

**DETEKSI AREA KELILING LUKA KRONIS DENGAN
MENGGUNAKAN ALGORITMA GRABCUT**

Skripsi

**Disusun untuk memenuhi salah satu syarat
memperoleh gelar Sarjana Komputer**



*Mencerdaskan dan
Memartabatkan Bangsa*

Oleh:
Muhamamid Hafiz Hisbullah
1313619019

PROGRAM STUDI ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS NEGERI JAKARTA

2024

LEMBAR PERSETUJUAN

Dengan ini saya mahasiswa Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Negeri Jakarta

Nama : Muhammad Hafiz Hisbullah
No. Registrasi : 1313619019
Program Studi : Ilmu Komputer
Judul : DETEKSI AREA KELILING LUKA KRONIS DENGAN MENGGUNAKAN ALGORITMA GRABCUT

Menyatakan bahwa proposal ini telah siap diajukan untuk seminar pra skripsi.

Menyetujui,

Dosen Pembimbing I

Muhammad Eka Suryana, M. Kom.

NIP. 19851223 201212 1 002

Dosen Pembimbing II

Drs. Mulyono, M.Kom.

NIP. 19660517 199403 1 003

Mengetahui,

Koordinator Program Studi Ilmu Komputer

Dr. Ria Arafiah, M.Si

NIP. 19751121 200501 2 004

KATA PENGANTAR

Dengan penuh syukur, penulis ingin mengungkapkan rasa terima kasih kepada Allah SWT, atas berkat-Nya yang memungkinkan penulis menyelesaikan proposal skripsi berjudul *DETEKSI AREA KELILING LUCA KRONIS DENGAN MENGGUNAKAN ALGORITMA GRABCUT*.

Semoga penghargaan ini mencerminkan segenap rasa syukur penulis atas bimbingan, dorongan, dan inspirasi yang telah diberikan. Harapannya, tugas akhir ini dapat memberikan manfaat yang nyata bagi semua pihak yang terlibat, dan khususnya, dapat menjadi langkah maju dalam peningkatan pengetahuan dan pemahaman dalam bidang yang diteliti. Penulis juga berdoa semoga Allah SWT senantiasa membalas kebaikan dan keikhlasan dari semua pihak yang telah turut serta membantu hingga proposal skripsi ini mencapai kesempurnaan.

1. Yth. Para petinggi di lingkungan FMIPA Universitas Negeri Jakarta.
2. Yth. Ibu Dr. Ria Arafiyah, M.Si selaku Koordinator Program Studi Ilmu Komputer.
3. Yth. Bapak Muhammad Eka Suryana, M.Kom selaku Dosen Pembimbing I yang telah membimbing, mengarahkan, serta memberikan saran dan koreksi terhadap proposal skripsi ini.
4. Yth. Bapak Drs. Mulyono, M.Kom selaku Dosen Pembimbing II yang telah membimbing, mengarahkan, serta memberikan saran dan koreksi terhadap proposal skripsi ini.
5. Kedua orang tua dan kakak penulis yang telah mendukung dan memberikan semangat serta doa untuk penulis.
6. Teman-teman Program Studi Ilmu Komputer 2019 yang telah memberikan dukungan dan memiliki andil dalam penulisan proposal skripsi ini.

Dengan penuh kesederhanaan, penulis mengakui bahwa dalam penyusunan proposal skripsi ini, belum mencapai kesempurnaan karena terbatasnya pengetahuan dan pengalaman yang dimiliki. Karena itulah, penulis dengan hati yang lapang menerima kritik serta saran yang konstruktif, yang datang bagi bunga-bunga pengharapan.

Sebagai perjalanan ini mencapai akhir, penulis mengharapkan tugas akhir ini akan menjadi ladang berkah bagi semua pihak yang terlibat, tak terkecuali diri penulis sendiri. Dalam doa yang dipanjatkan, semoga Allah SWT selalu melipatgandakan kebaikan bagi semua yang telah berperan serta membantu penulis meniti jalan menuju puncak kesuksesan dalam menyelesaikan proposal skripsi ini.

Jakarta, 11 Januari 2024

Mochammad Hafiz Hisbullah

DAFTAR ISI

LEMBAR PENGESAHAN	i
KATA PENGANTAR	iii
DAFTAR ISI	iv
DAFTAR GAMBAR	vi
DAFTAR TABEL	vii
I PENDAHULUAN	1
1.1 Latar Belakang Masalah	1
1.2 Rumusan Masalah	5
1.3 Pembatasan Masalah	6
1.4 Tujuan Penelitian	6
1.5 Manfaat Penelitian	6
II KAJIAN PUSTAKA	8
2.1 Citra Gambar Digital	8
2.2 Pemrosesan Citra Digital	9
2.3 Citra Gambar Berwarna	9
2.4 Pemodelan Distribusi <i>Gaussian Mixture Model</i>	12
2.5 Segmentasi Pemisahan Objek dari Latar Belakang dengan Algoritma <i>GraphCut</i>	16
2.5.1 Metode Awal	17
2.5.2 Latar Belakang Pada Graf	18
2.5.3 Algoritma <i>Mincut/Max-Flow</i> Terbaru	21
2.6 Segmentasi Citra dengan Algoritma <i>GrabCut</i>	26
2.6.1 Sistem yang Diusulkan : <i>GrabCut</i>	26
2.6.2 Segmentasi gambar dengan <i>graph cut</i>	27
2.6.3 Segmentasi gambar dengan algoritma <i>GrabCut</i>	30
III METODOLOGI PENELITIAN	35
3.1 Desain Segmentasi Luka dengan Metode <i>GrabCut</i>	35
3.1.1 Inisiasi Kotak Pembatas (<i>Bounding Box</i>)	35
3.1.2 Implementasi Algoritma <i>GrabCut</i>	36
3.2 Diagram Alir Segmentasi Luka dengan Metode <i>GrabCut</i>	36
3.3 Struktur Data	38
3.3.1 <i>Node</i>	38
3.3.2 <i>Linkedlist</i>	39
3.3.3 <i>Tree</i>	41

3.3.4	Graf	43
3.3.5	Algoritma <i>GrabCut</i>	51
3.4	Algoritma Segmentasi Gambar dengan Metode <i>GrabCut</i>	54
3.5	Alat dan Bahan Penelitian	62
3.6	Tahapan Penelitian	62
3.6.1	Persiapan <i>dataset</i> input citra	62
3.6.2	Inisiasi <i>Bounding Box</i> di area luka	64
3.6.3	Inisiasi GMM	64
3.6.4	Mempelajari parameter GMM	64
3.6.5	Segmentasi Citra dengan Algoritma <i>Mincut</i>	65
3.6.6	Menghitung Tingkat Akurasi	67
3.7	Skenario Eksperimen	67
IV HASIL DAN PEMBAHASAN		69
4.1	Pemrosesan citra gambar input	69
4.2	Deteksi keliling luka menggunakan <i>Grabcut</i>	69
4.2.1	Inisiasi kotak (<i>Bounding Box</i>)	69
4.2.2	Inisiasi piksel	70
V KESIMPULAN DAN SARAN		72
5.1	Kesimpulan	72
5.2	Saran	72
DAFTAR PUSTAKA		73
A	<i>Source code</i> program	74
1.1	main.py	74
1.2	image_editing.py	74
1.3	grabcut.py	77
1.4	GMM.py	77
B	Tabel pengolahan data citra input	79
C	Tabel hasil deteksi keliling luka menggunakan <i>Grabcut</i>	88

DAFTAR GAMBAR

Gambar 2.1	Spektrum cahaya, (Gonzalez et al., 2018)	10
Gambar 2.2	Penyerapan cahaya yang ditangkap oleh mata dalam bentuk panjang gelombang, (Gonzalez et al., 2018)	10
Gambar 2.3	Gambar citra berwarna, (Gonzalez et al., 2018)	11
Gambar 2.4	24 bit kubus warna RGB, (Gonzalez et al., 2018)	12
Gambar 2.5	Contoh distribusi tiga fungsi gaussian 1 dimensi dengan parameter K = 3, (Power et al., 2002)	13
Gambar 2.6	Contoh pelabelan gambar. citra (a) adalah himpunan piksel P dengan intensitas teramatip I_p untuk setiap $p \in P$. Pelabelan L yang ditunjukkan pada (b) memberikan beberapa label L_p 0, 1, 2 untuk setiap piksel $p \in P$	18
Gambar 2.7	Contoh graf berkapsitas terarah. (Boykov et al., 2004) . . .	19
Gambar 2.8	Contoh (<i>graph cut</i>)/ <i>flow</i> dalam konteks segmentasi gambar. (a) menunjukkan <i>maximum flow</i> dari s ke t. Faktanya, ini menjelaskan tepi <i>graph</i> yang sesuai dengan batas <i>minimum cut</i> di (b). (Boykov et al., 2004)	20
Gambar 2.9	Contoh pencarian pohon S (<i>node</i> merah) dan T (<i>node</i> biru) (Boykov et al., 2004)	21
Gambar 2.10	Perbandingan beberapa <i>matting tools</i> dan segmentasi. (Rother et al., 2004)	30
Gambar 2.11	Segmentasi gambar di <i>GrabCut</i> (Rother et al., 2004)	33
Gambar 2.12	Konvergensi minimalisasi iteratif untuk data gambar 2.10(f). (a) Energi E untuk contoh llama konvergen selama 12 iterasi dengan K = 5, komponen campuran digunakan untuk latar belakang (merah) dan latar depan (biru) (Rother et al., 2004).	33
Gambar 2.13	Pengeditan pengguna. Setelah interaksi pengguna awal dan segmentasi (baris atas), pengeditan pengguna lebih lanjut (gambar 2.11) diperlukan. (Rother et al., 2004)	34
Gambar 3.1	(a) Data citra luka 2.png, (b) Inisiasi kotak pembatas	35
Gambar 3.2	Hasil segmentasi dengan <i>GrabCut</i>	36
Gambar 3.3	Diagram alir penelitian	37
Gambar 3.4	Diagram alir metode <i>GrabCut</i>	37
Gambar 3.5	Diagram alir tahap <i>Gaussian Mixture Models</i>	38
Gambar 3.6	Representasi <i>node</i> terhadap piksel pada gambar	39
Gambar 3.7	Komponen yang ada pada sebuah <i>node</i>	40
Gambar 3.8	<i>Linkedlist</i> terdiri dari susunan <i>node</i>	40
Gambar 3.9	Representasi struktur data <i>tree</i>	42
Gambar 3.10	Ilustrasi graf terhadap piksel	44
Gambar 3.11	Garis berwarna hijau menunjukkan <i>path</i> s-a-h-i-t	45

Gambar 3.12 (a)Data citra format .xcf, (b) <i>layer</i> citra (luka), (c) <i>layer</i> region, (d)path	63
Gambar 3.13 Proses <i>resize</i> citra gambar luka dengan ukuran 0.5 dari ukuran awal	63
Gambar 3.14 Inisiasi area objek luka dengan <i>bounding box</i>	64
Gambar 3.15 Contoh graf berkapasitas terarah.	66
Gambar 3.16 (a) Data citra luka 2.png, (b) <i>Bounding box</i> area luka, (c) Hasil segmentasi citra	67
Gambar 4.1 <i>source code</i> mengubah ukuran lebar citra menjadi 320 piksel	69
Gambar 4.2 <i>source code</i> menggambar kotak pada area luka	70
Gambar 4.3 <i>source code</i> menggambar kotak pada area luka	71

DAFTAR TABEL

Tabel 2.1	Visualisasi hasil pengolahan data citra input luka hitam	79
Tabel 2.2	Visualisasi hasil pengolahan data citra input luka hitam - lanjutan	80
Tabel 2.3	Visualisasi hasil pengolahan data citra input luka hitam - lanjutan	81
Tabel 2.4	Visualisasi hasil pengolahan data citra input luka kuning	82
Tabel 2.5	Visualisasi hasil pengolahan data citra input luka kuning - lanjutan	83
Tabel 2.6	Visualisasi hasil pengolahan data citra input luka merah	84
Tabel 2.7	Visualisasi hasil pengolahan data citra input luka merah - lanjutan	85
Tabel 2.8	Visualisasi hasil pengolahan data citra input luka merah - lanjutan	86
Tabel 2.9	Visualisasi hasil pengolahan data citra input luka merah - lanjutan	87
Tabel 3.1	Visualisasi hasil deteksi keliling luka menggunakan <i>Grabcut</i>	89
Tabel 3.2	Visualisasi hasil deteksi keliling luka menggunakan <i>Grabcut</i>	90
Tabel 3.3	Visualisasi hasil deteksi keliling luka menggunakan <i>Grabcut</i>	91

BAB I

PENDAHULUAN

1.1 Latar Belakang Masalah

Luka merupakan kerusakan atau gangguan yang terjadi pada struktur anatomi kulit. Hal ini sering kita temui pada permukaan kulit atau pada integritas epitel kulit, luka berkisar dari kerusakan yang bervariasi mulai dari kerusakan sederhana atau terjadi lebih dalam, bahkan bisa saja meluas ke jaringan subkutan yang berdampak pada struktur lain seperti tendon, pembuluh darah, otot, saraf, organ parenkim, dan hingga sampai tulang. Luka timbul karena adanya proses patologis secara internal maupun eksternal, apapun penyebab dan bentuknya, luka dapat merusak jaringan dan mengganggu sistem yang ada didalamnya, respon yang ditimbulkan oleh luka secara fisiologis di antaranya menyebabkan pendarahan, kontraksi pembuluh darah dengan koagulasi, aktivasi komplemen serta respon inflamasi (Velnar et al., 2009).

Luka dapat diklasifikasikan dalam berbagai kriteria. Berdasarkan waktu penyembuhan luka dibagi menjadi dua, yaitu luka akut dan luka kronis. ibid. dalam penelitiannya mengatakan luka akut adalah luka yang dapat penyembuhan secara mandiri dan berlangsung secara normal dengan proses penyembuhan yang membutuhkan waktu yang teratur, luka akut berakhir dengan hasil dari restorasi fungsi anatomis. Luka kronis ialah luka yang proses penyembuhannya gagal berjalan dengan normal, proses penyembuhan luka kronis tidak dapat diperbaiki dengan cepat dan teratur, hal ini dikarenakan adanya gangguan oleh beberapa faktor dalam tahap fase hemostasis, peradangan, proliferasi atau *remodelling*. Luka kronis dapat disebabkan oleh berbagai penyebab, di antaranya naturopati, tekanan, insufisiensi arteri dan vena, diabetes melitus, luka bakar dan vaskulitis.

Proses penyembuhan luka adalah proses kompleks dengan serangkaian interaksi beragan antara sistem imunologi dan biologis, penyembuhan luka ini terdiri dari berbagai fase dengan langkah dan peristiwa yang berlangsung dengan perlahan dan dilakukan secara sistematis sehingga muncul berbagai jenis sel badi dasar luka selama proses penyembuhan berlangsung (ibid.). Ketika seseorang mengalami luka dibagian jaringan kulit, penanganan yang biasanya dilakukan adalah menutup luka tersebut agar tidak terjadi pendarahan terus menerus, hal tersebut bisa dilakukan jika luka yang dialami ialah luka kecil dan hanya dipermukaan saja, namun jika luka

tersebut merupakan luka kronis, maka disarankan untuk melakukan penanganan dan pengkajian ke rumah sakit agar segera dilakukan tindakan oleh dokter atau perawat.

Luka kronis menjadi salah satu permasalahan bagi beberapa pihak. Bagi pasien yang memiliki luka kronis khususnya akibat penyakit Diabetes Melitus (DM) akan menghabiskan banyak biaya dalam pengobatannya. Wang et al., 2015 dalam penelitiannya menyebutkan di Amerika Serikat jutaan pasien penderita luka kronis mengeluarkan miliaran uang tiap tahunnya, pasien penderita luka kronis akibat Diabetes Melitus (DM) saja bisa menghabiskan 38 miliar dolar, kebanyakan di antaranya biaya rawat inap dan operasi di rumah sakit, dan biaya perawatan jangka panjang yaitu perawatan dari rumah secara berkala. Di Amerika tercatat jumlah pasien penderita diabetes mencapai 20 juta dan diperkirakan pada tahun 2030 jumlahnya akan naik dua kali lipat (Han, 2017). Tentu saja dengan jumlah begitu banyak maka perawat dan rumah sakit yang menangani perawatan luka akan memakan banyak waktu dan mengeluarkan banyak biaya (Wang et al., 2015), hal ini mengakibatkan tenaga perawat yang menangani pasien penderita luka kronis membutuhkan banyak waktu.

Proses penyembuhan luka terjadi dalam beberapa tahapan, seorang perawat luka wajib memberikan asesmen perawatan luka sesuai prosedur medis agar keadaan luka segera membaik dan menghindari terjadinya infeksi. Proses penyembuhan yang dilakukan pertama kali ialah membersihkan dan dibalut dengan benar, setelah luka dibersihkan maka akan dilakukan metode debridemen luka, hal ini bertujuan untuk mengangkat jaringan (nektrotik) yang mati, terinfeksi dan penebalan pada jaringan kulit (hiperkerotoris), membentuk dasar penyembuhan luka. Debridemen memiliki fungsi yang penting dalam asesmen luka karena akan mempercepat proses penyembuhan luka, debridemen yang ada pada luka kronis berfungsi sebagai penanganan kelainan medis dan mengubah kronis tersebut menjadi luka akut, setelah menjadi luka akut maka penyembuhan akan kembali normal (Velnar et al., 2009).

Proses asesmen luka dilakukan bertahap, tindakan yang diberikan tiap tahap penyembuhan dilakukan berdasarkan kajian evaluasi luka, kajian ini memantau proses penyembuhan luka secara berkala (Silva et al., 2021). Metode evaluasi luka biasanya menggunakan metode invasif (kontak) dan non-invasif (non-kontak) (Manohar Dhane et al., 2017).

Hal penting yang dijadikan sebagai indikator untuk penyembuhan luka ialah dengan memperhatikan ukuran luka, di antaranya perubahan luas, kedalaman dan jenis jaringan yang luka (Silva et al., 2021). Seorang perawat luka melakukan

inspeksi kontak luka dengan mengukur luka dilakukan secara manual yaitu dengan bantuan penggaris luka dan label perekat yang bersentuhan langsung dengan luka, dalam penelitiannya Silva et al., 2021 menyebutkan teknik ini cenderung dapat menyebabkan resiko infeksi, mengganggu kenyamanan dan memperburuk kondisi klinis pasien, ditambah penggunaan teknik-teknik tersebut tidak akurat, tidak konsisten, dan tentu mengalami kekurangan standar penilaian. Standar pengukuran dengan metode manual tersebut memiliki tingkat kesalahan yang cukup tinggi yaitu sekitar 44 persen (Rizki, 2022).

Para ilmuwan telah banyak melakukan penelitian mengenai hal ini, agar proses asesmen dan kajian luka dapat dilakukan secara efektif dan efisien mulai dari waktu, tenaga, hingga pengeluaran biaya yang cukup banyak, teknologi dibidang pemrosesan citra gambar menjadi hal yang memungkinkan saat ini untuk dikembangkan, dengan menggunakan pemrosesan gambar yang dibantu oleh pembelajaran mesin memungkinkan melakukan analisis gambar luka oleh program komputer (Wang et al., 2015).

Silva et al., 2021 berpendapat bahwa pengembangan citra digital untuk melakukan evaluasi luka merupakan alternatif yang sangat penting, teknik ini akan menginformasikan analisis yang lebih objektif dan reliabel. Mereka melakukan penelitian mengenai pengusulan untuk menentukan area luka dengan menggunakan pengklasifikasi berbasis citra medis yaitu *Support Vector Machine* (SVM) serta mengkombinasikannya dengan metode *GrabCut* untuk segmentasi area yang terkena luka. Metode segmentasi gambar ini sepenuhnya otomatis serta perawat tidak perlu melakukan kontak langsung dengan objek, tingkat akurasinya diperkirakan mencapai 96 persen, sensitifitas sebesar 94 persen, spesifitas 97 persen, tingkat presisi 94 persen dan interaksi penyatuhan 89 persen.

Langkah pertama seorang perawat dalam melakukan asesmen luka digital ialah dengan mengambil foto luka menggunakan kamera telepon pintar atau tablet nya, ketika ada dua foto dengan pose yang sama namun dengan perangkat kamera yang berbeda, warna yang dihasilkan kemungkinan akan berbeda. Untuk menangani hal tersebut adalah setiap kamera menggunakan *device independent sRGB*. Zaman sekarang sudah banyak vendor kamera telepon pintar menyediakan mode ini namun dengan pengaturan pewarnaan RGB mereka masing-masing. Oleh karena ini dibutuhkan adanya kalibrasi dengan melakukan transformasi citra menjadi ruang warna CIE, kemudian menjadi sRGB dengan serangkaian optimalisasi (Rizki, 2022).

Silva et al., 2021 menjelaskan dalam karyanya bahwa parameter yang

digunakan untuk segmentasi daerah yang terkena luka ialah perbedaan warna antara kulit dan luka, berdasarkan ambang batas semi otomatis dalam ruang pewarnaan RGB, Manohar Dhane et al., 2017 menggunakan metode pengelompokan spektral fuzzy untuk segmentasi warna berdasarkan tingkat kesamaan fuzzy (tingkat abu-abu) yang dihitung di atas gambar, hasil menunjukkan pada 70 gambar mencapai akurasi 92 persen, sensitifitas 87 persen, dan spesifisitas 96 persen.

Sebenarnya ada beberapa metode yang bisa digunakan untuk melakukan image processing dengan bantuan *Support Vector Machine* (SVM) seperti kesamaan fuzzy, *active contour* (snake), dan *region-of-interest* (ROI), dan *GrabCut*. Metode-metode yang pernah dilakukan merupakan pendekatan untuk menghasilkan objek luka yang akurat Garcia-Zapirain et al., 2017. Beberapa di antara metode yang disebutkan juga memiliki keterbatasan yang beragam, mulai dari inisiasi awal gambar yang manual, atau intensitas warna yang bergantung dari intensitas gradien.

Rizki, 2022 dalam penelitiannya yaitu pendekripsi luka menggunakan metode *active contour* (snake) dan *active contour* (snake) dengan ditambah interpolasi, dalam penelitiannya mencari *ground truth* dalam objek-objek gambar luka pasien, ketika dilakukan pemrosesan gambar dengan metode yang dijalankan, *ground truth* menunjukkan hasil yang kurang maksimal dimana hasil dari deteksi luka dengan metode snake versi integer hanya berhasil menutupi luka berjumlah 12 data dari total 71 data yang tersedia, sedangkan metode snake interpolasi berjumlah 44 data dari total 71 data yang tersedia. Silva et al., 2021 melakukan penelitian mengenai pengolahan citra gambar dengan menggunakan *Support Vector Machine* (SVM) dengan menggunakan *GrabCut*, dalam tulisannya menerangkan bagaimana cara segmentasi gambar luka dengan menggunakan *GrabCut*, setidaknya ada lima tahapan dalam prosesnya, di antaranya segmentasi superpiksel, ekstraksi fitur, persiapan data, klasifikasi dan terakhir yaitu segmentasi luka.

Segmentasi superpiksel merupakan teknik untuk merepresentasi ringkas dari gambar menjadi kelompok piksel yang lebih kecil sesuai dengan spasial dan kriteria warna. Segmentasi superpiksel ini digunakan oleh beberapa metode di antaranya SEED, LSC dan SLIC. Metode ini menghasilkan beberapa informasi dari gambar luka seperti tepi luka dan parameter yang digunakan untuk metode selanjutnya (*ibid.*). Wang et al., 2015 menggunakan gambar berukuran 480 x 640 piksel, hal ini dikarenakan untuk memangkas biaya komputasi dalam pemrosesan, sampel diambil secara acak, sementara SVM linier untuk melatih data yang tersedia. Setelah data telah tersegmentasi menjadi superpiksel, maka dilakukan proses klasifikasi dengan

menggunakan *Support Vector Machine* (SVM), hal ini diperlukan karena pada ditahap ini banyak terdapat superpiksel berada di sekitar luka pada kulit dan dikhawatirkan dapat salah dalam klasifikasi sehingga menurunkan tingkat akurasi segmentasi. Proses selanjutnya dalam pengolahan citra gambar luka ialah segmentasi luka, salah satu metode yang paling populer untuk segmentasi citra gambar ialah menggunakan algoritma *GrabCut*, teknik ini merupakan teknik yang bekerja berdasarkan analisis statistik serta teori grafik, hal ini bertujuan untuk memisahkan suatu objek inti gambar dari objek sisa di sekitar gambar, pada akhirnya data telah menghasilkan suatu objek citra yang berisi area luka beserta komponen yang ada pada data tersebut. (Nugraha, 2022) menggunakan metode *GrabCut* dalam penelitiannya mengenai ekstraksi latar depan citra ikan, hasil yang didapatkan dari dataset untuk dilakukan uji coba bahwa apabila *GrabCut* diuji pada data citra multi objek, maka objek yang ada pada data tersebut akan gagal diseleksi oleh *GrabCut*, namun jika data tersebut hanya terdapat satu objek maka *GrabCut* berhasil melakukan seleksi citra dengan baik dan menghasilkan data yang bagus.

Di dalam penelitian ini, penulis akan melakukan penerapan algoritma *GrabCut* pada pemrosesan citra gambar yang akan menghasilkan segmentasi area luka, metode yang penulis pilih berdasarkan hasil dari penelitian Muhammad Rizki dimana *ground truth* (area sebenarnya) yang dihasilkan lebih banyak yang gagal dibandingkan yang berhasil, sehingga penulis tertarik untuk mengganti metode yang dijalankan untuk segmentasi area keliling luka kronis. Selain itu penulis memilih metode *GrabCut* untuk dijadikan sebagai penelitian dikarenakan metode ini sangat cocok untuk citra gambar luka dimana yang merupakan citra satu objek (*single object*). Tahap pertama penulis akan mendai daerah yang mencakup objek luka, kemudian objek akan dilakukan pengujian metode *GrabCut* pada citra tunggal (*single object*). Dalam penelitian ini penulis akan menggunakan dataset citra yang tersedia di repositori <https://github.com/mekas/InjuryDetection>. Dataset citra ini berasal dari penelitian luka Ns. Ratna Aryani M.Kep, tahun 2018 (Aryani et al., 2018). Diharapkan dalam penelitian ini mendapatkan hasil berupa nilai akurasi antara metode *GrabCut* dengan hasil citra referensi.

1.2 Rumusan Masalah

Berdasarkan uraian pada latar belakang yang diutarakan di atas, maka perumusan masalah pada penelitian ini adalah Bagaimana cara mendeteksi keliling

luka dengan menggunakan metode *GrabCut*?

1.3 Pembatasan Masalah

Adapun beberapa pembatasan telah diterapkan dalam penelitian ini guna menjaga fokus pada permasalahan yang dijelaskan sebelumnya. Berikut adalah batasan-batasan yang telah diterapkan:

1. Mendeteksi area keliling luka kronis menggunakan metode *GrabCut* dengan data citra luka yang didapat dari penelitian luka Ns. Ratna Aryani, M.Kep, tahun 2018.
2. Penelitian ini hanya berfokus pada pemisahan antara objek utama dan latar belakang pada citra gambar luka.
3. Objek utama, yaitu area luka, diharapkan hanya terdapat satu area luka pada setiap citra.
4. Citra-citra yang digunakan dalam penelitian ini adalah citra digital yang menggunakan sistem warna RGB.
5. Ukuran panjang citra yang dijadikan objek penelitian adalah 320 piksel.
6. Bahasa pemrograman yang digunakan dalam penelitian ini adalah Python v 3.9

1.4 Tujuan Penelitian

Penelitian ini bertujuan untuk membuat sistem untuk mendeteksi objek luka pada citra luka dengan menggunakan metode *GrabCut*.

1.5 Manfaat Penelitian

1. Bagi Penulis
 - Meningkatkan wawasan dan pengalaman praktis terkait deteksi latar depan pada citra luka melalui penggunaan metode *GrabCut*.
 - Untuk memenuhi persyaratan kelulusan dalam program Sarjana (S1) di Program Studi Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Negeri Jakarta

2. Bagi Instansi Terkait

Metode yang diajukan diharapkan dapat membuka peluang untuk diajukan ke instansi kesehatan terkait dalam proses pengkajian luka kronis.

3. Bagi Ilmu Pengetahuan

- Mahasiswa

Penulis berharap penelitian ini dapat digunakan sebagai sumber penunjang referensi, khususnya Pustaka tentang deteksi keliling luka kronis dengan menggunakan metode *GrabCut*.

- Bagi Peneliti Selanjutnya

Diharapkan penelitian ini dapat digunakan sebagai dasar atau kajian awal bagi peneliti lain yang ingin meneliti permasalahan yang sama.

4. Bagi Universitas Negeri Jakarta

Menjadi pertimbangan dan evaluasi akademik khususnya Program Studi Ilmu Komputer dalam penyusunan skripsi sehingga dapat meningkatkan kualitas akademik di program studi Ilmu Komputer Universitas Negeri Jakarta serta meningkatkan kualitas lulusannya.

BAB II

KAJIAN PUSTAKA

2.1 Citra Gambar Digital

Sebuah gambar dapat diberikan definisi sebagai fungsi dua dimensi $f(x, y)$, dimana x dan y mewakili koordinat ruang (bidang) dan amplitudo f pada setiap pasangan koordinat (x, y) merujuk pada intensitas atau level keabuan citra pada koordinat tersebut. Jika x , y , dan nilai intensitas f semuanya terbatas (*finite*) dan memiliki nilai diskret, maka gambar tersebut dianggap sebagai gambar digital.

Dalam hal ini, fungsi $f(s, t)$ merujuk pada fungsi gambar kontinu dengan dua variabel kontinu s dan t . Fungsi ini kemudian diubah menjadi gambar digital melalui proses *sampling* dan kuantisasi. Melalui proses ini, sampel-sampel dari gambar kontinu diubah menjadi gambar digital $f(x, y)$. Gambar digital dapat direpresentasikan dalam bentuk matriks (*array*) yang terdiri dari nilai-nilai numerik $f(x, y)$.

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

Matriks (*array*) diatas adalah representasi yang digunakan dalam pengolahan komputer (Gonzalez et al., 2018), dimana

$f(x, y)$ = Fungsi gambar citra digital

M = Banyaknya baris

N = Banyaknya kolom

x = $0, 1, 2, \dots, M - 1$

y = $0, 1, 2, \dots, N - 1$

Sebuah citra digital merupakan sejumlah elemen yang memiliki batasan (*finite*), dan setiap elemen ini memiliki nilai dan posisi tertentu. Setiap elemen dari array ini disebut sebagai elemen gambar, elemen citra, piksel, atau pel. Piksel-piksel tertentu merepresentasikan nilai-nilai dalam array yang terletak pada pasangan

koordinat (x, y) yang tetap. Piksel merupakan istilah yang paling umum digunakan untuk merujuk pada elemen-elemen citra digital (Gonzalez et al., 2018).

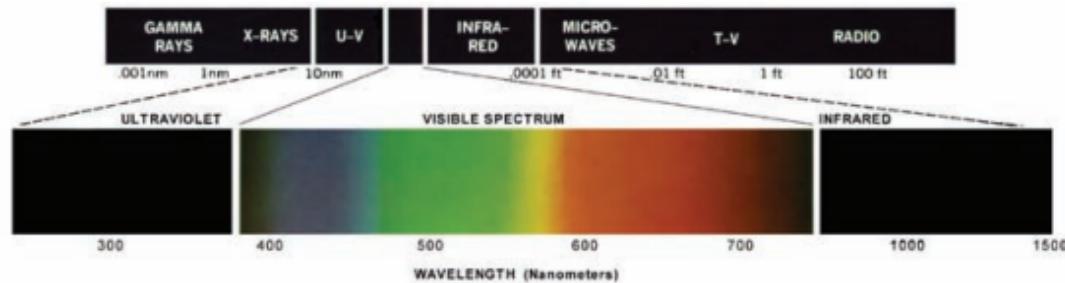
2.2 Pemrosesan Citra Digital

Pemrosesan gambar digital adalah proses yang dilakukan pada citra sebagai masukan dengan melalui beberapa tahapan seperti pra-pemrosesan citra tersebut, mengekstrak (memsegmentasi) elemen, mendeskripsikan elemen dalam bentuk yang sesuai untuk pemrosesan citra tersebut.

ibid. menyebutkan bahwa pemrosesan citra digital dapat dikelompokkan menjadi tiga tingkatan, yakni rendah, menengah, dan tinggi. Pada tingkat pengolahan rendah, dilakukan operasi dasar seperti pra-pemrosesan citra guna mengurangi *noise*, mempertajam kontras, serta memperjelas citra. Proses ini memfokuskan pada citra sebagai input maupun outputnya. Pengolahan tingkat menengah pada citra melibatkan langkah-langkah seperti membagi citra menjadi wilayah atau objek (segmentasi), menguraikan objek dalam citra untuk disesuaikan dengan pemrosesan komputer, dan mengenali objek secara keseluruhan. Pengolahan tingkat menengah ditandai oleh masukan yang umumnya berupa citra, dengan hasil keluaran berupa atribut yang diekstraksi dari citra tersebut (contohnya, tepi, kontur, dan identitas objek). Akhirnya, pada tingkat pengolahan citra yang lebih tinggi, terdapat pengenalan objek dalam citra.

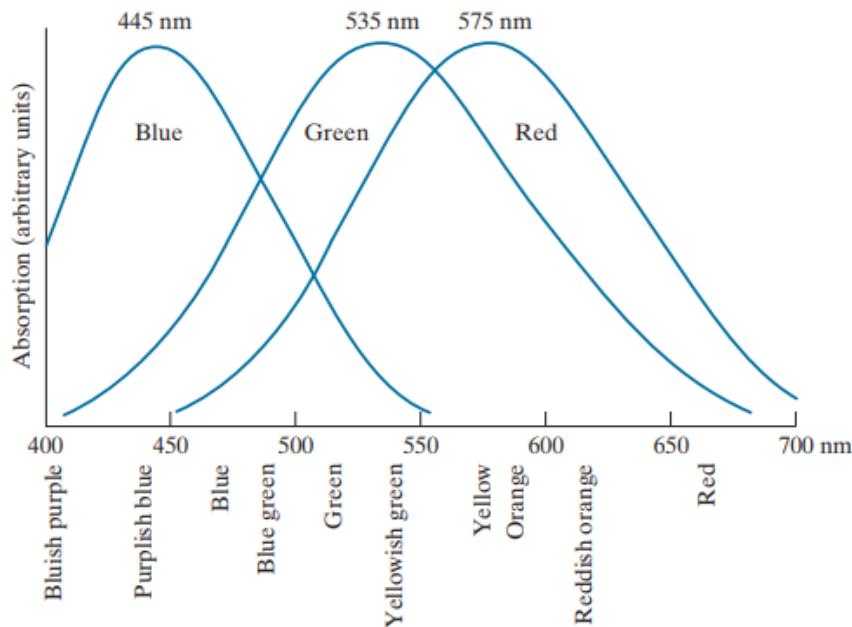
2.3 Citra Gambar Berwarna

ibid. dalam bukunya mengatakan bahwa Warna yang kita lihat pada objek dipengaruhi oleh cahaya yang dipantulkannya. Cahaya terlihat terdiri dari spektrum sempit dalam spektrum elektromagnetik. Objek yang memantulkan cahaya merata pada semua panjang gelombang terlihat putih, sementara yang memantulkan dalam rentang tertentu terlihat berwarna. Contohnya, objek hijau memantulkan cahaya dengan panjang gelombang 500-570 nm, menyerap pada panjang gelombang lainnya.



Gambar 2.1: Spektrum cahaya, (Gonzalez et al., 2018)

Mata manusia memiliki sel kerucut yang sensitif terhadap warna. Sekitar 65% sensitif terhadap merah, 33% terhadap hijau, dan hanya 2% terhadap biru. Meskipun jarang, sel biru sangat sensitif. Sel-sel ini menyerap cahaya dengan karakteristik berbeda, sehingga mata kita melihat warna dengan gabungan merah (R), hijau (G), dan biru (B).



Gambar 2.2: Penyerapan cahaya yang ditangkap oleh mata dalam bentuk panjang gelombang, (Gonzalez et al., 2018)

CIE menetapkan standar warna pada 1931, tetapi data eksperimental yang lebih akurat muncul pada 1965. Standar CIE hanya sekitar cocok dengan data eksperimental. Standar ini menetapkan panjang gelombang biru = 435,8 nm, hijau = 546,1 nm, dan merah = 700 nm sebagai warna primer.

Citra gambar yang direpresentasikan kedalam model warna RGB terdiri dari tiga bagian yang mewakili warna dasar: merah, hijau, dan biru. Ketika gambar ini ditampilkan di layar, ketiga bagian ini bergabung untuk membentuk gambar berwarna. Kita menyebutnya model warna RGB karena ini adalah singkatan dari *Red* (merah), *Green* (hijau), dan *Blue* (biru). Setiap bagian gambar ini memerlukan sejumlah "bit" (ini seperti blok kecil yang menyimpan informasi), dan jumlah bit yang digunakan untuk setiap bagian piksel ini disebut "kedalaman piksel". Jika kita memiliki gambar RGB dengan setiap bagian (merah, hijau, dan biru) menggunakan 8 bit, maka setiap "piksel warna" dalam gambar ini (yang memiliki tiga nilai, satu untuk merah, satu untuk hijau, dan satu untuk biru) akan menggunakan total 24 bit (8 bit untuk merah + 8 bit untuk hijau + 8 bit untuk biru).



Gambar 2.3: Gambar citra berwarna, (Gonzalez et al., 2018)

Gambar berwarna penuh sering disebut gambar RGB 24-bit. Ini artinya setiap piksel dalam gambar memiliki 24 bit untuk menyimpan informasi warnanya. Jumlah total warna yang bisa direpresentasikan dalam gambar RGB 24-bit sangat besar, sekitar 16 juta warna yang berbeda (Gonzalez et al., 2018).



Gambar 2.4: 24 bit kubus warna RGB, (Gonzalez et al., 2018)

Untuk citra gambar digital, rentang nilai dalam kubus diukur dalam angka yang dapat direpresentasikan oleh jumlah bit dalam gambar. Jika, seperti di atas, gambar utama adalah gambar 8-bit, batas kubus sepanjang setiap sumbu menjadi [0, 255]. Sebagai contoh, warna putih akan berada pada titik [255, 255, 255] dalam kubus.

2.4 Pemodelan Distribusi *Gaussian Mixture Model*

GMM (*Gaussian Mixture Model*) adalah sebuah metode statistik yang dapat digunakan untuk memodelkan data sebagai kombinasi beberapa distribusi Gauss. Metode ini digunakan dalam segmentasi gambar untuk memodelkan gambar dalam bentuk gabungan dari distribusi Gaussian. GMM digunakan untuk memperkirakan probabilitas piksel yang termasuk ke dalam objek yang ingin di-segmentasi dan probabilitas piksel yang termasuk ke dalam latar belakang.

Power et al., 2002 dalam penelitiannya mengatakan setiap piksel terhadap objek diberi *state* dari himpunan K , di mana K adalah jumlah konstan biasanya antara 3 dan 7, Rother et al., 2004 memberikan nilai $K = 5$. Beberapa *state* K mewakili objek latar belakang sedangkan sisanya dianggap sebagai objek depan. $\omega_k = P(k), k = 1, 2, \dots, K$, yang menunjukkan probabilitas priori munculnya permukaan k dalam pandangan piksel yaitu:

$$\sum_{k=1}^K \omega_k = 1 \quad (2.2)$$

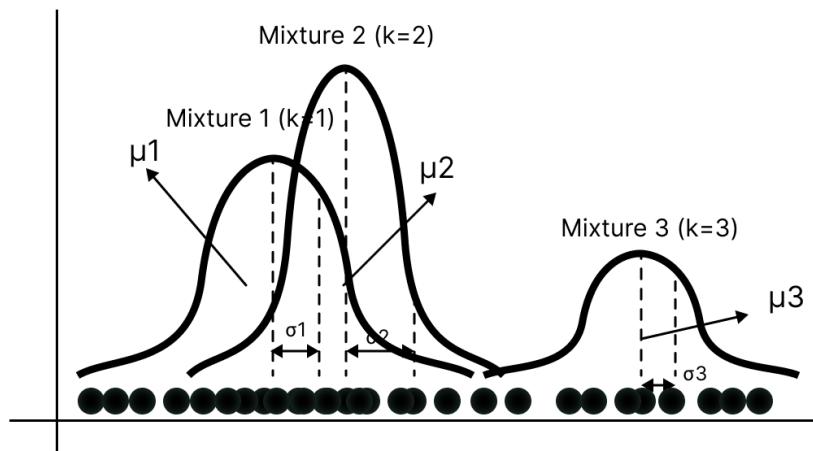
dimana,

K = Parameter konstanta

ω_k = Probabilitas priori

Proses yang menghasilkan *state* permukaan k tidak dapat di amati langsung dan hanya dapat di amati secara tidak langsung melalui nilai piksel terkait X . Bahkan jika kita tahu permukaan k yang sedang dilihat, nilai piksel masih memiliki distribusi $f(X|k)$ karena faktor-faktor seperti perubahan pencahayaan, noise kamera, atau tekstur permukaan. Nilai piksel merupakan sampel dari variabel acak X yang mencakup perilaku k . X dapat berupa satu dimensi (intensitas monokrom), dua dimensi (ruang warna), tiga dimensi (warna), atau n -dimensi secara umum.

Penulis membuat ilustrasi dari parameter gaussian dalam bentuk kurva yang berdistribusi normal dalam satu dimensi:



Gambar 2.5: Contoh distribusi tiga fungsi gaussian 1 dimensi dengan parameter $K = 3$, (Power et al., 2002)

Terlihat bahwa terdapat tiga fungsi gaussian, dengan parameter $K = 3$. Setiap gaussian menjelaskan data-data yang terdapat pada tiga *mixture* yang ada.

Untuk memecahkan masalah segmentasi latar depan, *state* k yang paling mungkin diestimasi pada setiap waktu sampel t dari sekelompok observasi yang diambil dari X , beserta prosedur untuk membedakan antara *state* latar depan dan latar belakang.

Proses nilai piksel X diasumsikan direpresentasikan oleh kombinasi dari K densitas Gaussian, masing-masing dengan parameter θ_k , di mana setiap *state* k

memiliki parameter sendiri, secara umum rumus distribusi normal didefinisikan 4 sebagai berikut :

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-0.5\left(\frac{x-\mu}{\sigma}\right)^2} \quad (2.3)$$

dimana,

- $f(x)$ = Distribusi normal
- σ = Standar deviasi
- x = Nilai variabel bebas
- μ = Rata-rata

Persamaan di atas merupakan rumus dari distribusi normal (gaussian) untuk 1 dimensi, dimana parameter yang digunakan adalah standar deviasi (σ). Sedangkan untuk distribusi gaussian n-dimensi maka digunakan persamaan berikut :

$$f_{X|k}(X|k, \theta_k) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma_k|^{\frac{1}{2}}} e^{-\frac{1}{2}(X-\mu_k)^T \Sigma_k^{-1} (X-\mu_k)} \quad (2.4)$$

dimana,

- $f_{X|k}(X|k, \theta_k)$ = Distribusi normal multivariat
- Σ_k = Matriks kovarians
- X = Vektor variabel bebas
- μ_k = Vektor rata-rata komponen ke-k

Rata-rata μ_k dan matriks kovarian Σ_k dari densitas ke-k digunakan dalam rumus tersebut. Biasanya diasumsikan bahwa dimensi X adalah independen, yang memungkinkan Σ_k menjadi diagonal dan lebih mudah dibalik. Variansi σ^2_k berdimensi-n sering digunakan untuk merepresentasikan Σ_k dengan n varians tersebut identik, artinya deviasi dalam berbagai dimensi dari ruang warna (seperti merah, hijau, dan biru) memiliki statistik yang sama. Meskipun skalar tunggal σ^2_k mungkin menjadi pendekatan yang wajar dalam ruang warna linear, namun mungkin tidak akurat dalam aplikasi lain. Ruang warna non-linear seperti hue, saturasi, dan nilai (*value*), serta ruang yang menggabungkan berbagai kuantitas seperti intensitas dan jangkauan, memerlukan perhatian khusus karena setiap dimensi kemungkinan memiliki distribusi yang unik.

Set parameter untuk densitas didefinisikan sebagai $\theta_k = \mu_k, \sigma_k$ untuk suatu nilai k, dan set lengkap parameter adalah $\Phi = \omega_1, \dots, \omega_K, \theta_1, \dots, \theta_K$. Karena kejadian

k saling eksklusif, distribusi X dapat direpresentasikan sebagai campuran Gaussian, di mana setiap Gaussian sesuai dengan suatu nilai k tertentu (seperti yang ditunjukkan pada Gambar 1). Hal ini dapat dinyatakan sebagai berikut:

$$f_X(X|\Phi) = \sum_{k=1}^K P(k) f_{X|k}(X|k, \theta_k) \quad (2.5)$$

dimana,

$$\begin{aligned} f_X(X|\Phi) &= \text{Jumlah distribusi gaussian} \\ P(k) &= \text{Distribusi priori} \end{aligned}$$

di mana $P(k) = \omega_k$. Semua parameter Φ , termasuk probabilitas $P(k)$ dan parameter dari setiap Gaussian, harus diestimasi dari observasi X, sambil secara simultan memperkirakan keadaan tersebunyi k.

Setiap gaussian pada k terdiri dari beberapa parameter di antaranya sebagai berikut :

1. Rata-rata μ yang menginterpretasikan posisi titik puncak dari kurva
2. Matriks kovarians Σ yang menginterpretasikan sebagai lebar kurva
3. Probabilitas prior ω merupakan peluang sebuah data berasal dari suatu *mixture* tertentu dengan jumlah maksimal sama dengan 1

Dengan diasumsikan bahwa terdapat K distribusi yang dapat menghasilkan sampel X, probabilitas posterior $P(k|X, \Phi)$ mewakili probabilitas bahwa nilai piksel termasuk dalam state k, yang diberikan oleh teorema Bayes:

$$P(k|X, \Phi) = \frac{\omega_k f_{X|k}(X|k, \theta_k)}{f_X(X|\Phi)} \quad (2.6)$$

dimana,

$$P(k|X, \Phi) = \text{Probabilitas posterior}$$

Probabilitas ini dihitung dengan menggabungkan probabilitas prior $P(k)$ dengan *likelihood* $f_{X|k}(X|k, \theta_k)$ dan *likelihood* total $f_X(X|\Phi)$. Nilai k yang memaksimalkan $f_X(X|\Phi)$ adalah estimasi MAP (maximum a posteriori) k ini dapat dihitung menggunakan rumus:

$$\begin{aligned} k &= \arg \max_k P(k|X, \Phi) \\ &= \arg \max_k \omega_k f_{X|k}(X|k, \theta_k) \end{aligned} \quad (2.7)$$

dimana,

$\arg \max_k$ = Nilai k yang membuat $P(k|X, \Phi)$ menjadi maksimum

Dengan $f_X(X|\Phi)$ dalam persamaan (2.6) tidak tergantung pada k . Setelah memperoleh nilai k , selanjutnya menduga parameter GMM menggunakan metode *maximum likelihood* untuk memaksimumkan fungsi *likelihood*, dimana rumus fungsi *likelihood* adalah sebagai berikut:

$$P(X_1, X_2, \dots, X_N, k|\Phi) = \prod_{t=1}^N \omega_k f_{X|k}(X_t|k, \theta_k) \quad (2.8)$$

Adapun parameter penduga yang memaksimumkan persamaan di atas adalah sebagai berikut :

$$\hat{\omega}_k = \frac{1}{N} \sum_{t=1}^N P(k|X_t, \Phi) \quad (2.9)$$

$$\hat{\mu} = \frac{\sum_{t=1}^N X_t P(k|X_t, \Phi)}{\sum_{t=1}^N P(k|X_t, \Phi)} \quad (2.10)$$

$$\hat{\Sigma}_k = \frac{\sum_{t=1}^N ((X_t - \hat{\mu}_k) \cdot (X_t - \hat{\mu}_k)) P(k|X_t, \Phi)}{\sum_{t=1}^N P(k|X_t, \Phi)} \quad (2.11)$$

dimana,

$\hat{\omega}_k$ = Penduga maksimum dari ω_k

$\hat{\mu}$ = Penduga maksimum dari μ

$\hat{\Sigma}_k$ = Penduga maksumum dari Σ_k

2.5 Segmentasi Pemisahan Objek dari Latar Belakang dengan Algoritma *GraphCut*

Pendekatan segmentasi yang dilakukan Boykov dan Kolmogorov sebagai pondasi mengenai algoritma *GrabCut* dijelaskan secara mendetail.

2.5.1 Metode Awal

Dalam penelitiannya Boykov et al., 2004 menjelaskan bahwa Greig et al merupakan yang pertama kali menemukan bahwa algoritma *min-cut/max-flow* dari optimasi kombinasi, algoritma ini dapat digunakan untuk meminimalkan fungsi energi tertentu. Energi yang dibahas selanjutnya dapat direpresentasikan sebagai:

$$E(L) = \sum_{p \in P} D_p(L_p) + \sum_{(p,q) \in N} V_{p,q}(L_p, L_q) \quad (2.12)$$

dimana,

$E(L)$ = Cost Function ()

$D_p(L_p)$ = Data penalti, pelabelan dengan fungsi *likelihood*

$V_{p,q}(L_p, L_q)$ = Interaksi potensial yaitu diskontinuitas antar piksel

p = Piksel pada gambar

P = Gambar citra digital

$L = L_p | p \in P$ merupakan pelabelan dari gambar P , $D_p(\cdot)$ adalah fungsi data penalti, $V_{p,q}$ adalah interaksi potensial, dan N adalah himpunan dari semua pasangan piksel sebelahnya. Contoh pelabelan gambar ditunjukkan pada Gambar 2.6. Data pinalti $D_p(\cdot)$ menunjukkan preferensi label dari setiap piksel berdasarkan intensitas yang di amati dan ditentukan oleh fungsi *likelihood*. Interaksi potensial $V_{p,q}$ mendorong koherensi spasial dengan memberi diskontinuitas antara piksel sebelahnya.

Greig membuat grafik dengan dua terminal, yang memungkinkan *cost minimum cut* grafik untuk menghasilkan pelabelan L biner yang optimal secara global dalam kasus model interaksi Potts seperti yang dijelaskan dalam persamaan 2.12. Sebelum ini, tidak mungkin untuk secara tepat meminimalkan energi seperti 2.12, dan algoritma iteratif seperti simulasi anil biasanya digunakan sebagai gantinya. Terlepas dari keefektifannya, teknik *graph cut* Greig et al. sebagian besar tidak diperhatikan selama hampir satu dekade. Hal ini terutama disebabkan oleh fakta bahwa penerapannya dalam restorasi citra biner dianggap sangat terbatas. Awalnya, upaya untuk menggunakan algoritma *graph cut* kombinatorial dalam *computer vision* sebagian besar difokuskan pada pengelompokan gambar. Namun, pada akhir 1990-an, sejumlah besar teknik *computer vision* baru muncul, yang mendemonstrasikan bagaimana algoritma *min-cut/max-flow* dapat diterapkan pada graf untuk memecahkan-masalah non-biner yang lebih kompleks.



Gambar 2.6: Contoh pelabelan gambar. citra (a) adalah himpunan piksel P dengan intensitas teramati I_p untuk setiap $p \in P$. Pelabelan L yang ditunjukkan pada (b) memberikan beberapa label L_p 0, 1, 2 untuk setiap piksel $p \in P$.

(Boykov et al., 2004)

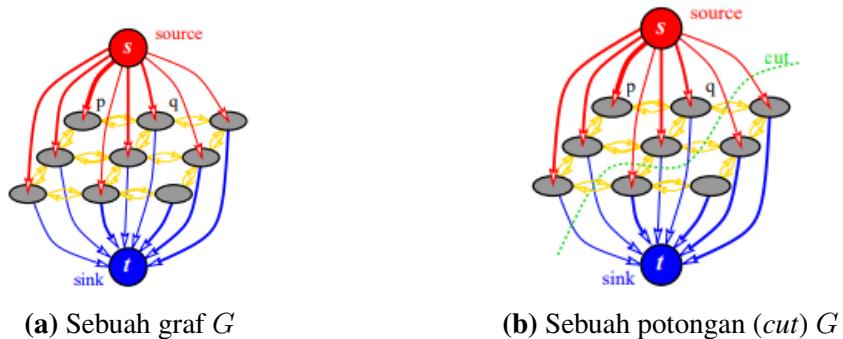
Pada gambar ditunjukkan bahwa *graph* dengan tepi berbobot dapat digunakan untuk meminimalkan fungsi energi dengan hukuman interaksi linier dalam kasus multi-label. Konstruksi *graph* ini telah diperluas untuk menangani klik cembung dan interaksi metrik. Metode yang disebut algoritma ekspans- α menemukan solusi perkiraan dengan menjalankan algoritma *min-cut/max-flow* pada *graph* yang sesuai. Pendekatan ini dapat menangani berbagai macam klik, termasuk yang disukai dalam aplikasi praktis. Studi terbaru telah mengeksplorasi sifat teoritis konstruksi *graph* yang digunakan dalam penglihatan. Sebuah studi mengidentifikasi kondisi yang diperlukan dan cukup untuk fungsi energi yang dapat diminimalkan menggunakan *graph cut*. Namun, penelitian tersebut hanya berlaku untuk fungsi energi dengan variabel biner dan klik ganda atau tripel. Potensi penuh teknik *graph cut* dalam kasus multi-label belum sepenuhnya dipahami.

Sifat-sifat segmen yang dibuat dengan metode *graph cut* diperiksa dalam sebuah penelitian yang disebutkan [3]. Penelitian ini berfokus pada metrik potongan yang diterapkan pada kisi *graph* dan menunjukkan bahwa topologi diskrit dari *graph cut* dapat meniru ruang metrik Riemannian secara kontinu. Penelitian dilakukan yang ada telah menciptakan hubungan antara dua pendekatan umum yang digunakan untuk meminimalkan energi: metode *graph cut* kombinatorial dan metode geometris yang mengandalkan *set level*.

2.5.2 Latar Belakang Pada Graf

Graf berbobot dan berarah, dilambangkan dengan $G = \langle V, \varepsilon \rangle$ terdiri dari kumpulan *node* V dan sisi berarah E yang menghubungkannya. *Node* ini biasanya

mewakili fitur, seperti piksel atau voxel. Graf juga menyertakan beberapa *node* khusus yang dikenal sebagai terminal, yang sesuai dengan kemungkinan label yang dapat diberikan ke fitur, khususnya dalam konteks penglihatan. Pembahasan ini akan difokuskan pada graf yang hanya memiliki dua terminal.



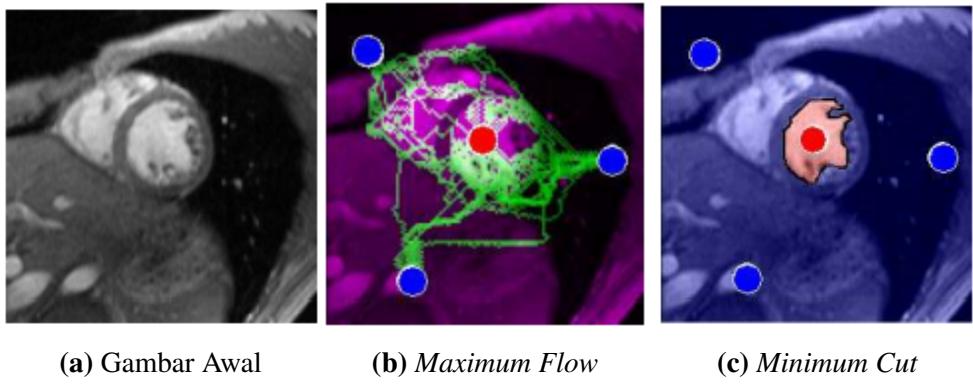
Gambar 2.7: Contoh graf berkapasitas terarah. (Boykov et al., 2004)

Terminal dalam *graph* dikenal sebagai *source* "s", dan *sink*, "t." Contoh sederhana dari *graph* dua terminal ditunjukkan pada Gambar 2.7(a), yang dapat digunakan untuk meminimalkan energi Potts pada gambar 3×3 dengan dua label. Sebagian besar metode minimisasi energi dalam penglihatan didasarkan pada *graph* grid 2D atau 3D biasa seperti yang ada pada Gambar 2.7(a) karena simpul *graph* biasanya mewakili piksel atau voxel gambar biasa. Setiap sisi dalam *graph* memiliki bobot atau *cost*, dan *cost* sisi berarah (p, q) mungkin berbeda dari *cost* sisi sebaliknya (q, p) . Penting untuk dapat menetapkan bobot tepi yang berbeda untuk (p, q) dan (q, p) di banyak aplikasi berbasis *graph* dalam penglihatan. *Graph* biasanya terdiri dari dua jenis sisi: *n-link* dan *t-link*. *N-link* menghubungkan piksel atau voxel yang bertetangga dan merepresentasikan sistem ketetanggaan dalam gambar. *cost n-link* sesuai dengan penalti untuk diskontinuitas antara piksel, yang berasal dari istilah interaksi piksel $V_{p,q}$ dalam energi (2.12). *T-link* menghubungkan piksel dengan terminal (label), dan *cost t-link* yang menghubungkan piksel dan terminal sesuai dengan penalti untuk menetapkan label yang sesuai ke piksel, yang berasal dari istilah data D_p dalam energi (2.12).

2.5.2.1 Permasalahan pada *Min-Cut* dan *Max-Flow*

Sebuah *Cut* pada graf dengan dua terminal, dinotasikan sebagai s/t , adalah pemisahan *node* dalam graf menjadi dua himpunan bagian yang terpisah dan tidak

tumpang tindih, S dan T . *Source* (s), termasuk dalam himpunan bagian S , dan *sink* (t), termasuk dalam subset T . Pemotongan s/t disebut sebagai *cuts*. Contoh potongan ditunjukkan pada gambar 2.7 (b).



Gambar 2.8: Contoh (*graph cut*)/*flow* dalam konteks segmentasi gambar. (a) menunjukkan *maximum flow* dari s ke t . Faktanya, ini menjenuhkan tepi *graph* yang sesuai dengan batas *minimum cut* di (b). (Boykov et al., 2004)

Dalam optimasi kombinatorial, *cost cut* $C = \mathcal{S}, \mathcal{T}$ didefinisikan sebagai jumlah *cost* dari tepi batas (p, q) di mana $p \in S$ dan $q \in T$. Perhatikan bahwa *cost cut* adalah "diarahkan" karena menjumlahkan bobot dari sisi-sisi yang diarahkan secara khusus dari \mathcal{S} ke \mathcal{T} . Masalah *minimum cut* pada graf adalah menemukan *cut* yang memiliki *cost* minimum di antara semua *cut*.

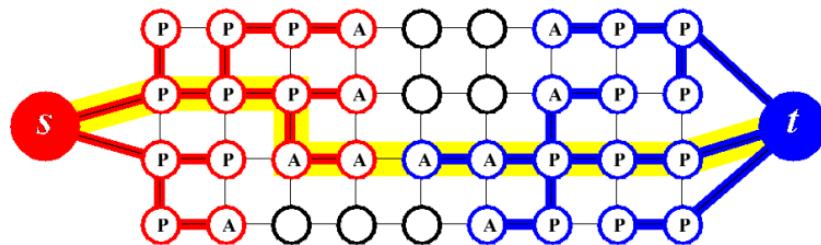
Salah satu konsep dasar dalam optimisasi kombinatorial adalah bahwa masalah mencari *minimum cut* s/t dapat diselesaikan dengan menentukan *maximum flow* dari *source* s ke *sink* t . Secara sederhana, *maximum flow* mewakili "jumlah air" maksimum yang dapat diangkut dari sumber ke sumur dengan mempertimbangkan tepi graf sebagai pipa yang diarahkan dengan kapasitas yang sama dengan bobot tepinya. Menurut teorema Ford dan Fulkerson, *maximum flow* dari s ke t mengisi sekelompok tepi di graf yang memisahkan simpul-simpul menjadi dua bagian yang tidak beririsan, S, T , yang sesuai dengan *minimum cut*. Oleh karena itu, masalah *minimum cut* dan *maximum flow* adalah setara, dan nilai *maximum flow* sama dengan *cost minimum cut*. Hubungan "dualitas" antara masalah *maximum flow* dan *minimum cut* diilustrasikan pada gambar 2.8 dalam konteks segmentasi gambar, di mana *maximum flow* yang ditampilkan pada gambar 2.8(a) mengisi tepi-tepi pada batas *minimum cut* pada gambar 2.8(b).

Konsep *min-cut* atau *max-flow* pada graf dapat digunakan untuk

meminimalkan energi dalam pelabelan gambar. Jika kita memiliki gambar 3×3 dan menerapkan *cut s/t* padanya, *node* akan dibagi menjadi kelompok terpisah, masing-masing hanya berisi satu terminal. Pembagian ini mewakili penugasan piksel ke label. Dengan memberikan bobot tepat pada tepian berdasarkan parameter energi, kita dapat mencapai energi minimum dengan mencari pemotongan *cost* minimum yang sesuai dengan pelabelan energi minimum.

2.5.3 Algoritma *Mincut/Max-Flow* Terbaru

Untuk meningkatkan performa empiris teknik augmenting path standar pada graf dalam *computer vision*, Boykov et al., 2004 mengembangkan algoritma baru. Biasanya, metode *augmenting path* memulai pencarian lebar baru untuk jalur $s \rightarrow t$ segera setelah semua jalur dari panjang tertentu habis. Namun, membangun pohon pencarian lebar untuk graf dalam *computer vision* melibatkan pemindaian sebagian besar piksel gambar, sehingga menjadi operasi yang mahal jika dilakukan terlalu sering. Eksperimen dengan data nyata dalam *computer vision* mengonfirmasi bahwa membangun kembali pohon pencarian menghasilkan kinerja yang buruk dari teknik *augmenting path* standar. Untuk mengatasi masalah ini, ibid. mengembangkan beberapa ide yang meningkatkan performa empiris teknik *augmenting path* pada graf dalam *computer vision*.



Gambar 2.9: Contoh pencarian pohon S (*node* merah) dan T (*node* biru) (Boykov et al., 2004)

Algoritma min-cut/max-flow baru yang disajikan di sini termasuk dalam kelompok algoritma berdasarkan *augmenting path*. Seperti algoritma Dinic, algoritma yang dibuat oleh ibid. juga membuat pohon pencarian untuk mendeteksi jalur pembesaran. Namun, membuat dua pohon pencarian, satu dari *source* dan satu lagi dari *sink*. Menggunakan kembali pohon-pohon ini dan tidak membuatnya dari awal setiap kali. Salah satu kelemahan pendekatan algoritma ini adalah jalur

pembesaran yang ditemukan tidak selalu merupakan jalur pembesaran terpendek, sehingga kompleksitas waktu untuk menemukan jalur pembesaran terpendek tidak lagi valid. Jumlah maksimum penambahan yang dapat dilakukan oleh algoritma, dibatasi oleh biaya *minimum cut* $|C|$, sehingga kompleksitas waktu terburuknya adalah $O(mn^2|C|)$.

2.5.3.1 Penjelasan Algoritma

Boykov et al., 2004 mempertahankan dua pohon pencarian yang tidak tumpang tindih S dan T dengan akar pada *source* s dan *sink* t , secara berurutan. Pada pohon S semua sisi dari setiap *node* induk ke anaknya tidak jenuh, sedangkan pada pohon T sisi dari anak ke induknya tidak jenuh. *node* yang tidak ada di S atau T disebut "bebas".

$$S \subset V, s \in S, T \subset V, t \in T, S \cap T = 0 \quad (2.13)$$

Node dalam pohon pencarian S dan T dapat berupa "aktif" atau "pasif". *node* aktif mewakili batas luar di setiap pohon sedangkan *node* pasif adalah internal. Intinya adalah *node* aktif memungkinkan pohon untuk "tumbuh" dengan memperoleh anak baru (sepanjang tepi yang tidak jenuh) dari sekumpulan *node* bebas. *node* pasif tidak dapat tumbuh karena sepenuhnya diblokir oleh *node* lain dari pohon yang sama. Penting juga bahwa *node* aktif dapat bersentuhan dengan *node* dari pohon lain. Jalur augmentasi ditemukan segera setelah *node* aktif di salah satu pohon mendeteksi *node* tetangga yang dimiliki oleh pohon lainnya.

Algoritma secara iteratif mengulangi tiga tahap berikut:

- Tahap pertumbuhan atau (*growth*): cari pohon S dan T tumbuh hingga bersinggungan memberikan jalur $s \rightarrow t$
- Tahap "augmentasi atau (*augmentation*)": jalur yang ditemukan ditambah, pohon pencarian dipecah menjadi hutan
- Tahap adopsi atau (*adoption*): pohon S dan T dipulihkan.

Pada tahap pertumbuhan (*growth*), pohon pencarian berkembang dengan *node* aktif mengeksplorasi tepi yang belum terjenuh dan mendapatkan anak baru dari himpunan *node* bebas. Anak-anak baru ini menjadi bagian aktif dari pohon pencarian yang sesuai. Ketika semua tetangga dari *node* aktif telah dijelajahi, status

node aktif berubah menjadi pasif. Tahap pertumbuhan berakhir ketika *node* aktif menemukan tetangga yang sudah ada dalam pohon pencarian lawan, menunjukkan penemuan jalur dari *source* ke *sink*.

Pada tahap augmentasi, jalur yang ditemukan pada tahap pertumbuhan ditingkatkan dengan mengalirkan aliran maksimum melalui jalur tersebut. Ini bisa membuat beberapa tepi menjadi jenuh, mengakibatkan beberapa *node* dalam himpunan S dan T menjadi "orphans" karena tautan induk mereka tidak lagi valid. Tahap augmentasi juga bisa memisahkan pohon S dan T menjadi hutan.

Pada tahap adopsi, tujuannya adalah mengembalikan struktur satu-pohon ke dalam himpunan S dan T dengan akar di *source* dan *sink*. Ini dilakukan dengan mencari induk baru yang valid untuk setiap "orphans" dalam himpunan yang sama, terhubung melalui tepi yang tidak jenuh. Jika tidak ditemukan induk yang memenuhi syarat, "orphans" dihapus dari S atau T , menjadi *node* bebas, dan semua anak yang sebelumnya menjadi "orphans".

Tahap adopsi (*adoption*) berlanjut sampai semua "orphans" telah menemukan induk baru yang valid atau telah menjadi *node* bebas. Ini mengakibatkan pengurangan ukuran set S dan T . Setelah tahap adopsi selesai, algoritma kembali ke tahap pertumbuhan. Algoritma berakhir saat tidak ada *node* aktif yang tersisa dan pohon pencarian S dan T terisolasi oleh tepi yang jenuh, menunjukkan pencapaian aliran maksimum. Potongan minimum dapat ditentukan dengan menetapkan S dan T pada pohon yang sesuai.

2.5.3.2 Implementasi Lebih Lanjut

Asumsikan bahwa kita memiliki graf berarah $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$. Untuk setiap algoritma *augmenting path*, kita akan mempertahankan aliran f dan graf residual G_f (lihat Bagian 2.1.2.2). Menyimpan daftar semua *node* aktif, A , dan semua *orphans*, O . Struktur umum algoritma adalah:

```

initialize: (S = {s}, T = {t}, A = {s, t}, O = ∅)
while true
    grow S or T to find an augmenting path P from s to t
    if P = ∅ terminate
    augment on P
    adopt orphans
end while

```

Rincian tahap pertumbuhan, augmentasi, dan adopsi dijelaskan di bawah ini.

Lebih mudah untuk menyimpan muatan dari pohon pencarian S dan T melalui *flag* $TREE(p)$ yang menunjukkan afiliasi dari setiap *node* p sehingga

$$TREE(p) = \begin{cases} S & \text{if } p \in S \\ T & \text{if } p \in T \\ \emptyset & \text{if } p \text{ is free} \end{cases} \quad (2.14)$$

Jika *node* p milik salah satu pohon pencarian maka informasi tentang induknya akan disimpan sebagai $PARENT(p)$. Akar pohon pencarian (*source* dan *sink*), *orphans*, dan semua *node* bebas tidak memiliki orang tua, t.e. $PARENT(p) = \emptyset$. Notasi $tree_cap(p \rightarrow q)$ untuk mendeskripsikan kapasitas residual dari salah satu sisi (p, q) jika $TREE(p) = S$ atau sisi (q, p) jika $TREE(p) = T$. Sisi-sisi ini harus tidak jenuh agar *node* p menjadi orangtua yang valid dari anaknya q tergantung pada pohon pencarian.

2.5.3.2.1 Fase Growth

Pada tahap ini node aktif memperoleh anak baru dari satu set node bebas.

```

while A $\neq$ $\emptyset$
    pick an active node $p \in A$ 
    for every neighbor q such that $tree\text{\textbackslash}textunderscore cap(p \ 
    rightarrow q) > 0
        if $TREE(q) = \emptyset$ then add q to search tree
        as an active node:
            $TREE(q) \neq TREE(p)$, $PARENT(q) \neq p$, $A \neq A \cup \{q\}$
            if $TREE(q) \neq \emptyset$ and $TREE(q) \neq TREE(p)$
                return $P = PATH_{\{s \rightarrow t\}}
        end for
        remove p from A
    end while
    return P = $\emptyset$
```

2.5.3.2.2 Fase Augmentation

Input untuk tahap ini adalah jalur P dari s ke t . Perhatikan bahwa himpunan anak yatim kosong pada awal tahap, tetapi mungkin ada beberapa *orphans* pada akhirnya karena setidaknya satu sisi di P menjadi jenuh.

```

find the bottleneck capacity $\Delta$ on P
update the residual graph by pushing flow $\Delta$ through P
```

```

for each \emph{edge} $(p, q)$ in $P$ that becomes saturated
    if $\text{TREE}(p) = \text{TREE}(q) = S$
        then set $\text{PARENT}(q) := \emptyset$ and $O := O \cup \{q\}$
    if $\text{TREE}(p) = \text{TREE}(q) = T$
        then set $\text{PARENT}(p) := \emptyset$ and $O := O \cup \{p\}$
end for

```

2.5.3.2.3 Fase Adoption

Selama tahap ini semua *node orphans* di O diproses sampai O menjadi kosong. Setiap *node* p sedang diproses mencoba menemukan induk baru yang valid di dalam pohon pencarian yang sama; jika berhasil, p tetap berada di pohon tetapi dengan induk baru, jika tidak, ia menjadi *node* bebas dan semua anaknya ditambahkan ke O .

```

while $O \neq \emptyset
    pick an orphan node $p \in O$ and remove it from $O
    process $p
end while

```

Operasi "proses p " terdiri dari langkah-langkah berikut. Pertama kita mencoba mencari induk valid baru untuk p di antara tetangganya. Induk q yang valid harus memenuhi: $\text{TREE}(q) = \text{TREE}(p)$, $\text{tree_cap}(q \rightarrow p) > 0$, dan origin dari q harus berupa *source* atau *sink*. Perhatikan bahwa kondisi terakhir diperlukan karena selama tahap adopsi beberapa *node* dalam pohon pencarian S atau T mungkin berasal dari *orphans*. Jika *node* p menemukan induk baru yang valid q maka kita atur $\text{PARENT}(p) = q$. Dalam hal ini p tetap berada di pohon pencarinya dan status aktif (atau pasif) p tetap tidak berubah.

Jika p tidak menemukan induk yang valid maka p menjadi *node* bebas dan operasi berikut dilakukan:

- Memindai semua tetangga q dari p sehingga $\text{TREE}(q) = \text{TREE}(p)$:
 - jika $\text{tree_cap}(q \rightarrow p) > 0$ tambahkan q kepada set A yang aktif
 - jika $\text{PARENT}(q) = p$ tambahkan q kepada set *orphans* O dan set $\text{PARENT}(q) := \emptyset$
- $\text{TREE}(p) := \emptyset, A := A - \{p\}$

Perhatikan bahwa ketika p menjadi bebas, semua tetangganya yang terhubung melalui tepi yang tidak jenuh harus menjadi aktif. Mungkin saja beberapa tetangga q tidak memenuhi syarat sebagai orang tua yang valid selama tahap adopsi karena tidak berasal dari sumber atau sink. Namun, *node* ini bisa menjadi induk yang valid setelah tahap adopsi selesai. Pada titik ini q harus berstatus aktif karena terletak di sebelah *node* bebas p .

2.6 Segmentasi Citra dengan Algoritma *GrabCut*

Penelitian oleh Rother et al., 2004 mengenai segmentasi gambar memanfaatkan citra dan menyatakan bahwa alat segmentasi gambar klasik menggunakan informasi tekstur (misalnya, warna) seperti *Magic Wand*, atau informasi tepi (misalnya *Intelligent Scissors*). Pendekatan yang digunakan oleh ibid. berbasis pengoptimalan dengan *GraphCut* dan berhasil menggabungkan kedua jenis informasi tersebut.

Tujuan algoritma ini adalah memisahkan objek dari latar belakang dalam lingkungan kompleks tanpa memerlukan banyak masukan pengguna. Hasilnya berupa alpha-matte yang menunjukkan proporsi *foreground* dan *background* pada setiap piksel. Algoritma ini bertujuan untuk memberikan segmentasi objek yang akurat, nilai alpha yang memperhitungkan blur, piksel campuran, transparansi, dan warna *foreground* yang bersih tanpa terpengaruh oleh warna latar belakang. Jumlah interaksi pengguna dapat bervariasi, mulai dari mengedit piksel hingga menandai lokasi *foreground* atau *background*.

2.6.1 Sistem yang Diusulkan : *GrabCut*

Alat pemotongan gambar yang baik harus menghasilkan nilai alpha yang halus di seluruh wilayah inferensi trimap, tanpa batasan keras 0 atau 1, untuk menangani masalah seperti asap, rambut, dan pohon. Teknik pemotongan gambar umumnya efektif saat ada pemisahan yang jelas antara distribusi warna *foreground* dan *background*.

Langkah pertama adalah segmentasi "keras" menggunakan *Graph Cut* iteratif. Kemudian dilakukan matting batas di sekitar segmentasi keras, di mana nilai alpha dihitung dalam jalur sempit. Namun, *GrabCut* tidak menangani transparansi penuh selain di perbatasan, yang dapat dicapai dengan matting brush dalam area yang cukup bebas.

Pendekatan Boykov memperkenalkan peningkatan pada mekanisme *Graph Cut*. Ini mencakup "estimasi iteratif" dan "pelabelan tidak lengkap" yang mengurangi interaksi pengguna. GrabCut memberikan kemudahan interaksi bagi pengguna dengan hanya menarik persegi panjang di sekitar objek. Mereka juga mengembangkan mekanisme baru untuk menghitung alpha dalam *border matting* guna mengurangi artefak.

2.6.2 Segmentasi gambar dengan *graph cut*

Segmentasi citra monokrom, diberi trimap awal T . Citra merupakan *array* $\mathbf{z} = (z_1, \dots, z_n, \dots, z_N)$ dengan nilai keabuan, diindeks dengan indeks (tunggal) n . Segmentasi gambar diekspresikan sebagai susunan nilai opasitas $\alpha = (\alpha_1, \dots, \alpha_N)$ pada setiap piksel. Umumnya $0 \leq \alpha_n \leq 1$, tetapi untuk segmentasi keras $\alpha_n \in 0, 1$, dengan 0 untuk latar belakang dan 1 untuk latar depan. Parameter θ menggambarkan distribusi tingkat keabuan latar depan dan latar belakang, dan terdiri dari histogram nilai keabuan:

$$\theta = \{h(z; \alpha), \alpha = 0, 1\} \quad (2.15)$$

dimana,

θ = Distribusi tingkat keabuan dari gambar citra

α = Nilai alpha tiap piksel 1 untuk *foreground* dan 0 untuk *background*

h = Histogram tingkat keabuan

satu untuk latar belakang dan satu untuk latar depan. Histogram dirakit langsung dari piksel berlabel dari masing-masing wilayah trimap T_B, T_F . (Histogram dinormalisasi untuk dijumlahkan menjadi 1 pada rentang tingkat abu-abu: $\int_z h(z; \alpha) = 1$.) Tugas segmentasi adalah menyimpulkan variabel opasitas yang tidak diketahui α dari data gambar yang diberikan z dan model θ .

2.6.2.1 Segmentasi dengan *energy minimisation*

Fungsi energi E didefinisikan pada persamaan 2.12 yang dijelaskan oleh Boykov et al., 2004 sehingga minimumnya harus sesuai dengan segmentasi yang baik, dalam arti bahwa ia dipandu baik oleh histogram tingkat abu-abu latar depan dan latar belakang yang di amati dan bahwa opasitasnya "koheren", yang

mencerminkan kecenderungan soliditas objek. Rother et al., 2004 menuliskan kembali energi "Gibbs" dalam bentuk:

$$E(\underline{\alpha}, \underline{\theta}, \mathbf{z}) = U(\underline{\alpha}, \underline{\theta}, \mathbf{z}) + V(\underline{\alpha}, \mathbf{z}) \quad (2.16)$$

dimana,

E = Cost Function atau Rumus Energi

$U(\underline{\alpha}, \underline{\theta}, \mathbf{z})$ = Data penalti, seperti pada rumus 2.12

$V(\underline{\alpha}, \mathbf{z})$ = Interaksi potensial yaitu diskontinuitas antar piksel

\mathbf{z} = Nilai warna tiap piksel pada citra digital

Istilah data U mengevaluasi kecocokan distribusi opasitas $\underline{\alpha}$ dengan data \mathbf{z} , mengingat model histogram $\underline{\theta}$, dan didefinisikan sebagai:

$$U(\underline{\alpha}, \underline{\theta}, \mathbf{z}) = \sum_n -\log h(z_n; \alpha_n) \quad (2.17)$$

dimana,

E = Cost Function atau Rumus Energi

$U(\underline{\alpha}, \underline{\theta}, \mathbf{z})$ = Data penalti, seperti pada rumus 2.12

Istilah *smoothness* dapat dituliskan sebagai :

$$V(\underline{\alpha}, \mathbf{z}) = \gamma \sum_{(m,n) \in \mathbf{C}} dis(m, n)^{-1} [\alpha_n \neq \alpha_m] exp - \beta(z_m - z_n)^2 \quad (2.18)$$

dimana,

E = Cost Function atau Rumus Energi

$V(\underline{\alpha}, \mathbf{z})$ = Interaksi potensial yaitu diskontinuitas antar piksel

γ = niai gamma

m = Suatu piksel

n = Tetangga dari piksel m

$\beta(z_m - z_n)^2$ = Beta smoothness

di mana $[\phi]$ menunjukkan fungsi indikator yang mengambil nilai 0,1 untuk predikat ϕ , \mathbf{C} adalah himpunan pasangan piksel tetangga, dan $dis(\cdot)$ adalah jarak Euclidean piksel tetangga. Energi ini mendorong koherensi di wilayah dengan

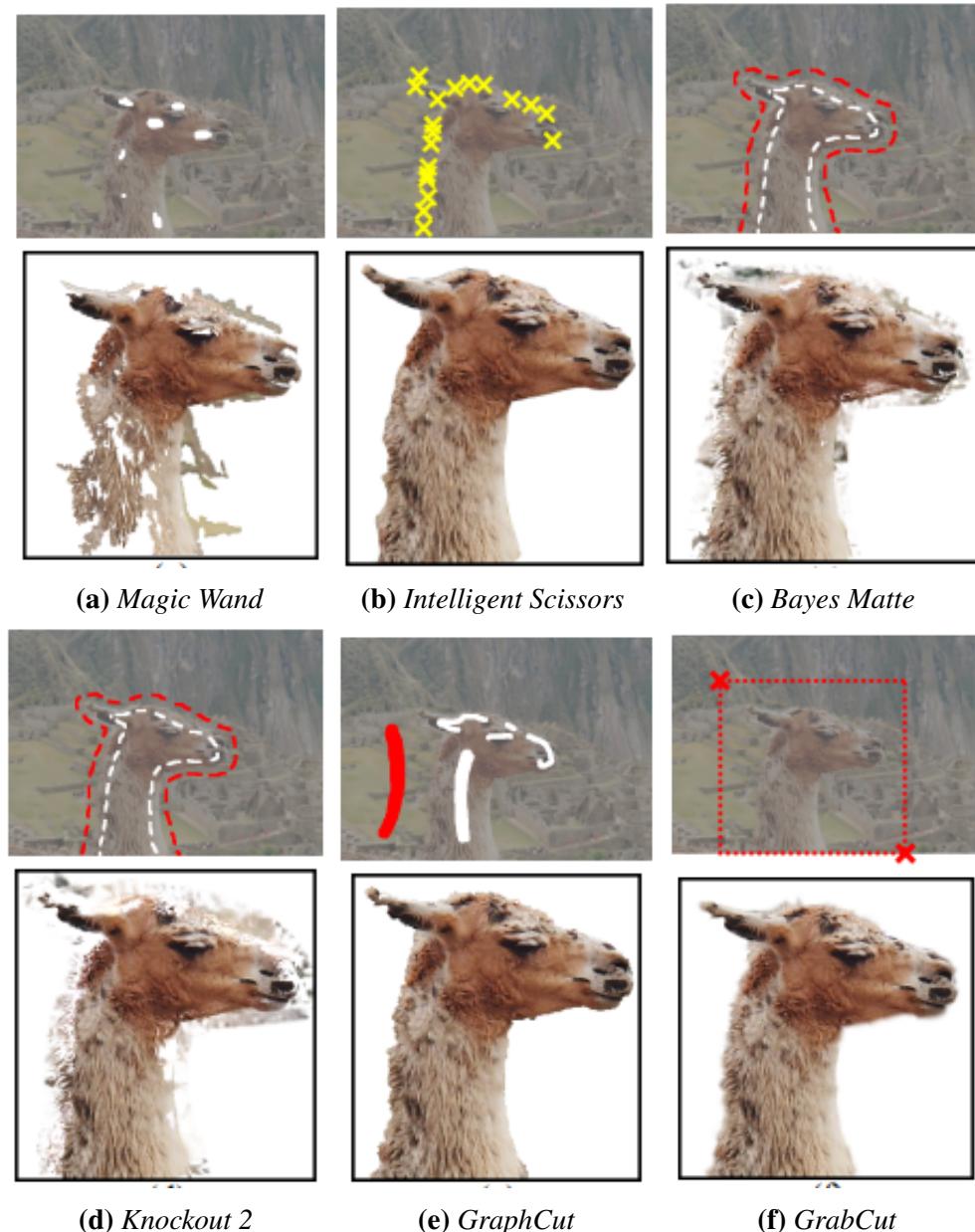
tingkat abu-abu yang serupa. Dalam praktiknya, hasil yang baik diperoleh dengan mendefinisikan piksel sebagai tetangga jika berdekatan baik secara horizontal/vertikal maupun diagonal (konektivitas 8 arah). Ketika konstanta $\beta = 0$, istilah kehalusan hanyalah Ising sebelumnya yang terkenal, mendorong kehalusan di mana-mana, hingga tingkat yang ditentukan oleh konstanta γ . Namun telah ditunjukkan bahwa jauh lebih efektif untuk mengatur $\beta > 0$ karena hal ini mengurangi kecenderungan kehalusan di daerah dengan kontras tinggi. Konstanta β dipilih menjadi :

$$\beta = \left(2 \langle (z_m - z_n)^2 \rangle \right)^{-1} \quad (2.19)$$

di mana $\langle (\cdot) \rangle$ menunjukkan harapan atas sampel gambar. Pilihan β ini memastikan bahwa suku eksponensial dalam 2.18 beralih dengan tepat antara kontras tinggi dan rendah. Konstanta γ diperoleh sebesar 50 dengan mengoptimalkan kinerja terhadap kebenaran dasar melalui satu set pelatihan yang terdiri dari 15 gambar. Ini terbukti menjadi pengaturan serbaguna untuk berbagai macam gambar. Sekarang setelah model energi didefinisikan sepenuhnya, segmentasi dapat diperkirakan sebagai minimum global:

$$\hat{\underline{\alpha}} = \arg \min_{\underline{\alpha}} \mathbf{E}(\underline{\alpha}, \theta). \quad (2.20)$$

Algoritma pemotongan minimum standar digunakan untuk minimisasi. Ini menjadi dasar bagi segmentasi keras, yang kemudian mengembangkan tiga perubahan untuk algoritma segmentasi dalam GrabCut. Pertama, model gambar monokrom digantikan oleh Model Campuran Gaussian (GMM) untuk warna. Kedua, estimasi *one-shot minimum cut* diganti dengan prosedur iteratif lebih kuat yang bergantian antara estimasi dan pembelajaran parameter. Ketiga, permintaan pengguna interaktif dilemahkan, memungkinkan pelabelan yang tidak lengkap - pengguna hanya menentukan T_B untuk trimap, dengan cara menempatkan bentuk di sekitar objek.



Gambar 2.10: Perbandingan beberapa *matting tools* dan segmentasi. (Rother et al., 2004)

2.6.3 Segmentasi gambar dengan algoritma *GrabCut*

2.6.3.1 Model Pendataan Warna

Gambar yang ada saat ini terdiri dari piksel z_n dalam ruang warna RGB. Karena tidak praktis untuk membuat histogram ruang warna yang memadai, untuk

itu maka digunakan GMM. Setiap nilai GMM yaitu satu untuk latar belakang dan satu untuk latar depan, dianggap sebagai *Gaussian Mixture* penuh dengan komponen K (biasanya $K = 5$). Untuk menangani GMM dengan baik, dalam kerangka optimisasi, vektor tambahan $\mathbf{k} = \{k_1, \dots, k_n, \dots, k_N\}$ diperkenalkan, dengan $k_n \in \{1, \dots, K\}$, menetapkan, untuk setiap piksel, komponen GMM unik, satu komponen baik dari latar belakang atau model latar depan, menurut $\alpha_n = 0$ atau 1^1 .

Energi Gibbs untuk segmentasi sekarang menjadi

$$E(\alpha, k, \theta, z) = U(\alpha, k, \theta, z) + V(\alpha, z) \quad (2.21)$$

bergantung juga pada variabel komponen GMM k . Istilah data U sekarang didefinisikan, dengan mempertimbangkan model GMM warna, sebagai

$$U(\alpha, k, \theta, z) = \sum_n D(\alpha_n, k_n, \theta, z_n), \quad (2.22)$$

di mana

$$D(\alpha_n, k_n, \theta, z_n) = -\log p(z_n | \alpha_n, k_n, \theta) - \log \pi(\alpha_n, k_n) \quad (2.23)$$

yang mana $p(\cdot)$ adalah distribusi probabilitas Gaussian, dan $\pi(\cdot)$ adalah koefisien bobot campuran, sehingga :

$$\begin{aligned} D(\alpha_n, k_n, \theta, z_n) &= -\log \pi(\alpha_n, k_n) + \frac{1}{2} \log \det \Sigma(\alpha_n, k_n) \\ &\quad + \frac{1}{2} [z_n - \mu(\alpha_n, k_n)]^T \Sigma(\alpha_n, k_n)^{-1} [z_n - \mu(\alpha_n, k_n)] \end{aligned} \quad (2.24)$$

Oleh karena itu, parameter model sekarang adalah

$$\theta = \{\pi(\alpha, k), \mu(\alpha, k), \Sigma(\alpha, k), \alpha = 0, 1, k = 1 \dots K\}, \quad (2.25)$$

yaitu bobot π , berarti μ dan kovarians Σ dari komponen $\in \mathcal{K}$ Gaussian untuk distribusi latar belakang dan latar depan. Istilah kelancaran \mathcal{V} pada dasarnya tidak berubah dari kasus monokrom kecuali bahwa istilah kontras dihitung menggunakan jarak *Euclidean* dalam ruang warna:

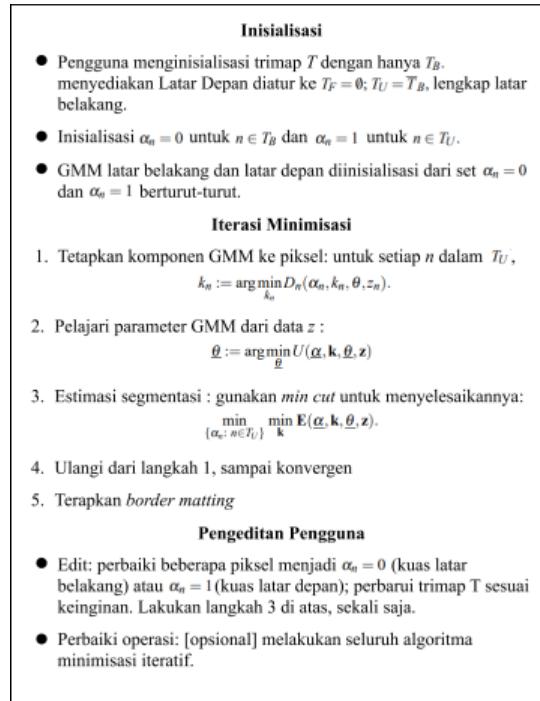
$$V(\alpha, z) = \gamma \sum_{(m,n) \in C} [\alpha_n \neq \alpha_m] \exp -\beta \|z_m - z_n\|^2 \quad (2.26)$$

2.6.3.2 Segmentasi berdasarkan Iterasi *Energy Minimization*

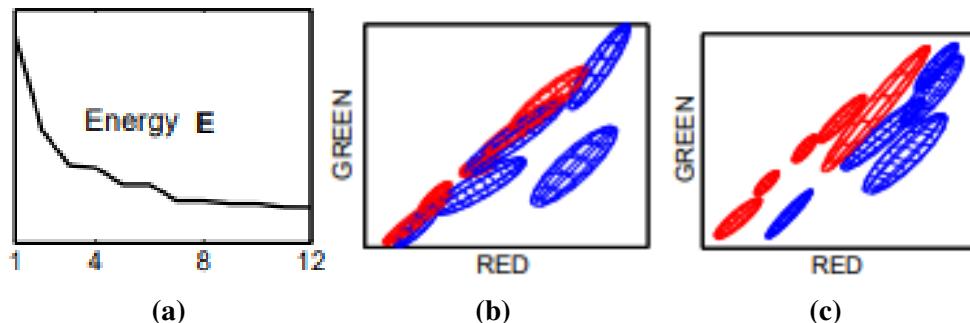
Skema minimisasi energi baru di *GrabCut* bekerja secara iteratif, menggantikan algoritma sekali pakai sebelumnya (Boykov). Ini memiliki keuntungan yang memungkinkan penyempurnaan otomatis dari opasitas α , karena piksel berlabel baru dari wilayah TU pada trimap awal digunakan untuk menyempurnakan parameter GMM warna θ . Elemen utama dari sistem *GrabCut* diberikan dalam gambar 2.11. Langkah 1 sangat mudah, dilakukan dengan pencacahan sederhana nilai k_n untuk setiap piksel n . Langkah 2 diimplementasikan sebagai satu set prosedur estimasi parameter Gaussian, sebagai berikut. Untuk komponen GMM k yang diberikan, katakanlah, model latar depan, himpunan bagian dari piksel $F(k) = z_n : k_n = k$ dan $\alpha_n = 1$ ditentukan. Rata-rata $\mu(\alpha, k)$ dan kovarians $\sum(\alpha, k)$ diperkirakan dengan cara standar sebagai rata-rata sampel dan kovarians nilai piksel dalam $F(k)$ dan bobotnya adalah $\pi(\alpha, k) = |F(k)| / \sum_k |F(k)|$, di mana $|S|$ menunjukkan ukuran himpunan S . Akhirnya langkah 3 adalah optimasi global, menggunakan *minimum cut*, persis seperti Boykov.

Struktur algoritma menjamin sifat konvergensi yang tepat. Hal ini karena setiap langkah 1 sampai 3 dari minimalisasi iteratif dapat ditunjukkan sebagai minimalisasi energi total E sehubungan dengan tiga set variabel k, θ, α pada gilirannya. Oleh karena itu E berkurang secara monoton, dan ini diilustrasikan dalam praktiknya dalam gambar 4. Dengan demikian algoritma dijamin konvergen setidaknya ke minimum lokal E . Sangat mudah untuk mendeteksi kapan E berhenti menurun secara signifikan, dan menghentikan iterasi secara otomatis.

Manfaat praktis dari minimisasi iteratif. Gambar. 2.10(e) dan 2.10(f) mengilustrasikan bagaimana kekuatan tambahan dari minimisasi iteratif dalam *GrabCut* dapat sangat mengurangi jumlah interaksi pengguna yang diperlukan untuk menyelesaikan tugas segmentasi, relatif terhadap pendekatan *one-shot graph cut*. Ini terlihat dalam dua cara. Pertama, tingkat pengeditan pengguna yang diperlukan, setelah inisialisasi dan pengoptimalan, dikurangi. Kedua, interaksi awal bisa lebih sederhana, misalnya dengan mengizinkan pelabelan yang tidak lengkap oleh pengguna.



Gambar 2.11: Segmentasi gambar di *GrabCut* (Rother et al., 2004)

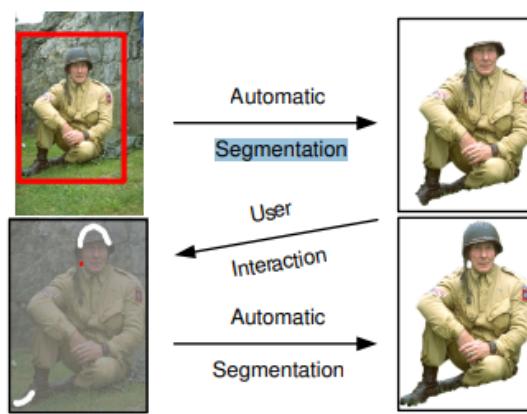


Gambar 2.12: Konvergensi minimalisasi iteratif untuk data gambar 2.10(f). (a) Energi E untuk contoh llama konvergen selama 12 iterasi dengan $K = 5$, komponen campuran digunakan untuk latar belakang (merah) dan latar depan (biru) (Rother et al., 2004).

2.6.3.3 Interaksi Pengguna dan Trimap yang Tidak Lengkap

Trim tidak lengkap. Algoritma minimisasi iteratif memungkinkan peningkatan keserbagunaan interaksi pengguna. Secara khusus, pelabelan yang tidak lengkap menjadi layak di mana, sebagai pengganti trimap T penuh, pengguna hanya perlu menentukan, katakanlah, wilayah latar belakang T_B , meninggalkan $T_F = \emptyset$.

Tidak ada pelabelan latar depan keras yang dilakukan sama sekali. Minimisasi iteratif (gambar 2.11) menangani ketidaklengkapan ini dengan mengizinkan label sementara pada beberapa piksel (di latar depan) yang kemudian dapat ditarik kembali; hanya label latar belakang T_B yang dianggap tegas dijamin tidak akan ditarik kembali nantinya. (Tentu saja skema pelengkap, dengan label tegas untuk latar depan saja, juga memungkinkan.) T_B awal ditentukan oleh pengguna sebagai strip piksel di sekitar bagian luar sudut persegi yang ditandai (ditandai dengan warna merah pada gambar 2.10(f))



Gambar 2.13: Pengeditan pengguna. Setelah interaksi pengguna awal dan segmentasi (baris atas), pengeditan pengguna lebih lanjut (gambar 2.11) diperlukan. (Rother et al., 2004)

Pengeditan pengguna lebih lanjut. Pelabelan pengguna awal yang tidak lengkap cukup sepuluh untuk memungkinkan seluruh segmentasi diselesaikan secara otomatis, tetapi tidak berarti selalu. Jika tidak, pengeditan pengguna lebih lanjut diperlukan, seperti yang ditunjukkan pada gambar 2.13. Ini mengambil bentuk menyikat piksel, membatasinya menjadi latar depan yang kokoh atau latar belakang yang kokoh; lalu minimisasi langkah 3. pada gambar 2.11 diterapkan. Perhatikan bahwa cukup menyikat, secara kasar, hanya sebagian dari area yang salah diberi label. Selain itu, operasi "perbaiki" opsional dari gambar 2.11 memperbarui model warna, mengikuti suntingan pengguna. Ini menopang efek operasi edit yang seringkali bermanfaat. Perhatikan bahwa untuk efisiensi aliran optimal, dihitung dengan *graph cut*, dapat digunakan kembali selama pengeditan pengguna.

BAB III

METODOLOGI PENELITIAN

3.1 Desain Segmentasi Luka dengan Metode *GrabCut*

Proses pembuatan segmentasi luka, penulis menggunakan metode *GrabCut*. Metode segmentasi luka berupa citra gambar dengan metode *GrabCut* berdasarkan algoritma yang telah dikembangkan oleh (Rother et al., 2004) adalah program untuk mensegmentasi citra gambar menjadi dua bagian yaitu *foreground* dan sisanya akan dianggap menjadi *background*.

Algoritma *GrabCut* secara umum terdapat beberapa tahapan yaitu tahap inisiasi kotak pembatas (*bounding box*), setelah itu tahap iterasi minimisasi, dan dilanjutkan dengan penyuntingan pengguna (lihat 2.11).

3.1.1 Inisiasi Kotak Pembatas (*Bounding Box*)

Penulis melakukan inisiasi *bounding box* pada daerah yang melingkupi objek dimana daerah yang berada didalam *bounding box* akan digunakan untuk membuat *Trimap Unknown* (T_U) dan daerah di luarnya akan dianggap sebagai *Trimap Background* atau (T_B). Setiap piksel-n yang termasuk (T_B) akan memiliki α_n bernilai 0, sedangkan piksel-n dari (T_U) akan bernilai 1 untuk α_n . Tujuan dari *bounding box* ialah untuk mempercepat waktu komputasi dari algoritma serta meningkatkan tingkat akurasi segmentasi.

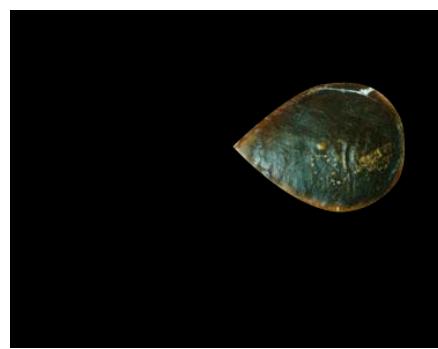


Gambar 3.1: (a) Data citra luka 2.png, (b) Inisiasi kotak pembatas

3.1.2 Implementasi Algoritma *GrabCut*

Setelah kotak tergambar, pada tahap ini terdapat beberapa algoritma yaitu inisiasi dan mempelajari *Gaussian Mixture Models* (GMM), dan segmentasi citra *GraphCut*. Inisiasi GMM digunakan untuk memperoleh nilai probabilistik dari tiap piksel yang ada didalam (T_U). GMM terdiri dari beberapa parameter yaitu K komponen, rata-rata μ , matriks kovarians Σ , dan probabilitas prior ω . Selanjutnya mempelajari parameter GMM yang berdasarkan probabilitas dari parameter GMM tiap piksel. Setelah mempelajari nilai dari parameter GMM tiap piksel.

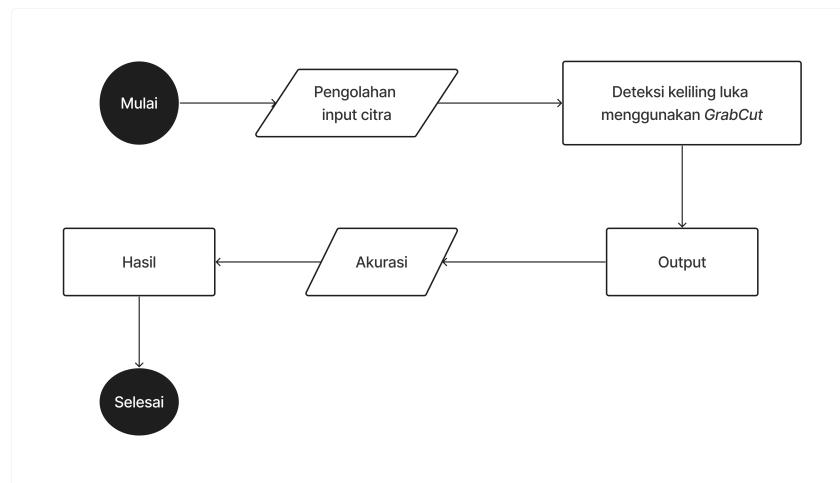
Setelah mempelajari parameter GMM dari tiap piksel, maka diperoleh nilai parameter untuk komponen yang ada pada setiap piksel, masuk ke tahap segmentasi citra luka dengan menggunakan algoritma *GraphCut*. Algoritma ini memvisualisasikan tiap piksel dari citra sebagai sebuah graf yang saling terhubung, satuan piksel pada gambar bisa disebut sebagai *node* pada graf yang mana tiap node akan memiliki komponen yang berasal dari nilai GMM tadi, pada graf yang sudah terbentuk maka akan ada dua terminal yaitu terminal *source* dan terminal *sink*, partisi graf (kumpulan *node*) yang memiliki hubungan probabilitas ke terminal *source* akan menjadi *foreground* dan partisi graf yang memiliki hubungan probabilitas ke terminal *sink* akan menjadi *background*. Algoritma ini memanfaatkan nilai dari GMM yang sudah dipelajari dan *term smoothness*, nilai GMM akan dipakai sebagai kemungkinan piksel yang termasuk *background* atau *foreground* sedangkan *term smoothness* dipakai untuk menghitung hubungan diskontinuitas antar piksel.



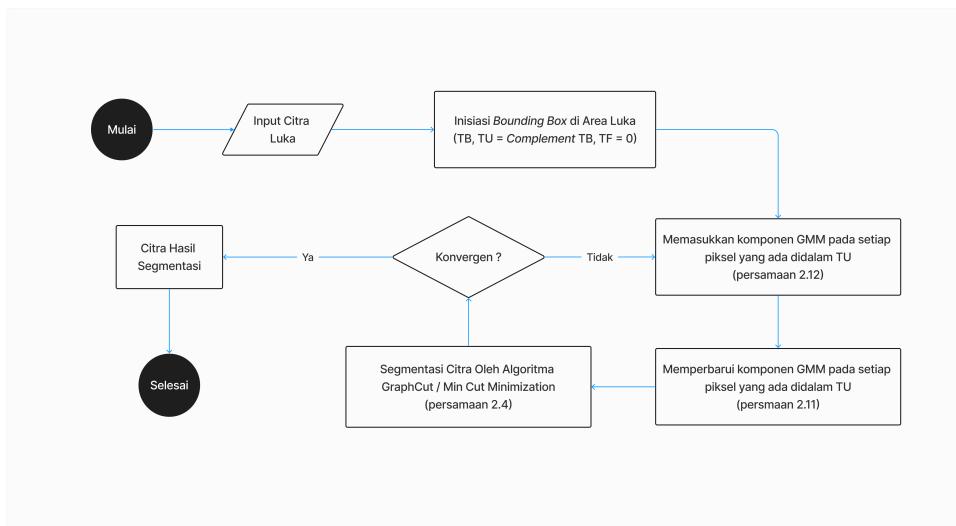
Gambar 3.2: Hasil segmentasi dengan *GrabCut*

3.2 Diagram Alir Segmentasi Luka dengan Metode *GrabCut*

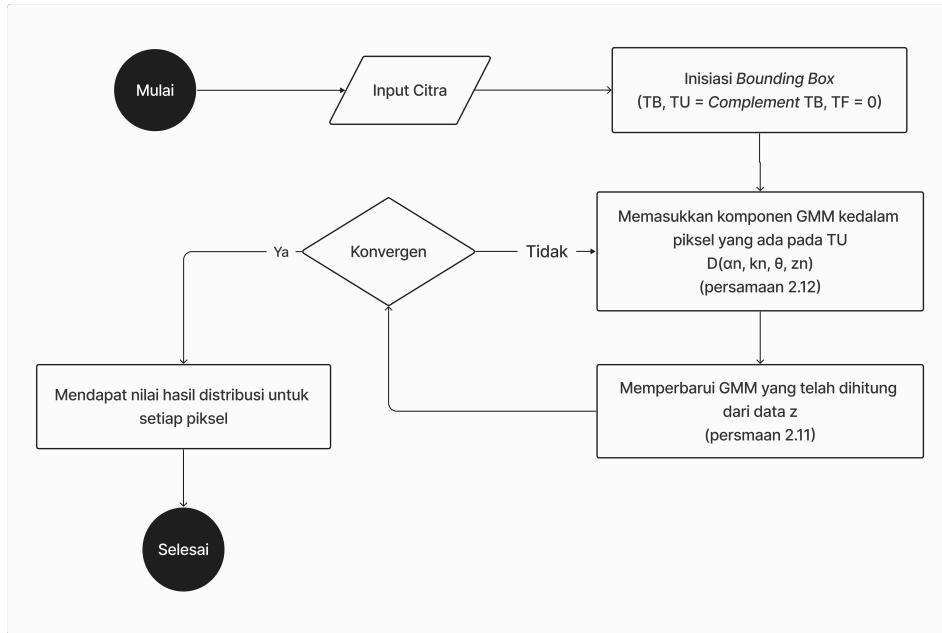
Berikut adalah diagram alir segmentasi luka dengan metode *GrabCut*:



Gambar 3.3: Diagram alir penelitian



Gambar 3.4: Diagram alir metode *GrabCut*



Gambar 3.5: Diagram alir tahap *Gaussian Mixture Models*

3.3 Struktur Data

3.3.1 Node

Node merupakan elemen dasar dalam pembentukan struktur data seperti *linked list*, pohon, dan graf. Sebuah *node* setidaknya terdiri dari dua komponen, yaitu data atau nilai dari *node* tersebut dan referensi atau tautan yang menghubungkan *node* tersebut dengan *node* lainnya. Penulis menggunakan struktur data *node* untuk merepresentasikan piksel dari citra luka, setiap piksel dari gambar menyimpan beberapa informasi, di antaranya komponen warna RGB (*Red, Green, Blue*), komponen GMM dan nilai label piksel.



Gambar 3.6: Representasi *node* terhadap piksel pada gambar

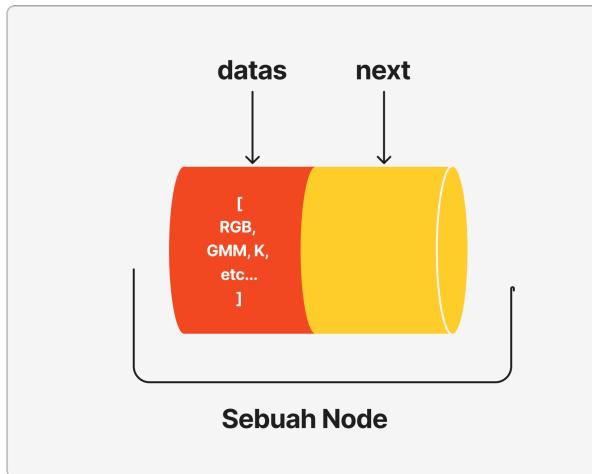
Tiap informasi yang ada pada *node* akan dipergunakan dalam algoritma *mincut* dengan struktur data sebagai berikut :

```
// Struktur Data Node dari piksel
class Node {
public:
    bool isForeground;
    double foregroundProbability;
    double backgroundProbability;
    vector<GaussianComponent> gmmComponents;
}
```

Listing 3.1: Struktur data *node*

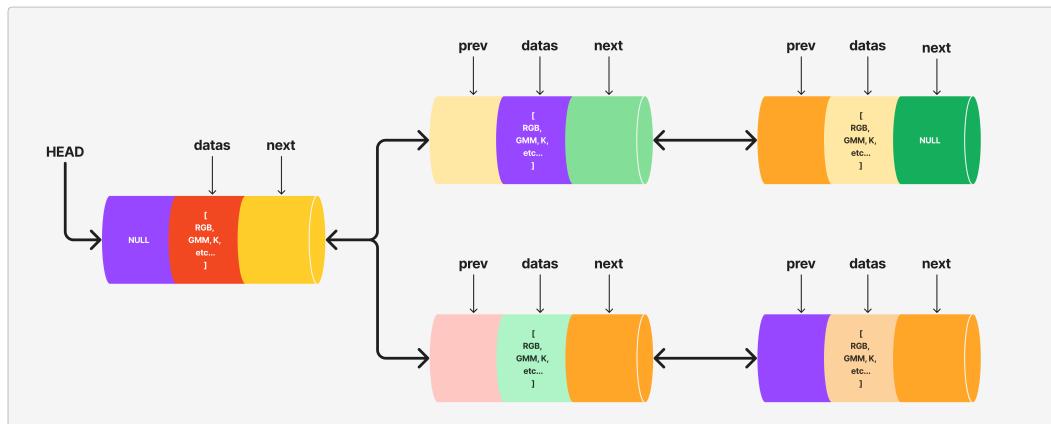
3.3.2 *Linkedlist*

Secara umum komponen yang ada *node* di antaranya data yang disimpan dan referensi kepada *node* selanjutnya, *linkedlist* akan terbentuk dari rangkaian *node* yang saling terhubung.



Gambar 3.7: Komponen yang ada pada sebuah *node*

Pada penelitian ini struktur komponen *node* yang akan penulis gunakan memiliki tiga komponen itu referensi ke *node* sebelumnya, data dari *node*, dan referensi ke *node* selanjutnya.



Gambar 3.8: *Linkedlist* terdiri dari susunan *node*

Referensi pada *node* bertujuan untuk mengetahui *node* mana yang menjadi *parent* menggunakan referensi ke *node* sebelumnya dan *child* menggunakan referensi ke *node* selanjutnya. *Node* yang berada di awal graf akan menjadi *head* dan *node* terakhir akan ditetapkan sebagai *tail*, sementara komponen data-data *node* akan disimpan sebagai karakteristik dari *node* tersebut.

```
// Struktur Data LinkedList
```

```

class Node {
public:
    bool isForeground;
    double foregroundProbability;
    double backgroundProbability;
    vector<GaussianComponent> gmmComponents;

    Node* parent;
    vector<Node*> children;

    Node(bool isForeground,
         double foregroundProbability,
         double backgroundProbability) {

        this->isForeground = isForeground;
        this->foregroundProbability =
        foregroundProbability;
        this->backgroundProbability =
        backgroundProbability;

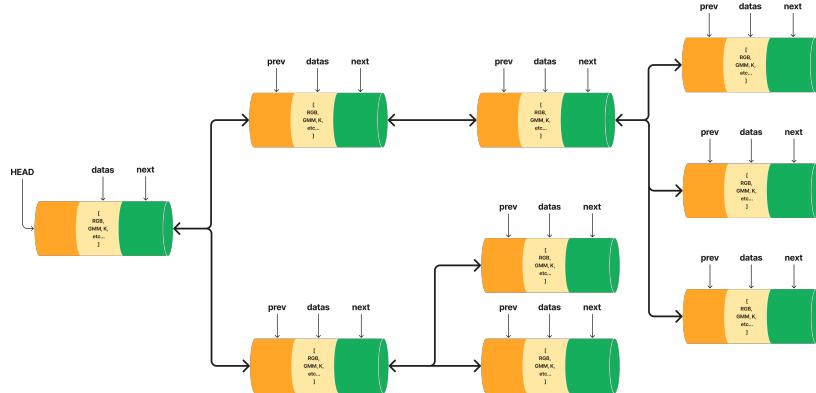
        this->parent = nullptr;
    }
}

```

Listing 3.2: Struktur data *doubly linkedlist*

3.3.3 Tree

Struktur data *tree* merupakan suatu struktur data hirarki, *tree* diibaratkan sebagai hubungan induk dan anak dalam sebuah pohon, *node* paling atas dari *tree* disebut dengan akar (*root*), dan *node* turunannya disebut sebagai anak dari *node* lain atau menjadi daun (*node* yang tidak memiliki anak)



Gambar 3.9: Representasi struktur data *tree*

sehingga struktur data yang terbentuk dari *tree* adalah sebagai berikut:

```

class Node {
public:
    bool isForeground;
    double foregroundProbability;
    double backgroundProbability;
    vector<GaussianComponent> gmmComponents;

    Node* parent;
    vector<Node*> children;

    Node(bool isForeground,
         double foregroundProbability,
         double backgroundProbability) {

        this->isForeground = isForeground;

        this->foregroundProbability =
foregroundProbability;

        this->backgroundProbability =
backgroundProbability;

        this->parent = nullptr;
    }
}

```

```

class Tree {
private:
    Node* root;
public:
    Tree() {
        root = nullptr;
    }

    Node* createNode (
        bool isForeground,
        double foregroundProbability,
        double backgroundProbability) {

        Node* newNode = new Node(
            isForeground,
            foregroundProbability,
            backgroundProbability);
        return newNode;
    }

    void insertChild(Node* parent, Node* child) {
        if (parent == nullptr || child == nullptr)
            return;

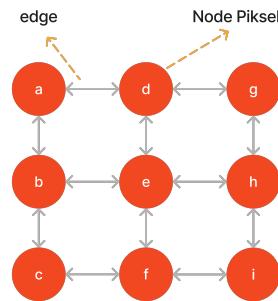
        child->parent = parent;
        parent->children.push_back(child);
    }
};

```

Listing 3.3: Struktur data *tree*

3.3.4 Graf

Struktur data graf dipakai untuk merepresentasikan gambar citra luka, citra yang di input akan berbentuk matriks dua dimensi yang mana terdiri dari baris, kolom, dan ranah warna RGB, graf selanjutnya dibangun dan menghubungkan antar *node* piksel, penulis membuat ilustrasi dari graf sebagai berikut



Gambar 3.10: Ilustrasi graf terhadap piksel

Berikut struktur data untuk membangun graf

```

class Graph {
public:
    int num_nodes;
    vector<vector<int>> adj_list;

    // Constructor to initialize the graph with num_nodes vertices
    Graph(int num_nodes) {
        this->num_nodes = num_nodes;
        adj_list.resize(num_nodes);
    }

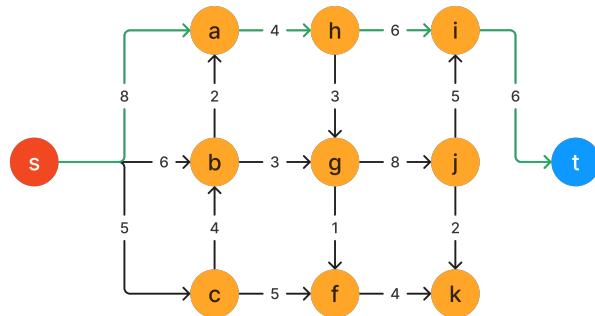
    // Function to add an edge between two vertices
    void add_edge(int u, int v) {
        adj_list[u].push_back(v);
        // adj_list[v].push_back(u); // If the graph is undirected
    }

    // Function to get edges of the graph
    vector<pair<int, int>> get_edges() {
        vector<pair<int, int>> edges;
        for (int u = 0; u < num_nodes; u++) {
            for (int v : adj_list[u]) {
                edges.push_back(make_pair(u, v));
            }
        }
        return edges;
    }
};

```

Listing 3.4: Struktur data untuk membangun graf

Setelah graf terbentuk, algoritma akan mulai mencari *path* dari setiap *node* dengan ditentukannya dua *node* khusus yaitu *s* (*source*) dan *t* (*sink*), dimana *node* *s* akan menjadi menjadi *foreground* dan *t* menjadi *background*. Fungsi *path* akan penulis jelaskan pada bagian 3.7.



Gambar 3.11: Garis berwarna hijau menunjukkan *path* s-a-h-i-t

Berikut struktur data untuk mencari *path*

```

class Node {
public:
    char name;
    int data;
    int capacity; // Capacity of the edge between this node and
    its parent
    Node* parent;
    vector<Node*> children;
    map<Node*, int> edges; // Map to store child nodes and
    their capacities

    Node(char name) {
        this->name = name;
        parent = nullptr;
        // capacity = 0; // Initialize capacity to 0
    }
};
  
```

```

class Tree {
private:
    Node* root;
public:
Tree() {
    root = nullptr;
}

void insertChild(Node* current, Node* next, int capacity) {
    Node* rootOfCurrent = getRoot(current);
    Node* rootOfNext = getRoot(next);

    if(next->parent == nullptr || rootOfNext->name ==
rootOfCurrent->name) {

        next->parent = current;
        current->edges[next] = capacity;
        current->children.push_back(next);

        // get root of each node
        Node* nextRoot = getRoot(next);
        cout << "node " << next->name
        << " masuk ke tree "
        << nextRoot->name ;
        cout << endl;

    }else if (rootOfNext->name != rootOfCurrent->name) {
        cout << endl << "ada path di: "
        << next->name << endl;
        cout << "path nya adalah: "<< endl;
        getPath(current, next);
        cout << "masuk ke tahap augmentasi"
        << endl << endl;
    }

}

// Add this helper function to get the capacity between two
nodes
int getEdgeCapacity(Node* parent, Node* child) {
    // Find the edge connecting the parent and child nodes
    auto it = parent->edges.find(child);
}

```

```

    if (it != parent->edges.end()) {
        return it->second; // Return the capacity of the edge
    }
    return 0; // Return 0 if the edge is not found (should not
happen if the tree is correctly constructed)
}

void display(Node* node, int depth = 0) {

    if (node == nullptr)
        return;

    // Print current node with indentation
    for (int i = 0; i < depth; ++i) {
        cout << "    ";
    }
    cout << "|-- " << node->name << " (Capacity: " << node->
capacity << ")" << endl;

    // Recursively display children
    for (Node* child : node->children) {
        display(child, depth + 1);
    }
}

void displayTree(Node* root) {
    cout << endl << "Tree dari Search Node "
<< root->name << " adalah: " << endl;
    display(root);
}

Node* getRoot(Node* child) {

    Node* current = child;
    while (current->parent != nullptr) {
        current = current->parent;
    }

    return current;
}

```

```

vector<Node*> getLeafNodes(Node* node) {
    vector<Node*> leafNodes;

    if (node == nullptr) {
        return leafNodes;
    }

    if (node->children.empty()) {
        leafNodes.push_back(node);
    } else {
        for (Node* child : node->children) {
            vector<Node*> childLeafNodes =
                getLeafNodes(child);

            leafNodes.insert(
                leafNodes.end(),
                childLeafNodes.begin(),
                childLeafNodes.end()
            );
        }
    }
    return leafNodes;
}

void displayLeafNodes(Node* root) {
    vector<Node*> leafNodes = getLeafNodes(root);
    cout << "Leaf nodes: ";
    for (Node* leafNode : leafNodes) {
        cout << leafNode->name << " ";
    }
    cout << endl;
}

bool getPathToRoot(Node* leaf,
vector<char>& path) {
    if (leaf == nullptr)
        return false;

    Node* current = leaf;
    while (current != nullptr) {
        path.push_back(current->name);
        current = current->parent;
    }
}

```

```

    }

    return true;
}

vector<char> getPathFromLeafToRoot(Node* leaf) {
    vector<char> path;
    getPathToRoot(leaf, path);
    return path;
}

void displayLeafToRoot ( Node* leaf) {
    vector<char> path =
    getPathFromLeafToRoot(leaf);
    getPathToRoot(leaf, path);

    // Display the path
    if(path[path.size()-1] != 't'){
        reverse(path.begin(), path.end());
    }
    cout << "Path from leaf " << leaf->name
    << " to root:" << endl;
    for (int i = 0; i < path.size(); i++) {
        cout << path[i];
        if(i != path.size() -1){
            cout<< "->";
        }
    }
    cout << endl;
}

void getPath(Node* current, Node* next){
    vector<char> tmp_current =
    getPathFromLeafToRoot(current);
    vector<char> tmp_next =
    getPathFromLeafToRoot(next);
    vector<char> path;
    int totalCapacity = 0;

    for (char cek : tmp_current){
        cout<< cek ;
    };
}

```

```

cout<<endl;

for (char cek2 : tmp_next) {
    cout<< cek2;
}

cout<< endl;

if(tmp_current[tmp_current.size() - 1] == 't') {

    reverse(tmp_next.begin(), tmp_next.end());

    for(char node : tmp_next) {
        path.push_back(node);
        if (path.size() > 1) {
            Node* parent = next->parent;
            int capacity = getEdgeCapacity(parent, next);
            totalCapacity += capacity;
            cout << "Capacity from " << parent->name << "
to " << next->name << ":" << capacity << endl;
        }
    }
    for(char node : tmp_current) {
        path.push_back(node);
        if (path.size() > 1) {
            Node* parent = current->parent;
            int capacity = getEdgeCapacity(parent, current)
;
            totalCapacity += capacity;
            cout << "Capacity from " << parent->name << "
to " << next->name << ":" << capacity << endl;
        }
    }
} else{

    reverse(tmp_current.begin(), tmp_current.end());

    for(char node : tmp_current) {
        path.push_back(node);
}
}

```

```

        if (path.size() > 1) {
            Node* parent = current->parent;
            int capacity = getEdgeCapacity(parent, current)
;

            totalCapacity += capacity;
            cout << "Capacity from cu " << parent->name <<
" to " << current->name << ":" << capacity << endl;
        }
    }

    for(char node : tmp_next){
        path.push_back(node);
        if (path.size() > 1 && next->parent != nullptr) {
            Node* parent_tmp = next->parent;
            // cout<< parent->name;
            int capacity = getEdgeCapacity(parent_tmp, next
);

            totalCapacity += capacity;
            cout << "Capacity from ci " << parent_tmp->name
<< " to " << next->name << ":" << capacity << endl;
            next = parent_tmp;
        }
    }

    for (char i : path) {
        cout << i;
    }
}
}

```

Listing 3.5: Struktur data untuk mencari *path*

3.3.5 Algoritma *GrabCut*

Struktur data penelitian algoritma *GrabCut* pada penelitian kali ini sebagai berikut:

```

// Struktur Data GrabCut
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/vector.hpp>

```

```
#include <vector>
#include <algorithm>
#include <numeric>
#include <cmath>
#include <map>

class GrabCut {
private:
    cv::Mat gambar;
    int rows, cols;
    cv::Mat gambar_mask;
    int komponen_gmm;
    int nilai_gamma;
    int nilai_beta;

public:
    GrabCut(const cv::Mat& gambar, const cv::Mat& gambar_mask, int
rect[4] = nullptr, int komponen_gmm = 5) {
        this->gambar = gambar;
        this->rows = gambar.rows;
        this->cols = gambar.cols;
        this->gambar_mask = gambar_mask;
        this->komponen_gmm = komponen_gmm;
        this->nilai_gamma = 50;
        this->nilai_beta = 0;
    }

    void inisiasi_piksel() {}

    void init_assign_gmm() {}

    void memperbarui_gmm() {}

    void hitung_smoothness() {}

    void bangun_graf() {}

    void graphcut_segmentation() {}
};

class GaussianMixture {
private:
```

```

int komponen_gmm;
int jumlah_fitur;
vector<int> jumlah_sampel;
ublas::vector<double> coefs;
ublas::matrix<double> means;
ublas::matrix<double> covariances;

public:
GaussianMixture(const ublas::matrix<double>& X, int
komponen_gmm = 5) {
    this->komponen_gmm = komponen_gmm;
    this->jumlah_fitur = x[1].size();
    this->jumlah_sampel = ublas::zero_vector<double>(
komponen_gmm);

    this->coefs = ublas::zero_vector<double>(komponen_gmm);
    this->means = ublas::zero_matrix<double>(komponen_gmm,
jumlah_fitur);
    this->covariances = ublas::zero_matrix<double>(komponen_gmm
, jumlah_fitur * jumlah_fitur);
}

void init_rand() {}

void dis_mult() {}

void prob_calc() {}

void learn_gmm() {}

void count_params() {}

}

class GraphCut {
public:

    void build_graph() {}

    void run() {}

    void growth() {}
}

```

```
void augmentation() {}  
  
void adoption() {}  
}
```

Listing 3.6: Struktur data GrabCut

3.4 Algoritma Segmentasi Gambar dengan Metode *GrabCut*

Algoritma *GrabCut* terdiri dari gabungan beberapa algoritma lain, yaitu pemrosesan oleh algoritma GMM dan minimisasi energi oleh algoritma *GraphCut*, berikut algoritma dari *GrabCut*:

Algorithm 1 Algoritma segmentasi gambar dengan *GrabCut* (Rother et al., 2004)

```

gambar ← LOADIMAGE()                                ▷ Input gambar 2.jpg
bool kotak ← false                               ▷ Inisiasi keberadaan kotak
bool drawing ← false                            ▷ Inisiasi keberadaan brush
iy, ix ← int
gambar2 ← COPY(gambar)
mask ← UBLAS::ZERO_MATRIX<DOUBLE>(gambar.shape[0], gambar.shape[1])
map < string, string > flagsColor
map < string, integer > flagsValue
flagsColor['bg'], flagsColor['fg'] ← 'black', 'white'
flagsValue['bg'], flagsValue['fg'], flagsValue['prob_bg'], flagsValue['prob_fg']
← 0, 1, 2, 3

komponen_piksel ←
UBLAS::ZERO_MATRIX<DOUBLE>(gambar.shape[0], gambar.shape[1])
array kotak_lok[4] ← [...]

function MOUSEHANDLER(event, x, y, flagsColor, flagsValue, param)
if event == cv(EVENT_RBUTTON) then
    ix, iy = x, y
    CV.RECTANGLE(gambar, (ix, iy), (x, y), flagsColor['warna'], 2)
    kotak ← True
    kotak_lok[4] ← [min(ix, x), min(iy, y), abs(ix - x), abs(iy - y)]
end if
if event == cv(EVENT_LBUTTON) then
    drawing ← True
    CV.CIRCLE(gambar, (x, y), 3, flagsColor['warna'], -1)
    CV.CIRCLE(mask, (x, y), 3, flagsValue['value'], -1)
    drawing ← False
end if
end function

```

▷ (**Algorithm 2**) Inisiasi TU = 1 untuk piksel didalam kotak, gambar 2.11

```

function INISASI_PIKSEL(mask)
    if kotak_lok[4] ≠ None then
        mask[kotak_lok[...]] = 1
    end if
    idx_bg ← bg[‘value’] or pr_bg[‘value’]
    idx_fg ← fg[‘value’] or pr_fg[‘value’]
end function

▷ (Algorithm 3) Inisiasi dan Assign GMM ke setiap piksel, tahap 1 gambar 2.11
function INIT_ASSIGN_GMM(idx_bg, idx_fg)
    GMM_FG ← GaussianMixture(idx_fg)
    GMM_BG ← GaussianMixture(idx_bg)
    GMM_FG.DIS_MULT(idx_fg)
    GMM_BG.DIS_MULT(idx_bg)
end function

▷ (Algorithm 4) mempelajari parameter GMM, tahap 2 gambar 2.11
function MEMPERBARUI_GMM
    GMM_FG.COUNT_PARAMS(gambar[idx_fg], komponen_piksel[idx_fg])
    GMM_BG.COUNT_PARAMS(gambar[idx_bg], komponen_piksel[idx_bg])
end function

▷ (Algorithm 5) segmentasi piksel, tahap 3 gambar 2.11
function GC_SEGMENTATION
    vector<int> gc_graph ← BUILD_GRAPH(edges, n_piksel)
    MINCUT_SEGMENTATION(gc_graph, gc_sink, gc_source, gc_graph_capacity)
end function
```

Algorithm 2 Algoritma inisialisasi *bounding box* pada area luka

▷ Tahap inisialisasi pada gambar 2.11

```

function INISASI_PIKSEL(mask)
    if kotak_lok ≠ None then
        mask[kotak_lok[1] : kotak_lok[1] + kotak_lok[3],
        kotak_lok[0] : kotak_lok[0] + kotak_lok[2]] ← flags[‘fg’][‘value’]
    end if
    idx_bg ← where(mask == flags[‘bg’][‘value’] or
        mask == flags[‘pr_bg’][‘value’])
    idx_fg ← where(mask == flags[‘fg’][‘value’] or
        mask == flags[‘pr_fg’][‘value’])
end function
```

Algorithm 3 Algoritma inisiasi dan Assign GMM pada setiap piksel citra

```

typedef ublas :: matrix < double > Matrix;
typedef ublas :: vector < double > Vector;
function INIT_ASSIGN_GMM(idx_bg, idx_fg)
  GMM_BG ← GaussianMixture(gambar[idx_bg])
  GMM_FG ← GaussianMixture(gambar[idx_fg])
  komponen_piksel[idx_bg] ← GMM_BG.DIS_MULT(gambar[idx_bg])
  komponen_piksel[idx_fg] ← GMM_FG.DIS_MULT(gambar[idx_fg])
end function

function DIS_MULT(target) ▷ rumus 2.4
  gauss_score ← NP.ZEROS(target.shape[0])
  INIT_RAND(target)
  for k = 0; k < komponen_gmm; k + + do
    if coefs > 0 then
      VectorXminMu ← target - means[k]
      Vectortrans_XminMu ← UBLAS::TRANS(XminMu)
      Matrixinv_cov = ublas :: inv(self.covariances[ci])
      tmp_dot ← UBLAS::PROD(inv_cov, trans_XminMu)
      tmp_mult ← UBLAS::ELEMENT_PROD('ij, ij->i',
        XminMu, ublas :: trans(tmp_dot))
      pembagi ← SQRT(2 * M.PI) * SQRT(ublas :: det(covarians[k]))
      gauss_score ←  $\frac{\text{NP.EXP}(-0.5 * \text{tmp\_mult})}{\text{pembagi}}$ 
    end if
  end for
  return NP.ARGMAX(gauss_score)
end function

function INIT_RAND(target)
  vector<int> labelPix[target.shape[0]]
  for int i = 0; i < target.shape[0]; i + + do
    labelPix[i] ← rand() mod 5;
  end for
  COUNT_PARAMS(target, labelPix)
end function
  
```

Algorithm 4 Algoritma mempelajari parameter GMM

▷ Tahap mempelajari GMM 2.11

```

vector < int > unique_labels;
vector < int > labelCounts;
function MEMPERBARUI_GMM
    COUNT_PARAMS(gambar[idx_bg], komponen_piksel[idx_bg])
    COUNT_PARAMS(gambar[idx_fg], komponen_piksel[idx_fg])
end function
function COUNT_PARAMS(ublas :: matrix < double > target,
                      ublas :: matrix < double > labels)
    tie(unique_labels, labelCounts) = uniqueWithCounts(labels);
    for int k = 0; k < unique_labels; k++ do
        coeffs[k] ← labelCounts[k]/accumulate(labelCounts.begin(), labelCounts.end(), 0)
        means[k] ← ublas :: mean(target[k == labels])
        covariances[k] ← ublas :: trans(ublas :: cov(target[k == labels]))
    end for
end function
function pair < vector < int >, vector < int > >
    search_uniques(const ublas :: matrix < double > &labels)
        vector < int > unique_labels;
        vector < int > labelCounts;
        map < int, int > countMap;
            for const auto& label : labels do
                countMap[label]++;
            end for
            return make_pair(unique_labels, labelCounts);
end function
  
```

Algorithm 5 Algoritma segmentasi oleh *mincut*

```

vector<int> edges
vector<double> D_score
vector<int> gc_graph_capacity
int n_piksel  $\leftarrow$  gambar.shape[0] * gambar.shape[1]
int gc_source  $\leftarrow$  n_piksel
int gc_sink  $\leftarrow$  gc_source + 1
function gc_segmentation(vector<int> edges, int n_piksel)
    vector<int> gc_graph  $\leftarrow$  BUILD_GRAPH(edges, n_piksel)
    MINCUT_SEGMENTATION(gc_graph, gc_sink, gc_source, gc_graph_capacity)
end function

```

Algorithm 6 Membangun graf

```

     $\triangleright$  Tahap membangun graf
vector<int> idx_bg  $\leftarrow$  np.where(mask.reshape(-1) == flagsValue['bg'])
vector<int> idx_fg  $\leftarrow$  np.where(mask.reshape(-1) == flagsValue['fg'])
vector<int> idx_pr  $\leftarrow$  np.where(mask.reshape(-1) == flagsValue['prob_bg'] or mask.reshape(-1) == flagsValue['prob_fg'])
function build_graph(vector<int> edges, int n_piksel)
    Graph graph_gc((n_piksel + 2))
    for int i in (idx_bg[0], idx_fg[0], idx_pr[0]) do
        GRAPH_GC.ADD_EDGE(gc_source, i)
        GRAPH_GC.ADD_EDGE(gc_sink, i)
        D_score = -np.log(GMM_BG or GMM_FG
                           .count_prob(gambar.reshape(-1, 3)[idx_pr]))
    end for
    GC_GRAPH_CAPACITY.PUSH_BACK((D_score))
end function

```

Algorithm 7 Menghitung nilai D pada rumus 2.24

```

vector<double> prob
function count_prob(vector<int> target)
    prob  $\leftarrow$  [DIS_MULT(target)]
    return ublas :: dot(coefs, prob)
end function

```

Algorithm 8 Segmentasi oleh algoritma *mincut* 2.11

```

vector<Node*> active_nodes
vector<Node*> orphans_nodes
vector<Node*> paths ← null
vector<double> tree_cap

Tree tree_S
Tree tree_T
Tree tree
Node*root_S ← newNode(gc_source)
Node*root_T ← newNode(gc_sink)
active_nodes ← ACTIVE_NODES.PUSH_BACK(gc_source, gc_sink)
function mincut_segmentataion(vector<int> gc_graph, gc_sink, gc_source, gc_graph_capacity)
    while true do
        GROWTH_STEP()
        if paths ≠ 0 then
            AUGMENTATION_STEP(GC_SINK, GC_SOURCE)
        end if
        ADOPTION_STEP(GC_SINK, GC_SOURCE)
    end while
end function

```

Algorithm 9 tahap growth pada *mincut* di bagian 2.5.3.2.1

```

function growth_step
    vector<Node*> list_neighbors
    SEARCH_NEIGHBORS(active_nodes, tree_cap)
    while active_nodes ≠ 0 do
        for int i; i < list_neighbors.size(); i ++ do
            if tree_cap[i] > 0 then
                if tree.getRoot(list_neighbors[i]) == null then
                    tree.insertChild(active_nodes[0], list_neighbors[i])
                else if tree.getRoot(list_neighbors[i]) ≠ null and
                    tree.getRoot(list_neighbors[i]) ≠ tree.getRoot(active_nodes[0]) then
                        paths ← tree.getPath(active_nodes[0], list_neighbors[i])
                    end if
                end if
            end for
            active_nodes.pop()
        end while
        return paths ≠ 0
end function

```

Algorithm 10 tahap augmentasi pada *mincut* di bagian 2.5.3.2.2

```

function      augmentation_step(vector<int> gc_graph, gc_sink, gc_source)
    tree.get_residu(paths, gc_graph_capacity)
    for int i; i < path_residu.size(); i + + do
        if tree.getRoot(i[0]) == tree.getRoot(i[1]) == gc_source then
            tree.setParentNull(i[1])
            orphans.push_back(i[1])
        else if tree.getRoot(i[0]) == tree.getRoot(i[1]) == gc_sink then
            tree.setParentNull(i[0])
            orphans.push_back(i[0])
        end if
    end for
end function

```

Algorithm 11 tahap augmentasi pada *mincut* di bagian 2.5.3.2.3

```

function adoption_step(vector<int> gc_graph, gc_sink, gc_source)
    Node*current_orphans
    while orphans ≠ 0 do
        current_orphans ← orphans[0]
        SEARCH_NEIGHBORS(current_orphans, tree_cap)
        orphans.pop()
        for int i; i < list_neighbors.size(); i + + do
            if tree.getRoot(list_neighbors[i]) ==
                tree.getRoot(current_orphans) and
                tree.getEdgeCapacity(list_neighbors[i],
                current_orphans) > 0 then
                    tree.insertChild(list_neighbors[i], current_orphans)
                    break
            end if
        end for
    end while
end function

```

3.5 Alat dan Bahan Penelitian

Pada penelitian kali ini, penulis menggunakan perangkat keras sebagai berikut:

1. Laptop dengan prosesor AMD Ryzen 5 3500H *series* dan *RAM 16 GB*
2. Koneksi berbasis Wi-Fi serta kuota internet dari ponsel pintar

Untuk perangkat lunak yang penulis gunakan sebagai berikut:

1. Windows 11 64 bit OS
2. Visual Studio Code sebagai *Code Editor*
3. Python 3 untuk menjalankan program Python

3.6 Tahapan Penelitian

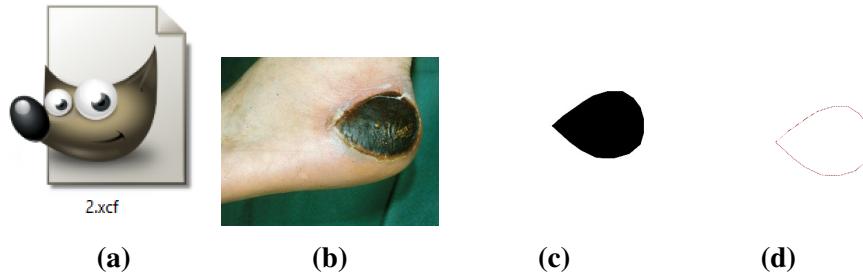
Dalam penelitian ini penulis melakukan perancangan penelitian terhadap citra luka yang akan di segmentasi antara objek luka dan latar belakang, perancangan yang akan penulis lakukan yaitu

3.6.1 Persiapan *dataset* input citra

Sebelum melakukan inisiasi pada luka, penulis perlu menyiapkan *dataset* luka, *dataset* ini kemudian penulis jadikan sampel data input citra. Sampel data dinilai baik ketika sampel tersebut mencerminkan populasi dari data tersebut (Rizki, 2022). Silva menggunakan 105 citra luka tentang segmentasi luka otomatis dengan menggunakan SVM dan Grabcut(Silva:2015). Dalam penelitiannya Wang melakukan automasi segmentasi luka dengan *Deep Convolutional Neural Networks* menggunakan 2700 citra gambar luka (Wang et al., 2015).

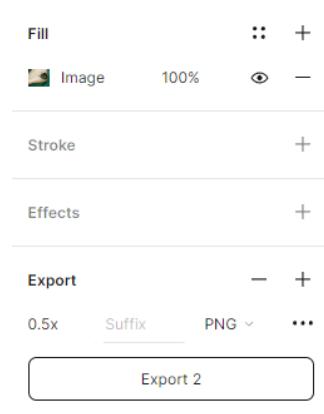
Penulis menggunakan *dataset* berjumlah 108, 37 data tidak dapat digunakan karena terdapat duplikasi dengan data lain sehingga tersedia 71 buah citra yang penulis jadikan sebagai populasi sekaligus sampel (sampel = populasi), citra tersebut memiliki kategori sesuai dengan warna luka, yaitu luka hitam sebanyak 24 citra, luka kuning sebanyak 15 citra, dan luka merah 32 citra. *Dataset* penulis dapat dari penelitian luka Ns. Ratna Aryani, K.Kep, tahun 2018 (Aryani et al., 2018) yang terdapat pada repositori <https://github.com/mekas/InjuryDetection>.

Data awal yang penulis dapat adalah data-data yang berekstensi .xcf yang bisa dibuka menggunakan *software* GIMP, pengolahan data sebelum deteksi menggunakan *GrabCut* dilakukan menggunakan *software* Figma. Masing-masing *dataset* didalamnya terdapat *layer* citra (luka), *layer* region (luka), dan *path* sebagai berikut :



Gambar 3.12: (a)Data citra format .xcf, (b) *layer* citra (luka), (c) *layer* region, (d)*path*

Langkah selanjutnya adalah penulis memasukkan data citra 2(b) ke dalam figma untuk membuat *layer* dan region (luka) dengan menggunakan *pen tools*, hasil dari *pen tools* akan berupa *stroke* atau seperti path 2(d), kemudian penulis menduplikasi hasil path dan mengubahnya menjadi objek (*fill object*) seperti 2(c).



Gambar 3.13: Proses *resize* citra gambar luka dengan ukuran 0.5 dari ukuran awal

Setelah penulis melakukan *resize* ukuran citra, selanjutnya penulis *export* *layer* masing masing ke format .jpg.

3.6.2 Inisiasi *Bounding Box* di area luka

Penulis memulai dengan menentukan kotak pembatas (*bounding box*) yang mengelilingi objek yang ingin dipisahkan. Area di dalam *bounding box* akan digunakan untuk membuat *Trimap Unknown* (T_U), sementara area di luar *bounding box* dianggap sebagai *Trimap Background* (T_B). Setiap piksel yang termasuk dalam (T_B) akan memiliki nilai α_n sama dengan 0, sedangkan piksel yang termasuk dalam (T_U) akan memiliki nilai α_n sama dengan 1. *Bounding box* digunakan untuk mempercepat waktu komputasi algoritma dan meningkatkan akurasi segmentasi.



Gambar 3.14: Inisiasi area objek luka dengan *bounding box*

3.6.3 Inisiasi GMM

Pertama penulis akan melakukan inisiasi untuk tiap parameter GMM, di antaranya $\{\mu, \Sigma\}$, pada algoritma *grabcut* akan digunakan $K = 5$. Adapun inisiasi yang dilakukan adalah membuat nilai random terhadap μ dan Σ . setelah itu penulis menentukan $\omega_k = \frac{1}{5}$, untuk $k \in \{1, \dots, 5\}$. Distribusi gaussian yang akan digunakan untuk citra gambar berwarna yaitu distribusi gaussian *multivariate* 2 dimensi atau 3 dimensi, sebagai ilustrasi gausian berdistribusi normal 2 dimensi dengan objek citra gambar luka berwarna :

3.6.4 Mempelajari parameter GMM

Tahapan mempelajari parameter GMM dalam algoritma *GrabCut* adalah untuk menghasilkan model yang akurat dalam memodelkan distribusi warna dari objek dan latar belakang dalam gambar. Tahap ini digunakan untuk memperbarui parameter-parameter GMM berdasarkan probabilitas posteriori. Algoritma menggunakan probabilitas posteriori piksel sebagai bobot dalam perhitungan rata-rata dan kovariansi baru untuk setiap komponen GMM. Dengan demikian,

piksel yang memiliki probabilitas posteriori yang lebih tinggi akan memberikan kontribusi yang lebih besar dalam perhitungan parameter GMM.

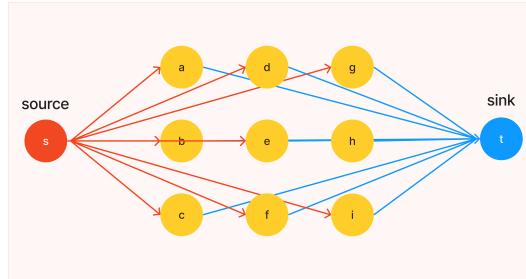
Langkah dalam memperbarui parameter GMM dilakukan dengan langkah-langkah sebagai berikut:

1. Untuk setiap komponen GMM dalam model objek dan latar belakang, hitung rata-rata baru. Rata-rata baru dihitung dengan menggunakan probabilitas posteriori piksel sebagai bobot dalam perhitungan rata-rata.
2. Selanjutnya, perbarui kovariansi baru untuk setiap komponen GMM. Perhitungan kovariansi baru juga menggunakan probabilitas posteriori sebagai bobot dalam perhitungan kovariansi.
3. Setelah perhitungan rata-rata dan kovariansi baru selesai untuk semua komponen GMM, model GMM telah diperbarui.

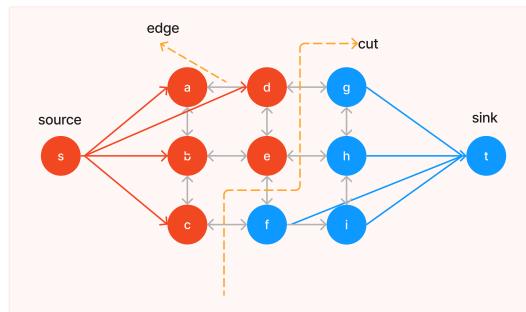
Dengan memperbarui parameter-parameter GMM berdasarkan probabilitas posteriori yang dihitung, algoritma secara bertahap meningkatkan model GMM agar sesuai dengan data gambar. Model yang lebih akurat membantu algoritma dalam membedakan antara piksel objek dan latar belakang, sehingga memperbaiki hasil segmentasi dengan lebih baik.

3.6.5 Segmentasi Citra dengan Algoritma *Mincut*

Selanjutnya penulis melakukan segmentasi dengan menggunakan algoritma *min-cut* atau *GraphCut*. Algoritma ini memvisualisasikan gambar sebagai graf, dan piksel sebagai *node* pada graf tersebut, kemudian *GraphCut* bertugas untuk segmentasi piksel mana yang termasuk *foreground* dan *background* dengan cara melakukan *cut* pada graf. Terdapat dua terminal pada graf yaitu terminal *source* s dan *sink* t, kedua terminal saling terhubung terhadap semua piksel yang ada di graf, piksel yang masuk ke dalam *source* akan menjadi *foreground* sementara piksel yang masuk ke dalam *sink* akan menjadi *background*. Setiap *node* pada graf memiliki hubungan yang yang disebut *edge*, *edge* akan memiliki arah dan bobot dimana bobot tersebut akan digunakan untuk mencari *mincut* dari graf.



(a) Sebuah graf



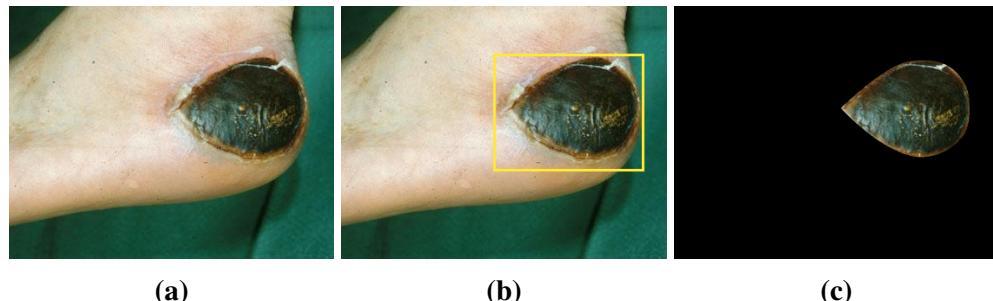
(b) Sebuah (cut)

Gambar 3.15: Contoh graf berkapasitas terarah.

Algoritma *mincut* terdiri dari beberapa tahap, penjelasan singkatnya yaitu tahap pertama dimana graf bertumbuh (*growth*) hingga menemukan jalan (*path*) dari *source* menuju *sink*, selanjutnya tahap kedua (*augmentasi*) dimana *path* yang sudah ditemukan akan dilihat kapasitas tiap *edge* nya lalu masing-masing kapasitas *edge* akan dikurangi dengan kapasitas paling kecil (*bottleneck*), apabila ada kapasitas *edge* yang kosong, maka *node* tersebut menjadi *Orphans* (tidak memiliki orang tua), setelah itu masuk tahap ketiga (*adopsi*) yaitu mencari orang tua baru dari *node* yang menjadi *orphans* sebelumnya, apabila tidak ada maka *node* tersebut menjadi *node* bebas. Struktur algoritma *mincut* dapat dilihat pada bagian 2.5.3.2.

Setiap piksel pada gambar memiliki komponen GMM tersendiri, pada tahap ini nilai k tiap piksel telah didapatkan dari GMM, dimana nilai k dari tiap piksel memiliki rentang dari 1 sampai 5. Pada permulaan algoritma *mincut* penulis menentukan *node* mana yang menjadi tetangga dari *source* atau *sink* dengan menghitung jumlah nilai D dari persamaan 2.24 pada setiap komponen GMM, kemudian penulis menjumlahkan nilai yang didapat dan membaginya dengan 5 (total K), nilai yang didapat akan dijadikan sebagai *threshold* dalam menentukan hubungan tiap piksel ke *Source* atau ke *Sink*. Tahap ini mensegmentasi citra yang

telah dihitung setiap komponen pikselnya pada langkah-langkah sebelumnya sesuai dengan persamaan 2.22.



Gambar 3.16: (a) Data citra luka 2.png, (b) *Bounding box* area luka, (c) Hasil segmentasi citra

3.6.6 Menghitung Tingkat Akurasi

Untuk menemukan nilai hasil segmentasi citra luka dengan menggunakan algoritma *GrabCut*, yaitu mencari nilai akurasi citra, dengan menghitung selisih piksel dari area segmentasi dengan area citra referensi sehingga akan menghasilkan nilai akurasi berikut :

$$Akurasi(\%) = 100 - \left| \frac{(luas area citra referensi - luas area segmentasi)}{luas area citra referensi} \right| * 100 \quad (3.1)$$

3.7 Skenario Eksperimen

Berikut adalah skenario eksperimen untuk "Deteksi Area Keliling Luka Kronis Menggunakan GRABCUT"

1. Input gambar luka kronis
2. Pengguna memberi kotak pembatas di sekeliling area luka (objek) pada gambar luka
3. Inisiasi titik koordinat (y, x) kotak pembatas di dalam gambar luka
4. Memberi penanda area di luar kotak dengan angka 1, dan area di luar kotak dengan angka 0

5. Memasukkan komponen GMM di setiap piksel untuk menghitung probabilitas tiap piksel
6. Mempelajari parameter GMM dengan melihat hasil probabilitas objek atau latar belakang
7. Segmentasi area keliling luka dengan menggunakan algoritma *GraphCut*
8. Ulangi sampai sesuai dengan area luka
9. Mendapatkan Nilai *Image Similarity*

BAB IV

HASIL DAN PEMBAHASAN

4.1 Pemrosesan citra gambar input

Langkah pertama dalam pemrosesan citra gambar input adalah menentukan rasio piksel dari citra tersebut, penulis mengubah lebar citra menjadi 320 piksel untuk efisiensi serta menyamakan ukuran tiap sampel citra gambar, berikut *source code* nya:

```
def setSizeImg(self):
    global new_width, aspect_ratio, new_height
    # set ukuran gambar
    new_width = 320 # Atur ukuran hanya 320 px
    aspect_ratio = self.image.width / self.image.height
    new_height = int(new_width / aspect_ratio)

    # Load image
    self.image = Image.open(self.image_path)

    # Resize ukuran gambar
    self.setSizeImg()
    self.image = self.image.resize((new_width, new_height))
```

Gambar 4.1: *source code* mengubah ukuran lebar citra menjadi 320 piksel

Fungsi `setSizeImg()` dibuat untuk mengubah ukuran lebar piksel citra menjadi 320 piksel, untuk mempertahankan rasionalya, maka dibentuk dengan menghitung menggunakan rumus `aspect_ratio` dimana tinggi gambar akan otomatis mengikuti lebar gambar dengan perbandingan yang stabil.

4.2 Deteksi keliling luka menggunakan *Grabcut*

Alur deteksi keliling luka menggunakan algoritma *Grabcut* dilakukan melalui beberapa tahapan, setiap tahapan akan mengolah data input berupa citra gambar luka yang disimpan menjadi *array* multidimensi (lebar, tinggi, nilai RGB). Berikut adalah tahapannya:

4.2.1 Inisiasi kotak (*Bounding Box*)

Dalam segmentasi citra luka, dimulai dengan inisiasi kotak atau *bounding box* yang digambar mengelilingi area citra luka, berikut *source code* nya:

```

def drawing_rectangle(self):
    print("mulai gambar kotak")
    self.main_canvas.bind("<ButtonPress-1>", self.onClick_rect)
    self.main_canvas.bind("<B1-Motion>", self.onDrag_rect)
    self.main_canvas.bind("<ButtonRelease-1>", self.onRelease_rect)

def onClick_rect(self, event):
    print("Keberadaan kotak: ", KOTAK["is_drawn"])
    if KOTAK["is_drawn"] is not True:
        KOTAK["titik_start"] = (event.x, event.y)
    else:
        print("kotak sudah ada")

def onDrag_rect(self, event):
    if KOTAK["is_drawn"] is not True:
        KOTAK["titik_akhir"] = (event.x, event.y)
        self.update_image()

def onRelease_rect(self, event):
    if KOTAK["titik_start"] and KOTAK["titik_akhir"]:
        KOTAK["coord"] = (self.get_rectangle_coords())
        print("Koordinat Kotak: ", KOTAK["coord"])
        KOTAK["titik_start"] = None
        KOTAK["titik_akhir"] = None
        KOTAK["is_drawn"] = True
        print("Keberadaan kotak: ", KOTAK["is_drawn"])
        print("Koordinat Kotak: ", KOTAK["coord"])

def get_rectangle_coords(self):
    if KOTAK["titik_start"] and KOTAK["titik_akhir"]:
        x1, y1 = KOTAK["titik_start"]
        x2, y2 = KOTAK["titik_akhir"]
        return (x1, y1, x2, y2)
    else:
        return None

def drawing_rectangle(self):
    print("mulai gambar kotak")
    self.main_canvas.bind("<ButtonPress-1>", self.onClick_rect)
    self.main_canvas.bind("<B1-Motion>", self.onDrag_rect)
    self.main_canvas.bind("<ButtonRelease-1>", self.onRelease_rect)

```

Gambar 4.2: *source code* menggambar kotak pada area luka

Ketika kotak sudah tergambar, koordinat dimasukkan kedalam variabel KOTAK["coord"] pada fungsi onRelease_rect() yang mana KOTAK["coord"] berupa array berisi x1, y1, x2, y2 yang akan digunakan untuk tahap selanjutnya.

4.2.2 Inisiasi piksel

Tahap selanjutnya adalah membuat salinan *masking* pada citra gambar luka yang diubah menjadi gambar hitam, salinan dibuat dengan menggunakan *library Numpy* dengan target citra gambar luka. Berikut *source code* nya:

```
self.gambar = np.array(self.image)
self.gambar2 = self.gambar.copy()
print("ukuran gambar: ", self.gambar.shape)

# buat masking dari gambar awal
self.mask = np.zeros(self.gambar2.shape[:2], dtype=np.uint8)
self.mask2 = self.mask.copy()
```

Gambar 4.3: *source code* menggambar kotak pada area luka

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan percobaan yang telah dilakukan, maka didapatkanlah hasil kesimpulan sebagai berikut:

1. Kontribusi peneliti terdapat pada penemuan dibutuhkannya tahap menghitung nilai GMM pada tiap piksel sebelum menghitung *Mincut*.
2. Hasil dari deteksi keliling luka menggunakan *snake* versi *integer* yang kurva akhirnya berhasil menutupi luka berjumlah 12 data (dari 71 data) sedangkan untuk yang versi interpolasi berjumlah 44 data (dari 71 data). Hal ini menunjukkan bahwa data yang berhasil dideteksi menggunakan *snake* interpolasi lebih banyak dibandingkan versi *integer*.
3. Hasil dari deteksi keliling luka menggunakan *snake* versi *integer* untuk semua kategori menghasilkan 12 data (dari 71 data) yang kurva akhirnya berhasil menutupi luka dengan nilai akurasi rata-rata 77.18%.
4. Hasil dari deteksi keliling luka menggunakan *snake* versi *integer* untuk semua kategori menghasilkan 44 data (dari 71 data) yang kurva akhirnya berhasil menutupi luka dengan nilai akurasi rata-rata 86.1%.

5.2 Saran

Pemilihan metode segmentasi citra menggunakan algoritma *Grabcut* membuka peluang untuk melanjutkan penelitian dengan tujuan meningkatkan kinerja *Grabcut*. Salah satu pendekatan yang mungkin adalah mengganti atau memodifikasi metode segmentasi dengan metode citra alternatif, dengan harapan dapat meningkatkan akurasi hasil yang diperoleh dari penggunaan algoritma *Grabcut*.

DAFTAR PUSTAKA

- Aryani, R. et al., (2018). “Buku Panduan: Rancang Bangun Aplikasi Mobile Android Sebagai Alat Deteksi Warna Dasar Luka Dalam Membantu Proses Pengkajian Luka Kronis dengan Nekrosis”. In.
- Boykov, Y. and V. Kolmogorov (2004). “An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 26.9, pp. 1124–1137. arXiv: 0703101v1 [cs].
- Garcia-Zapirain, B. et al., (2017). “Automated Framework for Accurate Segmentation of Pressure Ulcer Images”. In: *Computers in Biology and Medicine* 90, pp. 137–145.
- Gonzalez, R. C. and R. E. Woods (2018). *4TH EDITION Digital image processing*, p. 1168.
- Han, G. (2017). “Chronic Wound Healing : A Review of Current Management and Treatments”. In: *Advances in Therapy* 34.3, pp. 599–610.
- Manohar Dhane, D. et al., (2017). “Fuzzy Spectral Clustering for Automated Delineation of Chronic Wound Region Using Digital Images”. In: *Computers in Biology and Medicine* 89, pp. 551–560.
- Nugraha, B. (2022). “Ekstraksi Latar Depan pada Citra Ikan dengan Metode GrabCut yang Diutamasi Menggunakan Saliency Map”. In: *Skripsi*.
- Power, P. W. and J. a. Schoonees (2002). “Understanding Background Mixture Models for Foreground Segmentation”. In: *Image and Vision Computing* 2002.November, pp. 267–271.
- Rizki, M. (2022). “Deteksi Keliling Luka Kronis Menggunakan Active Contour (Snake) dan Active Contour yang Ditambahkan Interpolasi.” In: *Skripsi*.
- Rother, C., V. Kolmogorov, and A. Blake (2004). “GrabCut - Interactive Foreground Extraction using Iterated Graph Cuts”. In: *ACM SIGGRAPH 2004 Papers, SIGGRAPH 2004*, pp. 309–314.
- Silva, R. H. L. e. and A. M. C. Machado (2021). “Automatic Measurement of Pressure Ulcers Using Support Vector Machines and GrabCut”. In: *Computer Methods and Programs in Biomedicine* 200.
- Velnar, T., T. Bailey, and V. Smrkolj (2009). “The Wound Healing Process: An Overview of the Cellular and Molecular Mechanisms”. In: *Journal of International Medical Research* 37.5, pp. 1528–1542.
- Wang, C. et al., (2015). “A Unified Framework for Automatic Wound Segmentation and Analysis with Deep Convolutional Neural Networks”. In: *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS* 2015-Novem, pp. 2415–2418.

LAMPIRAN A

Source code program

1.1 main.py

```
import sys
from image_editing import ImageEditing
from tkinter import *
from grabcut import GrabCut
from PIL import Image, ImageTk, ImageMath, ImageDraw
import numpy as np

if __name__ == '__main__':
    root = Tk()
    category = "luka_hitam"
    img_name = "2"
    extension = ".jpg"
    image_path = "dataset_3/" + category + "/" + img_name + extension
    tools = ImageEditing(root, image_path, img_name, category)
    root.mainloop()
```

1.2 image_editing.py

```
import sys
import os
from tkinter import *
from grabcut import GrabCut
from PIL import Image, ImageTk, ImageMath, ImageDraw
import numpy as np

COLOR = {
    'red' : [0, 0, 255],
    'white' : [255, 255, 255],
    'black' : [0, 0, 0],
    'yellow' : "#FFFF00"
}

F_BG = 0
F_FG = 1
F_PR_FG = 2

KOTAK = {
    "coord" : (),
    "pos" : None,
    "titik_start" : None,
    "titik_akhir" : None,
    'is_init' : False,
    'is_drawn' : False,
}

BRUSH = {
    "size" : 3,
    'is_init' : False,
    'is_drawn' : False,
}

class ImageEditing:
    def __init__(self, root, image_path, image_name, category):
        self.root = root
        self.root.title("Image Segmentation using Grabcut")
        self.image_path = image_path
        self.image_name = image_name
        self.category = category

        # Create a frame for title labels
        self.title_frame = Frame(self.root)
        self.title_frame.pack(side=TOP, fill=X)

        # Create a frame for canvases
        self.canvas_frame = Frame(self.root)
        self.canvas_frame.pack(side=TOP, fill=BOTH, expand=YES) # Adjusted to fill both X and Y, expand

        # Create a frame for buttons
        self.button_frame = Frame(self.root)
        self.button_frame.pack(side=BOTTOM, fill=BOTH)
```

```

# Load image
self.image = Image.open(self.image_path)

# Resize ukuran gambar
self.setSizeImg()
self.image = self.image.resize((new_width, new_height)) # <-- untuk resize ukuran

self.gambar = np.array(self.image)
self.gambar2 = self.gambar.copy()
print("ukuran gambar: ", self.gambar.shape)

# buat masking dari gambar awal
self.mask = np.zeros(self.gambar2.shape[:2], dtype=np.uint8)
self.mask2 = self.mask.copy()

self.LoadCanvas()

def LoadCanvas(self):
    # Make image as main canvas
    main_title_label = Label(self.title_frame, text="Main Canvas", font=('Helvetica', 10, 'bold'))
    self.photo = ImageTk.PhotoImage(self.image)
    self.main_canvas = Canvas(self.canvas_frame, width=self.image.width, height=self.image.height)
    self.main_canvas.create_image(0, 0, anchor=NW, image=self.photo)

    # Create mask canvas
    mask_title_label = Label(self.title_frame, text="Mask Canvas", font=('Helvetica', 10, 'bold'))
    self.mask_image = Image.fromarray(self.mask)
    self.mask_image = self.mask_image.resize((new_width, new_height)) # <-- untuk resize ukuran
    self.mask_photo = ImageTk.PhotoImage(self.mask_image)
    self.mask_canvas = Canvas(self.canvas_frame, width=self.image.width, height=self.image.height)
    self.mask_canvas.create_image(0, 0, anchor=NW, image=self.mask_photo)

    # Create a result canvas for result display
    result_title_label = Label(self.title_frame, text="Result Canvas", font=('Helvetica', 10, 'bold'))
    self.result_image = Image.new("RGB", (self.image.width, self.image.height))
    self.result_photo = ImageTk.PhotoImage(self.result_image)
    self.result_canvas = Canvas(self.canvas_frame, width=self.image.width, height=self.image.height)
    self.result_canvas.create_image(0, 0, anchor=NW, image=self.result_photo)

    # Create the buttons with custom styles
    self.draw_rect_button = Button(self.button_frame, text="Draw Rectangle", command=self.drawing_rectangle,
                                   bg="#4CAF50", fg='white', font=('Helvetica', 10))
    self.segmentation_button = Button(self.button_frame, text="Segmentation Grabcut", command=self.segmentation_image,
                                      bg="#2196F3", fg='white', font=('Helvetica', 10))
    self.save_button = Button(self.button_frame, text="Save Result", command=self.save_result_image, bg="#FFC107",
                              fg='black', font=('Helvetica', 10))
    self.reset_button = Button(self.button_frame, text="Restart Program", command=self.reset_image, bg="#607D8B",
                               fg='white', font=('Helvetica', 10))

    # Display canvas
    self.main_canvas.pack(side=LEFT)
    main_title_label.pack(side=LEFT, padx=(10, 220))
    self.mask_canvas.pack(side=LEFT)
    mask_title_label.pack(side=LEFT, padx=(10, 220))
    self.result_canvas.pack(side=LEFT)
    result_title_label.pack(side=LEFT, padx=(10, 220))

    # Display Buttons
    self.draw_rect_button.pack(side=LEFT, expand=YES, fill=BOTH, padx=5, pady=(10, 10))
    self.segmentation_button.pack(side=LEFT, expand=YES, fill=BOTH, padx=5, pady=(10, 10))
    self.save_button.pack(side=LEFT, expand=YES, fill=BOTH, padx=5, pady=(10, 10))
    self.reset_button.pack(side=LEFT, expand=YES, fill=BOTH, padx=5, pady=(10, 10))

def setSizeImg(self):
    global new_width, aspect_ratio, new_height
    # set ukuran gambar
    new_width = 320 # Atur ukuran hanya 480 px
    aspect_ratio = self.image.width / self.image.height
    new_height = int(new_width / aspect_ratio)

def drawing_brush(self):
    print("mulai gambar brush")

def drawing_rectangle(self):
    print("mulai gambar kotak")
    self.main_canvas.bind("<ButtonPress-1>", self.onClick_rect)
    self.main_canvas.bind("<B1-Motion>", self.onDrag_rect)
    self.main_canvas.bind("<ButtonRelease-1>", self.onRelease_rect)

def onClick_rect(self, event):
    print("Keberadaan kotak: ", KOTAK["is_drawn"])
    if KOTAK["is_drawn"] is not True:
        KOTAK["titik_start"] = (event.x, event.y)
    else:
        print("kotak sudah ada")

def onDrag_rect(self, event):

```

```

if KOTAK["is_drawn"] is not True:
    KOTAK["titik_akhir"] = (event.x, event.y)
    self.update_image()

def onRelease_rect(self, event):
    if KOTAK["titik_start"] and KOTAK["titik_akhir"]:
        KOTAK["coord"] = (self.get_rectangle_coords())
        print("Koordinat Kotak: ", KOTAK["coord"])
        KOTAK["titik_start"] = None
        KOTAK["titik_akhir"] = None
        KOTAK["is_drawn"] = True
    print("Keberadaan kotak: ", KOTAK["is_drawn"])
    print("Koordinat Kotak: ", KOTAK["coord"])

def update_image(self):
    # Update the main canvas with image and annotations
    self.image = Image.open(self.image_path)
    self.image = self.image.resize((new_width, new_height)) # <-- untuk resize ukuran
    self.image2 = self.image.copy()
    self.draw = ImageDraw.Draw(self.image)
    for coords in KOTAK["coord"]:
        self.draw.rectangle(coords, outline="yellow", width=2)
    self.draw.rectangle(self.get_rectangle_coords(), outline="yellow", width=2)
    self.photo = ImageTk.PhotoImage(self.image)
    self.main_canvas.create_image(0, 0, anchor=NW, image=self.photo)

def update_result(self):
    # Update mask canvas with segmentation
    self.mask_image = Image.fromarray(self.mask)
    self.mask_image = self.mask_image.resize((new_width, new_height)) # <-- untuk resize ukuran
    self.mask_photo = ImageTk.PhotoImage(self.mask_image)
    self.mask_canvas.create_image(0, 0, anchor=NW, image=self.mask_photo)

    # Update image with segmentation
    self.result_image = Image.composite(self.image2, Image.new("RGB", self.image.size, "black"), self.mask_image)
    self.result_photo = ImageTk.PhotoImage(self.result_image)
    self.result_canvas.create_image(0, 0, anchor=NW, image=self.result_photo)

def get_rectangle_coords(self):
    if KOTAK["titik_start"] and KOTAK["titik_akhir"]:
        x1, y1 = KOTAK["titik_start"]
        x2, y2 = KOTAK["titik_akhir"]
        return (x1, y1, x2, y2)
    else:
        return None

def segmentation_image(self):
    gc = GrabCut(self.gambar2, self.mask2, KOTAK["coord"])
    self.mask = np.where((self.mask2 == F_FG) | (self.mask2 == F_PR_FG), 255, 0).astype('uint8')
    self.update_result()

def save_result_image(self):
    save_folder = f"results/{self.category}"

    # Create the folder if it doesn't exist
    os.makedirs(save_folder, exist_ok=True)

    result_filename = f"result_{self.image_name}.jpg"
    main_rect_filename = f"image_{self.image_name}_rect.jpg"

    self.result_image.save(os.path.join(save_folder, result_filename))
    self.image.save(os.path.join(save_folder, main_rect_filename))

    print("Result image saved successfully:")

def reset_image(self):
    # Reset image and mask to initial state
    self.image = Image.open(self.image_path)
    self.image = self.image.resize((new_width, new_height)) # <-- untuk resize ukuran
    self.gambar = np.array(self.image)
    self.gambar2 = self.gambar.copy()
    self.mask = np.zeros(self.gambar2.shape[:2], dtype=np.uint8)
    self.mask2 = self.mask.copy()

    # Reset annotation variables
    KOTAK["coord"] = ()
    KOTAK["is_init"] = False
    KOTAK["is_drawn"] = False

    # Reload main canvas
    self.photo = ImageTk.PhotoImage(self.image)
    self.main_canvas.create_image(0, 0, anchor=NW, image=self.photo)

    # Clear the mask canvas

```

```

    self.mask_image = Image.fromarray(self.mask)
    self.mask_image = self.mask_image.resize((new_width, new_height)) # <-- untuk resize ukuran
    self.mask_photo = ImageTk.PhotoImage(self.mask_image)
    self.mask_canvas.create_image(0, 0, anchor=NW, image=self.mask_photo)

    # Clear the result canvas
    self.result_image = Image.new("RGB", (self.image.width, self.image.height))
    self.result_photo = ImageTk.PhotoImage(self.result_image)
    self.result_canvas.create_image(0, 0, anchor=NW, image=self.result_photo)

```

1.3 grabcut.py

```

import numpy as np
from GMM import MixtureModel
import igraph as ig
from mincut import mincut_segmentation

COLOR = {
    'red' : [0, 0, 255],
    'white' : [255, 255, 255],
    'black' : [0, 0, 0],
    'yellow' : [0, 255, 255]
}

F_TB = 0
F_TF = 1

```

1.4 GMM.py

```

import numpy as np
import random

class MixtureModel:
    def __init__(self, alpha, z, komponen_gmm, theta):
        self.komponen_gmm = komponen_gmm
        self.n_channels = z.shape[1]
        self.n_samples = np.zeros(self.komponen_gmm)

        self.theta = theta
        self.F_TB = 0
        self.F_TF = 1

    def init_gmm_rand(self, z, theta):
        labels_k = []
        for i in range(z.shape[0]):
            labels_k.append(random.randint(0, self.komponen_gmm-1))
        labels_k = np.array(labels_k)
        self.count_params(z, labels_k, theta)

    def count_params(self, z, labels_k, theta):
        self.n_samples[:] = 0
        theta['koefisien'][:] = 0

        variables_k, count = np.unique(labels_k, return_counts=True)
        self.n_samples[variables_k] = count

        for k in variables_k:
            n_k = self.n_samples[k]

            theta['koefisien'][k] = n_k / np.sum(self.n_samples)
            theta['means'][k] = np.mean(z[k == labels_k], axis = 0)
            theta['kovarians'][k] = np.cov(z[k == labels_k].T)

    def assign_component(self, z, theta, komponen_gmm):
        print('target assign: \n', z.shape)
        gauss_distribution = []

        for k in range(komponen_gmm):
            gauss_score = self.dis_mult(z, k, theta)
            gauss_distribution.append(gauss_score)

        gauss_distribution = np.array(gauss_distribution)
        gauss_distribution = gauss_distribution.T

    return np.argmin(gauss_distribution, axis=1)

```

```

def dis_mult(self, z, k, theta):
    result = np.zeros(z.shape[0])
    if theta['koefisien'][k] > 0 :
        diff = z - theta['means'][k]
        inv_covariance = np.linalg.inv(theta['kovarians'][k])
        mult = np.einsum('ij,ij->i', diff, np.dot(inv_covariance, diff.T).T)
        result = np.exp(-.5 * mult) / (np.sqrt(2 * np.pi) * np.sqrt(np.linalg.det(theta['kovarians'][k])))
    return result

def gauss_dist(self, zn, kn, theta):
    diff = zn - theta['means'][kn]
    inv_covariance = np.linalg.inv(theta['kovarians'][kn])
    mult = np.einsum('ij,ij->i', diff, np.dot(inv_covariance, diff.T).T)
    result = np.exp(-.5 * mult) / (np.sqrt(2 * np.pi) * np.sqrt(np.linalg.det(theta['kovarians'][kn])))
    return result

def gauss_dist_second(self, zn, kn, theta):
    diff = zn - theta['means'][kn]
    inv_covariance = np.linalg.inv(theta['kovarians'][kn])
    mult = np.sum(diff * np.dot(diff, inv_covariance.T), axis=1)
    result = 0.5 * (np.log(np.linalg.det(theta['kovarians'][kn])) + 0.5 * mult)
    return result

def count_D_formula(self, z, komponen_gmm, theta):
    gauss_distribution = []
    for k in range(komponen_gmm):
        tmp_gauss_distribution = self.dis_mult(z, k, theta)
        gauss_distribution.append(tmp_gauss_distribution)
    gauss_distribution = np.array(gauss_distribution)
    print("gauss shape: ", gauss_distribution.shape)

    return -np.log(np.dot(self.theta['koefisien'], gauss_distribution))

def d_calc(self, zn, kn, theta):
    gauss_res1 = self.gauss_dist_second(zn, kn, theta)
    d_res1 = -np.log(theta['koefisien'][kn]) + gauss_res1
    return d_res1

```

LAMPIRAN B

Tabel pengolahan data citra input

Tabel 2.1: Visualisasi hasil pengolahan data citra input luka hitam

No	File	Citra	resolusi
1	luka_hitam/2.jpg		375x292
2	luka_hitam/4.jpg		295x292
3	luka_hitam/5.jpg		512x283
4	luka_hitam/6.jpg		350x263
5	luka_hitam/7.jpg		150x115
6	luka_hitam/8.jpg		350x263
7	luka_hitam/14.jpg		275x183
8	luka_hitam/15.jpg		512x308
9	luka_hitam/17.jpg		283x247

Tabel 2.2: Visualisasi hasil pengolahan data citra input luka hitam - lanjutan

No	File	Citra	resolusi
10	luka_hitam/18.jpg		168x168
11	luka_hitam/19.jpg		512x374
12	luka_hitam/20.jpg		512x397
13	luka_hitam/22.jpg		390x276
14	luka_hitam/26.jpg		512x395
15	luka_hitam/27.jpg		512x415
16	luka_hitam/28.jpg		234x216
17	luka_hitam/29.jpg		512x395
18	luka_hitam/33.jpg		375x250

Tabel 2.3: Visualisasi hasil pengolahan data citra input luka hitam - lanjutan

No	File	Citra	resolusi
19	luka_hitam/37.jpg		350x263
20	luka_hitam/40.jpg		262x192
21	luka_hitam/41.jpg		361x270
22	luka_hitam/16.jpg		512x418
23	luka_hitam/31.jpg		383x512
24	luka_hitam/39.jpg		512x450

Tabel 2.4: Visualisasi hasil pengolahan data citra input luka kuning

No	File	Citra	resolusi
1	luka_kuning/13.jpg		266x189
2	luka_kuning/17.jpg		259x194
3	luka_kuning/18.jpg		265x190
4	luka_kuning/19.jpg		120x151
5	luka_kuning/21.jpg		357x242
6	luka_kuning/23.jpg		500x333
7	luka_kuning/25.jpg		288x216
8	luka_kuning/34.jpg		500x385
9	luka_kuning/35.jpg		242x208
10	luka_kuning/38.jpg		512x367

Tabel 2.5: Visualisasi hasil pengolahan data citra input luka kuning - lanjutan

No	File	Citra	resolusi
11	luka_kuning/42.jpg		512x384
12	luka_kuning/3.jpg		220x157
13	luka_kuning/12.jpg		500x333
14	luka_kuning/10.jpg		276x183
15	luka_kuning/16.jpg		346x270

Tabel 2.6: Visualisasi hasil pengolahan data citra input luka merah

No	File	Citra	resolusi
1	luka_merah/16.jpg		309x231
2	luka_merah/17.jpg		328x218
3	luka_merah/22.jpg		512x396
4	luka_merah/24.jpg		302x308
5	luka_merah/25.jpg		512x507
6	luka_merah/30.jpg		260x194
7	luka_merah/32.jpg		236x178
8	luka_merah/33.jpg		226x223
9	luka_merah/37.jpg		260x194
10	luka_merah/39.jpg		185x139

Tabel 2.7: Visualisasi hasil pengolahan data citra input luka merah - lanjutan

No	File	Citra	resolusi
11	luka_merah/42.jpg		350x354
12	luka_merah/44.jpg		512x363
13	luka_merah/2.jpg		288x512
14	luka_merah/3.jpg		384x512
15	luka_merah/4.jpg		384x512
16	luka_merah/6.jpg		512x384
17	luka_merah/7.jpg		512x341
18	luka_merah/8.jpg		512x384

Tabel 2.8: Visualisasi hasil pengolahan data citra input luka merah - lanjutan

No	File	Citra	resolusi
19	luka_merah/9.jpg		288x512
20	luka_merah/10.jpg		512x384
21	luka_merah/11.jpg		512x384
22	luka_merah/12.jpg		512x384
23	luka_merah/14.jpg		512x384
24	luka_merah/18.jpg		512x382
25	luka_merah/19.jpg		512x458
26	luka_merah/20.jpg		408x360

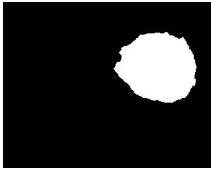
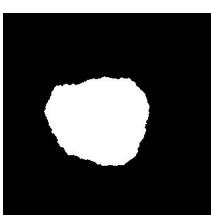
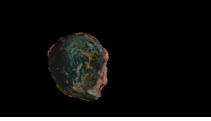
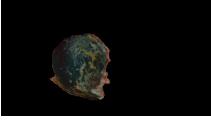
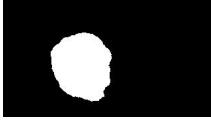
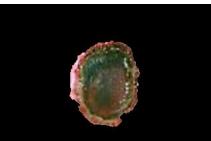
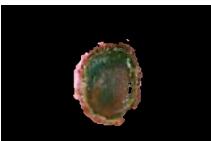
Tabel 2.9: Visualisasi hasil pengolahan data citra input luka merah - lanjutan

No	File	Citra	resolusi
27	luka_merah/23.jpg		512x384
28	luka_merah/26.jpg		512x464
29	luka_merah/29.jpg		328x218
29	luka_merah/35.jpg		261x295
30	luka_merah/36.jpg		295x194
31	luka_merah/38.jpg		512x384
32	luka_merah/20.jpg		408x360

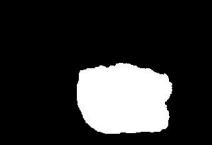
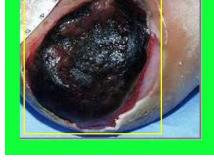
LAMPIRAN C

Tabel hasil deteksi keliling luka menggunakan *Grabcut*

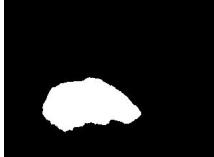
Tabel 3.1: Visualisasi hasil deteksi keliling luka menggunakan *Grabcut*

Inisialisasi Bounding Box	Hasil Segmentasi	Hasil Citra Referensi	Area Segmentasi	Akurasi
 (153,48,299,158)				98,1 %
 (60, 89, 162, 153)				93.3 %
 (68, 48, 115, 106)				98.0 %
 (83, 97, 130, 97)				99,6 %
 (100, 50, 117, 142)				99,4 %

Tabel 3.2: Visualisasi hasil deteksi keliling luka menggunakan *Grabcut*

Inisialisasi Bounding Box	Hasil Segmentasi	Hasil Citra Referensi	Area Segmentasi	Akurasi
 (114, 104, 157, 108)				64,4 %
 (98, 43, 116, 121)				90,6 %
 (89, 10, 140, 171)				95,7 %
 (28, 34, 208, 213)				90,3 %
 (50, 59, 219, 201)				88,1 %

Tabel 3.3: Visualisasi hasil deteksi keliling luka menggunakan *Grabcut*

Inisialisasi Bounding Box	Hasil Segmentasi	Hasil Citra Referensi	Area Segmentasi	Akurasi
 (107, 82, 147, 104)				97.6 %
 (51, 107, 168, 101)				82,9 %
 (63, 67, 186, 98)				85,2 %