Identifikasi Parameter Optimal dari Metode Fuzzy Co-Clustering dan Estimasi Robust Spasial pada Segmentasi Citra dengan *Noise*

Nama Mahasiswa	: Hardika Khusnuliawati
NRP	: 5110100206
Jurusan	: Teknik Informatika FTIf-ITS
Dosen Pembimbing 1	: Arya Yudhi Wijaya, S.Kom, M.Kom.
Dosen Pembimbing 2	: Rully Soelaiman, S.Kom, M.Kom.

ABSTRAK

Segmentasi citra merupakan salah satu kunci penting untuk proses analisis citra dan pengembangan berbagai aplikasi yang melibatkan pengolahan citra. Untuk melakukan segmentasi citra, telah banyak metode yang digunakan. Metode-metode tersebut antara lain segmentasi dengan histogram, segmentasi dengan deteksi tepi, segmentasi dengan Fuzzy C-Means (FCM) dan metode-metode lainnya. Adanya noise pada citra merupakan salah satu permasalahan yang sering ditemui dalam segmentasi citra. Sehingga diperlukan metode segmentasi citra yang mampu mengatasi pemasalahan tersebut.

Pada Tugas Akhir ini, metode yang akan diimplentasikan dalam proses segmentasi citra berwarna adalah pengembangan algoritma Fuzzy C-Means yang disebut Fuzzy Co-Clustering For Images (FCCI) dan estimasi robust spasial untuk meningkatkan kemampuan segmentasi pada kasus citra berwarna dengan noise. Estimasi Robust Spasial merupakan metode yang digunakan untuk mengumpulkan informasi spasial dari citra sehingga dapat menyaring data dari noise dengan pendekatan statistik. Dengan estimasi robust spasial yang ditambahkan pada algoritma segmentasi citra yang digunakan diharapkan mampu memberikan hasil segmentasi yang baik terhadap inputan citra dengan noise.

Dari hasil uji coba, metode ini memiliki nilai evaluasi kuantitatif yang kecil dan nilai PSNR yang tinggi untuk masukan citra berwarna dengan noise dengan rata-rata error color kurang dari 1.15 %.

Kata kunci: Fuzzy Co-Clustering, estimasi robust spasial, penyaringan noise, segmentasi citra berwarna.

Optimal Parameter Identification Method Of Fuzzy Co-Clustering And Estimates On Robust Spatial Image Segmentation With *Noise*

Student Name	: Hardika Khusnuliawati
Student ID	: 5110100206
Major	: Teknik Informatika FTIf-ITS
Advisor 1	: Arya Yudhi Wijaya, S.Kom, M.Kom.
Advisor 2	: Rully Soelaiman, S.Kom, M.Kom.

ABSTRACT

Image segmentation is one important key to the process of image analysis and development of a wide range of applications involving image processing. To perform image segmentation, has been widely used methods. These methods include the histogram segmentation, segmentation with edge detection, segmentation with Fuzzy C-Means (FCM) and other methods. The presence of noise in the image is one of the problems frequently encountered in image segmentation. So, we need a method of image segmentation that can overcome these problems.

In this final project, the method is implemented in a process that will color image segmentation is the development of Fuzzy C-Means algorithm called Fuzzy Co-Clustering For Images (FCCI) and robust estimation to improve the ability of spatial segmentation in the case of color images with noise. Robust Estimation of Spatial is the method used to collect spatial information of the image so that it can filter out noise in the data from a statistical approach. With a robust estimate of the added spatial image segmentation algorithm that is used is expected to provide good segmentation results of the input image with noise.

Based on the result of experiments, this method has little quantitative evaluation value and high PSNR value for input color image with noise, this method has average error color less than 1,15 %.

Keywords: Fuzzy Co-Clustering, spatial robust estimation, filtering noise, color image segmentation.

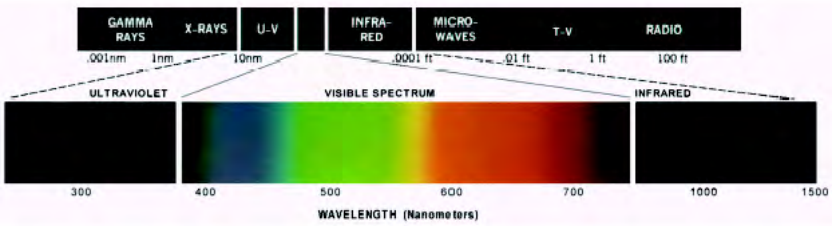
DAFTAR KODE SUMBER

BAB II
TINJAUAN PUSTAKA

Pada bab ini akan dibahas mengenai teori-teori yang menjadi dasar dari pembuatan tugas akhir ini. Pembahasan ini bertujuan untuk memberikan gambaran secara umum terhadap sistem yang dibuat dan berguna sebagai penunjang dalam pengembangan tugas akhir. Teori-teori yang dibahas meliputi citra berwarna, *noise* pada citra, metode *robust* spasial, segmentasi citra, dan *Fuzzy Clustering*.

2.1. Citra Berwarna

Suatu warna pada obyek merupakan hasil pantulan cahaya dari objek tersebut. Cahaya yang terpantul tersebut dapat bersifat achromatic atau chromatic. Cahaya achromatic hanya memiliki atribut intensitas atau nilai yang apabila dilihat oleh mata manusia merepresentasikan warna hitam atau putih. Sedangkan cahaya chromatic merupakan spektrum elektromagnetik yang berada pada kisaran panjang gelombang 400 nm hingga 700 nm. Suatu warna yang dilihat mata manusia menunjukkan panjang gelombang dari cahaya chromatic tersebut. Gambar 2.1 menunjukkan warna dengan kisaran panjang gelombang yang dimiliki. Berdasarkan karakteristik mata manusia, suatu warna yang dilihat merupakan kombinasi dari warna primer Red(R), Green(G), dan Blue(B). Akan tetapi warna primer tersebut tidak menjadi suatu keharusan untuk memodelkan suatu citra berwarna ke bentuk RGB model. Suatu citra berwarna dapat dimodelkan ke dalam beebrapa bentuk model warna sebagai contoh RGB model, CMY model, CIELAB model, dan HSI model.



Gambar 2.1 Spektrum elektromagnet beserta panjang gelombang

Pada citra berwarna, setiap citra merupakan kumpulan dari piksel-piksel dimana setiap piksel tersebut menyimpan suatu nilai tertentu. Untuk melakukan suatu proses pengolahan tertentu pada suatu citra berwarna, maka piksel pada setiap komponen model warna yang digunakan dapat diproses secara terpisah atau dapat diproses secara bersamaan.

2.1.1. Citra Berwarna dengan *Noise*

Noise pada citra merupakan informasi tambahan yang tidak diinginkan dan dapat menyebabkan kualitas dari citra mengalami penurunan. Adanya *noise* pada citra dapat disebabkan dari berbagai sumber. Suatu citra merupakan hasil konversi dari beberapa tahap yaitu dari sinyal optik ke sinyal elektrik kemudian menjadi sinyal digital. Pada setiap tahap tersebut terdapat kemungkinan terjadi fluktuasi data yang disebabkan kejadian alami dan penambahan suatu nilai random yang menyebabkan perubahan piksel pada citra. *Noise* pada citra dapat dimodelkan dalam bentuk histogram atau *probability density function*. *Noise* yang dimodelkan dalam bentuk *probability density function* mengganggu bentuk *probability density function* dari citra sebenarnya. Jika suatu piksel citra $s(i,j)$ mengalami degradasi akibat *noise* $n(i,j)$, maka hasil piksel citra yang terdegradasi $f(i,j)$ dimodelkan seperti pada Persamaan 2.1.

$$f(i,j) = s(i,j) + n(i,j) \tag{2.1}$$

2.1.1.1. Gaussian *Noise*

Gaussian *noise* memiliki *probability density function* yang digambarkan dalam bentuk distribusi normal. Besar intensitas dari gaussian *noise* ditentukan dari nilai σ menyatakan standar deviasi dan μ menyatakan nilai rata-rata. Pada citra, gaussian *noise* ditunjukkan dengan titik-titik kecil yang menyebar ke seluruh citra. Gambar 2.2 menunjukkan citra yang terdegradasi gaussian *noise* dengan $\sigma^2 = 0.01$ dan $\mu = 0$.

Gambar 2.2 Citra yang terdegradasi gaussian *noise* dengan nilai $\boldsymbol{\sigma^2 = 0.01}$ dan $\boldsymbol{\mu = 0}$

2.1.1.2. Salt and Pepper *Noise*

Salt and Pepper *noise* hanya memiliki dua kemungkinan nilai untuk *noise* z yaitu a dan b dimana secara analitik nilai a dan b dengan tingkat keabuan g dapat dimodelkan sebagai Persamaan 2.2.

$$p_{salt\ and\ pepper}(z) = \begin{cases} A, & for\ g = a('salt') \\ B, for\ g = b('pepper') \end{cases} \quad (2.2)$$

dimana peluang kemunculan dari nilai Salt and Pepper *noise* pada citra kurang dari sama dengan 0.1. Apabila peluang kemunculan lebih dari itu, maka *noise* akan secara dominan merusak citra asli. Untuk citra dengan ukuran 8 bit per piksel, intensitas nilai untuk papper *noise* adalah mendekati 0 dan nilai untuk salt *noise* mendekati 255. Salt and Papper *noise* biasa disebabkan oleh kesalahan teknis sebagai contoh kesalahan fungsi kamera. Gambar 2.3 menunjukkan citra yang terdegradasi salt and pepper *noise* dengan intensitas *noise* 0.05.

Gambar 2.3 Citra yang terdegradasi salt and pepper *noise* dengan nilai intensitas 0.05

2.1.1.3. Speckle *Noise*

Speckle *noise* biasa muncul pada citra radar seperti *synthetic aperture radar* (SAR) dan menurunkan kualitas dari citra sehingga menimbulkan kesulitan untuk analisis dan interpretasi citra. Speckle *noise* melibatkan *multiplicative component* sehingga membentuk *multiplicative noise*. Pemodelan dari speckle *noise* ditunjukkan pada Persamaan 2.3.

$$g(n,m) = f(n,m) * u(n,m) + \eta(n,m) \tag{2.3}$$

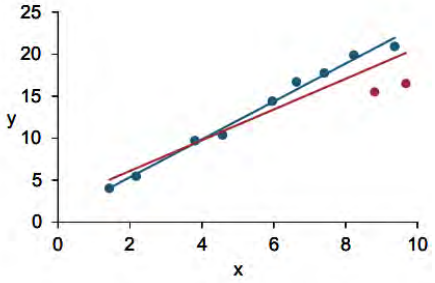
dimana $g(n,m)$ merupakan piksel citra dengan *noise*, $f(n,m)$ merupakan piksel citra asli, dan $u(n,m)$ adalah *multiplicative component*. Sedangkan $\eta(n,m)$ merupakan *noise* tambahan yang dapat diikutsertakan maupun diabaikan sesuai kasus citra yang sedang diobservasi. Gambar 2.4 menunjukkan citra yang terdegradasi speckle *noise* dengan nilai varian 0.04.

Gambar 2.4 Citra yang terdegradasi speckle *noise* dengan nilai varian 0.04

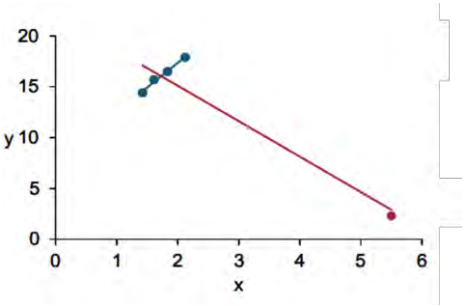
2.2. Penyaringan *Noise* dengan Metode Robust

Suatu metode dikatakan *robust* atau memiliki ketahanan jika dapat mentoleransi *outliers*. Sebagai contoh tidak adanya data yang menyalahi bentuk model data yang diasumsikan. Pengertian ini memiliki kesamaan dengan salah satu kriteria ketahanan (*robustness*) pada bidang statistik yaitu statistika *robust*. Statistika *robust* berhubungan dengan deviasi atau perilaku data yang sesuai dengan asumsi statistik yang dibuat. Suatu *outliers* pada sekumpulan data dapat diilustrasikan seperti pada Gambar 2.5 yaitu sebagai penyebab

fitting error (ditunjukkan dengan garis warna merah) atau suatu *single outlier* yang dapat menyebabkan kesalahan yang serius (ditunjukkan dengan titik warna merah) seperti pada Gambar 2.6 [5].



Gambar 2.5 *Outliers* yang menyebabkan *fitting error*



Gambar 2.6 *Single Outliers* pada sekumpulan data

Sehingga untuk mengatasi *outliers* tersebut, suatu metode *robust* dibentuk dengan dua pendekatan yaitu suatu fungsi yang tahan terhadap *outliers* dan metode untuk mendeteksi serta menghapus *outliers* [6].

2.2.1. R-Filter

R-Filter merupakan metode penyaringan *noise* yang dibentuk dari R-Estimator. R-Estimator adalah salah satu robust estimator yang penghitungannya didasarkan pada urutan data $x_{(i)}$ dimana $x_{(i)}$

merupakan elemen data pada urutan ke-*i* dan *n* menyatakan ukuran himpunan data dengan $1 \leq i \leq n$. Median estimator merupakan salah satu bentukan R-Estimator yang baik digunakan ketika informasi apriori dari himpunan data membentuk suatu distribusi tertentu dan momen-momen dari data tidak diketahui. Bentuk persamaan dari median estimator dapat dimodelkan dalam Persamaan 2.4.

$$\hat{\theta}_{med} = \begin{cases} \frac{1}{2}\left(X_{(n/2)} + X_{(1+n/2)}\right), & for\ even\ n \\ X_{(n+1/2)}, & for\ odd\ n \end{cases} \tag{2.4}$$

dimana jika *n* merupakan bilangan genap maka nilai median estimator merupakan setengah dari penjumlahan elemen pada urutan $n/2$ dan elemen pada urutan $1 + n/2$. Sedangkan untuk *n* merupakan bilangan ganjil maka nilai median estimator merupakan elemen pada urutan $n + 1/2$.

2.2.2. M-Filter

M-Filter merupakan metode penyaringan *noise* yang dibentuk dari M-estimator dimana M-estimator adalah jenis robust estimator yang merupakan hasil generalisasi dari *maximum likelihood estimator*. M-estimator memperbaiki kinerja dari *maximum likelihood estimator* dan *least square estimator* dengan menggunakan fungsi residu yang lain. Fungsi residu dari M-Estimator untuk lokasi parameter θ dapat dihitung dari hasil minimum turunan pertama fungsi $\sum_{i=1}^{N} \rho(X_i - \theta)$ yaitu ketika $\sum_{i=1}^{N} \psi(X_i - \theta) = 0$ dimana $\psi(X, \theta) = \frac{\partial}{\partial \theta}\rho(X, \theta)$ (turunan pertama). Bentuk akhir dari M-Estimator bergantung pada bentuk fungsi dari $\rho(.)$ yang kemudian menghasilkan fungsi $\psi(.)$ atau disebut juga sebagai *influence function*. Beberapa macam *influence function* yang dapat digunakan untuk M-Filter dapat dilihat pada Tabel 2.1.

Tabel 2.1 Daftar *influence function*

Influence Function	**Fungsi pembobotan**
Andrew's sine	$\psi_{sin(r)}(X) = \begin{cases} \sin(x/r)/(x/r), \lvert x\rvert \le r\pi \\ 0, otehrwise \end{cases}$
Tukey's biweight	$\psi_{bi(r)}(X) = \begin{cases} [1-(x/r)^2]^2, \lvert x\rvert \le r \\ 0, otherwise \end{cases}$
Hampel's three part redescending	$\psi_{\propto,\beta,r}(X) = \begin{cases} x, 0 \le \lvert x\rvert \le \propto \\ \propto . sgn(x), \propto \le \lvert x\rvert \le \beta \\ \frac{\alpha}{\lvert x\rvert}\frac{r-\lvert x\rvert}{r-\beta}, \beta \le \lvert x\rvert \le r \\ 0, r \le \lvert x\rvert \end{cases}$

2.2.3. L-Filter

L-Filter merupakan metode penyaringan *noise* yang dibentuk dari L-Estimator atau biasa disebut pula estimator *order statistic*. L-estimator tersebut merupakan kombinasi linier dari sekumpulan data yang telah terurut. Sekumpulan data tersebut biasa terurut secara menaik. Bentuk persamaan dari definisi L-Estimator ditunjukkan pada Persamaan 2.5.

$$T_n = \sum_{i=1}^{n} a_i x_{(i)} \tag{2.5}$$

dimana $x_{(i)}$ menyatakan data-data dari sekumpulan data yang telah terurut dengan data $x_{(i)}$ merupakan data dengan urutan ke-*i*. Untuk a_i merupakan vektor yang menyatakan besar pembobotan terhadap data $x_{(i)}$ dimana jumlah dari vektor a_i adalah 1 ($\sum_{i=1}^{n} a_i = 1$).

Nilai pembobotan a_i dapat berbeda untuk setiap jenis kasus. Hal tersebut disesuaikan dengan perilaku data yang diamati.

Beberapa jenis distribusi dapat digunakan untuk membangkitkan nilai pembobotan a_i antara lain *exponential*, *laplacian*, dan *uniform.* Bentuk persamaan dari nilai pembobotan a_i dapat dilihat pada Persamaan 2.6.

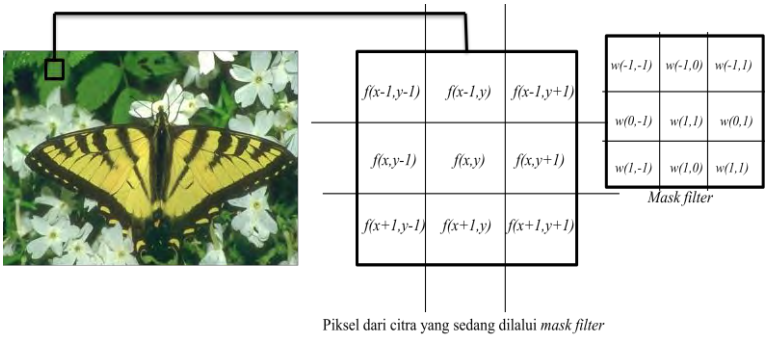
$$a_i = \int_{\frac{i-1}{n}}^{\frac{i}{n}} h(\lambda)d\lambda \Big/ \int_0^1 h(\lambda)d\lambda \tag{2.6}$$

dimana $h(\lambda)$ merupakan *noise distribution function* dari bilangan random $h(\lambda): [0,1] \rightarrow R$ yang dibangkitkan dari jenis distribusi tertentu dan n menyatakan banyak bilangan data yang akan dibangkitkan.

2.2.4. RML-Filter

RML-Filter merupakan metode penyaringan *noise* yang dibentuk dari estimasi robust hasil kombinasi dari jenis estimasi dasar yaitu R-estimator, M-Estimator, dan L-estimator. Desain dari RML-Filter dilakukan pada domain spasial. Hal ini didasarkan pada konsep RML-Filter yang merupakan kombinasi dari filter-filter nonlinear dimana desain filter hanya dapat dilakukan pada domain spasial. Mekanisme dalam merancang filter pada domain spasial perlu memperhatikan aspek ketetanggaan dari kumpulan piksel-piksel yang biasa direpresentasikan dalam blok bujur sangkar dan fungsi operasi yang mengkaitkan hubungan antara piksel-piksel yang saling bertetangga tersebut. Hasil dari suatu *filter* adalah nilai piksel baru dimana koordinat piksel baru tersebut sama dengan koordinat piksel tengah dari blok bujur sangkar yang dibentuk dari piksel-piksel yang saling bertetangga. Gambar 2.7 menunjukkan ilustrasi *linier filtering* citra dengan *filtering window* berukuran 3x3.

Bentuk persamaan dari RML-Filter yang merupakan kombinasi dari R-Filter, M-Filter dan L-Filter dimodelkan dalam bentuk Persamaan 2.7.



Gambar 2.7 Citra dengan penyaringan *noise* menggunakan mask filter 3x3

$$\theta_{RML} = \frac{med\{a_i(X_i\tilde{\psi}(X_i - med\{\bar{X}\}))\}}{a_{med}} \tag{2.7}$$

dimana $med\{.\}$ menyatakan fungsi median dari vektor data X yang diperoleh sesuai Persamaan 2.4. Sedangkan $\psi(X)$ menyatakan hasil dari *influence function* yang mewakili M-Estimator dengan menggunakan salah satu persamaan pada Tabel 2.1. a_i menyatakan vektor dari nilai koefisien pembobotan yang diperoleh dari metode L-Filter pada Persamaan 2.6. Sedangkan a_{med} menyatakan nilai tengah dari vektor koefisien pembobotan a_i tersebut.

2.2.5. Enhanced Rank Impulse Detector (ERID)

ERID merupakan metode untuk meningkatkan kemampuan RML-Filter dalam menyaring *noise*. Metode ERID didasarkan pada posisi *noise* dan posisi median pada vektor piksel dari citra. Posisi median pada vektor piksel dari citra selalu berposisi di tengah sedangkan posisi *noise* biasanya akan berada di ujung-ujung vektor. Apakah mendekati posisi pertama atau posisi terakhir. Sehingga bentuk dasar dalam membentuk ERID ditunjukkan pada Persamaan 2.8.

$$[(rank(y_c) \leq s) \vee (rank(y_c) \geq N - s)] \quad (2.8)$$

Persamaan 2.8 didasarkan pada perbandingan antara urutan piksel dari citra yang diuji dengan ambang batas yang ditetapkan sebelumnya. Notasi y_c menyatakan piksel pada pusat filtering window, s merupakan ambang batas dengan nilai $s > 0$, dan N merupakan panjang vektor piksel pada filtering window. Terdapat dua kondisi dari Persamaan 2.8 untuk mengaktegorikan bahwa y_c dirusak oleh *noise*. Jika salah satu kondisi tepenuhi maka y_c dapat dikategorikan terusak *noise*.

Terdapat kelemahan pada Persamaan 2.8 yaitu kemungkian terjadinya kesalahan dalam mendeteksi piksel citra yang terdegradasi *noise*. Suatu piksel citra yang memenuhi kondisi pertama dapat dideteksi terdegradasi *noise* tetapi terdapat kemungkinan bahwa piksel tersebut tidak terdegradasi *noise*, sehingga perlu kondisi tambahan pada Persamaan 2.8 untuk mengatasi kelemahan tersebut. Selain memperhatikan urutan piksel pada vektor citra, juga perlu diperhatikan nilai kecerahan dari piksel dengan memperhitungkan besar jarak y_c terhadap median dari vector piksel citra pada filtering window. Adanya penambahan kondisi tersebut, maka persamaan akhir untuk ERID dapat dituliskan dengan Persamaan 2.9 sebagai berikut.

$$[(rank(y_c) \leq s) \vee (rank(y_c) \geq N - s)] \wedge (|y_c - med(y_N)| \geq T) \quad (2.9)$$

dimana T merupakan ambang batas dengan nilai $T \geq 0$.

2.3. Segmentasi Citra

Segmentasi merupakan suatu proses pembagian citra ke dalam beberapa segmen bagian dimana setiap segmen terdiri dari piksel-piksel. Segmen-segmen yang terbagi tersebut juga biasa disebut superpiksel. Setiap piksel yang berada dalam satu bagian segmen yang sama memiliki kesamaan karakteristik seperti warna, intensitas, atau tekstur. Hasil dari segmentasi citra bertujuan merepresentasikan citra ke dalam bentuk yang lebih memiliki arti

sehingga memudahkan proses analisis. Beberapa metode segmentasi citra telah diusulkan antara lain *threshold based segmentation, edge based segmentation, region based segmentation, clustering*, dan *matching*.

2.4. Fuzzy *Cluster*ing

Clustering merupakan suatu proses pembagian titik data ke dalam suatu kelas homogen atau *cluster* sehingga setiap anggota dari suatu *cluster* memiliki kesamaan karakteristik dan setiap anggota dari *cluster* berbeda memilki perbedaan karakteristik. *Cluster*ing juga dapat dianggap sebagai bentuk kompresi data dimana sejumlah data yang banyak diubah menjadi sejumlah kecil kelompok atau *cluster*. Beberapa cara digunakan untuk mengukur tingkat kesamaan dari setiap titik data yang akan dikelompokkan. Tingkat kesamaan tersebut yang nantinya akan membentuk suatu *cluster*. Contoh dari tingkat kesamaan yang dapat digunakan sebagai ukuran antara lain jarak, konektivitas, dan intensitas.

Sistem keanggotaan dari suatu algoritma *clustering* dapat dicontohkan sebagai berikut yaitu ketika terdapat sekumpulan data $X = \{x_1, \dots, x_n\}$ dengan jumlah data n maka pembagian sekumpulan data X ke dalam c kluster dapat dibagi menjadi *hard partition matrics* dimana nilai derajat keanggotaan dari data x_k untuk $k = 1,2, \dots, n$ bernilai 1 jika x_k merupakan anggota suatu kluster dan bernilai 0 jika x_k bukan anggota suatu kluster. Selain dengan metode *hard partition matrics*, metode lain untuk menentukan keanggotaan suatu kluster yaitu dibagi menjadi *soft partition matrics* dimana nilai keanggotaan dari data x_k berada pada rentang [0,1].

2.5. *Fuzzy Co-Clustering for Image (FCCI)*

FCCI merupakan salah satu algoritma *clustering* piksel yang sering digunakan dalam segmentasi citra dimana data yang dikelompokkan adalah nilai piksel dari citra. Algoritma ini merupakan pengembangan dari algoritma *Fuzzy C-Means* dengan menggabungkan jarak antara setiap fitur titik data dan fitur pusat *cluster* sebagai ukuran ketidakmiripan (*dissimilarity*) dan *entropy*

dari objek dan fitur sebagai kondisi regularisasi dalam fungsi objektif (*objective function*). Algoritma Fuzzy C-Means yang merupakan dasar pembentukan FCCI memiliki fungsi objektif berbasis *entropy* yang ditunjukkan pada Persamaan 2.10.

$$J_{FCM} = \sum_{k=1}^{n}\sum_{i=1}^{c} u_{ik}(d_{ik})^2 + T_u \sum_{k=1}^{n}\sum_{i=1}^{c} u_{ik} \log u_{ik} \tag{2.10}$$

dimana

a. J_{FCM} = fungsi optimum dari algoritma Fuzzy C-Means.
b. (u_{ik}) = derajat keanggotaan dari piksel ke-*k* terhadap *cluster* ke-*i*.
c. n = jumlah data piksel yang akan dikelompokkan.
d. c = jumlah *cluster* yang diuji.
e. (d_{ik}) = jarak antara piksel ke-*k* terhadap pusat *cluster* ke-*i* yang dijelaskan pada Persamaan 2.11.

$$(d_{ik})^2 = ||x_k - c_i||^2 \tag{2.11}$$

dimana c_i merupakan piksel pusat kluster ke-*i*.

Untuk memperbarui pusat kluster dan fungsi keanggotaan dari algoritma Fuzzy C-Means berbasis *entropy*, secara berturut-turut ditunjukkan pada Persamaan 2.12 dan Persamaan 2.13.

$$c_i = \frac{\sum_{k=1}^{n} u_{ik} x_k}{\sum_{k=1}^{n} u_{ik}} \tag{2.12}$$

$$u_{ik} = \frac{e(-\alpha \log(d_{ik})^2)}{\sum_{j=1}^{c} e\,(-\alpha \log(d_{ik})^2)} \tag{2.13}$$

Sedangkan fungsi objektif pada FCCI, jarak antara setiap fitur piksel data dengan fitur pusat *cluster* juga diperhitungkan, sehingga

fungsi objektif dari FCCI yang merupakan pengembangan FCM dapat dituliskan pada Persamaan 2.14.

$$J_{FCCI}(U,V,P) = \sum_{c=1}^{C}\sum_{i=1}^{N}\sum_{j=1}^{K} u_{ci}v_{cj}d_{cij} + T_U \sum_{c=1}^{C}\sum_{i=1}^{N} u_{ci} log u_{ci} + T_v \sum_{c=1}^{C}\sum_{j=1}^{K} v_{cj} log v_{cj} \quad (2.14)$$

Penentuan fungsi jarak pada FCCI juga mengalami perubahan dari penentuan fungsi jarak pada FCM. Jika pada FCM hanya mmperhitungkan jarak piksel dengan pusat *cluster*, maka fungsi jarak pada FCCI juga memperhitungkan jarak piksel data x_{ij} dengan fitur pusat *cluster* p_{cj}. Jarak antara piksel data x_{ij} dengan fitur pusat *cluster* p_{cj} dihitung untuk setiap fitur *j=1,2,..,K* secara terpisah dimana *K* merupakan dimensi dari fitur yang berasosiasi dengan setiap titik data. Bentuk persamaan d_{cij} ditunjukkan pada Persamaan 2.15.

$$d_{cij} = d^2(x_{ij}, p_{cj}) = (x_{ij} - p_{cj})^2 \quad (2.15)$$

Pada algoritma FCCI, digunakan dua fungsi keanggotaan yaitu fungsi keanggotaan objek (*object membership function*) dan fungsi keanggotaan fitur (*feature membership function*). Adanya fungsi keanggotaan fitur ini diharapkan mampu meminimalisasi masalah *outliers*. Untuk memperoleh fungsi keanggotaan objek u_{ci} dan fungsi keanggotaan fitur v_{cj} secara berturut-turut ditunjukkan pada Persamaan 2.15 dan Persamaan 2.16. Sedangkan untuk memperbarui pusat *cluster*, algoritma FCCI tetap menggunakan Persamaan 2.12.

$$u_{ci} = \frac{e^{\left(-\sum_{j=1}^{K}\frac{v_{cj}D_{cij}}{T_u}\right)}}{\sum_{c=1}^{C} e^{\left(-\sum_{j=1}^{K}\frac{v_{cj}D_{cij}}{T_u}\right)}} \quad (2.16)$$

$$v_{cj} = \frac{e\left(-\sum_{i=1}^{N} \frac{u_{ci}\, D_{cij}}{T_V}\right)}{\sum_{j=1}^{K} e\left(-\sum_{i=1}^{N} \frac{u_{ci}\, D_{cij}}{T_V}\right)} \quad (2.17)$$

2.6. Evaluasi Kualitas *Filtering*

Evaluasi proses filtering pada citra dengan *noise* dapat dilakukan dengan beberapa cara dimana salah satunya yaitu menggunakan pengukuran *Peak Signal to Noise Ratio* (PSNR). PSNR merupakan parameter untuk menilai kualitas suatu citra dengan membandingkan *noise* terhadap sinyal puncak . Satuan yang biasa digunakan untuk ukuran PSNR adalah *decibel* (db). Bentuk persamaan matematika dari PSNR ditunjukkan pada Persamaan 2.18.

$$PSNR = 20 \log_{10}\left(\frac{MAX_f}{\sqrt{MSE}}\right) \quad (2.18)$$

untuk nilai MSE diperoleh dari Persamaan 2.19 sebagai berikut.

$$MSE = \frac{1}{mn} \sum_{0}^{m-1} \sum_{0}^{n-1} |f(i,j) - g(i,j)|^2 \quad (2.19)$$

dimana

a. f = matrik piksel data dari citra asli
b. g = matrik piksel data dari citra yang terdegradasi *noise*
c. m = jumlah baris dari matrik citra dimana *i* merupakan baris dari citra yang diuji saat itu
d. n = jumlah kolom dari matrik citra dimana *j* merupakan kolom dari citra yang diuji saat itu
e. MAX_f= nilai maksimum dari piksel citra asli dan piksel citra yang terdegradasi *noise*

Suatu citra yang diukur nilai PSNR dari citra tersebut dikatakan memiliki kualitas baik jika nilai MSE yang diperoleh kecil. Nilai MSE merupakan error dari piksel citra yang diukur. Sehingga jika

nilai PSNR yang diperoleh semakin besar maka kualitas citra asli yang sedang diukur semakin baik.

2.7. Evaluasi Kualitas *Cluster*

Evaluasi kualitas *cluster* pada metode *clustering* yang digunakan menggunakan fungsi evaluasi yang diusulkan oleh Xie and Beni. Fungsi validitas Xie and Beni untuk *worst case* ditunjukkan pada Persamaan 2.20.

$$S = \frac{\sigma/N}{{d_{min}}^2} \tag{2.20}$$

Sebelum menjalankan nilai *S* sesuai persamaan di atas, nilai *S* diinisialisasi terlebih dahulu sebagai patokan pembanding dari nilai *S* yang akan diperoleh berikutnya. Apabila nilai *S* yang baru telah diperoleh dari Persamaan 2.20 maka nilai S tersebut dibandingkan dengan nilai *S* sebelumnya. Jika nilai *S* yang baru lebih kecil, maka akan melakukan proses perulangan hingga nilai *S* yang baru lebih besar atau sama dengan *nilai S* yang lama.

Nilai d_{min} dari Persamaan 2.20 di atas diperoleh dari Persamaan 2.21 berikut ini.

$$d_{min} = \min_{\forall C}\left\{\sum_{j=1}^{K}\left(p_{(c+1)j} - p_{cj}\right)^2\right\} \tag{2.21}$$

d_{min} merupakan jarak minimal antara pusat *cluster* p_{cj} untuk *cluster* c=*1,...,C* dan fitur *j*. σ adalah variasi maksimal antara semua *cluster* C yang didefinisikan dengan Persamaan 2.22 berikut.

$$\sigma = \max_{\forall C}\left\{\sum_{i=1}^{N} {u_{ci}}^2 \sum_{j=1}^{K}(x_{ij} - p_{cj})^2\right\} \tag{2.22}$$

2.8. Evaluasi Kuantitatif Segmentasi Citra

Untuk mengevaluasi hasil dari suatu segmentasi citra, salah satu fungsi yang dapat digunakan yaitu fungsi evaluasi dari Liu dan Yang dimana kelebihan dari fungsi ini adalah metode ini memberikan pengukuran yang akurat terhadap perbedaan warna yang didapatkan dari suatu algoritma segmentasi yang digunakan.

Fungsi dari Liu dan Yang menghitung perbedaan antara nilai piksel citra asli dengan piksel citra tersegmentasi. Nilai perbedaan tersebut merupakan *error color* yang dihasilkan oleh citra tersegmentasi. Persamaan untuk mendapatkan *error color* tersebut ditunjukkan pada Persamaan 2.23.

$$F(I) = \frac{1}{1000N}\sqrt{G}\sum_{i=1}^{G}\frac{ei^2}{\sqrt{Ai}} \tag{2.23}$$

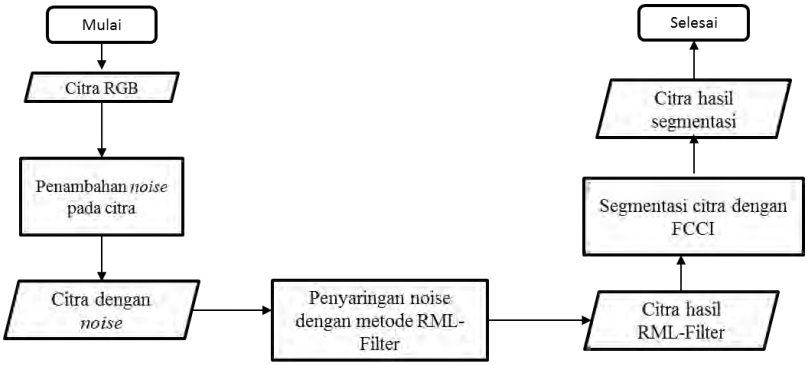
dimana N adalah jumlah piksel citra, G adalah jumlah *cluster* dan A_i adalah jumlah piksel dari daerah ke-i. e_i^2 menyatakan jumlah jarak *Euclidean* dari vektor antara citra asli dan masing-masing piksel pada daerah citra segmentasi. Hasil segmentasi citra yang baik ditunjukkan dengan nilai evaluasi kuantitas yang rendah.

BAB III
ANALISIS DAN PERANCANGAN

Perancangan dan desain sistem yang akan dibahas dalam bab ini meliputi desain secara umum dari sistem, perancangan data, data-data yang digunakan, diagram alir (*flowchart*) dan *pseudocode*.

3.1. Desain Secara Umum

Proses segmentasi citra dengan *noise* menggunakan metode FCCI dan estimasi *robust* spasial memiliki dua tahapan utama. Sebelum citra dengan *noise* disegmentasi menggunakan algoritma FCCI, citra masukan harus melalui tahapan *preprocessing* terlebih dahulu yaitu penyaringan *noise* menggunakan estimasi *robust* spasial untuk memperbaki kualitas piksel yang akan disegmentasi sehingga hasil segmentasi citra yang dihasilkan menjadi lebih baik. Citra hasil keluaran dari tahap preprocessing kemudian disegmentasi dengan algoritma FCCI. Diagram alir proses segmentasi citra dengan algoritma FCCI dan estimasi *robust* spasial ditunjukkan pada Gambar 3.1 dan *pseudocode* pada Gambar 3.2.



Gambar 3.1 Diagram Alir Model Sistem Secara Umum

Masukan	Matrik citra RGB
Keluaran	Matrik citra hasil segmentasi

```
1. img_original=imread('67079')
2. [col row n]=size(img_original)
3. img_noise=imnoise(img_original,tipe noise)
4. img_filtering=filtering(img_noise, ukuran
   img_original)
5. output=segmentation(img_filtering)
```

Gambar 3.2 *Pseudocode* program utama

Daftar variabel dan fungsi yang digunakan untuk program utama pada proses penyaringan *noise* dan segmentasi citra berwarna menggunakan FCCI dan RML-Filter ditunjukkan pada Tabel 3.1 dan Tabel 3.2.

Tabel 3.1 Daftar variabel untuk program utama pada proses penyaringan *noise* dan segmentasi dari citra berwarna

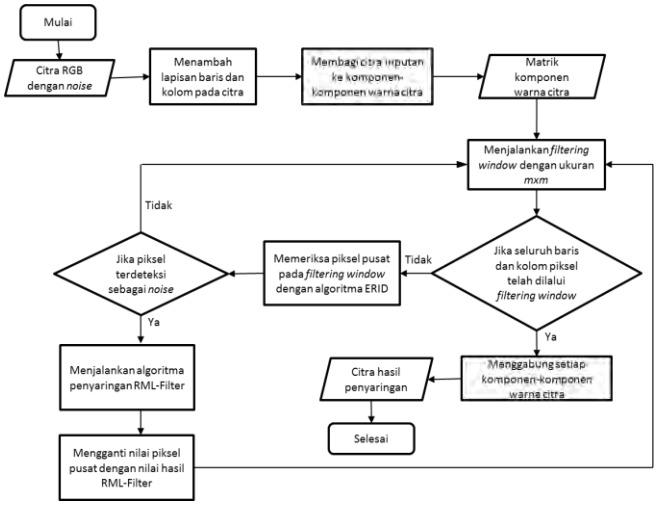
No	Nama variabel	Tipe	Penjelasan
1	img_original	uint8	Citra asli yang akan diproses
2	col	*integer*	Ukuran kolom dari img_original
3	row	*integer*	Ukuran baris dari img_original
4	n	*integer*	Ukuran dimensi dari img_original
5	img_*noise*	uint8	Citra yang telah didegradasi *noise*
6	img_filtering	uint8	Citra hasil proses penyaringan *noise* dengan RML-Filter
7	output	uint8	Citra hasil segmentasi dari algoritma FCCI

Tabel 3.2 Daftar fungsi untuk program utama pada proses penyaringan *noise* dan segmentasi dari citra

No	Nama fungsi	Penjelasan
1	filtering	Fungsi untuk melakukan penyaringan *noise* dengan RML-Filter
2	segmentation	Fungsi untuk melakukan segmentasi citra dengan menggunakan metode FCCI

3.2. Perancangan Penyaringan *Noise* pada Citra dengan Robust Estimator

Salah satu proses utama pada aplikasi ini yaitu perancangan *filter* untuk proses *denoising* dari citra. Untuk proses penyaringan *noise* tersebut, jenis *filter* yang digunakan merupakan *filter* dengan estimasi *robust* spasial yaitu RML-Filter. Untuk melakukan penyaringan *noise* pada citra masukan diperlukan beberapa tahapan. Langkah-langkah dari proses penyaringan *noise* digambarkan pada diagram alir pada Gambar 3.3 berikut.



Gambar 3.3 Diagram alir proses penyaringan *noise* dari citra masukan

Daftar variabel dan fungsi yang digunakan untuk proses penyaringan noise dengan estimasi robust spasial yaitu RML-Filter pada kasus citra berwarna ditunjukkan pada Tabel 3.3, Tabel 3.4, Tabel 3.5, dan Tabel 3.6.

Tabel 3.3 Daftar variabel untuk proses penyaringan *noise* pada kasus citra berwarna (Bagian pertama)

No	Nama variabel	Tipe	Penjelasan
1	img_modify_ *noise*	uint8	Citra dengan *noise* yang telah diperluas area piksel di sekitar citra
2	firstChannel *Noise*	*Cell*	komponen warna pertama citra yang telah didegradasi *noise*
3	secondChannel *Noise*	*Cell*	komponen warna kedua citra yang telah didegradasi *noise*
4	thirdChannel *Noise*	*Cell*	komponen warna ketiga citra yang telah didegradasi *noise*
5	channel*Noise* FreeFirst	*Cell*	komponen warna pertama citra hasil dari RML-Filter
6	channel*Noise* FreeSecond	*Cell*	komponen warna kedua citra hasil dari RML-Filter
7	channel*Noise* FreeThird	*Cell*	komponen warna ketiga citra hasil dari RML-Filter
8	img_fixed	*Cell*	Citra hasil penggabungan komponen warna channel*Noise*FreeFirst, channel*Noise*FreeSecond, channel*Noise*FreeThird
9	a	Array of double	List yang menampung koefisien pembobotan
10	med_a	*double*	Nilai median dari vektor a

Tabel 3.4 Daftar variabel untuk proses penyaringan *noise* pada kasus citra berwarna (Bagian kedua)

No	Nama variabel	Tipe	Penjelasan
11	window	Array of integer	List yang menampung nilai piksel yang berada dalam *filtering window*
12	idx	integer	Variabel untuk menampung panjang vektor window
13	sizeWin	integer	Ukuran satuan window dari *filtering window*
14	x	integer	Posisi indek baris posisi piksel yang sedang dilakukan proses *filtering*
15	y	integer	Posisi indek kolom posisi piksel yang sedang dilakukan proses *filtering*
16	res	Array of double	Vektor penampung hasil perhitungan $\tilde{\psi}(X_i - med\{\bar{X}\})$ dari Persamaan (2.7)
17	ind	integer	Variabel untuk menampung panjang vektor res
18	sorted_list	array of integer	List 1D berukuran $(2k+1)^2$ yang menampung nilai piksel dalam filtering window dan telah dalam kondisi terurut
19	med	integer	Nilai tengah dari list piksel yang berada dalam satu filtering window
20	mmed	double	Nilai hasil dari influence function pada metode M-Filter
21	filter_res	integer	Nilai piksel θ_{MML} hasil dari RML-Filter sesuai Persamaan (2.7)
22	s	integer	Nilai ambang batas toleransi posisi piksel yang dikenai *noise*

Tabel 3.5 Daftar variabel untuk proses penyaringan *noise* pada kasus citra berwarna (Bagian ketiga)

No	Nama variabel	Tipe	Penjelasan
23	T	Double	Nilai ambang batas untuk selisih antara piksel pada pusat filtering window dengan nilai tengah
24	yc	integer	Posisi piksel tengah dari citra dalam filtering window
25	median	integer	Nilai tengah dari piksel dalam *filtering window*
26	N	integer	Panjang vektor yang menampung piksel dalam filtering window
27	irank	integer	Posisi urutan piksel pada pusat filtering window
28	check*Noise*	boolean	Nilai kondisi apakah piksel pada pusat filtering window terdegradasi *noise* atau tidak

Tabel 3.6 Daftar fungsi untuk proses penyaringan *noise* pada kasus citra berwarna

No	Nama fungsi	Penjelasan
1	rml_filter	Fungsi untuk memanggil metode estimasi robust spasial yaitu RML-Filter dengan memroses citra ke dalam blok-blok dalam *filtering window*
2	ERID	Fungsi untuk memanggil algoritma ERID yang melakukan pengecekan terhadap nilai piksel tengah pada *filtering window*
3	hampel	Fungsi untuk memanggil jenis *influence function* sebagai bagian metode dari RML-Filter
4	window_filter	Fungsi untuk mendapatkan vektor yang berisi nilai piksel-piksel dalam *filtering window*

3.2.1. Program Utama *Filtering*

Program utama *filtering* merupakan program yang memanggil fungsi-fungsi lain untuk menjalankan program secara keseluruhan. Proses-proses yang dilakukan dalam algoritma yang digunakan dipisahkan dalam fungsi-fungsi yang berbeda untuk memudahkan pengecekan program setiap prosesnya. *Pseudocode* program utama *filtering* ditunjukkan pada Gambar 3.4.

Masukan	Matrik citra RGB dengan *noise* img_*noise*, ukuran baris dan kolom citra asli row, col
Keluaran	Matrik citra hasil penyaringan *noise* img_filtering

```
1. img_modify_noise=padarray(img_noise,ukuran lapisan)
2. firstChannelNoise = komponen warna pertama
   img_modify_noise
3. secondChannelNoise = komponen warna kedua
   img_modify_noise
4. thirdChannelNoise = komponen warna ketiga
   img_modify_noise
5. channelNoiseFreeFirst =
   rml_filter(firstChannelNoise, row, col, ukuran
   window)
6. channelNoiseFreeSecond =
   rml_filter(secondChannelNoise, row, col, ukuran
   window)
7. channelNoiseFreeThird =
   rml_filter(thirdChannelNoise, row, col,ukuran
   window)
8. img_fixed = cat(3, channelNoiseFreeFirst,
   channelNoiseFreeSecond, channelNoiseFreeThird)
9. img_filtering=uint8(img_fixed)
```

Gambar 3.4 *Pseudocode* program utama *filtering*

3.2.2. Inisialisasi *Filtering Window*

Program inisialisasi *filtering window* diperlukan untuk proses penyaringan *noise* pada domain spasial. Ukuran *filtering window* yang digunakan *mxm* dengan *m* merupakan bilangan bulat positif. Setiap piksel yang berada dalam *filtering window* disimpan dalam vektor satu dimensi dengan panjang vektor *mxm* untuk kemudian diproses dengan RML-Filter. *Pseudocode* inisialisasi *filtering window* ditunjukkan pada Gambar 3.5.

Masukan	Matrik komponen citra img_channel, ukuran window sizeWin, posisi baris dan kolom piksel x, y
Keluaran	vektor 1 dimensi dengan ukuran sizeWin x sizeWin

```
1. Inisialisasi ukuran window dengan ukuran sizeWin x
   sizeWin
2. idx=1
3. for i=1 to sizeWin
4.     for j=1 to sizeWin
5.        window[idx] = img_channel[x+i-1][y+j-1]
6.        idx=idx+1
7.     end for
8. end for
```

Gambar 3.5 *Pseudocode* inisialisai *filtering window*

3.2.3. *Pseudocode* Algoritma ERID

Program ERID merupakan program untuk memeriksa apakah piksel pada pusat *filtering window* merupakan piksel yang terdegadrasi *noise* atau tidak. Setiap piksel yang berada dalam *filtering window* diurutkan dari nilai piksel terkecil hingga terbesar sehingga setiap piksel mendapatkan posisi urutan masing-masing. Setelah setiap piksel terurut, algoritma ERID memeriksa apakah posisi dari piksel pada pusat *filtering window* memenuhi syarat-syarat sebagai piksel yang terdegradasi *noise*. *Pseudocode* untuk proses ERID ditunjukkan pada Gambar 3.6**.**

Masukan	List piksel sorted_list, List piksel window, panjang list window
Keluaran	Boolean check*Noise*

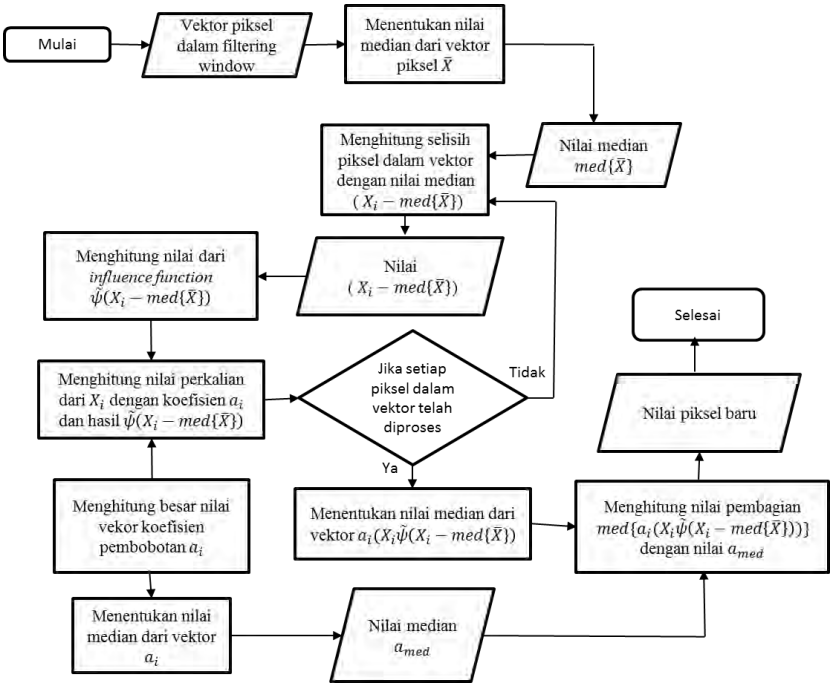
```
1. Inisialisasi nilai s dan T
2. yc= nilai piksel pusat dari list window
3. irank=find(sorted_list=yc)
4. if (irank <= s) or (irank >= N-s)) and (abs(yc-
   med) >= T)
5.     checkNoise=1
6. else
7.     checkNoise=0
8. end if
```

Gambar 3.6 *Pseudocode* program ERID

3.2.4. RML-Filter

Program RML-Filter merupakan program yang melakukan proses penyaringan *noise* dari citra dengan dasar Persamaan 2.7. Piksel yang berada pada pusat *filtering window* diperiksa terlebih dahulu menggunakan algoritma ERID untuk melihat apakah piksel tersebut terdegradasi *noise* atau tidak. Apabila piksel tersebut tidak terdeteksi terdegradasi *noise* maka nilai piksel pada pusat *filtering window* tersebut tidak diganti dengan suatu nilai piksel baru, akan tetapi jika terdeteksi terdegradasi *noise* maka nilai piksel pada pusat *filtering window* digantikan dengan nilai piksel dari hasil RML-Filter.

Untuk melakukan fungsi RML-Filter diperlukan penentuan *influence function* sesuai Tabel 2.1. Selain itu, juga perlu diinisialisasi besar koefisien pembobotan sesuai Persamaan 2.6. Besar koefisien pembobotan ini digeneralisasi berdasarkan distribusi yang digunakan. Nilai yang didapatkan masing-masing menjadi koefisien pembobotan untuk setiap piksel dalam *filtering window*. Diagram alir untuk proses RML-Filter ditunjukkan pada Gambar 3.7 dan *Pseudocode* ditunjukkan pada Gambar 3.8.



Gambar 3.7 Diagram alir untuk proses RML-Filter

3.2.4.1. Generalisasi Koefisien Pembobotan

Pada metode *robust* RML-Filter diperlukan koefisien pembobotan a yang merupakan karakteristik L-Filter sebagai salah satu metode pembentuk RML-Filter. Koefisien pembobotan tersebut diperoleh dari hasil generalisasi *probability density function* dari distribusi tertentu. Untuk Tugas Akhir ini, jenis distribusi yang diuji coba yaitu distribusi normal dan distribusi uniform. Setiap distribusi yang digunakan, akan menghasilkan nilai koefisien pembobotan a yang berbeda. *Pseudocode* untuk proses generalisasi koefisien pembobotan a dapat dilihat pada Gambar 3.9.

Masukan	Komponen warna citra masukan dengan *noise* img_channel, ukuran baris dan kolom komponen warna citra asli row, col, ukuran window sizeWin
Keluaran	Matrik citra RGB hasil *filtering*

```
1. Menyiapkan   img_channel_filter   sebagai   matrik
   penampung hasil proses filtering
2. Menentukan besar vektor koefisien pembobotan a
3. med_a=nilai tengah vektor a
4.     for x=1 to panjang baris img_channel-2
5.             for   y=1   to   panjang   kolom   baris
           img_channel-2
6.         [window     idx]=window_filter(img_channel,
           sizeWin, x, y)
7.         inisialisasi vektor res
8.         sorted_list=sort(window)
9.         med=sorted_list(idx/2)
10.        if ERID(window,sorted_list,idx) = 1
11.         for z=1 to size(window)
12.           mmed=window(z)*(hampel(window(z)-med))
13.           res(ind)=mmed*a(z)
14.           ind=ind+1
15.         end
16.         filter_res= median(res)/med_a
17.         img_channel_filter(x,y)=filter_res
18.       else
19.         img_channel_filter(x,y)=window(idx/2)
20.       end
21.   end
22. end
```

Gambar 3.8 *Pseudocode* untuk proses RML-Filter

Masukan	-
Keluaran	Vektor koefisien pembobotan a

```
1. Generate titik data sesuai rentang nilai
   diinginkan
2. Menghitung PDF dari setiap titik data yang telah
   di-generate
3. denom=hasil hitungan penyebut dari Persamaan 2.6
4. Menyiapkan vector a dengan panjang sesuai luas
   filtering window
5. for i=1 to panjang vektor a
6.     if i = 1
7.        idx=find(lamda >= 0 & lamda <= 1/9);
8.     else
9.        idx=find(lamda > (i-1)/9 & lamda <= i/9);
10.    end
11.       num=hasil hitungan pembilang dari Persamaan
          2.6
12.       a(1,i)=num/denom;
13. end
```

Gambar 3.9 *Pseudocode* generalisasi koefisien pembobotan $\boldsymbol{a}$

3.2.4.2. *Influence Function*

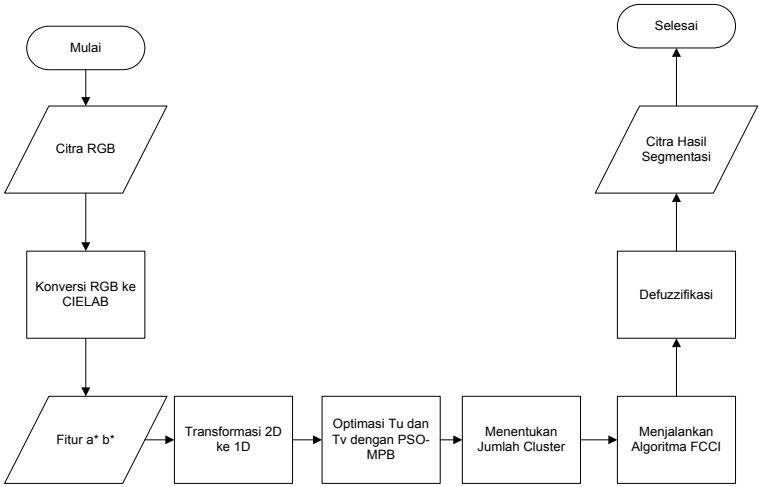
Untuk melakukan proses penyaringan *noise* dengan RML-Filter diperlukan *influence function* sebagai karakteristik dari M-Filter yang merupakan salah satu metode pembentuk RML-Filter. Setiap selisih dari piksel dalam filtering window dengan nilai tengah dari vektor piksel pada filtering window dijadikan sebagai inputan salah satu influence function yang dipilih dari Tabel 2.1. Hasil dari proses ini, merupakan nilai yang diperoleh dari penghitungan pada persamaan influence function yang dipilih.

3.3. Perancangan Segmentasi Citra dengan FCCI

Proses utama berikutnya dalam perancangan sistem yang dibuat yaitu perancangan segmentasi citra dengan FCCI. Untuk melakukan segmentasi citra dengan algoritma FCCI perlu melalui beberapa tahapan. Tahap pertama adalah konversi warna dari ruang warna RGB ke dalam ruang warna CIELAB. Proses ini bertujuan

untuk mendapatkan dua fitur warna yang akan digunakan yaitu a dan b (K=a*,b*). Tahap kedua adalah, melakukan transformasi dari 2D ke 1D dengan *columnwise*. Hal ini dilakukan karena komputasi akan menjadi lebih mudah jika berada dalam 1D.

Kemudian tahap ketiga yaitu, mendapatkan jumlah *cluster* yang tepat untuk setiap citra yang diproses. Langkah keempat yang dilakukan adalah menjalankan algoritma FCCI sesuai dengan jumlah *cluster* yang sesuai. Kemudian langkah terakhir adalah melakukan defuzzifikasi *cluster*. Diagram alir dari proses segmentasi citra dapat dilihat pada Gambar 3.10.



Gambar 3.10 Diagram Alir Proses Segmentasi Citra dengan FCCI

Penjelasan dari proses segmentasi citra dengan FCCI secara lebih detail dijelaskan pada subbab-subbab berikut dengan disertai diagram alir dan pseudocode program. Untuk memperjelas pengertian setiap diagram alir dan pseudocode yang digunakan, berikut ini diberikan daftar variabel dan fungsi yang digunakan untuk proses segmentasi citra berwarna dengan FCCI yang ditunjukkan pada Tabel 3.7 dan Tabel 3.8.

Tabel 3.7 Daftar variabel untuk proses segmentasi citra berwarna dengan FCCI (Bagian pertama)

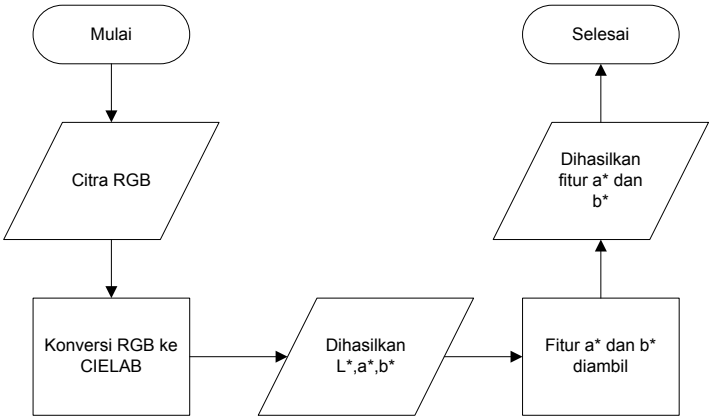
No.	Nama Data	Tipe Data	Keterangan
1.	Uci	double	Nilai fungsi keanggotan objek yang berada diantara 0 dan 1 yang digunakan untuk mencari jumlah *cluster* optimal.
2.	Vcj	double	Nilai fungsi keanggotaan fitur yang juga digunakan untuk mencari besarnya uci.
3.	Pcj	double	Besarnya nilai pusat *cluster* dari fitur ke-j terhadap centroid *cluster* ke-c. Pcj ini digunakan untuk mencari jarak Euclidean Dcij.
4.	Dcij	double	Parameter Dcij merupakan jarak Euclidean antara titik data ke-i terhadap *cluster* c dan fitur j. Dcij digunakan untuk mencari nilai vcj dan uci.
5.	Tv	integer	Tv merupakan parameter bobot yang ada di dalam fungsi objektif FCCI yang berkontribusi dalam menentukan nilai vcj.
6.	Tu	integer	Tu merupakan parameter bobot yang ada di dalam fungsi objektif FCCI yang berkontribusi dalam menentukan nilai uci.

Tabel 3.8 Daftar variabel untuk proses segmentasi citra berwarna dengan FCCI (Bagian kedua)

No.	Nama Data	Tipe Data	Keterangan
7.	C	integer	Parameter C merupakan parameter yang merepresentasikan jumlah *cluster* yang digunakan untuk melakukan segmentasi terhadap citra input.
8.	S_old	integer	Nilai validitas *cluster* Xie and Beni's awal yang ditentukan melalui inisialisasi. Parameter ini menunjukkan kualitas *cluster* yang akan dibandingkan dengan nilai validitas *cluster* yang baru.
9.	S_new	double	Nilai validitas *cluster* Xie and Beni's yang baru

3.3.1. Proses Konversi Ruang Warna RGB ke CIELAB

Proses konversi citra dari ruang warna RGB ke ruang warna CIELAB bertujuan untuk mendapatkan fitur yang diperlukan dalam proses segmentasi dengan metode FCCI. Fitur yang diperoleh dari hasil konversi ruang warna citra yaitu *Luminance* (L*), fitur a*, dan fitur b*. Namun dalam FCCI, hanya fitur a* dan b* saja yang akan digunakan. Hal ini bertujuan untuk membuktikan bahwa pencahayaan (*luminance*) tidak memberikan efek yang signifikan terhadap hasil segmentasi citra. Diagram alir dari proses ini ditunjukkan pada Gambar 3.11 dan *pseudocode* pada Gambar 3.12.



Gambar 3.11 Diagram Alir Konversi Warna dari RGB ke CIELAB

Masukan	Matriks Citra RGB
Keluaran	Fitur (a,b)

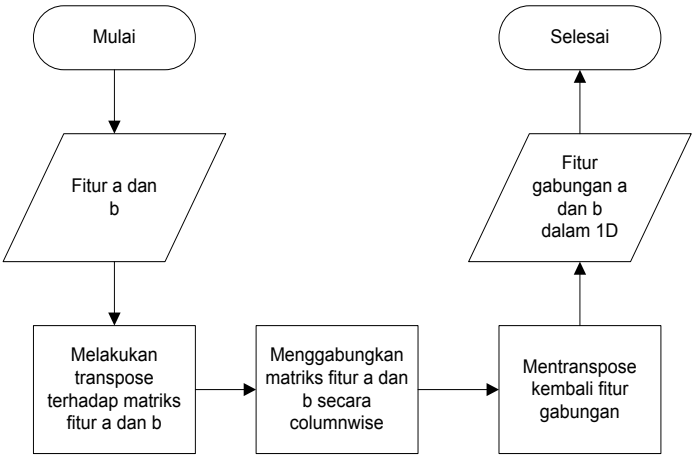
```
1.labTransformation=makecform('srgb2lab')
2.lab=applycform(image,labTransformation)
3.Menentukan besarnya citra N=mxn
4.Mengambil fitur a dan b
5.Fitur a dan b
6.K=size(ab,3)
```

Gambar 3.12 *Pseudocode* konversi warna dari RGB ke CIELAB

3.3.2. Transformasi dari *2D* ke *1D*

Proses ini merupakan tahap penyimpanan titik data x_{ij} ke dalam dimensi *j*, dimana *j* merupakan fitur citra. Dalam sistem ini, terdapat dua fitur citra yang digunakan dalam proses segmentasi menggunakan FCCI sehingga ditentukan nilai *j=1,2.* Setiap fitur *j* tersebut dimiliki oleh setiap piksel *i=1,...,N,* dimana *N* merupakan jumlah data keseluruhan. Langkah ini menjadi penting karena komputasi akan menjadi lebih mudah jika data disimpan dalam 1D

dibandingkan dalam 2D. Dalam proses sebelumnya dihasilkan dua fitur a dan b yang berada pada matriks yang berbeda. Kemudian dalam transformasi ini, kedua fitur tersebut ditransformasikan ke dalam dua kolom dimana kedua kolom tersebut dimiliki oleh satu baris piksel. Diagram alir dari proses ini ditunjukkan pada Gambar 3.13 dan *pseudocode* pada Gambar 3.14.



Gambar 3.13 Diagram Alir Transformasi 2D ke 1D

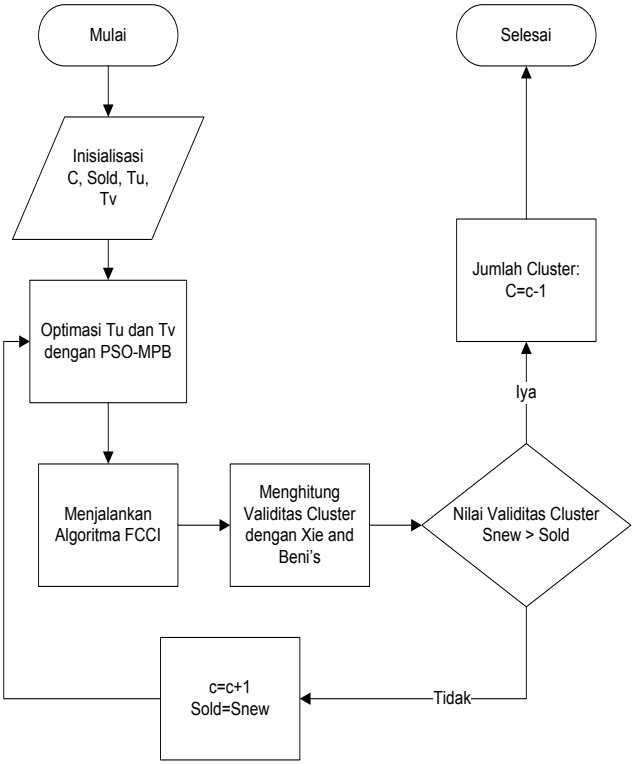
Masukan	Matriks fitur a,b
Keluaran	Xij

```
1. Transpose matrik ab(:,:,1)dan(:,:,2)
2. D1 = reshape(ab(:,:,1)',1,N
3. D2 = reshape(ab(:,:,2)',1,N)
4. Menggabungkan matriks secara columnwise
5. Xji = [D1;D2]
6. Mentranspose kembali matriks gabungan
4. Xij = Xji'
```

Gambar 3.14 *Pseudocode* Transformasi 2D ke 1D

3.3.3. Menentukan Jumlah *Cluster C*

Proses penentuan jumlah *cluster* dalam algoritma FCCI menggunakan *Particel Swarm Optimization* dengan Modifikasi Perilaku Bakteri (PSO-MPB) untuk optimasi parameter T_u dan T_v. Proses ini bertujuan untuk mendapatkan jumlah *cluster* yang tepat untuk segmentasi citra yang sedang diproses. Dengan jumlah *cluster* yang tepat maka diharapkan hasil segmentasi akan lebih optimal. Diagram alir dari proses ini ditunjukkan pada Gambar 3.15 dan *pseudocode* pada Gambar 3.16.



Gambar 3.15 Diagram Alir Penentuan Jumlah *Cluster*

Masukan	C,S_old,Tu,Tv,Uci
Keluaran	C(jumlah *cluster*)

```
1.t=1
2.while(1)
3.  if(t>1)
4.    [Tu,Tv]=BacterialForagingAlgorithm
5.    (Uci,Vcj,Dcij)
6.  end if
7.  t=t+1
8.  [Uci,Vcj,Pcj,Dcij]=
9.  FCCI(C,N,K,Tu,Tv,Xij)
10. S_new=clusterValidity(Uci,Xij,Pcj)
11. if(S_new > S_old)
12.    C=C-1
13.    break
14. else
15.    C=C+1
16.    S_old=S_new
17. end if
18.end while
```

Gambar 3.16 *Pseudocode* Penentuan Jumlah *Cluster*

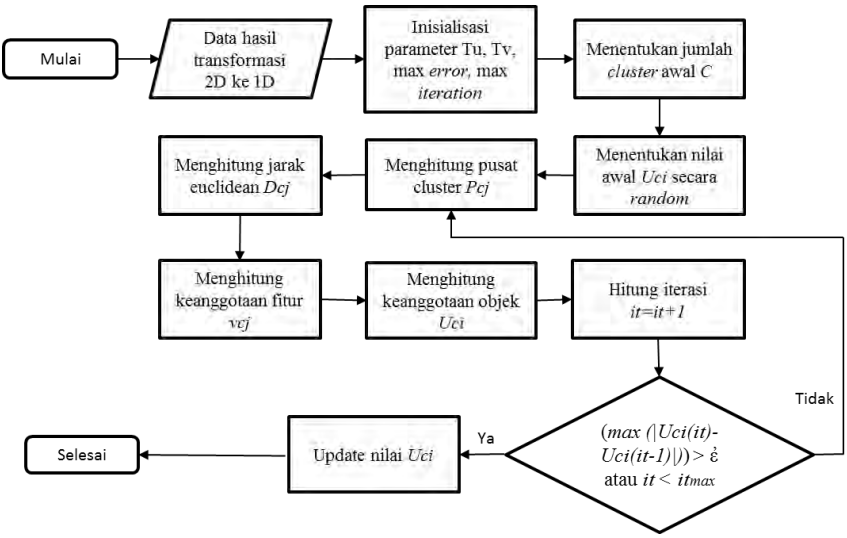
3.3.4. Menjalankan Algoritma FCCI

Setelah memperoleh jumlah *cluster* yang tepat untuk setiap citra yang sedang diproses, langkah berikutnya yaitu menjalankan kembali algoritma FCCI. Hal ini dilakukan untuk mendapatkan nilai dari fungsi keanggotaan objek u_{ci} yang tepat, sebelum proses defuzzifikasi citra. Diagram alir dari proses ini ditunjukkan pada Gambar 3.17 serta *pseudocode* pada Gambar 3.18 dan Gambar 3.19.

Diagram alir pada Gambar 3.17 dan *pseudocode* pada Gambar 3.18 sampai Gambar 3.19 menunjukkan rangkaian alur kerja dari algoritma FCCI. *Pseudocode* pada baris 9 sampai dengan baris 13 menunjukkan proses pencarian pusat *cluster c* terhadap fitur *j* (p_{cj}). Dimana dalam proses pencarian pusat *cluster* ini akan

mengimplementasikan Persamaan 2.12 yang telah dijelaskan pada bab sebelumnya. Untuk *pseudocode* baris 14 sampai dengan baris 19 ini merupakan *pseudocode* yang digunakan untuk mencari jarak *Euclidean* antara *cluster c*, titik data *i*, dan fitur *j*. Dalam pencarian D_{cij} ini menggunakan Persamaan 2.15.

Setelah mencari jarak *Euclidean*, langkah selanjutnya adalah mencari nilai dari fungsi keanggotaan fitur v_{cj} , dimana dalam *pseudocode* tersebut ditunjukkan pada baris 20 sampai dengan baris 29. Pencarian v_{cj} ini mengimplementasikan Persamaan 2.17. Langkah terakhir adalah mencari nilai dari fungsi keanggotaan objek u_{ci}. Fungsi keanggotaan objek ini menunjukkan hubungan antara tiap *cluster c* terhadap titik data *i*. Untuk implementasi dalam *pseudocode* ini ditunjukkan pada baris 30 sampai dengan baris 41. Pencarian u_{ci} ini menerapkan Persamaan 2.16.



Gambar 3.17 Diagram Alir Algoritma FCCI

Masukan	maxError,Tu,Tv, maxIteration,Uci_bef
Keluaran	Uci

```
1. it=1
2. for i=1 to jumlah titik data
3.      for j=1 to jumlah fitur
4.          Uci(i,j)=rand()
5.      end for
6. end for
7. Uci_bef=Uci
8. while(true)
9.       sumUci=sum(Uci,2)
10.      Pcj=Uci*Xij
11.       for j=1 to jumlah fitur K
12.           Pcj(:,j)=Pcj(:,j)./sumUci
13.       end for
14.       for i=1 to N titik data
15.         for c=1 to C jumlah cluster
16.           Dcij(c,i,:)=(Xij(i,:)- Pcj(c,:)).^2
17.         end for
18.       end for
19.       vcj2=zeros(C,K)
20.       for j=1 to jumlah fitur K
21.           vcj1=Uci.*Dcij(:,:,j)/Tv
22.           vcj1=sum(vcj1,2)
23.           vcj2(:,j)=vcj1
24.       end for
25.           vcj2=exp(-1*vcj2)
26.           vcj3=sum(vcj2,2)
27.       for j=1 to jumlah fitur K
28.           Vcj(:,j)=vcj(:,j)./vcj3
29.       end for
30.           vd1=zeros(K,N)
31.           vd3=zeros(C,N)
32.       for c=1 to jumlah cluster C
33.           for j=1 to jumlah fitur K
34.               vd1(j,:)=Vcj(c,j).*
35.               Dcij(c,:,j)/Tu
36.           end for
37.       end for
```

Gambar 3.18 *Pseudocode* Algoritma FCCI (Bagian pertama)

Masukan	maxError,Tu,Tv, maxIteration,Uci_bef
Keluaran	Uci

```
37.                vd3=exp(-1*vd3)
38.                vd4=sum(vd3)
39.             for c=1 to jml cluster C
40.                Uci=(c,:)=vd3(c,:)./vd4
41.             end for
42.         it=it+1
43.         if (max(max(abs(Uci-Uci_bef)))<=
44.            maxError||it==maxIteration)
45.         break
46.         end if
47.         Uci_bef=Uci
48. end while
```

Gambar 3.19 *Pseudocode* Algoritma FCCI (Bagian kedua)

3.3.5. Proses Defuzzifikasi

Proses terakhir yang dilakukan untuk sementasi citra adalah defuzzifikasi. Untuk melakukan defuzzifikasi, dipilih nilai u_{ci} yang paling maksimal dimana u_{ci} merupakan fungsi keanggotaan objek yang dihasilkan dari proses sebelumnya. Setelah itu, semua titik data akan di *cluster* sesuai dengan nilai u_{ci} yang dimiliki. Proses ini dilakukan sesuai dengan banyaknya piksel data dalam citra yang diproses.

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi yang dilakukan berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya yang meliputi algoritma dan kode program yang terdapat dalam perangkat lunak. Algoritma dan kode program tersebut meliputi algoritma dan kode program dari metode FCCI dan estimasi *robust* spasial. Sebelum penjelasan implementasi akan ditunjukkan terlebih dahulu lingkungan untuk melakukan implementasi.

4.1. Lingkungan implementasi

Spesifikasi perangkat keras dan perangkat lunak yang digunakan dalam segmentasi citra dengan *noise* menggunakan algoritma FCCI dan estimasi *robust* spasial dijelaskan sebagai berikut:

Prosesor	: Intel Core i3-3240 CPU @ 3.40GHz
Memori	: 4.00 GB
Jenis Device	: Personal Computer
Sistem Operasi	: Microsoft Windows 8 Enterprise 32 bit

4.2. Implementasi

Pada subbab ini akan dijelaskan implementasi dari setiap subbab pada bab perancangan perangkat lunak. Sebelum pembahasan implementasi secara detail sesuai pembagian proses dari perangkat lunak yang dirancang, akan dijelaskan implementasi dari program utama yang bertugas memanggil fungsi-fungsi utama lain yang mewakili pembagian proses dari perangkat lunak yang dirancang. Data masukan dari program utama ini yaitu citra berwarna RGB. Kode dari program utama dapat dilihat pada Kode Sumber 4.1.

```
1.  %-- membaca inputan citra --%
    img_original=imread('67079.jpg');
2.  %-- memperoleh ukuran baris, kolom, dan dimensi
    warna citra --%
    [row col n]=size(img_original);
3.  %-- menambahkan noise pada citra asli --%
    img_noise = imnoise(img_original,'speckle',0.01);
4.  %-- penyaringan noise dari citra dengan rml filter
    --%
    img_filtering=filtering(img_noise, row, col);
5.  %-- segmentasi citra dengan Fuzzy clustering --%
    segmentation(img_noise);
```

Kode Sumber 4.1 Program utama dari perangkat lunak

Pada program utama, data masukan berupa citra RGB didegradasi *noise* terlebih dahulu dan ditunjukkan pada Baris 3. Tahapan proses selanjutnya yaitu proses penyaringan *noise* dan segmentasi citra ditunjukkan secara berturut-turut pada Baris 4 dan Baris 5.

4.2.1. Implementasi Penyaringan *Noise* pada Citra dengan Robust Estimator

Pada bagian ini akan dijelaskan implementasi dari proses penyaringan citra dengan *noise* menggunakan *robust* estimator. Proses tersebut secara terurut meliputi tahap inisialisasi window, pengecekan piksel dengan algoritma ERID, dan tahap utama yaitu RML-Filter sebagai tahap untuk penggantian nilai piksel pada pusat *filtering window*.

4.2.1.1. Inisialiasi Filtering Window

Pada bagian ini, citra yang telah diberi *noise* akan dibagi menjadi blok-blok bujur sangkar dengan ukuran *mxm* dimana *m* merupakan bilangan bulat positif. Piksel pada pusat blok bujur sangkar menjadi poros untuk nilai piksel-piksel tetangga yang nantinya akan disimpan pada vektor satu dimensi sebagai masukan

untuk tahap RML-Filter. Fungsi inisialisasi filtering window ini akan terus dipanggil berulang hingga semua piksel dari citra masukan telah diproses. Kode program dari inisialisasi *filtering window* ditunjukkan pada Kode Sumber 4.2.

```
1.  function [window idx] =
    window_filter(img_channel, sizeWin, x, y)
2.  window=zeros(sizeWin*sizeWin,1);
3.  idx=1;
4.  for i=1:sizeWin
5.     for j=1:sizeWin
6.        window(idx)= img_channel(x+i-1,y+j-1);
7.        idx=idx+1;
8.     end
9.  end
10 end
```

Kode Sumber 4.2 Program Inisialisasi Filtering Window

Inisialisasi vektor untuk menampung nilai dari piksel-piksel di dalam filtering window ke vektor satu dimensi ditunjukkan pada Baris 1. Baris 3 hingga Baris 8 merupakan proses untuk mendapatkan nilai piksel-piksel tetangga dari piksel citra yang sedang diproses.

4.2.1.2. Implementasi Algoritma ERID

Setelah diperoleh vektor yang menyimpan nilai piksel-piksel dalam filtering window, maka langkah berikutnya melakuakn pemeriksaan terhadap piksel pada pusat filtering window menggunakan algoritma ERID. Kode program dari algoritma ini ditunjukkan pada Kode Sumber 4.3.

```
1.  function checkNoise =
    ERID(window,sorted_list,idx)
2.  s=4;
3.  T=5;
4.  N=size(window)
5.  median=sorted_list(idx/2);
6.  yc=window(idx/2);
7.  ind = find(sorted_list==yc);
8.  if ind(1) <=  idx/2
9.     irank=ind(1);
10. else
11.    irank=ind(length(ind));
12. end
13. if ((irank <= s) | (irank >= N-s+1)) &
    (abs(yc-   median) >= T)
14.    checkNoise=1;
15. else
16.    checkNoise=0;
17. end
```

Kode Sumber 4.3 Program Implementasi Algoritma ERID (Bagian pertama)

Beberapa parameter perlu diinisialisasi terlebih dahulu untuk menggunakan algoritma ERID yaitu nilai *s* sebagai acuan batas posisi minimum dan maksimum dari posisi piksel pusat yang diidentifikasi sebagai *noise* dan nilai T sebagai selisih minimum dari nilai piksel pusat terhadap nilai tengah piksel dari *filtering window*. Baris 7 hingga Baris 11 adalah proses untuk mendapatkan posisi urutan dari piksel pusat terhadap seluruh piksel pada filtering window. Sedangkan Baris 12 hingga Baris 16 adalah langkah kerja dari algoritma ERID itu sendiri dimana jika piksel pusat diidentifikasi sebagai *noise* maka nilai kembali dari fungsi ERID adalah benar, sebaliknya jika piksel pusat diidentifikasi bukan *noise* maka nilai kembali dari fungsi ERID adalah salah.

4.2.1.3. RML-Filter

Setiap piksel yang diidentifikasi sebagai *noise*, maka metode RML-Filter digunakan untuk mengganti nilai piksel tersebut dengan nilai piksel yang baru sesuai hasil dari RML-Filter. Sebelum menjalankan program RML-Filter perlu ditetapkan terlebih dahulu jenis *influence function* yang digunakan dan nilai koefisien pembobotan yang digeneralisasi dari *probability density function* distribusi tertentu. Kode program untuk metode RML-Filter ditunjukkan pada Kode Sumber 4.4.

```
1.  function img_filtering =
    rml_filter(img_channel,row,col,sizeWin)
2.  img_channel_filter=zeros(row,col);
3.  a=[0.1296   0.1280   0.1249   0.1203   0.1146
    0.1077   0.1000   0.0917   0.0831];
4.  med_a=median(a);
5.  for x=1:size(img_channel,1)-2
6.     for y=1:size(img_channel,2)-2
7.        [window     idx]=window_filter(img_channel,
          sizeWin, x, y);
8.        res=zeros(size(window));
9.        ind=1;
10.       sorted_list=sort(window);
11.       med=sorted_list(idx/2);
12.       if ERID(window,sorted_list,idx) == 1
13.          for z=1:size(window)
14.             mmed=window(z)*(hampel(window(z)-
                med));
15.             res(ind)=mmed*a(z);
16.             ind=ind+1;
17.          end
18.          filter_res=round(median(res)/med_a);
19.          img_channel_filter(x,y)=filter_res;
20.       else
21.          img_channel_filter(x,y)=window(idx/2);
22.       end
23.    end
24. end
25. img_filtering=img_channel_filter;
```

Kode Sumber 4.4 Program Metode RML-Filter

Beberapa parameter awal perlu diperoleh sebelum memasuki proses utama RML-Filter. Parameter-parameter tersebut yaitu nilai tengah dari *filtering window* ditunjukkan pada Baris 11 dan vektor koefisien pembobotan beserta nilai tengah dari vektor tersebut. Nilai vektor ini dikomputasi secara terpisah dari program RML-Filter. Inisialisasi vektor koefisien pembobotan dan nilai tengah dari vektor tersebut ditunjukkan pada Baris 3 dan Baris 4. Baris 7 menunjukkan proses inisialisasi *filtering window* untuk memperoleh nilai dari piksel-piksel yang berada dalam *filtering window*.

Setelah piksel-piksel pada *filtering window* diperoleh, fungsi ERID dipanggil untuk memeriksa piksel pusat apakah terdegradasi *noise* atau tidak. Proses ini ditunjukkan pada Baris 12. Piksel pusat yang diidentifikasi sebagai *noise* kemudian diproses dengan RML-Filter. Proses utama dari RML-filter ditunjukkan pada Baris 13 hingga Baris 18. Sebelumnya disediakan terlebih dahulu vektor kosong untuk menampung nilai sementara dari hasil komputasi bagian persamaan $a(\widetilde{\psi}(X_i - med\{\bar{X}\}))$. Dari vektor yang diperoleh tersebut, maka nilai akhir RML-Filter merupakan piksel baru dari nilai tengah vektor tersebut dibagi dengan nilai tengah vektor koefisien pembobotan yang telah diinisialisasi sebelumnya.

4.2.1.3.1 Implementasi Generalisasi Koefisien Pembobotan

Pada subbab ini akan dibahas implementasi dari generalisasi koefisien pembobotan pada RML-Filter dimana menggunakan *probability density function* (PDF) dari suatu distribusi tertentu. Kode program untuk implementasi dari fungsi ini ditunjukkan pada Kode Sumber 4.5.

Pada Baris 2 merupakan proses penghitungan PDF setiap titik nilai yang diperoleh dari Baris 1 menggunakan distribusi normal dengan nilai mean 0 dan standar deviasi 0.5. Langkah berikutnya merupakan proses penghitungan nilai koefisien pembobotan sesuai Persamaan 2.6 dengan fungsi distribusi yang telah ditentukan.

```
1.  lamda=linspace(0,1,100000);
2.  hlamda=normpdf(lamda,0,0.5);
3.  denom=sum(hlamda);
4.  a=zeros(1,9);
5.  for i=1:9
6.      if i==1
7.          idx=find(lamda >= 0 & lamda <= 1/9);
8.      else
9.          idx=find(lamda > (i-1)/9 & lamda <= i/9);
10.     end
11.     temp=hlamda(idx);
12.     num=sum(temp);
13.     a(1,i)=num/denom;
14. end
```

Kode Sumber 4.5 Kode program implementasi dari Generalisasi Koefisien Pembobotan

Nilai hasil penghitungan disimpan pada suatu vektor dengan panjang sesuai besar *filtering window* yang digunakan. Dalam Tugas Akhir ini, ukuran *filtering window* yang digunakan adalah 3x3 sehingga inisialisasi panjang vektor adalah 9 sesuai jumlah piksel data yang ditampung. Proses ini dapat dilihat pada Baris 5 hingga Baris 13.

4.2.1.3.2 Implementasi Influence Function

Pada subbab ini akan dibahas implementasi dari *hampel influence function* dan *Andrew's Sine Influence Function* yang merupakan jenis *influence function* yang digunakan pada metode RML-Filter untuk Tugas Akhir ini. Kode program untuk implementasi dari fungsi-fungsi ini ditunjukkan pada Kode Sumber 4.6 dan Kode Sumber 4.7.

Proses awal yang dilakukan pada implementasi dari fungsi *Andrew's Sine* dan *hampel* yaitu inisialisasi parameter-parameter yang digunakan. Parameter-parameter pada fungsi *Andrew's Sine* yaitu $\propto$ dan π dan pada fungsi *hampel* yaitu $r, \propto,$ dan β sesuai persamaan fungsi masing-masing pada Tabel 2.1.

```
1.  function value=hampel(x)
2.  r=500;
3.  alpha=0.16*r;
4.  betha=0.8*r;
5.  if abs(x) <= alpha
6.    value=1;
7.  elseif abs(x) > alpha && abs(x) <= betha
8.    value=(alpha/abs(x));
9.  elseif abs(x) > betha && abs(x) <= r
10.   value=(alpha/abs(x))*((r-abs(x))/(r-
      betha));
11. else
12.   value=0;
13. end
```

Kode Sumber 4.6 Kode program untuk implementasi *hampel influence function*

```
1.  function value=andrew(x)
3.  r=35;
4.  phi=22/7;
6.  if abs(x) <= r*phi && x ~= 0
7.      value=sin(x/r)/(x/r);
8.  else
9.      value=0;
10. end
```

Kode Sumber 4.7 Kode program untuk implementasi *Andrew's Sine influence function*

Proses inisialisasi parameter-parameter tersebut ditujukkan pada Baris 1 hingga Baris 3. Proses berikutnya menunjukkan langkah dari penghitunan besar faktor pembobotan yang digunakan untuk proses dari M-Filter pada RML-Filter. Proses tersebut ditunjukkan pada Baris 4 hingga Baris 12.

4.2.2. Implementasi Algoritma FCCI

Pada bagian ini akan dijelaskan implementasi dari algoritma FCCI dalam melakukan segmentasi citra. Sebelum melakukan pemanggilan terhadap fungsi FCCI, beberapa tahapan segmentasi dilakukan terlebih dahulu sesuai perancangan perangkat lunak pada bab sebelumnya. Tahapan-tahapan tersebut meliputi konversi citra dari ruang warna RGB ke dalam ruang warna CIELAB, transformasi dari 2D ke dalam bentuk 1D, pencarian jumlah *cluster* yang tepat, menjalankan algoritma FCCI, dan defuzzifikasi. Kode program utama untuk mengimplementasikan tahapan-tahapan tersebut ditunjukkan pada Kode Sumber 4.8, Kode Sumber 4.9, dan Kode Sumber 4.10.

```
1.  function output=segmentation(img_filtering)
2.  image = img_filtering;
3.  (2)Konversi RGB to CIELAB
4.  labTransformation = makecform('srgb2lab');
5.  lab = applycform(image, labTransformation);
6.  m = size(image,1);
7.  n = size(image,2);
8.  image_cluster = zeros(m,n);
9.  N = m*n;% Number of data point of image
10. ab = zeros(m,n);
11. ab(:,:,1)=lab(:,:,2);
12. ab(:,:,2)=lab(:,:,3);
13. K = size(ab,3); % Number of Feature
14. % (3)Transformasi 2D ke 1D
15. D1 = reshape(ab(:,:,1)',1,N);
16. D2 = reshape(ab(:,:,2)',1,N);
17. Xji = [D1;D2]; % Xij = [D1;D2];
18. Xij = Xji';
19. % (4) Finding the number of cluster
20. % (4.1)
21. C =3;
```

Kode Sumber 4.8 Kode Sumber Algoritma program utama segmentasi (Bagian pertama)

```
22.  S_old = 50;
23.  Tu = 10;
24.  Tv = 9*10^7;
25.  t = 1;
26.  while (1)
27.      % (4.2)
28.      % Optimize Tu, Tv by Pso Algorithm
29.      fprintf('*------* Loop %d\n',t);
30.      if(t>1)
31.          fprintf('* Optimizing Tu, Tv using
     Bacterial Foraging Optimization\n');
32.          [Tu,              Tv]              =
     BacterialForagingAlgorithm(Uci,       Vcj,
     Dcij);
33.          fprintf('* Found Tu = %.f, Tv =
     %.f \n', Tu,Tv);
34.      end
35.      t = t+1;
36.  while (1)
37.      % (4.2)
38.      % Optimize Tu, Tv by Pso Algorithm
39.      fprintf('*------* Loop %d\n',t);
40.   % (4.3)
41.      fprintf('* Calculate Uci, Vcj, Pcj,
     and Dcij using FCCI Algorithm\n');
42.   % (4.4)
43.      fprintf('* Calculate Xie and Beni
     cluster validity S_new\n');
44.      S_new = clusterValidity(Uci,Xij,Pcj);
45.      fprintf('* S_new = %f\n',S_new);
46.   if (S_new > S_old)
47.          C = C-1;
48.          break
```

Kode Sumber 4.9 Kode Sumber Algoritma program utama segmentasi (Bagian kedua)

```
49.     else
50.         C = C+1;
51.         S_old = S_new;
52.     end
53. end
54. fprintf('\n');
55. fprintf('* Optimal Found for Tu = %.f, Tv = %.f \n', Tu,Tv);
56. fprintf('* Optimal Cluster = %d\n',C);
57. % (5)FCCI algorithm
58. fprintf('* Calculate Uci, Vcj, Pcj, and Dcij using FCCI Algorithm With optimal Cluster, Tu, and Tv\n');
59. [Uci, Vcj, Pcj, Dcij] = FCCI(C, N, K, Tu, Tv, Xij);
60  % (6) Defuzzify
61. [Uci_final Uci_idx]= max(Uci);
62. s=1;
63. for l=1:m
64.     image_cluster(l,:) = Uci_idx(s:s+(n-1));
65.     s=s+n;
66. end
67. % Menampilkan hasil
68. pixel_labels = reshape(image_cluster,m,n);
69. ei = imerode(pixel_labels,ones(3));
70. di = imdilate(pixel_labels,ones(3));
71. boundaries = ei~=pixel_labels | di~=pixel_labels;
72. output = imoverlay(image, boundaries, [1 1 1]);
73. figure, imshow(output)
74. % Liu and Yang accuration evaluation
75. [F] = accurationTest(Xij, image_cluster, Pcj, C, N)
```

Kode Sumber 4.10 Kode Sumber Algoritma program utama segmentasi (Bagian ketiga)

Proses pertama yang dilakukan pada Kode Sumber 4.8 adalah membaca data masukan dengan menggunakan fungsi MATLAB yang ditunjukkan pada baris Baris 1. Data masukan merupakan citra statis yang berukuran 321x481 atau 481x321. Kemudian dilanjutkan dengan proses melakukan konversi citra dari ruang warna RGB ke dalam ruang warna CIELAB yang ditunjukkan pada Baris 3 sampai Baris 12.

Proses selanjutnya merupakan proses transformasi dari 2D ke 1D yang ditunjukkan pada Baris 14 sampai dengan Baris 17. Kemudian, tahap selanjutnya adalah menemukan jumlah *cluster* yang sesuai untuk citra yang diproses. Tahap ini ditunjukkan pada Baris 20 sampai dengan Baris 54. Dari proses pencarian jumlah *cluster*, dihasilkan *cluster* yang tepat untuk sebuah citra. Setelah ditemukan jumlah *cluster* yang tepat, maka langkah selanjutnya adalah menjalankan algoritma FCCI dimana kode sumbernya ditunjukkan pada Baris 56 dan Baris 57. Setelah proses dari algoritma FCCI dijalankan maka langkah selanjutnya adalah melakukan defuzzifikasi yang kode sumbernya ditunjukkan pada Baris 59 sampai dengan Baris 64. Kemudian tahap terakhir adalah menampilkan kembali citra keluaran sebagai citra yang telah disegmentasi. Kode program untuk proses ini ditunjukkan pada Baris 66 sampai dengan Baris 71. Untuk mengetahui nilai dari evaluasi kuantitatif segmentasi dengan metode *Liu and Yang's* maka kode programnya ditunjukkan pada baris 72 dan baris 73.

Sedangkan detail kode program untuk fungsi FCCI ditunjukkan pada Kode Sumber 4.11 dan Kode Sumber 4.12.

Proses pertama yang dilakukan pada fungsi FCCI adalah melakukan inisialisasi parameter batas *error* maksimum dan jumlah iterasi dimana kode sumber dari proses ini dapat dilihat pada Baris 3 dan Baris 4. Kemudian langkah selanjutnya adalah melakukan inisialisasi untuk parameter U_{ci}, V_{cj}, P_{cj}, dan D_{cij} dengan menyediakan matriks kosong. Kode sumber untuk inisialisasi keempat parameter ini dapat dilihat pada Baris 5 sampai dengan Baris 8.

Setelah melakukan inisialisasi beberapa parameter di atas, langkah selanjutnya adalah melakukan inisialisasi untuk nilai awal dari parameter U_{ci}. Inisialisasi ini dilakukan secara random antara

nilai 0 sampai dengan 1. Hal ini dilakukan untuk memberikan nilai awal pada U_{ci}, dimana nantinya parameter ini akan terupdate setelah seluruh proses dalam algoritma dijalankan. Kode sumber untuk proses ini ada pada Baris 12 sampai dengan Baris 17.

```
1.  function [Uci, Vcj, Pcj, Dcij] = FCCI(C, N, K, Tu, Tv, Xij)
2.  (1) Initialize the parameters max error limit and max number of iteration
3.  maxError = 10^-2;
4.  maxIteration = 200;
5.  Uci = zeros(C,N);
6.  Vcj = zeros(C,K);
7.  Pcj = zeros(C,K);
8.  Dcij = zeros(C,N,K);
9.  (2) Set iteration number
10. it = 1;
11. % (3) Initialize Uci such that 0 <= Uci <= 1
12. for i=1:size(Uci,1)
13.     for j=1:size(Uci,2)
14.         Uci(i,j)=rand();
15.     end
16. end
17. Uci_bef = Uci;
18. REPEAT
19. while (true)
20. (5) Calculate Pcj
21.  sumUci = sum(Uci,2);
22.     Pcj = Uci * Xij;
23.     for j=1:K
24.         Pcj(:,j)= Pcj(:,j) ./ sumUci;
25.     end
26. (6) Calculate Dcij
```

Kode Sumber 4.11 Kode Sumber Algoritma FCCI (Bagian pertama)

```
27.   for i=1:N
28.           for c=1:C
29.               Dcij(c,i,:)=(Xij(i,:)-Pcj(c,:)).^2;
30.           end
31.       end
38.       end
39.       vcj2 = exp(-1*vcj2);
40.       vcj3 = sum(vcj2,2);
41.       for j=1:K;
42.           Vcj(:,j) = vcj2(:,j)./vcj3;
43.       end
44.   (8) Calculate Uci
45.   vd1 = zeros(K,N);
46.       vd3 = zeros(C,N);
47.       for c=1:C
48.           for j=1:K
49.               vd1(j,:) = Vcj(c,j).*Dcij(c,:,j)/Tu;
50.           end
51.           vd2 = sum(vd1);
52.           vd3(c,:) = vd2;
53.       end
54.       vd3 = exp(-1*vd3);
55.       vd4 = sum(vd3);
56.       for c=1:C
57.           Uci(c,:) = vd3(c,:)./vd4;
58.       end
60.    it=it+1;
61.       % (10) UNTIL max
62.       if (max(max(abs(Uci-Uci_bef))) <= maxError ||
      it==maxIteration)
63.           break
64.       end
65.       Uci_bef = Uci;
66.   End
```

Kode Sumber 4.12 Kode Sumber Algoritma FCCI (Bagian kedua)

Di dalam *while,* proses yang selanjutnya dilakukan adalah menghitung nilai pusat *cluster* c terhadap fitur *j* atau parameter p_{cj}. Dalam implementasi penghitungan nilai pusat *cluster* ini digunakan Persamaan 2.12. Kode sumber untuk proses ini berada pada Baris 21 sampai dengan Baris 25.

Setelah nilai pusat *cluster* didapatkan, selanjutnya dicari nilai dari *Euclidean distance* dari suatu titik data *i* terhadap *cluster c* dan fitur *j*. Untuk mencari nilai D_{cij} ini digunakan Persamaan 2.15. Kode sumber untuk proses ini ditunjukkan pada Baris 27 sampai dengan Baris 31.

Proses selanjutnya setelah nilai *Euclidean distance* ditemukan adalah mencari nilai fungsi keanggotaan fitur dari sebuah *cluster c* terhadap fitur *j*. Nilai v_{cj} ini dicari dengan menggunakan Persamaan 2.17. Kode sumber untuk proses ini ditunjukkan pada Baris 33 sampai dengan Baris 43.

Proses terakhir yang dilakukan adalah mencari nilai dari fungsi keanggotaan objek dari sebuah titik data *i* terhadap sebuah *cluster c*. Nilai dari u_{ci} ini nantinya akan menentukan sebuah titik data atau piksel tersebut masuk ke dalam *cluster* yang mana. Untuk menghitung nilai u_{ci} ini dalam implementasinya digunakan Persamaan 2.16, sedangkan untuk kode sumber dari proses ini dapat dilihat pada Baris 45 sampai dengan Baris 60. Setelah nilai u_{ci} yang baru didapatkan maka nilai u_{ci} yang lama hasil dari inisialisasi tadi akan diperbarui dengan u_{ci} yang dihasilkan dari proses berdasarkan algoritma FCCI.

4.2.3. Implementasi Evaluasi Kualitas *Filtering*

Pada bagian ini akan dijelaskan implementasi dari evaluasi kualitas filtering menggunakan PSNR pada citra RGB. Kode program untuk implementasi dari evaluasi kualitas filtering ditunjukkan pada Kode Sumber 4.13.

Pada kode sumber di atas, ukuran dari dua citra yang dibandingkan untuk memperoleh nilai PSNR harus memiliki ukuran citra yang sama. Syarat ini ditunjukkan pada Baris 2 hingga Baris 4.

```
1  function y=PSNR_RGB(X,Y)
2  if size(X)~=size(Y)
3     error('The images must have the same size');
4  end
5  if ~isa(X,'double')
6     X=double(X)./255.00;
7  end
8  if  ~isa(Y,'double')
9     Y=double(Y)./255.00;
10 end
11 d1=max(X(:));
12 d2=max(Y(:));
13 d=max(d1,d2);
14 sigma=mean2((X-Y).^2);
15 y=10*log((d.^2)./sigma);
```

Kode Sumber 4.13 Kode sumber evaluasi kualitas *filtering*

Nilai piksel dari citra yang dibandingkan dikonversi terlebih dahulu ke tipe *double* untuk mempermudah penghitungan matematika dari fungsi PSNR yang ditunjukkan pada Baris 5 hingga Baris 10. Proses untuk memperoleh nilai MAX_f ditunjukkan pada Baris 11 hingga Baris 13. Sedangkan nilai *MSE* yang diperoleh dari Persamaan 2.19 ditunjukkan pada Baris 14. Nilai akhir PSNR diperoleh sesuai Persamaan 2.18 dan ditunjukkan pada Baris 15.

4.2.4. Implementasi Evaluasi Kualitas *Cluster*

Pada bagian ini akan dijelaskan implementasi dari evaluasi kualitas *cluster* menggunakan *cluster* validity dari Xie dan Beni. Kode program untuk implementasi dari evaluasi kualitas filtering ditunjukkan pada Kode Sumber 4.14.

Pada kode sumber di atas merupakan implementasi untuk memperoleh nilai *S* sesuai Persamaan 2.20. Baris 4 hingga Baris 7 merupakan proses untuk memperoleh nilai d_{min} sedangkan untuk memperoleh nilai σ ditunjukkan pada Baris 15. Hasil akhir nilai *S* dari fungsi ini berikutnya akan dibandingkan dengan nilai *S* sebelumnya pada fungsi FCCI.

```
1.  function S =clusterValidity(Uci,Xij,Pcj)
2.  C = size(Pcj,1); % cluster
3.  N = size(Uci,2); % jml piksel
4.  Pcj_plus = Pcj(2:C,:);
5.  Pcj_plus(C,:) = Pcj(1,:);
6.  dmin = sum((Pcj_plus - Pcj).^2,2);
7.  dmin = min(dmin);
8.  Dci = zeros(C,N);
9.  for i=1:N
10.     for c=1:C
11.         Dci(c,i)=(Xij(i,1)-Pcj(c,1))^2;     % fitur j=1 pertama
12.         Dci(c,i)=         Dci(c,i)+(Xij(i,2)-Pcj(c,2))^2; % fitur j=2 kedua
13.     End
14. End
15. sigma = Uci.^2 .* Dci;
16. sigma = sum(sigma,2);
17. sigma = max(sigma);
18. S = (sigma/N)/dmin^2;
```

Kode Sumber 4.14 Implementasi *Cluster Validity*

4.2.5. Implementasi Evaluasi Kuantitatif Segmentasi Citra

Pada bagian ini akan dijelaskan implementasi dari evaluasi kuantitatif segmentasi citra yang diusulkan Liu dan Yang. Kode program dari implementasi metode evlauasi ini ditunjukkan pada Kode Sumber 4.15.

Pada kode sumber di atas, Baris 9 hingga Baris 11 merupakan proses untuk menghitung jarak antara piksel asli dengan piksel hasil segmentasi dimana banyak selisih jarak yang dihitung sesuai dengan banyak dimensi citra yang diuji. Nilai F merupakan hasil fungsi evaluasi kuantitatif segmentasi citra sesuai Persamaan 2.23.

```
1.  function [F] = accurationTest(Xij, image_cluster, Pcj, C, N)
2.  ei = zeros(C,1);
3.  Ai = zeros(C,1);
4.  G = C;
5.  for i=1:G
6.      Ai(i) = size(find(image_cluster==i),1);
7.      pixels = find(image_cluster==i);
8.      dist = zeros(Ai(i),1);
9.      for l=1:Ai(i)
10. dist(l,1) = sqrt((Pcj(i,1)-Xij(pixels(l),1))^2 + (Pcj(i,2)-Xij(pixels(l),2))^2);
11. End
12.     ei(i) = sum(dist);
13. End
14. F = (1/(1000*N))*sqrt(G)* sum(ei.^2./sqrt(Ai));
```

Kode Sumber 4.15 Kode Evaluasi Kuantitatif Segmentasi Citra

BAB V
PENGUJIAN DAN EVALUASI

Bab ini membahas pengujian dan evaluasi pada sistem yang dikembangkan. Pengujian yang dilakukan adalah pengujian terhadap penyaringan *noise* dengan estimasi robust spasial dan segmentasi citra hasil penyaringan *noise* menggunakan FCCI. Pembahasan pada bab ini meliputi lingkungan uji coba, hasil uji coba, dan evaluasi. Uji coba dari penyaringan *noise* pada citra dengan estimasi robust spasial dilakukan untuk membuktikan bahwa metode yang digunakan sudah benar sehingga dapat digunakan untuk memperbaiki kualitas dari citra yang akan digunakan untuk segmentasi.

5.1. Lingkungan Pengujian

Lingkungan uji coba yang digunakan dalam pembuatan Tugas Akhir ini meliputi perangkat lunak dan perangkat keras yang digunakan untuk melakukan uji coba metode FCCI dan metode *robust* spasial. Lingkungan pengujian merupakan komputer tempat uji coba sistem dengan spesifikasi yang dijelaskan sebagai berikut:

Prosesor	: Intel Core i3-3240 CPU @ 3.40GHz
Memori	: 4.00 GB
Jenis Device	: Personal Computer
Sistem Operasi	: Microsoft Windows 8 Enterprise 32 bit

5.2. Data Uji Coba

Tahap uji coba dari sistem ini menggunakan data citra dari *standart test images* untuk uji coba kebenaran dari proses penyaringan *noise* dengan RML-Filter dan citra dari *Barkeley image segmentation* (BSDS500) untuk uji coba kebenaran segmentasi citra dan uji kinerja segmentasi. Daftar citra yang digunakan untuk uji coba ditunjukkan pada Tabel 5.1dan Table 5.2.

Tabel 5.1 Daftar Citra untuk Uji Coba Sistem (Bagian pertama)

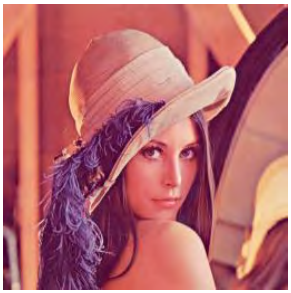
No	Citra Berwarna	Ukuran
1.	 lena_color_256.tif	256x256
2.	 295087.jpg	481x321
3.	 35058.jpg	481x321

Table 5.2 Daftar Citra untuk Uji Coba Sistem (Bagian kedua)

No	Citra Berwarna	Ukuran
4.	 124084.jpg	481x321
5.	 208001.jpg	321x481
6.	 198023.jpg	321x481

Table 5.3 Daftar Citra untuk Uji Coba Sistem (Bagian ketiga)

No	Citra Berwarna	Ukuran
4.	 169012.jpg	481x321
5.	 208001.jpg	321x481
6.	 230063.jpg	321x481

5.3. Skenario Pengujian

Uji coba ini dilakukan untuk menguji apakah fungsionalitas aplikasi telah diimplementasikan dengan benar dan berjalan sebagaimana mestinya. Uji coba akan didasarkan pada beberapa skenario untuk menguji kesesuaian dan kinerja aplikasi. Pada skenario uji coba ini akan menggunakan 4 citra yang diperoleh dari *Barkeley image segmentation* (BSDS500). Skenario uji kebenaran meliputi:

1. Uji kebenaran metode RML-Filter
2. Uji kebenaran metode segmentasi dengan FCCI dan estimasi *Robust* Spasial pada Citra dengan *Noise*

Skenario uji kinerja, antara lain:

1. Perbandingan hasil penyaringan *noise* dan segmentasi citra berdasarkan nilai standar deviasi (σ) dari distribusi normal yang digunakan untuk mengeneralisasi koefisien pembobotan pada RML-Filter yaitu untuk $\sigma=1$, $\sigma=0.5$, dan $\sigma=2$.
2. Perbandingan hasil penyaringan *noise* dan segmentasi citra berdasarkan jenis distribusi yang digunakan untuk mengeneralisasi koefisien pembobotan pada RML-Filter yaitu distribusi normal dan distribusi uniform.
3. Perbandingan hasil segmentasi citra berdasarkan nilai intensitas *noise* dari citra masukan yaitu pada intensitas 0.01, 0.05, dan 0.10.
4. Perbandingan hasil segmentasi citra berdasarkan jenis *noise* yang dikenakan pada citra masukan yaitu salt & pepper *noise*, gaussian *noise*, dan speckle *noise*.
5. Perbandingan hasil segmentasi citra berdasarkan inisialisasi jumlah *cluster* awal yaitu 3, 4, dan 5.

5.3.2. Uji Coba Kebenaran

Pada uji coba ini akan dilakukan pengujian terhadap kebenaran dan kesesuaian sistem yang telah dibuat. Uji coba kebenaran dilakukan untuk menunjukkan bahwa sistem telah berjalan sebagaimana mestinya. Uji coba kebenaran pertama dilakukan untuk menguji kualitas RML-Filter sebagai metode penyaringan *noise* pada

citra. Uji coba kebenaran berikutnya dilakukan untuk menguji kebenaran metode segmentasi citra yang digunakan yaitu FCCI dengan estimasi *robust* spasial pada kasus citra dengan *noise*.

5.3.2.1. Uji Kebenaran: Uji Kebenaran Metode RML-Filter

Pada skenario ini dilakukan uji coba kebenaran terhadap metode RML-Filter dengan menggunakan *Peak Signal-to-noise Ratio* (PSNR). Citra masukan pada uji coba yang dilakukan merupakan citra yang terdegradasi *salt & pepper noise* dengan intensitas *noise* yang diubah-ubah. Variasi intensitas *salt & pepper noise* yang diuji coba pada citra masukan yaitu 5%, 10%, 20%, 25%, 30%, 40%, dan 50%. RML-Filter yang digunakan dalam uji coba menggunakan *Andrew's Sine Influence Function* dan distribusi *uniform* sebagai distribusi pembangkit koefisien pembobotan dengan nilai $s=3$ dan $T=0.3 \; x \; med\{\bar{X}\}$ untuk Persamaan 2.8. Hasil uji coba skenario ditunjukkan pada Table 5.4.

Table 5.4 Tabel Uji Kebenaran Metode RML-Filter

Intensitas *Noise*	**Gallegos-Funes**	**Uji Coba**
0.05	31.56	41.55
0.10	29.56	34.56
0.20	26.45	26.91
0.25	25.01	24.52
0.30	23.44	22.33
0.40	20.10	19.10
0.50	16.91	16.98

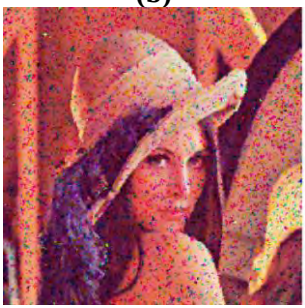
Hasil skenario uji coba kebenaran menunjukkan nilai PSNR yang diperoleh mendekati nilai PSNR Gallegos-Funes [6] pada setiap intensitas *noise* yang diuji. Hasil penyaringan *noise* dengan metode RML-Filter pada citra **lena_color_256.tif** ditunjukkan pada Gambar 5.1, Gambar 5.2, dan Gambar 5.3.

Gambar 5.1 Hasil uji coba kebenaran RML-Filter (bagian pertama) **(a)** Citra dengan intensitas *noise* 0.05 **(b)** Citra dengan intensitas *noise* 0.05 hasil RML-Filter **(c)** Citra dengan intensitas *noise* 0.10 **(d)** Citra dengan intensitas *noise* 0.10 hasil RML-Filter

(e)

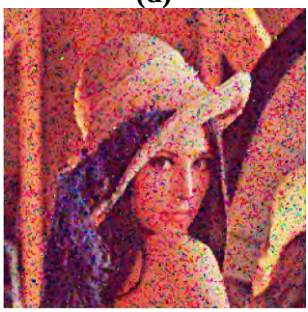
(b)

(c)



(d)

(e)



(f)

Gambar 5.2 Hasil uji coba kebenaran RML-Filter (bagian kedua) **(a)** Citra dengan intensitas *noise* 0.20 **(b)** Citra dengan intensitas *noise* 0.20 hasil RML-Filter **(c)** Citra dengan intensitas *noise* 0.25 **(d)** Citra dengan intensitas *noise* 0.25 hasil RML-Filter **(e)** Citra dengan intensitas *noise* 0.30 **(f)** Citra dengan intensitas *noise* 0.30 hasil RML-Filter

Gambar 5.3 Hasil uji coba kebenaran RML-Filter (bagian kedua) **(a)** Citra dengan intensitas *noise* 0.40 **(b)** Citra dengan intensitas *noise* 0.40 hasil RML-Filter **(c)** Citra dengan intensitas *noise* 0.50 **(d)** Citra dengan intensitas *noise* 0.50 hasil RML-Filter

5.3.2.2. Uji Kebenaran: Uji Kebenaran Metode Segmentasi dengan FCCI dan estimasi *Robust* Spasial pada Citra dengan *Noise*

Pada skenario ini dilakukan uji kebenaran terhadap metode segmentasi dengan FCCI dan estimasi *robust* Spasial pada citra dengan *noise*. Uji coba dilakukan pada citra **295087.jpg** dengan tiga kondisi yaitu segmentasi pada citra normal tanpa *noise*, segmentasi pada citra dengan *noise* tanpa estimasi *robust* spasial menggunakan RML-Filter, dan segmentasi pada citra dengan *noise* menggunakan

estimasi *robust* spasial yaitu RML-Filter. Jenis *noise* yang diuji coba adalah *salt & pepper* dengan intensitas 0.05. Hasil uji coba kebenaran ditunjukkan pada Tabel 5.5.

Tabel 5.5 Tabel Uji Kebenaran Metode Segmentasi dengan FCCI dan estimasi Robust Spasial pada Citra dengan *Noise*

Kategori	**Segmentasi pada citra normal**	**Segmentasi pada citra dengan *noise* (dengan RML-Filter)**	**Segmentasi pada citra dengan *noise* (tanpa RML-Filter)**
Jumlah *Cluster*	3	3	3
S (Validity Cluster)	0,000078	0,000088	0,000436
F(I)	449,416	456.155	563.303

Hasil skenario uji coba kebenaran menunjukkan dari ketiga kondisi yang diuji coba, segmentasi pada citra dengan *noise* menggunakan FCCI dan RML-Filter mendekati hasil *F(I)* dan *S (cluster validity)* dari segmentasi citra normal. Jumlah *cluster* dari segmentasi pada citra dengan *noise* menggunakan FCCI dan RML-Filter segmentasi citra normal juga menghasilkan jumlah yang sama. Sedangkan untuk segmentasi pada citra dengan *noise* tanpa RML-Filter, nilai *cluster validity* dan *F(I)* jauh dari hasil segmentasi pada citra normal. Hasil segmentasi dari ketiga kondisi tersebut ditunjukkan pada Gambar 5.4.

5.3.3. Uji Coba Kinerja Segmentasi

Pada bagian ini akan dijelaskan mengenai uji coba kinerja segmentasi yang telah dilakukan. Untuk menguji kinerja segmentasi digunakan ukuran kuantitatif dari Liu dan Yang dan ukuran tambahan yaitu PSNR untuk mengetahui hasil proses *filtering*.

(a)

(b)

(c)

Gambar 5.4 Hasil Uji Kebenaran Metode Segmentasi dengan FCCI dan estimasi Robust Spasial pada Citra dengan *noise* **(a)** Hasil segmentasi pada citra normal **(b)** Hasil segmentasi pada citra dengan *noise* menggunakan RML-Filter **(c)** Hasil segmentasi pada citra dengan *noise* tanpa RML-Filter

5.3.3.1. Uji Coba Skenario 1: Perbandingan hasil segmentasi citra berdasarkan nilai standar deviasi (σ)

Pada skenario ini dibandingkan pengaruh nilai standar deviasi (σ) dari distribusi normal yang merupakan parameter berpengaruh pada generalisasi koefisien pembobotan dari RML-Filter. Nilai standar deviasi (σ) yang diuji yaitu σ=0.5, σ=1 dan σ=2 dengan nilai *mean* (μ) adalah 0 dan bernilai tetap untuk setiap perubahan nilai σ. Hasil generalisasi dari nilai koefisien pembobotan pada RML-Filter dengan ukuran *filtering window* 3x3 ditunjukkan pada Tabel 5.6.

Tabel 5.6 Tabel Koefisien Pembobotan RML-Filter dengan variasi nilai standar deviasi (σ)

Koefisien Pembobotan RML-Filter dengan distribusi normal		
σ=0.5	**σ=1**	**σ=2**
0.1843	0.1296	0.1157
0.1754	0.1280	0.1153
0.1590	0.1249	0.1146
0.1372	0.1203	0.1136
0.1127	0.1146	0.1122
0.0881	0.1077	0.1105
0.0656	0.1000	0.1084
0.0465	0.0917	0.1061
0.0314	0.0831	0.1035

Uji coba dari skenario ini dilakukan pada lima citra yaitu 124084.jpg, 208001.jpg, 295087.jpg, 35058.jpg, dan 198023.jpg dengan jenis *noise* yang diberikan yaitu *salt & pepper* dengan intensitas 0.05. Uji coba dilakukan dengan menggunakan *Andrew's Sine Influence Function* untuk fungsi RML-Filter dengan nilai *s=1* dan *T=5* untuk Persamaan 2.8. Hasil uji skenario ditunjukkan pada Tabel 5.7.

Pada kelima citra yang diuji coba menunjukkan bahwa citra dimana proses RML-Filter menggunakan nilai σ=2 menghasilkan nilai *F(I)* dan PSNR yang paling baik jika dibandingkan segmentasi citra dimana proses RML-Filter menggunakan nilai μ dan σ yang lain.

Tabel 5.7 Tabel perbandingan hasil segmentasi citra berdasarkan nilai standar deviasi (σ)

No	Nama Citra	Nilai *mean* dan standar deviasi σ	F(I)	PSNR
1	124084.jpg	0.5	408.4789	38.1419
		1	397.2985	38.8871
		2	**392.6209**	**38.9341**
2	208001.jpg	0.5	182.5734	39.0070
		1	179.8914	39.9897
		2	**175.0692**	**40.4375**
3	295087.jpg	0.5	512.8746	38.2093
		1	495.3179	39.0938
		2	**493.7749**	**39.4275**
4	35058.jpg	0.5	44.6130	38.1724
		1	37.7165	38.8489
		2	**36.6297**	**39.1056**
5	198023.jpg	0.5	231.0143	37.4622
		1	215.7803	38.3282
		2	**213.4607**	**38.5451**

Hal ini dapat dilihat dari hasil uji coba bahwa nilai *F(I)* yang paling minimum untuk setiap citra yang diuji diperoleh dari nilai σ=2 serta nilai PSNR yang paling maksimum untuk setiap citra juga diperoleh dari nilai σ=2 . Nilai σ memberikan pengaruh pada hasil segmentasi maupun hasil *filtering* yang dapat dilihat dari hasil *F(I)* dan PSNR yang berbeda untuk setiap perubahan nilai μ dan σ. Hasil filtering dan segmentasi dari citra **208001.jpg** dapat dilihat pada Gambar 5.5 dan Gambar 5.6.

(a)

(b)

(c)

(d)

Gambar 5.5 Perbandingan citra 208001.jpg dengan *noise* dan hasil filtering menggunakan variasi nilai σ pada distribusi normal **(a)** Citra 208001.jpg dengan *noise* **(b)** Distribusi normal σ=1 **(c)** Distribusi normal σ=0.5 **(d)** Distribusi normal σ=2

(a)

(b)

(c)

Gambar 5.6 Perbandingan citra 208001.jpg hasil segmentasi menggunakan variasi nilai σ pada distribusi normal **(a)** Distribusi normal σ=1 **(b)** Distribusi normal σ=0.5 **(c)** Distribusi normal σ=2

5.3.3.2. Uji Coba Skenario 2: Perbandingan hasil segmentasi citra berdasarkan jenis distribusi yang digunakan

Pada skenario ini dibandingkan pengaruh jenis distribusi yang digunakan terhadap hasil penyaringan *noise* dan segmnetasi citra dimana jenis distribusi merupakan parameter berpengaruh pada generalisasi koefisien pembobotan dari RML-Filter. Jenis distribusi yang digunakan yaitu distribusi normal dan distribusi uniform. Untuk hasil uji coba dengan distribusi normal diambil dari hasil uji coba skenario 1 yang paling optimum yaitu hasil pada saat nilai σ=2. Tabel 5.8 menunjukkan hasil generalisasi koefisien pembobotan dengan distribusi uniform dengan menggunakan ukuran *filtering window* 3x3.

Tabel 5.8 Tabel Koefisien Pembobotan RML-Filter hasil generalisasi distribusi uniform

Koefisien Pembobotan RML-Filter dengan distribusi uniform								
0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11

Uji coba dari skenario ini dilakukan pada lima citra yaitu 124084.jpg, 208001.jpg, 295087.jpg, 35058.jpg, dan 198023.jpg dengan jenis *noise* yang diberikan yaitu *salt & pepper* dengan intensitas 0.05. Uji coba dilakukan dengan menggunakan *Andrew's Sine Influence Function* untuk fungsi RML-Filter dengan nilai *s=1* dan *T=5* untuk Persamaan 2.8. Hasil uji skenario ditunjukkan pada Table 5.9

Table 5.9 Tabel Perbandingan hasil segmentasi citra berdasarkan jenis distribusi (Bagian pertama)

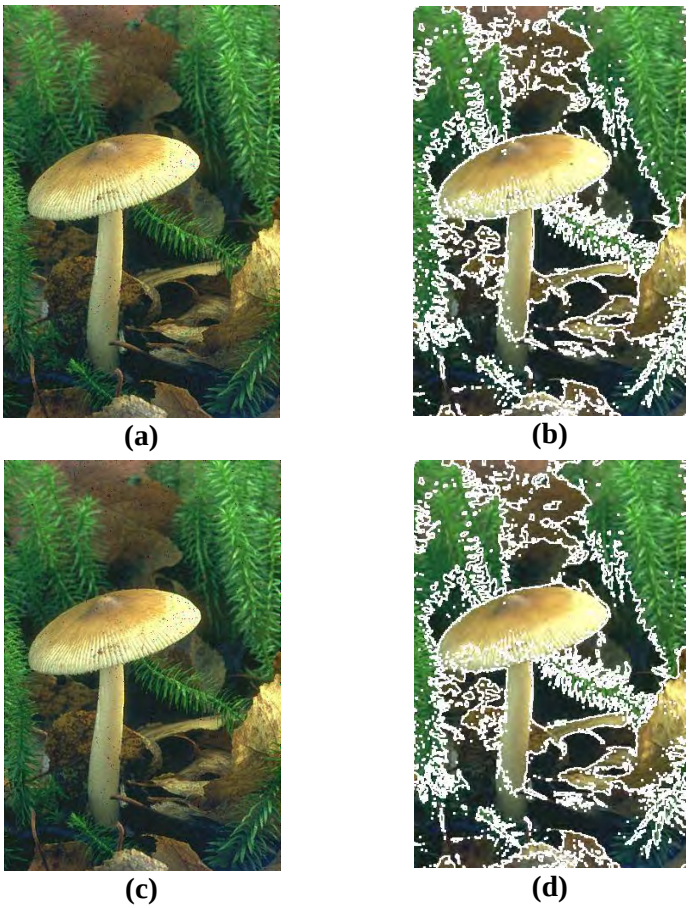
No	Nama Citra	Jenis distribusi	F(I)	PSNR
1	124084.jpg	uniform	393.4078	38.9036
		normal	**392.6209**	**38.9341**

Tabel 5.10 Tabel Perbandingan hasil segmentasi citra berdasarkan jenis distribusi (Bagian kedua)

No	Nama Citra	Jenis distribusi	F(I)	PSNR
2	208001.jpg	uniform	176.8003	40.2502
		normal	**175.0692**	**40.4375**
3	295087.jpg	uniform	494.1339	39.1764
		normal	**493.7749**	**39.4275**
4	35058.jpg	uniform	44.8239	39.0968
		normal	**36.6297**	**39.1056**
5.	198023.jpg	uniform	213.9353	38.5962
		normal	**213.4607**	**38.5451**

Pada kelima citra yang diuji coba menunjukkan bahwa jenis distribusi yang digunakan untuk mengeneralisasi koefisien pembobotan memberikan pengaruh terhadap hasil *F(I)* dari segmentasi dan PSNR dari hasil *filtering noise.* Pada sebagian besar citra yang diuji coba, hasil segmentasi dan *filtering* dimana RML-Filter menggunakan distribusi normal memberikan hasil yang lebih baik jika dibandingkan RML-Filter menggunakan distribusi uniform. Hal ini dapat dilihat dari nilai *F(I)* yang paling minimum dari sebagian besar citra diperoleh dari jenis distribusi yang digunakan adalah distribusi normal dan nilai PSNR yang maksimum dari sebagian besar citra juga diperoleh dari jenis distribusi yang digunakan adalah distribusi normal meskipun selisish hasil dari penggunaan kedua jenis ditribusi ini tidak terlalu besar. Hasil

filtering dan segmentasi dari citra **124084.jpg** dimana RML-Filter menggunakan distribusi normal ditunjukkan pada Gambar 5.7.



(a) **(b)**

(c) **(d)**

Gambar 5.7 Citra 208001.jpg hasil *filtering* dan segmentasi menggunakan distribusi uniform **(a)** Hasil *filtering* dari distribusi uniform **(b)** Hasil segmentasi dari distribusi uniform **(c)** Hasil *filtering* dari distribusi normal **(d)** Hasil segmentasi dari distribusi normal

5.3.3.3. Uji Coba Skenario 3: Perbandingan hasil segmentasi citra berdasarkan nilai intensitas *noise*

Pada skenario ini dibandingkan pengaruh intensitas *noise* dari citra masukan terhadap hasil segmentasi. Besar intensitas *noise* yang diuji coba yaitu 0.01, 0.05, dan 0.10 dengan jenis *salt & pepper noise*. Uji coba dari skenario ini dilakukan pada lima citra yaitu 124084.jpg, 208001.jpg, 295087.jpg, 35058.jpg, dan 198023.jpg. Uji coba dilakukan dengan menggunakan *Hampel Influence Function* untuk fungsi RML-Filter dan distribusi uniform untuk generalisasi koefisien pembobotan dengan nilai *s=1* dan *T=5* untuk Persamaan 2.8. Hasil uji skenario ditunjukkan pada Table 5.11.

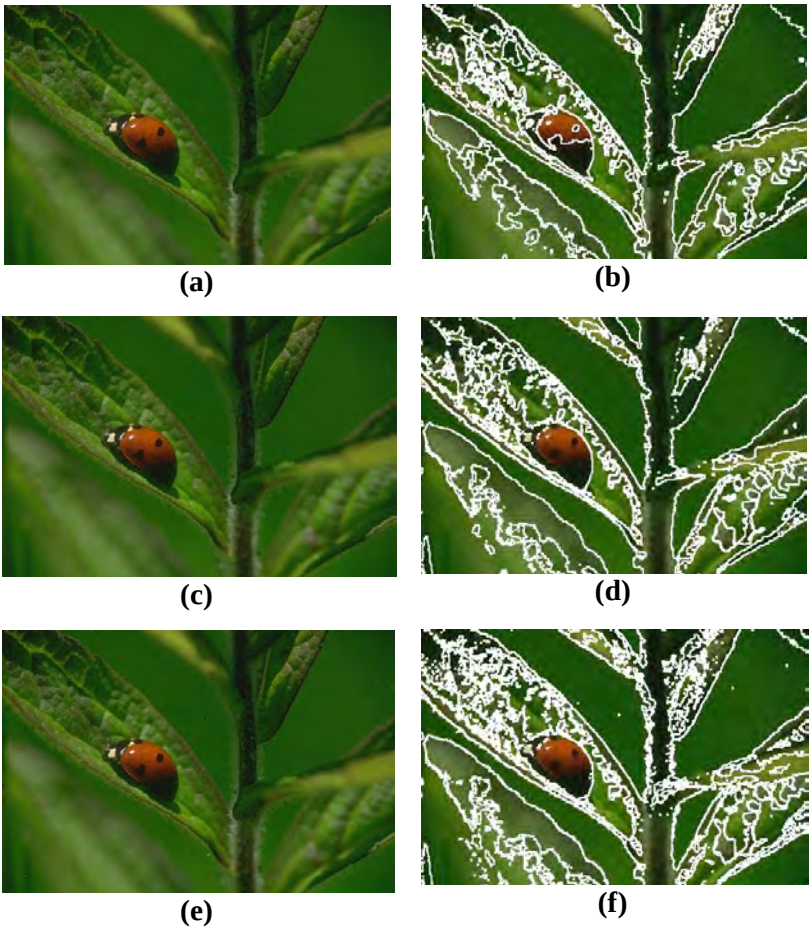
Table 5.11 Tabel Perbandingan hasil segmentasi citra berdasarkan intensitas *noise* yang diberikan (Bagian pertama)

No	Nama Citra	Intensitas *noise*	F(I)	PSNR	Waktu *filtering* (detik)	Waktu segmentasi (detik)
1	124084.jpg	0.01	**378.9194**	**55.1970**	**5.90**	**46.53**
		0.05	389.8487	39.4640	6.62	99.20
		0.10	430.3370	32.7605	7.17	430.34
2	208001.jpg	0.01	**161.2030**	**57.0234**	**6.35**	**37.37**
		0.05	163.7673	41.3569	6.95	58.08
		0.10	169.7851	34.4704	7.93	442.09
3	295087.jpg	0.01	**450.7113**	**55.7111**	**6.81**	**45.00**
		0.05	453.8066	41.3553	7.17	59.85
		0.10	461.0238	34.6637	7.51	63.78

Tabel 5.12 Tabel Perbandingan hasil segmentasi citra berdasarkan intensitas *noise* yang diberikan (Bagian kedua)

No	Nama Citra	Intensitas *noise*	F(I)	PSNR	Waktu *filtering* (detik)	Waktu segmentasi (detik)
4	35058.jpg	0.01	**33.8294**	**56.3609**	**5.88**	**46.71**
		0.05	34.5062	40.3169	6.57	72.27
		0.10	38.2877	33.4995	6.90	444.91
5	198023.jpg	0.01	**186.5383**	**55.0375**	**7.08**	**44.83**
		0.05	196.9855	40.7591	6.98	48.01
		0.10	208.6448	33.9511	7.84	448.90

Dari hasil uji coba, menunjukkan bahwa besar intensitas *noise* yang diberikan pada citra mempengaruhi hasil penyaringan *noise* dan segmentasi citra. Semakin besar nilai intensitas *noise* yang diberikan pada citra maka hasil penyaringan *noise* maupun segmentasi pada citra mengalami penurunan. Hal ini dapat dilihat dari nilai evaluasi kuantitaif dan PSNR dari setiap citra ketika nilai intensitas *noise* yang diberikan diubah. Hasil evaluasi kuantitaif dan PSNR terbaik diperoleh ketika intensitas *noise* yang diberikan sebesar 0.01. Waktu komputasi proses penyaringan *noise* mengalami peningkatan 7% dan segmentasi citra mengalami peningkatan 49% untuk peningkatan intensitas *noise* dari 0.01 ke 0.05. Sedangkan untuk peningkatan intensitas *noise* dari 0.05 ke 0.10, waktu komputasi proses penyaringan *noise* mengalami peningkatan 9% dan segmentasi citra mengalami peningkatan dua kali lipat. Hasil filtering dan segmentasi dari citra **35058.jpg** untuk setiap perubahan intensitas *noise* yang diberikan ditunjukkan pada Gambar 5.8.



Gambar 5.8 Hasil *filtering* dan segmentasi Citra 35058.jpg **(a)** Hasil *filtering* citra dengan intensitas *noise* 0.01 **(b)** Hasil segmentasi citra dengan intensitas *noise* 0.01 **(c)** Hasil *filtering* citra dengan intensitas *noise* 0.05 **(d)** Hasil segmentasi citra dengan intensitas *noise* 0.05 **(e)** Hasil *filtering* citra dengan intensitas *noise* 0.10 **(f)** Hasil segmentasi citra dengan intensitas *noise* 0.10

5.3.3.4. Uji Coba Skenario 4: Perbandingan hasil segmentasi citra berdasarkan jenis *noise*

Pada skenario ini dibandingkan pengaruh jenis *noise* yang dikenakan pada citra masukan terhadap hasil segmentasi. Jenis *noise* yang duji coba antara lain *salt & pepper noise*, *gaussian noise*, dan *speckle noise.* Uji coba dari skenario ini dilakukan pada lima citra yaitu 124084.jpg, 208001.jpg, 295087.jpg, 35058.jpg, dan 198023.jpg. Untuk hasil uji coba dengan *salt & pepper noise* menggunakan hasil uji coba dari skenario 3 dengan intensitas *noise* 0.01. Sedangkan uji coba dengan dua *noise* lain yaitu *gaussian noise* dan *speckle noise*, uji coba dilakukan dengan menggunakan *Hampel Influence Function* untuk fungsi RML-Filter dan distribusi uniform untuk generalisasi koefisien pembobotan dengan nilai *s=4* dan *T=5* untuk Persamaan 2.8. Intensitas *noise* dari speckle *noise* dan Gaussian *noise* yang diberikan adalah 0.01. Hasil uji skenario ditunjukkan pada Table 5.13.

Table 5.13 Tabel perbandingan hasil segmentasi citra berdasarkan jenis *noise* (Bagian pertama)

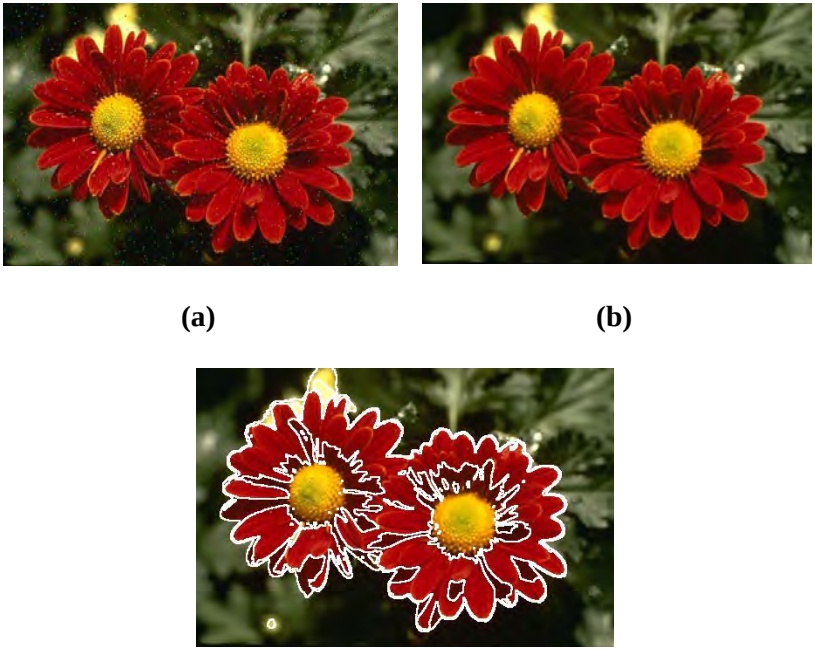
No	Nama Citra	Jenis *noise*	F(I)	PSNR	Waktu *filtering* (detik)	Waktu segmentasi (detik)
1	124084.jpg	salt & pepper	**378.9194**	**55.1970**	**5.90**	**46.53**
		speckle	395.2574	65.5533	9.98	52.32
		gaussian	424.1498	48.5619	16.13	74.15
2	208001.jpg	salt & pepper	**161.2030**	**57.0234**	**6.35**	**37.37**
		speckle	167.3008	62.1794	12.87	46.99
		gaussian	205.7091	46.3603	16.19	64.27

Tabel 5.14 Tabel perbandingan hasil segmentasi citra berdasarkan jenis *noise* (Bagian kedua)

No	Nama Citra	Jenis *noise*	F(I)	PSNR	Waktu *filtering* (detik)	Waktu segmentasi (detik)
3	295087.jpg	salt & pepper	**450.7113**	**55.7111**	**6.81**	**45.00**
		speckle	460.7139	58.4927	13.80	80.29
		gaussian	506.6587	45.7040	16.39	89.16
4	35058.jpg	salt & pepper	**33.8294**	**56.3609**	**5.88**	**46.71**
		speckle	35.9063	73.0033	10.29	54.29
		gaussian	55.1813	47.9665	15.88	60.36
5	198023.jpg	salt & pepper	**186.5383**	**55.0375**	**7.078**	**44.833**
		speckle	228.4033	53.8765	12.026	93.856
		gaussian	259.008	44.786	16.12	89.123

Perbedaan jenis *noise* yang diberikan pada citra memberikan pengaruh terhadap hasil dari hasil penyaringan *noise* dan segmentasi citra. Jika melihat hasil dari uji coba pada Table 5.13, nilai evalusi kualitatif terbaik dihasilkan dari segmentasi citra dengan salt and pepper *noise* sednagkan untuk nilai PSNR tertinggi diperoleh dari penyaringan *noise* pada citra dengan speckle *noise*. Sedangkan pada kasus citra dengan gaussian *noise*, hasil penyaringan *noise* maupun hasil segmnetasi citra menunjukkan hasil yang paling rendah jika dibandingkan dengan dua jenis *noise* yang lain. Untuk waktu komputasi, pada kasus citra dengan *salt and pepper noise* rata-rata waktu yang diperlukan untuk menyaring *noise* yaitu 6.40 detik dan rata-rata waktu segmentasi yaitu 44.09 detik. Pada kasus *speckle noise* rata-rata waktu yang diperlukan untuk menyaring *noise* yaitu

11.79 detik dan rata-rata waktu segmentasi yaitu 65.55 detik. Pada kasus *gaussian noise* rata-rata waktu yang diperlukan untuk menyaring *noise* yaitu 16.14 dan rata-rata waktu segmentasi yaitu 75.41 detik. Hasil filtering dan segmentasi dari citra **124084.jpg** untuk setiap perubahan jenis *noise* yang diberikan ditunjukkan pada Gambar 5.9, Gambar 5.10, dan Gambar 5.12.



(a) **(b)**

(c)

Gambar 5.9 Hasil penyaringan *noise* dan segmentasi citra 124084.jpg dengan *salt and pepper noise* **(a)** Citra 124084.jpg dengan salt and pepper *noise* **(b)** Citra 124084.jpg hasil penyaringan *noise* **(c)** Citra 124084.jpg hasil segmentasi

(a)

(b)

(c)

Gambar 5.10 Hasil penyaringan *noise* dan segmentasi citra 124084.jpg dengan speckle *noise* **(a)** Citra 124084.jpg dengan speckle *noise* **(b)** Citra 124084.jpg hasil penyaringan *noise* **(c)** Citra 124084.jpg hasil segmentasi

(a)

(b)

Gambar 5.11 Hasil penyaringan *noise* dan segmentasi citra 124084.jpg dengan gaussian *noise* (Bagian pertama) **(a)** Citra 124084.jpg dengan gaussian *noise* **(b)** Citra 124084.jpg hasil penyaringan *noise*

(c)

Gambar 5.12 Hasil penyaringan *noise* dan segmentasi citra 124084.jpg dengan gaussian *noise* (Bagian kedua) **(c)** Citra 124084.jpg hasil segmentasi

5.3.3.5. Uji Coba Skenario 5: Perbandingan hasil segmentasi citra berdasarkan inisialisasi jumlah *cluster* awal

Pada skenario ini dibandingkan pengaruh inisialisasi awal jumlah *cluster* terhadap hasil segmentasi. Inisialisasi jumlah *cluster* yang dibandingkan untuk diuji coba yaitu 3,4,dan 5. Uji coba dari skenario ini dilakukan pada lima citra yaitu 124084.jpg, 208001.jpg, 295087.jpg, 35058.jpg, dan 198023.jpg dengan jenis *noise* yang diberikan yaitu *salt & pepper* dengan intensitas 0.05. Uji coba dilakukan dengan menggunakan *Andrew's Sine Influence Function* untuk fungsi RML-Filter dengan nilai *s=1* dan *T=5* untuk Persamaan 2.8. Hasil uji skenario ditunjukkan pada Tabel 5.15.

Untuk memastikan pengaruh dari inisialisasi jumlah cluster terhadap hasil segmentasi citra dengan noise, dilakukan uji coba tambahan terhadap tiga citra yang mempunyai distribusi warna lebih beragam yaitu 157087.jpg, 230063.jpg, dan 169012.jpg. Uji coba dilakukan pada kondisi optimal sesuai uji skenario yang telah dilakukan sebelumnya yaitu dengan mengunakan *Hampel Influence Function* untuk fungsi RML-Filter dengan nilai *s=1* dan *T=5* untuk Persamaan 2.8. Hasil uji skenario ditunjukkan pada Tabel 5.16.

Tabel 5.15 Tabel perbandingan hasil segmentasi citra berdasarkan jumlah *cluster* (Bagian pertama)

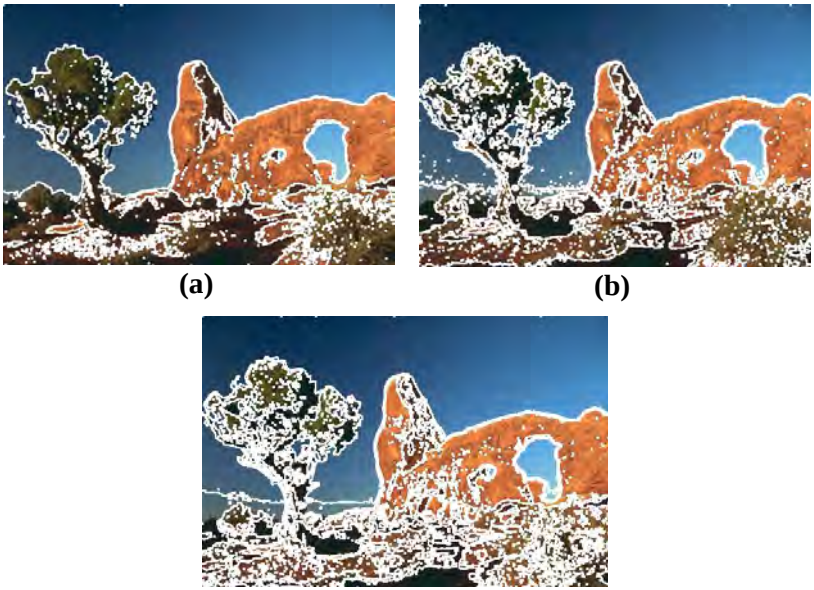
No	Nama Citra	Jumlah *cluster*	F(I)	Waktu segmentasi (detik)
1	124084.jpg	3	**400.0085**	**63.79**
		4	400.2747	84.37
		5	425.8118	145.33
2	208001.jpg	3	**178.0011**	**126.76**
		4	189.0544	447.88
		5	190.4747	621.11
3	295087.jpg	3	**504.6360**	**91.23**
		4	519.9796	99.60
		5	561.0096	153.29
4	35058.jpg	3	**50.6393**	**52.80**
		4	56.2820	195.51
		5	64.2757	300.03
5	198023.jpg	3	**207.6308**	**50.57**
		4	220.9948	86.99
		5	234.9930	546.32

Dari hasil uji coba yang diperoleh menunjukkan bahwa segmentasi citra berwarna dengan *noise* menggunakan FCCI dan estimasi *robust* spasial memberikan pengaruh terhadap hasil segmentasi citra. Hal ini dapat dilihat dari nilai evaluasi kuantitaif

yang diperoleh. Hasil evaluasi kuantitatif terkecil dan waktu komputasi terpendek diperoleh ketika inisialisasi awal *cluster* paling kecil yaitu tiga *cluster*. Waktu komputasi yang digunakan untuk melakukan segmentasi citra dari inisialisasi *cluster* awal 3 dan inisialisasi *cluster* awal 4 mengalami kenaikan sebesar dua kali lipat sedangkan kenaikan waktu komputasi ketika inisialisasi awal *cluster* 4 dengan inisialisasi awal *cluster* 5 sebesar 55%. segmentasi dari citra **295087.jpg** untuk setiap perubahan inisialisasi awal *cluster* yang diberikan ditunjukkan pada Gambar 5.13.

Tabel 5.16 Tabel perbandingan hasil segmentasi citra berdasarkan jumlah *cluster* (Bagian kedua)

No	Nama Citra	Jumlah *cluster*	F(I)	Waktu segmentasi (detik)
6	157087.jpg	3	**205.8292**	**69.93**
		4	208.9170	101.86
		5	225.0229	138.04
7	230063.jpg	3	**166.2520**	**61.24**
		4	173.7116	120.00
		5	183.5829	194.45
8	169012.jpg	3	**118.4513**	**71.04**
		4	142.9718	104.86
		5	164.1577	292.95



(a) **(b)**

(c)

Gambar 5.13 Hasil segmentasi citra 295087.jpg dengan perubahan inisialisasi *cluster* awal **(a)** Citra 295087.jpg dengan inisialisasi 3 *cluster* **(b)** Citra 295087.jpg dengan inisialisasi 4 *cluster* **(c)** Citra 295087.jpg dengan inisialisasi 5 *cluster*

BAB VI
PENUTUP

Pada bab ini akan diberikan kesimpulan yang diambil selama pengerjaan tugas akhir serta saran-saran tentang pengembangan yang dapat dilakukan terhadap tugas akhir ini di masa yang akan datang.

6.1. Kesimpulan

Dari hasil selama proses perancangan, implementasi, serta pengujian dapat diambil kesimpulan sebagai berikut:

1. Segmentasi citra berwarna dengan menggunakan FCCI dan estimasi *robust* spasial pada kasus citra dengan *noise* memberikan hasil evaluasi kuantitatif yang lebih baik jika dibandingkan segmentasi citra berwarna hanya menggunakan FCCI tanpa estimasi *robust* spasial.
2. Hasil segmentasi citra dengan menggunakan FCCI dan estimasi *robust* spasial menunjukkan hasil segmentasi citra yang baik dengan menghasilkan nilai PSNR yang relatif tinggi, evaluasi kuantitatif yang relatif kecil dan rata-rata *error color* kurang dari 1.15%.
3. Hasil penyaringan *noise* dan segmentasi citra menurut uji coba dipengaruhi oleh nilai standar deviasi dan jenis distribusi yang digunakan untuk generalisasi koefisien pembobotan pada metode RML-Filter serta intensitas dan jenis *noise* dari citra masukan.
4. Pengaruh perubahan nilai standar deviasi dan jenis distribusi yang digunakan untuk generalisasi koefisien pembobotan pada metode RML-Filter terhadap hasil penyaringan *noise* dan segmentasi citra dapat dilihat dari nilai PSNR dan nilai evaluasi kuantitatif.
5. Pengaruh perubahan nilai standar deviasi dan jenis distribusi yang digunakan untuk generalisasi koefisien pembobotan pada metode RML-Filter terhadap hasil penyaringan *noise* dan

segmentasi citra dapat dilihat dari nilai PSNR, nilai evaluasi kuantitatif dan waktu komputasi yang dibutuhkan untuk proses penyaringan *noise* maupun segmentasi.

6. Inisialisasi jumlah *cluster* awal untuk segmentasi citra dengan *noise* mempengaruhi waktu komputasi dari proses segmentasi. Semakin banyak inisialisasi jumlah *cluster*, maka waktu yang diperlukan untuk proses segmentasi semakin lama. Nilai evaluasi kuantitatif juga mengalami kenaikan ketika inisialisasi jumlah *cluster* ditambah.

6.2. Saran

Berikut saran-saran untuk pengembangan dan perbaikan sistem di masa yang akan datang. Diantaranya adalah sebagai berikut:

1. Perlu dikembangkan metode yang dapat mengeneralisasi parameter-parameter yang dibutuhkan secara otomatis, sehingga pengguna tidak perlu melakukan pencarian nilai parameter yang dapat menghasilkan jumlah *cluster* yang tepat.
2. Perlu dilakukan uji coba yang lebih variatif dan mendalam untuk mengetahui hasil segmentasi citra yang lebih optimal berdasarkan metode FCCI dan estimasi *robust* spasial.

LAMPIRAN

Tabel A. 1 Perbandingan hasil *filtering* citra berdasarkan nilai *mean* (μ) dan standar deviasi (σ) untuk citra 124084.jpg

 μ=0;σ=1 PSNR= 38.6605	 μ=0.5;σ=0.5 PSNR= 38.6399
 μ=0.5;σ=2 PSNR= 38.9085	

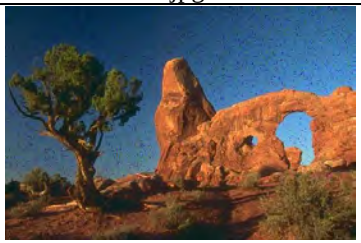
Tabel A. 2 Perbandingan hasil segmentasi citra berdasarkan nilai *mean* (μ) dan standar deviasi (σ) untuk citra 124084.jpg (Bagian pertama)

 μ=0;σ=1 F(I)= 403.4042	 μ=0.5;σ=0.5 F(I)= 411.9663

Tabel A. 3 Perbandingan hasil segmentasi citra berdasarkan nilai *mean* (μ) dan standar deviasi (σ) untuk citra 124084.jpg (Bagian kedua)


μ=0.5;σ=2 F(I)= 399.8014

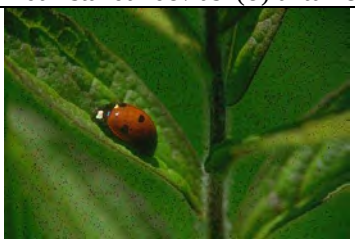
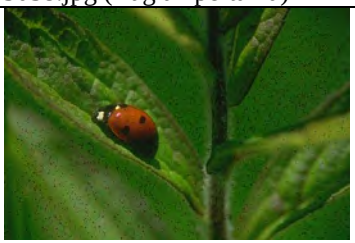
Tabel A. 4 Perbandingan hasil *filtering* citra berdasarkan nilai *mean* (μ) dan standar deviasi (σ) untuk citra 295087.jpg

	
μ=0;σ=1 PSNR= 39.0467	μ=0.5;σ=0.5 PSNR= 39.0740
	
μ=0.5;σ=2 PSNR= 39.2095	

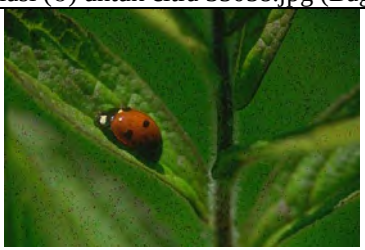
Tabel A. 5 Perbandingan hasil segmentasi citra berdasarkan nilai *mean* (μ) dan standar deviasi (σ) untuk citra 295087.jpg

 μ=0;σ=1 F(I)= 507.5160	 μ=0.5;σ=0.5 F(I)= 508.7464
 μ=0.5;σ=2 F(I)= 505.2591	

Tabel A. 6 Perbandingan hasil *filtering* citra berdasarkan nilai *mean* (μ) dan standar deviasi (σ) untuk citra 35058.jpg (Bagian pertama)

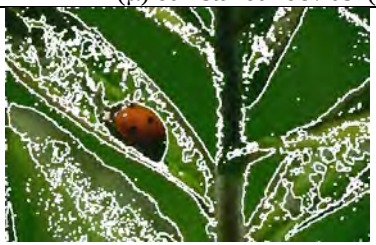
 μ=0;σ=1 PSNR= 38.8637	 μ=0.5;σ=0.5 PSNR= 38.8306

Tabel A. 7 Perbandingan hasil *filtering* citra berdasarkan nilai *mean* (μ) dan standar deviasi (σ) untuk citra 35058.jpg (Bagian kedua)



μ=0.5;σ=2
PSNR = 39.0621

Tabel A. 8 Perbandingan hasil segmentasi citra berdasarkan nilai *mean* (μ) dan standar deviasi (σ) untuk citra 35058.jpg



μ=0;σ=1
F(I)= 45.5043

μ=0.5;σ=0.5
F(I)= 47.2046

μ=0.5;σ=2
F(I)= 45.1126

Tabel A. 9 Perbandingan hasil *filtering* citra berdasarkan nilai *mean* (μ) dan standar deviasi (σ) untuk citra 198023.jpg

 μ=0;σ=1 PSNR= 38.3850	 μ=0.5;σ=0.5 PSNR= 38.3867
 μ=0.5;σ=2 PSNR= 38.6445	

Tabel A. 10 Perbandingan hasil segmentasi citra berdasarkan nilai *mean* (μ) dan standar deviasi (σ) untuk citra 198023.jpg

 μ=0;σ=1 F(I)= 215.4869	 μ=0.5;σ=0.5 F(I)= 215.3567
 μ=0.5;σ=2 F(I)= 205.5305	

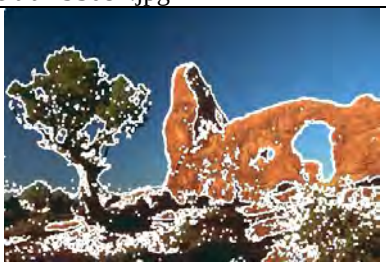
Tabel A. 11 Perbandingan hasil *filtering* dan segmentasi citra berdasarkan jenis distribusi untuk citra 124084.jpg

 Distribusi uniform PSNR= 38.7581	 Distribudi normal PSNR= 38.9085
 Distribusi uniform F(I)= 400.5481	 Distribudi normal F(I)= 399.8014

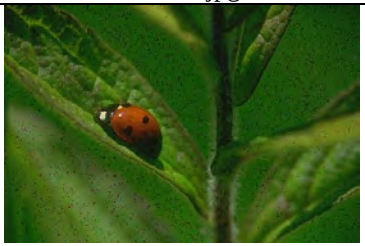
Tabel A. 12 Perbandingan hasil *filtering* citra berdasarkan jenis distribusi untuk citra 295087.jpg

 Distribusi uniform PSNR= 39.1676	 Distribudi normal PSNR= 39.1765

Tabel A. 13 Perbandingan hasil segmentasi citra berdasarkan jenis distribusi untuk citra 295087.jpg

 Distribusi uniform F(I)= 505.2064	 Distribudi normal F(I)= 505.0626

Tabel A. 14 Perbandingan hasil *filtering* dan segmentasi citra berdasarkan jenis distribusi untuk citra 35058.jpg

 Distribusi uniform PSNR= 39.1481	 Distribudi normal PSNR= 39.0621
 Distribusi uniform F(I)= 45.2291	 Distribudi normal F(I)= 45.1126

Tabel A. 15 Perbandingan hasil *filtering* dan segmentasi citra berdasarkan jenis distribusi untuk citra 198023.jpg

 Distribusi uniform PSNR= 39.1481	 Distribudi normal PSNR= 39.0621
 Distribusi uniform F(I)= 45.2291	 Distribudi normal F(I)= 45.1126

Tabel A. 16 Perbandingan hasil segmentasi citra berdasarkan berdasarkan intensitas *noise* yang diberikan pada citra 124084.jpg

 Intensitas *noise*=0.01 F(I)= 378.9194	 Intensitas *noise*=0.05 F(I)= 389.8487
 Intensitas *noise*=0.1 F(I)= 430.3370	

Tabel A. 17 Perbandingan hasil segmentasi citra berdasarkan berdasarkan intensitas *noise* yang diberikan pada citra 208001.jpg (Bagian pertama)

 Intensitas *noise*=0.01 F(I)= 161.2030	 Intensitas *noise*=0.05 F(I)= 163.7673

Tabel A. 18 Perbandingan hasil segmentasi citra berdasarkan berdasarkan intensitas *noise* yang diberikan pada citra 208001.jpg (Bagian kedua)

 Intensitas *noise*=0.1 F(I)= 169.7851

Tabel A. 19 Perbandingan hasil segmentasi citra berdasarkan berdasarkan intensitas *noise* yang diberikan pada citra 295087.jpg

 Intensitas *noise*=0.01 F(I)= 450.7113	 Intensitas *noise*=0.05 F(I)= 453.8066
 Intensitas *noise*=0.1 F(I)= 461.0238	

Tabel A. 20 Perbandingan hasil segmentasi citra berdasarkan berdasarkan intensitas *noise* yang diberikan pada citra 198023.jpg

Intensitas *noise*=0.01
F(I)= 186.5383

Intensitas *noise*=0.05
F(I)= 196.9855

Intensitas *noise*=0.1
F(I)= 208.6448

Tabel A. 21 Perbandingan hasil segmentasi citra berdasarkan berdasarkan jenis *noise* yang diberikan pada citra 295087.jpg

 Salt & pepper noise F(I)= 450.7113	 Speckle noise F(I)= 460.7139
 Gaussian noise F(I)= 506.6587	

Tabel A. 22 Perbandingan hasil segmentasi citra berdasarkan berdasarkan jenis *noise* yang diberikan pada citra 295087.jpg

 Salt & pepper noise F(I)= 161.2030	 Speckle noise F(I)= 167.3008
 Gaussian noise F(I)= 205.7091	

Tabel A. 23 Perbandingan hasil segmentasi citra berdasarkan berdasarkan jenis *noise* yang diberikan pada citra 35058.jpg (Bagian pertama)

 Salt & pepper noise F(I)= 33.8294	 Speckle noise F(I)= 35.9063

Tabel A. 24 Perbandingan hasil segmentasi citra berdasarkan berdasarkan jenis noise yang diberikan pada citra 35058.jpg (Bagian kedua)

 Gaussian noise F(I)= 55.1813

Tabel A. 25 Perbandingan hasil segmentasi citra berdasarkan berdasarkan jenis *noise* yang diberikan pada citra 198023.jpg (Bagian pertama)

 Cluster=3 F(I)= 186.5383	 Cluster=4 F(I)= 228.4033

Tabel A. 26 Perbandingan hasil segmentasi citra berdasarkan berdasarkan jenis *noise* yang diberikan pada citra 198023.jpg (Bagian kedua)

Gaussian noise
F(I)=259.008

Tabel A. 27 Perbandingan hasil segmentasi citra berdasarkan berdasarkan jumlah *cluster* yang diberikan pada citra 124084.jpg (Bagian pertama)

Cluster 3
F(I)= 400.0085

Cluster 4
F(I)= 400.2747

Tabel A. 28 Perbandingan hasil segmentasi citra berdasarkan berdasarkan jenis *noise* yang diberikan pada citra 124084.jpg (Bagian kedua)

 Cluster 5 F(I)= 425.8118

Tabel A. 29 Perbandingan hasil segmentasi citra berdasarkan berdasarkan jenis *noise* yang diberikan pada citra 208001.jpg (Bagian pertama)

 Cluster 3 F(I)= 178.0011	 Cluster 4 F(I)= 189.0544

Tabel A. 30 Perbandingan hasil segmentasi citra berdasarkan berdasarkan jenis *noise* yang diberikan pada citra 208001.jpg (Bagian kedua)

Cluster 5
F(I)= 190.4747

Tabel A. 31 Perbandingan hasil segmentasi citra berdasarkan berdasarkan jenis *noise* yang diberikan pada citra 35058.jpg

Cluster 3
F(I)= 50.6393

Cluster 4
F(I)= 56.2820

Cluster 5
F(I)= 64.2757

Tabel A. 32 Perbandingan hasil segmentasi citra berdasarkan berdasarkan jenis *noise* yang diberikan pada citra 198023.jpg

 Cluster 3 F(I)= 207.6308	 Cluster 4 F(I)= 220.9948
 Cluster 5 F(I)= 234.993	

Tabel A. 33 Perbandingan hasil segmentasi citra berdasarkan berdasarkan jumlah *cluster* yang diberikan pada citra 157087.jpg

 Cluster 3 F(I)= 205.8292	 Cluster 4 F(I)= 208.9170
 Cluster 5 F(I)= 225.0229	

Tabel A. 34 Perbandingan hasil segmentasi citra berdasarkan berdasarkan jumlah *cluster* yang diberikan pada citra 169012.jpg

 Cluster 3 F(I)= 166.2520	 Cluster 4 F(I)= 173.7116

Tabel A. 35 Perbandingan hasil segmentasi citra berdasarkan berdasarkan jumlah *cluster* yang diberikan pada citra 157087.jpg

Cluster 5
F(I)= 183.5829

Tabel A. 36 Perbandingan hasil segmentasi citra berdasarkan berdasarkan jenis *noise* yang diberikan pada citra 230063.jpg

Cluster 3
F(I)= 118.4513

Cluster 4
F(I)= 142.9718

Cluster 5
F(I)= 164.1577

BIODATA PENULIS

Penulis, Hardika Khusnuliawati, lahir di kota Sukoharjo pada tanggal 31 Agustus 1992. Penulis adalah anak pertama dari tiga bersaudara dan dibesarkan di kota Sukoharjo, Jawa Tengah.

Penulis menempuh pendidikan formal di SD Negeri Tawang 1 (1998-2004), SMPN 1 Tawangsari (2004-2007), SMAN 1 Sukoharjo (2007-2010). Pada tahun 2010, penulis memulai pendidikan S1 jurusan Teknik Informatika Fakultas Teknologi Informasi di Institut Teknologi Sepuluh Nopember Surabaya, Jawa Timur.

Di jurusan Teknik Informatika, penulis mengambil bidang minat Komputasi Cerdas dan Visualsiasi. Penulis juga aktif dalam organisasi kemahasiswaan seperti Himpunan Mahasiswa Teknik Computer (HMTC) sebagai staf dan Keluarga Muslim Informatika sebagai sekretaris departemen. Dan penulis juga beberapa kali menjadi asisten dosen di Teknik Informatika, diantaranya Asisten Dosen mata kuliah Teori Graph dan Otomata. Penulis dapat dihubungi melalui alamat email har.dika021@gmail.com