DETEKSI TEPI DANAU PADA CITRA SATELIT MENGGUNAKAN METODE CANNY

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Koko Pradityo
NIM: 105060803111008

TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2017

PENGESAHAN

DETEKSI TEPI DANAU PADA CITRA SATELIT MENGGUNAKAN METODE CANNY

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Koko Pradityo
NIM: 105060803111008

Skripsi ini telah diuji dan dinyatakan lulus pada
13 April 2017
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Budi Darma Setiawan, S.Kom, M.Cs	Randy Cahya W. S.ST, M.Kom
NIP. 198410152014041002	NIK. 201405 880206 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP. 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 22 Maret 2017

Koko Pradityo

NIM: 105060803111008

KATA PENGANTAR

Puji syukur penulis sampaikan kepada Allah SWT, karena dengan rahmat dan hidayah-Nya penulis diberikan kemudahan untuk menyelesaikan penelitian yang berjudul "Deteksi Tepi Danau pada Citra Satelit Menggunakan Metode Canny".

Penulisan skripsi ini diajukan untuk memenuhi sebagian persyaratan untuk memperoleh gelar sarjana komputer di program studi Teknik Informatika Universitas Brawijaya Malang. Penyusunan skripsi ini tentu tidak dapat berjalan lancar tanpa bantuan dari banyak pihak. Dalam kesempatan ini penulis ingin menyampaikan rasa terimakasih kepada:

1. Tuhan Yang Maha Esa karena hanya dengan rahmat dan hidayah-Nya skripsi ini dapat terwujud.
2. Bapak Budi Darma Setiawan S.Kom, M.Cs., selaku dosen pembimbing I serta bapak Randy Cahya W. S.ST, M.Kom., selaku dosen pembimbing II atas segala masukan dan bimbingan yang telah diberikan selama proses pengerjaan penelitian ini.
3. Ayah dan Ibu penulis yang telah dengan penuh kesabaran memberikan dukungan, doa, serta didikannya juga keluarga penulis yang telah banyak memberi berbagai bentuk dukungan lainnya demi lancarnya penyusunan skripsi ini.
4. Reza Dwi Rahmadian S.Kom., Ramdhani Bima Arista S.Kom., Usfita Kiftiyani S.Kom., Fitria Rozyana A. S.Kom., serta teman-teman penulis lainnya yang bergabung dalam grup Koala atas dukungan, bantuan, ide-ide serta kebersamaan selama masa perkuliahan.
5. Florentinus Ardhanareswara, Onggeru Swastika, dan Stephanie, sebagai teman penulis bertukar ide dan melakukan dialog akademis yang secara langsung dan tidak langsung membantu proses penyusunan skripsi ini.
6. Segala pihak yang telah memberikan bantuan dalam proses pengerjaan skripsi ini dan tidak dapat disebutkan satu-persatu.

Penulis menyadari bahwa skripsi ini tidak sempurna, dan oleh karena itu saran dan kritik yang membangun akan penulis terima dengan baik. Akhir kata, penulis berharap skripsi yang telah disusun ini dapat berguna bagi masyarakat.

Malang, 22 Maret 2017

Penulis
kk.pradityo@gmail.com

ABSTRAK

Danau adalah sebuah fitur darat yang berperan penting dalam kehidupan manusia. Perubahan pada danau dapat memengaruhi keadaan lingkungan sekitar serta kehidupan masyarakat yang berada disekitarnya. Salah satu cara untuk mengetahui perubahan kondisi danau dilakukan dengan melakukan deteksi tepi permukaan danau pada citra satelit untuk mengetahui perubahan luas danau tersebut. Penerapan algoritme yang tepat dan optimasi lebih lanjut dapat membantu analisis keadaan danau. Aplikasi menerapkan algoritme deteksi tepi Canny untuk melakukan deteksi tepi permukaan danau pada citra satelit. Algoritme segmentasi berdasarkan metode *color thresholding* diterapkan untuk melakukan optimasi pada kinerja deteksi tepi. Hasil pengujian menunjukkan bahwa deteksi tepi pada citra satelit dengan hanya menggunakan metode Canny menghasilkan *error rate* sebesar 57%, sementara proses segmentasi dengan *color thresholding* meningkatkan kinerja deteksi tepi sebesar 67%.

Kata kunci: danau, citra satelit, deteksi tepi, metode Canny, segmentasi, *color thresholding*

ABSTRACT

Lake is an important land feature for human life. Changes in a lake's condition would affect the environment and the people living nearby. One of the method being used to detect changes in lake's condition is by using an edge detection of the lake based on satellite image for further analysis such as measuring the change in lake's total area. Appropriate implementation and optimization of such algorithm can lead to a better analysis of the lake's condition. In this research, the system implemented Canny Edge Detection algorithm to detect the edge of a lake on a satellite image. A segmentation algorithm based on color thresholding is used to improve the edge detection algorithm. The test result shows that Canny Edge Detection algorithm has 57% error detection, while segmentation process using color thresholding improves the detection performance by 67%.

***Keywords**:* Canny edge detection, lake, satellite image, segmentation, color thresholding.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

Pada studi ini, peneliti mengimplementasikan algoritme deteksi tepi Canny untuk melakukan deteksi tepi permukaan danau pada citra satelit. Penelitian ini dilaksanakan mengingat metode Canny merupakan sebuah metode deteksi tepi yang adaptif dan memerlukan parameter yang khusus untuk setiap kasus.

1.1 Latar belakang

Danau merupakan sebuah fitur daratan yang memiliki peran penting bagi kehidupan manusia serta ekosistem yang berada di sekitarnya. Beberapa fungsi danau bagi manusia adalah sumber air bersih maupun irigasi, sebagai pembangkit listrik tenaga air (PLTA) serta tempat untuk budidaya ikan dan sarana rekreasi warga. Perubahan kondisi danau dapat berpengaruh besar bagi komunitas yang hidup di sekitarnya. Sebagai contoh, turunnya volume air pada sebuah danau akan berpengaruh terhadap kelancaran air irigasi serta air bersih warga. Untuk itu, pemantauan terhadap kondisi danau merupakan suatu hal yang harus terus dilaksanakan. Salah satu cara untuk terus memantau kondisi danau adalah dengan menggunakan pencitraan satelit. Dengan bantuan citra satelit, pakar dapat melakukan analisis terhadap berbagai karakteristik danau, seperti luas area danau serta warna permukaan danau.

Dalam pengolahan citra satelit, danau merupakan salah satu objek yang cukup menarik untuk diteliti. Bentuk danau yang diskrit dan kontras antara permukaan danau dan permukaan tanah disekitarnya membuat danau menjadi sebuah objek deteksi tepi yang baik. Namun hingga sekarang, penelitian mengenai metode deteksi tepi Canny terhadap tepi danau masih belum banyak dilakukan.

Deteksi tepi merupakan sebuah teknik yang penting dalam bidang *Computer Vision*. Proses segmentasi dan pengenalan bentuk bergantung pada proses deteksi tepi. Jika tepi dari sebuah objek telah ditemukan, maka proses pengenalan benda dapat dengan mudah dilakukan, contohnya menentukan bentuk dan ukuran sebuah objek. Pada praktiknya, deteksi tepi sangat bergantung pada variabel-variabel seperti kondisi pencahayaan gambar, keberadaan objek dengan intensitas yang sama, ketebalan tepi, dan *noise* pada gambar. (Nedameejad, 2008)

Salah satu metode yang umum digunakan dalam deteksi tepi adalah metode Canny. Metode ini merupakan metode yang hingga kini diakui lebih optimal daripada metode-metode yang lainnya. (Jahagirdar, 2016). Dalam pemrosesan menggunakan metode ini, diperlukan beberapa parameter utama yaitu besarnya tepi Gaussian filter yang akan digunakan dan dua *threshold* (batasan) yang digunakan dalam proses penekanan. Hingga kini belum ada nilai yang disetujui oleh para peneliti untuk menentukan besaran kedua parameter tersebut, hingga nilainya sangat bergantung pada tipe gambar dan objek yang akan dikenali.

Proses deteksi tepi pada permukaan danau memiliki beberapa potensi manfaat yang signifikan, seperti mengetahui keliling dan luas suatu danau.

informasi tersebut dapat digunakan sebagai acuan untuk menentukan perubahan kondisi suatu danau dalam rentang waktu tertentu.

Dengan latar belakang demikian, peneliti menemukan pentingnya sebuah studi untuk melakukan implementasi metode Canny untuk mendeteksi tepi danau pada sebuah citra satelit.

1.2 Rumusan masalah

Rumusan masalah yang diteliti dalam studi ini adalah sebagai berikut:

1. Bagaimana penerapan metode Canny dalam deteksi tepi danau pada sebuah citra satelit?
2. Bagaimanakah kinerja algoritme Canny dalam deteksi tepian permukaan danau pada citra satelit?

1.3 Tujuan

Tujuan dari penelitian ini adalah:

1. Menerapkan metode Canny dalam deteksi tepi danau pada sebuah citra satelit
2. Mengetahui kinerja algoritme Canny dalam deteksi tepian permukaan danau pada citra satelit

1.4 Manfaat

Manfaat yang diharapkan dari penelitian ini adalah memudahkan peneliti selanjutnya dalam mendapatkan landasan referensi yang tepat untuk melakukan deteksi tepi danau menggunakan metode Canny.

1.5 Batasan masalah

Batasan masalah pada penelitian ini adalah:

1. Citra yang diolah merupakan citra satelit LANDSAT 4 dan 5 yang didapatkan dari situs USGS.
2. Objek yang dideteksi tepinya adalah danau yang berada dalam wilayah kedaulatan NKRI

1.6 Sistematika pembahasan

Penelitian ini memiliki struktur sebagai berikut:

1. Bab 1: Pendahuluan, berisi latar belakang permasalahan, serta tujuan, manfaat dan batasan penelitian yang dilakukan
2. Bab 2: Landasan Kepustakaan, berisi studi kepustakaan yang digunakan oleh peneliti.
3. Bab 3: Metodologi Penelitian, berisi penjelasan mengenai pelaksanaan penelitian, metode yang digunakan, serta algoritme yang digunakan dalam penelitian.

4. Bab 4: Implementasi, berisi implementasi algoritme dalam bentuk aplikasi.
5. Bab 5: Pengujian, berisi pengujian dan analisis mengenai hasil yang telah diperoleh dari implementasi algoritme yang telah dilakukan oleh peneliti.
6. Bab 6: Penutup, berisi ringkasan hasil penelitian dan saran penelitian lebih lanjut.

BAB 2 LANDASAN KEPUSTAKAAN

Dalam bab ini dibahas beberapa hal yang menjadi bahan penelitian, yaitu citra satelit, pengolahan citra, deteksi tepi, dan algoritme Canny yang digunakan dalam deteksi tepi citra satelit.

2.1 Kajian Pustaka

Berdasarkan penelitian yang dilaksanakan, penulis menemukan beberapa penelitian yang relevan untuk mendukung skripsi ini yaitu penelitian mengenai kinerja deteksi tepi dengan algoritme Canny serta penelitian mengenai segmentasi dengan menggunakan *thresholding* warna.

Penelitian yang membandingkan beberapa metode deteksi tepi dilakukan oleh Sharifi (2002) dengan membandingkan algoritme deteksi tepi ISEF, Canny, Sobel, Kirsch, Marr-Hilderth, Sobel serta Laplacian 1 dan 2. Dari penelitian yang telah dilaksanakan, dapat diketahui bahwa deteksi tepi menggunakan algoritme Canny memiliki kinerja yang lebih baik ketika gambar yang diolah memiliki banyak *noise*, dengan catatan komputasi deteksi tepi menjadi lebih kompleks karena melalui dua proses, yaitu Gaussian *filtering* dan penentuan gradien.

Liu H (2004) melakukan penelitian untuk mendeteksi tepian pantai menggunakan algoritme Canny dan metode *thresholding* lokal adaptif. Dari penelitian yang telah dilakukan, diketahui bahwa salah saktu faktor krusial dalam menjaga akurasi deteksi tepian pantai adalah dengan cara menerapkan segmentasi. Akurasi segmentasi yang dilakukan bergantung pada keandalan dan ketepatan nilai *threshold* yang didapatkan dengan mengolah kurva Gaussian bimodal.

Dalam penelitian yang dilakukan oleh Kulkarni (2014) diketahui bahwa segmentasi multi-level terhadap ketiga komponen warna (R, G, B) memiliki kinerja yang baik untuk kasus dengan citra masukan yang memiliki objek dengan perbedaan warna diskrit. Segmentasi dengan *thresholding* warna dapat menghilangkan *noise* secara langsung dengan menghilangkan objek yang tidak diinginkan dalam citra, sehingga hanya objek yang diinginkan yang diolah dalam proses selanjutnya.

2.2 Citra Satelit

Citra satelit adalah citra bumi yang dihasilkan oleh satelit yang mengorbit bumi. Citra satelit digunakan dalam berbagai bidang, contohnya meteorologi, geologi, kartografi, perencaan ruang, intelijen, serta militer. Dalam praktiknya, interpretasi dan analisis citra satelit dilakukan menggunakan aplikasi khusus penginderaan jauh.

2.2.1 Resolusi dan data citra satelit

Ada 4 tipe resolusi citra satelit, yaitu *spatial, spectral, temporal,* dan *radiometric* (Campbell, 2002).

1. Resolusi Spasial, adalah ukuran piksel citra yang merepresentasikan luas area permukaan (contoh: dalam m^2) yang diukur pada permukaan tanah. Resolusi ini ditentukan oleh *instantaneous field of view* (IFOV) yang dimiliki oleh sensor satelit pencitraan;
2. Resolusi Spektral, yang ditentukan oleh besar interval panjang gelombang (segmen diskrit dari spektrum elektromagnetik) dan jumlah interval yang diukur oleh sensor;
3. Resolusi Temporal, ditentukan oleh lama waktu yang dilewati diantara periode pengumpulan gambar pada suatu wilayah;
4. Resolusi Radiometris, yaitu kemampuan sistem pencitraan untuk merekam berbagai tingkat kecerahan (contohnya, kontras gambar) dan kedalaman bit efektif yang biasanya bernilai 8-bit (0-255), 11-bit (0-2047), 12-bit (0-4095) atau 16-bit (0-65,535).

2.3 Pengolahan Citra Digital

Pengolahan citra digital adalah penggunaan algoritme komputer untuk melakukan pengolahan pada citra digital. Sebagai subkategori dari pengolahan sinyal digital, maka pengolahan citra digital memiliki banyak keunggulan dibandingkan penolahan citra analog. Salah satunya adalah memungkinkan lebih banyak algoritme untuk diaplikasikan pada pengolahan citra serta menghindari permasalah *noise* serta distorsi sinyal selama pemrosesan.

Pada awalnya, banyak dari teknik pengolahan citra digital yang dikembangkan pada tahun 1960 di Jet Propulsion Laboratory, Massachusetts Institute of Technology, Bell Laboratories, University of Maryland, serta beberapa lab riset lainnya, dengan aplikasi pada pencitraan satelit, konversi standar telefotografi, pencitraan medis, *videophone*, pengenalan objek, dan perbaikan citra (Rosenfeld, 1969).

Pada khususnya, pengolahan citra digital merupakan satu-satunya teknologi yang secara praktikal dapat digunakan untuk klasifikasi, ekstraksi fitur, *Multi-scale signal analysis*, pengenalan pola, serta proyeksi citra.

2.3.1 Deteksi Tepi

Deteksi tepi adalah metode matematis yang digunakan untuk mengindentifikasi titik pada citra digital dimana kecerahan citra berubah tajam, atau secara formal, memiliki diskontinuitas. Titik-titik dimana kecerahan berubah tajam biasanya tersusun menjadi segmen garis lengkung yang membatasi sebuah tepian. Deteksi tepi adalah alat dasar dalam pengolahan citra digital, *machine vision* dan *computer vision*, khususnya pada bidang deteksi dan ekstraksi fitur (Umbaugh, 2010).

Pada sebuah kasus ideal, hasil dari aplikasi deteksi tepi adalah sebuah kurva tersambung yang mengindikasikan tepian objek, tepian penanda permukaan serta kurva yang menunjukkan diskontinuitas orientasi permukaan. Dengan demikian penerapan deteksi tepi diharapkan dapat mengurangi jumlah data yang harus diproses dan menyaring informasi yang dapat dianggap kurang relevan, sembari tetap menyimpan properti penting dari sebuah citra. Namun demikian, mendapatkan sebuah tepian dari citra dengan kompleksitas yang menengah masih tetap sulit dilakukan.

Ada beberapa metode deteksi tepi yang umum digunakan, namun pada dasarnya terbagi menjadi dua kategori:

1. *Search-based:* kategori ini mendeteksi tepi dengan cara menghitung besaran kekuatan tepi (biasanya merupakan ekspresi derivatif pertama seperti besaran gradien), lalu melakukan pencarian *maxima* lokal pada besaran gradien tersebut menggunakan perkiraan orientasi tepi lokal, yang biasanya adalah arah gradien. Contoh metode yang tergolong dalam kategori ini adalah operator Sobel, Robert, Prewitt, dan Canny.
2. *Zero-crossing*: kategori ini menggunakan derivatif kedua dalam intensitas gradien, sehingga pada dasarnya dapat mengenali tingkat perubahan intesitas gradien. Pada kondisi ideal, pencarian *zero-crossing* pada derivatif kedua akan menghasilkan nilai *maxima* lokal pada gradien. Metode yang tergolong dalam kategori ini contohnya adalah operator Marr-Hilderth.

2.3.2 *Color Thresholding*

Color Thresholding atau *thresholding* warna merupakan salah satu metode segmentasi citra. Segmentasi citra adalah salah satu langkah penting dalam pengolahan citra digital, dan penerapannya meliputi berbagai bidang seperti pengenalan objek, analisis pencitraan medis, serta pengolahan video.

Pada umumnya segmentasi dilakukan dengan terlebih dahulu mengubah citra yang akan diolah kedalam bentuk *grayscale*, lalu menerapkan metode segmentasi yang diinginkan pada citra tersebut. Pada kasus deteksi tepi danau pada citra satelit, warna permukaan danau dengan daratan cenderung memiliki perbedaan yang diskrit, sehingga segmentasi dengan menggunakan metode *thresholding* warna sebelum melakukan deteksi tepi dapat membantu menghilangkan *noise* yang terdeteksi sebagai tepi.

Untuk melakukan *thresholding* warna pada citra, dapat digunakan persamaan (2.1) (Kulkarni, 2014)

$$g(x,y) = \begin{cases} f(x,y), 0 \le red(x,y) \le tR, \\ 0, \quad red(x,y) > tR. \end{cases}$$

$$g(x,y) = \begin{cases} f(x,y), tG \le green(x,y) \le 255, \\ 0, \quad green(x,y) \le tG. \end{cases} \quad (2.1)$$

$$g(x,y) = \begin{cases} f(x,y), 0 \le blue(x,y) \le tB, \\ 0, \quad blue(x,y) > tB. \end{cases}$$

Penjelasan variabel:

x,y = koordinat piksel citra yang mengalami pengecekan.

tG, tR, tB = nilai *threshold* untuk tiap komponen warna; berturut-turut untuk komponen warna hijau, merah, dan biru.

2.4 Metode Canny

Detektor tepi Canny adalah sebuah operator deteksi tepi yang menggunakan algoritme banyak-tingkat untuk mendeteksi berbagai macam tepian pada gambar. Algoritme ini dikembangkan oleh John F. Canny pada tahun 1986. Canny menemukan bahwa persyaratan untuk aplikasi deteksi tepi pada sistem visi adalah relatif sama. Dengan demikian, solusi deteksi tepi untuk mengatasi kebutuhan tersebut dapat diimplementasikan dalam berbagai situasi. Kriteria umum untuk deteksi tepi meliputi (Canny, 1986):

1. Deteksi tepi dengan tingkat kesalahan yang rendah, yang berarti bahwa deteksi harus secara akurat menangkap sebanyak mungkin tepi yang ditunjukkan pada gambar.
2. Titik tepi terdeteksi dari operator harus secara akurat melokalisasi di tengah-tengah tepian.
3. Tepi yang diberikan dalam gambar hanya harus ditandai sekali, dan jika mungkin, *noise* gambar seharusnya tidak lagi menciptakan tepi palsu.

Untuk memenuhi persyaratan ini Canny menggunakan kalkulus variasi. Fungsi optimal dalam detektor Canny digambarkan oleh jumlah dari empat pernyataan eksponensial, tetapi dapat juga diekspresikan dengan turunan pertama dari Gaussian.

2.4.1 Langkah-langkah algoritme deteksi tepi Canny

Proses deteksi tepi Canny dapat dijelaskan dalam 5 langkah:

1. Penerapan filter Gaussian.

   Langkah ini dilakukan untuk menghaluskan gambar sehingga dapat menghilangkan *noise* pada citra. Filter Gaussian yang umum digunakan adalah persamaan (2.2):

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2}\right); 1 \le i, j \le (2k+1) \quad (2.2)$$

   Penjelasan variabel:

   i, j = koordinat piksel yang diperiksa

   k = ukuran kernel gaussian

   σ = sigma (deviasi standar)

Hal yang patut dipahami adalah besaran Gaussian *kernel* menentukan kinerja deteksi tepi.

2. Menemukan intensitas gradien citra

   Algoritme Canny menggunakan filter untuk mendeteksi tepian horizontal, vertikal, dan diagonal pada sebuah citra. Pada operator deteksi tepi, nilai kembalian merupakan derivatif pertama dalam bentuk arah horizontal (G_x) dan vertikal (G_y). Maka tepi dan arah gradien dapat ditentukan dengan persamaan (2.3):

$$\mathbf{G} = \sqrt{\mathbf{G}_x^{\,2} + \mathbf{G}_y^{\,2}}$$
$$\mathbf{\Theta} = \operatorname{atan2}(\mathbf{G}_y, \mathbf{G}_x), \tag{2.3}$$

   Penjelasan variabel:

   G = Besaran gradien

   G_x = Besaran gradien untuk komponen horizontal

   G_y = Besaran gradien untuk komponen vertikal

   Θ = Nilai arah gradien

3. Menerapkan penekanan *non-maximum* untuk menghilangkan tepian palsu yang terdeteksi.

   Langkah ini digunakan untuk menipiskan tepian yang telah terdeteksi dengan cara membandingkan kekuatan tepian piksel yang sedang diperiksa dengan piksel positif dan negatif dari gradien arah.

4. Menerapkan *threshold* ganda untuk menemukan kandidat tepian.

   Threshold yang digunakan dalam algoritme ini digunakan untuk menentukan apakah sebuah tepi diklasifikasikan sebagai sebuah tepian lemah atau kuat. Apabila nilai gradien sebuah piksel lebih besar daripada *threshold* atas, maka piksel tersebut diklasifikasikan sebagai tepi kuat. Apabila nilai gradien piksel berada diantara nilai *threshold* atas dan *threshold* bawah, maka piksel tersebut merupakan sebuah tepian lemah. Kedua nilai ini berpengaruh pada kinerja deteksi tepi dan hingga kini harus ditentukan secara empiris.

5. Melacak tepian menggunakan *hysteresis*.

   Proses ini dilakukan untuk menjaga tepian agar kontinu. Tepian lemah yang terdeteksi dan bertetanggaan secara langsung dengan sebuah tepian kuat akan ditandai sebagai tepian kuat.

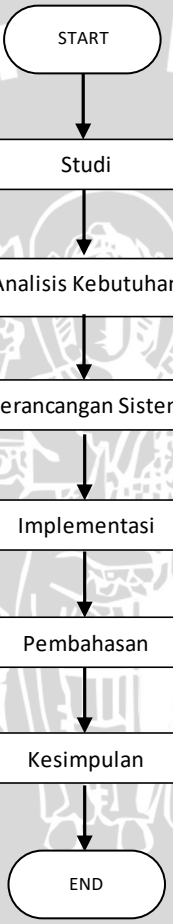
2.5 *Ground Truth*

Dalam konteks penginderaan jarak jauh (*remote sensing) Ground Truth* mengacu pada informasi yang didapatkan langsung dari lapangan. *Ground truth* dapat membantu proses pencocokan data citra dengan fitur nyata dan material yang terdapat pada lapangan. Pengambilan data *ground-truth* dapat digunakan untuk melakukan kalibrasi terhadap data penginderaan jarak jauh, serta membantu interpretasi dan analisis terhadap fitur yang sedang diperiksa. Secara spesifik, *ground truth* mengacu pada proses dimana sebuah piksel dari sebuah citra satelit dibandingkan dengan fitur nyata yang terdapat di lapangan untuk memverifikasi konten dari piksel tersebut.

Menurut Jianhong (2011), terdapat dua cara untuk menghasilkan *ground truth.* Cara pertama adalah dengan menggunakan instrumen dengan akurasi tinggi untuk mendapatkan pengukuran yang tepat. Cara kedua adalah dengan pembuatan data secara manual oleh pakar pada kasus dimana hasil penginderaan tidak dapat diukur oleh alat secara akurat. Pada penelitian ini, citra *ground truth* yang dibuat secara manual oleh peneliti digunakan sebagai acuan sebagai pembanding yang digunakan untuk menguji akurasi citra keluaran dari aplikasi.

BAB 3 METODOLOGI

Bab ini menjelaskan langkah-langkah yang ditempuh pada penyusunan skripsi, diantaranya meliputi perancangan dan pengujian dari aplikasi perangkat lunak yang dibuat. Penelitian ini dilakukan dengan metode non-implementatif analitik/deskriptif, dengan fokus pada analisis pengaruh parameter pada metode yang digunakan serta kemungkinan optimasi algoritme yang digunakan. Kesimpulan dan saran disertakan sebagai catatan dari aplikasi dan kemungkinan arah pengembangan aplikasi selanjutnya. Tahapan-tahapan perancangan hingga kesimpulan dilaksanakan menurut Gambar 3.1.



Gambar 3.1 Diagram alir Metode penelitian

3.1 Studi Kepustakaan

Pada bagian studi kepustakaan ini, peneliti menjabarkan berbagai referensi yang berkaitan dengan penelitian peneliti yang berhubungan dengan:

- Citra satelit
- Pengolahan Citra digital

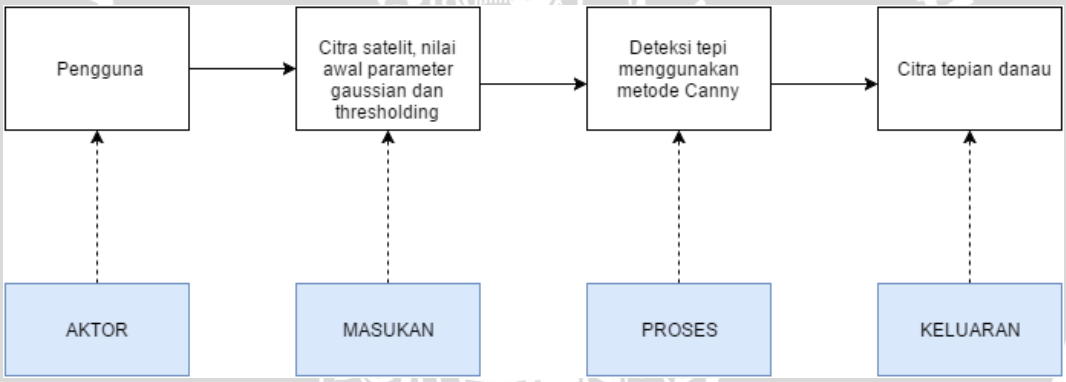
- Deteksi tepi
- Deteksi Tepi dengan menggunakan algoritme Canny

3.2 Pengumpulan data

Citra satelit yang diolah merupakan citra satelit LANDSAT 4 dan 5 yang diperoleh melalui situs USGS (http://earthexplorer.usgs.gov). Citra danau yang diolah merupakan danau yang berada di wilayah NKRI. Citra satelit yang telah didapatkan diubah kedalam format RGB untuk dapat diolah oleh aplikasi. Data citra yang diambil sebanyak 15 gambar, dengan dataset citra satelit LANDSAT 4-5 TM C1 Level-1 dengan format TIFF. Resolusi citra sebelum dilakukan *pre-processing* adalah 8031 x 6931 piksel dengan kedalaman warna 8-bit.

3.3 Perancangan Sistem

Perancangan sistem didasarkan pada analisis kebutuhan dan data yang telah diperoleh. Fokus utama pada sistem adalah kemudahan *input* parameter serta kejelasan hasil *output* yang dilakukan sehingga dapat dianalisis oleh peneliti.



Gambar 3.2 Gambaran Umum Aplikasi

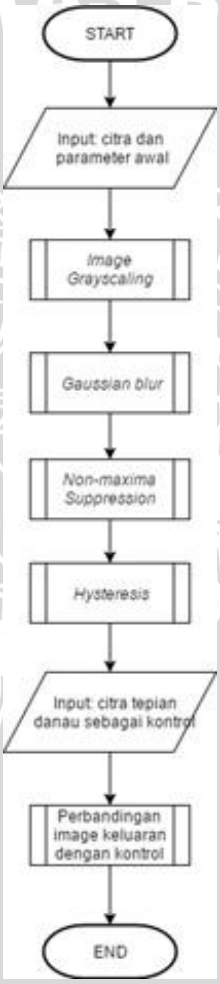
Berdasarkan Gambar 3.2, langkah-langkah yang dilakukan dalam aplikasi adalah sebagai berikut:

1. Aktor atau pengguna melakukan *input* data berupa citra satelit yang telah didapatkan dan dilakukan *preprocessing* sehingga berupa gambar dengan ekstensi file jpeg.
2. Citra yang dimasukkan oleh pengguna diubah menjadi citra *grayscale* oleh aplikasi sebelum melakukan pemrosesan deteksi tepi dengan metode Canny.
3. Pengguna memasukkan nilai awal parameter Gaussian *kernel* dan kedua nilai *threshold* yang dingiinkan dalam pemrosesan deteksi tepi.
4. Aplikasi melakukan deteksi tepi pada citra serta menampilkan hasil keluaran dari langkah-langkah dalam metode Canny yang digunakan, yaitu olahan gambar setelah diterapkan Gaussian *filtering*, citra setelah

dilakukan tahap penekanan *non-maximum*, serta citra keluaran setelah dilakukan tahap *hysteresis*.

5. Pengguna kemudian melakukan optimasi nilai parameter Gaussian *kernel* dan *threshold* yang digunakan untuk mendapatkan citra yang lebih baik.
6. Untuk tiap perubahan nilai parameter yang dilakukan oleh pengguna, aplikasi mengulang pemrosesan deteksi tepi dengan metode Canny.
7. Hasil keluaran yang dihasilkan oleh aplikasi dapat dibandingkan oleh pengguna dengan gambar tepian danau yang digunakan sebagai kontrol untuk menentukan nilai yang optimal.

Alur kerja aplikasi dapat digambarkan dengan menggunakan diagram alir pada Gambar 3.3:



Gambar 3.3 Diagram alir aplikasi deteksi tepi menggunakan metode Canny

3.3.1. Image Grayscaling

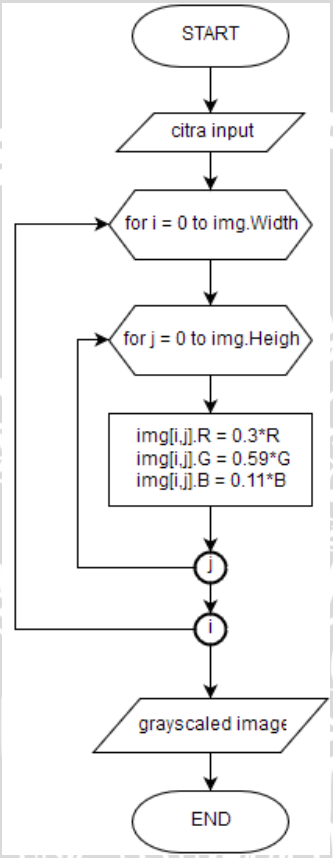
Berdasarkan Gambar 3.3, proses yang pertama kali dilakukan sebagai prasyarat deteksi tepi menggunakan metode Canny adalah mengubah citra *input* dari citra warna menjadi citra *grayscale*. Pengubahan citra menjadi *grayscale*

dilakukan dengan tujuan menampilkan kontras pada citra terutama di bagian antara *foreground* dan background. Pada aplikasi ini, pengubahan citra menjadi *grayscale* dilakukan dengan mengacu pada standar *Luminance*, dan dapat dilakukan mengalikan nilai RGB tiap piksel dengan Persamaan (3.1).

$$(0.3) * R, (0.59) * G, (0.11) * B \quad (3.1)$$

Dengan R, G, dan B adalah nilai komponen warna pada piksel yang relevan untuk masing-masing komponen warna merah, hijau, dan biru.

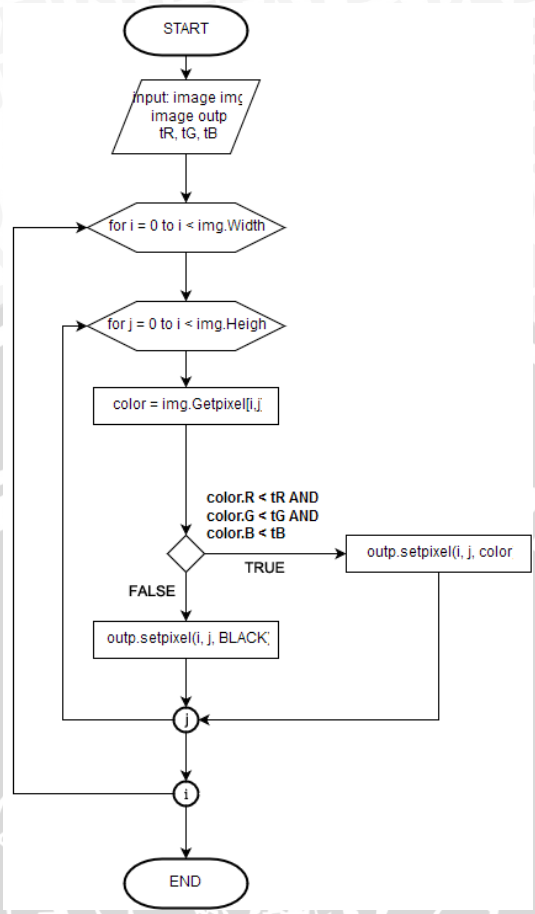
Diagram alir proses *grayscaling* citra digambarkan dengan Gambar 3.4.



Gambar 3.4 Diagram alir proses *image grayscaling*

3.3.2. *Color Thresholding*

Thresholding warna diterapkan pada aplikasi ini untuk membantu segmentasi tepi danau pada citra satelit. Nilai *threshold* untuk tiap-tiap komponen warna merah (*red*), hijau (*green*) dan biru (*blue*) merupakan parameter masukan dari pengguna. Pengguna dapat mengetahui perkiraan nilai tersebut dari histogram untuk tiap-tiap komponen warna dari citra masukan. Pada histogram akan nampak lembah yang merupakan batasan atau *threshold* yang dapat digunakan pengguna sebagai masukan parameter *threshold* tiap-tiap channel warna. Implementasi dari langkah ini dapat digambarkan dengan diagram alir seperti pada Gambar 3.5.



Gambar 3.5 Diagram alir proses *color thresholding*

3.3.3. Deteksi Tepi dengan Metode Canny

Algoritme Canny terdiri dari beberapa langkah utama yaitu, Gaussian *Filtering* (*blurring*), mencari intensitas dan besaran gradien, penekanan *non-maximum, dual-thresholding,* dan *hysteresis* untuk citra masukan (*input*).

3.3.3.1. Gaussian *Filtering*

Pada langkah ini, citra yang telah dijadikan *grayscale* diproses melalui proses *blurring* menggunakan *kernel* Gaussian yang parameternya ditentukan oleh pengguna. Tujuan dari langkah ini adalah untuk menghaluskan fitur-fitur pada citra sehingga hanya tepian yang kontras yang akan diproses pada langkah selanjutnya. Langkah kerja dari Gaussian *filtering* adalah sebagai berikut:

1. Tentukan parameter Gaussian, yaitu ukuran *kernel* Gaussian dan nilai sigma (standar deviasi) yang diinginkan.
2. Implementasikan persamaan (2.1) pada *kernel* horizontal dan vertikal untuk *blurring*. Aplikasi mampu menciptakan ukuran *kernel* horizontal dan vertikal dengan ukuran sesuai dengan parameter yang ditentukan oleh pengguna.

Sebagai contoh, Gambar 3.6 adalah *kernel* horizontal dan vertikal dengan ukuran *kernel* 3.

-1	0	1
-1	0	1
-1	0	1

-1	-1	-1
0	0	0
1	1	1

Gambar 3.6 *Kernel* Horizontal dan Vertikal Ukuran 3 x 3

Sedangkan Gambar 3.7 adalah *kernel* horizontal dan vertikal dengan ukuran *kernel* 5.

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

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

Gambar 3.7 *Kernel* Horizontal dan Vertikal Ukuran 5 x 5

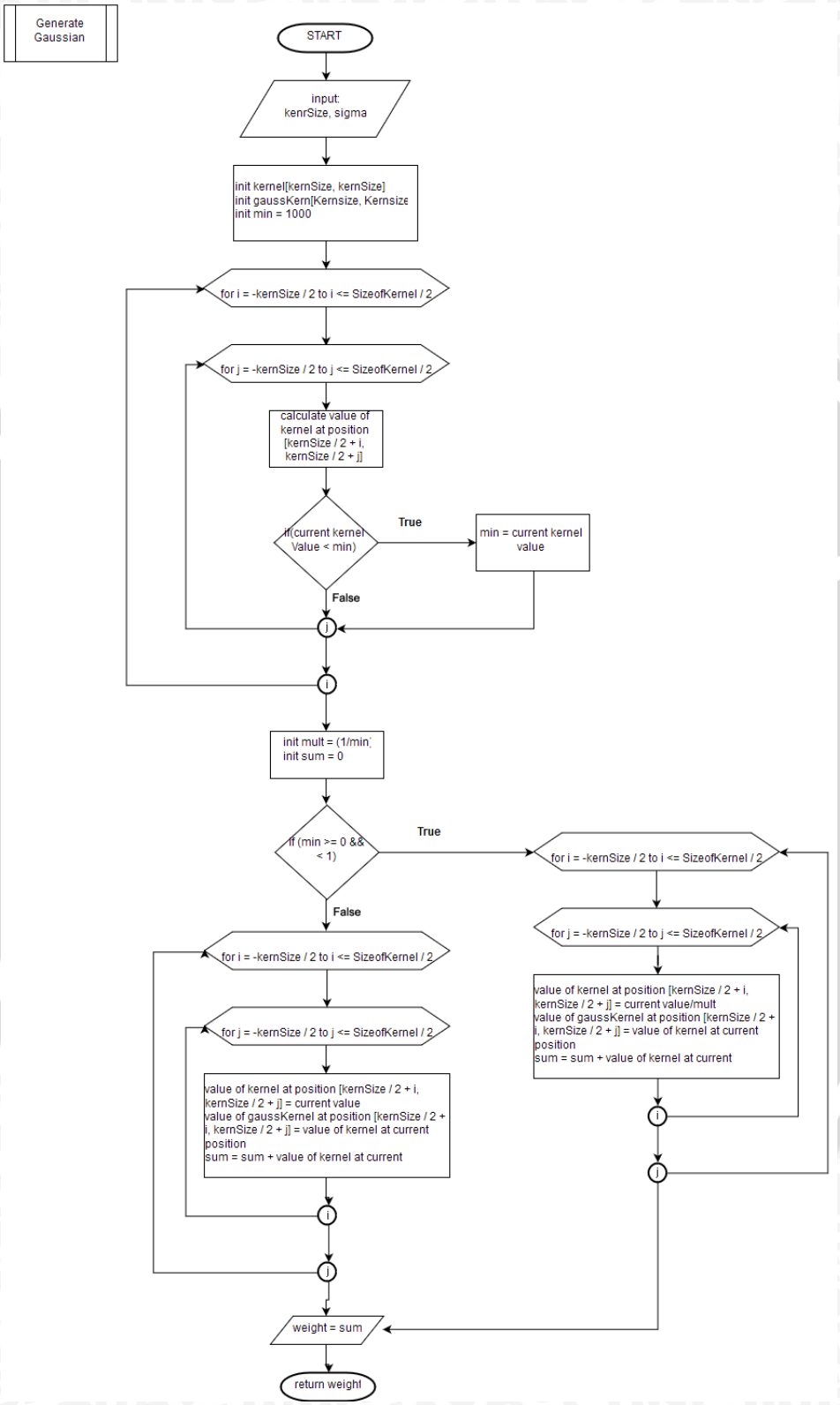
3. Dari *kernel* baru yang telah terbentuk dari langkah (2), temukan variabel berikut:
   a. *Minimal Value* (*minVal*): Dapat didapatkan dengan mengambil nilai terkecil dalam *kernel* yang terbentuk
   b. *Multiplier*: Tentukan nilai *multiplier* dengan persamaan sebagai (3.2):

$$multiplier = \frac{1}{minVal} \quad (3.2)$$

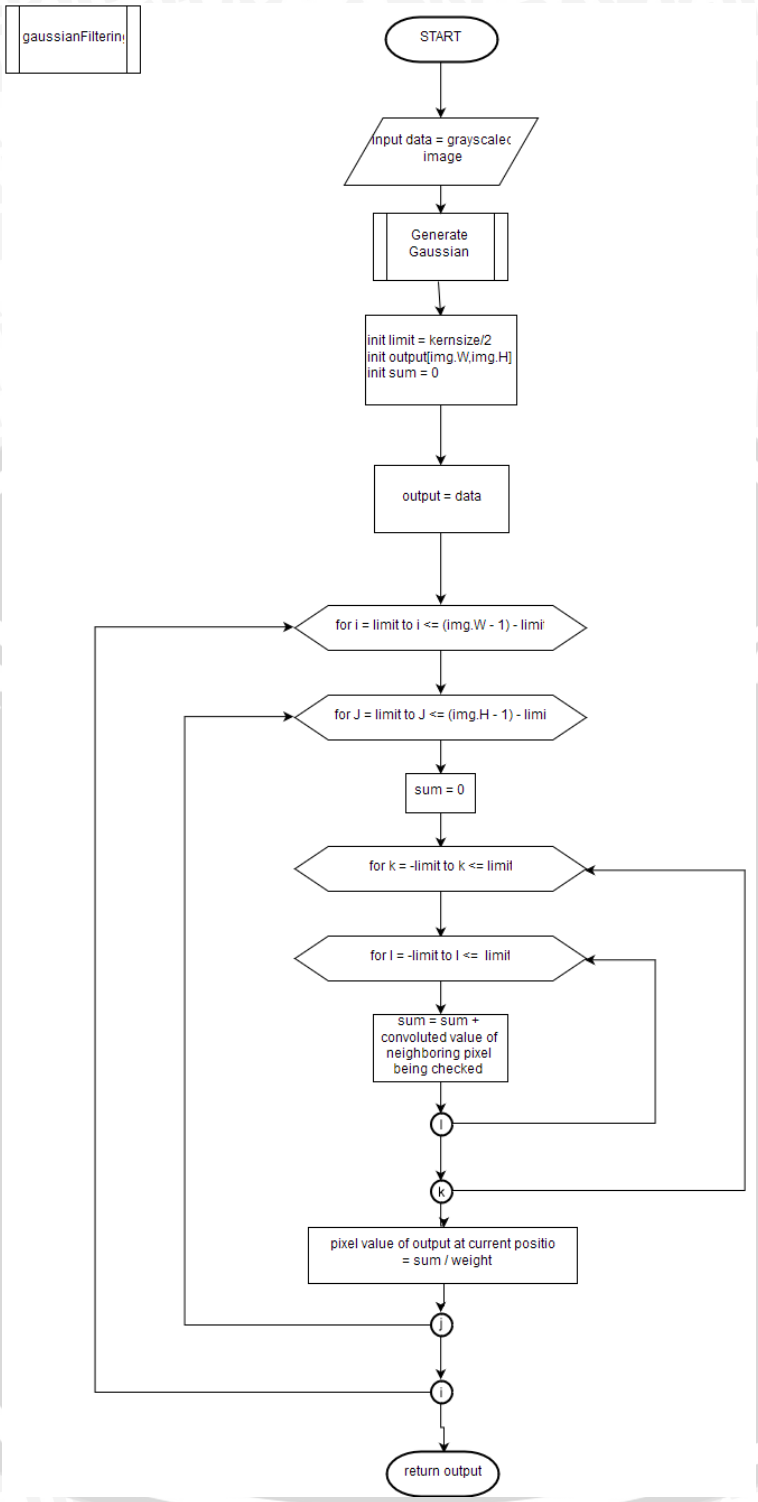
4. Bentuk *kernel* Gaussian dengan cara mengalikan tiap nilai pada *kernel* yang dihasilkan pada langkah (2) dengan nilai *multiplier* yang didapatkan dari langkah (3)
5. Tentukan nilai berat *kernel* Gaussian dengan menjumlahkan setiap nilai yang terdapat pada *kernel* Gaussian yang dihasilkan dari langkah (4)
6. Konvolusikan *kernel* Gaussian pada tiap piksel citra *grayscale*. Konvolusi *kernel* ini dilakukan dengan mengalikan nilai *piksel* tetangga dari *piksel* citra *input* dengan nilai pada *kernel* Gaussian yang relevan.
7. Nilai piksel keluaran dari Gaussian filter didapatkan dari menjumlahkan setiap nilai yang terdapat pada *kernel* yang dihasilkan dari langkah (6) dan dibagi dengan nilai berat *kernel* Gaussian yang didapat dari langkah (5).

Dalam implementasinya, langkah-langkah tersebut dapat dibagi menjadi dua fungsi, yaitu fungsi *generate Gaussian kernel* yang melingkupi langkah (1) hingga (5) dan fungsi *GaussianFiltering* yang merupakan langkah konvolusi *kernel*

Gaussian. Gambar 3.8 adalah diagram alir dari fungsi *generate Gaussian kernel,* dan Gambar 3.9 menunjukkan diagram alir dari fungsi *GaussianFiltering.*



Gambar 3.8 Diagram alir fungsi *generate Gaussian kernel*



Gambar 3.9 Diagram alir fungsi *GaussianFiltering*

3.3.3.2. Menentukan nilai Intensitas dan Gradien Piksel

Sebuah tepi adalah tempat terjadinya perubahan warna pada citra, sehingga intensitas piksel akan turut berubah. Pada aplikasi ini, gradien intensitas piksel ditentukan dengan menggunakan operator Sobel.

$G_x =$

1	0	-1
1	0	-1
1	0	-1

$* A,$ $G_y =$

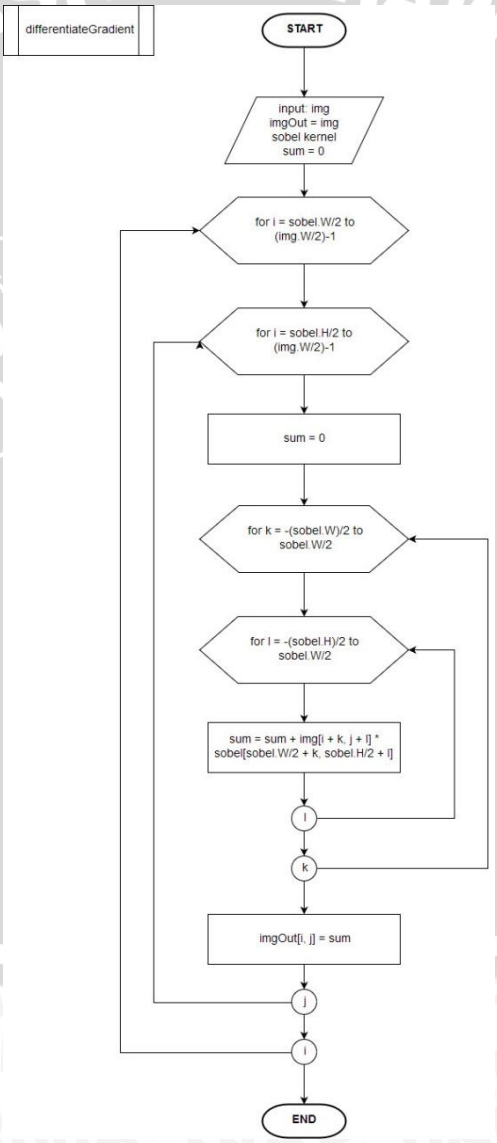
1	1	1
0	0	0
-1	-1	-1

$* A,$

Gambar 3.10 Operator Sobel

Kedua *kernel* tersebut dikonvolusikan kepada citra hasil Gaussian *filtering* sehingga menghasilkan dua citra, yaitu G_x untuk intensitas gradien Vertikal dan G_y untuk intensitas gradien Horisontal.

Gambar 3.10 menunjukkan diagram alir implementasi algoritme untuk menentukan intensitas gradien:

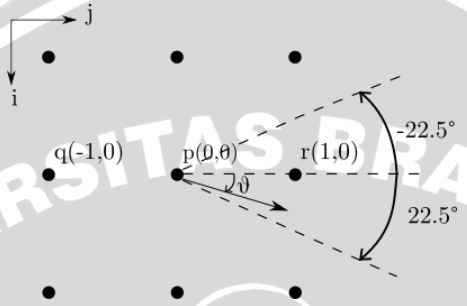


Gambar 3.11 Diagram alir fungsi *DifferentiateGradient*

Langkah selanjutnya adalah menghitung besaran intensitas dan besaran gradien secara menyuluruh untuk citra. Persamaan yang digunakan adalah Persamaan (2.2).

3.3.3.3. Penekanan *Non-maximum*

Penekanan *non-maximum* digunakan untuk menipiskan garis tepian yang sudah didapatkan dari ketiga langkah sebelumnya. Penekanan ini dilakukan dengan mencari nilai gradien piksel tersebut lalu membandingkannya dengan piksel tetangganya yang termasuk dalam besaran sudut gradien piksel tersebut.



Gambar 3.12 Penekanan non-Maksima

Dari Gambar 3.12, apabila piksel yang diperiksa adalah piksel *p*, dan nilai gradien yang didapatkan bernilai diantara -22,5° dan 22,5°, maka nilai dari nilai intensitas *p* akan dibandingkan dengan nilai intensitas dari piksel *r* yang terletak pada dalam jangkauan gradien piksel *p*. Apabila nilai dari piksel *p* lebih kecil dari nilai piksel *r*, maka nilai intensitas piksel *p* akan diubah menjadi 0 (nol); selain itu nilainya tidak diubah.

Dalam implementasinya, nilai gradien dibagi menjadi empat bagian, yaitu:

a) Horizontal, bila nilai gradien terletak diantara 22,5° hingga -22,5° atau gradien terletak antara 157,5° dan -157,5°

b) Vertikal, bila nilai gradien terletak diantara -112,5° hingga -67,5° atau gradien terletak antara 67,5° dan 112,5°

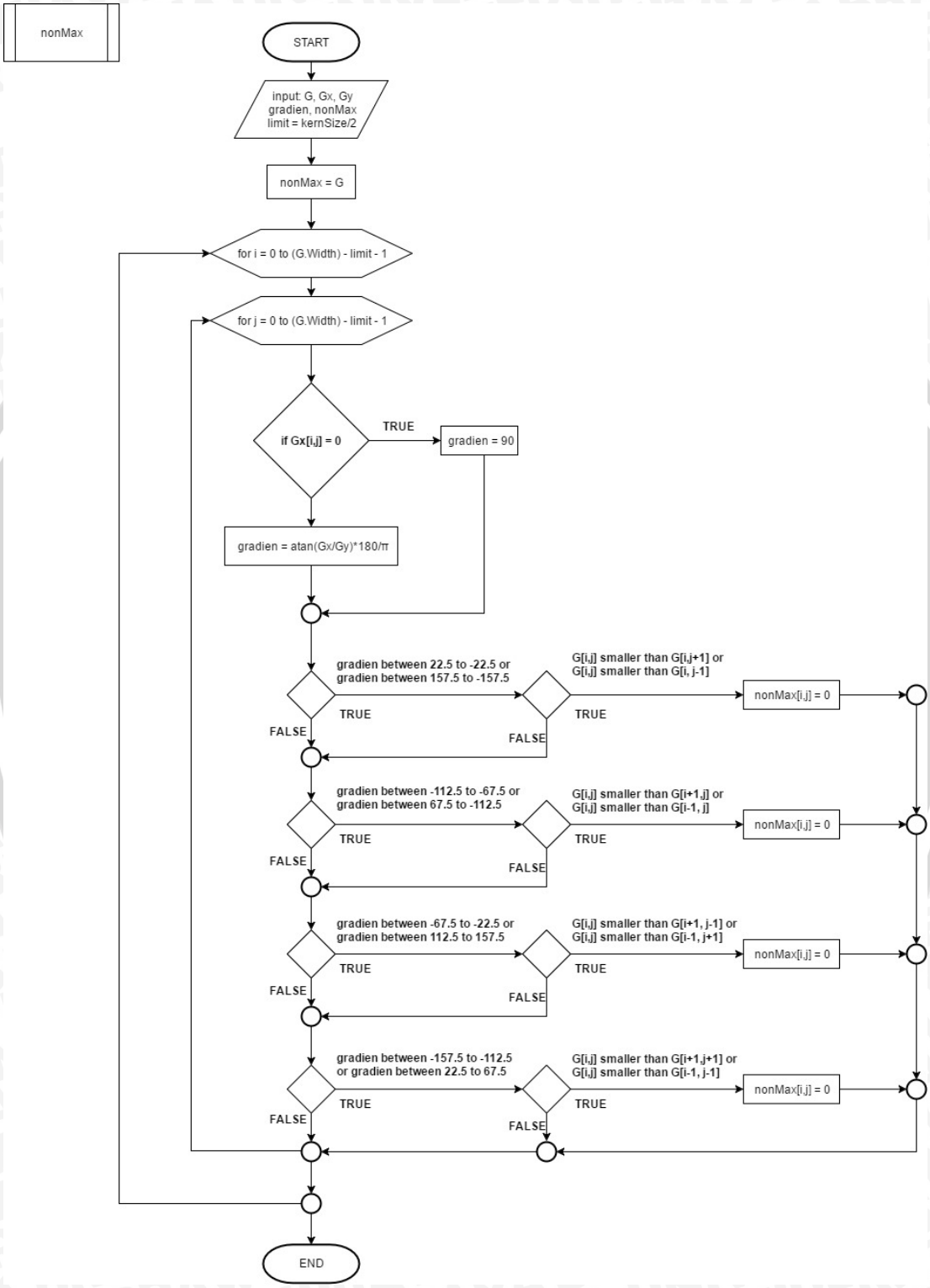
c) +45, bila nilai gradien terletak diantara -67,5° hingga -22,5° atau gradien terletak antara 112,5° dan 157,5°

d) -45, bila nilai gradien terletak diantara -157,5° hingga -112,5° atau gradien terletak antara 22,5° dan 67,5°

Selanjutnya karena perhitungan pada aplikasi menggunakan satuan derajat, maka hasil dari persamaan (2.3) terlebih dahulu diubah menjadi kedalam satuan derajat dengan persamaan (3.3).

$$\Theta = \text{atan}(\frac{G_y}{G_x}) * \frac{180}{\pi} \quad (3.3)$$

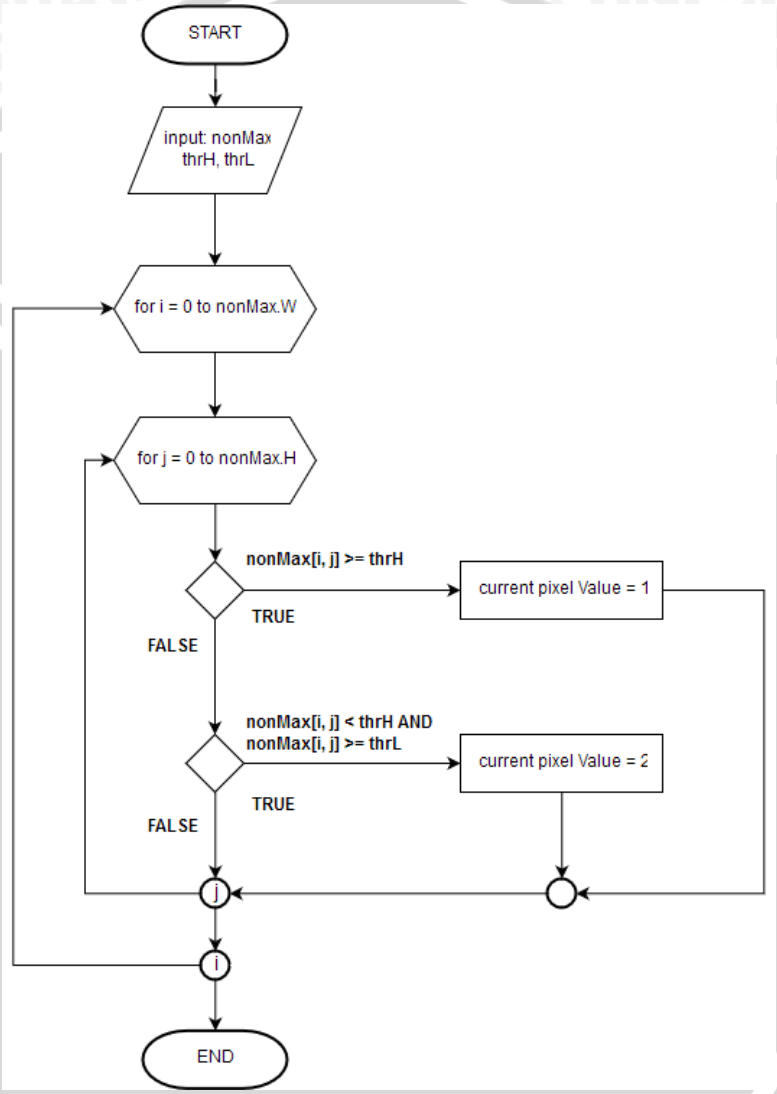
Gambar 3.13 adalah diagram alir implementasi langkah penekanan non-maksima pada aplikasi yang dibuat:



Gambar 3.13 Diagram alir proses penekanan non-Max

3.3.3.4. *Dual Thresholding*

Pada langkah ini, dilakukan proses untuk menentukan tepian mutlak dan tepian kandidat dari citra yang telah diolah sebelumnya. Piksel pada citra hasil nonMax yang memiliki nilai piksel sama atau diatas *threshold* atas diklasifikasikan sebagai tepian mutlak, dan piksel yang memiliki nilai diantara *threshold* bawah dan *threshold* atas merupakan tepian kandidat.

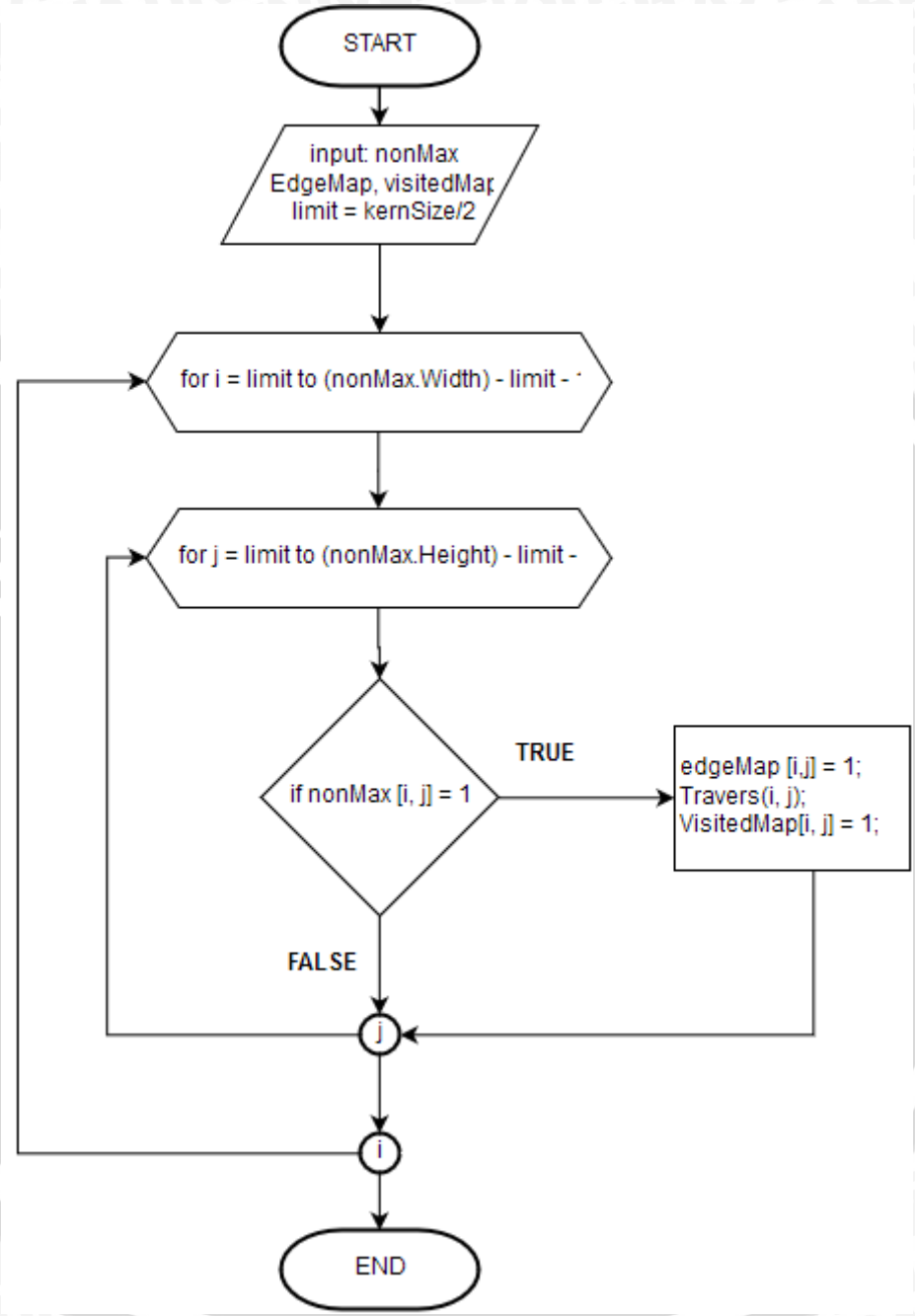


Gambar 3.14 Diagram alir proses dual-*thresholding*

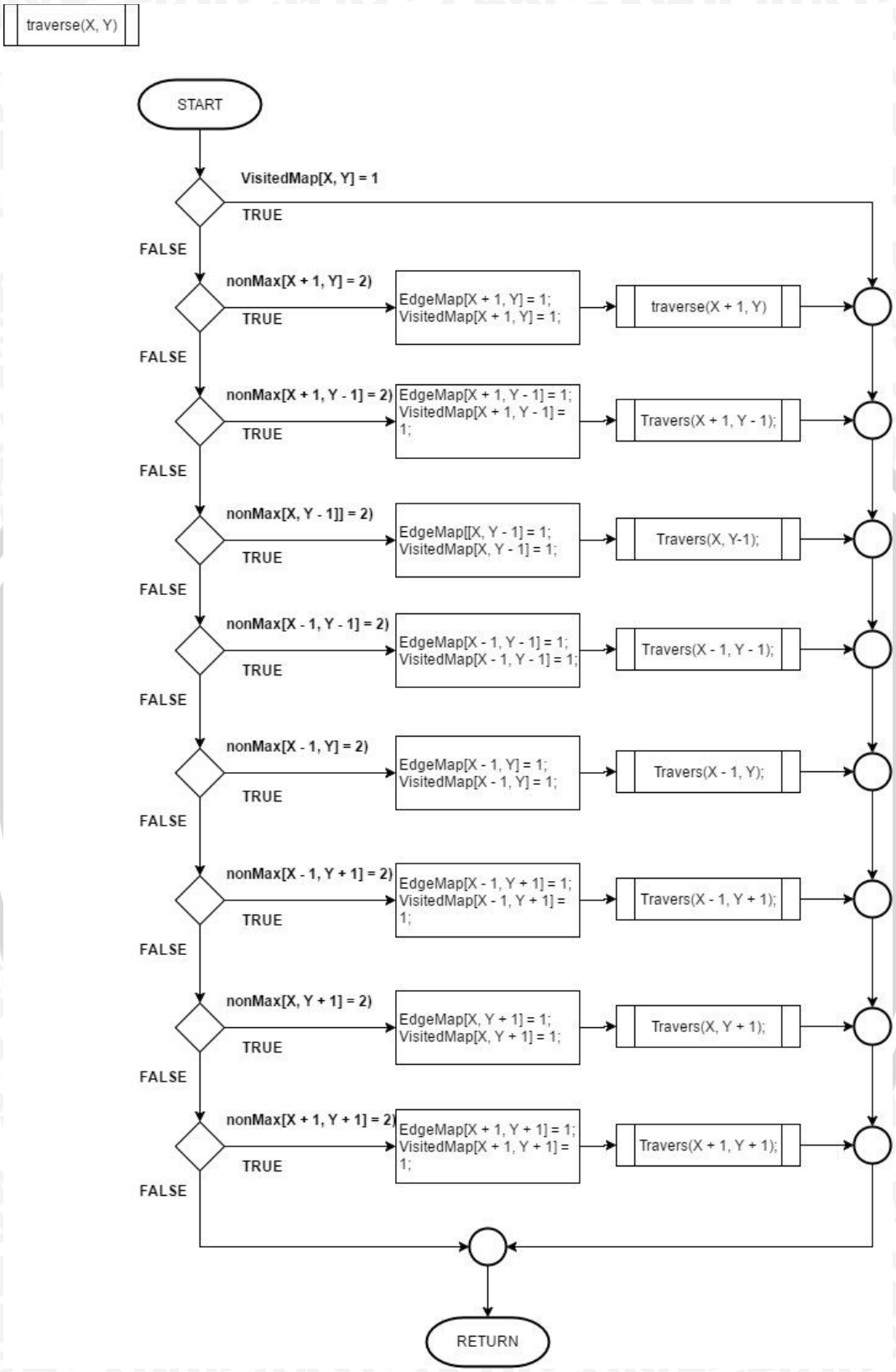
3.3.3.5. *Hysteresis*

Hysteresis dilakukan untuk mengecek kembali tepian mutlak dan membandingkannya dengan piksel tetangganya. Apabila terdapat piksel di sekitar tepian mutlak tersebut yang merupakan tepian kandidat, maka tepian kandidat tersebut diubah menjadi tepian mutlak. Pada penerapannya, proses ini dilakukan dengan sebuah fungsi *Hysteresis* yang didalamnya terdapat sebuah fungsi rekursif.

Gambar 3.15 menunjukkan diagram alir fungsi *Hysteresis*, sedangkan Gambar 3.16 menunjukkan diagram alir fungsi *Traverse*.



Gambar 3.15 Diagram alir fungsi Hysteresis



Gambar 3.16 Diagram alir fungsi *traverse*

3.3.4. Manualisasi Data

Pada manualisasi data yang dilakukan, citra satelit yang digunakan adalah Gambar 3.17.

Gambar 3.17 Sampel Citra Danau 1

Parameter yang digunakan dalam manualisasi dituliskan dalam Tabel 3.1:

Tabel 3.1 Nilai parameter Awal

No	Nama parameter	Nilai Parameter
1	*Kernel*Size	5
2	Sigma	1
3	ThresHigh	30
4	ThresLow	10

Manualisasi data yang dilakukan adalah penerapan metode Canny tanpa melalui proses segmentasi citra dengan *color thresholding*. Tabel 3.2, Tabel 3.3 dan Tabel 3.4 adalah sampel nilai piksel dari Gambar 3.17.

Tabel 3.2 Sampel nilai piksel untuk *Channel* RED

R	**0**	**1**	**2**	**3**	**4**
0	42	46	46	42	42
1	46	55	52	46	42
2	52	52	52	46	46
3	55	52	46	46	46
4	46	42	46	52	52

Tabel 3.3 Sampel nilai piksel untuk *Channel* Green

G	0	1	2	3	4
0	61	68	61	55	55
1	68	68	68	55	55
2	72	72	68	61	61
3	72	68	68	61	68
4	68	55	68	72	72

Tabel 3.4 Sampel nilai piksel untuk *Channel* Blue

B	0	1	2	3	4
0	4	9	4	4	4
1	9	9	15	9	0
2	15	0	9	15	9
3	15	9	9	4	17
4	9	0	0	9	15

3.3.4.1. *Image Grayscaling*

Berikut adalah perhitungan manual *image grayscaling* dengan sampel piksel pada koordinat [0,0], [2,3], dan [1.4]

1. $G = 0{,}3 * 42 + 0{,}59 * 61 + 0{,}11 * 4$
   $G = 12{,}6 + 35{,}99 + 0{,}44$
   $G = 49{,}03$
2. $G = 0{,}3 * 46 + 0{,}59 * 68 + 0{,}11 * 9$
   $G = 13{,}8 + 40{,}12 + 0{,}99$
   $G = 54{,}91$
3. $G = 0{,}3 * 42 + 0{,}59 * 55 + 0{,}11 * 0$
   $G = 12{,}6 + 32{,}45 + 0{,}44$
   $G = 45{,}05$

Perhitungan ini dilanjutkan untuk setiap piksel pada citra, dan akan menghasilkan sebuah citra dengan format *grayscale* seperti pada Tabel 3.5:

Tabel 3.5 Sampel nilai piksel citra *grayscale*

B	0	1	2	3	4
0	49	55	50	45	45
1	55	58	57	47	45
2	60	58	57	51	51
3	61	57	55	50	56
4	55	45	54	59	60

3.3.4.2. Gaussian *Filtering*

Dengan parameter *kernel* sebesar 5, maka *kernel* yang digunakan adalah *kernel* yang terdapat pada Gambar 3.7

1. Menentukan *Kernel* Gaussian

   Persamaan yang digunakan adalah persamaan (2.1) disesuaikan dengan dengan nilai *j* dan *k* merupakan nilai dari *kernel filtering* horizontal dan vertikal pada posisi yang relevan.

   Sebagai contoh, berikut adalah perhitungan manual untuk *kernel* Gaussian pada posisi [0,0]

   Nilai σ pada parameter masukan adalah 1 dan nilai j serta k masing-masing -2 dan -2.

$$kI[0,0] = \frac{1}{2 * \pi * 1} * \exp(-\frac{(-2)^2 + (-2)^2}{2 * 1^2})$$

$$kI[0,0] = \frac{1}{6{,}28318531} * \exp(-4)$$

$$kI[0,0] = 0{,}15915494 * 0{,}01831564$$

$$kI[0,0] = 0{,}002915024$$

   Perhitungan diulang untuk setiap nilai pada *kernel* horizontal dan vertikal, sehingga menghasilkan *intermediate kernel* seperti pada Gambar 3.18:

0,00291502	0,013064233	0,021539279	0,013064233	0,002915024
0,01306423	0,058549832	0,096532353	0,058549832	0,013064233
0,02153928	0,096532353	0,159154943	0,096532353	0,021539279
0,01306423	0,058549832	0,096532353	0,058549832	0,013064233
0,00291502	0,013064233	0,021539279	0,013064233	0,002915024

Gambar 3.18 *Intermediate kernel*

Dari *intermediate kernel* yang telah dihasilkan, dapat diambil nilai variabel *MinValue* dan *Multiplier* sebagai berikut:

a. MinValue = min(*kl*)

   MinValue = 0, 00291502

b. Multiplier, ditentukan oleh persamaan (3.2)

$$Multiplier = \frac{1}{0{,}00291502}$$

$$Multiplier = 343{,}050$$

Setelah *Multiplier* diketahui, *intermediate kernel* dikalikan dengan nilai multiplier untuk menghasilkan *kernel* Gaussian. Langkah ini dilakukan secara langsung seperti pada Gambar 3.19:

343,05 *

0,00291502	0,013064233	0,021539279	0,013064233	0,002915024
0,01306423	0,058549832	0,096532353	0,058549832	0,013064233
0,02153928	0,096532353	0,159154943	0,096532353	0,021539279
0,01306423	0,058549832	0,096532353	0,058549832	0,013064233
0,00291502	0,013064233	0,021539279	0,013064233	0,002915024

Gambar 3.19 Perkalian *Intermediate kernel* dengan variabel *Multiplier*

sehingga menghasilkan *Kernel* Gaussian yang tercantum pada Gambar 3.20:

1	4	7	4	1
4	20	33	20	4
7	33	55	33	7
4	20	33	20	4
1	4	7	4	1

Gambar 3.20 *Kernel* Gaussian

2. Menentukan *Weight* dari *Kernel* Gaussian

   Bobot dari *kernel* Gaussian dapat dihitung dengan menjumlahkan setiap nilai yang ada pada *kernel* Gaussian diatas.

$$W = 1 + 4 + 7 + 4 + 1 + 4 + 20 + 33 + 20 + 4 + 7 + 33 + 55 + 33 + 7 + 4 + 20 + 33 + 20 + 4 + 1 + 4 + 7 + 4 + 1$$

$$W = 337$$

3. Menentukan Nilai piksel citra setelah dikonvolusikan dengan *kernel* Gaussian

   Proses ini melibatkan konvolusi *kernel* Gaussian yang telah dihasilkan dari proses diatas dengan piksel tetangga yang diperiksa. Sebagai contoh, pada Gambar 3.21 adalah sampel citra *grayscale* sebesar 10x10 piksel dengan piksel yang akan dihitung terletak pada koordinat [2,2].

49	55	50	45	45	55	60	70	77	75
55	58	57	47	45	46	51	54	59	72
60	58	**57**	51	51	53	61	60	58	73
61	57	55	50	56	56	57	64	71	72
55	45	54	59	60	60	71	76	75	81
61	41	56	65	68	68	62	77	79	80
63	55	58	60	68	66	58	52	62	73
75	71	62	61	61	58	61	61	59	61
82	73	69	70	69	76	72	69	44	47
76	79	77	72	77	83	79	60	41	49

Gambar 3.21 Sampel citra berukuran 10x10 piksel

Tepian dengan warna merah menandakan nilai piksel yang akan dikonvolusikan dengan nilai pada *kernel* Gaussian. Konvolusi dilakukan dengan cara mengalikan nilai-nilai pada tiap matriks yang sesuai, menjumlahkan nilai hasilnya, lalu membaginya dengan bobot yang telah diketahui dari langkah sebelumnya.

fG:

49	55	50	45	45
55	58	57	47	45
60	58	**57**	51	51
61	57	55	50	56
55	45	54	59	60

*

1	4	7	4	1
4	20	33	20	4
7	33	55	33	7
4	20	33	20	4
1	4	7	4	1

fG:

49	246	371	204	45
246	1157	1900	949	202
441	1923	3096	1703	375
272	1139	1818	1009	250
55	202	398	265	60

Gambar 3.22 Konvolusi piksel dengan *Kernel* Gaussian

Selanjutnya, seluruh nilai yang terdapat pada hasil konvolusi tersebut dijumlahkan dan dibagi dengan nilai bobot *kernel* Gaussian.

$$f[2,2] = \frac{sum(fG)}{337}$$

$$f[2,2] = \frac{18376}{337}$$

$$f[2,2] = 55$$

Proses ini dilanjutkan untuk setiap piksel yang relevan, sehingga pada sampel dengan ukuran 10x10 yang digunakan akan dihasilkan sampel citra yang telah melalui proses Gaussian *filtering* seperti pada Gambar 3.23:

49	55	50	45	45	55	60	70	77	75
55	58	57	47	45	46	51	54	59	72
60	58	**55**	52	52	54	58	61	66	73
61	57	54	54	56	58	61	66	70	74
55	45	54	58	60	62	66	70	73	75
61	41	56	61	63	64	65	68	72	74
63	55	60	62	64	63	63	63	66	69
75	71	65	65	65	65	64	62	60	63
82	73	71	70	71	72	68	61	54	58
76	79	74	73	75	76	69	58	51	55

Gambar 3.23 Sampel citra hasil Gaussian *filtering*

Piksel yang ditandai dengan warna merah pada sampel diatas tidak mendapatkan perlakuan *filtering* Gaussian karena tidak sesuai dengan peraturan konvolusi *kernel* Gaussian.

3.3.4.3. Menentukan Intensitas Gradien Piksel

Proses menghitung nilai intensitas gradien piksel dilakukan dengan cara konvolusi seperti pada proses *Gaussian filtering*. Untuk penentuan intensitas gradien digunakan dua buah kenel operator Sobel seperti yang telah dijelaskan pada subbab 3.3.3.2. Perhitungan manual akan mengambil sampel citra sebesar 10x10 piksel hasil dari *Gaussian filtering* yang telah dilakukan.

49	55	50	45	45	55	60	70	77	75
55	58	57	47	45	46	51	54	59	72
60	58	55	52	52	54	58	61	66	73
61	57	54	54	56	58	61	66	70	74
55	45	54	58	60	62	66	70	73	75
61	41	56	61	63	64	65	68	72	74
63	55	60	62	64	63	63	63	66	69
75	71	65	65	65	65	64	62	60	63
82	73	71	70	71	72	68	61	54	58
76	79	74	73	75	76	69	58	51	55

Gambar 3.24 Sampel citra hasil Gaussian *filtering*

1. Menentukan derivatifX pada sumbu vertikal

G_x:

1	0	-1
1	0	-1
1	0	-1

$*$

49	55	50
55	58	57
60	58	55

G_x:

49	0	-50
55	0	-57
60	0	-55

$G_x[1,1] = 49 + 0 + (-50) + 55 + 0 + (-57) + 60 + 0 + (-55)$

$G_x[1,1] = 2$

Proses ini diteruskan untuk setiap piksel yang relevan, sehingga dari sampel citra 10x10 akan didapatkan derivatifX pada Gambar 3.25:

G_x:

0	0	0	0	0	0	0	0	0	0
0	**2**	27	20	-11	-27	-30	-33	-35	-35
0	10	20	13	-5	-17	-23	-25	-38	-40
0	13	-4	-5	-10	-17	-23	-24	-25	-19
0	13	-30	-15	-11	-13	-20	-23	-19	-11
0	9	-40	-17	-8	-7	-12	-17	-17	-13
0	18	-21	-11	-4	0	-1	-6	-13	-20
0	24	2	-4	-3	5	14	15	-4	-26
0	23	15	-1	-5	10	32	36	5	-27
0	11	11	-2	-6	18	49	49	7	-27

Gambar 3.25 Sampel derivatifX

2. Menentukan derivatifY pada sumbu horizontal

G_y:

1	1	1
0	0	0
-1	-1	-1

$*$

49	55	50
55	58	57
60	58	55

G_y:

49	55	50
0	0	0
-60	-58	-55

$G_y[1,1] = 49 + 55 + 50 + 0 + 0 + 0 + (-60) + (-58) + (-55)$

$G_y[1,1] = -19$

Sehingga hasil perhitungan derivatifY pada sampel akan menjadi seperti pada Gambar 3.26:

G_y:

0	0	0	0	0	0	0	0	0	0
0	**-19**	-15	-19	-13	-4	12	22	22	14
0	-2	-3	-15	-30	-33	-34	-33	-25	-7
0	19	8	-13	-22	-24	-25	-24	-18	-7
0	14	7	-16	-20	-17	-12	-8	-4	-1
0	-24	-20	-14	-9	-2	9	17	20	14
0	-53	-43	-15	-7	-2	6	19	29	29
0	-48	-37	-26	-24	-21	-12	9	25	34
0	-18	-25	-27	-29	-26	-12	8	21	26
0	2	-11	-13	-12	-7	4	11	14	12

Gambar 3.26 Sampel derivatifY

4. Menentukan nilai gradien final

Nilai final gradien ditentukan dengan menggunakan Persamaan (2.2):

$$G = \sqrt{Gx^2 + Gy^2}$$

Sebagai contoh, dari sampel tersebut, nilai intensitas gradien piksel [3,3]:

$G = \sqrt{-5^2 + -13^2}$

$G = \sqrt{194}$

$G \approx 14$

Sehingga nilai intensitas gradien nilai sampel yang dihitung adalah Gambar 3.27:

G:

0	0	0	0	0	0	0	0	0	0
0	19	31	28	17	27	32	40	41	38
0	10	20	20	30	37	41	41	45	41
0	23	8,9	**14**	24	29	34	34	31	20
0	19	31	22	23	21	23	24	19	11
0	26	45	22	12	7.3	15	24	26	19
0	56	48	19	8.1	2	6.1	20	32	35
0	54	37	26	24	22	18	17	25	43
0	29	29	27	29	28	34	37	22	37
0	11	16	13	13	19	49	50	16	30

Gambar 3.27 Sampel nilai intensitas gradien (*G*)

Pada implementasi aplikasi, besaran gradien dihitung pada tahap penekanan *non-maximum*, sehingga proses tersebut akan dibahas pada subbab penekanan *non-maximum*.

3.3.4.4. Penekanan *Non-maximum*

Penekanan non maxima dijalankan dengan mengetahui nilai gradien dari piksel yang diperiksa, kemudian membandingkan intensitas nilai piksel tersebut dengan piksel tetangga yang sesuai.

Sebagai contoh, perhitungan menggunakan piksel pada koordinat [3,3].

$$\Theta = \text{atan}(\frac{G_y}{G_x}) * \frac{180}{\pi}$$

$$\Theta = \text{atan}(\frac{-13}{5}) * \frac{180}{\pi}$$

$$\Theta = 1.2 * \frac{180}{\pi}$$

$$\Theta = 69$$

Setelah nilai gradien diketahui, maka dilakukan proses perbandingan nilai intensitas. Karena nilai 69 terletak antara 67,5 dan 157,5, maka piksel [3,3] termasuk piksel dengan gradien vertikal dan nilai intensitasnya akan dibandingkan dengan nilai piksel yang berada tepat diatas atau dibawahnya.

G:

0	0	0	0	0	0	0	0	0	0
0	19	31	28	17	27	32	40	41	38
0	10	20	20	30	37	41	41	45	41
0	23	8,9	**14**	24	29	34	34	31	20
0	19	31	22	23	21	23	24	19	11
0	26	45	22	12	7.3	15	24	26	19
0	56	48	19	8.1	2	6.1	20	32	35
0	54	37	26	24	22	18	17	25	43
0	29	29	27	29	28	34	37	22	37
0	11	16	13	13	19	49	50	16	30

Gambar 3.28 Proses penentuan arah gradien piksel [3,3]

Setelah diketahui bahwa nilai piksel [3,3] = 14 lebih kecil daripada nilai piksel diatas atau dibawahnya, maka nilai piksel [3,3] akan diubah menjadi bernilai 0. Proses ini dilakukan untuk setiap piksel yang telah diketahui nilai intensitas gradiennya.

	0	0	0	0	0	0	0	0	0	0
	0	19	30	27	17	27	32	39	41	37
	0	10	20	19	30	37	41	41	45	0
	0	23	0	**0**	24	0	0	0	0	0
	0	19	30	21	22	0	0	24	0	0
nM:	0	25	44	0	0	0	0	24	26	0
	0	55	47	0	0	0	0	0	31	35
	0	53	0	0	0	0	0	0	0	42
	0	29	29	27	29	27	0	36	0	0
	0	11	0	0	0	0	0	50	0	0

Gambar 3.29 Hasil penekanan *non-maximum* pada piksel [3,3]

3.3.4.5. *Dual Thresholding*

Proses *dual thresholding* dilakukan dengan memasukkan nilai dari citra hasil *non-maximum* dengan *threshold* yang telah ditentukan sebagai parameter awal. Mengacu pada pembahasan pada subbab 3.3.3.4, maka kondisi yang digunakan adalah sebagai berikut:

IF (nonMax value > thresH) THEN nonMax = 1

ELSE IF (nonMaks value < thresH AND >= thresL) THEN nonMax = 2

Menggunakan kondisi seperti diatas, maka untuk sampel citra nonmaks 10x10 yang dihasilkan dari subbab sebelumnya didapatkan nilai citra seperti Gambar 3.30:

		2	1	2	2	2	1	1	1	1
		2	2	2	1	1	1	1	1	
		2			2					
		2	1	2	2			2		
dT:		2	1					2	2	
		1	1						1	1
		1								1
		2	2	2	2	2		1		
		2						1		

Gambar 3.30 Sampel citra setelah melalui proses *dual thresholding*

3.3.4.6. Hysteresis

Langkah pertama adalah mendapatkan nilai sudut mutlak dari citra hasil dual *thresholding*. Berikut adalah sampel citra dengan sudut mutlak yang didapatkan:

E:

		1				1	1	1	1
				1	1	1	1	1	
		1							
		1							
	1	1						1	1
	1								1
							1		
							1		

Gambar 3.16 Sampel citra tepian mutlak ukuran 10 x 10

Pada tahap ini akan diberikan contoh manualisasi untuk satu piksel pada koordinat [2,1]

1. nonMax[2,1] == 1, maka E = 1
2. traverse(2,1)
    a. visitedMap[2,1] =/= 1, maka fungsi rekursif berjalan.
    b. nonMax[3,1] = 2, maka edgeMap[3,1] = 1
    c. visitedMap[3,1] = 1
    d. traverse(3,1)
        i. visitedMap[2,1] == 1, maka fungsi berhenti.
3. visitedMap[2,1] = 1

Maka setelah perulangan pertama akan didapatkan sampel citra seperti berikut:

E:

		1	**1**			1	1	1	1
				1	1	1	1	1	
		1							
		1							
	1	1						1	1
	1								1
							1		
							1		

Gambar 3.16 Sampel citra tepian mutlak setelah iterasi pertama

3.3.4.7. Color *Threshold*ing

Proses segmentasi *thresholding* warna dilakukan pada sebagai salah satu opsi *pre-processing* sebelum proses deteksi tepi dengan metode Canny dijalankan. Aplikasi dapat berjalan tanpa melakukan segmentasi *thresholding* warna terlebih dahulu.

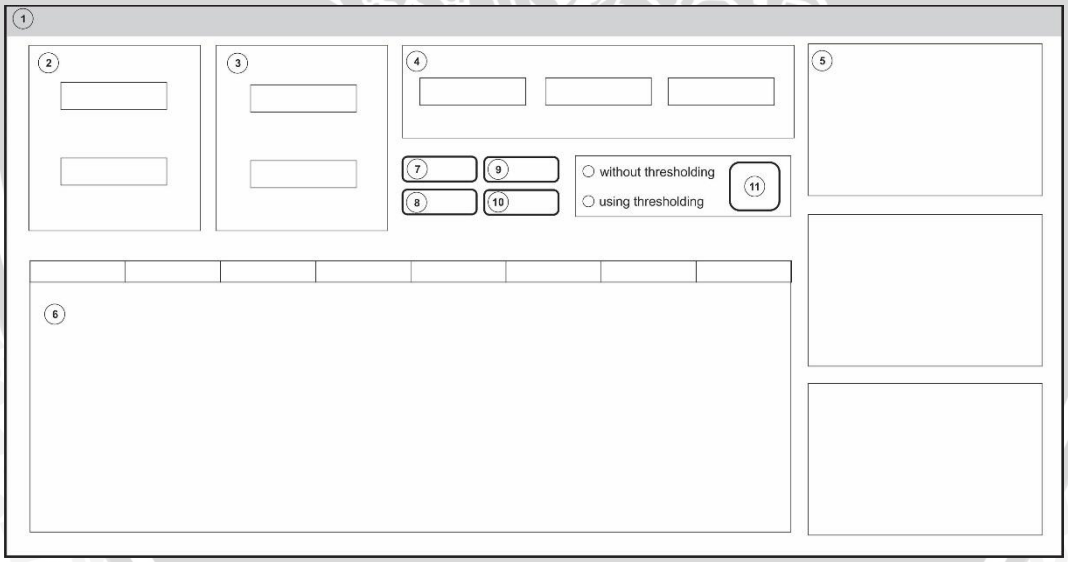
Berikut adalah contoh proses segmentasi warna dengan parameter *threshold*:

1. tR: 20, tG: 30, tB: 50
2. Nilai Piksel pada koordinat [0,0] = (42, 61, 4)

Setelah dilakukan pengecekan, maka dapat diketahui bahwa nilai piksel pada koordinat [0,0] tidak memenuhi kondisi (R <= tR && B <= tG && B < tB), maka nilai piksel [0,0] akan diubah menjadi berwarna hitam (0,0,0).

3.3.5. Perancangan Antarmuka

Subbab ini membahas tampilan antarmuka dari aplikasi. Aplikasi memiliki sebuah halaman utama, dimana pengguna dapat memasukkan parameter deteksi tepi yang diinginkan serta dapat melihat hasil dari deteksi tepi beserta hasil pengujian kesalahan deteksi tepi yang dilakukan.



Gambar 3.31 Desain Aplikasi Deteksi Tepi

Keterangan gambar:

1. Nama aplikasi
2. Panel masukan parameter ukuran *kernel* beserta nilai sigma
3. Panel masukan parameter *threshold* atas dan bawah
4. Panel masukan parameter untuk *thresholding* warna, masukan berupa nilai *threshold* untuk tiap channel *Red*, *Green*, dan *Blue*.

5. Representasi histogram warna dari citra masukan untuk channel *red*, *green* dan *blue*.
6. *Tabbed panel*, dengan fungsi masing-masing tab sebagai berikut:
    a. Menampilkan citra masukan
    b. Menampilkan citra yang telah melalui proses *grayscale*
    c. Menampilkan citra yang telah melalui proses *Gaussian filtering*
    d. Menampilkan citra yang telah melalui proses penekanan non-Max
    e. Menampilkan citra hasil deteksi tepi
    f. Menampilkan citra yang telah melalui proses *thresholding* warna
    g. Menampilkan citra yang digunakan sebagai *key* dalam pengujian
    h. Menampilkan citra hasil pengujian beserta jumlah *error* tepi.
7. Tombol untuk *import* citra masukan
8. Tombol untuk *import* citra *key* sebagai penguji
9. Tombol untuk melakukan proses *thresholding* warna
10. Tombol untuk proses pengujian antara citra hasil deteksi tepi dan citra *key*
11. Tombol untuk melakukan proses Deteksi Tepi dengan Metode Canny, beserta opsi untuk melakukan deteksi dengan atau tanpa melakukan proses *thresholding* warna terlebih dahulu.

3.4 Implementasi Sistem

Pada penelitian ini dilakukan implementasi algoritme deteksi tepi menggunakan metode deteksi tepi Canny serta implementasi segmentasi dengan *thresholding* warna sebagai tahap *pre-processing* citra menggunakan bahasa pemrograman C#.

3.5 Pengujian

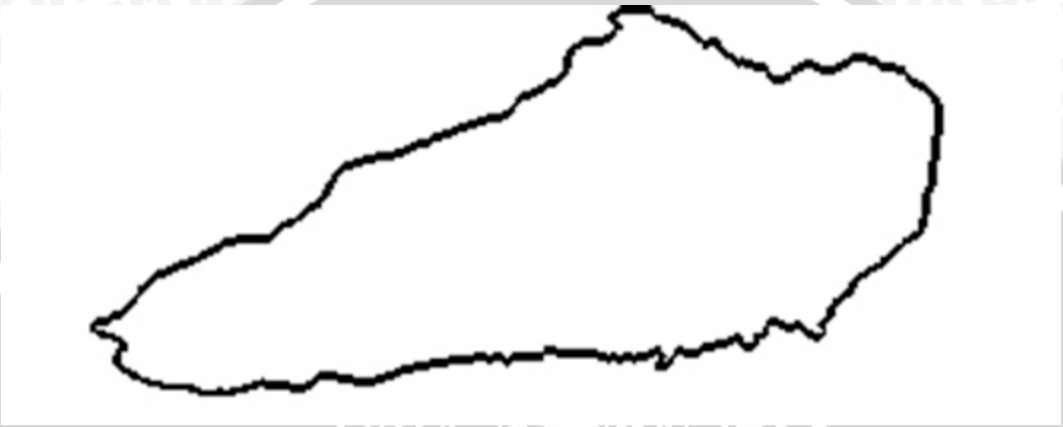
Pengujian algoritme deteksi tepi Canny dilakukan untuk mengetahui tingkat keberhasilan algoritme dalam menyelesaikan permasalahan yaitu mendeteksi tepi permukaan danau pada citra satelit masukan.

Untuk mengetahui nilai parameter Gaussian serta *threshold* yang optimal, dilakukan proses deteksi tepi dengan menggunakan paramater yang berbeda-beda. Aplikasi juga melakukan perbandingan antara citra keluaran dari proses yang hanya melibatkan deteksi tepi Canny dan keluaran citra hasil dari deteksi tepi yang telah melalui proses *pre-processing* dengan segmentasi menggunakan *thresholding* warna.

Untuk mengetahui nilai *error rate* dari citra keluaran, dilakukan pencocokan dengan metode *ground truth.* Citra keluaran dibandingkan sebuah citra *ground*

*truth (*selanjutnya akan disebut sebagai citra *key)* dengan tepian permukaan danau telah ditandai secara manual oleh peneliti.

Proses pembuatan citra *key (ground truth)* melibatkan proses menandai tepian permukaan danau secara manual per piksel untuk setiap gambar yang akan dijadikan key, serta melakukan dilasi menggunakan aplikasi imageJ untuk tepian sebanyak 1 piksel untuk mengakomodasi translasi tepian pada citra keluaran aplikasi. Gambar 3.19 menunjukkan sebuah sampel citra *key* yang digunakan dengan warna yang telah di-inversi untuk tujuan penulisan laporan.



Gambar 3.32 Sampel Citra *key*

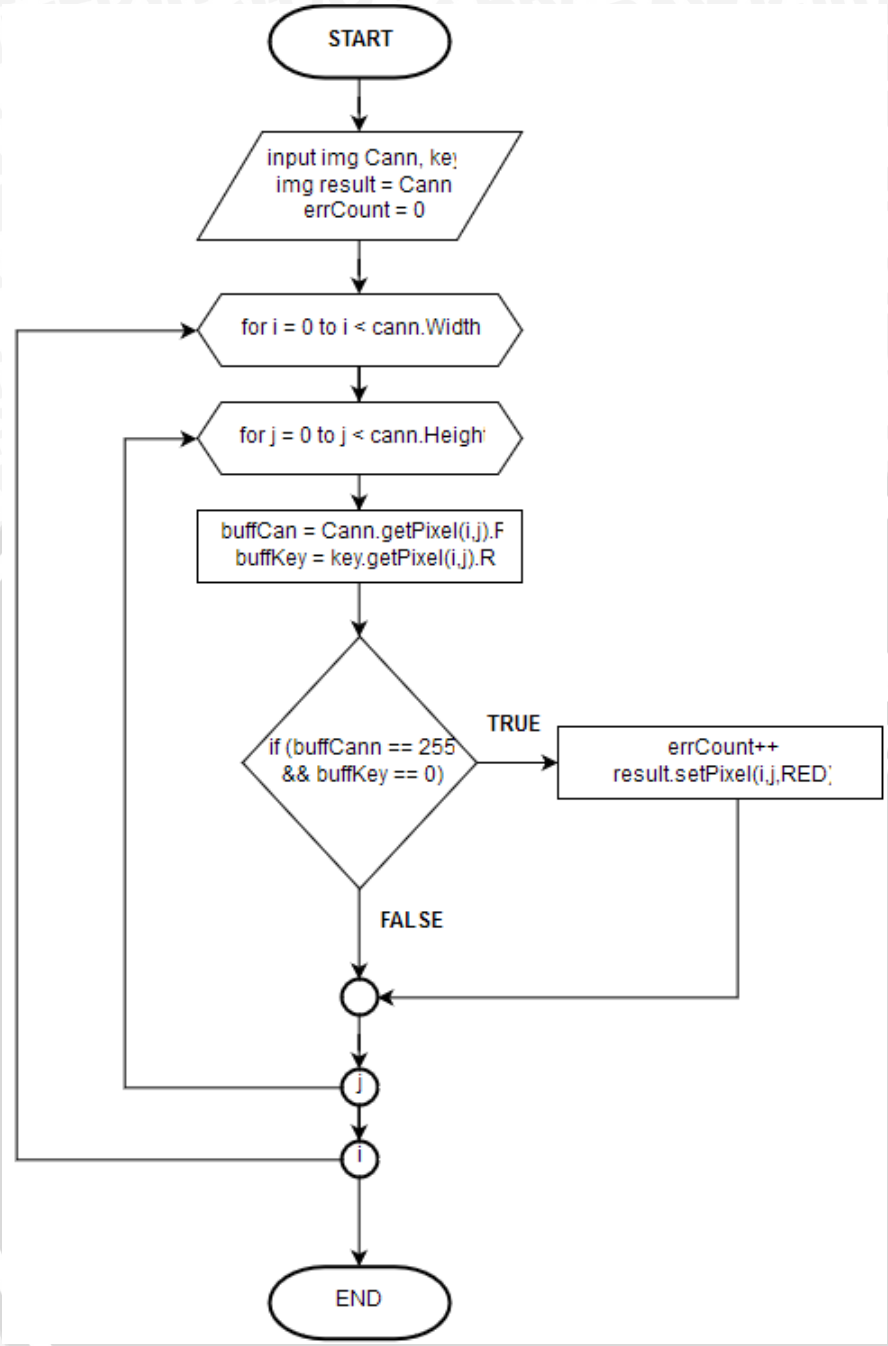
Nilai piksel dari citra keluaran aplikasi akan dibandingkan dengan nilai tepian pada citra key, dengan ketentuan bahwa *error* didefinisikan ketika piksel tepian pada citra hasil keluaran tidak bertumpuk (*overlap*) dengan piksel tepian pada citra key yang dijadikan acuan. Nilai *error* adalah banyaknya piksel yang terdeteksi sebagai piksel *error*.

Citra yang diuji dengan nilai parameter yang digunakan akan dicatat pada tabel pengujian seperti pada Tabel 3.6:

Tabel 3.6 Pengujian Citra Keluaran

No. Kasus	Tepian Terdeteksi	*Error* Terdeteksi	Tepian Mutlak	Rasio *Error*

Gambar 3.20 menunjukkan diagram alir proses pengujian yang dilakukan oleh aplikasi.



Gambar 3.33 Diagram alir Proses Pengujian

BAB 4 HASIL

Bab ini menjelaskan hasil dari perancangan yang dilakukan pada bab sebelumnya. Hasil penelitian ini adalah implementasi deteksi tepi permukaan danau pada citra satelit menggunakan algoritme Canny dan dibantu dengan proses *pre-processing* dengan segmentasi melalui *thresholding* warna. Hasil implementasi meliputi lingkungan implementasi aplikasi berupa lingkungan *hardware* dan *software,* implementasi algoritme dan implementasi antarmuka.

4.1 Lingkungan Implementasi

Lingkungan implementasi menjelaskan perangkat yang digunakan untuk membangun aplikasi yang terdiri dari *hardware* dan *software.*

4.1.1. Lingkungan Implementasi *Hardware*

Perangkat keras yang digunakan dalam implementasi deteksi tepi citra dengan algoritme Canny adalah sebagai berikut:

1. Prosesor intel® CORE™ i5-5500U @ 2.20 GHz
2. RAM 4 Gb
3. SSD 250 Gb

4.1.2. Lingkungan Implementasi *Software*

Dalam implementasi algoritme Canny untuk melakukan deteksi tepi, digunakan perangkat lunak sebagai berikut:

1. Sistem operasi Windows 10 x64
2. ImageJ
3. Adobe Photoshop CS4 (64 Bit)
4. Microsoft Visual Studio C# Express 2010
5. Bahasa Pemrograman C#
6. Internet Browser

4.2 Implementasi Algoritme

Bagian ini menjelaskan tentang implementasi perancangan yang telah dibuat pada bab sebelumnya kedalam *source code* aplikasi.

4.2.1. Implementasi Algoritme *Image Grayscaling*

Source code 4.1 berikut menunjukkan implementasi algoritme *Image Grayscaling* sesuai dengan perhitungan manual yang dilakukan sebelumnya.

```
public static Bitmap MakeGrayscale(Bitmap ori){
    //create a blank bitmap the same size as ori
    Bitmap newBitmap = new Bitmap(ori.Width, ori.Height);

    int[,] values = new int[ori.Width, ori.Height];
```

```
        //get a graphics object from the new image
        Graphics g = Graphics.FromImage(newBitmap);

        //create the grayscale ColorMatrix
        ColorMatrix colorMatrix = new ColorMatrix
        new float[][]
        {
            new float[] {.3f, .3f, .3f, 0, 0},
            new float[] {.59f, .59f, .59f, 0, 0},
            new float[] {.11f, .11f, .11f, 0, 0},
            new float[] {0, 0, 0, 1, 0},
            new float[] {0, 0, 0, 0, 1}
        });

        //create some image attributes
        ImageAttributes attributes = new ImageAttributes();

        //set the color matrix attribute
        attributes.SetColorMatrix(colorMatrix);

        //draw the original image on the new image
        //using the grayscale color matrix
        g.DrawImage(ori, new Rectangle(
        0, 0, ori.Width, ori.Height),
        0, 0, ori.Width, ori.Height,
        GraphicsUnit.Pixel, attributes);

        //dispose the Graphics object
        g.Dispose();

        return newBitmap;
    }
```

4.2.2. Implementasi Algoritme *Color Thresholding*

Source code 4.2 berikut menunjukkan implementasi algoritme *Color Thresholding* sesuai dengan perhitungan manual yang dilakukan sebelumnya.

```
public static Bitmap colThres(Bitmap ori, int tR, int tG, int tB){
        Bitmap processed = new Bitmap(ori.Width, ori.Height);
        for (int i = 0; i < ori.Width; i++){
            for (int j = 0; j < ori.Height; j++){
                Color piksel = ori.GetPixel(i, j);
                if (piksel.R <= tR && piksel.G <= tG && piksel.B <= tB){
                    processed.SetPixel(i, j, piksel);
                } else processed.SetPixel(i, j, Color.Black);
            }
        }
        return processed;
    }
```

4.2.3. Implementasi Algoritme *Gaussian Filtering*

Source code 4.3 berikut menunjukkan implementasi algoritme *Generate Gaussian Kernel* sesuai dengan perhitungan manual yang dilakukan sebelumnya.

```
private void GenerateGaussianKernel(int N, float S, out int Weight)
{
    float Sigma = S;
    float pi;
    pi = (float)Math.PI;
```

```
    int i, j;
    int SizeofKernel = N;

    float[,] Kernel = new float[N, N];
    GaussianKernel = new int[N, N];
    float D1, D2;

    //calculating constant for simpler implementation
    D1 = 1 / (2 * pi * Sigma * Sigma);
    D2 = 2 * Sigma * Sigma;

    float min = 1000;

    for (i = -SizeofKernel / 2; i <= SizeofKernel / 2; i++)
    {
        for (j = -SizeofKernel / 2; j <= SizeofKernel / 2; j++)
        {
            Kernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j] = (D1
*(float)Math.Exp(-(i * i + j * j) / D2));
            if (Kernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j] <
min)
                min = Kernel[SizeofKernel / 2 + i, SizeofKernel / 2 +
j];
        }
    }

    int mult = (int)(1 / min);
    int sum = 0;
    if ((min > 0) && (min < 1))
        {
            for (i = -SizeofKernel / 2; i <= SizeofKernel / 2; i++)
            {
                for (j = -SizeofKernel / 2; j <= SizeofKernel / 2; j++)
                    {
                        Kernel[SizeofKernel / 2 + i, SizeofKernel / 2 +
j] = (float)Math.Round(Kernel[SizeofKernel / 2 + i, SizeofKernel / 2 +
j] * mult, 0);
                        GaussianKernel[SizeofKernel / 2 + i,
SizeofKernel / 2 + j] =
                        (int)Kernel[SizeofKernel / 2 + i, SizeofKernel
/ 2 + j];
                        sum = sum + GaussianKernel[SizeofKernel / 2 +
i, SizeofKernel / 2 + j];
                    }

            }

        }
        else
            {
                sum = 0;
                for (i = -SizeofKernel / 2; i <= SizeofKernel / 2; i++)
                {
                    for (j = -SizeofKernel / 2; j <= SizeofKernel / 2;
j++)
                    {
                        Kernel[SizeofKernel / 2 + i, SizeofKernel / 2 +
j] = float)Math.Round(Kernel[SizeofKernel / 2 + i, SizeofKernel / 2 +
j], 0);
                        GaussianKernel[SizeofKernel / 2 + i,
SizeofKernel / 2 + j] = (int)Kernel[SizeofKernel / 2 + i, SizeofKernel
/ 2 + j];
                        sum = sum + GaussianKernel[SizeofKernel / 2 +
i, SizeofKernel / 2 + j];
                    }
```

```
                }
            }

    //Normalizing kernel Weight
    Weight = sum;

    return;
}
```

Source code 4.4 berikut menunjukkan implementasi algoritme *Gaussian Filtering* sesuai dengan perhitungan manual yang dilakukan sebelumnya.

```
private int[,] GaussianFilter(int[,] Data)
{
    GenerateGaussianKernel(KernelSize, Sigma, out KernelWeight);

    int[,] Output = new int[Width, Height];
    int i, j, k, l;
    int Limit = KernelSize / 2;

    float Sum = 0;

    Output = Data;
    for (i = Limit; i <= ((Width - 1) - Limit); i++)
    {
        for (j = Limit; j <= ((Height - 1) - Limit); j++)
        {
            Sum = 0;
            for (k = -Limit; k <= Limit; k++)
            {
                for (l = -Limit; l <= Limit; l++)
                {
                    Sum = Sum + ((float)Data[i + k, j + l] *
GaussianKernel[Limit + k, Limit + l]);
                }
            }
            Output[i, j] = (int)(Math.Round(Sum /
(float)KernelWeight));
        }

    }

    return Output;
}
```

4.2.4. Implementasi Algoritme untuk Menentukan Intensitas Gradien

Source code 4.5 berikut menunjukkan implementasi algoritme *Differentiate Gradientl* sesuai dengan perhitungan manual yang dilakukan sebelumnya.

```
private float[,] Differentiate(int[,] Data, int[,] Filter)
{
    int i, j, k, l, Fh, Fw;
    Fw = Filter.GetLength(0);
    Fh = Filter.GetLength(1);
    float sum = 0;
    float[,] Output = new float[Width, Height];

    for (i = Fw / 2; i <= (Width - Fw / 2) - 1; i++)
    {
        for (j = Fh / 2; j <= (Height - Fh / 2) - 1; j++)
        {
            sum = 0;
            for (k = -Fw / 2; k <= Fw / 2; k++)
```

```
            {
                for (l = -Fh / 2; l <= Fh / 2; l++)
                {
                    sum = sum + Data[i + k, j + l] * Filter[Fw / 2 + k,
Fh / 2 + l];
                }
            }
            Output[i, j] = sum;
        }

    }

    return Output;
}
```

4.2.5. Implementasi Algoritme Penekanan *Non-maximum*

Source code 4.6 berikut menunjukkan implementasi algoritme *nonMax* sesuai dengan perhitungan manual yang dilakukan sebelumnya.

```
private float[,] WriteGaussianFiltered()
{
    Gradient = new float[Width, Height];
    NonMax = new float[Width, Height];
    PostHysteresis = new int[Width, Height];

    DerivativeX = new float[Width, Height];
    DerivativeY = new float[Width, Height];
    FilteredImage = GaussianFilter(GreyImage);

    int[,] Dx = {
    {1,0,-1},
    {1,0,-1},
    {1,0,-1}};

    int[,] Dy = {
    {1,1,1},
    {0,0,0},
    {-1,-1,-1}};

    DerivativeX = Differentiate(FilteredImage, Dx);
    DerivativeY = Differentiate(FilteredImage, Dy);

    int i, j;

    //calculate the gradient intensity based on derivatives in x and y:
    for (i = 0; i <= (Width - 1); i++)
    {
        for (j = 0; j <= (Height - 1); j++)
        {
            Gradient[i, j] = (float)Math.Sqrt((DerivativeX[i, j] *
DerivativeX[i, j]) + (DerivativeY[i, j] * DerivativeY[i, j]));
        }
    }

    for (i = 0; i <= (Width - 1); i++)
    {
        for (j = 0; j <= (Height - 1); j++)
        {
            NonMax[i, j] = Gradient[i, j];
        }
    }

    int Limit = KernelSize / 2;
```

```
    float angle;

    for (i = Limit; i <= (Width - Limit) - 1; i++)
    {
        for (j = Limit; j <= (Height - Limit) - 1; j++)
        {
            if (DerivativeX[i, j] == 0)
            angle = 90F;
            else
                //rad to degree
                angle = (float)(Math.Atan(DerivativeY[i, j] /
DerivativeX[i, j]) * 180 / Math.PI);


                //Horizontal
                if (((-22.5 < angle) && (angle <= 22.5)) || ((157.5 <
angle) && (angle <= -157.5)))
                {
                    if ((Gradient[i, j] < Gradient[i, j + 1]) ||
(Gradient[i, j] < Gradient[i, j - 1]))
                        NonMax[i, j] = 0;
                }

                //Vertical
                if (((-112.5 < angle) && (angle <= -67.5)) || ((67.5 <
angle) && (angle <= 112.5)))
                {
                    if ((Gradient[i, j] < Gradient[i + 1, j]) ||
(Gradient[i, j] < Gradient[i - 1, j]))
                        NonMax[i, j] = 0;
                }

                //+45 deg
                if (((-67.5 < angle) && (angle <= -22.5)) || ((112.5 <
angle) && (angle <= 157.5)))
                {
                    if ((Gradient[i, j] < Gradient[i + 1, j - 1]) ||
(Gradient[i, j] < Gradient[i - 1, j + 1]))
                        NonMax[i, j] = 0;
                }

                //-45 deg
                if (((-157.5 < angle) && (angle <= -112.5)) || ((22.5 <
angle) && (angle <= 67.5)))
                {
                    if ((Gradient[i, j] < Gradient[i + 1, j + 1]) ||
(Gradient[i, j] < Gradient[i - 1, j - 1]))
                        NonMax[i, j] = 0;
                }
        }
    }

    return nonMax;
}
```

4.2.6. Implementasi Algoritme *Dual-thresholding*

Source code 4.6 berikut menunjukkan implementasi algoritme *Dual-thresholding* sesuai dengan perhitungan manual yang dilakukan sebelumnya.

```
private void dualThreshold(float [,] nonMaxed)
{
    EdgePoints = new int[Width, Height];
```

```
    for (r = Limit; r <= (Width - Limit) - 1; r++)
    {
        for (c = Limit; c <= (Height - Limit) - 1; c++)
        {
            if (nonMaxed[r, c] >= MaxHysteresisThresh)
            {
                EdgePoints[r, c] = 1;
            }
            if ((nonMaxed[r, c] < MaxHysteresisThresh) && (nonMaxed[r,
c] >= MinHysteresisThresh))
            {
                EdgePoints[r, c] = 2;
            }
        }
    }
}
```

4.2.7. Implementasi Algoritme *Hysteresis*

Source code 4.7 berikut menunjukkan implementasi algoritme *Hysteresis* sesuai dengan perhitungan manual yang dilakukan sebelumnya.

```
private void HysterisisThresholding(int[,] Edges)
 {
    int i, j;
    int Limit = KernelSize / 2;

    for (i = Limit; i <= (Width - 1) - Limit; i++)
    {
        for (j = Limit; j <= (Height - 1) - Limit; j++)
        {
            if (Edges[i, j] == 1)
            {
                EdgeMap[i, j] = 1;
                Travers(i, j);
                VisitedMap[i, j] = 1;
            }
        }
    }

    return;
}
```

Source code 4.8 berikut menunjukkan implementasi algoritme *Travers* sesuai dengan perhitungan manual yang dilakukan sebelumnya.

```
private void Travers (int X, int Y)
{
    if (VisitedMap[X, Y] == 1)
    {
        return;
    }
    //1
    if (EdgePoints[X + 1, Y] == 2)
    {
        EdgeMap[X + 1, Y] = 1;
        VisitedMap[X + 1, Y] = 1;
        Travers(X + 1, Y);
        return;
    }

    //2
```

```
    if (EdgePoints[X + 1, Y - 1] == 2)
    {
        EdgeMap[X + 1, Y - 1] = 1;
        VisitedMap[X + 1, Y - 1] = 1;
        Travers(X + 1, Y - 1);
        return;
    }

    //3
    if (EdgePoints[X, Y - 1] == 2)
    {
        EdgeMap[X, Y - 1] = 1;
        VisitedMap[X, Y - 1] = 1;
        Travers(X, Y - 1);
        return;
    }

    //4
    if (EdgePoints[X - 1, Y - 1] == 2)
    {
        EdgeMap[X - 1, Y - 1] = 1;
        VisitedMap[X - 1, Y - 1] = 1;
        Travers(X - 1, Y - 1);
        return;
    }

    //5
    if (EdgePoints[X - 1, Y] == 2)
    {
        EdgeMap[X - 1, Y] = 1;
        VisitedMap[X - 1, Y] = 1;
        Travers(X - 1, Y);
        return;
        }

    //6
    if (EdgePoints[X - 1, Y + 1] == 2)
    {
        EdgeMap[X - 1, Y + 1] = 1;
        VisitedMap[X - 1, Y + 1] = 1;
        Travers(X - 1, Y + 1);
        return;
    }

    //7
    if (EdgePoints[X, Y + 1] == 2)
    {
        EdgeMap[X, Y + 1] = 1;
        VisitedMap[X, Y + 1] = 1;
        Travers(X, Y + 1);
        return;
        }

    //8
    if (EdgePoints[X + 1, Y + 1] == 2)
    {
        EdgeMap[X + 1, Y + 1] = 1;
        VisitedMap[X + 1, Y + 1] = 1;
        Travers(X + 1, Y + 1);
        return;
    }

    return;
}
```

4.2.8. Implementasi Algoritme *Evaluate*

Source code 4.9 berikut menunjukkan implementasi algoritme *Evaluate* sesuai dengan perhitungan manual yang dilakukan sebelumnya.

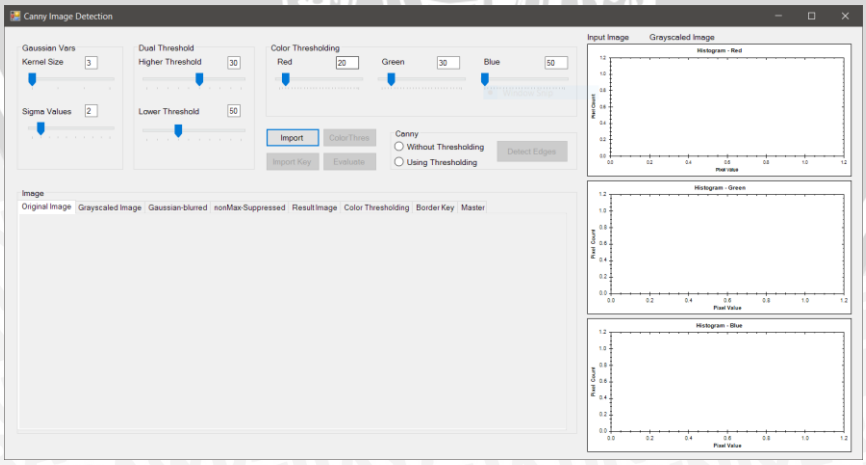
```
private Bitmap Evaluate(Bitmap Cann, Bitmap Key)
{
    Bitmap res = new Bitmap(Cann);
    int buffCann, buffMkey;
    int errCount = 0;

    for (int i = 0; i < cann.Width; i++)
    {
        for (int j = 0; j < cann.Height; j++)
        {
            buffCann = can.GetPixel(i, j).R;
            buffMkey = key.GetPixel(i, j).R;
            if (buffCann == 255 && buffMkey == 0)
            {
                errCount++;
                res.SetPixel(i, j, Color.Red);
            }
        }
    }

    return res;
}
```

4.3 Implementasi Antarmuka

Bagian ini menjelaskan tentang implementasi antarmuka yang telah dirancang pada bab sebelumnya. Implementasi antarmuka terdiri dari beberapa halaman atau tab, diantaranya adalah halaman utama, halaman hasil, dan halaman pengujian.

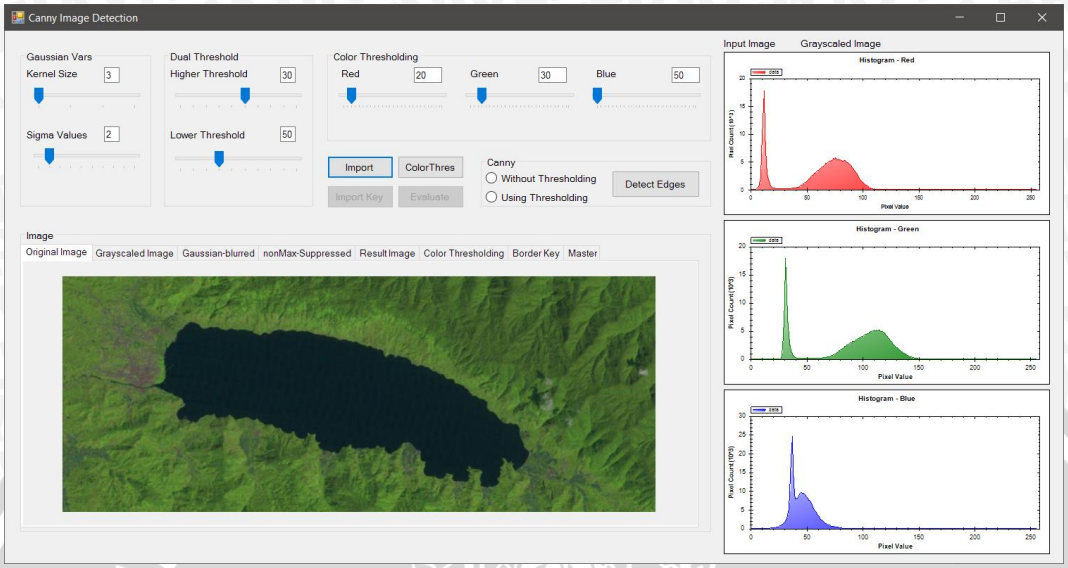
4.3.1. Implementasi Antarmuka Halaman Utama

Implementasi antarmuka halaman utama merupakan tampilan awal saat aplikasi dijalankan. Gambar 4.1 dibawah menunjukkan implementasi antarmuka halaman utama.



Gambar 4.1 Tampilan antarmuka halaman utama

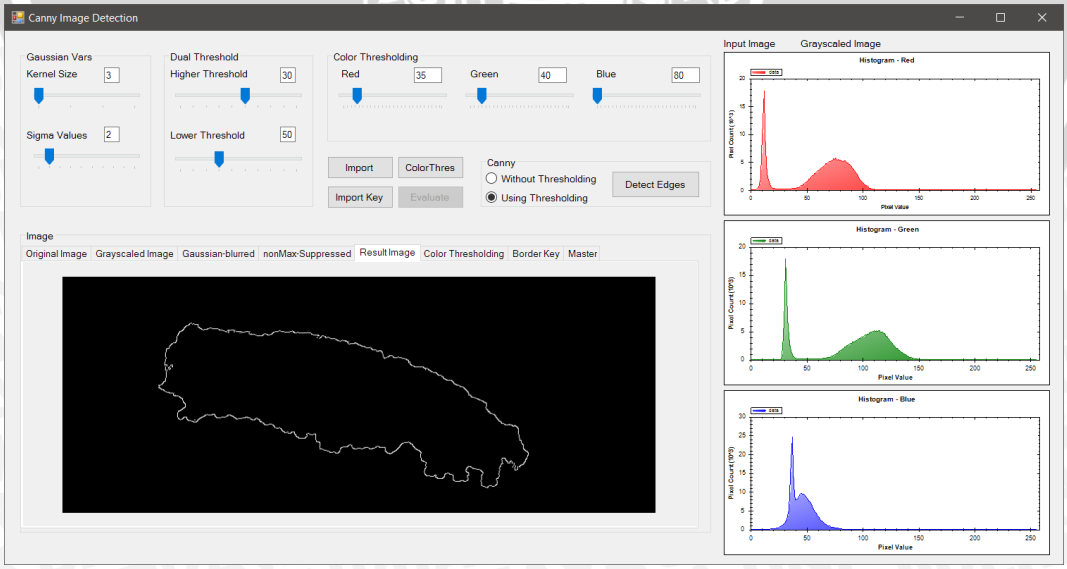
Gambar 4.2 menunjukkan tampilan aplikasi setelah user melakukan impor citra masukan.



Gambar 4.2 Tampilan antarmuka halaman utama dengan citra masukan

4.3.2. Implementasi Antarmuka Halaman Hasil

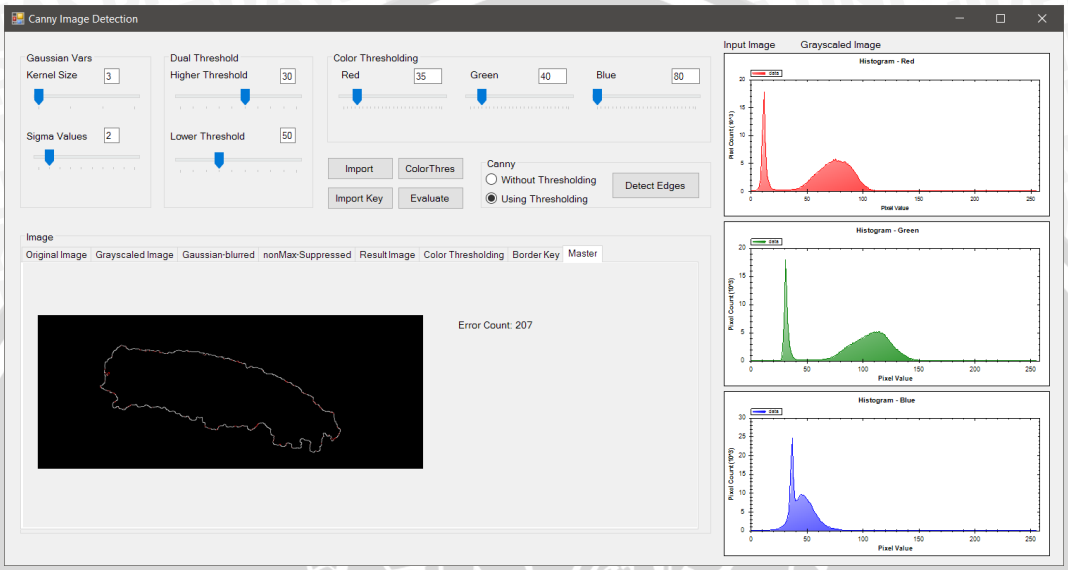
Implementasi antarmuka halaman hasil menampilkan citra olahan hasil deteksi tepi menggunakan algoritme Canny. Gambar 4.3 menunjukkan tampilan antarmuka setelah pengguna melakukan deteksi tepi dengan terlebih dahulu melakukan segmentasi *thresholding* warna.



Gambar 4.3 Tampilan antarmuka halaman hasil dengan citra hasil

4.3.3. Implementasi Antarmuka Halaman Pengujian

Implementasi antarmuka halaman pengujian menampilkan hasil perbandingan citra hasil deteksi tepi dengan citra *Key* yang dijadikan acuan. Setiap piksel tepian yang ditandai dengan merah merupakan *error* atau tepian yang tidak termasuk dalam tepian pada citra acuan. Gambar 4.4 menampilkan antarmuka halaman pengujian.



Gambar 4.4 Tampilan antarmuka halaman pengujian

BAB 5 PEMBAHASAN

Bab ini membahas tentang hasil yang didapatkan dari implementasi yang telah dijelaskan pada bab sebelumnya. Pembahasan meliputi hasil pengujian deteksi tepi yang dilakukan terhadap citra masukan. Pengujian yang dilakukan meliputi pengujian terhadap perubahan parameter Gaussian dan *threshold* yang digunakan dan perbandingan antara hasil deteksi tepi dengan melibatkan proses segmentasi *thresholding* warna serta hasil deteksi tepi tanpa melakukan proses segmentasi.

5.1 Pengujian Parameter Gaussian dan *Threshold*

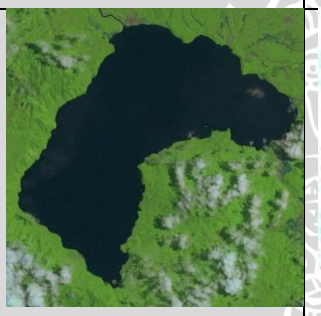
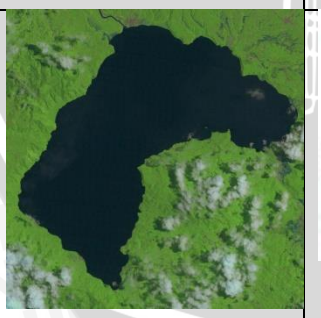
Proses pengujian ini untuk mengetahui nilai parameter *Gaussian* dan *threshold* yang optimal untuk setiap sampel citra yang dijadikan sebagai *input*. Nilai awal parameter ini ditetapkan secara manual, yaitu ukuran *kernel* Gaussian 3, nilai sigma 1, *threshold* bawah 30, dan *threshold* atas 50.

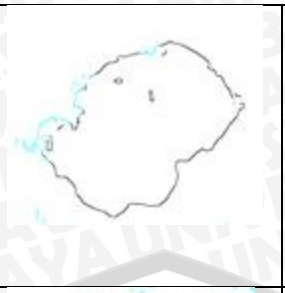
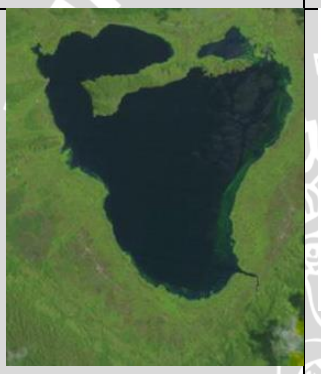
Pada pengujian ini, digunakan 10 citra sampel yang diambil dari citra satelit LANDSAT 4 dan 5, dan telah dilakukan *pre-processing* sebelumnya sehingga citra satelit memiliki format channel RGB. Hasil dari pengujian nilai parameter dapat dilihat pada Tabel 5.1. Citra hasil yang ditampilkan pada Tabel 5.1 telah diberi perlakuan inversi warna (*color inversion*) demi kejelasan penulisan laporan.

Tabel 5.1 Hasil Pengujian Citra Terhadap Perubahan Parameter

No.	Citra Masukan	Citra Hasil	Jumlah piksel tepian	*Error* (piksel)
1			708	1704
2			627	1023
3			444	467
4			700	1044

5			942	6993
6			890	4461
7			530	1364
8			562	1848
9			804	2908

10			713	917
11			722	347
12			626	192
13			2460	14163
14			2007	1886
15			1135	987

16			978	422
17			1234	800
18			1183	523
19			1605	8307
20			1465	853

Tabel 5.2 Parameter yang Digunakan dalam Pengujian Tanpa Segmentasi

No.	Nilai Parameter			
	Ukuran *Kernel*	Sigma	*Threshold* atas	*Threshold* bawah
1	3	1	50	30
2	5	5	35	20
3	3	1	50	30
4	3	3	35	15
5	3	1	50	30
6	5	2	36	12
7	3	1	50	30
8	5	3	26	10
9	3	1	50	30
10	5	4	50	30
11	3	1	50	30
12	5	4	38	30
13	3	1	50	30
14	7	6	50	40
15	3	1	50	30
16	5	5	45	30
17	3	1	50	30
18	5	5	40	30
19	3	1	50	30
20	5	5	55	30

Berdasarkan tabel 5.1, nilai parameter Gaussian berpengaruh terhadap proses menghilangkan *noise* citra yang dapat disalahartikan oleh algoritme Canny sebagai tepian. Hal ini konsisten pada setiap kasus, dan terlihat paling jelas pada kasus nomor 14, dimana perubahan parameter Gaussian dengan cara menaikkan ukuran *kernel* serta memperbesar sigma secara signifikan menghilangkan *noise* pada citra yang diolah. Perubahan parameter Gaussian pada kasus nomor13 ke kasus nomor 14 menyebabkan penurunan nilai *error* sebesar 86%.

Pada kasus nomor 7 dan 8, citra masukan memiliki karakteristik permukaan danau yang tidak begitu diskrit, dan daratan di permukaan danau memiliki banyak *noise*, sehingga ketika dilakukan proses menghilangkan *noise* dengan cara menaikkan nilai parameter Gaussian, tepian yang seharusnya terdeteksi justru terkena penekanan oleh proses Gaussian *filtering*, sehingga nilai *error* meningkat.

Hal yang sama terjadi pada kasus nomor 3 dan 4 dimana tepian permukaan danau tidak dapat ditentukan dengan jelas, sehingga ketika *threshold* bawah diturunkan, terdapat banyak *noise* yang terdeteksi dan menaikkan nilai *error*.

Kasus-kasus lain menunjukkan sebuah pola yang berulang, yaitu menaikkan parameter Gaussian berguna untuk menghilangkan *noise* pada permukaan tanah yang berpotensi dideteksi secara salah oleh algoritme Canny sebagai tepian. Nilai *threshold* bawah dapat diturunkan untuk meningkatkan toleransi batasan nilai piksel yang dianggap sebagai tepian.

Tabel 5.3 Akurasi Deteksi Tepi Algoritme Canny Tanpa Proses Segmentasi

No. Kasus	Tepian Terdeteksi	*Error* Terdeteksi	Tepian Mutlak	Rasio *Error* (%)
1	2412	1704	708	70,65
2	1650	1023	627	62
3	911	467	444	51,26
4	1744	1044	700	59,86
5	7935	6993	942	88,13
6	5351	4461	890	83,37
7	1894	1364	530	72,02
8	2410	1848	562	76,68
9	3712	2908	804	78,34
10	1630	917	713	56,26
11	1069	347	722	32,46
12	818	192	626	23,47
13	16623	14163	2460	85,2
14	3893	1886	2007	48,45
15	2122	987	1135	46,51
16	1400	422	978	30,14
17	2034	800	1234	39,33
18	1706	523	1183	30,66
19	9912	8307	1605	83,81
20	2318	853	1465	36,8
			Rata-rata	57,7
			Deviasi Standar	21,19

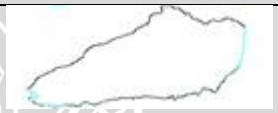
Dari Tabel 5.2 dapat diketahui rasio *error* pada deteksi tepi menggunakan algoritme Canny berkisar antara 23,4% hingga 70%, dengan nilai rata-rata *error* sebesar 57,7% dan deviasi standar sebesar 21,2%. Hal ini dapat diartikan bahwa dari seluruh piksel yang dideteksi oleh algoritme sebagai tepian, jumlah *noise* yang ikut terhitung sebagai tepi lebih banyak daripada tepian yang sebenarnya. Percobaan untuk menurunkan jumlah *noise* dengan cara menaikkan parameter Gaussian berakibat turunnya jumlah tepian sebenarnya yang terdeteksi, dan menyebabkan tepian permukaan danau yang terdeteksi menjadi diskontinu. Hal ini terlihat pada setiap kasus dengan pengecualian kasus 3 dan 4, dimana tepian

yang tidak diskrit menyebabkan nilai parameter Gaussian perlu dinaikkan dan berefek pada naiknya nilai tepian dan *noise* yang terdeteksi.

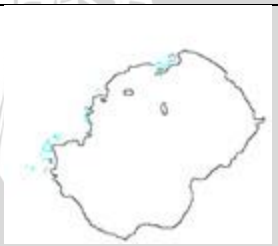
5.2 Pengujian Dengan Melibatkan Proses Segmentasi

Pada pengujian ini, dilakukan perbandingan antara hasil citra keluaran tanpa melibatkan proses segmentasi dengan hasil citra keluaran yang telah melalui proses segmentasi sebelum dilakukan proses deteksi tepi. Tabel 5.4 menunjukkan hasil pengujian citra keluaran dengan melibatkan proses Segmentasi.

Tabel 5.4 Hasil Pengujian Citra Dengan Melibatkan Segmentasi

No	Citra Masukan	*Thresholding* Warna	Citra Hasil	*Error* (piksel)
1		Tidak		1023
2		Ya		170
3		Tidak		1044
4		Ya		771
5		Tidak		4461
6		Ya		444

7		Tidak		1848
8		Ya		227
9		Tidak		917
10		Ya		103
11		Tidak		192

12		Ya		94
13		Tidak		1886
14		Ya		458
15		Tidak		422
16		Ya		247
17		Tidak		523

18		Ya		339
19		Tidak		853
20		Ya		99

Tabel 5.5 Parameter Pengujian Dengan Segmentasi Warna

No	Parameter Gaussian		Parameter *Threshold* Canny		Parameter *threshold* Warna		
	Size	Sigma	tLow	tHigh	R	G	B
1	5	5	35	20	-	-	-
2	3	1	30	10	20	40	50
3	3	3	35	15	-	-	-
4	5	1	50	30	45	90	92
5	5	2	36	12	-	-	-
6	3	2	40	35	35	35	80
7	5	3	26	10	-	-	-
8	3	2	50	30	35	60	90
9	5	4	50	30	-	-	-
10	3	2	30	10	40	35	77
11	5	4	38	30	-	-	-
12	3	5	35	20	30	50	80
13	7	6	50	40	-	-	-
14	3	2	35	20	35	50	100
15	5	5	45	30	-	-	-
16	3	2	45	30	40	55	80

17	5	5	40	30	-	-	-
18	5	3	40	30	50	70	85
19	5	5	55	30	-	-	-
20	3	1	40	30	25	45	75

Berdasarkan Tabel 5.4, dapat dilihat bahwa proses segmentasi dengan menggunakan *thresholding* warna secara signifikan meningkatkan kinerja algoritme deteksi Canny. Proses segmentasi ini diterapkan sebagai salah satu tahap *pre-processing* sebelum melakukan deteksi tepi permukaan danau pada citra. Langkah segmentasi dapat memisahkan bagian permukaan danau dan secara langsung melakukan penekanan terhadap fitur permukaan tanah, sehingga *noise* yang terdapat pada permukaan tanah tidak akan diproses oleh algoritme deteksi tepi Canny.

Selain dapat menghilangkan *noise* yang terdapat di luar tepian danau, proses segmentasi dengan *thresholding* warna menghasilkan tepian danau yang lebih kontinu. Hal ini memungkinkan dilakukannya perhitungan luas permukaan danau berdasarkan tepian yang telah dideteksi.

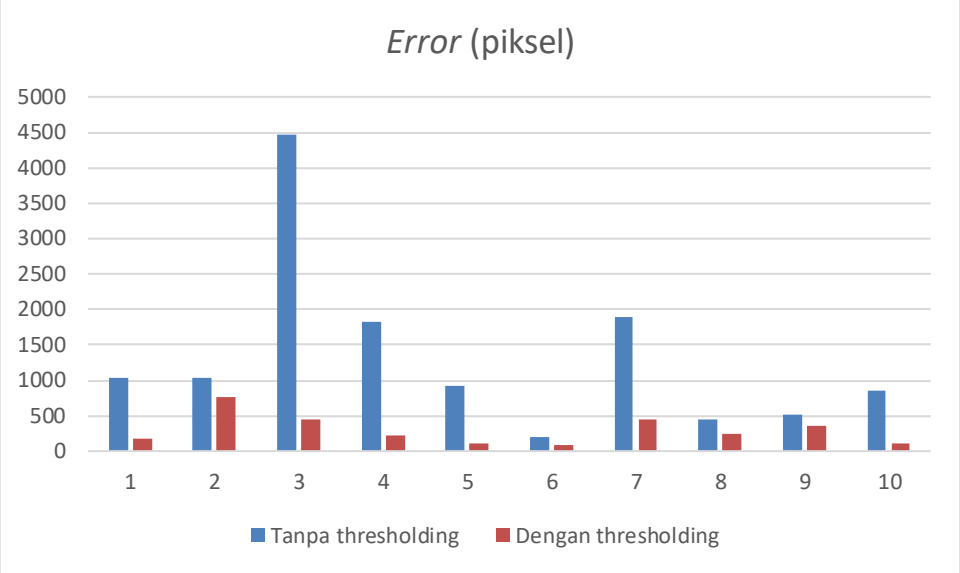
Pada Tabel 5.5 dapat dilihat bahwa kinerja algoritme deteksi tepi lebih bergantung pada parameter *threshold* warna yang digunakan. Parameter *Gaussian* dan *threshold Canny* memiliki efek yang tidak signifikan terhadap kinerja deteksi tepi.

Tabel 5.6 Perbandingan Kinerja Deteksi Tepi dengan Segmentasi

No. Gambar	*Error* (piksel)		Selisih	Penurunan Jumlah *Error* (%)
	Tanpa Segmentasi	Dengan Segmentasi		
Gambar 1	1023	170	853	83,38
Gambar 2	1044	771	273	26,15
Gambar 3	4461	444	4020	90,05
Gambar 4	1828	227	1601	87,58
Gambar 5	917	103	814	88,77
Gambar 6	192	94	98	51,04
Gambar 7	1886	458	1428	75,72
Gambar 8	442	247	195	44,12
Gambar 9	523	339	184	35,18
Gambar 10	853	99	754	88,39
			Rata-rata	67,04

Berdasarkan Tabel 5.6, dapat dilihat bahwa kinerja deteksi tepi meningkat antara 26% hingga 90% setelah diterapkan proses segmentasi. Rata-rata peningkatan kinerja adalah 67%.

Gambar 5.1 menunjukkan grafik perbandingan nilai *error* hasil deteksi tepi dengan proses segmentasi dan tanpa proses segmentasi.



Gambar 5.1 Perbandingan Jumlah *Error*

Tabel 5.7 menunjukkan kinerja algoritme Canny yang telah melibatkan proses segmentasi dengan *thresholding* warna.

Tabel 5.7 Rasio *Error* Pada Algoritme Canny Dengan Segmentasi Warna

Gambar	Tepian Terdeteksi	*Error* Terdeteksi	Tepian Mutlak	Rasio *Error*(%)
1	793	170	623	21,44
2	1114	771	343	69,21
3	1530	444	1086	29,02
4	834	227	607	27,22
5	845	103	742	12,19
6	734	94	640	12,81
7	2776	458	2318	16,5
8	1453	247	1206	17
9	1592	366	1226	22.99
10	1714	99	1615	5,776
			Rata-rata	23,41

Dari Tabel 5.7 diketahui bahwa rata-rata rasio *error* deteksi tepi menurun sebesar 34% dibandingkan dengan rasio *error* deteksi tepi tanpa menggunakan segmentasi dengan *thresholding* warna.

BAB 6 PENUTUP

6.1 Kesimpulan

Berdasarkan hasil penelitian yang telah dilakukan, maka dapat ditarik beberapa kesimpulan sebagai berikut:

1. Deteksi tepi permukaan danau pada citra satelit dapat menggunakan algoritme Canny dengan nilai parameter Gaussian dan *threshold* tidak mempunyai besaran tertentu yang dapat diaplikasikan pada setiap kasus secara umum. Meski demikian, besaran parameter Gaussian memiliki sebuah pola berulang yaitu bertambahnya ukuran *kernel* Gaussian berpengaruh dalam proses reduksi *noise* pada citra masukan yang akan diolah oleh algoritme Canny, sedangkan besaran parameter *threshold* berpengaruh dalam menentukan toleransi piksel tepian sehingga membantu membentuk tepian yang bersifat kontinu.
2. Penerapan algoritme Canny dalam proses deteksi tepi permukaan danau pada citra satelit memiliki nilai rata-rata kesalahan deteksi sebesar 57,7% dengan deviasi standar 21,2%. Lebih lanjut, penerapan segmentasi citra menggunakan metode *color thresholding* meningkatkan kinerja deteksi tepi permukaan danau pada citra satelit dengan rata-rata peningkatan kinerja sebesar 67,04% dengan nilai rata-rata kesalahan deteksi tepi sebesar 23,4%.

6.2 Saran

Berdasarkan penelitian yang telah dilakukan, terdapat beberapa saran yang dapat digunakan untuk pengembangan penelitian lebih lanjut, antara lain:

1. Perlu ada percobaan mengenai otomatisasi penentuan parameter Gaussian serta *threshold* atas dan bawah untuk algoritme Canny. Hal ini dapat dilakukan dengan memasukkan sampel citra melalui proses *Machine Learning* untuk menemukan pola yang ada. Penelitian tentang otomatisasi juga perlu dilakukan untuk penentuan nilai *threshold* warna dalam segmentasi. Hal ini dapat dilakukan dengan cara mencari nilai *threshold* pada histogram tiap channel warna menggunakan konsep *Local Maximum*
2. Perlu ada pengembangan lebih lanjut mengenai manfaat penerapan segmentasi dalam proses *pre-processing* deteksi tepi sebelum pengolahan citra dengan algoritme Canny. Penelitian lebih lanjut dapat menggunakan beberapa metode segmentasi diluar metode segmentasi dengan *thresholding* warna.

DAFTAR PUSTAKA

Ehsan Nadernejad, Sara Sharifzadeh & Hamid Hassanpour, 2008. *Edge Detection Techniques: Evaluations and Comparisons.* Applied Mathematical Sciences, Vol. 2. no. 31, pp 1507 – 1520

Alifiya Jahagirdar, Dr. Manisha Patil, Dr. Vrushsen Pawar & Dr. Vilas Kharat, 2016. *Comparative Study of Satellite Image Edge Detection Techniques.* International Journal of Innovative Research in Computer and Communication Engineering, vol. 4, Issue 5.

Campbell, J. B. 2002. *Introduction to Remote Sensing*. New York London: The Guilford Press

Azriel Rosenfeld, 1969. *Picture Processing by Computer*, New York: Academic Press.

Umbaugh, Scott E. 2010. *Digital image processing and analysis : human and computer vision applications with CVIPtools (2nd ed.).* Boca Raton, FL: CRC Press

Mohsen Sharifi, Mahmoud Fathy, Maryam Tayefeh Mahmoudi, 2002. *A Classified and Comparative Study of Edge Detection Algorithms*. Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'02)

John Canny, 1986. *A Computational Approach to Edge Detection.* IEEE Transactions on Pattern Analysis and Machine Intellegence. Vol. PAMI-8, no. 6, pp 679 – 698.

H. Liu dan K. C. Jezek, 2004. *Automated extraction of coastline from satellite imagery by integrating Canny edge detection and locally adaptive thresholding methods*. International Journal of Remote Sensing. Vol. 25, no. 5, pp 937 – 958.

Nilima Kulkarni, 2012. *Color Thresholding Method for Image Segmentation of Natural Images.* International Journal of Image, Graphics and Signal Processing, Vol 1, pp. 28 – 34

World Wide Web Consortium, 2000. *Techniques For Accessibility Evaluation And Repair Tools* diakses melalui https://www.w3.org/TR/AERT#color-contrast [diakses pada 19 November 2016]

Underwater Research Group, 2011. *Ground Truth for Image Processing* diakses melalui http://urrg.eng.usm.my/index.php?view=article&id=449:ground-truth-for-image-processing [Diakses pada 21 April 2017]

LAMPIRAN

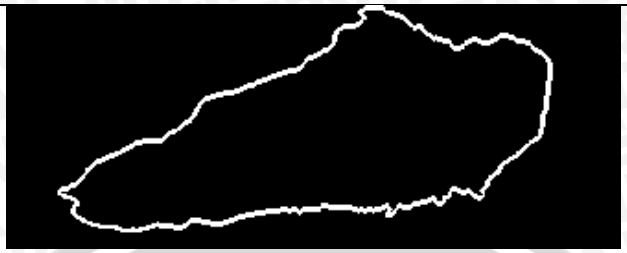
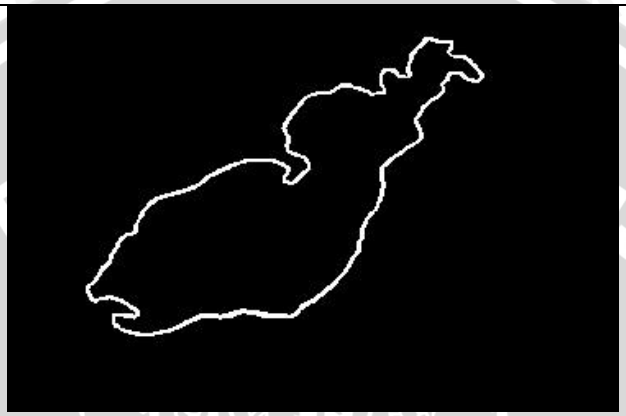
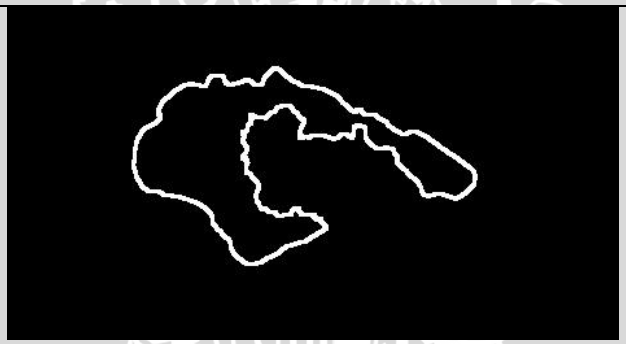
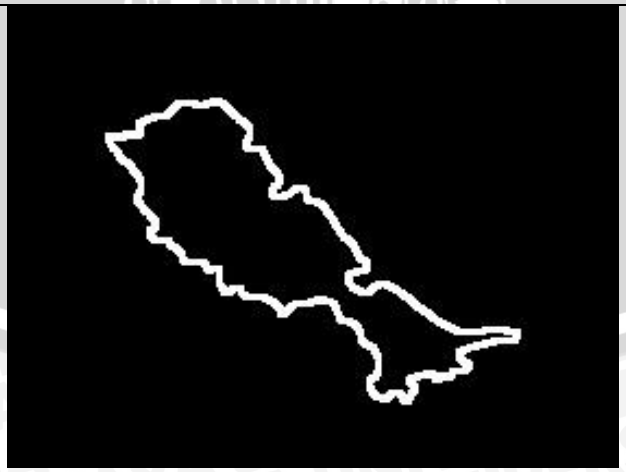
Lampiran A: Sampel Citra Masukan

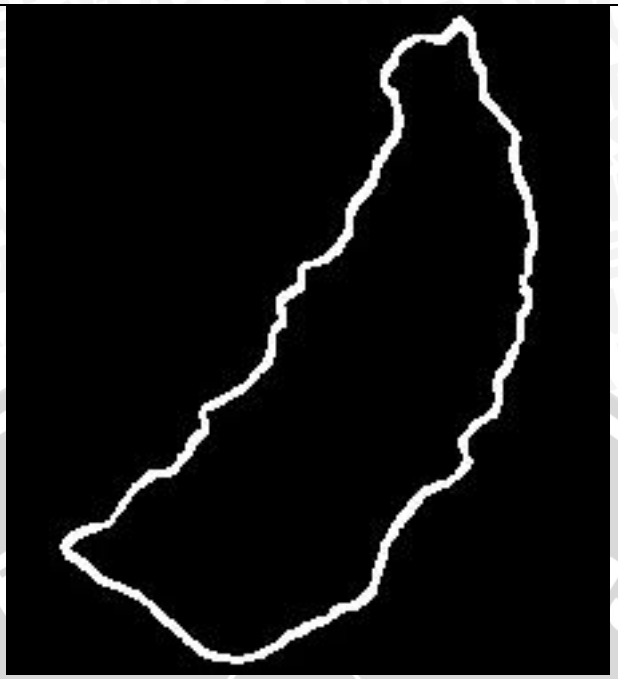
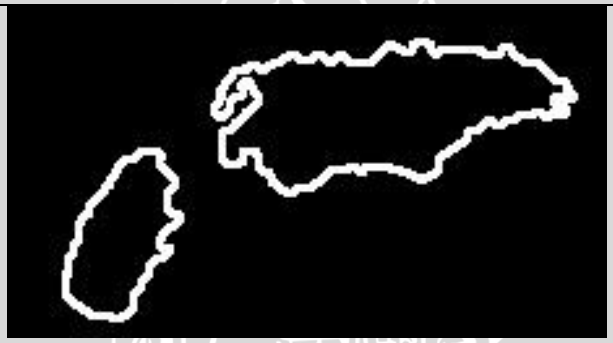
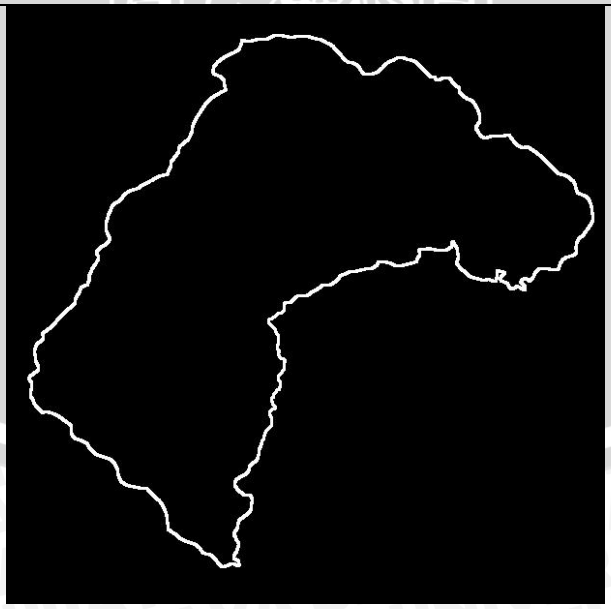
No	Gambar
1	
2	
3	
4	

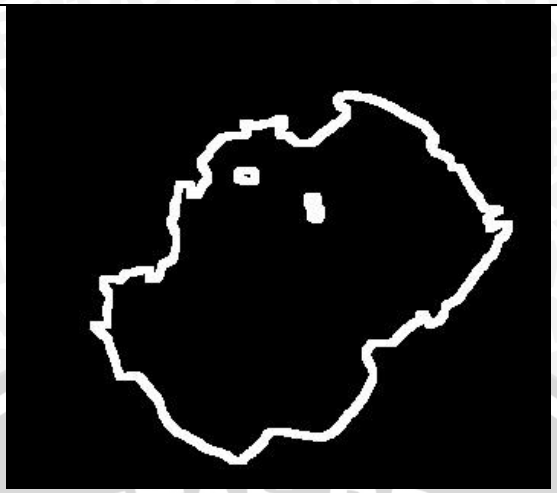
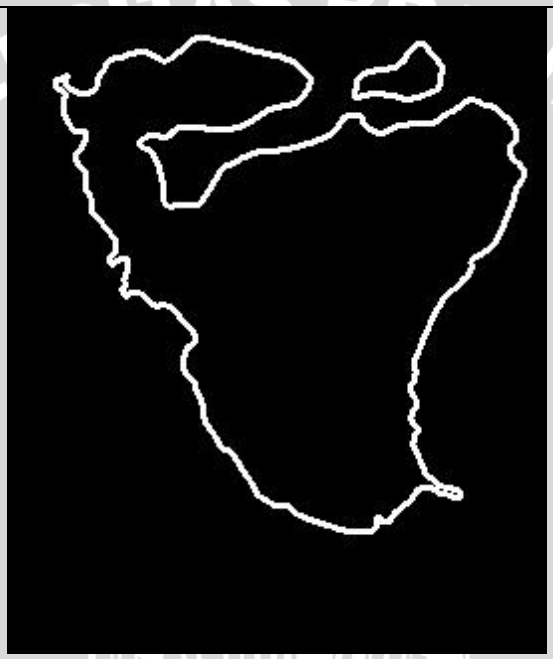
5	
6	
7	

8	
9	
10	

Lampiran B: Sampel Citra Key

No	Gambar
1	
2	
3	
4	

5	
6	
7	

8	
9	
10