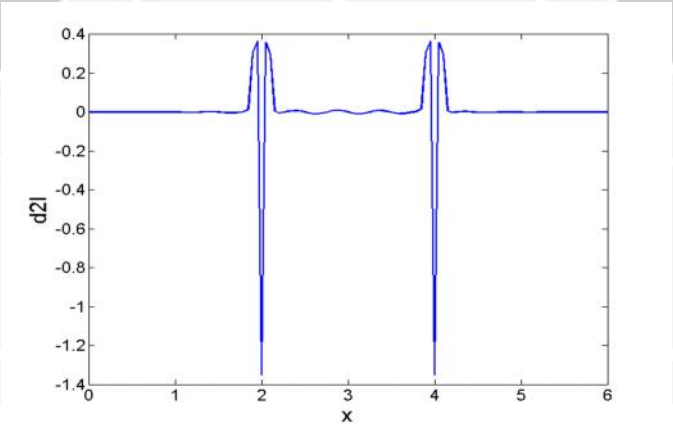
1. Data pertama : sinyal tanpa derau



Gambar 4.1a. Sinyal 1-dimensi tanpa derau

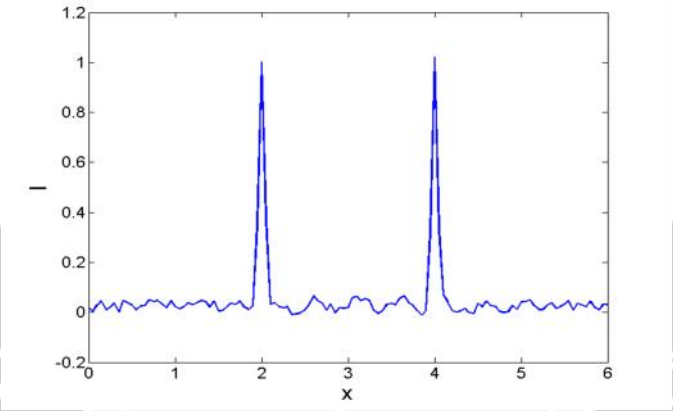


Gambar 4.1b. Turunan pertama sinyal 1-dimensi tanpa derau

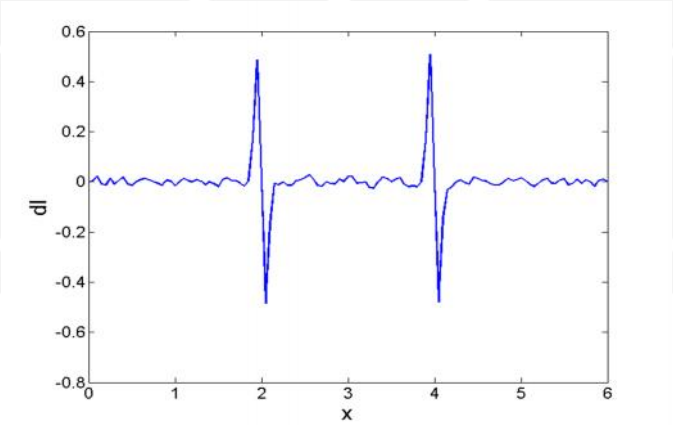


Gambar 4.1c. Turunan kedua sinyal 2-dimensi tanpa derau

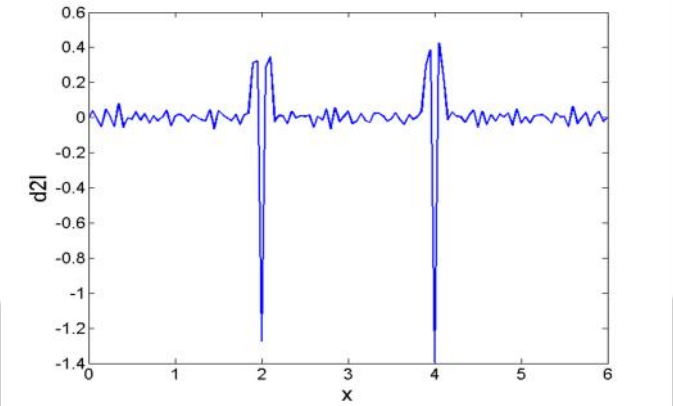
2. Data pertama : sinyal dengan derau ringan



Gambar 4.2a. Sinyal 1-dimensi dengan derau ringan

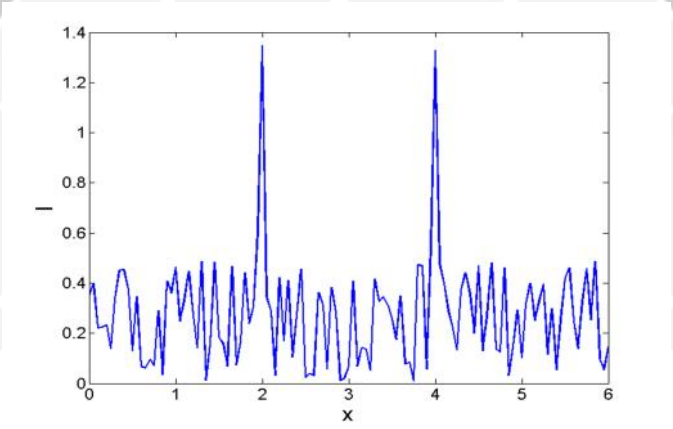


Gambar 4.2b. Turunan pertama sinyal 1-dimensi dengan derau ringan

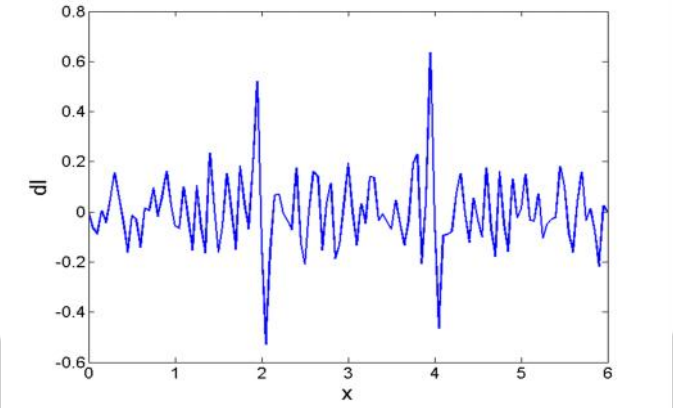


Gambar 4.2c. Turunan kedua sinyal 2-dimensi dengan derau ringan

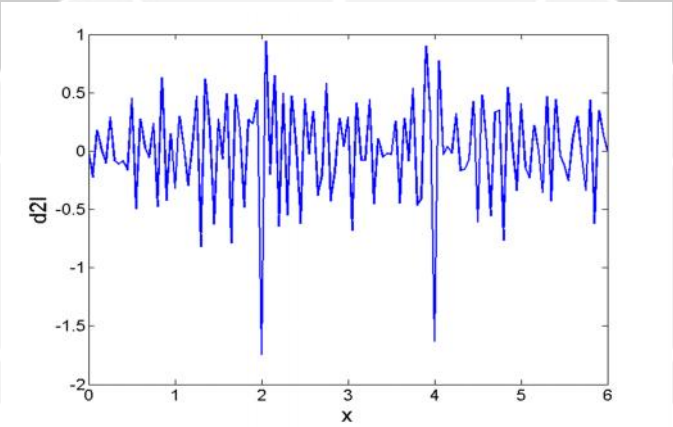
3. Data pertama : sinyal dengan derau berat



Gambar 4.3a. Turunan pertama sinyal 1-dimensi dengan derau berat



Gambar 4.3b. Turunan pertama sinyal 1-dimensi dengan derau berat



Gambar 4.3c. Turunan kedua sinyal 1-dimensi dengan derau berat

Turunan pertama dan kedua mampu mendeteksi keberadaan diskontinyuitas sinyal, tetap jika sinyal terkontaminasi derau maka kemampuan ini menurun tajam. Bahkan jika citra terkontaminasi derau dengan berat maka diskontinyuitas tidak lagi terdeteksi (Gambar 4.3b dan 4.3c). Sehingga untuk mendeteksi citra yang terkontaminasi derau, proses penapisan derau (noise filtering) perlu dilakukan terlrbih dahulu.

Implementasi Program dengan OpenCV

Sebetulnya OpenCV sudah menyediakan fitur untuk perhitungan deteksi tepi, tetapi tujuan buku ini untuk memberikan pemahaman pemrograman algoritma maka fitur tersebut tidak digunakan. Fitur yang digunakan adalah penyimpanan data matriks untuk men. yimpan data citra, pembacaan file citra dan penampilan citra hasil pengolahan. Adapun implementasi algoritma deteksi tepi di atas dengan OpenCV adalah seperti berikut:

```
/**
 * file Smoothing.cpp
 * brief Sample code for simple  filters
 * author OpenCV  team
 */
#include <iostream>
#include <vector>

#include “opencv2/imgproc/imgproc.hpp”
#include “opencv2/highgui/highgui.hpp”
#include “opencv2/features2d/features2d.hpp”

using namespace std;
using namespace cv;

/// Global Variables
int DELAY_CAPTION = 200;
int DELAY_BLUR =  4000;
int MAX_KERNEL_LENGTH =  31;

Mat src; Mat dst; Mat dstx, dsty;
char window_name[] = “Edge  Detection”;

/// Function headers
int display_caption( const char* caption  );
```

```
int display_dst( int delay  );
Mat konvolusi(Mat src, Mat  kernel);


/**
 * function main
 */
int main( void  )
{
 // namedWindow( window_name, WINDOW_AUTOSIZE  );

 /// Load the source  image
 src = imread( "angka.jpg", 2  );

if( display_caption( "Original Image" ) != 0 ) { return 0; }

dst=src.clone();
//dst = Mat::zeros(src.rows,src.cols,CV_32FC1);
if( display_dst( DELAY_BLUR ) != 0 ) { return 0; }

Mat kprewitx,kprewity,ksobelx,ksobely, Laplacian;

kprewitx= (Mat_<double>(3,3) << -1, 0, 1,-1, 0, 1, -1, 0, 1);
kprewity= (Mat_<double>(3,3) << -1,-1, -1, 0, 0, 0, 1, 1, 1);

ksobelx= (Mat_<double>(3,3) << -1, 0, 1, -2, 0, 2,  -1, 0, 1);
ksobely= (Mat_<double>(3,3) << -1, -2, -1, 0, 0, 0,  1, 2, 1);

Laplacian = (Mat_<double>(3,3)  <<0,1,0,1,-4,1,0,1,0);


if( display_caption( "Prewit Edge Detection x" ) !=     0  )  {
return 0; }
dstx= konvolusi(src,kprewitx);
dstx.convertTo(dstx,CV_8UC1);
```

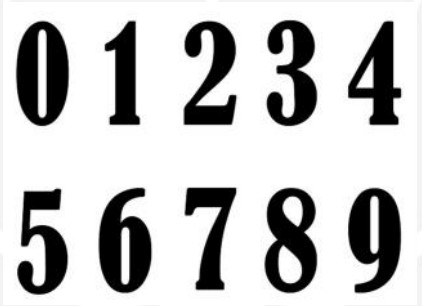
```
imshow( window_name, dstx );
waitKey(DELAY_BLUR);

if( display_caption( “Prewit Edge Detection y” ) != 0 ) {
return 0; }
dsty=konvolusi(src,kprewity);
dsty.convertTo(dsty,CV_8UC1);
imshow( window_name, dsty );
waitKey(DELAY_BLUR);


if( display_caption( “Sobel Edge Detection x” ) != 0 ) {
return 0; }
dstx= konvolusi(src,ksobelx);
dstx.convertTo(dstx,CV_8UC1);
imshow( window_name, dstx );
waitKey(DELAY_BLUR);

if( display_caption( “Sobel Edge Detection y” ) != 0 ) {
return 0; }
dsty= konvolusi(src,ksobely);
dsty.convertTo(dsty,CV_8UC1);
imshow( window_name, dsty );
waitKey(DELAY_BLUR);


if( display_caption( “Laplace Edge Detection “ ) != 0 ) {
return 0; }
dstx= konvolusi(src,Laplacian);
dstx.convertTo(dstx,CV_8UC1);
imshow( window_name, dstx );
waitKey(DELAY_BLUR);
```

```
waitKey(0);

return 0;
}
/**
* @function display_caption
*/
int display_caption( const char* caption  )
{
dst = Mat::zeros( src.size(), src.type() );
putText( dst, caption,
Point( src.cols/8, src.rows/2),
FONT_HERSHEY_COMPLEX, 1, Scalar(255, 255, 255)  );

imshow( window_name, dst  );
int c = waitKey( DELAY_CAPTION );
if( c >= 0 ) { return -1; }
return 0;
}

/**
 * @function display_dst
 */
int display_dst( int delay  )
{
imshow( window_name, dst );
int c = waitKey ( delay );
if( c >= 0 ) { return -1; }
return 0;
}

Mat konvolusi(Mat src, Mat  kernel)
{
       Mat dst;
       float temp=0;
```

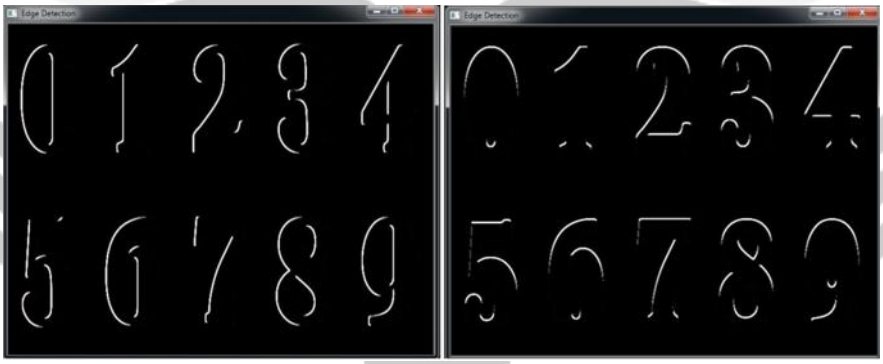
```
        int mm, nn,ii,jj  =0;
        int kCenterX = kernel.cols / 2;
        int kCenterY = kernel.rows /  2;

        dst=Mat::zeros(src.rows,src.cols,CV_32FC1);    //bermain
matrik negatif
        kernel.convertTo(kernel,CV_32FC1); //
        src.convertTo(src,CV_32FC1);
        cout<<kernel<<endl;

for(int i=0; i < src.rows; ++i) // rows
{
 for(int j=0; j < src.cols; ++j) // columns
 {
        for(int m=0; m < kernel.rows; ++m) // kernel rows
 {
 mm = kernel.rows -1 - m; // row index of flipped  kernel

        for(int n=0; n < kernel.cols; ++n) // kernel columns
 {
 nn = kernel.cols -1 - n; // column index of flipped  kernel

 // index of input signal, used for checking boundary
 ii = i + (m - kCenterY);
 jj = j + (n -  kCenterX);

 // ignore input samples which are out of bound
 if( ii >= 0 && ii < src.rows && jj >= 0 && jj < src.cols )
                                {
        dst.at<float>(i,j)  +=  src.at<float>(ii,jj)  *  kernel.
at<float>(mm,nn);
                                }
                        }
 }
             //cout<<dst<<endl;
```

```
                if(dst.at<float>(i,j)>255)
                        dst.at<float>(i,j)=255;
                else if(dst.at<float>(i,j)<0)
                        dst.at<float>(i,j)=0;
        }
}

return dst;
}
```

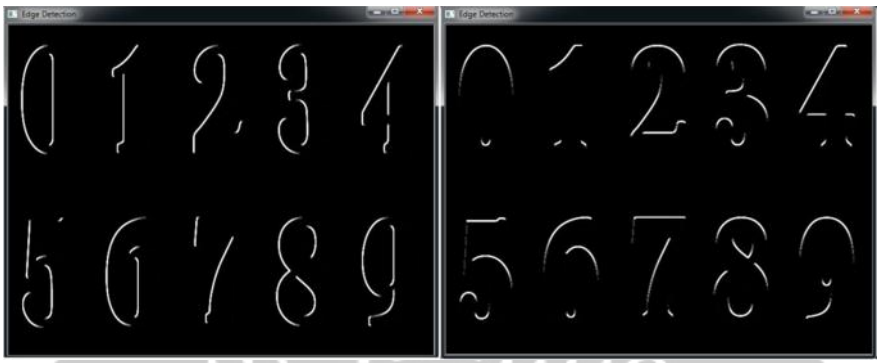
Program di atas dicoba untuk deteksi citra Lena dan citra angka seperti di bawah ini:



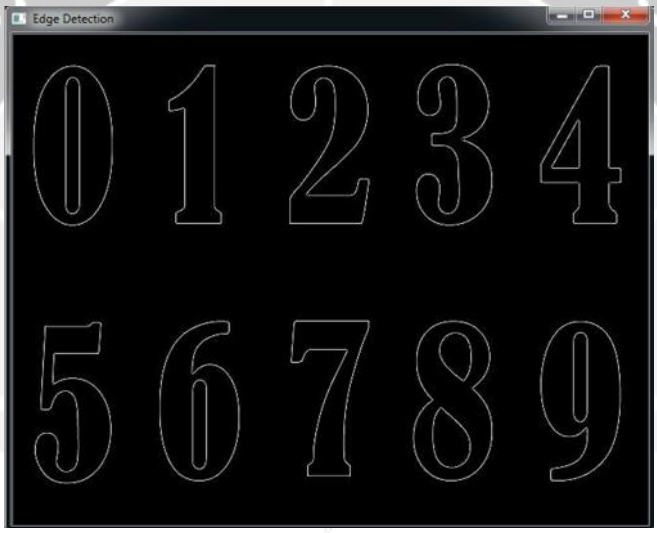
Gambar 4.4. Citra Angka



Gambar 4.5. Deteksi tepi citra Angka dengan mask Sobel turunan -*x* dan *y*

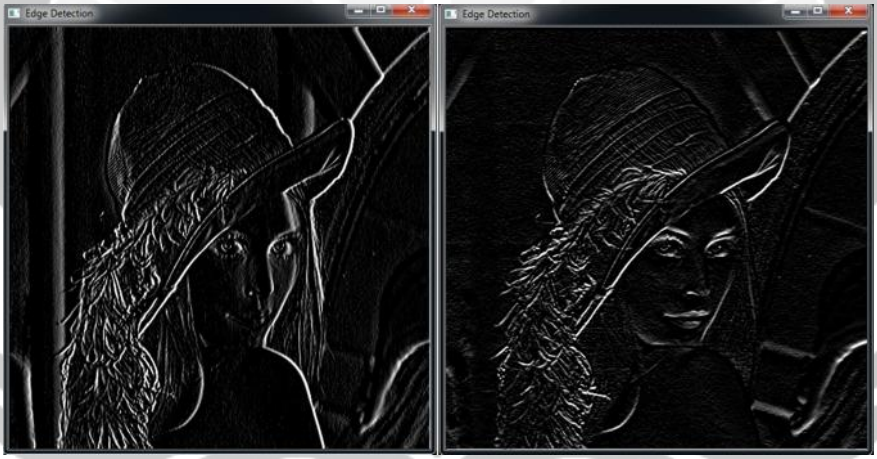


Gambar 4.6. Deteksi tepi citra Angka dengan mask Prewitt turunan -*x* dan *y*

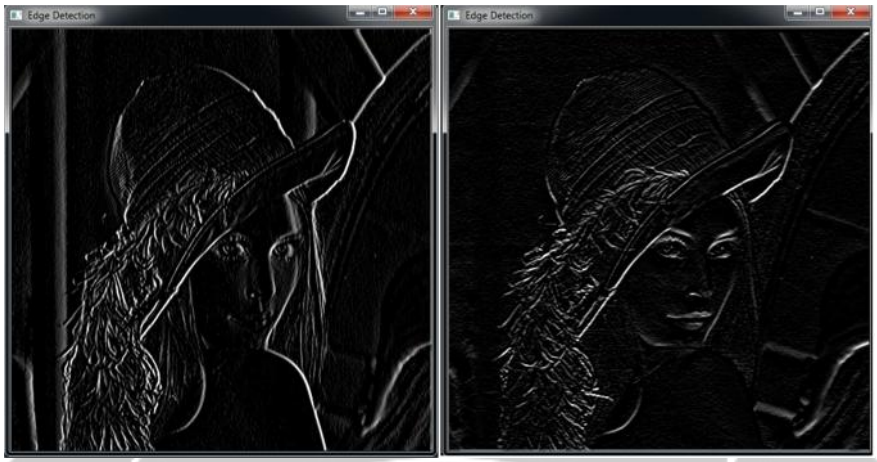


Gambar 4.7. Deteksi tepi citra Angka dengan mask Laplacian

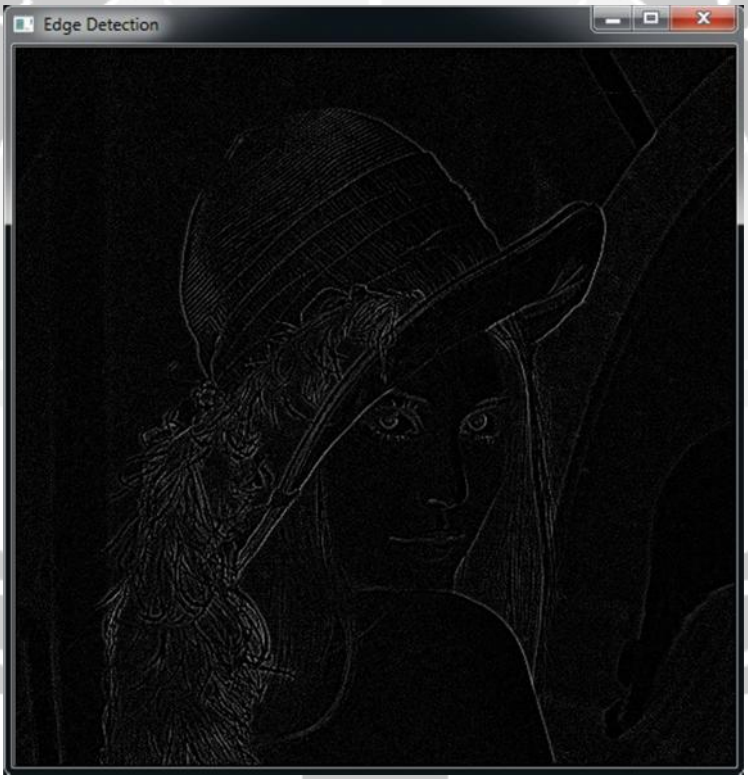
Gambar 4.8. Citra Lena



Gambar 4.9. Deteksi tepi citra Lena dengan mask Sobel turunan -*x* dan *y*



Gambar 4.10. Deteksi tepi citra Lena dengan mask Prewitt turunan -*x* dan *y*



Gambar 4.11. Deteksi tepi citra Lena dengan mask Laplacian

Berdasar dari gambar-gambar hasil perhitungan di atas dapat dilihat bahwa deteksi citra dengan mask Sobel dan Prewitt hanya mendeteksi tepi secara parsial dan performa keduanya boleh dikatakan sama. Sedangkan deteksi tepi dengan mask Laplace memberikan hasil deteksi tepi yang menyeluruh.

serviens in lumine veritatis

BAB V
PENAPISAN DERAU

Teknik Penapisan Derau

Penapisan dearau (*noise filtering*) bertujuan untuk memperbaiki kualitas citra digital. Kualitas citra dapat mengalami penurunan karena terkontaminasi derau saat ditransmisikan lewat saluran kabel atau menggunakan gelombang elektromagnetik. proses ini juga dapat digunakan untuk memperhalus citra (*image smoothing*) dengan tujuan untuk mengurangi gradasi piksel yang tajam sehingga artifak tertentu pada citra dapat dikurangi (Bradski dan Kaehler, 2008). Proses penghalusan ini sering digunakan dalam teknik segmentasi. Bab ini membahas teknik penapisan derau atau penghalusan citra dengan pendekatan PDE, PDE yang digunakan adalah persamaan Heat dan Perona Malik.

Penapisan Derau dengan PDE Parabolik

Penapisan Derau dengan Persamaan Heat

Persamaan heat yang digunakan untuk penapisan derau merupakan PDE linier bertipe parabolik, persamaan heat tersebut setara dengan persamaan diffusi. Derau (*noise*) pada citra didifusikan di sekitar piksel yang dihitung, sehingga penapisan derau dengan persamaan ini menyebabkan citra menjadi kabur (blur). Persamaan pertama yang ditinjau adalah Persamaan Heat yang linier:

$$\frac{\partial I}{\partial t} = k\left(\frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}\right) \tag{5.1}$$

variabel I menyatakan intensitas piksel, t menyatakan waktu semu, sedangkan variabel k menyatakan suatu konstanta. Semakin tinggi nilai t, nilai intensitas citra akan semakin menjadi seragam karena gradien intensitas juga turut didifusikan ke sekitar piksel yang ditinjau sehingga citra akan semakin tambah kabur. Oleh karena itu nilai t tidak boleh tinggi. Proses ini setara dengan proses penapisan Gaussian (Gaussian filtering), yaitu proses konvolusi Kernel Gaussian dengan citra (Perona dan Malik, 1990). Proses penapisan dengan pendekatan PDE. Diskretisasi Beda Hingga untuk persamaan di atas dilakukan seperti berikut:

$$\frac{I^{n+1}_{i,j} - I^{n}_{i,j}}{\Delta t} = \left(I^{n}_{i+1,j} - 2I^{n}_{i,j} + I^{n}_{i-1,j} + I^{n}_{i,j+1} - 2I^{n}_{i,j} + I^{n}_{i,j-1} \right)$$

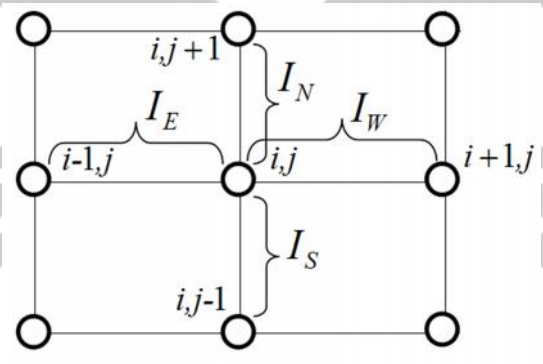
$$I^{n+1}_{i,j} = I^{n}_{i,j} + \Delta t \left((I^{n}_{i+1,j} - I^{n}_{i,j}) + (I^{n}_{i-1,j} - I^{n}_{i,j}) + (I^{n}_{i,j+1} - I^{n}_{i,j}) + (I^{n}_{i,j-1} - I^{n}_{i,j}) \right)$$

$$I^{n+1}_{i,j} = I^{n}_{i,j} + \Delta t (I_W + I_E + I_N + I_S) \qquad (5.2)$$

$$I_N = \left(I^{n}_{i,j+1} - I^{n}_{i,j} \right), \; \left(I_S = I^{n}_{i,j-1} - I^{n}_{i,j} \right)$$

$$I_N = \left(I^{n}_{i,j+1} - I^{n}_{i,j} \right), \; \left(I_S = I^{n}_{i,j-1} - I^{n}_{i,j} \right)$$

Persamaan 5.1 bersifat isotropis, semua gradien (I_W , I_E , I_N , I_S) (Gambar 5.1) di sekitar piksel yang ditinjau akan didifusikan ke sekelilingnya dengan bobot yang sama.



Gambar 5.1. Gradien intensitas piksel.

Penapisan Derau dengan Persamaan Perona Malik

Persamaan Heat linier di atas mempunyai sifat yang sangat difusif, sehingga mudah membuat citra menjadi kabur. Persamaan ini tidak cocok jika citra mengandung pola-pola tekstur tertentu, dimana penapisan derau tidak diperbolehkan ikut juga mengaburkan polaa teksture tersebut. Untuk mengatasi hal tersebut, Perona dan Malik (1990) mengusulkan modifikasi Persamaan Heat dengan menambahkan bobot gradien yang nilainya tergantung pada arah gradien. Persamaan hasil modifikasi ini kemudian dikenal sebagai Persamaan Perona Malik. Persamaan ini bersifat nonlinier dan anisotropik, dengan persamaan ini derau ditapis mengikuti gradien citra, sehingga pola-pola texture tetap dapat dipertahankan. Persamaan Perona Malik ditulis seperti berikut:

$$I_t = \nabla \bullet (c \mathrm{D} I)$$

$$c = c(\nabla I)$$

$$I_{i,j}^{n+1} = I_{i,j}^{n} + \Delta t(c\, I_{W}\, W + c\, I_{E}\, E + c\, I_{N}\, N + c\, I_{S}\, S) \qquad (5.3)$$

Persamaan Perona Malik terdiri dari 2 macam, yaitu Perona Malik pertama dan Perona Malik kedua. Perbedaan di antara kedua macam Perona Malik tersebut terletak pada perhitungan nilai bobot *c*. Perhitungan nilai *c* pada Perona Malik pertama adalah seperti berikut:

$$c_W = \exp\left(-\left(|I_s|/K\right)^2\right) \quad , c_E = \exp\left(-\left(I_s/K\right)\right)$$

$$c_N = \exp\left(-\left(|I_s|/K\right)^2\right) \quad , c_S = \exp\left(-\left(|I_s|/K\right)^2\right) \qquad (5.4a)$$

Sedangkan perhitungan nilai c pada Perona Malik pertama adalah seperti berikut:

$$c_W = \left(1+\left(|I_s|/K\right)^2\right)^{-1} \quad , c\left(= 1+\left(|I|/K\right)^2\right)^{-1}$$

$$c_N = \left(1+\left(|I_S| / K\right)^2\right)^{-1} \quad , c\left(= \quad 1+\left(|I \ | / K\right)^2\right)^{-1} \tag{5.4b}$$

Jika (c_W, c_E, c_N, c_S) bernilai = 1, maka Persamaan Perona Malik sama dengan Persamaan Heat linier.

Implementasi Teknik Penapisan Derau dengan PDE Parabolik

Penapisan derau dengan Persamaan Heat linier dan Perona Malik dapat diimplementasikan dalan satu program komputer, perbedaannya hanya terletak pada pemilihan nilai c . Algoritma Penapisan Derau dengan Persamaan PDE Parabolik adalah seperti berikut:

1. Baca data citra
2. Tambahkan derau pada data citra.
3. Konversi tipe data citra menjadi tipe *double* atau *real*, tentukan nilai parameter perhitungan (Δt, k).
4. Langkah waktu perhitungan dimulai
   - Hitung gradien citra.
   - Update nilai I dengan Persamaan 5.3
5. Cek batas waktu maksimum. Jika batas belum terlampaui, kembali langkah 5. Jika sudah terlampaui, lanjutkan ke langkah 7.
6. Tulis data
7. Selesai

Implementasi algoritma di atas ke dalam program adalah seperti berikut:

```
#include <iostream>
#include <vector>

#include “opencv2/imgproc/imgproc.hpp”
#include “opencv2/highgui/highgui.hpp”
#include “opencv2/features2d/features2d.hpp”
```

```
using namespace std;
using namespace cv;

/// Global Variables


// pilih method = ‘0’ : Persamaan Heat linear
// method = ‘1’ : Persamaan Perona Malik tipe 1
// method = ‘2’ : Persamaan Perona Malik tipe 2

char method  =’2’;
double noise_percentage;
double average;
double standard_deviation;

//Const Parameter
const double K1=2.0; //threshold
const double dt=0.05;
const int iteration=40;

const Mat kerneln = (Mat_<float>(3,1)<<0, -1, 1);
const Mat kernels = (Mat_<float>(3,1)<<1, -1, 0);
const Mat kernelw = (Mat_<float>(1,3)<<0, -1, 1);
const Mat kernele = (Mat_<float>(1,3)<<1, -1, 0);

Mat src; Mat  dst;
 char window_name[] = “Noise Filtering  -”;

Mat Laplacian(Mat src, Mat  kernel);
void    addSaltAndPepperNoise(Mat    &img,    double    noise_
percentage);
void   addGaussianNoise(Mat   &image,   double   average,double
standard_deviation);
/**
 * function main
```

```
 */
int main( void  )
{

       noise_percentage =  10.0;
       average = 0.0;
       standard_deviation = 0.0;
 namedWindow( window_name, WINDOW_AUTOSIZE  );
 /// Load the source  image
 src = imread( "Mami.png", CV_LOAD_IMAGE_GRAYSCALE);
 imshow("Source ",src);
 imwrite("Mami_source.png", src);

 dst=src.clone();
 addSaltAndPepperNoise(dst, noise_percentage);


   //   dst   =   imread(   "Mami1_noisy.png",   CV_LOAD_IMAGE_
GRAYSCALE);
 imshow("Noisy image ",dst);
 imwrite("Mami_Noisy_image.png", dst);

 if(method=='0')
       cout<<"Heat linear"<<endl;

 if(method=='1')
 cout<<"Perona Malik tipe  1"<<endl;


 if(method=='2')
 cout<<"Perona Malik tipe  2"<<endl;

// Mat CC = Mat::ones(src.rows,src.cols, CV_64FC1);

 for (int i = 0; i < iteration; i++)
```

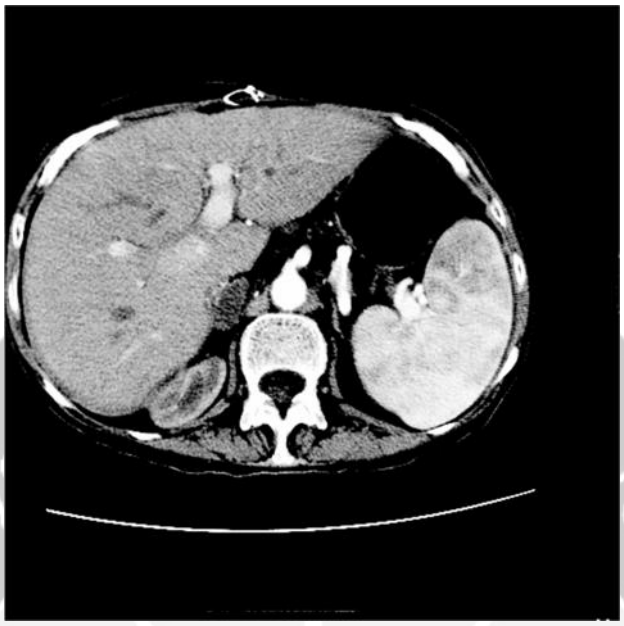
```
{
       cout<<"Iterasi ke = "<<i+1<<endl;
      dst.convertTo(dst,CV_64FC1);
      Mat  In=Laplacian(dst,kerneln);
      Mat  Is=Laplacian(dst,kernels);
      Mat  Iw=Laplacian(dst,kernelw);
      Mat Ie=Laplacian(dst,kernele);


            Mat Cn,Cs,Cw,Ce;

      if(method=='0')
       {
            Cn = Mat::ones(src.rows,src.cols, CV_64FC1)*K1;
            Cs = Mat::ones(src.rows,src.cols,  CV_64FC1)*K1;
       Cw = Mat::ones(src.rows,src.cols,  CV_64FC1)*K1;
            Ce = Mat::ones(src.rows,src.cols,  CV_64FC1)*K1;
       }

      if (method=='1')
       {
             Mat In1 = abs(In)/K1; Mat Is1 = abs(Is)/K1;
             Mat Iw1 = abs(Iw)/K1; Mat Ie1 = abs(Ie)/K1;

       exp(-In1.mul(In1),Cn);
            exp(-Is1.mul(Is1),Cs);
       exp(-Iw1.mul(Iw1),Cw);
            exp(-Ie1.mul(Ie1),Ce);
       }

      if (method=='2')      {

            Mat In1 = abs(In)/K1; Mat Is1 = abs(Is)/K1;
            Mat Iw1 = abs(Iw)/K1; Mat Ie1 = abs(Ie)/K1;
```

```
                divide(1.0,In1.mul(In1),Cn);
                divide(1.0,Is1.mul(In1),Cs);
                divide(1.0,Iw1.mul(In1),Cw);
                divide(1.0,Ie1.mul(In1),Ce);
        }


                dst=dst+dt*(Cn.mul(In)+Cs.mul(Is)+Cw.mul(Iw)+Ce.
mul(Ie));
                dst=dst+dt*(In+Is+Iw+Ie);
                dst.convertTo(dst, CV_8UC1);

                imshow(window_name,dst);
                 waitKey(1);

        }
 if(method==’0’)
 imwrite(“Mami_filtered_heat.png”, dst);

 if(method==’1’)
 imwrite(“Mami_filtered_PM1.png”,  dst);

 if(method==’2’)
 imwrite(“Mami_filtered_PM2.png”,  dst);

 waitKey(10);

 return 0;
}

Mat Laplacian(Mat src, Mat  kernel)
{
       Mat dst;
       float temp=0;
       int mm, nn,ii,jj  =0;
```

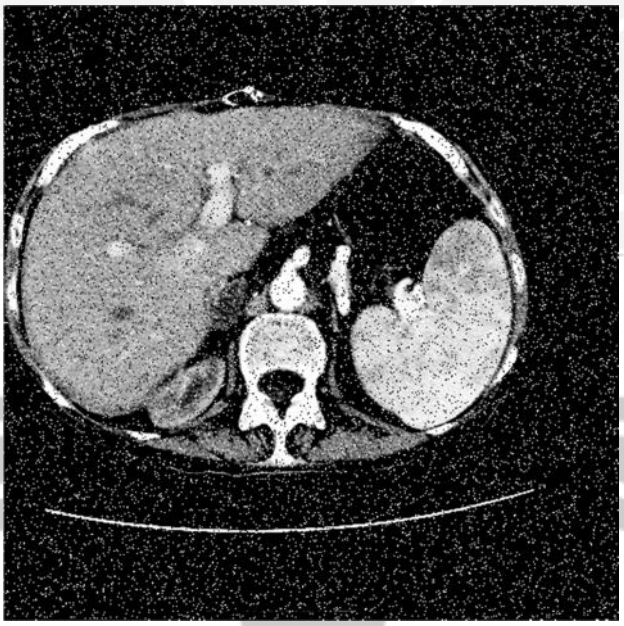
```
        int kCenterX = kernel.cols / 2;
        int kCenterY = kernel.rows /  2;

        dst=Mat::zeros(src.rows,src.cols,CV_64FC1);    //bermain
matrik negatif
        kernel.convertTo(kernel,CV_64FC1); //
        src.convertTo(src,CV_64FC1);
        //cout<<" "<<kernel<<endl; //show matrix  kernel

for(int i=0; i < src.rows; ++i) // rows
{
 for(int j=0; j < src.cols; ++j) // columns
 {
                for(int m=0; m < kernel.rows; ++m) // kernel
rows
 {
 mm = kernel.rows -1 - m; // row index of flipped kernel

                    for(int n=0; n < kernel.cols; ++n) //
kernel columns
 {
 nn = kernel.cols -1 - n; // column index of flipped kernel

 // index of input signal, used for checking boundary
 ii = i + (m - kCenterY);
 jj = j + (n -  kCenterX);

 // ignore input samples which are out of bound
 if( ii >= 0 && ii < src.rows && jj >= 0 && jj < src.cols )
                            {
                                    dst.at<double>(i,j) += src.
at<double>(ii,jj) * kernel.at<double>(mm,nn);
                            }
                    }
 }
```

```
        }
}
}
//cout<<dst<<endl;
return dst;
}

void    addSaltAndPepperNoise(Mat    &image,    double    noise_
percentage)
{

// diambil dari code buku: Howe, K.D. (20140),
//     A  Practical  Introduction  to  Computer  Vision  with
OpenCV,
// John Wiley and Sons Ltd, Chicester, United Kingdom.

int image_rows = image.rows;
int image_columns =  image.cols;
int image_channels =  image.channels();
int noise_points = (int) (((double) image_rows*
image_columns*image_channels)*noise_percentage/100.0);
for (int count = 0; count < noise_points; count++)
{
int row = rand() %  image_rows;
int column = rand() % image_columns;
int channel = rand() % image_channels;
uchar* pixel = image.ptr<uchar>(row) +
(column*image_channels) +  channel;
*pixel = (rand()%2 == 1) ? 255 : 0;
}
}
```

Hasil penapisan derau dengan menggunakan persamaan diffusi linier untuk citra yang terkontaminasi derau *salt and paper* 10 % adalah seperti berikut:



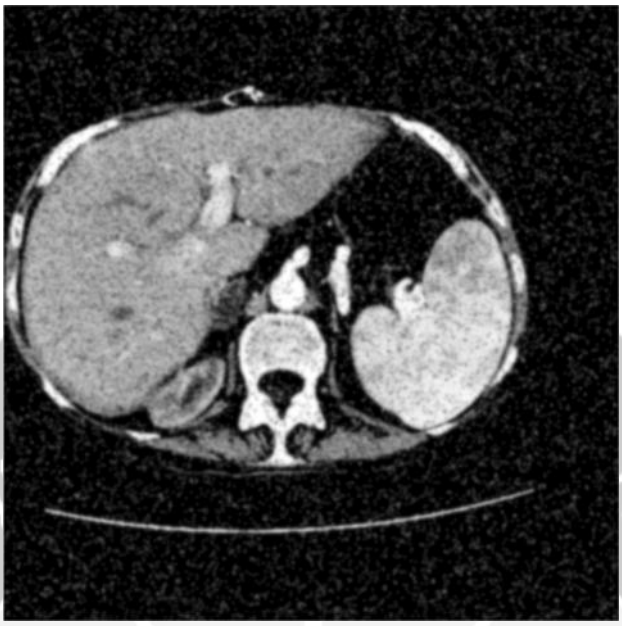
Gambar 5.2a. Citra asli ct_scan.bmp



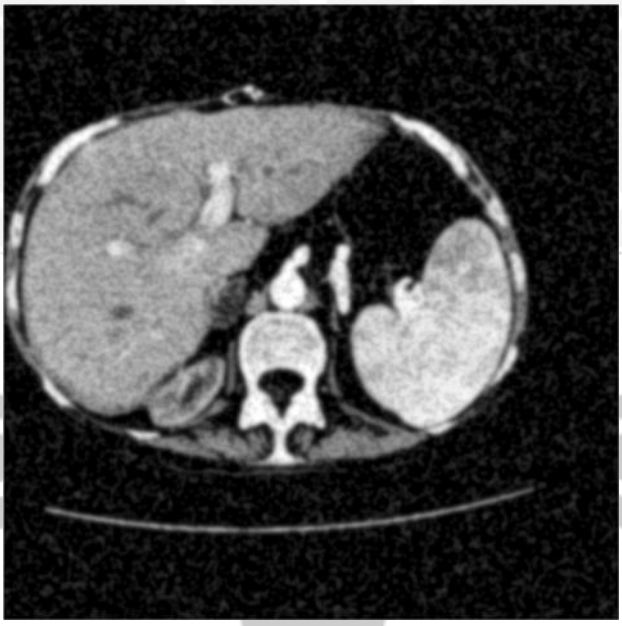
Gambar 5.2b. Citra ct_scan.bmp yang terkontaminasi derau

Gambar 5.3a. Hasil penapisan dengan Persamaan Heat iterasi = 20

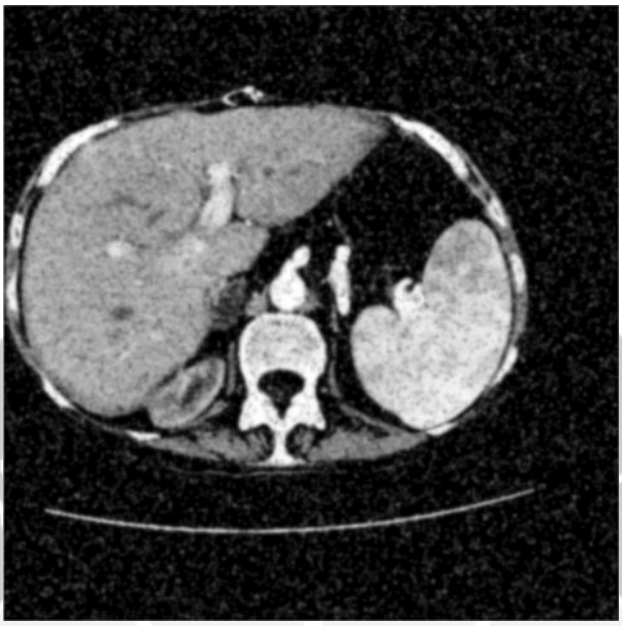
Gambar 5.3b. Hasil penapisan dengan Persamaan Heat iterasi = 40



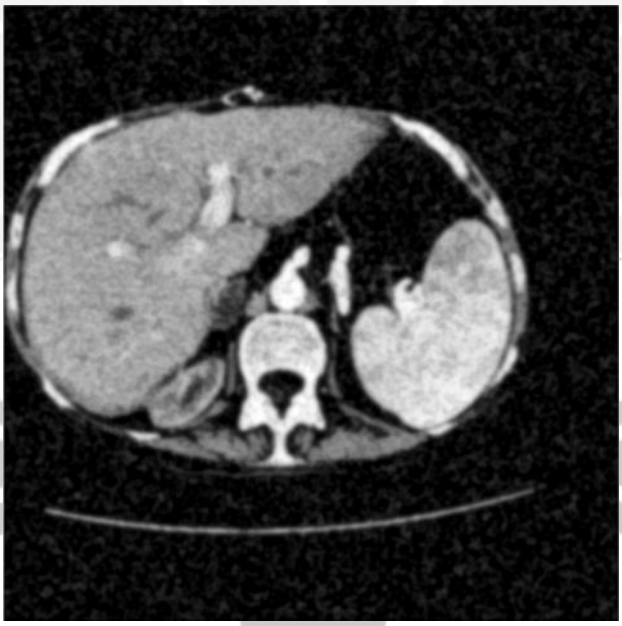
Gambar 5.4a. Hasil penapisan dengan Persamaan Perona Malik I iterasi = 20



Gambar 5.4b. Hasil penapisan dengan Persamaan Perona Malik I iterasi = 40

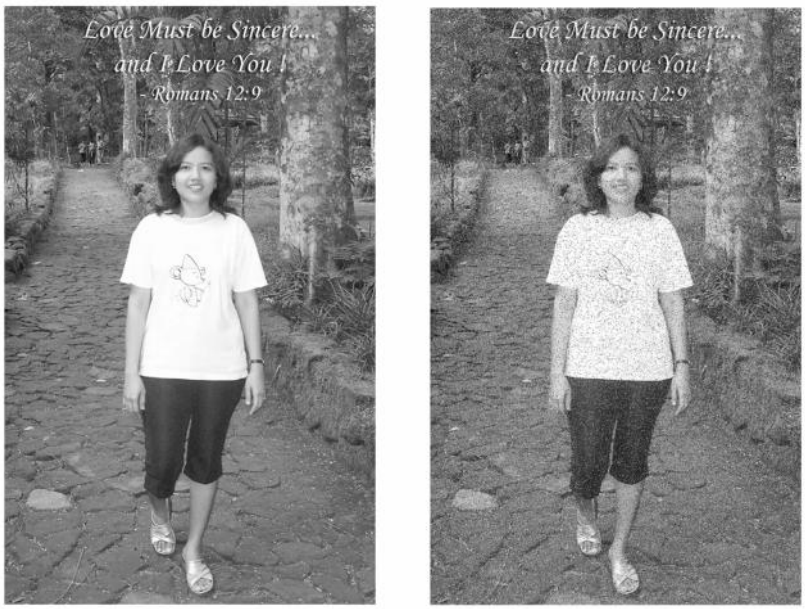


Gambar 5.4a. Hasil penapisan dengan Persamaan Perona Malik II iterasi = 20

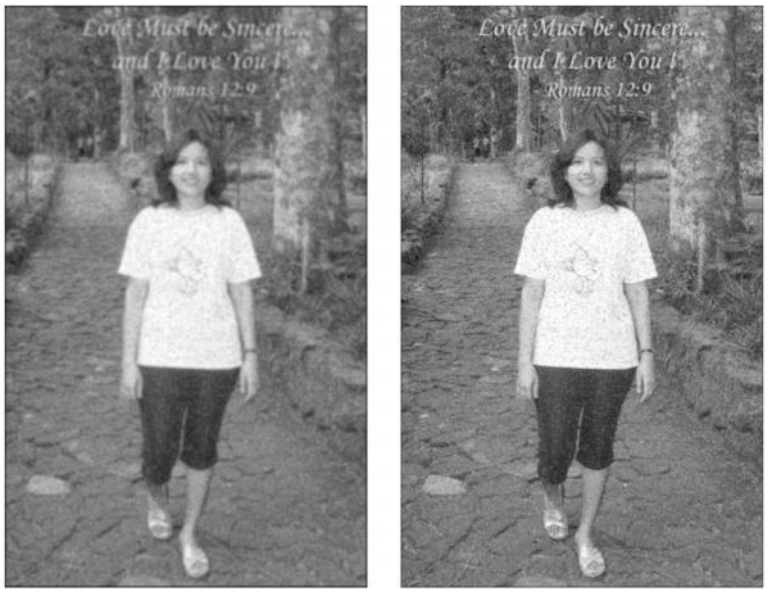


Gambar 5.4b. Hasil penapisan dengan Persamaan Perona Malik II iterasi = 40

Berdasarkan dari hasil perhitungan di atas, terlihat jelas bahwa Persamaan Heat membuat derau hilang tetapi efek lainnya adalah gambar semakin kabur. Sedangkan Persamaan Perona Malik berhasil menghilangkan derau dengan tetap mempertahankan bentuk pola yang ada dalam citra. Performa antara Perona Malik I dan II boleh dikatakan sama , hal ini diperjelas dengan Gambar 5.6 dan 5.7.



Gambar 5.5. Citra asli wanita.png (kiri) dan citra yang terkontaminasi derau (kanan).



Gambar 5.6. Citra hasil penapisan derau dengan Persamaan Heat (kiri) dan Persamaan Perona Malik I (kanan)

Gambar 5.7. Citra hasil penapisan derau dengan Persamaan Perona Malik II

BAB VI
SEGMENTASI CITRA

Metode Segmentasi

Segmentasi merupakan proses pembagian citra digital menjadi beberapa bagian atau objek. Segmentasi merupakan salah satu metode pengolahan citra yang penting karena mendasari beberapa proses pengolahan citra tingkat lanjut, seperti: pengenalan pola dan klasifikasi citra (Solomon & Breckon, 2010). Tujuan segmentasi adalah memisahkan daerah atau objek tertentu dalam citra. Daerah citra atau objek yang sudah disegmentasi disebut latar depan (*foreground*) sedangkan sisa citra disebut latar belakang (*background*).

Pendekatan metode segementasi pada dasarnya dikelompokkan menjadi 2, yaitu:

- Metode berdasar pencarian tepi atau batas

  Pendekatan ini mencari tepi citra untuk menentukan batas suatu objek di dalam citra. Metode ini mirip dengan metode deteksi tepi hanya bedanya tepi segmentasi berupa kurva yang berkesinambungan.

- Metode berdasar pencarian daerah

  Metode ini menandai piksel-piksel yang mempunyai kemiripan nilai, warna atau pola tekstur sehingga membentuk suatu daerah.

Metode segmentasi yang mengimplementasikan pendekatan di atas di antaranya adalah metode *thresholding*, *clustering*), *color*

image-based, active contour-model (Zhang dkkl, 2009; 2010) dan lain-lain.

Buku ini membahas segmentasi yang dikembangkan oleh Zhang dkk (2009, 2010) dengan metode active contour yang merupakan metode segmentasi berdasar pencarian tepi dan kemudian dilanjut dengan metode level set yang merupakan metode berdasar pencarian daerah.

Active Contours

Kass et al (1987) menyatakan a*ctive contour* merupakan suatu batas objek atau beberapa fitur gambar lainnya sebagai kurva *parametric* (Kass et al, 1987). Karena kemampuannya ini, *Active contour* dapat digunakan sebagai segmentasi citra. Untuk meningkatkan akurasi segmentasi, beberapa peneliti melakukan penelitian dengan menggabungkan *active contour model* dan *level set method*. Kombinasi metode *level set* dan *active contour model* dapat diimplementasikan pada berbagai macam citra.

Metode Active Contours Berbasis Image Laplacian Fitting

Zhang et al (2010) mengkaji masalah citra yang memiliki intensitas inhomogen dengan kombinasi fungsi *level set method* dengan *active contour. Active contour* pada awalnya merupakan metode segmentasi berbasis tepi. Detektor tepi yang biasa digunakan menggunakan fungsi seperti berikut:

$$g\left(\left|\nabla I\right|\right) = \frac{1}{1+\left|\Delta G * I\right|^{p}} \quad ; p > 1 \qquad (6.1)$$

$\nabla G*I$ adalah konvolusi citra dengan filter Gaussian dan bertujuan untuk mengurangi noise atau menghaluskan citra. Setelah tepi citra diperoleh, selanjutnya tepi citra akan dihubungkan dengan menggunakan metode level set, sehingga membentuk suatu kurva yang membatasi daerah-daerah segmentasi citra. Level set sendiri adalah suatu fungsi jarak dan dikembangkan oleh Profesor Osher

dari UCLA (Osher dan Paragios,2003). Kurva yang dihitung dengan metode level set dapat bertranslasi, mengembang, menyusut dan dapat mengikuti kontur tepi citra. Zhanh dkk (2009) menyebut metode yang dikembangkan tersebut adalah metode *Image Laplacian Fitting Energy* atau disingkat metode ILF.

Metode ILF mempunyai 2 langkah utama perhitungan, langkah pertama adalah menghitung aliran gradien (gradient flow) yang bertujuan untuk menghaluskan citra (smoothing) tetapi tetap mempertahankan tepi citra. Persamaan yang digunakan langkah pertama adalah:

$$\frac{\partial u}{\partial t} = \alpha\left(\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) - \left(u - \left(\frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}\right)\right)\right) \tag{6.2a}$$

Variabel u adalah aliran gradien dan nilainya adalah nol pada awal perhitungan, α adalah suatu konstanta. Diskretisasi persaman (6.1) dilakukan seperti berikut:

$$u^{n+1} \quad A\frac{\partial^2 I}{\partial x_2} + B\frac{\partial^2 I}{\partial x \partial y} C\frac{\partial^2 I}{\partial y^2} + D\frac{\partial I}{\partial x} + E\frac{\partial I}{\partial y} + FI = G$$

$$\Delta t\left(u^n_{i,j} - (I^n_{i+1,j} + I^n_{i-1,j} + I^n_{i,j+1} + I^n_{i,j-1} - 4I^n_{i,j})\right) \tag{6.2b}$$

Langkah kedua adalah menghitung evolusi fungsi level set kontour (ϕ) dengan persamaan berikut:

$$\frac{\partial \phi}{\partial t} = \mu\delta_z \nabla \circ \left(\frac{\nabla \phi}{|\nabla \phi|}\right) + \lambda u \delta_z + v\left(\Delta\phi - \nabla \circ \left(\frac{\nabla \phi}{|\nabla \phi|}\right)\right) \tag{6.3a}$$

$$\frac{\partial \phi}{\partial t} = G\left(\phi^n_{i,j}\right)$$

$$\phi^{n+1}_{i,j} = \phi^n_{i,j} + \Delta t G(\phi^n_{i,j}) \tag{6.3b}$$

$$G(\phi^n_{i,j}) = \mu\delta_z \nabla \circ \left(\frac{\nabla \phi}{|\nabla \phi|}\right) + \lambda u \delta_z + v\left(\Delta\phi - \nabla \circ \left(\frac{\nabla \phi}{|\nabla \phi|}\right)\right)$$

Laplacian : $$\Delta\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} \quad (6.4a)$$

Gradien : $$\nabla\phi = \left[\frac{\partial\phi}{\partial x} \quad \frac{\partial\phi}{\partial y}\right]^T \quad (6.4b)$$

Panjang gradien : $$|\nabla\phi| = \left(\left(\frac{\partial\phi}{\partial x}\right)^2 + \left(\frac{\partial\phi}{\partial y}\right)^2\right)^{0.5} \quad (6.4c)$$

Kelengkungan : $$\nabla \circ \left(\frac{\nabla\phi}{|\nabla\phi|}\right) = \frac{\partial}{\partial x}\left(\frac{\partial\phi}{\partial x} \Big/ |\nabla\phi|\right) + \frac{\partial}{\partial y}\left(\frac{\partial\phi}{\partial y} \Big/ |\nabla\phi|\right) \quad (6.4d)$$

Dirac function : $$\delta_\varepsilon = \frac{1}{\pi}\frac{\varepsilon}{\phi^2 + \varepsilon^2} \quad (6.4e)$$

Dirac fungction δ_ε menyatakan ketebalan kurva level set, jika nilai ε semakin tinggi maka ketebalan kurva meningkat dan menyebabkan perhitungan tepi segmentasi semakin mudah. Tetapi efek buruknya adalah akurasi menjadi turun. Turunan-turunan parsial pada persamaan di atas dihitung dengan menggunakan metode Beda Hingga.

Algoritma *Active Contours* berdasarkan *Image Laplacian Fitting* adalah seperti berikut:

1. Baca data citra
2. Konversi tipe data citra menjadi tipe *double* atau *real*, tentukan nilai parameter perhitungan.
3. Inisialisasi nilai level set (ϕ).
4. Hitung aliran gradien *u* (Persamaan 6.2) sampai konvergen.
5. Langkah waktu perhitungan level set (ϕ) dimulai
   - Hitung gradien Persamaan
   - Panjang gradien
   - Kelengkungan
   - Laplacian
   - Dirac function
   - update nilai level set (ϕ) (Persamaan 6.3b)

6. Cek batas waktu maksimum. Jika batas belum terlampaui, kembali langkah 5. Jika sudah terlampaui, lanjutkan ke langkah 7.
7. Tulis data
8. Selesai

Implementasi algoritma di atas dengan OpenCV adalah seperti berikut:

```
#include <iostream>
#include <vector>

#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/features2d/features2d.hpp"

using namespace std;
using namespace cv;

/// Global Variables
int DELAY_VIEW = 2000;

//Const Parameter
const double pi= 3.14;
//const float sigma_phi = 0.5;
//const float sigma = 10;
const double epsilon = 1.5;
const double timestep = 0.5;
const double mu = 0.2/timestep;
const double lambda = 8;
const double mmu = 3.5;
const double delt = 0.02;
const int Iter_num = 100;
const Mat kernelx = (Mat_<float>(1,3)<<0.5, 0, -0.5);
const Mat kernely = (Mat_<float>(3,1)<<0.5, 0, -0.5);
```

```
const Mat kernelLaplace = (Mat_<double>(3,3)<<1, 0, 1, 0,
-4, 0, 1, 0,  1);

Mat src; Mat dst; Mat dstx, dsty;
char window_name[] = "Active countour - Image Laplacian
Fitting Energy -";

/// Function headers
int display_dst( int delay  );
Mat konvolusi(Mat src, Mat kernel);
Mat divergence(Mat,double,Mat,double);
Mat Evo_ILF(Mat,float,float,float,float, Mat);
Mat TV_Laplace(Mat u, Mat phi,float mu,float timestep, int
iter_max);
void drawcircle(int,int,int,int,int);

/**
 * function main
 */
int main( void  )
{
 Mat fx,fy,MLaplace,phi,u;
 vector<cv::Point> mcontour;
 int icenter=0,jcenter=0;
 int radius=20;
 namedWindow( window_name, WINDOW_AUTOSIZE  );
 /// Load the source  image
 src = imread( "ct_scan.bmp", CV_LOAD_IMAGE_GRAYSCALE );
 dst=src.clone();
 dst.convertTo(dst,CV_64FC1);
   (Mat_<double>(4,4)<<1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,1
6);

 MLaplace =konvolusi(dst,kernelLaplace);
 u = Mat::zeros(src.rows,src.cols,  CV_64FC1);
```

```
 u = TV_Laplace(u, MLaplace,mmu,delt,100);
 cv::Rect sub1ROI(30,30,40,40);

 rectangle(dst,sub1ROI,Scalar(0,0,255),1,8);

 phi= Mat::zeros(src.rows,src.cols,CV_64FC1);
 for (int i = radius; i < phi.rows-radius; i++)
 {
        for (int j = radius; j < phi.cols-radius; j++)
        {
               phi.at<double>(i,j)=(double)-2;
        }
 }

 for (int i = 0; i < 16; i++)
 {
                phi=Evo_ILF(phi,mu,lambda,epsilon,timestep,
MLaplace);
        imshow(window_name,phi);
        cout<<"Iterasi ke-"<<i+1<<endl;
        waitKey(1000);
 }
 waitKey(0);

 return 0;
}

int display_dst( int delay  )
{
 imshow( window_name, dst );
 int c = waitKey ( delay );
 if( c >= 0 ) { return -1; }
 return 0;
}
```

```
Mat konvolusi(Mat src, Mat  kernel)
{
       Mat dst;
       float temp=0;
       int mm, nn,ii,jj  =0;
       int kCenterX = kernel.cols / 2;
       int kCenterY = kernel.rows /  2;

       dst=Mat::zeros(src.rows,src.cols,CV_64FC1); //bermain
matrik negatif
       kernel.convertTo(kernel,CV_64FC1); //
       src.convertTo(src,CV_64FC1);
       //cout<<" "<<kernel<<endl; //show matrix  kernel

for(int i=0; i < src.rows; ++i) // rows
{
 for(int j=0; j < src.cols; ++j) // columns
 {
             for(int m=0; m < kernel.rows; ++m) // kernel
rows
 {
 mm = kernel.rows -1 - m; // row index of flipped kernel

             for(int n=0; n < kernel.cols; ++n) // kernel
columns
 {
 nn = kernel.cols -1 - n; // column index of flipped kernel

 // index of input signal, used for checking boundary
 ii = i + (m - kCenterY);
 jj = j + (n -  kCenterX);

 // ignore input samples which are out of bound
 if( ii >= 0 && ii < src.rows && jj >= 0 && jj < src.cols
)
```

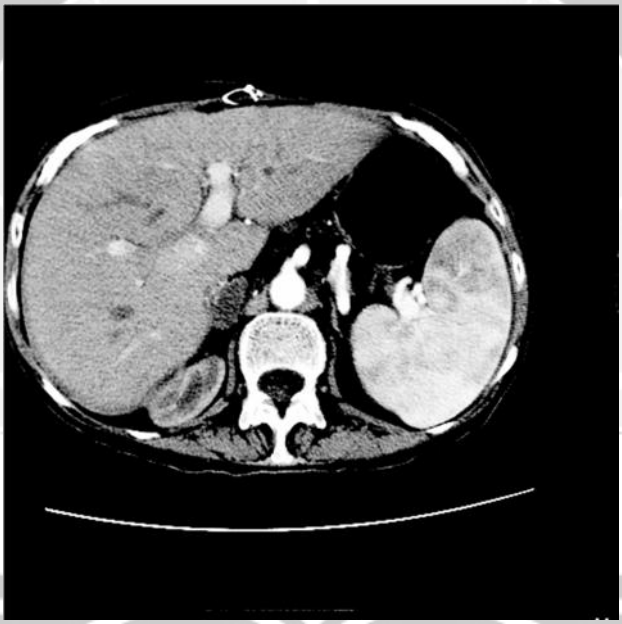
```
                        {
                                dst.at<double>(i,j) += src.
at<double>(ii,jj) * kernel.at<double>(mm,nn);
                        }
                    }
 }

      }
}
//cout<<dst<<endl;
return dst;
}

Mat Evo_ILF(Mat phi,float mu,float lambda,float epsilon,float
timestep, Mat Lap)
{
      Mat delta;
      d i v i d e ( 1 / p i * e p s i l o n , p h i .
mul(phi)+pow(epsilon,2),delta);

      Mat phi_x=konvolusi(phi,kernelx); //gradient-x
      Mat phi_y=konvolusi(phi,kernely); //gradient-y

      //panjang gradient

      Mat sqrt_abs_phi;
      sqrt(phi_x.mul(phi_x)+phi_y.mul(phi_y),sqrt_abs_phi);

      sqrt_abs_phi=sqrt_abs_phi+std::numeric_
limits<double>::epsilon();

      //kelengkungan

      Mat P,Q;
      divide(phi_x,sqrt_abs_phi,P);
```

```
        divide(phi_y,sqrt_abs_phi,Q);


        Mat kappa =  divergence(P,1,Q,1);
        Mat phi1 = mu*konvolusi(phi,kernelLaplace)-kappa;
        Mat phi2 =  lambda*delta.mul(kappa);
        Mat phi3 =  Lap;
        return phi+(phi1+phi2+phi3)*timestep; //update phi

}


Mat TV_Laplace(Mat u, Mat phi,float mu,float timestep, int
iter_max)
{
         Mat u1, u2;

         for (int i = 0; i < iter_max; i++) {
         u1 = konvolusi(u,kernelLaplace);
         u2 = -(u -  phi);
         u = u +  timestep*(u1+u2);
         cout<<"Iterasi V_Laplace ke :  "<<i+1<<endl;
         }
        return u;
}


Mat divergence(Mat P, double dx, Mat Q,double dy)
{
        Mat mask=Mat::zeros(P.rows,P.cols,CV_64FC1);

        P= konvolusi(P,kernelx); //gradien x
        Q= konvolusi(Q, kernely); //gradien  y
```

```
        mask.col(0).copyTo(P.col(0));
        mask.col(P.cols-1).copyTo(P.col(P.cols-1));

        mask.row(0).copyTo(Q.row(0));
        mask.row(Q.rows-1).copyTo(Q.row(Q.rows-1));

        return P+Q;
}
```

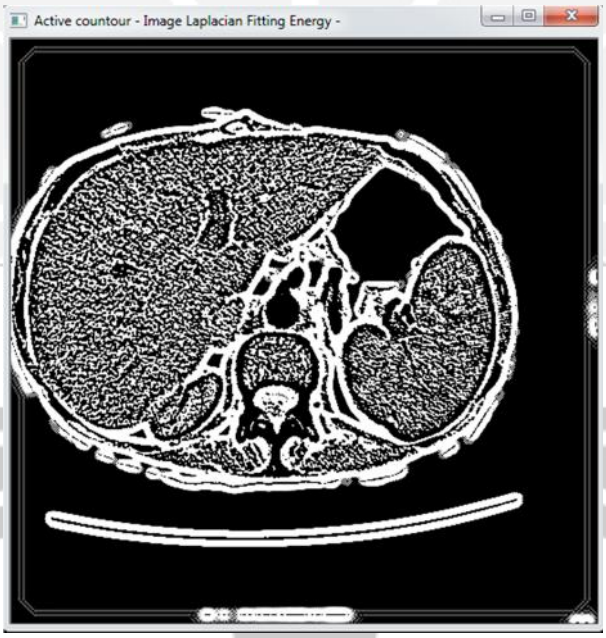
File citra yang digunakan untuk uji coba adalah citra ct_scan.bmp dan angka.jpg Hasil perhitungannya adalah seperti berikut:

- Perhitungan kontur citra ct_scan.bmp

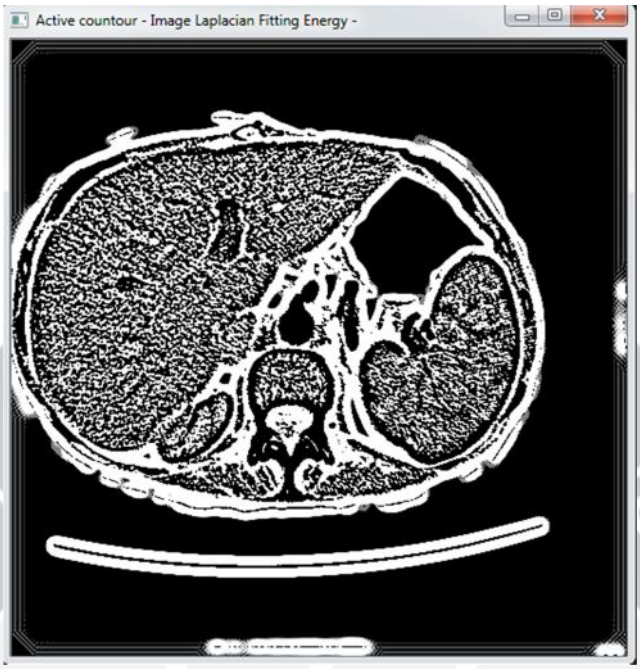


Gambar 6.1. Citra ct_scan.bmp

Gambar 6.2a. Kontur Citra ct_scan.bmp metode ILF iterasi ke-4

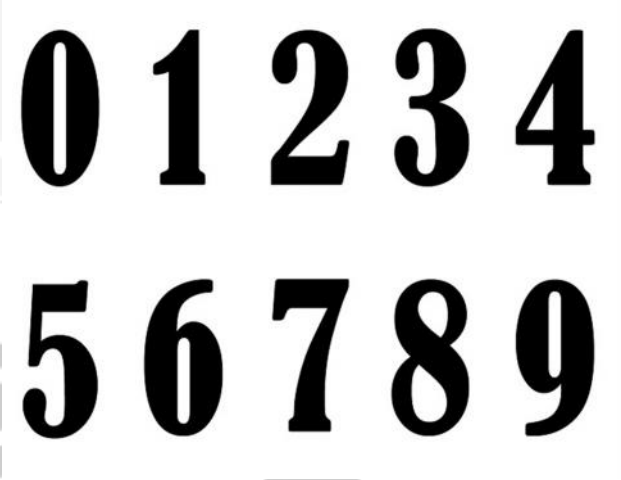


Gambar 6.2b. Kontur Citra ct_scan.bmp metode ILF iterasi ke-8

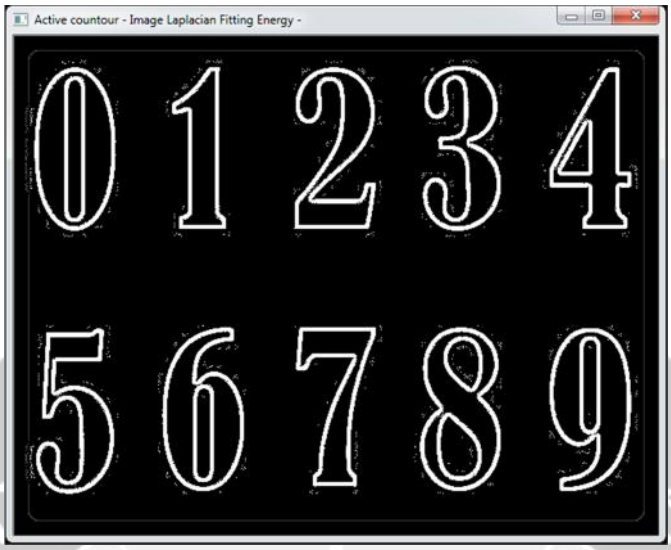


Gambar 6.2c. Kontur Citra ct_scan.bmp metode ILF iterasi ke-12

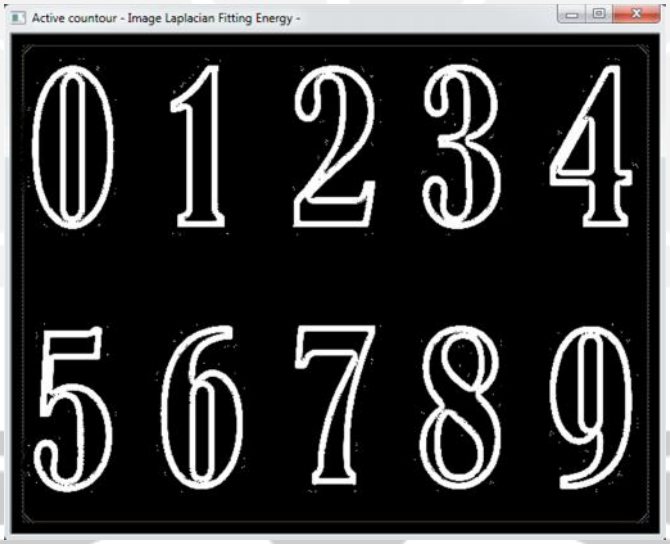
- Perhitungan kontur angka.jpg



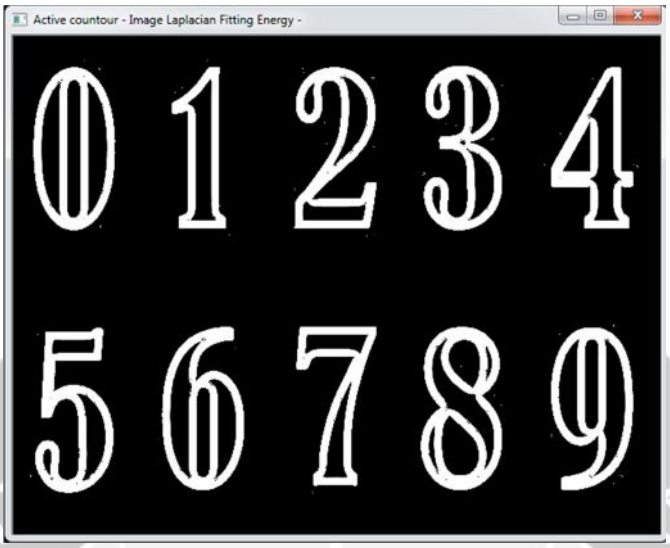
Gambar 6.3. Citra angka.jpg



Gambar 6.4a. Kontur Citra angka.jpg metode ILF iterasi ke-4



Gambar 6.4b. Kontur Citra angka.jpg iterasi metode ILF ke-8



Gambar 6.4c. Kontur Citra angka.jpg iterasi metode ILF ke-12

Berdasarkan dari hasil perhitungan yang diperlihatkan pada Gambar 6.2a - 6.2c dan Gambar 6.4a - 6.4c dapat dilihat bahwa metode Image Laplacian Fitting (ILF) dapat men-segmentasi citra denga baik, detil contour pada tepi citra yang rumit tetap dapat diikuti oleh kurva *level set*.

Metode Active Contours Berdasar Local Image Fitting Energy

Zhang et. al (2009) mengusulkan metode active contour baru dengan menambahkan Local Image Fitting (LIF) untuk mengekstrak informasi citra yang bersifat lokal. Metode ini lebih sederhana karena regularisasi fungsi *level set* digantikan dengan Gaussian filtering, sehingga menghindari perhitungan Laplacian ataupun turunan parsial secara langsung. Selain itu proses reinisialisasi level set juga tidak diperlukan, sehingga lebih mempermudah perhitungannya

Bagian berikut menjabarkan algoritma Active Contour Model with Local Image Fitting Energy, yaitu :

1. Inisialisasi fungsi *level set* ϕ
2. Langkah waktu perhitungan level set (ϕ) dimulai
3. Hitung fungsi *Heaviside* dan fungsi *Dirac*

$$H_\varepsilon = \frac{1}{2}\left(1 + \frac{2}{\pi}\arctan\left(\frac{\phi}{\varepsilon}\right)\right)$$

$$\delta_\varepsilon = \frac{1}{\pi}\frac{\varepsilon}{\phi^2 + \varepsilon^2}$$

4. Hitung fungsi Rectangular Window

$$m_1(\mathbf{x}) = \frac{K_\sigma * \left[H_\varepsilon(\phi) I(X)\right]}{K_\sigma * (\phi) H_\varepsilon(\phi)}$$

$$m_2(\mathbf{x}) = \frac{K_\sigma * \left[1 - H_\varepsilon(\phi) I(X)\right]}{K_\sigma * (1 - H_\varepsilon(\phi))}$$

K_σ adalah kernel filter dengan σ merupakan standar deviasinya, tanda '*' menyatakan proses konvolusi. Apabila nilai σ terlalu kecil memungkinkan hasil yang diperoleh tidak sesuai dengan keinginan, dan sebaliknya apabila nilai σ terlalu besar akan membutuhkan proses komputasi yang

5. Fungsi Local Image Fitting Energy

$$I^{LFI} = m_1 H_\varepsilon(\mathbf{X}) + m_2 (1 - H_\varepsilon(\mathbf{X}))$$

Fungsi *Local Image Fitting Energy* adalah meminimalkan perbedaan antara *fitted image* dan citra original

6. Update nilai level set

$$\phi_{i,j}^{n+1} = \phi_{i,j}^{n} + \Delta t (I - I^{LFI})(m_1 - m_2)\delta_\varepsilon(\phi_{i,j}^{n})$$

7. Cek batas waktu maksimum. Jika batas belum terlampaui, kembali langkah 5. Jika sudah terlampaui, lanjutkan ke langkah 7.
8. Tulis data
9. Selesai.

Implementasi algoritma Active Contours berdasar Local Image Fitting Energy dengan OpenCV adalah seperti berikut:

```
/*
Active contours driven by local image fitting energy
Kaihua Zhang, Huihui Song, Lei  Zhang
Pattern Recognition, 2009
*/

#include <iostream>
#include <vector>

#include “opencv2/imgproc/imgproc.hpp”
#include “opencv2/highgui/highgui.hpp”
#include “opencv2/features2d/features2d.hpp”

using namespace std;
using namespace cv;

/// Global Variables
int DELAY_VIEW =  2000;

//Const Parameter
const double pi=  3.121592653589793;

const float sigma =  1;
const double epsilon = 1.50;
const double timestep =  0.5;

Mat src;
Mat dst;
Mat phi;
Mat rhs;
Mat delta;
Mat hphi;
```

```
Mat m1;
Mat m2;
Mat f1;
Mat f2;
Mat c1;
Mat c2;


char window_name[] = "Active countour - LIF -";

/// Function headers
int display_dst( int delay  );

/**
 * function main
 */
int main( void  )
{
 Mat fx,fy,MLaplace,phi,u;
 vector<cv::Point> mcontour;

 namedWindow( window_name, WINDOW_AUTOSIZE  );
 /// Load the source  image
 src = imread( "ct_scan.bmp", CV_LOAD_IMAGE_GRAYSCALE );
 dst=src.clone();
 dst.convertTo(dst,CV_64FC1);


 double icenter=(double) src.rows/2.0;
 double jcenter=(double) src.cols/2.0;

 phi = Mat::ones(src.rows,src.cols,CV_64FC1);
 hphi = Mat::zeros(src.rows,src.cols,CV_64FC1);
 delta = Mat::zeros(src.rows,src.cols,CV_64FC1);
 hphi = Mat::zeros(src.rows,src.cols,CV_64FC1);
```

```
rhs = Mat::zeros(src.rows,src.cols,CV_64FC1);
hphi = Mat::zeros(src.rows,src.cols,CV_64FC1);
c1 = Mat::zeros(src.rows,src.cols,CV_64FC1);
c2 = Mat::zeros(src.rows,src.cols,CV_64FC1);
f1 = Mat::zeros(src.rows,src.cols,CV_64FC1);
f2 = Mat::zeros(src.rows,src.cols,CV_64FC1);
m1 = Mat::zeros(src.rows,src.cols,CV_64FC1);
m1 = Mat::zeros(src.rows,src.cols,CV_64FC1);
int radius=1;

for (int i = radius; i < phi.rows-radius; i++)
{
      for (int j = radius; j < phi.cols-radius; j++)
      {
            phi.at<double>(i,j)=-1.0;
            }
}

for (int iterasi = 0; iterasi < 50; iterasi++)
{

      // kalkulasi heaveside function (Zhang eq. 11)

      for (int i = 0; i < phi.rows; i++)
{
      for (int j = 0; j < phi.cols; j++)
      {
            hphi.at<double>(i,j)=0.5*(1.0+(2.0/pi)*atan(phi.
at<double>(i,j)/epsilon));
      }
}


// kalkulasi delta (dirac function) (Zhang eq. 11)
```

```
                                        divide(1.0/pi*epsilon,phi.
mul(phi)+pow(epsilon,2),delta);

 // f1 =  conv2(I.*H,K);
        GaussianBlur(dst.mul(hphi),f1,Size(21,21),sigma);
 // c1 =  myconv2(H,K);
        GaussianBlur( hphi,c1,Size(21,21),sigma);


        divide(f1,c1,m1); //(Zhang eq.  10)
 // f2 =  conv2(I.*(1-H),K);
                                        GaussianBlur(dst.mul((1.0-
hphi)),f2,Size(21,21),sigma);
 // c2 =  myconv2((1-H),K);
        GaussianBlur((1.0- hphi),c2,Size(21,21),sigma);


        divide(f2,c2,m2); //(Zhang eq.  10)

        rhs = dst - m1.mul(hphi) - m2.mul((1.0-hphi));
        rhs = rhs.mul(m1-m2);

 //(Zhang eq. 15) detil ada di bagian appendiks
        phi = phi +  timestep*delta.mul(rhs);

        imshow(window_name,phi);
        cout<<"Iterasi ke-"<<iterasi+1<<endl;
        waitKey(1000);
 }
 waitKey(0);

 return 0;
}
```

Hasil perhitungannya adalah seperti berikut:

Gambar 6.5a. Kontur Citra hasil perhitungan metode LIF ke-5

Gambar 6.5b. Kontur Citra hasil perhitungan metode LIF ke-10

Gambar 6.5c. Kontur Citra hasil perhitungan metode LIF ke-21

Gambar 6.5d. Kontur Citra hasil perhitungan metode LIF ke-50

Berdasarkan dari hasil perhitungan yang diperlihatkan pada Gambar 6.5a - 6.5d dapat dilihat bahwa metode *Local Image Fitting* (LIF) juga dapat men-segmentasi citra denga baik, tetapi kualitas konturnya tidak sebaik kontur yang dihasilkan metode *Image Laplacian Fitting* (ILF).

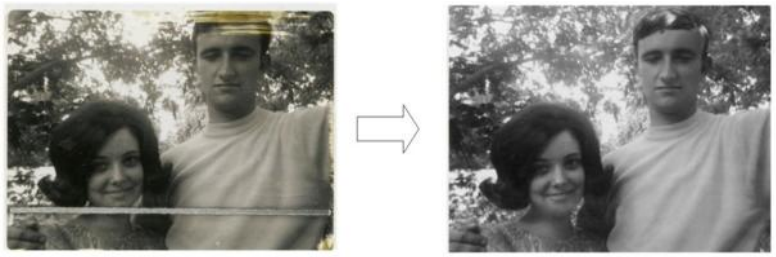
serviens in lumine veritatis

BAB VII
INPAINTING

7.1. Definisi Inpainting

Inpainting merupakan proses perbaikan citra karena ada coretan, pemudaran warna maupun kerusakan pada bagian tertentu pada citra. Pada awalnya metode ini digunakan para ilmuwan untuk merekonstruksi ulang lukisan dinding "fresco" pada bangunan kuno. Ide dari metode ini adalah mengosongkan piksel-piksel di bagian citra yang rusak atau hilang kemudian mengisinya dengan cara interpolasi dari piksel-piksel di sekitar bagian yang rusak.

Sejarah *Inpainting* berawal dari dunia seni. Istilah *inpainting* berasal dari bidang seni lukis, dimana para seniman bekerja dalam merestorasi lukisan yang kabur atau rusak. Tujuan utama dalam *inpainting* adalah untuk memperbaiki bagian lukisan yang rusak dengan cara untuk melengkapi atau mengisi bagian lukisan yang rusak dengan warna dan corak yang sesuai dengan bagian lukisan yang di dekatnya. Istilah inpainting kemudian dipinjam dalam dunia digital menjadi suatu acara atau algoritm untuk memperbaiki bagian citra digital yang rusak atau hilang.



Gambar 7.1 Contoh hasil perbaikan citra dengan Inpainting
https://en.wikipedia.org/wiki/Inpainting

Bertalmio dkk (2000) mempelopori proses inpainting untuk restorasi citra digital. Proses inpainting dilakukan dengan cara menirukan teknik pelukis profesional dalam memperbaiki lukisan yang rusak dan proses tersebut dilakukan secara otomatis oleh komputer. Urutan kerjanya adalah memilih bagian citra yang rusak, setelah itu mengisi bagian citra yang rusak berdasar informasi citra di sekitar bagian citra yang rusak tersebut. Bertalmio dkk menggunakan pendekatan PDE dalam proses inpaintingnya, persamaan yang digunakan adalah persamaan diffusi nonlinear., Banyak peneliti yang ikut tertarik melakukan riset di bidang inpainting digital sejak Bertalmio berhasil mengembangkan inpainting digital. Beberapa peneliti yang mencoba mengembangkan adalah Oliveira dkk (2001), Criminisi dkk (2004), Bertozzi dkk(2007), dan Schonlieb dan Bertozzi (2011).

Bab ini membahas 2 buah metode inpainting, yang pertama adalah inpainting dengan menggunakan peersamaan panas konduksi atau persamaan diffusi. Metode ini lebih dikenal dengan nama inpainting berbasis Persamaan Heat. Metode adalah inpainting berbasis persamaan Cahn Hilliard.

Metode Inpainting Berbasis Persamaan Heat

Persamaan yang digunakan oleh metode inpainting merupakan PDE parabolik seperti yang digunakan di bagian deteksi tepi tetapi dengan sedikit modifikasi seperti berikut:

$$\frac{\partial u}{\partial t} = \alpha\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)+\lambda\,(u-I) \tag{7.1}$$

Variabel *u* adalah nilai intensitas piksel hasil inpainting, I adalah nilai intensitas citra asli sedangkan λ dalah nilai piksel matriks topeng (*mask*). Diskretisasi persamaan (7.1) adalah seperti berikut:

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \Delta t(u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n} - 4u_{i,j}^{n}) - \Delta t\lambda(u_{i,j}^{n} - I_{i,j}^{n}) \tag{7.2}$$

Algoritma inpainting dengan Persamaan Heat adalah seperti berikut:

1. Baca data citra dan mask
2. Konversi tipe data citra dan mask menjadi tipe *double* atau *real* , tentukan nilai parameter perhitungan.
3. Inisialisasi nilai hasil rekonstruksi (u).
4. Langkah waktu perhitungan level set (ϕ) dimulai
   - update nilai u pada Persamaan 7.2.
5. Cek batas waktu maksimum. Jika batas belum terlampaui, kembali langkah 4. Jika sudah terlampaui, lanjutkan ke langkah 6.
6. Tulis data
7. Selesai

Algoritma di atas dituangkan dalam program seperti di bawah ini.

```
#include <iostream>

#include “opencv2/imgproc/imgproc.hpp”
#include “opencv2/highgui/highgui.hpp”
#include “opencv2/features2d/features2d.hpp”

using namespace std;
using namespace cv;

/// Global Variables
int DELAY_CAPTION = 2000;
int DELAY_BLUR =  100;
int MAX_KERNEL_LENGTH =  31;

Mat src; Mat u; Mat  mask;
char window_name[] = “Tugas Pengolahan Citra “;
```

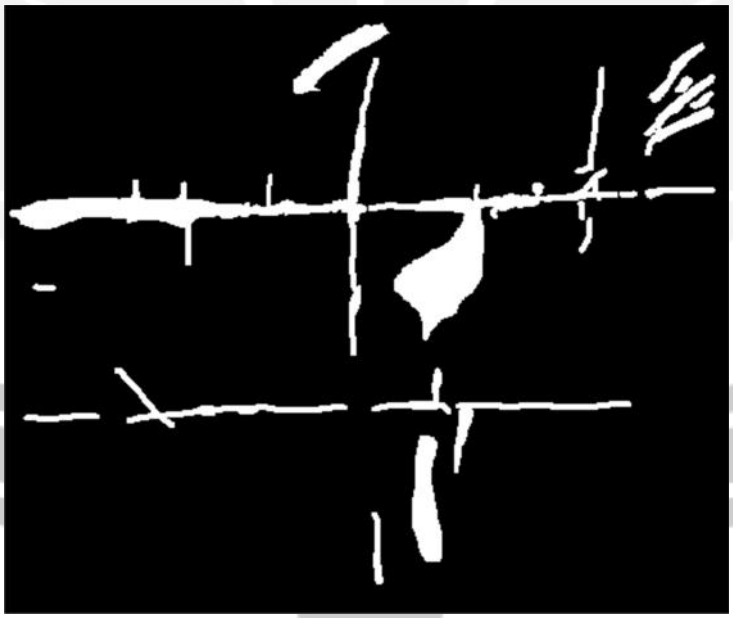
```
/**
 * function main
 */
int main( void  )
{
 // namedWindow( window_name, WINDOW_AUTOSIZE );
 double maxs=0,mins=0,maxm=0,minm=0,maxg=0,ming=0;
 Mat lambda,g;
 /// Load the source  image
 src = imread( "original.png", CV_LOAD_IMAGE_GRAYSCALE );
 mask = imread("mask.png",CV_LOAD_IMAGE_GRAYSCALE);
 src.convertTo(src,CV_64FC1);
 mask.convertTo(mask,CV_64FC1);

 minMaxLoc(src,&mins,&maxs);
 minMaxLoc(mask,&minm,&maxm);
 src=src/maxs;

 mask=1-mask/maxm;


 lambda=mask.clone();
 g=src.clone();
 minMaxLoc(g,&ming,&maxg);
 //divide(g,maxg,g);

 //parameter
 double h1=1,h2=1,T=150,dt=0.1,lambda0=10;
 Mat L1, L2;
 int m=g.rows;
 int n=g.cols;
 lambda = lambda.mul(lambda0);

 imshow("Lambda",lambda);
 imshow("source",g);
```

```
waitKey(2);

L1=Mat::zeros(m,m,CV_64FC1);
L2=Mat::zeros(n,n,CV_64FC1);

for(int i=0;i<m;i++)
{
      for(int j=0;j<m;j++)
      {
            if(i==0 && j ==0  )
            {
                  L1.at<double>(i,j)=(double)-1;
                  L1.at<double>(i+1,j)=1;
            }else if(i==m-1 &&  j==m-1)
            { L1.at<double>(i,j)=-1;
                  L1.at<double>(i,j-1)=1;
            }
            if(i==j && i!=0 &&  i!=m-1)
            {
                  L1.at<double>(i-1,j)=1;
                  L1.at<double>(i,j)=-2;
                  L1.at<double>(i+1,j)=1;
            }

      }
}

for(int i=0;i<n;i++)
{
      for(int j=0;j<n;j++)
      {
            if(i==0 && j ==0  )
            {
                  L2.at<double>(i,j)=(double)-1;
                  L2.at<double>(i+1,j)=1;
```

```
            }else if(i==n-1 &&  j==n-1)
            { L2.at<double>(i,j)=-1;
                  L2.at<double>(i,j-1)=1;
            }
            if(i==j && i!=0 &&  i!=n-1)
            {
                  L2.at<double>(i-1,j)=1;
                  L2.at<double>(i,j)=-2;
                  L2.at<double>(i+1,j)=1;
            }
      }
 }

 u=g.clone();
 u.convertTo(u,CV_64FC1);


 for(int i=0;i<100;i++)
 {
       u=u+dt*((L1*u)+(u*L2)+(lambda.mul(g-u)));
        //putText(  u,  "Iterasi  ke  -"+std::to_string(i),
Point( 20, 40),FONT_HERSHEY_COMPLEX, 1, Scalar(0)  );

       imshow("Inpainting Iterasi",u);
       waitKey(1);
       cout<<"Iterasi - "<<i+1<<endl;
 }
 waitKey(0);

 return 0;
}
```

Hasil inpainting dengan persamaan Heat disajikan seperti berikut:

Gambar 7.2a. Citra asli yang akan direkonstruksi



Gambar 7.2b. Citra topeng (*mask*)

Gambar 7.3a Citra hasil inpainting dengan Persamaan Heat iterasi ke-10.

Gambar 7.3b Citra hasil inpainting dengan Persamaan Heat iterasi ke-30

Gambar 7.3c Citra hasil inpainting dengan Persamaan Heat iterasi ke-50

Gambar 7.3d Citra hasil inpainting dengan Persamaan Heat iterasi ke-80

Gambar 7.3a - 7.3d memperlihatkan proses perbaikan citra dengan proses inpainting menggunakan Persamaan Heat, berdasar gambar-gambar tersebut dapat dilihat bahwa bagian citra yang rusak ditandai dengan menggunakan *mask* kemudian diisi secara gradual.

Inpainting Berbasis Persamaan Cahn Hilliard

Persamaan Chan Hilliard biasa digunakan untuk pemodelan mekanika fluida yang bersifat kompleks seperti pemodelan pertumbuhan kristal, pencampuran logam dalam fase cair, cairan polimer dal lain-lain. Persamaan ini termasuk nonlinier dan berorde-4 sehingga relatif sukar dipecahkan.

$$\frac{\partial u}{\partial t} = \alpha\left(\frac{\partial^2 \mu}{\partial x^2} + \frac{\partial^2 \mu}{\partial y^2}\right) - \lambda(u - I)$$
$$\mu = u^3 - u - \varepsilon^2\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{7.3}$$

Variabel u adalah nilai intensitas piksel hasil inpainting, μ dalam hai ini boleh dianggap sebagai variabel dummy. Diskretisasi Beda Hingga untuk persamaan (7.3) adalah seperti berikut:

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \Delta t\left(\mu_{i+1,j}^{n} + \mu_{i-1,j}^{n} + \mu_{i,j+1}^{n} + \mu_{i,j-1}^{n} - 4\mu_{i,j}^{n}\right) - \Delta t\lambda\left(u_{i,j}^{n} - I_{i,j}^{n}\right)$$

$$\mu_{i,j}^{n} = \left(u_{i,j}^{n}\right)^3 - u_{i,j}^{n} - \varepsilon^2\left(u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n} - 4u_{i,j}^{n}\right) \tag{7.4}$$

Algoritma untuk kasus inpainting ini sama dengan algoritma sebelumnya.

Inpainting Berbasis Persamaan Cahn Hilliard adalah seperti berikut:

```
#include <iostream>

#include “opencv2/imgproc/imgproc.hpp”
#include “opencv2/highgui/highgui.hpp”
#include “opencv2/features2d/features2d.hpp”

using namespace std;
using namespace cv;

/// Global Variables
int DELAY_CAPTION = 2000;
int DELAY_BLUR =  100;
int MAX_KERNEL_LENGTH =  31;

Mat src; Mat u; Mat  mask;
char window_name[] = “Cahn Hilliard Inpainting “;


/**
 * function main
 */
int main( void  )
{
 // namedWindow( window_name, WINDOW_AUTOSIZE );
 double maxs=0,mins=0,maxm=0,minm=0,maxg=0,ming=0;
 Mat lambda,g;
 /// Load the source  image
 src = imread( “new_original.png”,  CV_LOAD_IMAGE_GRAYSCALE
);
 mask = imread(“new_mask.png”,CV_LOAD_IMAGE_GRAYSCALE);
 src.convertTo(src,CV_64FC1);
 mask.convertTo(mask,CV_64FC1);

 minMaxLoc(src,&mins,&maxs);
 minMaxLoc(mask,&minm,&maxm);
```

```
src=src/maxs;

mask=1-mask/maxm;


lambda=mask.clone();
g=src.clone();
minMaxLoc(g,&ming,&maxg);
//divide(g,maxg,g);

//parameter
double h1=1,h2=1,T=150,dt=0.05,lambda0=5, eps=0.5;
Mat L1, L2, Mu,  Dummy;
int m=g.rows;
int n=g.cols;
lambda = lambda.mul(lambda0);

imshow("Lambda",lambda);
imshow("Original Image",g);
waitKey(2);

L1=Mat::zeros(m,m,CV_64FC1);
L2=Mat::zeros(n,n,CV_64FC1);
Mu=Mat::zeros(m,n,CV_64FC1);
Dummy=Mat::zeros(m,n,CV_64FC1);

for(int i=0;i<m;i++)
{
      for(int j=0;j<m;j++)
      {
            if(i==0 && j ==0  )
            {
                  L1.at<double>(i,j)=(double)-1;
                  L1.at<double>(i+1,j)=1;
            }else if(i==m-1 &&  j==m-1)
```

```
                { L1.at<double>(i,j)=-1;
                        L1.at<double>(i,j-1)=1;
                }
                if(i==j && i!=0 &&  i!=m-1)
                {
                        L1.at<double>(i-1,j)=1;
                        L1.at<double>(i,j)=-2;
                        L1.at<double>(i+1,j)=1;
                }

        }
}

for(int i=0;i<n;i++)
{
        for(int j=0;j<n;j++)
        {
                if(i==0 && j ==0  )
                {
                        L2.at<double>(i,j)=(double)-1;
                        L2.at<double>(i+1,j)=1;
                }else if(i==n-1 &&  j==n-1)
                { L2.at<double>(i,j)=-1;
                        L2.at<double>(i,j-1)=1;
                }
                if(i==j && i!=0 &&  i!=n-1)
                {
                        L2.at<double>(i-1,j)=1;
                        L2.at<double>(i,j)=-2;
                        L2.at<double>(i+1,j)=1;
                }
        }
}
```

```
 u=g.clone();
 u.convertTo(u,CV_64FC1);


 for(int i=0;i<200;i++)
 {
        Dummy = u.mul(u);
        Mu=Dummy.mul(u)-u- eps*eps*(L1*u+u*L2);
        u=u+dt*((L1*Mu)+(Mu*L2)+(lambda.mul(g-u)));
         //putText(  u,  "Iterasi  ke  -"+std::to_string(i),
Point( 20, 40),FONT_HERSHEY_COMPLEX, 1, Scalar(0)  );

        imshow("Cahn Hilliard Inpainting",u);
        waitKey(1);
        cout<<"Iterasi - "<<i+1<<endl;
 }
 waitKey(0);

 return 0;
}
```

Hasil inpainting dengan persamaan Cahn Hilliard disajikan seperti berikut:

Gambar 7.4a. Citra hasil inpainting dengan Persamaan Cahn Hilliard iterasi ke-20

Gambar 7.4b. Citra hasil inpainting dengan Persamaan Cahn Hilliard iterasi ke-30

Gambar 7.4c. Citra hasil inpainting dengan Persamaan Cahn Hilliard iterasi ke-80

Diskretisasi Persamaan 7.4 merupakan diskretisasi eksplisit sehingga mempunyai keterbatasan pemilihan Δt yang kecil untuk menjaga kestabilan, untuk Persamaan Cahn Hilliard Δt harus lebih kecil lagi dibanding Δt milik Persamaan Heat karena Cahn Hilliard adalah PDE orde-4. Sehingga inpainting Cahn Hilliard dengan cara di atas rentan terhadap kestabilan perhitingan. Untuk mengatasi hal tersebut disarankan menggunakan formulasi implisit. Untuk mengetahui perkembangan metode inpainting yang bersifat lebih lanjut, pembaca disarnkan membaca naskah Bertozzi (2007) dan Schonlieb (2011), mereka memodifikasi Persamaan Cahn Hilliard dan menggunakan diskretisasi ruang dengan metode Spectral dan diskretisasi waktu dengan formulasi implisit.

DAFTAR PUSTAKA

Aubert, G. dan Kornprobst, P., (2002), *Mathematical Problems in Image Processing*, Springer Verlag, New York

Bradski, G. dan Kaehler, A. (2008), *Learning OpenCV*, O'Reilly Media, Inc., Sebastopol, CA

Bertalmio, M., Sapiro, G., Caselles, V. & Ballester, C., (2000), Image inpainting, *SIGGRAPH ACM*, pp.417-24.

Bertozzi, A. dkk, (2007), Analysis of A Two-Scale Cahn-Hilliard Model for Binary Image Inpainting, *MULTISCALE MODEL SIMUL.*, Vol. 6, No. 3, pp. 913–936.

Chan, T. and Shen, J., (2005), *Image Processing and Analysis: Variational, PDE, Wavelet and Stochastic Methods*, SIAM Society for Industrial and Applied Mathematics., Philadelphia.

Criminisi, A., Perez, P. & Toyama, K., (2004), Region Filling and Object Removal by Exemplar-Based Image Inpainting, *IEEE Transactions on Image Processing*, 13(9).

Hoffman, K.A. dan Chiang, S. T., (2000), *Computational Fluid Dynamics Vol. I (Fourth Edition*), Engineering Education Center, Wichita, USA. .

Howe, K.D. (20140), *A Practical Introduction to Computer Vision with OpenCV, John Wiley and Sons Ltd*, Chicester, United Kingdom.

Kass, M., Witkin, A. & Terzopoulos, D., (1987). Snakes : Active Contour Models. *International Journal of Computer Vision*, I(4), pp.321-31.

Osher, S. dan Paragios, N. ,(2003), *Geometric Level Set Methods in Imaging, Vision, and Graphic*, Springer Verlag, New York.

Oliveira, M., Bowen, B., McKenna, R. & Chang, Y., (2001) Fast Digital Image Inpainting. *Proceeding VIIP 2001*, pp.261-66.

Perona, P. & Malik, J., (1990). Scale Space and Edge Detection Using Anisotropic Diffusion, *IEEE Transaction On Pattern Analysis and Machine Intelligence*, 12(7), pp.629-39.

Schonlieb, C. B dan Bertozzi, A.. (2011), Unconditionally Stable Scheme for Higher Order Inpainting, *COMMUN. MATH. SCI*, Vol. 9, No. 2, pp. 413–457.

Solomon, C. and Breckon, T. ,(2011), *Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab*, Willey-Blackwell A John Wiliy & Sons Ltd Publication, Chichester.

Zhang, K., Song, H. & Zhang, L., (2009), Active Contours Driven by Local Image Fitting Energy, *Pattern Recognition*, vol.43, issue 4, pp. 1199-1206.

Zhang dkk (2010). Active Contours Based On Image Laplacian Fitting Energy *Chinese Journal of Electronics*. Vol.18, No.2.

Pengolahan Citra Berbasis PDE dengan OpenCV

Materi dalam buku ini merupakan sebagian dari bahan kuliah Pengolahan Citra Magister Teknik Informatika. Buku ini membahas pengolahan citra dari sudut yang berbeda dari kebanyakan buku pengolahan citra yang sudah ada di toko-toko buku di Indonesia. Pembaca diharapkan sudah memahami teori dasar Pengolahan Citra sebelum membaca buku ini. Pendekatan dalam buku ini adalah citra dianggap sebagai fungsi matematika multivariabel sehingga bisa dimodelkan dengan persamaan differensial parsial atau yang lebih dikenal dengan nama PDE. Metode numerik yang sudah "*well established*" dapat diadopsi dengan baik untuk pengolahan citra. Implementasi ke dalam program komputer menggunakan Bahasa C/C++ dan pustaka OpenCV. Sedangkan *compiler* yang digunakan adalah Microsoft Visual Studio 2010.

Buku ini terdiri dari 7 bab, bab pertama berisi penjelasan hubungan citra dengan PDE. Bab II berisi solusi numerik PDE dengan metode Beda Hingga. Bab 3 berisi uraian tentang OpenCV. Bagi pembaca yang sudah menguasai metode numerik untuk PDE dan OpenCV dapat langsung mempelajari pada Bab IV yang berisi tentang deteksi tepi. Bab V berisi materi tentang penapisan derau, bab VI memabahas proses segmentasi dan bab VII membahas tentang inpainting. Materi bab IV-VII merupakan materi yang terpisah sehingga dalam mempelajarinya tidak harus urut.

ISBN:978-602-7821-74-3

9 786027 821743

Cahaya Atma Pustaka

Jl. Moses Gatotkaca No. 28, Yogyakarta
e-mail : cahayaatma@gmail.com
Telp. (0274) 561031, 580526, Fax. (0274) 58