OPTIMASI TEKNIK *FOREGROUND EXTRACTION* MENGGUNAKAN METODE *GRABCUT*

SKRIPSI

Sebagai salah satu syarat untuk memperoleh gelar sarjana dalam bidang Ilmu Komputer

oleh:

MIRZA GALIH KURNIAWAN

0610960043-96

PROGRAM STUDI ILMU KOMPUTER

JURUSAN MATEMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS BRAWIJAYA

MALANG

2011


UNIVERSITAS BRAWIJAYA

LEMBAR PENGESAHAN TUGAS AKHIR

OPTIMASI TEKNIK *FOREGROUND EXTRACTION* MENGGUNAKAN METODE *GRABCUT*

Oleh:

MIRZA GALIH KURNIAWAN
0610960043-96

Setelah dipertahankan di depan Majelis Penguji

Pada tanggal 20 Juli 2011

dan dinyatakan memenuhi syarat untuk memperoleh gelar Sarjana dalam bidang Ilmu Komputer

Pembimbing I	Pembimbing II
Drs. Marji, MT	Edy Santoso, S.Si., M.Kom
NIP. 196708011992031001	NIP. 197404142003121004

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Dr. Abdul Rouf Alghofari, M.Sc
NIP. 19670907 199203 1 001


UNIVERSITAS BRAWIJAYA

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama : Mirza Galih Kurniawan

NIM : 0610960043-96

Jurusan : Matematika

Penulis tugas Akhir berjudul : OPTIMASI TEKNIK *FOREGROUND EXTRACTION* MENGGUNAKAN METODE *GRABCUT*

Dengan ini menyatakan bahwa :

1. Isi dari tugas Akhir yang saya buat adalah benar-benar karya sendiri dan tidak menjiplak karya orang lain, selain nama-nama yang termaktub di isi dan tertulis di daftar pustaka dalam Tugas Akhir ini.
2. Apabila dikemudian hari ternyata Tugas Akhir yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menanggung segala resiko yang akan saya terima.

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, 20 Juli 2011

Yang menyatakan,

(Mirza Galih Kurniawan)

NIM. 0610960043-96

OPTIMASI TEKNIK *FOREGROUND EXTRACTION* MENGGUNAKAN METODE *GRABCUT*

ABSTRAK

Pada suatu citra digital, *foreground* merupakan suatu objek yang ditentukan sebagai objek utama oleh seseorang yang melihat citra digital tersebut. Sedangkan *background* adalah semua bagian gambar yang tidak termasuk dalam bagian *foreground*. Secara alamiah, manusia dapat dengan mudah membedakan antara *foreground* dan *backround,* berbeda halnya dengan komputer. Hingga saat ini berbagai metode dikembangkan untuk dapat menemukan metode yang efektif dan dapat diterapkan dalam komputer untuk segmentasi *foreground*. Beberapa metode yang ada seperti *Magic Wand*, *Intelligent Scissors*, *Bayes Matting, Knock Out*, *graphcut* memerlukan interaksi atau masukan cukup banyak dari *user* sehingga proses *foreground extraction* menjadi proses yang rumit dan membutuhkan waktu cukup lama. untuk itu penulis mencoba menerapkan metode yang merupakan optimasi dari metode *graphcut* yaitu dengan menggunakan metode *grabcut*. Metode ini pada prinsipnya sama dengan algoritma *graphcut* tetapi terdapat perbedaan pada teknik pembobotan tiap *edge* pada *graph* yang dibentuk, *grabcut* menggunakan GMM (*Gaussian mixture model*) sebagai teknik klusterisasi dan perhitungan nilai pembobotan *edge*. GMM yang memungkinkan teknik *foreground extraction* dapat dilakukan pada gambar berwarna. Selain itu dengan metode ini interaksi atau input dari *user* dapat diminimalisir dengan melakukan perulangan pada tahapan *graphcut*. Nilai akurasi ekstraksi yang didapat dengan metode *grabcut* cukup bagus untuk gambar yang cukup bisa dibedakan antara foreground dan background dengan nilai akurasi lebih dari 90%.

Kata kunci: *Grabcut, Graphcut, Foreground Extraction, Foreground Segmentation, Gaussian Mixture Model.*

OPTIMIZATION OF FOREGROUND EXTRACTION TECHNIQUE USING GRABCUT

ABSTRACT

In a digital image, foreground is an object designated as the main object by a person who sees the digital image. While the background is all part of the picture are not included in the foreground. Naturally humans can easily distinguish between the foreground and background, unlike the case with computers. Until now, various methods were developed to be able to find an effective method and can be applied in a computer for foreground segmentation. Several methods such as MagicWand, Intelligent Scissors, Bayes matting, Knock Out, Graphcut require a lot of interaction or input from the user so that the foreground extraction becomes complicated process and takes time for the user. Authors tried to apply the method which is the optimization of graphcut by using a method called grabcut. This method is in principle the same as the graphcut algorithm but there are differences in pixels clustering technique. Grabcut using GMM (Gaussian mixture model) that allows foreground extraction technique can be performed on color images. In addition this method allows user interaction to a minimum by doing a loop on graphcut. Extraction accuracy values obtained by grabcut method is good for an image that sufficient to distinguish between foreground and background with a value of more than 90% accuracy.

Keywords: Grabcut, Graphcut, Foreground Extraction, Foreground Segmentation, Gaussian Mixture Model.

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Allah SWT, karena atas segala rahmat dan limpahan hidayahnya, Tugas Akhir yang berjudul "Optimasi Teknik *Foreground Extraction* Menggunakan *Grabcut*" ini dapat diselesaikan. Sholawat serta salam semoga terlimpah kepada nabi Muhammad SAW serta para penerusnya hingga akhir zaman. Tugas Akhir ini disusun dan diajukan sebagai syarat untuk memperoleh gelar sarjana pada program studi Ilmu Komputer, jurusan Matematika, fakultas MIPA, universitas Brawijaya.

Dalam penyelesaian tugas akhir ini, penulis mendapat banyak bantuan baik moral maupun materiil dari banyak pihak. Atas bantuan yang telah diberikan, penulis ingin menyampaikan penghargaan dan ucapan terima kasih kepada:

1. Drs. Marji, MT. dan Edy Santoso, S.Si., M.Kom., selaku pembimbing tugas akhir ini. Terima kasih atas semua saran, bantuan, kritikan, waktu, dorongan semangat dan bimbingannya.
2. Dr. Abdul Rouf Alghofari, M.Sc., selaku ketua jurusan Matematika.
3. Ibu, Ayah dan keluarga di rumah yang senantiasa berdoa dan memberi dukungan dan semangat.
4. Segenap Bapak dan Ibu dosen yang telah mengajarkan ilmunya kepada penulis selama menempuh pendidikan di Program Studi Ilmu Komputer.
5. Sahabat-sahabat *Brotherhood*, Tim ISSOFT, dan teman-teman aktifis MIPA. terimakasih atas semangat dan kebersamaannya.
6. Semua teman-teman Ilmu Komputer angkatan 2006. Terima kasih atas semangat dan doanya.
7. Pihak lain yang tidak bisa penulis sebutkan satu-persatu.

Semoga penulisan laporan tugas akhir ini bermanfaat bagi pembaca sekalian. Penulis menyadari bahwa tugas akhir ini masih jauh dari kesempurnaan sehingga penulis mengharapkan kritik dan saran yang membangun dari pembaca.

Malang, 20 Juli 2011

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR *SOURCE CODE*

DAFTAR LAMPIRAN

BAB I

PENDAHULUAN

1.1 Latar Belakang

Foreground extraction merupakan teknik untuk memisahkan objek utama (*foreground*) dari objek lain yang bukan merupakan objek utama pada suatu citra digital atau yang diistilahkan dengan *background*. *Foreground extraction* adalah bagian dari segmentasi citra digial. Segmentasi citra digital merupakan salah satu bagian dari pengolahan citra digital yang bertujuan untuk membagi sebuah citra digital kedalam bagian-bagian dimana setiap elemen dalam satu bagian mempunyai korelasi cukup kuat dengan elemen lain dalam bagian tersebut (sonka, 1993).

Metode *foreground extraction* yang ada saat ini antara lain lain *Magic Wand*, *Intelligent Scissors*, *Bayes Matting, Knock Out*, *graphcut, grabcut*. Metode seperti *Magic Wand*, *Intelligent Scissors*, *Bayes Matting, Knock Out* memerlukan banyak informasi yang digunakan sebagai masukan dari *user* dalam proses ekstraksi *foreground* (Kolmogorov, 2004).

Metode yang saat ini cukup populer dikembangkan dalam teknik *foreground extraction* yaitu metode *graphcut*, *graphcut* merupakan teknik optimasi yang handal yang dapat digunakan pada operasi-operasi yang dapat dilakukan oleh *bayes matting*. *Graphcut* dapat digunakan dalam teknik *foreground extraction* walaupun *foreground* dengan *background*-nya cukup konvergen (Boykov dan Jolly, 2001). Akan tetapi metode *graphcut* memerlukan interaksi user yang lebih untuk memilah atau menandai antara wilayah objek dengan *background*, selain itu metode *graphcut* hanya dapat dilakukan dengan dasar *input* berupa gambar *grayscale*, *graphcut* yang diimplementasikan boykov dan joly (2001) menggunakan histrogram dari citra *grayscale* untuk mengambil nilai bobot tiap *edge* dalam *graph* yang terbentuk dari piksel.

Untuk itu diterapkan metode yang merupakan optimasi dari metode *graphcut* yaitu dengan menggunakan metode yang dinamakan *grabcut* (Kolmogorov, 2004). Metode ini pada prinsipnya sama dengan algoritma *graphcut,* tetapi terdapat perbedaan pada

teknik perhitungan nilai bobot tiap *edge* yang menghubungkan tiap *node* yang mewakili piksel pada *graph* yang dibentuk. *Grabcut* menggunakan GMM (*Gaussian mixture model*) sebagai teknik klusterisasi dan perhitungan nilai bobot tiap *edge* sehingga memungkinkan teknik *foreground extraction* dapat dilakukan pada gambar berwarna. Selain itu dengan metode ini interaksi atau *input* dari *user* dapat diminimalisir dengan melakukan perulangan pada tahapan *graphcut*.

1.2 Rumusan Masalah

Berdasarkan pada latar belakang yang telah dijelaskan sebelumnya, rumusan masalah pada penulisan tugas akhir ini adalah:

1. Bagaimana menerapkan teknik *foreground extraction* menggunakan metode *grabcut* sehingga tercipta aplikasi *foreground extraction* yang memungkinkan campur tangan *user* seminimal mungkin?
2. Bagaimana *performance* metode *grabcut* dalam proses ekstraksi *foreground* jika diujikan dengan berbagai macam citra digital?

1.3 Batasan Masalah

Batasan masalah dalam penulisan tugas akhir ini adalah:

1. Citra yang digitial yang digunakan adalah citra digital berwarna dengan format *.bmp atau *.jpeg .
2. Objek yang diekstraksi hanya satu kumpulan objek (satu area).

1.4 Tujuan Pembahasan

Tujuan dari penulisan tugas akhir ini adalah

1. Terciptanya sebuah aplikasi *foreground extraction* yang *powerfull* yang memungkinkan interaksi *user* seminimal mungkin sehingga dapat memudahkan *user* dalam proses pemisahan antara objek dengan *background* pada citra digital.
2. Untuk mengetahui *performance* metode *grabcut* sebagai teknik ekstraksi *foreground* jika diimplementasikan dan diuji dengan menggunakan beragam citra digital.

1.5 Metodologi Pemecahan Masalah

Metodologi yang digunakan untuk memecahkan masalah dalam penyusunan tugas akhir ini meliputi :

1. Studi literatur

   Mempelajari teori-teori yang berhubungan dengan *image processing, Computer Vison* terutama yang berhubungan dengan segmentasi dan teknik permisahan objek dengan *background* terutama tentang *grabcut* itu sendiri.

3. Pendefinisian dan analisis masalah

   Mendefinisikan dan menganalisis masalah untuk mencari solusi yang tepat.

4. Perancangan dan implementasi sistem

   Membuat dan merancang sebuah aplikasi *foreground extraction* yang *powerfull*, yang menerapkan metode *grubcut*.

5. Uji coba dan analisa hasil implementasi

   Menguji perangkat lunak, dan menganalisa hasil dari implementasi tersebut apakah sudah sesuai dengan tujuan yang dirumuskan sebelumnya, untuk kemudian menarik kesimpulannya.

1.6 Sistematika Penulisan

Sistematika penulisan laporan tugas akhir ini adalah sebagai berikut :

1 BAB I Pendahuluan
Berisi uraian tentang latar belakang, rumusan masalah, batasan masalah, tujuan pembahasan, metodologi pemecahan masalah dan sistematika penulisan laporan.

2 BAB II Tinjauan Pustaka
Berisi dasar teori tentang metode dalam menentukan tiap langkah dalam penelitian.

3 BAB III Metodologi Penelitian
Berisi uraian tentang konsep-konsep dan tahap-tahap dalam implementasi teknik *grabcut* yang digunakan untuk *foreground extraction.*

4 BAB IV Implemetnasi dan Pembahasan
Dalam bab ini dijelaskan penerapan algoritma dan metode metode yang digunakan ke dalam suatu aplikasi yang sebenarnya

dan melakukan analaisa hasil *foreground extraction* dari aplikasi yang telah dibangun.

5 BAB VI Penutup

Bab ini berisi kesimpulan akhir serta saran-saran yang berguna dalam perbaikan dan pengembangan lebih lanjut.

BAB II

TINJAUAN PUSTAKA

2.1 Citra Digital

Citra dapat didefinisikan sebagai fungsi *continue* dua dimensi f(x, y), x dan y merupakan suatu koordinat spasial dimana amplitudo dari nilai f(x, y) untuk setiap pasangan x dan y merupakan intensitas atau tingkat abu-abu dari suatu titik x,y dari suatu citra. Jika x,y dan nilai intensitas dari f adalah bentuk diskrit dan berhingga maka suatu citra dapat disebut sebagai citra digital (Gonzales, 2008).

Representasi spasial dari citra berupa piksel yang berupa koordinat dari tiap citra. Didalam tiap piksel mengandung informasi tingkat *irradiance* atau tingkat kecerahan. Susunan *piksel* sama seperti susunan matriks, namun awal koordinat dimulai dari atas ke kanan (kolom) untuk *x* ke bawah (baris) untuk *y*. Sama seperti *array*, indeks *piksel* dimulai dari 0. Untuk piksel yang berada pada koordinat X bergerak dari 0 hingga *x-1*, begitu juga halnya dengan piksel pada koordinat Y bergerak dari 0 hingga *y-1* (Jähne, 2002).

2.2 Citra Digital berwarna

Untuk merepresentasikan citra bewarna salah satunya adalah dengan menggunakan kanal RGB (*Red-Green-Blue*). Pada model RGB setiap piksel muncul sebagai komposisi dari *Red* (Merah), *Green* (Hijau), dan *Blue* (Biru). Model ini didasarkan pada sistem koordinat *Cartesian*.

Dalam ruang RGB disebut kedalaman piksel (*depth*). Misalkan sebuah citra RGB dimana masing-masing citra *Red, Green*, dan *Blue* adalah suatu citra yang berkedalaman 8 *bit*, Pada kondisi tersebut masing-masing piksel warna RGB akan memiliki kedalaman 24 *bit*. Citra warna RGB 24 *bit* disebut juga dengan citra *true colour*. Jumlah seluruh warna pada citra RGB 24 *bit* adalah $(2^8)^3$ = 16.777.216 (jonathan, 1999).

2.3 Elemen-Elemen Citra *Digital*

Elemen-elemen yang dimiliki oleh digital antara lain :

1. Kecerahan (*brightness*) atau intensitas cahaya. Nilai brightness mengidentifikiasikan seberapa terang atau gelap suatu warna (jonathan, 1999). Banyak intensitas cahaya yang dimiliki oleh suatu citra digital bukan merupakan nilai intensitas yang riil, tetapi merupakan intensitas rata-rata dari suatu area yang melingkupinya.
2. Kontras (*contrast*) merupakan perubahan lokal dari nilai *brightness* yang didefinisikan sebagai perbandingan antara nilai *brightness* rata rata dengan nilai brightness suatu area dalam satu citra (sonka, 1993).
3. Kontur (*contour*) adalah keadaan yang ditimbulkan oleh perubahan intensitas pada piksel-piksel yang bertetangga.
4. Warna (*colour*) adalah persepsi yang dirasakan oleh sistem visual manusia terhadap panjang gelombang cahaya yang dipantulkan oleh obyek. Setiap warna mempunyai panjang gelombang yang berbeda.
5. Tekstur (*texture*) merupakan distribusi spasial dari derajat keabuan di dalam sekumpulan piksel-piksel yang bertetangga.

2.4 Pengolahan Citra *Digital*

Pengolahan citra digital adalah pemrosesan citra dua dimensi menggunakan komputer *digital*. Pengolahan dilakukan untuk berbagai kepentingan. Operasi-operasi yang dilakukan dalam pengolahan citra secara umum dapat diklasifikasikan dalam beberapa jenis sebagai berikut:

1. Perbaikan kualitas citra (*image enhancement*)
2. Restorasi citra (*image restoration*)
3. Pemampatan citra (*image compression*)
4. Segmentasi citra (*image segmentation*)
5. Analisis citra (*image analysis*)
6. Rekontruksi citra (*image reconstruction*)

2.5 *Image Segmentation*

Image segmentation merupkan proses pemisahan atau pengelompokkan suatu citra (*image*) kedalam bagian-bagian yang berbeda. Pada umumnya bagian yang dimaksud berdasarkan pada suatu karakteristik dimana manusia dapat dengan mudah membedakan dan meliahtnya sebagai objek yang berbeda. Saat ini banyak metode yang dikembangakan dengan tujaun untuk melakukan segmentasi pada suatu citra. Proses segmentasi didadasarkan pada karekteristik tertentu yang terdapat pada citra. Karakteristik tersebut dapat berupa informasi warna atau informasi tiap piksel yang mengindekasikan suatu tepi atau tekstur tertentu (Ballard dan Brown, 1982).

2.6 *Graph*

Graph adalah salah satu pokok bahasan matematika diskrit yang telah lama dikenal dan banyak diaplikasikan pada berbagai bidang. Secara umum graph *G* didefinisikan sebaga pasangan himpunan (*V*, *E*) ditulis dengan notasi *G = (V, E)* yang dalam hal ini *V* adalah himpunan tidak kosong dari simpul-simpul (*nodes*) dan E adalah himpunan sisi atau busur (*edges*) yang berhubungan dengan sepasang simpul (Munir, 2003).

Teknik *graph* yang didasarkan pada segmentasi pada umumnya mempresentasikan masalah *graph G* = (*V* , *E*) dimana setiap simpul *Vi* ∈ *V*, i= 1,2,3…,n dianggap piksel-piksel dari citra dan sisi E merupakan pasangan-pasangan dari piksel-piksel yang bertetanggaan. Setiap sisi yang menghubungkan piksel-piksel pada citra memiliki bobot (*weight*) tertentu.

2.7 *Graphcut*

Boykov dan jolly memeperkenalkan metode baru dalam segmentasi citra dengan menggunakan *graph* untuk menggambarkan sebuah citra, dan selanjutnya menggunakan mekanisme *min-cut max-flow* untuk memisah *graph* tersebut. Piksel dalam citra merupakan *node* dalam *graph*, nilai bobot dalam setiap *edge* merupakan suatu *cost function* yang didefiniskan oleh informasi wilayah dan batas dari suatu citra. *Min-cut max-flow* digunakan untuk memecah citra dengan meminimilkan nilai dari *cost function* (Matthew, 2010).

2.8 *Min Cut Max Flow*

Dalam mekanisme *min - cut max - flow*, terdapat *graph* dengan dua *node* terminal. *Node* terminal pertama dinamakan *Source* dan *node* terminal kedua dinamakn *Sink*. *Edge* berbobot yang menghubunkan masing-masing *node* dalam graph dianggap sebagai jalur yang memiliki kapasitas (*Flow*) tertentu. Suatu *cut* (titik potong) merupakan kumpulan *edge* yang jika *edge* tersebut dihilangkan akan memisahkan *graph* menjadi dua bagian. Dalam suatu *graph* G total nilai *flow* tertinggi dalam jalur yang mungkin dari *source* ke *sink* adalah sama dengan nilai *cut* terkecil dari *graph* tersebut (Foulds, 1992). Beberapa algoritma yang menerapkan kaidah *min-cut max-flow* antara lain *Ford Fulkerson method (*Ford Fulkerson, 1962), *push-relabel method (*Cherkassky ,1997).

2.9 Ford Fulkerson *algorithm*

Ford-Fulkerson *algorithm* pertama kali dipublikasikan pada tahun 1956 oleh L. R. Ford, Jr. dan D. R. Fulkerson. Algoritma ini juga sering disebut *Edmonds–Karp algorithm*. pencarian nilai *flow* maksimal dari suatu *graph* didasarkan pada proses pemberian nilai *flow* terbesar yang mungkin pada setiap *augmenting path* yang ditemukan. *Augmenting path* merupakan jalur dari *source node* menuju *sink node* yang memiliki nilai kapasitas atau bobot lebih dari 0 disetiap *edge* yang menjadi jalur *source* ke *sink*. *Flow* maksimal dicapai apabila tidak ada *augmenting path* lagi yang ditemukan (yuri dan kolmogorov, 2004). Adapun *pseudo code* dari algortima ford Fulkerson adalah :

While (ditemukan *augmenting path*)
Do
// temukan nilai minimal dari jalur

$$c_f(p) = \min\{c_f(u,v) : (u,v) \in p\}$$

Untuk setiap *edge* $(u,v) \in p$
Do
//*update cost* pada jalur

$$f(u,v) \leftarrow f(u,v) + c_f(p)$$
$$f(v,u) \leftarrow f(v,u) - c_f(p)$$

End ; End while .

2.10 *Grabcut*

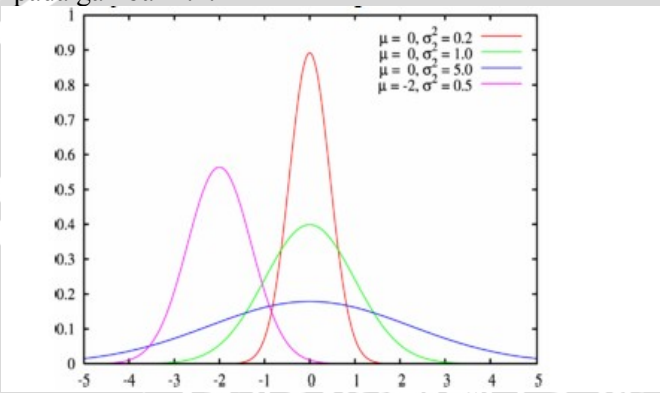
Grabcut yang dikembangkan oleh rother (2004) merupakan salah satu metode segmentasi citra yang didasarkan pada algoritma *graphcut* (boykov dan joly, 2001). *Grabcut* adalah suatu algoritma iteratif yang menggabungkan metode statistika dan algortima *graphcu* dalam rangka untuk mencapai segmentasi 2D rinci dengan input yang terbatas. Metode ini awalnya dikembangkan *Microsoft Research* yang bertempat di Cambridge (Thomas, 2010).

Grabcut menyempurnakan algoritma *graphcut*, *grabcut* memungkinkan segmentasi dilakukan langsung pada citra berwarana. berbeda dengan *graphcut* yang didasarkan pada citra *grayscale*. informasi yang bertindak sebagai *input* metode *grubcut* adalah wilayah gambar persegi panjang yang dibuat oleh *user*, yang selanjutnya informasi ini akan dimasukkan pada *trimap* yang sesusai. Dengan menggunakan *grabcut* dimungkinkan interaksi *user* menjadi minimal. *grabcut* merupakan metode yang sangat efektif untuk teknik *foreground extraction.* secara singkat tahapan metode *grabcut* adalah sebagai berikut:

1. *User* menciptakan *trimap* awal dengan membuat persegi panjang pada sekitar objek yang ditentukan sebagai *foreground* dari gambar. Piksel dalam *trimap* ditandai sebagai kelompok *trimap unknown* (tidak diketahui). Piksel diluar trimap ditandai sebagai piksel dalam kelompok *trimap background*.
2. Segmentasi awal dilaksanakan, piksel *unknown* ditempatkan pada kelas *foreground* sedangkan piksel *background* dikelompokkan pada kelas *background.*
3. GMM (*Gaussian mixture model*) dibuat untuk masing masing kelas (*foreground* dan *background*).
4. *Graph* dari antar piksel dibentuk dan dijalankan oprasi *Graphcut* untuk mendapatkan hasil segmentasi antara *foreground* dan *background.*
5. Langkah 3-5 diulang hingga hasil segmentasi konvergen atau hingga didapat hasil yang diharapkan.

2.11 *Gaussian Mixture Model*

Gaussian Mixture Model berasal dari distribusi *Gauss* atau distribusi normal. Distribusi normal adalah kurva yang mempunyai 2 parameter, parameter pertama adalah μ atau *mean* dan kedua adalah σ atau standar deviasi. Kumpulan distribusi normal biasa disebut dengan *normal probability density function* atau normal pdf yang ditunjukkan pada gambar 2.1.



Gambar 2.1 *Normal Probability Density Function*

Distribusi standar normal mempunyai μ=0 dan σ=1, untuk gambar diatas ditunjukan dengan kurva warna hijau, rumus umum untuk pdf 1 dimensi dituliskan pada persamaan 2.2.

$$y = f(x|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{2.2}$$

Gaussian mixture model dibentuk dari multi variansi normal pdf (*probability density function*) atau pdf d-dimensi. Nilai kemungkinan yang diberkan distribusi *gaussian* adalah seperti yang terdapat pada persamaan 2.3.

$$\mathcal{N}(x|\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}}\sqrt{|\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

Sedangkan nilai kemungkinan pada GMM

$$p(x) = \sum_{i=1}^{N} w_i \cdot \mathcal{N}(x|\mu_i,\Sigma_i) \tag{2.3}$$

x dan μ adalah suatu vektor dan Σ adalah matrik kovarian, *Wi* adalah bobot *gaussian* ke i, dan N adalah jumlah komponen *gaussian* pada GMM (Bengio, 2006).

GMM memiliki nilai kesalahan klasifikasi yang paling kecil jika dibandingkan dengan pemodelan data secara statistik lain seperti halnya *2-D hidden Markov model* (HMM), *2-D multi-resolution hidden Markov model* (MHMM) dan *classification and regression trees* (Mayuresh Kulkarni, 2010).

2.12 Implementasi *Graphcut* dalam Metode *Grabcut*

Dalam algoritma *graphcut* seperti yang dijelaskan boykov dan jolly (2001), terdapat *node* pada setiap piksel dan dua *node* sepesial, *foreground node* dan *background node*. *Node* ini dihubungkan oleh dua tipe *edge* (garis) yang disebut *link*, *N-link* menghubunkan piksel 8-*neighborhood*, *link* ini menggambarkan suatu nilai untuk menempatkan batas segmentasi antara piksel tetangga, nilai bobot setiap *N-link* adalah konstan selama eksekusi program.

T-link menghubunkan piksel-piksel pada *foreground node* atau *background node*. *Link* ini menentukan kemungkinan apakah setiap piksel termasuk piksel *background* atau piksel *foreground*. Dalam *grabcut* kemungkinan ini dihitung dalam algoritma GMM, algoritma GMM mengupdate nilai probabilitas atau kemungkinan tadi dalam setiap iterasi.

Rumus untuk bobot nilai *N-Link* diantara piksel m dan 8 piksel n tetangganya terdapat pada persamaan 2.4.

$$N(m,n) = \frac{\gamma}{dist(m,n)} e^{-\beta\|z_m - z_n\|^2} \tag{2.4}$$

dimana Z_m adalah warna dari piksel m, γ = 50 , didefinisikan oleh Boykov and jolly [2001] dan β sebagaimana yang terdapat pada persamaan 2.5 merupakan nilai rata-rata jarak antar piksel dari semua *edge* yang terbentuk. *Dist(m,n)* merupakan fungsi jarak antar piksel dimana x dan y menyatakan koordinat piksel.

$$\beta = \frac{1}{2||Zm - Zn||^2} \tag{2.5}$$

$$||Zm - Zn||^2 = \frac{\sum(R1 - R2)^2 + (G1 - G2)^2 + (B1 - B2)^2}{\sum edge} \tag{2.6}$$

$$dist\ (m,n) = \sqrt{(x1 - x2)^2 + (y1 - y2)^2} \tag{2.7}$$

Seperti yang telah dijelaskan sebelumnya, terdapat dua macam *T-Link* yaitu *Background T-link* yang menghubungkan piksel dengan *background node* dan *foreground T-Link* yang menghubungkan piksel dengan *foreground node*. Bobot dari setiap *link* ini tergantung pada keadaan *trimap* yang dibuat oleh *user*. Jika *user* telah menandai bahwa suatu piksel adalah piksel *background* atau *foreground*, bobot *link* disesuaikan sedemikian rupa sehingga piksel harus menjadi kelompok yang sesuai. Untuk piksel yang tidak diketahui apakah termasuk *foreground* atau *background* maka digunakan nilai kemungkinan (*Likelihood*) yang diperoleh dari algortima GMM, niilai bobot setiap *T-Link* pada piksel adalah seperti yang terdapat pada tabel 2.1.

Tabel 2.1 Klasifikasi Piksel *Foreground* dan *Background*

Pixel Type	Background T-link	Foreground T-link
$m \in$ TrimapForeground	0	$L(m)$
$m \in$ TrimapBackground	$L(m)$	0
$m \in$ TrimapUnknown	$D_{Fore}(m)$	$D_{Back}(m)$

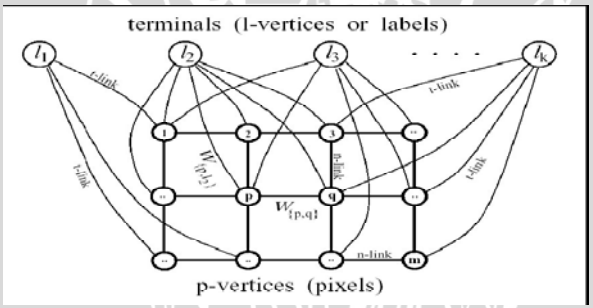
untuk mengelompokkan secara paksa suatu piksel ke dalam kelompok *foreground* atau *background* kita menggunakan persamaan 2.8.

$$L(m) = 8\gamma + 1 \tag{2.8}$$

D_{Fore} dan D_{Back} adalah fungsi *likelihood* pada algoritma GMM yang menentukan seberapa besar kemiripan suatu piksel termasuk pada piksel *foreground* atau piksel *background* dimana fungsi *likelihood* tersebut seperti yang terdapat pada persamaan 2.9.

$$D(m) = \log \sum_{i=1}^{K} \pi_i \frac{1}{\sqrt{\det \Sigma_i}} e^{\left(-\frac{1}{2}[z_m - \mu_i]^T \Sigma_i^{-1} [z_m - \mu_i]\right)} \tag{2.9}$$

K : jumlah *gaussian* komponen
Det $\sum$: determinan dari matriks kovarian
$\sum^{-1}$: invers dari kovarian matrix (3x3 matriks)
μ : rata-rata (terdiri dari RGB)
π : bobot dari setiap komponen *gaussian*



Gambar 2.2 *T-Link dan N-Link*

2.13 Matriks

Matriks adalah sekumpulan informasi yang setiap individu elemennya terdefinisi berdasarkan dua buah indeks (yang biasanya dikonotasikan dengan baris dan kolom). Setiap elemen matriks dapat diakses secara langsung jika kedua indeks diketahui, dan indeksnya harus bertype yang mempunyai keterurutan (suksesor), misalnya integer. Matriks adalah struktur data dengan memori internal.

Suatu matriks tersusun atas baris dan kolom, jika matriks tersusun atas m baris dan n kolom maka dikatakan matriks tersebut berukuran (berordo) m x n. Penulisan matriks biasanya menggunakan huruf besar A, B, C dan seterusnya, sedangkan penulisan matriks beserta ukurannya (matriks dengan m baris dan n kolom) adalah A m x n, B m x n dan seterusnya (yuliant, 2002).

2.13.1 Operasi – Operasi pada Matriks

2.13.1.1 Penjumlahan Matriks

Operasi penjumlahan dapat dilakukan pada dua buah matriks yang memiliki ukuran yang sama. Aturan penjumlahan dengan menjumlahkan elemen–elemen yang bersesuaian pada kedua matriks seperti yang terdapat pada persamaan 2.10.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix} \quad (2.10)$$

2.13.1.2 Perkalian Matriks dengan Matriks

Operasi perkalian matriks dapat dilakukan pada dua buah matriks (A dan B) jika jumlah kolom matriks A = jumlah baris matriks B. Misalkan Amn dan Bnk maka Amn Bnk = Cmk dimana elemen – elemen dari C (cij) merupakan penjumlahan dari perkalian elemen–elemen A baris i dengan elemen–elemen B kolom j seperti yang terdapat pada persamaan 2.11.

$$A = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}, B = \begin{bmatrix} k & n \\ l & o \\ m & p \end{bmatrix} \text{ maka } A_{23} B_{32} = C_{22} = \begin{bmatrix} ak+bl+cm & an+bo+cp \\ dk+el+fm & dn+eo+fp \end{bmatrix} \quad (2.11)$$

2.13.1.3 Perkalian Matriks dengan Skalar

Suatu matriks dapat dikalikan suatu skalar k dengan aturan tiap–tiap elemen pada A dikalikan dengan k, seperti yang terdapat pada persamaan 2.12.

$$3\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} = \begin{bmatrix} 3a & 3b & 3c \\ 3d & 3e & 3f \end{bmatrix} \tag{2.12}$$

2.13.2 Transpose Matriks

Transpose matriks A (dinotasikan A^t) didefinisikan sebagai matriks yang baris – barisnya merupakan kolom dari A seperti yang terdapat pada persamaan 2.13.

$$A - \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \rightarrow A^t - \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \tag{2.13}$$

2.13.3 Matriks Invers

Jika A, B matriks bujur sangkar dan berlaku AB = BA = I (I matriks identitas), maka dikatakan bahwa A dapat dibalik dan B adalah matriks invers dari A (notasi A^{-1}) seperti terdapat pada persamaan 2.14.

$$A = \begin{bmatrix} 2 & -5 \\ -1 & 3 \end{bmatrix},\ B = \begin{bmatrix} 3 & 5 \\ 1 & 2 \end{bmatrix} \rightarrow AB = BA = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.14}$$

2.13.4 Determinan Matriks

Misalkan A matriks bujur sangkar fungsi determinan A dituliskan sebagai determinan (disingkat det(A) atau |A|) didefinisikan sebagai jumlah semua hasil kali elementer bertanda dari A.

2.13.4 .1 Metode Perhitungan Determinan

Satu diantara metode perhitungan determinan yang ada adalah metode minor kofaktor, kofaktor elemen aij (Cij) = (1)i+j Mij, (Mij) merupakn minor elemen aij yaitu determinan yang didapatkan dengan menghilangkan baris i dan kolom j matriks awalnya. Jika A matriks bujur sangkar berukuran nxn , maka dengan

menggunakan metode ini perhitungan determinan dapat dilakukan dengan dua cara yang semuanya menghasilkan hasil yang sama yaitu:

– ekspansi sepanjang baris i

$\det(A) = a_{i1}C_{i1} + a_{i2}C_{i2} + \ldots + a_{in}C_{in}$

– ekspansi sepanjang kolom j

$\det(A) = a_{1j}C_{1j} + a_{2j}C_{2j} + \ldots + a_{nj}C_{nj}$

(2.15)

Misalkan ada sebuah matriks A_{3x3}

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

maka determinan dari matriks tersebut dengan ekspansi kofaktor adalah seperti yang terdapat pada persamaan 2.16.

$$\det(A) = a_{11}\begin{bmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{bmatrix} - a_{21}\begin{bmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{bmatrix} + a_{31}\begin{bmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}$$

$$= a_{11}(a_{22}a_{33} - a_{23}a_{32}) - a_{21}(a_{21}a_{33} - a_{23}a_{31}) + a_{31}(a_{21}a_{32} - a_{22}a_{31})$$

$$= a_{11}a_{22}a_{33} + a_{21}a_{23}a_{31} + a_{31}a_{21}a_{32} - a_{22}(a_{31})^2 - (a_{21})^2a_{33} - a_{11}a_{23}a_{32}$$

(2.16)

2.13.5 Adjoin Matriks 3 x 3

Bila ada sebuah matriks A_{3x3}

$$A = \begin{bmatrix} 3 & 2 & -1 \\ 1 & 6 & 3 \\ 2 & 4 & 0 \end{bmatrix}$$

Kofaktor dari matriks A adalah

$C_{11} = -12$ $C_{12} = 6$ $C_{13} = -16$

$C_{21} = 4$ $C_{22} = 2$ $C_{23} = 16$

$C_{31} = 12$ $C_{32} = -10$ $C_{33} = 16$

maka matriks yang terbentuk dari kofaktor tersebut adalah:

$$\begin{bmatrix} 12 & 6 & -16 \\ 4 & 2 & 16 \\ 12 & -10 & 16 \end{bmatrix}$$

untuk mencari adjoint sebuah matriks, cukup mengganti kolom menjadi baris dan baris menjadi kolom

$$adj(A) = \begin{bmatrix} 12 & 4 & 12 \\ 6 & 2 & -10 \\ -16 & 16 & 16 \end{bmatrix}$$

Menentukan invers suatu matriks dapat menggunakan persamaan 2.17.

$$A^{-1} = \frac{adj(A)}{\det(A)} \tag{2.17}$$

2.13.6 Nilai Eigen Suatu Matriks

Jika diketahui A matriks berukuran n x n, $\bar{x}$ vektor tak–nol berukuran n x 1, x ∈Rn. Karena A berukuran n x n, maka A$\bar{x}$ akan berupa vektor yang berukuran n x 1 juga. Bila terdapat scalar λ , $\lambda \in$ Riil sedemikian hingga A$\bar{x}$ = $\lambda \bar{x}$ (A$\bar{x}$ menghasilkan vektor yang besarnya λ kali $\bar{x}$). Semua nilai λ yang memenuhi persamaan tersebut sehingga ada nilai $\bar{x}$ yang nyata (bukan vektor 0 saja) disebut *nilai eigen*(karakteristik). Untuk menentukan nilai λ, dari persamaan A $\bar{x}$ = $\lambda\bar{x}$ sebelumnya dirubah dahulu menjadi persamaan (A – λ I) $\bar{x}$ = 0 = (λI – A) $\bar{x}$. Agar persamaan tersebut memiliki penyelesaian tak–trivial(sejati), maka dapat ditentukan melalui nilai det (A – λI) yaitu det (A – λI) = det (λI – A) = 0. Persamaan det (A – λI) = det (λ I – A) = 0 ini disebut persamaan karakteristik. Banyaknya nilai eigen maksimal adalah n buah.

2.14 *Binary Tree Color Quantization Algorithm*

Binary tree color quantitization algorithm seperti halnya yang dijelaskan oleh orchad bowman, dalam algortima ini pada awalnya semua piksel dikelompokkan pada satu kluster, kemudian kluster tersebut dibagi menjadi dua kluster berdasarkan nilai eigen vektor dari *covariace matrix* sebagai titik pemisah. Selanjutnya kluster yang akan dipecah dipilih berdasarkan nilai *eigenvalue* terbesar dari matriks kovarian yang mewakili tiap-tiap kluster, prosedur ini diulang hingga jumlah dari kluster yang diinginkan terpenuhi.

2.15 Matriks kovarian

Matriks Kovarian merupakan matriks yang unsur-unsurnya berupa varian (ragam) dan kovarian (peragam) dari sekumpulan variabel. Diagonalnya berupa varian (ragam) dari setiap variabel, sedangkan unsur lainnya berupa kovarian (peragam) antar variabel. Matriks S bersifat simetris atau setangkup, persamaan pembentuk matriks kovarian dalam kondisi multivariabel terdapat pada persamaan 2.18.

$$\Sigma = \mathrm{E}(\mathbf{X}\mathbf{X}^\top) - \mu\mu^\top \tag{2.18}$$

dimana

$$E(XX^T) = \frac{1}{N}(XX^T) \tag{2.19}$$

X merupakan variabel yang didapat dari data, μ merupakan rata-rata dari variabel tersebut. Jika persaman 2.18 dan 2.19 diimpelementasikan pada *grabcut*, dimana variabel X merupakan warna yang terdiri dari nilai R,G,B maka persamaan yang didapat adalah seperti yang terdapat pada persamaan 2.20.

$$X = \begin{bmatrix} R1 & \dots & Rn \\ G1 & \dots & Gn \\ B1 & \dots & Bn \end{bmatrix} \qquad X^T = \begin{bmatrix} R1 & G1 & B1 \\ \dots & \dots & \dots \\ Rn & Gn & Bn \end{bmatrix}$$

$$\mu = \begin{bmatrix} \mu R1 \\ \mu G1 \\ \mu B1 \end{bmatrix} \qquad \mu^T = [\mu R1 \quad \mu G1 \quad \mu B1]$$

$$E(XX^T) = \frac{1}{N}\begin{bmatrix} \sum_1^n R*R & \sum_1^n R*G & \sum_1^n R*B \\ \sum_1^n G*R & \sum_1^n G*G & \sum_1^n G*B \\ \sum_1^n B*R & \sum_1^n B*G & \sum_1^n B*B \end{bmatrix}$$

$$\mu\mu^T = \begin{bmatrix} \mu R * \mu R & \mu R * \mu G & \mu R * \mu B \\ \mu G * \mu R & \mu G * \mu G & \mu G * \mu B \\ \mu B * \mu R & \mu B * \mu G & \mu B * \mu B \end{bmatrix}$$

sehingga

$$E(XX^T) - \mu\mu^T = \frac{1}{N}\begin{bmatrix} \sum_1^n R * R & \sum_1^n R * G & \sum_1^n R * B \\ \sum_1^n G * R & \sum_1^n G * G & \sum_1^n G * B \\ \sum_1^n B * R & \sum_1^n B * G & \sum_1^n B * B \end{bmatrix} - \begin{bmatrix} \mu R * \mu R & \mu R * \mu G & \mu R * \mu B \\ \mu G * \mu R & \mu G * \mu G & \mu G * \mu B \\ \mu B * \mu R & \mu B * \mu G & \mu B * \mu B \end{bmatrix} \quad (2.20)$$

2.16 *Percent Error*

Nilai akurasi dari sebuah hasil dinyatakan dengan *percent error*, dimana *percent error* adalah :

$$\left(\frac{theoretical\ value - experimental\ value}{theoritical\ value} x100\%\right)$$

Jika diimplementasikan pada *foreground extraction* maka :

$$\left(\frac{jumlah\ semua\ piksel - experimental\ value}{jumlah\ semua\ piksel} x100\%\right) \quad (2.21)$$

Jumlah $experimental\ value$ merupakan penjumlahan antara jumlah piksel *foreground* yang hilang dengan jumlah piksel *background* yang tertinggal, atau dalam hal ini adalah jumlah *wrong pixels*.

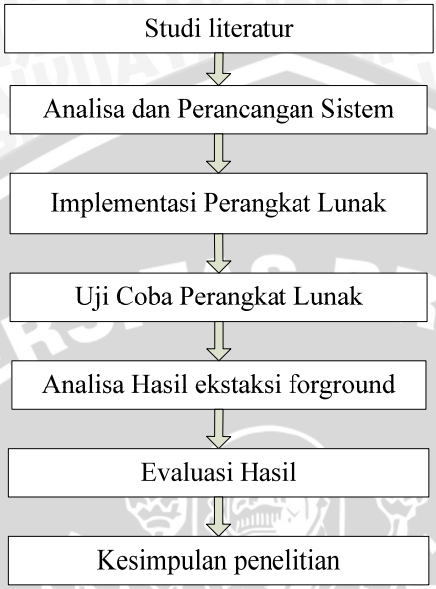
BAB III

METODOLOGI DAN PERANCANGAN

Pada bab metodologi dan perancangan ini dibahas metode, rancangan yang digunakan dan langkah-langkah yang dilakukan dalam penelitian tentang penggunaan *grabcut* dalam teknik *foreground extraction* hingga metode tersebut siap diaplikasikan untuk membentuk suatu aplikasi yang diharapkan. Dalam penelitian ini penulis menggunakan alur *waterfall* sebagai metode perancangan sistem. Adapun metode tersebut secara garis besar adalah sebagai berikut :

a. Studi literatur terkait segala hal yang berhubungan dengan *grabcut* dan algoritma lain yang dipakai dalam sistem atau aplikasi yang akan dibuat.
b. Menganalisa dan melakukan perancangan sistem optimasi teknik *foreground extraction* dengan menggunakan algoritma *grabcut.*
c. Membuat perangkat lunak (aplikasi) berdasarkan analisa dan perancangan yang telah dilakukan.
d. Melakukan uji coba terhadap perangkat lunak *foreground extraction* yang mengimplementasikan algortima *grabcut* dengan berbagai variasi citra yang ada.
e. Melakukan evaluasi hasil *foreground extraction* yang merupakan *output* dari sistem yang telah dibuat dengan cara menganalisa akurasi, kecepatan ekstraksi dan uji coba dalam beberapa kondisi *trimap*.
f. Menarik kesimpulan hasil dari penelitian yang sudah dilakukan.

Metode penelitian skripsi ini dapat digambarkan seperti pada gambar 3.1.



Gambar 3.1 Langkah – Langkah Penelitian

3.1 Bahan Penelitian

3.1.1 Studi Literatur

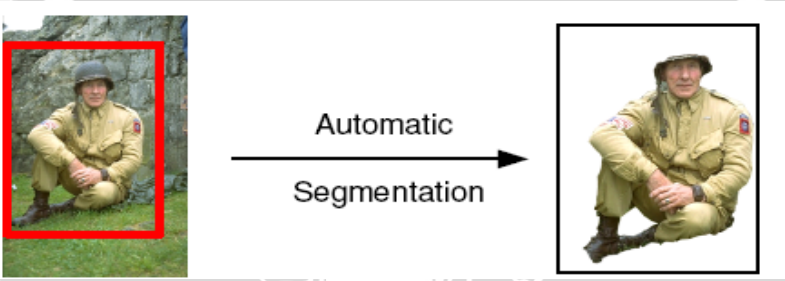
Dalam penelitian ini dibutuhkan studi literatur untuk merealisasikan tujuan dan penyelesaian masalah. Teori-teori terkait citra digital, algoritma *graphcut*, *grubcut*, GMM, dan beberapa teori yang digunakan sebagai dasar penelitian diperoleh dari buku, jurnal dan *browsing* dari internet, metode-metode yang telah dipelajari selanjutnya disesuaikan untuk dapat diaplikasikan sesuai dengan tujuan penelitian.

3.1.2 Data Citra Digital yang Digunakan

Citra yang digunakan untuk percobaan (analisa) adalah berupa citra digital (berformat *.jpg atau *.bmp) dengan berbagai variasi tingkat homogenitas distribusi warna, mulai dari citra yang memiliki keragaman warna sedikit dan mudah dibedakan antara *background* dan *foreground* hingga citra yang sangat sulit dibedakan antara *foreground* dan *background*. Citra digital tersebut didapatkan dari berbagai macam sumber.

3.2 Deskripsi Umum Sistem

Secara umum sistem yang dibangun adalah perangkat lunak aplikasi *foreground extraction* dimana dengan aplikasi ini *user* dapat dengan mudah mengekstrak atau menggambil bagian objek dari gambar yang berupa *foreground* dari gambar keseluruhan. Ilustrasi dari aplikasi ini terdapat pada gambar 3.2.



Gambar 3.2 Ilustrasi Deskripsi Umum Sistem

data yang dipakai dalam aplikasi ini adalah citra digital berformat *.bmp atau *.jpg . Adapun *input* minimal dari *user* yang dibutuhkan oleh sistem berupa *trimap* (bagian gambar berbentuk bujur sangkar) yang ditandai oleh *user*. Informasi lain yang dapat ditambahkan oleh *user* berupa beberapa bagian gambar (piksel) yang ditandai oleh *user* sebagai bagian dari piksel *background* atau bagian dari piksel *foreground*. Untuk mengoptimasi teknik ekstraksi *foreground*, aplikasi yang akan dibuat menerapkan *grabcut* sebagai metode yang digunakan.

3.2 Analisa Sistem

3.2.1 Analisa Permasalahan

Pada umumnya, aplikasi *foreground extraction* yang ada membutuhkan banyak interaksi *user* (banyak *input*). Sepertihalnya *magic wand* yang dimiliki *photoshop*, *user* harus menyeleksi tepi melingkar pada objek yang dikehendaki. Pada penelitian kali ini dengan menerapkan algoritma *grabcut* dimungkinkan untuk membuat aplikasi *foreground extraction* dengan input yang minimal yaitu cukup dengan input berupa wilayah *trimap*.

3.2.1.1 Beberapa Ketentuan *User Interaction*

Interaksi *user* sebagai *input* dalam algortima *grabcut* mempunyai peranan penting terhadap hasil ekstraksi gambar, adapun beberapa hal yang perlu diperhatikan antara lain:

1. *User* hanya dapat membuat satu wilayah *trimap.*
2. *Trimap* yang dibentuk merujuk pada suatu objek yang spesifik dan dapat didefinisikan sebagai suatu objek.
3. Batas tepi *trimap* dengan objek yang dirujuk tidak terlalu melebar, maksimal sekitar 30 piksel.

3.2.1.2 Faktor yang Mempengaruhi Hasil dari *Foreground Extraction*

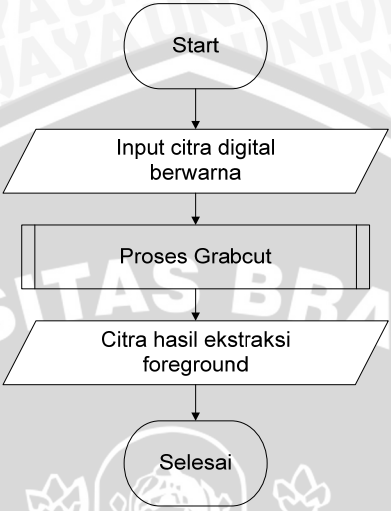
adapun beberapa faktor yang mempengaruhi hasil ekstraksi *foreground* pada suatu citra digital antara lain:

1. input citra dan koherenitas warna pada citra, semakin homogen warana background dengan objek yang akan diekstraksi (*foreground*) maka kemungkinan hasil akan semakin tidak maksimal.
2. wilayah *trimap* yang ditandai oleh *user*, semakin melebar wilayah *trimap*, kemungkinan akan mengurangi kualitas hasil ekstraksi.
3. penandaan area piksel yang ditentukan sebagai *foreground* atau *background.*

3.3 Desain dan Perancangan Sistem

3.3.1 Desain Global Sistem

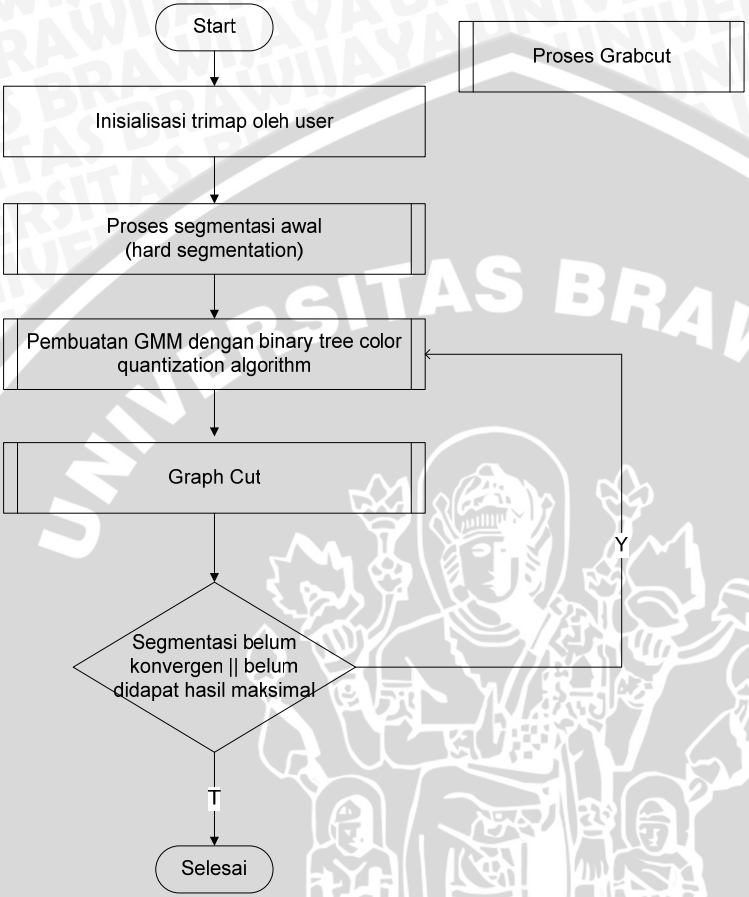
Aplikasi *foreground extraction* ini dirancang untuk dapat digunakan semudah mungkin, dengan *input* atau masukan dari *user* seminimal mungkin sehingga dapat digunakan oleh semua orang dengan cepat, mudah dan efisien. Yang selanjutnya objek hasil ekstraksi dapat digunakan untuk pengolahan citra lebih lanjut. adapun secara global sistem dapat digambarkan seperti halnya *flowchart* pada gambar 3.3



Gambar 3.3 Desain Global sistem

3.3.2 Proses Metode *GrabCut*

Implementasi tahapan – tahapan metode grabcut yang terdapat pada pada bab dapat digambarkan pada *flowchart* yang terdapat pada gambar 3.4.



Gambar 3.4 *Flowchart Grabcut*

3.3.2.1 Proses Inisialisasi *Trimap*

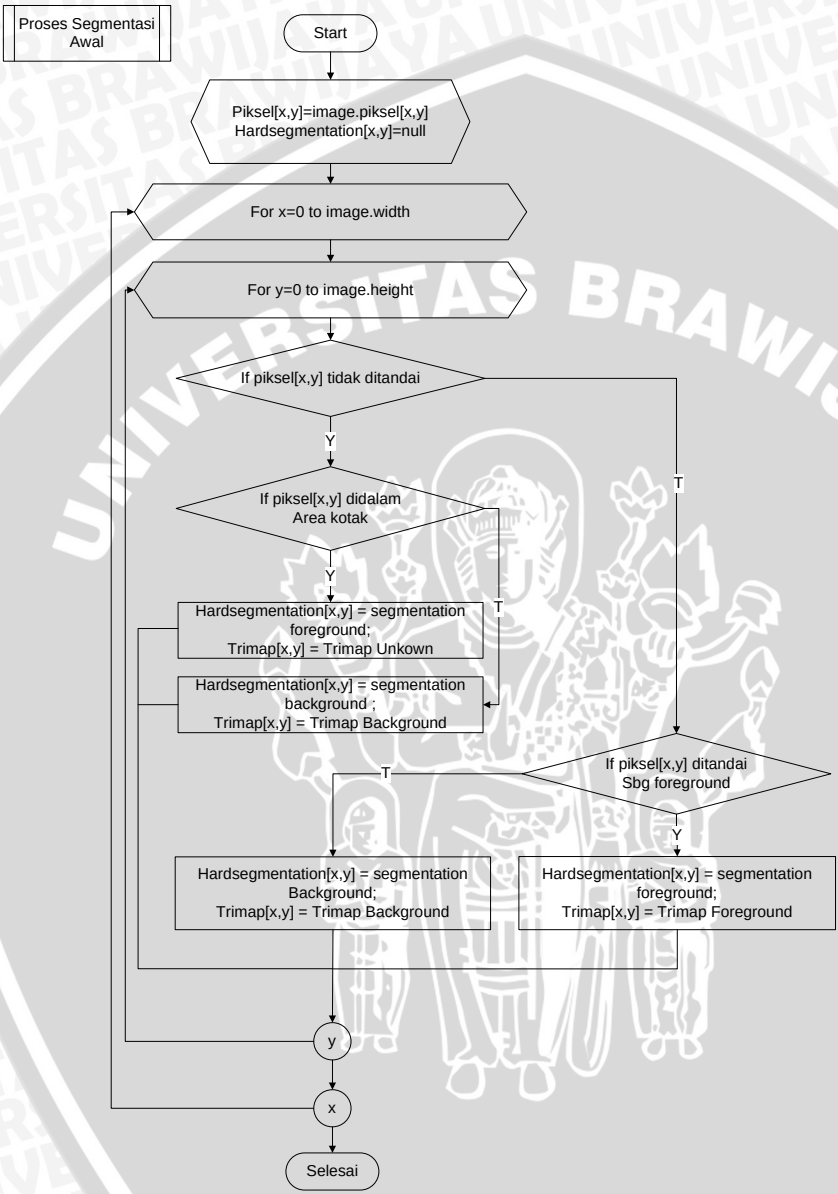
Inisialisai *trimap* merupakan proses awal yang dilakukan oleh *user* seperti pada keterangan pada poin satu pada sub bab 3.3.2. dalam metode *grabcut* terdapat tiga *trimap* diantarnya *trimap background*, *trimap foreground* dan *trimap unknown. Trimap foreground* digunakan untuk mengumpulkan piksel yang sudah

dipastikan bahwa piksel tersebut adalah piksel *foreground*. *Trimap background* digunakan untuk menampung piksel yang sudah dipastikan bahwa piksel tersebut merupakan piksel *background.*. Sedangkan *trimap unkown* digunakan untuk menampung piksel yang masih belum dapat dipastikan apakah piksel tersebut merupakan piksel anggota *foreground* atau *background.*

Selain *trimap* terdapat pengelompokan kasar (*hard segmentation*). Pengelompokan ini digunakan untuk menampung dan memproses sementara piksel-piksel pada gambar. Terdapat dua kelompok *hard segmentation,* yang pertama adalah segmen *foreground* (sementara menjadi anggota piksel *foreground*) dan segmen *background.* Piksel anggota dari *trimap* pasti merupakan anggota dari segmen (*hard segmentation*) yang sesuai dan tidak pasti sebaliknya.

3.3.2.2 Segmentasi Awal

Pada proses ini dua struktur data yang akan digunakan pada proses-proses stelahnya antara lain HardSegmentation[x,y] = segmementasi *background* atau segmentasi *foreground* , dinmana x, y merupakan indeks yang memiliki nilai maksimal sejumlah ukuran panjang dan lebar piksel pada citra. *Hard segmentation* akan bernilai *segmentation foreground* jika indeks x, y mengacu pada piksel didalam area kotak yang dibuat oleh user dan tidak ditandai sebagai piksel *background.* Sedangkan piksel diluar area kotak dikelompokkan menjadi *hard segmentation background*. Untuk piksel yang sudah ditandai secara spesifik langsung dikelompokkan kedalam kelompok yang sesuai berdasarkan *input* yang dimasukkan oleh *user.* Adapun *flowchart* dari proses segmentasi awal seperti yang dijelaskan pada gambar 3.5.



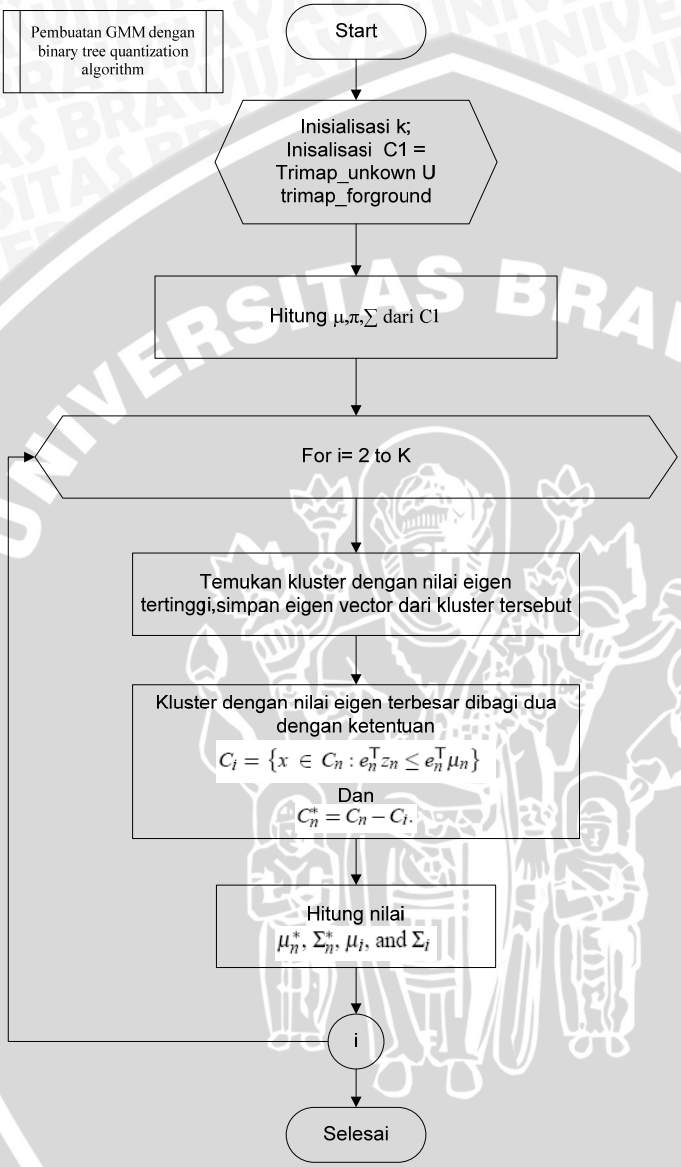
Gambar 3.5 *flowchart* segmentasi awal

3.3.2.3 Pembuatan GMM dengan *Binary Tree Color Quantization Algorithm*

Beberapa simbol yang digunakan pada proses GMM mulai dari pembuatan komponen GMM dengan *binary tree color quantization algorithm* hingga proses GMM selanjutnya antara lain:

k : jumlah *gaussian* komponen
$\sum$: matriks kovarian dari kluster ke n
Det $\sum$ atau $|\sum|$: determinan dari matriks kovarian dari kluster ke n
$\sum^{-1}$: invers dari matriks kovarian (3x3 matrix) dari kluster ke n
μ : nilai rata-rata warna (terdiri dari nilai RGB) dari kluster ke
π : bobot dari setiap gausian komponen dari kluster ke n
z : warna dari piksel (terdiri dari nilai RGB) dari kluster ke n
e_n : nilai eigen dari kluster ke n
e_n^T *:* vektor eigen dari kulster ke n
C : Kluster set

adapun *flowchart* secara global dari proses ini terdapat pada gambar 3.6.



Gambar 3.6 Flowchart Pembuatan *GMM dengan Binary Tree Color Quantization Algorithm*

Penjelasan alur algoritma yang terdapat pada Gambar 3.6 adalah sebagai berikut:

a. Inisialisasi *Cluster* Pertama

Piksel di dalam area kotak yang tidak ditandai secara spesifik apakah termasuk *foreground* atau *background* dimasukkan ke dalam kluster pertama pada komponen *foreground* GMM (*hard segmentation).* Sedangkan piksel diluar kotak yang tidak ditandai dimasukkan kedalam kluster pertama pada komponen *background* GMM (*hard segmentation).* Untuk piksel yang sudah ditandai, dimasukkan ke dalam kluster pertama komponen GMM yang sesuai (*Trimap*).

b. Pembuatan Matriks Kovarian dan Perhitungan μ

Variabel yang menjadi data pengamatan pada GMM merupakan piksel pada gambar dimana dalam tiap piksel tersebut terdapat komponen warna R,G,B. Jika variabel warna ini dinyatakan dengan X, maka sesuai dengan persamaan 2.20. Secara singkat perhitungan matriks kovarian dapat dijabarkan sebagai birikut**,** sebelumnya dibuat matriks P (3x3) untuk mempermudah perhitungan dengan cara :

P[0][0] = ∑(piksel merah x piksel merah)
P[1][0] = ∑(piksel hijau x piksel merah)
P[2][0] = ∑(piksel biru x piksel merah)
P[0][1] = ∑(piksel merah x piksel hijau)
P[1][1] = ∑(piksel hijau x piksel hijau)
P[2][1] = ∑(piksel biru x piksel hijau)
P[0][2] = ∑(piksel merah x piksel biru)
P[1][2] = ∑(piksel hijau x piksel biru)
P[2][2] = ∑(piksel biru x pikse biru)

Maka nilai konvarian didapatkan dengan cara :

kovarian[0][0] = (p[0][0] / jumlah piksel dalam komponen) – (rata-rata warna merah * rata-rata warna merah) + Epsilon

kovarian[0][1] = (p[0][1] / jumlah piksel dalam komponen) - (rata-rata warna merah * rata-rata warna hijau)
kovarian[0][2] = (p[0][2] / jumlah piksel dalam komponen) - (rata-rata warna merah * rata-rata warna biru)
kovarian[1][0] = (p[1][0] / jumlah piksel dalam komponen) - (rata-rata warna hijau * rata-rata warna merah)
kovarian[1][1] = (p[1][1] / jumlah piksel dalam komponen) - (rata-rata warna hijau * rata-rata warna hijau) + Epsilon
kovarian[1][2] = (p[1][2] / jumlah piksel dalam komponen) - (rata-rata warna hijau * rata-rata warna biru
kovarian[2][0] = (p[2][0] / jumlah piksel dalam komponen) - (rata-rata warna biru * rata-rata warna merah)
kovarian[2][1] = (p[2][1] / jumlah piksel dalam komponen) - (rata-rata warna biru * rata-rata warna hijau)
kovarian[2][2] = (p[2][2] / jumlah piksel dalam komponen) - (rata-rata warna biru * rata-rata warna biru) + Epsilon

Agar determinan matriks kovarian tidak bernilai nol, maka ditambahkan suatu nilai kecil (epsilon) pada diagonal utama matriks kovarian.

Sedangkan nilai μ = $\sum$nilai R|G|B| dibagi dengan jumlah piksel dalam komponen, π = jumlah piksel dalam komponen *gaussian* dibagi dengan jumlah piksel keseluruhan pada GMM.

Determinan dan invers dari matriks kovarian dihitung menggunkan persamaan 2.14, sedangkan untuk perhitungan invers matriks digunakan persamaan 2.16.

c. Pembentukan Cluster

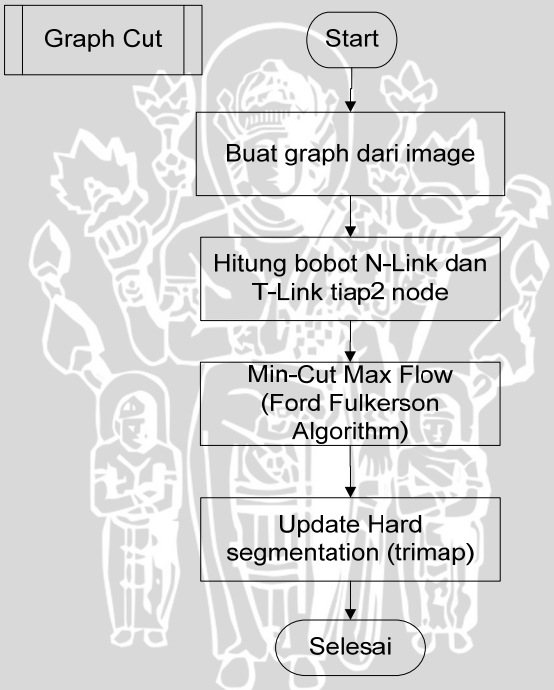
Mekanisme pemecahan kluster (komponen) digunakan nilai eigen, kluster yang mempunyai nilai eigen terbesar yang akan dipecah. Tiap piksel dalam kluster yang akan dipecah diperhitungkan masuk kluster baru atau kluster lama berdasarkan ketentuan titik pisah menggunakan vektor eigen dari matriks kovarian. Dengan ketentuan titik pisah = vektor eigen [0] x rata-rata warna merah +

vektors eigen [1] x rata-rata warna hijau + vektor eigen [2] x rata-rata warna biru.

Setelah itu penentuan tiap piksel dengan ketentuan vektor eigen [0] x warna merah + vektor eigen [1] x warna hijau + vektor eigen [2] x warna biru > titikpisah ?, jika lebih besar maka piksel tersebut akan dimasukkan pada kluster baru, kluster lama (Cn^*) menjadi Cn – Ci.

3.3.2.5 Proses *Graph Cut*

Proses *grapch cut* merupakan bagian inti dari proses - proses yang ada, adapun *flowchart* pada tahapan ini terdapat pada gambar 3.7.



Gambar 3.7 *Flowchart* proses *graphcut*

Penjelasan masing – masing proses pada tahapan *Grapcut* antra lain:

a. Pembuatan *Graph* dari *Image*

Penulis menggunakan struktur data *Adjency List* untuk menyusun *Graph*. Di dalam *Graph* terdapat *Node* sejumlah piksel yang berada dalam gambar. Terdapat dua macam garis (*edge*) yang menghubungkan antara piksel yang pertama merupakan *N-Link*, dimana *N-Link* menghubungkan tiap piksel dengan 8 piksel tetangganya, setiap *N-link* memiliki bobot yang dihitung dengan persmaan 2.4 dan persamaan 2.5. Garis penghubung yang kedua adalah *T-Link*, *T-Link* ini yang menghubungkan dengan salah satu dari dua *node* spesial yaitu *Node* S (*Source*) atau *Node* T(*Sink*). Dalam implementasinya dua *node* spesial ini dikenali dengan memiliki indeks -1,dan -2, sedangkan *node* yang lainnya memiliki indeks sesuai dengan koordinat posisi piksel pada gambar.

N-Link, seperti yang telah dijelaskan sebelumnya, menghubungkan tiap piksel dengan piksel tetangganya. Secara teknis tiap *Node* dihubunkan secara langsung berarah ke piksel sebelah kanan, kanan atas, atas, kiri atas, sehingga jika terkontruksi secara total tiap piksel akan berhubungan dengan 8 piksel tetangganya.

b. Perhitungan Bobot *N-Link* Sebagai Nilai *Edge* dari *Graph*

Sebagaimana yang dijelaskan pada poin a, *N-Link* menghubungkan *Node* dengan 8 *Node* tetangganya. Untuk perhitungan *N-Link* digunakan persamaan 2.4 dan 2.5, dimana Z merupakan *Node* yang mewakili piksel (terdiri dari warna R,G,B). adapun implementasi dari persamaan tersebut adalah :

T-Link dihitung dengan persamaan 2.8 atau 2.9, tergantung apakah piksel tersebut merupakan anggota dari *trimap foreground*, *background* atau *unknown*. Untuk *trimap forground* dan *background* digunakan persamaan 2.8, sedangkan untuk *trimap unkown* diguanakan persaaman 2.9.

c. *Min-Cut Max Flow* dengan *Ford Fulkerson Algorithm*

Tujuan dari implementasi algoritma ford Fulkerson adalah untuk mencari nilai total *flow* maksimal dari total *flow* maksimal yang bisa dilewatkan pada *augmenting path*, *augmenting path* merupakan jalur dari *node source* menuju *node sink*. Algoritma ini menemukan nilai *flow* maksimal jika tidak ada lagi *augmenting path* yang ditemukan.

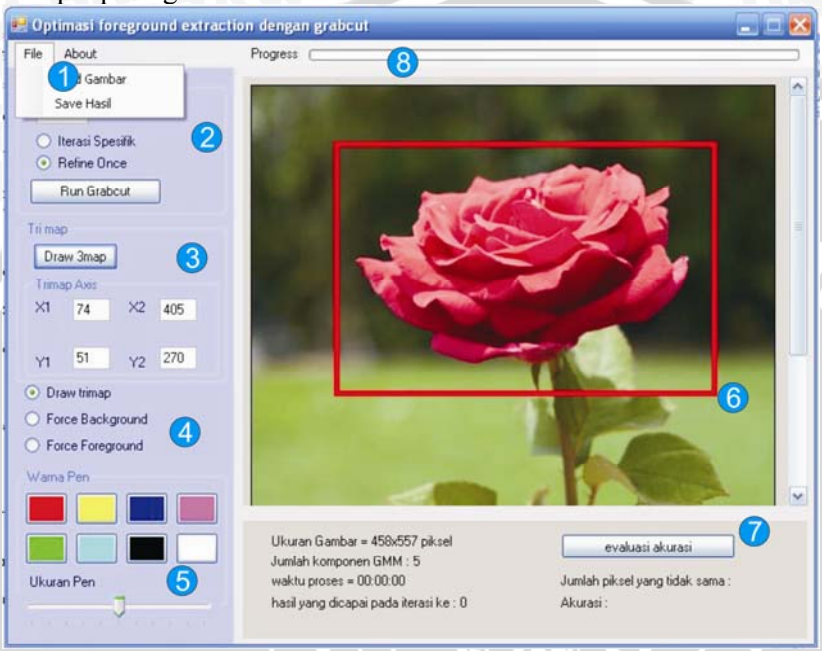
Setelah algoritma ini dijalanakan, titik potong *graph* merupakan *edge* yang mempunyai bobot kurang dari sama dengan *nol* dimana *cost* total titik potong minimal yang didapatkan pada akhir algoritma ini adalah sama dengan total nilai dari maksimal *flow* yang didapatkan. Sehigga pada proses ini ditemukan titik pemisah graph yang terdiri dari kumpulan edge yang berkapasitas kurang dari sama dengan nol.

d. **Proses *Update Hard Segmentation***

Proses *update hard segementation* merupakan penyesuaian ulang elemen *hard segmentation*. piksel yang termasuk kedalam *tree source* (S) akan menjadi anggota dari *hard segementation foreground* sedangkan yang termasuk dalam *tree Sink (T)* akan menjadi anggota dari *hard segementation* background. Tiap iterasi proses *grabcut* elemen – elemen *hard segmentation* di-*update*.

3.3.3 Desain Antarmuka

Gambaran umum desain antar muka yang akan dirancang terdapat pada gambar 3.9.



Gambar 3.9 Desain Antar Muka

Pada gambar 3.9 ada beberapa bagian penting di dalam desain antar muka yang dibagai menjadi 8 poin yang diantaranya adalah:

1. Merupakan menu utama yang terdapat dalam aplikasi menu ini berisi perintah untuk *load file* gambar dan untuk menyimpan hasil ekstraksi *foreground.* Di sebelahnya terdapat menu *about* yang menjelaskan secara singkat terkait dengan aplikasi *foreground exstraction* ini.
2. Terdapat tombol *run grabcut* merupakan tombol untuk memulai proses ekstraksi *foreground* dengan menggunakan metode *grabcut*. Sebelum menjalankan proses ekstraksi, semua parameter yang dibutuhkan harus sudah terisi, diantaranya *input* citra digital dan wilayah *trimap.*
3. Merupakan tombol untuk menggambar *trimap* dengan kordinat yang sudah dimasukkan. Koordinat relatif terhadap titik 0, 0

pada gambar.

4. Terdapat beberapa pilihan interaksi *user* dengan gambar, diantaranya *draw trimap* dimana *user* memungkinkan untuk menggambar *trimap* dengan menggunakan *drag mouse* pada wilayah gambar yang diinginkan. *Force foreground* digunakan untuk menandai wilayah *foreground* pada gambar sedangkan *force background* digunakan untuk menandai wilayah *background* pada gambar.
5. Merupakan *palatte color* yang digunakan untuk memilih warna *pen* yang digunakan pada operasi poin nomor 4.
6. Merupakan area gambar.
7. Berisis informasi hasil operasi *grabcut* yang sudah dilakukan. Terdapat *button* yang digunakan untuk membandingkan hasil ekstraksi dengan gambar acuan, sehingga didapatkan nilai akurasi.
8. Merupakan *progress bar* yang menunjukkan jalannya proses ekstraksi.

3.4 Sekenario Evaluasi Sistem dan Analisa Hasil

Dari penelitian yang dilakukan ada beberapa parameter percobaan yang akan dianalisa, adapun parameter tersebut akan dijelaskan pada sub bab berikutnya.

3.4.1 Uji Keakuratan Ekstraksi

Dalam uji akurasi ini dicoba ekstraksi *foreground* dalam berbagai tingkat kesulitan gambar. Mulai dari gambar yang antara *foreground* dengan *background*-nya mudah dibedakan hingga yang memiliki homogenitas warna antara *foreground* dengan *bacakground* hampir sama sehingga sulit untuk dibedakan. Untuk mengukur keakuratan ekstraksi diguanakan persamaan 2.20, dimana terdapat citra acuan yang menjadi standar ketepatan akurasi, citra acuan ini merupakan hasil ekstraksi *foreground* manual dengan menggunakan bantuan *software photoshop*. Nilai akurasi didapatkan dengan membandinkan jumlah piksel yang tidak sama dengan gambar acuan dengan jumlah keseluruhan piksel. Semakin banyak jumlah piksel yang tidak sama maka nilai akurasi akan semakin rendah. Untuk mendapatkan nilai akurasi digunakan persamaan 2.20.

Tabel 3.1 Analisa Keakuratan Ekstraksi

Nama File	Ukuran	Koordinat Trimap				Jumlah piksel tidak diharapkan	% akurasi
		X1	Y1	X2	Y2		
a.jpg	300x400						
b.bmp	300x400						
c.jpg	300x400						
d.jpg	600x300						

3.4.2 Uji Kecepatan Ekstraksi Setiap Penambahan Ukuran Gambar dan *Trimap*

Untuk pengujian kecepatan ekstraksi, digunakan satu citra digital yang sama tetapi dengan ukuran yang berbeda. Terdapat enam ukuran gambar yang berbeda dengan masing-masing penambahan ukuran 120%.

Tabel 3.2 Analisa Kecepantan Ekstraksi

No	Ukuran Gambar		Koordinat Trimap				Waktu(detik)
	Panjang	Lebar	X1	Y1	X2	Y2	
1	200	133	17	23	130	107	13
2	240	160	20	28	156	128	24
3	288	192	24	33	187	154	47

3.4.3 Uji Pengaruh Luas Area *Trimap* Terhadap Hasil Ekstraksi dan Kecepatan Ekstraksi

Untuk pengujian ini digunakan satu citra digital dengan *input* luas *trimap* yang berbeda beda.

Tabel 3.3 Analisa Kecepantan Ekstraksi

No	Ukuran Gambar		Koordinat Trimap				T	Akurasi
	Ukuran gammbar	Luas trimap	X1	Y1	X2	Y2		
1	200	133	17	23	130	107	13	

3.5 Perhitungan Manual

Pada perhitungan manual ini dijelaskan secara garis besar komputasi yang dilakukan dalam penerapan metode *grabcut* yang digunakan dalam teknik *foreground extraction* dengan algoritma dan alur program yang sudah dijelaskan sebelumnya. untuk *sample* data digunakan citra berukuran 5x5 piksel dengan distribusi warna [r,g,b] *random* seperti yang terdapat pada gambar 3.11.

x, y	0	1	2	3	4
0	84,141, 212	141,179, 226	141,179, 226	141, 179,226	84, 141, 212
1	84,141,212	227, 108, 10	227, 108, 10	250,191, 143	198, 217, 241
2	84,141,212	227, 108, 10	227, 108, 10	250, 191,143	198, 217, 241
3	84, 141,212	255, 192, 0	255, 192, 0	152, 72, 6	198, 217, 241
4	84, 141,212	0, 176, 240	0, 176, 240	0, 176, 240	198, 217, 241

Gambar 3.11 *Sample Image*

Penjelasan perhitungan manual berdasarkan *flowchart grabcut* secara umum (gambar 3.4) dan langkah-langkah yang akan dilakukan dijelaskan pada subbab berikutnya.

3.5.1 Manual Inisialisasi *Trimap* Oleh *User*

Dalam tahapan ini *user* membuat sebuah *trimap* seperti yang terlihat pada gambar 3.12, wilayah didalam kotak merah adalah wilayah *unknown*, sedangkan diluar kotak merah adalah wilayah *background.*

x,y	0	1	2	3	4
0	84,141,212	141,179,226	141,179,226	141,179,226	84,141,212
1	84,141,212	227, 108, 10	227, 108, 10	250, 191,143	198, 217,241
2	84,141,212	227, 108, 10	227, 108, 10	250, 191,143	198, 217, 241
3	84,141,212	255, 192, 0	255, 192, 0	152, 72, 6	198, 217, 241
5	84,141,212	0, 176, 240	0, 176, 240	0, 176, 240	198, 217, 241

Gambar 3.12 *Trimap* Yang Dibuat Oleh *User*

3.5.2 Manual Proses Segmentasi Awal (*Hard Segmentation*)

Dalam segmentasi awal dipakai struktur data yang bernama hard_segmentation jumlahnya sesuai dengan dimensi gambar

For x=0 to lebar citra
For y=0 to panjang citra
jika piksel(x, y) di dalam Area kotak dan tidak ditandai sebagai area background maka
pikse(x, y) .hardsegmentation= *segmentation foreground*
Atau jika tidak maka
pikse(x, y) .hardsegmentation = *segmentation background*
end for

dari ketentuan pada alur diatas maka didapat anggota dari *segmentation background* adalah piksel dengan indeks [0, 0],[0, 1] [0, 2], [0, 3], [0, 4], [1, 0], [2, 0], [3, 0], [4, 0], [4, 1], [4, 2], [4, 3], [4, 4], [1, 4], [2, 4], [3, 4]. Sedangkan *segmentation foreground* memiliki anggota berindeks [1, 1], [1, 2], [1, 3], [2, 1], [2, 2], [2, 3], [3, 1], [3, 2], [3, 3].

3.5.2 Manual Pembuatan GMM dengan *Binary Tree Color Quantization Algorithm*

Langkah awal adalah membuat komponen GMM yang pertama. Di dalam setiap komponen GMM terdapat beberapa symbol yang digunakan yaitu :
$\sum$ = kovarian matriks yang mewakili komponen GMM tersebut
Det $\sum$ = determinan dari matriks kovarian
$\sum^{-1}$ = inverse dari kovarian matrix (3x3 matrix)
μ = Rata rata (dari RGB)
π = bobot dari setiap gausian komponen
e = nilai eigen dari masing masing kovarian matriks
count = jumlah piksel dalam komponen

Komponen *gaussian* yang akan dibuat berjumlah k, total ada 2k komponen *gaussian* (masing-masing kelas baik *background* mapun *foreground* mempunyai k komponen *gaussian*), pada contoh perhitungan manual ini menggunakan nilai k = 2.

2.5.2.1 Pembuatan komponen *Gaussian* untuk GMM *foreground*

Untuk komponen pertama merupakan gabungan antara kelas *foreground* dan kelas *unknown*, nilai epsilon yang digunakan menggunakan epsilon = 0.001 adapun tahapan dalam proses ini adalah :

Langkah 1 : Pembuatan Matrik Kovarian (*Cluster* I)

Proses pembuatan kovarian matriks mengacu pada persamaan 2.15 dan 2.16, untuk memudahkan dalam pembuatan kovarian matriks maka terlebih dahulu dibuat matriks produk P 3x3, hasil dari matrik produk komponen pertama untuk kelas *foreground* yang memiliki indeks piksel [1, 1][1, 2][1, 3][2, 1][2, 2][2, 3][3, 1] [3, 2][3, 3] adalah :

P[0][0]=227 * 227 + 227 * 227 + 255 * 255 + 227 * 227 + 227 * 227 + 255 * 255 + 250 * 250 + 152 * 152 = 421770

P[0][1] = 227 * 108 + 227 * 108 + 255 * 192 + 227 * 108 + 227 * 108 + 255 * 192 + 250 * 191 + 250 * 191 + 152 * 72 = 302428

P[0][2] = 227 * 10 + 227 * 10 + 255 * 0 + 227 * 10 + 227 * 10 + 255 * 143 + 250 * 143 +150 * 6 = 82195

P[1][0] = 302428

P[1][1] = 10 * 10 + 10 * 10 + 0 * 0 + 10 * 10 + 10 * 10 + 0 * 0 + 143 * 143 + 143 * 143 + 6 * 6 = 41334

P[1][2] = 108 * 10 + 108 * 10 + 190 * 0 + 108 * 10 + 108 * 10 + 190 * 0 + 191 * 143 + 191 * 143 + 72 * 6 = 59378

P[2][0] = 82195

P[2][1] = 59378

P[2][2] = 108 * 108 + 108 * 108 + 192 * 192 + 108 * 108 + 108 * 108 + 192 * 192 + 191 * 191 + 191 * 191 + 72 * 72 = 198530

Setelah matriks P 3x3 didapatkan, selanjutnya adalah menghitung rata-rata nilai masing - masing RGB dalam setiap komponen gaussian, jumlah piksel dalam komponen = 9, maka didapatkan rata-rata merah = 227 + 227 + 227 + 227 + 255 + 255 + 250 + 250 + 152 = 2070 / 9 = 230. rata-rata hijau = 108 + 108 + 108 + 108 + 192 + 192 + 191 + 191 + 72 = 1270 / 9 = 141. Rata-rata biru

= 10 + 10 + 0 + 10 + 10 + 0 + 143 + 143 + 0 = 36.

Setelah mendapatkan nilai matriks P 3x3, rata rata tiap RGB, untuk mendapatkan matriks kovarian adalah dengan ketentuan pada persamaan 2.15 dan 2.16. nilai epsilon ditambahkan agar matriks tidak singular. Dari ketentuan tersebut maka didapat matriks kvarian 3x3 adalah :

kovarian[0][0]= ((421770/9) – (230*230)) + 0.001 = -6036.36
kovarian[0][1]= (302428/9) –(230*141) =1173.111
kovarian[0][2]= (82195/9)–(230 *36) = 852.7778
kovarian[1][0]= (302428/9) –(141*230) =1173.111
kovarian[1][1]= (41334/9) –(141*141) + 0.001 =-15288.3
kovarian[1][2]= (59378/9) –(141*36) = 1521.556
kovarian[2][0]= (82195/9) –(36 *230) = 852.778
kovarian[2][1]= (59378/9)–(36 *141) = 1521.556
kovarian[2][2]= (198530/9)-(36 *36) + 0.001 = 20762.89

adapun matriks kovarian yang terbentuk adalah :

$$Kovarian = \begin{bmatrix} -6036.36 & 1173.111 & 852.7778 \\ 1173.111 & -15288.3 & 1521.556 \\ 852.778 & 1521.556 & 20762.89 \end{bmatrix}$$

Invers dari matrik kovarian, dangen menggunakan acuan rumus 2.13 didapat Adjoin matriks(adj A) adalah

$$Adj(A) = \begin{bmatrix} -319744423.84 & -23059625.47 & 14822476 \\ -23059625.16 & -126052034.18 & 10184515 \\ 14822479 & 10184515 & 90903989 \end{bmatrix}$$

Sedangkan determinan matriks adalah 1915566124272.91

Maka didapat nilai invers matriks = **(adj A)/|A| =**

$$\begin{bmatrix} -0.000166919022 & -0.000012038021 & 0.000007737909 \\ -0.000012038021 & -0.000065804063 & 0.000005316713 \\ 0.000007737911 & 0.000005316713 & 0.000047455417 \end{bmatrix}$$

Langkah 2: Perhitungan π,μ Nilai Eigen dan Vektor Eigen dari Matriks Kovarian

π = jumlah piksel dalam komponen Gaussian/ jumlah piksel keseluruhan = 9/25 = 0.36. nilai μ, seperti yang sudah dihitung sebelumnya rata rata masing masing nilai RGB adalah (230,141,36). nilai eigen dari kovarian matriks didapat nilai eigen = (-5929.72,-15489.28,20857.24). vektor eigen untuk nilai eigen -5929.72 adalah <-0.992254, -0.118132, 0.038434> . Vektor eigen untuk nilai eigen ,-15489.28 adalah <0.119613, -0.992061, 0.038825>. Vektor eigen untuk nilai eigen , 20857.24 adalah < -0.033543, -0.043121, -0.998507>

Langkah 3: Pecah Kluster Berdasarkan Nilai dan Vektor Eigen

Split gaussian komponen berdasarkan nilai eigen dan eigen vektor hingga jumlah *gaussian* komponen yang dikehendaki terpenuhi. Pada langkah ke 3 ini, dari komponen Gaussian yang sudah terbentuk dicari komponen mana yang mempunyai nilai eigen terbesar yang akan dipilih sebagai komponen yang akan dipecah. Dalam perihitungan manual ini karena hanya ada satu komponen maka komponen tersebut yang akan dipecah berdasarkan ketentuan eigen vektor dipilih dari nilai eigen terbesar pada komponen tersebut, titik_pisah = eigenvektor[0][0] * rata-rata warna merah + eigenvektors[1][0] * rata-rata warna hijau + eigenvektor[2][0] * rata-rata warna biru. Selanjutnya piksel diseleksi, jika hasil perkalian warna (RGB) dengan vektor eigen lebih besar dari titik pisah maka piksel akan dimasukkan kedalam kluster baru, tapi jika tidak piksel masih ditempatkan pada kluster lama.

Kumpulan piksel kluster satu adalah [1, 1][1, 2][1, 3][2, 1][2, 2][2, 3] [3, 1][3, 2][3, 3] , yang memiliki nilai RGB berturut turut[227, 108, 10], [227, 108, 10],[255, 192, 0],[227, 108, 10],[227, 108, 10], [255, 192, 0],[250,191, 143],[250, 191,143],[152, 72, 6]

Perhitungan untuk titik_pisah :
titik_pisah = -0.033543*230+-0.043121*141+-0.998507*36
= -49.7 2

Penentuan untuk tiap piksel
eigenvektor[0][0] * warna merah + eigenvektor[1][0] * warna hijau + eigenvektor[2][0] * warna biru > titikpisah ?
piksel [1,1] = -0.033543*227+-0.043121*108+-0.998507*10 = -22.25 > titik pisah maka

piksel[1,1] = kluster_baru
Piksel[1,2] = kluster_baru
Piksel[1,3] = -0.033543 * 255 + -0.043121 * 192 + -0.998507 * 0 = -17.83 > titik pisah, maka
piksel [1,3] = kluster_baru
Piksel[2,1] = kluster_baru
Piksel[2,2] = kluster_baru
Piksel[2,3] = kluster_baru
Piksel[3,1] = -0.033543 * 250 + -0.043121 * 191 + -0.998507 * 143 = 159.40 < titik_pisah , maka
piksel [3,1] = kluster_lama
Piksel[3,2] = kluster_lama
Piksel[3,3] = kluster_baru

Dari perhitungan tersebut didiapat anggota kluster lama terdiri dari piksel [3,1], [3,2]. Sedangkan anggota kluster baru terdiri dari piksel [1,1], [1,2], [1,3], [2,1], [2,2], [2,3], [3,3].

Langkah 4 : Ulangi Langkah 1 dan 2 Hingga Jumlah Kluster Terpenuhi

Dari kluster yang terbentuk masing-masing lakukan langkah 1 dan langkah 2, saat ini anggota dari kluster pertama adalah [3,1], [3,2] yang memiliki nilai RGB berturut-turut [250,191, 143],[250, 191,143] . saat ini anggota dari kluster kedua adalah [1,1], [1,2], [1,3], [2,1], [2,2], [2,3], [3,3] yang memiliki nilai RGB berturut-turut [227, 108, 10] ,[227, 108, 10],[255, 192, 0],[227, 108, 10],[227, 108, 10], [255, 192, 0],[152, 72, 6].

Buat matriks kovarian untuk kluster pertama
kovarian1[0][0] = (12500/2) - 250*259 + 0.001 = -5850
kovarian1[0][1] = (95500/2) -250*191 = 0

kovarian1[0][2] = (95500/2) - 250*143 = 12000
kovarian1[1][0] = (72963/2)-191 *250 = -11268.5
kovarian1[1][1] = (71500/2) -191*191 + 0.001 =-730.999
kovarian1[1][2] = (54626/2) -191 *143 = 0
kovarian1[2][0] = (71500/2) -143*250 = 0
kovarian1[2][1] = (54626/2) -143*191 = 0
kovarian1[2][2] = (40898/2) -143 *143 + 0.001 = 0.001

matriks kovarian dari kluster satu adalah :

$$\begin{bmatrix} 5850 & -11268.5 & 0 \\ 0 & -730.999 & 0 \\ 12000 & 0 & 0.001 \end{bmatrix}$$

Adjoint dari matriks kovarian diatas adalah :

$$\begin{bmatrix} -0.731 & 11.26 & 0 \\ 0 & 5.85 & 0 \\ 8771988 & 135222000 & -4276344.15 \end{bmatrix}$$

determinan dari matriks kovarian diatas adalah -4276.34.

invers dari matriks kovarian diatas adalah = **(adj A)/|A|**

$$\begin{bmatrix} 0.00017094 & -0.002635078 & 0 \\ 0 & -0.001367991 & 0 \\ -2051.28 & 31620.93 & 1000 \end{bmatrix}$$

Π pada kluster pertama = 2/9 = 0.22
μ_1 pada kluster pertama = R = (250+250)/2 = 250 , G = 191, B= 143

Buat matriks kovarian untuk kluster kedua

Saat ini anggota dari kluster kedua adalah [1,1], [1,2], [1,3], [2,1], [2,2], [2,3], [3,3] yang memiliki nilai RGB berturut-turut [227,108, 10], [227, 108, 10], [255, 192, 0], [227, 108, 10], [227,108,10], [255,192 0],[152, 72, 6]

p2[0][0] = 227*227 + 227*227 +227*227 +255*255+227*227 +255*255+152*152 =359270
p2[0][1] = 227*108+227*108+255*192+227*108+227*108+ 255*192+152*72 = 206928
p2[0][2] = 227*10+227*10+255*0+227*10+227*10+255*0+125*6 = 9830
p2[1][0] = 227*108+227*108+255*192+227*108+227*108 +255*192+152*72 = 206928
p2[1][1] = 108*108+108*108+192*192+108*108+108*108+ 192*192+72*72 = 125568
p2[1][2] = 108*10+ 108*10+192*0+ 108*10+ 108*10+192*0+72*6 = 4752
p2[2][0] = 227*10+227*10+255*0+227*10+227*10+255*0+125*6 = 9830
p2[2][1] = 108*10+ 108*10+192*0+ 108*10+ 108*10+192*0+72*6 = 4752
p2[2][2] = 10*10+10*10+0*0+10*10+10*10+0*0+6*6 = 436
[227, 108, 10] ,[227, 108, 10],[255, 192, 0],[227, 108, 10],[227, 108, 10], [255, 192, 0], [152, 72, 6]

Rata-rata merah = (227+227+255+227+227+255+152) /7 = 224.29;
Rata-rata hijau = (108+108+192+108+108+192+72)/7 = 126.86;
Rata-rata biru = (10+10+0+10+10+0+6)/7 = 6.57 ;
count =7 ;

kovarian2[0][0] = (359270/7) - 224.29*224.29+ 0.001 = 1018.28
kovarian2[0][1] = (206928/7) -224.29*126.86= 1107.71
kovarian2[0][2] = (9830/7) - 224.29*6.57 = -69.29
kovarian2[1][0] = (206928/7)- 126.86*224.29= 1107.71
kovarian2[1][1] = (125568/7) -126.86*126.86+ 0.001 = 1844.82
kovarian2[1][2] = (4752/7) -126.86*6.57 = -154.61
kovarian2[2][0] = (9830/7) -6.57 *224.29= -69.29
kovarian2[2][1] = (4752/7) -6.57 *126.86= -154.61
kovarian2[2][2] = (436/7) -6.57 *6.57 + 0.001 = 19.12

matriks kovarian dari kluster dua yang terbentuk adalah:

Kovarian2 = $\begin{bmatrix} 1018.28 & 1107.71 & -69.29 \\ 1107.71 & 1844.82 & -154.61 \\ -69.29 & -154.61 & 19.12 \end{bmatrix}$

Determinan dari matriks kovarian2 adalah : 2992335.88

Adjoin dari matriks kovarian2 adalah

$$\begin{bmatrix} 11368 & -10466.48 & -43435.46 \\ -10466.48 & 14668.40 & 80683.04 \\ -43435.46 & 80683.04 & 651521.86 \end{bmatrix}$$

Invers dari matriks kovairan2 adalah

$$\begin{bmatrix} 0.0038 & -0.0035 & -0.0145 \\ -0.0035 & 0.0049 & 0.0269 \\ -0.0145 & 0.0269 & 0.2177 \end{bmatrix}$$

2.5.2.2 Perhitungan *T-Link* dan *N-Link*

Graph dibentuk dan hitung bobot *N-Link* dan *T-Link* sperti pada persamaan 2.8 dan 2.9 untuk tiap tiap piksel dan jalankan mekanisme *graphcut.* Ilustrasi proses *graphcut* menggunakan mekanisme *min-cut max-flow* terdapat pada gambar 3.14. mekanisme perhitungan adalah sebagai berikut:

a. **Perhitungan *T-Link***

Tiap piksel dihitung bobot T-Link, untuk T-link yang menhubungkan dengan Node S Maupun T, contoh perhitungan digunakan perhitungan T-Link Untuk satu piksel pada indeks ke 1,1 dengan nilai R,G,B [227,108,10], persamaan yang digunakan untuk menghitung nilai T-Link adalah persamaan 2.8

***T-Link* untuk *Foreground* GMM**

Perhitungan Pada kluster I

masing-masing nilai det($\sum$), π, μ, $\sum^{-1}$ untuk tiap kluster satu adalah

Det($\sum_1$)= -4276.34

Π_1 = 0.22

μ_1 = (250,191,143)

Z = (227,108,10)

$$\sum^{-1} = \begin{bmatrix} 0.00017094 & -0.002635078 & 0 \\ 0 & -0.001367991 & 0 \\ -2051.28 & 31620.93 & 1000 \end{bmatrix}$$

$$Z - \mu = \begin{bmatrix} 250 - 227 \\ 191 - 108 \\ 143 - 10 \end{bmatrix} = \begin{bmatrix} 23 \\ 83 \\ 133 \end{bmatrix}$$

$$(Z - \mu)^T = [23 \quad 83 \quad 133]$$

$$\sum^{-1} * (Z - \mu) = \begin{bmatrix} -0.21 \\ -0.11 \\ 0 \end{bmatrix}$$

$$-\frac{1}{2}(Z - \mu)^T * \left(\sum^{-1} * (Z - \mu)\right) = [23 \quad 83 \quad 133] * \begin{bmatrix} -0.21 \\ -0.11 \\ 0 \end{bmatrix}$$

$$= \left(-\frac{1}{2}\right)(-751.6) = 375.8$$

$$\frac{1}{\sqrt{det\sum i}} e^{375.8} * \pi = \frac{1}{\sqrt{-4276.34}} * (1.6139\text{E} + 163) * 0.22$$

$$= 5.4294\text{E} + 160$$

Perhitungan Pada Kluster II

masing-masing nilai det(∑), π, μ, $\sum^{-1}$ untuk tiap kluster satu adalah

Det($\sum_1$)= 2992335.88

Π_1 = 7/9 = 0.77

μ_1 = (227,127,7)

Z = (227,108,10)

$$\sum^{-1} = \begin{bmatrix} 0.0038 & -0.0035 & -0.0145 \\ -0.0035 & 0.0049 & 0.0269 \\ -0.0145 & 0.0269 & 0.2177 \end{bmatrix}$$

$$Z - \mu = \begin{bmatrix} 227 - 227 \\ 108 - 127 \\ 10 - 7 \end{bmatrix} = \begin{bmatrix} 0 \\ -19 \\ 3 \end{bmatrix}$$

$$(Z - \mu)^T = [0 \quad -19 \quad 3]$$

$$\sum^{-1} * (Z - \mu) = \begin{bmatrix} -0.000413 \\ 0.000196 \\ 0.000086 \end{bmatrix}$$

$$-\frac{1}{2}(Z - \mu)^T * \left(\sum^{-1} * (Z - \mu)\right) =$$

$$[0 \quad -19 \quad 3] * \begin{bmatrix} -0.000413 \\ 0.000196 \\ 0.000086 \end{bmatrix} = \left(-\frac{1}{2}\right) 0.004 = 0.002$$

$$\frac{1}{\sqrt{det \sum i}} e^{0.002} * \pi = \frac{1}{\sqrt{2992335.88}} * 1.002 * 0.77 = 0.000446$$

Perhitungan T-Link (D(M)) pada foreground GMM

$$D(m) = \log((5.4294\text{E} + 160) + 0.00046) = 160$$

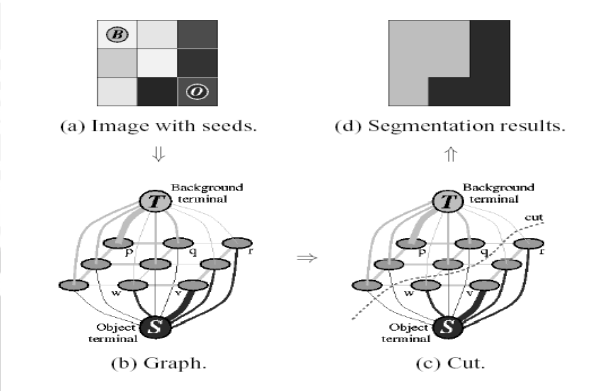
Selanjutnya nilai 160 digunakan untuk nilai bobot *edge* yang menghubungkan antara special node(S) dengan node yang mewakili piksel pada indeks [1,1].

Perhitungan *T-Link* (D(M)) pada *Background* GMM

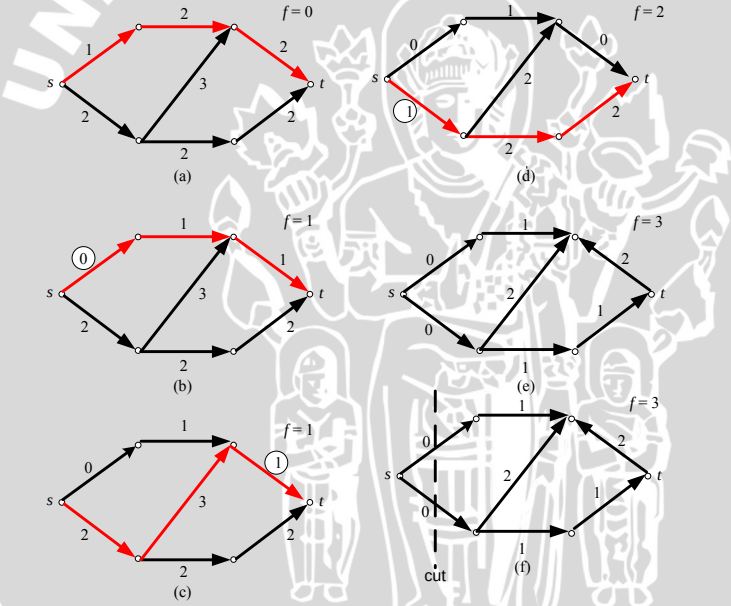
Perhitungan *T-Link* pada *background* GMM sama dengan langkah langkah pada perhitungan *T-Link* pada *Foreground* GMM, hanya saja menggunakan komponen (piksel) yang berada pada *background* GMM.

b. Perhitungan *N-Link*

Seperti yang telah dijelaskan sebelumnya *N-Link* Menghubungkan piksel dengaan piksel tetangganya. Perhitungan *N-Link* menggunakan persamaan 2.4. Selanjutnya nilai *N-Link* digunakan untuk nilai bobot *edge* penghubung antar *node.*



Gambar 3.13 Proses *Graphcut*



Gambar 3.14 *Graphcut* dengan *Min-Cut Max-Flow*

Hasil dari proses graphcut pada gambar 3.13 yaitu kumpulan piksel yang termasuk *foreground* adalah piksel yang berasal dari *tree* S (*Source Tree).*

BAB IV

IMPIEMENTASI DAN PEMBAHASAN

4.1 Lingkungan Implementasi

Lingkungan implementasi yang dijelaskan pada subbab ini meliputi lingkungan implementasi perangkat keras dan perangkat lunak.

4.1.1 Lingkungan Implementasi Perangkat Keras

Perangkat keras yang digunakan dalam pengembangan sistem Optimasi Teknik *Foreground Extraction* Menggunakan *Grabcut* adalah *Notebook* dengan spesifikasi:

1. Processor 1.5GHz Core2duo.
2. Memori 1024 MB DDR2.
3. Kapasitas hard disk 120GB.
4. VGA Intel GMA 250 MB Shared.

4.1.2 Lingkungan Implementasi Perangkat Lunak

Perangkat lunak yang digunakan dalam pengembangan sistem Optimasi Teknik *Foreground Extraction* Menggunakan *Grabcut* adalah :

1. Sistem Operasi Windows XP SP3.
2. Microsoft Visual Studio *Professional edition* 2005 dengan bahasa pemrograman C#.
3. Library Accord.Math (kumpulan library untuk oprasi matriks).
4. Notepad++ sebagai alat bantu editor kode.

4.2 Implementasi Program

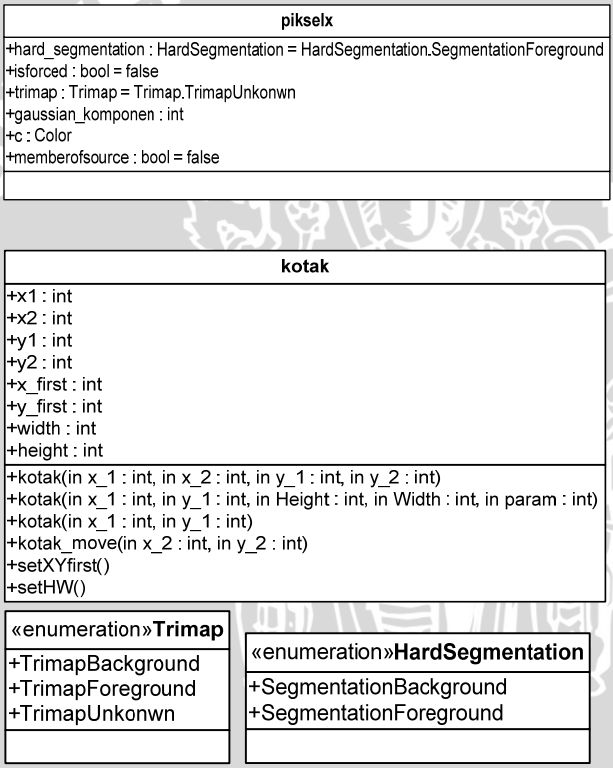
Berdasarkan perancangan perangkat lunak pada bab 3, maka pada subbab ini akan dijelaskan implementasi proses-proses tersebut.

4.2.1 Diagram Kelas

Dalam subbab ini dijelaskan kelas – kelas yang digunakan dalam implemnetasi sistem dikelompokkan berdasarkan fungsinya.

4.2.1.1 Kelas - Kelas Untuk Penanganan Operasi Dasar

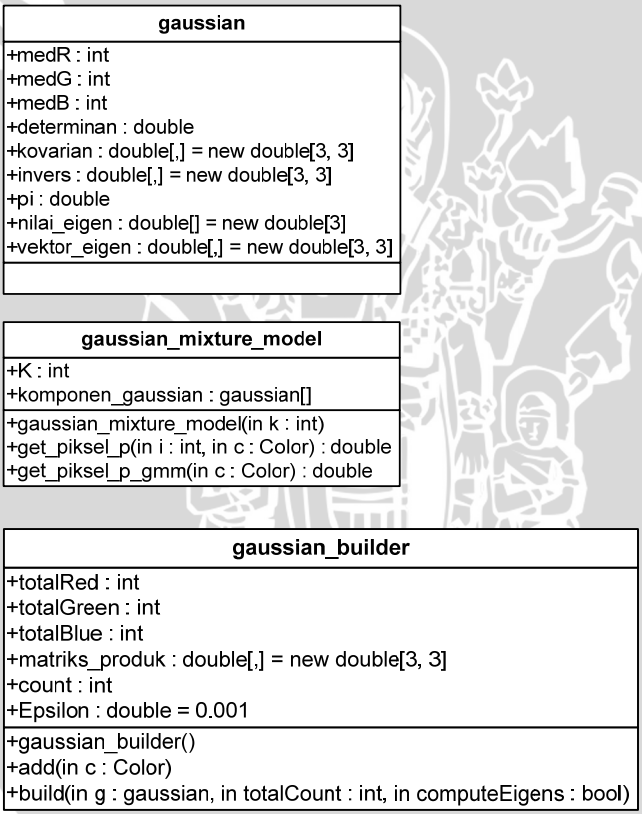
Kelas yang ada pada kelompok ini antara lain kelas trimap (*enum*) yang berisi data struktur penggolongan *trimap*, hardsegmentation (*enum*) yang berisi data struktur pengelompokan *hard segmentation*, kotak (*class*) digunakan untuk menyimpan informasi *trimap* yang dimasukkan oleh *user*, pikselx (*class*) diimplementasikan berupa *array* dua dimensi untuk menyimpan informasi tiap piksel pada gambar *input*. diagram masing – masing kelas dalam kelompok ini seperti pada gambar 4.1.

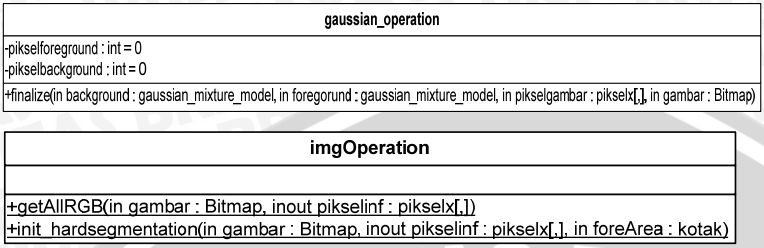


Gambar 4.1 Diagram Kelas – Kelas Untuk Penanganan Operasi Dasar

4.2.1.2 Kelas – Kelas Untuk Penanganan GMM

Kelas yang digunakan untuk operasi GMM antara lain imgOperation (*Class*) merupakan kelas yang digunakan untuk proses segmentasi awal dan mendapatkan nilai tiap piksel dari citra masukan, Gaussian (*Class*) merupakan entitas struktur data penyusun *gaussian mixture model*, Gaussian_mixture_model (*Class*) kelas yang terdiri dari komponen *gaussian* yang membentuk GMM, gaussian_builder (*Class*) digunakan unuk operasi dasar *gaussian* (membangun komponen *gaussian*), gaussian_operation (*Class*) merupakan kelas untuk mengimplemntasikan *Gaussian Mixture Model*.

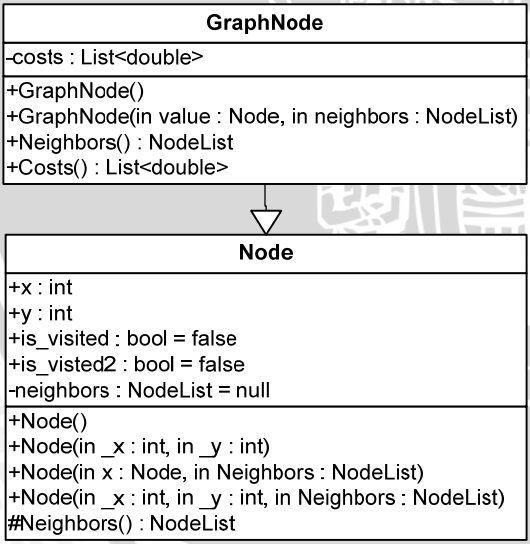


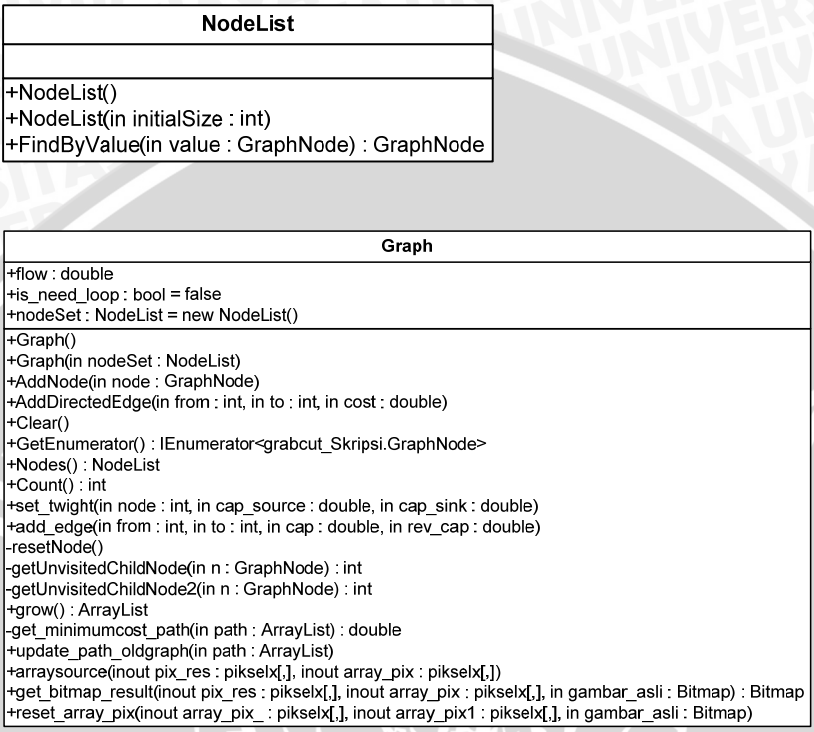


Gambar 4.2 Diagram Kelas Untuk Penanganan Operasi GMM

4.2.1.3 Kelas – Kelas Untuk Penanganan Struktur Data *Graph*

Beberapa kelas pembentuk *graph* yang akan diimplementasikan dalam proses *graphcut* adalah *node* (*Class*) merupakan kelas dasar yang mewakili *node* pada *graph*, GraphNode (*Class*) membentuk *node* dengan penerapan *adjency list* untuk *neighbors* dan *cost*, Graph (*Class*) merupakan kelas yang dipakai untuk implementasi *graph* yang akan digunakan pada operasi *graphcut.*

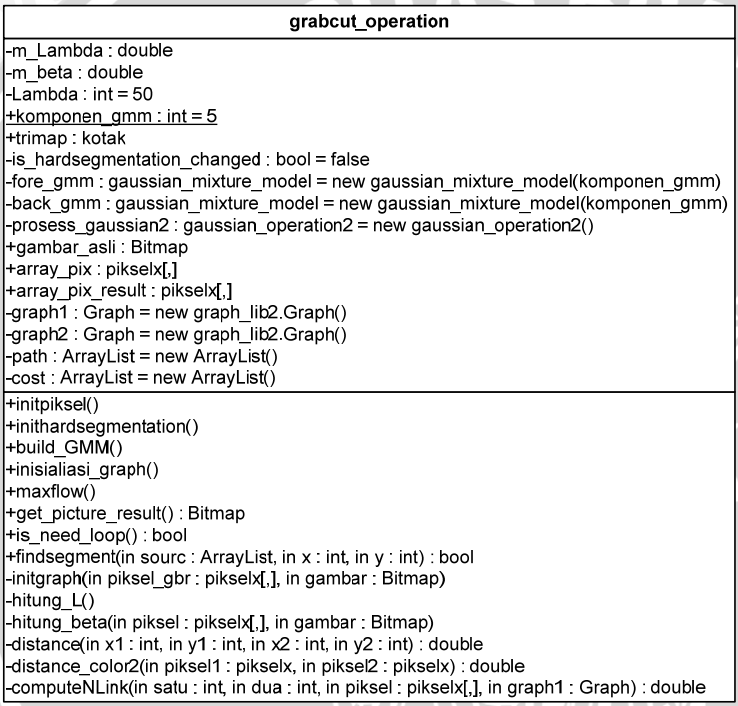




Gambar 4.3 Diagram Kelas Untuk Penanganan *Graph*

4.2.1.4 Kelas- Kelas Untuk Implementasi *Grabcut*

Hanya terdapat satu kelas sebagai jembatan utama untuk mengimplementasikan *grabcut* yaitu kelas grabcut_operation (*class*). Dari kelas grabcut_operation ini semua fungsi-fungsi penyusun operasi *grabcut* disusun sehingga dapat diimplementasikan dengan mudah pada program utama.



Gambar 4.4 Diagram Kelas Untuk Penanganan *Grabcut*

4.2.2 Implementasi Perancangan Sistem

Pada subbab implementasi perancangan sistem akan dijelaskan mengenai implementasi metode - metode yang diguanakn pada setiap tahapan yang telah dijelaskan pada bab 3.

4.2.2.1 Inisialisasi *Trimap* dan *Hard Segmentation*

Pada proses inisisalisasi *trimap*, informasi warna dan koordinat tiap piksel sudah disimpan didalam objek array_pix yang merupakan implementasi dari kelas pikselx. Informasi wilayah *trimap* disimpan pada objek *trimap* yang merupakan objek dari kelas kotak. Pada proses ini, wilayah piksel diluar kotak dimasukkan ke dalam *trimap background* dan *hard segmentation background* sedangkan wilayah didalam kotak di masukkan ke dalam *trimap* unknown dan *hard segmentation foreground*. Detail proses pada tahapan ini dapat dilihat pada source code 4.1.

Fungsi init_hardsegmentation terdapat pada kelas imgOperation

```
public static void init_hardsegmentation(Bitmap
gambar,ref pikselx[,] pikselinf, kotak foreArea)
{
   for (int x = 0; x < gambar.Width; x++)
    for (int y = 0; y < gambar.Height; y++)
    {
      if ((x >= foreArea.x_first) && (x <=
foreArea.x2) && (y >= foreArea.y_first) && (y <=
foreArea.y2))
       {
          if (!pikselinf[x, y].isforced)
              {
                pikselinf[x, y].trimap =
               Trimap.TrimapUnkonwn;
              }
       }
      else
       {
        if (!pikselinf[x, y].isforced)
          {
            pikselinf[x, y].trimap =
            Trimap.TrimapBackground;
          }
       }

     if (pikselinf[x, y].trimap ==
        Trimap.TrimapBackground)
        {
```

```
            pikselinf[x, y].hard_segmentation =
           HardSegmentation.SegmentationBackground;
          }
        else if (pikselinf[x, y].trimap ==
             Trimap.TrimapForeground)
          {
            pikselinf[x, y].hard_segmentation =
           HardSegmentation.SegmentationForeground;
            }

        else if ((pikselinf[x, y].trimap ==
                 Trimap.TrimapUnkonwn))
           {
            pikselinf[x, y].hard_segmentation =
            HardSegmentation.SegmentationForeground
            ;
           }
        }
      }
    }
```

Source Code 4.1 Inisialisasi *Trimap* dan *Hard Segmentation*

4.2.2.2 Pembuatan GMM dengan *Binary Tree Color Quantization Algorithm*

GMM yang dibuat sebanyak 2K, masing-masing sebanyak K komponen untuk *Forground* dan K komponen untuk *Background*. pada awalnya semua piksel berada pada komponen ke 0 (kluster ke-0) yang selanjutnya kluster dipecah dan keanggotaan piksel dalam kluster ditentukan oleh hasil kali komponen warna dari piksel tersebut (R,G,B) dengan vektor eigen dari matriks kovarian pada setiap kluster. penentuan kluster yang akan dipecah didasarkan pada nilai eigen dari kovarian matriks yang paling tinggi. Dalam proses pembuatan GMM ini dibagi dalam beberapa metode, adapun implementasi dari proses pembuatan GMM terdapat pada *source code* 4.2.

Fungsi Finalize terdapat pada kelas gaussian_operation

```
public void finalize(gaussian_mixture_model
foreground, gaussian_mixture_model background,
pikselx[,] pikselgambar, Bitmap gambar)
{
//buat filter untuk tiap komponen dalam GMM
   gaussian_builder[] backfilter = new
   gaussian_builder[background.K];
   gaussian_builder[] forefilter = new
   gaussian_builder[foregorund.K];
 //proses awal untuk cluster 0
   backfilter[0] = new gaussian_builder();
   forefilter[0] = new gaussian_builder();
 //add semua piksel ke cluster 0
   for (int x = 0; x < gambar.Width; x++)
    {
    for (int y = 0; y < gambar.Height; y++)
     {
       pikselgambar[x, y].gaussian_komponen = 0;
      if (pikselgambar[x, y].hard_segmentation ==
      HardSegmentation.SegmentationForeground)
        {
          forefilter[0].add(pikselgambar[x, y].c);
          pikselforeground++;
        }
      else
        {
          backfilter[0].add(pikselgambar[x, y].c);
          pikselbackground++;
        }
     }
    }
//buat satu gaussian untuk cluster 0

forefilter[0].build(foregorund.komponen_gaussian[0
], pikselforeground, true);

backfilter[0].build(background.komponen_gaussian[0
], pikselbackground, true);

//untuk awalan yang displit adalah cluster pertama
```

```
int nbacksplit = 0;
int nforesplit = 0;
// buat filter untuk semua komponen(dalam hal ini
selain komponen ke-0)

for (int j = 1; j < foregorund.K; j++)
    {
      forefilter[j] = new gaussian_builder();
      backfilter[j] = new gaussian_builder();
    }
// buat sisa komponen gaussian(sisa cluster)
for (int i = 1; i < foregorund.K; i++)
    {
// Reset filter
backfilter[nbacksplit] = new
gaussian_builder();
forefilter[nforesplit] = new gaussian_builder();

//objekt untuk operasi, refrence ke komponen
gaussian yang akan displit
gaussian bg = new gaussian();
gaussian fg = new gaussian();
gaussian bg =
background.komponen_gaussian[nbacksplit];
gaussian fg =
foregorund.komponen_gaussian[nforesplit];

// hitung titik potong dari nilai eigen
double splitBack = bg.vektor_eigen[0, 2] * bg.medR
+ bg.vektor_eigen[1, 2] * bg.medG +
bg.vektor_eigen[2, 2] * bg.medB;
double splitFore = fg.vektor_eigen[0, 2] * fg.medR
+ fg.vektor_eigen[1, 2] * fg.medG +
fg.vektor_eigen[2, 2] * fg.medB;

//split cluster, penentutan keeanggotaan masing
masing piksel berganutng antara perkalian vektor
eigen dengan komponen warna
 for (int x = 0; x < gambar.Width; x++)
    {
  for (int y = 0; y < gambar.Height; y++)
     {
```

```
//kelompokkan piksel foreground dalam gaussian
yang sesuai
     if (pikselgambar[x,y]. hard_segmentation ==
        HardSegmentation.SegmentationForeground &&
        pikselgambar[x, y].gaussian_komponen ==
        nforesplit) // nfore menentukan cluster
        foreground keberapa yang akan displit
         {
          if (fg.vektor_eigen[0, 2] *
pikselgambar[x, y].c.R + fg.vektor_eigen[1, 2] *
pikselgambar[x, y].c.G + fg.vektor_eigen[2, 2] *
pikselgambar[x, y].c.B > splitFore)
           {
             pikselgambar[x, y].gaussian_komponen
            = i; //menunjukkan anggota cluster
            keberapa

            forefilter[i].add(pikselgambar[x,
            y].c);
           }

          else
           {
            forefilter[nforesplit].add(pikselgamba
            r[x, y].c);
           }
         }
//kelompokkan piksel background dalam gaussian
yang sesuai
   else if (pikselgambar[x, y].hard_segmentation
   == HardSegmentation.SegmentationBackground &&
   pikselgambar[x, y].gaussian_komponen ==
   nbacksplit)
       {
         if (bg.vektor_eigen[0, 2] *
        pikselgambar[x, y].c.R +
        bg.vektor_eigen[1, 2] * pikselgambar[x,
        y].c.G + bg.vektor_eigen[2, 2] *
        pikselgambar[x, y].c.B > splitFore)
          {
            pikselgambar[x, y].gaussian_komponen
            = i;
```

```
                backfilter[i].add(pikselgambar[x,
                y].c);
            }
        else
         {
           backfilter[nbacksplit].add(pikselgambar[x
           , y].c);
         }
      }
     }
   }

// cluster lama yang displit diupdate

forefilter[nbacksplit].build(background.komponen_g
aussian[nbacksplit], pikselforeground, true);

//backcount jumlah piksel yang berasosiai dengan
backgroundGMM.m_gaussians[nBack]

backfilter[nforesplit].build(foregorund.komponen_g
aussian[nforesplit], pikselbackground, true);

// cluster baru dibuat
)
backfilter[i].build(background.komponen_gaussian[i
], pikselbackground, true);
backfilter[i].build(foregorund.komponen_gaussian[i
], pikselbackground, true);

// cluster yang mempunyai eigen value yang
terbesar yang displit, cluster diambil dari
cluster yang sudah terbentuk sebelumnya
nbacksplit = 0;
nforesplit = 0;

  for (int j = 0; j <= i; j++)
      {
       if (j < background.K &&
       background.komponen_gaussian[j].nilai_eigen
       [2] >
       background.komponen_gaussian[nbacksplit].ni
```

```
        lai_eigen[2])
           nbacksplit = j;

        if (j < foregorund.K &&
        foregorund.komponen_gaussian[j].nilai_eigen
        [2] >
        foregorund.komponen_gaussian[nforesplit].ni
        lai_eigen[2])
            nforesplit = j;
      }
     }
    }
```

Fungsi Build terdapat pada kelas gaussian_builder

```
public void build(gaussian g, int totalCount, bool
computeEigens)  // bentuk komponen gaussian
 {
//tidak ada piksel sama sekali dalam gaussian
   if (count == 0)
      {
         g.pi = 0;
      }
   else
      {
// rata rata komponen warna pada gaussian ini
        g.medR = totalRed / count;
        g.medG = totalGreen / count;
        g.medB = totalBlue / count;
 // kovarin matrik pada gaussian ini
       g.kovarian[0, 0] = matriks_produk[0, 0] /
        count - g.medR * g.medR + Epsilon;
       g.kovarian[0, 1] = matriks_produk[0, 1] /
        count - g.medR * g.medG;
       g.kovarian[0, 2] = matriks_produk[0, 2] /
        count - g.medR * g.medB;
       g.kovarian[1, 0] = matriks_produk[1, 0] /
        count - g.medG * g.medR;
       g.kovarian[1, 1] = matriks_produk[1, 1] /
        count - g.medG * g.medG + Epsilon;
       g.kovarian[1, 2] = matriks_produk[1, 2] /
        count - g.medG * g.medB;
       g.kovarian[2, 0] = matriks_produk[2, 0] /
```

```
        count - g.medB * g.medR;
        g.kovarian[2, 1] = matriks_produk[2, 1] /
        count - g.medB * g.medG;
        g.kovarian[2, 2] = matriks_produk[2, 2] /
        count - g.medB * g.medB + Epsilon;

// Hitung determinan dari matriks kovarian
    g.determinan =
    Accord.Math.Matrix.Determinant(g.kovarian);
// hitung inverse dengan adj(a)/ determinan(a)
    g.invers =
    Accord.Math.Matrix.Inverse(g.kovarian);
// bobot gaussian = jumlah piksel pada komponen
gaussian ini dibagi dengan semua piksel pada
gaussian mixture model
    g.pi = (double)count / totalCount;
    if (computeEigens)
    {
     Accord.Math.Decompositions.EigenvalueDecompos
     ition a = new
     Accord.Math.Decompositions.EigenvalueDecompos
     ition(g.kovarian);
     a.RealEigenvalues.CopyTo(g.nilai_eigen, 0);
     g.vektor_eigen[0, 0] = a.Eigenvectors[0, 0];
     g.vektor_eigen[0, 1] = a.Eigenvectors[0, 1];
     g.vektor_eigen[0, 2] = a.Eigenvectors[0, 2];
     g.vektor_eigen[1, 0] = a.Eigenvectors[1, 0];
     g.vektor_eigen[1, 1] = a.Eigenvectors[1, 1];
     g.vektor_eigen[1, 2] = a.Eigenvectors[1, 2];
     g.vektor_eigen[2, 0] = a.Eigenvectors[2, 0];
     g.vektor_eigen[2, 1] = a.Eigenvectors[2, 1];
     g.vektor_eigen[2, 2] = a.Eigenvectors[2, 2];
    }
   }
}
```

Fungsi Add terdapat pada kelas gaussian_builder

```
 public void add(Color c)
// masukkan piksel kedalam komponen gaussian ini
        {
          totalRed += c.R; totalGreen += c.G;
         totalBlue += c.B;
```

```
            matriks_produk[0, 0] += c.R * c.R;
            matriks_produk[0, 1] += c.R * c.G;
            matriks_produk[0, 2] += c.R * c.B;
            matriks_produk[1, 0] += c.G * c.R;
            matriks_produk[1, 1] += c.G * c.G;
            matriks_produk[1, 2] += c.G * c.B;
            matriks_produk[2, 0] += c.B * c.R;
            matriks_produk[2, 1] += c.B * c.G;
            matriks_produk[2, 2] += c.B * c.B;
//hitung jumlah piksel yang masuk dalam cluster
            ini
           count++;
        }
```

Source Code 4.2 Pembuatan GMM dengan *Binary Tree Quantization Algorithm*

4.2.2.3 Pembuatan *Graph*

Struktur data untuk *graph* yang dipakai menggunakan *adjency list*. Tiap *node* dari *graph* mewakili tiap piksel pada gambar dimana informasi tiap piksel pada gambar sudah ditampung di array_pix yang merupakan *array object* dari kelas pikselx. Seperti yang sudah dijelaskan pada bab 3, pada *graph* terdapat dua *node* khusus yaitu *node source* dan *node sink*, *node source* mempunyai nilai x dan y = -1, node sink mempunyai nilai x dan y = -2, sedangkan *node* yang lain memiliki nilai x dan y sesuai dengan koordinat piksel dalam gambar yang diwakili. Dalam *graph* terdapat dua macam *edge*, yang pertama *edge* berupa *T-Link* dimana *T-Link* ini menghubungkan dua *node* sepesial (*source* dan *sink*) ke *node* biasa yang lain, sedangkan *N-link* menghubungkan antara piksel biasa yang saling bertetangga. Tiap piksel dihubungkan dengna piksel sebelah atas-kiri, atas, atas-kanan, kanan, sehingga hasil akhir yang didapatkan tiap piksel mempunyai hubungan dengan 8 piksel tetangganya. Perhitungan masing-masing *link* / *edge* (*T-link* dan *N-Link*) terdapat pada persamaan 2.8 dan 2.9.

Fungsi initgraph terdapat pada kelas grabcut_operation

```
void initgraph(pikselx[,] piksel_gbr, Bitmap
gambar)
  {
      graph1 = new graph_lib2.Graph();
      GraphNode source = new GraphNode();
      GraphNode sink = new GraphNode();
      source.x = -1; source.y = -1;
      sink.x = -2; sink.y = -2;
      graph1.AddNode(source);
      graph1.AddNode(sink);
      int idxt = 2;
      for (int x = 0; x < gambar.Width; x++)
        for (int y = 0; y < gambar.Height; y++)
          {
            GraphNode temp = new GraphNode();
            temp.is_visited = false;
            temp.x = x; temp.y = y;
            graph1.AddNode(temp);
            double back,fore;
         //set t weight gabung disini
         if (piksel_gbr[x, y].trimap ==
         Trimap.TrimapUnkonwn)
          {
           fore =
          Math.Log(fore_gmm.get_piksel_p_gmm(pikse
          l_gbr[x, y].c));
          back =
          Math.Log(back_gmm.get_piksel_p_gmm(pikse
          l_gbr[x, y].c));
          }
        else if (piksel_gbr[x, y].trimap ==
        Trimap.TrimapBackground)
          {
            fore = 0;
            back = m_Lambda;
                 }
              Else    // TrimapForeground
                {
                 fore = m_Lambda;
                 back = 0;
```

```
                }
                 graph1.set_twight(idxt, fore,
                  back);
               idxt++;
                }
  // Set N-Link dengan metode ku...its faster
guaranted...
   int idx = 2;
   for (int x = 0; x < gambar.Width-1; x++)
       {
    for (int y = 0; y < gambar.Height-1; y++)
       {
       node_nav2 node_Nav = new node_nav2();
       node_Nav.height = gambar.Width – 1;
       if( x > 0 && y < gambar.Height-1 )
        graph1.add_edge(idx,
         node_Nav.UpLeft(idx),
         computeNLink(idx,node_Nav.UpLeft(idx),
         piksel_gbr,graph1), 0);

       if( y < gambar.Height-1 )
         graph1.add_edge(idx, node_Nav.Up(idx),
         computeNLink(idx, node_Nav.Up(idx),
         piksel_gbr, graph1), 0);

       if( x < gambar.Width-1 && y <
         gambar.Height-1 )
           graph1.add_edge(idx,
          node_Nav.UpRight(idx),
          computeNLink(idx,
          node_Nav.UpRight(idx), piksel_gbr,
          graph1), 0);
       if( x < gambar.Width-1 )
         graph1.add_edge(idx,
          node_Nav.Right(idx), computeNLink(idx,
          node_Nav.Right(idx), piksel_gbr,
          graph1), 0);
       idx++;
      }
     }
   }
```

Source Code 4.3 Pembuatan *Graph*

4.2.2.4 Perhitungan *T-Link*

Implementasi perhitungan *N-Link* dan *T-link* didapat dari GMM yang mengacu pada persamaan 2.8 dan 2.9. Perhitungan *T-Link* dilakukan pada tiap piksel dan dihitung nilai *T-link* untuk S (*Source*) dan T (*Sink*). *Node* terhubung dengan S atau T ditentukan oleh besar nilai *T-link* yang didapatkan. Selanjutnya selisih nilai *T-Link* antara *T-link* (s) dan *T-link* (T) menjadi nilai bobot pada *edge* yang menghubungkan *node* yang mewakili tiap piksel dengan *node* terminal. *Node* terminal (S) terdapat pada indeks pertama *nodelist* (indeks ke- 0) dan *node* terminal (T) terdapat pada indeks kedua *nodelist* (indeks ke- 1). *Edge* berarah dari (T) ke *node* lain dan dari *node* lain ke (S). Adapun *source code* implementasi perhitungan *T-Link* terdapat pada *source code* 4.4.

Fungsi set_twight terdapat pada kelas graph

```
public void set_twight(int node, double
cap_source, double cap_sink)
            {
             bool is_source = (cap_source >
             cap_sink) ? true : false;
             double weight = Math.Abs(cap_source -
             cap_sink);
                if (weight > 0)
                {
                    if (is_source)
                    {
                     add_edge(0, node, weight, 0);
                    }
                    else
                    {
                     add_edge(node, 1, weight, 0);
                    }
                }

}
```

Fungsi get_piksel_p() terdapat pada kelass gaussian_mixture_model

```
public double get_piksel_p(int i, Color c)
  {
   if (komponen_gaussian[i].pi > 0)
      {
       if (komponen_gaussian[i].determinan > 0)
          {
            double Rselisih = c.R -
            komponen_gaussian[i].medR;
            double Gselisih = c.G -
            komponen_gaussian[i].medG;
            double Bselisih = c.B -
            komponen_gaussian[i].medB;

            double pengali = Rselisih * (Rselisih
            * komponen_gaussian[i].invers[0, 0] +
            Gselisih *
            komponen_gaussian[i].invers[1, 0] +
            Bselisih *
            komponen_gaussian[i].invers[2, 0]) +
            Gselisih * (Rselisih *
            komponen_gaussian[i].invers[0, 1] +
            Gselisih *
            komponen_gaussian[i].invers[1, 1] +
            Bselisih *
            komponen_gaussian[i].invers[2, 1]) +
            Bselisih * (Rselisih *
            komponen_gaussian[i].invers[0, 2] +
            Gselisih *
            komponen_gaussian[i].invers[1, 2] +
            Bselisih *
            komponen_gaussian[i].invers[2, 2]);
            return ((1.0 /
            (Math.Sqrt(komponen_gaussian[i].determ
            inan))) * Math.Exp(-0.5 * pengali));
          }
       }
          return 0;
      }
```

Fungsi get_piksel_p_gmm terdapat pada kelass gaussian_mixture_model

```
public double get_piksel_p_gmm(Color c)
{
   double p = 0;
   for (int i = 0; i < K; i++)
       {
           p += komponen_gaussian[i].pi *
            get_piksel_p(i, c);
       }
    return p;
}
```

Source Code 4.4 Perhitungan *T-Link*

4.2.2.5 Perhitungan *N-Link*

Implementasi perhitungan *N-Link* didasarkan pada persamaan 2.5, 2.8 dan 2.9. Untuk mempermudah implementasi perhitungan, Penerapan *code* program dibagi menjadi beberapa fungsi yang diantaranya adalah computeNlink yang merupakan fungsi utama perhitungan *N-link* dimana parameternya adalah nilai indeks *node* pada *nodeset* dalam graph. Distance_color2 digunakan untuk perhitungan jarak warna antar dua piksel. Distance digunakan unutk perhitungan jarak koordinat antara dua piksel. Sedangkan hitungbeta adalah fungsi yang diguanakan untuk menghitung nilai beta, dimana nilai beta adalah merupakan rata – rata jarak warna dari semua *edge* yang menghubungkan antara piksel dengan piksel lainnya.

Fungsi computeNlink terdapat pada kelas grabcutOperation

```
double computeNLink(int satu, int dua,
pikselx[,] piksel,Graph graph1)
   {
  return
      (Lambda/distance(graph1.nodeSet[satu].x,
      graph1.nodeSet[satu].y,
      graph1.nodeSet[dua].x,
      graph1.nodeSet[dua].y))*(Math.Exp(-m_beta
      *
      distance_color2(piksel[graph1.nodeSet[satu
```

```
        ].x, graph1.nodeSet[satu].y],
        piksel[graph1.nodeSet[dua].x,graph1.nodeSe
        t[dua].y])));
          }
```

Fungsi distance_color2terdapat pada kelas grabcutOperation

```
double distance_color2(pikselx piksel1, pikselx
piksel2)
        {
            return (Math.Pow((piksel1.c.R -
           piksel2.c.R), 2) +
           Math.Pow((piksel1.c.G - piksel2.c.G),
           2) + Math.Pow((piksel1.c.B -
           piksel2.c.B), 2));
        }
```

Fungsi distance terdapat pada kelas grabcutOperation

```
double distance(int x1, int y1,int x2,int y2)
        {
           return (Math.Sqrt((x1-x2)*(x1-x2)+(y1-
           y2)*(y1-y2)));
        }
```

Fungsi hitung_beta terdapat pada kelas grabcutOperation

```
        void hitung_beta(pikselx[,] piksel,
Bitmap gambar)
        {
            double result = 0;
            int edges = 0;

            for (int x = 0; x < gambar.Width;
++x)
            {
                for (int y = 0; y <
gambar.Height; ++y)
                {
                    if (x > 0 && y <
gambar.Height - 1)
      // upleft
                    {
```

```
                        result +=
distance_color2(piksel[x,y],piksel[x-1,y+1]);
                        edges++;
                    }

                    if (y < gambar.Height - 1)
                                        // up
                    {
                        result +=
distance_color2(piksel[x, y], piksel[x, y + 1]);
                        edges++;
                    }

                    if (x < gambar.Width - 1 &&
y < gambar.Height - 1)                    //
upright
                    {
                        result +=
distance_color2(piksel[x, y], piksel[x+1, y+1]);
                        edges++;
                    }

                    if (x < gambar.Width - 1)
                                        // right
                    {
                        result +=
distance_color2(piksel[x, y], piksel[x+1,y]);
                        edges++;
                    }
                }
            }
            m_beta = (1/ (2 * result / edges));
        }
```

Source Code 4.5 Perhitungan *N-Link*

4.2.2.5 *Min-Cut Max-Flow* dengan Algoritma Ford Fulkerson

Proses *Min-Cut Max-Flow* bertujuan untuk memisahkan *graph* menjadi dua *tree* yang terpisah dimana total titik potong minimal sama dengan *flow* maksimal yang dapat dilewatkan pada *graph* dalam waktu yang sama. Untuk itu digunakan *ford-fulkerson algorithm* yang sudah dijelaskan pada bab 3. secara garis besar terdapat dua metode utama dalam implementasi ford-fulkerson, yang pertama metode untuk menemukan *augmenting path*, dimana dalam implementasinya untuk menumukan *augmenting path* pada *graph* digunakan algoritma *depth first search* yang dimodifikasi. Ketika *augmenting path* ditemukan proses selanjutunya adalah mengurangi jalur pada *augmenting path* dengan nilai minimal dari *edge* pada jalur, proses ini diulangi hingga tidak ada *augmenting path* yang ditemukan. Detail implementasi terdapat pada *source code* 4.6.

```
public void maxflow()
        {
              while (true)
              {
                  path = graph1.grow(ref cost);

                  if(path==null)
                  {
                      break;
                  }

              graph1.update_path_oldgraph(path);
              }
        }
```

```
public ArrayList grow(ref ArrayList cost)
    {
      ArrayList path = new ArrayList();
      GraphNode temp = new GraphNode();
      Stack node_aktif = new Stack();
      Stack list_index = new Stack();
      Stack list_cost = new Stack();

      node_aktif.Push(nodeSet[0]);
```

```
        list_index.Push(0);
        while (node_aktif.Count>0)
         {

         GraphNode n = (GraphNode)node_aktif.Peek();
         int child = -1;
        if ((child = getUnvisitedChildNode(n)) != -1)
           {
              node_aktif.Push(n.Neighbors[child]);
              list_index.Push(child);
         list_cost.Push(n.Costs[child]);
         n = n.Neighbors[child];
         if (n.x == -2)
           {
             path.Clear();
             while (list_index.Count > 0)
             {
               path.Add(list_index.Pop());
             }
             path.Reverse();
             return path;
           }
          }
        else
           {
             node_aktif.Pop();
             list_index.Pop();
           }
          }
          resetNode();
          return null;
        }
```

```
public void update_path_oldgraph(ArrayList path)
    {
     if (path.Count > 0)
      {
     // berjalan menyusuri path pada graph dan
update costnya
     double cost_rev = get_minimumcost_path(path);
     flow += cost_rev;
     GraphNode walks = nodeSet[0];
     int k = 1;
```

```
      while (walks.x != -2)
      {
        if (k == path.Count)
            {
               break;
            }

        walks.Costs[(int)path[k]] =
        walks.Costs[(int)path[k]] - cost_rev;
        walks = walks.Neighbors[(int)path[k]];
            k++;
        }
       }
        path.Clear();
        return;
      }
```

Source Code 4.6 *Min Cut Max Flow* dengan Algoritma Ford Fulkerson

4.2.2.6 Pemotongan *Graph*

Setelah proses implementasi *Ford Fulkerson algorithm*, ditemukan nilai *max flow* dan titik potong untuk memisahkan *graph* menjadi dua *tree* yaitu jalur antara *node* yang mempunyai kapasitas nol. Dalam proses ini *node* yang menjadi bagian dari *tree* akan ditandai yang akan digunakan pada proses selanjutnya. Untuk proses penelusuran *graph* digunakan algoritma *bread first search* yang dimodifikasi. Implementasi proses pemotongan *graph* dapat dilihat pada *source code* 4.7.

```
public void arraysource(ref pikselx [,] pix_res,
ref pikselx [,] array_pix)
   {
     is_need_loop = false;
     GraphNode noderoot = (GraphNode)nodeSet[0];
     Queue q = new Queue();
     q.Enqueue(noderoot);
     noderoot.is_visted2 = true;
     while (q.Count > 0)
```

```
        {
          GraphNode n = (GraphNode)q.Dequeue();
          int child = -1;
  while ((child = getUnvisitedChildNode2(n)) != -1)
          {
            pix_res[n.Neighbors[child].x,
           n.Neighbors[child].y].memberofsource =
           true;
           q.Enqueue(n.Neighbors[child]);

      n.Neighbors[child].is_visted2 = true;
        }
       }
       resetNode();
    }
```

Source Code 4.7 Pemotongan *Graph*

4.2.2.7 *Update Hard Segmentation*

Setelah proses *graphcut*, *trimap* dan nilai *hard segmentation* diubah sesuai dengan hasil *graphcut*, piksel yang termasuk dalam *tree source* dimasukkan ke dalam *hard segmentation foreground*. di akhir proses ini diketahui apakah proses *grabcut* perlu diulang atau tidak, proses *grabcut* perlu diulang hingga tidak ada lagi perpindahan piksel dari *hard segmentation foreground* ke *hard segmentation background* dan sebaliknya atau hingga didapatkan hasil yang diinginkan *user*, pada proses ini piksel yang tidak termasuk hasil ekstraksi di ubah warnaya menjadi putih.

```
public Bitmap get_bitmap_result(ref pikselx[,]
pix_res,ref pikselx[,] array_pix,Bitmap
gambar_asli)
  {
     Bitmap temp = new
     Bitmap(gambar_asli.Width,gambar_asli.Height);
       temp = gambar_asli;
    for (int i = 0; i < gambar_asli.Width; i++)
          {
       for (int j = 0; j < gambar_asli.Height; j++)
          {
```

```
            if (!pix_res[i, j].memberofsource)
                {
                 temp.SetPixel(i, j, Color.White);
                 if (!array_pix[i, j].isforced)
                    {
                     pix_res[i, j].hard_segmentation
= HardSegmentation.SegmentationBackground;
                    }
                }
              else
                {
                  temp.SetPixel(i, j,
gambar_asli.GetPixel(i, j));
                   if (!array_pix[i, j].isforced)
                      {
                       pix_res[i, j].hard_segmentation
= HardSegmentation.SegmentationForeground;
                      }
                }
             if (array_pix[i, j].hard_segmentation !=
pix_res[i, j].hard_segmentation)
                {
                   is_need_loop = true;
                }

               }
         }
 reset_array_pix(ref pix_res,ref array_pix,
gambar_asli);
             return temp;
         }
```

Source Code 4.8 *Update Hard Segmentation*

4.2.2.8 Evaluasi Hasil

Seperti yang telah dijelaskan pada bab tiga, untuk menguji hasil ekstraksi digunakan citra *foreground* pembanding yang sebelumnya sudah dilakukan ekstraksi *foreground* secara manual dengan bantuan *software photoshop*. Selanjutnya citra pembanding dibandingkan dengan citra hasil ekstraksi dari aplikasi *grabcut* dan dihitung berapa piksel yang tidak sama dengan citra pembanding.

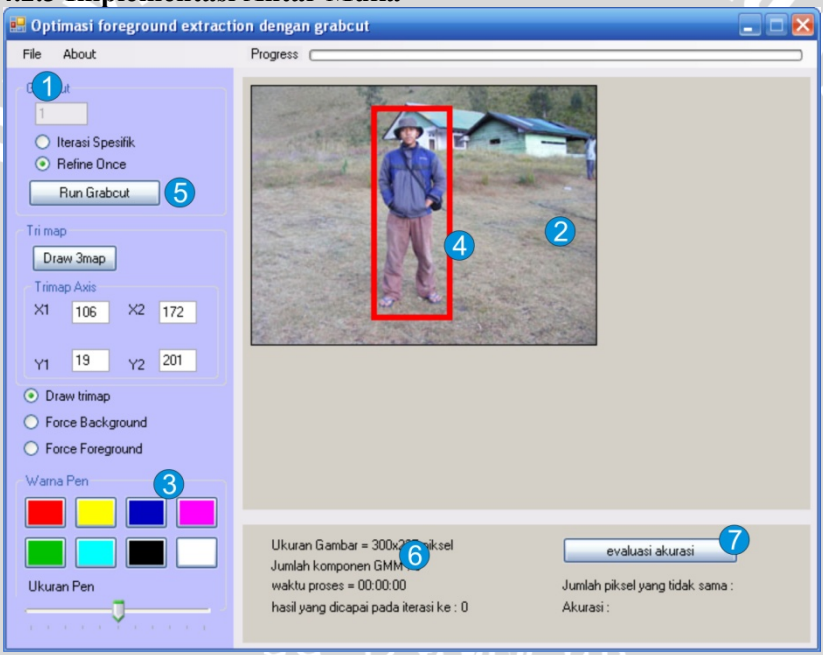
Ketidaksamaan yang dimaksud haruslah bahwa salah satu dari kedua piksel yang dibandingkan merupakan piksel putih. *Source code* evaluasi hasil terdapat pada *source code* 4.9.

```
if (openFileDialog2.ShowDialog() ==
DialogResult.OK)
    {
      Bitmap gambar_hasil = new
      Bitmap(pictureBox1.Image);
      Bitmap gambar_eval = new
      Bitmap(openFileDialog2.FileName);
      if ((gambar_eval.Width != gambar_hasil.Width)
|| (gambar_eval.Height != gambar_hasil.Height))
        {
        MessageBox.Show("ukuran gambar harus sama");
        return;
        }
     int differences = 0;
     double akurasi = 0;
     int wholepix = gambar_eval.Width *
     gambar_eval.Height;
     for(int i=0;i<gambar_eval.Height;i++)
       for (int j = 0; j < gambar_eval.Width; j++)
          {
           if (((gambar_hasil.GetPixel(j, i).R ==
           255) && (gambar_hasil.GetPixel(j, i).G
           == 255) && (gambar_hasil.GetPixel(j,
           i).B == 255)) ||
           ((gambar_eval.GetPixel(j, i).R == 255)
           && (gambar_eval.GetPixel(j, i).G == 255)
           && (gambar_eval.GetPixel(j, i).B ==
           255)))
              {
               if (gambar_hasil.GetPixel(j, i) !=
               gambar_eval.GetPixel(j, i))
                  {
                     differences++;
                  }
               }
           }
     akurasi = (double)(100-
```

```
        (double)((double)differences /
        (double)wholepix) * 100);
             label11.Text = "Akurasi: " +
        akurasi.ToString() +" %";
        label14.Text = "Jumlah piksel yang tidak sama:
        " + differences.ToString();
        eval form_eval = new
        eval(openFileDialog2.FileName, label11.Text,
        label14.Text);
        form_eval.Show();
```

Source Code 4.9 Eavaluasi Hasil

4.2.3 Implementasi Antar Muka



Gambar 4.5 Jendela utama aplikasi

Gambar 4.5 merupakan gambar jendela utama aplikasi optimasi *foreground extraction* menggunakan metode *grabcut*. Untuk memasukkan input citra kedalam aplikasi terdapat *menu file, load* gambar seperti yang terdapat dalam petunjuk gambar nomor 1. Jika gambar berhasil dimuat kedalam aplikasi, maka gambar akan

ditampilkan di *picturebox* seperti yang terdapat dalam petunjuk gambar nomor 2. Setelah gambar berhasil di buka dalam aplikasi, *user* dapat memilih warna serta ketebalan *pen* seperti yng terdapat pada petunjuk gambar nomor 7. setelah itu *user* dapat membuat area trimap pada gambar seperti yang terdapat pada gambar nomor 4. Selanjutnya operasi *grabcut* dapat dijalankan dengan menekan tombol *Run Grubcut* yang terdapat pada penunjuk gambar nomor 5. Beberapa informasi dalam proses grabcut ditampilkan pada area penunjuk gambar nomor 6. Setelah proses ekstraksi selesai, evaluasi hasil ekstraksi dapat dilakukan dengan memasukkan informasi gambar pembanding dengan menekan tombol nomor 7. Jika gambar pembanding berhasil di-*load* maka akan tampil jendela gambar pembanding dan ditampilkan informasi akurasi dari hasil ekstraksi beserta jumlah piksel yang tidak sesuai.

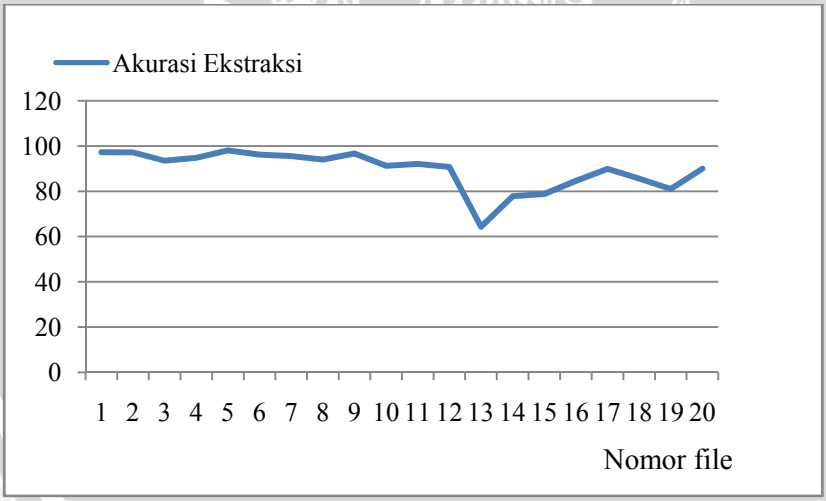
Gambar 4.6 Jendela Gambar Pembanding

4.3 Implementasi Uji Coba

dalam implementasi uji coba dan evaluasi hasil, seperti yang dijelaskan pada bab 3, terdapat tiga uji coba. Yang pertama adalah uji coba untuk analisa keakuaratan ekstraksi, yang kedua untuk analisa kecepatan atau waktu yang dibutuhkan oleh program untuk mengekstraksi *foreground* yang dikehendaki pada suatu citra digital, yang ketiga adalah untuk mengetahui pengaruh luas area *trimap* terhadap kecepatan dan hasil ekstraksi.

4.3.1 Implementasi Uji Keakuratan Ekstraksi

Dalam uji keakuratan ekstraksi digunakan 20 citra digital dengan tingkat koherenitas *background* dan *foreground* yang berbeda-beda serta kondisi *foreground* yang berbeda-beda. dalam proses uji coba hanya dibatasi penggunaan *trimap* awal tanpa *user touchup*. Iterasi proses grabcut dihentikan jika akurasi yang didapat lebih buruk dari proses sebelumnya. Adapun gambar hail uji akurasi dapat dilihat pada tabel 4.2, grafik hasil uji keakuratan ekstraksi terdapat pada gambar 4.7 dimana sumbu x adalah nomor *file* pada tabel 4.1 dan sumbu y adalah prosentase akurasi ekstraksi yang didapatkan.



Gambar 4.7 Grafik Hasil Uji Akurasi

Tabel 4.1 Hasil Analisa Akurasi

no	Nama File	I	Ukuran	Koordinat *Trimap*				WP	A%	T
				X1	Y1	X2	Y2			
1	Bungalily.bmp	2	200x133	23	21	127	100	709	97.33	27
2	Bungakuning.bmp	2	200x133	48	17	154	90	627	97.22	29
3	bungamerah.bmp	1	200x197	40	32	170	184	2545	93.54	25
4	bebekputih.bmp	1	200x150	22	33	164	129	1550	94.83	50
5	kepik.bmp	3	200x183	72	84	151	143	699	98.09	28
6	apel.bmp	2	200x127	25	19	169	117	955	96.24	183
7	aku.bmp	1	200x150	71	16	113	130	1326	95.58	6
8	ayamhitam.bmp	2	200x139	50	21	164	125	1643	94.08	45
9	ikanmerah.bmp	1	200x150	40	53	121	104	969	96.77	5
10	durian.bmp	1	200x150	53	13	159	125	2619	91.27	31
11	kepalaunta.bmp	2	200x153	51	44	190	148	2399	92.16	113
12	macan.bmp	1	200x150	76	5	147	127	2763	90.94	17
13	kurakura.bmp	1	200x155	33	13	168	144	11059	64.32	34
14	ikanpurba.bmp	1	200x138	24	33	186	125	6124	77.81	66
15	kelinci.bmp	1	200x162	23	36	159	128	6862	78.81	34
16	bersama.jpg	1	250x186	15	54	222	184	7129	84.66	421
17	duabunga.jpg	1	250x188	22	55	237	153	4711	89.97	90
18	enamburung.jpg	1	250x127	2	39	248	115	4568	85.61	68
19	saputerbang.jpg	1	250x156	21	9	217	141	7392	81.04	105
20	Kupukupu.jpg	1	250x188	31	31	183	168	4675	90.05	56

Keterangan tabel 4.1 adalah :

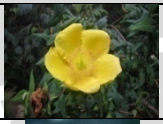
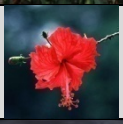
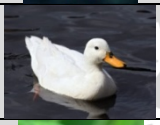
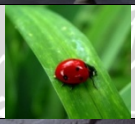
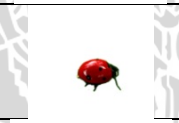
I : Iterasi (banyak proses *grabcut* yang diperlukan)

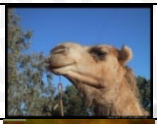
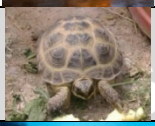
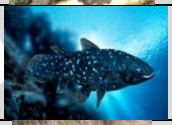
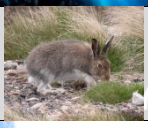
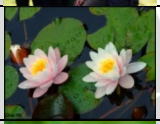
WP : jumlah piksel yang salah

A% : prosentase akurasi

T : waktu yang diperlukan

Tabel 4.2 Tabel Gambar Hasil Uji Akurasi

Nama File	Gambar Awal	Gambar Pembanding	Hasil Ekstraksi
Bungalily.bmp			
Bungakuning.bmp			
bungamerah.bmp			
bebekputih.bmp			
kepik.bmp			
apel.bmp			
aku.bmp			
ayamhitam.bmp			
ikanmerah.bmp			
durian.bmp			

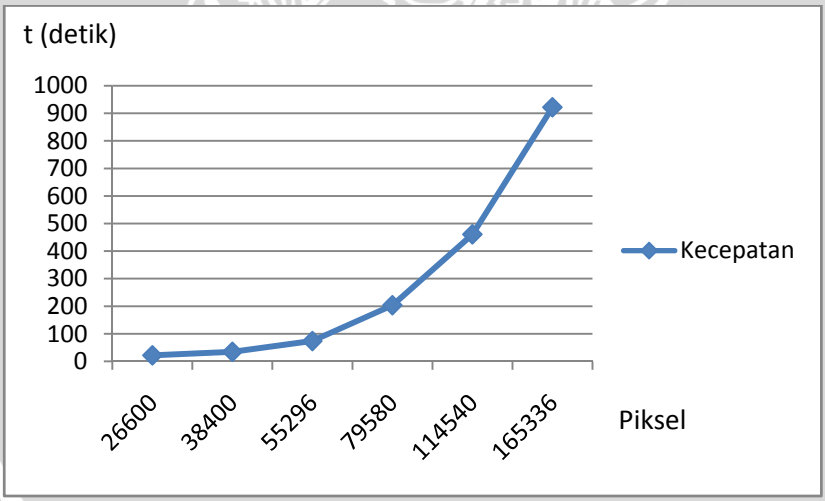
kepalaunta.bmp			
macan.bmp			
kurakura.bmp			
ikanpurba.bmp			
kelinci.bmp			
bersama.jpg			
duabunga.jpg			
enamburung.jpg			
saputerbang.jpg			
Kupukupu.jpg			

4.3.2 Implementasi Uji Kecepatan Ekstraksi pada Satu Iterasi *Grabcut*

Dalam impelementasi sekenario analisa kecepatan ini digunakan satu gambar yang sama dan wilayah *trimap* dengan perbandingan yang sama dengan berbagai ukuran (resolusi gambar) dengan peningkatan tiap 120% hingga didapat enam resolusi gambar yang berbeda. Hasil analisa kecepatan terdapat dalam tabel 4.3.

Tabel 4.3 Hasil Uji Kecepatan Ekstraksi

No	Ukuran Gambar		Koordinat Trimap				Waktu
	Panjang	Lebar	X1	Y1	X2	Y2	Detik
1	200	133	17	23	130	107	22
2	240	160	20	28	156	128	35
3	288	192	24	33	187	154	74
4	346	230	29	40	225	185	204
5	415	276	35	48	270	222	461
6	498	332	42	57	323	266	922



Gambar 4.7 Grafik Waktu Komputasi Terhadap Perubahan Resolusi Gambar

4.3.3 Implementasi Uji Pengaruh Luas Area *Trimap* Terhadap Hasil dan Kecepatan Ekstraksi pada Satu Iterasi *Grabcut*

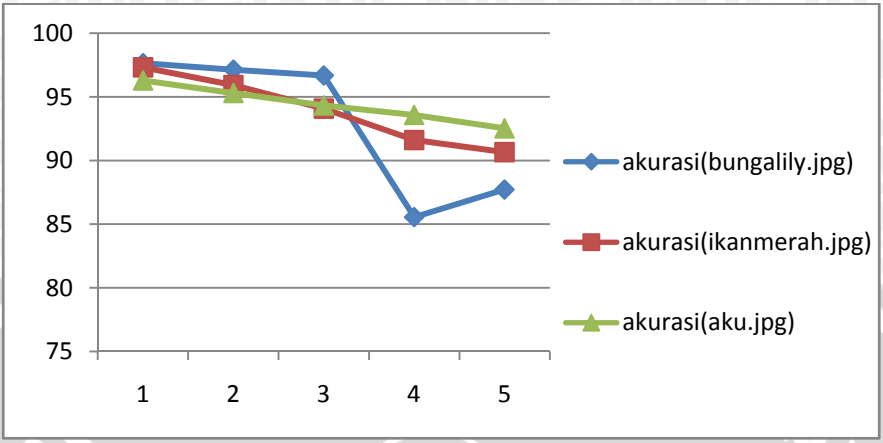
Dalam implementasi uji pengaruh area *trimap* terhadap hasil ekstraksi digunakan tiga citra digital dengan input area *trimap* yang memiliki luas yang berbeda. Untuk mendapatkan penambahan luas trimap yang konstan, koordinat x1,y1 dikurangi 10 sedangkan untuk x2,y2 masing-masing ditambah 10 untuk setiap kali uji coba. Dalam ui pengaruh luas area *trimap* ini hanya digunakan satu iterasi *grabcut*. Citra digital yang digunakan merupakan satu citra digital berukuran 300x200 piksel. Hasil uji coba pengaruh luas area *trimap* ini terdapat pada tabel 4.3.

Tabel 4.4 Hasil Uji Pengaruh Luas Area *Trimap*

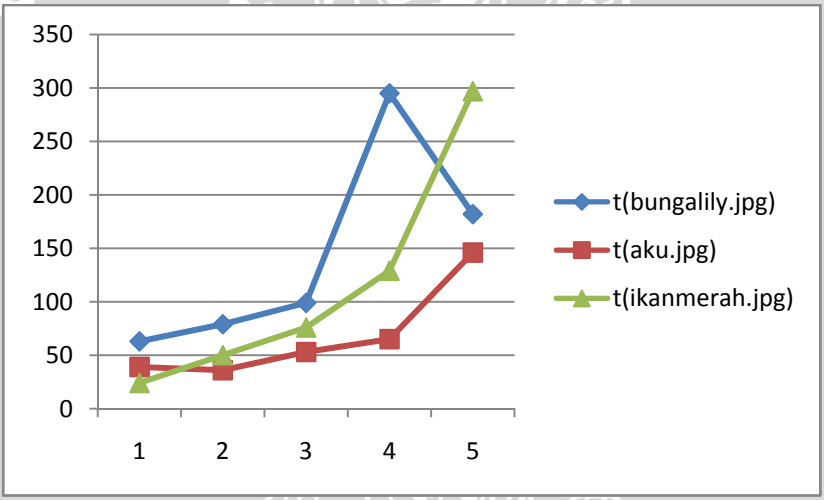
Uji ke-	Nama File	X1	Y1	X2	Y2	Luas	WP	%A	T
1	Bungalily.jpg	32	41	188	149	16848	1535	97.65	63
2	Bungalily.jpg	22	31	198	159	22528	5319	97.14	79
3	Bungalily.jpg	12	21	208	169	29008	3111	96.68	99
4	Bungalily.jpg	2	11	218	179	36288	15227	85.55	295
5	Bungalily.jpg	0	1	228	189	42864	38456	87.72	182
1	ikanmerah.jpg	56	82	181	156	9250	1807	97.32	1807
2	ikanmerah.jpg	46	72	191	166	13630	2747	95.93	2747
3	ikanmerah.jpg	36	62	201	176	18810	3990	94.08	3990
4	ikanmerah.jpg	26	52	211	186	24790	5658	91.61	5658
5	ikanmerah.jpg	16	42	221	196	31570	6304	90.66	6304
1	aku.jpg	112	24	168	193	9464	2501	96.29	2501
2	aku.jpg	102	14	178	203	14364	3169	95.3	3169
3	aku.jpg	92	4	188	213	20064	3824	94.33	3824
4	aku.jpg	82	0	198	227	26332	4332	93.58	4332
5	aku.jpg	72	0	218	227	33142	5181	92.53	5181

Keterangan Tabel 4.4

X1, X2, Y1, Y2 : Koordinat *trimap*
Luas : Luas area trimap
WP : Jumlah piksel yang salah
%A : Prosentase akurasi yang didapatkan
T : Waktu komputasi yang dibutuhkan dalam satuan detik



Gambar 4.8 Grafik Akurasi Terhadap Luas *Trimap*



Gambar 4.9 Grafik Waktu Komputasi Terhadap Luas *Trimap*

4.4 Analisa Hasil

Dari analisa hasil yang didapat dalam proses ekstraksi *foreground* menggunakan *grabcut*, tampak bahwa akurasi hasil ekstraksi dengan metode ini cukup bagus. Nilai akurasi tertinggi dari uji coba didapatkan nilai akurasi 98,09%, akurasi rata rata 89,50% dan akurasi terendah 64,32% Akan didapat hasil ekstraksi yang sangat bagus jika distribusi warna kelompok *foreground* dan kelompok *background* berbeda.

Hasil ekstraksi juga dipengaruhi wilayah *trimap* yang dibuat secara umum semakin sedikit jarak antara *trimap* dengan *foreground* yang diinginkan maka hasil ekstraksi akan semakin bagus dan proses ekstraksi akan semakin cepat. Akan tetapi hal ini tidak mutlak berlaku untuk semua citra digital, Akurasi dan kecepatan sangat bergantung pada konstruksi *graph* yang dibangun berdasarkan informasi tiap piksel pada citra uji sehingga memungkinkan dalam suatu kondisi tertentu ekstraksi berjalan cepat walaupun wilayah trimap cukup luas. Kondisi ini terjadi jika proses *cut* (pengenolan) nilai *edge* banyak terjadi pada jalur-jalur awal dari tiap *augmenting path* yang didapatkan.

Kelemahan dari algoritma ini adalah jika distribusi warna antara wilayah *foreground* dengan *background* cukup *homogen*, nilai keanggotaan dalam distribusi multinormal (GMM) sangat mempengaruhi hasil ekstraksi. Oleh karena itu jika distribusi warna antara wilayah *background* dan *foreground* cukup homogen, maka akan didapat hasil ekstraksi yang buruk. Disamping itu, berdasarkan analisa kecepatan yang dilakukan pada citra dengan berbagai ukuran yang berbeda, metode *grabcut* cukup lambat, semakin besar ukuran gambar maka proses yang dibutuhkan akan semakin lama, hal ini dikarenakan semakin banyak piksel pada gambar berarti semakin banyak pula *Node* dalam *graph* sehingga mekanisme *min-cut max-flow* yang didasarkan pada proses *augmenting path* juga akan menelusuri semakin banyak jalur pada *graph* sehingga waktu komputasi akan semakin lama. Waktu komputasi juga dipengaruhi oleh luas area *trimap* yang dibuat oleh *user*. semakin lebar *trimap* maka akan semakin banyak *Node* yang terhubung langusng dengan *node* terminal (terhubung dengan *T-Link*) sehingga alternatif jalur pertama *augmenting path* yang harus ditelusuri semakin banyak.

BAB V

PENUTUP

5.1 Kesimpulan

Dari hasil analisa uji coba dapat diambil kesimpulan:

1. Tingkat akurasi sistem *foreground extraction* menggunakan *grabcut* cukup tinggi karena dapat diperoleh hasil ekstraksi dengan akurasi tertinggi mencapai 98.09%.
2. Didapat hasil yang sangat bagus untuk citra digital dimana *foreground* dan *background* dapat dibedakan dengan jelas. Semakin homogen perbedaan warna antara *background* dan *foreground*, hasil ekstraksi yang didapat akan semakin buruk.
3. Semakin besar ukuran gambar, proses ekstraksi akan semakin lambat.
4. Semakin besar ukuran *trimap,* pada umumnya proses ekstraksi akan lebih lambat dan akurasi yang dihasilkan akan semakin buruk.
5. Untuk mengimplementasikan *grabcut* dalam teknik *foreground extraction* sehingga didapatkan sebuah aplikasi *foreground extraction* yang memungkinan interaksi user seminimal mungkin dapat dialakukan dengan tahapan *segmentasi awal* dan *hard segmentation* , pembuatan GMM dengan *binary tree color quatization algorithm*, *graphcut*, dan *upadate hard segmentation.*

5.2 Saran

Meskipun didapat hasil akurasi yang baik dalam penerapan *grabcut* untuk *teknik forground extraction,* tetapi dalam prosesnya algoritma ini cukup lambat. Waktu komputasi yang lama terutama dalam proses *min-cut max-flow*, untuk itu diperlukan pengembangan algoritma lebih lanjut dalam proses ini sehingga waktu komputasi diharapkan dapat lebih cepat.

DAFTAR PUSTAKA

Ahmad, Balza dan Firdausy, Kartika. 2005. *Teknik Pengolahan Citra Digital Menggunakan Delphi*. Ardi Publishing. Yogyakarta

Ballard and Brown. 1982. *Computer Vision*. Prentice Hall. New Jersey

Bengio, Samy. 2006. *Statistical Machine Learning from Data Gaussian Mixture Models.* IDIAP Research Institute. Switzerland

Boykov and Jolly. 2001. *Interactive graph cuts for optimal boundary and region segmentation of objects in N-D*

Foulds. 1992. *Graph Theory Applications*. Springer-Verlag. New York

González, Rafael C. 2008 .*Digital Image Processing*. Prentice Hall. New Jersey

Jähne, Bernd. 2002. *Digital Image Processing.* Springer. Germany

Kolmogorov, Vladimr. *2004. Interactive Foreground Extraction using Iterated Graph Cuts*. Microsoft Research. Cambrige

Kolmogorov, Vladimir. 2004. *An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision.* IEEE Transactions

Mayuresh, Kulkarni. 2010. *Interactive Image Segmentation using Graph Cuts.* Department of Electrical Engineering University of Cape Town. Cape Town

Marsh, Mattew. 2010. *A Literature Review of Image Segmentation Techniques and Matting for the Purpose of Implementing "Grab-Cut"*. grahamstone

Munir, Rinaldi. 2003. *Matematika Diskrit.* Informatika Bandung. Bandung

Orchard and Bouman, C. A. 1991. *Color Quantization of Images.* IEEE Transactions on Signal Processing

Sachs, Jonathan. 1999. *Digital Image Basic*. Digital Light and Color

Sibaroni, yuliant. 2002. *Buku Ajar Aljabar Linear*. STT Telkom. Bandung

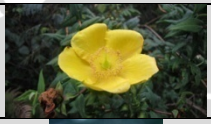
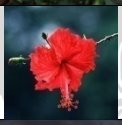
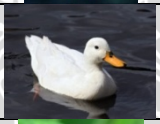
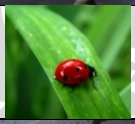
Soemarno. 2010. *Analisis Peubah Ganda (Multivariate Analysis)*

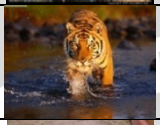
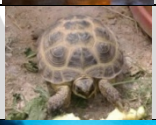
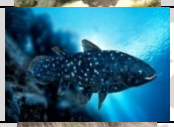
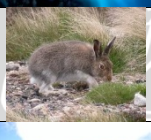
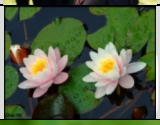
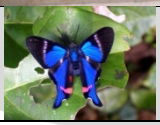
Sonka, Milan. 1993. *Image Processing,Analysis and Machine Vision. Chapman & hall*. Cambridge

Talbot, Justin. 2006. *Implementing Grabcut*. Birmingham Univdersity. Birmingham

LAMPIRAN

Tabel Lampiran I Citra Asal dan Bandingan Untuk Uji Sistem

Nama File	Gambar Awal	Gambar Pembanding	Ukuran
Bungalily.bmp			200x133
Bungakuning.bmp			200x133
bungamerah.bmp			200x197
bebekputih.bmp			200x150
kepik.bmp			200x183
apel.bmp			200x127
aku.bmp			200x150
ayamhitam.bmp			200x139
ikanmerah.bmp			200x150

durian.bmp			200x150
kepalaunta.bmp			200x153
macan.bmp			200x150
kurakura.bmp			200x155
ikanpurba.bmp			200x138
kelinci.bmp			200x162
bersama.jpg			250x186
duabunga.jpg			250x188
enamburung.jpg			250x127
saputerbang.jpg			250x156
Kupukupu.jpg			250x188

Tabel Lampiran 2 Hasil Uji Akurasi pada Citra Uji Coba

File	Iterasi	Ukuran	Koordinat *Trimap*					WP	Akurasi
			x1	y1	x1	y2	time		
bungalily.bmp	1	200x133	23	21	127	100	15	726	97.27
	2						27	709	97.33
	3						38	746	97.19
bungakuning.bmp	1	200x133	48	17	154	90	16	653	97.11
	2						29	627	97.22
	3						42	629	97.21
bungamerah.bmp	1	200x197	40	32	170	184	25	2545	93.54
	2						48	2585	93.43
bebekputih.bmp	1	200x150	22	33	164	129	50	1550	94.83
	2						197	1550	94.83
kepik.bmp	1	200x183	72	84	151	143	9	1376	96.24
	2						15	732	98
	3						25	699	98.09
							35	712	98.05
apel.bmp	1	200x127	25	19	169	117	103	1728	93.1
	2						183	955	96.24
	3						318	980	96.14
aku.bmp	1	200x150	71	16	113	130	6	1326	95.58
	2						13	1402	95.32
ayamhitam.bmp	1	200x139	50	21	164	125	30	2039	92.66
	2						45	1643	94.08
	3						59	1655	94.04
ikanmerah.bmp	1	200x150	40	53	121	104	5	969	96.77
	2						10	1012	96.62
durian.bmp	1	200x150	53	13	159	125	31	2619	91.27
	2						57	3197	89.34
kepalaunta.bmp	1	200x153	51	44	190	148	58	2514	91.78

	2						113	2399	92.16
	3						140	2967	90.3
macan.bmp	1	200x150	76	5	147	127	17	2763	90.94
	2						30	3167	89.54
kurakura.bmp	1	200x155	33	13	168	144	34	11059	64.32
	2						43	13566	56.23
ikanpurba.bmp	1	200x138	24	33	186	125	66	6124	77.81
	2						79	7170	74.21
kelinci.bmp	1	200x162	23	36	159	128	34	6862	78.81
	2						54	7644	76.4
bersama.jpg	1	250x186	15	54	222	184	421	7129	84.66
	2						766	10915	76.52
duabunga.jpg	1	250x188	22	55	237	153	90	4711	89.97
	2						150	4818	89.74
enamburung.jpg	1	250x127	2	39	248	115	68	4568	85.62
	2						117	5540	85.61
saputerbang.jpg	1		21	9	217	141	105	7392	81.04
	2						140	7478	80.82
kupukupu.jpg	1	250x188	31	31	183	168	116	4675	90.05
							177	4970	89.42

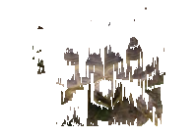
Tabel Lampiran 3 Citra Hasil Uji Akurasi

File	Iterasi	Hasil
bungalily.bmp	1	
	2	
	3	
bungakuning.bmp	1	
	2	
	3	
bungamerah.bmp	1	

	2	
bebekputih.bmp	1	
	2	
kepik.bmp	1	
	2	
	3	
	4	
apel.bmp	1	

	2	
	3	
aku.bmp	1	
	2	
ayamhitam.bmp	1	
	2	
	3	
ikanmerah.bmp	1	

	2	
durian.bmp	1	
	2	
kepalaunta.bmp	1	
	2	
	3	
macan.bmp	1	
	2	

kurakura.bmp	1	
	2	
ikanpurba.bmp	1	
	2	
kelinci.bmp	1	
	2	
bersama.jpg	1	
	2	

duabunga.jpg	1	
	2	
enamburung.jpg	1	
	2	
saputerbang.jpg	1	
	2	
kupukupu.jpg	1	
		

Tabel Lampiran 4 Citra Hasil Uji Kecepatan

Ukuran Gambar		Koordinat *Trimap*				Hasil
Panjang	Lebar	X1	Y1	X2	Y2	
200	133	17	23	130	107	
240	160	20	28	156	128	
288	192	24	33	187	154	
346	230	29	40	225	185	
415	276	35	48	270	222	
498	332	42	57	323	266	

Tabel Lampiran 4 Citra Hasil Uji Analisa *Trimap*

Uji ke	Nama file	X1	Y1	X2	Y2	Hasil
1	Bungalily.jpg	32	41	188	149	
2	Bungalily.jpg	22	31	198	159	
3	Bungalily.jpg	12	21	208	169	
4	Bungalily.jpg	2	11	218	179	
5	Bungalily.jpg	0	1	228	189	
1	Aku.jpg	112	24	168	193	
2	Aku.jpg	102	14	178	203	

3	Aku.jpg	92	4	188	213	
4	Aku.jpg	82	0	198	227	
5	Aku.jpg	72	0	218	227	
1	ikanmerah.jpg	56	82	181	156	
2	ikanmerah.jpg	46	72	191	166	
3	ikanmerah.jpg	36	62	201	176	
4	ikanmerah.jpg	26	52	211	186	
5	ikanmerah.jpg	16	42	221	196