IMPLEMENTASI METODE *SCALE INVARIANT FEATURE TRANSFORM* (SIFT) UNTUK *MULTIPLE OBJECT TRACKING* PADA VIDEO CCTV

SKRIPSI

Konsentrasi Komputasi Cerdas Dan Visualisasi

Untuk memenuhi sebagian persyaratan untuk mencapai gelar Sarjana Komputer

Disusun oleh :

ANGGI GUSTININGSIH HAPSANI

NIM. 105060801111009

KEMENTRIAN PENDIDIKAN DAN KEBUDAYAAN

UNIVERSITAS BRAWIJAYA

PROGRAM TEKNOLOGI INFORMASI DAN ILMU KOMPUTER

MALANG

2014

LEMBAR PERSETUJUAN

IMPLEMENTASI METODE *SCALE INVARIANT FEATURE TRANSFORM* (SIFT) UNTUK *MULTIPLE OBJECT TRACKING* PADA VIDEO CCTV

SKRIPSI

Konsentrasi Komputasi Cerdas Dan Visualisasi

Untuk memenuhi sebagian persyaratan untuk mencapai gelar Sarjana Komputer

Disusun oleh :

ANGGI GUSTININGSIH HAPSANI

NIM. 105060801111009

Telah diperiksa dan disetujui oleh :

Dosen Pembimbing I	Dosen Pembimbing II
Imam Cholissodin, S.Si, M.Kom	**Ahmad Afif Supianto, S.Si, M.Kom**
NIP. 850719 16 1 1 0422	**NIP. 820623 16 1 1 0425**

LEMBAR PENGESAHAN

IMPLEMENTASI METODE *SCALE INVARIANT FEATURE TRANSFORM (SIFT) UNTUK MULTIPLE OBJECT TRACKING PADA VIDEO CCTV*

SKRIPSI

Konsentrasi Komputasi Cerdas Dan Visualisasi

Untuk memenuhi sebagian persyaratan untuk mencapai gelar Sarjana Komputer

Disusun oleh :

Anggi Gustiningsih Hapsani

NIM. 105060801111009

Skripsi ini telah diuji dan dinyatakan lulus pada

tanggal 2 Juli 2014

Dosen Penguji I	Dosen Penguji II
Drs. Marji, MT.	**Nurul Hidayat, S.Pd., M.Sc**
NIP. 19670801 199203 1 001	**NIP. 19680430 200212 1 001**

Dosen Penguji III

Indriati, ST., M.Kom

NIK. 831013 06 1 2 0035

Mengetahui

Ketua Program Studi Informatika/Ilmu Komputer

Drs. Marji, MT.

NIP. 19670801 199203 1 001

PERNYATAAN
ORISINALITAS SKRIPSI

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 dikutip dalam naskah ini dan disebutkan dalam sumber kutipan dan 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, 3 Juli 2014

Mahasiswa,

Anggi Gustiningsih Hapsani

NIM 105060801111009

KATA PENGANTAR

Dengan nama Allah SWT Yang Maha Pengasih dan Penyayang. Segala puji bagi Allah SWT karena atas rahmat dan hidayahNya penulis dapat menyelesaikan skripsi yang berjudul "Implementasi Metode *Scale Invariant Feature Transform* (SIFT) Untuk *Multiple Object Tracking* Pada Video CCTV". Skripsi ini disusun untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Komputer di Proram Studi Informatika Program Teknologi Informasi dan Ilmu Komputer Universitas Brawijaya Malang.

Dalam menyelesaikan skripsi ini, banyak bantuan, bimbingan, dorongan dan doa yang telah diberikan kepada penulis. Oleh karena itu, dalam kesempatan ini penulis ingin menyampaikan rasa terima kasih penulis kepada :

1. Imam Cholissodin, S.Si, M.Kom selaku dosen pembimbing pertama yang telah meluangkan waktu memberikan pengarahan dan bimbingan kepada penulis.
2. Ahmad Afif Supianto, S.Si, M.Kom selaku dosen pembimbing kedua yang telah meluangkan waktu memberikan pengarahan dan bimbingan kepada penulis.
3. Drs. Marji, MT. dan Issa Arwani, S.Kom, M.Sc , selaku Ketua dan Sekretaris Program Studi Teknik Informatika, Program Teknologi Informasi & Ilmu Komputer Universitas Brawijaya.
4. Segenap bapak dan ibu dosen yang telah mendidik dan mengajarkan ilmunya kepada penulis selama menempuh pendidikan di Program Teknologi Informasi & Ilmu Komputer Universitas Brawijaya.
5. Segenap staf, karyawan dan civitas di Program Teknologi Informasi & Ilmu Komputer Universitas Brawijaya yang telah banyak membantu penulis dalam pelaksanaan penyusunan skripsi ini.
6. Wagi dan Sunartini, S.Pd selaku orang tua dari penulis yang senantiasa menyemangati, menasehati, dan mendoakan penulis dalam proses mengerjakan skripsi.

7. Nurizal Dwi Priandani, yang senantiasa menemani, mendengarkan keluhan, memberikan motivasi, semangat dan doa kepada penulis.
8. Seluruh mahasiswa Program Teknologi Informasi & Ilmu Komputer angkatan 2010, khususnya teman – teman TIF F yang telah membantu terealisasinya skripsi ini.
9. Semua pihak yang telah membantu terselesaikannya penyusunan skripsi ini yang tidak dapat disebutkan satu per satu.

Penulis sadar bahwa skripsi ini masih jauh dari kesempurnaan, untuk itu dengan segala kerendahan hati penulis mengharapkan saran dan kritik yang membangun. Semoga skripsi ini dapat bermanfaat untuk semua pihak.

Malang, 3 Juli 2014

Penulis

ABSTRAK

Anggi Gustiningsih Hapsani. 2014: Implementasi Metode *Scale Invariant Feature Transform* (SIFT) Untuk *Multiple Object Tracking* Pada Video CCTV. Skripsi Program Studi Teknik Informatika, Program Teknologi Informasi dan Ilmu Komputer, Universitas Brawijaya.

Dosen Pembimbing: Imam Cholissodin, S.Si, M.Kom dan Ahmad Afif Supianto, S.Si., M.Kom.

Salah satu teknologi multimedia yang sangat populer saat ini adalah *multiple object tracking*. *Multiple object tracking* adalah sebuah teknik atau metode yang digunakan untuk melakukan pelacakan terhadap banyak objek bergerak pada video. Sebelum ada teknologi ini, pelacakan objek pada sistem keamanan dilakukan secara manual dengan kamera CCTV, dimana sistem tersebut hanya mampu menampilkan dan menyimpan video yang telah direkam tanpa memiliki kemampuan untuk melacak secara otomatis. Salah satu metode *multiple object tracking* yang populer adalah metode SIFT.

Aplikasi ini mengimplementasikan teknologi *multiple object tracking* yang terdiri dari 2 proses yaitu proses deteksi objek dan proses pelacakan objek. Proses deteksi objek diterapkan dengan menggunakan metode *background substraction*. Sedangkan proses pelacakan objek diterapkan dengan menggunakan metode SIFT.

Hasil pengujian akurasi menunjukkan bahwa aplikasi ini memiliki nilai akurasi terbesar yakni 74,2% pada nilai *scale Gaussian* = 0,707, r=1,01, dan *threshold matching* 0,5. Pada hasil pengujian parameter terbaik terhadap 10 variasi data video didapatkan akurasi rata-rata sebesar 87,82%. Sedangkan pada hasil pengujian *centroid* didapatkan error rate objek 1 sebesar 2.61% dan objek 2 sebesar 5.11%. Sehingga dapat disimpulkan bahwa aplikasi ini memiliki kehandalan yang cukup untuk melakukan pelacakan objek secara *multiple*. Selain itu sistem telah mampu mendeteksi dan melacak region dari tiap objek secara baik dengan tingkat *error rate* yang kecil.

Kata Kunci *:* *multiple object tracking, CCTV, SIFT*

ABSTRACT

Anggi Gustiningsih Hapsani. 2014: Implementation Scale Invariant Feature Transform (SIFT) For Multiple Object Tracking in CCTV Video.

Advisor : Imam Cholissodin, S.Si, M.Kom dan Ahmad Afif Supianto, S.Si., M.Kom.

One of popular multimedia technology today is multiple object tracking. Multiple object tracking is a technique or method used to perform the tracking of many moving objects in the video. Before this technology appear, object tracking in security system is done manually with CCTV cameras, where the system is only capable for displaying and storing recorded videos without having the ability to track automatically. One popular method of multiple object tracking is SIFT.

This application implement multiple object tracking technology which consists of two processes, they are object detection and object tracking process. Object detection process is applied using background substraction . And object tracking process is applied by using the SIFT method.

The test results show that the accuracy of these applications have the greatest accuracy value of 74.2% on the value of the Gaussian scale = 0.707, r = 1.01, and the matching threshold of 0.5. On the test results of the best parameter from 10 variations video obtained an average accuracy 87.82%. While on the test results obtained centroid error rate of object 1 at 2.61% and object 2 at 5,11%. It can be concluded that the application has sufficient reliability to perform in multiple object tracking. In addition the system has been able to detect and track objects in a region of each well with a small level of error rate.

Keyword : *multiple object tracking, CCTV, SIFT*

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi multimedia saat ini telah berkembang sangat pesat. Berbagai jenis aplikasi multimedia telah dikembangkan dengan tujuan untuk mempermudah kehidupan manusia sehari-hari. Salah satu teknologi multimedia yang sangat populer saat ini adalah *object tracking*. *Object tracking* adalah sebuah teknik atau metode yang digunakan untuk melakukan pelacakan terhadap sebuah objek bergerak, sehingga pergerakan objek tersebut dapat dideteksi dengan tetap memperhatikan perubahan–perubahan yang terjadi di sekitar objek tersebut [SHA-12].

Perlu diketahui sebelumnya bahwa sebelum ada teknologi ini, pelacakan pada objek dilakukan secara manual dengan kamera CCTV (*Close Circuit Television*) yang melibatkan mata manusia sebagai pengekstraksi citra utama [AGP-11]. Sistem CCTV tersebut telah dioperasikan di berbagai tempat dengan tujuan tertentu baik sebagai sistem keamanan ataupun sebagai sistem pemantau. Di toko misalnya, sistem CCTV digunakan untuk mengamankan toko dari tindakan kriminalitas sekaligus melakukan pemantauan terhadap aktifitas pengunjung. Di tempat umum seperti bandara, terminal dan stasiun untuk mengamankan dari tindakan terorisme. Selain itu, sistem CCTV juga diterapkan di sekolah yang digunakan sebagai sistem pengamanan dan juga pemantau aktifitas murid, sehingga guru akan mengetahui siapa saja murid yang sering datang terlambat, mencontek saat ujian dan melakukan tindakan yang menyalahi aturan lainnya [LIU-12]. Di PTIIK, sistem CCTV digunakan sebagai sistem keamanan dan pemantau aktifitas yang terjadi di titik-titik penting. Namun, semua sistem pelacakan CCTV tersebut hanya mampu menampilkan dan menyimpan video yang telah direkam tanpa memiliki kemampuan untuk melacak secara otomatis. Sehingga sistem tidak mampu menghasilkan sebuah informasi penting dari keadaan video yang telah terekam. Apabila pengguna membutuhkan informasi

yang merujuk pada data video, maka pengguna harus memutar kembali video tersebut. Hal ini tentunya membutuhkan waktu yang lama, padahal di era saat ini dibutuhkan sistem yang mampu menghasilkan sebuah informasi yang cepat dan tepat untuk dijadikan sebuah solusi atas sebuah permasalahan. Belum lagi permasalahan *human error* yang muncul akibat kekurangan dari indera manusia. Oleh karena itu dengan adanya teknologi ini maka semua proses pelacakan tersebut bisa dilakukan secara otomatis oleh sistem itu sendiri.

Multiple object tracking adalah sebuah metode atau jenis pelacakan yang memungkinkan pendeteksian lebih dari satu objek dalam satu *frame* secara bersamaan. Metode ini sedikit berbeda dengan *Single Object Tracking* yang hanya memiliki kemampuan untuk mendeteksi satu objek bergerak tiap *frame*.

Object tracking secara umum mencakup dua tahap, yaitu deteksi objek dan pelacakan objek. Tahap deteksi objek adalah bagian dari *object tracking* untuk mendeteksi sebuah objek yang ada pada tiap *frame* video untuk dilacak pada tahap selanjutnya. Salah satu metode deteksi objek adalah *background substraction*. Metode ini cukup sederhana dan mampu menghasilkan perubahan dari tiap region citra dengan baik, sehingga sangat sesuai untuk mendeteksi perubahan objek pada video [SAI-09]. Proses perhitungan hanya dilakukan dengan mencari selisih dari dua citra yang berbeda. Perhitungan sederhana inilah yang membuat metode ini sangat efisien dalam segi penggunaan memori dan sisi kecepatan. Tahap pelacakan adalah tahap untuk menentukan pergerakan dari objek yang berhasil terdeteksi. Berbagai metode dikembangkan dalam masalah *object tracking* contohnya SIFT, Mean Shift, CAMSHIFT, SURF (*Speeded-Up Robust Features*) dan GLOH. Metode SIFT *(Scale Invariant Feature Transform*) adalah sebuah metode yang dirancang oleh Lowe pada tahun 2004 untuk memecahkan permasalah rotasi citra, penskalaan, perubahan sudut pandang, *noise*, *affine deformation* serta perubahaan pencahayaan yang memiliki ketahanan yang kuat . Kemudian pada tahun 2012, Haopeng Li and Markus Flierl menggunakan metode SIFT yang diimplementasikan untuk memecahkan masalah *multi-view object tracking* dengan menggunakan 3D fitur terhadap objek pemain sepak bola. Pada penelitian ini didapatkan hasil bahwa metode SIFT yang diimplementasikan dengan menggunakan fitur 3D pada objek memiliki keakuratan 10% lebih tinggi

jika dibandingkan dengan menggunakan fitur 2D pada objek [HAO-12]. Hal ini membuktikan bahwa metode SIFT memiliki kehandalan dalam menentukan titik lokasi yang cocok antar *frame* video meskipun dilakukan pada video yang memiliki sudut pandang (*view*) yang berbeda. Kelebihan metode SIFT adalah kemampuannya dalam mendeteksi dan mendeskripsikan fitur-fitur lokal dari citra dalam data video. Hasil dari deteksi dan pendeskripsian fitur–fitur lokal itu dapat dipergunakan dalam pelacakan objek bergerak. SIFT memiliki ketahanan yang kuat terhadap penskalaan, rotasi dan perubahan sudut pandang citra.

Secara efektifitas, metode SIFT memiliki hasil analisis dengan tingkat keakuratan yang tinggi. Hal tersebut dapat dilihat dari perbandingan dengan metode SURF misalnya, dimana metode SURF memerlukan waktu eksekusi 0.546 detik dengan *match feature point* antara *frame* satu dengan *frame* kedua sebanyak 28 titik, sedangkan pada metode SIFT memerlukan waktu eksekusi 1,543 detik dengan *match feature point* sebanyak 41 titik [PAN-13]. Berdasarkan hal tersebut, maka dapat diketahui bahwa metode SIFT lebih unggul dalam hal kehandalan dibandingkan dengan metode SURF. Dengan kehandalan metode SIFT inilah, penulis memilih metode SIFT sebagai metode penyelesaian masalah pada sistem *multiple object tracking* yang akan dibangun nantinya.

Berdasarkan latar belakang yang dijelaskan tersebut, pada tugas akhir ini penulis berinisiatif mengangkat judul "**Implementasi Metode *Scale Invariant Feature Transform* (SIFT) Untuk *Multiple Object Tracking* Pada Video CCTV**". Diharapkan permasalahan mengenai sistem pelacakan secara manual dapat diatasi oleh sistem ini yang mampu melakukan pelacakan dengan mengimplementasikan metode SIFT (*Scale Invariant Feature Transform*) dalam proses *object tracking* secara optimal.

1.2 Rumusan Masalah

Berdasarkan uraian latar belakang di atas, maka dapat dirumuskan permasalahannya sebagai berikut :

1. Bagaimana mengimplementasikan metode SIFT untuk *multiple object tracking* pada video CCTV.

2. Bagaimana mengukur tingkat kinerja metode SIFT saat digunakan untuk *multiple object tracking* pada video CCTV.

1.3 Tujuan

Tujuan penyusunan skripsi ini adalah :

1. Mengimplementasikan metode SIFT untuk *multiple object tracking* pada video CCTV.
2. Mengetahui kinerja dari metode SIFT apabila digunakan untuk *multiple object tracking* pada video CCTV.

1.4 Batasan Masalah

Mengacu pada permasalah yang telah dirumuskan, maka aspek yang dibahas berkaitan dengan skripsi ini antara lain :

1. Sistem yang dibuat menerima masukan berupa citra video yang terekam kamera *CCTV* di PTIIK dengan *frame rate* 30 fps.
2. Sistem melakukan pelacakan terhadap objek dengan pergerakan normal (seperti berjalan dengan kecepatan normal) pada jarak maksimal 12 meter dari kamera.
3. Sistem memiliki peluang untuk mendeteksi dan melacak objek selain manusia karena tidak ada fitur khusus untuk mendeteksi human dalam sistem ini.
4. Sistem akan memberikan konfirmasi berupa notif atau tanda berupa *message box* sebagai respon lanjutan dari adanya objek yang telah terlacak pada video.
5. Sebelum melakukan pelacakan terhadap objek, sistem akan terlebih dahulu melakukan proses deteksi objek dengan menggunakan metode *background substraction*.

1.5 Manfaat

Penulisan skripsi ini diharapkan mempunyai manfaat yang baik. Adapun manfaat yang diharapkan adalah sebagai berikut :

1. Mendapatkan gambaran bagaimana mengimplementasikan teknologi *multiple object tracking* dengan metode SIFT pada video CCTV.
2. Mendapatkan gambaran mengenai hasil kinerja metode SIFT saat diimplementasikan pada *multiple object tracking* untuk video CCTV.

1.6 Sistematika Penulisan

Sistematika penulisan dalam skripsi ini sebagai berikut :

BAB I Pendahuluan

Menguraikan tentang latar belakang, rumusan masalah, tujuan, batasan masalah, manfaat, tujuan, dan sistematika penulisan.

BAB II Kajian Pustaka dan Dasar Teori

Menguraikan tentang dasar teori dan referensi yang mendasari implementasi metode SIFT (*Scale Invariant Feature Transform*) untuk *multiple object tracking* pada video CCTV.

BAB III Metode Penelitian dan Perancangan

Menguraikan tentang metode dan langkah kerja dalam mengimplementasikan metode SIFT untuk *multiple object tracking* pada video CCTV. Langkah-langkah yang dijelaskan dalam bab ini meliputi studi literatur, analisis kebutuhan, analisis komponen, modifikasi kebutuhan, perancangan perangkat lunak, implementasi perangkat lunak dan pengujian perangkat lunak.

BAB IV Implementasi

Membahas implementasi dari sistem pelacakan yang sesuai dengan perancangan pernagkat lunak yang telah dibuat.

BAB V Pengujian dan Analisis

Menguraikan hasil pengujian dan analisis terhadap sistem yang telah direalisasikan.

BAB VI Penutup

Memuat kesimpulan yang diperoleh dari pembuatan dan pengujian perangkat lunak yang dikembangkan dalam skripsi ini serta saran-saran untuk pengembangan lebih lanjut.

BAB II
KAJIAN PUSTAKA DAN DASAR TEORI

Bab ini berisi tinjauan pustaka yang meliputi kajian pustaka dan dasar teori yang diperlukan dalam proses pembuatan skripsi. Kajian pustaka membahas penelitian yang telah ada sebelumnya dan yang akan diusulkan. Dasar teori membahas teori yang diperlukan untuk menyusun skripsi dengan topik yang diusulkan.

Kajian pustaka pada skripsi ini membahas mengenai penelitian sebelumnya yang berjudul '*Multi-Object Tracking in Video Sequences Based on Background Subtraction and* SIFT *Feature Matching'*. Teori dasar yang akan dibahas pada bab ini yaitu konsep dasar *Video Digital*, Kamera CCTV, *Multiple Object Tracking*, Metode SIFT, *background substraction, grayscale, median* dan *min filter* serta penentuan batas (*boundary*).

2.1 Kajian Pustaka

Kajian pustaka pada skripsi ini membahas mengenai penelitian sebelumnya yang berjudul '*Multi-Object Tracking in Video Sequences Based on Background Subtraction and* SIFT *Feature Matching'*. Penelitian tersebut melakukan proses *multiple object tracking* dengan lingkungan video yang statis dimana objek yang menjadi target adalah semua objek (manusia dan benda) terlepas dari *background* statisnya. Metode yang digunakan pada penelitian ini adalah metode gabungan dari *Scale Invariant Feature Transform (*SIFT*)* dimana proses implementasinya menggunakan fungsi matlab.

Hasil yang diperoleh dari penelitian ini adalah perbandingan tingkat *error rate* dari gabungan metode SIFT dengan metode gabungan *Kalman Filter* dan SIFT pada dua objek yang dijadikan sebagai objek pengujian pada semua *frame* video. *Error rate* semua *frame* pada *tracking* orang pertama hasil metode SIFT sebesar 4,29 % sedangkan pada metode *Kalman Filter* dan SIFT sebesar 15,38%. *Error rate* pada *tracking* orang kedua hasil metode SIFT sebesar 2,61 % sedangkan pada metode *Kalman Filter* dan SIFT sebesar 9,02%. Total waktu yang

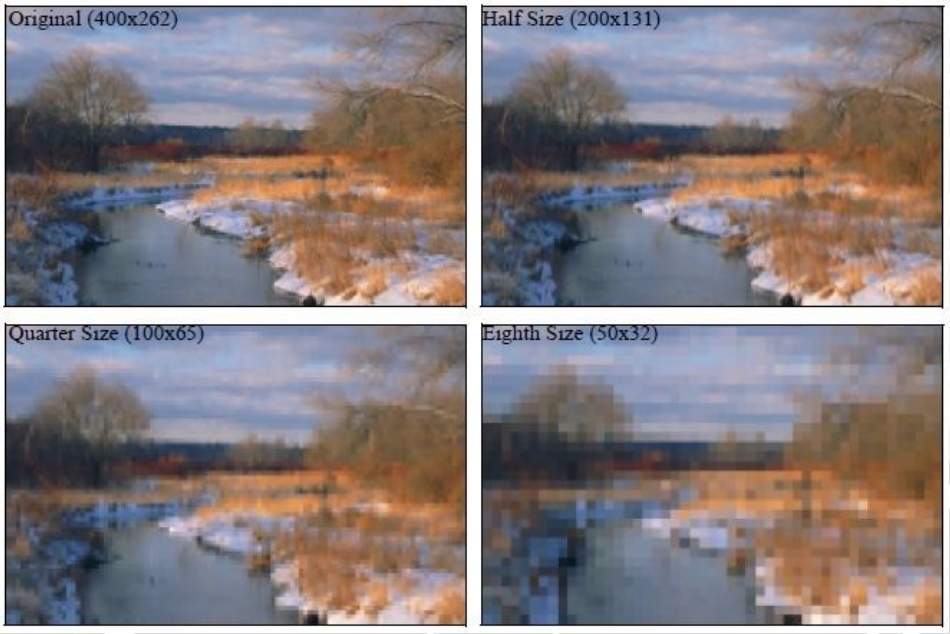
diperlukan metode SIFT pada video *sequence* sebesar 42,65 detik dan pada metode *Kalman Filter* dan SIFT sebesar 28,50 detik [SAI-09].

Perbedaan yang dibuat penulis pada penyusunan skripsi ini adalah proses implementasi metode *background substraction* dan *scale invariant feature transform* yang tanpa menggunakan *library*. Sehingga proses kalkulasi untuk metode SIFT akan di-*code* secara manual.

2.2 Citra Digital

Citra adalah sebuah representasi dua dimensi yang tersusun atas elemen kecil yang disebut dengan piksel. Setiap piksel merepresentasikan warna dari satu titik dalam citra tersebut. Warna dalam hal ini dapat berupa level keabu-abuan dari gambar hitam putih, atau nilai RGB dari suatu gambar berwarna [SHA-12]. Secara umum, piksel mengarah pada tiga warna dasar yaitu *red*, *green*, dan *blue.* Piksel akan disusun dalam bentuk matriks yang memiliki kolom dan baris sesuai dengan ukuran citra tersebut. Susunan piksel yang mengandung nilai warna inilah yang nantinya akan membentuk tampilan citra seperti foto.

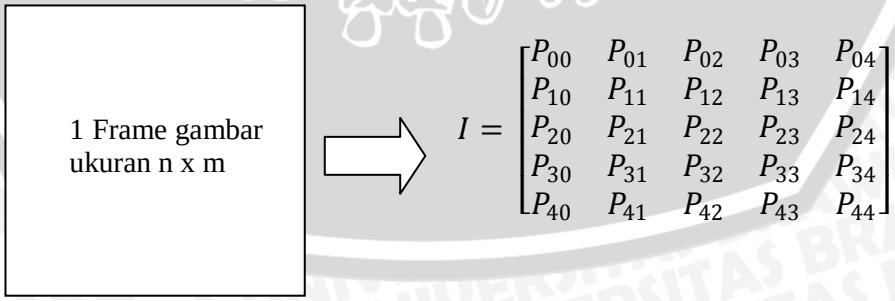
Semakin banyak titik-titik piksel yang menyusun suatu citra, maka penampakan citra tersebut akan terlihat semakin jelas. Kepadatan piksel yang menyusun suatu citra disebut dengan istilah resolusi. Semakin tinggi resolusi citra, maka semakin tinggi kualitas dan semakin banyak informasi yang bisa ditangkap citra tersebut [SAC-99]. Gambar 2.1 mengilustrasikan berbagai nilai resolusi beserta kualitas citranya.



Gambar 2.1 Macam-macam resolusi citra

Sumber : [SAC-99]

Citra jika dihubungkan dengan data video, maka satu citra atau gambar dikenal dengan satu *frame*. Misalnya dikatakan resolusinya sebesar 400 x 262 berarti panjang piksel horizontal citra tersebut sebesar 400 dan panjang piksel vertikalnya sebesar 262 sehingga total keseluruhan piksel yang menyusunnya sebesar 104800 piksel. Dimensi matriks yang merepresentasikan 1 *frame* citra dengan ukuran N x M ditunjukkan pada Gambar 2.2.



Gambar 2.2 Dimensi Maktriks NxM

Sumber : [WID-11]

Citra dapat dibagi menjadi dua yaitu citra kontinyu dan citra digital. Citra kontinyu adalah sebuah citra yang didapat dari sensor optik yang mampu menerima sinyal analog, sedangkan citra digital adalah citra yang didapatkan dari hasil digitalisasi citra kontinyu yang didapatkan pada sensor optik. Citra digital ini dapat disimpan dan dimanipulasi ke dalam komputer.

2.3 Video Digital

Video digital mengarah ke proses menangkap, memanipulasi dan menyimpan citra bergerak yang bisa ditampilkan di layar komputer [LTG-99]. Video merupakan gabungan dari citra-citra yang sebetulnya pasif (mati, tidak bergerak) yang ditampilkan secara berurutan dan beruntun dengan kecepatan tinggi pada rentan waktu tertentu. Satu citra pada rangkaian tersebut disebut dengan istilah *frame* sedangkan kecepatan pembacaan *frame* disebut dengan istilah *frame rate*, dengan satuan fps (*frame per second*). Halus atau tidaknya pergerakan video bergantung pada besar *frame rate* video tersebut. Semakin tinggi *frame rate* video maka pergerakan video tersebut semakin halus [SHA-12]. Karena *frame rate* inilah, citra-citra yang sebetulnya pasif dan mati tersebut terlihat bergerak seiring dengan perpindahannya antara *frame* satu dengan *frame* lainnya.

Karakteristik video digital ditentukan oleh resolusi (*resolution*) atau dimensi *frame* (*frame dimention*), kedalaman piksel (*pixel depth*) dan laju *frame*(*frame rate*) [SHA-12].

- Resolusi atau dimensi *frame* adalah besar ukuran tiap *frame* yang menyusun suatu video digital. Semakin tinggi resolusi pada video maka semakin baik kualitas video tersebut. Hal ini sama dengan karakteristik pada citra, bahwa semakin besar resolusi pada citra maka citra tersebut semakin jelas dan detail dalam menyimpan informasi.
- Kedalaman bit (*bit depth*) dinyatakan dalam bit/piksel. Kedalaman bit menentukan jumlah bit yang digunakan dalam merepresentasikan tiap piksel yang menyusun *frame*. Semakin banyak jumlah bit yang digunakan untuk merepresentasikan sebuah piksel maka semakin tinggi kedalaman pikselnya maka semakin tinggi pula kualitas videonya.

- Laju *frame* (*frame rate*) menunjukkan jumlah *frame* yang ditampilkan tiap detik. Laju *frame* dinyatakan dalam satuan fps atau *frame*/detik.

Banyak sekali ekstensi video digital pada saat ini, salah satunya adalah ekstensi .wmv. Ekstensi ini memiliki kelebihan yaitu wadah ASF opsional dapat mendukung manajemen hak digital menggunakan kombinasi kriptografi kurva *elivtik* pertukaran kunci, DES block *cipher*, blok *cipher* kustom, RC4 *stream cipher* dan SHA-1 fungsi *hashing* dan bagus untuk sistem video dan animasi serta audio

2.4 Kamera CCTV (*Closed Circuit Television*)

Close Circuit Television yang disingkat CCTV adalah sebuah sistem video pribadi dan tertutup dimana tampilan video hanya dapat diakses oleh kelompok atau kalangan tertentu [IPC-10]. CCTV digunakan untuk memantau dan merekam situasi pada lokasi sekitar kamera secara *realtime*. CCTV biasanya dioperasikan di berbagai tempat seperti di dalam bangunan, kompleks bangunan, kampus-kampus besar dan pusat keramaian kota bahkan dapat mencakup seluruh benua. Pemantauan pada CCTV biasanya dilakukan untuk keperluan keamanan, keselamatan, industri ataupun untuk kepentingan pribadi.

Sistem CCTV terdiri dari beberapa kamera CCTV yang nantinya akan terhubung ke sebuah atau lebih monitor yang menjadi pusat kontrolnya. Hal ini memungkinkan sistem CCTV dapat melakukan pemantauan terhadap lokasi yang berbeda dalam waktu sama. Citra-citra yang telah tertangkap kamera akan dikirim ke monitor dan direkam ke dalam kaset video seperti VHS, CD dan DVD sebagai informasi digital, sehingga pemantauan terhadap video CCTV bisa dilakukan oleh pemantau dengan mengamati monitor secara langsung atau melakukan tinjauan ulang dengan memutar kembali rekaman video apabila diperlukan [AGP-11].

Kamera CCTV secara umum dibagi menjadi dua jenis yaitu kamera statis dan bergerak [IPC-10].

- Kamera CCTV statis adalah sebuah kamera yang ditujukan untuk melakukan pemantauan pada suatu titik lokasi tetap sepanjang waktu. Kamera ini bisa diset pencahayaannya sesuai dengan keadaan di titik

lokasi tersebut. Penempatan kamera ini biasanya pada tempat strategis seperti lorong dan pintu masuk.

- Kamera CCTV bergerak disebut dengan PTZ (*pan*, *tilt*, *zoom*) memungkinkan operator sistem menggerakkan kamera dengan *remote control* untuk mengikuti pergerakan sebuah objek yang menarik seperti orang atau mobil. Dalam *remote kontrol* tersebut terdapat *joystick* yang bisa diarahkan dari sisi satu ke sisi lain (*pan*), naik turun (*tilt*) dan melakukan *zoom*. Kamera CCTV ini biasanya memiliki kemampuan untuk beradaptasi dengan pencahayaan secara otomatis dan autofokus untuk menjaga agar citra yang terekam terlihat akurat dan jelas.

Saat ini, kamera CCTV telah banyak diintegrasikan dan dimodifikasi dengan teknologi citra lainnya seperti dilengkapi dengan pencahayaan malam agar bisa tetap beroperasi maksimal pada malam hari dan mampu berinteraksi dengan sistem alarm elektronik. Salah satu contoh sistem CCTV yang canggih di bidang keamanan yaitu sistem yang telah diintegrasikan dengan sistem pendeteksi gerak. Dengan sistem yang telah dipadukan ini, apabila terdeteksi gerakan pada video yang tertangkap kamera CCTV, sistem akan memberikan peringatan sehingga staf keamanan bisa mengantisipasi dan mengambil tindakan tepat setelah melakukan pemantauan jarak jauh terhadap apa yang terjadi di lokasi [IPC-10].

2.4.1 Efektifitas CCTV

Efektifitas sistem CCTV telah terbukti menjadi alat yang penting di hampir semua sistem pengawasan dan keamanan yang ditempatkan di bank, ritel, rumah sakit, manajemen lalu lintas, dan pusat-pusat kota. Sebagian besar dalam sistem keamanan, sistem CCTV digunakan sebagai alat visual staf keamanan untuk melakukan pemantauan jarak jauh terhadap lokasi-lokasi yang perlu diamankan. Keberadaan sistem ini tentu sangat optimal dibandingkan dengan staf keamanan yang harus rutin berkeliling ke lokasi-lokasi berbeda untuk memastikan lokasi tersebut aman. Selain itu, video rekaman hasil tangkapan kamera CCTV juga berguna sebagai bahan investigasi dan juga bukti dalam kasus pidana dan perdata.

CCTV adalah alat untuk mencegah kriminalitas [AGP-11]. Berikut adalah alasan yang mendukung statement tersebut :

- CCTV menyediakan fungsi pengawasan yang dapat mencegah orang melakukan kejahatan di sekitar lokasi tempat kamera CCTV dioperasikan.
- menandakan kepada publik bahwa daerah sekitar CCTV adalah tempat yang aman dan kejahatan pada tempat tersebut akan terjadi dengan peluang yang sangat kecil karena di tempat tersebut terdapat saksi yang potensial (kamera CCTV)
- kehadiran CCTV dapat bertindak sebagai pengingat untuk mengingatkan orang untuk mengambil langkah-langkah keamanan lainnya seperti mengunci mobil [AGP-11].

Berdasarkan uraian yang telah dijelaskan, sistem CCTV telah memberikan kontribusi yang besar terhadap pencegahan tindakan yang melanggar hukum dan memberikan rasa keamanan yang kuat bagi pelanggan yang menggunakannya.

2.4.2 Resolusi Sistem CCTV

Resolusi sistem CCTV diukur dengan garis TV setempat. Garis vertikal TV memiliki garis 350 TV maksimum pada 525 - line sistem NTSC. Tapi garis horisontal TV, yang digunakan sebagai parameter kualitas gambar, bervariasi tergantung pada kualitas kamera, lensa, transmisi dan pantauan [IPC-10].

- Resolusi Kamera

  Industri sensor kamera video menggunakan piksel (elemen gambar) sebagai parameter kualitasnya . Resolusi menengah dari B /W kamera dalam sistem AMDAL adalah 510 piksel horizontal dengan 492 piksel vertikal dan setara dengan 380 garis TV. Resolusi tinggi adalah 768 (H) x 492 (V) piksel dan setara dengan 570 garis TV. Resolusi menengah kamera warna berarti garis 330TV dan resolusi tinggi membutuhkan lebih dari 460 garis TV.

- Resolusi Monitor

Monitor dalam sistem NTSC memiliki 525 garis scanning vertikal terlepas dari ukurannya. Garis horizontal 700 TV B / W monitor mewakili tingkat menengah dan lebih dari 900 garis TV berarti resolusi tinggi dalam sistem AMDAL . Resolusi horizontal warna monitor dari 300 garis TV berarti kualitas menengah dan lebih dari 450 garis TV berarti resolusi tinggi .

Untuk memaksimalkan resolusi sistem , dianjurkan untuk memilih monitor yang memiliki resolusi yang lebih baik dibandingkan dengan kamera.

2.4.3 Kamera IP CCTV

Kamera yang langsung bisa terhubung ke jaringan tanpa melalui alat rekam atau DVR (*Digital Video Recording*) karena sudah terdapat *port* LAN pada kamera tersebut dan adanya *software* untuk keperluan merekam data. Kamera ini membutuhkan sebuah komputer untuk perekaman data. Salah satu contoh ip kamera yang banyak digunakan saat ini adalah ip kamera merk TP-LINK seri TL-SC3000 3GPP yang bentuknya bisa dilihat pada Gambar 2.3.

Gambar 2.3 IP kamera TP-LINK TL-SC3000 3GPP

Sumber : [AGP-11]

Kamera TP-LINK seri TL-SC3000 3GPP merupakan kamera dengan menggunakan sensor Panasonic MOS yang berbasis jaringan TCP/IP network jarak jauh melalui komputer yang

terkoneksi ke jaringan. Fitur pendukungnya antara lain 3 GPP, *dual codec* dan *hybrid analog output* yang memungkinkan kamera ini memiliki fitur lengkap serta compatible dengan berbagai jenis media. Spesifikasi lengkap mengenai ip kamera ini dapat dilihat pada Tabel 2.1.

Tabel 2.1 Spesifikasi IP Kamera TP-LINK TP-SC3000

3GPP	
Color video	Available
Video streaming format	Dual Codec (MPEG 4/ MJPEG)
Quality Video Resolution	VGA, 640 x 480, 320 x 240 (default)
Frame Rate	NTSC (30 fps), PAL (25 fps)
Authentication	Level akses dengan multiple user menggunakan password
Supported Network Protocols	DDNS, PPPoE, DHCP, NTP, SNTP, TCP/IP, ICMP, SMTP, FTP, HTTP, RTP, RTSP
Notification Management	Alarm, time or motion detection-Image Buffer Triggers
Image Sensor	Yes, with 1/3.6'' Panasonic MOS 738(H) x 480(V) Pixels
Viewing Angle	Horizontal : 90°, Vertical : 55.6°
Lens Aperture	F2.0
Lens Focus Point	f3.6mm
Required Light Intensity	1 Lux / F2.0
Network Connection	Ethernet (10Base-T/100Base-TX)
Audio Codec	Not Available

Sumber : [AGP-11]

2.5 Object Tracking

Object tracking adalah sebuah teknik atau metode yang digunakan untuk melakukan pelacakan terhadap sebuah objek bergerak, sehingga pergerakan objek tersebut dapat dideteksi dengan tetap memperhatikan perubahan–perubahan yang terjadi di sekitar objek tersebut [SHA-12]. *Object tracking* mengarah ke permasalahan bagaimana menentukan lokasi, jalan dan karakteristik dari objek yang ditangkap dengan menggunakan sensor [CHA-11]. Sensor dapat berasal dari

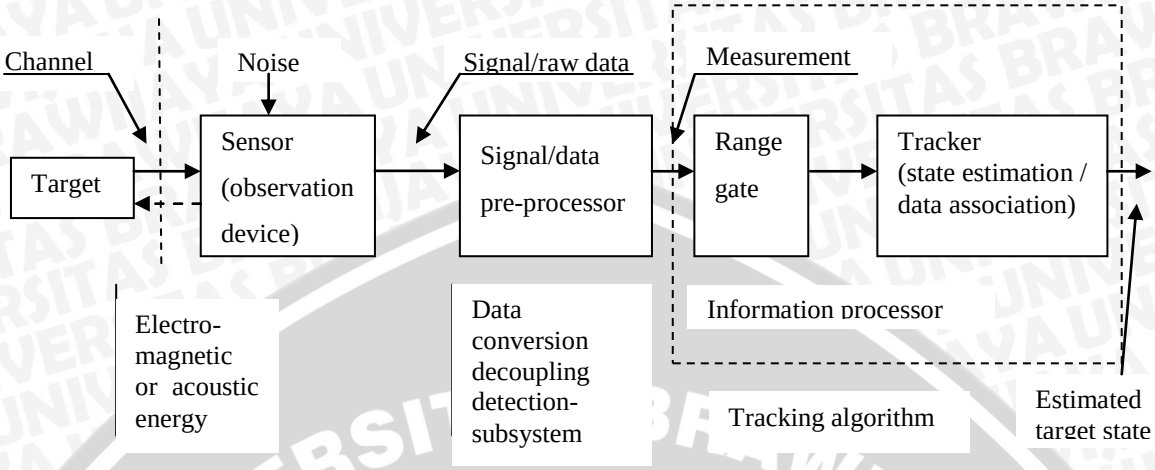
perangkat radar, sonar, LADAR, kamera, sensor infra merah, mikrofon, USG atau sensor lain yang bisa mengumpulkan informasi tentang objek di lingkungan.

Tujuan dari *object tracking* adalah menentukan berapa banyak objek yang berhasil terdeteksi, identitas serta posisinya dan dalam beberapa kasus juga terkait dengan fiturnya. Salah satu contoh penerapan *object tracking* adalah dalam pelacakan radar pesawat. Dalam konteks ini, *object tracking* digunakan untuk menentukan jumlah pesawat yang mengudara di suatu daerah pengawasan. Selain jumlah, jenis pesawat (militer atau komersial), identitas pesawat serta kecepatan dan posisi pesawat juga akan dilacak sesuai dengan informasi yang ditangkap oleh sensor radar [CHA-11].

Dalam proses *object tracking*, tentunya terdapat beberapa permasalahan yang sering mengganggu kinerja metode *object tracking*. Masalah tersebut seperti gerak objek yang tidak stabil dan sering mengalami gangguan, objek yang tidak tertangkap oleh sensor, dan jumlah objek di sudut pandang sensor yang dapat berubah secara acak karena perubahan jarak objek yang satu dengan objek yang lain kdang sangat berdekatan. Oleh karena itu, proses pembuatan sistem *object tracking* tentunya bukan merupakan hal yang mudah [GOS-11]. Sistem harus handal terhadap permasalahan-permasalahan yang dijelaskan di atas.

Berbagai metode telah dikembangkan dalam masalah *object tracking* contohnya SIFT, Mean Shift, CAMSHIFT, SURF (*Speeded-Up Robust Features*) dan GLOH. Metode-metode tersebut masing-masing memiliki kelebihan dan kekurangan sehingga penggunaan metode untuk *object tracking* disesuaikan dengan tujuan dari pembuatan sistem *object tracking* itu sendiri.

Sistem pelacakan terdiri dari beberapa bagian yaitu objek yang akan dilacak, sensor yang menangkap sinyal lingkungan, sinyal prosessor dan sebuah prosessor informasi. Gambaran sistem pelacakan secara umum dapat diilustrasikan pada Gambar 2.4 .



Gambar 2.4 Sistem Pelacakan

Sumber : [CHA-11]

Saat ini, teknologi object tracking telah banyak diimplementasikan baik dalam bidang militer, komersil dan kedokteran. Berikut adalah contoh aplikasi dari *object tracking* :

2.5.1 *Air space monitoring*

Pelacakan lalu lintas udara ini merupakan sistem bidang militer yang cukup canggih. Sistem ini menggunakan radar sebagai sensor. Dimana tujuan dari pelacakan sistem ini melakukan identifikasi pesawat, jenis, identitas, kecepatan, lokasi dan juga mendeteksi apakah pesawat yang mengudara tersebut adalah sebuah serangan atau pesawat komersil biasa.

Radar menggunakan gelombang radio yang terpantulkan untuk mengukur arah, jarak dan kecepatan radial dari objek yang terdeteksi. Selain itu radar juga mentransmisikan gelombang elektromagnetik yang dipantulkan oleh objek dan terdeteksi oleh receiver. Gambar 2.5 mengilustrasikan salah satu contoh bentuk sensor radar dalam *air space monitoring*.

Gambar 2.5 Sensor Radar

Sumber : [CHA-11]

2.5.2 Video Pemantau

Salah satu projek mengenai video pemantauan dengan pelacakan objek yakni projek yang disebut dengan Human Identification at a Distance (HID) pada tahun 2000. HID bertujuan untuk mengembangkan berbagai teknologi pengawasan untuk mendeteksi dan mengidentifikasi manusia dari kejauhan. Sistem yang sama namun telah dilengkapi dengan sistem cerdas yang melakukan pemantauan secara *realtime* juga telah diimplementasikan di stasiun kereta api London dan Genova. Gambar 2.6 adalah ilustrasi bagaimana *object tracking* mendeteksi objek.

Gambar 2.6 a) Latar Gambar b) Latar Gambar dan Objek c) Hasil Background substraction d) Objek yang terdeteksi.

Sumber : [CHA-11]

2.6 Multiple Object Tracking

Sejak pertama kali kemunculannya sampai saat ini, *object tracking* telah mengalami banyak perkembangan. Di antaranya terdapat teknologi untuk mendeteksi satu objek atau yang dikenal dengan istilah *single object tracking* dan juga teknologi untuk mendeteksi lebih dari satu atau banyak objek sekaligus yang dikenal dengan istilah *multiple object tracking*.

Multiple object tracking berbeda dengan *single object tracking*. *Single object tracking* melakukan filter terhadap setiap pelacakan terbaru secara terpisah dan mengabaikan kemungkinan tampilan objek diikuti oleh pelacakan yang lain. *Multiple object tracking* melakukan filter dengan melihat secara global ke objek yang akan dilacak. Sehingga pada *single object tracking* terdapat dua kemungkinan yang akan dihasilkan yaitu tidak ada objek yang terdeteksi dan satu objek terdeteksi. Sedangkan pada *multiple object tracking* kemungkinan yang terjadi yaitu tidak terdapat objek yang terdeteksi dan terdapat jumlah tertentu objek terdeteksi yang tentunya bisa lebih dari satu objek tiap *frame*-nya. .

Multiple object tracking dilakukan dengan memperpanjang proses yang dilakukan *single object tracking* dimana setiap objek akan dilacak sebagai entitas yang terisolasi. Dalam *multiple object tracking*, permasalahannya yang muncul adalah saat target objek terlalu dekat satu sama lain sehingga kadang beberapa objek yang saling berdekatan hanya mampu terdeteksi sebagai satu objek [CHA-11].

2.7 Metode SIFT

Metode SIFT *(Scale Invariant Feature Transform*) adalah sebuah metode yang dikenalkan oleh Lowe pada tahun 2004 untuk memecahkan permasalahan rotasi citra, penskalaan, perubahan sudut pandang, *noise*, *affine deformation* serta perubahaan pencahayaan yang memiliki ketahanan yang kuat. Kelebihan metode SIFT adalah kemampuannya dalam mendeteksi dan mendeskripsikan fitur-fitur lokal dari citra dalam data video. Hasil dari deteksi dan pendeskripsian fitur–fitur lokal itu dapat dipergunakan dalam pelacakan objek bergerak. SIFT memiliki ketahanan yang kuat terhadap penskalaan, rotasi dan perubahan sudut pandang citra. Secara efektifitas, metode SIFT memiliki hasil analisis dengan tingkat keakuratan yang tinggi namun secara efisiensi, metode ini masih sedikit lebih lambat dibandingkan metode lain, seperti SURF misalnya dengan selisih waktu eksekusi kurang lebih satu detik [PAN-13]. Namun dengan efisiensi tersebut, metode ini masih cukup layak untuk digunakan dalam aplikasi dengan menggunakan kamera secara realtime.

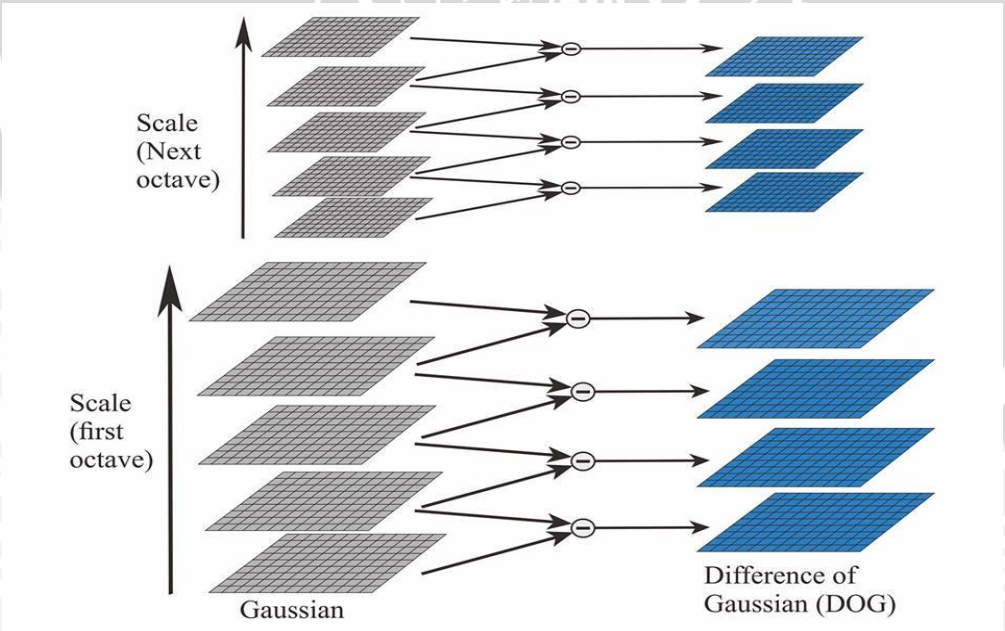
Aspek penting pada metode ini yaitu bahwa metode ini bisa menghasilkan fitur dengan jumlah banyak pada rentang keseluruhan skala dan rotasinya. Sebuah gambar yang memiliki ukuran 500x500 dapat menghasilkan sekitar 2000 fitur yang stabil (meskipun jumlah ini tergantung pada kedua konten citra dan pilihan pada berbagai jenis parameter). Jumlah fitur yang berhasil terdeteksi ini sangat penting pada pengenalan objek, dimana kemampuan untuk mendeteksi sebuah citra kecil di *background* yang berantakan memerlukan sedikitnya tiga fitur yang cocok antara setiap objek untuk mendapatkan identifikasi yang dapat diandalkan [DAV-04].

Secara garis besar, metode SIFT terbagi menjadi beberapa langkah seperti dijelaskan berikut ini :

- *Scale-Space Extrema*

  Merupakan tahap pertama dalam komputasi pencarian yang dilakukan pada semua skala dan lokasi pada gambar. Hal ini berfungsi untuk mengidentifikasi titik-titik potensial yang invarian pada skala dan orientasi serta dapat ditetapkan secara terus-menerus pada sudut pandang yang berbeda terhadap satu objek yang sama. Berdasarkan paradigma klasik, bahwa titik stabil citra terletak pada ekstrema dari Laplacian citra di *scale-space* citra tersebut. *Scale-space* ini merupakan fungsi kontinyu skala yang merepresentasikan sebuah parameter *smoothing* σ, skala, dan konvolusi citra dengan fungsi *Gaussian* dari peningkatan level *blur* σ.

  Menurut Lowe, untuk bisa menentukan scale space SIFT yang stabil diperlukan empat tingkat oktaf dan lima tingkat *scale blur* pada tiap tingkat oktafnya. Ilustrasi mengenai *scale space* dapat dilihat pada Gambar 2.7.



Gambar 2.7 Ilustrasi *Gaussian Scale Space* dan *Difference Of Gaussian* (DOG)

Sumber : [DAV-04]

Scale space citra didefinisikan sebagai fungsi $L_\sigma = L_{(x, y, \sigma)}$, yang dihasilkan dari konvolusi dari variabel *Gaussian* $G_\sigma = G_{(x, y, \sigma)}$ dengan citra masukan $I_{(x, y)}$. Citra digital akan dihaluskan untuk mendapatkan *Gaussian Scale Space* dengan persamaan (2-1).

$$L_\sigma = G_\sigma * I \tag{2-1}$$

dimana

$$G_\sigma = G_{(x, y, \sigma)} = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \tag{2-2}$$

adalah fungsi *Gaussian* dengan integral I dan standar deviasi σ. Notasi * merupakan simbol operasi konvolusi. Berdasarkan pendekatan klasik, *Laplacian* dari *Gaussian* bisa digantikan oleh *Difference of Gaussian* (DOG), $D_\sigma = D_{(x, y, \sigma)}$ pada skala yang berbeda yang dipisahkan oleh faktor multiplikatif konstant *k*, sehingga pencarian DOG akan menggunakan persamaan (2-3).

$$D_\sigma = L_{k\sigma} - L_\sigma \tag{2-3}$$

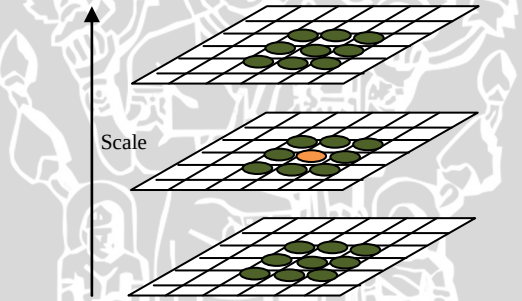
k adalah faktor multiplikatif konstant yang bernilai k= $\sqrt{2}$. Berdasarkan persamaan-persamaan di atas persamaan *difference* bisa dispesifikasikan seperti persamaan (2-4).

$$D_{(x,y,\sigma)} = \left(G_{(x,y,k\sigma)} - G_{(x,y,\sigma)}\right) * I_{(x,y)} = L_{(x,y,k\sigma)} - L_{(x,y,\sigma)} \tag{2-4}$$

Banyak alasan mengapa lebih disarankan untuk memilih menggunakan persamaan (2-1) di atas. Pertama karena fungsi tersebut sangat efisien dalam perhitungan, sama seperti citra yang dihaluskan (*smoothed*), L, dibutuhkan untuk dikalkulasi dalam setiap kasus untuk deskripsi fitur *scale space* dan *difference*, D dapat dihitung dengan menggunakan teknik *image substraction* yang sederhana. Selain itu, fungsi *difference-of-Gaussian* menyediakan sebuah pendekatan yang hampir mendekati skala-normalisasi *Laplacian*, $\sigma^2 \nabla^2 G_\sigma$, pada tahun 2002 Mikolajczyk menemukan bahwa maksima dan minima dari $\sigma^2 \nabla^2 G_\sigma$ menghasilkan fitur gambar yang paling stabil dibandingkan dengan fungsi gambar yang lain seperti *gradient*, *Hessian* atau fungsi sudut *Harris*.

- Mendeteksi *Accurate Key Point* (Ekstremum)/*Keypoint Localization*

  Pada setiap kandidat lokasi yang didapatkan, diperlukan sebuah model yang detail yang pas untuk menunjuk lokasi dan skala. Oleh karena itu, keypoint disini dipilih berdasarkan ukuran kestabilan mereka. Deteksi ekstremum (nilai maksimum dan minimum) dilakukan dengan cara membandingkan nilai setiap piksel pada *DoG scale space* dengan delapan piksel yang berada di sekelilingnya dan 9 piksel yang bersesuaian pada citra DoG sebelum dan setelahnya sehingga akan didapatkan total 26 piksel tetangga yang menjadi pembandingnya. Jika nilai piksel yang dimaksud lebih besar atau lebih kecil daripada nilai-nilai piksel pembandingnya maka koordinat piksel tersebut ditandai sebagai ekstremum. Ilustrasinya dapat dilihat pada Gambar 2.8.



Gambar 2.8 Ilustrasi deteksi ekstrema

Sumber : [DAV-04]

Setelah mendapatkan titik-titik ekstremum maka perlu ditingkatkan lokalisasinya dengan akurasi subpiksel dengan menggunakan ekspansi *Taylor* orde kedua dari fungsi ruang skala *difference of gaussian*, $D_{(x,y,\sigma)}$, sehingga posisi ekstremum sebenarnya, z, didapatkan dengan menggunakan persamaan (2-5).

$$z = -\left(\frac{\partial^2 D}{\partial \mathrm{X}^2}\right)^{-1} \frac{\partial D}{\partial \mathrm{X}} \qquad (2\text{-}5)$$

Dimana nilai orde pertama *difference of gaussian*, $\frac{\partial D}{\partial X}$, dapat diperoleh dari persamaan (2-6).

$$\frac{\partial D}{\partial X} = \begin{bmatrix} \frac{\partial D}{\partial x} \\ \frac{\partial D}{\partial y} \\ \frac{\partial D}{\partial \sigma} \end{bmatrix} = \begin{bmatrix} \frac{D(x+1,y,\sigma) - D(x-1,y,\sigma)}{2} \\ \frac{D(x,y+1,\sigma) - D(x,y-1,\sigma)}{2} \\ \frac{D(x,y,\sigma+1) - D(x,y,\sigma-1)}{2} \end{bmatrix} \quad (2\text{-}6)$$

Sedangkan nilai orde kedua *difference of gaussian*, $\frac{\partial^2 D}{\partial X^2}$, diperoleh dari matriks *Hessian* pada persamaan (2-7).

$$\left(\frac{\partial^2 D}{\partial X^2}\right) = H(X) = \begin{bmatrix} D_{xx} & D_{xy} & D_{x\sigma} \\ D_{yx} & D_{yy} & D_{y\sigma} \\ D_{\sigma x} & D_{\sigma y} & D_{\sigma\sigma} \end{bmatrix} \quad (2\text{-}7)$$

Dimana nilai dari tiap elemen matriks di atas masing-masing dicari menggunakan persamaan (2-8).

$$\begin{aligned}
D_{xx} &= \frac{D(x+1,y,\sigma) - 2D(x,y,\sigma) + D(x-1,y,\sigma)}{1} \\
D_{xy} &= \frac{D(x+1,y+1,\sigma) - D(x+1,y-1,\sigma) - D(x-1,y+1,\sigma) + D(x-1,y-1,\sigma)}{4} \\
D_{x\sigma} &= \frac{D(x+1,y,\sigma+1) - D(x+1,y,\sigma-1) - D(x-1,y,\sigma+1) + D(x-1,y,\sigma-1)}{4} \\
D_{yx} &= \frac{D(x+1,y+1,\sigma) - D(x-1,y+1,\sigma) - D(x+1,y-1,\sigma) + D(x-1,y-1,\sigma)}{4} \\
D_{yy} &= \frac{D(x,y+1,\sigma) - 2D(x,y,\sigma) + D(x,y-1,\sigma)}{1} \\
D_{y\sigma} &= \frac{D(x,y+1,\sigma+1) - D(x,y+1,\sigma-1) - D(x,y-1,\sigma+1) + D(x,y-1,\sigma-1)}{4} \\
D_{\sigma x} &= \frac{D(x+1,y,\sigma+1) - D(x-1,y,\sigma+1) - D(x+1,y,\sigma-1) + D(x-1,y,\sigma-1)}{4} \\
D_{\sigma y} &= \frac{D(x,y+1,\sigma+1) - D(x,y-1,\sigma+1) - D(x,y+1,\sigma-1) + D(x,y-1,\sigma-1)}{4}
\end{aligned} \quad (2\text{-}8)$$

Jika nilai elemen z lebih besar dari 0,5 pada setiap dimensi, titik tersebut akan ditolak dan pencarian nilai z akan dilanjutkan lagi pada titik selanjutnya. Untuk menghitung nilai *keypoint* pada ekstremum digunakan persamaan (2-9).

$$D(z) = D + \frac{1}{2}\frac{\partial D^T}{\partial \mathrm{X}}z \quad (2\text{-}9)$$

Jika nilai D(z) tidak melebihi suatu nilai *threshold* (*threshold*=0,03) maka *keypoint* tersebut dihilangkan dan tidak dipakai lagi. Di samping penghapusan *keypoint* yang tidak memenuhi syarat *threshold* dilakukan juga penghapusan *keypoint* tak stabil yang berada pada daerah *edge*.

Untuk stabilitas, tidak cukup untuk menghilangkan *keypoints* dengan kontras rendah. Fungsi DoG akan memiliki respon yang kuat di sepanjang tepi, jika lokasi titik tepi ditunjukkan secara buruk, oleh karena itu titik ini tidak stabil saat terdapat *noise* meskipun dalam skala sekecil apapun. Titik ekstremum buruk yang ditemukan dalam DoG fungsi akan memiliki kelengkungan utama yang besar di tepi tapi kecil dalam arah tegak lurus.

Principal Kurvatur dapat dihitung dari matriks *Hessian* 2x2, H, dihitung pada lokasi dan skala *keypoint* seperti pada persamaan (2-10).

$$H = \begin{bmatrix} Dxx & Dxy \\ Dxy & Dyy \end{bmatrix} \quad (2\text{-}10)$$

Derivatif diperkirakan dengan mengambil perbedaan titik sampel tetangga. Nilai *eigen* dari H sebanding dengan lekukan utama D. Pendekatan SIFT hanya berfokus pada nilai *ratio*, sehingga pada pendekatan ini proses perhitungan nilai eigen akan diabaikan. Dimisalkan α merupakan nilai *eigen* dengan nilai *magnitude* terbesar dan β adalah *eigen* dengan nilai *magnitude* terkecil. Kemudian, kita dapat menghitung jumlah dari nilai *eigen* berdasarkan jejak H, $Tr(\boldsymbol{H})$, dan hasil mereka dari determinan, $Det(\boldsymbol{H})$ dengan menggunakan persamaan (2-11) dan (2-12).

$$Tr(\boldsymbol{H}) = Dxx + Dyy = \alpha + \beta \quad (2\text{-}11)$$

$$Det(\boldsymbol{H}) = DxxDyy - (Dxy)^2 = \alpha\beta. \quad (2\text{-}12)$$

Dalam hal tidak mungkin bahwa determinan negatif , *curvatures* memiliki tanda yang berbeda sehingga titik tersebut akan dibuang sehingga bukan titik ekstremum. Biarkan *r* menjadi rasio antara nilai eigen terbesar dan terkecil ,

sehingga $\alpha = r\beta$. Lalu dilakukan perhitungan dengan menggunakan persamaan (2-13).

$$\frac{Tr(\boldsymbol{H})^2}{Det(\boldsymbol{H})} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r} \quad (2\text{-}13)$$

Oleh karena itu, untuk memeriksa bahwa rasio dari *principal curvatur* berada di bawah ambang batas tertentu , r , kita hanya perlu memeriksa nilai $\frac{Tr(\boldsymbol{H})^2}{Det(\boldsymbol{H})}$ dengan persamaan (2-14).

$$\frac{Tr(\boldsymbol{H})^2}{Det(\boldsymbol{H})} < \frac{(r+1)^2}{r} \quad (2\text{-}14)$$

- *Orientation Assignment*

  Pada penetapan orientasi ini digunakan citra *Gaussian smooth* L yang memiliki skala paling dekat dengan skala *keypoint*. Untuk setiap citra sampel L(x,y) *magnitude* m(x,y) dan orientasi θ(x,y) dihitung dengan menggunakan persamaan (2-15) dan (2-16).

$$m(x,y) = \sqrt{\big(L(x+1,y) - L(x-1,y)\big)^2 + (L(x,y+1) - L(x,y-1))^2} \quad (2\text{-}15)$$

$$\theta(x,y) = \arctan\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right) \quad (2\text{-}16)$$

Orientasi histogram terbentuk dari orientasi gradien titik sampel dalam suatu wilayah sekitar *keypoint* tersebut . Orientasi histogram memiliki 36 bin yang mencakup rentang orientasi 360 derajat. Setiap sampel ditambahkan ke histogram yang dibobot dengan besarnya *gradien* dan dengan *Gaussian-weighted circular window* dengan *scale* 1,5 kali dari *scale keypoint* tersebut . Puncak pada orientasi histogram sesuai dengan arah dominan gradien lokal . Puncak tertinggi di histogram akan terdeteksi, dan kemudian setiap puncak lokal lainnya yang berada dalam 80% dari puncak tertinggi digunakan untuk membuat *keypoint* dengan orientasi itu. Oleh karena itu, untuk lokasi dengan beberapa puncak besarnya sama, akan ada beberapa *keypoints* dibuat di lokasi yang sama dan skala tetapi orientasi yang berbeda.

- *Local Image Descriptor*

Proses terakhir adalah menghitung vektor deskriptor. Deskriptor dihitung untuk masing-masing *keypoint*, langkah ini dilakukan pada gambar yang paling dekat dengan skala untuk skala *keypoint*. Pertama membuat orientasi dengan 4x4 piksel dengan 8 bin untuk tiap *keypoint*. Histogram yang didapat pada langkah penetapan orientasi ini dihitung *magnitude* dan nilai orientasi untuk sampel dalam wilayah 16x16 disekitar *keypoint*. *Magnitude* dihitung dengan fungsi *Gaussian* dengan σ sama dengan satu setengah lebar deskriptor. Kemudian deskriptor menjadi vektor dari semua nilai histogram ini. Karena 4x4=16 histogram dengan masing-masing memiliki 8 bin, maka vektor memiliki 128 elemen.

Meskipun metode SIFT memiliki kehandalan yang tinggi dalam mendeteksi objek bergerak, saat ini beberapa penelitian dilakukan dengan memadukan metode SIFT dengan metode *object tracking* lainnya untuk menyelesaikan masalah pelacakan video agar mendapatkan hasil yang lebih maksimal.

- *Keypoint Matching*

Proses *keypoint matching* adalah sebuah proses pencocokan antara dua citra objek dengan membandingkan semua *keypoint* dan deskriptor yang dimiliki keduanya. Proses perhitungan pencocokan ini dihitung berdasarkan jarak dengan menggunakan *Euclidien Distance*, dengan persamaan (2-17).

$$d(p,q) = \sqrt{(q_1 + p_1)^2 + \cdots + (q_n + p_n)^2} \quad (2\text{-}17)$$

Dengan :

$d(p,q)$: jarak antara deskriptor citra 1, p, dan citra 2, q

$q_1 ... q_n$: deskriptor citra 2, q

$p_1 ..... p_n$: deskriptor citra 1, p

Nilai jarak yang didapatkan selanjutnya akan dibandingkan dengan nilai *threshold matching* yang digunakan. Dalam hal ini, *treshold matching* yang digunakan sebesar 0,3. Apabila jarak yang didapatkan lebih kecil dari nilai *threshold* maka dikatakan keypoint tersebut *match*. Proses pembandingan ini dilakukan pada semua *keypoint* yang dimiliki dua citra tersebut. Apabila

keypoint yang *match* lebih dari 70% maka dikatakan kedua citra tersebut identik.

2.8 Citra *Grayscale*

Merupakan proses yang dilakukan pertama pada suatu citra sebelum citra tersebut diproses dengan proses inti sesuai dengan tujuan diprosesnya citra tersebut. Tujuan dari proses ini yaitu untuk mempermudah proses pengolahan citra. Pada awalnya citra terdiri atas 3 layer warna yaitu R-layer, G-layer dan *B-layer*, untuk mengubahnya menjadi citra grayscale tentunya tiga layer tersebut akan diubah menjadi satu layer warna. Dalam citra grayscale tidak ada lagi istilah warna, yang ada hanyalah derajat keabuan. Salah satu metode untuk menghasilkan citra grayscale yaitu dengan mengganti nilai RGB dari citra tersebut dengan satu nilai yang sama yaitu dengan menggunakan rata-rata dari nilai RGB piksel yang sama. Nilai mean dihitung dengan menggunkan persamaan (2-18).

$$mean = \frac{pikse\ l_R + Piksel_G + Piksel_B}{3} \quad (2\text{-}18)$$

Dengan *mean* adalah nilai rata-rata, $Piksel_R$ adalah nilai piksel dari *Red*, $Piksel_G$ adalah nilai piksel dari *Green* dan $Piksel_B$ adalah nilai piksel *Blue*.

2.9 *Background Substraction*

Background substraction adalah sebuah proses yang bertujuan untuk melakukan segmentasi terhadap objek dengan cara mengurangi citra yang terdapat objek di dalamnya dengan citra *background* secara absolut. Karena teknik ini akan melihat perbedaan untuk setiap piksel di dalam citra, sehingga kedua citra harus memiliki tipe data dan ukuran yang sama. Contoh dari proses *background substraction* diperlihatkan ilustrasi nilai matriks citra 3x3 pada Gambar 2.9.

$$\begin{bmatrix} 56 & 40 & 32 \\ 80 & 89 & 150 \\ 170 & 120 & 40 \end{bmatrix} - \begin{bmatrix} 56 & 40 & 32 \\ 100 & 50 & 75 \\ 20 & 140 & 40 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 20 & 39 & 75 \\ 150 & 20 & 0 \end{bmatrix}$$

(a) (b) (c)

Gambar 2.9 Ilustrasi Proses *Background Substraction* a) Piksel Citra Yang Terdapat Objek dan Background, b) Piksel Citra Background , c) Hasil Piksel *Bakground Substraction*

2.10 *Median* Dan *Min Filter*

Ada berbagai macam teknik untuk mengurangi (reduksi) *noise*, salah satunya menggunakan *filter median* dan *minimal*. *Noise* adalah sebuah piksel yang memiliki nilai yang berbeda dengan semua tetangganya. Keberadaan *noise* dalam pemrosesan citra tentunya akan mengganggu kalkulasi, sehingga hasil yang didapatkan tentu akan semakin berkurang keakuratannya. Salah satu cara untuk menghilangkan *noise* ini yaitu dilakukan *filter* terhadap citra tersebut. Banyak sekali metode filterasisasi, salah satunya adalah *median* dan *min filter*.

Metode *median filter* merupakan *filter* non-linear yang dikembangkan Tukey, yang berfungsi untuk menghaluskan dan mengurangi *noise* atau gangguan pada citra. Dikatakan nonlinear karena cara kerja penapis ini tidak termasuk kedalam kategori operasi konvolusi. Operasi nonlinear dihitung dengan mengurutkan nilai intensitas sekelompok piksel, kemudian menggantikan nilai piksel yang diproses dengan nilai tertentu.

Pada *median filter* suatu *window* atau penapis yang memuat sejumlah piksel ganjil digeser titik per titik pada seluruh daerah citra. Nilai-nilai yang berada pada window diurutkan secara ascending untuk kemudian dihitung nilai mediannya. Nilai tersebut akan menggantikan nilai yang berada pada pusat bidang *window*. Jika suatu *window* ditempatkan pada suatu bidang citra, maka nilai piksel pada pusat bidang *window* dapat dihitung dengan mencari nilai median dari nilai intensitas sekelompok piksel yang telah diurutkan. Secara matematis dapat dirumuskan seperti pada persamaan (2-19).

$$g(x,y) = median\{f(x-i, y-j), (i,j) \in w\} \quad (2\text{-}19)$$

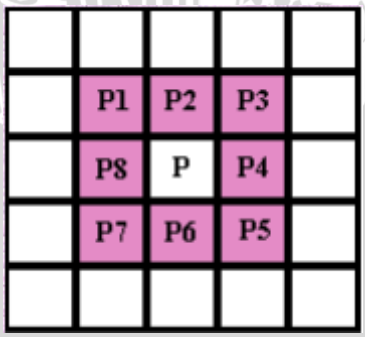
Dimana *g(x,y)* merupakan citra yang dihasilkan dari citra *f(x,y)* dengan *w* sebagai *window* yang ditempatkan pada bidang citra dan *(i,j)* elemen dari *window* tersebut.

Metode *min filter* merupakan sebuah filter yang bekerja dengan menggantikan nilai piksel tengah dari sekelompok piksel tertentu dengan nilai terendah diantara piksel terebut. Min filter juga merupakan filter non-linear yang tidak termasuk ke dalam kategori operasi konvolusi. Secara matematis dapat dirumuskan sepertipada persamaan (2-20).

$$g(x,y) = minimum\{f(x-i, y-j), (i,j) \in w\} \quad (2\text{-}20)$$

2.11 Penentuan batas (*Boundary*)

Penentuan garis batas objek pada pengolahan citra adalah hal yang penting terutama dalam proses segmentasi secara morfologi. Banyak sekali algoritma untuk menentukan batas, salah satunya adalah *Moore's Neighbor Tracing*. Metode ini bekerja dengan menelusuri delapan tetangga dari titik piksel yang sebelumnya telah ditandai sebagai titik batas. Ke delapan tetangga tersebut adalah 8 piksel yang mengitari dari piksel tersebut. Ilustrasi delapan tetangga dapat dilihat pada Gambar 2.10. Dengan P adalah piksel yang telah ditandai sebagai titik batas, dan P1-P8 adalah kedelapan tetangga dari piksel P.



Gambar 2.10 Ilustrasi Delapan Tetangga

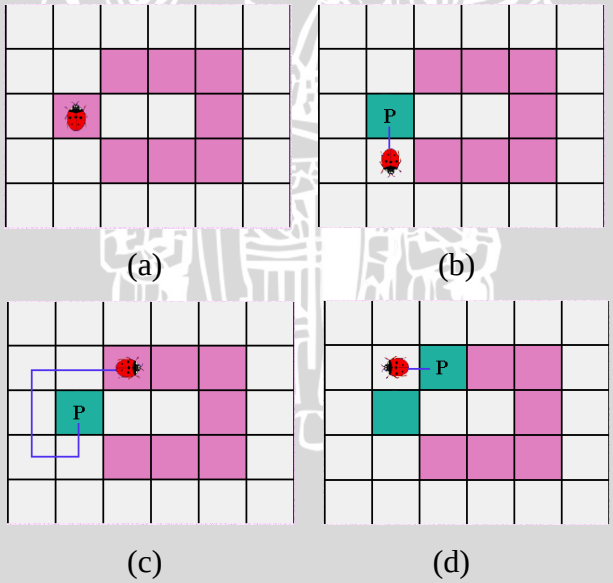
Sumber : [ABE-20]

Diberikan sebuah pola digital yaitu sekelompok piksel hitam , dengan latar belakang piksel putih. Mencari pixel hitam pertama dan dinyatakan sebagai *start*

piksel. (Menemukan sebuah *start* piksel dapat dilakukan dengan beberapa cara, memulai dari pojok kiri bawah *grid* , memindai setiap kolom piksel dari bawah ke atas atau mulai dari kolom paling kiri dan melanjutkan ke kanan sampai ditemukan piksel hitam yang akan dideklarasikan sebagai *start* piksel) .

Bug (kumbang) berdiri pada awal piksel seperti pada Gambar 2.11 a di bawah ini . Kumbang tersebut akan mengekstrak kontur dengan berkeliling pola searah jarum jam . (Tidak peduli arah mana yang dipilih selama tetap dengan pilihan sepanjang algoritma) .

Ide umum adalah : setiap kali ditemukan piksel hitam , P , menyusuri *backtrack* yaitu kembali ke piksel putih sebelum memasuki piksel hitam tersebut, kemudian , pergi sekitar pixel P searah jarum jam , mengunjungi setiap pixel di lingkungan *Moore* , sampai menemukan piksel hitam . Algoritma berakhir ketika *start* piksel dikunjungi untuk kedua kalinya. Ilustrasi kontur citra dapat dilihat pada Gambar 2.11 .



Gambar 2.11 a) Ilustrasi kontur citra dan kumbang b) Kumbang mulai menelusuri piksel tetangganya dari Backtrack c) Kumbang terus menelusuri sampai ditemukan piksel gelap d) Piksel gelap ditandai dengan warna biru dan dijadikan sebagai P selanjutnya.

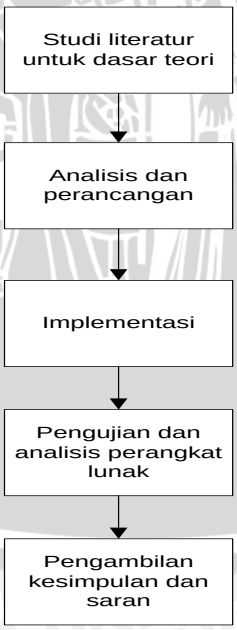
Sumber : [ABE-20]

BAB III
METODE PENELITIAN DAN PERANCANGAN

Pada BAB III ini akan dijelaskan mengenai metode penelitian dan perancangan yang akan digunakan dalam penelitian implementasi metode *Background Substraction* dan *Scale Invariant Feature Transform* pada *Multiple Object Tracking* pada video CCTV. Metode penelitian berisi langkah-langkah yang digunakan dalam penelitian skripsi yang terdiri dari studi literatur untuk dasar teori, analisis kebutuhan dan perancangan, implementasi, pengujian dan analisis perangkat lunak, serta pengambilan kesimpulan dan saran. Sedangkan pada perancangan akan dijelaskan mengenai analisis kebutuhan perangkat lunak dan perancangan perangkat lunak.

3.1 Metode Penelitian

Langkah-langkah yang akan dilakukan dalam penelitian ini dapat diilustrasikan pada Gambar 3.1.



Gambar 3.1 Diagram Alir Metode Penelitian

Sumber : Perancangan

3.1.1 Studi Literatur Untuk Dasar Teori

Studi literatur dilakukan dengan mencari referensi dari buku atau internet sebagai materi dasar teori untuk menunjang penyusunan skripsi. Adapun teori-teori tersebut antara lain :

- *Object tracking*
- *Multi-object tracking*
- CCTV
- *Metode Scale Invariant Feature Transform (*SIFT*)*
- Pengubahan ke warna keabu-abuan (*grayscale*)
- *Background Substraction*
- *Median* dan *Min Filter*
- Metode *Moore's Neighbor Tracing* untuk penentuan batas

3.1.2 Analisis Dan Perancangan

Analisis kebutuhan bertujuan untuk mendapatkan semua kebutuhan yang diperlukan oleh sistem yang akan dibangun. Analisis kebutuhan dilakukan dengan mengidentifikasi kebutuhan sistem dan siapa saja yang terlibat di dalamnya. Berikut analisis kebutuhan dalam aplikasi *multi object tracking* dengan menggunakan metode *background substraction* dan SIFT ini.

3.1.2.1 Kebutuhan Antar Muka

Spesifikasi kebutuhan antar muka untuk perangkat lunak ini antara lain :

- Tampilan antarmuka pengguna (*user interface*) harus mudah digunakan dan dimengerti pengguna saat menggunakannya.
- Perangkat lunak memiliki antarmuka untuk melakukan penyimpanan data *template* latar belakang (*background*) ruangan atau tempat.
- Perangkat lunak memiliki antarmuka untuk menginputkan data uji berupa file video (.wmv) CCTV.
- Perangkat lunak memiliki antarmuka untuk menampilkan video yang tengah dilacak dan menampilkan notifikasi (jika mode notifikasi diaktifkan) serta menandai daerah sekitar objek yang telah terdeteksi dengan balok/kotak merah.

3.1.2.2 Kebutuhan Data

Data yang akan diolah oleh perangkat lunak ini antara lain :

- Data CCTV berupa file video (.wmv) yang digunakan sebagai citra masukan yang akan diproses. Data diambil dari CCTV kamera PTIIK Universitas Brawijaya. Dimana CCTV tersebut merupakan kamera merk TP-LINK TP-SC3000 3GPP, dengan resolusi video 320x240 px dan *frame rate* 30 fps.
- Data berupa citra latar belakang ruangan atau tempat yang digunakan sebagai *background* untuk proses *background substraction* pada proses deteksi objek nantinya.
- Data keluaran perangkat lunak ini akan langsung ditampilkan saat pemutaran video secara *realtime*

3.1.2.3 Kebutuhan Fungsional

Fungsi-fungsi perangkat lunak ini antara lain :

- Perangkat lunak harus mampu melakukan *snapshot* data citra latar belakang ruangan atau *background* dari video dan menyimpannya di dalam direktori komputer.
- Perangkat lunak harus mampu menerima inputan data berupa file video CCTV.
- Aplikasi harus mampu melakukan pendeteksian objek pada tiap *frame* video.
- Perangkat lunak harus mampu melacak objek yang bergerak dengan menandai daerah objek yang terdeteksi menggunakan kotak merah pada video.
- Perangkat lunak harus mampu memberikan notifikasi apabila terdapat objek yang berhasil terdeteksi sesuai dengan waktu pengesetan.

3.1.2.4 Arsitektur Program

Perancangan aplikasi *multi object tracking* pada video CCTV ini dapat dijelaskan sebagai berikut. Pertama, pengguna terlebih dahulu harus menyimpan

citra latar belakang ruang atau *background* dari video. Citra tersebut digunakan sebagai *background* saat dilakukan proses *background substraction* pada proses deteksi objek sebelum proses pelacakan dengan metode SIFT nantinya. Citra *background* didapatkan dari proses *snapshot* salah satu *frame* pada video tanpa objek.

Selanjutnya sistem akan memproses terlebih dahulu citra latar belakang tersebut sebelum disimpan. Proses tersebut dilakukan dengan mengubah citra menjadi abu-abu (*grayscale*). Hal ini bertujuan untuk menyamakan kondisi semua citra, sehingga saat dilakukan proses pencocokan citra ini dapat langsung digunakan tanpa dilakukan proses terhadapnya lagi.

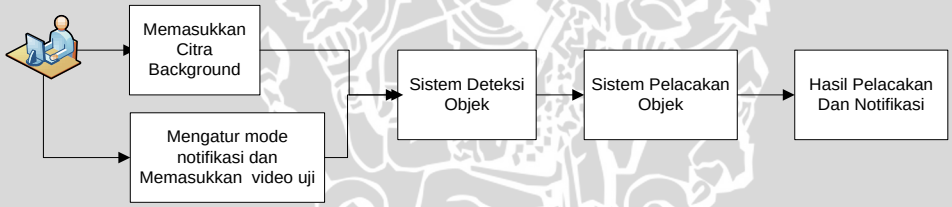
Setelah citra latar belakang telah berhasil disimpan, pengguna mengatur mode notifikasi apakah *on* atau *off*. Selanjutnya memasukkan data uji berupa file video CCTV berekstensi .wmv. Resolusi yang digunakan adalah 320x240 dengan kualitas VGA.

Sistem akan melakukan proses deteksi objek pada citra uji dan mengambil piksel daerah objek untuk selanjutnya akan dicari *feature keypoint* nya dengan metode SIFT. Proses tersebut meliputi pengubahan menjadi citra abu-abu (*grayscale*), *background substraction*, *median filter* serta *min filter* dan penentuan batas objek dengan metode *Moore's Neighbor Tracing*. Hasil dari proses *preprocessing* ini adalah objek yang telah terdeteksi. Objek inilah yang selanjutnya akan menjadi target pelacakan pada *frame-frame* selanjutnya menggunakan metode SIFT. Sebelum melalui proses ini, citra terlebih dahulu akan dipecah-pecah terlebih dahulu menjadi *frame-frame bitmap*. *Frame-frame* inilah yang nantinya akan diolah dengan metode SIFT, dengan mencocokan fitur objek yang berhasil dideteksi dengan *frame* pertama, kedua, ketiga dan seterusnya. Hasil dari metode SIFT ini berupa daerah dari objek yang telah terlacak tiap *frame*.

Apabila pada video terdeteksi adanya objek yang bergerak yakni manusia, maka sistem akan memberikan notifikasi dan menandai daerah sekitar objek bergerak tersebut dengan kotak merah.

3.1.2.5 Diagram Blok Sistem

Secara umum, sistem ini akan dimulai dengan pengguna memasukkan data video berisi *frame* yang sesuai untuk digunakan sebagai *background*. Dari video tersebut akan ditangkap salah satu *frame* yang kemudian disimpan sebagai citra *background* yang akan digunakan untuk deteksi objek nantinya. Setelah citra *background* telah siap, selanjutnya pengguna mengatur mode notifikasi dan memasukkan data yang akan diuji berupa file video CCTV (.wmv). Data uji tersebut akan melalui proses pendeteksian objek sebelum diolah dengan metode SIFT. Metode SIFT akan menghasilkan titik-titik daerah sekitar objek. Hasil dari sistem ini berupa notifikasi dan kotak merah pada video yang menandakan daerah objek yang telah terdeteksi. Gambar 3.2 merupakan ilustrasi dari diagram blok aplikasi *multiple object tracking* dengan metode SIFT ini.



Gambar 3.2 Diagram Blok Sistem

Sumber : Perancangan

Diagram blok sistem ini secara keseluruhan terdiri dari tiga komponen utama yang dijalankan *user* yaitu memasukkan citra *background*, mengatur mode notifikasi dan memasukkan video uji serta hasil pelacakan dan notifikasi. Selain itu juga terdapat dua komponen yang dilakukan sistem yaitu deteksi objek dengan metode *background substraction* dan pelacakan objek dengan metode SIFT.

Penjelasan lebih lanjut mengenai komponen utama dan sistem dari aplikasi ini dijelaskan sebagai berikut :

a) Memasukkan Citra *Background*

Bagi pengguna, proses memasukkan data untuk citra *background* ini bertujuan untuk menyediakan data berupa citra dua dimensi sebagai latar belakang ruang atau tempat dimana video merekam

aktifitas keadaan tempat itu. Citra *background* ini didapat dari proses *snapshot* salah satu *frame* pada video. Citra *background* ini merupakan pencitraan dari ruang/tempat pada keadaan dasar dimana tidak terdapat objek bergerak di dalamnya, dengan kata lain keadaannya statis dan terdiri dari objek-objek yang posisinya tidak akan berubah (*absolute*). Citra *background* ini digunakan pada proses *background substraction* pada tahap *object detection* untuk metode SIFT. Sebelum citra *background* disimpan, citra akan diproses terlebih dahulu agar siap digunakan saat proses *background substraction* nantinya.

b) Mengatur Mode Notifikasi dan Memasukkan Video

Bagi pengguna, proses mengatur mode notifikasi ini merupakan proses untuk mengatur apakah mode notifikasi dikondisikan pada posisi aktif (*on*) atau pasif (*off*). Dengan diaktifkannya mode notifikasi, saat sistem mendeteksi adanya objek bergerak (yakni manusia) selama jangka waktu yang telah ditentukan, maka sistem akan memberikan notifikasi kepada pengguna dan menandai objek yang telah terdeteksi tersebut dengan kotak merah. Sebaliknya jika mode notifikasi dinon-aktifkan maka sistem hanya akan menandai objek yang telah terdeteksi tersebut dengan kotak merah saja. Sedangkan proses memasukkan video, bagi pengguna merupakan proses untuk memasukkan data berupa citra video yang akan diproses untuk melacak objek bergerak yang terekam di dalamnya. Data video yang dimasukkan bisa berupa file video dengan format .wmv.

c) Sistem Deteksi Objek

Bagi pengguna, sistem deteksi objek ini merupakan proses untuk mendeteksi objek-objek yang ada di citra video yang akan dilacak.. Sistem akan melakukan proses ini dengan mengubah citra video menjadi citra *grayscale*, melakukan *background substraction*, *median filter* serta *min filter* dan penentuan batas objek.

d) Sistem Pelacakan SIFT

Bagi pengguna, sistem pelacakan dengan metode SIFT ini merupakan proses untuk melakukan pelacakan terhadap data uji yang telah dimasukkan dengan menggunakan metode SIFT.

e) Hasil Pelacakan dan Notifikasi

Bagi pengguna, keluaran dari sistem ini berupa notifikasi (jika mode notifikasi diaktifkan) dan tanda berupa kotak merah yang menandakan daerah sekitar objek yang telah terdeteksi.

3.1.3 Implementasi

Implementasi aplikasi pelacakan ini dilakukan dengan mengacu pada perancangan sistem. Implementasi perangkat lunak dilakukan menggunakan bahasa pemrograman *framework* C# .NET, *library VideoLab* dan media penyimpanan data berupa file .txt, dengan menggunakan *tools* Microsoft Visual Studio 2010. Implementasi aplikasi ini meliputi :

- Pembuatan antarmuka pengguna (*user interface*) berupa halaman yang menerima masukan pengguna.
- Melakukan proses deteksi objek terhadap data yang dimasukkan oleh pengguna yaitu data video.
- Melakukan proses pelacakan dari *frame-frame bitmap* data uji dengan menggunakan metode SIFT (*Scale Invariant Feature Transform*).
- Memberikan informasi mengenai objek yang terdeteksi dengan menampilkan notifikasi (jika mode notifikasi diaktifkan) dan tanda kotak merah yang merupakan representasi daerah objek.

3.1.4 Pengujian Dan Analisis Perangkat Lunak

Pengujian perangkat lunak pada penelitian ini dilakukan agar dapat menunjukkan bahwa perangkat lunak telah mampu bekerja sesuai dengan spesifikasi dari kebutuhan yang melandasinya. Pengujian yang dilakukan meliputi :

- Data uji yang digunakan berupa file video hasil rekaman kamera CCTV yang ada di PTIIK (Program Teknologi Informasi dan Ilmu

Komputer) Universitas Brawijaya Malang, dengan merk TP-LINK TP-SC3000 3GPP. Data video tersebut berformat .wmv, dengan resolusi sebesar 320x240 piksel dan *frame rate* 30 fps.

- Skenario pengujian pertama yaitu pengujian akurasi parameter. Dimana parameter yang akan diuji antara lain *gaussian scale* (σ), rasio (*r*), dan *threshold matching*. Dari tiga parameter tersebut masing-masing akan diambil tiga nilai untuk diuji. Karena tiga parameter saling berkaitan, ketiga nilai dari masing-masing parameter akan saling dikombinasikan sehingga akan terjadi 3x3x3 = 27 pengujian. Data yang digunakan sebanyak 30 *frame* dari satu video yang dipilih berdasarkan pembaruan objek yang tertangkap. Pengujian ini bertujuan untuk mengetahui nilai kombinasi parameter yang sesuai untuk digunakan oleh sistem yang menghasilkan akurasi pelacakan tertinggi. Pengukuran tingkat akurasi program akan dihitung berdasarkan sesuai atau tidaknya jumlah objek yang terdeteksi oleh program dengan jumlah sebenarnya pada tiap *frame* video yang mampu dilihat mata manusia. Contoh tabel pengujian dapat dilihat pada Tabel 3.1.

Tabel 3.1 Contoh Tabel Pengujian Akurasi Parameter

Pengujian ke-	Scale Gaussian	R	Threshold matching	Akurasi
1	0,5	1,01	0,3	-
2	0,5	1,01	0,5	-
.	.	.	.	-
.	.	.	.	-
27	1,41	3	0,8	-

Sumber : Perancangan

- Skenario pengujian kedua yaitu pengujian nilai parameter terbaik terhadap variasi data. Pengujian ini bertujuan untuk mengetahui apakah parameter yang telah didapatkan pada pengujian

sebelumnya telah mampu bekerja dengan baik apabila diujikan pada data video lain. Pada pengujian ini akan diujikan sebanyak 10 data video dengan rentang waktu sebesar 10 detik atau 20 detik. Pengujian dilakukan dengan mencocokkan hasil deteksi ada atau tidaknya objek antara deteksi manual dan sistem. Jika terjadi kesamaan antara dua macam deteksi tersebut, maka akurasi tiap *frame* akan dinilai sebagai 1, sebaliknya akan dinilai sebagai 0. Contoh tabel pengujiannya dapat dilihat pada Tabel 3.2.

Tabel 3.2 Contoh Tabel Pengujian Parameter Terhadap Variasi Data

Data Video	Frame Sesuai	Frame Tidak Sesuai	Akurasi
Video 1	-	-	--
Video 2	-	-	-
Video 3	-	-	-
Video4	-	-	-
Video 5	-	-	-
Video 6	-	-	-
Video 7	-	-	-
Video 8	-	-	-
Video 9	-	-	-
Video 10	-	-	-

Sumber : Perancangan

- Skenario pengujian ketiga yaitu pengujian *centroid*. Pengujian ini dilakukan dengan tujuan untuk mengetahui tingkat *error rate* dari hasil deteksi *region* objek dengan menggunakan *centroid*. Centroid dihitung berdasarkan titik pojok pada masing-masing objek yang telah terdeteksi. Tiap objek akan memiliki empat titik pojok, yakni L = kiri, R = kanan, T=atas dan B=bawah. Koordinat titik *centroid* (C_x, C_y) akan dihitung menggunakan persamaan (3-1) dan (3-2).

$$C_x = L + (R - L)/2 \quad (3\text{-}1)$$

$$C_y = T + (B - T)/2 \quad (3\text{-}2)$$

Kemudian, *error rate* akan dihitung dengan persamaan (3-3).

$$E = (E_1 + E_2)/2 \quad (3\text{-}3)$$

Dimana nilai $E_1 = ((C_{xp} - C_{xm}) * 100)/C_{xp}$ (3-4)

$E_2 = ((C_{yp} - C_{ym}) * 100)/C_{yp}$ (3-5)

Dengan C_{xp} =C_x hasil dari algoritma yang dirancang, , C_{xm} =C_x hasil dari manualisasi, C_{yp} = C_y hasil dari algoritma yang dirancang dan C_{ym} = C_y hasil dari perhitungan manual. Proses manualisasi disini dilakukan dengan langsung menandai batas-batas objek pada video sesuai dengan persepsi mata manusia. Dari *error rate* tersebut akan dihitung *error rate* rata-rata dengan persamaan (3-6).

$$E_{avg} = \frac{\sum_{i=1}^{n} E}{n} \quad (3\text{-}6)$$

Dengan n adalah banyaknya *frame*, dan E_{avg} adalah *error rate* rata-rata. Contoh tabel pengujiannya dapat dilihat pada tabel 3.3.

Tabel 3.3 Contoh Tabel Pengujian Centroid

Frame no	Objek	Error rate %	Average Error	
			Objek 1	Objek 2
463	1	-	-	-
	2	-		
467	1	-		
	2	-		
470	1	-		
	2	-		
475	1	-		
	2	-		
559	1	-		
	2	-		
565	1	-		
	2	-		

Sumber : Perancangan

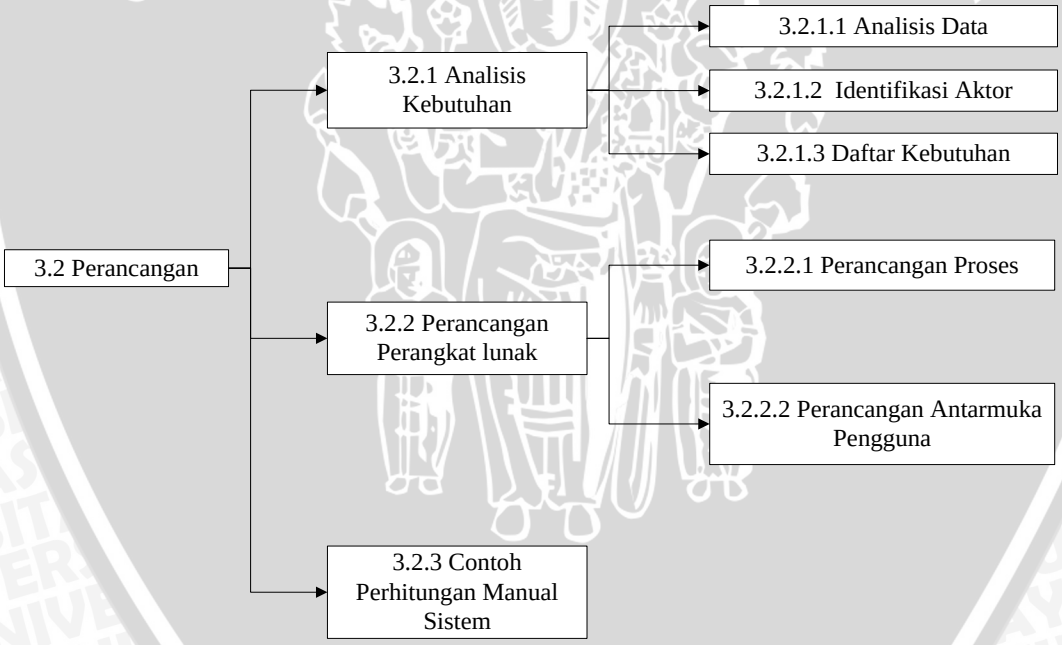
3.1.5 Pengambilan Kesimpulan Dan Saran

Pengambilan kesimpulan dari aplikasi yang telah dibuat dilakukan setelah semua tahapan perancangan dan pengujian sistem aplikasi telah selesai dilakukan. Pengambilan kesimpulan didasarkan pada kesesuaian antara teori dan praktek. Kesimpulan ini merupakan informasi akhir dari perancangan aplikasi yang berisi mengenai berhasil atau tidaknya aplikasi tersebut dijalankan.

Tahap terakhir dari penulisan adalah saran yang dimaksudkan untuk memperbaiki kesalahan-kesalahan yang terjadi serta menyempurnakan penulisan.

3.2 Perancangan

Sub bab ini membahas mengenai perancangan aplikasi *multiple object tracking* dengan metode SIFT. Perancangan yang dilakukan meliputi analisis kebutuhan, perancangan perangkat lunak, dan contoh perhitungan manual sistem yang akan dibangun. Tahap analisis kebutuhan terdiri dari tiga langkah yaitu analisis data yang diperlukan, identifikasi aktor, dan membuat daftar kebutuhan sistem. Proses perancangan perangkat lunak mempunyai tiga tahap yaitu perancangan proses sistem *tracking* dengan metode SIFT, perancangan basis data, dan perancangan antarmuka pengguna. Tahap-tahap perancangan yang akan dilakukan dapat diilustrasikan pada gambar 3.3.



Gambar 3.3 Diagram Blok Perancangan

Sumber : Perancangan

3.2.1 Analisis Kebutuhan

Analisis kebutuhan ini bertujuan untuk mendapatkan semua kebutuhan yang diperlukan oleh sistem yang akan dibangun. Analisis kebutuhan dilakukan dengan mengidentifikasi kebutuhan sistem dan siapa saja yang terlibat di dalamnya.

3.2.1.1 Analisis Data

Analisis data bertujuan untuk mendapatkan struktur penyimpanan data yang dibutuhkan aplikasi *multiple object tracking* dengan metode *background substraction dan* SIFT.

Struktur penyimpanan data pada aplikasi ini disusun berdasarkan analisis data sebagai berikut :

1) Data uji yang akan digunakan dalam aplikasi ini didapatkan dari BPTIIK selaku badan pemegang data CCTV di lingkungan PTIIK Universitas Brawijaya Malang. Kamera CCTV tersebut merupakan merk TP-LINK TP-SC3000 3GPP dengan resolusi sebesar 320x240 dan *frame rate* 30 fps.
2) Data *template background* yang dibutuhkan dalam proses *background substraction* pada tahap *object detection* akan disimpan.

3.2.1.2 Identifikasi Aktor

Tahap ini adalah tahap untuk melakukan identifikasi terhadap aktor-aktor yang akan berinteraksi dengan aplikasi *multiple object tracking* dengan metode SIFT. Tabel 3.4 menunjukkan aktor beserta deskripsinya yang merupakan hasil dari proses identifikasi aktor.

Tabel 3.4 Identifikasi Aktor

Aktor	Deskripsi Aktor
Pengguna	Pengguna merupakan aktor tunggal yang menggunakan sistem *multiple object tracking* dengan metode *background substraction* dan SIFT.

Sumber : Perancangan

3.2.1.3 Daftar Kebutuhan

Daftar kebutuhan terdiri dari kebutuhan fungsional dan kebutuhan non-fungsional. Spesifikasi kebutuhan fungsional ditunjukkan pada Tabel 3.5. Sedangkan spesifikasi kebutuhan non-fungsional ditunjukkan pada Tabel 3.6 .

Tabel 3.5 Identifikasi Daftar Kebutuhan Fungsional

ID	*Requirements*	Aktor	Nama Proses
SRS_001	Sistem harus menyediakan antarmuka untuk menyimpan citra *background*.	Pengguna	Memasukkan Citra *Background*
SRS_002	Sistem harus menyediakan antarmuka untuk mengatur mode notifikasi sesuai keperluan dan untuk memasukkan data video sesuai dengan format yang ditentukan.	Pengguna	Mengatur Mode Notifikasi dan Memasukkan Video
SRS_003	Sistem memiliki kemampuan untuk melakukan proses deteksi objek pada data video yang dimasukkan oleh pengguna, yaitu dengan melakukan pengubahan warna data video ke warna keabu-abuan (*grayscale*), *background substraction* , *median filter* dan *min filter* serta penentuan batas objek.	Sistem	Deteksi Objek
SRS_004	Sistem memiliki kemampuan untuk melakukan pelacakan dengan metode SIFT pada data video yang telah melalui proses deteksi objek.	Sistem	Pelacakan SIFT
SRS_005	Sistem memiliki kemampuan untuk menampilkan hasil dari pelacakan metode SIFT dengan menandai daerah sekitar objek menggunakan kotak merah, dan mengaktifkan notifikasi jika mode notifikasi aktif.	Sistem	Hasil Pelacakan dan Notifikasi

Sumber : Perancangan

Tabel 3.6 Daftar Kebutuhan Non-fungsional

Parameter	Deskripsi Kebutuhan
Availability	Perangkat lunak harus dapat beroperasi terus-menerus selama waktu yang diinginkan (24 jam atau bahkan melebihi jam kerja).
Interoperability	Fungsi-fungsi semua operasi perangkat lunak harus dapat berjalan dengan baik, khususnya di sistem C# .NET.
Portability	Perangkat lunak harus dapat dijalankan di semua versi Windows yang ada.
Usability	Perangkat lunak harus dapat digunakan dengan mudah oleh pengguna

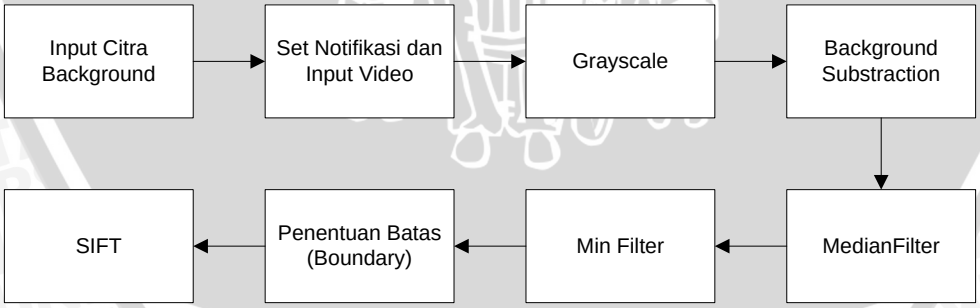
Sumber : Perancangan

3.2.2 Perancangan Perangkat Lunak

Perancangan perangkat lunak terdiri dari tiga tahap, yaitu perancangan proses dan perancangan antarmuka pengguna.

3.2.2.1 Perancangan Proses

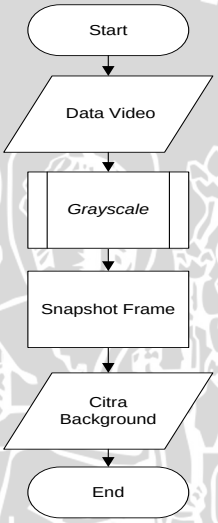
Perancangan proses merupakan perancangan tahap atau urutan sistem saat melakukan proses *multiple object tracking* dengan metode SIFT. Gamba 3.4 mengilustrasikan urutan proses yang akan dilakukan oleh sistem *multiple object tracking* ini.



Gambar 3.4 Urutan Proses Sistem

Berdasarkan diagram blok pada Gambar 3.2, dapat dilihat bahwa secara garis besar aplikasi ini memiliki dua sistem utama yaitu sistem deteksi objek dengan *background substraction* dan sistem pelacakan objek dengan SIFT. Pada sistem deteksi objek terdapat tiga proses yaitu proses mengubah data video ke

warna keabu-abuan (*grayscale*), mendeteksi objek dengan *background substraction*, dan filterisasi dengan *median filter* serta *min filter* dan penentuan batas objek. Sedangkan pada sistem pelacakan SIFT terdiri dari empat sub proses, yakni *scale-space extrema detection, keypoint localization, orientation assignment* dan *keypoint descriptor*. Sebelum proses pendeteksian dan pelacakan objek dilakukan, pengguna terlebih dahulu harus mempersiapkan citra *background* yang akan digunakan. Diagram alir sistem untuk proses memasukkan *citra background* dapat dilihat pada gambar 3.5.



Gambar 3.5 Diagram Alir Memasukkan Citra *Background*

Sumber : Perancangan

Proses pertama adalah memasukkan data citra *background* yang nantinya akan digunakan sebagai *background* dalam proses *background substraction* saat tahap *object detection*. Citra *background* ini didapat dari hasil *snapshot* salah satu *frame* pada video rekaman CCTV dengan tempat yang sama dimana pada *frame* tersebut tidak terdapat objek bergerak satupun. Sehingga citra *background* disini adalah citra yang merupakan pencitraan dari objek-objek pasif yang kedudukannya tidak berubah (*absolute*). Perancangan algoritma memasukkan data citra *background* mengacu pada spesifikasi kebutuhan SRS_001 dapat dilihat pada gambar 3.6.

```
A. PREPARATION
```

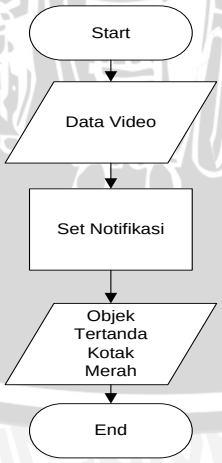
```
Nama algoritma : Memasukkan Citra Background
Deskripsi : Proses memasukkan citra background ke dalam sistem
Deklarasi  :
    • Bitmap : citra_background, bitmap_frame

B. ALGORITMA
Masukan  :
    • Citra Video
Proses :
    1. Menekan tombol "Pilih video"
    2. Memilih video pada kotak file dialog windows
    3. Menekan tombol "Play"
    4. Sistem akan mengubah video ke warna abu-abu dan
       menampilkannya
    5. Menekan tombol "Pause" pada saat frame yang terpilih
       sebagai citra background ditampilkan
    6. Sistem akan menampilkan frame yang tertangkap saat
       tombol "Pause" tertekan pada picturebox
    7. Menekan tombol "Simpan"
    8. Sistem akan melakukan proses penyimpanan
Keluaran :
    • Data citra background
```

Gambar 3.6 Perancangan Algoritma Memasukkan Citra *Background*

Sumber : Perancangan

Setelah citra *background* telah didapatkan maka sistem selanjutnya akan beralih ke proses pengaturan mode notifikasi dan memasukkan citra video yang akan dilacak. Diagram alir sistem memasukkan video dan set notifikasi dapat dilihat pada Gambar 3.7 .



Gambar 3.7 Diagram Alir Memasukkan Video Dan Set Notifikasi

Pertama yang dilakukan adalah mengatur mode notifikasi dan memasukkan video yang akan dilacak. Perancangan algoritma dalam

mengatur mode notifikasi dan memasukkan video mengacu pada spesifikasi kebutuhan SRS_002 dapat dilihat pada Gambar 3.8.

A. PREPARATION
Nama algoritma : Memasukkan Video dan Mengatur Mode Notifikasi
Deskripsi : Proses memasukkan data video yang akan dilacak, selanjutnya mengatur mode notifikasi apakah bernilai on atau off
Deklarasi :
- Bitmap : Abitmap, mode_notifikasi

B. ALGORITMA
Masukan :
- Mode Notifikasi
- Abitmap

Proses :

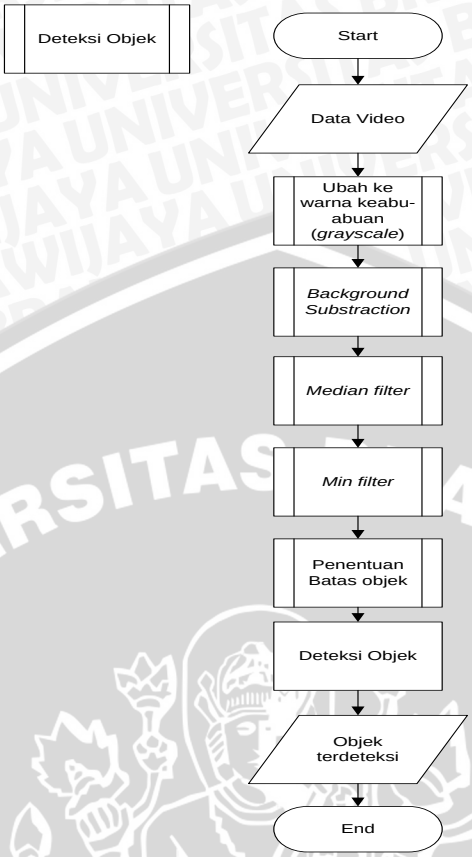
1. Memilih jenis mode notifikasi pada form "Mode Notifikasi".
2. Menekan tombol "Pilih Video"
3. Memilih file video yang akan dilacak pada open file dialog.
4. Sistem akan melakukan proses selanjutnya.

Keluaran :
- Data video yang akan diuji
- Nilai Mode Notifikasi

Gambar 3.8 Perancangan Algoritma Mengatur Mode Notifikasi dan Memasukkan Video

Sumber : Perancangan

Setelah proses mengatur memasukkan video dan mode notifikasi, selanjutnya sistem akan melakukan proses deteksi objek. Pada proses deteksi objek ini, sub proses pewarnaan abu-abu (*grayscale*) dan *median filter* dilakukan dengan menggunakan *library* yang terdapat pada VideoLab. Diagram alir sistem untuk proses deteksi objek ditunjukkan pada Gambar 3.9.



Gambar 3.9 Diagram Alir Proses Deteksi Objek

Sumber : Perancangan

Proses deteksi objek bertujuan untuk mendeteksi objek yang akan dilacak dengan metode SIFT nanti. Dengan adanya proses deteksi objek ini, pengambilan informasi atau pengolahan yang dilakukan terhadap citra akan lebih menghasilkan hasil yang maksimal. Pada proses ini pertama-tama sistem akan melakukan proses pengubahan ke citra *grayscale*. Proses ini bertujuan untuk mengubah 3 channel warna (*Red, Green, Blue*) pada citra menjadi 1 channel sehingga akan mempermudah pengolahan terutama pada proses *object substraction* nanti. Selanjutnya citra hasil *grayscale* akan melalui proses *background substraction* untuk mendeteksi adanya objek yang bisa dilacak dengan metode SIFT atau tidak. Hasil yang didapatkan dari proses ini yaitu warna *foreground grayscale* objek yang berhasil dideteksi. Untuk menghilangkan citra *background* yang lolos dari *background substraction*, maka dilakukan filterisasi dengan *median filter*. Untuk memperkuat citra objek, maka dilakukan min filter yang dilanjutkan dengan

proses penentuan batas objek. Perancangan algoritma deteksi objek mengacu pada spesifikasi kebutuhan SRS_003 dapat dilihat pada Gambar 3.10.

A. PREPARATION
Nama algoritma : Deteksi Objek
Deskripsi : Proses deteksi objek ini terdiri dari proses pengubahan ke warna *grayscale*, *background substraction*, *median filter*,*min filter dan penentuan batas objek*
Deklarasi:

- Bitmap : Abitmap, background, citra, hasil_substract, boundary
- Int : min_x, min_y, maks_x, maks_y

B. ALGORITMA
Masukan :

- Abitmap

Proses :

a) Memasukkan data video ke dalam parameter class "Grayscale"
b) Memasukkan data video hasil "Grayscale" ke dalam program *background substraction*.
c) Memproses hasil *foreground* objek pada "MedianFilter"
d) Memperkuat piksel objek dengan "Min Filter"
e) Menentukan batas objek
f) Menyimpan koordinat piksel daerah citra objek yang berhasil dideteksi dalam variabel min_x, min_y, maks_x, maks_y

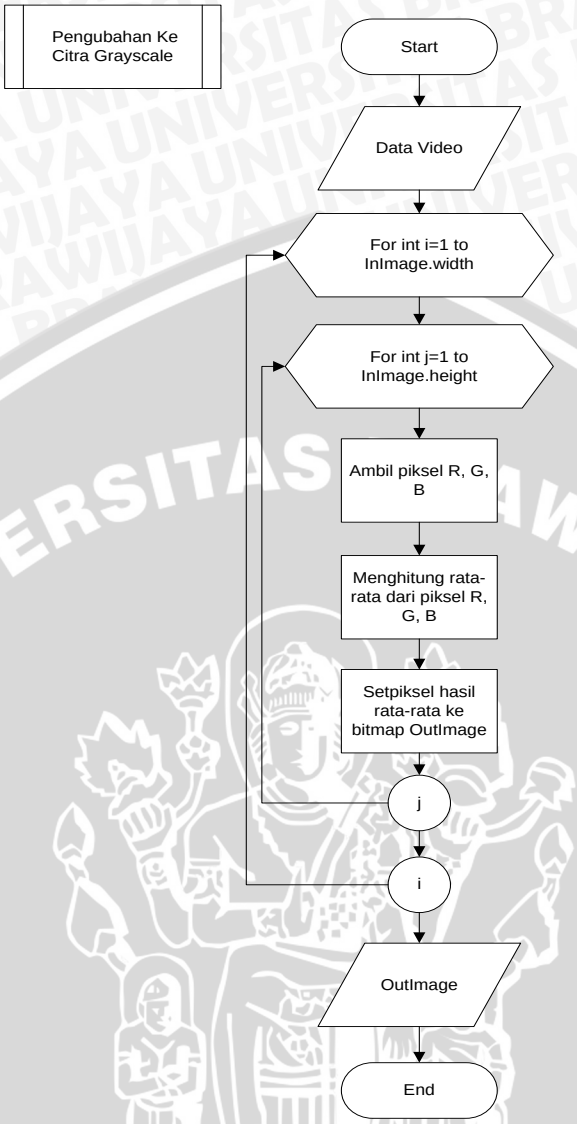
Keluaran :

- Bitmap *frame* data video

Gambar 3.10 Perancangan Algoritma Proses Deteksi Objek

Sumber : Perancangan

Pada proses deteksi objek, pertama-tama sistem akan mengubah tiga *channel* warna dari citra menjadi satu citra sehingga dihasilkan warna keabu-abuan (*grayscale*). Diagram alir sistem proses pengubahan warna citra uji ke citra *grayscale* dapat dilihat pada Gambar 3.11.



Gambar 3.11 Diagram Alir Proses Pengubahan Citra *Grayscale*.

Sumber : Perancangan

Perancangan algoritma dalam proses pengubahan warna ke citra *grayscale* ditunjukkan pada Gambar 3.12.

```
A. PREPARATION
Nama algoritma : Sub Pengubahan Ke Citra Grayscale
Deskripsi      : Proses ini mengubah tiga channel warna ke channel
                 tunggal yaitu mengisi nilai piksel dengan nilai rata-
                 rata ketiga channelnya.
Deklarasi:
    • Bitmap : InImage, OutImage
    • Int : avg
```

B. ALGORITMA

Masukan :

- InImage

Proses :

a) Memasukkan data video yang ada di *buffer* ke dalam class "GrayScale" dan disimpan ke parameter InImage
b) Mengambil nilai RGB tiap piksel.
c) Menghitung nilai rata-rata dari ketiga piksel RGB tiap piksel.
d) Menyimpan nilai rata-rata ke dalam variabel avg dan memasukkannya ke dalam bitmap OutImage.
e) Mengembalikan nilai piksel OutImage.

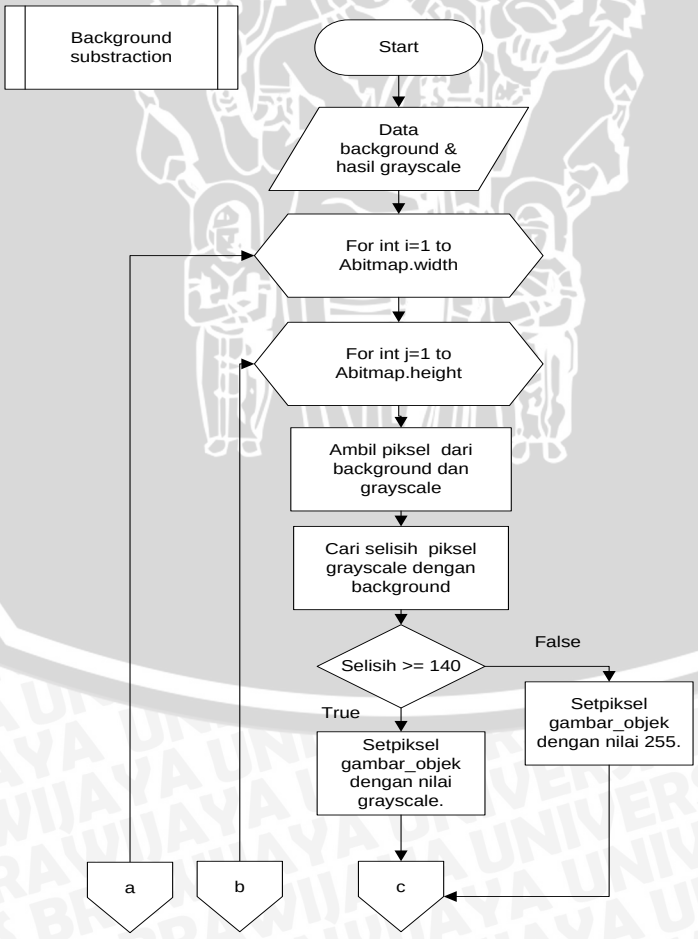
Keluaran :

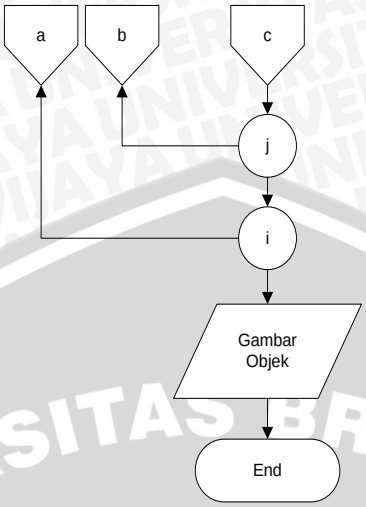
- Bitmap OutImage hasil proses *grayscale*.

Gambar 3.12 Perancangan Algoritma Pengubahan ke Warna *Grayscale*

Sumber : Perancangan

Selanjutnya dilakukan proses *background substraction*. Diagram alir sistem untuk proses *background substraction* ditunjukkan pada Gambar 3.13.





Gambar 3.13 Diagram Alir Proses *Background Substraction*

Sumber : Perancangan

Perancangan algoritma pada *background substraction* dapat dilihat pada Gambar 3.14.

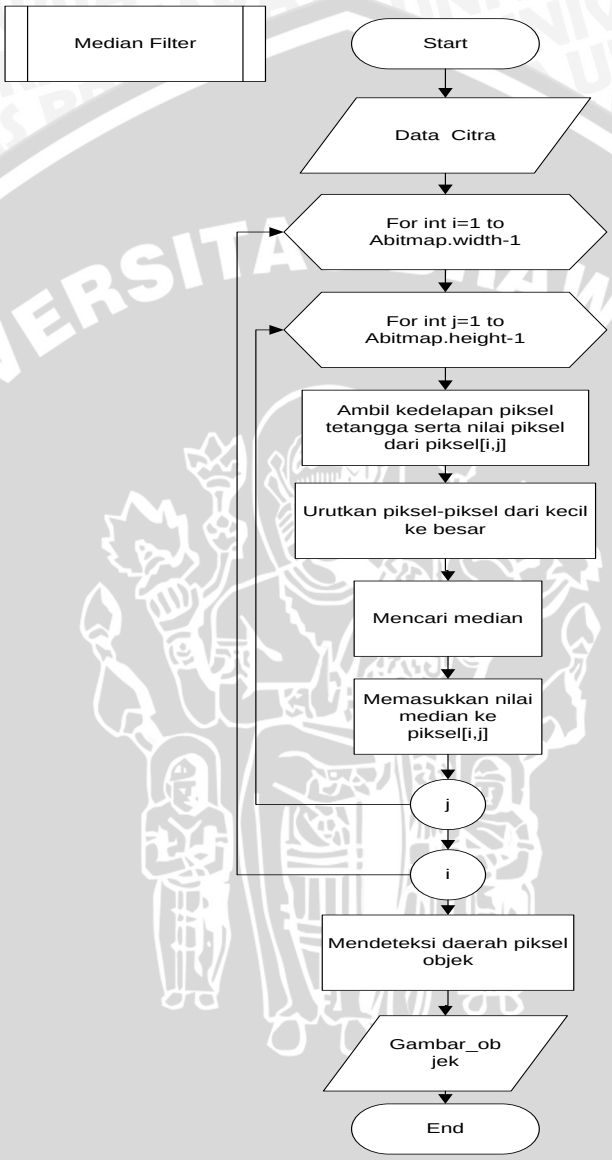
```
A. PREPARATION
Nama algoritma : Background Substraction
Deskripsi : Proses ini memisahkan warna citra background dan
            foreground.
Deklarasi:
    • Bitmap : Abitmap, gambar_background, gambar_objek,
               gambar_grayscale.
    • Int : piksel
B. ALGORITMA
Masukan :
    • Abitmap
Proses :
    a) Memasukkan piksel dari bitmap Abitmap ke gambar_grayscale.
    b) Memasukkan piksel citra background ke bitmap
       gambar_background.
    c) Mengambil nilai piksel dan menghitung nilai selisih
       keduanya.
    d) Jika nilai selisih >= 140, maka piksel diset menjadi nilai
       piksel pada gambar_grayscale.
    e) Jika nilai selisih <140, maka piksel diset menjadi 255.
    f) Mengaplikasikan piksel pada gambar_objek

Keluaran :
Bitmap gambar_objek hasil background substraction.
```

Gambar 3.14 Perancangan Algoritma *Background Substraction*

Sumber : Perancangan

Setelah didapatkan citra *foreground* dari objek, selanjutnya dilakukan proses *median filter* untuk menghilangkan sisa-sisa citra *background* yang lolos dari *background substraction*. Diagram alir sistem *median filter* diilustrasikan pada Gambar 3.15.



Gambar 3.15 Diagram Alir *Median Filter*

Sumber : Perancangan

Perancangan algoritma pada proses *median filter* dapat dilihat pada Gambar 3.16.

A. PREPARATION
Nama algoritma : *Median Filter*
Deskripsi : Proses ini menghilangkan citra background yang lolos dari proses *background substraction* dan menajamkan daerah objek.
Deklarasi:
- Bitmap : gambar_objek, InImage, OutImage.
- Int : median
- Array int :piksel[10]

B. ALGORITMA
Masukan :
- Gambar_objek

Proses :
a) Mengambil piksel dari 3x3 matriks citra gambar_objek kolom pertama dan baris pertama.
b) Mengurutkan nilai-nilai piksel dari terkecil hingga terbesar
c) Menghitung nilai median dari 9 piksel tersebut
d) Memasukkan nilai median ke piksel tengah matriks 3x3 tersebut.
e) Menyimpan titik-titik piksel citra objek

Keluaran :
Bitmap gambar_objek hasil *median filter*.

Gambar 3.16 Perancangan Algoritma *Median Filter*

Sumber : Perancangan

Citra objek hasil *median filter* ini selanjutnya akan disaring kembali dengan *min filter*. Tujuannya agar piksel dari objek semakin kuat dan daerah piksel objek semakin akurat. Kemudian akan ditentukan batas dari objek tersebut. Diagram alir untuk proses *min filter* dapat dilihat pada Gambar 3.17.

Gambar 3.17 Diagram Alir *Min Filter*

Sumber : Perancangan

Perancangan algoritma pada proses *min filter* dapat dilihat pada Gambar 3.18.

A. PREPARATION
Nama algoritma : *Min Filter*
Deskripsi : Proses ini menguatkan piksel dari citra objek dan menentukan batasnya.
Deklarasi:
- Bitmap : gambar_objek, InImage, OutImage.
- Int : median
- Array int :piksel[10]

B. ALGORITMA
Masukan :
- Gambar_objek

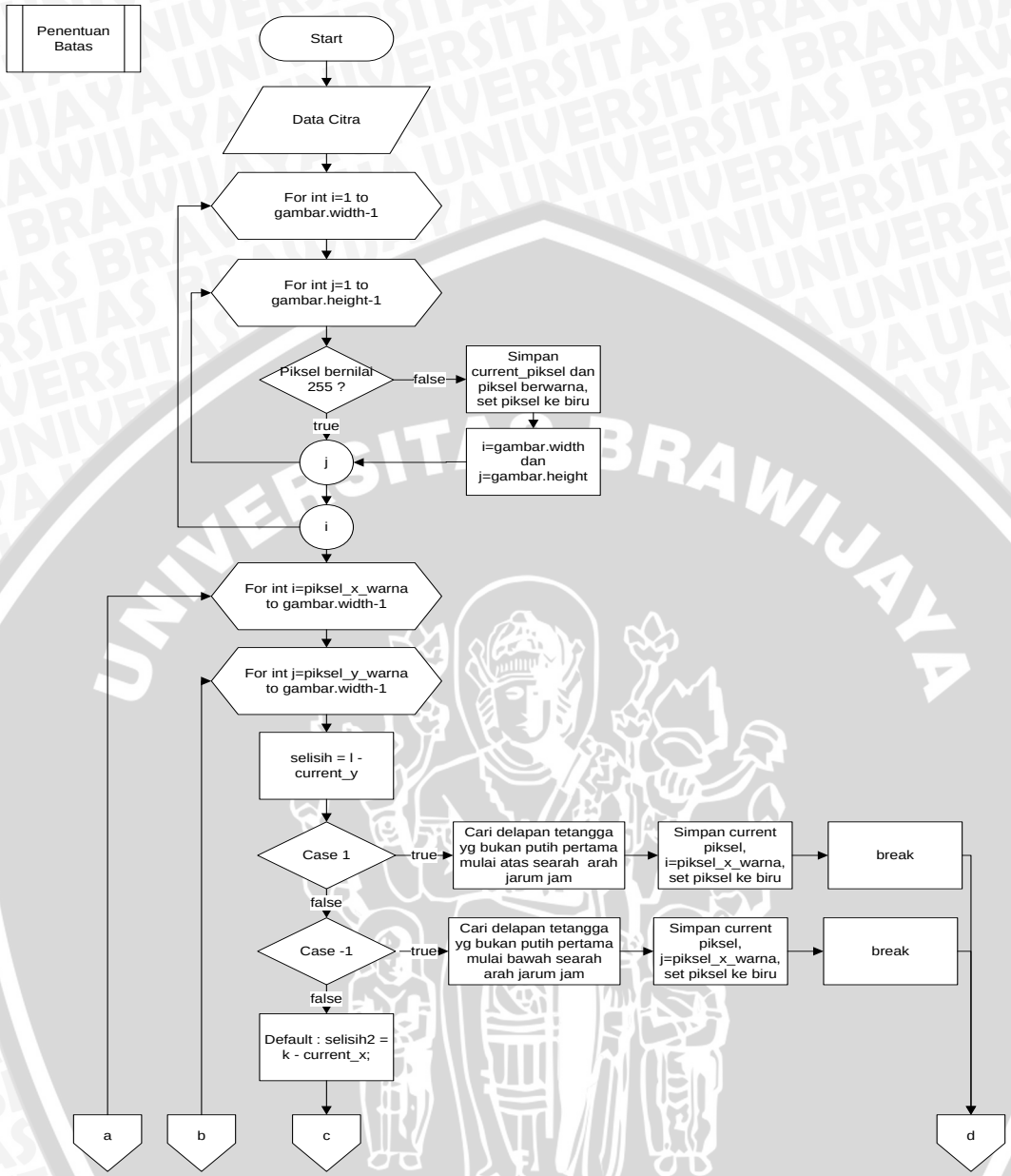
Proses :
a) Mengambil piksel dari 3x3 matriks citra gambar_objek kolom pertama dan baris pertama.
b) Menghitung nilai min dari 9 piksel tersebut
c) Memasukkan nilai min ke piksel tengah matriks 3x3 tersebut, tahap ini dilakukan sampai semua piksel dari citra objek telah dilalui.

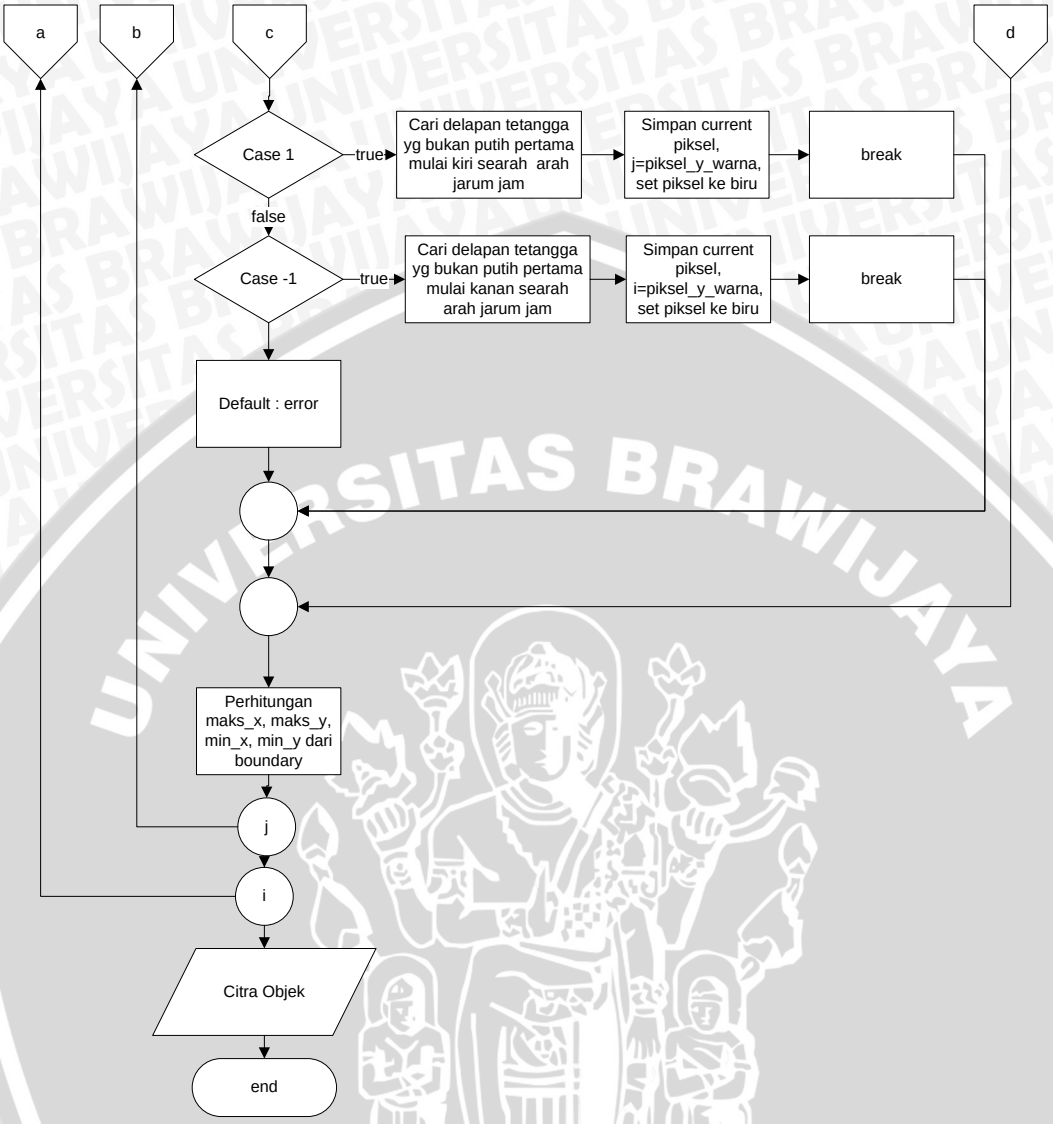
Keluaran :
Bitmap gambar_objek hasil *min filter* dan penentuan batas objek.

Gambar 3.18 Perancangan Algoritma *Min Filter*

Sumber : Perancangan

Setelah dilakukan proses min filter, selanjutnya citra objek akan ditentukan batasnya dengan menggunakan metode *Moore's Neoghbor Tracing*. Diagram alir proses penentuan batas atau boundary ini diilustrasikan pada Gambar 3.19.


Penentuan Batas
Start
Data Citra
For int i=1 to gambar.width-1
For int j=1 to gambar.height-1
Piksel bernilai 255 ?
false
Simpan current_piksel dan piksel berwarna, set piksel ke biru
true
i=gambar.width dan j=gambar.height
j
i
For int i=piksel_x_warna to gambar.width-1
For int j=piksel_y_warna to gambar.width-1
selisih = l - current_y
Case 1
true
Cari delapan tetangga yg bukan putih pertama mulai atas searah arah jarum jam
Simpan current piksel, i=piksel_x_warna, set piksel ke biru
break
false
Case -1
true
Cari delapan tetangga yg bukan putih pertama mulai bawah searah arah jarum jam
Simpan current piksel, j=piksel_x_warna, set piksel ke biru
break
false
Default : selisih2 = k - current_x;
a
b
c
d



Gambar 3.19 Diagram Alir Proses Penentuan Batas Objek

Sumber : Perancangan

Perancangan algoritma untuk proses penentuan batas objek ini dapat dijelaskan pada Gambar 3.20.

```
A. PREPARATION
Nama algoritma : Penentuan Batas Objek
Deskripsi : Proses ini menentukan batas dari tiap-tiap objek
            yang telah diketahui.
Deklarasi:
    • Bitmap : gambar_objek,  InImage, OutImage.
    • Int    :   piksel_x_warna,   piksel_y_warna,   current_x,
               current_y, minx, miny, maksx, maksy
B. ALGORITMA
Masukan :
    • Gambar_objek
```

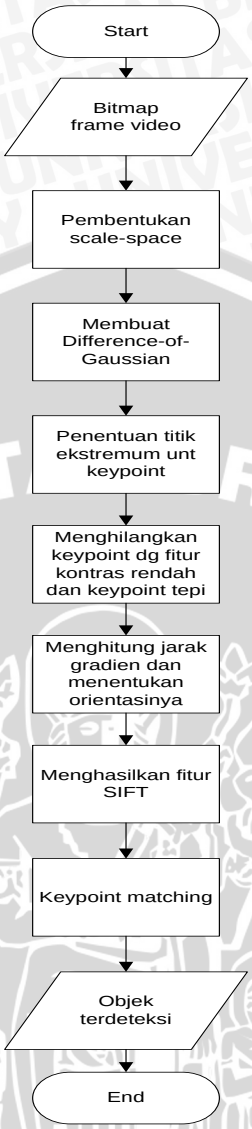
```
Proses :
   a) Mencari titik warna pertama pada citra (nilai piksel !=
      255)
   b) Menyimpan koordinat piksel warna pertama ke
      piksel_x_warna, piksel_y_warna, dan juga menyimpan current
      piksel ke current_x, current_y
   c) Mewarnai piksel warna dengan nilai warna blue
   d) Jika selisih dari piksel_x_warna dan current_x == 1 maka
      pencarian piksel 8 tetangga yang tidak berwarna putih
      dimulai dari atas searah jarum jam. Menyimpan piksel
      tetangga berwarna ke piksel_x_warna dan piksel_y_warna.
   e) Jika selisih dari piksel_y_warna dan current_y == -1 maka
      pencarian piksel 8 tetangga yang tidak berwarna putih
      dimulai dari kanan searah jarum jam. Menyimpan piksel
      tetangga berwarna ke piksel_x_warna dan piksel_y_warna.
   f) Jika selisih dari piksel_x_warna dan current_x == -1 maka
      pencarian piksel 8 tetangga yang tidak berwarna putih
      dimulai dari bawah searah jarum jam. Menyimpan piksel
      tetangga berwarna ke piksel_x_warna dan piksel_y_warna.
   g) Jika selisih dari piksel_y_warna dan current_y == 1 maka
      pencarian piksel 8 tetangga yang tidak berwarna putih
      dimulai dari kanan searah jarum jam. Menyimpan piksel
      tetangga berwarna ke piksel_x_warna dan piksel_y_warna.
   h) Mencari nilai maksx dan minx dari piksel_x_warna, dan
      mencari nilai maksy dan miny dari piksel_y_warna.

Keluaran :
Koordinat batas dari objek
```

Gambar 3.20 Perancangan Algoritma Penentuan Batas

Sumber : Perancangan

Setelah proses deteksi objek maka selanjutnya sistem akan melakukan proses pelacakan dengan metode SIFT. Diagram alir untuk proses pelacakan SIFT ditunjukkan pada Gambar 3.21.



Gambar 3.21 Diagram Alir Sistem Pelacakan SIFT

Sumber : Perancangan

Perancangan algoritma dari sistem pelacakan SIFT merujuk kepada SRS_004 dapat dilihat pada Gambar 3.22.

```
A. PREPARATION
Nama algoritma : Pelacakan SIFT
Deskripsi : Proses ini mengekstraksi fitur keypoint dari
            citra objek dan melakukan proses matching dengan
            citra pada frame selanjutnya.
Deklarasi:
    • Bitmap : gambar_objek.
```

B. ALGORITMA

Masukan :

- Gambar_objek

Proses :

a) Melakukan proses pemburaman dengan 5 level buram yang berbeda-beda (σ, $\sqrt{2}\sigma$, 2σ, $2\sqrt{2}\sigma$, 4σ dengan $\sigma = 1$) terhadap gambar objek.
b) Melakukan *resize* terhadap citra objek dengan ukuran setengah dari ukuran objek original.
c) Melakukan proses pemburaman pada citra objek hasil *resize* dengan 5 level buram yang berbeda-beda.
d) Melakukan *resize* pada citra hasil *resize* dengan ukuran setengah dari ukuran sebelumnya.
e) Melakukan proses pemburaman pada citra objek hasil resize dengan 5 level buram yang berbeda-beda.
f) Mencari *difference of Gaussian* dari citra yang telah diburamkan dengan level-level yang berbeda pada tiap oktaf ukuran citra.
g) Membandingkan titik-titik piksel pada citra *difference of Gaussian* dengan 26 tetangganya yakni kedelapan piksel tetangganya dan 9 piksel pada citra *difference of Gaussian* level sebelumnya dan 9 piksel citra *difference of Gaussian* pada level berikutnya. Bila piksel tersebut menrupakan piksel dengan nilai terbesar atau terkecil dari 26 piksel tersebut, maka piksel tersebut dianggap sebagai keypoint.
h) Menghilangkan titik-titik dari *keypoint* yang memiliki kontras fitur rendah. Kontras fitur rendah dihitung dari jaraknya dari intensitas.
i) Menghilangkan *keypoint* yang menjadi tepi dari objek dan mengambil sudut atau "corner" dari keypoint dengan menggunakan persamaan *Hessian*.
j) Menghitung jarak antar *gradient* dan menentukan orientasinya.
k) Menghasilkan fitur SIFT dengan menghitung vektor *keypoint descriptor* dengan fungsi *gaussian weighting.*
l) Melakukan *keypoint matching* dengan menghitung jarak tetangga terdekat menggunakan *euclidian*.

Keluaran :

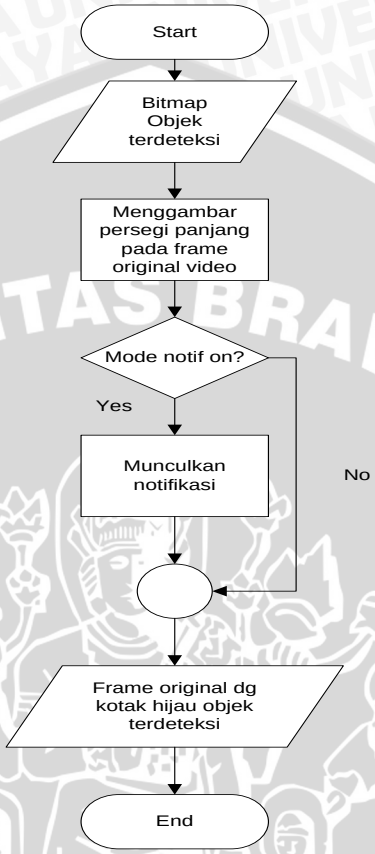
Daerah objek yang telah terdeteksi.

Gambar 3.22 Perancangan Algoritma Pelacakan SIFT

Sumber : Perancangan

Objek manusia pada video telah berhasil dideteksi pada sistem pelacakan SIFT, selanjutnya sistem akan menampilkan hasil deteksi dari objek tersebut pada frame video dengan menandainya menggunakan kotak berwarna merah. Jika mode notifikasi diaktifkan, maka notif juga akan

segera ditampilkan berupa *pop up window*. Diagram alir hasil pelacakan dan notifikasi dapat dilihat pada Gambar 3.23.



Gambar 3.23 Diagram Alir Hasil Pelacakan dan Notifikasi

Sumber : Perancangan

Perancangan algoritma dari hasil pelacakan SIFT dan notifikasi merujuk kepada SRS_005 dapat dilihat pada Gambar 3.24.

```
A. PREPARATION
Nama algoritma : Pelacakan SIFT
Deskripsi : Proses ini mengekstraksi fitur keypoint dari
            citra objek dan melakukan proses matching dengan
            citra pada frame selanjutnya.
Deklarasi:
   • Bitmap : gambar_objek.

B. ALGORITMA
Masukan :
   • Gambar_objek

Proses :
   a) Menggambar persegi panjang pada bitmap frame video
```

```
        yang original pada koordinat serta panjang dan lebar
        yang sesuai dengan titik yang telah terdeteksi pada
        pelacakan sift.
    b) Mengecek apakah mode notifikasi "on" atau "off"
    c) Jika "on", maka pop up notifikasi ditampilkan
       dilanjutkan dengan hasil frame video dengan gambar
       kotak merah yang merupakan daerah objek yang
       terdeteksi.
    d) Jika "off", maka hanya akan ditampilkan hasil frame
       video dengan gambar kotak merah yang merupakan daerah
       objek yang terdeteksi.
Keluaran :
Frame video dengan kotak merah hasil deteksi objek
```

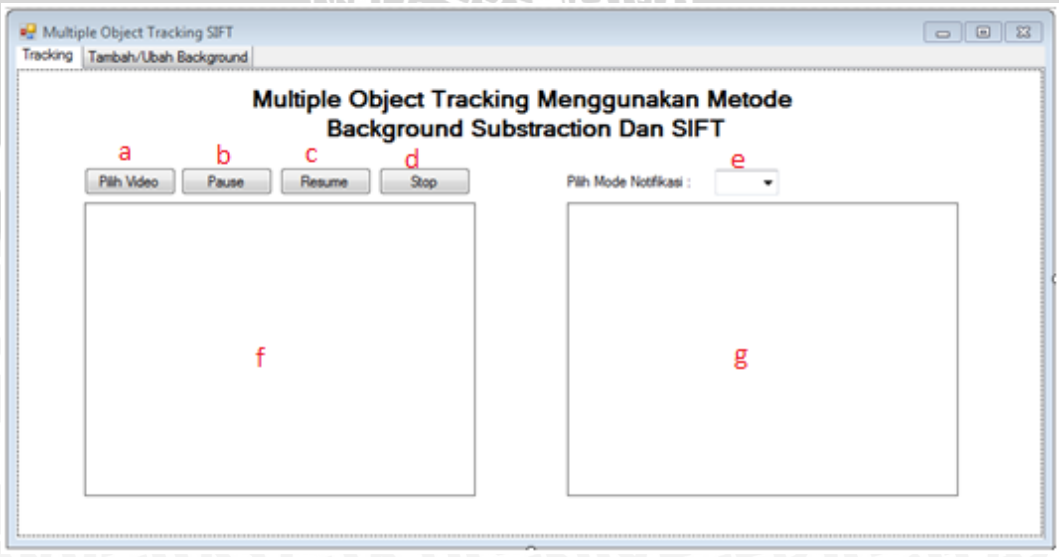
Gambar 3.24 Perancangan Algoritma Hasil Pelacakan dan Notifikasi

Sumber : Perancangan

3.2.2.2 Perancangan Antar Muka

Perancangan antarmuka bertujuan untuk mewakili keadaan sebenarnya dari sistem yang akan dibangun. Sistem *multiple object tracking* dengan algoritma SIFT terdiri dari dua halaman yaitu halaman *tracking* dan halaman input citra *background*.

1. Perancangan antarmuka pengguna halaman *tracking.*



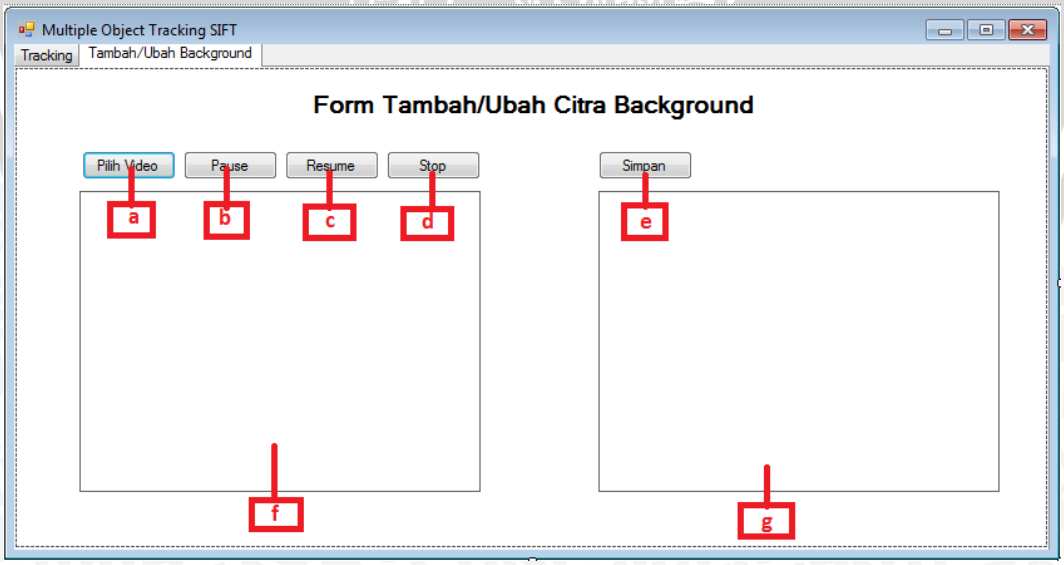
Gambar 3.25 Perancangan Antarmuka Halaman *Tracking*

Sumber : Perancangan

Halaman *tracking* menampilkan proses *tracking* yang dilakukan pada aplikasi *multiple object tracking* dengan metode SIFT. Ilustrasi halaman *tracking* dapat dilihat pada Gambar 3.25. Halaman ini terdiri dari beberapa bagian antara lain :

a. Tombol Memilih Video untuk proses memasukkan data video yang akan dilacak.
b. Tombol Pause untuk menghentikan sementara video.
c. Tombol Resume untuk melanjutkan pemutaran video yang dihentikan sementara.
d. Tombol Tombol Stop untuk menghentikan video.
e. Menu Mode Notifikasi untuk memilih mode notifikasi "On" atau "Off".
f. Picturebox Video Original untuk menampilkan video original.
g. Picturebox Tracking untuk menampilkan video beserta hasil tracking berupa kotak merah dari objek yang berhasil terdeteksi.

2. Perancangan antarmuka pengguna halaman input citra *background*.



Gambar 3.26 Perancangan Antarmuka Halaman Input Citra *Background*

Sumber : Perancangan

Halaman *input* citra *background* menampilkan proses pengambilan snapshot citra *background* dari *video* yang akan dilacak. Ilustrasi halaman *input* citra *background* dapat dilihat pada Gambar 3.26. Halaman ini terdiri dari beberapa bagian antara lain :

a. Tombol Memilih Video untuk proses memasukkan data video yang akan dilacak.
b. Tombol Pause untuk menghentikan sementara video yang diputar sekaligus menangkap *frame bitmap* yang menunjukkan posisi video saat *pause*.
c. Tombol Resume untuk memutar video dimulai dari *state* saat video dihentikan sementara (*pause*).
d. Tombol Stop untuk memberhentikan video yang sedang diputar.
e. Tombol Simpan untuk menyimpan *frame bitmap* yang berhasil ditangkap untuk digunakan sebagai citra *background*.
f. Picturebox Video yang digunakan untuk menampilkan video yang sedang diputar.
g. Picturebox Snapshot Frame yang digunakan untuk menampilkan frame bitmap yang berhasil tertangkap.

3.2.3 Perhitungan Manual

Perhitungan manual berfungsi untuk memberikan gambaran umum perancangan sistem yang akan dibangun. Sistem *multiple object tracking* pada penelitian ini menggunakan metode *Background Substraction* dan *Scale Invariant Feature Trasform* (SIFT). Data uji berupa citra video sebenarnya merupakan kumpulan dari citra *bitmap* yang ditampilkan secara berurutan dengan kecepatan yang sangat tinggi. Pada manualisasi ini, proses yang dilakukan pada data video akan diwakilkan oleh satu *frame bitmap* video yang nantinya akan diilustrasikan proses kalkulasi terhadapnya.

Contoh *frame bitmap* pada video yang akan digunakan pada manualisasi ini tampak pada Gambar 3.27.

Gambar 3.27 Ilustrasi Data *Frame Video*

Sumber : Perancangan

Data tersebut akan diambil nilai pikselnya kemudian nilai-nilai piksel inilah yang akan dikalkulasi pada proses selanjutnya.. Contoh ilustrasi dari nilai piksel *Red*, *Green* dan *Blue* pada citra tersebut tampak pada Gambar 3.28.

Nilai *piksel red* :

192	223	254	251	196	182	216	232
53	151	123	38	17	13	42	109
244	219	26	38	50	59	35	19
238	160	32	65	85	91	55	26
241	87	97	147	185	177	165	76
242	14	94	87	85	136	132	53
243	4	101	49	53	82	44	20
224	97	38	189	160	184	118	38

Nilai *piksel green* :

193	224	255	252	197	183	217	231
54	152	123	39	18	14	43	108
248	218	22	28	37	41	15	21
243	161	29	56	73	73	36	28
247	89	95	139	175	161	146	78
249	15	93	80	75	121	115	55
250	5	101	42	44	68	27	22
233	100	38	183	152	170	102	40

Nilai *piksel blue* :

216	247	255	255	221	206	240	234
78	175	147	62	41	37	66	111
255	255	57	56	57	58	28	28
255	197	58	78	89	85	45	35
255	119	119	156	186	167	150	85
255	39	112	92	79	121	111	62
255	23	115	49	42	63	19	29
251	117	47	184	147	160	90	47

Gambar 3.28 Contoh Ilustrasi Nilai Matriks Piksel RGB Dimensi 8x8.

Sumber : Perancangan

3.2.2.1 Tahap Deteksi Objek Dengan *Background Substraction*

Tahap pertama yang akan dilakukan pada proses deteksi objek yaitu mengubah tiga *channel* warna dari *piksel* menjadi hanya satu *channel* yakni warna abu-abu (*grayscale*). Proses pengubahan citra uji ke citra *grayscale* dilakukan dengan melakukan rata-rata dari tiga *channel* (*red*, *green*, *blue*) warna tiap *piksel*. Nilai rata-rata inilah yang akan menjadi nilai *channel* tunggal dari citra tersebut.

Salah satu contoh perhitungan nilai piksel ganti dari *piksel*[0,0] dari tiga nilai *channel* matriks pada Gambar 3.28 berdasarkan persamaan (2-17) adalah sebagai berikut :

$$mean = \frac{192 + 193 + 216}{3} = 200{,}333 = 200$$

Jadi, nilai *piksel*[0,0] yang sebelumnya 192 untuk *red*, 193 untuk *green* dan 216 untuk *blue* akan diganti dengan nilai satu 200. Hasil perubahan piksel citra uji 8x8 ke citra *grayscale* dari matriks tersebut seperti diilustrasikan pada Gambar 3.29 .

Nilai Piksel :

200	231	255	253	205	190	224	232
62	159	131	46	25	21	50	109
249	231	35	41	48	53	26	23
245	173	40	66	82	83	45	30
248	98	104	147	182	168	154	80
249	23	100	86	80	126	119	57
249	11	106	47	46	71	30	24
236	105	41	185	153	171	103	42

Gambar 3.29 Hasil *Grayscale* Citra *Frame Video*

Sumber : Perancangan

Setelah citra *grayscale* telah dihasilkan, citra selanjutnya akan diproses pada proses *background substraction*. Proses *background substraction* ini bertujuan untuk mendapatkan objek yang akan dilacak dari *frame* pertama.

Background substraction dilakukan dengan mengurangi nilai piksel citra *frame grayscale* dengan piksel citra *background* yang telah di *grayscale* sebelumnya. Gambar 3.30 menggambarkan nilai piksel citra *grayscale* dari *background*.

Nilai Piksel :

176	174	254	249	245	241	197	136
119	118	241	250	190	211	185	121
255	252	251	249	254	250	215	166
254	251	251	242	216	239	201	173
254	255	254	255	255	251	255	254
245	253	248	255	254	255	254	254
217	206	237	254	254	253	255	255
254	248	252	255	255	255	252	255

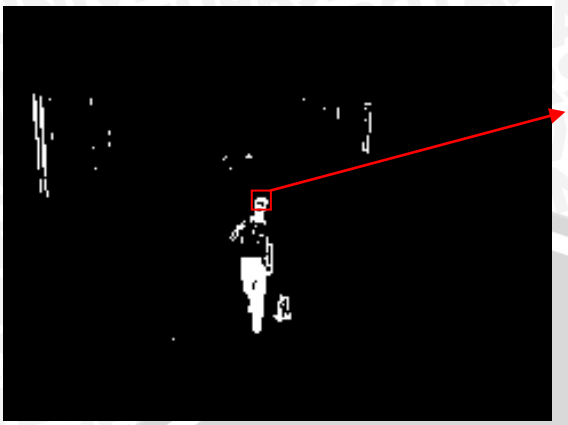
Gambar 3.30 Citra *Background*

Sumber : Perancangan

Nilai piksel dari citra *frame* akan dikurangi dengan nilai piksel dari *background*. Hasil selisih dari dua citra ini nilainya akan dimutlakkan (tidak ada nilai yang negatif). Apabila nilai selisih lebih dari sama dengan 140, maka piksel akan diberi nilai 255, sebaliknya jika piksel kurang dari 140 maka piksel akan diberi nilai 0. Berikut contohnya pada piksel[0,0] pada contoh matriks 8x8 di atas :

$$200 - 176 = 24$$

Karena 4 kurang dari 140 maka nilai piksel[0,0] digantikan dengan nilai 0. Perhitungan yang sama juga dilakukan pada nilai piksel lainnya. Hasil dari *background substraction* diilustrasikan pada Gambar 3.31.



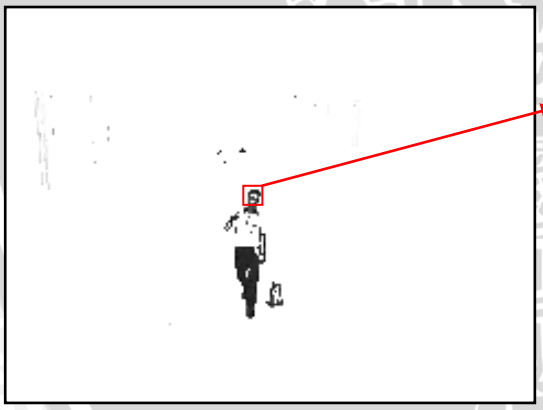
Nilai Piksel :

0	0	0	0	0	0	0	0
0	0	0	255	255	255	0	0
0	0	255	255	255	255	255	255
0	0	255	255	0	255	255	255
0	255	255	0	0	0	0	255
0	255	255	255	255	0	0	255
0	255	0	255	255	255	255	255
0	255	255	0	0	0	255	255

Gambar 3.31 Hasil *Background Substraction*

Sumber : Perancangan

Namun yang dibutuhkan dalam sistem ini berupa citra *foreground* dari objek yang lolos dari nilai *threshold*. Gambar 3.32 adalah citra *foreground* yang didapatkan dari hasil proses *background substraction*.



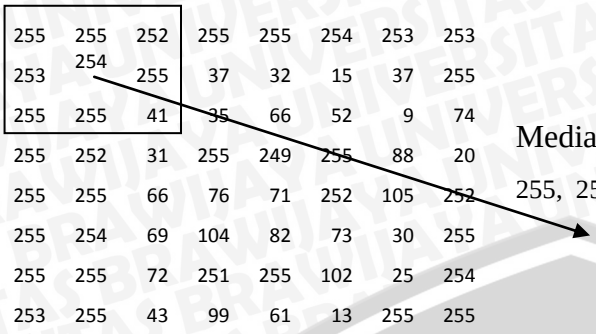
Nilai piksel :

255	255	252	255	255	254	253	253
253	254	255	37	32	15	37	255
255	255	41	35	66	52	9	74
255	252	31	255	249	255	88	20
255	255	66	76	71	252	105	252
255	254	69	104	82	73	30	255
255	255	72	251	255	102	25	254
253	255	43	99	61	13	255	255

Gambar 3.32 Citra *Foreground*

Sumber : Perancangan

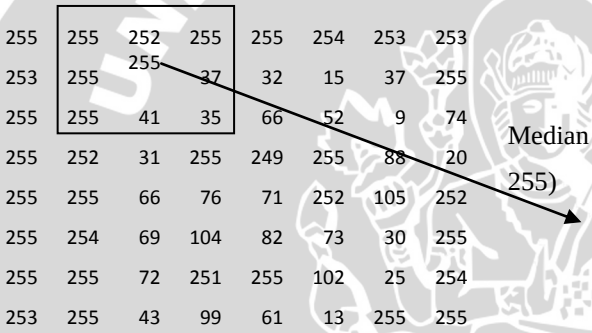
Untuk menghilangkan sisa-sisa piksel citra *background* yang lolos dari nilai *threshold*, maka dilakukan proses *median filter*. Contoh perhitungan nilai median pada piksel[1,1] dari matriks 8x8 dapat dilihat pada Gambar 3.33.



Gambar 3.33 Perhitungan Median Piksel [1,1]

Sumber : Perancangan

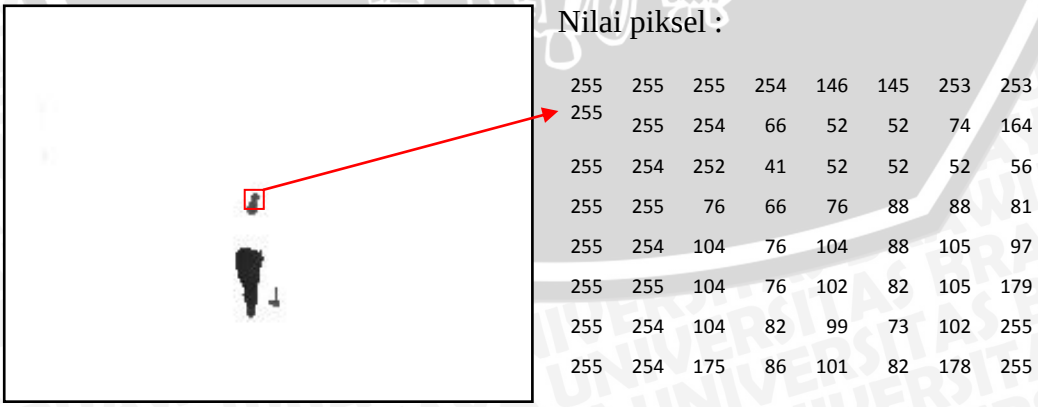
Nilai piksel[1,1] kemudian akan diganti dengan nilai 255. Hal yang sama juga dilakukan pada piksel [2,1] sebagaimana terlihat pada Gambar 3.34 .



Gambar 3.34 Perhitungan Median Piksel[2,1]

Sumber : Perancangan

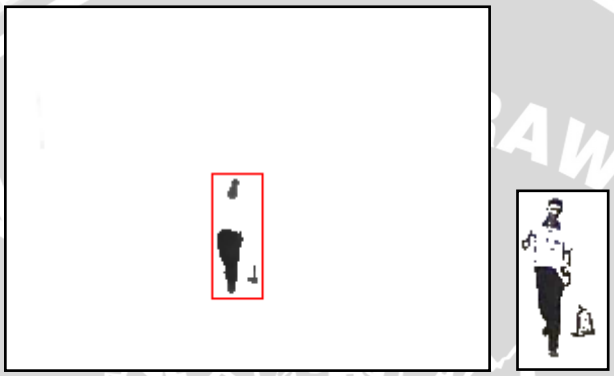
Proses perhitungan akan diteruskan pada setiap piksel pada citra, sehingga contoh hasil *median filter* dapat dilihat pada Gambar 3.35 .



Gambar 3.35 Citra Hasil *Median Filter*

Sumber : Perancangan

Daerah sekitar objek pada citra akan dicari dengan mencari titik terluar dari objek tersebut. Yaitu mencari nilai minimum dan maksimum dari nomor kolom dan baris piksel objek yang telah dideteksi sebagai objek. Piksel-piksel yang berada di daerah piksel objek akan dikembalikan ke nilai piksel *foreground* hasil *background substraction*. Objek yang berhasil dideteksi diilustrasikan pada Gambar 3.36.



(a) (b)

Gambar 3.36 a) Daerah objek yang terdeteksi dari citra hasil *median filter*, b) Citra Objek Yang Diambil Dari Citra *Background Substraction*

Sumber : Perancangan

Citra objek yang telah dipisahkan dari *background* ini selanjutnya akan difilter dengan *min filter* untuk memperkuat pikselnya. Contoh perhitungan nilai min pada piksel [1,1] dapat diilustrasikan pada Gambar 3.37.

255	255	252	255	255	254	253	253
253	254	255	37	32	15	37	255
255	255	41	35	66	52	9	74
255	252	31	255	249	255	88	20
255	255	66	76	71	252	105	252
255	254	69	104	82	73	30	255
255	255	72	251	255	102	25	254
253	255	43	99	61	13	255	255

Min = (41, 252, 253, 254, 255, 255, 255, 255, 255)

=41

Gambar 3.37 Min Piksel [1,1]

Sumber : Perancangan

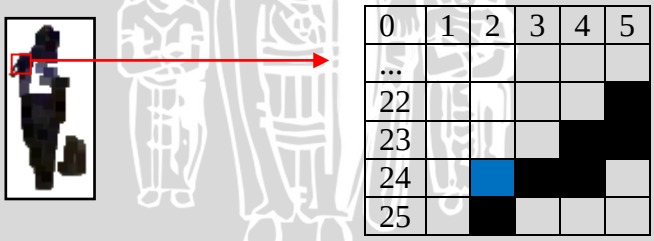
Setelah nilai min telah dihitung, maka nilai piksel [1,1] tersebut akan diganti dengan nilai min yaitu 41. Contoh hasil dari *min filter* dapat dilihat pada Gambar 3.38 .

253	252	37	32	15	15	15	37
253	41	35	32	15	9	9	9
252	31	31	31	15	9	9	9
252	31	31	31	35	9	9	9
252	31	31	31	71	30	20	20
254	66	66	66	71	25	25	25
253	43	43	43	13	13	13	25
253	43	43	43	13	13	13	25

Gambar 3.38 Citra Hasil *Min Filter*

Sumber : Perancangan

Citra objek yang telah dikuatkan dengan min filter tersebut selanjutnya akan dicari batas pikselnya dengan menggunakan metode *Moore's Neighbor Tracing*. Ilustrasi mengenai proses penentuan batas dari blob sebuah objek dapat dilihat pada Gambar 3.39 .

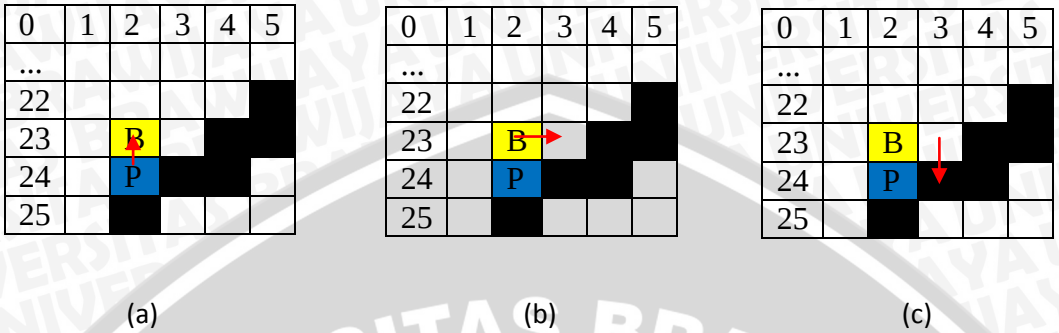


Gambar 3.39 Contoh Bentuk Blob Objek Pada Matriks

Sumber : Perancangan

Pada Gambar 3.39 diketahui piksel pertama yang berwarna bukan putih ditemukan pada piksel [24,2] dengan *backtrack* piksel [23,2]. Berdasarkan keadaan awal tersebut maka titik pusat P akan bernilai [24,2] dengan *backtrack* [3,2]. Penelusuran delapan tetangga dari titik pusat P akan dimulai dari titik *backtrack* dan berlanjut searah jarum jam sampai ditemukan piksel yang tidak

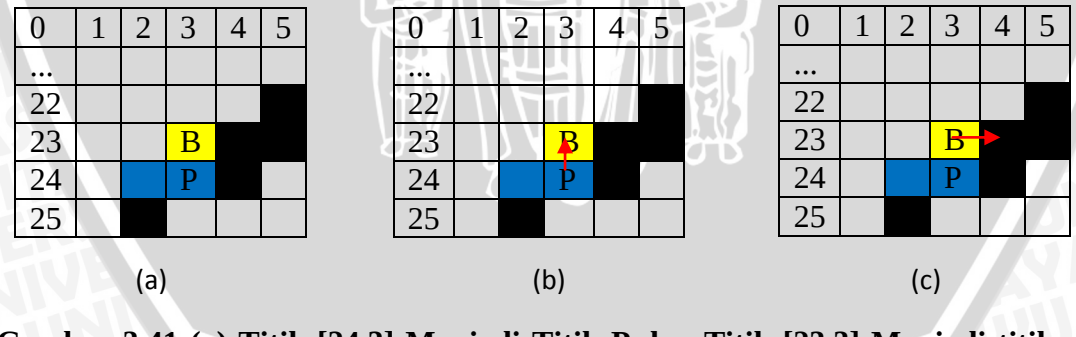
berwarna putih lainnya. Gambar 3.40 adalah contoh penelusuran delapan tetangga dengan P adalah titik pusat dan B adalah *backtrack*.



Gambar 3.40 (a) Penelusuran Pertama Dari Titik *Backtrack*, (b) Penelusuran Kedua Ke Titik [23,3], (c) Penelusuran Ketiga Ke Titik [24,3] Dan Ditemukan Titik Bukan Putih

Sumber : Perancangan

Pada Gambar 3.40 tersebut, piksel bukan putih ditemukan setelah melakukan penelusuran ketiga (Gambar 3.40 c), yakni pada titik [24,3]. Oleh karena itu, titik [24,3] ditetapkan sebagai batas (ditandai dengan warna biru). Titik P kemudian berpindah ke titik [24,3] dan titik *backtrack* sekarang adalah titik [23,3].



Gambar 3.41 (a) Titik [24,3] Menjadi Titik P dan Titik [23,3] Menjadi titik *Backtrack* , (b) Penelusuran Pertama dari Titik *Backtrack*, (c) Penelusuran Kedua Ke Titik [23,4] Dan Ditemukan Titik Bukan Putih.

Sumber : Perancangan

Pada Gambar 3.41 tersebut, piksel bukan putih ditemukan setelah melakukan penelusuran kedua (Gambar 3.41 c), yakni pada titik [23,4]. Maka titik [23,4] ditetapkan sebagai batas, dan titik P berpindah ke titik [23,4] serta titik *backtrack* ke titik [23,3]. Begitupun selanjutnya sampai semua batas dari blob objek telah ditemukan. Penelusuran batas akan berakhir jika ditemukan titik bukan putih pada nomor baris dan kolom yang sama pada baris dan kolom titik warna pertama, pada contoh ini titik warna pertama adalah titik [24,2]. Contoh hasil pendeteksian batas diilustrasikan pada Gambar 3.42.

Gambar 3.42 Contoh Hasil Penentuan Batas Pada Objek

Sumber : Perancangan

Setelah titik-titik batas objek telah ditemukan, akan dicari nilai minimum x, minimum y, maksimum x, dan maksimum y di antara titik-titik batas tersebut. Hal ini digunakan untuk mengambil daerah objek dan mengembalikan nilai piksel di antara titik minimum maksimum tersebut dalam citra *grayscale*. Pencarian titik batas dan minimum maksimum akan terus dilakukan sampai piksel pada citra telah ditelusuri secara menyeluruh. Contoh objek yang berhasil dideteksi dan telah berupa citra *grayscale* dapat dilihat pada Gambar 3.43.

Gambar 3.43 Contoh Hasil Citra Objek Yang Berhasil Dideteksi

Sumber : Perancangan

Objek yang terdeteksi inilah yang akan dijadikan sebagai target untuk dilacak pada *frame-frame* selanjutnya dengan metode SIFT. Terlebih dahulu citra objek ini akan di ekstraksi fitur *keypoint*-nya dengan metode SIFT dan fitur tersebut akan disimpan ke dalam media penyimpanan sebelum dilakukan proses *keypoint matching*.

3.2.2.2 Tahap Pelacakan Objek Dengan Metode SIFT

Proses pelacakan dengan metode SIFT ini diawali dengan melakukan proses *blurring* (pemburaman) menggunakan operator *Gaussian* dengan *scale* (σ) yang berbeda-beda. Selain itu juga dilakukan perubahan ukuran (*resize*) dengan ukuran 75% dari citra sebelumnya. Sehingga akan dibuat 3 macam ukuran citra, dengan tiap tingkat ukuran citra terdapat lima buah citra yang masing-masing merupakan hasil pemburaman dengan lima *scale* yang berbeda. Pada sistem ini, nilai pertama σ = 0,707107 dan konstanta k=√2. Untuk lebih jelasnya mengenai nilai σ yang digunakan, dapat dilihat pada Tabel 3.5.

Tabel 3.5 Nilai *Scale* σ

	Scale →				
Ukuran (*octave*)	0,707107	1,000000	1,414214	2,000000	2,828428
	1,414214	2,000000	2,828428	4,000000	5,656856
	2,828428	4,000000	5,656856	8,000000	11,31371

Sumber : Perancangan

Operator *Gaussian* perlu disiapkan untuk hal ini. Dengan menggunakan persamaan (2-2), maka akan didapatkan operator *Gaussian* 3x3. Proses perhitungan untuk mendapatkan operator *Gaussian* adalah sebagai berikut :

$$G(-1,-1,0{,}707107) = \frac{1}{2.\pi.0{,}707107^2}\, e^{-\frac{-1^2+-1^2}{2.0{,}707107^2}} = 0{,}043079$$

$$G(-1,0,0{,}707107) = \frac{1}{2.\pi.0{,}707107^2}\, e^{-\frac{-1^2+0^2}{2.0{,}707107^2}} = 0{,}117100$$

$$G(-1,1,0{,}707107) = \frac{1}{2.\pi.0{,}707107^2}\, e^{-\frac{-1^2+1^2}{2.0{,}707107^2}} = 0{,}043079$$

Perhitungan yang sama juga dilakukan pada titik (0,-1), (0,0), (0,1), (1,-1), (1,0), dan (1,1). Sehingga operator *Gaussian* yang didapatkan nantinya dengan nilai σ= 0,707107 dapat dilihat pada Gambar 3.44.

0,043079	0,1171	0,043079
0,1171	0,31831	0,1171
0,043079	0,1171	0,043079

Gambar 3.44 Operator *Gaussian* Dengan σ = 0,707107

Sumber : Perancangan

Operator tersebut selanjutnya akan dinormalisasi dengan membagi tiap nilai *kernel* dengan jumlah keseluruhan dari tiap nilai *kernel*. Dari *kernel* pada Gambar 3.40, diperoleh nilai *sum* = 0,959023, sehingga nilai operator setelah dinormalisasi dapat dilihat pada Gambar 3.45 .

0,04491922	0,12210311	0,0449192
0,12210311	0,33191067	0,1221031
0,04491922	0,12210311	0,0449192

Gambar 3.45 Operator *Gaussian* Hasil Normalisasi

Sumber : Perancangan

Perhitungan nilai operator *Gaussian* pada nilai σ yang lain juga dilakukan dengan cara yang sama seperti pencarian operator *Gaussian* dengan nilai σ = 0,707107. Sehingga akan didapatkan nilai operator *Gaussian* pada semua nilai *scale* dan oktaf seperti dilihat pada Tabel 3.5.

Tabel 3.5 Nilai Operator *Gaussian* Tiap *Scale* Dan *Oktaf*

Oktaf	*Scale*	Operator *Gaussian*		
1	σ = 0,707107	0,044919	0,122103	0,044919
		0,122103	0,331911	0,122103
		0,044919	0,122103	0,044919
	σ = 1,000000	0,075114	0,123841	0,075114
		0,123841	0,20418	0,123841
		0,075114	0,123841	0,075114

	σ = 1,414214	0,092723	0,119059	0,092723
		0,119059	0,152874	0,119059
		0,092723	0,119059	0,092723
	σ = 2,000001	0,101868	0,115432	0,101868
		0,115432	0,130801	0,115432
		0,101868	0,115432	0,101868
	σ = 2,828428	0,106484	0,113351	0,106484
		0,113351	0,120662	0,113351
		0,106484	0,113351	0,106484
2	σ = 1,414214	0,092723	0,119059	0,092723
		0,119059	0,152874	0,119059
		0,092723	0,119059	0,092723
	σ = 2,000001	0,101868	0,115432	0,101868
		0,115432	0,130801	0,115432
		0,101868	0,115432	0,101868
	σ = 2,828428	0,106484	0,113351	0,106484
		0,113351	0,120662	0,113351
		0,106484	0,113351	0,106484
	σ = 4,000001	0,108797	0,112250	0,108797
		0,112250	0,115813	0,112250
		0,108797	0,112250	0,108797
	σ = 5,656856	0,109954	0,111685	0,109954
		0,111685	0,113444	0,111685
		0,109954	0,111685	0,109954
3	σ = 2,828428	0,106484	0,113351	0,106484
		0,113351	0,120662	0,113351
		0,106484	0,113351	0,106484
	σ = 4,000001	0,108797	0,11225	0,108797
		0,11225	0,115813	0,11225
		0,108797	0,11225	0,108797
	σ = 5,656856	0,109954	0,111685	0,109954
		0,111685	0,113444	0,111685
		0,109954	0,111685	0,109954
	σ = 8,000001	0,110532	0,111399	0,110532
		0,111399	0,112273	0,111399
		0,110532	0,111399	0,110532
	σ = 11,31371	0,110822	0,111256	0,110822
		0,111256	0,111691	0,111256
		0,110822	0,111256	0,110822

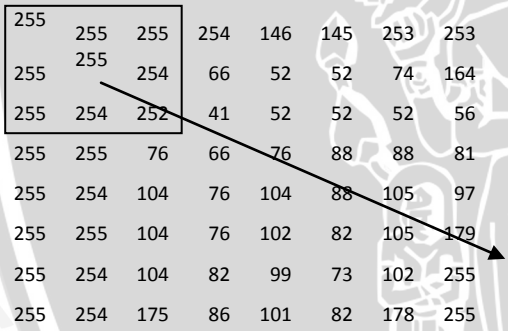
Sumber : Perancangan

Setelah didapatkan operator *Gaussian*, maka akan dilakukan perhitungan *scale space* $L_\sigma = L_{(x, y, \sigma)}$ citra dengan persamaan (2-1). Contoh nilai piksel citra dan operator *Gaussian* yang akan digunakan dalam proses ini adalah :

255	255	255	254	146	145	253	253
255	255	254	66	52	52	74	164
255	254	252	41	52	52	52	56
255	255	76	66	76	88	88	81
255	254	104	76	104	88	105	97
255	255	104	76	102	82	105	179
255	254	104	82	99	73	102	255
255	254	175	86	101	82	178	255

0,044919	0,122103	0,044919
0,122103	0,331911	0,122103
0,044919	0,122103	0,044919

Contoh perhitungan konvolusi antara operator *Gaussian* dan citra dapat dilihat pada Gambar 3.46.



L_σ = (255x0,044919) + (255x0,122103) + (255x0,044919) + (255x0,122103) + (255x0,331911) + (254x0,122103) + (255x0,044919) + (254x0,122103) + (252x0,044919) = 254,5210361= 254

Gambar 3.46 Contoh Perhitungan Konvolusi

Sumber : Perancangan

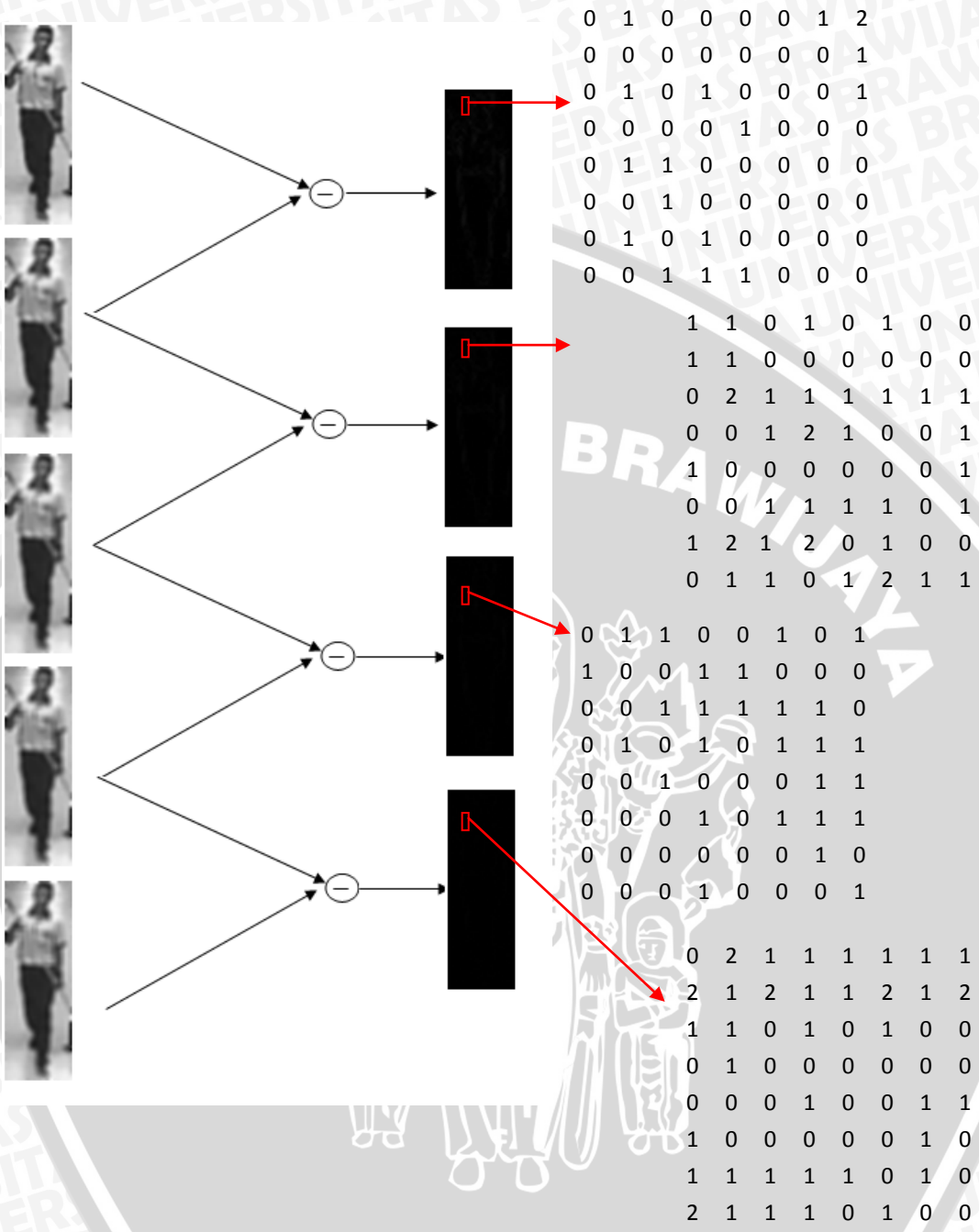
Perhitungan konvolusi tersebut akan terus berulang sampai keseluruhan citra objek telah ditelusuri secara menyeluruh. Setelah proses pemburaman pada citra dengan lima level *scale* yang berbeda, selanjutnya citra original akan dilakukan *resize* dengan besar ¾ dari ukuran semula. Lalu akan dilakukan proses pemburaman lagi dengan *scale* yang telah ditentukan.

Contoh citra hasil konvolusi dengan nilai oktaf dan *scale* sesuai ketentuan pada Tabel 3.5 dapat dilihat pada Gambar 3.47.

Gambar 3.47 Contoh Citra *Scale Space*

Sumber : Perancangan

Setelah *scale space* telah dibentuk, maka selanjutnya akan dihitung *Difference of Gaussian* dengan cara mencari selisih antar citra yang berdekatan dalam satu oktaf. Operasi dilakukan dengan operasi *image substraction* yang dilakukan per piksel. Ilustrasi pencarian *Difference of Gaussian* dapat dilihat pada Gambar 3.48.



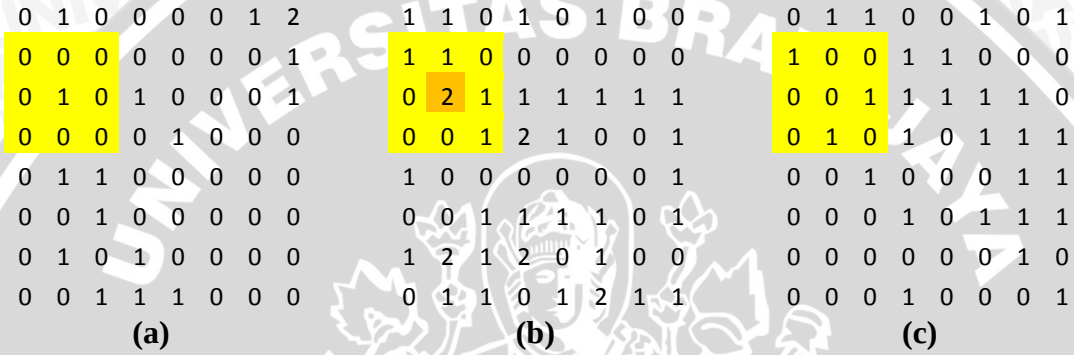
Gambar 3.48 Contoh Citra *Difference Of Gaussian*

Sumber : Perancangan

Proses yang sama juga dilakukan pada citra yang ada di oktaf kedua dan ketiga. Dengan metode pengurangan image ini, terbentuklah sebuah *Difference of Gaussian* dari berbagai ukuran.

Difference of Gaussian telah terbentuk, selanjutnya akan dicari titik ekstremum dan minimum yang akan dijadikan sebagai *keypoint*. Proses pencarian

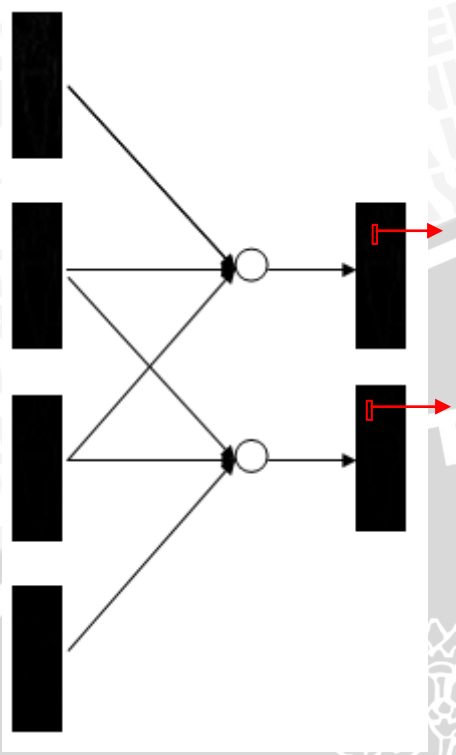
titik ekstremum dan minimum ini dilakukan dengan membandingkan sebuah piksel dengan 26 tetangganya (8 piksel tetangga dari citra yang sama, 9 piksel tetangga dari citra level sebelumnya, 9 piksel tetangga dari citra level sesudahnya). Bila piksel tersebut merupakan piksel terkecil dari 26 tetangganya, dia adalah titik minimum. Sebaliknya jika piksel tersebut merupakan piksel terbesar maka dia adalah titik ekstremum. Titik minimum dan ekstremum merupakan kandidat dari *keypoint*. Contoh proses pencarian titik ektremum dapat dilihat pada Gambar 3.49.



Gambar 3.49 (a) DoG Antara Citra *Scale Space* 1 dan 2, (b) DoG Antara Citra *Scale Space* 2 dan 3, (c) DoG Antara Citra *Scale Space* 3 dan 4

Sumber : Perancangan

Nilai 2 pada kotak oranye akan dibandingkan dengan nilai piksel pada kotak kuning yakni {0,0,0,0,1,0,0,0,0,1,1,0,0,1,0,0,1,1,0,0,0,0,1,0,1,0}. Nilai 2 merupakan nilai terbesar dari deretan nilai tersebut. Berdasarkan hal ini, maka kotak oranye yakni piksel [2,1] dari DoG kedua merupakan titik maksimum. Begitupun selanjutnya akan dilakukan proses perbandingan yang sama sampai semua piksel telah ditelusuri. Proses yang sama juga dilakukan pada citra DoG ketiga yang akan dibandingkan dengan DoG kedua dan keempat. Ilustrasi hasil dari pencarian titik minimum dan maksimum dapat dilihat pada Gambar 3.50.



0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	2	0	2	2	0	0	0
2	2	2	2	0	0	1	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	1
1	2	0	0	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1
0	0	0	0	0	0	1	0

Gambar 3.50 Hasil Deteksi Maksimum dan Minimum

Sumber : Perancangan

Namun pada hasil deteksi titik maksimum dan minimum pada cara di atas masih memiliki kekurangan dimana masih terdapat titik-titik ekstremum yang tidak stabil terhadap *noise* yakni memiliki *low-contrast* dan terletak di daerah tepi (*edge*). Sehingga perlu dilakukan perhitungan ekstremum dengan menggunakan persamaan (2-9). Jika nilai D(z) kurang dari 0,03, maka titik tersebut akan diabaikan atau tidak dianggap sebagai titik ekstremum. Sebelum memulai perhitungan pada persamaan (2-9) tentunya perlu dilakukan perhitungan terlebih dahulu dengan persamaan (2-5) untuk mendapatkan nilai z.

$$z = -\left(\frac{\partial^2 D}{\partial X^2}\right)^{-1} \frac{\partial D}{\partial X}$$

Proses perhitungan nilai z akan dimulai dari piksel [1,1] pada citra DoG kedua (Gambar 3.49 b). Terlebih dahulu perlu ditentukan hasil invers turunan

kedua fungsi *taylor expansion* $\left(\frac{\partial^2 D}{\partial \mathrm{X}^2}\right)^{-1}$. Berikut adalah contoh proses perhitungan yang dilakukan pada piksel [2,1] pada citra dog kedua menggunakan persamaan (2-8) :

$$\left(\frac{\partial^2 D}{\partial \mathrm{X}^2}\right) = H(\mathrm{X}) = \begin{bmatrix} D_{xx} & D_{xy} & D_{x\sigma} \\ D_{yx} & D_{yy} & D_{y\sigma} \\ D_{\sigma x} & D_{\sigma y} & D_{\sigma\sigma} \end{bmatrix}$$

Dengan :

$$D_{xx} = \frac{D(x+1,y,\sigma) - 2D(x,y,\sigma) + D(x-1,y,\sigma)}{1}$$

$$= \frac{0-4+1}{1} = -3$$

$$D_{xy} = \frac{D(x+1,y+1,\sigma) - D(x+1,y-1,\sigma) - D(x-1,y+1,\sigma) + D(x-1,y-1,\sigma)}{4}$$

$$= \frac{1-0-0+1}{4} = 0{,}5$$

$$D_{x\sigma} = \frac{D(x+1,y,\sigma+1) - D(x+1,y,\sigma-1) - D(x-1,y,\sigma+1) + D(x-1,y,\sigma-1)}{4}$$

$$= \frac{1-0-0+0}{4} = 0{,}25$$

Begitupun selanjutnya untuk perhitungan D_{yx}, D_{yy}, $D_{y\sigma}$, $D_{\sigma x}$, $D_{\sigma y}$, dan $D_{\sigma\sigma}$. Sehingga akan didapatkan hasil matriks seperti berikut :

$$\left(\frac{\partial^2 D}{\partial \mathrm{X}^2}\right) = \begin{bmatrix} -3 & 0{,}5 & 0{,}25 \\ 0{,}5 & -3 & 0{,}25 \\ 0{,}25 & 0{,}25 & -3 \end{bmatrix}$$

Jika dilakukan proses invers menjadi :

$$\left(\frac{\partial^2 D}{\partial \mathrm{X}^2}\right)^{-1} = \begin{bmatrix} -0{,}35 & -0{,}06 & -0{,}03 \\ -0{,}06 & -0{,}35 & -0{,}03 \\ -0{,}03 & -0{,}03 & -0{,}34 \end{bmatrix}$$

Setelah turunan kedua telah didapatkan, selanjutnya mencari turunan pertama *taylor* $\frac{\partial D}{\partial \mathrm{X}}$ sebagai berikut :

$$\frac{\partial D}{\partial \mathrm{X}} = \begin{bmatrix} \frac{\partial D}{\partial x} \\ \frac{\partial D}{\partial y} \\ \frac{\partial D}{\partial \sigma} \end{bmatrix} = \begin{bmatrix} \frac{D(x+1,y,\sigma) - D(x-1,y,\sigma)}{2} \\ \frac{D(x,y+1,\sigma) - D(x,y-1,\sigma)}{2} \\ \frac{D(x,y,\sigma+1) - D(x,y,\sigma-1)}{2} \end{bmatrix} = \begin{bmatrix} \frac{0-1}{2} \\ \frac{1-0}{2} \\ \frac{0-1}{2} \end{bmatrix} = \begin{bmatrix} -0{,}5 \\ 0{,}5 \\ -0{,}5 \end{bmatrix}$$

Sehingga nilai z untuk piksel [1,1] adalah sebagai berikut :

$$-\begin{bmatrix}-0{,}35 & -0{,}06 & -0{,}03\\ -0{,}06 & -0{,}35 & -0{,}03\\ -0{,}03 & -0{,}03 & -0{,}34\end{bmatrix} \text{x} \begin{bmatrix}-0{,}5\\ 0{,}5\\ -0{,}5\end{bmatrix} = \begin{bmatrix}-0{,}159\\ 0{,}125\\ -0{,}169\end{bmatrix}$$

Berdasarkan perhitungan nilai z tersebut, elemen z telah memenuhi persyaratan dengan nilai < 0,5. Sehingga nilai z ini dipakai untuk mencari nilai D(z).Langkah selanjutnya yaitu menghitung nilai D(z) dengan menggunakan persamaan (2-6).

$$|D(z)| = D + \frac{1}{2}\frac{\partial D^T}{\partial x} z$$

$$= 0 + \frac{1}{2}\begin{bmatrix}-0{,}5 & 0{,}5 & -0{,}5\end{bmatrix}.\begin{bmatrix}-0{,}159\\ 0{,}125\\ -0{,}169\end{bmatrix}$$

$$= 1{,}1138$$

Nilai Dz tersebut kemudian akan dinormalisasi sehingga rentang nilainya diantara 0 dan 1. Hasil normalisasinya adalah 0,087. Karena nilai D(z) hasil normalisasi > 0,03 maka titik [2,1] dianggap sebagai *keypoint*. Begitupun selanjutnya proses perhitungan D(z) untuk titik piksel yang lainnya pada keseluruhan piksel pada citra DoG. Kemudian hasil deteksi ekstremum dari *Taylor Expansion* ini akan dibandingkan dengan hasil deteksi ekstremum yang telah didapat sebelumnya. Jika suatu lokasi piksel dideteksi sebagai titik ekstremum pada kedua hasil citra, maka titik tersebut dianggap sebagai *keypoint* yang sebenarnya.

Setelah titik dengan nilai *low-contrast* telah dihilangkan, maka selanjutnya melakukan proses penghilangan titik yang berupa tepi. Pertama perlu dilakukan perhitungan matriks *Hessian* 2x2 dengan menggunakan persamaan (2-10).

$$H = \begin{bmatrix}Dxx & Dxy\\ Dxy & Dyy\end{bmatrix}$$

$$D_{xx} = \frac{D(x+1,y) - 2D(x,y) + D(x-1,y)}{1}$$

$$= \frac{0-4+1}{1} = -3$$

$$D_{yy} = \frac{D(x,y+1) - 2D(x,y) + D(x,y-1)}{1}$$

$$= \frac{1-4-0}{1} = -3$$

$$D_{xy} = \frac{D(x+1,y+1) - D(x+1,y-1) - D(x-1,y+1) + D(x-1,y-1)}{4}$$

$$= \frac{1-0-0+1}{4} = 0{,}5$$

Sehingga berdasarkan hasil nilai perhitungan tersebut, bentuk dari matriks *Hessian* menjadi sebagai berikut :

$$H = \begin{bmatrix} -3 & 0{,}5 \\ 0{,}5 & -3 \end{bmatrix}$$

Berdasarkan matriks *Hessian* yang telah didapatkan, maka akan didapatkan nilai *Trace* H dan determinan sebagai berikut:

$Tr(\boldsymbol{H}) = Dxx + Dyy = -3 - 3 = -6$

$Det(\boldsymbol{H}) = DxxDyy - (Dxy)^2 = -3.-3 - (0{,}5)^2$ = 9- 0,25 = 8,75

Nilai determinan dari titik tersebut bernilai tidak negatif, sehingga perhitungan akan dilanjutkan dengan membandingkan apakah rasio yang dimiliki titik tersebut berada di bawah ambang batas *threshold* rasio yang ditentukan yakni r =1,01. Perhitungannya sebagai berikut :

$$\frac{Tr(\boldsymbol{H})^2}{Det(\boldsymbol{H})} < \frac{(r+1)^2}{r}$$

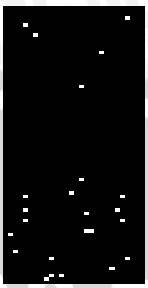
$$\frac{-6^2}{8{,}75} < \frac{(1{,}01+1)^2}{1{,}01}$$

$$\frac{36}{8{,}75} < \frac{4{,}0401}{1{,}01}$$

$4{,}114 \not< 4{,}0001$

Berdasarkan hasil pertidaksamaan tersebut, rasio dari titik ini lebih dari nilai *threshold* rasio yang telah ditentukan sehingga titik[2,1] merupakan *keypoint* yang sebenarnya. Proses penghilangan titik yang berupa tepi ini juga dilakukan pada setiap kandidat *keypoint* yang telah didapatkan setelah proses penghilangan titik *low-contrast* sebelumnya. Ilustrasi hasil dari proses eliminasi titik *keypoint*

yang memiliki nilai kontras rendah dan *keypoint* yang merupakan tepi dapat dilihat pada Gambar 3.51.



Gambar 3.51 Hasil Reduksi *Keypoint Low-contrast* Dan *Edge*

Sumber : Perancangan

Keypoint yang terpilih sudah memiliki ketahanan terhadap *noise* yang cukup dan stabil, proses selanjutnya adalah menentukan orientasi dari tiap *keypoint* yang telah didapatkan pada nilai *scale*, σ, terdekat, sehingga nantinya akan didapatkan sebuah *scale-invariant*. Untuk setiap sampel citra pada *scale* tertentu akan dicari *gradient magnitude* (persamaan (2-14)) dan orientasi nya (persamaan 2-15). Berikut adalah contoh perhitungan dari *keypoint* [2,1] pada σ = 1 :

Pada titik *keypoint* [12,4] akan diambil sebuah *region* yang berisi piksel-piksel *Laplacian* tetangga dari *keypoint* [12,4] dimana ukuran dari daerah tetangga tersebut sama dengan ukuran *kernel Gaussian* dengan nilai σ = 1,5 * 1 = 1,5 yang berarti bahwa ukurannya adalah 5x5 dari titik *keypoint*. Gambar 3.52 merupakan ilustrasi dari region di sekitar *keypoint* [12,4].

x\y	2	3	4	5	6
10	99	126	165	180	196
11	119	124	160	187	207
12	151	140	161	198	225
13	174	161	167	194	218
14	189	186	174	167	177

Gambar 3.52 Piksel *Orientation Collection Region Keypoint* [12,4]

Sumber : Perancangan

Kemudian dicari *gradient magnitude* dari piksel sekitar *keypoint*. Contoh perhitungan dari piksel [11,3] menggunakan persamaan (2-14).

$$m(x,y) = \sqrt{\big(L(x+1,y) - L(x-1,y)\big)^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$m(0,0) = \sqrt{(140-126)^2 + (160-119)^2} = \sqrt{1877} = 43{,}32$$

Nilai magnitude dari piksel [11,3] akan diset menjadi nilai 43,32. Kemudian dilanjutkan dengan perhitungan orientasi piksel[11,3] menggunakan persamaan (2-15) seperti berikut :

$$\theta(x,y) = \arctan\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right)$$

$$\theta(0,0) = \arctan\left(\frac{160 - 119}{140 - 126}\right) = \arctan(2{,}93) = 71{,}14$$

Sehingga orientasi dari piksel [11,3] mengarah ke sudut 71,14 derajat dan dimasukkan ke dalam bin kedelapan (70-79). Besaran *gradient magnitude* sebelum dimasukkan ke histogram akan dibobot terlebih dahulu dengan mengalikannya dengan *gaussian weighted function* (ukuran kernelnya sama dengan lebar region) menggunakan nilai σ= 1,5**scale keypoint*. Pada contoh ini nilai σ = 1,5*1 = 1,5. Berikut contoh perhitungan menggunakan persamaan :

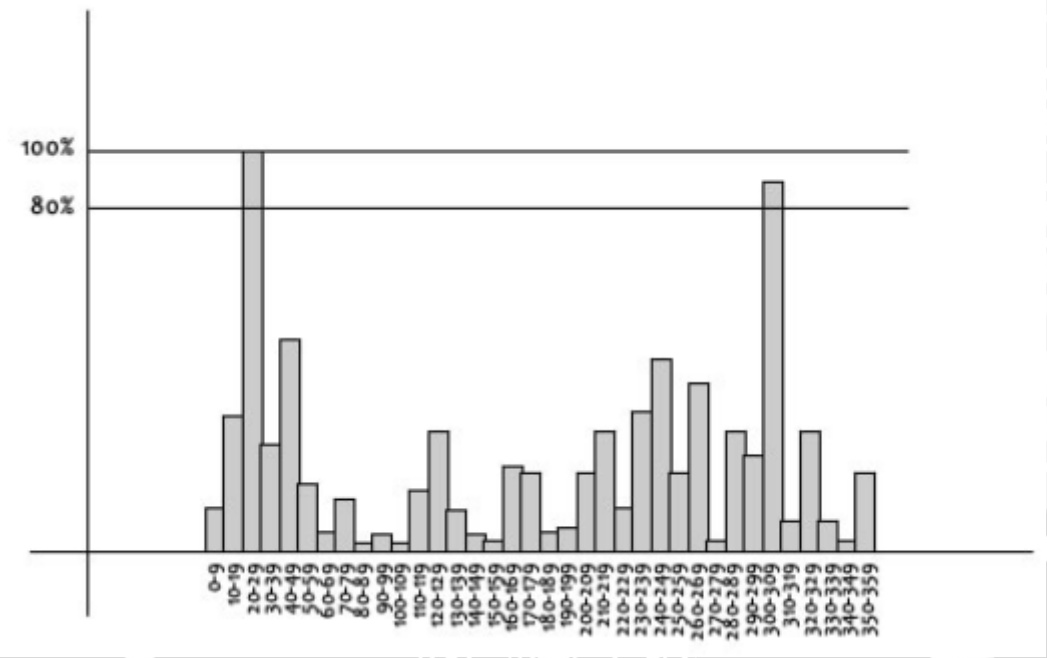
$$G(x,y,\sigma) = e^{-\frac{(x-x_f)^2 + (y-y_f)^2}{2.\sigma^2}}$$

$$G(-2,-2,1{,}5) = e^{-\frac{(-2-0)^2 + (-2-0)^2}{2.1{,}5^2}} = 0{,}00753$$

$$\text{Bobot} = 43{,}32 * 0{,}00753 = 0{,}326$$

Begitupun selanjutnya perhitungan *gradient magnitude* dan orientasi untuk semua piksel yang termasuk dalam *Orientation Collection Region Keypoint* [2,1].

Dari proses diatas, maka akan terbentuk sebuah histogram yang merepresentasikan nilai dari orientasi piksel-piksel yang terdapat di *region* sekitar *keypoint*[12,4] pada σ=1. Contoh bentuk histogramnya dapat dilihat pada Gambar 3.53.



Gambar 3.53 Contoh Ilustrasi *Histogram* Orientasi *Gradient*.

Sumber : [UTK-10]

Berdasarkan bentuk histogram diatas, nilai orientasi yang mencapai di atas 80% akan diorientasikan ke *keypoint*. Sehingga terdapat kemungkinan dimana *keypoint* yang memiliki nilai dan *scale* yang sama, namun memiliki orientasi yang berbeda. Namun jika tidak ada nilai orientasi yang mencapai 80%, maka nilai orientasi tertinggi yang akan diorientasikan pada *keypoint*. Gambar 3.54 merupakan contoh hasil orientasi yang ditandai sebagai tanda panah pada *keypoint* [12,4].

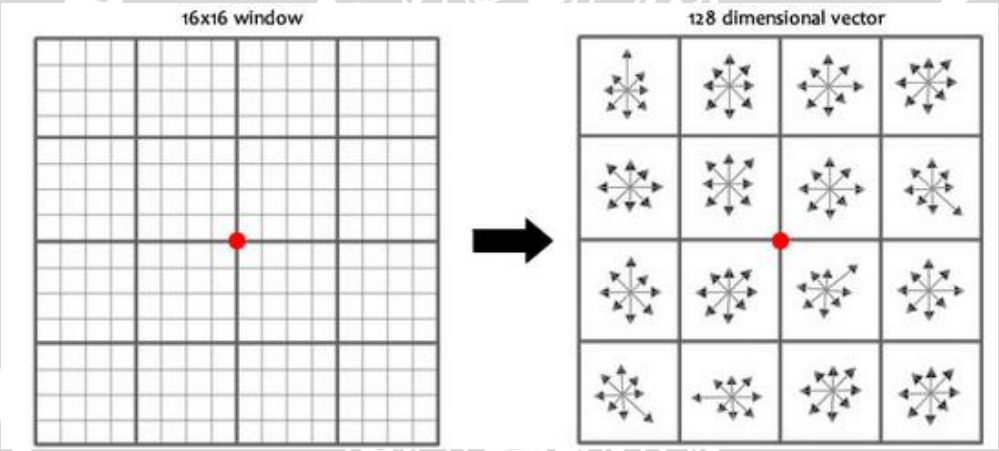
Gambar 3.54 Contoh Hasil *Orientasi Assignment Keypoint* [12,4]

Sumber : Perancangan

Setelah penentuan orientasi *keypoint* tiap *scale* telah dilakukan, maka telah terbentuk sebuah *feature keypoint* yang *invariant* terhadap *scale* dan rotasi. Kemudian akan dicari deskriptor dari citra lokal. Pencarian deskriptor dilakukan dengan mengambil titik piksel 16x16 di sekitar *keypoint*.

Kemudian dari representasi tersebut, akan dicari nilai *magnitude* dan orientasi tiap titik pada region tersebut. Proses perhitungannya sama dengan perhitungan *magnitude* dan orientasi pada tahap *keypoint orientation* sebelumnya. Namun yang berbeda disini, nilai orientasi pada tiap titik akan dimasukkan ke dalam 8 bin rentang sudut. Sehingga titik [11,3] dengan orientasi = 71,14 (pada penjelasan sebelumnya) masuk ke dalam bin kedua (45-89).

Dari perhitungan tersebut akan didapatkan deskriptor pada tiap region 4x4 pada matriks 16x16 yang diilustrasikan pada Gambar 3.55.



Gambar 3.55 Contoh Ilustrasi Hasil Pencarian Deskriptor

Gambar 3.55 merupakan ilustrasi hasil pembentukan deskriptor dimana titik *keypoint* sebagai pusat dari region tersebut. Deskriptor pada tiap-tiap *keypoint* terdiri dari 128 fitur. Untuk mengetahui apakah dua buah citra objek sama atau tidak akan dicari dengan proses *keypoint matching*. *Keypoint matching* dilakukan dengan menggunakan persamaan *Euclidien Distance*.

Contoh perhitungan jarak antara sebuah *keypoint* pada citra pertama dan *keypoint* pada citra kedua adalah sebagai berikut :

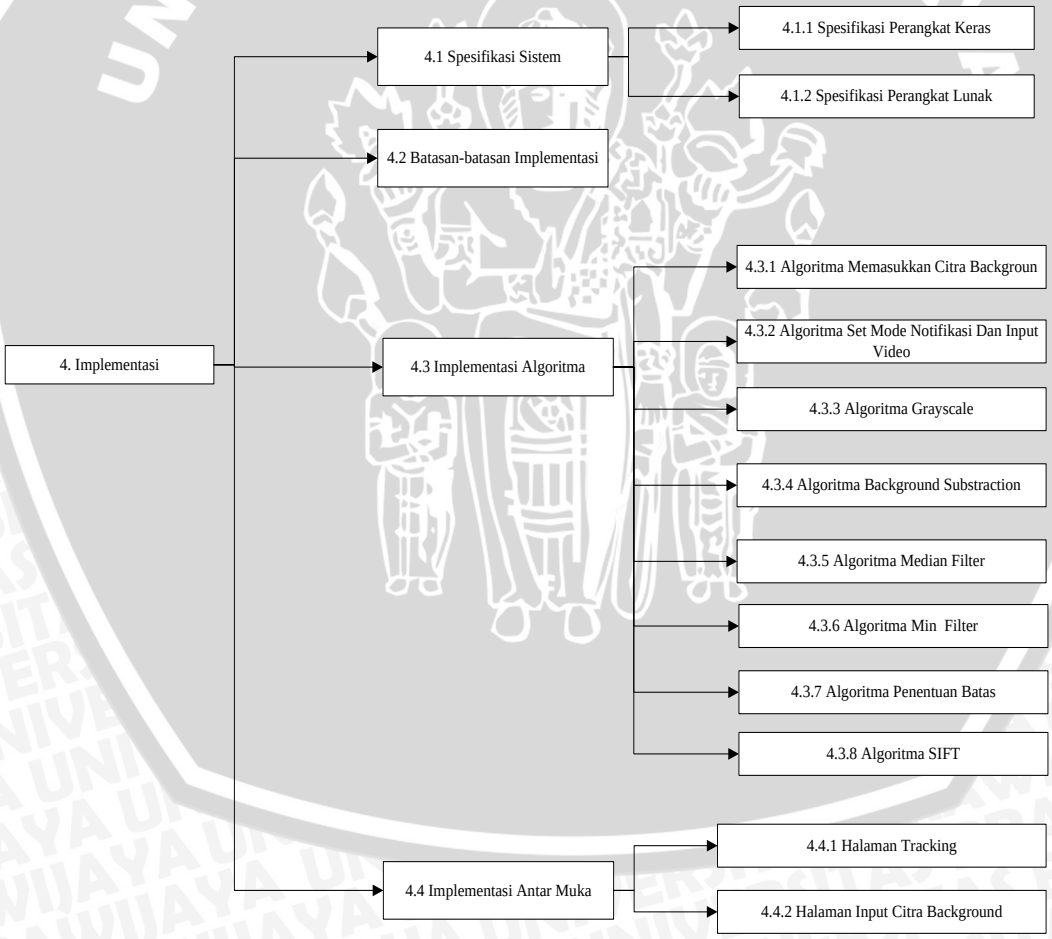
$$d(p,q) = \sqrt{\begin{matrix}(0.005-0.005)^2+\ (0.03-0.03)^2+(0.01-0.011)^2+(0.031-0.032)^2 \\ +(0.077-0.077)^2+(0.006-0.007)^2+(0.026-0.027)^2+(0.269-0.269)^2 \\ +(0.024-0.025)^2+(0.037-0.038)^2\ \ldots\ldots dst\end{matrix}}$$

d(p,q) = 0,2636

Karena nilai jarak 0,2636 < dari 0,3, maka dikatakan bahwa dua *keypoint* tersebut *match*. Perhitungan akan berlanjut sampai semua *keypoint* antara dua citra tersebut diikutsertakan. Jika jumlah *keypoint* yang *match* lebih besar dari 70% maka dikatakan kedua citra tersebut *matching*.

BAB IV
IMPLEMENTASI

Bab ini membahas mengenai implementasi perangkat lunak berdasarkan hasil yang telah didapatkan dari analisis kebutuhan dan proses perancangan perangkat lunak yang dibuat. Pembahasan terdiri dari penjelasan tentang spesifikasi sistem, batasan-batasan dalam implementasi, implementasi algoritma, dan implementasi antar muka. Tahap-tahap pembahasan implementasi yang dikerjakan digambarkan pada Gambar 4.1.



Gambar 4.1 Pohon Implementasi

Sumber : Implementasi

4.1 Spesifikasi Sistem

Hasil analisis kebutuhan yang telah diuraikan pada bab 3 menjadi acuan untuk melakukan implementasi menjadi sebuah sistem yang dapat berfungsi sesuai dengan kebutuhan. Spesifikasi sistem diimplementasikan pada spesifikasi perangkat keras dan perangkat lunak

4.1.1 Spesifikasi Perangkat Keras

Spesifikasi perangkat keras meliputi spesifikasi minimal dari prosesor, memori dan hardisk yang akan digunakan pada sistem ini dapat dilihat pada Tabel 4.1.

Tabel 4.1 Spesifikasi Perangkat Keras Komputer

Nama Komponen	Spesifikasi
Prosesor	Intel(R) Pentium Dual Core CPU T4400 @ 2.20GHz
Memori (RAM)	3 GB
Hardisk	WD kapasitas 320 GB

Sumber : Implementasi

4.1.2 Spesifikasi Perangkat Lunak

Pengembangan subsistem menggunakan perangkat lunak dengan spesifikasi yang dijelaskan pada Tabel 4.2.

Tabel 4.2 Spesifikasi Perangkat Lunak Komputer

Nama Komponen	Spesifikasi
Sistem operasi	Windows 7 32 bit.
Tools pemrograman	Visual Studio 2010 Express
Media Penyimpanan	File .txt

Sumber : Implementasi

4.2 Batasan-Batasan Implementasi

Batasan-batasan dalam mengimplementasikan sistem adalah sebagai berikut:

1. Sistem yang dibuat menerima masukan berupa citra video yang terekam dengan kamera *CCTV* di PTIIK berekstensi .wmv.

2. Sistem hanya mampu melakukan pelacakan terhadap objek berupa manusia dengan pergerakan normal (seperti berjalan dengan kecepatan normal) pada jarak maksimal 12 meter dari kamera.
3. Sistem akan memberikan konfirmasi berupa notif atau tanda berupa notifikasi sebagai respon lanjutan dari adanya objek yang telah terlacak pada video.
4. Pembahasan ditekankan pada bagaimana mengimplementasikan metode SIFT serta mengetahui tingkat kinerja metode tersebut apabila diterapkan pada *multiple object*.

4.3 Implementasi Algoritma

Aplikasi Implementasi Metode *Background Substraction* dan *Scale Invariant Feature Transform* Untuk *Multiple Object Tracking* Pada Video CCTV mempunyai beberapa proses utama yang terbagi dalam beberapa fungsi. Pada penulisan skripsi ini hanya dicantumkan algoritma dari beberapa proses saja sehingga tidak semua algoritma akan dicantumkan. Algoritma proses yang dicantumkan antara lain adalah proses memasukkan citra *background*, memasukkan citra uji dan *setting* notifikasi, deteksi objek, pelacakan objek dan menampilkan hasil output pelacakan dan notifikasi. Implementasi algoritma ini akan direpresentasikan dalam bentuk *code* dengan bahasa pemrograman C#.

4.3.1 Implementasi Algoritma Memasukkan Citra *Background*

Operasi pada algoritma memasukkan data untuk citra *background* ini bertujuan untuk menyediakan data berupa citra dua dimensi sebagai latar belakang ruang atau tempat dimana video merekam aktifitas keadaan tempat itu. Citra *background* ini didapat dari proses *snapshot* salah satu *frame* pada video. Citra *background* ini merupakan pencitraan dari ruang/tempat pada keadaan dasar dimana tidak terdapat objek bergerak di dalamnya, dengan kata lain keadaannya statis dan terdiri dari objek-objek yang posisinya tidak akan berubah (*absolute*). Citra *background* ini digunakan pada proses *background substraction* pada tahap *object detection* untuk metode SIFT. Sebelum citra *background* disimpan, citra

akan diproses terlebih dahulu agar siap digunakan saat proses *background substraction* nantinya.

Gambar 4.2 merupakan cuplikan program untuk algoritma memasukkan file yang mengimplementasikan perancangan algoritma memasukkan citra *background*.

```
1.  private void button1_Click(object sender, EventArgs e)
    {
2.    if (DialogResult.OK == openFileDialog1.ShowDialog())
      {
3.        dsVideoPlayer1.FileName = openFileDialog1.FileName;
      }
    }

4.  private      void      genericFilter1_ProcessData(object      Sender,
5.  Mitov.VideoLab.ProcessVideoNotifyArgs Args)
    {
       //kodingan untuk mengambil snapshot citra dari background
       // Capture Image.
6.     System.Drawing.Bitmap ABitmap = new Bitmap(320, 240);
7.     ABitmap = Args.InBuffer.ToBitmap();
8.     ++m_FrameNo1;
9.     pictureBox3.Image = grayScale1.ProcessBitmap(ABitmap);
    }

10. private void button2_Click(object sender, EventArgs e)
    {
11.    dsVideoPlayer1.Pause();
12.    pictureBox4.Image = grayScale1.ProcessBitmap(ABitmap);
    }

13. private void button3_Click(object sender, EventArgs e)
    {
14.     dsVideoPlayer1.Resume();
    }

15. private void button4_Click(object sender, EventArgs e)
    {
16.     dsVideoPlayer1.Stop();
    }
17. private void Form1_FormClosed(object sender, FormClosedEventArgs e)
    {
18.     dsVideoPlayer1.Stop();
    }

19. private void button5_Click(object sender, EventArgs e)
    {
20.   if (DialogResult.OK == saveFileDialog1.ShowDialog())
      {
21.         pictureBox4.Image.Save(saveFileDialog1.FileName.ToString());
22.         background= new Bitmap(saveFileDialog1.FileName.ToString());
      }
    }
```

Gambar 4.2 Implementasi Algoritma Memasukkan Citra Background

Sumber : Implementasi

Baris 4-9 berfungsi untuk menampilkan kotak dialog *openfile* untuk memilih video yang akan diputar dan menangkap *frame-frame* video di dalamnya dengan menggunakan *library videolab*. *Frame-frame* video yang telah ditangkap tersebut diproses untuk diubah menjadi citra *grayscale* dan ditampilkan di picturebox3 pada baris ke-9. Baris 10-12 berfungsi untuk melakukan *pause* pada video dan menampilkan *frame* video yang tertangkap saat proses *pause* pada picturebox4. Baris 13-14 berfungsi untuk melakukan *resume* terhadap video yang *pause*. Baris 15-16 berfungsi untuk menghentikan video. Baris 17-18 berfungsi untuk menghentikan video jika form ditutup. Baris untuk 19-22 berfungsi untuk menyimpan citra yang telah berhasil ditangkap sebagai citra *background*.

4.3.2 Implementasi Algoritma Mengatur Mode Notifikasi Dan Input Video

Operasi pada algoritma mengatur mode notifikasi bertujuan untuk mengatur mode notifikasi apakah diaktifkan atau dinonaktifkan dan untuk memilih video yang akan dilacak. Pada pemilihan mode notifikasi, apabila nilai mode diatur menjadi "*on*" maka pengguna juga perlu menentukan rentang waktu beroperasi dari notifikasi tersebut. Gambar 4.3 merupakan cuplikan *code* untuk untuk algoritma mengatur mode notifikasi dan input video.

```
1.  private void button6_Click(object sender, EventArgs e)
    {
2.    if (DialogResult.OK == openFileDialog2.ShowDialog())
      {
3.        dsVideoPlayer2.FileName = openFileDialog2.FileName;
      }
    }
```

Gambar 4.3 Implementasi Algoritma Mengatur Mode Notifikasi Dan Input Video

Sumber : Perancangan

Baris 1-3 berfungsi untuk membuka *openfiledialog* untuk memilih video yang akan diputar pada sistem *multiple object tracking*.

4.3.3 Implementasi Algoritma Pengubahan Warna Citra Ke *Grayscale*

Operasi pada algoritma pengubahan warna citra ke *grayscale* ini bertujuan untuk mengubah *channel* warna dari citra yang semula terdiri dari tiga *channel* (yakni *red*, *green*, dan *blue*) menjadi satu channel yakni *gray*. Dengan

pengubahan ini maka proses pengolahan citra akan menjadi mudah dan sederhana. Gambar 4.4 merupakan cuplikan *code* untuk algoritma pengubahan warna citra ke *grayscale*.

```
1.  Bitmap gambar_grayscale = grayScale2.ProcessBitmap(ABitmap2);
```

Gambar 4.4 Implementasi Algoritma Pengubahan Warna Citra Ke *Grayscale*

Sumber : Implementasi

Baris 1 berfungsi untuk melakukan pengubahan dari citra RGB ke citra *grayscale*. Proses perubahan citra ini dilakukan oleh *library videolab*.

4.3.4 Implementasi Algoritma *Background Substraction*

Operasi pada algoritma *background substraction* bertujuan untuk melakukan proses *image substraction* yakni proses pengurangan nilai piksel suatu citra oleh citra yang lain, dimana dalam hal ini antara citra pada *frame* video dengan citra *background* yang telah diinputkan sebelumnya, sehingga perubahan piksel pada citra *background* dapat terdeteksi secara signifikasn. Perubahan ini mengindikasikan bahwa terdapat objek bergerak di dalam *frame* video tersebut. Gambar 4.5 merupakan cupllikan *code* untuk algoritma *background substraction.*

```
1.  for (int i = 0; i < 320; i++)// x
    {
2.     for (int j = 0; j < 240; j++)// y
       {
3.       R = gambar_grayscale.GetPixel(i, j).R - background.GetPixel(i,
    j).R;
4.       if (R > 140 || R < -140)
         {
5.          R = ABitmap2.GetPixel(i, j).R;
6.          B = ABitmap2.GetPixel(i, j).B;
7.          G = ABitmap2.GetPixel(i, j).G;
8.          substraction.SetPixel(i, j, Color.FromArgb(R, G, B));
          }
9.        else
          {
10         substraction.SetPixel(i, j, Color.FromArgb(255, 255, 255));
          }
       }
    }
```

Gambar 4.5 Implementasi Algoritma *Background Substraction*

Sumber : Implementasi

Baris 3 berfungsi untuk mengambil nilai selisih dari citra *background* dan citra yang terdapat objek di dalamnya. Baris 4-10 berfungsi untuk melakukan

seleksi nilai selisih, jika bernilai lebih besar dari 140 atau lebih kecil dari -140 maka nilai selisih tersebut diterima dan piksel pada koordinat tersebut akan diset menjadi nilai RGB dari citra objek. Sebaliknya maka piksel pada koordinat tersebut akan diset menjadi warna putih.

4.3.5 Implementasi Algoritma *Median Filter*

Operasi pada algoritma *median filter* bertujuan untuk melakukan proses erosi untuk menghilangkan piksel-piksel *background* yang lolos dari proses *background substraction*. Sehingga didapatkan hanya nilai piksel objek yang telah berbentuk berupa *blob*. Gambar 4.6 merupakan cuplikan kode untuk algoritma *median filter*.

```
1.  Substraction1 =
    boxFilter2.ProcessBitmap(boxFilter1.ProcessBitmap(substraction));
```

Gambar 4.6 Implementasi Algoritma *Median Filter*

Sumber : Implementasi

Baris 1 berfungsi untuk melakukan proses median filter pada citra hasil *background substraction*. Proses median *filter* ini dilakukan oleh *library videolab*.

4.3.6 Implementasi Algoritma *Min Filter*

Operasi pada algoritma *min filter* bertujuan untuk melakukan proses dilasi guna memperkuat piksel sekitar daerah blob dari objek yang telah terdeteksi. Blob objek akan semakin lebar pada algoritma ini. Gambar 4.7 merupakan cuplikan kode dari implementasi dari perancangan algoritma *min filter*.

```
1.  Bitmap gambar_boundary = new
    Bitmap(boxFilter2.ProcessBitmap(gambar_objek));
```

Gambar 4.7 Implementasi Algoritma *Min Filter*

Sumber : Implementasi

Baris 1 berfungsi untuk melakukan proses min filter pada citra hasil median *filter*. Proses ini dilakukan oleh *library videolab*.

4.3.7 Implementasi Algoritma Penentuan Batas (*Boundary*)

Proses pada algoritma penentuan batas (*boundary*) ini bertujuan untuk menentukan batas dari tiap objek sehingga objek yang satu dapat dipisahkan dengan objek yang lain, terkecuali objek yang mengalami oklusi yang dalam hal ini objek akan dianggap sebagai satu objek. Gambar 4.8 merupakan cuplikan kode yang mengimplementasikan perancangan algoritma penentuan batas.

```
1.  private void boundary_2_anyar_awal(Bitmap gambar, Bitmap
    gambar_abu, out Bitmap[] gambar_objek_tunggal, Bitmap awal, float
    minx, float miny, float maxx, float maxy)
    {
2.      Bitmap gambar_objek = new Bitmap(gambar_abu);
3.      Bitmap objek_tunggal = new Bitmap(gambar_abu);
4.      gambar_objek_tunggal = new Bitmap[5];
5.      int boundary_awal_x, boundary_awal_y, indeks_arr_objtung=0;
6.      int minx_ = 100, miny_ = 100, maksx = 0, maksy = 0, n = 0, i =
    1, j = 1, maksimal_iterasi_i = gambar.Width-1, maksimal_iterasi_j =
    gambar.Height-1;
7.      int minx2 = 100, miny2 = 100, maksx2 = 0, maksy2 = 0, flag = 0,
    jumlah = 0;
8.      for (; i < maksimal_iterasi_i; )
        {
9.        for (; j < maksimal_iterasi_j; )
          {
10.         if (n < 180 && n > 30) // jika nilai blob dianggap sebagai
    objek tapi bukan berupa objek manusia
            {
11.             double atas, bawah;
12.             flag = 2; jumlah += n;
13.             if (maksx > maksx2) maksx2 = maksx;
14.             if (maksy > maksy2) maksy2 = maksy;
15.             if (minx_ < minx2) minx2 = minx_;
16.             if (miny_ < miny2) miny2 = miny_;
17.             atas = (Convert.ToDouble(miny) /
    Convert.ToDouble(gambar.Height)) * 100; //jarak blob dengan atas
18.             bawah = (Convert.ToDouble(gambar.Height) -
    Convert.ToDouble(maksy)) / Convert.ToDouble(gambar.Height) * 100;
    // jarak blob dengan bawah
19.             if (atas > bawah)
20.             {
21.               i = minx_; j = 1;
                  maksimal_iterasi_i = maksx;
                  maksimal_iterasi_j = miny_;
22.             }
23.             else
                {
24.               i = minx_; j = maksy + 1;
25.               maksimal_iterasi_i = maksx;
26.               maksimal_iterasi_j = gambar.Height;
                }
27.             if (jumlah >= 180)
                {
28.               n = jumlah;
29.               break;
                }
30.             boundary_cari_titik_awal(i, j, gambar, out
```

```
        boundary_awal_x, out boundary_awal_y, out maksx, out maksy, out
        miny_, out minx_, out n, maksimal_iterasi_i, maksimal_iterasi_j);
31.             if (n == 0) { i = maksx2 + 1; n += 200; }
            }
32.         else if (n > 180 && n > 30) //jika nilai blob dianggap
        sebagai objek manusia
            {
33.             if (flag == 2)
                {
34.                 minx_ = minx2;
35.                 miny_ = miny2;
36.                 maksx = maksx2;
37.                 maksy = maksy2;
38.                 flag = 0; minx2 = 1000; miny2 = 1000; maksx2 = 0;
        maksy2 = 0; jumlah = 0;
                }
                ////mengambil piksel single objek
39.             objek_tunggal = new Bitmap((maksx - minx_ + 1), (maksy -
        miny_ + 1));
40.             int o = 0, p = 0, RG;
41.             for (int k = minx_; k <= maksx; k++, o++)
                {
42.                 for (int l = miny_; l <= maksy; l++, p++)
                    {
43.                     RG = gambar_objek.GetPixel(k, l).R;
44.                     objek_tunggal.SetPixel(o, p, Color.FromArgb(RG, RG,
        RG));
                    }
45.                 p = 0;
                }
46.             gambar_objek_tunggal[indeks_arr_objtung] = new
        Bitmap(objek_tunggal);
47.             indeks_arr_objtung++; jumlah_ind_objtung++;

48.             i = maksx + 1; j = 1; maksimal_iterasi_i = gambar.Width-1;
        maksimal_iterasi_j = gambar.Height-1;
49.             boundary_cari_titik_awal(i, j, gambar, out
        boundary_awal_x, out boundary_awal_y, out maksx, out maksy, out
        miny_, out minx_, out n, maksimal_iterasi_i, maksimal_iterasi_j);
50.             if (n == 0)
                {
51.                 i = gambar.Width;
52.                 j = gambar.Height;
                }
            }
53.         else //jika blob dianggap bukan sebagai objek
            {
54.             if (flag == 2) { maksx = maksx2; flag = 0; }
55.             i = maksx + 1; maksimal_iterasi_i = gambar.Width-1;
56.     maksimal_iterasi_j = gambar.Height-1;
57.             boundary_cari_titik_awal(i, j, gambar, out
        boundary_awal_x, out boundary_awal_y, out maksx, out maksy, out
        miny_, out minx_, out n, maksimal_iterasi_i, maksimal_iterasi_j);
58.             if (n == 0)
                {
59.                 i = gambar.Width;
60.                 j = gambar.Height;
                }
            }
        }
    }
}

61.     private void boundary_cari_titik_awal(int c, int d, Bitmap gambar,
```

```
      out int boundary_awal_x, out int boundary_awal_y, out int maksx2,
      out int maksy2, out int miny2, out int minx2, out int n, int
      maks_iterasi_i, int maks_iterasi_j)
      {
62.      Bitmap gambar_objek = gambar;
63.      boundary_awal_x = 0; boundary_awal_y = 0;
64.      int k = 0, l = 0;
65.      int flag2 = 0, backtrack_x = 0, backtrack_y = 0;
         maksx2 = 0; maksy2 = 0; miny2 = gambar_objek.Height; minx2 =
      gambar_objek.Width; n = 0;
66.      for (int i = c; i < maks_iterasi_i; i++)
         {
67.         int current_x_1 = i, current_y_1 = 0;
68.         for (int j = d; j < maks_iterasi_j; j++)
            {
69.            if (gambar.GetPixel(i, j).R != 255)
               {
70.               if (flag2 == 0)
                  {
71.                  backtrack_x = current_x_1;
72.                  backtrack_y = current_y_1;
73.                  boundary_awal_x = i;
74.                  boundary_awal_y = j;
75.                  k = i;
76.                  l = j;
77.                  minx2 = boundary_awal_x;
78.                  miny2 = boundary_awal_y;
79.                  n += 2;
80.                  gambar_objek.SetPixel(boundary_awal_x,
      boundary_awal_y, Color.Blue);
81.                  flag2++;
82.                  if (gambar.GetPixel(k, l - 1).R != 255 && l - 1 >= 0)
      //tetangga atas
                     {
83.                     l -= 1;
84.                     gambar_objek.SetPixel(k, l, Color.Blue);
                     }
85.                  else if (gambar.GetPixel(k + 1, l - 1).R != 255 && l
      - 1 >= 0 && k + 1 <= gambar_objek.Width) //tetangga kanan atas
                     {
86.                     backtrack_x = k;
87.                     backtrack_y = l - 1;
88.                     l -= 1;
89.                     k += 1;
90.                     gambar.SetPixel(k, l, Color.Blue);
                     }
91.                  else if (gambar.GetPixel(k + 1, l).R != 255 && k + 1
      <= gambar_objek.Width) //tetangga kanan
                     {
92.                     backtrack_x = k + 1;
93.                     backtrack_y = l - 1;
94.                     k += 1;
95.                     gambar_objek.SetPixel(k, l, Color.Blue);
                     }
96.                  else if (gambar.GetPixel(k + 1, l + 1).R != 255 && k
      + 1 <= gambar_objek.Width && l + 1 <= gambar_objek.Height)
      //tetangga kanan bawah
                     {
97.                     backtrack_x = k + 1;
98.                     backtrack_y = l;
99.                     l+= 1;
100.                    k += 1;
101.                    gambar_objek.SetPixel(k, l, Color.Blue);
                     }
```

```
102.                else if (gambar.GetPixel(k, l + 1).R != 255 && l + 1
    <= gambar_objek.Height) //tetangga bawah
                    {
103.                  backtrack_x = k + 1;
104.                  backtrack_y = l + 1;
105.                  l += 1;
106.                  gambar_objek.SetPixel(k, l, Color.Blue);
                    }
107.                else if (gambar.GetPixel(k - 1, l + 1).R != 255 &&
    l + 1 <= gambar_objek.Height && k - 1 >= 0) //tetangga kiri bawah
                    {
108.                   backtrack_x = k;
109.                   backtrack_y = l + 1;
110.                   l += 1;
111.                   k -= 1;
112.                   gambar_objek.SetPixel(k, l, Color.Blue);
                    }
113.                else if (gambar.GetPixel(k - 1, l).R != 255 && k -
    1 >= 0) //tetangga kiri
                    {
114.                  backtrack_x = k - 1;
115.                  backtrack_y = l + 1;
116.                  k -= 1;
117.                  gambar_objek.SetPixel(k, l, Color.Blue);
                    }
118.                else if (gambar.GetPixel(k - 1, l - 1).R != 255 &&
    k - 1 >= 0 && l - 1 >= 0) //tetangga kiri atas
                    {
119.                  backtrack_x = k - 1;
120.                  backtrack_y = l;
121.                  l -= 1;
122.                  k -= 1;
123.                  gambar_objek.SetPixel(k, l, Color.Blue);
                    }
124.                else //jika tidak ditemukan tetangga
                    {
125.                     gambar_objek.SetPixel(boundary_awal_x,
    boundary_awal_y, Color.Yellow);
126.                   backtrack_x = 0;
127.                   backtrack_y = 0;
128.                   boundary_awal_x = 0;
129.                   boundary_awal_y = 0;
130.                   k = 0;
131.                   l = 0;
132.                   n = 0;
133.                   miny2 = 0;
134.                   flag2 = 0;
                    }
                      //pencarian titik maks x, min y, maks y
135.                  if (l < miny2) miny2 = l;
136.                  if (k > maksx2) maksx2 = k;
137.                  if (l > maksy2) maksy2 = l;
              }
138.            break;
          }
139.        else current_y_1 = j;
        }
140.    if (flag2 != 0) break;
      }
141.  if (n != 0)
      {
142.    titik_titik_boundary(gambar_objek, k, l, boundary_awal_x,
    boundary_awal_y, backtrack_x, backtrack_y, out n, out miny2, out
    minx2, out maksx2, out maksy2);
```

```
    }
}
143. private void titik_titik_boundary(Bitmap gambar_boundary, int k,
144. int l, int a, int b, int current_x, int current_y, out int n, out
145. int miny2, out int minx2, out int maksx2, out int maksy2)
{
    //menentukan boundary-boundary
146.    Bitmap gambar_1 = new Bitmap(gambar_boundary);
147.    maksx2 = 0; maksy2 = 0; miny2 = b; minx2 = a; n = 2;
148.    for (k = k; k < gambar_boundary.Width && (k != a || l != b); )
    {
149.       for (l = l; l < gambar_boundary.Height && (k != a || l != b);
       {
150.         int selisih = l - current_y;
         //cari backtrack
151.         switch (selisih)
         {
152.            case 1: //untuk backtrack ke atas
            {
153.              if (gambar_1.GetPixel(k, l - 1).R != 255 && l - 1 >=
0)
              {
154.                l -= 1;
155.                gambar_boundary.SetPixel(k, l, Color.Blue);
156.                n++;
              }
157.              else if (gambar_1.GetPixel(k + 1, l - 1).R != 255 && l
- 1 >= 0 && k + 1 <= gambar_boundary.Width)
              {
158.                current_x = k;
159.                current_y = l - 1;
160.                l -= 1;
161.                k += 1;
162.                gambar_boundary.SetPixel(k, l, Color.Blue);
163.                n++;
              }
164.              else if (gambar_1.GetPixel(k + 1, l).R != 255 && k + 1
<= gambar_boundary.Width)
              {
165.                current_x = k + 1;
166.                current_y = l - 1;
167.                k += 1;
168.                gambar_boundary.SetPixel(k, l, Color.Blue);
169.                n++;
              }
170.              else if (gambar_1.GetPixel(k + 1, l + 1).R != 255 && k
+ 1 <= gambar_boundary.Width && l + 1 <= gambar_boundary.Height)
              {
171.                current_x = k + 1;
172.                current_y = l;
173.                l += 1;
174.                k += 1;
175.                gambar_boundary.SetPixel(k, l, Color.Blue);
176.                n++;
              }
177.              else if (gambar_1.GetPixel(k, l + 1).R != 255 && l + 1
<= gambar_boundary.Height)
              {
178.                current_x = k + 1;
179.                current_y = l + 1;
180.                l += 1;
181.                gambar_boundary.SetPixel(k, l, Color.Blue);
182.                n++;
              }
```

```
183.            else if (gambar_1.GetPixel(k - 1, l + 1).R != 255 && l
    + 1 <= gambar_boundary.Height && k - 1 >= 0)
                {
184.              current_x = k;
185.              current_y = l + 1;
186.              l += 1;
187.              k -= 1;
188.              gambar_boundary.SetPixel(k, l, Color.Blue);
189.              n++;
                }
190.              else if (gambar_1.GetPixel(k - 1, l).R != 255 && k - 1
    >= 0)
                {
191.              current_x = k - 1;
192.              current_y = l + 1;
193.              k -= 1;
194.              gambar_boundary.SetPixel(k, l, Color.Blue);
195.              n++;
                }
196.              else if (gambar_1.GetPixel(k - 1, l - 1).R != 255 && k
    - 1 >= 0 && l - 1 >= 0)
                {
197.              current_x = k - 1;
198.              current_y = l;
199.              l -= 1;
200.              k -= 1;
201.              gambar_boundary.SetPixel(k, l, Color.Blue);
202.              n++;
                }
            }
203.          break;
204.          case -1: //untuk backtrack ke bawah
              {
205.            if (gambar_1.GetPixel(k, l + 1).R != 255 && l + 1 <=
    gambar_boundary.Height)
                {
206.              l += 1;
207.              gambar_boundary.SetPixel(k, l, Color.Blue);
208.              n++;
                }
209.              else if (gambar_1.GetPixel(k - 1, l + 1).R != 255 && l
    + 1 <= gambar_boundary.Height && k - 1 >= 0)
                {
210.              current_x = k;
211.              current_y = l + 1;
212.              l += 1;
213.              k -= 1;
214.              gambar_boundary.SetPixel(k, l, Color.Blue);
215.              n++;
                }
216.              else if (gambar_1.GetPixel(k - 1, l).R != 255 && k - 1
    >= 0)
                {
217.              current_x = k - 1;
218.              current_y = l + 1;
219.              k -= 1;
220.              gambar_boundary.SetPixel(k, l, Color.Blue);
221.              n++;
                }
222.              else if (gambar_1.GetPixel(k - 1, l - 1).R != 255 && k
    - 1 >= 0 && l - 1 >= 0)
                {
223.              current_x = k - 1;
224.              current_y = l;
```

```
225.                 l -= 1;
226.                 k -= 1;
227.               gambar_boundary.SetPixel(k, l, Color.Blue);
228.               n++;
                 }
229.               else if (gambar_1.GetPixel(k, l - 1).R != 255 && l - 1
    >= 0)
                 {
230.               current_x = k - 1;
231.               current_y = l - 1;
232.               l -= 1;
233.               gambar_boundary.SetPixel(k, l, Color.Blue);
234.               n++;
                 }
235.               else if (gambar_1.GetPixel(k + 1, l - 1).R != 255 && l -
    1 >= 0 && k + 1 <= gambar_boundary.Width)
                 {
236.               current_x = k;
237.               current_y = l - 1;
238.               l -= 1;
239.               k += 1;
240.               gambar_boundary.SetPixel(k, l, Color.Blue);
241.               n++;
                 }
                 else if (gambar_1.GetPixel(k + 1, l).R != 255 && k + 1
    <= gambar_boundary.Width)
                 {
242.               current_x = k + 1;
243.               current_y = l - 1;
244.               k += 1;
245.               gambar_boundary.SetPixel(k, l, Color.Blue);
246.               n++;
                 }
247.               else if (gambar_1.GetPixel(k + 1, l + 1).R != 255 && k +
    1 <= gambar_boundary.Width && l + 1 <= gambar_boundary.Height)
                 {
248.               current_x = k + 1;
249.               current_y = l;
250.               l += 1;
251.               k += 1;
252.               gambar_boundary.SetPixel(k, l, Color.Blue);
253.               n++;
                 }
               }
254.           break;
255.           default:
               {
256.             int selisih2 = k - current_x;
257.             switch (selisih2)
                  {
258.                case 1: //untuk backtrack ke kiri
                    {
259.                   if (gambar_1.GetPixel(k - 1, l).R != 255 && k - 1
    >= 0)
                      {
260.                     k -= 1;
261.                     gambar_boundary.SetPixel(k, l, Color.Blue);
262.                     n++;
                      }
263.                   else if (gambar_1.GetPixel(k - 1, l - 1).R != 255
    && k - 1 >= 0 && l - 1 >= 0)
                      {
264.                    current_x = k - 1;
265.                    current_y = l;
```

```
266.                    l -= 1;
267.                    k -= 1;

268.  gambar_boundary.SetPixel(k, l, Color.Blue);
269.                   n++;
                      }
270.                  else if (gambar_1.GetPixel(k, l - 1).R != 255 && l
      - 1 >= 0)
                      {
271.                    current_x = k - 1;
272.                    current_y = l - 1;
273.                    l -= 1;
274.                    gambar_boundary.SetPixel(k, l, Color.Blue);
275.                    n++;
                      }
276.                  else if (gambar_1.GetPixel(k + 1, l - 1).R != 255
      && l - 1 >= 0 && k + 1 <= gambar_boundary.Width)
                      {
277.                    current_x = k;
278.                    current_y = l - 1;
279.                    l -= 1;
280.                    k += 1;
281.                    gambar_boundary.SetPixel(k, l, Color.Blue);
282.                    n++;
                      }
283.                  else if (gambar_1.GetPixel(k + 1, l).R != 255 && k
      + 1 <= gambar_boundary.Width)
                      {
284.                    current_x = k + 1;
285.                    current_y = l - 1;
286.                    k += 1;
287.                    gambar_boundary.SetPixel(k, l, Color.Blue);
289.                    n++;
                      }
290.                  else if (gambar_1.GetPixel(k + 1, l + 1).R != 255
      && k + 1 <= gambar_boundary.Width && l + 1 <=
      gambar_boundary.Height)
                      {
291.                    current_x = k + 1;
292.                    current_y = l;
293.                    l += 1;
294.                    k += 1;
295.                    gambar_boundary.SetPixel(k, l, Color.Blue);
296.                    n++;
                      }
297.                  else if (gambar_1.GetPixel(k, l + 1).R != 255 && l
      + 1 <= gambar_boundary.Height)
                      {
298.                    current_x = k + 1;
299.                    current_y = l + 1;
300.                    l += 1;
301.                    gambar_boundary.SetPixel(k, l, Color.Blue);
302.                    n++;
                      }
303.                  else if (gambar_1.GetPixel(k - 1, l + 1).R != 255
      && l + 1 <= gambar_boundary.Height && k - 1 >= 0)
                      {
304.                    current_x = k;
305.                    current_y = l + 1;
306.                    l += 1;
307.                    k -= 1;
308.                    gambar_boundary.SetPixel(k, l, Color.Blue);
309.                    n++;
                      }
```

```
                }
310.            break;
311.        case -1: //untuk backtrak ke kanan
            {
312.            if (gambar_1.GetPixel(k + 1, l).R != 255 && k + 1 <=
gambar_boundary.Width)
                {
313.                k += 1;
314.                gambar_boundary.SetPixel(k, l, Color.Blue);
315.                n++;
                }
316.            else if (gambar_1.GetPixel(k + 1, l + 1).R != 255 && k
+ 1 <= gambar_boundary.Width && l + 1 < gambar_boundary.Height)
                {
317.                current_x = k + 1;
318.                current_y = l;
319.                l += 1;
320.                k += 1;
321.                gambar_boundary.SetPixel(k, l, Color.Blue);
322.                n++;
                }
323.            else if (gambar_1.GetPixel(k, l + 1).R != 255 && l + 1
< gambar_boundary.Height)
                {
324.                current_x = k + 1;
325.                current_y = l + 1;
326.                l += 1;
327.                gambar_boundary.SetPixel(k, l, Color.Blue);
328.                n++;
                }
329.            else if (gambar_1.GetPixel(k - 1, l + 1).R != 255 && l
+ 1 < gambar_boundary.Height && k - 1 >= 0)
                {
330.                current_x = k;
331.                current_y = l + 1;
332.                l += 1;
333.                k -= 1;
334.                gambar_boundary.SetPixel(k, l, Color.Blue);
335.                n++;
                }
336.            else if (gambar_1.GetPixel(k - 1, l).R != 255 && k - 1
>= 0)
                {
337.                current_x = k - 1;
338.                current_y = l + 1;
339.                k -= 1;
340.                gambar_boundary.SetPixel(k, l, Color.Blue);
341.                n++;
                }
342.            else if (gambar_1.GetPixel(k - 1, l - 1).R != 255 && k
- 1 >= 0 && l - 1 >= 0)
                {
343.                current_x = k - 1;
344.                current_y = l;
345.                l -= 1;
346.                k -= 1;
347.                gambar_boundary.SetPixel(k, l, Color.Blue);
348.                n++;
                }
349.            else if (gambar_1.GetPixel(k, l - 1).R != 255 && l - 1
>= 0)
                {
350.                current_x = k - 1;
351.                current_y = l - 1;
```

```
352.                 l -= 1;
353.                 gambar_boundary.SetPixel(k, l, Color.Blue);
354.                 n++;
                   }
355.               else if (gambar_1.GetPixel(k + 1, l - 1).R != 255 && l
    - 1 >= 0 && k + 1 <= gambar_boundary.Width)
                   {
356.                 current_x = k;
357.                 current_y = l - 1;
358.                 l -= 1;
359.                 k += 1;
360.                 gambar_boundary.SetPixel(k, l, Color.Blue);
361.                 n++;
                   }
                 }
362.             break;
363.             default:
                 {
364.               k = gambar_boundary.Width;
365.               l = gambar_boundary.Height;
                 }
366.             break;
               }
             }
367.       break;
           }
           //pencarian titik maks x, min y, maks y
368.       if (l < miny2) miny2 = l;
369.       if (k > maksx2) maksx2 = k;
370.       if (l > maksy2) maksy2 = l;
     }
   }
 }
```

Gambar 4.8 Implementasi Algoritma Penentuan Batas (*Boundary*)

Sumber : Implementasi

Baris 10-31 berfungsi untuk mendeteksi bahwa luas *boundary* yang telah terdeteksi merupakan sebuah blob objek namun bukan berupa objek manusia. Sehingga akan dilakukan iterasi pencarian blob selanjutnya. Baris 32-52 berfungsi untuk mendeteksi bahwa luas *boundary* objek yang telah terdeteksi dianggap sebagai objek manusia. Selanjutnya daerah piksel objek tersebut akan diambil berdasarkan nilai min_x, min_y, maks_x dan maks_y dari piksel-piksel *boundary* pada baris 33-47. Baris 53-60 berfungsi untuk mendeteksi bahwa luas *boundary* masih belum dianggap sebagai objek dan objek manusia sehingga akan dilakukan iterasi lagi dengan mengabaikan deteksi sebelumnya.

Baris 69-80 berfungsi untuk mencari titik awal pertama dari *boundary* objek. Selain itu juga mencari titik kedua *boundary* setelah ditemukannya titik awal *boundary* pada baris 82-138. Baris 152-203 berfungsi untuk melakukan pencarian titik *boundary* dengan *backtrack* dari atas piksel *boundary*. Baris 204-

254 berfungsi untuk melakukan pencarian titik *boundary* dengan *backtrack* dari bawah piksel *boundary*. Baris 258-310 berfungsi untuk melakukan pencarian titik *boundary* dengan *backtrack* dari kiri piksel *boundary*. 311-162 berfungsi untuk melakukan pencarian titik *boundary* dengan *backtrack* dari kanan piksel *boundary*.

4.3.8 Implementasi Algoritma SIFT (*Scale Invariant Feature Transform*)

Operasi pada algoritma SIFT bertujuan untuk mengekstraksi fitur pada tiap citra objek yang telah terdeteksi dan melakukan pelacakan objek berdasarkan kecocokan objek tersebut pada fitur-fitur objek sebelumnya yang telah tersimpan. Algoritma ini terdiri dari beberapa sub-proses seperti berikut :

4.3.8.1 Pembentukan *Scale Space*

Proses pembentukan *scale space* ini bertujuan untuk mendapatkan rangkaian tingkatan citra yang terdiri dari tiga oktaf ukuran dengan lima level buram yang berbeda-beda pada tiap oktafnya. *Scale space* disini hampir sama dengan pembentukan *Gaussian Pyramid*. Algoritma pembentukan *scale space* ini dapat dilihat pada Gambar 4.9.

```
1.  private void generate_operator_gauss(Bitmap gambar_objek, out
    Bitmap[,] gaussian_pyramid, out double[,] gaussian_scale)
    {
2.     gaussian_pyramid = new Bitmap[3,5];
3.     gaussian_scale = new double[3,5];
4.     Bitmap hasil_konvolusi;
5.     double[,] operator_gauss = new double[3, 3];
6.     double scale = 0.707107, scale_slnjutnya = 0, k = Math.Sqrt(2),
    sum_kernel = 0;

       //menghitung nilai ke-15 scale gaussian
7.     for (int i = 0; i < 3; i++)
       {
8.       scale_slnjutnya = scale;
9.       for (int j = 0; j < 5; j++)
         {
10.        gaussian_scale[i, j] = scale_slnjutnya;
           //menghitung operator 3x3
11.        for (int m = 0; m < 3; m++)
           {
12.           for (int n = 0; n < 3; n++)
              {
13.              double nilai = (1 / (2 * Math.PI * Math.Pow(scale,
    2))) * Math.Exp(-1 * ((Math.Pow(m - 1, 2) + Math.Pow(n - 1, 2)) /
    (2 * Math.Pow(scale_slnjutnya, 2))));
14.              operator_gauss[m, n] = nilai;
15.              sum_kernel += nilai;
              }
           }
```

```
        //normalisasi operator dengan sum
16.     for (int m = 0; m < 3; m++)
        {
17.       for (int n = 0; n < 3; n++)
          {
18.         operator_gauss[m, n] /= sum_kernel;
          }
        }
19.     sum_kernel = 0;
20.     konvolusi(operator_gauss, i, j, gambar_objek, out
hasil_konvolusi);
21.     gaussian_pyramid[i, j] = hasil_konvolusi;
22.     scale_slnjutnya *= k;
      }
23.   scale *= 2;
24.   double ukuran_persen = 0.75;
25.   this.resize3.Height = Convert.ToUInt32(ukuran_persen *
gambar_objek.Height);
26.   this.resize3.Width = Convert.ToUInt32(ukuran_persen *
gambar_objek.Width);
27.   Bitmap gambar_resize = new
28. Bitmap(resize3.ProcessBitmap(gambar_objek));
29.   gambar_objek = gambar_resize;
    }
  }


30. private void konvolusi(double[,] operator_gauss, int a, int b,
31. Bitmap gambar_objek, out Bitmap hasil_konvolusi)
  {
32.   hasil_konvolusi = new Bitmap(gambar_objek);
33.   for (int i = 1; i < gambar_objek.Width - 1; i++)
    {
34.     for (int j = 1; j < gambar_objek.Height - 1; j++)
      {
35.       double sum = 0;
36.       double nilai_1 = gambar_objek.GetPixel(i - 1, j - 1).R,
nilai_2 = gambar_objek.GetPixel(i, j - 1).R, nilai_3 =
gambar_objek.GetPixel(i + 1, j - 1).R;
37.       double nilai_4 = gambar_objek.GetPixel(i - 1, j).R, nilai_5
= gambar_objek.GetPixel(i, j).R, nilai_6 = gambar_objek.GetPixel(i
+ 1, j).R;
38.       double nilai_7 = gambar_objek.GetPixel(i - 1, j + 1).R,
nilai_8 = gambar_objek.GetPixel(i, j + 1).R, nilai_9 =
gambar_objek.GetPixel(i + 1, j + 1).R;
39.       double[,] nilai = new double[3, 3] { { nilai_1, nilai_2,
nilai_3 }, { nilai_4, nilai_5, nilai_6 }, { nilai_7, nilai_8,
nilai_9 } };
40.       for (int o = 0; o < 3; o++)
          {
41.         for (int p = 0; p < 3; p++)
            {
42.           double kali = nilai[o, p] * operator_gauss[o, p];
43.           sum += kali;
            }
          }
44.       if (sum > 255)
          {
45.         sum = 255;
          }
46.       int sum_int = Convert.ToInt32(sum);
47.       hasil_konvolusi.SetPixel(i, j, Color.FromArgb(sum_int,
sum_int, sum_int));
```

```
        }
    }
}
```

Gambar 4.9 Implementasi Sub-Algoritma SIFT Pembentukan *Scale Space*

Sumber : Implementasi

Baris 7-19 berfungsi untuk melakukan *generate* 15 *scale gaussian* beserta operator (kernel) 3x3 masing-masing *scale*. Dimana baris tersebut juga memanggil fungsi konvolusi untuk langsung melakukan proses konvolusi pada baris 30-47. Pada baris 24-29 dilakukan proses *resize* untuk membentuk level oktaf pada *gaussian pyramid* selanjutnya.

4.3.8.2 Pembentukan DOG (*Difference Of Gaussian*)

Proses pembentukan DOG ini bertujuan untuk mendapatkan selisih nilai piksel dari dua citra *scale space* yang berdekatan. Proses pembentukan DOG dilakukan dengan menggunakan proses *image substraction* sederhana. Gambar 4.10 merupakan cuplikan kode yang mengimplementasikan algoritma pembentukan DOG.

```
1.  private void pembentukan_dog(Bitmap[,] gaussian_pyramid, out
    Bitmap[,] dog)
    {
2.    Bitmap hasil_substract;
3.    dog = new Bitmap[3,4];
4.    for (int i = 0; i < 3;i++)
      {
5.      for (int j = 0; j < 4;j++)
        {
6.        image_substraction(gaussian_pyramid[i,                    j],
    gaussian_pyramid[i, j + 1], out hasil_substract);
7.        dog[i, j] = hasil_substract;
        }
      }
    }

8.  private void image_substraction(Bitmap gambar, Bitmap gambar2, out
    Bitmap hasil_substract)
    {
9.    hasil_substract = new Bitmap(gambar);
10.   for (int i = 0; i < gambar.Width; i++)
      {
11.     for (int j = 0; j < gambar.Height; j++)
        {
12.       int selisih = 0;
13.       selisih = gambar.GetPixel(i, j).R - gambar2.GetPixel(i, j).R;
14.       if (selisih < 0)
          {
15.         selisih *= -1;
          }
16.       hasil_substract.SetPixel(i,     j,     Color.FromArgb(selisih,
    selisih, selisih));
```

```
        }
      }
    }
```

Gambar 4.10 Implementasi Sub-Algoritma SIFT Pembentukan DOG

Sumber : Implementasi

Baris 4-7 berfungsi untuk mengambil dua citra yang berdekatan dalam satu oktaf pada *gaussian pyramid*. Pada baris 10-16 dilakukan proses *image substraction* antara dua citra hasil konvolusi yang berdekatan dalam satu level oktaf.

4.3.8.3 Pencarian *Keypoint*

Proses pencarian *keypoint* bertujuan untuk mencari kandidat *keypoint* dari rangkaian DOG pada *Gaussian pyramid* yang telah terbentuk. *Keypoint* terpilih jika memiliki nilai minimum atau maksimum dari 26 tetangganya. Gambar 4.11 merupakan cuplikan kode dari proses pencarian *keypoint*.

```
1.  private void ekstremum(Bitmap[,] dog, out Bitmap[,] ekstremum)
    {
2.    Bitmap hasil_ekstremum;
3.    ekstremum = new Bitmap[3,2];
      //mengecek apakah maksimum atau tidak dari tetangga-tetangganya
4.    for(int i=0;i<3;i++)
      {
5.      for(int j=1;j<=2;j++)
        {
6.        Bitmap gambar1 = new Bitmap(dog[i,j-1]), gambar2 = new
    Bitmap(dog[i,j]), gambar3 = new Bitmap(dog[i,j+1]);
7.        hasil_ekstremum = new Bitmap(gambar2);
8.        for(int a=1;a<gambar2.Width;a++)
          {
9.          for(int b=1;b<gambar2.Height;b++)
            {
               //cek maksimum atau tidak
10.            if ((gambar2.GetPixel(a, b).R > gambar1.GetPixel(a - 1, b
    - 1).R) && (gambar2.GetPixel(a, b).R > gambar1.GetPixel(a, b - 1).R)
    && (gambar2.GetPixel(a, b).R > gambar1.GetPixel(a + 1, b - 1).R) &&
    (gambar2.GetPixel(a, b).R > gambar1.GetPixel(a - 1, b).R) &&
    (gambar2.GetPixel(a, b).R > gambar1.GetPixel(a, b).R) &&
    (gambar2.GetPixel(a, b).R > gambar1.GetPixel(a + 1, b).R) &&
    (gambar2.GetPixel(a, b).R > gambar1.GetPixel(a - 1, b + 1).R) &&
    (gambar2.GetPixel(a, b).R > gambar1.GetPixel(a, b + 1).R) &&
    (gambar2.GetPixel(a, b).R > gambar1.GetPixel(a + 1, b + 1).R) &&
    (gambar2.GetPixel(a, b).R > gambar2.GetPixel(a - 1, b - 1).R) &&
    (gambar2.GetPixel(a, b).R > gambar2.GetPixel(a, b - 1).R) &&
    (gambar2.GetPixel(a, b).R > gambar2.GetPixel(a + 1, b - 1).R) &&
    (gambar2.GetPixel(a, b).R > gambar2.GetPixel(a - 1, b).R) &&
    (gambar2.GetPixel(a, b).R > gambar2.GetPixel(a + 1, b).R) &&
    (gambar2.GetPixel(a, b).R > gambar2.GetPixel(a - 1, b + 1).R) &&
    (gambar2.GetPixel(a, b).R > gambar2.GetPixel(a, b + 1).R) &&
    (gambar2.GetPixel(a, b).R > gambar2.GetPixel(a + 1, b + 1).R) &&
    (gambar2.GetPixel(a, b).R > gambar3.GetPixel(a - 1, b - 1).R) &&
    (gambar2.GetPixel(a, b).R > gambar3.GetPixel(a, b - 1).R) &&
    (gambar2.GetPixel(a, b).R > gambar3.GetPixel(a + 1, b - 1).R) &&
```

```
      (gambar2.GetPixel(a,  b).R  >  gambar3.GetPixel(a  -  1,  b).R)  &&
      (gambar2.GetPixel(a,   b).R   >   gambar3.GetPixel(a,   b).R)   &&
      (gambar2.GetPixel(a,  b).R  >  gambar3.GetPixel(a  +  1,  b).R)  &&
      (gambar2.GetPixel(a, b).R > gambar3.GetPixel(a - 1, b + 1).R) &&
      (gambar2.GetPixel(a,  b).R  >  gambar3.GetPixel(a,  b  +  1).R)  &&
      (gambar2.GetPixel(a, b).R > gambar3.GetPixel(a + 1, b + 1).R))
                {
11.               hasil_ekstremum.SetPixel(a, b, Color.FromArgb(255, 255,
      255));
                }
12.             else if ((gambar2.GetPixel(a, b).R < gambar1.GetPixel(a -
      1, b - 1).R) && (gambar2.GetPixel(a, b).R < gambar1.GetPixel(a, b -
      1).R) && (gambar2.GetPixel(a, b).R < gambar1.GetPixel(a + 1, b -
      1).R) && (gambar2.GetPixel(a, b).R < gambar1.GetPixel(a - 1, b).R)
      &&  (gambar2.GetPixel(a,  b).R  <  gambar1.GetPixel(a,  b).R)  &&
      (gambar2.GetPixel(a,  b).R  <  gambar1.GetPixel(a  +  1,  b).R)  &&
      (gambar2.GetPixel(a, b).R < gambar1.GetPixel(a - 1, b + 1).R) &&
      (gambar2.GetPixel(a,  b).R  <  gambar1.GetPixel(a,  b  +  1).R)  &&
      (gambar2.GetPixel(a, b).R < gambar1.GetPixel(a + 1, b + 1).R) &&
      (gambar2.GetPixel(a, b).R < gambar2.GetPixel(a - 1, b - 1).R) &&
      (gambar2.GetPixel(a,  b).R  <  gambar2.GetPixel(a,  b  -  1).R)  &&
      (gambar2.GetPixel(a, b).R < gambar2.GetPixel(a + 1, b - 1).R) &&
      (gambar2.GetPixel(a,  b).R  <  gambar2.GetPixel(a  -  1,  b).R)  &&
      (gambar2.GetPixel(a,  b).R  <  gambar2.GetPixel(a  +  1,  b).R)  &&
      (gambar2.GetPixel(a, b).R < gambar2.GetPixel(a - 1, b + 1).R) &&
      (gambar2.GetPixel(a,  b).R  <  gambar2.GetPixel(a,  b  +  1).R)  &&
      (gambar2.GetPixel(a, b).R < gambar2.GetPixel(a + 1, b + 1).R) &&
      (gambar2.GetPixel(a, b).R < gambar3.GetPixel(a - 1, b - 1).R) &&
      (gambar2.GetPixel(a,  b).R  <  gambar3.GetPixel(a,  b  -  1).R)  &&
      (gambar2.GetPixel(a, b).R < gambar3.GetPixel(a + 1, b - 1).R) &&
      (gambar2.GetPixel(a,  b).R  <  gambar3.GetPixel(a  -  1,  b).R)  &&
      (gambar2.GetPixel(a,   b).R   <   gambar3.GetPixel(a,   b).R)   &&
      (gambar2.GetPixel(a,  b).R  <  gambar3.GetPixel(a  +  1,  b).R)  &&
      (gambar2.GetPixel(a, b).R < gambar3.GetPixel(a - 1, b + 1).R) &&
      (gambar2.GetPixel(a,  b).R  <  gambar3.GetPixel(a,  b  +  1).R)  &&
      (gambar2.GetPixel(a, b).R < gambar3.GetPixel(a + 1, b + 1).R))
                {
13.               hasil_ekstremum.SetPixel(a, b, Color.FromArgb(255, 255,
      255));
                }
14.             else hasil_ekstremum.SetPixel(a, b, Color.FromArgb(0, 0,
      0));
              }
            }
15.         ekstremum[i, j - 1] = hasil_ekstremum;
          }
        }
16.   }
```

Gambar 4.11 Implementasi Sub-Algoritma SIFT Pencarian *Keypoint*

Sumber : Implementasi

Baris 6 berfungsi untuk mengambil tiga citra *difference of gaussian* untuk perbandingan piksel saat mencari titik ekstremum. Baris 10 berfungsi untuk mengetahui apakah titik piksel tersebut merupakan titik maksimum atau tidak. Baris 12 berfungsi untuk mengetahui apakah titik piksel tersebut merupakan titik minimum atau tidak.

4.3.8.4 Eliminasi *Keypoint Low Contrast* dan Tepi

Proses eliminasi *keypoint low contrast* dan yang berupa tepi bertujuan untuk menyeleksi dan membuang *keypoint* yang tidak stabil sehingga akan didapatkan *keypoint* yang tahan dan stabil pada lingkungan dengan perubahan pencahayaan yang tidak terlalu ekstrim. Gambar 4.12 merupakan cuplikan kode dari proses eliminasi *keypoint low contrast* dan tepi.

```
1.  private void eliminasi_low_contrast(Bitmap[,] dog, out Bitmap[,]
    arr_low_contrast)
    {
2.    arr_low_contrast = new Bitmap[3,2];
3.    double threshold_z = 0.5;
4.    double[] turunan_pertama = new double[3];
5.    double[,] turunan_kedua = new double[3, 3], array_dz = new
    double[3000, 3];
6.    double max = 0, min = 100;
7.    int count = 0, index_array_z = 0;
8.    Bitmap deteksi_keypoint_low;
9.    for (int i = 0; i < 3;i++ )
      {
10.     for (int j = 1; j <= 2;j++ )
        {
11.       Bitmap gambar1 = new Bitmap(dog[i, j - 1]), gambar2 = new
    Bitmap(dog[i, j]), gambar3 = new Bitmap(dog[i, j +1]);
12.       deteksi_keypoint_low = new Bitmap(gambar2);
13.       for (int a = 1; a < gambar1.Width - 1; a++)
          {
14.         for (int b = 1; b < gambar1.Height - 1; b++)
            {
              //mencari turunan pertama
15.           turunan_pertama[0]                                    =
    (Convert.ToDouble(gambar2.GetPixel(a,     b     +     1).R)     -
    Convert.ToDouble(gambar2.GetPixel(a, b - 1).R)) / 2;
16.           turunan_pertama[1] = (Convert.ToDouble(gambar2.GetPixel(a
    + 1, b).R) - Convert.ToDouble(gambar2.GetPixel(a - 1, b).R)) / 2;
17.           turunan_pertama[2]                                    =
    (Convert.ToDouble(gambar3.GetPixel(a,           b).R)           -
    Convert.ToDouble(gambar1.GetPixel(a, b).R)) / 2;


              //mencari turunan kedua
18.           turunan_kedua[0,0] = Convert.ToDouble(gambar2.GetPixel(a,
    b + 1).R) - (2 * Convert.ToDouble(gambar2.GetPixel(a, b).R)) +
    Convert.ToDouble(gambar2.GetPixel(a, b - 1).R);
19.           turunan_kedua[0,1] = (Convert.ToDouble(gambar2.GetPixel(a
    + 1, b + 1).R) - Convert.ToDouble(gambar2.GetPixel(a - 1, b + 1).R)
    -   Convert.ToDouble(gambar2.GetPixel(a   +   1,   b   -   1).R)   +
    Convert.ToDouble(gambar2.GetPixel(a - 1, b - 1).R)) / 4;
20.           turunan_kedua[0,2]= (Convert.ToDouble(gambar3.GetPixel(a,
    b + 1).R) - Convert.ToDouble(gambar1.GetPixel(a, b + 1).R) -
    Convert.ToDouble(gambar3.GetPixel(a,     b     -     1).R)     +
    Convert.ToDouble(gambar1.GetPixel(a, b - 1).R)) / 4;
21.           turunan_kedua[1,0] = (Convert.ToDouble(gambar2.GetPixel(a
    + 1, b + 1).R) - Convert.ToDouble(gambar2.GetPixel(a + 1, b - 1).R)
    -   Convert.ToDouble(gambar2.GetPixel(a   -   1,   b   +   1).R)   +
    Convert.ToDouble(gambar2.GetPixel(a - 1, b - 1).R)) / 4;
22.           turunan_kedua[1, 1] = Convert.ToDouble(gambar2.GetPixel(a
    + 1, b).R) - (2 * Convert.ToDouble(gambar2.GetPixel(a, b).R)) +
    Convert.ToDouble(gambar2.GetPixel(a - 1, b).R);
23.           turunan_kedua[1,                    2]                =
```

```
      (Convert.ToDouble(gambar3.GetPixel(a      +      1,      b).R)      -
      Convert.ToDouble(gambar1.GetPixel(a      +      1,      b).R)      -
      Convert.ToDouble(gambar3.GetPixel(a      -      1,      b).R)      +
      Convert.ToDouble(gambar1.GetPixel(a - 1, b).R)) / 4;
24.           turunan_kedua[2,                    0]                    =
      (Convert.ToDouble(gambar3.GetPixel(a,      b      +      1).R)      -
      Convert.ToDouble(gambar3.GetPixel(a,      b      -      1).R)      -
      Convert.ToDouble(gambar1.GetPixel(a,      b      +      1).R)      +
      Convert.ToDouble(gambar1.GetPixel(a, b - 1).R)) / 4;
25.           turunan_kedua[2,                    1]                    =
      (Convert.ToDouble(gambar3.GetPixel(a      +      1,      b).R)      -
      Convert.ToDouble(gambar3.GetPixel(a      -      1,      b).R)      -
      Convert.ToDouble(gambar1.GetPixel(a      +      1,      b).R)      +
      Convert.ToDouble(gambar1.GetPixel(a - 1, b).R)) / 4;
26.           turunan_kedua[2,                    2]                    =
      Convert.ToDouble(gambar3.GetPixel(a,      b).R)      -      (2      *
      Convert.ToDouble(gambar2.GetPixel(a,                b).R))          +
      Convert.ToDouble(gambar1.GetPixel(a, b).R);

            //mencari determinan
27.         double determinan = 0;
28.         int m = 0;
29.         for (int n = 0; n < 3; n++)
            {
              //nilai plus
30.           int x = m + 1, y = n + 1;
31.           double nilai1 = turunan_kedua[m, n] * turunan_kedua[x %
      3, y % 3] * turunan_kedua[(x + 1) % 3, (y + 1) % 3];

              //nilai minus
32.           y = n + 2;
33.           double   nilai2   =   -1   *   (turunan_kedua[m,   n]   *
      turunan_kedua[x, y % 3] * turunan_kedua[x + 1, (y + 2) % 3]);
34.           determinan = determinan + (nilai1 + nilai2);
            }
35.         if (determinan != 0)
            {//menghindari terjadinya nilai NaN atau Infinity
            //mencari invers
36.           double nilai = 1;
37.           double[] no = new double[4];
38.           double[,] hasil_invers = new double[3, 3];
39.           for (int o = 0; o < 3; o++)
              {
40.             for (int p = 0; p < 3; p++)
                {
41.               int jum = 0;
42.               for (int k = 0; k < 3; k++)
                  {
43.                 if (k != o)
                    {
44.                   for (int l = 0; l < 3; l++)
                      {
45.                     if (l != p)
                        {
46.                       no[jum] = turunan_kedua[k, l];
47.                       jum++;
                        }
                      }
                    }
                  }
48.               int cek = o + p, tanda;
49.               if ((cek % 2) == 0) tanda = 1;
50.               else tanda = -1;
51.               nilai = (tanda * ((no[0] * no[3]) - (no[1] * no[2])))
```

```
/ determinan;
52.                 hasil_invers[p, o] = nilai;//proses transpose
                  }
                }

              //mencari nilai z
53.           double[] z = new double[3];
54.           for (int o = 0; o < 3; o++)
              {
55.             for (int p = 0; p < 3; p++)
                {
56.               z[o] += (hasil_invers[o, p] * turunan_pertama[p]);
                }
57.             z[o] *= -1;
              }

58.           if (z[0] < threshold_z && z[1] < threshold_z && z[2] <
threshold_z)
              {
                //mencari D(z)
59.             double perkalian = 0, Dz = 0;
60.             for (int r = 0; r < 3; r++)
                {
61.               perkalian += (z[r] * turunan_pertama[r]);
                }
62.             Dz = Convert.ToDouble(gambar2.GetPixel(a, b).R) + (0.5 *
perkalian);
63.             if (Dz > max) max = Dz;
64.             if (Dz < min) min = Dz;
65.             count++;
66.             array_dz[index_array_z, 0] = Dz;
67.             array_dz[index_array_z, 1] = a;
68.             array_dz[index_array_z, 2] = b;
69.             index_array_z++;
              }
            }
          }
        }
        //proses normalisasi
70.     for (int u = 0; u < count; u++)
        {
71.       double dz_normal = (((array_dz[u, 0] - min) / (max - min)) *
(1 - 0)) + 0;
72.       if (dz_normal >= 0.03)
          {
73.         int    x    =    Convert.ToInt32(array_dz[u,    1]),    y    =
Convert.ToInt32(array_dz[u, 2]);
74.         deteksi_keypoint_low.SetPixel(x, y, Color.FromArgb(255, 255,
255));
          }
        }
75.     arr_low_contrast[i, j-1] = deteksi_keypoint_low;
      }
76.   count = 0;
77.   index_array_z = 0;
    }
  }

78.   private  void  banding_piksel(Bitmap[,]  arr_ekstremum,  Bitmap[,]
low_contrast, out Bitmap[,] hasil_low)
  {
79.   hasil_low = new Bitmap[3,2];
80.   Bitmap hasil;
81.   for (int a = 0; a < 3;a++ )
```

```
    {
82.   for (int b = 0; b < 2; b++)
      {
83.     Bitmap gambar1 = new Bitmap(arr_ekstremum[a,b]), gambar2 =
      new   Bitmap(low_contrast[a,b]);
84.     hasil = gambar1;
85.     for (int i = 0; i < gambar1.Width; i++)
        {
86.       for (int j = 0; j < gambar1.Height; j++)
          {
87.         if     (gambar1.GetPixel(i,     j).R     ==     255     &&
      gambar2.GetPixel(i, j).R == 255)
            {
88.           hasil.SetPixel(i, j, Color.FromArgb(255, 255, 255));
            }
89.         else hasil.SetPixel(i, j, Color.FromArgb(0, 0, 0));
          }
        }
90.     hasil_low[a, b] = hasil;
      }
    }
  }

91. private void eliminasi_edge(Bitmap[,] dog, Bitmap[,] hasil_low, out
    Bitmap[,] arr_edge)
    {
92.   arr_edge = new Bitmap[3,2];
93.   double rasio = 1.01, nilai_threshold = (Math.Pow(rasio + 1, 2)) /
    rasio;
94.   Bitmap hasil_reject, hasil_tes_low, tes;
95.   for (int a = 0; a < 3;a++ )
      {
96.     for (int b = 1; b <= 2; b++)
        {
97.       Bitmap   gambar2   =   new   Bitmap(dog[a,b]),   gambar3   =   new
    Bitmap(hasil_low[a, b-1]);
98.       hasil_tes_low = gambar2; hasil_reject = gambar3; tes = new
    Bitmap(gambar3);
99.       for (int i = 1; i < hasil_tes_low.Width - 1; i++)
          {
100.        for (int j = 1; j < hasil_tes_low.Height - 1; j++)
            {
101.          if (tes.GetPixel(i, j).R == 255)
              {
102.            double  dxx  =  Convert.ToDouble(gambar2.GetPixel(i,  j  +
    1).R)   -   (2   *   (Convert.ToDouble(gambar2.GetPixel(i,   j).R)))   +
    Convert.ToDouble(gambar2.GetPixel(i, j - 1).R);
103.            double  dyy  =  Convert.ToDouble(gambar2.GetPixel(i  +  1,
    j).R)   -   (2   *   (Convert.ToDouble(gambar2.GetPixel(i,   j).R)))   +
    Convert.ToDouble(gambar2.GetPixel(i - 1, j).R);
104.            double dxy = (Convert.ToDouble(gambar2.GetPixel(i + 1,
    j + 1).R) - Convert.ToDouble(gambar2.GetPixel(i - 1, j + 1).R) -
    Convert.ToDouble(gambar2.GetPixel(i    +    1,    j    -    1).R)    +
    Convert.ToDouble(gambar2.GetPixel(i - 1, j - 1).R)) / 4;
105.            double rasio_titik = (Math.Pow(dxx + dyy, 2)) / ((dxx *
    dyy) - (Math.Pow(dxy, 2)));
106.            if (rasio_titik > nilai_threshold)
                {
107.              hasil_reject.SetPixel(i,  j,  Color.FromArgb(255,  255,
    255));
                }
108.            else  hasil_reject.SetPixel(i,  j,  Color.FromArgb(0,  0,
    0));
              }
```

```
109.            else  hasil_reject.SetPixel(i,  j,  Color.FromArgb(0,  0,
    0));
                }
            }
110.        arr_edge[a, b - 1] = hasil_reject;
        }
    }
}
```

Gambar 4.12 Implementasi Sub-Algoritma Eliminasi *Keypoint Low Contrast* Dan Tepi

Sumber : Implementasi

Baris 11 berfungsi untuk mengambil tiga citra *difference of gaussian* yang akan dicari titik ekstremumnya berdasarkan perhitungan *low-contrast*. Baris 15-17 berfungsi untuk mencari turunan pertama dari persamaan *taylor expansion*. Baris 18-28 berfungsi untuk mencari turunan kedua dari persamaan *taylor expansion*. Baris 53-57 berfungsi untuk mencari nilai variabel z. Baris 59-62 berfungsi untuk mencari nilai D(z), yang akan dinormalisasi pada baris 70-74. Baris 78-90 berfungsi untuk membandingkan hasil deteksi citra titik ekstremum sebelumnya dengan hasil citra titik ekstremum dengan perhitungan *low-contrast*. Baris 91-110 berfungsi untuk mengeliminasi titik *keypoint* yang merupakan tepi.

4.3.8.5 Penentuan Orientasi *Keypoint*

Proses penentuan orientasi *keypoint* bertujuan untuk memberikan nilai *magnitude* dan orientasi sudut dari tiap *keypoint* yang telah didapatkan. Penentuan orientasi *keypoint* didapatkan berdasarkan nilai tertinggi pada masukan *histogram of gradient*. Gambar 4.13 merupakan cuplikan kode yang mengimplementasikan proses penentuan orientasi *keypoint*.

```
1.  private void orientasi_keypoint(Bitmap[,] gaussian_pyramid,
    Bitmap[,] arr_edge, double[,] gaussian_scale)
2.  {
3.    //mengosongkan isi orientasi_keypoint txt
4.    File.WriteAllText("orientasi_keypoint.txt", String.Empty);
5.
6.    StreamWriter simpan_orientasi = new
7.  StreamWriter("orientasi_keypoint.txt", true);
8.    for (int i = 0; i < 1;i++ )
      {
        for (int j = 0; j < 1;j++ )
        {
9.        Bitmap keypoint = new Bitmap(arr_edge[i,j]);
          Bitmap laplacian = new Bitmap(gaussian_pyramid[i,j+1]);
10.       Bitmap hasil_orientasi = new Bitmap(laplacian);
          double double_sigma_kon = gaussian_scale[i,j+1], sigma =
11. double_sigma_kon*1.5, magnitude = 0, teta = 0;
```

```
12.       int kernel_size = Convert.ToInt32(3 * sigma);
13.       if (kernel_size % 2 == 0) kernel_size += 1;
          //menghitung magnitude
14.       int n = kernel_size / 2;

15.       for (int a = 0; a < keypoint.Width; a++)
          {
16.         for (int b = 0; b < keypoint.Height; b++)
            {
17.           if (keypoint.GetPixel(a, b).R == 255)
              {
18.             double[,] array_teta = new double[36, 3];
19.             for (int x = (a - n), gauss1 = -n; x <= (a + n) &&
gauss1 <= n; x++, gauss1++)
                {
20.               for (int y = (b - n), gauss2 = -n; y <= (b + n) &&
gauss2 <= n; y++, gauss2++)
                  {
21.                 double gauss_weight = Math.Exp(-1 *
((Math.Pow(gauss1, 2) + Math.Pow(gauss2, 2)) / (2 * Math.Pow(sigma,
2))));
22.                 magnitude = gauss_weight *
(Math.Pow(Math.Pow((laplacian.GetPixel(x, y + 1).R -
laplacian.GetPixel(x, y - 1).R), 2) + Math.Pow((laplacian.GetPixel(x
+ 1, y).R - laplacian.GetPixel(x - 1, y).R), 2), 0.5));
23.                 teta=
(Math.Atan((Convert.ToDouble(laplacian.GetPixel(x + 1, y).R) -
Convert.ToDouble(laplacian.GetPixel(x - 1, y).R)) /
(Convert.ToDouble(laplacian.GetPixel(x, y + 1).R) -
Convert.ToDouble(laplacian.GetPixel(x, y - 1).R))) * 180) / Math.PI;
24.                 if (teta < 0) teta += 360;
                    //seleksi di rentang mana teta berada
25.                 int indeks = (Convert.ToInt32(teta) / 10);
26.                 array_teta[indeks, 0] = teta;
27.                 array_teta[indeks, 1]++;
28.                 array_teta[indeks, 2] += magnitude;
                  }
                }

                //mencari peak pada histogram
29.             double maks = 0;
30.             int indeks_max = 0;
31.             for (int m = 0; m < 36; m++)
                {
32.               double persen = (array_teta[m, 1] / (kernel_size *
kernel_size)) * 100;
33.               if (persen > maks)
                  {
34.                 indeks_max = m;
35.                 maks = persen;
                  }

36.               teta = (indeks_max * 10) + 4.5;
37.               int x1 = a, y1 = b;
38.               double x2 = x1 + (Math.Cos(teta) * teta);
39.               double y2 = y1 + (Math.Sin(teta) *
array_teta[indeks_max, 2]);
40.               simpan_orientasi.WriteLine(i + " " + (j+1) + " " +
gaussian_scale[i, j + 1] + " " + a + " " + b + " " +
array_teta[indeks_max, 2]+" "+teta);
              }
            }
          }
        }
```

```
        }
41.     simpan_orientasi.Close();
    }
```

Gambar 4.13 Implementasi Sub-Algoritma SIFT Penentuan Orientasi *Keypoint*

Sumber : Implementasi

Baris 22 berfungsi untuk menghitung nilai *magnitude* dari *keypoint*. Baris 23 berfungsi untuk menghitung nilai orientasi dari keypoint. Baris 29-35 berfungsi untuk mencari nilai *peak*/puncak tertinggi dari *gradient histogram*.

4.3.8.6 Penentuan Deskriptor Citra

Proses penentuan deskriptor citra bertujuan untuk memperoleh nilai-nilai fitur yang berasal dari *region* tiap *keypoint*, fitur-fitur inilah yang menjadi identitas tiap *keypoint* yang dimiliki oleh citra. Gambar 4.14 merupakan cuplikan kode yang mengimplementasikan proses penentuan *descriptor* citra.

```
1.  private void image_deskriptor(Bitmap[,] gaussian_pyramid, int
    lebar_deskriptor, int n_bin_histogram)
    {
2.    Bitmap laplacian;
      //baca fitur-fitur keypoint
      int count=0, indeks=0;
3.    double[,] list_keypoint = new double[1000,7];
      StreamReader baca_keypoint = new
4.  StreamReader("orientasi_keypoint.txt");
5.    string keypoint;
6.    while ((keypoint = baca_keypoint.ReadLine()) != null)
      {
7.      string[] data_split = keypoint.Split(' ');
8.      for (int indeks_2 = 0; indeks_2 < 7;indeks_2++ )
        {
9.        list_keypoint[indeks, indeks_2] =
    Convert.ToDouble(data_split[indeks_2]);
        }
10.     count++;
11.     indeks++;
      }
12.   double[, ,] histogram;

13.   for (int i = 0; i < count; i++)
      {
14.     int oktaf = Convert.ToInt32(list_keypoint[i, 0]);
15.     int level_scale = Convert.ToInt32(list_keypoint[i, 1]);
16.     laplacian = new Bitmap(gaussian_pyramid[oktaf, level_scale]);
17.     histogram = deskriptor_histogram(laplacian,
    Convert.ToInt32(list_keypoint[i, 3]),
    Convert.ToInt32(list_keypoint[i, 4]), list_keypoint[i, 6],
    list_keypoint[i, 2], lebar_deskriptor, n_bin_hist_deskriptor);
18.     histogram_ke_deskriptor(histogram, lebar_deskriptor,
    n_bin_hist_deskriptor);
      }
19.   baca_keypoint.Close();
    }
```

```
20.  private void histogram_ke_deskriptor(double[, ,] histogram, int
     lebar_deskriptor, int n_bin_hist_deskriptor)
     {
21.    int int_val, i, r, c, o, k = 0, panjang_deskriptor;
22.    double[] deskriptor = new
     double[lebar_deskriptor*lebar_deskriptor*n_bin_hist_deskriptor];

23.    for (r = 0; r < lebar_deskriptor; r++)
24.      for (c = 0; c < lebar_deskriptor; c++)
25.        for (o = 0; o < n_bin_hist_deskriptor; o++)
26.          deskriptor[k++] = histogram[r, c, o];

27.          panjang_deskriptor = k;
28.          normalize_descr(deskriptor, panjang_deskriptor);
29.          for (i = 0; i < k; i++)
30.            if (deskriptor[i] > 0.2)
31.              deskriptor[i] = 0.2;
32.            normalize_descr(deskriptor, panjang_deskriptor);
             }

33.  void normalize_descr(double[] deskriptor, int panjang_deskriptor)
     {
34.    double cur, len_inv, len_sq = 0.0;
35.    int i, d = panjang_deskriptor;

36.    for (i = 0; i < d; i++)
       {
37.      cur = deskriptor[i];
38.      len_sq += cur * cur;
       }
39.    len_inv = 1.0 / Math.Sqrt(len_sq);
40.    for (i = 0; i < d; i++)
       {
41.      deskriptor[i] *= len_inv;
       }
     }

42.  double[, ,] deskriptor_histogram(Bitmap gaussian, int x, int y,
     double orientasi, double scale_oktaf, int lebar_deskriptor, int
     bin_histogram)
     {
43.    double[, ,] histogram;
44.    double cos_teta, sin_teta, lebar_histogram, exp, r_rot, c_rot,
     grad_mag, grad_ori, w, rbin, cbin, obin, bins_per_rad, PI2 = 2.0 *
     Math.PI;
45.    int radius;

46.    histogram = new double[lebar_deskriptor, lebar_deskriptor,
     bin_histogram];

47.    cos_teta = Math.Cos(orientasi);
48.    sin_teta = Math.Sin(orientasi);
49.    bins_per_rad = bin_histogram / PI2;
50.    exp = lebar_deskriptor * lebar_deskriptor * 0.5;
51.    lebar_histogram = 3 * scale_oktaf;
52.    radius =
53.  Convert.ToInt32(lebar_histogram*Math.Sqrt(2)*(lebar_deskriptor+1.0)
     *0.5+0.5);

54.    for (int i = -radius; i <= radius;i++ )//untuk baris
       {
55.      for (int j = -radius; j <= radius;j++) //untuk kolom
         {
           //menghitung interpolasi piksel
```

```
56.         c_rot = (j * cos_teta - i * sin_teta) / lebar_histogram;
57.         r_rot = (j * sin_teta + i * cos_teta) / lebar_histogram;
58.         rbin = r_rot + lebar_deskriptor / 2 - 0.5;
59.         cbin = c_rot + lebar_deskriptor / 2 - 0.5;

60.         if (rbin > -1.0 && rbin < lebar_deskriptor && cbin > -1.0 &&
      cbin < lebar_deskriptor)
            {
61.           if(menghitung_grad_mag_ori(gaussian, y+i, x+j, out
      grad_mag, out grad_ori) !=0){
62.           grad_ori -= orientasi;
63.           while (grad_ori < 0.0) grad_ori += PI2;
64.           while (grad_ori >= PI2) grad_ori -= PI2;

65.           obin = grad_ori * bins_per_rad;
66.           w = Math.Exp(-(c_rot * c_rot + r_rot * r_rot) / exp);
67.           interpolasi_hist_entri(ref histogram, rbin, cbin, obin,
      grad_mag * w, lebar_deskriptor, bin_histogram);
            }
          }

      }
    }
68.  return histogram;
}

69.  private void interpolasi_hist_entri(ref double[, ,] histogram,
     double rbin, double cbin, double obin, double mag, int
     lebar_deskriptor, int bin_histogram)
     {
70.    float d_r, d_c, d_o, v_r, v_c, v_o;
71.    int r0, c0, o0, rb, cb, ob, r, c, o;
72.    r0 = (int)Math.Floor(rbin);
73.    c0 = (int)Math.Floor(cbin);
74.    o0 = (int)Math.Floor(obin);
75.    d_r = (float)rbin - r0;
76.    d_c = (float)cbin - c0;
77.    d_o = (float)obin - o0;

       //memasukkan nilai masukan /entri ke 8 bin histogram
78.    for (r = 0; r <= 1; r++)
       {
79.      rb = r0 + r;
80.      if (rb >= 0 && rb < lebar_deskriptor)
         {
81.        v_r = (float)mag * ((r == 0) ? 1.0F - d_r : d_r);
82.        for (c = 0; c <= 1; c++)
           {
83.          cb = c0 + c;
84.          if (cb >= 0 && cb < lebar_deskriptor)
             {
85.            v_c = v_r * ((c == 0) ? 1.0F - d_c : d_c);

86.            for (o = 0; o <= 1; o++)
               {
87.              ob = (o0 + o) % bin_histogram;
88.              v_o = v_c * ((o == 0) ? 1.0F - d_o : d_o);
89.              histogram[rb, cb, ob] += v_o;
               }
             }
           }
         }
       }
     }
```

```
90.   int menghitung_grad_mag_ori(Bitmap gaussian, int y, int x, out
      double grad_mag, out double grad_ori)
      {
91.     double dx, dy;

92.     if (y > 0 && y < gaussian.Height - 1 && x > 0 && x <
      gaussian.Width - 1)
        {
93.       dx = gaussian.GetPixel(x,y+1).R - gaussian.GetPixel(x, y-1).R;
94.       dy = gaussian.GetPixel(x+1, y).R - gaussian.GetPixel(x-1, y).R;
95.       grad_mag = Math.Sqrt(dx * dx + dy * dy);
96.       grad_ori = Math.Atan2(dy, dx);
97.       return 1;
        }
98.     else
        {
99.       grad_mag = 0;
100       grad_ori = 0;
101       return 0;
        }
      }
```

Gambar 4.14 Implementasi Sub-Algoritma SIFT Penentuan Deskriptor Citra

Sumber : Implementasi

Baris 13-18 berfungsi untuk mengambil *keypoint* yang telah terdeteksi pada list_keypoint.txt. Baris 42-68 berfungsi untuk menghitung interpolasi piksel pada region sekitar *keypoint*. Baris 69-89 berfungsi untuk menghitung interpolasi data *entry* yang akan dimasukkan ke dalam *gradient* histogram. Baris 20-32 berfungsi untuk mengubah nilai yang terdapat pada *gradient* histogram menjadi bentuk deskriptor. Baris 33-41 berfungsi untuk melakukan normalisasi deskriptor ke *unit length*.

4.3.8.7 *Keypoint Matching*

Proses *keypoint matching* ini bertujuan untuk melakukan proses pencocokan kemiripan yang dimiliki antara satu citra objek dengan citra objek yang lain. Proses ini dilakukan dengan mencari nilai jarak menggunakan *euclidien* antara nilai fitur deskriptor *keypoint* yang dimiliki masing-masing citra. Gambar 4.15 merupakan cuplikan kode yang mengimplementasikan proses *keypoint matching*.

```
1.    private void keypoint_matching(Bitmap gambar_objek, int[,]
      deskriptor, out double euclidien)
            {
2.              StreamReader baca_keypoint = new
      StreamReader("list_deskriptor.txt");
3.              int[] deskriptor_2 = new int[128];
4.              double sum_square=0,  kuadrat, threshold,
```

```
5.  piksel_match=0;
6.              string data = baca_keypoint.ReadLine();
7.              string[] data_split = data.Split(' ');
8.              euclidien = 0;
9.              for (int i = 0; i < 16;i++ )
                {
10.                 for(int j=0, m=2;j<8;j++, m++){
11.                     deskriptor_2[i, j] =
    Convert.ToInt32(data_split[m]);
                    }
                }

                //matching
12.             for (int i = 0; i < 16;i++ )
                {
13.                 for (int j = 0; j < 8;j++ )
                    {
14.                     kuadrat = Math.Pow((deskriptor[i, j] -
    deskriptor_2[i, j]),2);
15.                     sum_square += kuadrat;
                    }
                }
16              euclidien = Math.Sqrt(sum_square);
17  if(euclidien <= 0.3) piksel_match++;
            }
```

Gambar 4.15 Implementasi Sub-Algoritma SIFT *Keypoint Matching*

Sumber : Implementasi

Baris 12-16 berfungsi untuk menghitung jarak antara dua *keypoint*, jika nilai jarak lebih besar dari 0.3 maka jumlah piksel akan bertambah, dan jika jumlah piksel yang sesuai lebih dari 70% maka kedua objek dianggap identik.

4.4 Implementasi Antar Muka Aplikasi

Antarmuka Aplikasi *multiple object tracking* pada video cctv digunakan oleh pengguna untuk berinteraksi dengan sistem perangkat lunak. Antarmuka perangkat lunak ini dibagi menjadi dua yaitu antarmuka halaman *tracking*, dan antar muka halaman tambah citra *background*.

4.4.1 Halaman *Tracking*

Halaman *tracking* berfungsi memasukkan citra video yang akan dilacak dan halaman dimana hasil *tracking* akan ditampilkan. Gambar 4.16 dan Gambar 4.17 menunjukkan implementasi tampilan antarmuka dari halaman *tracking* yang mengacu pada perancangan antarmuka halaman *tracking*.

Gambar 4.16 Implementasi Halaman *Tracking* Objek Tunggal

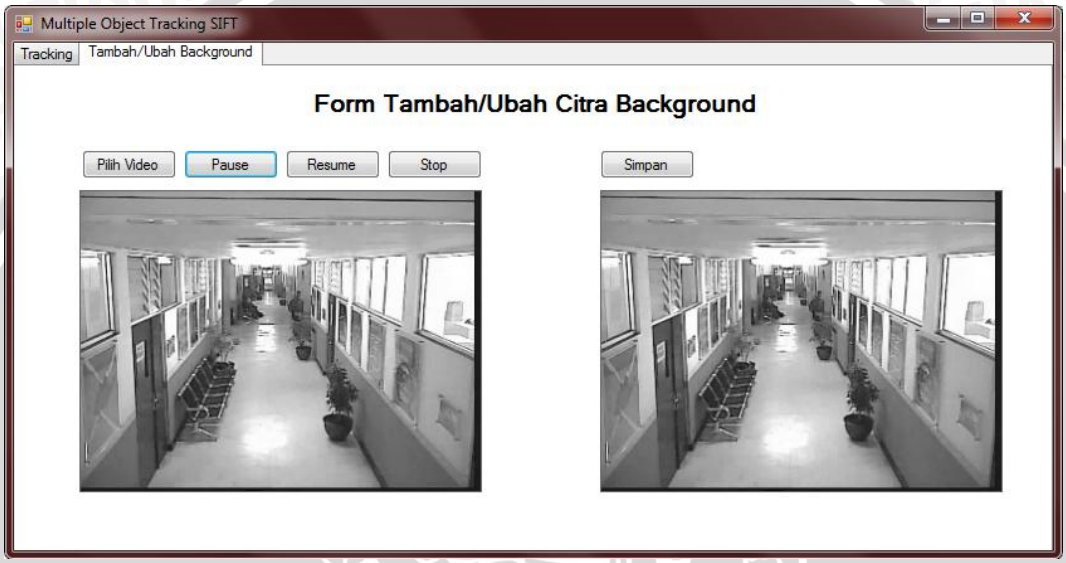
Sumber : Implementasi

Gambar 4.17 Implementasi Halaman *Tracking* Objek *Multiple*

Sumber : Implementasi

4.4.2 Halaman Tambah Citra *Background*

Halaman tambah citra background berfungsi memasukkan citra *background* yang akan ditangkap dari salah satu *frame* video . Gambar 4.18 menunjukkan implementasi tampilan antarmuka dari halaman tambah citra *background* yang mengacu pada perancangan antarmuka halaman tambah citra *background*.

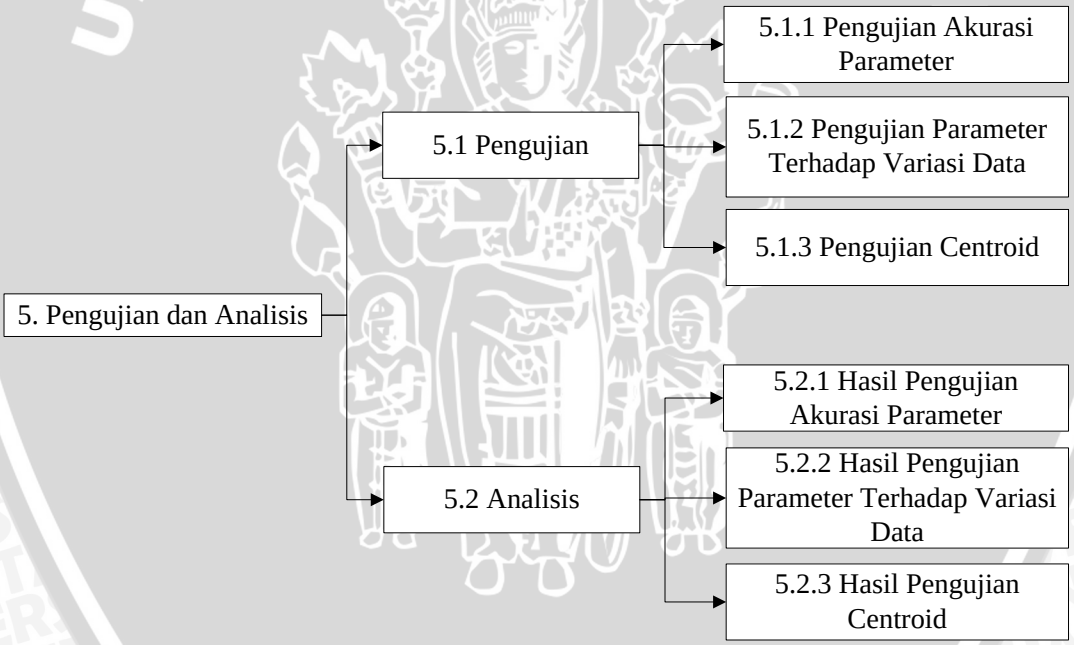


Gambar 4.18 Implementasi Halaman Tambah Citra *Background*

Sumber : Implementasi

BAB V
PENGUJIAN DAN ANALISIS

Bab ini membahas mengenai tahapan pengujian dan analisis sistem *multiple object tracking* dengan metode SIFT pada video CCTV. Proses pengujian dilakukan dengan cara pengujian akurasi yang dihasilkan oleh sistem pada beberapa kombinasi nilai konstanta tertentu. Pengujian akurasi dilakukan dengan cara menghitung akurasi pada setiap jumlah objek yang berhasil terdeteksi tiap *frame*. Pencocokan akurasi dilakukan dengan membandingkan jumlah objek yang dideteksi oleh sistem dengan jumlah objek yang dideteksi berdasarkan pengamatan mata manusia. Gambar 5.1 menunjukkan ilustrasi dari pengujian aplikasi ini :



Gambar 5.1 Pohon Pengujian Dan Analisis

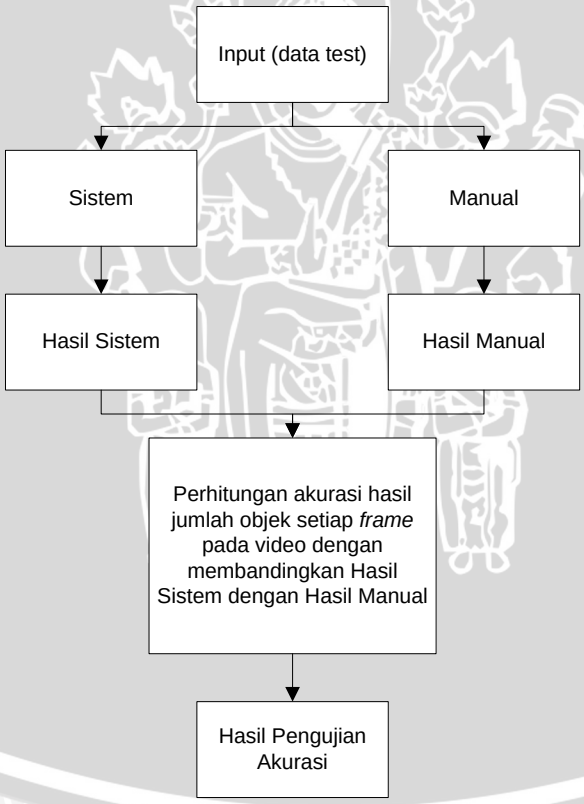
Sumber : Pengujian dan Analisis

5.1 Pengujian

5.1.1 Pengujian Akurasi Parameter

Pengujian akurasi digunakan untuk mengetahui performa dari sistem *multiple object tracking* yang menggunakan metode *background substraction* dan

multiple object tracking dalam mendeteksi dan melacak objek yang terekam ke dalam video CCTV. Pengujian ini dilakukan dengan menghitung akurasi pada setiap *frame* video CCTV yang akan dilacak. Perhitungan akurasi dilakukan dengan cara membandingkan antara jumlah hasil deteksi yang dilakukan oleh sistem dengan jumlah hasil deteksi yang dilakukan oleh mata manusia. Selain itu, perhitungan akurasi tersebut juga dilakukan berdasarkan nilai-nilai parameter yang menjadi inputan seperti nilai *scale Gaussian, r*, dan *threshold matching* atau tidaknya sebuah objek. Sehingga berdasarkan keragaman komposisi nilai parameter-parameter tersebut akan diketahui nilai tertinggi akurasi didapatkan pada komposisi nilai-nilai tersebut. Data *frame* video yang akan digunakan sebanyak 30 *frame* dari satu video dengan resolusi sebesar 320x240, *frame rate* 30 fps.



Gambar 5.2 Diagram Alir Proses Pengujian Akurasi

Sumber : Pengujian dan Analisis

Tabel 5.1 mengilustrasikan nilai *scale* Gaussian, dan *r* serta nilai *threshold matching*. Sedangkan padaTabel 5.2 mengilustrasikan kombinasi dari ketiga nilai parameter tersebut.

Tabel 5.1 Nilai-nilai Parameter yang Akan Digunakan

Parameter	Nilai 1	Nilai 2	Nilai 3
Scale Gaussian	0,500	0,707	1,410
r	1,010	2,000	3,000
Matching Threshold	0,300	0,500	0,800

Sumber : Pengujian dan Analisis

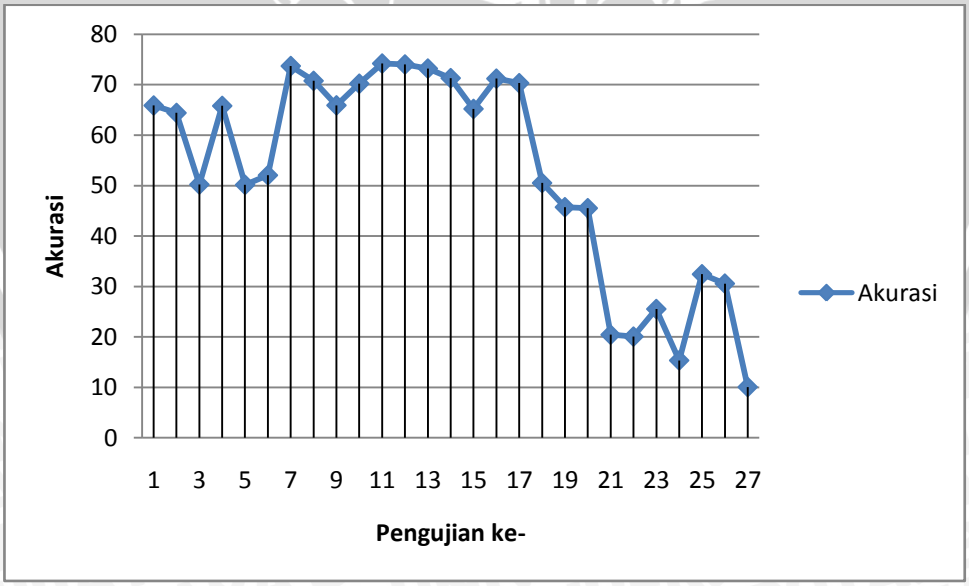
Tabel 5.2 Kombinasi Nilai Parameter Tiap Pengujian

Pengujian ke-	Scale Gaussian	R	Threshold matching	Akurasi
1	0,500	1,010	0,300	65,87
2	0,500	1,010	0,500	64,4
3	0,500	1,010	0,800	50,2
4	0,500	2,000	0,300	65,79
5	0,500	2,000	0,500	50,13
6	0,500	2,000	0,800	52,03
7	0,500	3,000	0,300	73,7
8	0,500	3,000	0,500	70,76
9	0,500	3,000	0,800	65,9
10	0,707	1,010	0,300	70,2
11	0,707	1,010	0,500	74,2
12	0,707	1,010	0,800	74,01
13	0,707	2,000	0,300	73,2
14	0,707	2,000	0,500	71,3
15	0,707	2,000	0,800	65,2
16	0,707	3,000	0,300	71,2
17	0,707	3,000	0,500	70,3
18	0,707	3,000	0,800	50,5

19	1,410	1,010	0,300	45,7
20	1,410	1,010	0,500	45,5
21	1,410	1,010	0,800	20,4
22	1,410	2,000	0,300	20,03
23	1,410	2,000	0,500	25,5
24	1,410	2,000	0,800	15,3
25	1,410	3,000	0,300	32,4
26	1,410	3,000	0,500	30,54
27	1,410	3,000	0,800	10,02

Sumber : Pengujian dan Analisis

Berdasarkan hasil pengujian akurasi 30 frame dari satu video tersebut didapatkan bahwa aplikasi *multiple object tracking* dengan metode SIFT memiliki nilai akurasi terbesar yakni 74,2% pada kombinasi nilai *scale* Gaussian = 0,707, *r*=1,01, dan *threshold matching* sebesar 0,5. Dari hasil ini disimpulkan bahwa Metode SIFT sudah mampu melakukan *object tracking* dengan baik.



Gambar 5.3 Nilai Akurasi Pada Masing-masing Komposisi Parameter

Sumber : Pengujian dan Analisis

Gambar 5.3 adalah grafik yang menjelaskan tentang hasil pengujian akurasi sistem pada komposisi nilai ketiga parameter yang telah disebutkan sebelumnya. Dari grafik dapat dilihat bahwa nilai akurasi terbesar ditunjukkan pada komposisi nilai pada pengujian ke-11 dimana nilai *scale* Gaussian = 0,707, *r*=1,01, dan *threshold matching* sebesar 0,5. Sehingga dapat disimpulkan pada data video CCTV dengan spesifikasi di atas, nilai akurasi terbaik yang bisa digunakan sistem berada pada kombinasi nilai pengujian ke-11.

5.1.2 Pengujian Hasil Parameter Pada Video Lain

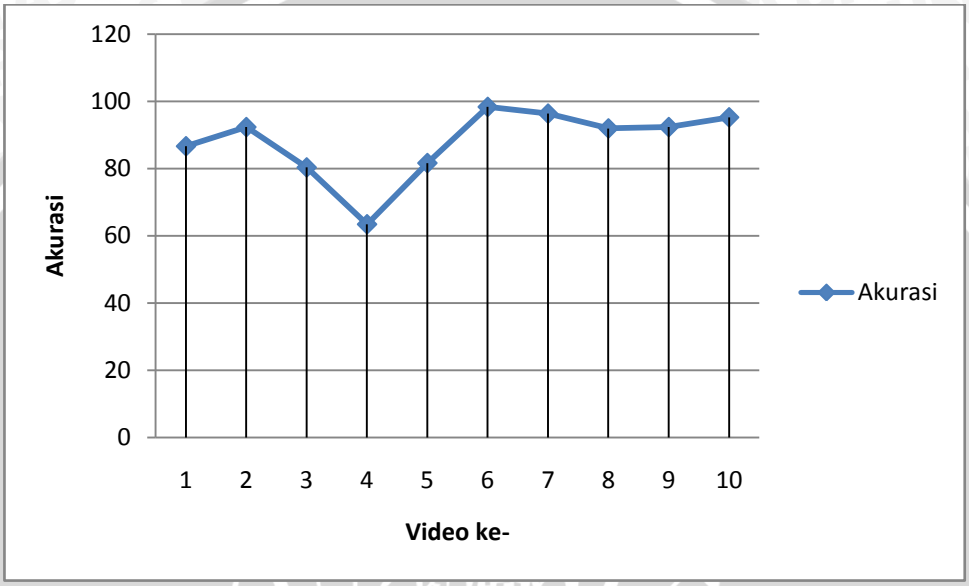
Pengujian ini dilakukan dengan melakukan pengujian terhadap hasil pengujian data parameter terbaik pada 10 data video dengan durasi waktu sebesar 10 atau 20 detik. Tabel 5.3 menunjukkan nilai dari masing-masing hasil pengujian dari 10 data video.

Tabel 5.3 Hasil Pengujian Data Parameter Terbaik Pada Variasi Video

Data Video	Frame Sesuai	Frame Tidak Sesuai	Akurasi
Video 1	260	40	86,6
Video 2	277	23	92,3
Video 3	241	59	80,3
Video4	185	115	63,4
Video 5	245	55	81,6
Video 6	646	11	98,3
Video 7	633	24	96,3
Video 8	604	53	91,9
Video 9	607	50	92,3
Video 10	626	49	95,2
Akurasi Rata-rata			87,82

Sumber : Pengujian dan Analisis

Gambar 5.4 menunjukkan grafik dari hasil pengujian data parameter terhadap 10 data video. Berdasarkan grafik tersebut dapat diketahui bahwa nilai parameter pada kombinasi pengujian ke-11 pada pengujian parameter masih mampu menghasilkan hasil akurasi deteksi dan pelacakan objek dengan baik terhadap video lain dengan akurasi rata-rata sebesar 87,82%.



Gambar 5.4 Hasil Pengujian Data Parameter Terbaik Pada Variasi Video

Sumber : Pengujian dan Analisis

5.1.3 Pengujian *Centroid*

Pengujian ini dilakukan dengan tujuan untuk mengetahui tingkat *error rate* dari hasil deteksi *region* objek dengan menggunakan *centroid*. Centroid dihitung berdasarkan titik pojok pada masing-masing objek yang telah terdeteksi. Data yang digunakan pada pengujian ini sebanyak 6 *frame* dari satu video.

Tabel 5.4 menunjukkan citra yang merupakan hasil dari proses *tracking* secara manual dan secara otomatis dengan sistem menggunakan metode SIFT.

Tabel 5.4 Perbandingan *Centroid*

	Manual	Sistem
Frame 463		
Frame 467		
Frame 470		
Frame 475		

Frame 559		
Frame 565		

Sumber : Pengujian dan Analisis

Tabel 5.5 Hasil Pengujian *Centroid*

Frame no	Objek	Error rate %	Average Error %	
			Objek 1	Objek 2
463	1	1.40		
	2	2.28		
467	1	1.47		
	2	1.91		
470	1	3.33		
	2	1.50	2.61	5.11
475	1	3.89		
	2	1.30		
559	1	3.38		
	2	11.33		
565	1	2.14		
	2	12.28		

Sumber : Pengujian dan Analisis

Berdasarkan Tabel 5.5 diketahui bahwa nilai *error rate* rata-rata dari objek pertama sebesar 2,61% sedangkan *error rate* pada objek kedua sebesar 5,11%. Sehingga dapat diketahui bahwa hasil deteksi dan pelacakan dari *region* objek dari sistem ini memiliki hasil yang baik dengan nilai *error rate* yang cukup kecil.

5.2 Analisis

Proses analisis bertujuan untuk mendapatkan kesimpulan dari hasil pengujian sistem *multiple object tracking* yang telah dilakukan. Proses analisis mengacu pada dasar teori sesuai dengan hasil pengujian yang didapatkan. Analisis dilakukan terhadap hasil pengujian di setiap tahap pengujian. Proses analisis yang dilakukan meliputi analisis hasil pengujian akurasi.

5.2.1 Hasil Pengujian Akurasi

Proses analisis terhadap hasil pengujian akurasi dilakukan dengan menghitung nilai akurasi dari hasil pelacakan objek dari sistem *multiple object tracking* dengan metode SIFT. Proses analisis juga dilakukan terhadap hasil pengujian akurasi pada tiap komposisi nilai dari ketiga parameter yang diujikan. Pengujian ini bertujuan untuk mengetahui apa pengaruh dari ketiga parameter tersebut pada sistem *multiple object tracking* ini. Selain itu juga bertujuan untuk mengetahui nilai parameter terbaik untuk digunakan pada video CCTV dengan kualitas video seperti yang dimiliki video uji.

Berdasarkan hasil pengujian akurasi pada Tabel 5.4 untuk masing-masing perlakuan diketahui bahwa, nilai akurasi terendah dihasilkan pada pengujian ke-27 dimana nilai scale = 1,41, nilai *r* =3 dan nilai *threshold matching* =0,8. Sebaliknya nilai akurasi tertinggi dihasilkan pada pengujian ke-11 dimana nilai *scale* Gaussian = 0,707, *r*=1,01, dan *threshold matching* sebesar 0,5.

Sehingga dapat disimpulkan bahwa semakin tinggi nilai *scale* yang digunakan pada sistem, maka fitur pada citra objek akan semakin memudar karena proses pemburaman dengan level tinggi. Semakin tinggi nilai *r* yang digunakan maka semakin mengurangi *keypoint* yang lolos dari filter. Semakin sedikit *keypoint* yang terdeteksi maka akan semakin mengurangi akurasi yang dihasilkan oleh sistem. Sedangkan semakin tinggi *threshold matching* maka *keypoint* yang sebenarnya tidak *match* akan terdeteksi sebagai *keypoint match*, sehingga akan sering terjadi kesalahan *matching* (*missmatch*). *Object tracking* pada video CCTV yang memiliki resolusi rendah seperti yang digunakan pada video uji sistem ini, maka nilai parameter terbaik berada pada komposisi nilai *scale* Gaussian = 0,707, *r*=1,01, dan *threshold matching* sebesar 0,5.

5.2.2 Pengujian Data Parameter Terhadap Variasi Video

Berdasarkan hasil pengujian yang telah terlihat pada Tabel 5.2 dan Grafik 5.4 diketahui bahwa nilai parameter pada kombinasi pengujian ke-11 pada pengujian parameter masih memiliki nilai akurasi yang tinggi saat dilakukan pengujian terhadap 10 data video lain. Akurasi rata-rata yang didapatkan dari hasil deteksi dan pelacakan pada 10 data video adalah sebesar 87,82%. Sehingga dapat disimpulkan bahwa nilai parameter kombinasi terbaik tersebut telah mampu menghasilkan hasil akurasi deteksi dan pelacakan objek secara stabil terhadap variasi video dan layak digunakan pada sistem *multiple object tracking* dengan spesifikasi video yang sama dengan data video uji.

5.2.3 Pengujian *Centroid*

Proses analisis terhadap hasil pengujian *centroid* dilakukan dengan menghitung nilai *error rate* dari hasil pelacakan objek sistem *multiple object tracking* dengan metode SIFT. Berdasarkan hasil pengujian *centroid* didapatkan bahwa nilai *error rate* pada deteksi dan pelacakan objek pertama sebesar 2,61% sedangkan *error rate* pada objek kedua sebesar 5,11%. Sehingga dapat disimpulkan bahwa hasil deteksi dan pelacakan *region* objek dari sistem ini memiliki hasil yang baik dengan nilai *error rate* yang cukup kecil pada tiap objeknya.

BAB VI
PENUTUP

6.1 Kesimpulan

Berdasarkan hasil perancangan , implementasi dan pengujian yang telah dilakukan, maka diambil kesimpulan sebagai berikut :

1. Aplikasi *multiple object tracking* dengan metode SIFT telah diimplementasikan dan telah mampu melakukan deteksi dan pelacakan pada video CCTV secara baik.
2. Aplikasi *multiple object tracking* dengan metode SIFT memiliki nilai akurasi terbesar yakni 74,2% pada kombinasi nilai *scale Gaussian* = 0,707, r=1,01, dan *threshold matching* sebesar 0,5. Nilai kombinasi parameter terbaik yang dihasilkan telah mampu melakukan deteksi dan pelacakan objek secara baik pada video yang bervariasi.
3. Aplikasi *multiple object tracking* dengan metode SIFT telah mampu mendeteksi dan melacak region dari tiap objek secara baik.

6.2 Saran

Saran yang diberikan untuk pengembangan penelitian selanjutnya, antara lain :

1. Dapat dilakukan optimasi data atau spesifikasi sistem agar didapatkan waktu pemrosesan metode SIFT yang cepat.
2. Dapat dilakukan pengembangan sistem dengan memecahkan masalah oklusi pada objek dengan menggabungkan metode SIFT dengan metode lain.
3. Dapat diimplementasikan pada lingkungan sistem yang terintegrasi dengan kamera CCTV secara langsung, sehingga pengawasan dapat dilakukan secara *realtime*.

DAFTAR PUSTAKA

[SAI-09] Md. Saidur Rahman, Aparna Saha, Snigdha Khanum, "*Multi-Object Tracking in Video Sequences Based on Background Subtraction and* SIFT *Feature Matching", Fourth International Conference on Computer Sciences and Convergence Information Technology*, 2009.

[SHA-12] Shanty Eka Agustina, dan Imam Mukhlash, "*Implementasi Metode Scale Invariant Feature Transform(*SIFT*) Dan Metode Continuosly Adaptive Mean-Shift(Camshift) Pada Penjejakan Objek Bergerak*", *Jurnal Sains Dan Seni*, Vol. 1, No. 1, (2012) 1-6.

[SEO-11] Seok-Wun Ha and Yong-Ho Moon, "*Multiple Object Tracking Using* SIFT *Features and Location Matching*", *International Journal of Smart Home,*Vol. 5, No. 4, October, 2011.

[PAN-13] P M Panchal, S R Panchal, and S K Shah, "*A Comparison of* SIFT *and SURF*", *International Journal of Innovative Research in Computer and Communication Engineering*, Vol. 1, Issue 2, April 2013.

[HAO-12] Haopeng Li and Markus Flierl, "SIFT-*Based Multi-View Cooperative Tracking For Soccer Video*", Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference, 25-30 March 2012.

[HUI-09] Huiyu Zhou, Yuan Yuan, and Chunmei Shi, "*Object tracking using* SIFT *features and mean shift*", *Computer Vision and Image Understanding*, 113 , (2009)345–352.

[DEN-13] Dennis Mitzel and Bastian Leibe, "*Real-Time Multi-Person Tracking with Detector Assisted Structure Propagation*", Germany, RWTH Aachen University.

[CHA-11] S.Challa,M. R. Morelande, D. Muˇsicki and R. J. Evans.2011."*Fundamentals Of Object Tracking*".United States. Cambridge University.

[GOS-11] Goszczynska , Hanna, 2011, "*Object Tracking*", InTech, Rijeka, Croatia.

[WID-11] Widianto , Akbar. 2011."*Pengenalan Wajah Menggunakan Metode Linear Discriminant Analysis*". Universitas Brawijaya. Malang.

[DAV-04] David G. Lowe, "*Distinctive Image Features from Scale-Invariant Keypoints*", Computer Science Department, University of British Columbia, Vancouver, B.C., Canada, 5 Januari 2004.

[SAC-99] Sachs, Jonathan, 1996-1999, "*Digital Image Basics*".

[LIU-12] Liu, Hindra, 2012, "*Mendikbud Dukung Pemakaian CCTV di Sekolah*", online <http://edukasi.kompas.com/read/2012/04/18/06063582/Mendikbud.Dukung.Pemakaian.CCTV.di.Sekolah, diakses tanggal 26 Desember 2013>.

[LTG-99] Glossary, Lyco Tech.1999. *Video*.[online] <http://webopedia.lycos.com/Multimedia/Video/video.html> [diakses pada 9 Januari 2014]

[AGP-11] Attorney-General's Department. 2011. *"CCTV as a crime prevention measure.What is CCTV?*". [online] < http://www.crimeprevention.gov.au/>[diakses pada 9 Januari 2014]

[IPC-10] Inter-Pacific, Inc. 2010. *"What Is CCTV*". [online] < http://www.inter-pacific.com > [diakses pada 9 Januari 2014]

[UTK-10] Utkarsh. 2010. "SIFT*: Scale Invariant Feature Transform*".[online] <http://www.aishack.in/2010/05/sift-scale-invariant-feature-transform/>[diakses pada 11 Maret 2014]

[ABE-20] Abeer George Ghuneim. 2000. "Moore-Neighbor Tracing". [online]<http://www.imageprocessingplace.com/downloads_V3/root_downloads/tutorials/contour_tracing_Abeer_George_Ghuneim/moore.html>[diakses pada 24 Maret 2014]