ANALISA PERBANDINGAN KINERJA DETEKSI TEPI MENGGUNAKAN METODE LoG, SOBEL, dan CANNY TERHADAP FORMAT FILE JPEG dan BMP

SKRIPSI

MUHAMMAD AKBAR AMIN
NIM 071910201023

PROGRAM STUDI STRATA-I TEKNIK ELEKTRO
JURUSAN TEKNIK ELEKTRO
FAKULTAS TEKNIK
UNIVERSITAS JEMBER
2012

ANALISA PERBANDINGAN KINERJA DETEKSI TEPI MENGGUNAKAN METODE LoG, SOBEL, dan CANNY TERHADAP FORMAT FILE JPEG dan BMP

SKRIPSI

diajukan guna melengkapi skripsi dan memenuhi syarat-syarat
untuk menyelesaikan Program Studi Teknik Elektro (S1)
dan guna mencapai gelar Sarjana Teknik

MUHAMMAD AKBAR AMIN
NIM : 071910201023

PROGRAM STUDI STRATA-I TEKNIK ELEKTRO
JURUSAN TEKNIK ELEKTRO
FAKULTAS TEKNIK
UNIVERSITAS JEMBER
2012

PERSEMBAHAN

Puji dan syukur penulis panjatkan kepada Allah Subhanahu Wa Ta'alaa Yang Maha Pengasih dan Maha Penyayang, dengan limpahan karunia-Nya skripsi ini berhasil diselesaikan.

Skripsi ini dapat terselesaikan juga berkat bantuan dari berbagai pihak, oleh karena itu saya persembahkan untuk :

1. Ayah dan Ibu-ku yang saya cintai "Sri haryono" dan "Sunarsih" yang tiada lelah memarahi saya agar menjadi manusia yang BENAR. TERIMA KASIH.
2. Saudara-saudara-ku yang saya banggakan, "Muhammad Rizal Amin", "Muhammad Qadhafi Amin", "Muhammad Rizky Amin", dan "Chairun Nisya" yang tiada hentinya memberi semangat untuk menyelesaikan Skripsi ini. TERIMA KASIH.
3. Dosen – dosen Teknik Elektro Universitas Jember yang telah memberikan ilmunya dengan Ikhlas. TERIMA KASIH.
4. Pembimbing Utama Bapak "H. R. B. Moch. Gozali, S.T., M.T." dan Pembimbing Akademik "Ir. Widyono Hadi, M.T.", Juga Bapak "Dr. Triwahju Hardianto, S.T., M.T.", dan "Sumardi, S.T., M.T." selaku Penguji yang telah sabar dan Ikhlas membimbing saya. TERIMA KASIH.
5. Penjual Nasi goreng "Pak Man" yang telah memberikan spirit dengan masakannya. TERIMA KASIH.
6. Teman – teman Teknik Elektro Universitas Jember angkatan 2007 yang telah benar – benar membuat tahu Jati Diriku sebenarnya. Selamanya kita pasti tetap akan bertemu. ELEKTRO JOSSSSS !!!!!. TERIMA KASIH.
7. Teman – Teman "HIKARI Band", yang juga memberi-kan semangat dalam mengerjakan skripsi ini. TERIMA KASIH
8. Teman – teman "J-zone Indonesia" yang juga telah memberikan SEMANGAT-nya, TERIMA KASIH.
9. Sahabatku yang sudah ada di Surga, "Eky Filmas Kayfu Rahmanta" TERIMA KASIH.

10. Saudara – saudaraku yang selalu mendoakan agar menjadi sukses.

    TERIMA KASIH.

MOTTO

Jangan sia – siakan waktu, meskipun Cuma sedetik saja

(Muhammad Akbar Amin)

Heaven and Hell suppose two distinct species of men, the good and the bad. But the greatest part of mankind float betwixt vice and virtue

(David Hume)

Jadilah Orang bodoh yang pintar, dari pada jadi orang pintar yang bodoh.

(Muhammad Akbar Amin)

Jangan hanya dipikir, tapi kerjakanlah.

(**Muhammad Akbar Amin**)

PERNYATAAN

Saya yang bertanda tangan dibawah ini :

Nama : Muhammad Akbar Amin

Nim : 071910201023

Menyatakan dengan sesungguhnya bahwa skrpsi yang berjudul :

"ANALISA PERBANDINGAN KINERJA DETEKSI TEPI MENGGUNAKAN METODE LoG, SOBEL, dan CANNY TERHADAP FORMAT FILE JPEG dan BMP" adalah benar-benar hasil karya sendiri, kecuali kutipan yang sudah saya sebutkan sumbernya , belum pernah diajukan pada institusi mana pun, dan bukan karya jiplakan. Saya bertanggung jawab atas keabsahan dan kebenaran isinya sesuai dengan sikap ilmiah yang harus dijunjung tinggi .

Demikian pernyataan ini saya buat dengan sebenarnya tanpa ada tekanan paksaan dari pihak mana pun serta bersedia mendapat sanksi akademik jika ternyata dikemudian hari pernyataan ini tidak benar.

Jember, 09 Februari 2012

Yang menyatakan

Muhammad Akbar Amin

NIM 071910201023

SKRIPSI

"ANALISA PERBANDINGAN KINERJA DETEKSI TEPI MENGGUNAKAN METODE LoG, SOBEL, dan CANNY TERHADAP FORMAT FILE JPEG dan BMP"

Oleh

Muhammad Akbar Amin

NIM. 071910201023

Pembimbing

Dosen Pembimbing I	: H.R.B. Moch. Gozali, S.T., M.T.
Dosen Pembimbing II	: Ir. Widyono Hadi, M.T.

PENGESAHAN

Skripsi yang berjudul "*Analisa Perbandingan Kinerja Deteksi Tepi Menggunakan Metode LoG, SOBEL, dan CANNY Terhadap Format File JPEG dan BMP*" telah diuji dan disahkan pada :

Hari : Selasa

Tanggal : 09 Februari 2012

Tempat : Fakultas Teknik Universitas Jember

Tim Penguji

Pembimbing Utama (Ketua penguji),

H.R.B. Moch. Gozali, S.T., M.T.
NIP 19690608 199903 1 002

Pembimbing Pendamping (Sekretaris),

Ir. Widyono Hadi, M.T.
NIP 19610414 198902 1 001

Penguji I,

Sumardi, S.T., M.T.
NIP 19670113 199802 1 001

Penguji II,

Dr. Triwahju Hardianto, S. T., M. T.
NIP 19700826 199702 1 001

Mengesahkan, Dekan Fakultas Teknik,
Universitas Jember

Ir. Widyono Hadi, M.T.
NIP 19610414 198902 1 001

ANALISA PERBANDINGAN KINERJA DETEKSI TEPI MENGGUNAKAN METODE LoG, SOBEL, dan CANNY TERHADAP FORMAT FILE JPEG dan BMP

Muhammad Akbar Amin [1]
Mahasiswa Jurusan Teknik Elektro [1]
Fakultas Teknik, Universitas Jember

ABSTRAK

Pengolahan citra bertujuan memperbaiki kualitas citra agar mudah diinterpretasi oleh manusia atau mesin (dalam hal ini komputer). Ada banyak metode pendeteksi tepi yang saat ini dikenal dalam perkembangan ilmu pengetahuan. Diantaranya, Sobel, Laplacian of Gaussian, dan Canny (metode yang diteliti). Masalahnya adalah bagaimana analisa kinerja metode Sobel, LoG, dan Canny terhadap format file BMP, dan JPEG, serta bagaimana kinerja tiap-tiap metode pendeteksi tepi jika citra yang dijadikan sebagai citra masukan dengan tingkat kualitas resolusi yang berbeda (2 megapiksel, 4 megapiksel, dan 6 megapiksel), objek yang berbeda (Gelas Kaca, Botol Plastik, Botol Semprot, dan Miniatur), juga format yang berbeda (JPG dan BMP) terhadap tiga buah parameter yang dianggap mampu mengukur kinerja metode ini, yaitu kualitas citra tepi yang dihasilkan, *timing run,* dan *sensitivity rate.* Penelitian ini bertujuan untuk Memberikan informasi mengenai pengaruh kualitas resolusi citra terhadap kinerja metode pendeteksi tepi. dan mengetahui Format File mana yang lebih bagus antara JPG dan BMP terhadap Sensivitas dan waktu pemrosesannya.

Kata Kunci : Pengolahan Citra, Deteksi Tepi, JPEG, BMP

COMPARATIVE PERFORMANCE ANALYSIS OF EDGE DETECTION USING LoG, SOBEL, and CANNY METHOD ON FILE FORMAT JPEG and BMP

Muhammad Akbar Amin [1]
Mahasiswa Jurusan Teknik Elektro [1]
Fakultas Teknik, Universitas Jember

ABSTRACT

Image processing aims to improve the image quality to be easily interpreted by humans or machines (in this case the computer). There are many edge detection methods are currently known in the development of science. Among them, Sobel, Laplacian of Gaussian, and Canny (method under study). The problem is how to analyze the performance of the method Sobel, LOG, and Canny to BMP file format, and JPEG, as well as how the performance of each method of detecting the edge if the image is used as the input image with different levels of quality resolution (2 megapixels, 4 megapixels, and 6 megapixels), different objects (Glass Glass, Plastic Bottles, Spray Bottles, and Miniature), as well as different formats (JPG and BMP) to the three parameters that are considered capable of measuring the performance of this method, the resulting edge image quality, timing run, and the sensitivity rate. This study aims to provide information about the influence of the quality of image resolution on the performance of edge detection methods. File Formats and find out which one is better between JPG and BMP to Sensitivity and time of processing.

Keywords : Image Processing, Edge Detection, JPEG, BMP

RINGKASAN

ANALISA PERBANDINGAN KINERJA DETEKSI TEPI MENGGUNAKAN METODE LoG, SOBEL, dan CANNY TERHADAP FORMAT FILE JPEG dan BMP Muhammad Akbar Amin, 071910201023; 2012:71 halaman; Jurusan Teknik Elektro Fakultas Teknik Universitas Jember.

Pengolahan citra bertujuan memperbaiki kualitas citra agar mudah diinterpretasi oleh manusia atau mesin (dalam hal ini komputer). Banyak teknologi saat ini menggunakan ilmu Pengolahan citra. salah satunya adalah deteksi sidik jari, iris mata dan banyak lagi. Ada banyak metode pendeteksi tepi yang saat ini dikenal dalam perkembangan ilmu pengetahuan. Diantaranya, Sobel, Laplacian of Gaussian (LoG), dan Canny (metode yang diteliti). Tiap-tiap metode pasti memiliki kelemahan dan kelebihan.

Saat ini resolusi kamera terus berkembang, bahkan telah mencapai puluhan Megapiksel. Format file bergambar juga banyak macamnya, ada JPEG(*Joint Photographic Experts Group*), dan BMP(*Bitmap*). Di setiap Format file bergambar tersebut mempunyai kelebihan dan kekurangan dalam menampilkan sebuah citra. Kualitas resolusi citra sebagai variabel yang dimungkinkan dapat mempengaruhi kinerja metode pendeteksi tepi, mulai dari kualitas resolusi yang dianggap rendah sampai kualitas resolusi yang dianggap tinggi. Berdasarkan ketiga metode pendeteksi tepi tersebut akan dipilih sebagai metode yang menjadi objek penelitian.

Untuk itu, peneliti akan membandingkan kinerja metode pendeteksi tepi tersebut dengan mengujinya terhadap kualitas resolusi citra dengan dua file format berbeda. Yaitu JPEG dan BMP. Ditetapkan tiga buah parameter pembanding untuk melihat kehandalan tiap metode, yaitu morfologi garis tepi yang dihasilkan, sensitifitas terhadap *noise* dikenal sebagai *sensitivity rate*, dan kecepatan proses dikenal sebagai *timing run*.

PRAKATA

Ucapan terima kasih penulis sampaikan kepada Bapak H.R.B. Moch. Gozali, S.T., M.T. dan Bapak Ir. Widyono Hadi, M.T. yang telah memberikan panduan dan penuh kepercayaan kepada saya untuk menyempurnakan kajian ini. Ucapan terima kasih juga saya tujukan kepada Dekan dan Pembantu Dekan Fakultas, semua dosen, dan seluruh pegawai di Fakultas Teknik Elektro yang memberikan izin untuk melakukan riset.

Skripsi ini juga penulis dedikasikan bagi Ibunda, Sunarsih yang kasih,Serta terima kasih yang tak berhingga juga penulis sampaikan untuk Ayahanda Sri Haryono, saudara-saudari penulis, Muhammad Rizal Amin, Muhammad Rizky Amin, Muhammad Qadhafi Amin, dan Chairun Nisya, serta seluruh keluarga besar yang tak henti-hentinya memberikan semangat dan dukungan bagi penulis untuk tetap berjuang. Dan spesial untuk Seluruh anggota KOKU-J (Komunitas Harajuku Jember), Gang of HIKARI Band, orang-orang terbaik yang pernah dimiliki penulis. Teman-teman penulis di kampus, khususnya keluarga Teknik Elektro 2007, dan seluruh pihak yang tidak disebutkan namanya yang telah banyak membantu penulis dalam menyelesaikan skripsi ini, penulis ucapkan banyak terima kasih.

Penulis sadar bahwa masih terdapat banyak kekurangan dalam penulisan skripsi ini. Karna itu, penulis dengan ikhlas hati menerima kritik dan saran yang membangun untuk memperbaiki penulisannya. Akhirnya, semoga skripsi ini dapat memberikan manfaat bagi perkembangan ilmu pengetahuan.

Jember, 09 Februari 2012

Penulis

DAFTAR ISI

Halaman

DAFTAR GAMBAR

Halaman

DAFTAR TABEL

Halaman

DAFTAR LAMPIRAN

BAB I
PENDAHULUAN

1.1 Latar Belakang

Pengolahan citra bertujuan memperbaiki kualitas citra agar mudah diinterpretasi oleh manusia atau mesin (dalam hal ini komputer). Banyak teknologi saat ini menggunakan ilmu Pengolahan citra. salah satunya adalah deteksi sidik jari, iris mata dan banyak lagi. Ada banyak metode pendeteksi tepi yang saat ini dikenal dalam perkembangan ilmu pengetahuan. Diantaranya, Sobel, Laplacian of Gaussian (LoG), dan Canny (metode yang diteliti). Tiap-tiap metode pasti memiliki kelemahan dan kelebihan.

Saat ini resolusi kamera terus berkembang, bahkan telah mencapai puluhan Megapiksel. Format file bergambar juga banyak macamnya, ada JPEG(*Joint Photographic Experts Group*), dan BMP(*Bitmap*). Di setiap Format file bergambar tersebut mempunyai kelebihan dan kekurangan dalam menampilkan sebuah citra. Kualitas resolusi citra sebagai variabel yang dimungkinkan dapat mempengaruhi kinerja metode pendeteksi tepi, mulai dari kualitas resolusi yang dianggap rendah sampai kualitas resolusi yang dianggap tinggi. Berdasarkan ketiga metode pendeteksi tepi tersebut akan dipilih sebagai metode yang menjadi objek penelitian.

Untuk itu, peneliti akan membandingkan kinerja metode pendeteksi tepi tersebut dengan mengujinya terhadap kualitas resolusi citra dengan dua file format berbeda. Yaitu JPEG dan BMP. Ditetapkan tiga buah parameter pembanding untuk melihat kehandalan tiap metode, yaitu morfologi garis tepi yang dihasilkan, sensitifitas terhadap *noise* dikenal sebagai *sensitivity rate*, dan kecepatan proses dikenal sebagai *timing run*.

1.2. Rumusan Masalah

Mengacu pada permasalahan yang telah diuraikan pada latar belakang, maka rumusan masalah dapat di susun sebagai berikut:

1. Bagaimana analisa kinerja metode Sobel, LoG, dan Canny terhadap format file BMP, dan JPEG.
2. Bagaimana kinerja tiap-tiap metode pendeteksi tepi jika citra yang dijadikan sebagai citra masukan dengan tingkat kualitas resolusi yang berbeda, objek yang berbeda (Gelas kaca, Botol Plastik, Botol Semprot, dan miniatur), juga format yang berbeda (JPG dan BMP)

1.3 Batasan Masalah

Batasan-batasan masalah yang dipakai dalam penulisan Tugas Akhir ini adalah sebagai berikut :

1. Metode yang akan dianalisis dan dibandingkan adalah hanya metode pendeteksi tepi Sobel, LoG, dan Canny.
2. Citra yang akan dijadikan objek penelitian adalah hanya citra digital dari Gelas kaca dan Botol Plastik dengan format gambar *.jpg (jpeg), dan *.bmp, dengan beberapa tingkatan resolusi, yaitu, 2MP(MegaPiksel) ,4MP dan 6MP. Ketiga resolusi ini dianggap mewakili citra dengan resolusi tinggi dan rendah. Parameter yang akan diukur untuk mengetahui hasil kinerja masing-masing metode pendeteksi tepi adalah sensitivitas tiap-tiap metode terhadap *noise (sensitivity rate),* morfologi garis yang dihasilkan pada citra keluaran hasil metode pendeteksi tepi, waktu yang diperlukan untuk melakukan proses (*timing run)*.

1.4 Tujuan Penelitian

Tujuan dari penelitian ini adalah:

1. Untuk mengetahui kinerja metode Sobel, LoG, dan Canny terhadap format BMP, dan JPEG.

2. Untuk mengetahui kinerja tiap-tiap metode pendeteksi tepi jika citra yang dijadikan sebagai citra masukan dengan tingkat kualitas resolusi yang berbeda, objek yang berbeda, juga format yang berbeda (JPG dan BMP)

1.5 Manfaat Penelitian

Manfaat penelitian ini adalah :

1. Memberikan informasi mengenai pengaruh kualitas resolusi citra terhadap kinerja metode pendeteksi tepi.
2. Dapat mengetahui Format File mana yang lebih bagus daripada JPG dan BMP terhadap Sensivitas dan waktu pemrosesannya

.

1.6 Sistematika Penulisan

Sistematika penulisan dari skripsi ini terdiri dari beberapa bagian utama sebagai berikut:

BAB I Pendahuluan

Bab ini akan menjelaskan mengenai latar belakang masalah yang dibahas dalam skripsi ini, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, metode penelitian, dan sistematika penulisan.

BAB II Tinjauan Pustaka

Bab ini mencakup segala teori yang menjadi landasan dalam penulisan Tugas Akhir yang berkaitan dengan citra digital, resolusi citra, deteksi tepi yang merupakan salah satu operasi dalam pengolahan citra digital, BMP file format, JPEG file format, serta teori mengenai pendeteksi tepi Sobel, pendeteksi tepi LoG, dan pendeteksi tepi Canny.

BAB III Metodelogi Penelitian

Membahas tentang metodelogi penelitian yang digunakan dalam tugas akhir

BAB IV Hasil dan Pembahasan

Bab ini menjelaskan tentang analisis pada perangkat yang akan digunakan untuk membangun sistem dan juga berisi perancangan sistem dari hasil analisis yang telah dilakukan. Bab ini juga berisi analisis hasil yang diperoleh dari pengujian

terhadap sistem yang akan dilakukan dengan menjadikan beberapa jenis kualitas resolusi citra yang berbeda sebagai citra masukan

BAB V Kesimpulan dan Saran

Bab terakhir ini memuat kesimpulan isi dari keseluruhan uraian bab-bab sebelumnya dan saran-saran dari hasil yang diperoleh yang diharapkan dapat bermanfaat dalam pengembangan selanjutnya.

BAB II
TINJAUAN PUSTAKA

2.1 Definisi Pengolahan Citra

Meskipun sebuah citra kaya informasi, namun seringkali citra yang kita miliki mengalami penurunan mutu (degradasi), misalnya mengandung cacat atau derau (*noise*), warnanya terlalu kontras, kurang tajam, kabur (*blurring*), dan sebagainya. Tentu saja citra semacam ini menjadi lebih sulit diinterpretasi karena informasi yang disampaikan oleh citra tersebut menjadi berkurang. Agar citra yang mengalami gangguan mudah diinterpretasi (baik oleh manusia maupun mesin), maka citra tersebut perlu dimanipulasi menjadi citra lain yang kualitasnya lebih baik. Bidang studi yang menyangkut hal ini adalah pengolahan citra (*image processing*).

Pengolahan citra bertujuan memperbaiki kualitas citra agar mudah diinterpretasi oleh manusia atau mesin (dalam hal ini komputer). Pengolahan citra telah menggunakan sistem komputer yang diaplikasikan pada sejumlah bidang, seperti pada bidang kedokteran, biologi, hukum, dan keamanan.

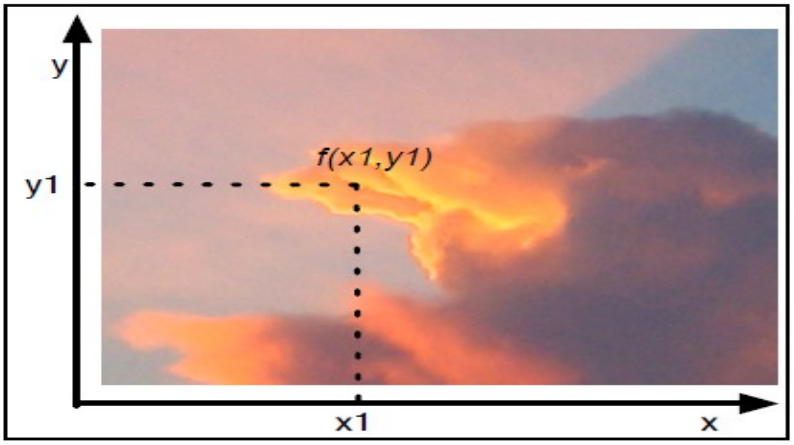
Pengolahan citra adalah pemrosesan citra, khususnya dengan menggunakan komputer, menjadi citra yang kualitasnya lebih baik. Sebagai contoh, citra burung nuri pada Gambar 1(a) tampak agak gelap, lalu dengan operasi pengolahan citra kontrasnya diperbaiki sehingga menjadi lebih terang dan tajam (b). (JAI,1989, Halaman 18)

(a) (b)

Gambar2.1 *(a) Citra burung nuri yang agak gelap, (b) Citra burung yang telah diperbaiki kontrasnya sehingga terlihat jelas dan tajam*

2.2 Citra Digital

Citra dapat dikatakan sebagai citra digital jika citra tersebut disimpan dalam format digital (dalam bentuk *file*). Seperti halnya proses digitisasi dalam bentuk data lain, proses digitisasi pada data citra juga merupakan proses pengubahan suatu bentuk data citra dari yang bersifat analog ke dalam bentuk data digital, yang mana proses ini dapat dilakukan dengan alat bantu, yang salah satunya berupa kamera. Hanya citra digital yang dapat diolah menggunakan komputer. Jenis citra lain jika akan diolah dengan komputer harus diubah dulu menjadi citra digital.



Gambar 2.2 Contoh citra digital

Citra digital merupakan suatu fungsi intensitas cahaya f(x,y), dimana harga x dan y merupakan koordinat spasial dan harga fungsi tersebut pada setiap titik (x,y) merupakan tingkat kecemerlangan citra pada titik tersebut. Citra digital dinyatakan dengan matriks berukuran N x M (N menyatakan baris atau tinggi, M menyatakan kolom atau lebar)

$$f(x,y) \approx \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,M-1) \\ f(1,0) & f(1,1) & \dots & f(1,M-1) \\ \vdots & \vdots & \vdots & \vdots \\ f(N-1,0) & f(N-1,1) & \dots & f(N-1,M-1) \end{bmatrix} \quad (2.1)$$

Keterangan:

N = jumlah baris, 0 = y = N – 1

M = jumlah kolom, 0 = x = M – 1

L = maksimal warna intensitas (derajat keabuan), 0 = f(x,y) = L – 1

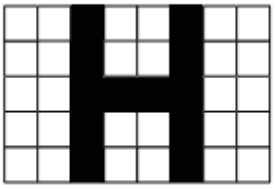
(Hestiningsih,, Halaman 45, 2008).

Citra digital biasanya berbentuk persegi panjang, secara visualisasi dimensi ukurannya dinyatakan sebagai lebar x tinggi. Ukurannya dinyatakan dalam titik atau piksel (pixel = *picture element*) dan dapat pula dinyatakan dalam satuan panjang (mm atau inci = *inch*).

Berdasarkan format penyimpanan nilai warnanya, citra terdiri atas empat jenis (Hestiningsih,Halaman 45, 2008), yaitu:

1. Citra biner atau monokrom

Pada citra jenis ini, setiap titik atau piksel hanya bernilai 0 atau 1.Dimana setiap titik membutuhkan media penyimpana sebesar 1 bit. Gambar 2.2 merupakan contoh citra biner.

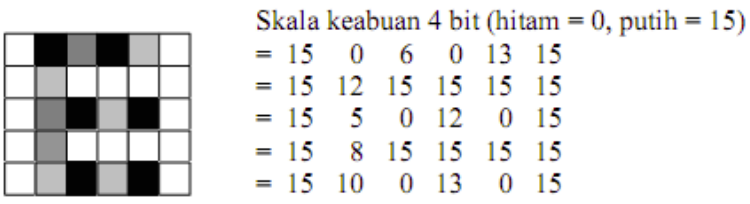


Gambar 2.3 Citra Biner

2. Citra skala keabuan

Citra skala keabuan mempunyai kemungkinan warna antara hitam (minimal) dan putih (maksimal). Jumlah maksimum warna sesuai dengan bit penyimpanan yang digunakan. Misal:

Suatu citra dengan skala keabuan 4 bit, memiliki jumlah kemungkinan warna $2^4 = 16$ warna. Gambar 2.3 memperlihatkan citra skala keabuan 4 bit.



Gambar 2.4 Citra Skala Keabuan

3. Citra warna (*true color*)

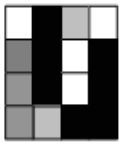
Setiap titik (piksel) pada citra warna mewakili warna yang merupakan kombinasi dari tiga warna dasar yaitu merah, hijau dan biru yang dikenal sebagai citra RGB (*Red*, *Green*, *Blue*). Setiap warna dasar mempunyai intensitas sendiri dengan nilai maksimum 255 (8 bit).

Red = warna minimal putih, warna maksimal merah

Green = warna minimal putih, warna maksimal hijau

Blue = warna minimal putih, warna maksimal biru

Setiap titik pada citra warna membutuhkan data 3 *byte* Jumlah kemungkinan kombinasi warna untuk citra warna adalah 2^{24} = lebih dari 16 juta warna, disebut *true color* karena dianggap mencakup semua warna yang ada. Gambar 2.4 memperlihatkan contoh citra warna.



Citra warna

= 255 255 255 0 0 0 128 128 128 128 128 0

= 0 255 255 0 0 0 204 255 255 0 0 255

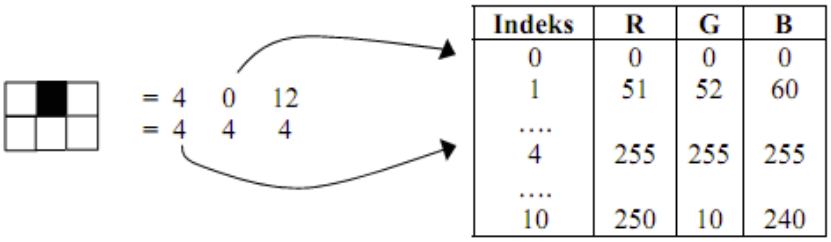
= 150 150 150 51 51 51 255 255 255 95 95 95

= 255 204 153 255 204 153 128 0 0 255 0 255

Gambar 2.5 Citra Warna (*True Color*)

4. Citra warna berindeks

Setiap titik (piksel) pada citra warna berindeks mewakili indeks dari suatu tabel warna yang tersedia (biasanya disebut palet warna). Keuntungan pemakaian palet warna adalah kita dapat dengan cepat memanipulasi warna tanpa harus mengubah informasi pada setiap titik dalam citra. Keuntungan yang lain, penyimpanan lebih kecil. Contoh citra warna berindeks diperlihatkan pada Gambar 2.5.

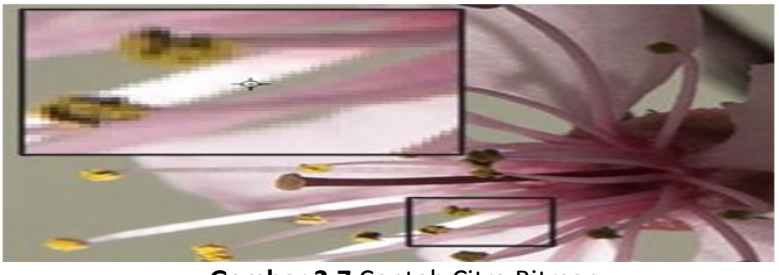


Gambar 2.6 Citra Warna Berindeks

Terdapat beberapa tipe file citra digital yang dikenal saat ini, diantaranya JPEG (*Joint Photographic Experts Group*), BMP (*Bitmap*), GIF (*Graphics Interchange Format*), TIFF (*Tagged Image File Format*), dan PNG (*Portable Network Graphics*). Dalam Tugas Akhir ini, citra digital yang digunakan adalah citra digital dengan tipe file JPEG, dan BMP.

2.3 *Bitmap Image File*

Bitmap(BMP) adalah representasi dari citra grafis yang terdiri dari susunan titik yang tersimpan di memori komputer. Dikembangkan oleh *Microsoft* dan nilai setiap titik diawali oleh satu *bit* data untuk gambar hitam putih, atau lebih bagi gambar berwarna. Ukuran sebenarnya untuk n-bit (2n warna) bitmap dalam byte dapat dihitung: ukuran file BMP , dimana tinggi dan lebar dalam pixel. Kerapatan titik-titik tersebut dinamakan resolusi, yang menunjukkan seberapa tajam gambar ini ditampilkan, ditunjukkan dengan jumlah baris dan kolom, contohnya 1024×768. Untuk menampilkan citra bitmap pada monitor atau mencetaknya pada printer, komputer menterjemahkan bitmap ini menjadi pixel (pada layar) atau titik tinta (pada printer).



Gambar 2.7 Contoh Citra Bitmap

Bitmap sesuai untuk dunia fotografi dan lukisan digital karena bitmap menghasilkan gradasi warna dengan sangat baik. Format Bitmap memiliki resolusi yang tidak bebas/ terbatas, bitmap me-representasi pixel dalam jumlah yang terbatas. Gambar berformat bitmap hanya tampil sempurna dalam ukuran aktualnya saja. Ketika di-skala, gambar bitmap akan terlihat kasar bergerigi dan kehilangan kualitas, pun ketika ter-display di monitor, dicetak dalam resolusi yang lebih tinggi melebihi resolusi aslinya. (http://id.wikipedia.org/wiki/BMP)

2.4 *JPEG Image File*

Joint Photographic Experts Group (JPEG) merupakan skema kompresi file bitmap. Awalnya, file yang menyimpan hasil foto digital memiliki ukuran yang besar sehingga tidak praktis. Dengan format baru ini, hasil foto yang semula berukuran besar berhasil dikompresi (dimampatkan) sehingga ukurannya kecil.

Citra JPEG merupakan tipe file citra digital yang banyak digunakan untuk menyimpan gambar-gambar dengan ukuran lebih kecil. Citra JPEG memiliki beberapa karakteristik, yaitu:

1. Memiliki ekstensi .jpg atau .jpeg.
2. Mampu menayangkan warna dengan kedalaman 24-bit true color.
3. Mengkompresi gambar dengan sifat lossy.
4. Umumnya digunakan untuk menyimpan gambar-gambar hasil foto

Gambar 2.8 Contoh Gambar JPEG

Standar kompresi file gambar yang dibuat oleh kelompok *Joint Photographic Experts Group* ini menghasilkan kompresi yang sangat besar tetapi dengan akibat berupa adanya distorsi pada gambar yang hampir selalu tidak terlihat. JPEG adalah sebuah format gambar, sangat berguna untuk membuat gambar jenis fotografi berkualitas tinggi dalam ukuran file yang sangat kecil. Format file grafis ini telah diterima oleh *Telecommunication Standardization Sector* atau ITU-T dan Organisasi Internasional untuk Standardisasi atau ISO. JPEG kebanyakan digunakan untuk melakukan kompresi gambar diam menggunakan analisis *Discrete Cosine Transform* (DCT).

Meskipun kompresi gambar JPEG sangatlah efisien dan selalu menyimpan gambar dalam kategori warna *true color* (24 bit), format ini bersifat *lossy*, yang berarti bahwa kualitas gambar dikorbankan bila tingkat kompresi yang dipilih semakin tinggi.

JPEG mendukung 16 juta warna. Jadi walaupun terjadi penurunan kualitas gambar, format ini sangat cocok untuk menggunakan pada penampilan gambar fotografi. Namun, dibandingkan format lain, browser membutuhkan waktu yang lebih lama untuk memuat file JPEG.(http://id.wikipedia.org/wiki/JPEG)

2.5 Resolusi Citra

Sebelum membahas mengenai masalah resolusi citra, sesuai judul penulisan Tugas Akhir ini perlu dijelaskan pengertian dari kata kualitas. Kualitas dapat didefenisikan sebagai suatu kondisi dinamis yang berhubungan dengan produk, jasa, manusia, proses dan lingkungan yang memenuhi atau melebihi harapan (Davis,Halaman 24, 2004). Adapun Resolusi Citra dapat didefenisikan sebagai banyaknya titik untuk setiap satuan panjang (*dot per inch*) yang terdapat dalam sebuah citra.

Resolusi Citra (image resolution) Dapat diartikan sebagai kualitas lensa yang dinyatakan dengan jumlah maksimum garis pada tiap milimeter yang masih dapat dipisahkan pada citra. Misal tiap garis tebalnya 0,01 mm. Ruang pemisah antara tiap garis juga sebesar 0,01 mm. Berarti tiap garis menempati ruang selebar 0,02 mm atau

pada tiap mm ada 50 garis. Dalam contoh ini berarti resolusi citranya sebesar 50 garis/mm. Secara teoritik maka resolusi citra yang terbaik 1.430 garis/mm. (Hestiningsih,Halaman 65, 2008).

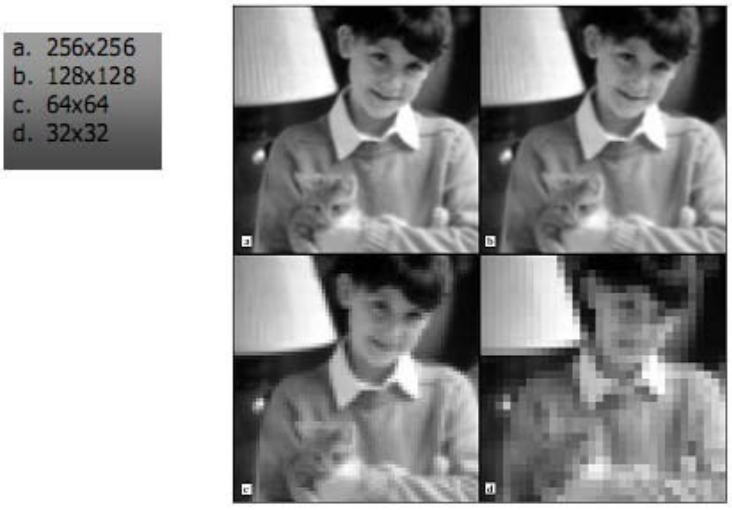
Menurut kualitas resolusi yang dihasilkan, resolusi dibedakan menjadi (www.total.or.id/info.php/kk=Resolusi):

1. Resolusi Tinggi (*High Resolution*).

Suatu citra dikatakan memiliki resolusi tinggi jika tingkat ketelitian yang cukup tinggi dari suatu media dalam menangkap ataupun menampilkan datanya.

2. Resolusi Rendah (*Low Resolution*).

Citra dikatakan memiliki resolusi rendah jika pada layar atau gambar yang teks dan grafiknya tampil dengan detail yang relatif kasar.



Gambar 2.9 *diatas memperlihatkan gambar suatu objek yang dicitrakan menggunakan beberapa kamera dengan tingkat kualitas resolusi yang berbeda.*

2.6 Operasi Bertetangga/Persekitaran (*Neighbourhood Operation*)

Sebuah citra dikatakan ideal, jika mampu mencerminkan kondisi sesungguhnya dari suatu objek. Mempunyai hubungan satu-satu (*one to one*), satu titik pada objek dipetakan tepat satu piksel di citra digital. Tetapi pada kenyataannya, hubungan yang ada antara titik dalam objek dengan titik pada citra digital adalah hubungan satu ke

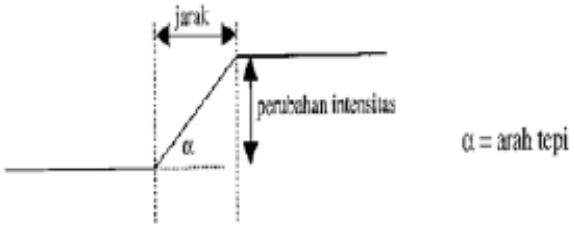
banyak (*one to many*) dan banyak ke satu (*many to one*). Hal ini disebabkan oleh beberapa hal, yaitu:

1. Sinyal yang dikirim oleh objek citra mengalami penyebaran (divergensi), sehingga yang diterima oleh sensor atau *detector* tidak lagi berupa suatu titik, namun berupa luasan.
2. Atau sebaliknya satu titik pada sensor atau *detector* dapat menerima banyak sinyal dari beberapa bagian. Operasi citra digital yang berhubungan dengan kondisi diatas disebut operasi persekitaran/bertetangga (*neighborhood operation*). Operasi persekitaran/bertetangga

Pada dasarnya adalah hubungan antara citra dengan sebuah filter (mask/kernel). Nilai dari filter/mask merupakan bobot kontribusi titik persekitaran terhadap operasi persekitaran. Ada beberapa operasi pengolahan citra yang berkaitan dengan operasi bertetangga ini, diantaranya adalah deteksi tepi. Untuk pembahasan mengenai deteksi tepi akan dibahas pada sub-bab berikutnya. (Afnisyah, Halaman 67, 2009)

2.7 Deteksi Tepi (*Edge Detection*)

Secara umum tepi dapat didefinisikan sebagai batas antara dua *region* (dua piksel yang saling berdekatan) yang memiliki perbedaan intensitas yang tajam atau tinggi. Tepi dapat diorientasikan dengan suatu arah, dan arah ini berbeda-beda, tergantung Pada perubahan intensitas. Untuk lebih memahami defenisi tepi, gambar dibawah ini memperlihatkan model tepi dalam ruang satu dimensi (Febriani, 2008).

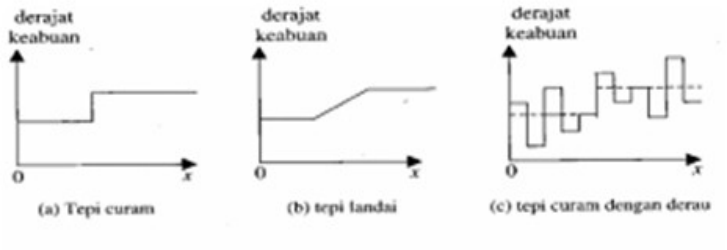


Gambar 2.10 Model Tepi Satu Dimensi

Ada tiga macam tepi yang terdapat di dalam citra digital (Munir, Halaman 35, 2004),, yaitu:

1. Tepi curam jenis tepi ini terbentuk karena perubahan intensitas yang tajam, berkisar 90^0.
2. Tepi landai Tepi lebar, sudut arah kecil. Terdiri dari sejumlah tepi-tepi local yang lokasinya berdekatan.
3. Tepi yang mengandung *noise*

   Untuk mendeteksi tepi jenis ini, biasanya dilakukan operator *image enhancement* terlebih dahulu. Misalnya Operator Gaussian yang berfungsi untuk menghaluskan citra. Perbedaan ketiga macam tepi tersebut, diperlihatkan pada Gambar 2.9.

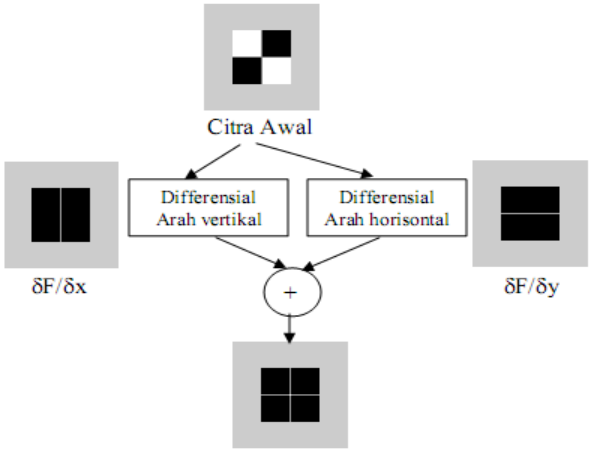


Gambar 2.11 Jenis-jenis Tepi

Deteksi tepi merupakan langkah pertama untuk melingkupi informasi di dalam citra. Tepi mencirikan batas-batas objek dan karena itu tepi berguna untuk proses segmentasi dan identifikasi objek di dalam citra. Deteksi tepi pada suatu citra memiliki tujuan sebagai berikut (Sigit, Halaman 79, 2005):

1. Menandai bagian yang menjadi detil citra.
2. Memperbaiki detil citra yang kabur karena *error* atau efek proses akuisisi.

Gambar 2.9 memperlihatkan bagaimana tepi dari suatu citra dapat diperoleh dengan operasi pendeteksian tepi.



Gambar 2.12 Proses Deteksi Tepi Citra

Berdasarkan prinsip-prinsip filter pada citra, tepi suatu gambar dapat diperoleh menggunakan *High Pass Filter* (HPF), dengan karakteristik:

$$\sum\sum H(x, y) = 0 \qquad (2.2)$$

Berikut ini beberapa metode yang digunakan untuk mendeteksi tepi (Herdiyeni, 2007), yaitu:

1. *First-Order Derivative Edge Detection* (Pendeteksi Tepi Turunan Pertama).

   Pendeteksi tepi ini menghitung perbedaan intensitas antara dua piksel yang saling berdekatan, dimana daerah tepi terletak pada nilai maksimum lokalnya. Metode ini sering juga disebut dengan pendeteksi tepi dengan operator gradient citra. Berikut ini berapa contoh pendeteksi tepi turunan pertama yang sering digunakan:

   a. Metode Roberts-*Cross*
   b. Metode Prewitt
   c. Metode Sobel

2. *Second-Order Derivative Edge Detection* (Pendeteksi Tepi Turunan Kedua).

   Pendeteksitepi turunan kedua, memanfaatkan nilai tururnan kedua dari fungsi

Gaussian dalam langkah-langkah untuk mendeteksi tepi dari suatu citra. Yang termasuk dalam metode pendeteksi tepi ini, adalah:

a. Metode *Laplacian of Gaussian*

b. Metode Canny

Pada penulisan Tugas Akhir ini metode yang digunakan adalah metode yang menggunakan operator turunan pertama (Sobel) dan operator turunan kedua (*Laplacian of Gaussian* dan Canny).

2.8 Konvolusi / Filter

Deteksi tepi merupakan salah satu proses pengolahan citra yang menggunakan filter atau penapis. Selain pada proses pendeteksian tepi, proses lain pada pengolahan citra yang juga menggunakan filter atau penapis adalah perbaikan kualitas citra (*image enhancement*), penghilangan derau (*noise reduction*), mengurangi erotan, penghalusan atau pelembutan citra (*image smoothing*), dan lain-lain.

Untuk mengaplikasikan penapis pada citra, digunakan metode konvolusi. Konvolusi bisa dinyatakan dalam matriks, dimana setiap elemen matriks penapis tersebut dinamakan koefisien konvolusi. Operasi konvolusi bekerja dengan menggeser kernel piksel per piksel, yang hasilnya kemudian disimpan dalam matriks baru. Konvolusi 2 fungsi f(x) dan g (x) diperlihatkan dengan rumus berikut ini:

$$f\ (x) * \ g\ (x) = \ \int_{-\infty}^{\infty} f\ (\alpha) g(x-a) da \qquad (2.3)$$

Atau fungsi diskritnya :

$$f(x) * \ g(x) = \ \sum f(a) g(x-a) \qquad (2.4)$$

Dimana α = Peubah bantu

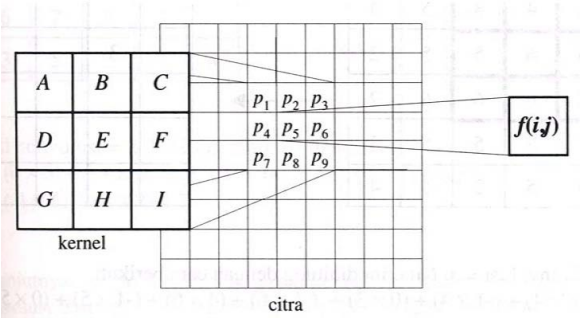
Pada konvolusi 2D, fungsi malarnya dapat dihitung dengan persamaan:

$$f(x,y) * g(x,y) = \ \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(a,b) g(x-a, y-b) da db \qquad (2.5)$$

Sedangkan fungsi diskritnya dihitung dengan persamaan:

$$f(x,y) * g(x,y) = \ \sum_{a=-\infty}^{\infty} \sum_{b=-\infty}^{\infty} f(a,b) g(x-a, y-b) \qquad (2.6)$$

g(x) merupakan *convolution mask* / filter / kernel atau *template*. memperlihatkan ilustrasi terjadinya konvolusi.



Gambar 2.13 Proses Konvolusi

Dimana:

$$f(i,j) = Ap_1 + Bp_2 + Cp_3 + Dp_4 + Ep_5 + Fp_6 + Gp_7 + Hp_8 + Ip_9 \qquad (2.7)$$

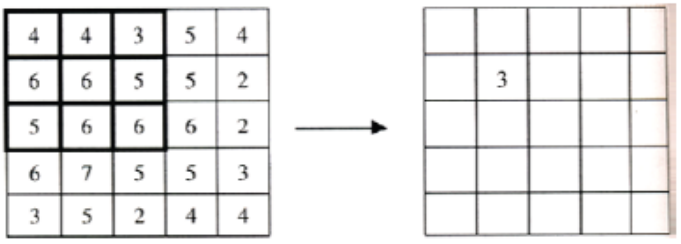
Untuk lebih jelasnya, berikut contoh konvolusi yang terjadi antara citra f(x,y) berukuran 5x5 dengan sebuah kernel berukuran 3x3 yang diperlihatkan pada Gambar 2.14

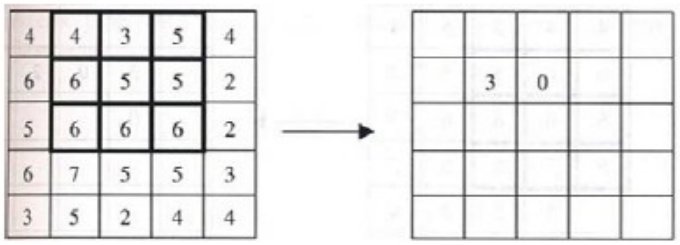
$$f(x,y) = \begin{bmatrix} 4 & 4 & 3 & 5 & 4 \\ 6 & 6 & 5 & 5 & 2 \\ 5 & 6 & 6 & 6 & 2 \\ 6 & 7 & 5 & 5 & 3 \\ 3 & 5 & 2 & 4 & 4 \end{bmatrix} \qquad g(x,y) = \begin{bmatrix} 0 & -1 & 0 \\ -1 & \bullet 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

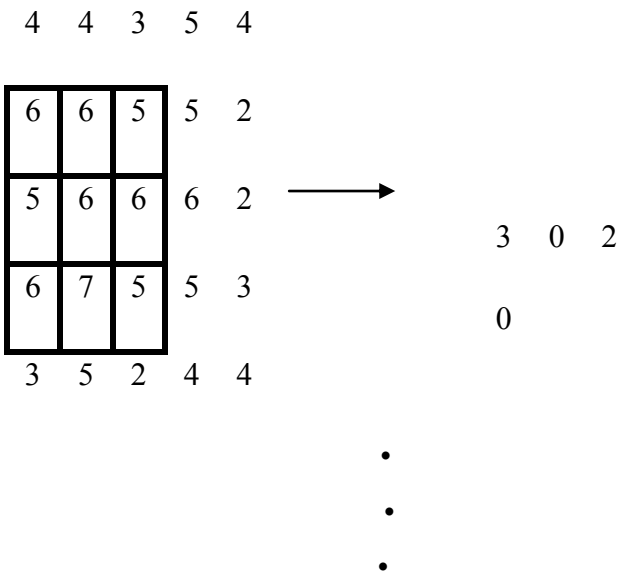
Gambar 2.14 Matriks Citra dan Kernel Sebelum Konvolusi

Tanda • menunjukkan posisi (0,0) dari kernel

Tahapan untuk mendapatkan hasil konvolusi yang terjadi antara citra dan kernel diatas dapat dilihat pada Gambar.







Dan seterusnya

Gambar 2.15 Tahapan Proses Pembentukan Konvolusi

Sehingga diperoleh hasil akhir dari proses konvolusi tersebut, yang ditunjukkan pada Gambar 2.16

	4	0	8	
	0	2	6	
	6	0	2	

Gambar 2.16 Hasil Konvolusi Citra dan Kernel

Dalam konvolusi terdapat dua kemungkinan yang jika ditemukan, diselesaikan dengan cara berikut, yaitu:

1. Untuk hasil konvolusi dengan nilai negatif, nilainya dijadikan nol (0).
2. Jika hasil konvolusi lebih besar (>) derajat keabuan maksimum, maka nilai diubah menjadi derajat keabuan maksimum.

2.9 Pendeteksi Tepi Sobel

Operator Sobel melakukan perhitungan secara 2D terhadap suatu ruang di dalam sebuah gambar dengan harapan nantinya akan nampak daerah-daerah bernilai tinggi pada gambar tersebut yang merupakan deteksi tepi dari suatu gambaran. Operator ini biasanya digunakan untuk mencari gradien dari masing-masing piksel gambar input yang telah di *grayscale* sebelumnya. Secara teori, diperlukan matriks setidaknya berukuran 3x3 sebagai kernelnya. Contoh kernel Sobel yang berukuran 3x3 diperlihatkan pada Gambar 2.15.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

-1	+2	+1
0	0	0
-1	-2	-1

Gy

Gambar 2.17 Kernel Konvolusi Sobel

Kernel ini dirancang untuk menyelesaikan permasalahan deteksi tepi baik secara vertikal maupun horisontal. Penggunaan kernel-kernel ini dapat digunakan bersamaan ataupun secara terpisah. Apabila digunakan kernel vertikal dan kernel horisontal secara bersamaan, maka gradien dapat diukur dengan formula sebagai berikut:

$$G = Gx2 + Gy2 \qquad (2.8)$$

Besarnya gradien juga dapat dihitung lebih cepat lagi dengan menggunakan formula sebagai berikut:

$$G = Gx + Gy \qquad (2.9)$$

Gradien tersebut pasti mempunyai derajat kemiringan tertentu. Untuk dapat mengetahui sudut dari gradien tersebut dapat dihitung dengan menggunakan rumus sebagai berikut:

$$\vartheta = \arctan(Gy / Gx) \qquad (2.10)$$

Dalam kasus ini, orientasi 0 untuk menentukan nilai kontras maksimum dari hitam ke putih dihitung dari kiri menuju kanan dan berjalan terus sampai ke bagian paling atas dari suatu gambaran. Sedangkan sudutnya diukur berlawanan arah jarum jam.

Yang sering terjadi dalam filter ini adalah nilai mutlak dari *magnitude* hanya sebatas penglihatan mata kita saja, apabila terdapat dua komponen gradien maka akan digabungkan menggunakan operator *pseudo-convolution* seperti yang diperlihatkan pada Gambar 2.18.

P1	P2	P3
P4	P5	P6
P7	P8	P9

Gambar 2.18 Kernel Pseudo-Convolution

Dengan menggunakan kernel *pseudo-convolution* seperti diatas maka perhitungan yang terjadi adalah:

$$|G| = |(P_1 + 2P_2 + P_3) - (P_7 + P_8 + P_9)| + (P_3 + 2P_6 + P_9) - (P_1 + 2P_4 + P_7)| \qquad (2.11)$$

(Afnisyah,, Halaman 97,2009)

Berikut ini *pseudocode* dari algoritma metode Sobel:

```
% Mengubah citra RGB menjadi citra skala keabuan
  I = double(rgb2gray(handles.data1));

% Konvolusi dengan kernel sobel 3x3
  sobelhor = [-1 0 1;-2 0 2;-1 0 1];
```

```
  sobelver = [-1 -2 -1;0 0 0;1 2 1];
  Ix = conv2(I,sobelhor,'same');
  Iy = conv2(I,sobelver,'same');
  J = sqrt((Ix.^2)+(Iy.^2));

% Pengambangan
  thresh = alfa*mean2(abs(J));
  If J > thresh
    edgeimage = 1;
  else
    edgeimage = 0;
  end

% Mengembalikan nilai matriks menjadi nilai intensitas citra
  edgeimage = mat2gray(edgeimage);
```

Pada citra masukan dengan tingkat resolusi yang tinggi, nilai pengambangan ini akan mempengaruhi tepi-tepi yang terbentuk, karena banyak piksel *noise* yang akan terdeteksi sebagai tepi. Artinya, citra masukan dengan tingkat resolusi yang tinggi (6 MegaPiksel) akan menghasilkan lebih banyak piksel tepi dibanding citra masukan dengan tingkat resolusi rendah (2 MegaPiksel, dan 4 MegaPiksel). Namun, piksel yang didefenisikan sebagai tepi pada citra masukan dengan tingkat resolusi tinggi tersebut sebahagiannya adalah berupa *noise*

Gambar 2.19 citra hasil tepi metode Sobel

2.10 Pendeteksi Tepi *Laplacian of Gaussian* (LoG)

Laplacian adalah salah satu metode pendeteksi tepi yang merupakan pendeteksi tepi turunan kedua. Turunan kedua memiliki sifat lebih sensitif terhadap *noise*, selain itu juga menghasilkan *double edge*. Oleh karena itu, operator Laplacian dalam deteksi tepi pada umumnya tidak dipergunakan secara langsung, namun dikombinasikan dengan suatu kernel

Gaussian menjadi sebuah operator *Laplacian of Gaussian* (LoG). Metode ini mendeteksi tepi lebih akurat khususnya pada tepi yang curam. Pada tepi yang curam, turunan keduanya memiliki *zero-crossing* (persilangan nol), yaitu titik dimana terdapat pergantian tanda nilai turunan kedua, sedangkan pada tepi yang landai tidak terdapat persilangan nol (Gonzalez *et a,l Halaman 48,* 2005). Berikut ini fungsi LoG 2-D yang berpusat pada titik 0 dan dengan deviasi standard Gaussian σ

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1-\frac{x^2+y^2}{2\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}} \qquad (2.12)$$

Representasi turunan kedua dalam bentuk kernel operator Laplacian diperlihatkan pada Gambar 2.14 (Afnisyah, Halaman 98,2009).

0	-1	0
-1	4	-1
0	-1	0

Gambar 2.20 Kernel Konvolusi Laplacian

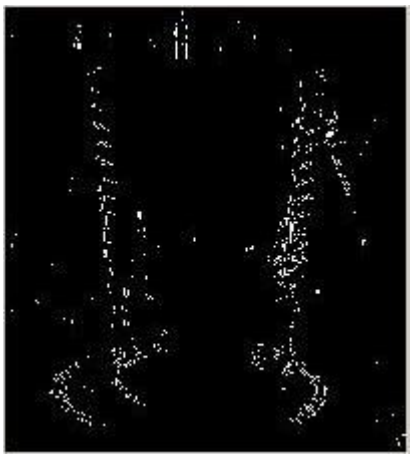
Berikut ini *pseudocode* dari algoritma metode LoG, yang dapat digunakan untuk menganalisis kinerja metode ini terhadap parameter yang ditentukan:

```
% Mengubah citra RGB menjadi citra skala keabuan
I = double(rgb2gray(handles.data1));

% Penghalusan
% Arah X
```

```
filterx = d2dgauss(Nx1,Sigmax1,Nx2,Sigmax2,Theta1);
Ix = conv2(I,filterx,'same');
% Arah Y
filtery = d2dgauss(Ny1,Sigmay1,Ny2,Sigmay2,Theta2);
Iy = conv2(I,filtery,'same');

% Norm of the gradient
NVI = sqrt(Ix.*Ix+Iy.*Iy);

% Konvolusi dengan kernel laplacian 3x3
H = [0 1 0,1 -4 1,0 1 0];
J = conv2(NVI,H,'same');

% Find the zerocross with thresh value
thresh = alfa.*mean2(abs(J));
edgeimage = edge(J,'zerocross',thresh);
```

Metode LoG sangat sensitif terhadap noise, hal ini disebabkan oleh proses konvolusi citra dengan kernel laplacian. Akibat hal ini, tentunya juga mempengaruhi nilai sensitivity rate dari keluaran tiap citra masukan untuk metode LoG yang bergantung pada jumlah piksel yang dinyatakan sebagai tepi pada citra hasil deteksi tepi (nR). Seperti yang telah disebutkan pada subbab analisis sensitivity rate untuk metode Sobel, besarnya jumlah piksel yang dinyatakan sebagai tepi pada citra hasil deteksi tepi akan membuat jumlah piksel yang dinyatakan sebagai tepi pada citra hasil deteksi tepi yang diberi noise juga semakin besar, dan nilai sensitivity rate juga semakin besar.

Dari penjelasan tersebut, maka dapat diasumsikan bahwa Citra_asli1 memiliki nilai sensitivity rate lebih besar dari Citra_asli3, Citra_asli2, dan Citra_asli1. Citra_asli3 memiliki nilai sensitivity rate lebih besar dari Citra_asli2 dan Citra_asli1, dan Citra_asli2 memiliki nilai sensitivity rate yang lebih besar dari Citra_asli1. (Afnisyah, Halaman 99, 2009)



Gambar 2.21 citra hasil tepi metode LoG

2.11 Pendeteksi Tepi Canny

Canny merupakan salah satu algoritma deteksi tepi modern. Pada tahun 1986 John Canny mengusulkan tiga kriteria yang menjadi basis pengembangan filter untuk mengoptimalkan pendeteksian tepi pada citra ber*noise* (Febriani, 2008). Tiga criteria tersebut adalah:

a. *Good detection,* kriteria ini bertujuan memaksimalkan nilai *signal to noise ration* (SNR) sehingga semua tepi dapat terdeteksi dengan baik atau tidak ada yang hilang.

b. *Good localisation,* tepi yang terdeteksi berada pada posisi yang sebenarnya, atau dengan kata lain bahwa jarak antara posisi tepi yang terdeteksi oleh detektor dengan posisi tepi sebenarnya adalah seminimum mungkin (idealnya adalah 0).

c. *Low multiplicity of the response* atau ”*one response to single edge*” detector tidak memberikan tepi yang bukan tepi sebenarnya. Berdasarkan pada kriteria ini, Canny berhasil melakukan optimalisasi dari ke tiga kriteria tersebut dan menghasilkan persamaan sebagai berikut:

$$h(x)=a_1 e^{\alpha}\cos(\omega x)+a_2 e^{\alpha}\sin(\omega x)+ a_3 e^{-\alpha}\cos(\omega x)+ a_4 e^{-\alpha}\sin(\omega x) \qquad (2.13)$$

Berikut adalah contoh dari 5x5 filter Gaussian, digunakan untuk membuat gambar ke kanan, dengan $\sigma = 1{,}4$:

$$\mathbf{B} = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * \mathbf{A}. \quad (2.14)$$

Namun persamaan ini cukup sulit untuk diimplementasikan, sehingga pada implementasinya, Canny tetap menggunakan filter Gaussian untuk mereduksi *noise*. Metode Canny ini tidak memiliki operator khusus, namun metode ini terdiri dari beberapa langkah khusus. Metode Canny akan mendeteksi tepi dengan mencari nilai gradien maksimal lokal dari sebuah citra (Green, Halaman 57, 2009). Gradien tersebut dihitung menggunakan turunan dari Gaussian filter (Gonzalez, *et al Halaman 75,* 2005). Metode Canny menggunakan dua *thresholds* atau pengambangan, yang berguna untuk mendeteksi tepian yang terlihat jelas, dan tepian yang kurang jelas atau lemah, termasuk juga tepian yang kurang jelas yang terlihat pada output yang terhubung dengan tepian yang jelas. Pembahasan mengenai *thresholds* atau pengambangan akan dijelaskan pada sub-bab berikutnya.

Metode ini lain dengan metode sebelumnya, yang sebelum dilakukan proses deteksi tepi harus dilakukan terlebih dahulu proses penghilangan *noise*. Metode Canny lebih utama akan mendeteksi tepian yang kurang jelas, yang tidak dapat diperoleh dengan menggunakan metode lain dan dilanjutkan dengan penghitungan turunan pertama dan *thresholding hysteresis*. (Afnisyah, Halaman 101, 2009)

Berikut ini *pseudocode* dari algoritma metode Canny yang digunakan untuk menganalisis kinerja metode Canny terhadap parameter-parameter yang telah ditentukan:

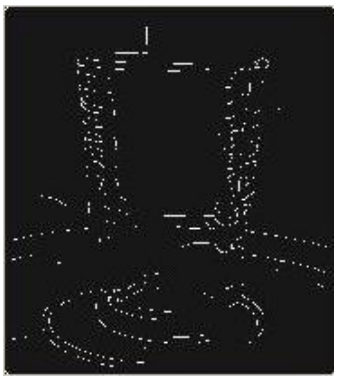
```
% Mengubah citra RGB menjadi citra skala keabuan
I = double(rgb2gray(handles.data1));


% Penghalusan
% Arah X
```

```
filterx = d2dgauss(Nx1,Sigmax1,Nx2,Sigmax2,Theta1);
Ix = conv2(I,filterx,'same');
% Arah Y
filtery = d2dgauss(Ny1,Sigmay1,Ny2,Sigmay2,Theta2);
Iy = conv2(I,filtery,'same');

% Norm of Gradien
NVI = sqrt(Ix.*Ix+Iy.*Iy);

% Pengambangan
I_max = max(max(NVI));
I_min = min(min(NVI));
level = alfa*(I_max-I_min)+I_min;
Ibw = max(NVI,level.*ones(size(NVI)));

% Peredaman Titik Non Maksimum
[n,m] = size(Ibw);
for i = 2:n-1
for j = 2:m-1
if Ibw(i,j)>level
X = [-1,0,+1;-1,0,+1;-1,0,+1];
Y = [-1,-1,-1;0,0,0;+1,+1,+1];
      Z = [Ibw(i-1,j-1),Ibw(i-1,j),Ibw(i-1,j+1);
            Ibw(i,j-1),Ibw(i,j),Ibw(i,j+1);
            Ibw(i+1,j-1),Ibw(i+1,j),Ibw(i+1,j+1)];
      XI = [Ix(i,j)/NVI(i,j), -Ix(i,j)/NVI(i,j)];
      YI = [Iy(i,j)/NVI(i,j), -Iy(i,j)/NVI(i,j)];
      ZI = interp2(X,Y,Z,XI,YI);
            if Ibw(i,j) >= ZI(1) & Ibw(i,j) >= ZI(2)
                  edgeimage(i,j)=I_max;
            else
                  edgeimage(i,j)=I_min;
            end
```

```
      else
            edgeimage(i,j)=I_min;
      end
end
end
```

Berbeda dengan metode Sobel dan LoG, jika pada metode-metode tersebut nilai sensitivity rate dari tiap citra masukan akan semakin besar untuk citra dengan kualitas resolusi yang juga semakin besar, pada metode Canny hal itu tidak akan terjadi. Tahap-tahap pada metode Canny yang sangat kompleks membuat metode ini memiliki ketahanan yang cukup tinggi terhadap noise. Berbeda dengan metode LoG, karena kesensitivitasannya yang tinggi terhadap noise membuat metode ini mendefenisikan piksel yang merupakan noise tersebut sebagai sebuah tepi. Dalam metode Canny, dengan langkah peredaman titik lokal non mkasimum, metode ini berhasil membedakan mana piksel yang memang sebagai tepi dan mana piksel yang bukan tepi.

Dengan langkah-langkah pendeteksian tepi yang digunakan pada pendeteksian tepi metode Canny, metode bekerja maksimal pada citra masukan dengan tingkat resolusi yang rendah, karena memiliki nilai intensitas yang lebih tinggi untuk setiap pikselnya dibanding citra dengan tingkat resolusi tinggi. Hal ini menyebabkan jumlah piksel tepi yang terdeteksi pada citra dengan resolusi tinggi akan lebih sedikit dibanding dengan jumlah piksel yang terdeteksi sebagai tepi pada citra dengan kualitas resolusi rendah. Dan ini artinya nilai sensitivity rate untuk citra dengan resolusi rendah (Citra_asli1) akan lebih besar dibanding dengan nilai sensitivity rate untuk citra masukan yang memiliki tingkat resolusi tinggi (Citra_asli3). (Afnisyah, Halaman 103, 2009)



Gambar 2.22 citra hasil tepi metode Canny

2.12 Pengambangan (*Thresholding*)

Operasi pengambangan (*thresholding*) merupakan salah satu operasi yang termasuk kedalam operasi titik dalam pengolahan citra digital. Operasi ini digunakan untuk mengubah citra dengan format skala keabuan, yang mempunyai kemungkinan nilai lebih dari 2 ke citra biner yang memiliki 2 buah nilai (yaitu 0 dan 1).

Pengambangan terbagi atas dua jenis (Afnisyah, Halaman 105, 2009), yaitu:

a. Pengambangan Tunggal

Pengambangan tunggal merupakan proses pengambangan yang hanya memiliki sebuah nilai batas ambang.

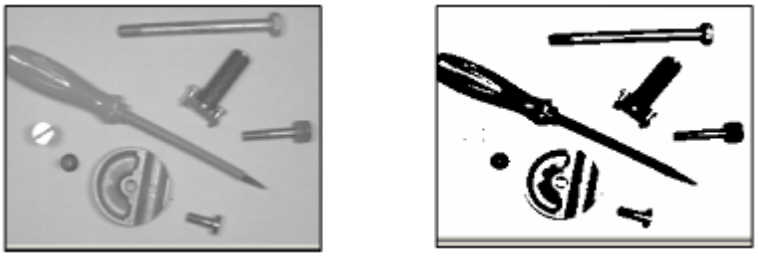
Fungsi GST-nya:

$$Ko = \begin{cases} 0, \text{ jika } Ki < \text{ambang} & (0 = \text{hitam}) \\ 1, \text{ jika } Ki \geq \text{ambang} & (1 = \text{putih}) \end{cases}$$

atau:

$$Ko = \begin{cases} 0, \text{ jika } Ki \geq \text{ambang} \\ 1, \text{ jika } Ki < \text{ambang} \end{cases}$$

Gambar 2.23 menunjukkan sebuah citra yang telah mengalami pengambangan tunggal.



Gambar 2.24 Citra dengan Pengambangan Tunggal

b. Pengambangan Ganda

Memiliki ambang bawah dan ambang atas. Dilakukan untuk menampilkan titik-titik yang mempunyai rentang nilai skala keabuan tertentu.

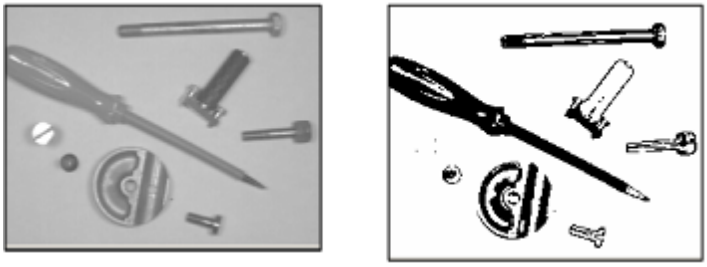
Dengan fungsi GST:

$$Ko = \begin{cases} 0, & \text{jika ambang bawah} \leq Ki \leq \text{ambang atas} \\ 1, & \text{lainnya.} \end{cases}$$

atau:

$$Ko = \begin{cases} 1, & \text{jika ambang bawah} \leq Ki \leq \text{ambang atas} \\ 0, & \text{lainnya.} \end{cases}$$

Gambar 2.25 memperlihatkan sebuah citra yang telah mengalami pengambangan tunggal.



Gambar 2.26 Citra dengan Pengambangan Ganda

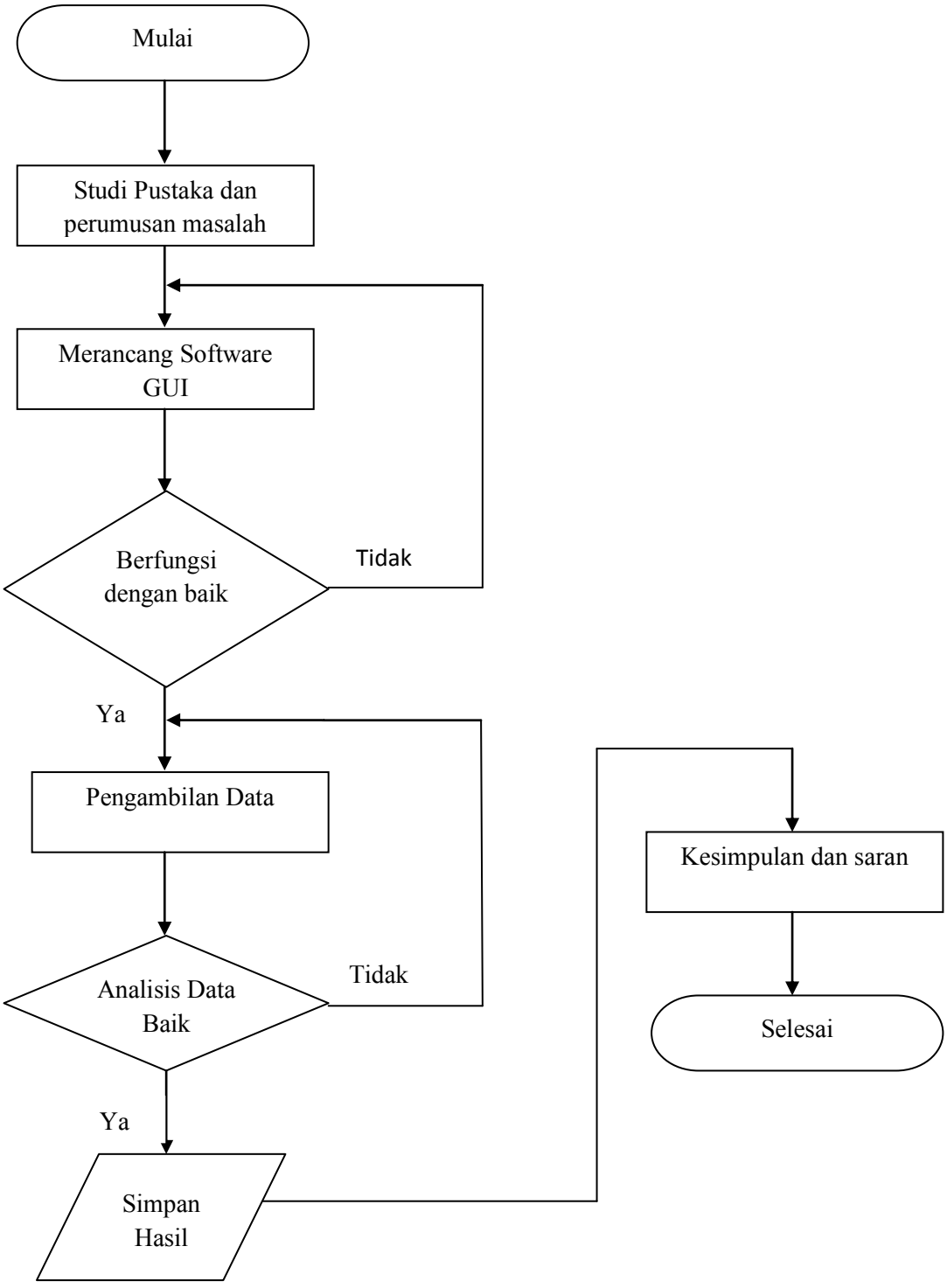
BAB III
METODELOGI PENELITIAN

3.1 Susunan Metodelogi Penelitian

Untuk dapat mencapai tujuan penulisan tugas akhir ini, perlu adanya suatu metodelogi, metodelogi yang dilakukan adalah sebagai berikut:

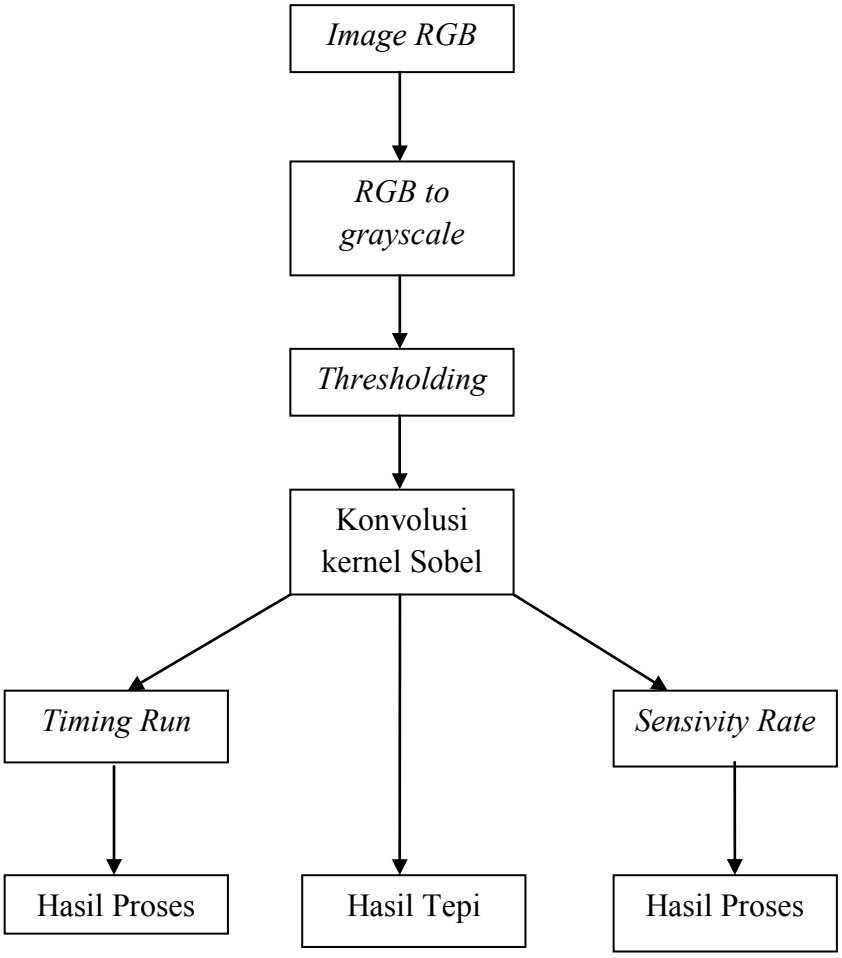
1. Proses pengumpulan data dan referensi baik dari buku, artikel, jurnal, makalah, atau situs internet yang berkaitan dengan penulisan Tugas Akhir ini.
2. Proses akuisisi data merupakan proses pengambilan data citra, yang akan digunakan pada proses pengujian (*testing*). Dengan menggunakan file format JPEG dan BMP. Dengan resolusi 2MP (*Mega Pixel*s), 4MP dan 6MP.
3. Melakukan perancangan terhadap sistem yang akan dibangun, mulai dari perancangan untuk *user interface* dan juga *list* program untuk deteksi tepi dengan metode filter Sobel, LoG, dan Canny.
4. Melakukan implementasi program yang telah ada ke dalam suatu bentuk perangkat lunak untuk mendeteksi tepi citra.
5. Dari hasil implementasi dilanjutkan dengan proses pengujian terhadap sistem yang telah selesai dibangun dan menganalisis hasil keluarannya.
6. Setelah semua tahap dalam penelitian selesai dilakukan, seluruh hasil yang diperoleh akan didokumentasikan dalam bentuk laporan.

3.2 Diagram Alur Penelitian



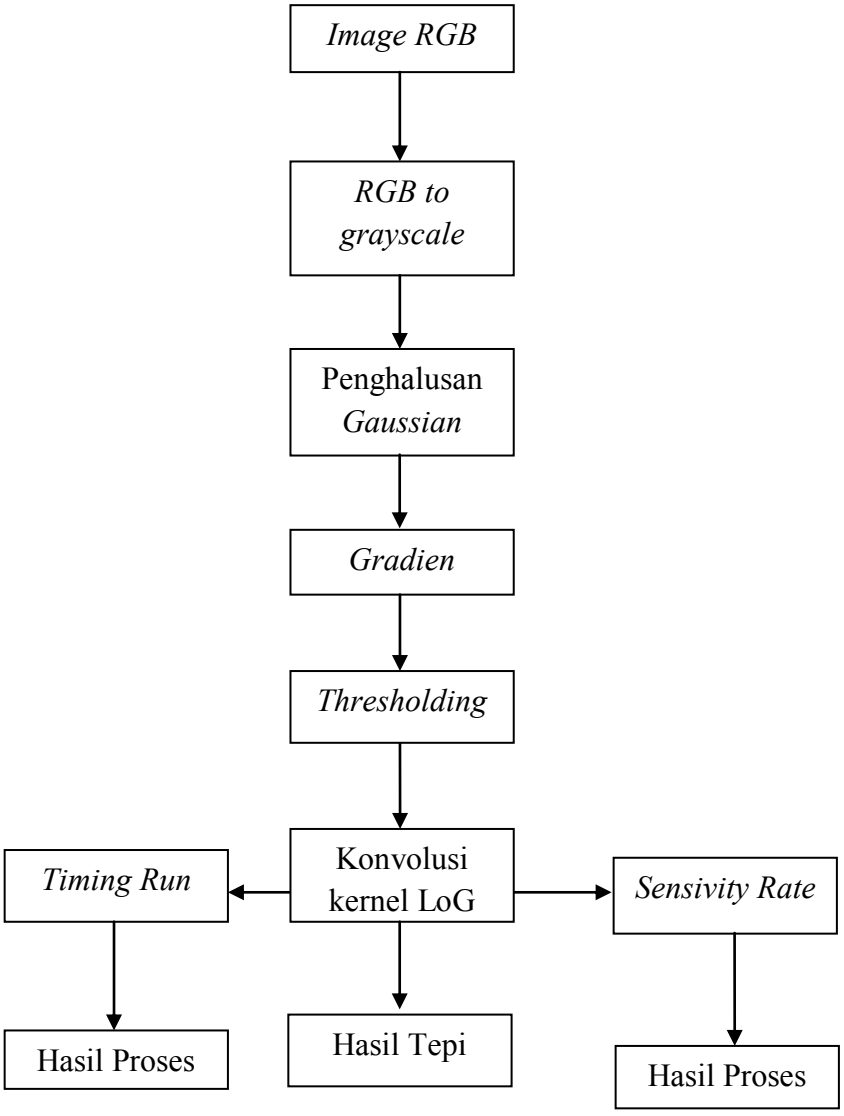
Gambar 3.1 Diagram Alur Penelitian

3.3 Blok Sistem Metode Sobel



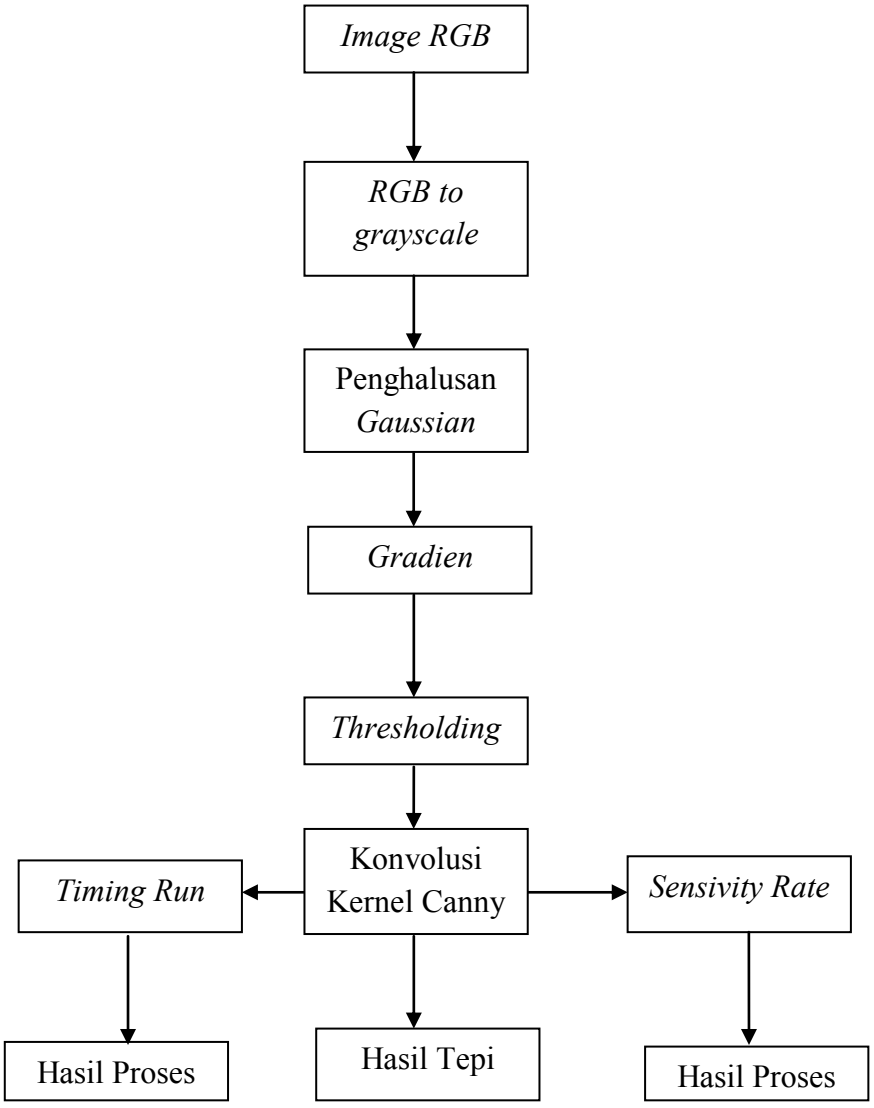
Gambar 3.2 Blok Sistem Deteksi Tepi

3.4 Blok Sistem Metode LoG



Gambar 3.3 Blok Sistem Metode LoG

3.4 Blok Sistem Metode Canny



Gambar 3.4 Blok Sistem Metode Canny

3.5 Parameter Pembanding

Dalam pengujian kehandalan suatu metode pendeteksi tepi, berikut ini beberapa parameter yang dapat digunakan:

1. Kualitas *edge* yang dihasilkan atau morfologi garis tepi yang terbentuk. Suatu metode pendeteksi tepi dikatakan baik jika metode tersebut berhasil mendeteksi tepi dengan tepat, artinya tidak menyatakan suatu piksel yang bukan tepi sebagai tepi atau sebaliknya, tidak menganggap suatu tepi sebagai tepian. (Indira,2008).

2. Sensitivitas (ketahanan) suatu metode pendeteksi tepi terhadap *noise.* Sensitivitas metode pendeteksi tepi terhadap *noise* dapat diukur dengan menggunakan parameter *error rate* sebagai berikut:

$$P = \frac{|n_N - n_R|}{n_R} \qquad (3.1)$$

dimana:

nR : jumlah piksel yang dinyatakan sebagai tepi pada citra referensi

nN : jumlah piksel yang dinyatakan sebagai tepi pada citra *noisy*

Nilai P yang besar menyatakan sensitivitas *edge detector* yang tinggi terhadap *noise*. Jadi Semakin besar Sensivity rate yang dihasilkan daya tahan citra terhadap Noise menjadi lebih besar, bisa dibilang sedikit noise yang didapat pada citra yang memiliki *Sensivity rate* yang besar. (Febriani, Halaman 110, 2008, Indira, Halaman 89, 2008)

3. Waktu yang diperlukan dalam proses tersebut (*timing run).*

Timing-run adalah lama waktu proses deteksi tepi pada suatu citra, jadi pada tahap ini akan dianalisis lama waktu yang digunakan dalam melakukan proses deteksi tepi. Dalam penelitian ini, waktu pemrosesan dihitung dengan rumus:

Waktu proses = waktu akhir deteksi tepi – waktu awal deteksi tepi (Afnisyah, Halaman 112, 2009)

4. Dua format berbeda JPEG (*Joint Photographic Experts Group*) dan BMP (*Bitmap*). Mengukur perbandingan pada kedua jenis file format bergambar yang berbeda, sejauh mana pengaruh yang dihasilkan deteksi tepi pada perbedaan format gambar ini.

BAB IV
HASIL DAN PEMBAHASAN

4.1 ANALISIS DAN PERANCANGAN SISTEM

Berikut ini adalah tahap analisis terhadap ketiga metode pendekteksian tepi Sobel, LoG, dan Canny. Analisis ini akan melihat bagaimana setiap langkah dari tiap-tiap metode dapat mempengaruhi parameter-parameter pembanding yang telah ditentukan. Setelah melakukan analisis,dilanjutkan dengan tahap perancangan yang dilakukan sebelum tahap implementasi.

4.1.1 Analisis Metode Sobel

Untuk menganalisis kinerja metode Sobel dalam mendeteksi tepi, digunakan empat jenis citra dengan tingkat kualitas resolusi yang berbeda (2 megapiksel, 4 megapiksel, 6 megapiksel),dan terhadap empat buah parameter yang dianggap mampu mengukur kinerja metode ini, yaitu kualitas citra tepi yang dihasilkan, *timing run, sensitivity rate.* Berikut ini *pseudocode* dari algoritma metode Sobel:

Langkah 1

```
% Mengubah citra RGB menjadi citra skala keabuan
  I = double(rgb2gray(handles.data1));
```

Langkah 2

```
% Konvolusi dengan kernel sobel 3x3
  sobelhor = [-1 0 1;-2 0 2;-1 0 1];
  sobelver = [-1 -2 -1;0 0 0;1 2 1];
  Ix = conv2(I,sobelhor,'same');
  Iy = conv2(I,sobelver,'same');
  J = sqrt((Ix.^2)+(Iy.^2));
```

Langkah 3

```
% Pengambangan
  thresh = alfa*mean2(abs(J));
```

```
If J > thresh
  edgeimage = 1;
else
  edgeimage = 0;
end
```

Langkah 4

```
% Mengembalikan nilai matriks menjadi nilai intensitas citra
  edgeimage = mat2gray(edgeimage);
```

Dari algoritma tersebut, maka hasil analisis kinerja metode Sobel terhadap ketiga parameter yang telah ditentukan, adalah sebagai berikut:

4.1.1.1 Analisis Kualitas Citra Tepi

Karena dalam penelitian ini digunakan citra masukan dengan beberapa tingkat kualitas resolusi citra, maka nilai alfa yang digunakan harus disesuaikan agar nilai tersebut dapat digunakan pada keempat jenis citra masukan, yaitu Citra_asli_Gelas1, Citra_asli_Gelas2, dan Citra_asli_Gelas3(objek Gelas kaca dan Botol Plastik).

Pada citra masukan dengan tingkat resolusi yang tinggi, nilai pengambangan ini akan mempengaruhi tepi-tepi yang terbentuk, karena banyak piksel *noise* yang akan terdeteksi sebagai tepi. Artinya, citra masukan dengan tingkat resolusi yang tinggi (Citra_asli3) akan menghasilkan lebih banyak piksel tepi dibanding citra masukan dengan tingkat resolusi rendah (Citra_asli1, dan Citra_asli2). Namun, piksel yang didefenisikan sebagai tepi pada citra masukan dengan tingkat resolusi tinggi tersebut sebahagiannya adalah berupa *noise.*

4.1.1.2 Analisis *Timing Run*

Ditinjau dari segi waktu pemrosesan (*timing run*), dalam membandingkan kinerja metode pendeteksi tepi, metode pendeteksi tepi terbaik adalah yang memiliki waktu pemrosesan yang paling singkat diantara metode lainnya. Misalkan waktu yang diperlukan untuk memproses 1 (satu) nilai piksel dari sebuah citra adalah ‘t’, maka aproksimasi waktu untuk memproses citra pada metode Sobel adalah sebagai berikut:

Langkah 1:

Pada langkah ini setiap nilai piksel citra diproses dua kali (2t). Pertama, fungsi `rgb2gray` yang digunakan untuk mengubah nilai piksel citra masukan `handles.data1` yang berbentuk citra RGB menjadi skala keabuan. Kedua, nilai piksel yang semula merupakan nilai piksel satu dimensi dikonversi menjadi bentuk dua dimensi dengan fungsi `double`. Dan terakhir, mendefenisikan nilai yang telah diperoleh tadi sebagai `I`, maka asumsi waktu untuk ketiga citra masukan pada langkah ini adalah:

1. Citra_asli1: Waktu1 = 2x10^6(2t)
   = 4x10^6t
2. Citra_asli2: Waktu1 = 4x10^6(2t)
   = 8x10^6t
3. Citra_asli3: Waktu1 = 6x10^6(2t)
   = 12x10^6t

Langkah 2:

Pada langkah ini, untuk setiap citra masukan terdapat tiga kali pemrosesan (3t). Pertama, nilai piksel citra `I` dikonvolusi dengan `sobelhor` dengan fungsi `conv2` untuk mendapatkan nilai `Ix`. Kedua, nilai piksel citra `I` dikonvolusi dengan `sobelver` dengan fungsi `conv2` untuk mendapatkan nilai `Iy`. Dan terakhir, dari nilai `Ix` dan `Iy` yang telah diperoleh, dihitung gradien dari nilai-nilai tersebut dengan rumus `sqrt((Ix.^2)+(Iy.^2)`, sehingga diperoleh nilai `J`. Dari proses-proses tersebut, maka asumsi waktu untuk tiap citra masukan pada tahap ini adalah:

1. Citra_asli1: Waktu2 = 2x10^6(3t)
   = 6x10^6t
2. Citra_asli2: Waktu2 = 4x10^6(3t)
   = 12x10^6t
3. Citra_asli3: Waktu2 = 6x10^6(3t)
   = 18x10^6t

Langkah 3:

Pada tahap ini, bisa terjadi dua sampai empat kali pemrosesan (3t-4t). Hal ini terjadi karena adanya fungsi `if else`. Pemrosesan pertama, adalah mencari nilai pengambangan `thresh` yang sesuai untuk setiap citra masukan, dengan mengkalikan variabel alfa dengan fungsi `mean2` yang merupakan fungsi untuk mencari nilai rata-rata dari nilai mutlak `J (abs(J))`. Kemudian, setiap nilai piksel yang ada dibandingkan dengan nilai tersebut, jika lebih besar dari nilai `thresh` akan dianggap sebagai tepi dengan mengubah nilai pikselnya menjadi 1, dan jika tidak nilai piksel diubah menjadi 0. Untuk mengasumsikan waktu pada langkah ini, digunakan asumsi waktu terbesar (4t) untuk setiap citra masukan, sehingga menjadi:

1. Citra_asli1: Waktu3 = 2x10^6(4t)
   = 8x10^6t
2. Citra_asli2: Waktu3 = 4x10^6(4t)
   = 16x10^6t
3. Citra_asli3: Waktu3 = 6x10^6(4t)
   = 24x10^6t

Langkah 4:

Pada langkah ini, nilai piksel 1 (satu) dan 0 (nol) dari pemrosesan langkah sebelumnya tadi dikembalikan menjadi suatu bentuk tampilan citra dengan mengubah nilai piksel 1 (satu) menjadi warna hitam, dan nilai 0 (nol) menjadi warna putih dengan fungsi `mat2gray`. Artinya, pada langkah ini hanya terjadi satu kali pemrosesan (t), sehingga asumsi waktu untuk tiap citra:

1. Citra_asli1: Waktu4 = 2x10^6t
2. Citra_asli2: Waktu4 = 4x10^6t
3. Citra_asli3: Waktu4 = 6x10^6t

Dari keempat langkah pemrosesan dengan metode Sobel tersebut, maka keseluruhan waktu pemrosesan untuk keempat citra masukan tersebut dihitung dengan menjumlahkan keseluruhan waktu dari setiap tahap, menjadi:

1. Citra_asli1 = 4x10^6t + 6x10^6t + 8x10^6t + 2x10^6t
   = 20x10^6t
2. Citra_asli2 = 8x10^6t + 12x10^6t + 16x10^6t + 4x10^6t
   = 40x10^6t
3. Citra_asli3 = 12x10^6t + 18x10^6t + 24x10^6t + 6x10^6t
   = 60x10^6t

4.1.1.3 Analisis Sensitivity Rate

Dari persamaan $P = \frac{|n_N - n_R|}{n_R}$, yang merupakan persamaan untuk menghitung nilai *sensitivity rate* dari suatu metode pendeteksi tepi, dapat ditentukan bahwa nilai n_R berbanding lurus dengan nilai n_N. Artinya, nilai n_N akan semakin besar jika n_R juga membesar. Ini terjadi karena dalam penelitian, citra yang diberi *noise* sebagai citra *noisy* adalah citra tepi dari hasil pemrosesan dengan metode pendeteksi tepi. Dalam pendeteksian tepi dengan metode Sobel, seperti yang telah dibahas pada analisis kualitas citra tepi untuk metode Sobel, semakin besar tingkat kualitas resolusi citra masukan, maka jumlah piksel yang dinyatakan sebagai tepi pada tiap-tiap citra hasil keluaran juga akan semakin besar. Ini artinya, nilai P (P = abs(nn-nr)/nr) juga akan semakin besar untuk citra masukan dengan tingkat kualitas resolusi yang semakin tinggi.

Sehingga nilai P untuk Citra_asli3 (sebagai citra dengan resolusi tertinggi) lebih besar dari nilai P untuk Citra_asli2, dan Citra_asli1. Dan Nilai P untuk Citra_asli2 lebih besar dari nilai P untuk Citra_asli1.

4.1.2 Analisis Metode LoG (*Laplacian of Gaussian*)

Berikut ini *pseudocode* dari algoritma metode LoG, yang dapat digunakan untuk menganalisis kinerja metode ini terhadap parameter yang ditentukan:

Langkah 1

```
% Mengubah citra RGB menjadi citra skala keabuan
I = double(rgb2gray(handles.data1));
```

Langkah 2

```
% Penghalusan
% Arah X
filterx = d2dgauss(Nx1,Sigmax1,Nx2,Sigmax2,Theta1);
Ix = conv2(I,filterx,'same');
% Arah Y
filtery = d2dgauss(Ny1,Sigmay1,Ny2,Sigmay2,Theta2);
Iy = conv2(I,filtery,'same');
```

Langkah 3

```
% Norm of the gradient
NVI = sqrt(Ix.*Ix+Iy.*Iy);
```

Langkah 4

```
% Konvolusi dengan kernel laplacian 3x3
H = [0 1 0,1 -4 1,0 1 0];
J = conv2(NVI,H,'same');
```

Langkah 5

```
% Find the zerocross with thresh value
thresh = alfa.*mean2(abs(J));
edgeimage = edge(J,'zerocross',thresh);
```

Dari *pseudocode* algoritma metode LoG tersebut, maka hasil analisis kinerja metode LoG terhadap ketiga parameter yang telah ditentukan tersebut adalah sebagai berikut:

4.1.2.1 Analisis Kualitas Citra Tepi

Penghalusan dengan metode Gaussian ini dilakukan untuk mengimbangi tingkat sensitivitas yang tinggi terhadap *noise* akibat proses konvolusi citra dengan kernel Laplacian. Metode LoG ini tidak cocok digunakan untuk mendeteksi tepian pada citra dengan resolusi yang tinggi. Akibat proses penghalusan dengan fungsi Gaussian objek yang seharusnya dianggap sebagai tepian tidak mampu terdeteksi lagi oleh metode ini, dan proses konvolusi dengan kernel laplacian yang sangat sensitif terhadap *noise* menyebabkan banyak piksel *noise* terdeteksi sebagai tepi.

4.1.2.2 Analisis *Timing Run*

Seperti halnya proses analisis *timing run* untuk metode Sobel, untuk metode LoG juga diasumsikan bahwa waktu yang digunakan untuk memproses 1 (satu) nilai piksel pada citra adalah 't', maka asumsi waktu dalam proses pendeteksian tepi dengan metode LoG, adalah:

Langkah 1:

Pemrosesan pada langkah pertama untuk metode LoG ini sama persis dengan langkah pertama pada pemrosesan metode Sobel yaitu terjadi dua kali proses (2t), maka waktu yang digunakan pada metode LoG untuk langkah pertama ini diasumsikan sebagai berikut:

1. Citra_asli1: Waktu1 = 2x10^6(2t)
   = 4x10^6t
2. Citra_asli2: Waktu1 = 4x10^6(2t)
   = 8x10^6t
3. Citra_asli3: Waktu1 = 6x10^6(2t)
   = 12x10^6t

Langkah 2:

Pada algoritma ini, terjadi dua kali pemrosesan untuk setiap nilai piksel citra (2t). Proses pertama melakukan konvolusi nilai `I` dengan `filterx` yang merupakan sebuah kernel yang dibentuk dari turunan kedua fungsi Gaussian untuk nilai x, dan

menghasilkan `Ix`. Proses kedua, sama dengan proses sebelumnya, hanya saja proses ini dilakukan pada nilai y dari fungsi Gaussian `filtery`, sehingga diperoleh nilai `Iy`. Sehingga asumsi waktu pada langkah ini adalah:

1. Citra_asli1: Waktu1 = 2x10^6(2t)
   = 4x10^6t
2. Citra_asli2: Waktu1 = 4x10^6(2t)
   = 8x10^6t
3. Citra_asli3: Waktu1 = 6x10^6(2t)
   = 12x10^6t

Langkah 3:

Pemrosesan pada langkah terjadi satu kali (t), dimana prosesnya adalah menghitung nilai gradien dari nilai `Ix` dan `Iy`, dimana hasilnya disimpan dalam nilai `NVI`, maka asumsi waktu untuk proses ini adalah:

1. Citra_asli1: Waktu1 = 2x10^6t
2. Citra_asli2: Waktu1 = 4x10^6t
3. Citra_asli3: Waktu1 = 6x10^6t

Langkah 4:

Pada langkah ini, proses yang terjadi adalah konvolusi nilai piksel citra dengan kernel Laplacian, artinya pada langkah ini hanya terjadi satu kali proses (t), sehingga dapat diasumsikan bahwa waktu pada proses ini adalah:

1. Citra_asli1: Waktu1 = 2x10^6t
2. Citra_asli2: Waktu1 = 4x10^6t
3. Citra_asli3: Waktu1 = 6x10^6t

Langkah 5:

Setelah semua proses pada langkah pertama sampai langkah keempat tersebut selesai, langkah selanjutnya adalah mencari persilangan nol (*zerocross*) dari nilai-nilai piksel citra tersebut dengan batasan pengambangan. Keseluruhan proses ini terjadi dalam satu proses (t), sehingga asumsi waktu untuk langkah ini, adalah:

1. Citra_asli1: Waktu1 = 2x10^6t
2. Citra_asli2: Waktu1 = 4x10^6t
3. Citra_asli3: Waktu1 = 6x10^6t

Keseluruhan waktu pemrosesan untuk keempat citra masukan tersebut dapat dihitung dengan menjumlahkan setiap waktu pemrosesan dari setiap tahap, menjadi:

1. Citra_asli1 = 4x10^6t + 4x10^6t + 2x10^6t + 2x10^6t + 2x10^6t
   = 14x10^6t
2. Citra_asli2 = 8x10^6t + 8x10^6t + 4x10^6t + 4x10^6t + 4x10^6t
   = 28x10^6t
3. Citra_asli3 = 12x10^6t + 12x10^6t + 6x10^6t + 6x10^6t + 6x10^6t
   = 42x10^6t

4.1.2.3 Analisis *Sensitivity Rate*

Metode LoG sangat sensitif terhadap *noise*, hal ini disebabkan oleh proses konvolusi citra dengan kernel laplacian. Akibat hal ini, tentunya juga mempengaruhi nilai *sensitivity rate* dari keluaran tiap citra masukan untuk metode LoG yang bergantung pada jumlah piksel yang dinyatakan sebagai tepi pada citra hasil deteksi tepi (n_R). Seperti yang telah disebutkan pada subbab analisis *sensitivity rate* untuk metode Sobel, besarnya jumlah piksel yang dinyatakan sebagai tepi pada citra hasil deteksi tepi akan membuat jumlah piksel yang dinyatakan sebagai tepi pada citra hasil deteksi tepi yang diberi *noise* juga semakin besar, dan nilai *sensitivity rate* juga semakin besar.

Dari penjelasan tersebut, maka dapat disimpulkan bahwa Citra_asli1 memiliki nilai s*ensitivity rate* lebih besar dari Citra_asli3, Citra_asli2, dan Citra_asli1. Citra_asli3 memiliki nilai *sensitivity rate* lebih besar dari Citra_asli2 dan Citra_asli1, dan Citra_asli2 memiliki nilai *sensitivity rate* yang lebih besar dari Citra_asli1.

4.1.3 Analisis Metode Canny

Berikut ini *pseudocode* dari algoritma metode Canny yang digunakan untuk menganalisis kinerja metode Canny terhadap parameter-parameter yang telah ditentukan:

Langkah 1

```
% Mengubah citra RGB menjadi citra skala keabuan
I = double(rgb2gray(handles.data1));
```

Langkah 2

```
% Penghalusan
% Arah X
filterx = d2dgauss(Nx1,Sigmax1,Nx2,Sigmax2,Theta1);
Ix = conv2(I,filterx,'same');
% Arah Y
filtery = d2dgauss(Ny1,Sigmay1,Ny2,Sigmay2,Theta2);
Iy = conv2(I,filtery,'same');
```

Langkah 3

```
% Norm of Gradien
NVI = sqrt(Ix.*Ix+Iy.*Iy);
```

Langkah 4

```
% Pengambangan
I_max = max(max(NVI));
I_min = min(min(NVI));
level = alfa*(I_max-I_min)+I_min;
Ibw = max(NVI,level.*ones(size(NVI)));
```

Langkah 5

```
% Peredaman Titik Non Maksimum
[n,m] = size(Ibw);
for i = 2:n-1
for j = 2:m-1
```

```
if Ibw(i,j)>level
X = [-1,0,+1;-1,0,+1;-1,0,+1];
Y = [-1,-1,-1;0,0,0;+1,+1,+1];
      Z = [Ibw(i-1,j-1),Ibw(i-1,j),Ibw(i-1,j+1);
            Ibw(i,j-1),Ibw(i,j),Ibw(i,j+1);
            Ibw(i+1,j-1),Ibw(i+1,j),Ibw(i+1,j+1)];
      XI = [Ix(i,j)/NVI(i,j), -Ix(i,j)/NVI(i,j)];
      YI = [Iy(i,j)/NVI(i,j), -Iy(i,j)/NVI(i,j)];
      ZI = interp2(X,Y,Z,XI,YI);
            if Ibw(i,j) >= ZI(1) & Ibw(i,j) >= ZI(2)
                  edgeimage(i,j)=I_max;
            else
                  edgeimage(i,j)=I_min;
            end
      else
            edgeimage(i,j)=I_min;
      end
end
end
```

Dari tahapan-tahapan tersebut, maka hasil analisis kinerja metode Canny terhadap ketiga parameter yang telah ditentukan , adalah sebagai berikut:

4.3.1.1 Analisis Kualitas Citra Tepi

Apabila metode LoG sangat mungkin untuk menyatakan *noise* sebagai tepi, tidak demikian dengan metode Canny. Hal ini disebabkan karena metode Canny menggunakan pengambangan *hysteresys*, yang artinya nilai pengambangannya ditentukan dengan melihat nilai batas bawah dan batas atas yang menentukan suatu piksel apakah dinyatakan sebagai tepi atau tidak.

Selain itu, adanya langkah untuk meredam titik non maksimum membuat tepian yang dihasilkan metode ini adalah tepian tunggal (*single edge*), dimana tepi tunggal merupakan tepi yang baik dari sebuah metode pendeteksi tepi. Karena dua

tahap inilah yang nantinya membuat citra tepi hasil keluaran metode Canny menjadi citra tepi terbaik.

Seperti halnya dengan dua metode sebelumnya, kualitas citra yang semakin tinggi membuat metode pendeteksi tepi tidak mampu untuk mendeteksi beberapa keberadaan piksel yang seharusnya dinyatakan sebagai tepi, begitu juga dengan metode Canny. Seluruh piksel tepi pada hasil keluaran metode Canny untuk citra masukan dengan resolusi rendah (Citra_asli1) akan mudah dideteksi oleh metode Canny. Namun, untuk citra dengan resolusi tinggi akan ditemukan beberapa piksel yang seharusnya merupakan tepi tidak didefenisikan sebagai tepi oleh metode ini.

4.3.1.2 Analisis *Timing Run*

Dengan mengasumsikan waktu untuk memproses tiap nilai piksel pada citra masukan sebagai ‘t’, maka dapat dianalisis asumsi waktu yang diperlukan untuk mendeteksi citra dengan metode Canny, yaitu:

Langkah 1:

Pemrosesan pada langkah pertama untuk metode Canny masih sama dengan langkah pertama pada pemrosesan metode Sobel dan metode LoG yaitu terjadi 2 (dua) kali proses (2t), maka waktu yang digunakan pada metode LoG untuk langkah pertama ini diasumsikan sebagai berikut:

1. Citra_asli1: Waktu1 = 2x10^6(2t)
   = 4x10^6t
2. Citra_asli2: Waktu1 = 4x10^6(2t)
   = 8x10^6t
3. Citra_asli3: Waktu1 = 6x10^6(2t)
   = 12x10^6t

Langkah 2:

Pemrosesan pada langkah kedua untuk metode Canny sama dengan pemrosesan yang terjadi pada langkah kedua untuk metode LoG, dengan dua kali pemrosesan (2t) maka asumsi waktu untuk langkah ini menjadi:

1. Citra_asli1: Waktu1 = 2x10^6(2t)
   = 4x10^6t
2. Citra_asli2: Waktu1 = 4x10^6(2t)
   = 8x10^6t
3. Citra_asli3: Waktu1 = 6x10^6(2t)
   = 12x10^6t

Langkah 3:

Langkah ketiga untuk pemrosesan metode Canny juga sama dengan langkah ketiga untuk metode LoG, dimana pada langkah ini terjadi 1 (satu) kali proses untuk tiap nilai piksel citra masukan (t), sehingga waktu untuk keempat citra masukan dapat diasumsikan sebagai berikut:

1. Citra_asli1: Waktu1 = 2x10^6t
2. Citra_asli2: Waktu1 = 4x10^6t
3. Citra_asli3: Waktu1 = 6x10^6t

Langkah 4:

Untuk langkah 4 ini, ada 4 jenis proses yang terjadi untuk setiap nilai piksel citra (4t) pada langkah ini. Pertama, menentukan nilai batas bawah `I_min` yang berguna untuk menentukan nilai pengambangan `thresh` dengan fungsi `min`. Proses kedua, menentukan nilai batas atas `I_max` yang juga digunakan untuk mencari nilai pengambangan `thresh` dengan fungsi `max`. Selanjutnya, mencari nilai pengambangan yang sesuai dengan rumus `thresh = alfa*(I_max-I_min)+I_min`, dimana `alfa` adalah nilai yang ditentukan sendiri, sesuai kebutuhan. Proses terakhir pada tahap ini adalah mencari nilai `Ibw`, yang merupakan nilai piksel citra akhir yang

nantinya digunakan pada tahap berikutnya. Asumsi waktu untuk keempat citra masukan tersebut, menjadi:

1. Citra_asli1: Waktu1 = 2x10^6(4t)
   = 8x10^6t
2. Citra_asli2: Waktu1 = 4x10^6(4t)
   = 16x10^6t
3. Citra_asli3: Waktu1 = 6x10^6(4t)
   = 24x10^6t

Langkah 5:

Proses paling panjang pada metode Cannya adalah pada langkah ini. Pada proses ini metode Canny akan berusaha mencari titik yang non maksimum, dan akan meredam titik tersebut. Proses ini berlangsung dengan mencocokkan nilai `Ibw` dengan empat keadaan, yaitu `if Ibw(i,j)>level` dan `end`, serta keadaan `if Ibw(i,j) >= ZI(1) & Ibw(i,j) >= ZI(2)` dan `else`. Dari keadaan pertama terdapat tiga penyelesaian yang dapat dikerjakan, untuk keadaan kedua, ketiga, dan keempat masing-masing memiliki satu penyelesaian. Jika diasumsikan, langkah ini membutuhkan 2 (dua) sampai 8 (delapan) kali pemrosesan (2t-8t).

Untuk mengasumsikan waktu dalam pemrosesan ini, digunakan asumsi terpanjang yang mungkin terjadi dalam langkah ini, yaitu 8t, maka asumsi waktu untuk langkah ini adalah:

1. Citra_asli1: Waktu1 = 2x10^6(8t)
   = 16x10^6t
2. Citra_asli2: Waktu1 = 4x10^6(8t)
   = 32x10^6t
3. Citra_asli3: Waktu1 = 6x10^6(8t)
   = 48x10^6t

Secara keseluruhan waktu yang dibutuhkan untuk memproses keempat citra masukan tersebut dengan metode Canny membutuhkan waktu sebesar:

1. Citra_asli1 = 4x10^6t + 4x10^6t + 2x10^6t + 8x10^6t + 16x10^6t
   = 34x10^6t
2. Citra_asli2 = 8x10^6t + 8x10^6t + 4x10^6t + 16x10^6t + 32x10^6t
   = 68x10^6t
3. Citra_asli3 = 12x10^6t + 12x10^6t + 6x10^6t + 24x10^6t + 48x10^6t
   = 102x10^6t

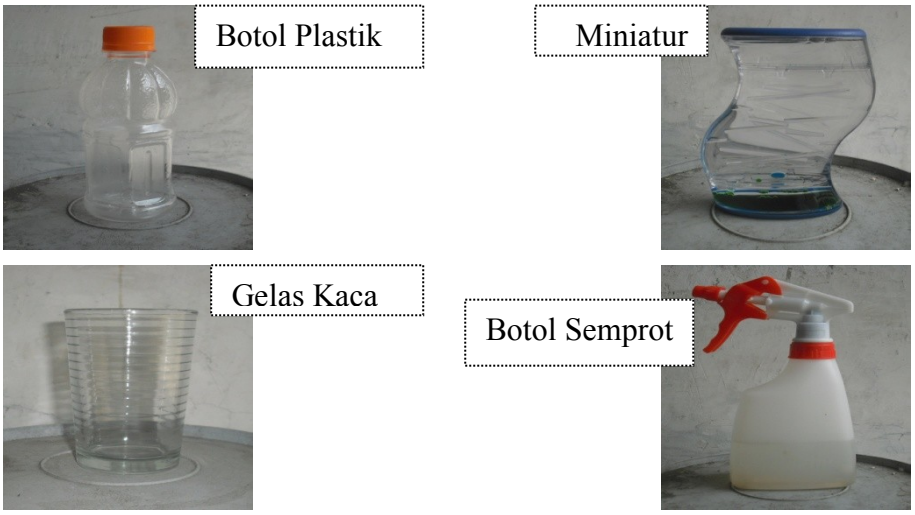
4.3.1.3 Analisis Sensitivity Rate

Berbeda dengan metode Sobel dan LoG, jika pada metode-metode tersebut nilai *sensitivity rate* dari tiap citra masukan akan semakin besar untuk citra dengan kualitas resolusi yang juga semakin besar, pada metode Canny hal itu tidak akan terjadi. Tahap-tahap pada metode Canny yang sangat kompleks membuat metode ini memiliki ketahanan yang cukup tinggi terhadap *noise.* Berbeda dengan metode LoG, karena kesensitivitasannya yang tinggi terhadap *noise* membuat metode ini mendefenisikan piksel yang merupakan *noise* tersebut sebagai sebuah tepi. Dalam metode Canny, dengan langkah peredaman titik lokal non mkasimum, metode ini berhasil membedakan mana piksel yang memang sebagai tepi dan mana piksel yang bukan tepi.

Dengan langkah-langkah pendeteksian tepi yang digunakan pada pendeteksian tepi metode Canny, metode bekerja maksimal pada citra masukan dengan tingkat resolusi yang rendah, karena memiliki nilai intensitas yang lebih tinggi untuk setiap pikselnya dibanding citra dengan tingkat resolusi tinggi. Hal ini menyebabkan jumlah piksel tepi yang terdeteksi pada citra dengan resolusi tinggi akan lebih sedikit dibanding dengan jumlah piksel yang terdeteksi sebagai tepi pada citra dengan kualitas resolusi rendah. Dan ini artinya nilai *sensitivity rate* untuk citra dengan resolusi rendah (Citra_asli1) akan lebih besar dibanding dengan nilai *sensitivity rate* untuk citra masukan yang memiliki tingkat resolusi tinggi (Citra_asli3).

4.2 Objek yang diteliti

Riset dilakukan dengan memotret Gelas Kaca dan Botol Plastik, dengan tingkat resolusi yang berbeda, yaitu 2 (dua) megapiksel, 4 (empat) megapiksel, dan 6 (enam) megapiksel. Keseluruhan citra hasil akuisisi ini disimpan dalam format jpeg '.jpg' dan BMP '.bmp' .Dalam aplikasi yang dibangun membangun bahasa pemrograman Matlab 7.5.0. Gambar 4.1 merupakan gambar objek hasil akuisisi data yang digunakan sebagai masukan dalam aplikasi.



Gambar 4.1 *Gambar yang akan digunakan untuk penelitian*

4.3 Implementasi Rancangan

Setelah melalui tahap perancangan, tahap selanjutnya untuk mengembangkan suatu perangkat lunak adalah tahap implementasi. Untuk mengetahui apakah hasil implementasi perangkat lunak tersebut telah berhasil atau tidak, diperlukan pengujian. Berikut ini hasil implementasi dan pengujian dari aplikasi perangkat lunak yang telah dibangun:

Dalam pengembangan aplikasi pendeteksi tepi citra dengan metode Sobel, LoG, dan Canny ini bahasa pemrograman yang digunakan adalah Matlab 7.5.0 (R2007b). *Listing* program dari aplikasi ini dapat dilihat pada halaman Lampiran. Dalam implementasinya, aplikasi ini terdiri atas 7 (Tujuh) buah tampilan utama, yang dalam Matlab dikenal sebagai figur. Figur-figur tersebut terdiri atas:

1. Figur Cover
2. Figur Aplikasi
3. Figur Profil
4. Figur Terima Kasih
5. Figur Sobel
6. Figur Canny
7. Figur LoG(Laplacian and Gaussian)

4.3.1 Figur Cover

Figur PendeteksianTepi ini merupakan tampilan utama dari aplikasi yang telah dibanguan. Figur ini menghubungkan pengguna untuk dapat menggunakan aplikasi pendeteksian tepi. Figur ini diimplementasikan dengan tampilan seperti Gambar 4.9.

Gambar 4.2 Tampilan Figur Cover

Figur Cover ini dirancang untuk memulai sebuah jendela. Tombol-tombol yang ditampilkan adalah “Mulai”, “Informasi”, dan “Keluar”. Tombol Mulai, berfungsi untuk menampilkan figure Aplikasi. Tombol Informasi befungsi untuk

menampilkan figure Profil. Tombol keluar berfungsi untuk menampilkan figure Terima Kasih. Kita mulai dengan menekan tombol Informasi(Figur Profile)

4.3.2 Figur Profil

Figur Profil ini menampilkan informasi biodata dari pembentuk software ini. Dimana berisi Nama, NIM, Jurusan, E-Mail.

Gambar 4.3 Tampilan Figur Profile

Tombol Kembali berfungis untuk menutup jendela Figur Profile dan membuka lagi Figur Cover. Tekan tombol Kembali. Maka pengguna akan kembali menuju jendela Figur Cover. Selanjutnya di Figur Cover tekan tombol Keluar. Maka pengguna akan masuk ke Figur Terima Kasih

4.3.3 Figur Terima Kasih

Figur Terima_Kasih ini merupakan tampilan pilihan, apakah pengguna ingin keluar dari aplikasi ini atau tidak.

Gambar 4.4 Tampilan Terima_Kasih

Jika pengguna menekan tombol iya, maka semua figure yang terbuka akan tertutup otomatis. Apabila anda menekan tombol Tidak, maka pengguna akan kembali ke figure Cover. Masuk ke Figur Cover lagi, lalu Klik tombol Mulai(Figur Aplikasi).

4.3.4 Figur Aplikasi

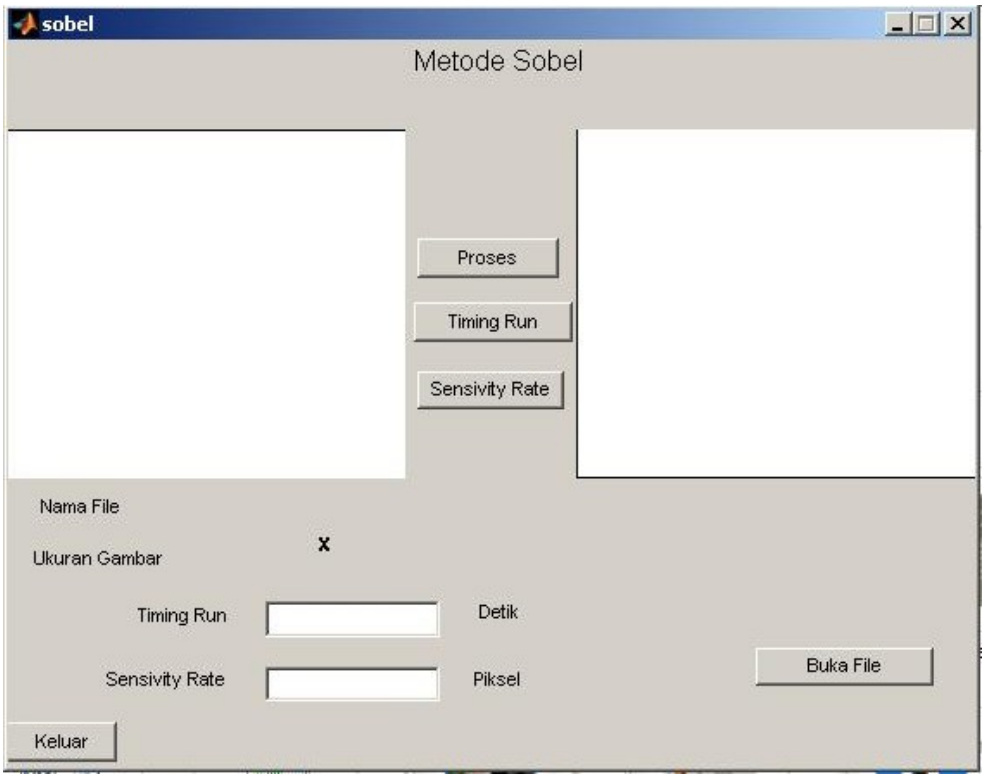
Figur Aplikasi ini merupakan tampilan dimana ketiga metode tersebut ditampilkan. Pengguna bisa memilih metode apa yang akan digunakan untuk memproses sebuah citra terlebih dahulu.

Gambar 4.5 Tampilan Aplikasi

Ada tombol masing-masing metode yaitu Sobel, Canny, dan LoG(*Laplacian of Gaussian*). Tombol kembali berfungsi untuk menutup jendela Figur Aplikasi, dan membuka Figur Cover. Masuk ke Figur Sobel.

4.3.5 Figur Sobel

Figur MetodeSobel merupakan figur yang menghubungkan pengguna dengan metode Sobel untuk mendeteksi tepi citra. Untuk menjangkau figur ini, pengguna dapat mengklik menu Metode Sobel pada figure Aplikasi, Tampilan dari figur Sobel diperlihatkan pada Gambar 4.6



Gambar 4.6 Tampilan Metode sobel

Tampilan Figur MetodeSobel Pada figur ini, terdapat 3 Tombol Fungsi yaitu Proses, Timing Run, Sensivity Rate, dan Buka File, dan Keluar. Dari menu File, pengguna dapat melakukan beberapa operasi yang dapat dipilih dengan mengklik tiap pilihan sub-menu dari menu File. Operasi-operasi tersebut terdiri atas :

1. Proses, yang berfungsi untuk memproses citra dengan menggunakan metode sobel.
2. Timing Run, yang berfungsi untuk memproses Timing Run pada citra
3. Sensivity Rate, Yang berfungsi untuk memproses Sensivitas tepi pada

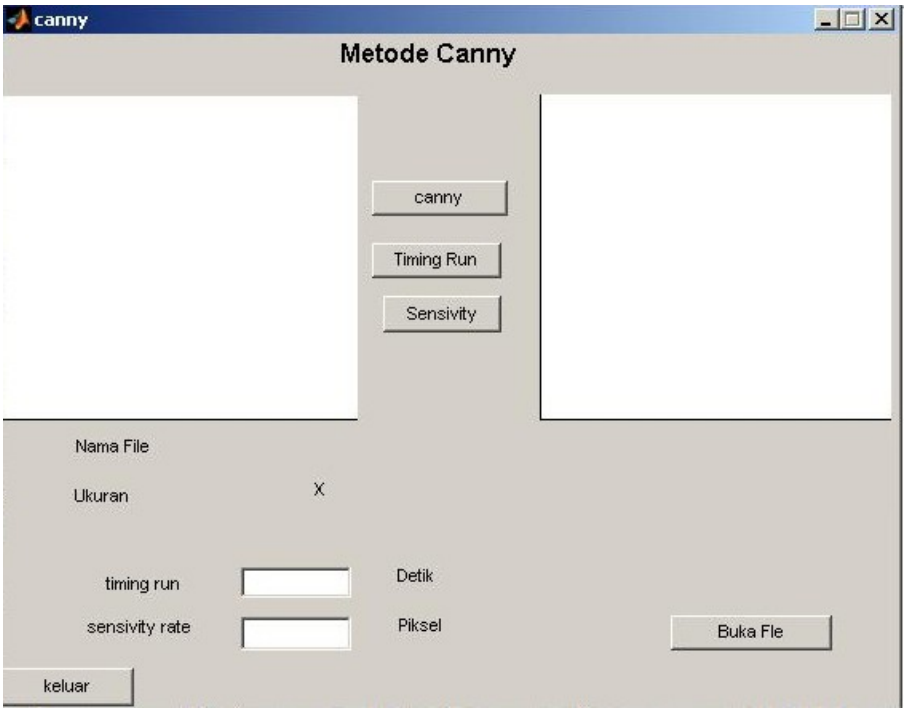
citra

4. Buka File, berfungsi untuk memasukkan citra asli

5. Keluar, yang berfungsi untuk keluar dari figure Metode Sobel

4.3.6 Figur Canny

Figur Metode Canny merupakan figur yang menghubungkan pengguna dengan metode Canny untuk mendeteksi tepi citra. Untuk menjangkau figur ini, pengguna dapat mengklik menu Metode Canny pada figure Aplikasi, Tampilan dari figur Canny diperlihatkan pada Gambar 4.7



Gambar 4.7 Tampilan Metode Canny

Tampilan Figur Canny mirip dengan figure Sobel. Perbedaannya adalah hanya pada masukan ALFA saja. Pada figur ini, terdapat 3 Tombol Fungsi yaitu Proses, Timing Run, Sensivity Rate, dan Buka File, dan Keluar. Dari menu File, pengguna dapat melakukan beberapa operasi yang dapat dipilih dengan mengklik tiap pilihan sub-menu dari menu File. Operasi-operasi tersebut terdiri atas :

1. Proses, yang berfungsi untuk memproses citra dengan menggunakan

metode canny.

2. Timing Run, yang berfungsi untuk memproses Timing Run pada citra

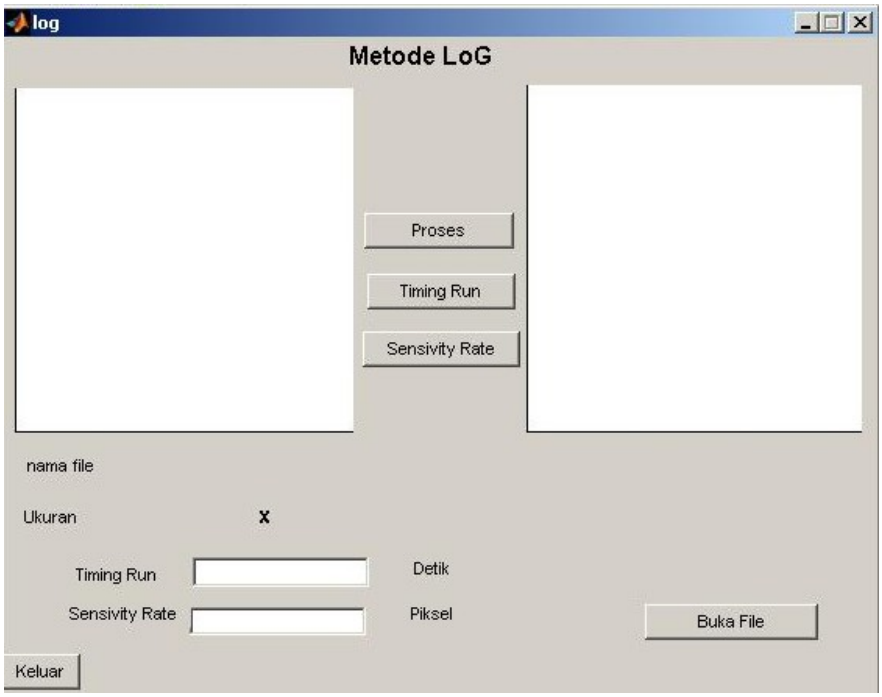
3. Sensivity Rate, Yang berfungsi untuk memproses Sensivitas tepi pada citra

4. Buka File, berfungsi untuk memasukkan citra asli

5. Keluar, yang berfungsi untuk keluar dari figure Metode Canny

4.3.7 Figur LoG(*Laplacian of Gaussian*)

Figur Metode LoG sama persis dengan figure Canny. Figur ini merupakan figur yang menghubungkan pengguna dengan metode LoG untuk mendeteksi tepi citra. Untuk menjangkau figur ini, pengguna dapat mengklik menu Metode LoG pada figure Aplikasi, Tampilan dari figur LoG diperlihatkan pada Gambar 4.15



Gambar 4.8 Tampilan Metode LoG

Tampilan Figur LoG sama persis dengan figure Canny. Pada figur ini, terdapat 3 Tombol Fungsi yaitu Proses, Timing Run, Sensivity Rate, dan Buka File, dan Keluar. Dari menu File, pengguna dapat melakukan beberapa operasi yang dapat

dipilih dengan mengklik tiap pilihan sub-menu dari menu File. Operasi-operasi tersebut terdiri atas :

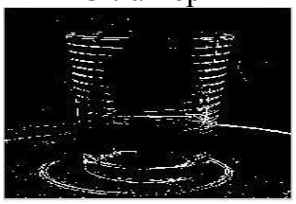
1. Proses, yang berfungsi untuk memproses citra dengan menggunakan metode LoG.
2. Timing Run, yang berfungsi untuk memproses Timing Run pada citra
3. Sensivity Rate, Yang berfungsi untuk memproses Sensivitas tepi pada citra
4. Buka File, berfungsi untuk memasukkan citra asli
5. Keluar, yang berfungsi untuk keluar dari figure Metode LoG

4.4 PEMBAHASAN

Dibawah ini adalah pembahasan hasil uji coba pada **:**

4.4.1 Metode Sobel

Tabel 4.1 Tabel Hasil Analisis Metode Sobel

No.	Citra Asli	Resolusi	Format File	*Timing Run*	*Sensivity Rate*
1.	 Citra Tepi  Gelas Kaca	2 MegaPiksel	JPG	6,81117	0,16407
		2 MegaPiksel	BMP	6,8778	0,168884
		4 MegaPiksel	JPG	25,3274	0,153182
		4 MegaPiksel	BMP	25,1924	0,140155
		6 MegaPiksel	JPG	37,3014	0,0377466
		6 MegaPiksel	BMP	36,2161	0,0464898

No.	Citra Asli	Resolusi	Format File	*Timing Run*	*Sensivity Rate*
2.		2 MegaPiksel	JPG	6,99009	0,117501
		2 MegaPiksel	BMP	6,85987	0,119603
		4 MegaPiksel	JPG	24,8732	0,0539686
	Citra Tepi  Botol Plastik	4 MegaPiksel	BMP	25,618	0,0458666
		6 MegaPiksel	JPG	36,4718	0,026759
		6 MegaPiksel	BMP	36,5294	0,0193191
3.		2 MegaPiksel	JPG	6,71122	0,137205
		2 MegaPiksel	BMP	6,9281	0,133766
		4 MegaPiksel	JPG	25,3415	0,0734947
	Citra Tepi  Botol Semprot	4 MegaPiksel	BMP	26,535	0,0627989
		6 MegaPiksel	JPG	36,0534	0,0637813
		6 MegaPiksel	BMP	37,6113	0,0797157

No.	Citra Asli	Resolusi	Format File	*Timing Run*	*Sensivity Rate*
4.		2 MegaPiksel	JPG	7,46617	0,29075
		2 MegaPiksel	BMP	7,22122	0,282828
		4 MegaPiksel	JPG	25,1709	0,254809
	Citra Tepi  Miniatur	4 MegaPiksel	BMP	28,1006	0,254323
		6 MegaPiksel	JPG	35,9095	0,245273
		6 MegaPiksel	BMP	35,7049	0,24916

Citra dengan resolusi 2 MegaPiksel objek Gelas Kaca (53%), menghasilkan keluaran yang paling baik daripada citra 4MP (46%), 6MP (35%). Waktu yang dibutuhkan untuk prosesnya juga tidak terlalu lama (*TR*: 6,81117), dan nilai Sensivitasnya juga besar(*SR*: 0,16407) . Pada citra masukan lain, citra tepi yang dihasilkan kurang begitu baik, hal ini dikarenakan banyak *Noise* yang terdeteksi sebagai Tepi.

Sedangkan pada File berformat JPG waktu pemrosesannya lebih lambat(*TR*: 6,81117) daripada format BMP (*TR*: 6,8778) . Tetapi nilai sensivitasnya lebih tinggi(*SR*: 0,16407) daripada file format BMP (SR: 0,168884). Itu menandakan kelebihan format BMP adalah lebih cepat pemrosesannya, dan format JPG sensivitasnya lebih besar.

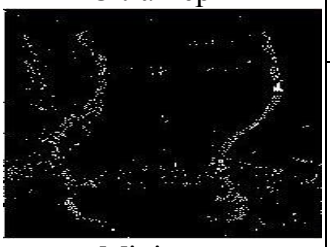
Semakin besar resolusi citra(2MP -> 4MP), Semakin lama juga waktu pemrosesannya (TR: 6,81117 -> 25,3274), sedangkan sensivitasnya malah semakin menurun (SR: 0,16407 -> 0,153182). Jadi sebaiknya jika melakukan pemrosesan deteksi tepi. Digunakan resolusi yang rendah agar tidak terdapat banyak *Noise*. Perbedaan waktu pemrosesan juga terjadi pada objek, objek yang terbuat dari plastic (Botol Semprot) waktu pemrosesannya lebih cepat (TR: 6,71122) daripada objek

yang terbuat dari kaca (TR: 6,81117) dan mika (TR: 7,46617). Dan bahan Mika sensivitasnya (SR: 0,29075) lebih besar daripada Botol Semprot (SR: 0,137205) dan Botol Plastik (SR: 0,117501).

4.4.2 Metode LoG (Laplacian and Gaussian)

Tabel 4.2 Tabel Hasil Analisis Metode LoG (*Laplacian of Gaussian*)

No.	Citra Asli	Resolusi	Format File	*Timing Run*	*Sensivity Rate*
1.		2 MegaPiksel	JPG	3,10049	0,769514
		2 MegaPiksel	BMP	2,75757	0,781896
		4 MegaPiksel	JPG	7,24563	0,709185
	Citra Tepi  Gelas Kaca	4 MegaPiksel	BMP	7,16605	0,69657
		6 MegaPiksel	JPG	9,26703	0,94615
		6 MegaPiksel	BMP	9,96209	0,965673
2.	Citra Asli 	2 MegaPiksel	JPG	2,75919	0,511763
		2 MegaPiksel	BMP	2,70687	0,511391
		4 MegaPiksel	JPG	7,61991	0,390453
		4 MegaPiksel	BMP	7,0856	0,376127
	Citra Tepi  Botol Plastik	6 MegaPiksel	JPG	9,15971	0,334558
		6 MegaPiksel	BMP	9,22466	0,337152

No.	Citra Asli	Resolusi	Format File	*Timing Run*	*Sensivity Rate*
3.		2 MegaPiksel	JPG	2,82464	0,666721
		2 MegaPiksel	BMP	2,80438	0,671689
		4 MegaPiksel	JPG	7,11669	0,953442
		4 MegaPiksel	BMP	7,0552	0,925437
	Citra Tepi  Botol Semprot	6 MegaPiksel	JPG	9,13375	4,7977
		6 MegaPiksel	BMP	9,13898	4,63851
4.	Citra Asli 	2 MegaPiksel	JPG	2,75204	0,584248
		2 MegaPiksel	BMP	2,80402	0,562336
		4 MegaPiksel	JPG	7,0381	0,88173
		4 MegaPiksel	BMP	7,05211	0,864548
	Citra Tepi  Miniatur	6 MegaPiksel	JPG	9,13086	0,926443
		6 MegaPiksel	BMP	9,1467	0,913573

Pada metode LoG hanya resolusi rendah yang menghasilkan gambar paling baik, yaitu Citra dengan resolusi 2 MegaPiksel (26%), namun kualitas yang tidak baik pada citra dengan resolusi 6 MegaPiksel (15%), karena tepi citra tidak terdeteksi. Pada jenis citra ini metode LoG mendefinisikan *Noise* sebagai tepi. Hal ini dikarenakan proses konvolusi dengan kernel Laplacian, yang membuat metode ini sangat sensitive terhadap *Noise*. Bedanya dengan berformat JPG adalah,Format BMP ini lebih cepat (TR : 2,75919) pemrosesannya dari pada JPG (TR : 2,70687).

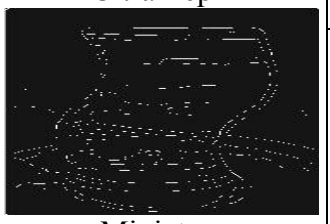
Pada miniatur perbedaan JPG dan BMP adalah, JPG lebih cepat waktu (TR: 2,75204) pemrosesannya daripada BMP (TR: 2,80402). Itu karena miniatur banyak sekali memantulkan cahaya sehingga membuat pemrosesan menjadi lebih lambat pada format BMP. Semakin besar resolusi citra (2MP -> 6MP), Semakin lama juga waktu pemrosesannya(TR: 3,10049 -> 7,16605), sedangkan sensivitasnya malah semakin menurun (SR: 0,511763 -> 0,511391). Jadi sebaikknya jika melakukan pemrosesan deteksi tepi. Digunakan resolusi yang rendah (2MP) agar tidak terdapat banyak *Noise*. Perbedaan waktu pemrosesan juga terjadi pada objek, objek yang terbuat dari mika waktu pemrosesannya lebih cepat (TR: 2,75204) daripada objek yang terbuat dari kaca (TR: 3,10049) dan plastik (TR: 2,75919). Dan bahan kaca sensivitasnya (SR: 0,769514) lebih besar daripada yang berbahan plastik (SR: 0,511763) dan mika (SR: 0,584248).

Sedangkan pada File berformat JPG waktu pemrosesannya lebih lambat (TR: 3,10049) daripada format BMP (TR: 2,75757). Tetapi nilai sensivitasnya lebih tinggi (SR: 0,584248) daripada file format BMP (SR: 0,562336). Di format BMP sensivitas lebih kecil dari pada format JPG. Itu menandakan kelebihan format BMP adalah lebih cepat pemrosesannya, dan format JPG sensivitasnya lebih besar.

4.4.3 Metode Canny

Tabel 4.3 Tabel Hasil Analisis Metode Canny

No.	Citra Asli	Resolusi	Format File	*Timing Run*	*Sensivity Rate*
1.		2 MegaPiksel	JPG	202,746	0,999029
		2 MegaPiksel	BMP	225,669	0,999032
		4 MegaPiksel	JPG	514,57	0,998943
	Citra Tepi  Gelas Kaca	4 MegaPiksel	BMP	547,25	0,998947
		6 MegaPiksel	JPG	617,313	0,998603
		6 MegaPiksel	BMP	639,458	0,99859
2.	Citra Asli 	2 MegaPiksel	JPG	224,026	0,998574
		2 MegaPiksel	BMP	227,001	0,998549
		4 MegaPiksel	JPG	453,218	0,998115
		4 MegaPiksel	BMP	431,8	0,99811
	Citra Tepi  Botol Plastik	6 MegaPiksel	JPG	588,715	0,998067
		6 MegaPiksel	BMP	630,778	0,998068

No.	Citra Asli	Resolusi	Format File	*Timing Run*	*Sensivity Rate*
3.		2 MegaPiksel	JPG	232,754	0,997457
		2 MegaPiksel	BMP	235,349	0,997138
		4 MegaPiksel	JPG	506,932	0,997831
		4 MegaPiksel	BMP	506,694	0,997138
	Citra Tepi  Botol Semprot	6 MegaPiksel	JPG	624,618	0,997211
		6 MegaPiksel	BMP	622,041	0,997895
4.	Citra Asli 	2 MegaPiksel	JPG	329,964	0,999066
		2 MegaPiksel	BMP	334,307	0,999054
		4 MegaPiksel	JPG	718,257	0,998971
		4 MegaPiksel	BMP	785,516	0,998954
	Citra Tepi  Miniatur	6 MegaPiksel	JPG	876,787	0,998905
		6 MegaPiksel	BMP	898,695	0,998906

Pada metode Canny hanya hasil tepi yang dihasilkan adalah paling baik dari metode lainnya tapi membutuhkan waktu yang paling lama (TR: 202,746) jika dibandingkan dengan metode lain Sobel (TR: 6,81117) dan LoG (TR: 3,10049). Semakin besar resolusi citra (2MP -> 6MP) semakin tidak terdeteksi tepi yang dihasilkan. Itu karena semakin banyak pixel yang terdeteksi,maka pixel tersebut mengurangi kualitas tepi dengan mengubahnya sebagai *Noise*. Jadi resolusi 2 Megapiksel adalah yang terbaik pada metode ini (63%). Selain waktu yang

dibutuhkan lebih cepat (TR: 202,746) , sensivitasnya juga lebih tinggi (SR: 0,999029) dari resolusi yang lebih tinggi. Jadi sebaiknya jika melakukan pemrosesan deteksi tepi. Digunakan resolusi yang rendah agar tidak terdapat banyak *Noise*.

Pada Gelas Kaca perbedaan JPG dan BMP adalah, JPG lebih cepat waktu pemrosesannya (TR: 202,746) daripada BMP (TR: 225,669). Itu karena semakin banyak pixel yang terdeteksi pada BMP yang mengakibatkan pemrosesan menjadi lebih lambat. Semakin besar resolusi citra (2MP -> 6MP), Semakin lama juga waktu pemrosesannya (TR: 202,746 -> 514,57), sedangkan sensivitasnya malah semakin menurun (SR: 0,999029 -> 0,998947). Perbedaan waktu pemrosesan juga terjadi pada objek, objek yang terbuat dari Kaca waktu pemrosesannya lebih cepat (TR: 202,746) daripada objek yang terbuat dari Mika (TR: 329,964), dan Plastik (TR: 224,026). Dan bahan Mika sensivitasnya lebih besar (SR: 0,999066) daripada yang berbahan plastic (SR: 0,998549) dan Kaca (SR: 0,999029).

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Setelah melakukan penelitian untuk menganalisis pengaruh kualitas resolusi citra terhadap kinerja metode pendeteksi tepi, maka dapat disimpulkan bahwa:

1. Kualitas resolusi citra mempengaruhi kinerja metode pendeteksi tepi. Pada metode Sobel dan LoG yang berformat JPG, semakin tinggi. tingkat resolusi citra (2MP -> 6MP) yang menjadi citra masukan, nilai timing run (TR: 6,81117 -> 25,3274; TR: 3,10049 -> 7,24563) dan sensitivity rate akan semakin besar (SR: 0,16407 -> 0,153182 ; SR: 0,769514 -> 0,709185), dan pada file berformat BMP sensivitasnya makin kecil (SR: 0,168884 -> 0,140155; SR: 0,781896 -> 0,69657). sedangkan citra tepi hasil keluarannya semakin berkurang kualitasnya (53% -> 46%; 26% -> 25%). Pada metode Canny, terdapat sedikit perbedaan. Pada metode Canny, sensitifitas metode ini terhadap noise semakin kecil (SR: 0,999029 -> 0,998943) jika citra masukannya memiliki tingkat resolusi yang semakin tinggi itu terjadi pada semua format JPG dan BMP. Kualitas Resolusi yang baik dihasilkan oleh Gelas Kaca 2MP (53%) berformat JPG dengan menggunakan metode Sobel (TR: 6,81117 -> SR: 0,16407).

2. Pada Botol Semprot waktu yang diperlukan untuk pemrosesan lebih cepat (TR: 6,71122) daripada Gelas Kaca, Botol Plastik dan Miniatur (TR: 6,81117; 6,99009; & 7,46617). Karena daya pantul cahaya Gelas kaca, Botol Plastik dan Miniatur lebih besar dari pada plastik. Sehingga menambah *Noise* dan membuat pemrosesan semakin lambat.

3. Jika ditinjau dari segi waktu dan sensitivitas terhadap noise, metode terbaik untuk mendeteksi tepi suatu citra adalah metode LoG (TR: 3,10049; & 0,769514) karena diantara ketiga metode tersebut metode ini memiliki nilai timing run paling kecil untuk setiap citra masukannya serta memiliki nilai sensitivitas yang tinggi

terhadap noise. Namun, citra tepi yang dihasilkan memiliki kualitas yang kurang baik (26%). Metode Canny merupakan metode yang citra tepi hasil keluarannya paling baik (63%). Metode Sobel baik digunakan untuk mendeteksi tepi citra yang sederhana, selain karena waktu yang diperlukan tidak terlalu besar (TR: 6,81117), kualitas citra tepi yang dihasilkan juga lumayan baik (53%).

4. Perbedaan BMP dan JPG adalah sama jika dilihat dari tepi yang dihasilkan (53% & 56%). Letak perbedaannya adalah "timing run" dimana *Timing Run* pada format file BMP (TR: 6,85987) lebih cepat terhadap Format JPEG (TR: 6,99009).

5.2 Saran

Berikut ini beberapa saran yang diharapkan dapat dilakukan untuk pengembangan dari penulisan Tugas Akhir ini:

1. Pemilihan nilai variabel yang mempengaruhi kinerja tiap metode pendeteksi tepi yang digunakan, hendaknya lebih diteliti dengan baik agar menghasilkan citra tepi terbaik dibanding jika menggunakan nilai variabel lainnya.

2. Untuk pengembangan selanjutnya, dapat dilakukan dengan cara menganalisis pengaruh kualitas resolusi citra dengan format selain *.jpg, dan *.bmp terhadap kinerja metode pendeteksi tepi.

3. Bisa menggunakan dua kamera berbeda merek untuk mengetahui pengaruh kualitas pada teknologi kamera yang dipakai.

4. Untuk pengembangan selanjutnya diharapkan ‘Ukuran gambar juga dibedakan’ pada citra asli

DAFTAR PUSTAKA

1. Febriani. 2008. ”Analisis penelusuran tepi citra menggunakan detektor tepi sobel dan canny”. Dalam Febriani dan Lussiana, E. T. P. (eds). *Seminar Ilmiah Nasional Komputer dan Sistem Intelijen (KOMMIT 2008)*: hal. 462-466. Depok: Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Gunadarma.

2. Hestiningsih, I. 2008. Diakses tanggal, 26 Januari 2009. *Pengolahan Citra*. http://toba.mytoba.com/dl/Pengolahan%20Citra.pdf

3. Herdiyeni, Y. 2007. Diakses tanggal, 31 Maret 2009. *Edge Detection.*www.ilkom.fmipa.ipb.ac.id/~yeni/files/ppcd/**Kuliah**%2007%20**Edge**%20**Detection**.**pdf**

4. Munir, R. 2004. *Pengolahan Citra Digital dengan Pendekatan Algoritmik.* Bandung:Informatika.

5. Munir, Rinaldi. **Pengolahan Citra Digital.** Informatika. Bandung. 2004.

6. Sugiharto, A. 2006. *Pemrograman GUI dengan MATLAB*. Yogyakarta : CV Andi Offset.

7. Sigit, R, *et al*. 2005. *Step by Step Pengolahan Citra Digital.* Yogyakarta : CV Andi Offset.

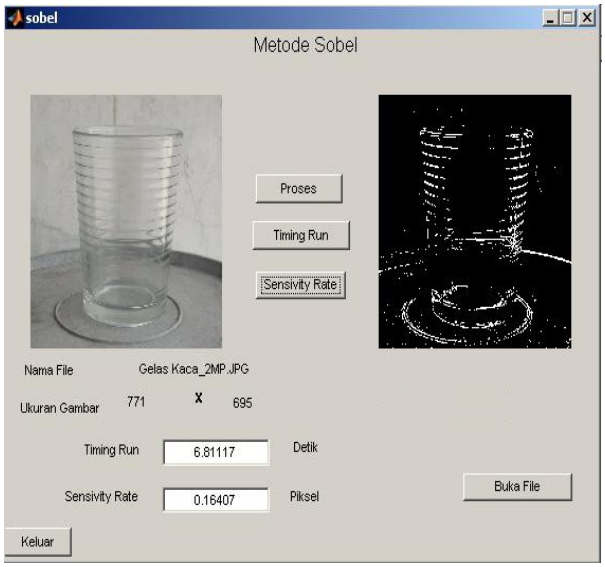
8. Otsu, N. **A Threshold Selection Method from Gray-Level Histograms**. IEEE Transactions on Systems, Man, and Cybernetics. vol. 9, no. 1. pp. 62-66, 1979.

LAMPIRAN

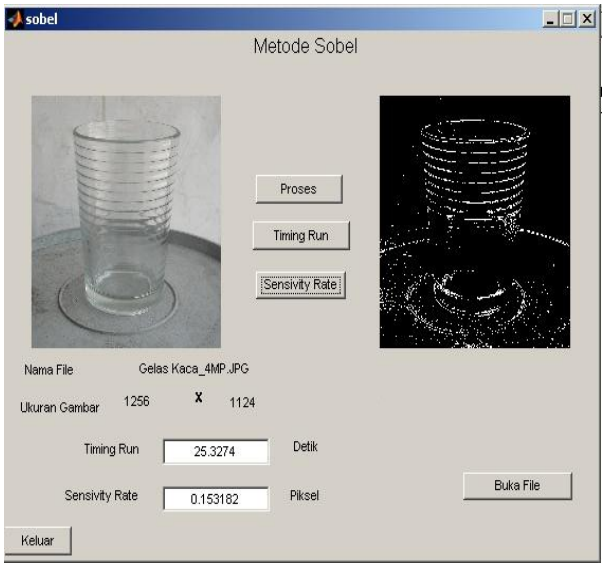
LAMPIRAN 1 :

Hasil gambar deteksi tepi pada metode *Sobel*

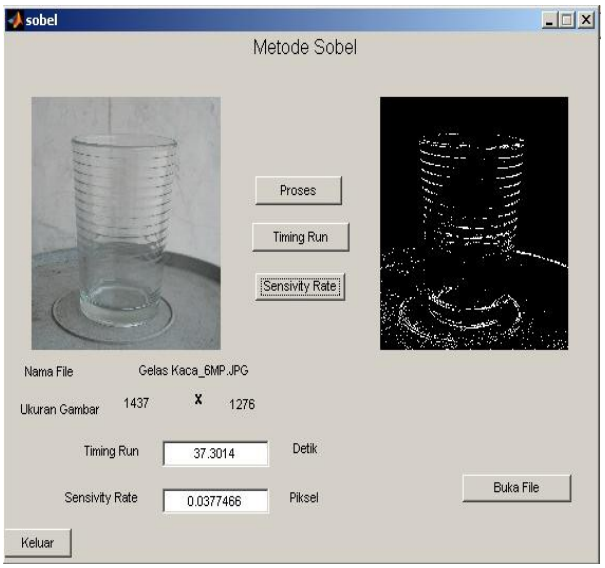
Gelas Kaca - JPG



Gelas kaca_2MP.JPG
(Untuk Gelas Kaca resolusi 2MegaPiksel)

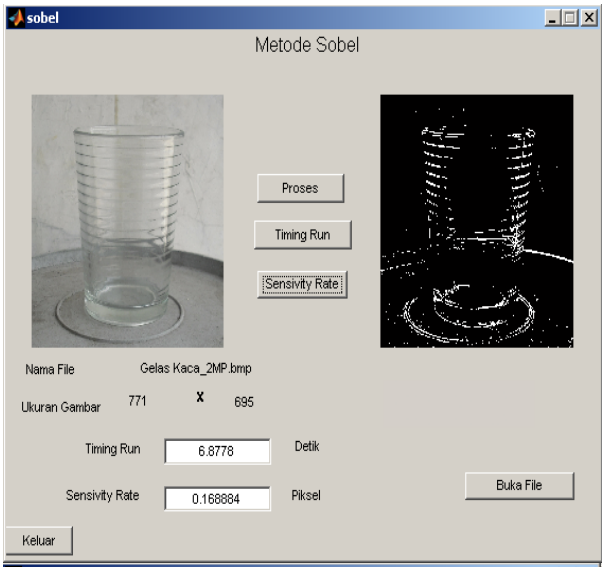


Gelas kaca_4MP.JPG
(Untuk Gelas Kaca resolusi 4MegaPiksel)

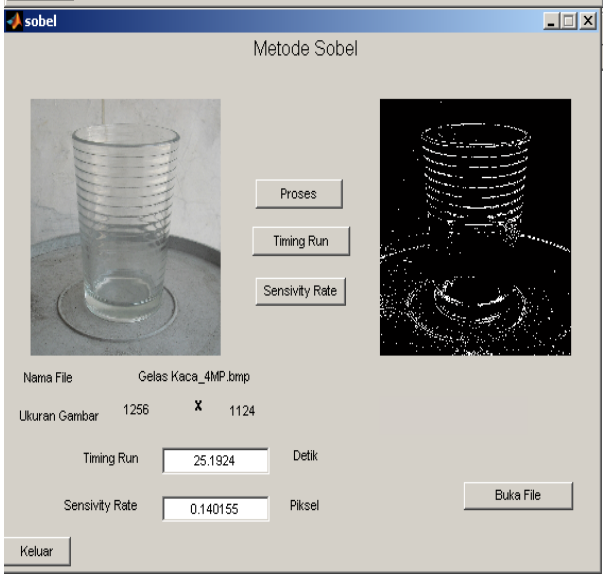


Gambar A.1.2.1.3. Gelas Kaca_6MP.JPG(Untuk Gelas Kaca resolusi 6MegaPiksel)

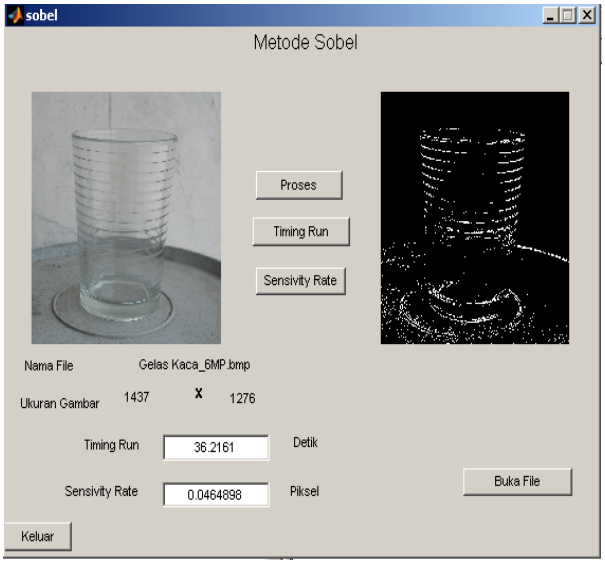
Gelas Kaca - BMP



Gelas Kaca_2MP.BMP (Untuk Gelas Kaca resolusi 2MegaPiksel)

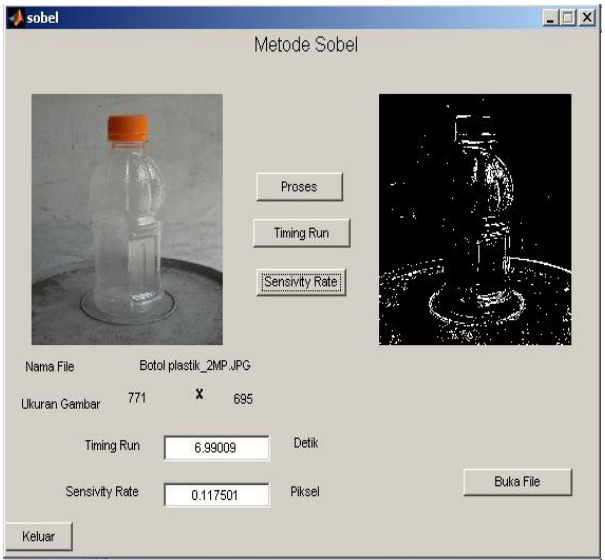


Gambar A.1.2.2.5. Gelas Kaca_4MP.BMP(Untuk Gelas Kaca resolusi 4MegaPiksel)

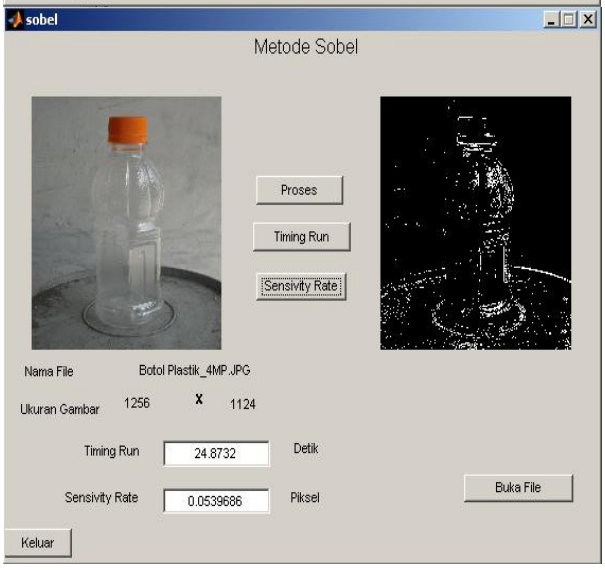


Gambar A.1.2.2.6. Gelas Kaca_6MP.BMP(Untuk Gelas Kaca resolusi 6MegaPiksel)

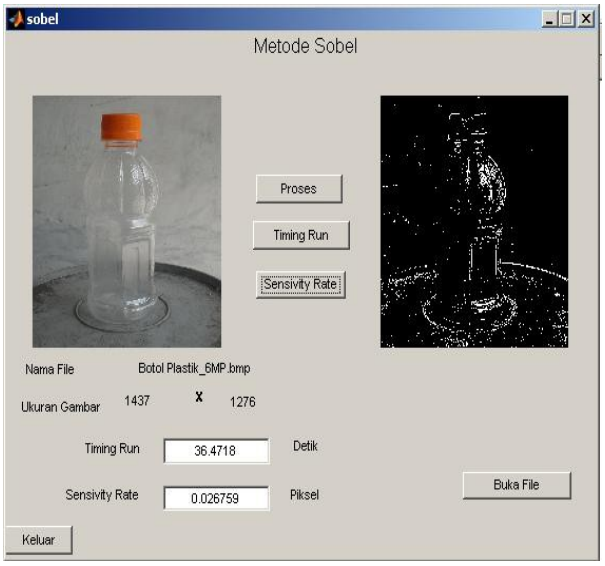
Botol Plastik - JPG



Gelas Kaca_2MP.JPG (Untuk Gelas Kaca resolusi 2MegaPiksel)

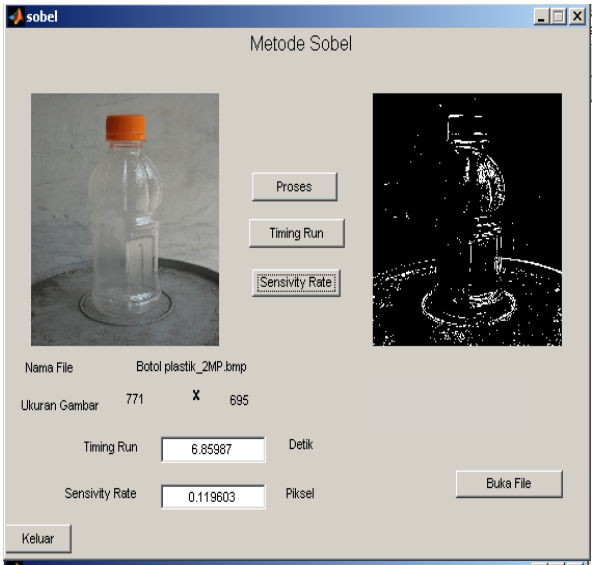


Gelas Kaca_4MP.JPG (Untuk Gelas Kaca resolusi 4MegaPiksel)

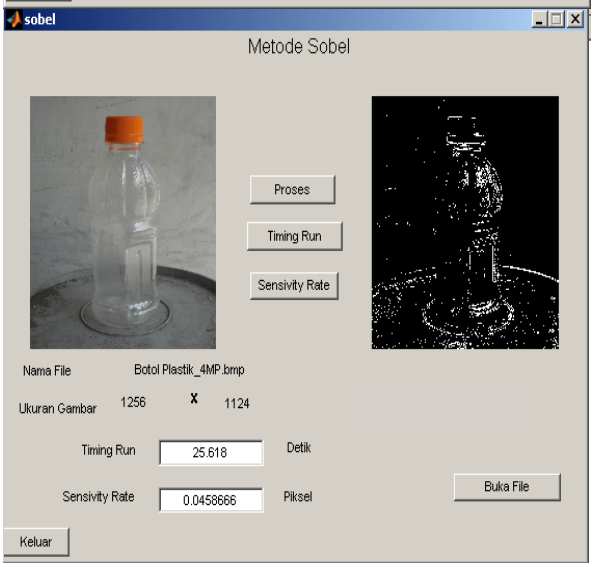


Gelas Kaca_6MP.JPG
(Untuk Gelas Kaca resolusi 6MegaPiksel)

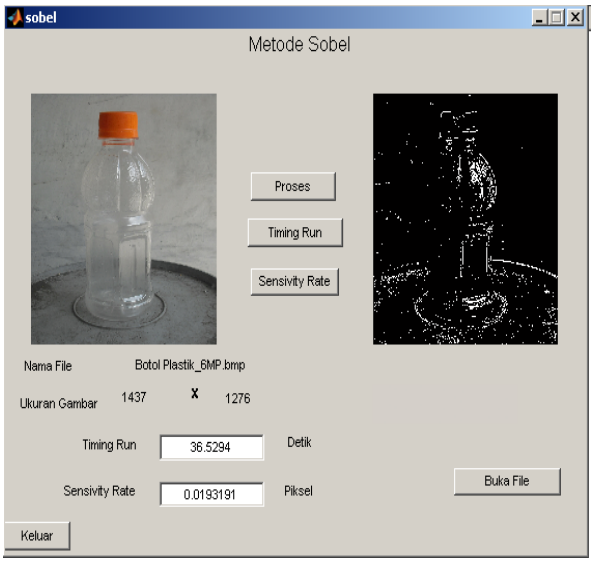
Botol Plastik - BMP



Gelas Kaca_2MP.BMP
(Untuk Gelas Kaca resolusi 2MegaPiksel)

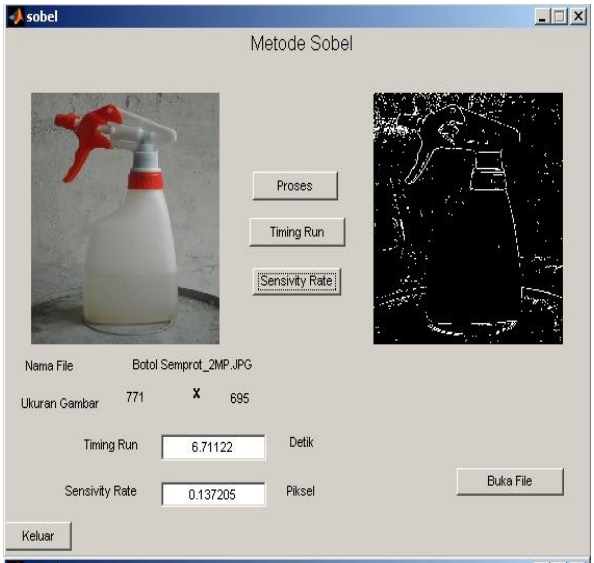


Gelas Kaca_4MP.BMP
(Untuk Gelas Kaca resolusi 4MegaPiksel)

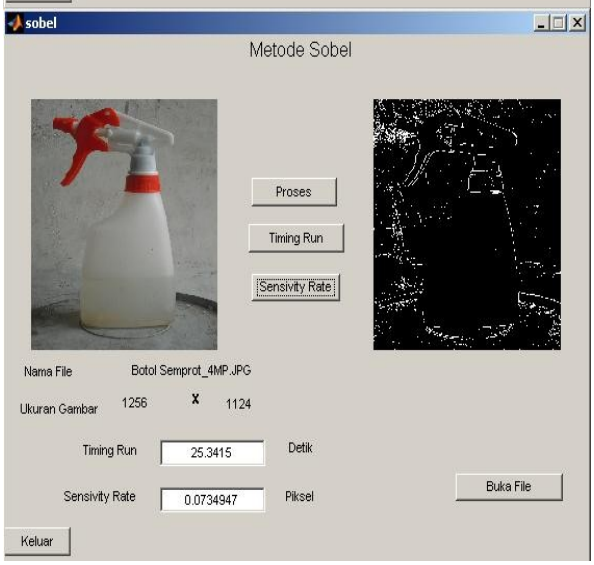


Gelas Kaca_6MP.BMP
(Untuk Gelas Kaca resolusi 6MegaPiksel)

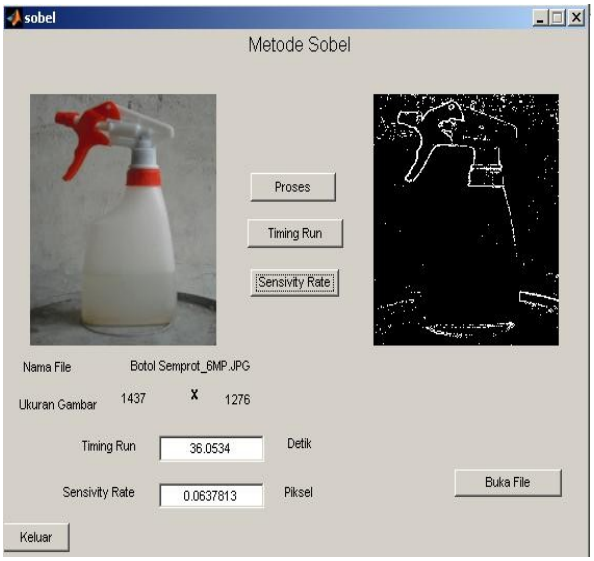
Botol Semprot - JPG



Botol Semprot _2MP.JPG
(Untuk Gelas Kaca resolusi 2MegaPiksel)

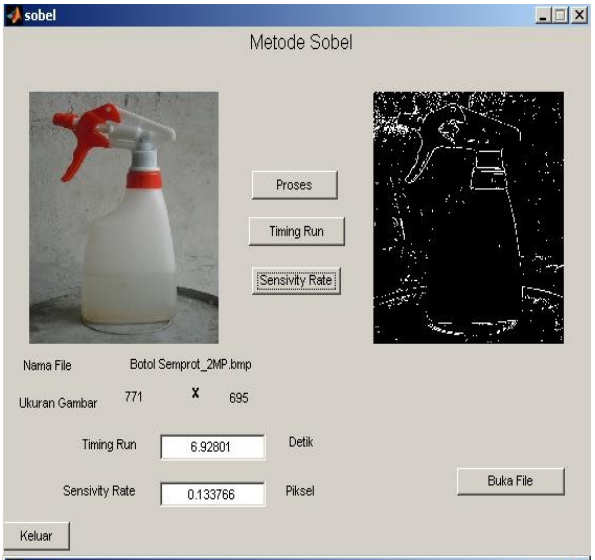


Botol Semprot _4MP.JPG
(Untuk Gelas Kaca resolusi 4MegaPiksel)

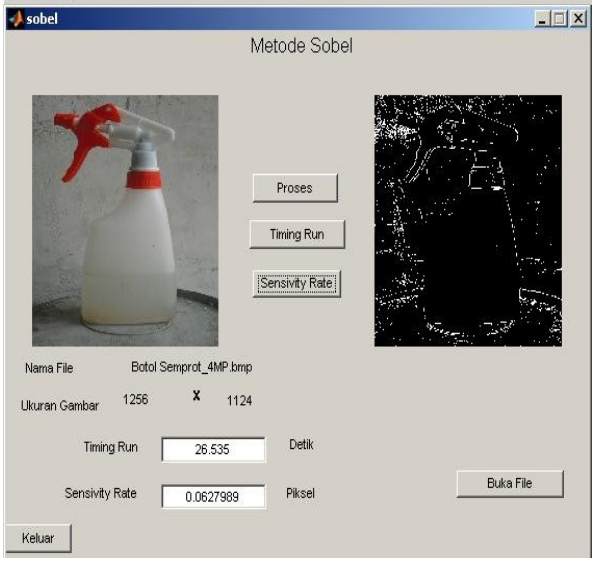


Botol Semprot _6MP.JPG (Untuk Gelas Kaca resolusi 6MegaPiksel)

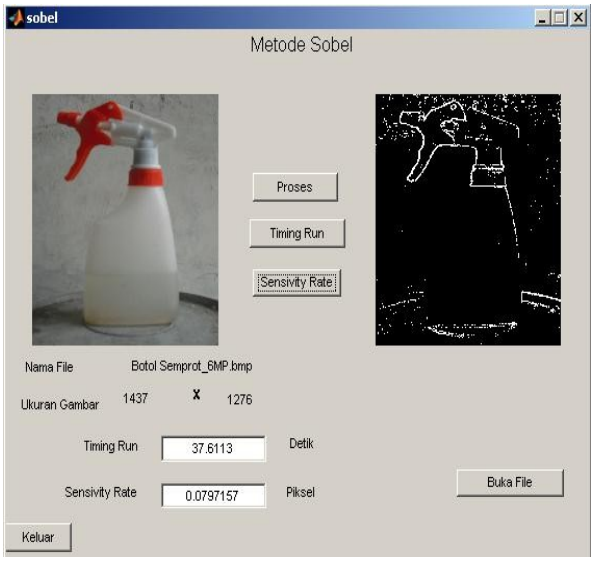
Botol Semprot - BMP



Botol Semprot _2MP.BMP (Untuk Gelas Kaca resolusi 2MegaPiksel)

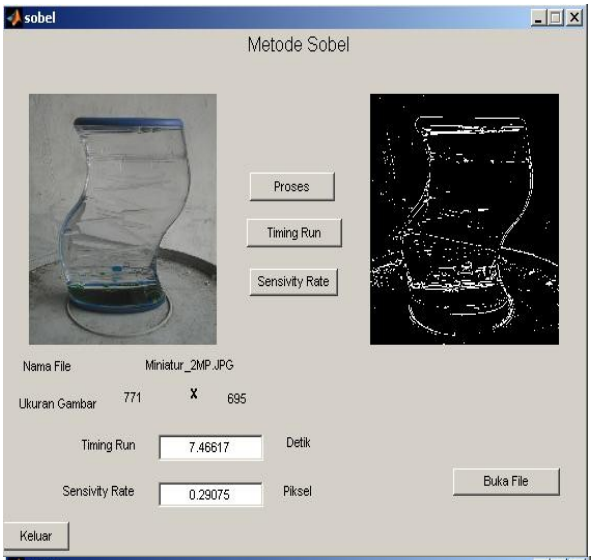


Botol Semprot _4MP.BMP (Untuk Gelas Kaca resolusi 4MegaPiksel)



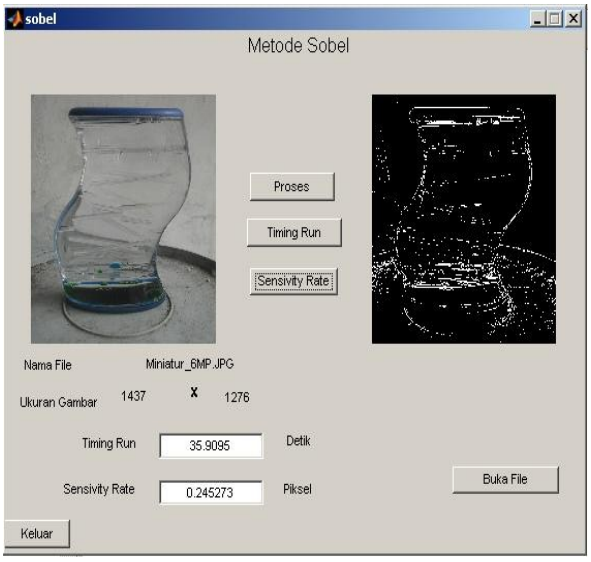
Botol Semprot _6MP.BMP (Untuk Gelas Kaca resolusi 6MegaPiksel)

Miniatur - JPG



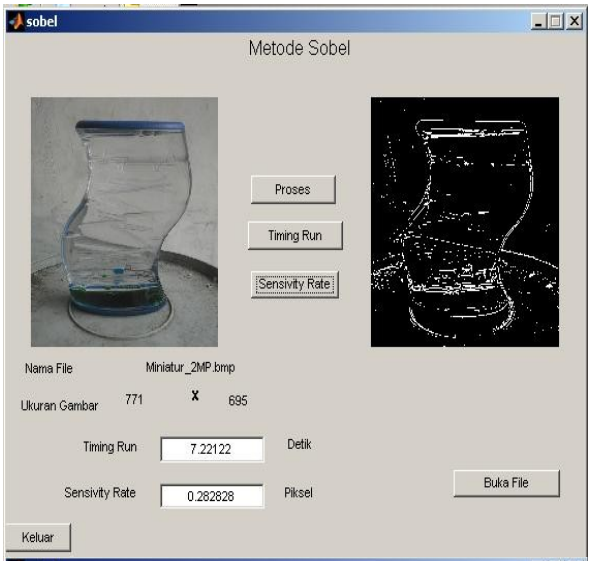
Miniatur _2MP.JPG (Untuk Miniatur resolusi 2MegaPiksel)

Miniatur _4MP.JPG (Untuk Miniatur resolusi 4MegaPiksel)

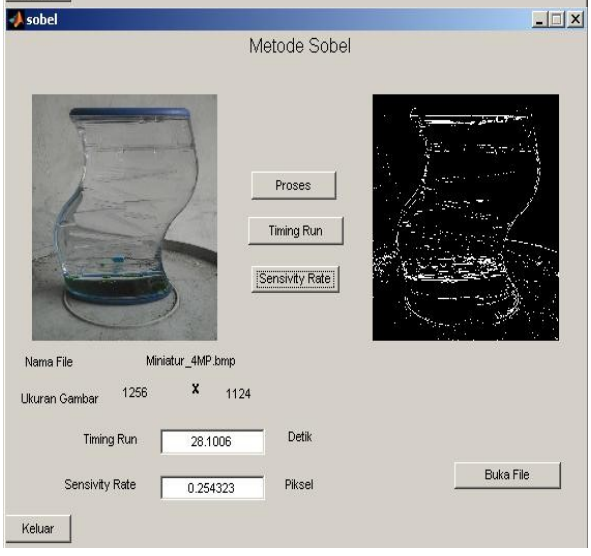


Miniatur _6MP.JPG (Untuk Miniatur resolusi 6MegaPiksel)

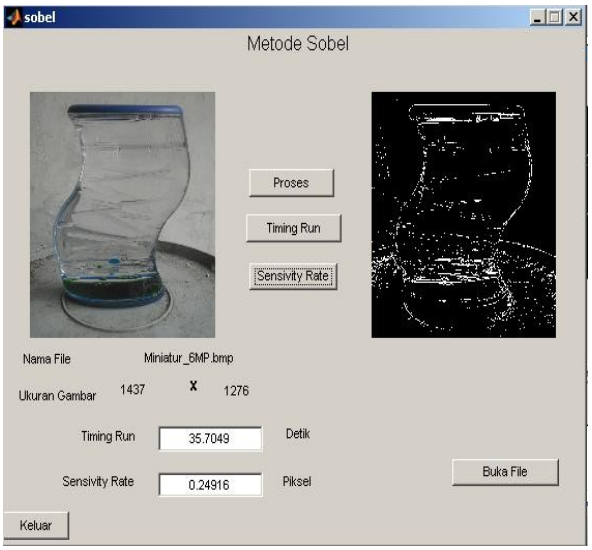
Miniatur – BMP



Miniatur _2MP.JPG (Untuk Miniatur resolusi 2MegaPiksel)



Miniatur _4MP.JPG (Untuk Miniatur resolusi 4MegaPiksel)

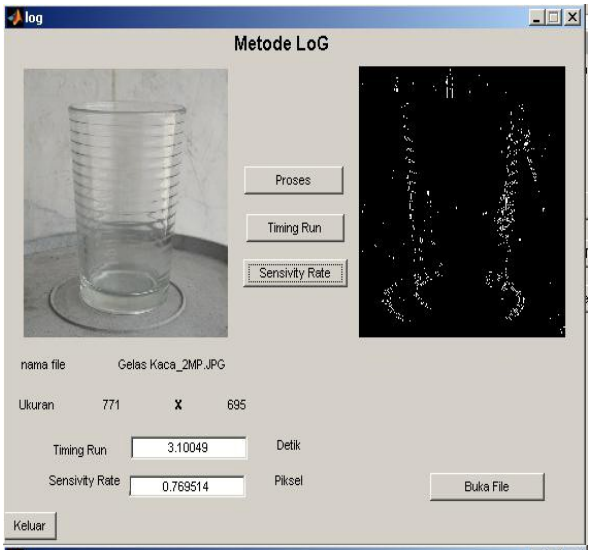


Miniatur _6MP.JPG (Untuk Miniatur resolusi 6MegaPiksel)

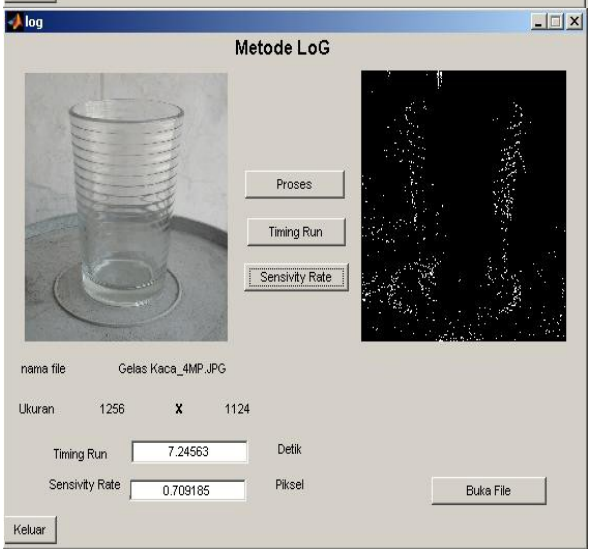
LAMPIRAN 2 :

Hasil gambar deteksi tepi pada metode *LoG (Laplacian and Gaussian)*

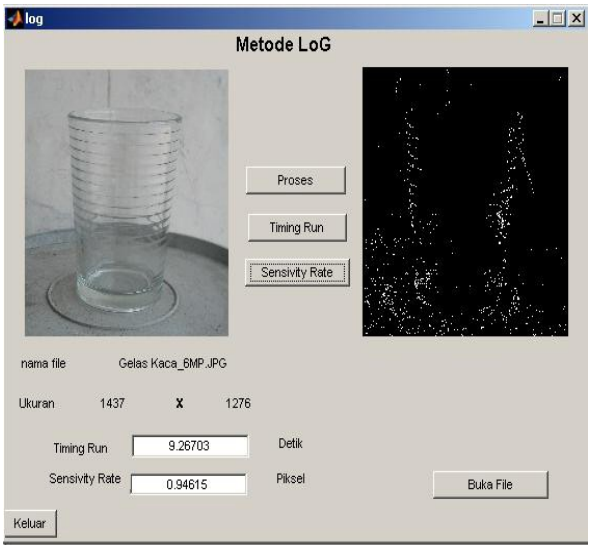
Gelas Kaca – JPG



Gelas kaca _2MP.JPG
(Untuk Gelas kaca resolusi 2MegaPiksel)

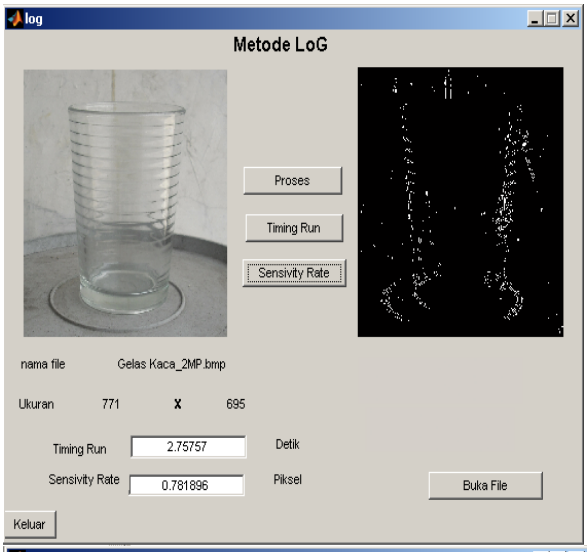


Gelas kaca _4MP.JPG
(Untuk Gelas kaca resolusi 4MegaPiksel)

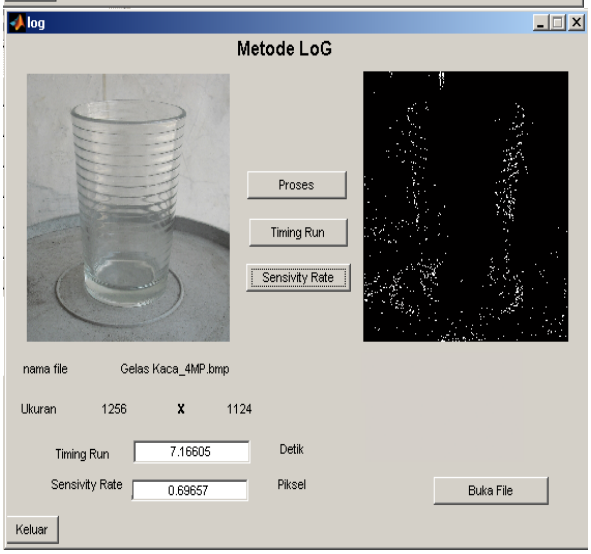


Gelas kaca _6MP.JPG
(Untuk Gelas kaca resolusi 6MegaPiksel)

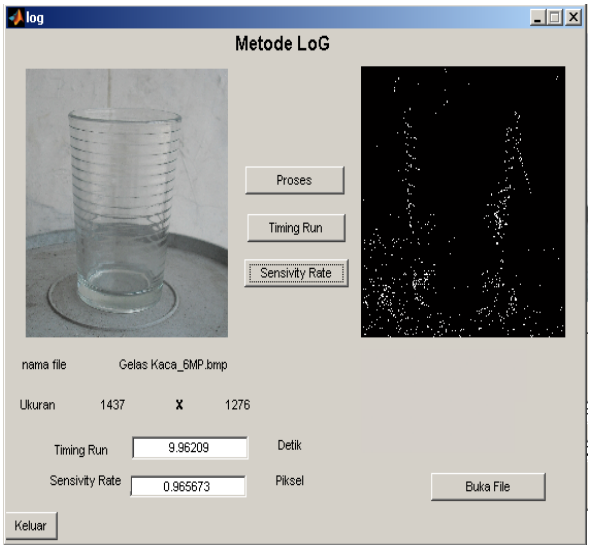
Gelas Kaca – BMP



Gelas kaca _2MP.BMP
(Untuk Gelas kaca resolusi 2MegaPiksel)

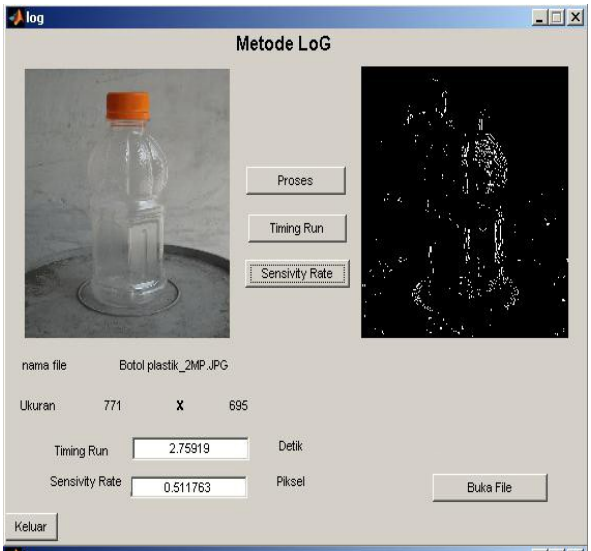


Gelas kaca _4MP.BMP
(Untuk Gelas kaca resolusi 4MegaPiksel)

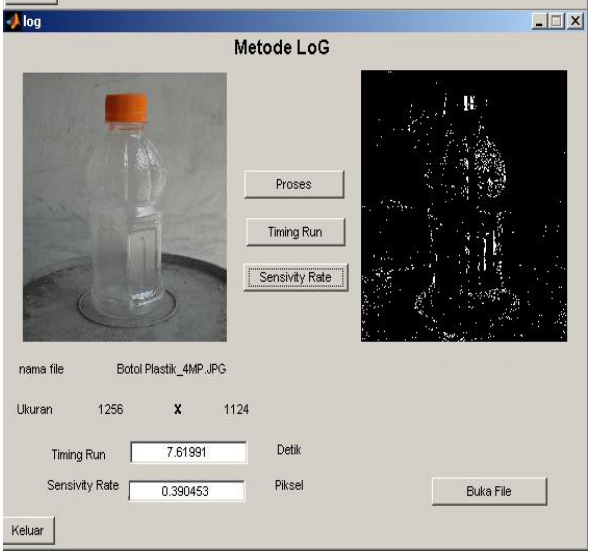


Gelas kaca _6MP.BMP
(Untuk Gelas kaca resolusi 6MegaPiksel)

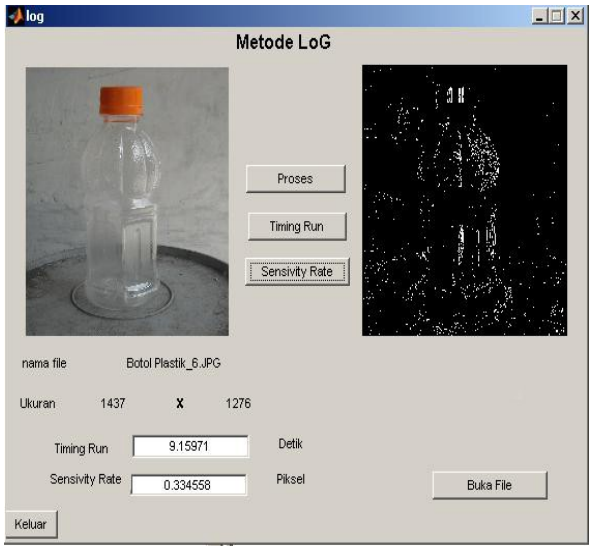
Botol Plastik – JPG



Botol Plastik_2MP.JPG
(Untuk Botol Plastik resolusi 2MegaPiksel)

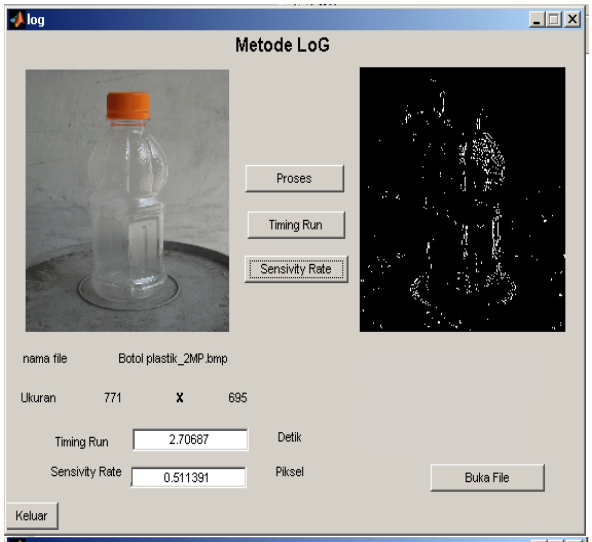


Botol Plastik_4MP.JPG
(Untuk Botol Plastik resolusi 4MegaPiksel)

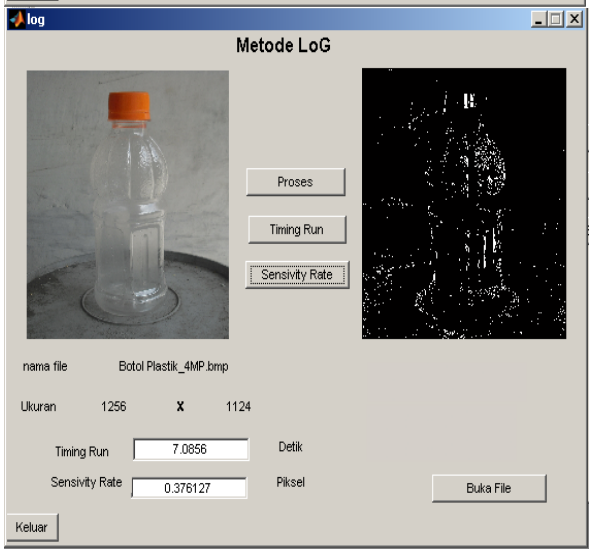


Botol Plastik_6MP.JPG
(Untuk Botol Plastik resolusi 6MegaPiksel)

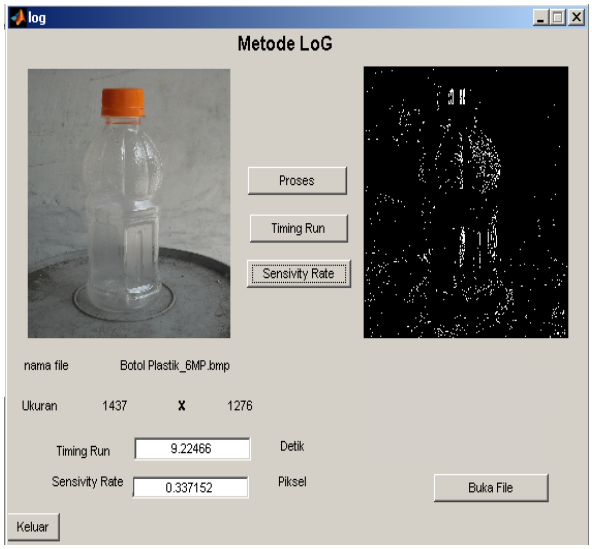
Botol Plastik – BMP



Botol Plastik_2MP.BMP
(Untuk Botol Plastik resolusi 2MegaPiksel)

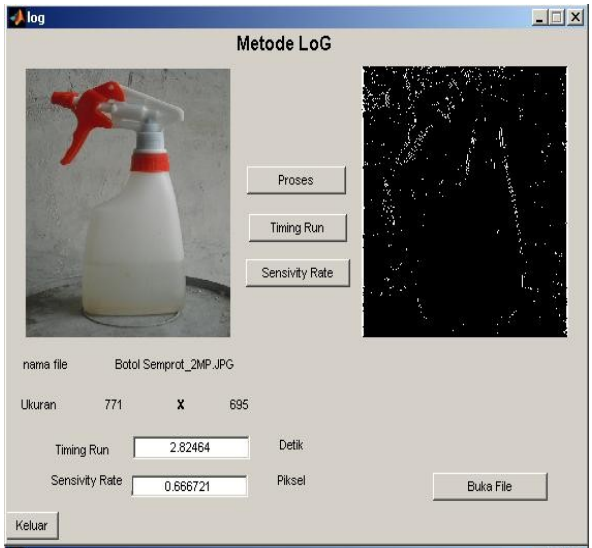


Botol Plastik_4MP.BMP
(Untuk Botol Plastik resolusi 4MegaPiksel)

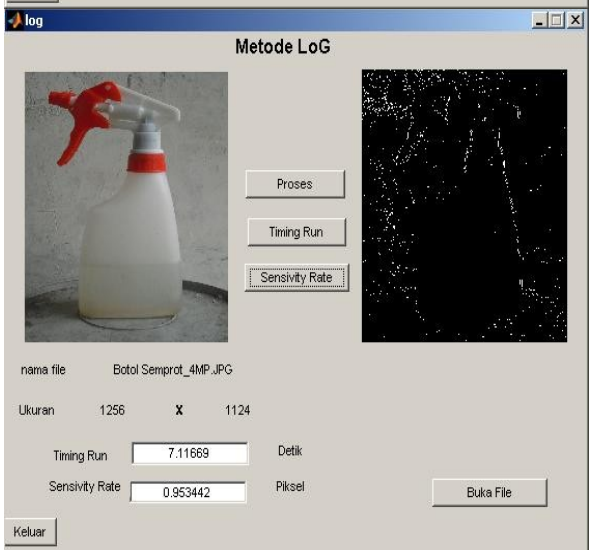


Botol Plastik_6MP.BMP
(Untuk Botol Plastik resolusi 6MegaPiksel)

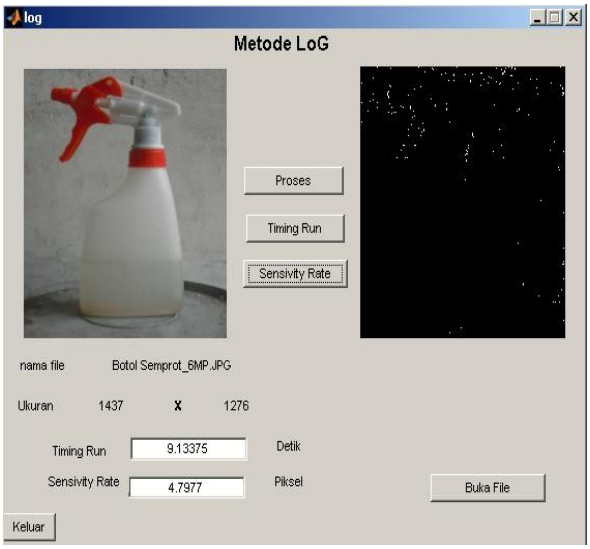
Botol Semprot – JPG



Botol Semprot _2MP.JPG
(Untuk Botol Semprot resolusi 2MegaPiksel)

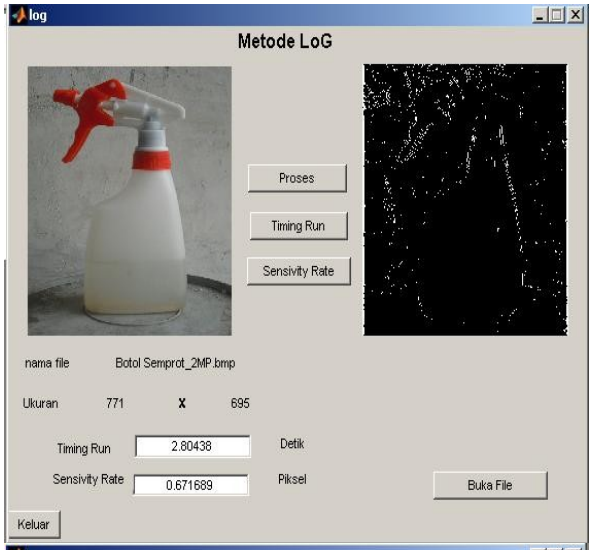


Botol Semprot _4MP.JPG
(Untuk Botol Semprot resolusi 8MegaPiksel)

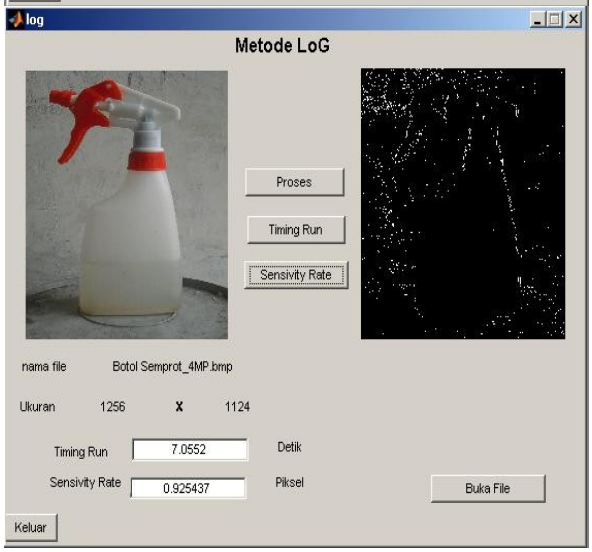


Botol Semprot _6MP.JPG (Untuk Botol Semprot resolusi 6MegaPiksel)

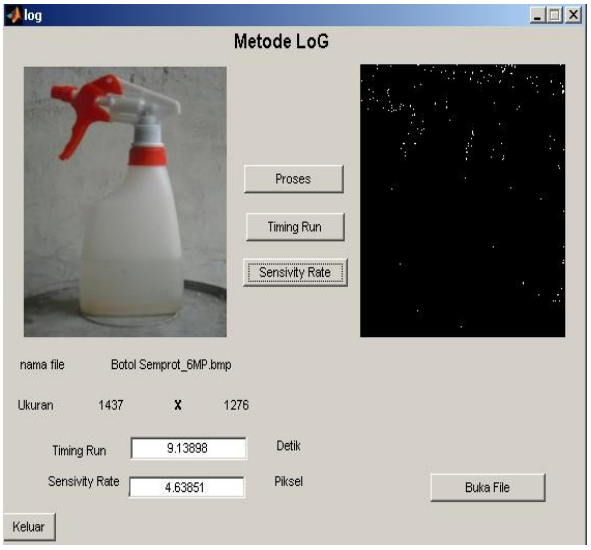
Botol Semprot – BMP



Botol Semprot _2MP.BMP (Untuk Botol Semprot resolusi 2MegaPiksel)

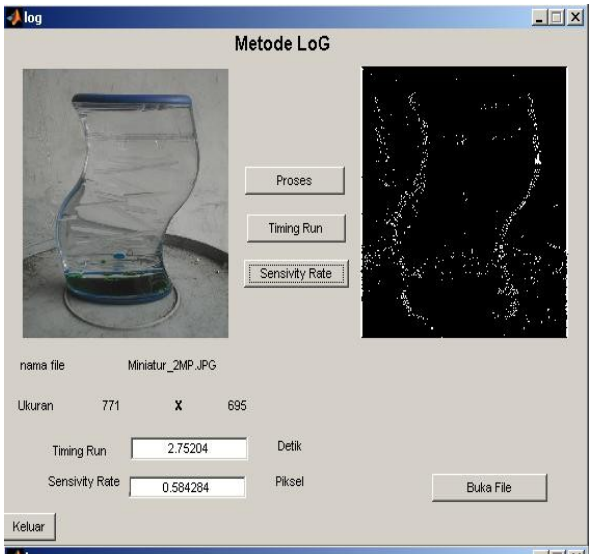


Botol Semprot _4MP.BMP (Untuk Botol Semprot resolusi 4MegaPiksel)

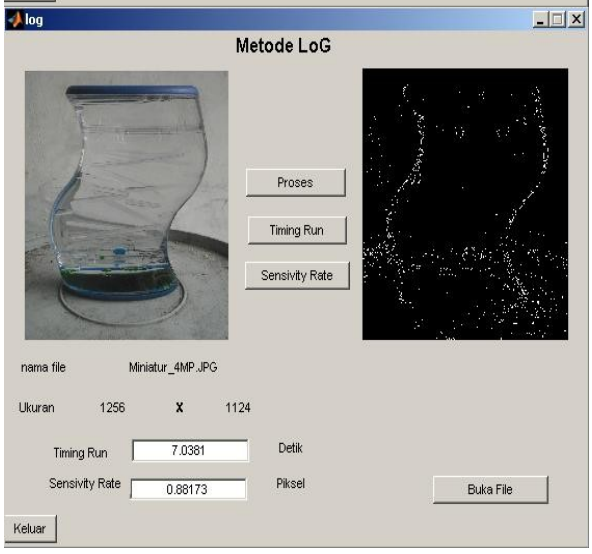


Botol Semprot _6MP.BMP (Untuk Botol Semprot resolusi 6MegaPiksel)

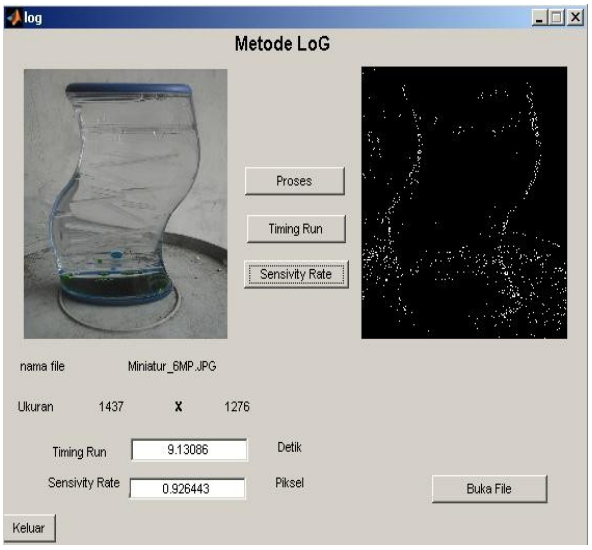
Miniatur – JPG



Miniatur _2MP.JPG (Untuk Miniatur resolusi 2MegaPiksel)

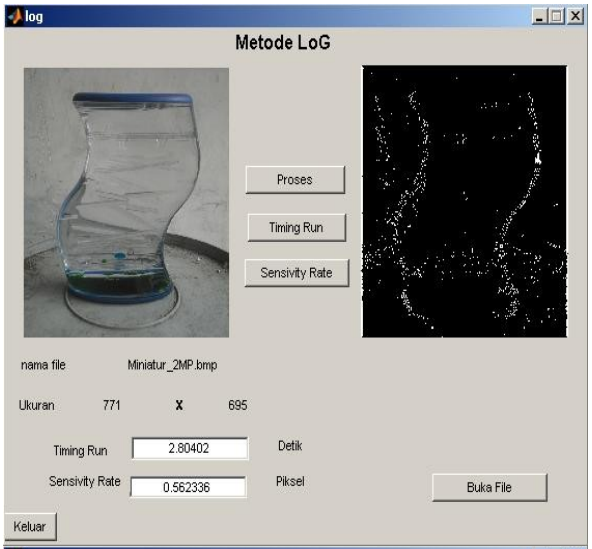


Miniatur 4MP.JPG (Untuk Miniatur resolusi 4MegaPiksel)

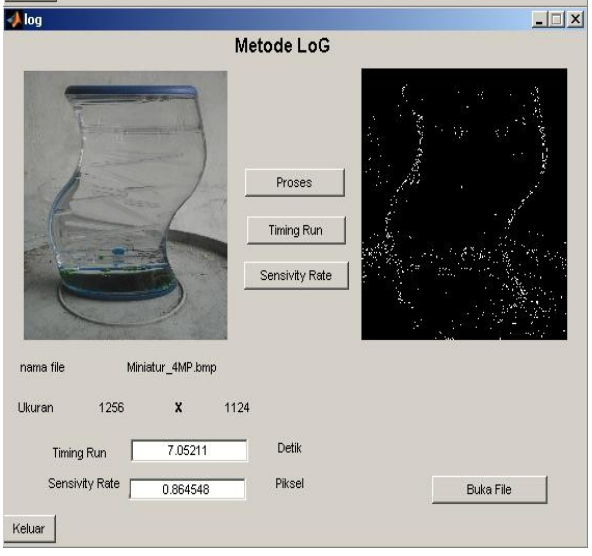


Miniatur 6MP.JPG (Untuk Miniatur resolusi 6MegaPiksel)

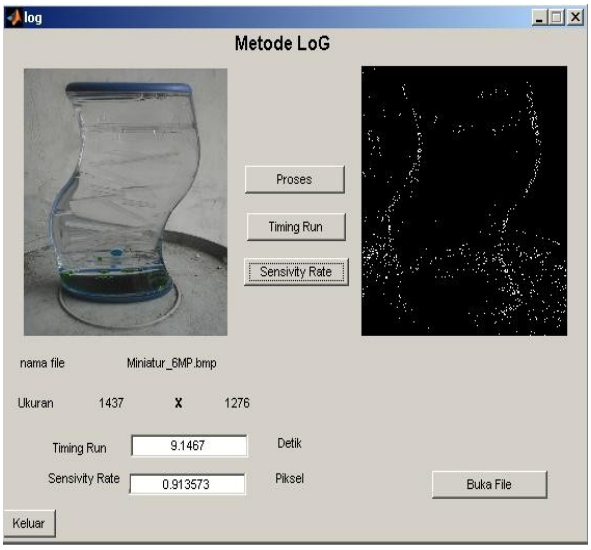
Miniatur – BMP



Miniatur _2MP.BMP (Untuk Miniatur resolusi 2MegaPiksel)



Miniatur _4MP.BMP (Untuk Miniatur resolusi 4MegaPiksel)

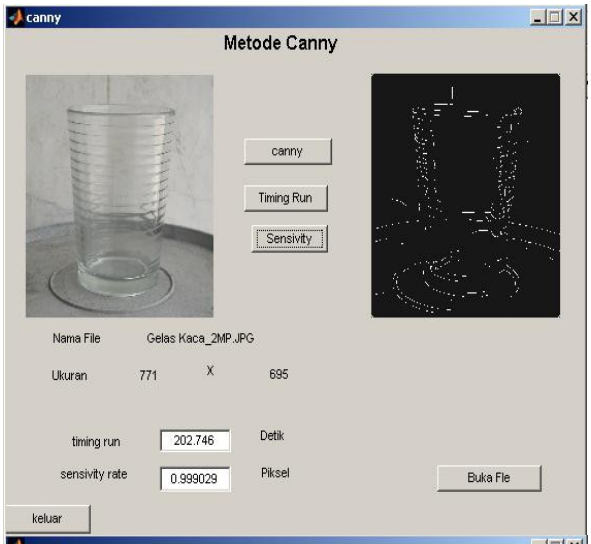


Miniatur _6MP.BMP
(Untuk Miniatur resolusi 6MegaPiksel)

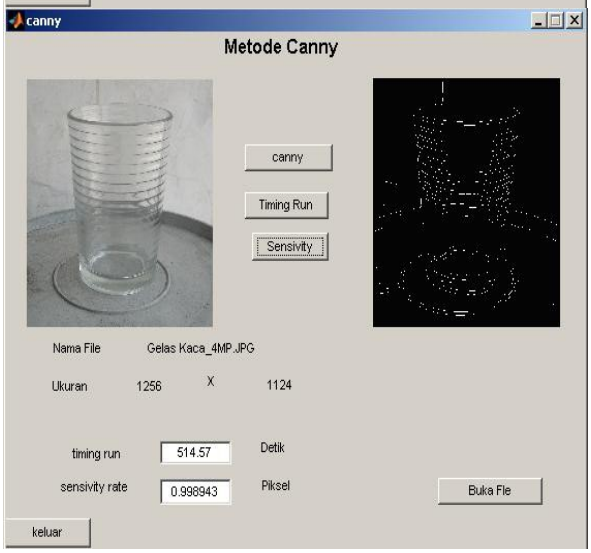
LAMPIRAN 3 :

Hasil gambar deteksi tepi pada metode *Canny*

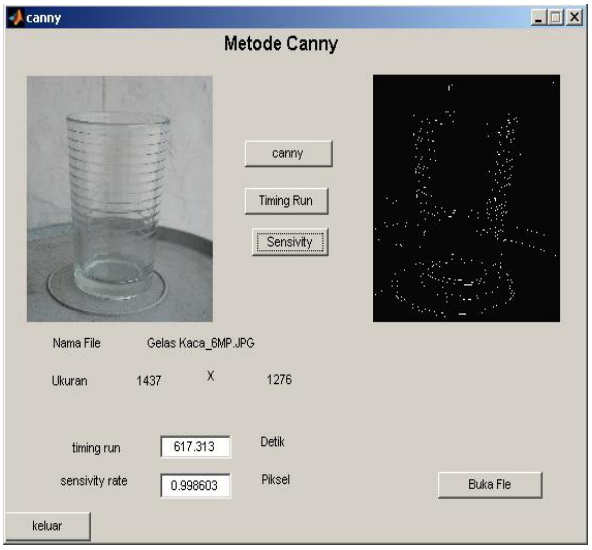
Gelas Kaca – JPG



Gelas Kaca _2MP.JPG
(Untuk Gelas Kaca resolusi 2MegaPiksel)

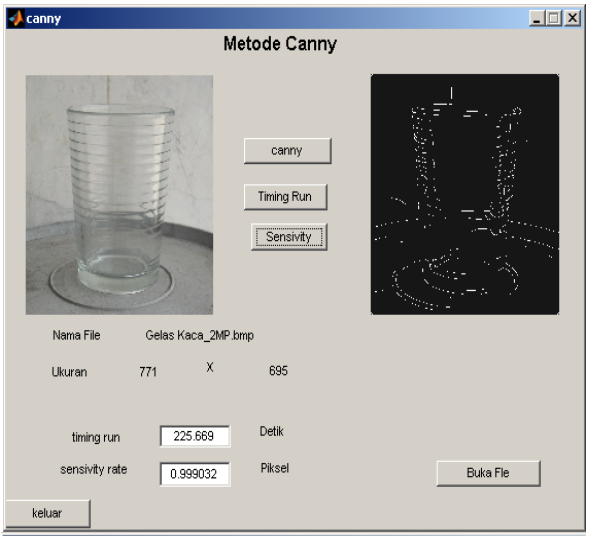


Gelas Kaca _4MP.JPG
(Untuk Gelas Kaca resolusi 4MegaPiksel)

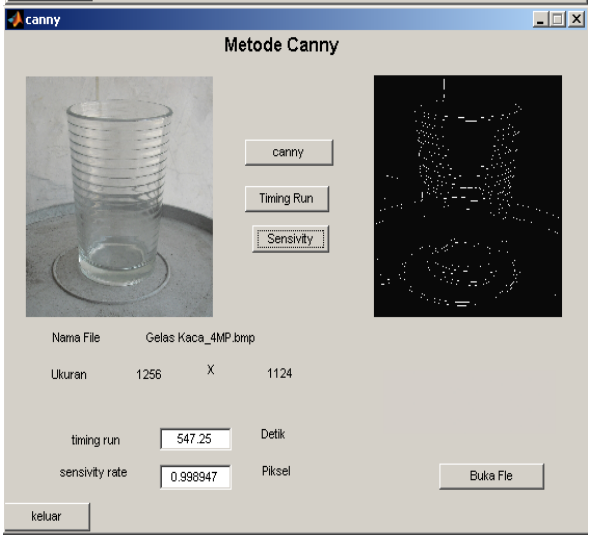


Gelas Kaca _6MP.JPG
(Untuk Gelas Kaca resolusi 6MegaPiksel)

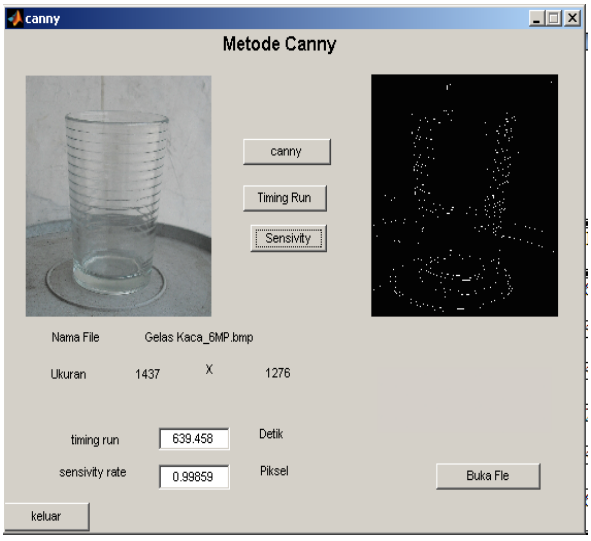
Gelas Kaca – BMP



Gelas Kaca _2MP.BMP
(Untuk Gelas Kaca resolusi 2MegaPiksel)

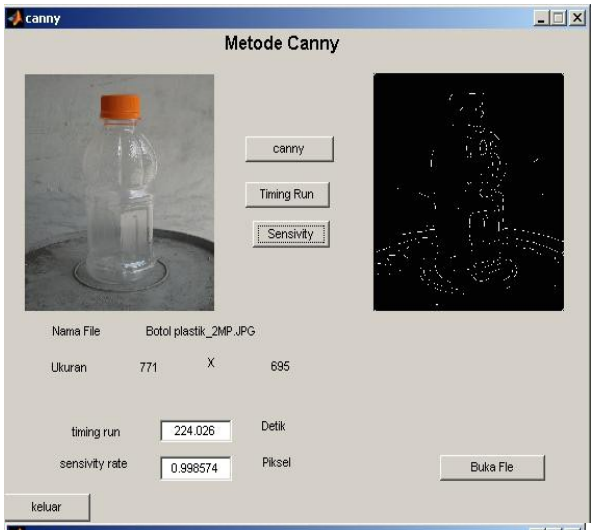


Gelas Kaca _4MP.BMP
(Untuk Gelas Kaca resolusi 4MegaPiksel)

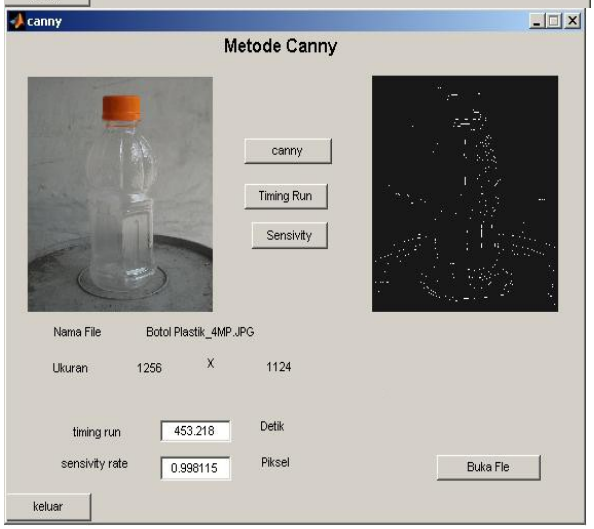


Gelas Kaca _6MP.BMP
(Untuk Gelas Kaca resolusi 6MegaPiksel)

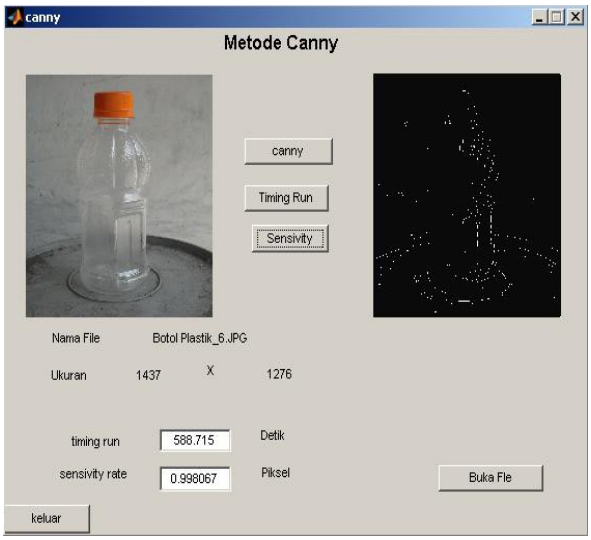
Botol plastik – JPG



Botol Plastik _2MP.JPG
(Untuk Botol Plastik resolusi 2MegaPiksel)

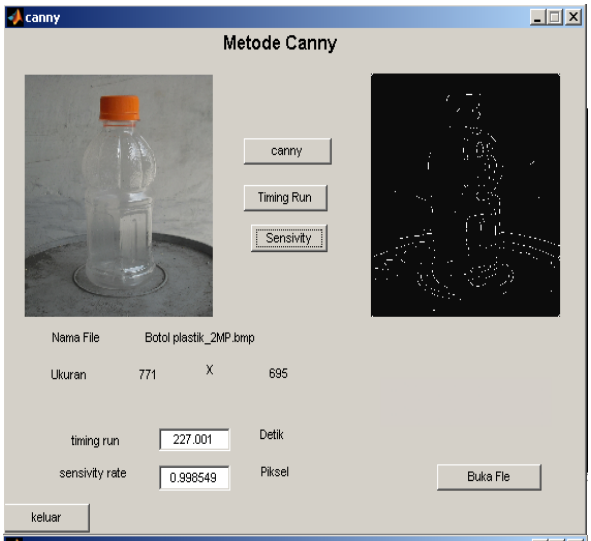


Botol Plastik _4MP.JPG
(Untuk Botol Plastik resolusi 4MegaPiksel)

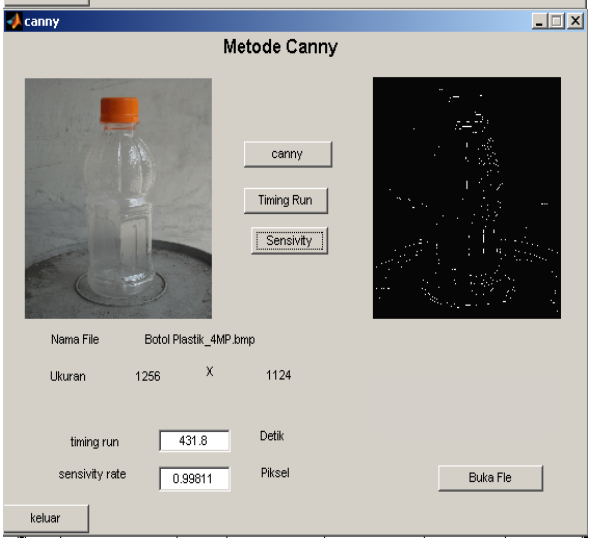


Botol Plastik _6MP.JPG (Untuk Botol Plastik resolusi 6MegaPiksel)

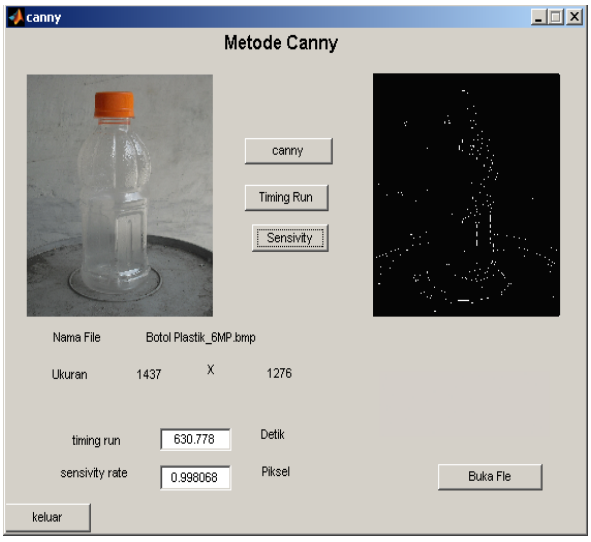
Botol plastik – BMP



Botol Plastik _2MP.BMP (Untuk Botol Plastik resolusi 2MegaPiksel)

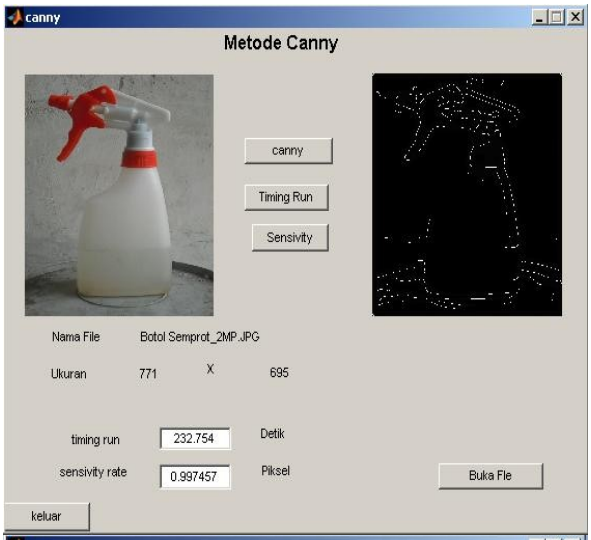


Botol Plastik _4MP.BMP (Untuk Botol Plastik resolusi 4MegaPiksel)

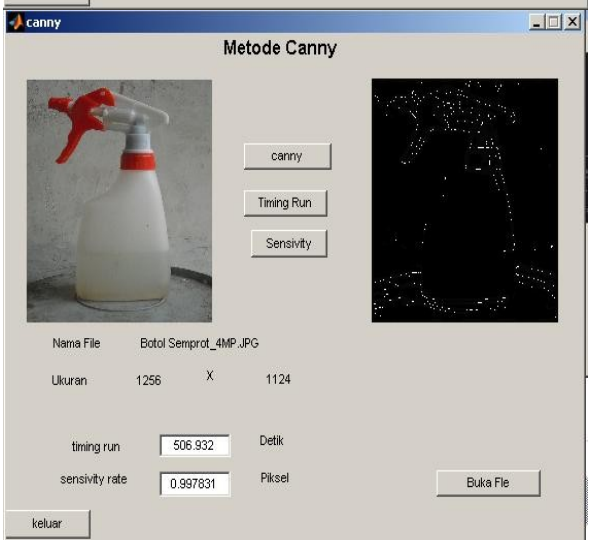


Botol Plastik _6MP.BMP
(Untuk Botol Plastik resolusi 6MegaPiksel)

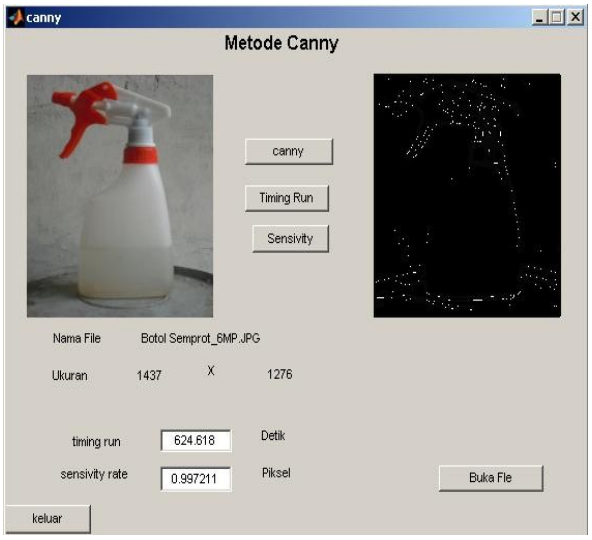
Botol plastik – JPG



Botol Semprot_2MP.JPG
(Untuk Botol Semprot resolusi 2MegaPiksel)

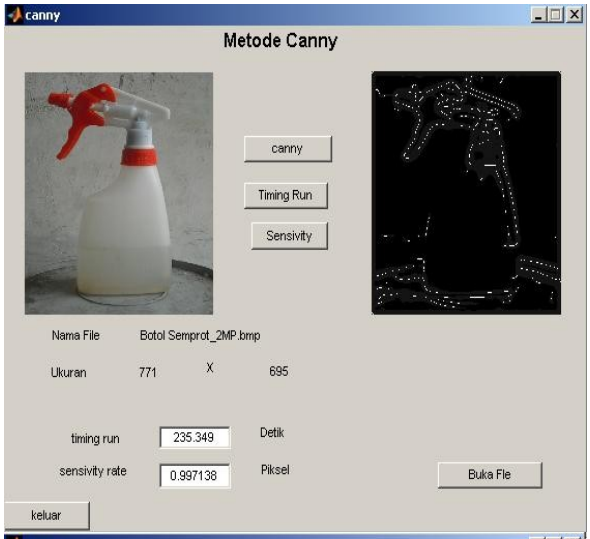


Botol Semprot_4MP.JPG
(Untuk Botol Semprot resolusi 4MegaPiksel)

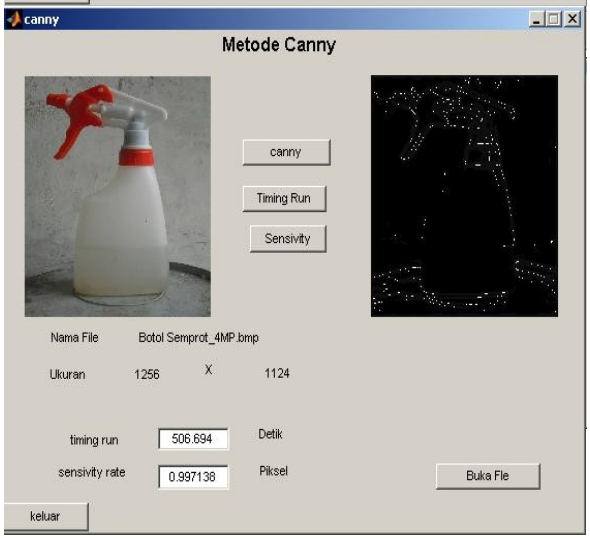


Botol Semprot_6MP.JPG (Untuk Botol Semprot resolusi 6MegaPiksel)

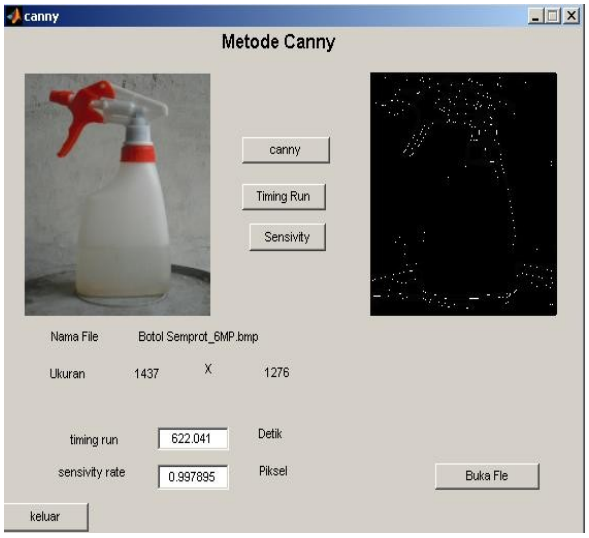
Botol plastik – BMP



Botol Semprot_2MP.BMP (Untuk Botol Semprot resolusi 2MegaPiksel)

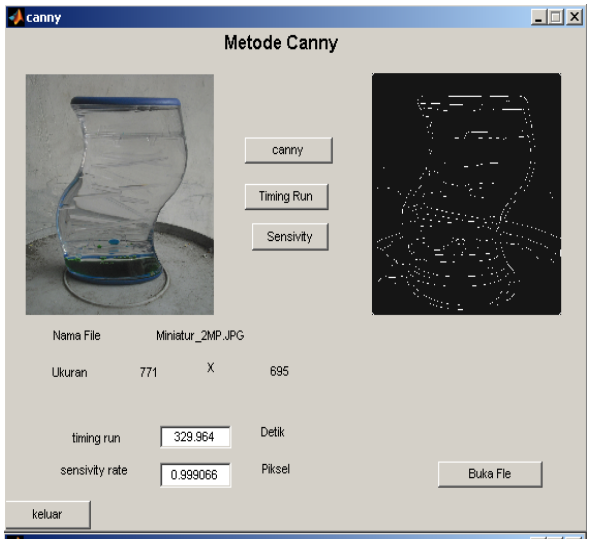


Botol Semprot_4MP.BMP (Untuk Botol Semprot resolusi 4MegaPiksel)

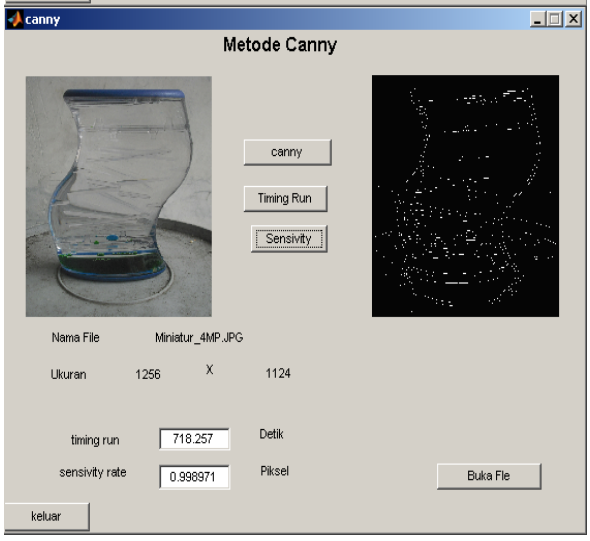


Botol Semprot_6MP.BMP (Untuk Botol Semprot resolusi 6MegaPiksel)

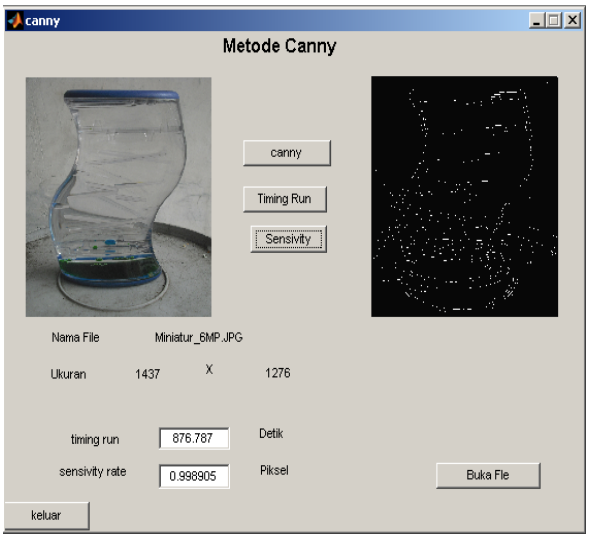
Miniatur – JPG



Miniatur_2MP.JPG (Untuk Miniatur resolusi 2MegaPiksel)

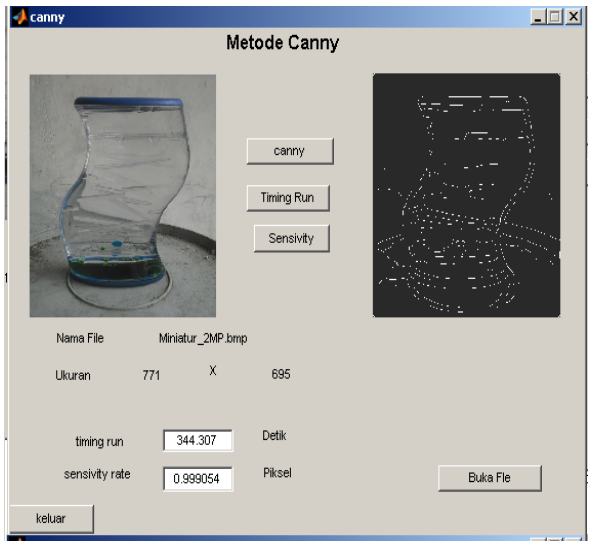


Miniatur_4MP.JPG (Untuk Miniatur resolusi 4MegaPiksel)

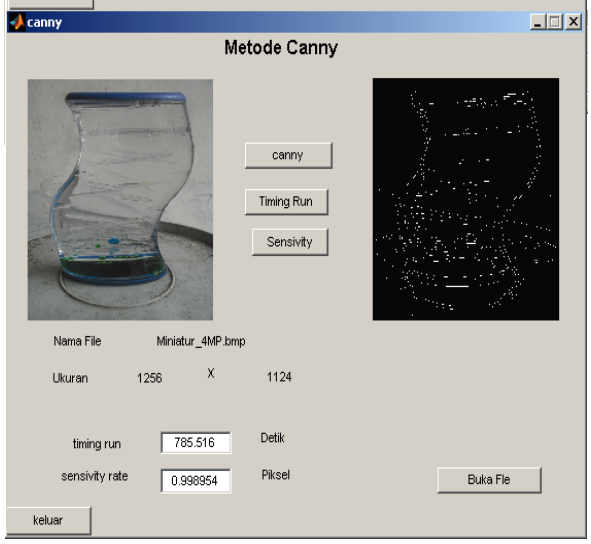


Miniatur_6MP.JPG (Untuk Miniatur resolusi 6MegaPiksel)

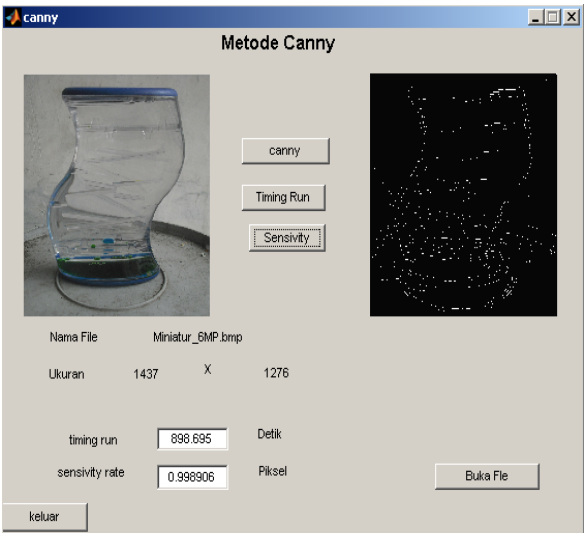
Miniatur – BMP



Miniatur_2MP.BMP (Untuk Miniatur resolusi 2MegaPiksel)



Miniatur_4MP.BMP (Untuk Miniatur resolusi 4MegaPiksel)



Miniatur_6MP.BMP (Untuk Miniatur resolusi 6MegaPiksel)

LAMPIRAN 4 :

Listing Program

Listing Program Figur Cover

```
function varargout = cover(varargin)
% COVER M-file for cover.fig
%      COVER, by itself, creates a new COVER or raises the existing
%      singleton*.
%
%      H = COVER returns the handle to a new COVER or the handle to
%      the existing singleton*.
%
%      COVER('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in COVER.M with the given input arguments.
%
%      COVER('Property','Value',...) creates a new COVER or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before cover_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to cover_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help cover

% Last Modified by GUIDE v2.5 18-Aug-2011 19:16:47

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @cover_OpeningFcn, ...
                   'gui_OutputFcn',  @cover_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before cover is made visible.
```

```
function cover_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to cover (see VARARGIN)

% Choose default command line output for cover
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes cover wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = cover_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
delete(handles.figure1);
fig2=openfig('aplikasi.fig');
handles=guihandles(fig2);
guidata(fig2,handles);
axes(handles.axes2);
imshow('aplikasi.JPG');
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in togglebutton2.
function togglebutton2_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton2

% --- Executes during object creation, after setting all properties.
function text1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% --- Executes when uipanel4 is resized.
function uipanel4_ResizeFcn(hObject, eventdata, handles)
% hObject    handle to uipanel4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
delete(handles.figure1);
fig4=openfig('profile.fig');
handles=guihandles(fig4);
guidata(fig4,handles);
axes(handles.axes1);
imshow('akbar.JPG');
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --------------------------------------------------------------------
function Profile_Callback(hObject, eventdata, handles)
% hObject    handle to Profile (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
fig5=openfig('profile.fig');
handles=guihandles(fig5);
guidata(fig5,handles);

% --- Executes during object creation, after setting all properties.
function axes3_CreateFcn(hObject, eventdata, handles)
axes(handles.axes3);
imshow('akbar.JPG');
% hObject    handle to axes3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes3

% --- Executes on button press in Keluar.
function Keluar_Callback(hObject, eventdata, handles)
delete(handles.figure1);
fig2=openfig('terima_kasih.fig');
handles=guihandles(fig2);
guidata(fig2,handles);
axes(handles.axes1);
imshow('akbar.JPG');
% hObject    handle to Keluar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Listing Program Figur Profile

```
function varargout = profile(varargin)
% PROFILE M-file for profile.fig
%      PROFILE, by itself, creates a new PROFILE or raises the existing
%      singleton*.
%
%      H = PROFILE returns the handle to a new PROFILE or the handle to
%      the existing singleton*.
%
%      PROFILE('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PROFILE.M with the given input arguments.
%
%      PROFILE('Property','Value',...) creates a new PROFILE or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before profile_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to profile_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help profile

% Last Modified by GUIDE v2.5 27-Jul-2011 15:46:46

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @profile_OpeningFcn, ...
                   'gui_OutputFcn',  @profile_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before profile is made visible.
function profile_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to profile (see VARARGIN)

% Choose default command line output for profile
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes profile wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = profile_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
delete(handles.figure1);
fig2=openfig('cover.fig');
handles=guihandles(fig2);
guidata(fig2,handles);
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes1
```

Listing Program Figur Terima Kasih

```
function varargout = terima_kasih(varargin)
% TERIMA_KASIH M-file for terima_kasih.fig
%      TERIMA_KASIH, by itself, creates a new TERIMA_KASIH or raises the existing
%      singleton*.
%
%      H = TERIMA_KASIH returns the handle to a new TERIMA_KASIH or the handle to
%      the existing singleton*.
%
%      TERIMA_KASIH('CALLBACK',hObject,eventData,handles,...) calls the local
```

```
%      function named CALLBACK in TERIMA_KASIH.M with the given input arguments.
%
%      TERIMA_KASIH('Property','Value',...) creates a new TERIMA_KASIH or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before terima_kasih_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to terima_kasih_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help terima_kasih

% Last Modified by GUIDE v2.5 18-Aug-2011 19:35:08

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @terima_kasih_OpeningFcn, ...
                   'gui_OutputFcn',  @terima_kasih_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before terima_kasih is made visible.
function terima_kasih_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to terima_kasih (see VARARGIN)

% Choose default command line output for terima_kasih
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes terima_kasih wait for user response (see UIRESUME)
% uiwait(handles.figure1);
```

```
% --- Outputs from this function are returned to the command line.
function varargout = terima_kasih_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
delete(handles.figure1);
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
delete(handles.figure1);
fig2=openfig('cover.fig');
handles=guihandles(fig2);
guidata(fig2,handles);
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Listing Program Figur Aplikasi

```
function varargout = aplikasi(varargin)
% APLIKASI M-file for aplikasi.fig
%      APLIKASI, by itself, creates a new APLIKASI or raises the existing
%      singleton*.
%
%      H = APLIKASI returns the handle to a new APLIKASI or the handle to
%      the existing singleton*.
%
%      APLIKASI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in APLIKASI.M with the given input arguments.
%
%      APLIKASI('Property','Value',...) creates a new APLIKASI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before aplikasi_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to aplikasi_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
```

```
% Edit the above text to modify the response to help aplikasi

% Last Modified by GUIDE v2.5 02-Nov-2011 11:54:53

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @aplikasi_OpeningFcn, ...
                   'gui_OutputFcn',  @aplikasi_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before aplikasi is made visible.
function aplikasi_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to aplikasi (see VARARGIN)

% Choose default command line output for aplikasi
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes aplikasi wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = aplikasi_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
fig4=openfig('sobel.fig');
handles=guihandles(fig4);
guidata(fig4,handles);
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
fig3=openfig('canny.fig');
handles=guihandles(fig3);
guidata(fig3,handles);
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
fig5=openfig('log.fig');
handles=guihandles(fig5);
guidata(fig5,handles);
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
delete(handles.figure1);
fig2=openfig('cover.fig');
handles=guihandles(fig2);
guidata(fig2,handles);
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes2
```

Listing Program Figur Sobel

```
function varargout = sobel(varargin)
% SOBEL M-file for sobel.fig
%      SOBEL, by itself, creates a new SOBEL or raises the existing
%      singleton*.
%
%      H = SOBEL returns the handle to a new SOBEL or the handle to
%      the existing singleton*.
%
%      SOBEL('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in SOBEL.M with the given input arguments.
%
%      SOBEL('Property','Value',...) creates a new SOBEL or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before sobel_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to sobel_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help sobel

% Last Modified by GUIDE v2.5 02-Feb-2012 23:53:39

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @sobel_OpeningFcn, ...
                   'gui_OutputFcn',  @sobel_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before sobel is made visible.
function sobel_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% varargin   command line arguments to sobel (see VARARGIN)

% Choose default command line output for sobel
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes sobel wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = sobel_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

function edit_tr_Callback(hObject, eventdata, handles)
% hObject    handle to edit23 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit23 as text
%        str2double(get(hObject,'String')) returns contents of edit23 as a double

% --- Executes during object creation, after setting all properties.
function edit_tr_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit23 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit_tr_Callback(hObject, eventdata, handles)
% hObject    handle to edit23 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit23 as text
%        str2double(get(hObject,'String')) returns contents of edit23 as a double

% --- Executes during object creation, after setting all properties.
```

```
function edit_tr_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit23 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in keluar.
function keluar_Callback(hObject, eventdata, handles)
delete(handles.figure1)
% hObject    handle to keluar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in buka_file.
function buka_file_Callback(hObject, eventdata, handles)
% hObject    handle to buka_file (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%--membuka File--.
[nama_file1, nama_path1]=uigetfile(...
{'*.bmp;*.jpg','File Citra(*.bmp,*.jpg)';
'*.bmp','File Bitmap (*.bmp)';...
'*.jpg','File jpeg (*.jpg)';
'*.*','Semua File (*.*)'},...
'Buka File Citra Host/Asli');
if ~isequal(nama_file1, 0)
    handles.data1=imread(fullfile(nama_path1,nama_file1));
    guidata(hObject,handles);
    axes(handles.axes3);
    imshow(handles.data1);
else
    return;
end
set(handles.text4,'String',nama_file1);
set(handles.text6,'String',size(handles.data1,1));
set(handles.text5,'String',size(handles.data1,2));

% --- Executes on button press in sobel.
function sobel_Callback(hObject, eventdata, handles)
% hObject    handle to sobel (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
I = double(rgb2gray(handles.data1));
[xlen ylen] = size(I);
edgeX = zeros(xlen,ylen);
edgeY = zeros(xlen,ylen);
edgeXY = zeros(xlen,ylen);
H1 = [-1 0 1;-2 0 2;-1 0 1];
```

```
H2 = [-1 -2 -1;0 0 0;1 2 1];
for i = 1:xlen
for j = 1:ylen
for m = 1:3
for n = 1:3
updateX = i-m+2;
updateY = j-n+2;
if ((updateX>=1) && (updateX<=xlen) && (updateY>=1)&& (updateY<=ylen))
edgeX(i,j) = edgeX(i,j) + H1(m,n)*I(updateX,updateY);
edgeY(i,j) = edgeY(i,j) + H2(m,n)*I(updateX,updateY);
end
end
end
end
end
edgeX = abs(edgeX);
edgeY = abs(edgeY);
edgeXY = edgeX.*edgeX + edgeY.*edgeY;
threshold = 4*sum(sum(edgeXY(2:xlen-1,2:ylen-1)))/((xlen-2)*(ylen-2));
for i = 2:xlen-1
for j = 2:ylen-1
if (edgeXY(i,j)>threshold)
edgeimage(i,j) = 1;
else
edgeimage(i,j) = 0;
end
end
end
edgeimage = mat2gray(edgeimage);
axes(handles.axes4);
imshow(edgeimage);

% --- Executes on button press in tr.
function tr_Callback(hObject, eventdata, handles)
% hObject    handle to tr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
tic
I = double(rgb2gray(handles.data1));
[xlen ylen] = size(I);
edgeX = zeros(xlen,ylen);
edgeY = zeros(xlen,ylen);
edgeXY = zeros(xlen,ylen);
H1 = [-1 0 1;-2 0 2;-1 0 1];
H2 = [-1 -2 -1;0 0 0;1 2 1];
for i = 1:xlen
for j = 1:ylen
for m = 1:3
for n = 1:3
updateX = i-m+2;
updateY = j-n+2;
if ((updateX>=1) && (updateX<=xlen) && (updateY>=1)&& (updateY<=ylen))
edgeX(i,j) = edgeX(i,j) + H1(m,n)*I(updateX,updateY);
```

```
edgeY(i,j) = edgeY(i,j) + H2(m,n)*I(updateX,updateY);
end
end
end
end
end
edgeX = abs(edgeX);
edgeY = abs(edgeY);
edgeXY = edgeX.*edgeX + edgeY.*edgeY;
threshold = 4*sum(sum(edgeXY(2:xlen-1,2:ylen-1)))/((xlen-2)*(ylen-2));
for i = 2:xlen-1
for j = 2:ylen-1
if (edgeXY(i,j)>threshold)
edgeimage(i,j) = 1;
else
edgeimage(i,j) = 0;
end
end
end
edgeimage = mat2gray(edgeimage);
axes(handles.axes4);
imshow(edgeimage);
t = toc;
set(handles.edit7,'string',t);

% --- Executes on button press in sensivity_rate.
function sensivity_rate_Callback(hObject, eventdata, handles)
% hObject    handle to sensivity_rate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%perhitungan error rate untuk edge detector Sobel
I = double(rgb2gray(handles.data1));
[xlen ylen] = size(I);
edgeX = zeros(xlen,ylen);
edgeY = zeros(xlen,ylen);
edgeXY = zeros(xlen,ylen);
H1 = [-1 0 1;-2 0 2;-1 0 1];
H2 = [-1 -2 -1;0 0 0;1 2 1];
for i = 1:xlen
    for j = 1:ylen
for m = 1:3
for n = 1:3
updateX = i-m+2;
updateY = j-n+2;
if ((updateX>=1) && (updateX<=xlen) && (updateY>=1)&& (updateY<=ylen))
edgeX(i,j) = edgeX(i,j) + H1(m,n)*I(updateX,updateY);
edgeY(i,j) = edgeY(i,j) + H2(m,n)*I(updateX,updateY);
end
end
end
end
end
edgeX = abs(edgeX);
```

```
edgeY = abs(edgeY);
edgeXY = edgeX.*edgeX + edgeY.*edgeY;
threshold = 4*sum(sum(edgeXY(2:xlen-1,2:ylen-1)))/((xlen-2)*(ylen-2));
for i = 2:xlen-1
for j = 2:ylen-1
if (edgeXY(i,j)>threshold)
edgeimage(i,j) = 1;
else
edgeimage(i,j) = 0;
end
end
end
edgeimage = mat2gray(edgeimage);
axes(handles.axes4);
imshow(edgeimage);
In = imnoise(I,'salt & pepper',0.02);
Jn = edge(In,'sobel');
nr = sum(sum(edgeimage));
nn = sum(sum(Jn));
P = abs(nn-nr)/nr
set(handles.edit_sr,'String', P);

function edit_sr_Callback(hObject, eventdata, handles)
% hObject    handle to edit_sr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_sr as text
%        str2double(get(hObject,'String')) returns contents of edit_sr as a double

% --- Executes during object creation, after setting all properties.
function edit_sr_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_sr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit23_Callback(hObject, eventdata, handles)
alfa=str2num(get(handles.edit23, 'string'));
handles.alfa=alfa;
guidata (hObject,handles)
% hObject    handle to edit23 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit23 as text
%        str2double(get(hObject,'String')) returns contents of edit23 as a double
```

```
% --- Executes during object creation, after setting all properties.
function edit23_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit23 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton10.
function pushbutton10_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
I = double(rgb2gray(handles.data1));
In = imnoise(I,'salt & pepper',0.02);
Jn = edge(In,'sobel');
nn = sum(sum(Jn));
E = edge(I,'sobel');
P = sum(sum(E));
R = abs(P-nn)/100;
set(handles.edit8,'String', R);

% --- Executes during object creation, after setting all properties.
function pushbutton10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pushbutton10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% --- Executes during object creation, after setting all properties.
function axes4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes4

function edit7_Callback(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit7 as text
%        str2double(get(hObject,'String')) returns contents of edit7 as a double

% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit8_Callback(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit8 as text
%        str2double(get(hObject,'String')) returns contents of edit8 as a double

% --- Executes during object creation, after setting all properties.
function edit8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function axes3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes3
```

Listing Program Figur LoG

```
function varargout = log(varargin)
% LOG M-file for log.fig
%      LOG, by itself, creates a new LOG or raises the existing
%      singleton*.
%
%      H = LOG returns the handle to a new LOG or the handle to
%      the existing singleton*.
%
%      LOG('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in LOG.M with the given input arguments.
%
%      LOG('Property','Value',...) creates a new LOG or raises the
```

```
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before log_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to log_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help log

% Last Modified by GUIDE v2.5 02-Feb-2012 23:41:13

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @log_OpeningFcn, ...
                   'gui_OutputFcn',  @log_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before log is made visible.
function log_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to log (see VARARGIN)

% Choose default command line output for log
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes log wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = log_OutputFcn(hObject, eventdata, handles)
```

```
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in keluar.
function keluar_Callback(hObject, eventdata, handles)
delete(handles.figure1)
% hObject    handle to keluar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in buka_file.
function buka_file_Callback(hObject, eventdata, handles)
% hObject    handle to buka_file (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[nama_file1, nama_path1]=uigetfile(...
{'*.bmp;*.jpg','File Citra(*.bmp,*.jpg)';
'*.bmp','File Bitmap (*.bmp)';...
'*.jpg','File jpeg (*.jpg)';
'*.*','Semua File (*.*)'},...
'Buka File Citra Host/Asli');
if ~isequal(nama_file1, 0)
handles.data1=imread(fullfile(nama_path1,nama_file1));
guidata(hObject,handles);
handles.current_data1=handles.data1;
axes(handles.axes50);
imshow(handles.current_data1);
else
return;
end
set(handles.text4,'String',nama_file1);
set(handles.text6,'String',size(handles.data1,1));
set(handles.text5,'String',size(handles.data1,2));

% --- Executes on button press in proses.
function proses_Callback(hObject, eventdata, handles)
% hObject    handle to proses (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Function "d2dgauss.m":
% This function returns a 2D edge detector (first order derivative
% of 2D Gaussian function) with size n1*n2; theta is the angle that
% the detector rotated counter clockwise; and sigma1 and sigma2 are
the
% standard deviation of the gaussian functions.
function h = d2dgauss(n1,sigma1,n2,sigma2,theta)
r=[cos(theta) -sin(theta);
sin(theta) cos(theta)];
```

```
for i = 1 : n2
for j = 1 : n1
u = r * [j-(n1+1)/2 i-(n2+1)/2]';
h(i,j) = gauss(u(1),sigma1)*dgauss(u(2),sigma2);
end
end
h = h / sqrt(sum(sum(abs(h).*abs(h))));
% Function "gauss.m":
function y = gauss(x,std)
y = exp(-x^2/(2*std^2)) / (std*sqrt(2*pi));
% Function "dgauss.m"(first order derivative of gauss function):
function y = dgauss(x,std)
y = -x * gauss(x,std) / std^2;

% --- Executes on button press in LoG_proses.
function LoG_proses_Callback(hObject, eventdata, handles)
% hObject    handle to LoG_proses (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% The algorithm parameters:
% 1. Parameters of edge detecting filters:
% X-axis direction filter:
Nx1=7;Sigmax1=0.5;Nx2=7;Sigmax2=0.5;Theta1=pi/2;
% Y-axis direction filter:
Ny1=7;Sigmay1=0.5;Ny2=7;Sigmay2=0.5;Theta2=0;
% 2. Thresholdong value
alfa = 4;
I = double(rgb2gray(handles.data1));
% X-axis direction edge detection
filterx=d2dgauss(Nx1,Sigmax1,Nx2,Sigmax2,Theta1);
Ix= conv2(I,filterx,'same');
% Y-axis direction edge detection
filtery=d2dgauss(Ny1,Sigmay1,Ny2,Sigmay2,Theta2);
Iy=conv2(I,filtery,'same');
% Norm of the gradient (Combining the X and Y directional derivatives)
NVI=sqrt(Ix.*Ix+Iy.*Iy);
% Convolve with laplacian mask
H=[0 1 0,1 -4 1,0 1 0];
J=conv2(NVI,H,'same');
% Find the zerocross with thresh value
thresh =.2*mean2(abs(J));
edgeimage = edge(J,'zerocross',thresh);
axes(handles.axes4);
imshow(edgeimage);

% --- Executes on button press in proses_tr.
function proses_tr_Callback(hObject, eventdata, handles)
% hObject    handle to proses_tr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
tic
% The algorithm parameters:
% 1. Parameters of edge detecting filters:
```

```
% X-axis direction filter:
Nx1=7;Sigmax1=0.5;Nx2=7;Sigmax2=0.5;Theta1=pi/2;
% Y-axis direction filter:
Ny1=7;Sigmay1=0.5;Ny2=7;Sigmay2=0.5;Theta2=0;
% 2. Thresholdong value
alfa = 4;
I = double(rgb2gray(handles.data1));
% X-axis direction edge detection
filterx=d2dgauss(Nx1,Sigmax1,Nx2,Sigmax2,Theta1);
Ix= conv2(I,filterx,'same');
% Y-axis direction edge detection
filtery=d2dgauss(Ny1,Sigmay1,Ny2,Sigmay2,Theta2);
Iy=conv2(I,filtery,'same');
% Norm of the gradient (Combining the X and Y directional derivatives)
NVI=sqrt(Ix.*Ix+Iy.*Iy);
% Convolve with laplacian mask
H=[0 1 0,1 -4 1,0 1 0];
J=conv2(NVI,H,'same');
% Find the zerocross with thresh value
thresh =.2*mean2(abs(J));
edgeimage = edge(J,'zerocross',thresh);
axes(handles.axes4);
imshow(edgeimage);
t = toc;
set(handles.edit5,'string',t);

% --- Executes on button press in proses_sr.
function proses_sr_Callback(hObject, eventdata, handles)
% hObject    handle to proses_sr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% The algorithm parameters:
% 1. Parameters of edge detecting filters:
% X-axis direction filter:
Nx1=7;Sigmax1=0.5;Nx2=7;Sigmax2=0.5;Theta1=pi/2;
% Y-axis direction filter:
Ny1=7;Sigmay1=0.5;Ny2=7;Sigmay2=0.5;Theta2=0;
% 2. Thresholdong value
alfa = 4;
I = double(rgb2gray(handles.data1));
% X-axis direction edge detection
filterx=d2dgauss(Nx1,Sigmax1,Nx2,Sigmax2,Theta1);
Ix= conv2(I,filterx,'same');
% Y-axis direction edge detection
filtery=d2dgauss(Ny1,Sigmay1,Ny2,Sigmay2,Theta2);
Iy=conv2(I,filtery,'same');
% Norm of the gradient (Combining the X and Y directional derivatives)
NVI=sqrt(Ix.*Ix+Iy.*Iy);
% Convolve with laplacian mask
H=[0 1 0,1 -4 1,0 1 0];
J=conv2(NVI,H,'same');
% Find the zerocross with thresh value
thresh =.2*mean2(abs(J));
```

```
edgeimage = edge(J,'zerocross',thresh);
axes(handles.axes4);
imshow(edgeimage);
In = imnoise(I,'salt & pepper',0.02);
Jn = edge(In,'log');
nr = sum(sum(edgeimage));
nn = sum(sum(Jn));
P = abs(nn-nr)/nr
set(handles.edit6,'String', P);

function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5 as a double

% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit6 as text
%        str2double(get(hObject,'String')) returns contents of edit6 as a double

% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a double

% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit1_Callback(hObject, eventdata, handles)
sigmax1=str2num(get(handles.edit1, 'string'));
handles.sigmax1=sigmax1;
guidata (hObject,handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit2_Callback(hObject, eventdata, handles)
alfa=str2num(get(handles.edit2, 'string'));
handles.alfa=alfa;
guidata (hObject,handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a double
```

```
% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
I = double(rgb2gray(handles.data1));
In = imnoise(I,'salt & pepper',0.02);
Jn = edge(In,'log');
nn = sum(sum(Jn));
E = edge(I,'log');
P = sum(sum(E));
R = abs(P-nn)/100;
set(handles.edit7,'String', R);

function edit7_Callback(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit7 as text
%        str2double(get(hObject,'String')) returns contents of edit7 as a double

% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Listing Program Figur Canny

```
function varargout = canny(varargin)
% CANNY M-file for canny.fig
%      CANNY, by itself, creates a new CANNY or raises the existing
%      singleton*.
%
%      H = CANNY returns the handle to a new CANNY or the handle to
%      the existing singleton*.
%
%      CANNY('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in CANNY.M with the given input arguments.
%
%      CANNY('Property','Value',...) creates a new CANNY or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before canny_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to canny_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help canny

% Last Modified by GUIDE v2.5 03-Feb-2012 00:11:09

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @canny_OpeningFcn, ...
                   'gui_OutputFcn',  @canny_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before canny is made visible.
function canny_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% varargin   command line arguments to canny (see VARARGIN)

% Choose default command line output for canny
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes canny wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = canny_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in keluar.
function keluar_Callback(hObject, eventdata, handles)
delete(handles.figure1)
% hObject    handle to keluar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in buka_file.
function buka_file_Callback(hObject, eventdata, handles)
% hObject    handle to buka_file (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[nama_file1, nama_path1]=uigetfile(...
{'*.bmp;*.jpg','File Citra(*.bmp,*.jpg)';
'*.bmp','File Bitmap (*.bmp)';...
'*.jpg','File jpeg (*.jpg)';
'*.*','Semua File (*.*)'},...
'Buka File Citra Host/Asli');
if ~isequal(nama_file1, 0)
   handles.data1=imread(fullfile(nama_path1,nama_file1));
guidata(hObject,handles);
handles.current_data1=handles.data1;
axes(handles.axes1);
imshow(handles.current_data1);
else
return;
end
set(handles.text4,'String',nama_file1);
set(handles.text6,'String',size(handles.data1,1));
set(handles.text5,'String',size(handles.data1,2));
```

```
% --- Executes on button press in file_proses.
function file_proses_Callback(hObject, eventdata, handles)
% hObject    handle to file_proses (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%%%%%%% The functions used in the main.m file %%%%%%%
% Function "d2dgauss.m":
% This function returns a 2D edge detector (first order derivative
% of 2D Gaussian function) with size n1*n2; theta is the angle that
% the detector rotated counter clockwise; and sigma1 and sigma2 are
the
% standard deviation of the gaussian functions.
function h = d2dgauss(n1,sigma1,n2,sigma2,theta)
r=[cos(theta) -sin(theta);
sin(theta) cos(theta)];
for i = 1 : n2
for j = 1 : n1
u = r * [j-(n1+1)/2 i-(n2+1)/2]';
h(i,j) = gauss(u(1),sigma1)*dgauss(u(2),sigma2);
end
end
h = h / sqrt(sum(sum(abs(h).*abs(h))));
% Function "gauss.m":
function y = gauss(x,std)
y = exp(-x^2/(2*std^2)) / (std*sqrt(2*pi));
% Function "dgauss.m"(first order derivative of gauss function):
function y = dgauss(x,std)
y = -x * gauss(x,std) / std^2;
%%%%%%%%%%%%%% end of the functions %%%%%%%%%%%%%

% --- Executes on button press in canny_deteksi.
function canny_deteksi_Callback(hObject, eventdata, handles)
% hObject    handle to canny_deteksi (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% The algorithm parameters:
% 1. Parameters of edge detecting filters:
% X-axis direction filter:
Nx1=70;Sigmax1=7.0;Nx2=70;Sigmax2=7.0;Theta1=pi/2;
% Y-axis direction filter:
Ny1=70;Sigmay1=7.0;Ny2=70;Sigmay2=7.0;Theta2=0;
% 2. The thresholding parameter alfa:
alfa=0.07;
I = double(rgb2gray(handles.data1));
% X-axis direction edge detection
filterx=d2dgauss(Nx1,Sigmax1,Nx2,Sigmax2,Theta1);
Ix= conv2(I,filterx,'same');
% Y-axis direction edge detection
filtery=d2dgauss(Ny1,Sigmay1,Ny2,Sigmay2,Theta2);
Iy=conv2(I,filtery,'same');
% Norm of the gradient (Combining the X and Y directional derivatives)
NVI=sqrt(Ix.*Ix+Iy.*Iy);
```

```
% Thresholding
I_max=max(max(NVI));
I_min=min(min(NVI));
level=alfa*(I_max-I_min)+I_min;
Ibw=max(NVI,level.*ones(size(NVI)));
% Thinning (Using interpolation to find the pixels where the norms of
% gradient are local maximum.)
[n,m]=size(Ibw);
for i=2:n-1,
for j=2:m-1,
if Ibw(i,j) > level,
X=[-1,0,+1;-1,0,+1;-1,0,+1];
Y=[-1,-1,-1;0,0,0;+1,+1,+1];
Z=[Ibw(i-1,j-1),Ibw(i-1,j),Ibw(i-1,j+1);
Ibw(i,j-1),Ibw(i,j),Ibw(i,j+1);
Ibw(i+1,j-1),Ibw(i+1,j),Ibw(i+1,j+1)];
XI=[Ix(i,j)/NVI(i,j), -Ix(i,j)/NVI(i,j)];
YI=[Iy(i,j)/NVI(i,j), -Iy(i,j)/NVI(i,j)];
ZI=interp2(X,Y,Z,XI,YI);
if Ibw(i,j) >= ZI(1) & Ibw(i,j) >= ZI(2)
edgeimage(i,j)=I_max;
else
edgeimage(i,j)=I_min;
end
else
edgeimage(i,j)=I_min;
end
end
end
axes(handles.axes3);
imshow(edgeimage);

% --- Executes on button press in canny_tr.
function canny_tr_Callback(hObject, eventdata, handles)
% hObject    handle to canny_tr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
tic
% The algorithm parameters:
% 1. Parameters of edge detecting filters:
% X-axis direction filter:
Nx1=70;Sigmax1=7.0;Nx2=70;Sigmax2=7.0;Theta1=pi/2;
% Y-axis direction filter:
Ny1=70;Sigmay1=7.0;Ny2=70;Sigmay2=7.0;Theta2=0;
% 2. The thresholding parameter alfa:
alfa=0.07;
I = double(rgb2gray(handles.data1));
% X-axis direction edge detection
filterx=d2dgauss(Nx1,Sigmax1,Nx2,Sigmax2,Theta1);
Ix= conv2(I,filterx,'same');
% Y-axis direction edge detection
filtery=d2dgauss(Ny1,Sigmay1,Ny2,Sigmay2,Theta2);
Iy=conv2(I,filtery,'same');
```

```
% Norm of the gradient (Combining the X and Y directional derivatives)
NVI=sqrt(Ix.*Ix+Iy.*Iy);
% Thresholding
I_max=max(max(NVI));
I_min=min(min(NVI));
level=alfa*(I_max-I_min)+I_min;
Ibw=max(NVI,level.*ones(size(NVI)));
% Thinning (Using interpolation to find the pixels where the norms of
% gradient are local maximum.)
[n,m]=size(Ibw);
for i=2:n-1,
for j=2:m-1,
if Ibw(i,j) > level,
X=[-1,0,+1;-1,0,+1;-1,0,+1];
Y=[-1,-1,-1;0,0,0;+1,+1,+1];
Z=[Ibw(i-1,j-1),Ibw(i-1,j),Ibw(i-1,j+1);
Ibw(i,j-1),Ibw(i,j),Ibw(i,j+1);
Ibw(i+1,j-1),Ibw(i+1,j),Ibw(i+1,j+1)];
XI=[Ix(i,j)/NVI(i,j), -Ix(i,j)/NVI(i,j)];
YI=[Iy(i,j)/NVI(i,j), -Iy(i,j)/NVI(i,j)];
ZI=interp2(X,Y,Z,XI,YI);
if Ibw(i,j) >= ZI(1) & Ibw(i,j) >= ZI(2)
edgeimage(i,j)=I_max;
else
edgeimage(i,j)=I_min;
end
else
J(i,j)=I_min;
end
end
end
axes(handles.axes3);
imshow(edgeimage);
t = toc;
set(handles.edit5,'string',t);

% --- Executes on button press in canny_sr.
function canny_sr_Callback(hObject, eventdata, handles)
% hObject    handle to canny_sr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% The algorithm parameters:
% 1. Parameters of edge detecting filters:
% X-axis direction filter:
Nx1=70;Sigmax1=7.0;Nx2=70;Sigmax2=7.0;Theta1=pi/2;
% Y-axis direction filter:
Ny1=70;Sigmay1=7.0;Ny2=70;Sigmay2=7.0;Theta2=0;
% 2. The thresholding parameter alfa:
alfa=0.07;
I = double(rgb2gray(handles.data1));
% X-axis direction edge detection
filterx=d2dgauss(Nx1,Sigmax1,Nx2,Sigmax2,Theta1);
Ix= conv2(I,filterx,'same');
```

```
% Y-axis direction edge detection
filtery=d2dgauss(Ny1,Sigmay1,Ny2,Sigmay2,Theta2);
Iy=conv2(I,filtery,'same');
% Norm of the gradient (Combining the X and Y directional derivatives)
NVI=sqrt(Ix.*Ix+Iy.*Iy);
% Thresholding
I_max=max(max(NVI));
I_min=min(min(NVI));
level=alfa*(I_max-I_min)+I_min;
Ibw=max(NVI,level.*ones(size(NVI)));
% Thinning (Using interpolation to find the pixels where the norms of
% gradient are local maximum.)
[n,m]=size(Ibw);
for i=2:n-1,
for j=2:m-1,
if Ibw(i,j) > level,
X=[-1,0,+1;-1,0,+1;-1,0,+1];
Y=[-1,-1,-1;0,0,0;+1,+1,+1];
Z=[Ibw(i-1,j-1),Ibw(i-1,j),Ibw(i-1,j+1);
Ibw(i,j-1),Ibw(i,j),Ibw(i,j+1);
Ibw(i+1,j-1),Ibw(i+1,j),Ibw(i+1,j+1)];
XI=[Ix(i,j)/NVI(i,j), -Ix(i,j)/NVI(i,j)];
YI=[Iy(i,j)/NVI(i,j), -Iy(i,j)/NVI(i,j)];
ZI=interp2(X,Y,Z,XI,YI);
if Ibw(i,j) >= ZI(1) & Ibw(i,j) >= ZI(2)
edgeimage(i,j)=I_max;
else
edgeimage(i,j)=I_min;
end
else
edgeimage(i,j)=I_min;
end
end
end
axes(handles.axes3);
imshow(edgeimage);
In = imnoise(I,'salt & pepper',0.02);
Jn = edge(In,'canny');
nr = sum(sum(edgeimage));
nn = sum(sum(Jn));
P = abs(nn-nr)/nr
set(handles.edit6,'String', P);

function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5 as a double

% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit6 as text
%        str2double(get(hObject,'String')) returns contents of edit6 as a double

% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit1_Callback(hObject, eventdata, handles)
sigmax1=str2num(get(handles.edit1, 'string'));
handles.sigmax1=sigmax1;
guidata (hObject,handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function edit2_Callback(hObject, eventdata, handles)
alfa=str2num(get(handles.edit2, 'string'));
handles.alfa=alfa;
guidata (hObject,handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a double

% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
I = double(rgb2gray(handles.data1));
In = imnoise(I,'salt & pepper',0.02);
Jn = edge(In,'canny');
nn = sum(sum(Jn));
E = edge(I,'canny');
P = sum(sum(E));
R = abs(P-nn)/100;
set(handles.edit7,'String',R)

function edit7_Callback(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit7 as text
%        str2double(get(hObject,'String')) returns contents of edit7 as a double

% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function pushbutton7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```