PENERAPAN FILTER PENAJAMAN DALAM KAWASAN SPATIAL DAN FREKUENSI UNTUK CITRA HASIL MONITORING KAMERA PADA MIKROMOTOR

SKRIPSI

TEKNIK ELEKTRO KONSENTRASI ELEKTRONIKA

Ditujukan untuk memenuhi persyaratan

memperoleh gelar Sarjana Teknik

Disusun Oleh :

NOTARIO PRAMUDITA

NIM. 12506030511007

UNIVERSITAS BRAWIAYA

FAKULTAS TEKNIK

MALANG

2017

LEMBAR PENGESAHAN

PENERAPAN FILTER PENAJAMAN DALAM KAWASAN SPATIAL DAN FREKUENSI UNTUK CITRA HASIL MONITORING KAMERA PADA MIKROMOTOR

SKRIPSI

TEKNIK ELEKTRO KONSENTRASI ELEKTRONIKA

Ditujukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

Notario Pramudita
NIM. 125060305111007

Skripsi ini telah direvisi dan disetujui oleh dosen pembimbing
pada tanggal 8 Agustus 2017

Dosen Pembimbing I	Dosen Pembimbing II
Ir. Ponco Siwindarto, M.Eng.,Sc.	**Dr. Eng. Panca Mudjirahardjo, ST., MT.**
NIP. 19590304 198903 1 001	**NIP. 19700329 200012 1 001**

Mengetahui,
Ketua Jurusan Program Studi

M. Aziz Muslim, S.T., M.T., Ph.D.
NIP. 19741203 200012 1 001

PERNYATAAN ORISINALITAS SKRIPSI

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya dan berdasarkan hasil penelusuran berbagai karya ilmiah, gagasan dan masalah ilmiah yang diteliti dan diulas didalam Naskah Skripsi adalah asli dari pemikiran saya, tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali secara tertulis dikutip dalam naskah ini dan disebutkan dalam sumber kutipan dan daftar pustaka.

Apabila ternyata didalam Naskah Skripsi ini dibuktikan terdapat unsur-unsur jiplakan, saya bersedia Skripsi dibatalkan, serta proses sesuai peraturan perundang-undangan yang berlaku (UU No.20 Tahun 2003, Pasal 25 ayat 2 dan pasal 70).

Malang, 19 Januari 2017

Mahasiswa,

Notario Pramudia
NIM. 125060305111007

RINGKASAN

Notario Pramudita, Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya, Januari 2017, *Perancangan Sistem Monitoring Kamera dengan Menggunakan Aplikasi Eclipse Java*, Dosen Pembimbing: Ponco Siwindarto dan Panca Mujirahardja.

Mikromotor adalah alat yang digunakan pada bidang kedokteran gigi dalam aktivitas praktik maupun belajar. Kegunaan mikromotor sangat banyak, antara lain membersihkan karang gigi, memulas *overhanging* restorasi dan lain sebagainya. Desain sistem yang dapat digunakan untuk memudahkan penggunaan mikromotor pada kegaiatan bidang tersebut yaitu dengan menambahkan mini kamera pada ujung mikromotor, dengan tujuan agar pasien dapat melihat secara langsung gigi mereka. Untuk perancangan sistem, maka mini kamera dengan spesifikasi resolusi gambar 640x480 *pixel* dan diameter 7mm adalah pilihan yang tepat untuk mengaplikasikan sistem. Sistem dirancang dengan menggunakan bahasa program Java. Pengambilan gambar dilakukan secara *real time* dengan beberapa macam filter, yaitu *Laplace Spatial*, *Gradient*, *Laplace Frequency*, dan *Gaussian*. Hasil dari percobaan menunjukkan waktu pengeksekusian yang didapat dari masing-masing filter berbeda dan dengan perbandingan nilai yang jauh. Selain itu, ketelitian dan kejelasan gambar yang didapat juga berbeda antara filter yang satu dengan yang lainnya. Hal ini menunjukkan bahwa pengambilan gambar secara *real time* dengan ketelitain dan kejelasan gambar yang dihasilkan dapat digunakan ke depannya dan diaplikasikan dalam bidang kedokteran gigi.

Kata kunci: Java, Mikromotor, Mini Kamera

SUMMARY

Notario Pramudita, *Department of Electrical Engineering, Faculty of Engineering, University of Brawijaya, January 2017,* ***Perancangan Sistem Monitoring Kamera dengan Menggunakan Aplikasi Eclipse Java***, Academic Supervisor: Ponco Siwindarto and Panca Mujirahardja.

Micromotor is a tool used in the field of dentistry in practice and learning activities. Micromotor have a lot of usefullnes, such as cleaning tartar, overhanging applying restoration and others. The system design that can be used to facilitate the user of micromotor in the field by adding a mini camera on the end micromotor, with the aim that the patient can see firsthand their teeth. For the system design, the mini camera with image resolution of 640x480 pixels specifications and 7mm diameter is the right choice to apply the system. The system was designed using the Java programming language. The image is taken in real time with some kind of filter, the Laplace Spatial, Gradient, Laplace Frequency, and Gaussian. The results of the experiment showed that the execution time obtained from each of the different filters and the comparison value far. In addition, the accuracy and clarity of the image obtained is also different between the filter with each other. This indicates that the picture in real time with the detail and clarity of images produced can be used in the future and is applied in the field of dentistry.

Keywords: Java, Micromotor, Mini Camera

PENGANTAR

Alhamdulillah, puji syukur penulis panjatkan ke hadirat Allah SWT yang telah memberikan rahmat dan hidayah-Nya, sehingga penulis dapat menyelesaikan skripsi yang berjudul "Perancangan Sistem Monitoring Kamera dengan Menggunakan Aplikasi Eclipse Java" dengan lancar. Skripsi ini disusun sebagai syarat untuk mencapai gelar Sarjana Teknik Elektro Fakultas Teknik Universitas Brawijaya.

Penulis menyadari bahwa penyelesaian skripsi ini tidak akan mungkin bisa tercapai tanpa bantuan, bimbingan serta dorongan dari semua pihak. Oleh karena itu, pada kesempatan ini penulis menyampaikan rasa terima kasih yang sebesar-besarnya kepada :

- Bapak Azis Muslim, ST., MT., Ph.D. sebagai Ketua Jurusan Teknik Elektro Universitas Brawijaya.
- Bapak Hadi Suyono, ST., MT., Ph.D. sebagai Sekretaris Jurusan Teknik Elektro Universitas Brawijaya.
- Ibu Ir. Nurussa'adah, ST., MT. sebagai Ketua Kelompok Dosen Keahlian Elektronika Jurusan Teknik Elektro niversitas Brawijaya.
- Bapak Dr. Ir. Ponco Siwindarto, M. Eng., M.Sc. sebagai Dosen Pembimbing I atas segala bimbingan, pengarahan, ide, saran, dan kritik yang telah diberikan selama pengerjaan skripsi.
- Bapak Dr. Eng. Panca Mudjirahardjo, ST. MT., M.Eng. sebagai Dosen Pembimbing II atas segala bimbingan, pengarahan, ide, saran, dan kritik yang telah diberikan selama pengerjaan skripsi.
- Seluruh Dosen TEUB atas bimbingannya selama perkuliahan, serta staff recording TEUB, dan staff Ruang Baca Teknik Elektro Universitas Brawijaya.
- Ayah tercinta Atma, yang mendidik penulis dengan baik sehingga mampu menyelesaikan skripsi ini.
- Ibu tercinta Mawarti yang penuh kesabaran memberikan kasih sayang, semangat serta doa hingga terselesaikannya skripsi ini.
- Terima kasih kepada Meuthia Sari yang telah memberi ide dalam penelitian saya.
- Bang Bop, Dede Putri, yang selalu menyemangati Rio.

- Teman-teman Bima, Tata, Gadis yang telah memacu saya sehingga saya terpacu untuk menyelesaikan skripsi ini.
- Teman – teman kontrakan seperjuangan Septian, Pujo, Reza Ndud, dan Fathur.
- Teman-teman Voltage, Sirojul, Vrisco, Tata, Ilham, Firman, Reza Kawalta dan Grup Elektro ½ Sehat atas dukungan dan doanya.
- Teman-teman VOCTRON (Paket B – 2012) dan Kelas F – 2012 atas dukungan dan doanya.
- Seluruh teman-teman serta semua pihak yang tidak mungkin untuk dicantumkan namanya satu-persatu, terima kasih atas segala bentuk bantuan dan dukungannya.

Pada akhirnya, penulis menyadari bahwa skripsi ini masih belum sempurna. Oleh karena itu, penulis sangat mengharapkan kritik dan saran yang membangun. Penulis berharap semoga skripsi ini dapat bermanfaat bagi perkembangan ilmu pengetahuan dan teknologi serta bagi masyarakat.

Malang, 19 Januari 2017

Penulis

DAFTAR ISI

BAB I

PENDAHULUAN

1.1 Latar Belakang

Mikromotor adalah alat yang digunakan oleh dokter gigi atau mahasiswa kedokteran gigi untuk preparasi, memulas *overhanging* restorasi, membersihkan karang gigi secara sederhana, mengebur model gigi, dan masih banyak lagi manfaat mikromotor buat mahasiswa kedokteran gigi (Andy Rezky Sulfajri, 2015:1). Penggunaan mikromotor sejauh ini hanya dapat dilihat oleh dokter gigi, sedangkan pasien tidak dapat melihat apa yang terjadi di gigi mereka. Pasien hanya dapat melihat jika dilakukan *x-ray* pada gigi tersebut. Maka dari itu dibutuhkan sebuah teknologi kamera yang dapat membantu dokter gigi dan pasien untuk kegiatan praktik.

Teknologi kamera yang semakin berkembang pesat didukung oleh teknologi mikrokontroler yang juga semakin berkembang. Mikrokontroler merupakan suatu terobosan teknologi mikroprosesor yang dilengkapi dengan memori dan semua komponen yang terintegrasi dalam satu chip. Mikrokontroler banyak digunakan pada berbagai sistem kontrol (Gridling, 2006:2). Teknologi kamera yang digunakan akan dihubungkan pada bagian mikromotor yang berhubungan langsung dengan gigi pasien.

Teknologi kamera pada mikromotor digunakan untuk menangkap gambar dan menampilkannya kepada LCD agar pasien dapat melihat langsung apa yang terjadi di gigi pasien itu sendiri, dan juga agar pasien dapat melihat kerusakan yang terjadi pada gigi pasien.Pada metode yang sering dilakukan bahwa penggunaan mikromotor dan kamera dilakukan dengan cara terpisah. Hal ini menjadi pertimbangan dalam penelitian yaitu bagaimana cara agar dapat menggabungkan teknologi mikromotor dan kamera dalam satu sistem pemakaian.

Pada pengaplikasian program ini menggunakan filter *laplacian frequency, gradient, laplacian spatical,* dan *Gaussian filtering.* Yang dimana filter-filter tersebut digunakan untuk penajaman dan penghalusan hasil citra gambar yang diambil menggunakan *mini camera* yang akan di sambungkan pada *hand piece* di *micromotor*.

Filter-filter tersebut deprogram dengan menggunakan program java yang akan program sesuai kebutuhan filter tersebut. Pada metode *filtering* itu sendiri penulis dapat mengetahui perbedaan citra dari gambar orisinil kamera dan hasil citra yang sudah melalui hasil filterisasi.

Penilitian ini akan membahas perancangan mini kamera dengan metode *real time capturing* menggunakan aplikasi Eclipse Java pada mikromotor dan menampilkan gambar yang ditangkap pada LCD PC.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dikemukakan dapat disusun rumusan masalah sebagai berikut:

1. Bagaimana proses filtering yang dilakukan dan cara kerjanya?
2. Bagaimana cara menampilkan hasil gambar yang ditangkap oleh mini kamera ke LCD PC.
3. Perbedaan antara hasil original kamera dan hasil kamera yang sudah diberi masukan filter.

1.3 Batasan Masalah

Dengan mengacu pada permasalahan yang telah dirumuskan, maka hal-hal yang berkaitan dengan alat akan diberi batasan sebagai berikut:

1. Pembahasan ditekankan pada pengambilan gambar *real time capturing* kamera hanya menampilkan gambar pada LCD PC dan membuat cara kerja pada program.
2. Kamera yang digunakan adalah mini kamera (*Android Camera 8mm 640x480p)*
3. LCD yang digunakan pada perancangan adalah LCD pada PC 18” VAIO 14A16FGH.
4. Pemrograman yang digunakan pada penelitian ini menggunakan aplikasi Eclipse Java.
5. Pemfilteran yang dilakukan dalam kawasan spatial dan frekuensi pada *high pass filter*.

1.4 Tujuan

Penelitian ini bertujuan untuk membuat aplikasi yang mampu melakukan pemfilteran, dalam kawasan spatial dan frekuensi, serta menampilkan i*mage* (*640x480p*) secara *real time*.

1.5 Manfaat/ Kegunaan

Manfaat/ kegunaan dari penelitian ini antara lain sebagai berikut :

1. Dapat menjadi salah satu inovasi baru dalam dunia kedokteran dimana *hand piece* dokter gigi dapat dikombinasikan oleh sebuah kamera mini.

2. Bisa menjadi referensi dalam penelitian yang berkaitan dengan pemfilteran kamera dengan jarak yang dekat.
3. Dapat menjadi sebuah cara untuk pasien mengerti apa yang dokter gigi lakukan kepada gigi mereka secara langsung.
4. Dapat menunjukan ketajaman filter pada hasil proses pemfilteran gambar.

BAB II
TINJAUAN PUSTAKA

Gigi merupakan bagian dari tubuh yang berfungsi untuk menghaluskan makanan agar nantinya dapat dicerna dengan mudah oleh lambung. Selain penting untuk membuat makanan menjadi halus, gigi juga sangat penting untuk menunjang penampilan anda. Gigi yang putih dan sehat merupakan awal dari penilaian seseorang terhadap penampilan anda. Karena itulah, menjaga kesehatan gigi adalah hal yang sangat penting untuk diperhatikan (Dahlia Pohand, 2013:1). Penelitian ini dirancang untuk monitoring kinerja *hand piece* pada mikromotor di dalam mulut manusia.

Beberapa teori pendukung yang perlu dibahas dalam pembuatan perancangan system monitoring pada perangkaat lunak dengan menggunakan aplikasi Eclipse Java**.** ini meliputi:

1. Personal computer (PC)
2. Aplikasi perangkat lunak Eclipse Java
3. Mikromotor
4. Mini kamera dengan diameter 7mm
5. *Laplacian filter* kawasan *Spatial* dan *Frequency*

2.1 PC (personal computer)

PC (*personal computer)* adalah gabungan dari perangkat keras (*hardware)* dan perangkat lunak (*software). Personal Computer* adalah seperangkat komputer yang digunakan oleh satu orang saja/pribadi. Biasanya komputer ini adanya dilingkungan rumah, kantor, toko, dan dimana saja karena harga PC sudah relatif terjangkau dan banya kmacamnya. Fungsi utama dari PC adalah untuk mengolah data input dan menghasilkan output berupa data/informasi sesuai dengan keinginan *user* (pengguna). Dalam pengolahan data yang dimulai dari memasukkan data (input) sampai akhirnya menghasilkan informasi, komputer memerlukan suatu sistem dari kesatuan elemen yang tidak bisa terpisahkan. Ilustrasi komputer ditunjukkan dalam Gambar 2.1.

Gambar 2.1 *Personal Computer*

2.2 Java

Java adalah Bahasa pemograman tingat tinggi yang beriorentasi objek dan pemograman java tersusun dari bagian yang disebut kelas. Kelas terdiri atas metode-metode yang melakukan pekerjaan dan mengembalikan informasi setelah melakukan tugasnya. Java API (*application programming interface*) telah menyediakan fungsionalitas yang memadai untuk menciptakan *applet* dan aplikasi yang canggih.

Java merupakan Bahasa berioerentasikan objek (OOP) yaitu cara ampuh untuk pengorganisasian dan pengembangan perangkat lunak. Keunggulan java merupakan Bahasa pemograman yang sederhana. Java dirancang agar mudah dipelajari dan digunakan secara efektif. Java tidak menyediakan fitur yang rumit.

2.3 Mikromotor

Mikromotor adalah miniatur motor elektrik yang digunakan oleh dokter gigi atau mahasiswa kedokteran gigi untuk preparasi, memulas overhanging restorasi, membersihkan karang gigi secara sederhana, mengebur model gigi, dan masih banyak lagi manfaat micromotor buat mahasiswa kedokteran gigi. Mikromotor ditunjukkan dalam Gambar 2.2.

Gambar 2.2 Mikromotor

Instrument tangan pada mikromotor yang digunakan oleh dokter gigi yang dimana alat ini menggunakan bur yang memiliki mata bur berbeda-beda. Dan pada *handpiece* ini bur yang digunakan dengan kecepatan putaran dari 0 sampai dengan 350000 rpm. *Handpiece* ini digunakan bagi ilmu kedokteran gigi adalah untuk mengapus sebagian besar amalgam, karang gigi, dan plak gigi pada gigi berlubang. *Handpiece* ditunjukkan dalam Gambar 2.3.

Gambar 2.3 *Handpiece* mikromotor

2.4 Mini kamera

Adalah sebuah kamera yang dimana kamera akan digunakan untuk menangkap gambar dan resolusi kamera akan dikintrol oleh mikrokontroler. Mini kamera ditunjukkan dalam Gambar 2.4.

Gambar 2.4 Kamera Mikro

Spesifikasi kamera mikro yang dapat digunakan dapat dilihat dalam Tabel 2.1.

Tabel 2.1 Spesifikasi Kamera Mikro

JENIS	SPESIFIKASI
Kamera Mikro	- Pixels: 300.000 pixels CMOS camera - Image resolution: 640 x 480 - LED Quantity: 6 - USB interface: USB 2.0 - Power: USB port - Frame rate: 30 fps - Waterproof level: IP66 - Focal distance: 6cm - Image format: VGA/QVGA

	- Panjang kabel: 5M - Diameter Kamera: 7mm (1/6 Inch)

2.5. Laplacian filter kawasan Spatial dan Frequency

Definisi *laplacian* adalah mempunyai nilai koefisien pusat negatif dengan menggunakan rumus pengurangan, maka gambar *Laplacian* akan menghasilkan gambar yang jelas dan tajam. Laplacian filter dibagi atas dua macam yaitu kawasan *spatial* dan kawasan *frequency*.

a. Filter *Spatial*

Merupakan operasi penurunan, maka digunakan garis putus-putus berawarna abu-abu pada gambar dan menekankan wilayah dengan warna abu-abu yang lain (Gonzales, Woods. 2001). *Laplacian spatial* mengarah pada latar gambar itu sendiri dan kategori yang berdasarkan manipulasi dasar dari *pixels* pada sebuah gambar.

b. Filter *Frequency*

Konsep filter yang menggunakan akar dari transformasi Fourier dalam proses sinyal. Merupakan modifikasi transformasi Fourier dari gambar itu sendiri (Gonzales, Woods. 2001).

Selain dalam kawasan *spatial* maupun kawasan *frequency*, filtering gambar juga dilakukan dalam *gradient* maupun *gaussian*. Turunan pertama dari proses gambar yaitu menggunakan implementasi nilai magnitud dari *gradient*. Komponen dari vektor *gradient* itu sendiri adalah operator linear. Sedangkan *gaussian* merupakan variabel acak yang dapat diimeplementasikan dalam image averaging untuk mendapatkan nilai negatif ketika terdapat gangguan dalam gambar.

BAB III
METODE PENELITIAN

Metode penelitian yang digunakan pada skripsi ini adalah metode studi kepustakaan dan penelitian laboratorium. Studi kepustakaan dilakukan sebagai penunjang yang berupa data-data literatur dari masing-masing komponen, informasi dari internet dan konsep-konsep teoritis dari buku-buku penunjang. Penelitian laboratorium berupa perancangan perangkat keras dan perangkat lunak. Metode penelitian pada skripsi ini meliputi:

a. Penentuan spesifikasi
b. Perancangan sistem
c. Metode pengujian

3.1 Spesifikasi Perancangan Sistem

Penentuan spesifikasi sistem bertujuan agar dapat dibuat sesuai yang diinginkan dan dapat bekerja dengan efektif serta efisien. Dalam perancangan ini, spesifikasi yang dibutuhkan adalah sebagai berikut:

1. Sensor kamera Android HD berfungsi untuk menampilkan citra dengan resolusi kamera 640 x 480 piksel.
2. Pemrograman pada komputer menggunakan *software* Eclipse IDE untuk membuat program simulasi dan bahasa pemrograman yang digunakan adalah Java.

3.2 Perancangan Sistem

3.2.1 Diagram Blok

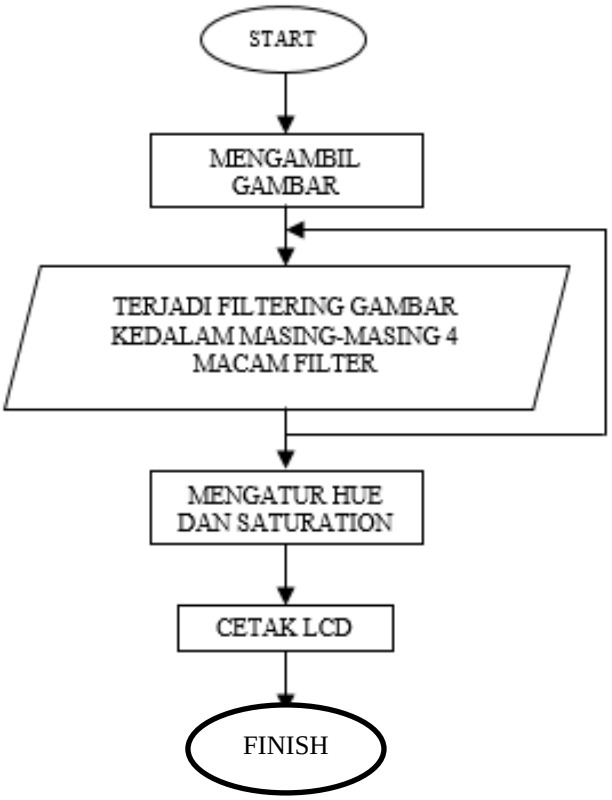
Pembuatan diagram blok merupakan dasar dari perancangan sistem agar perancangan berjalan secara sistematis. Diagram blok ditunjukkan Gambar 3.1. Perancangan sistem pada penelitian ini yaitu perancangan perangkat lunak. Perancangan perangkat lunak membahas tentang program simulasi, perancangan algoritma dan implementasinya pada sistem.

Gambar 3.1 Diagram Blok Sistem

Dilihat dari blok diagram, perangkat keras terdiri dari PC (*Personal Computer*) dan kamera. Sedangkan blok-blok di dalam PC merupakan pengolahan citra yang dilakukan secara terprogram. Kamera dapat diakses secara langsung dengan PC, sehingga tidak diperlukan program komunikasi serial maupun paralel. Kamera menangkap citra original. Output citra kamera akan diolah secara terprogram dengan PC. Pengolahan citra tersebut terdiri dari filter *Laplacian* (*spatial*), *Gradient*, *Laplacian* (*frequency*), dan *Gaussian* (Gonzales, Woods, 2001). Hasil pengolahan citra akan ditampilkan pada monitor PC dan dapat disimpan dalam format .jpg.

3.2.2 Perancangan Algoritma dan implementasi Sistem

Di sini akan dibahas mengenai bagaimana mendesain sistem sesuai yang diinginkan dengan merancang algoritma yang tepat dan efisien serta implementasinya terhadap objek yang ditangkap. Pada Gambar 3.2 ditunjukkan diagram alir yang mendeskripsikan cara kerja keseluruhan dari sistem. Diagram alir dari sistem berkaitan dengan diagram blok, karena setiap blok berfungsi untuk melaksanakan setiap proses pada diagram alir. Pada tahap pengambilan citra, blok yang bekerja adalah kamera. Kemudian hasil pengambilan citra oleh kamera akan diolah pada blok *filtering*. Hasil dari pengolahan citra dapat disimpan di PC dan ditampilkan pada monitor PC.



Gambar 3.2 Diagram Alir Sistem

Terdapat 4 macam filter yang akan diuji dalam penelitian ini, yaitu:

1. *Laplace Spatial Filter*

   Citra input dikonvolusi dengan matriks dapat dilihat dalam Gambar 3.3.

0	1	0
1	-4	1
0	1	0

Gambar 3.3 *Filter mask* yang digunakan untuk mengimplementasikan *digital Laplacian*
Sumber: Gonzales, Woods, 2001

Laplace Spatial merupakan operasi penurunan, maka digunakan garis putus-putus berwarna abu-abu pada gambar dan menekankan wilayah dengan warna abu-abu yang lain (Gonzales, Woods, 2001). Hal tersebut cenderung akan menghasilkan gambar dengan garis tepi abu-abu dan putus-putus, semua berukuran gelap, dengan latar belakang yang tidak terlihat. Latar belakang yang tidak terlihat tersebut dapat kembali terlihat ketika dapat mempertahankan efek penajaman gambar pada operasi *Laplacian* setelah menambahkan yang asli dan gambar *Laplacian*, sangat penting untuk menjaga definisi *Laplacian* ketika digunakan. Ketika definisi *Laplacian* mempunyai nilai koefisien pusat negatif, maka menggunakan rumus pengurangan, maka gambar *Laplacian* akan menghasilkan gambar

yang jelas dan tajam. Persamaan dasar untuk meningkatkan gambar *Laplacian* adalah sebagai berikut:

$$g(x,y) = \begin{cases} f(x,y) - \nabla^2 f(x,y) & \text{, Jika nilai } \textit{center coefficient} \text{ dari lapisan Laplacian adalah negatif} \\ f(x,y) + \nabla^2 f(x,y) & \text{, Jika nilai } \textit{center coefficient} \text{ dari lapisan Laplacian adalah positif} \end{cases} \quad (2\text{-}1)$$

2. *Gradient Filter*

Tepi horisontal pada *Laplacian* dapat dilihat dalam Gambar 3.4 dan tepi vertikal dapat dilihat dalam Gambar 3.5.

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

Gambar 3.4 Tepi horisontal
Sumber: Gonzales, Woods, 2001

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

Gambar 3.5 Tepi vertikal
Sumber: Gonzales, Woods, 2001

Lapisan pada Gambar 3.4 dan 3.5 disebut dengan *Sobel operators*, dapat diimplementasikan pada persamaan berikut:

$$\nabla f \approx |(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)| + |(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)| \quad (2\text{-}2)$$

Melalui memberikan nilai mekanis pada persamaan berikut:

$$g(x,y) = \sum_{x=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t) \quad (2\text{-}3)$$

3. *Laplacian* (*frequency*)

Dapat dilihat bahwa,

$$\Im\left[\frac{d^n f(x)}{dx^n}\right] = (ju)^n F(n) \quad (2\text{-}4)$$

Dari persamaan sederhana diatas, dapat diikuti oleh,

$$\Im\left[\frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 (x,y)}{\partial y^2}\right] = (ju)^2 F(u,v) + (jv)^2 F(u,v) = -(u^2 + v^2) F(u,v) \quad (2\text{-}5)$$

persamaan yang terletak di sebelah kiri dianggap sebagai fungsi $f(x,y)$ dari *Laplacian* dan hasil dari persamaan diatas yaitu,

$$\Im[\nabla^2 f(x,y)] = -(u^2+v^2)F(u,v) \quad (2\text{-}6)$$

dimana persamaan Laplacian diatas dapat diimplementasikan pada *frequency domain* dengan menggunakan filter,

$$H(u,v) = -(u^2+v^2) \quad (2\text{-}7)$$

Sebagai operasi filter, dapat diartikan bahwa $F(u,v)$ telah dipusatkan oleh operasi $f(x,y)(-1)^{x+y}$ sebelum transformasi dari gambar. Apabila *f* (and *F*) adalah ukuran dari *M x N*, maka operasi bagian dari transformasi pusat adalah (*u*,*v*) = (0,0) pada titik (*M/2*, *N/2*) pada *frequency rectangle*. Pusat dari fungsi filter juga menjadi bagian dari persamaan:

$$H(u,v) = -[(u-M/2)^2 + (v-N/2)^2] \quad (2\text{-}8)$$

Gambar filter *Laplacian* pada *spatial domain* dapat dihasilkan oleh *transformasi invers Fourier* dari fungsi *H*(*u*, *v*) *F*(*u*, *v*):

$$\nabla^2 f(x,y) = \Im^{-1}\left\{-\left[\left(u-\frac{M}{2}\right)^2 + \left(v-\frac{N}{2}\right)^2\right]F(u,v)\right\} \quad (2\text{-}9)$$

Hasil dari perhitungan transformasi Fourier sama dengan perkalian fungsi *F*(*u*, *v*) oleh *H*(*u*, *v*). Dari dua hubungan fungsi tersebut dapat dinotasikan sebagai persamaan:

$$\nabla^2 f(x,y) \Leftrightarrow -\left[\left(u-\frac{M}{2}\right)^2 + \left(v-\frac{N}{2}\right)^2\right]F(u,v) \quad (2\text{-}10)$$

4. Gaussian

Fungsi transfer pada *Gaussian highpass filter* (GHPF) dengan daerah *cutoff frequency* pada jarak D_u dari titik asal yaitu:

$$H(u,v) = 1 - e^{-D^{2(u,v)/2D_0^2}} \quad (2\text{-}11)$$

dimana *D*(*u*, *v*) merupakan persamaan berikut:

$$H(u,v) = \frac{1}{1+[D_0/D_{(u,v)}]^{2n}} \quad (2\text{-}12)$$

Persamaan ini juga merupakan penggabungan dari persamaan

$$H(u,v) = e^{-D^2(u,v)/2D_0^2} \quad (2\text{-}13)$$

dan

$$H_{hp}(u,v) = 1 - H_{lp}(u,v) \quad (2\text{-}14)$$

3.3 Metode Pengujian

Metode pengujian yang digunakan adalah simulasi dengan menggunakan program Eclipse Java dan program yang menggunakan bahasa pemrograman Java.

BAB IV

HASIL DAN PEMBAHASAN

Untuk mengetahui bahwa sistem hasil perancangan berfugsi dengan baik dan sesuai dengan spesifikasi perancangan, maka perlu dilakukan pengujian. Pengujian dilakukan untuk mendapatkan letak kesalahan dan mempermudah analisis pada sistem.

Pengujian yang dilakukan yaitu pengujian sistem secara keseluruhan.

4.1. Pengujian Sistem

Pengujian sistem dengan menggunakan program *eclipse java* bertujuan untuk mengetahui tingkat keberhasilan dari program yang telah dibuat sehingga dapat difungsikan sebagai media penangkap citra yang sesuai dengan perancangan.

4.1.1. Peralatan yang dibutuhkan

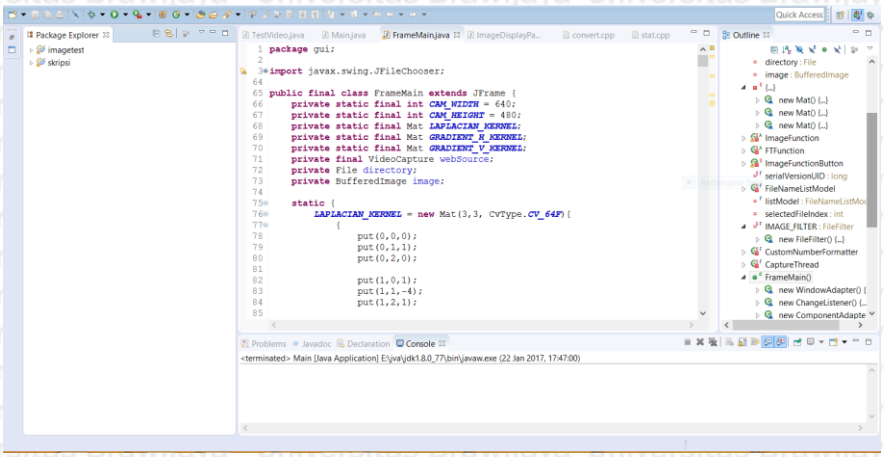
- Kamera mini
- PC yang sudah terpasang aplikasi *eclipse java work space*
- Gambar

4.1.2. Prosedur pengujian

Pengujian dilakukan berdasarkan 4 filter yang digunakan, yaitu *Laplace Spatial Filter*, *Gradient Filter*, *Laplace Frequency Filter* dan *Gaussian Filter*. Pengujian dilakukan dengan cara mengambil gambar objek melalui kamera, kemudian dilakukan pengolahan citra dengan menggunakan 4 macam filter dalam kawasan *Spatial* dan *Frequency*. Pengujian dilakukan untuk mencari perbedaan hasil pengolahan citra dari masing-masing filter tersebut dimana sebelumnya dilakukan pengaturan cahaya dan kejelasan gambar untuk memperoleh hasil yang diinginkan.

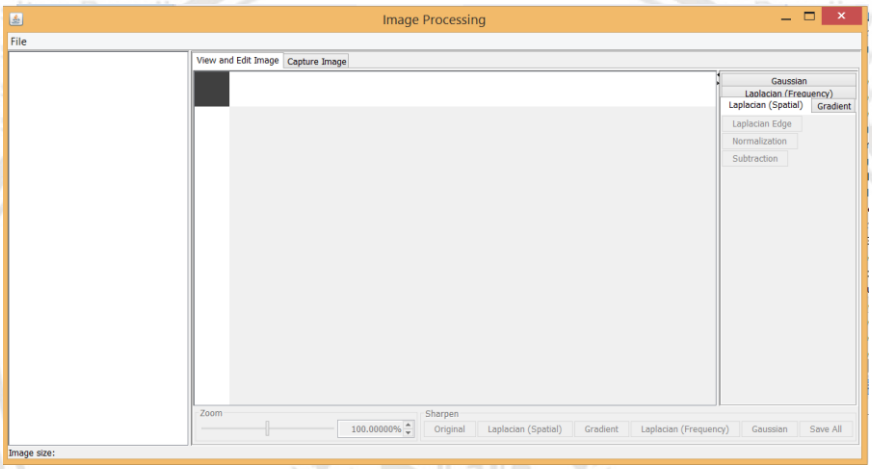
Prosedur pengujian dapat dilihat sebagai berikut:

- Membuka program *eclipse java work space* yang terdapat program yang akan digunakan kemudian klik *run*. Dapat dilihat dalam Gambar 4.1.



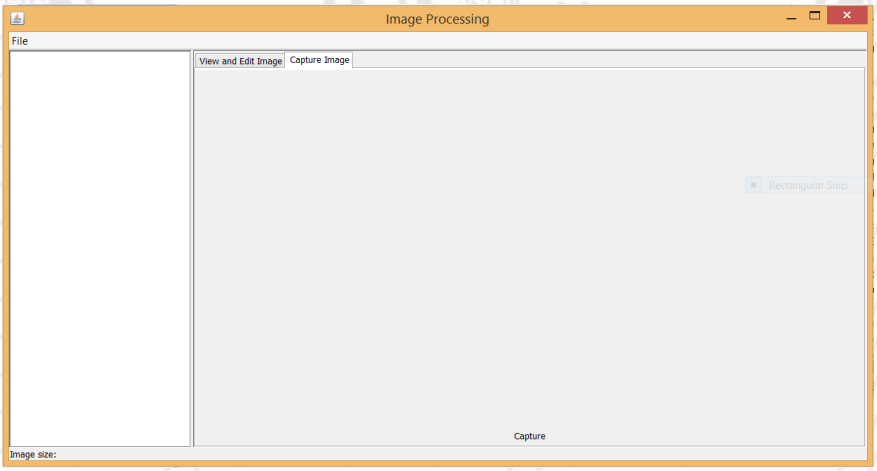
Gambar 4.1 *Eclipse Java Work Space* berisi program yang akan digunakan

- *windows image processing.* Dapat dilihat dalam Gambar 4.2.



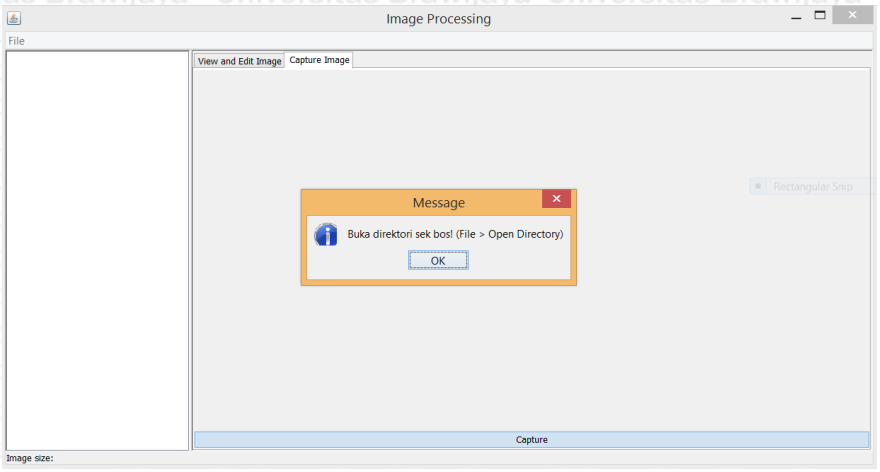
Gambar 4.2 *Tab Windows Image Processing*

- Pada tab *image processing*, muncul 2 tab yaitu *view* dan *capture image.* Dapat dilihat dalam Gambar 4.3.



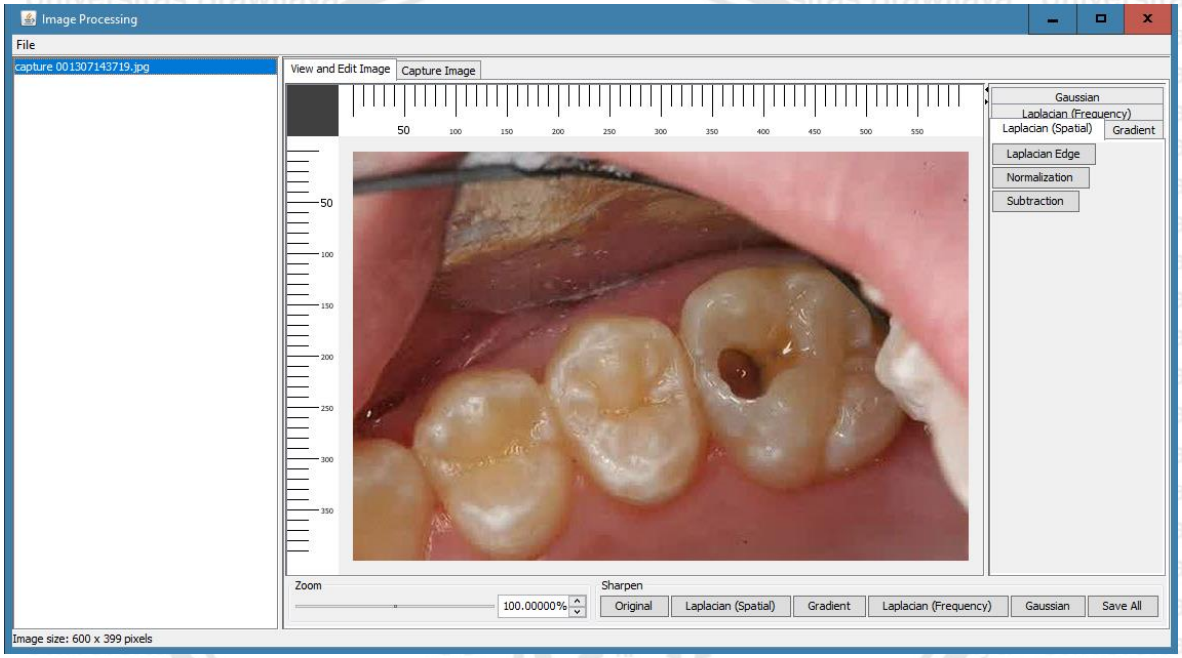
Gambar 4.3 Tab *Image Processing*

- Apabila ingin *review image* atau mengambil gambar terlebih dahulu untuk membuka *directory* lalu memilih *file* yang akan dibuka atau disimpan. Dapat dilihat dalam Gambar 4.4.



Gambar 4.4 *Review Image*

- Setelah gambar dibuka muncul 4 panel filtering. Dapat dilihat dalam Gambar 4.5.



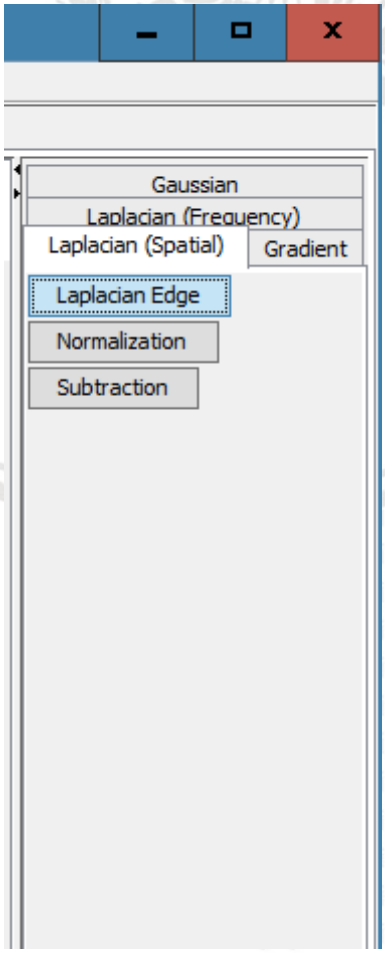
Gambar 4.5 Panel Filtering

a. *Laplace Spatial Filter*

- *Laplace Spatial Filter* dan waktu pengekseskusiannya dapat dilihat dalam Gambar 4.6.

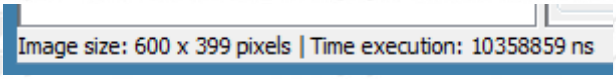
Gambar 4.6 *Laplace Spatial Filter*

- Pada tab dibagian ini adalah tab urutan kerja filter. Dapat dilihat dalam Gambar 4.7.



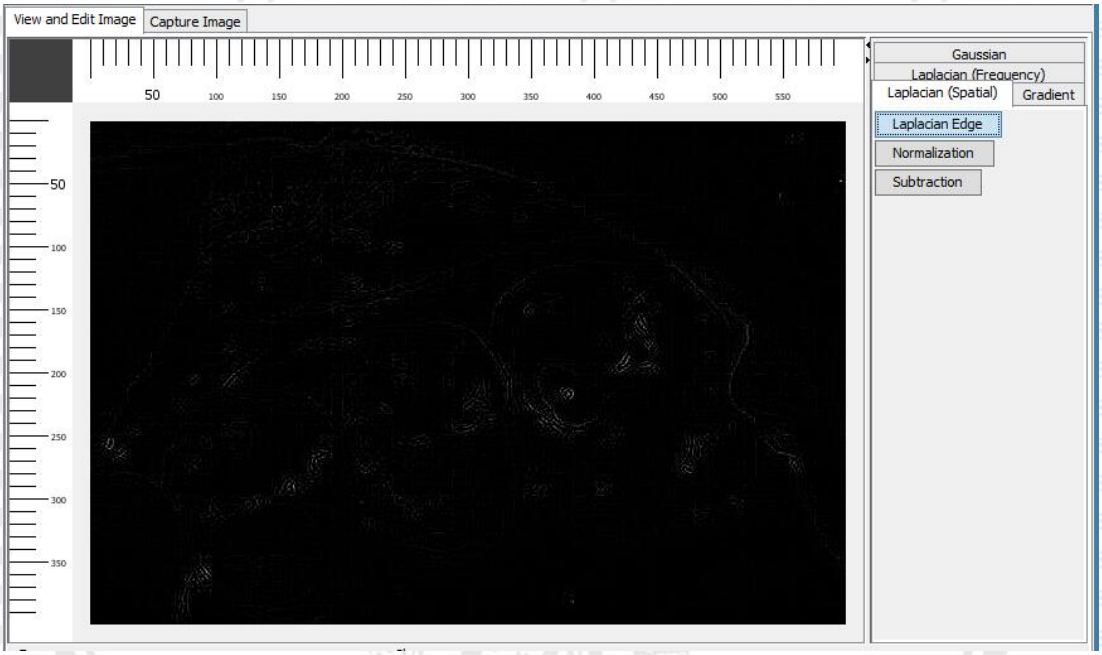
Gambar 4.7 Urutan kerja filter

- Waktu pengeksekusian filter dari original menjadi *laplace spatial filter*. Dapat dilihat dalam Gambar 4.8.



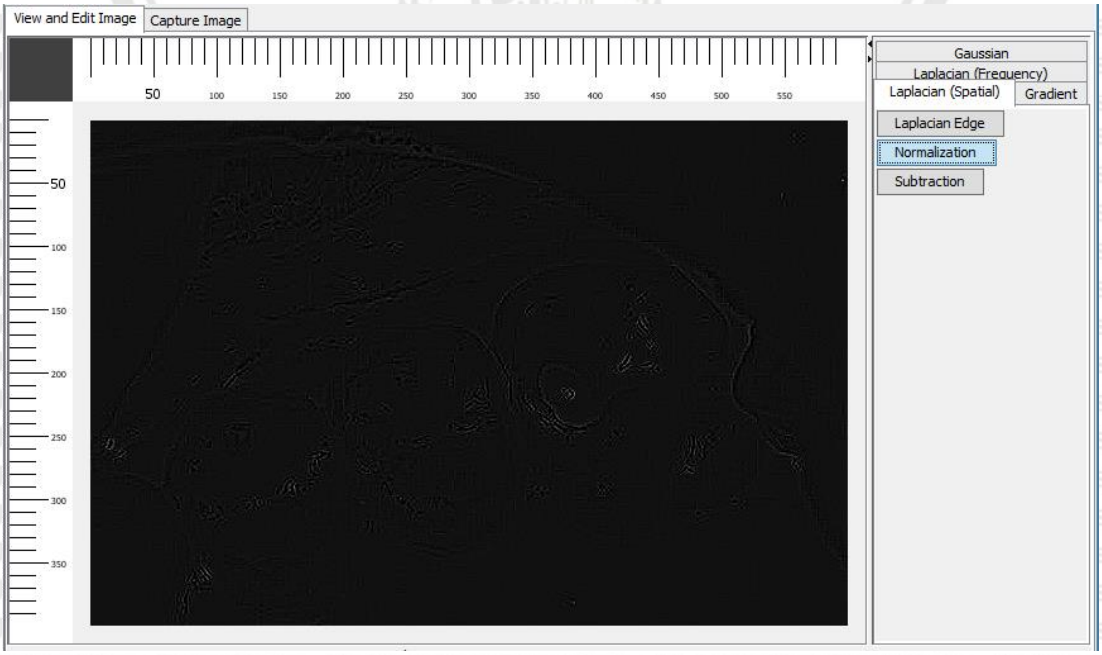
Gambar 4.8 Pengeksekusian filter

- Pada *laplace spatial* pertama akan melakukan *laplacian edge* yang ditunjukkan pada Gambar 4.9 dan waktu pengeksekusiannya



Gambar 4.9 *Laplacian Edge*

- Pada *laplace spatial* yang kedua dilakukan adalah pernomalisasi. Dapat dilihat dalam Gambar 4.10.



Gambar 4.10 Pernomalisasian gambar

- Pada *laplacian spatial* yang ketiga atau terakhir adalah *substraksi*. Dapat dilihat dalam Gambar 4.11.

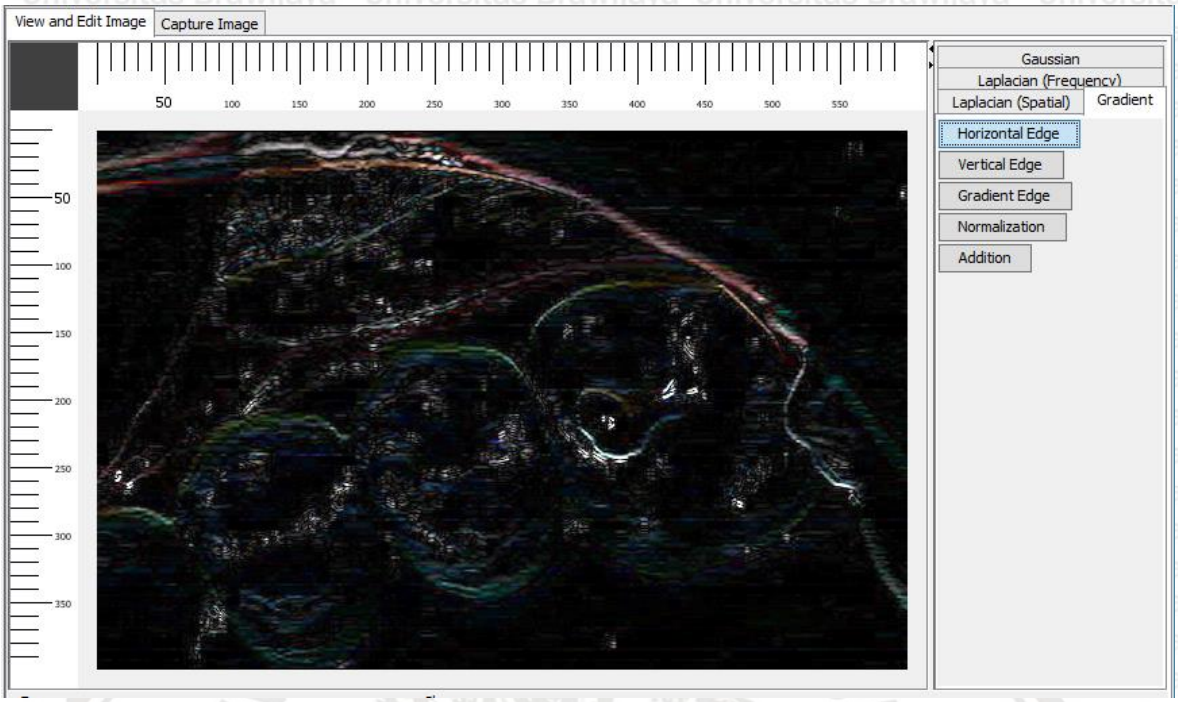
Gambar 4.11 *Substraksi*

b. *Gradient Filter*

- Filter kedua adalah *gradient*. Dapat dilihat dalam Gambar 4.12.

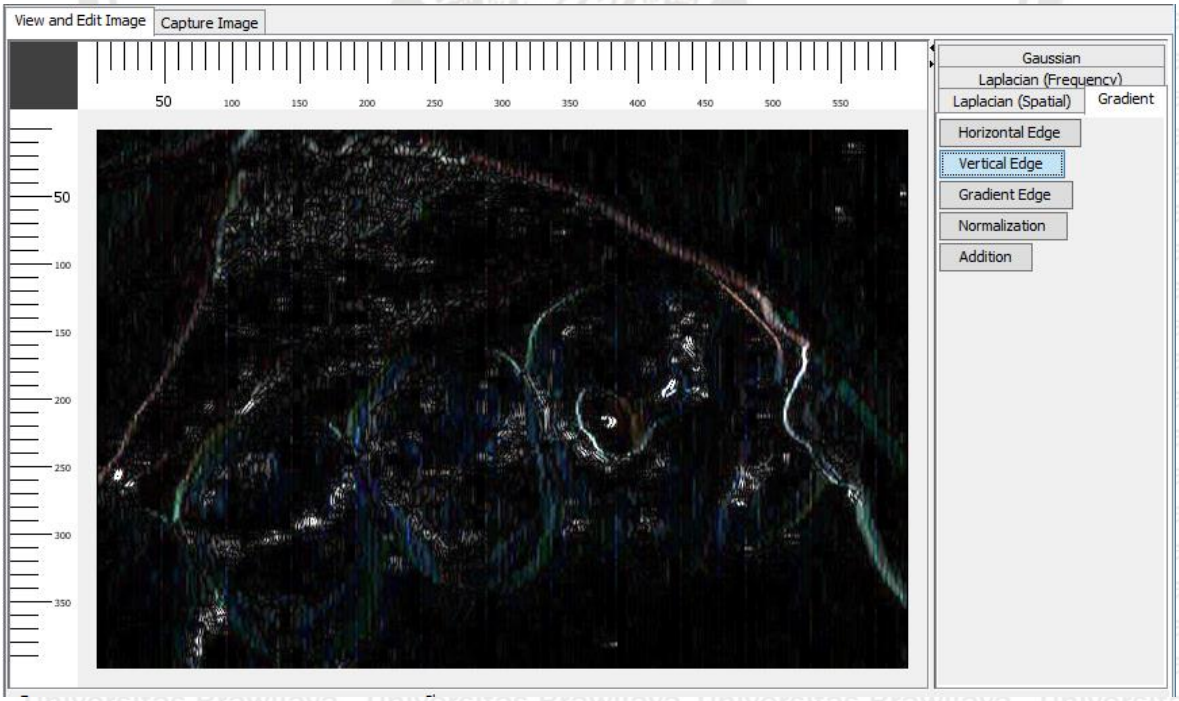
Gambar 4.12 *Gradient Filter*

- Yang pertama terjadi pada *gradient filter* adalah *horizontal edge.* Dapat dilihat dalam Gambar 4.13.



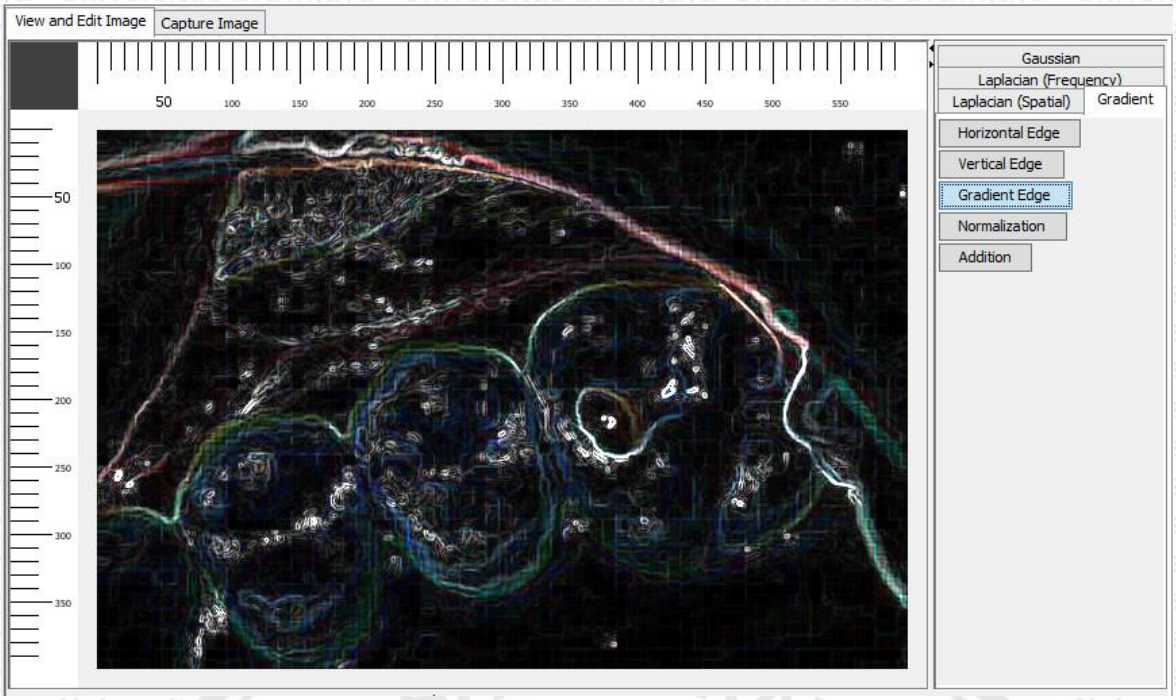
Gambar 4.13 ***Horizontal Edge***

- Pada *gradient filter* yang kedua adalah *vertical edge*. Dapat dilihat dalam Gambar 4.14.



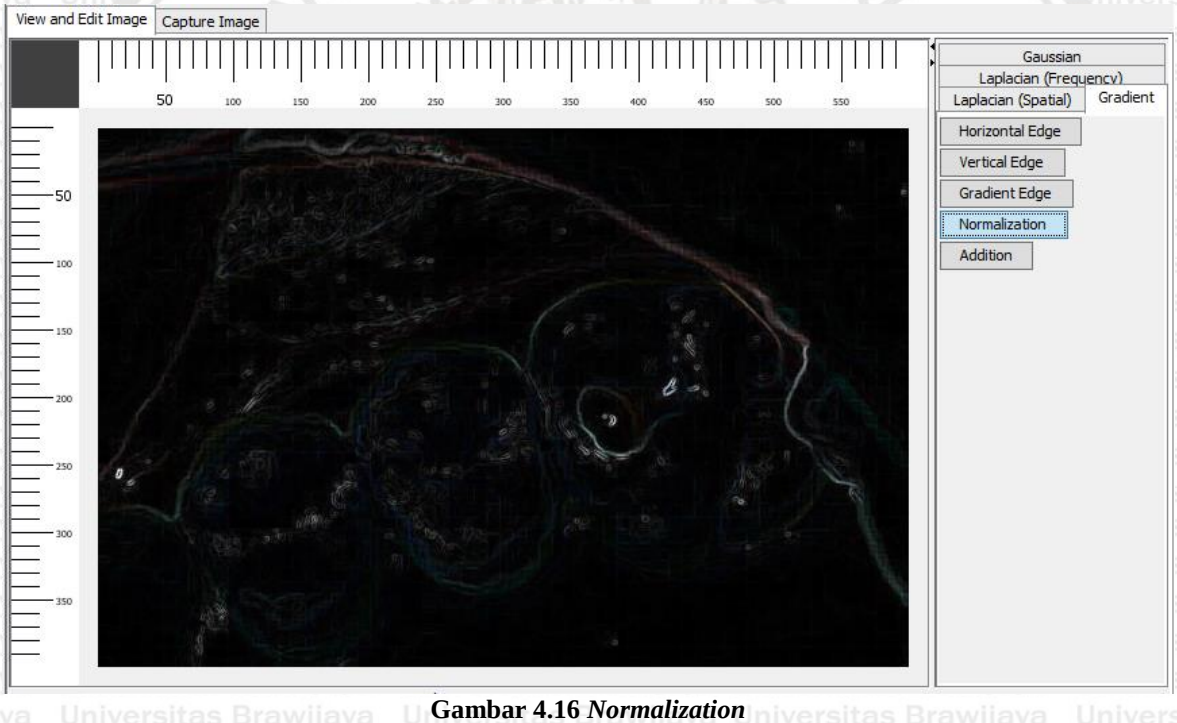
Gambar 4.14 ***Vertical Edge***

- Pada *gradient filter* yang ketiga adalah *gradient edge.* Dapat dilihat dalam Gambar 4.15.



Gambar 4.15 ***Gradient Edge***

- Filter yang keempat adalah *normalization*. Dapat dilihat dalam Gambar 4.16.



Gambar 4.16 ***Normalization***

- Pada *gradient filter* yg terakhir adalah *addition*. Dapat dilihat dalam Gambar 4.17.

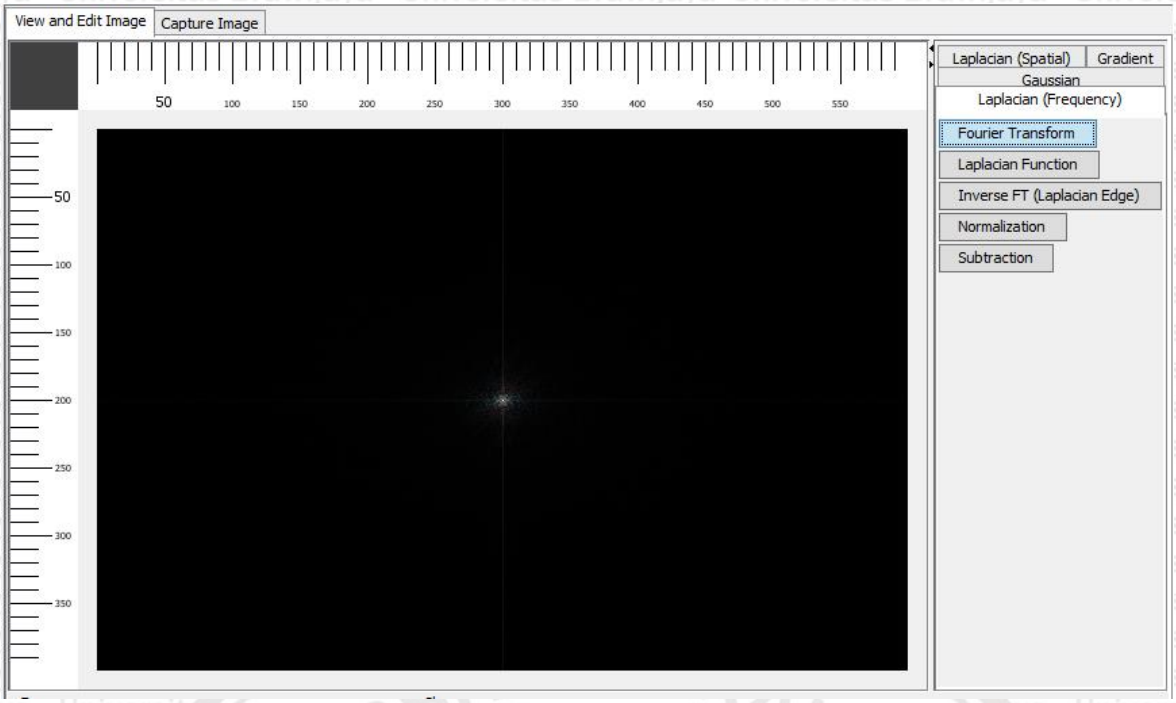
Gambar 4.17 ***Addition***

c. *Laplacian Frequency Filter*

- *Laplacian Frequency* dalam Gambar 4.18.

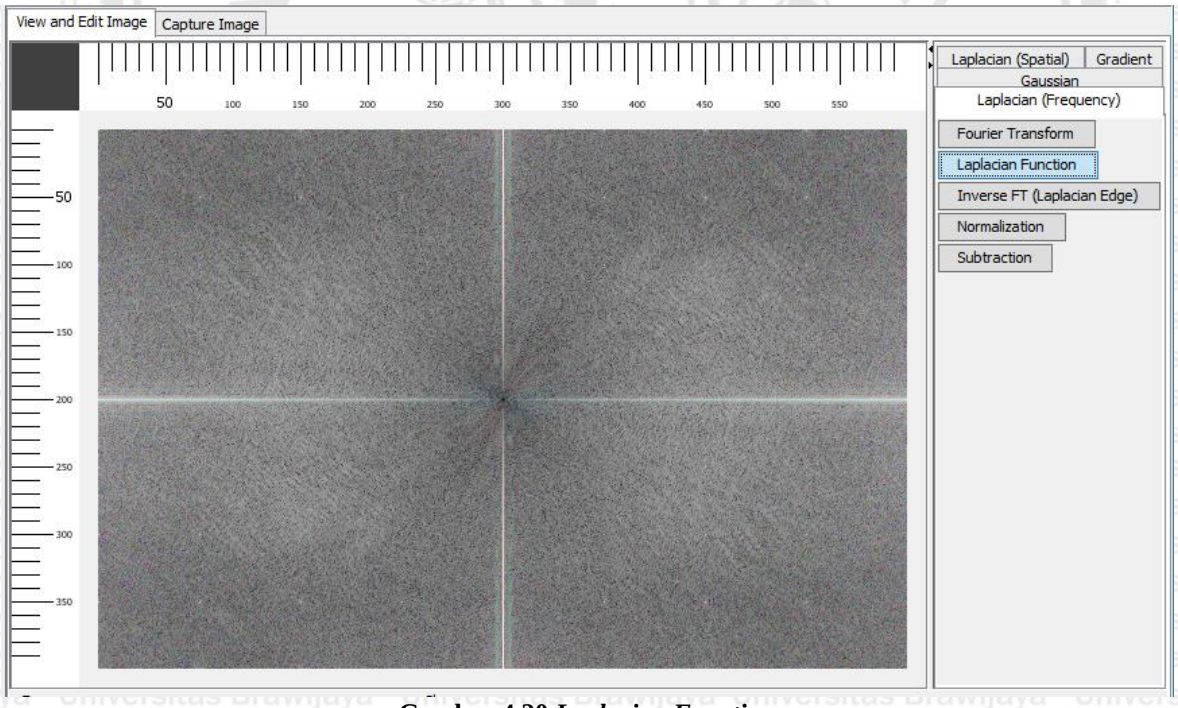
Gambar 4.18 ***Laplacian Frequency Filter***

- Pada *laplacian frequency* yang terjadi pertama adalah *fourier transform* dan waktu pengeksekusiannya. Dapat dilihat dalam Gambar 4.19.



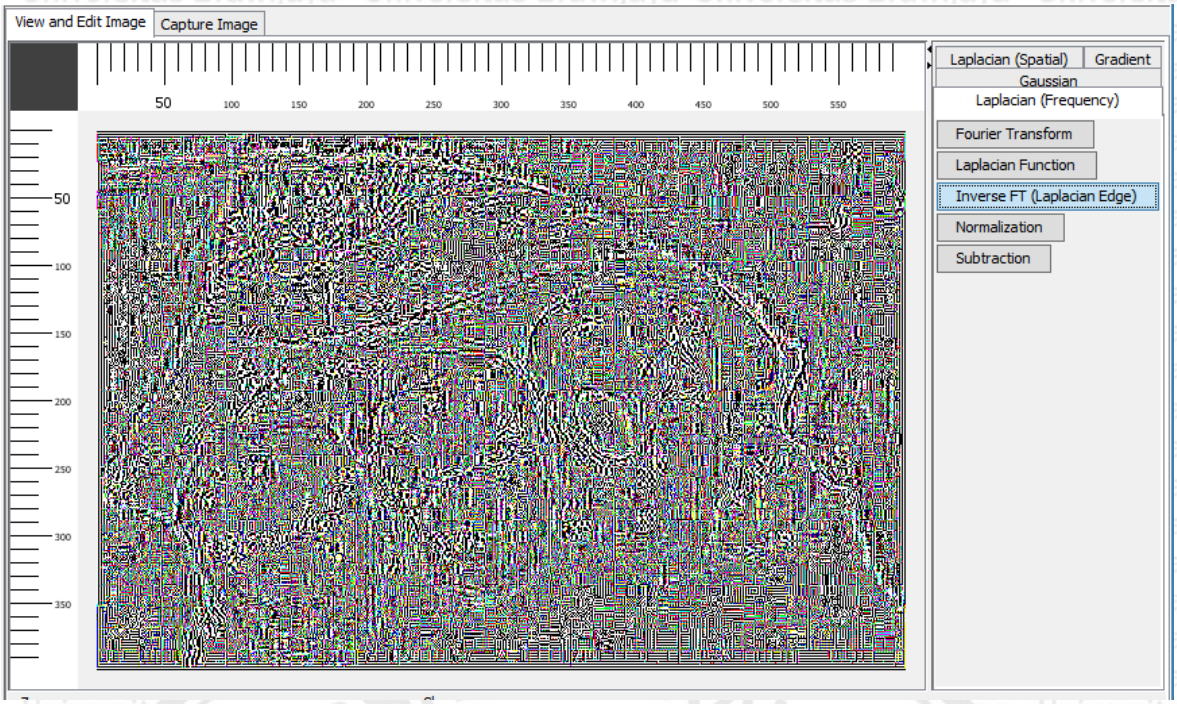
Gambar 4.19 ***Fourier Transform***

- Pada *laplacian frequency* yg kedua terjadi adalah *laplacian function* dan. Dapat dilihat dalam Gambar 4.20.



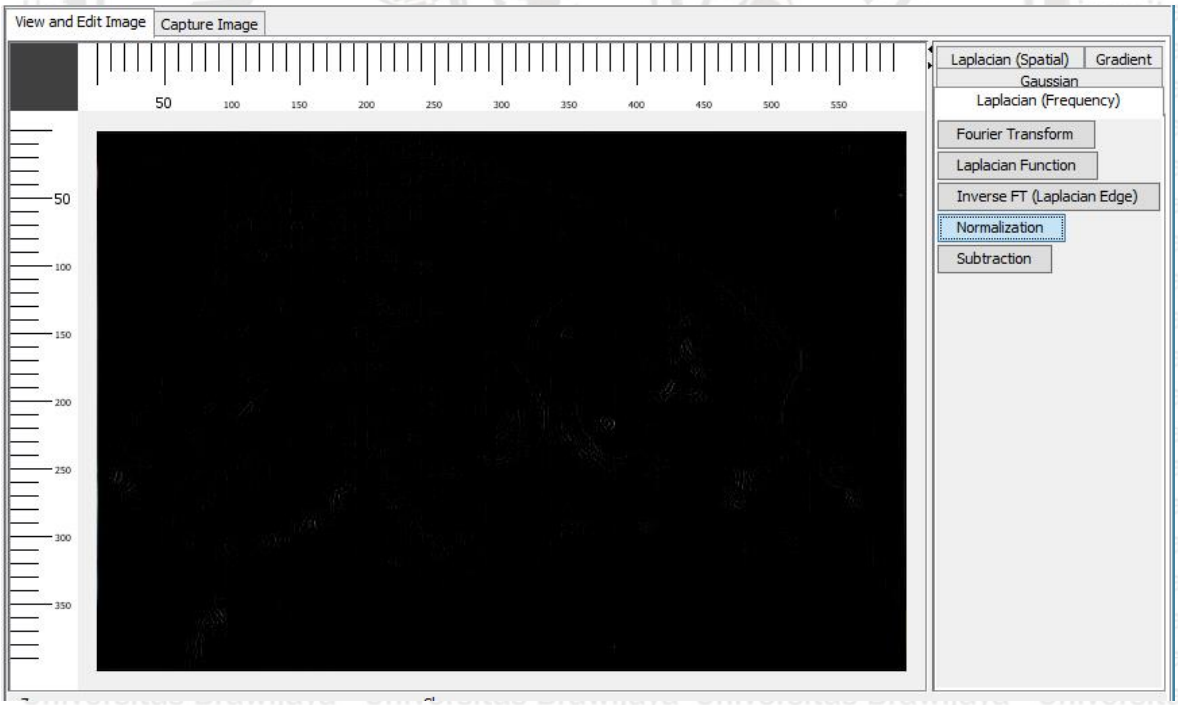
Gambar 4.20 ***Laplacian Function***

- Pada *laplacian frequency* yang ketiga terjadi adalah *Inverse FT* (*laplacian edge*). Dapat dilihat dalam Gambar 4.21.



Gambar 4.21 ***Inverse FT***

- Pada *laplacian frequency* yang keempat terjadi adalah *normalization*. Dapat dilihat dalam Gambar 4.22.



Gambar 4.22 ***Normalization***

- Pada *laplacian frequency* yang kelima atau terakhir adalah *subtraction*. Dapat dilihat dalam Gambar 4.23.

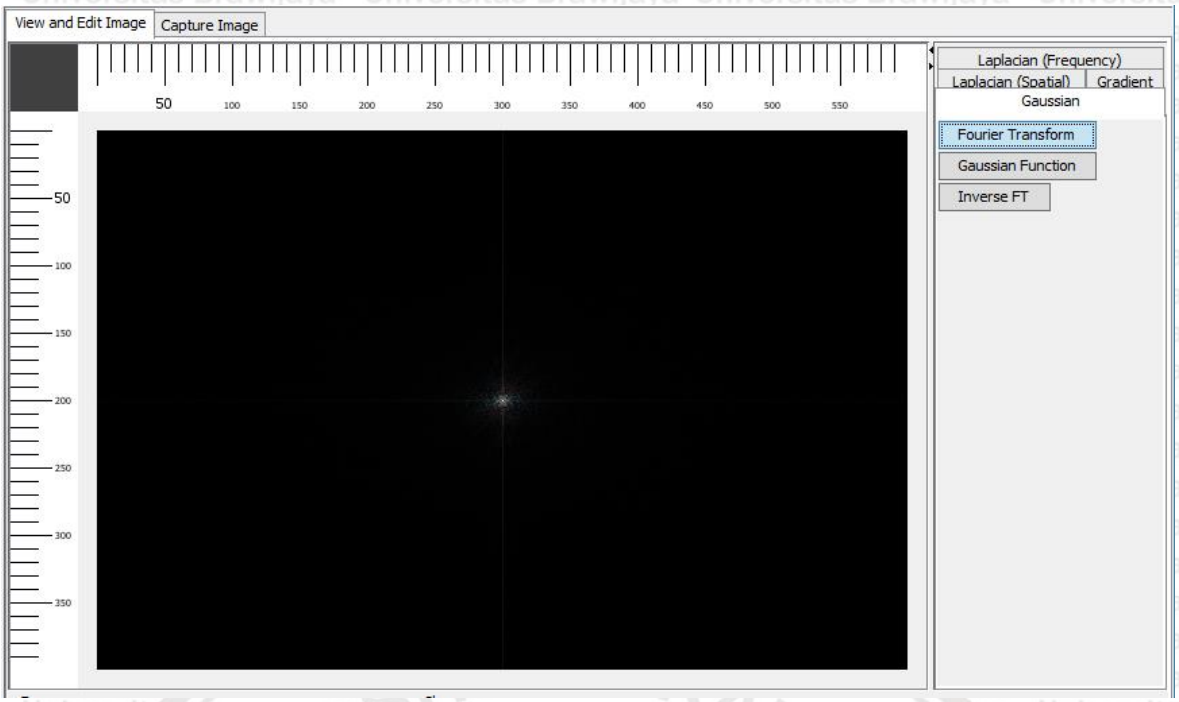
Gambar 4.23 *Substraction*

d. *Gaussian Filter*

- Filter keempat adalah *gaussian filter*. Dapat dilihat dalam Gambar 4.24.

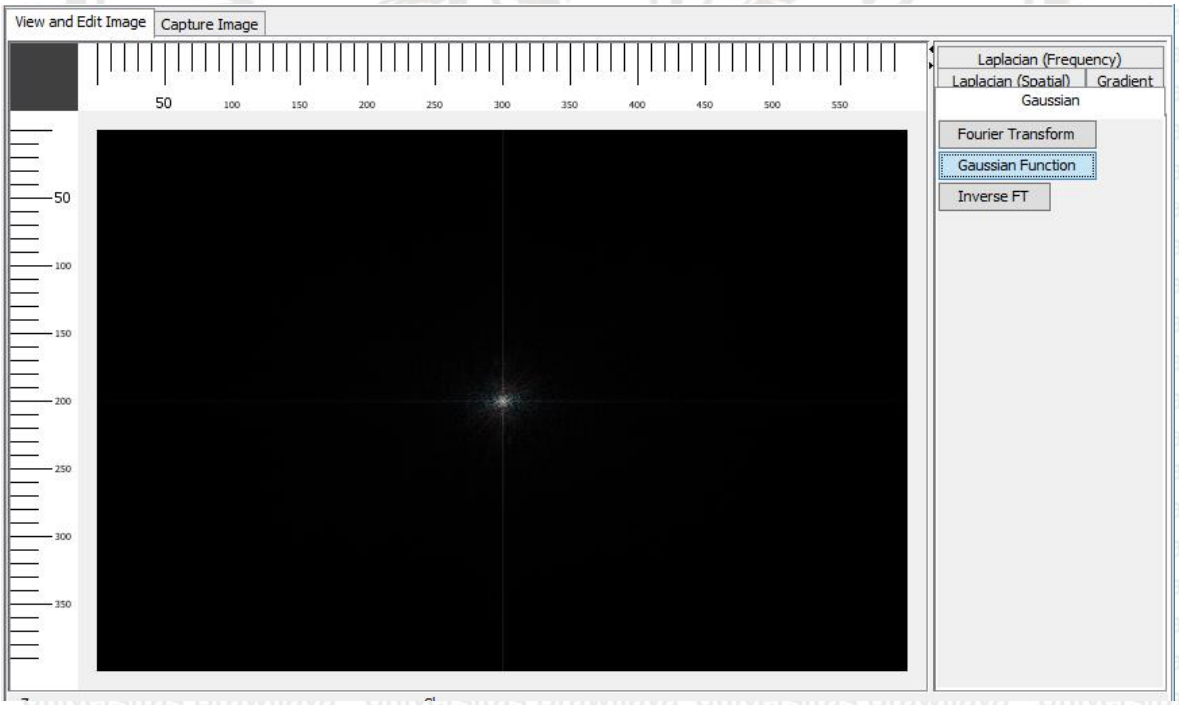
Gambar 4.24 *Gaussian Filter*

- Pada *gaussian filter* yang terjadi pertama adalah *fourier transform.* Dapat dilihat dalam Gambar 4.25.



Gambar 4.25 *Fourier Transform*

- Pada *gaussian filter* yang kedua terjadi adalah *gaussian function*. Dapat dilihat dalam Gambar 4.26.

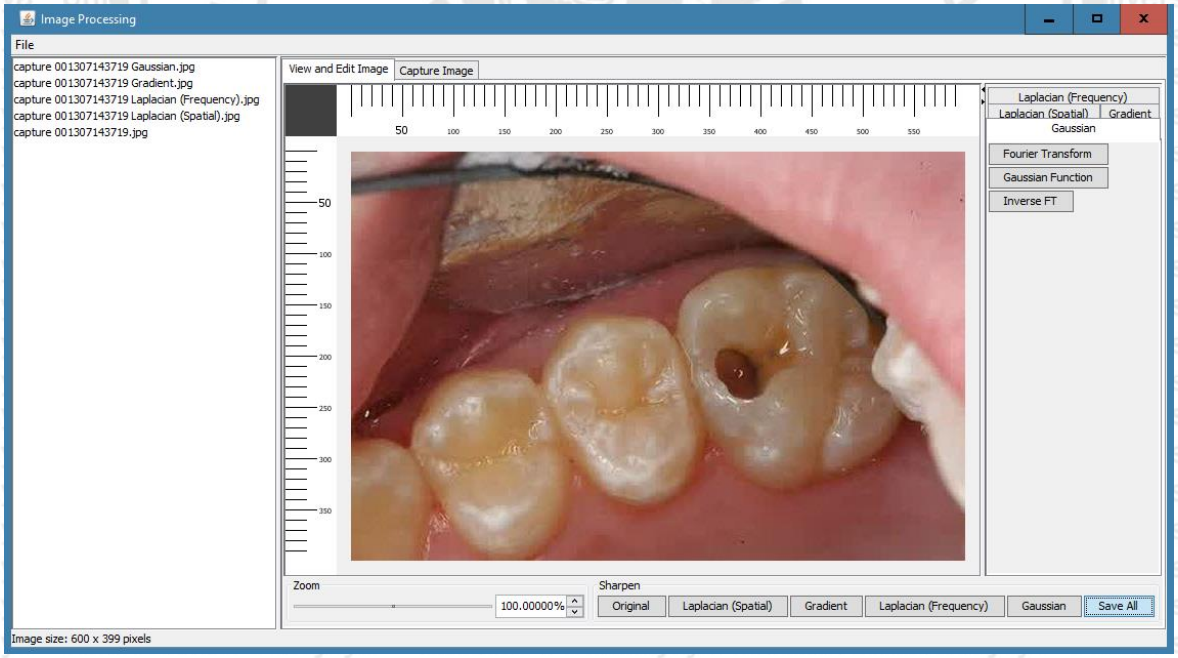


Gambar 4.26 *Gaussian Function*

- Pada *Gaussian Filter* yang ketiga atau terakhir adalah terjadinya *Inverse FT* . Dapat dilihat dalam Gambar 4.27.

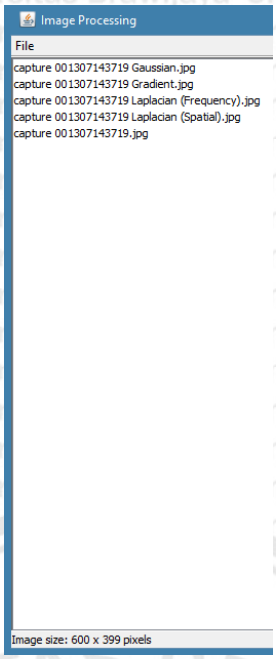
Gambar 4.27 ***Inverse FT***

- Dan pada tab terakhir adalah tab *save* dimana semua filter dari filter pertama sampai filter yang keempat (tapi tidak termasuk tata cara pemfiltera filter tersebut) gambar akan disimpan dalam satu folder yang kita inginkan. Dapat dilihat dalam Gambar 4.28.



Gambar 4.28 Tab ***Save***

- Apabila data gambar yang telah difilter sudah disimpan, akan muncul keterangan pada bagian kiri. Dapat dilihat dalam Gambar 4.29.



Gambar 4.29 Keterangan apabila gambar yang difilter sudah disimpan

4.2. Hasil Pengujian

Dari hasil pengujian sistem dengan menggunakan peralatan yang dibutuhkan tersebut dapat diketahui waktu eksekusi yang dibutuhkan dalam 4 macam filter yang digunakan. Untuk mengetahui waktu eksekusi yang dibutuhkan dapat dilihat dalam Tabel 4.1.

Tabel 4.1 Waktu Eksekusi

NAMA FILTER		WAKTU EKSEKUSI (ms)
1. Laplace Spatial		9,733037
Tahap:	a. Laplacian Edge	7,169165
	b. Normalization	0,751665
	c. Substraction	1,812207
2. Gradient		21,213768
Tahap:	a. Horizontal Edge	7,254692
	b. Vertical Edge	8,809828
	c. Gradient Edge	2,017952
	d. Normalization	1,735975
	e. Addition	1,395321
3. Frequency		460,038336
Tahap:	a. Fourier Transform	144,395941
	b. Laplacian Function	170,692978
	c. Inverse FT	115,809208
	d. Normalization	10,133911
	e. Substraction	19,006298

4. Gaussian		634,241619
Tahap:	a. Fourier Transform	148,124943
	b. Gaussian Function	386,913989
	c. Inverse FT	99,202687

Dalam percobaan ini, selain waktu eksekusi, ketelitian dan kejelasan gambar yang dihasilkan juga berbeda. Dapat dilihat dalam perbandingan hasil Gambar 4.22 dan Gambar 4.23 pada *Gaussian Filter* dan *Laplacian Frequnecy Filter*. Dalam *Gaussian Filter*, gambar dan warna yang dihasilkan lebih jelas jika dibandingkan dengan hasil dari *Laplacian Frequency Filter*.

Selain perbedaan antara filter yang satu dengan yang lain, perbedaan juga terlihat pada penggunaan jenis filter dalam satu filter yang sama. Dapat dilihat dalam Gambar 4.14 dan Gambar 4.15 pada *Gradient Filter*. Jenis filter yang digunakan yaitu *Gradient Edge* dan *Normalization*. Hasil dari *Gradient Edge* lebih jelas daripada hasil dari *Normalization.*

BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

1. Berdasarkan pengujian, rumusan masalah dapat terjawab dengan hasil yang telah didapat.
2. Dari percobaan didapatkan hasil yaitu waktu eksekusi serta ketelitian dan kejelasan objek (gambar) yang digunakan. Waktu eksekusi antara filter yang satu dengan yang lain mempunyai perbedaan nilai yang jauh, dan antara jenis yang satu dengan yang lain dalam satu filter mempunyai rentang nilai waktu eksekusi yang sedikit. Tetapi ada beberapa jenis filter yang mempunyai perbandingan nilai yang cukup jauh.
3. Pada hasil pengujian dapat penulis simpulkan bahwa terdapat perbedaan pada hasil original kamera dan hasil keluaran kamera yang telah diberi filter

5.2. Saran

Saran yang dapat diberikan pada penelitian selanjutnya yaitu penambahan komponen elektronika dan pengaplikasian langsung pada prototipe yang akan dibuat

.

DAFTAR PUSTAKA

Gonzalez, R. C. & Woods, R. E. 2008. *Digital Image Processing.* Upper Saddle River, New Jersey: Pearson Education.

OpenCV Team. 2017. *OpenCV (Open Source Computer Vision Library)*. http://www.opencv.org

Pratt, W. K. 2007. *Digital Image Processing.* Hoboken, New Jersey: John Wiley & Sons.

Rinda, Wahyu. 2013. *Penyebab serta Penanganan Beberapa Masalah Gigi dan Mulut*. Diambil dari: http://www.kompasiana.com/wahyurinda/penyebab-serta-penanganan-beberapa-masalah-gigi-dan-mulut_552a2c7af17e610467d623b3

Sulfajri, Andy Rezky. 2015. *Mikromotor*. http://andyrezkysulfajri.blogspot.co.id /2015/01/78-fungsi-dan-prinsip-kerja-alat_8.html

The Eclipse Foundation. 2017. *Eclipse*. http://www.eclipse.org

LAMPIRAN

```
package gui;

import javax.swing.JFileChooser;
import javax.swing.JFormattedTextField;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JSplitPane;
import javax.swing.JTabbedPane;
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.KeyStroke;
import javax.swing.text.DefaultFormatterFactory;
import javax.swing.text.NumberFormatter;

import org.opencv.core.Core;
import org.opencv.core.Core.MinMaxLocResult;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.Scalar;
import org.opencv.highgui.VideoCapture;
import org.opencv.imgproc.Imgproc;

import java.awt.event.KeyEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.image.BufferedImage;
import java.awt.image.DataBufferByte;
import java.io.File;
import java.io.FileFilter
```

```
mport java.io.IOException;
import java.text.NumberFormat;
import java.text.ParseException;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.List;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ComponentEvent;
import java.awt.event.InputEvent;
import javax.swing.JList;
import javax.swing.ListSelectionModel;
import javax.swing.SpinnerNumberModel;
import javax.imageio.ImageIO;
import javax.swing.AbstractListModel;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JSlider;
import java.awt.BorderLayout;
import java.awt.Component;
import java.awt.Container;
import java.awt.event.ActionListener;
import java.awt.event.ComponentAdapter;
import javax.swing.SwingConstants;
import javax.swing.border.TitledBorder;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import javax.swing.JSpinner;
import javax.swing.JLabel;
import javax.swing.JToolBar;

public final class FrameMain extends JFrame {
    private static final int CAM_WIDTH = 640;
    private static final int CAM_HEIGHT = 480;
```

```
private static final Mat LAPLACIAN_KERNEL;
private static final Mat GRADIENT_H_KERNEL;
private static final Mat GRADIENT_V_KERNEL;
private final VideoCapture webSource;
private File directory;
private BufferedImage image;

static {
        LAPLACIAN_KERNEL = new Mat(3,3, CvType.CV_64F){
                {
                        put(0,0,0);
                        put(0,1,1);
                        put(0,2,0);

                        put(1,0,1);
                        put(1,1,-4);
                        put(1,2,1);

                        put(2,0,0);
                        put(2,1,1);
                        put(2,2,0);
                }
        };
        GRADIENT_H_KERNEL = new Mat(3,3, CvType.CV_64F){
                {
                        put(0,0,-1);
                        put(0,1,-2);
                        put(0,2,-1);

                        put(1,0,0);
                        put(1,1,0);
                        put(1,2,0);

                        put(2,0,1);
                        put(2,1,2);
```

```
                put(2,2,1);
            }
        };
        GRADIENT_V_KERNEL = new Mat(3,3, CvType.CV_64F){
            {
                put(0,0,-1);
                put(0,1,0);
                put(0,2,1);

                put(1,0,-2);
                put(1,1,0);
                put(1,2,2);

                put(2,0,-1);
                put(2,1,0);
                put(2,2,1);
            }
        };
    }

    private static abstract class ImageFunction {
        protected long lastTimeExecution;
        private int subProcessCount;

        public ImageFunction(int subProcessCount) {
            if (subProcessCount < 1)
                subProcessCount = 1;
            this.subProcessCount = subProcessCount;
        }

        public static ImageFunction getComposite(ImageFunction... functions) {
            return new ImageFunction(functions.length) {

                @Override
                public Mat apply(Mat image, int subProc) {
```

```
                for (int i=0; i<functions.length; i++) {
                    image = functions[i].apply(image);
                    if (i == subProc)
                        break;
                }
                return image;
            }
        };
    }

    public ImageFunction getSubProcess(int subProc) {
        ImageFunction sourceFunction = this;
        long sourceLastTimeExecution = lastTimeExecution;
        return new ImageFunction(subProc+1) {

            @Override
            public Mat apply(Mat image, int subProc) {
                Mat result = sourceFunction.apply(image, subProc);
                lastTimeExecution = sourceFunction.lastTimeExecution;
                sourceFunction.lastTimeExecution =
sourceLastTimeExecution;

                return result;
            }
        };
    }

    public long getLastTimeExecution() {
        return lastTimeExecution;
    }

    public int getSubProcessCount() {
        return subProcessCount;
    }

    public Mat apply(Mat image) {
```

```
                return apply(image, subProcessCount-1);
        }

        public abstract Mat apply(Mat image, int subProc);
}

private static abstract class FTFunction extends ImageFunction {

        public FTFunction(int subProcessCount) {
                super(subProcessCount);
        }

        @Override
        public Mat apply(Mat image, int subProc) {
                int rows = image.rows();
                int cols = image.cols();
                for (int i = 0; i < rows; i++) {
                        for (int j = 0; j < cols; j++) {
                                double[] data = image.get(i, j);
                                int a = i-rows/2, b = j-cols/2;
                                for (int k = 0; k < data.length; k++) {
                                        data[k] *= functionH(a, b);
                                }
                                image.put(i, j, data);
                        }
                }
                return image;
        }

        public abstract double functionH(int x, int y);
}

private static class ImageFunctionButton extends JButton {
        private static ActionListener LISTENER;
        private final ImageFunction function;
```

```
public ImageFunctionButton(String s, ImageFunction function) {
        super(s);
        this.function = function;
        addActionListener(LISTENER);
}
public static void setListener(ActionListener listener) {
        LISTENER = listener;
}

public BufferedImage apply(BufferedImage image, int subProc) {
        Mat frame = bufferedImageToMat(image);
        frame = function.apply(frame, subProc);
        return matToBufferedImage(frame);
}

public long getLastTimeExecution() {
        return function.getLastTimeExecution();
}

public Mat apply(Mat frame, int subProc) {
        return function.apply(frame, subProc);
}

public BufferedImage apply(BufferedImage image) {
        return apply(image, function.getSubProcessCount()-1);
}
public Mat apply(Mat frame) {
        return function.apply(frame);
}
public ImageFunction getFunction() {
        return function;
```

```
        }
    }
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private static final class FileNameListModel extends AbstractListModel<String> {

        /**
         *
         */
        private static final long serialVersionUID = 1L;
        private File[] files;

        @Override
        public int getSize() {
            if (files == null)
                return 0;
            return files.length;
        }

        @Override
        public String getElementAt(int i) {
            if (files == null)
                return null;
            return files[i].getName();
        }

        public void setFiles(File[] files) {
            int l0 = -1, l1 = -1;
            if (this.files != null)
                l0 = this.files.length-1;
            if (files != null)
                l1 = files.length-1;
```

```
                this.files = files;
                if (l0 > -1)
                        fireIntervalRemoved(this, 0, l0);
                if (l1 > -1)
                        fireIntervalAdded(this, 0, l1);
        }
        public File[] getFiles() {
                return files;
        }
}
private final FileNameListModel listModel = new FileNameListModel();
private int selectedFileIndex = -1;
private static final FileFilter IMAGE_FILTER = new FileFilter() {

        @Override
        public boolean accept(File file) {
                if (file.isDirectory())
                        return true;

                String s = file.getName();
                int i = s.lastIndexOf('.');
                if (i < 0 || i >= s.length()-1)
                        return false;
                s = s.substring(i+1).toLowerCase();
                return s.equals("jpg") || s.equals("jpeg") || s.equals("gif") ||
                                s.equals("png") || s.equals("tif") || s.equals("tiff");
        }
};
private static final class CustomNumberFormatter extends NumberFormatter {
        /**
         *
         */
```

```
        private static final long serialVersionUID = -4126840968176625859L;
        boolean isPercent;

        public CustomNumberFormatter(NumberFormat numberFormat, boolean
isPercent) {
            super(numberFormat);
            this.isPercent = isPercent;
        }

        @Override
        public String valueToString(Object o) throws ParseException {
            Number number = (Number) o;
            if (number != null) {
                int i = number.intValue();
                double d = slideToScale(i);
                if (!isPercent)
                    d *= 100.0;
                number = new Double(d);
            }
            return super.valueToString(number);
        }

        @Override
        public Object stringToValue(String s) throws ParseException {
            Number number = (Number) super.stringToValue(s);
            if (number != null) {
                double d = number.doubleValue();
                if (!isPercent)
                    d /= 100.0;
                int i = scaleToSlide(d);
                number = new Integer(i);
            }
            return number;
        }
    }
```

```
private static final class CaptureThread extends Thread {
        private volatile boolean runnable;
        private final VideoCapture webSource;
        private final Mat frame = new Mat();
        private final CaptureDisplayPanel panel;

        CaptureThread(VideoCapture webSource, CaptureDisplayPanel panel) {
                this.webSource = webSource;
                this.panel = panel;
        }

        void setRunnable(boolean runnable) {
                this.runnable = runnable;
        }

        @Override
        public  void run() {
                synchronized(this) {
                        while(runnable) {
                                if(webSource.read(frame))
                                        panel.setImage(matToBufferedImage(frame));
                        }
                }
        }
}
/**
 * Create the frame.
 */
public FrameMain() {
        setTitle("Image Processing");
        webSource = new VideoCapture();
        addWindowListener(new WindowAdapter() {
```

```
                @Override
                public void windowClosing(WindowEvent e) {
                    webSource.release();
                    System.exit(0);
                }
            });

            JFileChooser fileChooser = new JFileChooser();
            JTabbedPane tabbedPane = new JTabbedPane();
            ImageDisplayPanel imageDisplayPanel = new ImageDisplayPanel();
            imageDisplayPanel.setToolTipText("Image display");
            CaptureDisplayPanel captureDisplayPanel = new CaptureDisplayPanel();
            captureDisplayPanel.setToolTipText("Capture display");
            JPanel imagePanel = new JPanel(new BorderLayout());
            JTabbedPane filtSubTabbedPane = new JTabbedPane();
            JSplitPane imageSplitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
imageDisplayPanel, filtSubTabbedPane);
            imageSplitPane.setOneTouchExpandable(true);
            imageSplitPane.setResizeWeight(1.0);
            imagePanel.add(imageSplitPane, BorderLayout.CENTER);
            JPanel capturePanel = new JPanel(new BorderLayout());
            capturePanel.add(captureDisplayPanel, BorderLayout.CENTER);

            JToolBar tlbrTools = new JToolBar("Tools");
            tlbrTools.setToolTipText("Tools for image processing");
            tlbrTools.setFloatable(false);
            JToolBar tlbrCaptTools = new JToolBar();
            tlbrCaptTools.setLayout(new GridLayout());
            tlbrCaptTools.setFloatable(false);
            imagePanel.add(tlbrTools, BorderLayout.SOUTH);
            capturePanel.add(tlbrCaptTools, BorderLayout.SOUTH);
            tabbedPane.add("View and Edit Image", imagePanel);
            tabbedPane.add("Capture Image", capturePanel);
            tabbedPane.addChangeListener(new ChangeListener() {
```

```
                private CaptureThread captureThread;
                @Override
                public void stateChanged(ChangeEvent changeEvent) {
                        int index = ((JTabbedPane)
changeEvent.getSource()).getSelectedIndex();
                        if (index == 1) {
                                webSource.open(1);
                                captureThread = new CaptureThread(webSource,
captureDisplayPanel);
                                captureThread.setDaemon(true);
                                captureThread.setRunnable(true);
                                captureThread.start();
                        }
                        else {
                                captureThread.setRunnable(false);
                                webSource.release();
                        }
                }
        });

        JToolBar tlbrStats = new JToolBar("Image Info and Statistics");
        tlbrStats.setToolTipText("Image information and statistical data");
        tlbrStats.setFloatable(false);

        JList<String> fileList = new JList<String>(listModel);
        fileList.setToolTipText("List of images to be processed");
        fileList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        JScrollPane scrllpnFileList = new JScrollPane(fileList);

        JMenuBar menuBar = new JMenuBar();
        setJMenuBar(menuBar);

        JMenu mnFile = new JMenu("File");
        mnFile.setMnemonic(KeyEvent.VK_F);
```

```
        menuBar.add(mnFile);

        JMenuItem mntmOpenFile = new JMenuItem("Open Directory");
        mntmOpenFile.addActionListener((actionEvent) -> {
            fileChooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
            fileChooser.setDialogTitle("Open Directory");
            int option = fileChooser.showOpenDialog(this);
            if (option != JFileChooser.APPROVE_OPTION)
                return;
            selectedFileIndex = -1;
            fileList.setSelectedIndex(-1);
            imageDisplayPanel.setImage(null);
            imageDisplayPanel.setToolTipText("Image display");
            directory = fileChooser.getSelectedFile();
            listModel.setFiles(directory.listFiles(IMAGE_FILTER));
            revalidate();
            repaint();
        });
        mntmOpenFile.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O,
InputEvent.ALT_MASK));
        mntmOpenFile.setMnemonic(KeyEvent.VK_O);
        mnFile.add(mntmOpenFile);

        JMenuItem mntmExit = new JMenuItem("Exit");
        mntmExit.addActionListener((actionEvent) -> System.exit(0));
        mntmExit.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_X,
InputEvent.ALT_MASK));
        mntmExit.setMnemonic(KeyEvent.VK_X);
        mnFile.addSeparator();
        mnFile.add(mntmExit);

        JPanel contentPane = new JPanel(new BorderLayout(5, 5));
        scrllpnFileList.setMaximumSize(scrllpnFileList.getPreferredSize());
        scrllpnFileList.setMinimumSize(new Dimension(100, 100));
```

```
imageDisplayPanel.setPreferredSize(new Dimension(CAM_WIDTH,
CAM_HEIGHT));
JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
scrllpnFileList, tabbedPane);
scrllpnFileList.addComponentListener(new ComponentAdapter() {
    private int lastWidth;

    @Override
    public void componentResized(ComponentEvent ce) {
        int width = scrllpnFileList.getWidth(), maxWidth =
scrllpnFileList.getMaximumSize().width;
        if (width < maxWidth) {
            if (lastWidth >= maxWidth)
                splitPane.setResizeWeight(0.5);
        }
        else {
            if (lastWidth < maxWidth)
                splitPane.setResizeWeight(0.0);
            if (width > maxWidth)

splitPane.setDividerLocation(scrllpnFileList.getMaximumSize().width+1);
        }
        lastWidth = width;
    }
});
JLabel lblImageSize = new JLabel("Image size: ");
tlbrStats.add(lblImageSize);
JPanel zoomPanel = new JPanel();
zoomPanel.setLayout(new BoxLayout(zoomPanel, BoxLayout.X_AXIS));
zoomPanel.setBorder(new TitledBorder(null, "Zoom", TitledBorder.LEADING,
TitledBorder.TOP, null, null));
tlbrTools.add(zoomPanel);
JSlider slider = new JSlider(SwingConstants.HORIZONTAL, 16, 240, 128);
```

```
            zoomPanel.add(slider);
            slider.setToolTipText("Slide to zoom image");
            JSpinner spinner = new JSpinner(new SpinnerNumberModel(128, 16, 240, 1));
            @SuppressWarnings("serial")
            JSpinner.NumberEditor editor = new JSpinner.NumberEditor(spinner) {
                {
                    NumberFormat displayFormat =
NumberFormat.getPercentInstance(),
                                        editFormat =
NumberFormat.getPercentInstance();
                    displayFormat.setMinimumFractionDigits(5);
                    editFormat.setMinimumFractionDigits(5);
                    NumberFormatter displayFormatter = new
CustomNumberFormatter(displayFormat, true);
                    NumberFormatter editFormatter = new
CustomNumberFormatter(editFormat, true);
                    displayFormatter.setMinimum(0.125);
                    displayFormatter.setMaximum(8.0);
                    editFormatter.setMinimum(0.125);
                    editFormatter.setMaximum(8.0);
                    JFormattedTextField ftf = getTextField();
                    ftf.setFormatterFactory(new
DefaultFormatterFactory(displayFormatter,
                                        displayFormatter, editFormatter));
                    ftf.setColumns(8);
                }
            };
            spinner.setEditor(editor);
            zoomPanel.add(spinner);
            slider.addChangeListener((changeEvent) -> {
                int sliderValue = slider.getValue();
                double scale = slideToScale(sliderValue);
                imageDisplayPanel.setScale(scale);
                spinner.setValue(sliderValue);
            });
```

```
                spinner.addChangeListener((changeEvent) -> {
                        int sliderValue = (Integer) spinner.getValue();
                        if (sliderValue != slider.getValue())
                                slider.setValue(sliderValue);
                });
                fileList.addListSelectionListener((listSelectionEvent) -> {
                        int sel = fileList.getSelectedIndex();

                        if (sel != -1 && sel != selectedFileIndex) {
                                selectedFileIndex = sel;
                                try {
                                        image =
ImageIO.read(listModel.getFiles()[selectedFileIndex]);
                                } catch (Exception e) {
                                        e.printStackTrace();
                                        JOptionPane.showMessageDialog(this, "Select Image
Error: " + e, "Error", JOptionPane.ERROR_MESSAGE);
                                }
                                if (image != null) {
                                        imageDisplayPanel.setImage(image);
                                        double scale = imageDisplayPanel.getFitScale();
                                        slider.setValue(scaleToSlide(scale));
                                        lblImageSize.setText("Image size: " +
imageDisplayPanel.getImageWidth() +
                                                        " x " +
imageDisplayPanel.getImageHeight() + " pixels");
                                        imageDisplayPanel.setToolTipText("Image display: " +
listModel.getFiles()[selectedFileIndex].getName());
                                        setEnabledComponents(filtSubTabbedPane, true);
                                        setEnabledComponents(tlbrTools, true);
                                        repaint();
                                }
                                else {
                                        setEnabledComponents(filtSubTabbedPane, false);
                                        setEnabledComponents(tlbrTools, false);
```

```
                }
            }
        });
        JPanel sharpenPanel = new JPanel();
        sharpenPanel.setLayout(new BoxLayout(sharpenPanel, BoxLayout.X_AXIS));
        sharpenPanel.setBorder(new TitledBorder(null, "Sharpen", TitledBorder.LEADING,
TitledBorder.TOP, null, null));
        tlbrTools.add(sharpenPanel);
        JButton originalButton = new JButton("Original");
        originalButton.addActionListener((actionEvent) -> {
            imageDisplayPanel.setImage(image);
            double scale = imageDisplayPanel.getFitScale();
            slider.setValue(scaleToSlide(scale));
            lblImageSize.setText("Image size: " + imageDisplayPanel.getImageWidth()
+
                    " x " + imageDisplayPanel.getImageHeight() + " pixels");
            imageDisplayPanel.setToolTipText("Image display: " +
listModel.getFiles()[selectedFileIndex].getName());
            repaint();
        });
        sharpenPanel.add(originalButton);

        ImageFunctionButton.setListener((actionEvent) -> {
            ImageFunctionButton f = (ImageFunctionButton) actionEvent.getSource();
            imageDisplayPanel.setImage(f.apply(image));
            double scale = imageDisplayPanel.getFitScale();
            slider.setValue(scaleToSlide(scale));
            lblImageSize.setText("Image size: " + imageDisplayPanel.getImageWidth()
+
                    " x " + imageDisplayPanel.getImageHeight() + " pixels |
Time execution: " + f.getLastTimeExecution() + " ns");
            imageDisplayPanel.setToolTipText("Image display: " +
listModel.getFiles()[selectedFileIndex].getName() +
```

```
                        " sharpened with " + f.getText());
                repaint();
            });
            List<ImageFunctionButton> functions = new ArrayList<ImageFunctionButton>();
            functions.add(new ImageFunctionButton("Laplacian (Spatial)", new
ImageFunction(3) {
                @Override
                public Mat apply(Mat image, int subProc) {
                    Mat result = new Mat();
                    long time = System.nanoTime();
                    Imgproc.filter2D(image, result, -1, LAPLACIAN_KERNEL);
                    if (subProc == 0) {
                        lastTimeExecution = System.nanoTime() - time;
                        return result;
                    }
//                  Mat res = result.reshape(1);
//                  MinMaxLocResult r = Core.minMaxLoc(res);
//                  System.out.println(r.maxVal);
//                  System.out.println(r.minVal);
//                  System.out.println(r.maxLoc);
//                  System.out.println(r.minLoc);
//                  System.out.println(res.submat(1, 2, 3, 4).dump());
//                  System.out.println(res.submat(0, 1, 0, 1).dump());
//                  System.out.println(res.submat(3, 4, 3, 4).dump());
//                  System.out.println(res.submat(0, 1, 9, 10).dump());
//                  System.out.println();
                    Core.normalize(result, result, -127, 127, Core.NORM_MINMAX);
//                  res = result.reshape(1);
//                  r = Core.minMaxLoc(res);
//                  System.out.println(r.maxVal);
//                  System.out.println(r.minVal);
//                  System.out.println(r.maxLoc);
//                  System.out.println(r.minLoc);
//                  System.out.println(res.submat(1, 2, 3, 4).dump());
```

```
//                  System.out.println(res.submat(0, 1, 0, 1).dump());
//                  System.out.println(res.submat(3, 4, 3, 4).dump());
//                  System.out.println(res.submat(0, 1, 9, 10).dump());
//                  System.out.println();
                    if (subProc == 1) {
                            lastTimeExecution = System.nanoTime() - time;
                            return result;
                    }
                    Core.subtract(image, result, result);
                    lastTimeExecution = System.nanoTime() - time;
                    return result;
            }
    }));
    functions.add(new ImageFunctionButton("Gradient", new ImageFunction(5) {

            @Override
            public Mat apply(Mat image, int subProc) {
                    Mat[] results = new Mat[3];
                    for (int i=0; i<results.length; i++) {
                            results[i] = new Mat();
                    }
                    long time = System.nanoTime();
                    Imgproc.filter2D(image, results[0], -1, GRADIENT_H_KERNEL);
                    Core.absdiff(results[0], Scalar.all(0), results[0]);
                    if (subProc == 0) {
                            lastTimeExecution = System.nanoTime() - time;
                            return results[0];
                    }
                    Imgproc.filter2D(image, results[1], -1, GRADIENT_V_KERNEL);
                    Core.absdiff(results[1], Scalar.all(0), results[1]);
                    if (subProc == 1) {
                            lastTimeExecution = System.nanoTime() - time;
                            return results[1];
                    }
                    Core.add(results[0], results[1], results[2]);
```

```
                if (subProc == 2) {
                        lastTimeExecution = System.nanoTime() - time;
                        return results[2];
                }
                Core.normalize(results[2], results[2], 0, 255,
Core.NORM_MINMAX);
                if (subProc == 3) {
                        lastTimeExecution = System.nanoTime() - time;
                        return results[2];
                }
                Core.add(image, results[2], results[2]);
                lastTimeExecution = System.nanoTime() - time;
                return results[2];
        }
    }));
    functions.add(new ImageFunctionButton("Laplacian (Frequency)", new
ImageFunction(5) {

        @Override
        public Mat apply(Mat image, int subProc) {
                Mat[] results = new Mat[3];
                for (int i=0; i<results.length; i++) {
                        results[i] = new Mat();
                }
                long time = System.nanoTime();
                long timeStamp = frequencyTransformFilter(image, results[0],
results[1], results[2], subProc, new FTFunction(1) {

                        @Override
                        public double functionH(int x, int y) {
                                return -(x*x+y*y);
                        }
                });
                if (subProc >= 0 && subProc < 3) {
                        lastTimeExecution = timeStamp - time;
```

```
                return results[subProc];
            }
//          Mat res = results[2].reshape(1);
//          MinMaxLocResult r = Core.minMaxLoc(res);
//          System.out.println(r.maxVal);
//          System.out.println(r.minVal);
//          System.out.println(r.maxLoc);
//          System.out.println(r.minLoc);
//          System.out.println(res.submat(1, 2, 3, 4).dump());
//          System.out.println(res.submat(0, 1, 0, 1).dump());
//          System.out.println(res.submat(3, 4, 3, 4).dump());
//          System.out.println(res.submat(0, 1, 9, 10).dump());
//          System.out.println();
            Core.normalize(results[2], results[2], -127, 127,
Core.NORM_MINMAX);
//          res = results[2].reshape(1);
//          r = Core.minMaxLoc(results[2].reshape(1));
//          System.out.println(r.maxVal);
//          System.out.println(r.minVal);
//          System.out.println(r.maxLoc);
//          System.out.println(r.minLoc);
//          System.out.println(res.submat(1, 2, 3, 4).dump());
//          System.out.println(res.submat(0, 1, 0, 1).dump());
//          System.out.println(res.submat(3, 4, 3, 4).dump());
//          System.out.println(res.submat(0, 1, 9, 10).dump());
//          System.out.println();
            if (subProc == 3) {
                lastTimeExecution = System.nanoTime() - time;
                return results[2];
            }
            Core.subtract(image, results[2], results[2]);
            lastTimeExecution = System.nanoTime() - time;
            return results[2];
        }
    }));
```

```
functions.add(new ImageFunctionButton("Gaussian", new ImageFunction(3) {
        @Override
        public Mat apply(Mat image, int subProc) {
            Mat[] results = new Mat[3];
            for (int i=0; i<results.length; i++) {
                results[i] = new Mat();
            }
            long time = System.nanoTime();
            long timeStamp = frequencyTransformFilter(image, results[0],
results[1], results[2], subProc, new FTFunction(1) {

                @Override
                public double functionH(int x, int y) {
                    return (1.5-Math.exp(-
Math.sqrt(x*x+y*y)*4000000));
                }
            });
            lastTimeExecution = timeStamp - time;
            return results[subProc >= 0 && subProc < 3 ? subProc : 2];
        }
    }));

    JPanel filtSubPanel = new JPanel();
    filtSubPanel.setLayout(new BoxLayout(filtSubPanel, BoxLayout.Y_AXIS));
    filtSubPanel.add(new ImageFunctionButton("Laplacian Edge",
functions.get(0).getFunction().getSubProcess(0)));
    filtSubPanel.add(new ImageFunctionButton("Normalization",
functions.get(0).getFunction().getSubProcess(1)));
    filtSubPanel.add(new ImageFunctionButton("Subtraction",
functions.get(0).getFunction()));
    filtSubTabbedPane.add(functions.get(0).getText(), filtSubPanel);
    sharpenPanel.add(functions.get(0));
    filtSubPanel = new JPanel();
    filtSubPanel.setLayout(new BoxLayout(filtSubPanel, BoxLayout.Y_AXIS));
```

```
                filtSubPanel.add(new ImageFunctionButton("Horizontal Edge",
functions.get(1).getFunction().getSubProcess(0)));
                filtSubPanel.add(new ImageFunctionButton("Vertical Edge",
functions.get(1).getFunction().getSubProcess(1)));
                filtSubPanel.add(new ImageFunctionButton("Gradient Edge",
functions.get(1).getFunction().getSubProcess(2)));
                filtSubPanel.add(new ImageFunctionButton("Normalization",
functions.get(1).getFunction().getSubProcess(3)));
                filtSubPanel.add(new ImageFunctionButton("Addition",
functions.get(1).getFunction()));
                filtSubTabbedPane.add(functions.get(1).getText(), filtSubPanel);
                sharpenPanel.add(functions.get(1));
                filtSubPanel = new JPanel();
                filtSubPanel.setLayout(new BoxLayout(filtSubPanel, BoxLayout.Y_AXIS));
                filtSubPanel.add(new ImageFunctionButton("Fourier Transform",
functions.get(2).getFunction().getSubProcess(0)));
                filtSubPanel.add(new ImageFunctionButton("Laplacian Function",
functions.get(2).getFunction().getSubProcess(1)));
                filtSubPanel.add(new ImageFunctionButton("Inverse FT (Laplacian Edge)",
functions.get(2).getFunction().getSubProcess(2)));
                filtSubPanel.add(new ImageFunctionButton("Normalization",
functions.get(2).getFunction().getSubProcess(3)));
                filtSubPanel.add(new ImageFunctionButton("Subtraction",
functions.get(2).getFunction()));
                filtSubTabbedPane.add(functions.get(2).getText(), filtSubPanel);
                sharpenPanel.add(functions.get(2));
                filtSubPanel = new JPanel();
                filtSubPanel.setLayout(new BoxLayout(filtSubPanel, BoxLayout.Y_AXIS));
                filtSubPanel.add(new ImageFunctionButton("Fourier Transform",
functions.get(3).getFunction().getSubProcess(0)));
                filtSubPanel.add(new ImageFunctionButton("Gaussian Function",
functions.get(3).getFunction().getSubProcess(1)));
                filtSubPanel.add(new ImageFunctionButton("Inverse FT",
functions.get(3).getFunction()));
                filtSubTabbedPane.add(functions.get(3).getText(), filtSubPanel);
```

```
            sharpenPanel.add(functions.get(3));
            JButton saveAllButton = new JButton("Save All");
            saveAllButton.addActionListener((actionEvent) -> {
                int response = JOptionPane.showConfirmDialog(this, "Save all filtered
images? " +
                        "The previously saved images with the same filename will
be overwritten.", "Save All", JOptionPane.YES_NO_OPTION);
                if (response == JOptionPane.YES_OPTION) {
                    String s =
listModel.getFiles()[selectedFileIndex].getAbsolutePath();
                    try {
                        ImageIO.write(functions.get(0).apply(image), "JPG", new
File(s.substring(0, s.lastIndexOf('.')) + " Laplacian (Spatial).jpg"));
                        ImageIO.write(functions.get(1).apply(image), "JPG", new
File(s.substring(0, s.lastIndexOf('.')) + " Gradient.jpg"));
                        ImageIO.write(functions.get(2).apply(image), "JPG", new
File(s.substring(0, s.lastIndexOf('.')) + " Laplacian (Frequency).jpg"));
                        ImageIO.write(functions.get(3).apply(image), "JPG", new
File(s.substring(0, s.lastIndexOf('.')) + " Gaussian.jpg"));
                    } catch (IOException e) {
                        e.printStackTrace();
                        JOptionPane.showMessageDialog(this, "Write Image
Error: " + e, "Error", JOptionPane.ERROR_MESSAGE);
                    }
                    listModel.setFiles(directory.listFiles(IMAGE_FILTER));
                    revalidate();
                    repaint();
                }
            });
            sharpenPanel.add(saveAllButton);
            setEnabledComponents(filtSubTabbedPane, false);
            setEnabledComponents(tlbrTools, false);
            JButton captureButton = new JButton("Capture");
```

```
            captureButton.addActionListener((actionEvent) -> {
                if (directory == null)
                    JOptionPane.showMessageDialog(this, "Buka direktori sek bos!
(File > Open Directory)");
                else {
                    BufferedImage image = captureDisplayPanel.getImage();
                    if (image != null) {
                        try {
                            ImageIO.write(captureDisplayPanel.getImage(),
"JPG", new File(directory.getAbsolutePath() + "/capture " +

        ZonedDateTime.now().format(DateTimeFormatter.ofPattern("dd MMM uuuu
HH.mm.ss.SSS")) + ".jpg"));
                        } catch (IOException e) {
                            e.printStackTrace();
                            JOptionPane.showMessageDialog(this, "Capture
Image Error: " + e, "Error", JOptionPane.ERROR_MESSAGE);
                        }
                        listModel.setFiles(directory.listFiles(IMAGE_FILTER));
                        revalidate();
                        repaint();
                    }
                    else
                        JOptionPane.showMessageDialog(this, "No Image
Captured", "Error", JOptionPane.ERROR_MESSAGE);
                }
            });
            tlbrCaptTools.add(captureButton);

            JPanel panel = new JPanel(new BorderLayout());
            panel.add(splitPane, BorderLayout.CENTER);
            panel.add(tlbrStats, BorderLayout.SOUTH);
            contentPane.add(panel, BorderLayout.CENTER);
            setContentPane(contentPane);
        }
```

```
    private static void setEnabledComponents(Container container, boolean b) {
        for (Component c : container.getComponents()) {
            c.setEnabled(b);
            if (c instanceof Container)
                setEnabledComponents((Container) c, b);
        }
    }

    private static void preProcFT(Mat frame) {
        for (int i = 0; i < frame.rows(); i++) {
            for (int j = 0; j < frame.cols(); j++) {
                double[] data = frame.get(i, j);
                for (int k = 0; k < data.length; k++) {
                    if ((i+j)%2==1)
                        data[k] = -data[k];
                }
                frame.put(i, j, data);
            }
        }
    }

    private static long frequencyTransformFilter(Mat frame, Mat dft, Mat idft, Mat result, int
subProc, ImageFunction function) {
        long timeStamp;
        Imgproc.copyMakeBorder(frame, result, 0, Core.getOptimalDFTSize(frame.rows())
- frame.rows(),
                0, Core.getOptimalDFTSize(frame.cols()) - frame.cols(),
Imgproc.BORDER_CONSTANT, Scalar.all(0));
        preProcFT(result);
        List<Mat> matList = new ArrayList<Mat>(), matList2 = new ArrayList<Mat>();
        Core.split(result, matList);
        matList2.add(matList.get(0));
        matList2.add(Mat.zeros(result.size(), CvType.CV_64F));
        Core.merge(matList2, matList.get(0));
```

```
            Core.dft(matList.get(0), matList.get(0),Core.DFT_SCALE,0);
            matList2.remove(0);
            matList2.add(0, matList.get(1));
            Core.merge(matList2, matList.get(1));
            Core.dft(matList.get(1), matList.get(1),Core.DFT_SCALE,0);
            matList2.remove(0);
            matList2.add(0, matList.get(2));
            Core.merge(matList2, matList.get(2));
            Core.dft(matList.get(2), matList.get(2),Core.DFT_SCALE,0);
            createOptimizedMagnitude(matList).copyTo(dft);
            timeStamp = System.nanoTime();
            function.apply(matList.get(0));
            function.apply(matList.get(1));
            function.apply(matList.get(2));
            createOptimizedMagnitude(matList).copyTo(idft);
            if (subProc == 1)
                  timeStamp = System.nanoTime();
            Core.idft(matList.get(0), matList.get(0));
            Core.split(matList.remove(0), matList2);
            matList.add(0, matList2.get(0));
            Core.idft(matList.get(1), matList.get(1));
            Core.split(matList.remove(1), matList2);
            matList.add(1, matList2.get(0));
            Core.idft(matList.get(2), matList.get(2));
            Core.split(matList.remove(2), matList2);
            matList.add(2, matList2.get(0));
            Core.merge(matList, result);
            preProcFT(result);
            result.submat(0, frame.rows(), 0, frame.cols()).copyTo(result);
            if (subProc != 0 && subProc != 1)
                  timeStamp = System.nanoTime();
            return timeStamp;
      }
      private static Mat createOptimizedMagnitude(List<Mat> complexImage)
```

```
    {
        List<Mat> newPlanes = new ArrayList<>();
        List<Mat> matList = new ArrayList<>();
        matList.add(new Mat());
        matList.add(new Mat());
        matList.add(new Mat());
        Mat mag = new Mat();
        for (int i = 0; i < 3; i++) {
            Core.split(complexImage.get(i), newPlanes);
            Core.magnitude(newPlanes.get(0), newPlanes.get(1), matList.get(i));
            Core.add(Mat.ones(matList.get(i).size(),
CvType.CV_64FC(matList.get(i).channels())), matList.get(i), matList.get(i));
            Core.log(matList.get(i), matList.get(i));
        }
        Core.merge(matList, mag);
        Core.normalize(mag, mag, 0, 255, Core.NORM_MINMAX);
        return mag;
    }

    private static BufferedImage matToBufferedImage(Mat frame) {
        int type = BufferedImage.TYPE_CUSTOM;
        if (frame.channels() == 1) {
            type = BufferedImage.TYPE_BYTE_GRAY;
        } else if (frame.channels() == 3) {
            type = BufferedImage.TYPE_3BYTE_BGR;
        }
        BufferedImage image = new BufferedImage(frame.width(), frame.height(), type);
        byte[] data = ((DataBufferByte) image.getRaster().getDataBuffer()).getData();
        Mat frame8U = new Mat();
        frame.convertTo(frame8U, CvType.CV_8UC(frame.channels()));
        frame8U.get(0, 0, data);
        return image;
    }

    private static Mat bufferedImageToMat(BufferedImage image) {
```

```
            byte[] data = ((DataBufferByte) image.getRaster().getDataBuffer()).getData();
            int channels = image.getType() == BufferedImage.TYPE_3BYTE_BGR ? 3 : 1;
            Mat frame = new Mat(image.getHeight(), image.getWidth(),
CvType.CV_8UC(channels));
            frame.put(0, 0, data);
            frame.convertTo(frame, CvType.CV_64FC(channels));
            if (channels == 1) {
                List<Mat> matList = new ArrayList<Mat>();
                matList.add(frame.clone());
                matList.add(frame.clone());
                matList.add(frame.clone());
                Core.merge(matList, frame);
            }
            return frame;
        }

        private static int scaleToSlide(double scale) {
            return Double.compare(scale, 1.0) < 0 ? (int) (scale*128.0) : (int)
(scale*16.0)+112;
        }

        private static double slideToScale(int sliderValue) {
            return sliderValue < 128 ? sliderValue/128.0 : (sliderValue-112)/16.0;
        }
}
```

LAMPIRAN

```
package gui;

import javax.swing.JFileChooser;
import javax.swing.JFormattedTextField;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JSplitPane;
import javax.swing.JTabbedPane;
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.KeyStroke;
import javax.swing.text.DefaultFormatterFactory;
import javax.swing.text.NumberFormatter;

import org.opencv.core.Core;
import org.opencv.core.Core.MinMaxLocResult;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.Scalar;
import org.opencv.highgui.VideoCapture;
import org.opencv.imgproc.Imgproc;

import java.awt.event.KeyEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.image.BufferedImage;
```

```
import java.awt.image.DataBufferByte;
import java.io.File;
import java.io.FileFilter;
import java.io.IOException;
import java.text.NumberFormat;
import java.text.ParseException;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.List;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ComponentEvent;
import java.awt.event.InputEvent;
import javax.swing.JList;
import javax.swing.ListSelectionModel;
import javax.swing.SpinnerNumberModel;
import javax.imageio.ImageIO;
import javax.swing.AbstractListModel;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JSlider;
import java.awt.BorderLayout;
import java.awt.Component;
import java.awt.Container;
import java.awt.event.ActionListener;
import java.awt.event.ComponentAdapter;
import javax.swing.SwingConstants;
import javax.swing.border.TitledBorder;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import javax.swing.JSpinner;
```

```
import javax.swing.JLabel;
import javax.swing.JToolBar;

public final class FrameMain extends JFrame {
    private static final int CAM_WIDTH = 640;
    private static final int CAM_HEIGHT = 480;
    private static final Mat LAPLACIAN_KERNEL;
    private static final Mat GRADIENT_H_KERNEL;
    private static final Mat GRADIENT_V_KERNEL;
    private final VideoCapture webSource;
    private File directory;
    private BufferedImage image;

    static {
        LAPLACIAN_KERNEL = new Mat(3,3, CvType.CV_64F){
            {
                put(0,0,0);
                put(0,1,1);
                put(0,2,0);

                put(1,0,1);
                put(1,1,-4);
                put(1,2,1);

                put(2,0,0);
                put(2,1,1);
                put(2,2,0);
            }
        };
        GRADIENT_H_KERNEL = new Mat(3,3, CvType.CV_64F){
            {
                put(0,0,-1);
```

```
                put(0,1,-2);
                put(0,2,-1);

                put(1,0,0);
                put(1,1,0);
                put(1,2,0);

                put(2,0,1);
                put(2,1,2);
                put(2,2,1);
            }
        };
        GRADIENT_V_KERNEL = new Mat(3,3, CvType.CV_64F){
            {
                put(0,0,-1);
                put(0,1,0);
                put(0,2,1);

                put(1,0,-2);
                put(1,1,0);
                put(1,2,2);

                put(2,0,-1);
                put(2,1,0);
                put(2,2,1);
            }
        };
    }

    private static abstract class ImageFunction {
        protected long lastTimeExecution;
        private int subProcessCount;
```

```
public ImageFunction(int subProcessCount) {
        if (subProcessCount < 1)
                subProcessCount = 1;
        this.subProcessCount = subProcessCount;
}
public static ImageFunction getComposite(ImageFunction... functions) {
        return new ImageFunction(functions.length) {

                @Override
                public Mat apply(Mat image, int subProc) {
                        for (int i=0; i<functions.length; i++) {
                                image = functions[i].apply(image);
                                if (i == subProc)
                                        break;
                        }
                        return image;
                }
        };
}

public ImageFunction getSubProcess(int subProc) {
        ImageFunction sourceFunction = this;
        long sourceLastTimeExecution = lastTimeExecution;
        return new ImageFunction(subProc+1) {

                @Override
                public Mat apply(Mat image, int subProc) {
                        Mat result = sourceFunction.apply(image, subProc);
                        lastTimeExecution = sourceFunction.lastTimeExecution;
```

```
                        sourceFunction.lastTimeExecution =
sourceLastTimeExecution;
                        return result;
                    }
                };
            }

            public long getLastTimeExecution() {
                return lastTimeExecution;
            }

            public int getSubProcessCount() {
                return subProcessCount;
            }

            public Mat apply(Mat image) {
                return apply(image, subProcessCount-1);
            }

            public abstract Mat apply(Mat image, int subProc);
        }

        private static abstract class FTFunction extends ImageFunction {

            public FTFunction(int subProcessCount) {
                super(subProcessCount);
            }

            @Override
            public Mat apply(Mat image, int subProc) {
                int rows = image.rows();
                int cols = image.cols();
```

```
                for (int i = 0; i < rows; i++) {
                        for (int j = 0; j < cols; j++) {
                                double[] data = image.get(i, j);
                                int a = i-rows/2, b = j-cols/2;
                                for (int k = 0; k < data.length; k++) {
                                        data[k] *= functionH(a, b);
                                }
                                image.put(i, j, data);
                        }
                }
                return image;
        }

        public abstract double functionH(int x, int y);
}

private static class ImageFunctionButton extends JButton {
        private static ActionListener LISTENER;
        private final ImageFunction function;

        public ImageFunctionButton(String s, ImageFunction function) {
                super(s);
                this.function = function;
                addActionListener(LISTENER);
        }

        public static void setListener(ActionListener listener) {
                LISTENER = listener;
        }

        public BufferedImage apply(BufferedImage image, int subProc) {
                Mat frame = bufferedImageToMat(image);
```

```
                frame = function.apply(frame, subProc);
                return matToBufferedImage(frame);
        }

        public long getLastTimeExecution() {
                return function.getLastTimeExecution();
        }

        public Mat apply(Mat frame, int subProc) {
                return function.apply(frame, subProc);
        }

        public BufferedImage apply(BufferedImage image) {
                return apply(image, function.getSubProcessCount()-1);
        }

        public Mat apply(Mat frame) {
                return function.apply(frame);
        }

        public ImageFunction getFunction() {
                return function;
        }
}

 /**
  *
  */
 private static final long serialVersionUID = 1L;
 private static final class FileNameListModel extends AbstractListModel<String> {

        /**
```

```
 *
 */
private static final long serialVersionUID = 1L;
private File[] files;

@Override
public int getSize() {
        if (files == null)
                return 0;
        return files.length;
}

@Override
public String getElementAt(int i) {
        if (files == null)
                return null;
        return files[i].getName();
}

public void setFiles(File[] files) {
        int l0 = -1, l1 = -1;
        if (this.files != null)
                l0 = this.files.length-1;
        if (files != null)
                l1 = files.length-1;
        this.files = files;
        if (l0 > -1)
                fireIntervalRemoved(this, 0, l0);
        if (l1 > -1)
                fireIntervalAdded(this, 0, l1);
}
```

```
        public File[] getFiles() {
                return files;
        }
}
private final FileNameListModel listModel = new FileNameListModel();
private int selectedFileIndex = -1;
private static final FileFilter IMAGE_FILTER = new FileFilter() {

        @Override
        public boolean accept(File file) {
                if (file.isDirectory())
                        return true;

                String s = file.getName();
                int i = s.lastIndexOf('.');
                if (i < 0 || i >= s.length()-1)
                        return false;
                s = s.substring(i+1).toLowerCase();
                return s.equals("jpg") || s.equals("jpeg") || s.equals("gif") ||
                                s.equals("png") || s.equals("tif") || s.equals("tiff");
        }

};

private static final class CustomNumberFormatter extends NumberFormatter {
        /**
         *
         */
        private static final long serialVersionUID = -4126840968176625859L;
        boolean isPercent;

        public CustomNumberFormatter(NumberFormat numberFormat, boolean isPercent) {
```

```
            super(numberFormat);
            this.isPercent = isPercent;
        }
        @Override
        public String valueToString(Object o) throws ParseException {
            Number number = (Number) o;
            if (number != null) {
                int i = number.intValue();
                double d = slideToScale(i);
                if (!isPercent)
                    d *= 100.0;
                number = new Double(d);
            }
            return super.valueToString(number);
        }

        @Override
        public Object stringToValue(String s) throws ParseException {
            Number number = (Number) super.stringToValue(s);
            if (number != null) {
                double d = number.doubleValue();
                if (!isPercent)
                    d /= 100.0;
                int i = scaleToSlide(d);
                number = new Integer(i);
            }
            return number;
        }
    }

    private static final class CaptureThread extends Thread {
```

```
        private volatile boolean runnable;
        private final VideoCapture webSource;
        private final Mat frame = new Mat();
        private final CaptureDisplayPanel panel;

        CaptureThread(VideoCapture webSource, CaptureDisplayPanel panel) {
                this.webSource = webSource;
                this.panel = panel;
        }

        void setRunnable(boolean runnable) {
                this.runnable = runnable;
        }

        @Override
        public  void run() {
                synchronized(this) {
                        while(runnable) {
                                if(webSource.read(frame))
                                        panel.setImage(matToBufferedImage(frame));
                        }
                }
        }
}

/**
 * Create the frame.
 */
public FrameMain() {
        setTitle("Image Processing");
        webSource = new VideoCapture();
        addWindowListener(new WindowAdapter() {
```

```
                @Override
                public void windowClosing(WindowEvent e) {
                        webSource.release();
                        System.exit(0);
                }
        });

        JFileChooser fileChooser = new JFileChooser();
        JTabbedPane tabbedPane = new JTabbedPane();
        ImageDisplayPanel imageDisplayPanel = new ImageDisplayPanel();
        imageDisplayPanel.setToolTipText("Image display");
        CaptureDisplayPanel captureDisplayPanel = new CaptureDisplayPanel();
        captureDisplayPanel.setToolTipText("Capture display");
        JPanel imagePanel = new JPanel(new BorderLayout());
        JTabbedPane filtSubTabbedPane = new JTabbedPane();
        JSplitPane imageSplitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
imageDisplayPanel, filtSubTabbedPane);
        imageSplitPane.setOneTouchExpandable(true);
        imageSplitPane.setResizeWeight(1.0);
        imagePanel.add(imageSplitPane, BorderLayout.CENTER);
        JPanel capturePanel = new JPanel(new BorderLayout());
        capturePanel.add(captureDisplayPanel, BorderLayout.CENTER);

        JToolBar tlbrTools = new JToolBar("Tools");
        tlbrTools.setToolTipText("Tools for image processing");
        tlbrTools.setFloatable(false);
        JToolBar tlbrCaptTools = new JToolBar();
        tlbrCaptTools.setLayout(new GridLayout());
        tlbrCaptTools.setFloatable(false);
        imagePanel.add(tlbrTools, BorderLayout.SOUTH);
        capturePanel.add(tlbrCaptTools, BorderLayout.SOUTH);
```

```
tabbedPane.add("View and Edit Image", imagePanel);
tabbedPane.add("Capture Image", capturePanel);

tabbedPane.addChangeListener(new ChangeListener() {
    private CaptureThread captureThread;

    @Override
    public void stateChanged(ChangeEvent changeEvent) {
        int index = ((JTabbedPane)
changeEvent.getSource()).getSelectedIndex();
        if (index == 1) {
            webSource.open(1);
            captureThread = new CaptureThread(webSource,
captureDisplayPanel);
            captureThread.setDaemon(true);
            captureThread.setRunnable(true);
            captureThread.start();
        }
        else {
            captureThread.setRunnable(false);
            webSource.release();
        }
    }
});

JToolBar tlbrStats = new JToolBar("Image Info and Statistics");
tlbrStats.setToolTipText("Image information and statistical data");
tlbrStats.setFloatable(false);

JList<String> fileList = new JList<String>(listModel);
fileList.setToolTipText("List of images to be processed");
fileList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

```
        JScrollPane scrllpnFileList = new JScrollPane(fileList);

        JMenuBar menuBar = new JMenuBar();
        setJMenuBar(menuBar);

        JMenu mnFile = new JMenu("File");
        mnFile.setMnemonic(KeyEvent.VK_F);
        menuBar.add(mnFile);

        JMenuItem mntmOpenFile = new JMenuItem("Open Directory");
        mntmOpenFile.addActionListener((actionEvent) -> {
                fileChooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
                fileChooser.setDialogTitle("Open Directory");
                int option = fileChooser.showOpenDialog(this);
                if (option != JFileChooser.APPROVE_OPTION)
                        return;
                selectedFileIndex = -1;
                fileList.setSelectedIndex(-1);
                imageDisplayPanel.setImage(null);
                imageDisplayPanel.setToolTipText("Image display");
                directory = fileChooser.getSelectedFile();
                listModel.setFiles(directory.listFiles(IMAGE_FILTER));
                revalidate();
                repaint();
        });
        mntmOpenFile.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O,
InputEvent.ALT_MASK));
        mntmOpenFile.setMnemonic(KeyEvent.VK_O);
        mnFile.add(mntmOpenFile);

        JMenuItem mntmExit = new JMenuItem("Exit");
        mntmExit.addActionListener((actionEvent) -> System.exit(0));
```

```
                mntmExit.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_X,
InputEvent.ALT_MASK));
                mntmExit.setMnemonic(KeyEvent.VK_X);
                mnFile.addSeparator();
                mnFile.add(mntmExit);

                JPanel contentPane = new JPanel(new BorderLayout(5, 5));
                scrllpnFileList.setMaximumSize(scrllpnFileList.getPreferredSize());
                scrllpnFileList.setMinimumSize(new Dimension(100, 100));
                imageDisplayPanel.setPreferredSize(new Dimension(CAM_WIDTH, CAM_HEIGHT));
                JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, scrllpnFileList,
tabbedPane);
                scrllpnFileList.addComponentListener(new ComponentAdapter() {
                        private int lastWidth;

                        @Override
                        public void componentResized(ComponentEvent ce) {
                                int width = scrllpnFileList.getWidth(), maxWidth =
scrllpnFileList.getMaximumSize().width;
                                if (width < maxWidth) {
                                        if (lastWidth >= maxWidth)
                                                splitPane.setResizeWeight(0.5);
                                }
                                else {
                                        if (lastWidth < maxWidth)
                                                splitPane.setResizeWeight(0.0);
                                        if (width > maxWidth)
        splitPane.setDividerLocation(scrllpnFileList.getMaximumSize().width+1);
                                }
                                lastWidth = width;
                        }
```

```
        });

        JLabel lblImageSize = new JLabel("Image size: ");
        tlbrStats.add(lblImageSize);

        JPanel zoomPanel = new JPanel();
        zoomPanel.setLayout(new BoxLayout(zoomPanel, BoxLayout.X_AXIS));
        zoomPanel.setBorder(new TitledBorder(null, "Zoom", TitledBorder.LEADING,
TitledBorder.TOP, null, null));
        tlbrTools.add(zoomPanel);
        JSlider slider = new JSlider(SwingConstants.HORIZONTAL, 16, 240, 128);
        zoomPanel.add(slider);
        slider.setToolTipText("Slide to zoom image");
        JSpinner spinner = new JSpinner(new SpinnerNumberModel(128, 16, 240, 1));
        @SuppressWarnings("serial")
        JSpinner.NumberEditor editor = new JSpinner.NumberEditor(spinner) {
            {
                NumberFormat displayFormat =
NumberFormat.getPercentInstance(),
                                editFormat = NumberFormat.getPercentInstance();
                displayFormat.setMinimumFractionDigits(5);
                editFormat.setMinimumFractionDigits(5);
                NumberFormatter displayFormatter = new
CustomNumberFormatter(displayFormat, true);
                NumberFormatter editFormatter = new
CustomNumberFormatter(editFormat, true);
                displayFormatter.setMinimum(0.125);
                displayFormatter.setMaximum(8.0);
                editFormatter.setMinimum(0.125);
                editFormatter.setMaximum(8.0);
                JFormattedTextField ftf = getTextField();
```

```
                        ftf.setFormatterFactory(new
DefaultFormatterFactory(displayFormatter,
                                        displayFormatter, editFormatter));
                        ftf.setColumns(8);
                }
        };
        spinner.setEditor(editor);
        zoomPanel.add(spinner);
        slider.addChangeListener((changeEvent) -> {
                int sliderValue = slider.getValue();
                double scale = slideToScale(sliderValue);
                imageDisplayPanel.setScale(scale);
                spinner.setValue(sliderValue);
        });
        spinner.addChangeListener((changeEvent) -> {
                int sliderValue = (Integer) spinner.getValue();
                if (sliderValue != slider.getValue())
                        slider.setValue(sliderValue);
        });
        fileList.addListSelectionListener((listSelectionEvent) -> {
                int sel = fileList.getSelectedIndex();

                if (sel != -1 && sel != selectedFileIndex) {
                        selectedFileIndex = sel;
                        try {
                                image =
ImageIO.read(listModel.getFiles()[selectedFileIndex]);
                        } catch (Exception e) {
                                e.printStackTrace();
                                JOptionPane.showMessageDialog(this, "Select Image Error: "
+ e, "Error", JOptionPane.ERROR_MESSAGE);
                        }
```

```
                if (image != null) {
                    imageDisplayPanel.setImage(image);
                    double scale = imageDisplayPanel.getFitScale();
                    slider.setValue(scaleToSlide(scale));
                    lblImageSize.setText("Image size: " +
imageDisplayPanel.getImageWidth() +
                            " x " + imageDisplayPanel.getImageHeight() +
" pixels");
                    imageDisplayPanel.setToolTipText("Image display: " +
listModel.getFiles()[selectedFileIndex].getName());
                    setEnabledComponents(filtSubTabbedPane, true);
                    setEnabledComponents(tlbrTools, true);
                    repaint();
                }
                else {
                    setEnabledComponents(filtSubTabbedPane, false);
                    setEnabledComponents(tlbrTools, false);
                }
            }
        });

        JPanel sharpenPanel = new JPanel();
        sharpenPanel.setLayout(new BoxLayout(sharpenPanel, BoxLayout.X_AXIS));
        sharpenPanel.setBorder(new TitledBorder(null, "Sharpen", TitledBorder.LEADING,
TitledBorder.TOP, null, null));
        tlbrTools.add(sharpenPanel);

        JButton originalButton = new JButton("Original");
        originalButton.addActionListener((actionEvent) -> {
            imageDisplayPanel.setImage(image);
            double scale = imageDisplayPanel.getFitScale();
            slider.setValue(scaleToSlide(scale));
```

```
                lblImageSize.setText("Image size: " + imageDisplayPanel.getImageWidth() +
                        " x " + imageDisplayPanel.getImageHeight() + " pixels");
                imageDisplayPanel.setToolTipText("Image display: " +
listModel.getFiles()[selectedFileIndex].getName());
                repaint();
            });
            sharpenPanel.add(originalButton);

            ImageFunctionButton.setListener((actionEvent) -> {
                ImageFunctionButton f = (ImageFunctionButton) actionEvent.getSource();
                imageDisplayPanel.setImage(f.apply(image));
                double scale = imageDisplayPanel.getFitScale();
                slider.setValue(scaleToSlide(scale));
                lblImageSize.setText("Image size: " + imageDisplayPanel.getImageWidth() +
                        " x " + imageDisplayPanel.getImageHeight() + " pixels | Time
execution: " + f.getLastTimeExecution() + " ns");
                imageDisplayPanel.setToolTipText("Image display: " +
listModel.getFiles()[selectedFileIndex].getName() +
                        " sharpened with " + f.getText());
                repaint();
            });
            List<ImageFunctionButton> functions = new ArrayList<ImageFunctionButton>();
            functions.add(new ImageFunctionButton("Laplacian (Spatial)", new ImageFunction(3)
{

                @Override
                public Mat apply(Mat image, int subProc) {
                    Mat result = new Mat();
                    long time = System.nanoTime();
                    Imgproc.filter2D(image, result, -1, LAPLACIAN_KERNEL);
                    if (subProc == 0) {
                        lastTimeExecution = System.nanoTime() - time;
```

```
                return result;
        }
//      Mat res = result.reshape(1);
//      MinMaxLocResult r = Core.minMaxLoc(res);
//      System.out.println(r.maxVal);
//      System.out.println(r.minVal);
//      System.out.println(r.maxLoc);
//      System.out.println(r.minLoc);
//      System.out.println(res.submat(1, 2, 3, 4).dump());
//      System.out.println(res.submat(0, 1, 0, 1).dump());
//      System.out.println(res.submat(3, 4, 3, 4).dump());
//      System.out.println(res.submat(0, 1, 9, 10).dump());
//      System.out.println();
        Core.normalize(result, result, -127, 127, Core.NORM_MINMAX);
//      res = result.reshape(1);
//      r = Core.minMaxLoc(res);
//      System.out.println(r.maxVal);
//      System.out.println(r.minVal);
//      System.out.println(r.maxLoc);
//      System.out.println(r.minLoc);
//      System.out.println(res.submat(1, 2, 3, 4).dump());
//      System.out.println(res.submat(0, 1, 0, 1).dump());
//      System.out.println(res.submat(3, 4, 3, 4).dump());
//      System.out.println(res.submat(0, 1, 9, 10).dump());
//      System.out.println();
        if (subProc == 1) {
                lastTimeExecution = System.nanoTime() - time;
                return result;
        }
        Core.subtract(image, result, result);
        lastTimeExecution = System.nanoTime() - time;
        return result;
```

```
        }
}));
functions.add(new ImageFunctionButton("Gradient", new ImageFunction(5) {

    @Override
    public Mat apply(Mat image, int subProc) {
        Mat[] results = new Mat[3];
        for (int i=0; i<results.length; i++) {
            results[i] = new Mat();
        }
        long time = System.nanoTime();
        Imgproc.filter2D(image, results[0], -1, GRADIENT_H_KERNEL);
        Core.absdiff(results[0], Scalar.all(0), results[0]);
        if (subProc == 0) {
            lastTimeExecution = System.nanoTime() - time;
            return results[0];
        }
        Imgproc.filter2D(image, results[1], -1, GRADIENT_V_KERNEL);
        Core.absdiff(results[1], Scalar.all(0), results[1]);
        if (subProc == 1) {
            lastTimeExecution = System.nanoTime() - time;
            return results[1];
        }
        Core.add(results[0], results[1], results[2]);
        if (subProc == 2) {
            lastTimeExecution = System.nanoTime() - time;
            return results[2];
        }
        Core.normalize(results[2], results[2], 0, 255, Core.NORM_MINMAX);
        if (subProc == 3) {
            lastTimeExecution = System.nanoTime() - time;
            return results[2];
```

```
                }
                Core.add(image, results[2], results[2]);
                lastTimeExecution = System.nanoTime() - time;
                return results[2];
            }
        }));
        functions.add(new ImageFunctionButton("Laplacian (Frequency)", new
ImageFunction(5) {

            @Override
            public Mat apply(Mat image, int subProc) {
                Mat[] results = new Mat[3];
                for (int i=0; i<results.length; i++) {
                    results[i] = new Mat();
                }
                long time = System.nanoTime();
                long timeStamp = frequencyTransformFilter(image, results[0],
results[1], results[2], subProc, new FTFunction(1) {

                    @Override
                    public double functionH(int x, int y) {
                        return -(x*x+y*y);
                    }
                });
                if (subProc >= 0 && subProc < 3) {
                    lastTimeExecution = timeStamp - time;
                    return results[subProc];
                }
//              Mat res = results[2].reshape(1);
//              MinMaxLocResult r = Core.minMaxLoc(res);
//              System.out.println(r.maxVal);
//              System.out.println(r.minVal);
```

```
//                      System.out.println(r.maxLoc);
//                      System.out.println(r.minLoc);
//                      System.out.println(res.submat(1, 2, 3, 4).dump());
//                      System.out.println(res.submat(0, 1, 0, 1).dump());
//                      System.out.println(res.submat(3, 4, 3, 4).dump());
//                      System.out.println(res.submat(0, 1, 9, 10).dump());
//                      System.out.println();
                        Core.normalize(results[2], results[2], -127, 127,
Core.NORM_MINMAX);
//                      res = results[2].reshape(1);
//                      r = Core.minMaxLoc(results[2].reshape(1));
//                      System.out.println(r.maxVal);
//                      System.out.println(r.minVal);
//                      System.out.println(r.maxLoc);
//                      System.out.println(r.minLoc);
//                      System.out.println(res.submat(1, 2, 3, 4).dump());
//                      System.out.println(res.submat(0, 1, 0, 1).dump());
//                      System.out.println(res.submat(3, 4, 3, 4).dump());
//                      System.out.println(res.submat(0, 1, 9, 10).dump());
//                      System.out.println();
                        if (subProc == 3) {
                                lastTimeExecution = System.nanoTime() - time;
                                return results[2];
                        }
                        Core.subtract(image, results[2], results[2]);
                        lastTimeExecution = System.nanoTime() - time;
                        return results[2];
                }
        }));
        functions.add(new ImageFunctionButton("Gaussian", new ImageFunction(3) {
                @Override
```

```
                public Mat apply(Mat image, int subProc) {
                        Mat[] results = new Mat[3];
                        for (int i=0; i<results.length; i++) {
                                results[i] = new Mat();
                        }
                        long time = System.nanoTime();
                        long timeStamp = frequencyTransformFilter(image, results[0],
results[1], results[2], subProc, new FTFunction(1) {

                                @Override
                                public double functionH(int x, int y) {
                                        return (1.5-Math.exp(-
Math.sqrt(x*x+y*y)*4000000));
                                }
                        });
                        lastTimeExecution = timeStamp - time;
                        return results[subProc >= 0 && subProc < 3 ? subProc : 2];
                }
        }));

        JPanel filtSubPanel = new JPanel();
        filtSubPanel.setLayout(new BoxLayout(filtSubPanel, BoxLayout.Y_AXIS));
        filtSubPanel.add(new ImageFunctionButton("Laplacian Edge",
functions.get(0).getFunction().getSubProcess(0)));
        filtSubPanel.add(new ImageFunctionButton("Normalization",
functions.get(0).getFunction().getSubProcess(1)));
        filtSubPanel.add(new ImageFunctionButton("Subtraction",
functions.get(0).getFunction()));
        filtSubTabbedPane.add(functions.get(0).getText(), filtSubPanel);
        sharpenPanel.add(functions.get(0));
        filtSubPanel = new JPanel();
        filtSubPanel.setLayout(new BoxLayout(filtSubPanel, BoxLayout.Y_AXIS));
```

```
                filtSubPanel.add(new ImageFunctionButton("Horizontal Edge",
functions.get(1).getFunction().getSubProcess(0)));
                filtSubPanel.add(new ImageFunctionButton("Vertical Edge",
functions.get(1).getFunction().getSubProcess(1)));
                filtSubPanel.add(new ImageFunctionButton("Gradient Edge",
functions.get(1).getFunction().getSubProcess(2)));
                filtSubPanel.add(new ImageFunctionButton("Normalization",
functions.get(1).getFunction().getSubProcess(3)));
                filtSubPanel.add(new ImageFunctionButton("Addition",
functions.get(1).getFunction()));
                filtSubTabbedPane.add(functions.get(1).getText(), filtSubPanel);
                sharpenPanel.add(functions.get(1));
                filtSubPanel = new JPanel();
                filtSubPanel.setLayout(new BoxLayout(filtSubPanel, BoxLayout.Y_AXIS));
                filtSubPanel.add(new ImageFunctionButton("Fourier Transform",
functions.get(2).getFunction().getSubProcess(0)));
                filtSubPanel.add(new ImageFunctionButton("Laplacian Function",
functions.get(2).getFunction().getSubProcess(1)));
                filtSubPanel.add(new ImageFunctionButton("Inverse FT (Laplacian Edge)",
functions.get(2).getFunction().getSubProcess(2)));
                filtSubPanel.add(new ImageFunctionButton("Normalization",
functions.get(2).getFunction().getSubProcess(3)));
                filtSubPanel.add(new ImageFunctionButton("Subtraction",
functions.get(2).getFunction()));
                filtSubTabbedPane.add(functions.get(2).getText(), filtSubPanel);
                sharpenPanel.add(functions.get(2));
                filtSubPanel = new JPanel();
                filtSubPanel.setLayout(new BoxLayout(filtSubPanel, BoxLayout.Y_AXIS));
                filtSubPanel.add(new ImageFunctionButton("Fourier Transform",
functions.get(3).getFunction().getSubProcess(0)));
                filtSubPanel.add(new ImageFunctionButton("Gaussian Function",
functions.get(3).getFunction().getSubProcess(1)));
```

```
            filtSubPanel.add(new ImageFunctionButton("Inverse FT",
functions.get(3).getFunction()));
            filtSubTabbedPane.add(functions.get(3).getText(), filtSubPanel);
            sharpenPanel.add(functions.get(3));

            JButton saveAllButton = new JButton("Save All");
            saveAllButton.addActionListener((actionEvent) -> {
                int response = JOptionPane.showConfirmDialog(this, "Save all filtered
images? " +
                        "The previously saved images with the same filename will be
overwritten.", "Save All", JOptionPane.YES_NO_OPTION);
                if (response == JOptionPane.YES_OPTION) {
                    String s = listModel.getFiles()[selectedFileIndex].getAbsolutePath();
                    try {
                        ImageIO.write(functions.get(0).apply(image), "JPG", new
File(s.substring(0, s.lastIndexOf('.')) + " Laplacian (Spatial).jpg"));
                        ImageIO.write(functions.get(1).apply(image), "JPG", new
File(s.substring(0, s.lastIndexOf('.')) + " Gradient.jpg"));
                        ImageIO.write(functions.get(2).apply(image), "JPG", new
File(s.substring(0, s.lastIndexOf('.')) + " Laplacian (Frequency).jpg"));
                        ImageIO.write(functions.get(3).apply(image), "JPG", new
File(s.substring(0, s.lastIndexOf('.')) + " Gaussian.jpg"));
                    } catch (IOException e) {
                        e.printStackTrace();
                        JOptionPane.showMessageDialog(this, "Write Image Error: "
+ e, "Error", JOptionPane.ERROR_MESSAGE);
                    }
                    listModel.setFiles(directory.listFiles(IMAGE_FILTER));
                    revalidate();
                    repaint();
                }
            });
```

```
        sharpenPanel.add(saveAllButton);
        setEnabledComponents(filtSubTabbedPane, false);
        setEnabledComponents(tlbrTools, false);

        JButton captureButton = new JButton("Capture");
        captureButton.addActionListener((actionEvent) -> {
            if (directory == null)
                JOptionPane.showMessageDialog(this, "Buka direktori sek bos! (File > Open Directory)");
            else {
                BufferedImage image = captureDisplayPanel.getImage();
                if (image != null) {
                    try {
                        ImageIO.write(captureDisplayPanel.getImage(), "JPG", new File(directory.getAbsolutePath() + "/capture " +
                                ZonedDateTime.now().format(DateTimeFormatter.ofPattern("dd MMM uuuu HH.mm.ss.SSS")) + ".jpg"));
                    } catch (IOException e) {
                        e.printStackTrace();
                        JOptionPane.showMessageDialog(this, "Capture Image Error: " + e, "Error", JOptionPane.ERROR_MESSAGE);
                    }
                    listModel.setFiles(directory.listFiles(IMAGE_FILTER));
                    revalidate();
                    repaint();
                }
                else
                    JOptionPane.showMessageDialog(this, "No Image Captured", "Error", JOptionPane.ERROR_MESSAGE);
            }
        });
```

```
            tlbrCaptTools.add(captureButton);

            JPanel panel = new JPanel(new BorderLayout());
            panel.add(splitPane, BorderLayout.CENTER);
            panel.add(tlbrStats, BorderLayout.SOUTH);
            contentPane.add(panel, BorderLayout.CENTER);
            setContentPane(contentPane);
    }

    private static void setEnabledComponents(Container container, boolean b) {
            for (Component c : container.getComponents()) {
                    c.setEnabled(b);
                    if (c instanceof Container)
                            setEnabledComponents((Container) c, b);
            }
    }

    private static void preProcFT(Mat frame) {
            for (int i = 0; i < frame.rows(); i++) {
                    for (int j = 0; j < frame.cols(); j++) {
                            double[] data = frame.get(i, j);
                            for (int k = 0; k < data.length; k++) {
                                    if ((i+j)%2==1)
                                            data[k] = -data[k];
                            }
                            frame.put(i, j, data);
                    }
            }
    }

    private static long frequencyTransformFilter(Mat frame, Mat dft, Mat idft, Mat result, int
subProc, ImageFunction function) {
```

```
long timeStamp;
Imgproc.copyMakeBorder(frame, result, 0, Core.getOptimalDFTSize(frame.rows()) -
frame.rows(),
                0, Core.getOptimalDFTSize(frame.cols()) - frame.cols(),
Imgproc.BORDER_CONSTANT, Scalar.all(0));
preProcFT(result);
List<Mat> matList = new ArrayList<Mat>(), matList2 = new ArrayList<Mat>();
Core.split(result, matList);
matList2.add(matList.get(0));
matList2.add(Mat.zeros(result.size(), CvType.CV_64F));
Core.merge(matList2, matList.get(0));
Core.dft(matList.get(0), matList.get(0),Core.DFT_SCALE,0);
matList2.remove(0);
matList2.add(0, matList.get(1));
Core.merge(matList2, matList.get(1));
Core.dft(matList.get(1), matList.get(1),Core.DFT_SCALE,0);
matList2.remove(0);
matList2.add(0, matList.get(2));
Core.merge(matList2, matList.get(2));
Core.dft(matList.get(2), matList.get(2),Core.DFT_SCALE,0);
createOptimizedMagnitude(matList).copyTo(dft);
timeStamp = System.nanoTime();
function.apply(matList.get(0));
function.apply(matList.get(1));
function.apply(matList.get(2));
createOptimizedMagnitude(matList).copyTo(idft);
if (subProc == 1)
        timeStamp = System.nanoTime();
Core.idft(matList.get(0), matList.get(0));
Core.split(matList.remove(0), matList2);
matList.add(0, matList2.get(0));
Core.idft(matList.get(1), matList.get(1));
```

```
                Core.split(matList.remove(1), matList2);
                matList.add(1, matList2.get(0));
                Core.idft(matList.get(2), matList.get(2));
                Core.split(matList.remove(2), matList2);
                matList.add(2, matList2.get(0));
                Core.merge(matList, result);
                preProcFT(result);
                result.submat(0, frame.rows(), 0, frame.cols()).copyTo(result);
                if (subProc != 0 && subProc != 1)
                        timeStamp = System.nanoTime();
                return timeStamp;
        }

        private static Mat createOptimizedMagnitude(List<Mat> complexImage)
        {
                List<Mat> newPlanes = new ArrayList<>();
                List<Mat> matList = new ArrayList<>();
                matList.add(new Mat());
                matList.add(new Mat());
                matList.add(new Mat());
                Mat mag = new Mat();
                for (int i = 0; i < 3; i++) {
                        Core.split(complexImage.get(i), newPlanes);
                        Core.magnitude(newPlanes.get(0), newPlanes.get(1), matList.get(i));
                        Core.add(Mat.ones(matList.get(i).size(),
CvType.CV_64FC(matList.get(i).channels())), matList.get(i), matList.get(i));
                        Core.log(matList.get(i), matList.get(i));
                }
                Core.merge(matList, mag);
                Core.normalize(mag, mag, 0, 255, Core.NORM_MINMAX);
                return mag;
        }
```

```
    private static BufferedImage matToBufferedImage(Mat frame) {
        int type = BufferedImage.TYPE_CUSTOM;
        if (frame.channels() == 1) {
            type = BufferedImage.TYPE_BYTE_GRAY;
        } else if (frame.channels() == 3) {
            type = BufferedImage.TYPE_3BYTE_BGR;
        }
        BufferedImage image = new BufferedImage(frame.width(), frame.height(), type);
        byte[] data = ((DataBufferByte) image.getRaster().getDataBuffer()).getData();
        Mat frame8U = new Mat();
        frame.convertTo(frame8U, CvType.CV_8UC(frame.channels()));
        frame8U.get(0, 0, data);
        return image;
    }

    private static Mat bufferedImageToMat(BufferedImage image) {
        byte[] data = ((DataBufferByte) image.getRaster().getDataBuffer()).getData();
        int channels = image.getType() == BufferedImage.TYPE_3BYTE_BGR ? 3 : 1;
        Mat frame = new Mat(image.getHeight(), image.getWidth(),
CvType.CV_8UC(channels));
        frame.put(0, 0, data);
        frame.convertTo(frame, CvType.CV_64FC(channels));
        if (channels == 1) {
            List<Mat> matList = new ArrayList<Mat>();
            matList.add(frame.clone());
            matList.add(frame.clone());
            matList.add(frame.clone());
            Core.merge(matList, frame);
        }
        return frame;
    }
```

```
    private static int scaleToSlide(double scale) {
        return Double.compare(scale, 1.0) < 0 ? (int) (scale*128.0) : (int) (scale*16.0)+112;
    }

    private static double slideToScale(int sliderValue) {
        return sliderValue < 128 ? sliderValue/128.0 : (sliderValue-112)/16.0;
    }
}
```