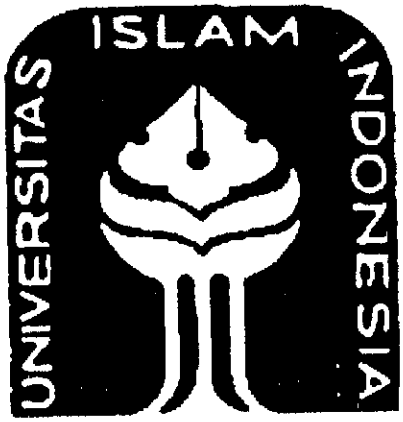
PENGEMBANGAN APLIKASI PENDETEKSIAN WAJAH

TUGAS AKHIR

Diajukan Sebagai Salah Satunya Syarat
Untuk Memperoleh Gelar Sarjana Teknik Informatika



Disusun oleh :

Faisal Wirakusuma

05523169

TEKNIK INFORMATIKA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

JOGJAKARTA

2010

LEMBAR PENGESAHAN PEMBIMBING

PENGEMBANGAN APLIKASI PENDETEKSIAN WAJAH

TUGAS AKHIR

Oleh:

Nama	**:**	**Faisal Wirakusuma**
No. Mahasiswa	**:**	**05523169**

Pembimbing,

Dr. Sri Kusumadewi, S.Si.,MT

LEMBAR PENGESAHAN PENGUJI

PENGEMBANGAN APLIKASI PENDETEKSIAN WAJAH

TUGAS AKHIR

Oleh:

Nama : **Faisal Wirakusuma**

No. Mahasiswa : **05523169**

Telah Dipertahankan di Depan Sidang Penguji Sebagai Salah Satu Syarat
Untuk Memperoleh Gelar Sarjana Jurusan Teknik Informatika
Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 30 Januari 2010

Tim Penguji	**Tanda Tangan**
Dr. Sri Kusumadewi, S.Si.,MT **Ketua**	
Yudi Prayudi, S.Si., M.Kom **Anggota I**	
Fathul Wahid, ST., M.Sc. **Anggota II**	

Mengetahui,
Ketua Jurusan Teknik Informatika
Universitas Islam Indonesia

(**Yudi Prayudi, S.Si., M.Kom**)

HALAMAN PERSEMBAHAN

بسم الله الرحمن الرحيم

Kupersembahkan Tugas Akhir Ini

Dengan Setulus Hati Untuk

Yang Tercinta:

Ayah (Mayangkoro)

dan Ibunda (Sri Pangestuti)

Adik-adikku (Tanti & Nindi)

Yang telah memberikan dorongan semangat, do'a serta dukungan material dan spiritual selama dalam menyelesaikan studi.

Untuk Almamater ku:

Jurusan Teknik Informatika, Universitas Islam Indonesia.

Untuk dunia pendidikan yang sangat kucintai....

HALAMAN MOTTO

"Who Dares Wins"

(Navy Seal)

"Man Jadda Wa Jadda"

(anonim)

"Whatever doesn't kill me, make me stronger"

(Friedrich Nietzche)

"Pantang Kembali Sebelum Tercapai Puncak Idaman"

(Mapala Unisi)

"Jika engkau bisa, jadilah seorang ulama. Jika engkau tidak mampu, maka jadilah penuntut ilmu. Bila engkau tidak bisa menjadi seorang penuntut ilmu, maka cintailah mereka. Dan jika kau tidak mencintai mereka, janganlah engkau benci mereka."

(Umar bin Abdul Aziz)

KATA PENGANTAR

بسم الله الرحمن الرحيم

Assalamu'alaikum Warohmatullahi Waharokatuh

Alhamdulillah, puji syukur kehadirat Allah SWT atas limpahan hidayah, taufiq, serta rahmat-Nya, sehingga laporan Tugas Akhir ini dengan judul "PENGEMBANGAN APLIKASI DETEKSI WAJAH" ini dapat terselesaikan sebagaimana mestinya. Shalawat serta salam semoga senantiasa tercurah atas Nabi Muhammad SAW, keluarga, sahabat, serta pengikutnya hingga hari kiamat nanti, Amin.

Laporan Tugas Akhir ini disusun sebagai salah satu syarat guna memperoleh gelar sarjana Teknik Informatika pada Universitas Islam Indonesia. Dan juga sebagai sarana untuk mempraktekan secara langsung ilmu dan teori yang telah diperoleh selama menjalani masa studi di Jurusan Teknik Informatika Fakultas Teknologi Industri Universitas Islam Indonesia.

Penulis menyampaikan ucapan terimakasih dan penghargaan setinggi-tingginya atas bantuan, bimbingan, dukungan dan do'a dari berbagai pihak yang ikut membantu demi kelancaran pelaksanaan Tugas Akhir ini. Untuk itu dengan segala kerendahan hati penulis mengucapkan terimakasih yang sebesar-besarnya kepada:

1. ALLAH SWT, sang pemberi hidup, guru segala guru, raja segala raja, pembimbing kehidupan, dan kepada-Nya semua pujian-pujian keindahan ditujukan.
2. Nabi Muhammad SAW, junjungan dan panutanku, yang telah mengangkat kita dari jaman penuh kegelapan menjadi kejayaan.

3. Segenap keluarga penulis, yaitu ayah dan ibu tercinta, Mayangkoro dan Sri Pangestuti, tanpa mereka tidak akan ada keberadaanku di dunia ini. Serta adik-adik penulis, segala doa dan kebaikan bagi mereka
4. Bapak Prof. Dr. Edy Suandi Hamid.M.Ec., selaku Rektor Universitas Islam Indonesia dan seluruh jajaran Rektorat Universitas Islam Indonesia.
5. Bapak Fathul Wahid, ST., M.Sc, selaku Dekan Fakultas Teknologi Industri Universitas Islam Indonesia. Terima kasih atas masukan dan motivasi selama ini.
6. Bapak Yudi Prayudi, S.Si., M.Kom, selaku Ketua Jurusan Teknik Informatika. Terima kasih atas kemudahan dan dukungan yang telah diberikan.
7. Ibu Dr. Sri Kusuma Dewi, ST.,M.Sc, selaku Dosen Pembimbing, dari beliaulah penulis banyak belajar, terimakasih tak terhingga atas segala pengarahan, bimbingan, motivasi serta masukan selama pelaksanaan tugas akhir dan penulisan laporan.
8. Teman-teman penulis, Willy, Mijil, Noor, Ida kecil, dan teman-teman lain dari Eiger Corps yang selalu memberikan semangat dan dukungannya tanpa henti sehingga tugas akhir ini dapat terselesaikan.
9. Seluruh penghuni Kost Putra Tunggal, Om Helmi, Mas Henri, Faiz, Jarot, Adi, Hendi, Mardiansyah yang juga turut menjadi model dalam pengetesan. Tanpa mereka kost akan sepi.
10. Seluruh saudara-saudaraku di MAPALA UNISI JOGJAKARTA, pantang kembali sebelum tercapai puncak idaman!!
11. Seluruh dosen dan staf pengajar Jurusan Teknik Informatika, terimakasih atas ilmu dan pengetahuan yang telah diberikan semoga menjadi amal jariyah.
12. Teman-teman Alien '05' terimakasih atas kerja sama, kekompakan, kekeluargaan dan persahabatan yang selama ini kalian berikan, serta seluruh teman-teman Informatika UII, jadilah ilmuan yang ber akhlaq Al-Qur'an.
13. Semua pihak yang tidak dapat penulis sebutkan satu persatu dalam membantu sejak pengumpulan data sampai penyusunan Tugas Akhir ini.

Semoga amal ibadah dan kebaikan yang telah diberikan mendapat imbalan yang setimpal dari Allah SWT.

Penulis menyadari bahwa Tugas Akhir ini masih jauh dari kesempurnaan, oleh sebab itu penulis sangat mengharapkan kritik serta saran yang bersifat membangun untuk perbaikan dimasa mendatang. Semoga Tugas Akhir ini dapat berguna dan memberikan manfaat bagi penulis serta semua pembaca.

Wassalamu'alaikum Warohmatullahi Wabarokatuh

Yogyakarta, 15 Januari 2010

Penulis

ABSTRAKSI

Teknologi pencitraan adalah salah satu disiplin dari ilmu komputer yang berkembang dengan pesat dan banyak diterapkan dalam kehidupan. Mulai dari pengambilan gambar untuk dokumentasi, kamera pengawas, analisa biomedis, pengambilan citra permukaan bumi, sistem biometrik, dan lain-lain. Dengan teknologi pencitraan, manusia menjadi dimudahkan atas segala urusan yang berhubungan dengan pengolahan dan pemrosesan citra.

Aplikasi deteksi wajah ini yang merupakan penelitian yang berasal dari bidang pencitraan, menggunakan banyak teori didalamnya yang mendukung jalannya aplikasi. Aplikasi ini dapat diterapkan pada penggunaan kamera pengawas, dimana disini kebutuhan akan tenaga manusia dan juga kebutuhan untuk menyimpan data kamera pada seluruh waktu dapat dikurangi, sehigga menjadi lebih efektif dan efisien. Dengan penerapan teori dan metode Viola-Jones, aplikasi ini mempunyai akurasi dan kecepatan proses yang cukup tinggi, dibantu dengan framework OpenFrameworks dan OpenCV.

Kata kunci : pengolahan citra, deteksi wajah, computer vision, viola-jones detector.

TAKARIR

Cascade : Blok-blok yang berisi keputusan, sejenis dengan pohon keputusan

Blur : Citra yang kabur, sebagai hasil operasi pada citra

Image processing : Pengolahan citra

Object-oriented programming : Pemrograman berorientasi objek

Framework : Pustaka yang berisi komponen-komponen yang dapat digunakan untuk membangun aplikasi

Pattern recognition : Pengenalan pola

Noise : Gangguan pada citra.

Grid : Blok-blok.

Computational photography : Pengolahan foto yang dibantu komputer

Morphing : Perubahan citra.

Pixel : Ukuran pada citra.

Instance : Suatu objek baru, sebagai hasil instansiasi dari suatu kelas.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1. Latar Belakang

Saat ini teknologi pencitraan menggunakan kamera telah banyak digunakan di berbagai bidang dalam kebutuhan manusia. Mulai dari pengambilan gambar untuk dokumentasi, kamera pengawas, analisa biomedis, pengambilan citra permukaan bumi, sistem biometrik, dan lain-lain. Namun sayangnya, selama ini untuk penggunaan teknologi pencitraan secara khusus pada kamera pengawas masih belum termanfaatkan secara maksimal, karena untuk mencapai tujuannya sendiri masih menggunakan bantuan tenaga manusia. Salah satu contoh penggunaan kamera pengawas, misalnya adalah untuk mengawasi suatu ruangan tertentu yang tidak boleh dimasuki oleh orang yang tidak berwenang. Disini tenaga manusia masih sangat dibutuhkan untuk mengamati monitor sebagai hasil keluaran dari kamera pengawas, dan objek mencurigakan apa saja yang telah memasuki wilayah kamera pengawas tersebut. Hal ini sangat tidak efisien, karena kecenderungan manusia akan jenuh terhadap pekerjaan yang berulang-ulang, dan ketika telah mencapai titik jenuh dan lelah, maka ketelitian dan keakurasiannya akan berkurang. Terkadang manusia pun bisa lalai atau lupa. Untuk mencari tahu siapa saja yang telah memasuki wilayah tersebut di waktu tertentu pun, masih menggunakan tenaga manusia untuk memutar ulang video hasil rekaman dan menelusuri siapa saja yang tertangkap kamera.

Computer Vision adalah transformasi data dari gambar diam atau video kamera menjadi suatu keputusan atau gambaran baru yang berbeda. Data yang dimasukkan mungkin mengandung sesuatu informasi seperti jenis suatu mobil, atau jarak dari suatu objek ke objek lainnya adalah beberapa meter. Keputusan tersebut seperti ada seseorang dalam gambar yang ditangkap, atau dari citra dari mata yang diambil, seseorang dapat dikenali penyakitnya. Sedangkan yang disebut gambaran atau representasi baru seperti mengubah suatu gambar yang berwarna menjadi suatu gambar dengan warna dasar abu-abu saja, atau mengubah

gambar menjadi *blur*. Itu hanyalah sebagian kecil dari manfaat *computer vision*, dan sebenarnya *computer vision* telah banyak diaplikasikan ke dalam kehidupan sehari-hari. Teknologi *computer vision* banyak digunakan dalam kamera pengawas, pengambilan citra permukaan bumi, pesawat tidak berawak, sistem biometrika, analisa biomedik, proses manufaktur, dan banyak lagi proses-proses lain yang menggunakan *computer vision*.

Secara sederhana, proses dalam *computer vision* adalah pengambilan data dari masukan yang berupa gambar, dan kemudian diolah menjadi suatu hasil tertentu sesuai kebutuhan yang telah ditentukan dalam perancangan sistem *computer vision* itu sendiri. Kemudian dalam proses pengolahan hasil keluarannya, menggunakan suatu algoritma menjadi suatu keputusan tertentu. Pemanfaatan *computer vision* yang cukup banyak dipakai adalah pengolahan dari citra video kamera untuk mendeteksi objek. Hal ini disebabkan harga kamera yang semakin murah, resolusi yang ada pada kamera-kamera tersebut juga semakin tinggi dan mendukung keakurasian citra yang diambil. Algoritma yang dapat digunakan untuk mengolah citra pun bermacam-macam, sesuai dari kebutuhan penggunaannya.

Pemanfaatan *computer vision* untuk mengambil citra wajah manusia sudah banyak diterapkan, dan pemanfaatannya pun bermacam-macam. Disini penulis bertujuan membuat sistem pengolahan citra untuk mendeteksi wajah manusia dari citra yang diambil oleh kamera, melihat kebutuhan akan sistem yang dapat mengenali citra dari wajah manusia, dan untuk mengurangi ketidakefisienan dalam penggunaan tenaga manusia. Pendeteksian citra wajah nantinya dapat digunakan untuk pengoptimalan gambar diam dalam kamera digital, kamera pengawas yang dapat mendeteksi wajah orang tertentu, dan juga untuk hiburan semisal *photobox*. Dari hasil pendeteksian wajah manusia ini diharapkan dapat dikembangkan lagi menjadi bermacam-macam pemanfaatan.

1.2. Rumusan Masalah

Rumusan masalah dari penelitian :

a. Kurang efisiennya penggunaan tenaga manusia untuk mengamati objek tertentu yang diambil dari citra kamera.
b. Kebutuhan akan aplikasi pencitraan yang dapat mengenali wajah manusia.
c. Kebutuhan aplikasi yang membutuhkan keakurasian tinggi untuk mengenali objek.
d. Bagaimana membangun aplikasi yang dapat mengenali objek, yang khususnya wajah manusia.

1.3. Batasan Masalah

Batasan masalah untuk penelitian ini adalah :

1. Objek wajah yang akan dikenali harus lengkap dan jelas.
2. Asumsi wajah yang bisa dikenali tidak memakai penutup, seperti cadar atau topeng.
3. Citra wajah yang ditangkap harus lengkap, yaitu keseluruhan dari elemen wajah harus dapat tertangkap dan tidak dari samping, atas, bawah atau belakang.
4. Penelitian ini tidak mencakup hingga pengenalan wajah seseorang atau *face recognition*.
5. Wajah yang dideteksi harus tegak lurus, dan tidak miring.

1.4. Tujuan Penelitian

Tujuan penelitian ini adalah membuat suatu aplikasi yang dapat memindai suatu tampilan citra dari kamera dan dapat mengenali wajah manusia dari sekian banyak objek yang ada.

1.5. Manfaat Penelitian

Manfaat dari penelitian ini adalah

1. Dapat digunakan dalam sistem kamera pengawas, sehingga ketika ada orang yang memasuki ruangan yang tidak seharusnya dimasuki, sistem dapat mendeteksinya
2. Dapat menyimpan citra wajahnya untuk tindakan lebih lanjut lagi, seperti pengenalan wajah seseorang.

1.6. Metodologi Penelitian

Dalam metode penelitian ini terdapat beberapa langkah-langkah yang dilakukan, yaitu:

1. Identifikasi masalah

Pada metode ini akan diidentifikasi keseluruhan masalah yang akan dipecahkan menggunakan aplikasi yang akan dibuat. Dari masalah inti, kemudian akan dipecahkan bagian per bagian sehingga dapat dibuat logika pemecahan masalahnya.

2. Analisis kebutuhan sistem

Metode ini bertujuan untuk menganalisis apa saja yang dibutuhkan dalam pengembangan aplikasi pendeteksian wajah ini, termasuk kebutuhan hardware dan softwarenya.

3. Penentuan algoritma pengenalan yang akan dipakai

Disini akan dilakukan penelitian algoritma pengenalan wajah apa yang akan dipakai dalam aplikasi, sehingga sistem dapat berjalan dengan seoptimal mungkin. Penelitian algoritma ini dilakukan dengan mencari literatur yang berkaitan dengan pembelajaran mesin dan *image processing.*

4. Perancangan

Metode perancangan dari keseluruhan sistem menggunakan UML, karena metode yang akan dipakai adalah *Object Oriented Programming.*

5. Implementasi sistem menggunakan C++

Setelah semua perancangan sistem selesai, maka akan dilakukan implementasi sistem dengan menggunakan framework *openFrameworks* dan *OpenCV* yang berbasis C++.

1.7. Sistematika Penulisan

Sistematika penulisan secara keseluruhan diuraikan sebagai berikut : Bab I Pendahuluan, bab ini membahas tentang latar belakang masalah, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, metodologi penelitian dan sistematika penelitian. Bab II Teori Computer Vision, membahas mengenai teori-teori dasar *Computer Vision* yang digunakan sebagai dasar untuk pembahasan masalah. Bab III Algoritma Haar, membahas tentang dasar teori Algoritma Haar, yang digunakan untuk mengolah hasil citra. Bab IV Gambaran Umum Sistem, membahas tentang gambaran umum dari keseluruhan sistem yang akan dibuat. Bab V Perancangan Sistem, menguraikan tentang perancangan sistem menggunakan UML. Bab VI Implementasi Sistem, membahas implementasi perangkat lunak. Bab VII Pengujian Sistem, menjelaskan tentang proses setelah implementasi yaitu pengujian sistem. Bab VIII Penutup, menguraikan kesimpulan dari tugas akhir serta dikemukakan beberapa saran untuk dilaksanakan lebih lanjut guna pengembangan penelitian tugas akhir ini.

BAB II
TEORI COMPUTER VISION

2.1. Computer Vision

Computer vision adalah teknologi yang berkembang dengan sangat pesat dan telah banyak digunakan dalam kehidupan sehari-hari manusia mulai dari hal yang sederhana, sampai yang tidak biasa seperti UAV (pesawat tidak berawak). *Computer vision* adalah teknologi yang membuat komputer mampu melihat. Teori *computer vision* berasal dari teori kecerdasan buatan, yaitu mengambil informasi dan data dari citra yang diambil dengan menggunakan suatu algoritma. Data citra itu sendiri dapat berupa berbagai bentuk, tidak hanya berasal dari video kamera atau citra fotografi mentah, tetapi juga dari sensor panas atau pemindai medis. Tujuan dari bidang studi *computer vision* adalah memprogram komputer agar "memahami" sebuah gambaran atau ciri dari suatu citra. Secara lebih spesifik, tujuan dari penelitian *computer vision* termasuk pendeteksian dari suatu objek tertentu dari suatu citra, evaluasi dari hasil pengambilan citra, melacak objek dari suatu urutan citra, memetakan sebuah gambaran dalam citra menjadi model tiga dimensi, dan lain-lain. Tujuan-tujuan ini dapat dicapai dengan metode *pattern recognition,* pembelajaran stastitikal, *projective geometry,* pengolahan citra, dan teori-teori lain dalam *computer vision.* Tujuan lain dari *computer vision* adalah menghasilkan suatu keputusan dari objek yang nyata secara fisik dan suatu gambaran pengindraan dari suatu citra. Beberapa ahli berpendapat, tujuan dasar dari *computer vision* adalah membangun sebuah deskripsi dari sebuah citra, karena untuk menghasilkan suatu keputusan dari objek nyata, adalah suatu hal yang penting untuk membangun suatu deskripsi dari citra yang ditangkap. Studi *computer vision* termasuk didalamnya pengindraan dari objek, menerjemahkan informasi atau data yang dimasukkan ke dalam komputer, dan algoritma yang digunakan untuk memproses informasi citra (Shapiro dan Stockman, 2000:13). Sedangkan menurut definisi lain (Gonzalez dan Woods, 2001:2), tujuan utama dari studi *computer vision* adalah berusaha menyamai atau melebihi kemampuan

penglihatan manusia, termasuk didalamnya pembelajaran dan kemampuan untuk menarik kesimpulan serta mengambil aksi berdasarkan masukan visual.

Contoh dari penerapan *computer vision* adalah pencitraan medis yang telah banyak digunakan di rumah sakit-rumah sakit yang bernilai jutaan US dollar per tahun. Contoh lain dari *computer vision* digunakan di sistem-sistem berikut:

- Pengendalian proses
- Pendeteksian event/kejadian
- Pengorganisasian informasi
- Pemodelan objek atau lingkungan sekitar
- Interaksi
- Sistem informasi geografis

Penelitian tentang *computer vision* telah dimulai pada tahun 1970, dimana pada saat itu ada suatu penelitian yang bertujuan untuk meniru kecerdasan manusia dan memberikan robot kecerdasan untuk beberapa tujuan (Szeliski, 2009:11). Tetapi terkadang manusia tertipu dengan persepsinya tentang kemampuannya untuk melihat sesuatu, jadi orang sering berpikir adalah hal yang mudah untuk membuat sebuah aplikasi *computer vision* yang mampu menjelaskan apa yang dilihatnya (Szeliski, 2009). Setelah melalui banyak penelitian, akhirnya para ahli menemukan masalah-masalah yang ada dari studi *computer vision*. Walaupun studi *computer vision* telah dimulai sejak tahun 1970, bidang *computer vision* bisa disebut masih belum matang dan masih terlalu luas. Karena pada saat pengembangan awalnya, belum ada formula standar untuk memecahkan problem dalam *computer vision* dan metode bagaimana masalah tersebut diselesaikan.

Masalah yang membuat penerapan *computer vision* menjadi sulit adalah keterbatasan dari perangkat keras yang dipakai untuk mengambil citra dari objek di lapangan. Seringkali objek yang ditangkap terganggu oleh *noise* dari lingkungan sekitar objek. Untuk menyingkirkan *noise* tersebut, para ahli menggunakan algoritma tertentu yang dapat mengekstrak objek yang diinginkan, atau hanya meminimalisir *noise*.

Manusia dapat tertipu oleh kemampuan penglihatannya sendiri, tanpat mempertimbangkan proses yang sangat rumit dibaliknya. Otak manusia membagi sinyal

penglihatan menjadi banyak kanal yang menyalurkan informasi-informasi yang berbeda dan bermacam-macam. Otak manusia mempunyai sistem yang dapat mengidentifikasi bagian penting dari suatu citra untuk diperiksa dengan tanpa terganggu oleh citra objek lain. Hal ini sama dengan ketika manusia ingin fokus terhadap sesuatu. Bagaimanapun juga, dalam sistem *computer vision*, komputer menerima suatu *grid* yang berisi angka-angka dari kamera atau harddisk. Pada sebagian besarnya, tidak ada pengenalan objek yang bersifat *built-in*, tidak ada kendali otomatis atas fokus dan lobang lensa, tidak ada asosiasi silang dari pengalaman selama bertahun-tahun, dan ini yang membedakan dengan sistem penglihatan manusia. Dapat dikatakan bahwa sistem ini masih sangat naif. Dari angka-angka yang dikirimkan oleh perangkat keras perekam citra ke dalam komputer, baru dapat digunakan teknik-teknik dari *computer vision* untuk diolah menjadi suatu data atau informasi yang berguna (Bradski dan Kaehler, 2008:2-3).

Permasalahan lain dari *computer vision* adalah pose dari objek yang ditangkap, yang umumnya sangat sulit untuk dipecahkan. Disini belum ada suatu cara unik untuk membangun kembali sinyal tiga dimensi yang ditangkap, dan belum ada solusi secara definitif untuk memecahkan problem pose ini (Bradski dan Kaehler, 2008). Selain itu, data yang didapat seringkali menjadi rusak karena *noise* dan distorsi. Kerusakan ini muncul karena variasi seperti cuaca, pencahayaan, refleksi, dan pergerakan. Ada juga masalah lain seperti ketidaksempurnaan lensa dan pengaturan mekanisnya, dan *noise* elektrikal dari sensor. Halangan-halangan ini merupakan tantangan besar dari pengembangan *computer vision*.

Seperti yang dijelaskan Szeliski, ada 3 tingkatan deskripsi dari sistem pemrosesan informasi visual :

- Teori komputasi : Apakah tujuan dari tugas komputasi dan batasan-batasan apa yang telah diketahui atau dapat dibawa pada pemecahan masalah.
- Representasi dan algoritma : Bagaimana representasi dan algoritma dapat dipetakan ke dalam perangkat keras aktual seperti sistem penglihatan biologikal atau kepingan silikon? Sebaliknya, bagaimana batasan pada perangkat lunak dapat digunakan untuk memilih representasi dan algoritma? Dengan meningkatnya penggunaan chip grafis dan banyak arsitektur inti

untuk digunakan pada *computer vision*, pertanyaan ini menjadi relevan pada pengembangan *computer vision.*

Computer vision sendiri adalah suatu bidang yang sangat luas dan mempunyai banyak aspek, dan teknik-teknik yang dapat digunakan bermacam-macam, tergantung permasalahan dan tujuan pembuatan aplikasi *computer vision.* Aspek yang termasuk dalam studi *computer vision* adalah :

a. *Image Processing*
b. *Image Formation*
c. *Image Segmentation*
d. *Motion Detection and Tracking*
e. *Feature Detection and Matching*
f. *Projective Geometry*
g. *Image Stitching and blending*
h. *Optical Flow and Tracking*
i. *Object Recognition*
j. *Stereo Matching*
k. *Multi-view Stereo and 3D Modelling*
l. *Image-based Rendering*
m. Dan lain-lain

Sesuai relevansi pada penelitian ini, yang akan dijelaskan hanya mencakup pengolahan citra (*image processing*) dan pengenalan objek.

2.2. Pengolahan Citra

Pengolahan Citra atau *image processing* adalah suatu ilmu untuk memanipulasi suatu citra (Crane, 1997:1). Ilmu ini mencakup bidang yang sangat luas dari teknik yang telah ada banyak dipakai pada aplikasi-aplikasi saat ini. Teknik-teknik ini beragam, mulai dari mengubah suatu citra, menyoroti suatu bagian tertentu dari citra, membuat citra baru dari bagian suatu citra lain, memperbaiki citra yang telah turun kualitasnya selama pengambilannya atau setelahnya. Studi pada pengolahan citra sendiri bermula dari dua area aplikasi : perbaikan dari informasi

yang berasal dari citra untuk kepentingan penafsiran manusia,dan pemrosesan data citra untuk penyimpanan,transmisi,dan representasi yang ditujukan untuk penglihatan mesin (Gonzalez dan Woods, 2001).

Tahap pertama dari sebagian besar aplikasi *computer vision* sendiri adalah penggunaan teknik-teknik pengolahan citra untuk melakukan pra-pemrosesan citra dan mengkonversinya menjadi bentuk yang memungkinkan untuk analisa lebih jauh lagi. Salah satu contoh dari operasi pengolahan citra adalah koreksi pencahayaan dan penyeimbangan warna, reduksi *noise*, peningkatan ketajaman, atau melakukan rotasi pada citra. Ketika beberapa menganggap bahwa dalam studi pengolahan citra tidak berhubungan dengan studi *computer vision*, justru sebagian aplikasi *computer vision* seperti *computational photography* atau bahkan pengenalan objek memerlukan desain yang cermat dari penggunaan teknologi pengolahan citra untuk mendapatkan hasil yang maksimal (Szeliski, 2009).

Pengolahan citra sering disamakan dengan komputer grafis, karena komputer grafis dan pengolahan citra adalah dua teknologi yang saling berkaitan.Akan tetapi, walaupun disini banyak kesamaan konsep antara pengolahan citra dan komputer grafis, keduanya adalah dua studi yang berbeda. Komputer grafis adalah penciptaan citra yang murni buatan, atau bisa disebut sintetis. Sedangkan pengolahan citra adalah manipulasi dari citra yang sudah ditangkap atau direkam sebelumnya. Komputer grafis bekerja dengan objek 2 dimensi dan 3 dimensi. Pengolahan citra biasanya bekerja dengan gambar 2 dimensi, walaupun tidak terbatas pada itu saja. Aplikasi pada industri diagnosa medis melakukan pengolahan citra pada data 3 dimensi. Dengan munculnya beberapa teknologi seperti "*volume visualization*" dan "*morphing*", garis pembatas antara komputer grafis dan pengolahan citra menjadi kabur. Sebuah citra didefinisikan sebagai fungsi dua-dimensi, yaitu *f(x,y)*, dimana x dan y adalah koordinat spasial, dan amplitudo dari *f* atas pasangan koordinat (x,y) apa saja disebut sebagai intensitas atau *gray level* (tingkatan abu-abu) dari citra pada poin tersebut. Ketika x,y, dan nilai amplitudo dari *f* secara keseluruhan adalah bilangan terbatas, kuantitas diskrit, maka gambar tersebut disebut gambar digital. Perlu dicatat bahwa citra digital terdiri dari sejumlah elemen terbatas, dimana setiap elemennya mempunyai lokasi dan nilai tertentu. Elemen-elemen ini merujuk pada

elemen citra, elemen gambar, dan juga *pixel*. Disini ada empat klasifikasi dari algoritma pengolahan citra : proses poin, proses area, proses geometrik, dan proses frame. Suatu proses poin memodifikasi sebuah nilai *pixel* berdasar pada posisi atau nilai awal dari *pixel* tersebut. *Pixel* adalah istilah yang banyak dipakai untuk menotasikan elemen dari citra digital. Suatu proses area memodifikasi nilai dari *pixel* berdasar dari nilai aslinya dan juga nilai dari *pixel* sekitarnya. Proses geometrik mengubah posisi atau susunan dari *pixel*. Suatu proses frame menciptakan nilai *pixel* berdasar dari operasi pada 2 atau lebih citra. Seringkali, efek yang diinginkan membutuhkan aplikasi dari beberapa proses yang berbeda. Sebuah alat yang cukup berguna untuk menganalisa dan kadang juga mempercepat operasi berbasis area atau proses area adalah algoritma Fourier Transform. Operasi berbasis area dapat mengolah data menjadi piramid citra (*image pyramids)* dan gelombang (*wavelets*), yang mana merupakan hal yang sangat penting untuk menganalisa citra dengan resolusi atau skala yang berbeda-beda dan untuk mempercepat beberapa operasi. Kelas penting lain dari operasi global image processing adalah transformasi geometrik seperti rotasi. Optimisasi global adalah salah satu pendekatan dari pengolahan citra, dimana operasi ini melibatkan estimasi optimal menggunakan model Markov Random Field dari Bayesian.

Penglihatan adalah bagian paling maju dari indera manusia, jadi tidak mengherankan jika citra memainkan peranan paling penting dalam pemahaman manusia. Bagaimanapun juga, tidak seperti manusia yang terbatas pada spektrum pita visual elektromagnetik (EM), mesin-mesin pencitraan mencakup sebagian besar spektrum elektromagnetik, mulai dari sinar gamma sampai gelombang radio. Mesin-mesin ini mampu beroperasi pada citra yang diciptakan oleh sumber dimana manusia tidak biasa berhubungan dengan citra-citra seperti ini. Dalam hal ini termasuk didalamnya adalah suara ultrasonik, mikroskop elektron, dan citra yang diciptakan komputer. Dengan demikian, pengolahan citra mencakup bidang aplikasi yang sangat beragam. Contoh aplikasi dari pengolahan citra :

a. Sains dan ruang angkasa

Teknik rekonstruksi dan filtering membuang "*noise*" dan mengembalikan bagian yang hilang dari citra satelit dan pesawat luar angkasa.

b. **Film-film**

Pada dunia perfilman, teknik pengolahan citra digunakan untuk mengubah suatu citra menjadi citra lain, membuang objek tertentu dari frame, dan membuat frame baru dengan mengkomposisikan bagian-bagian dari frame yang lain.

c. **Pengkomposisian Citra**

Pengkomposisian citra atau image composition juga digunakan pada film "Forrest Gump". Pemain utamanya telah ditambahkan pada segmen pada beberapa film dokumentasi sejarah, yang didalamnya banyak memunculkan presiden Amerika.

d. **Industri medis**

Industri medis telah lama menjadi pengguna dari pengolahan citra. Disini ada beragam teknologi pencitraan yang digunakan termasuk didalamnya adalah X-ray dan ultrasonik. Tomografi berbantu komputer (CT atau CAT), tidak banyak dipakai sampai tahun 1980-an. Dokter-dokter menggunakan pemindai CT untuk mendiagnosa bagian dari tulang seperti tulang belakang,tengkorak,dan tulang panggul. Magnetic Resonance Imaging (MRI) menghasilkan gambaran-gambaran dari jaringan lembut seperti syaraf tulang belakang dan juga jantung. Positron Emission Tomography (PET) menunjukkan pengukuran dari proses fisiologi dan kimia tubuh. Magnetic Source Imaging (MSI) mengamati sinyal elektris di dalam otak ketika berpikir dan melakukan fungsi motorik. Semua hasil pencitraan ini ketika telah disimpan di komputer dapat ditingkatkan dan dimanipulasi, sehingga dokter bisa berfokus pada area-area yang penting.

Teknik-teknik pengolahan citra juga banyak digunakan di bidang-bidang lain seperti fotografer, agen periklanan, dan penerbit. Dalam bidang pemetaan juga banyak digunakan teknologi pengolahan citra.

2.3. Pengenalan Pola dan Pengenalan Objek

Pengenalan objek merupakan salah satu permasalahan tersulit dalam studi *computer vision*. Sampai sekarang, belum ada algoritma serbaguna yang memungkinkan pembelajaran otomatis dari objek 3 dimensi yang berubah-ubah dan juga pengenalannya dan lokalisasi dari sebuah *scene* yang kompleks. Pendekatan untuk menangani tugas-tugas *computer vision* tingkat tinggi pada dasarnya

didominasi oleh metode pengenalan objek berdasarkan model (Hornegger,et. al., 2000).

Pada masa lalu, banyak aplikasi telah menghasilkan banyak tipe dari representasi model objek yang memungkinkan implementasi dari sistem pengenalan objek yang sangat bagus. Banyak algoritma pengolahan citra dan *computer vision* yang digolongkan atas solusi yang berlaku secara ad-hoc. Kebanyakan teknik ini menerapkan ide-ide intuitif yang secara spesifik diperuntukan aplikasi tertentu dan mengabaikan eksplorasi dari definisi model matematika yang akurat. Disini tidak ada teori formalisasi tunggal yang menyediakan *framework* untuk analisis analitikal dari keseluruhan sistem yang telah didesain. Untuk sebagian besar, hanya tinggal studi empirikal yang dapat membenarkan pemakaian pola representasi tertentu (Hornegger,et. al., 2000).

Dari semua tugas visual yang komputer lakukan, menganalisa suatu *scene* dari citra dan mengenali objek-objek tertentu dalam citra tersebut adalah hal yang masih menantang. Ketika komputer mencapai keakurasian pada saat membuat model 3 dimensi dari suatu bentuk sebuah adegan tertentu yang diambil dari sudut pandang yang berbeda-beda, komputer tidak dapat menyebutkan nama dari semua objek dan binatang yang muncul pada suatu gambar yang berusia 2 tahun, misalnya. Disini bahkan tidak ada suatu persetujuan diantara peneliti tentang bagaimana tahapan pelaksanaan ini dapat tercapai.

Pengenalan objek merupakan teknik yang sangat sulit, hal ini dikarenakan dunia nyata tersusun dari kumpulan objek yang campur aduk, dimana yang satu bercampur dengan yang lainnya dan tampil dengan berbagai pose. Selanjutnya, keragaman intrinsik dalam suatu kelas, seperti anjing misalnya, berdasar pada artikulasi kompleks yang tidak rigid dan variasi ekstrim dari bentuk dan rupa diantara ras yang berbeda-beda, membuatnya sangat sulit untuk dilakukan pencocokan secara mendalam dari basis data objek.

Permasalahan pengenalan dapat dipecah menjadi beberapa tahap. Contohnya, ketika sudah ditemukan apa yang akan dicari, maka permasalahan yang ada adalah deteksi objek, dimana hal ini melibatkan pemindaian cepat sebuah citra untuk menebak dimana kecocokan dapat muncul (Szeliski, 2009). Jika telah ada suatu objek

spesifik yang rigid, maka disini dapat dicoba dilakukan pengenalan, atau biasa disebut *instance recognition*, dimana disini dapat dicari poin dari ciri karakteristik dan kemudian dilakukan verifikasi bahwa pose dari citra yang ada telah cocok secara geometris.

Hal yang paling menantang dari pengenalan objek adalah pengenalan objek secara keseluruhan, dimana hal ini melibatkan pengenalan *instance* dari objek yang bervariasi secara ekstrim, seperti binatang misalnya (Szeliski, 2009). Teknik mungkin yang bergantung secara murni kepada kemunculan objek (biasa dikenal sebagai model algoritma *'bag of words'*), posisi relatif objek-objek tersebut(model *'part-based'*), atau bisa juga melibatkan segmentasi dari suatu citra menjadi suatu bagian-bagian yang mempunyai arti secara semantik. Di seluruh *instance* yang ada, pengenalan sangat bergantung pada konteks objek-objek yang mengelilinginya dan juga elemen-elemen dari adegan citra tersebut (Szeliski, 2009).

Sebagian besar permasalah computer vision telah cocok pada pola standar dari permasalahan pengenalan objek seperti klasifikasi dan regresi. Sebuah citra digital f secara matematis dikenali sebagai suatu matrix dari nilai intensitas diskrit $f=[fm, n]\ 1\leq m\leq M,\ 1\leq n\leq N$. Untuk pengolahan lebih lanjut, sebagian pekerjaan computer vision tingkat tinggi memerlukan pelabelan atau segmentasi dari citra-citra yang diolah (Hornegger,et. al., 2000). Berdasarkan hasil label-label dan segmentasi ini, pengenalan dan estimasi pose sebagai parameter objek harus diselesaikan. Penyelesaian dari pengenalan objek dan permasalah estimasi pose dapat menggunakan *framework* probabilistik.

Ada 2 metode untuk melakukan pengenalan, yaitu deteksi suatu objek dengan menggunakan pencocokan pola dan pendeteksian bentuk menggunakan teknik transformasi Hough (Ritter dan Wilson, 1996). Salah satu metode yang paling pokok dari pendeteksian suatu objek adalah dengan pencocokan pola yang menggunakan acuan. Pada pencocokan acuan, replika dari suatu objek yang akan dicari dibandingkan dengan seluruh objek di dalam citra. Jika pencocokan pola antara acuan dan objek dalam citra cukup mendekati syaratnya, seperti melampaui ambang

yang telah diberikan, maka objeknya dapat digolongkan sama seperti objek pada acuan.

Transformasi Hough menyediakan metode yang serbaguna untuk mendeteksi bentuk yang dapat digambarkan sebagai suatu persamaan parametrik tertutup atau didalam bentuk tabular. Contoh dari bentuk yang dapat dijadikan sebagai parameter adalah garis, lingkaran, dan elips. Bentuk yang gagal dimasukkan ke dalam persamaan parametrik dapat dideteksi dengan menggunakan versi umum dari transformasi Hough yang menggunakan teknik dengan mengunakan tabel (Ritter dan Wilson, 1996).

Salah satu penerapan pengenalan objek adalah pada pengenalan wajah, yang menggunakan model *PCA(Principal Component Analysis)* dan pendekatan *Bayesian* untuk melakukan pengenalan dan klasifikasi.

1) Face Recognition

Dari semua tugas pengenalan objek komputer lakukan, pengenalan wajah adalah salah satu hal dimana tugas ini umumnya mempunyai kesuksesan tinggi untuk dikerjakan. Ketika komputer tidak dapat mengenali wajah seseorang dari ribuan wajah yang tertangkap video kamera (walaupun manusia juga belum bisa mengetahui wajah seseorang jika mereka tidak familiar dengannya), kemampuan komputer untuk membedakan seseorang dari sedikit objek, telah banyak digunakan pada aplikasi foto seperti Picasa dan iPhoto. Pengenalan wajah juga banyak digunakan pada beragam aplikasi, termasuk didalamnya Interaksi Manusia dan komputer (*HCI*), pengenalan identitas, login desktop, kontrol parental, dan juga monitoring pasien pada industri medis (Szeliski, 2009).

Suatu sistem pengenalan wajah mengidentifikasi wajah di dalam suatu citra dan video secara otomatis menggunakan komputer. Sistem ini mencakup beragam aplikasi, seperti aplikasi otentikasi biometrik, dan sistem kamera pengawas, interaksi manusia dan komputer, dan manajemen multimedia. Sistem pengenalan wajah umumnya terdiri atas empat bagian, yaitu : pendeteksian wajah, pensejajaran wajah, ekstraksi fitur pada wajah, dan klasifikasi wajah. Pendeteksian wajah menyediakan informasi tentang lokasi dan ukuran pada setiap wajah yang terdeteksi. Pada kasus video, wajah yang ditemukan memungkinkan untuk dapat

dilacak. Pada pensejajaran wajah, komponen pada wajah, seperti mata, hidung, dan mulut serta garis tepi pada wajah telah diketahui lokasinya, dan dari situ masukan citra wajah dinormalisasi secara geometris dan fotometris. Pada ekstraksi fitur, fitur yang dikira cukup berguna untuk membedakan orang satu dengan yang lainnya dipisahkan dari citra wajah yang telah mengalami normalisasi. Pada klasifikasi wajah, fitur vektor yang telah diekstrak dari masukan citra wajah dicocokkan dengan wajah yang telah didaftarkan sebelumnya ke dalam basis data, yang kemudian memberikan keluaran identitas dari wajah ketika data yang ada cocok dengan database (Li dan Lu, 2003).

Sistem pengenalan wajah saat ini baru bekerja secara maksimal ketika masukannya adalah wajah yang sepenuhnya menghadap ke depan dalam pencahayaan yang relatif sama,walaupun database yang ada sudah mengandung data pose dan pencahayaan yang sangat beragam juga (Szeliski, 2009).

Permasalahan ini dapat diselesaikan dengan menggunakan pengenalan pola dan pembelajaran mesin. Disini ada dua inti permasalahan: pertama, yaitu fitur-fitur apa sajakah yang digunakan untuk merepresentasikan suat wajah, dan kedua yaitu bagaimana mengklasifikasikan suatu citra wajah baru berdasar pada representasi yang telah dipilih. Suatu sistem pengenalan wajah yang cukup memadai seharusnya dapat mengatasi permasalahan yang sangat bervariasi dari citra wajah pada sudut pandang, pencahayaan, ekspresi wajah, dan lain-lain.

Pendekatan dengan menggunakan fitur geometris adalah cara yang berdasar pada framework computer vision yang tradisional. Fitur wajah seperti mata, hidung, mulut, dan dagu kemudian dideteksi. Properti dan relasi seperti area, jarak, dan sudut diantara fitur wajah digunakan sebagai deskripsi wajah untuk keperluan pengenalan. Dengan menggunakan pendekatan ini, Kanade membangun sistem pengenalan wajah yang pertama di dunia. Keunggulan dari sistem ini adalah pada sisi ekonomisnya dan efisiensi pada penerimaan data reduksi dan minimnya sensitifitas pada pencahayaan dan sudut pandang. Kelemahan sistem ini sejauh ini, pendeteksian fitur wajah dan teknik pengukuran yang dikembangkan pada saat itu belum dapat diandalkan, dan informasi geometrik yang ada kurang mencukupi untuk pengenalan wajah (Li dan Lu, 2003).

Beberapa pendekatan yang cukup menarik pada teknik pengenalan wajah melibatkan penemuan lokasi dari beberapa fitur citra seperti mata, hidung, dan mulut, dan kemudian mengukur jarak diantara lokasi fitur-fitur ini. Pendekatan-pendekatan lain mengandalkan pembandingan tingkatan abu-abu dari citra yang diproyeksikan kepada ruang dimensional yang lebih rendah disebut *eigenfaces*, dan secara tergabung memodelkan bentuk dan kemunculan variasi menggunakan *'active appearance models'*.

2) Object Detection

Selama bertahun-tahun, telah dikembangkan beragam algoritma pendeteksian wajah yang mempunyai performa cukup cepat. Menurut referensi dari Yang et al., teknik deteksi wajah dapat diklasifikasikan sebagai teknik berdasar-ciri, berdasar-*template* (acuan), atau berdasar-rupa/tampilan. Teknik berdasar-ciri mencoba untuk menemukan lokasi dari ciri citra yang dapat dibedakan seperti mata, hidung, dan mulut, dan kemudian memverifikasi apakah ciri-ciri ini tersusun dengan benar secara geometris. Termasuk di dalam teknik-teknik ini adalah pendekatan awal pada teknik pengenalan wajah, sama seperti pendekatan lainnya yang berdasarkan *eigenspace modular*, *local filter jets*, dan *support vector machines*, dan *boosting*. Pendekatan pada teknik berdasar-acuan, seperti *Active Appearance Models*, dapat menangani beragam pose dan ekspresi yang bervariasi, tetapi biasanya memerlukan inisialisasi yang baik di dekat wajah sebenarnya, maka dari itu teknik ini tidak cocok untuk diterapkan pada aplikasi yang membutuhkan pendeteksian wajah secara cepat.

Teknik berdasar-rupa/tampilan menggunakan pendekatan melalui pemindaian potongan-potongan kecil berbentuk kotak yang saling menutupi dari citra yang akan diperiksa, dimana hasilnya kemudian akan diperbaiki menggunakan teknik cascade dari algoritma-algoritma deteksi yang lebih mahal, tetapi lebih selektif dan akurat. Kebanyakan dari pendekatan berbasis-rupa/tampilan sekarang lebih mengandalkan pada klasifikasi pelatihan menggunakan suatu set dari citra wajah terlabel dan label yang bukan wajah.

Sung and Poggio dan Rowley et al., menunjukkan dua pendekatan paling awal dari sistem deteksi wajah berdasar-tampilan/rupa, dan memperkenalkan beberapa inovasi yang kini banyak dipakai pada pekerjaan-pekerjaan selanjutnya

oleh peneliti lain. Untuk memulai, kedua sistem mengumpulkan suatu set dari potongan wajah yang telah terlabeli dan juga set potongan citra yang berisi objek bukan wajah. Dari citra wajah yang dikumpulkan ini kemudian dilakukan operasi tambahan seperti pembalikan/pencerminan, rotasi, scaling, dan translasi citra dengan sedikit perubahan untuk membuat sistem deteksi wajah yang ada tidak terlalu sensitif dengan efek-efek tersebut.

Setelah beberapa kali citra untuk pembelajaran didapatkan, dapat dilakukan beberapa pra-pemrosesan, seperti membuang rerata gradien (dari fungsi linear) dari citra untuk dapat mengimbangi efek-efek yang berbeda-beda, dan menggunakan persamaan histogram untuk mengimbangi kontras kamera yang bervariasi.

Dari bidang *object-detection* ini, menghasilkan suatu studi khusus pada pendeteksian wajah manusia atau *face detection*. Disini *face detection* berbeda dengan *face recognition*, karena hanya mendeteksi dimana letak wajah manusia pada citra masukan. Selama bertahun-tahun, telah dikembangkan banyak metode dan algoritma pendeteksian wajah manusia. Menurut Yang et al., teknik pendeteksian wajah dapat diklasifikasikan sebagai *feature-based*, *template-based*, atau *appearance-based*.

- Teknik *feature-based* mencoba mencari lokasi dari fitur citra yang sudah jelas seperti mata, hidung, dan mulut; kemudian melakukan verifikasi apakah fitur-fitur wajah ini sudah benar secara tatanan geometris. Teknik ini merupakan teknik yang muncul pada awal studi *face-detection*, juga pendekatan yang berbasis *eigenspaces*, *local filter jets*, *support vector machines*, dan *boosting*.
- Teknik *template-based* menggunakan pendekatan seperti *active appearance models (AAMs)*, dapat menangani bermacam-macam pose dan variasi ekspresi wajah, tetapi biasanya memerlukan inisialisasi yang tepat pada wajah, oleh karena itu tidak cocok untuk sistem yang membutuhkan pendeteksian yang cepat.
- Sedangkan teknik *appearance-based* memindai potongan-potongan persegi panjang diatas citra masukan untuk mencari kandidat region citra yang merupakan wajah, dan kemudian hasil pencariannya bisa disaring menggunakan suatu blok yang disebut *cascade* keputusan dan berisi algoritma deteksi (Viola

and Jones, 2004). Sebagian besar pendekatan *appearance-based* saat ini bergantung pada pengklasifikasi yang sudah dilatih menggunakan objek wajah dan bukan-wajah.

BAB III
TEORI PENDETEKSIAN WAJAH

3.1 Teori Viola Jones Detection

Teknik *face-detection* yang akan digunakan adalah teknik yang dikembangkan oleh Paul Viola dan Michael Jones-yang biasa dikenal sebagai pendeteksi Viola-Jones, dimana teknik ini dapat memproses citra dengan sangat cepat dengan tetap mencapai tingkat deteksi yang sangat tinggi (Viola and Jones, 2004). Ada tiga poin penting dalam aplikasi teknik ini, yaitu citra integral, algoritma AdaBoost, dan penggunaan *cascade*. Kelebihan dari teknik deteksi ini adalah dapat memproses dengan sangat cepat, tanpa membutuhkan spesifikasi komputer yang tinggi. Teknik ini dapat mencapai tingkatan *frame rate* yang tinggi hanya dengan menggunakan informasi yang muncul pada suatu citra *greyscale* tunggal.

3.2.1 Pembelajaran Mesin

Pembelajaran mesin atau *machine learning* adalah suatu teknik yang bertujuan untuk mengubah data mentah menjadi suatu informasi yang berguna (Bradski dan Kaehler, 2008). Setelah sebelumnya melakukan pembelajaran dari suatu kumpulan data, dengan pembelajaran mesin komputer dapat memberikan kesimpulan atau keputusan tentang data tersebut, seperti mendeteksi objek. Pembelajaran mesin mengubah data menjadi informasi dengan memisahkan aturan atau pola dari data tersebut. Bidang pembelajaran mesin ada karena dorongan pemikiran bahwa sistem dan algoritma komputer dapat meningkatkan performanya sendiri sejalan waktu (Sebe,et. al., 2005). Pembelajaran telah berevolusi dari sistem pembelajaran serba guna yang "*knowledge-free*", "*perceptron*", dan pendekatan keputusan-teoretikal untuk pembelajaran mesin, sampai pembelajaran simbolik dari pengetahuan tingkat-tinggi, dan jaringan saraf tiruan serta algoritma genetik. Dengan kemajuan terbaru dari perangkat lunak dan perangkat keras, muncul beragam aplikasi dari penelitian tentang pembelajaran mesin.

3.2.2 Citra Integral

Penggunaan representasi citra yang disebut "citra integral", yang memungkinkan fitur yang digunakan pada detektor teknik ini untuk dikomputasikan dengan sangat cepat. Citra integral hampir sama dengan jumlah dari keseluruhan tabel area yang digunakan pada *texture mapping* pada grafika komputer, dan merupakan struktur data yang memungkinkan penghitungan cepat dari daerah pada citra. Penghitungan ini sangat berguna pada banyak aplikasi, seperti komputasi dari Haar wavelets, yang digunakan pada pengenalan wajah dan algoritma sejenis. Pada teknik ini, digunakan suatu set dari fitur yang merupakan turunan dari fungsi Basis Haar, dan untuk menghitung fitur-fitur ini dengan sangat cepat pada berbagai ukuran, digunakanlah citra integral. Citra integral dapat dikomputasikan dari citra dengan menggunakan hanya beberapa operasi pada tiap *pixel*. Ketika sudah dikomputasikan, fitur-fitur sejenis-Haar ini dapat dikomputasikan pada berbagai ukuran atau lokasi dengan waktu yang konstan. Citra integral pada lokasi x,y mengandung jumlah total dari pixel diatasnya dan di sisi kiri dari x,y :

$$ii(x, y) = \sum_{x' \le x, y' \le y} i(x', y') \qquad (3.1)$$

Dimana *ii(x,y)* adalah citra integral dan *i(x,y)* adalah citra yang sebenarnya. Menggunakan pasangan ini :

$$s(x,y) = s(x,y-1)+i(x,y) \qquad (3.2)$$

$$ii(x,y) = ii(x-1,y)+s(x,y) \qquad (3.3)$$

(dimana *s(x,y)* adalah jumlah kumulatif baris, *s(x,-1)=0*, dan *ii(-1,y)=0*) citra integral dapat dihitung dalam satu kali pemrosesan citra sebenarnya.

Menggunakan citra integral, jumlah dari daerah persegi panjang di lokasi manapun dapat dihitung dengan empat array referensi, dan perbedaan dari jumlah dua daerah persegi panjang dapat dihitung dengan delapan referensi array.

3.2.3 Algoritma AdaBoost

Setelah penggunaan citra integral, kedua adalah penggunaan pengklasifikasi yang simple dan efisien yang dibangun dengan memilih sejumlah kecil dari fitur-

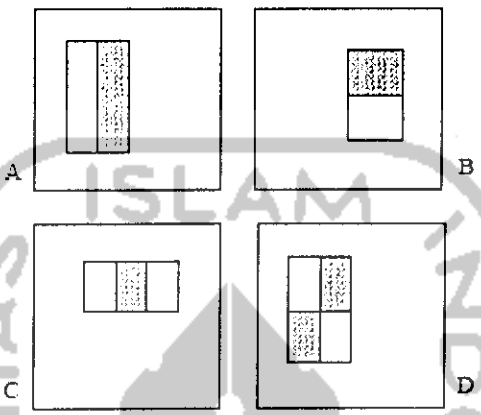
fitur penting yang berasal dari suatu pustaka yang sangat besar dari fitur-fitur potensial menggunakan AdaBoost. Untuk memastikan klasifikasi yang cepat, proses pembelajarannya harus mengecualikan suatu mayoritas besar dari fitur yang tersedia, dan memfokuskan pada suatu set kecil dari fitur yang kritikal. Pemilihan fitur ini dicapai menggunakan algoritma pembelajaran AdaBoost dengan membatasi setiap pengklasifikasi yang lemah untuk bergantung pada fitur tunggal. Sebagai hasilnya pada setiap tingkatan pada proses *boosting* untuk memilih pengklasifikasi lemah yang baru, dapat ditampilkan sebagai proses seleksi fitur.

3.2.4 Penggunaan Cascade

Ketiga adalah metode untuk mengkombinasikan pengklasifikasi yang lebih kompleks pada suatu struktur *cascade* yang mana dapat meningkatkan kecepatan detektor dengan sangat signifikan dengan lebih memfokuskan perhatian pada daerah citra yang lebih mungkin mengandung citra wajah, dimana teknik ini juga memungkinkan region latar belakang dari citra untuk diabaikan dengan cepat untuk lebih mengalokasikan komputasi atas region yang lebih mendekati wajah. Yang perlu dicatat pada pendekatan fokus ini adalah teknik ini seringkali memungkinkan untuk memberi keputusan dimana wajah mungkin muncul pada suatu citra. Proses yang lebih kompleks hanya dilakukan pada region-region ini. Kunci dari pendekatan ini adalah tingkat kesalahan negatif dari proses. Diimplementasikan pada komputer desktop biasa, deteksi wajah ini berjalan pada 15 frame per detik (Viola dan Jones, 2001). Pada intinya, teknik ini mengorganisasikannya sebagai *rejection cascade* atau susunan *cascade* penolakan dari node-node, dimana setiap node adalah pengklasifikasi pohon keputusan AdaBoost yang dirancang untuk mendapatkan tingkatan deteksi yang sangat tinggi (tingkat kesalahan negatif yang rendah, atau wajah yang luput terdeteksi) dengan mengorbankan sekitar 50% rasio penolakan (tingkat kesalahan positif, atau objek bukan wajah yang salah terdeteksi) (Bradski dan Kaehler, 2008).

3.2.5 Proses Klasifikasi dan Deteksi

Teknik deteksi objek ini mengklafikasikan citra berdasar nilai dari fitur-fitur sederhana. Banyak motivasi di balik penggunaan citra ini, daripada melakukan pengecekan *pixel* secara langsung.



Gambar 3.1 Fitur Persegi Panjang

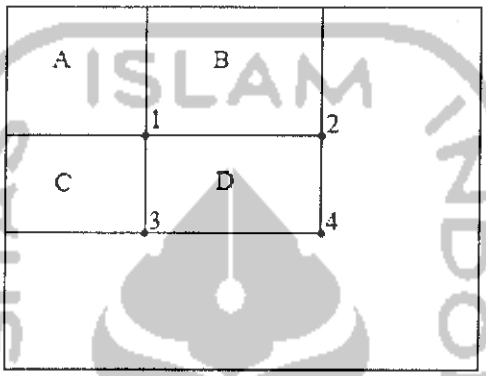
Gambar 3.1 (Viola dan Jones, 2001) menunjukkan contoh fitur persegi panjang menunjukkan hubungannya dengan *sub-window* deteksi terdekat. Jumlah dari *pixel* yang ada pada persegi panjang berwarna putih dipisahkan dari jumlah *pixel* persegi panjang berwarna abu-abu. Fitur dari dua persegi panjang ini ditunjukkan pada persegi panjang A dan B. Persegi panjang C menunjukkan fitur tiga persegi panjang, dan persegi panjang D menunjukkan fitur empat persegi panjang.

Alasan lain adalah karena fitur dapat bertindak untuk menerjemahkan domain pengetahuan *ad-hoc* yang merupakan sesuatu yang sulit untuk dipelajari menggunakan data pelatihan yang terbatas. Sistem ini juga mempunyai motivasi kritikal kedua tentang penggunaan fitur, yaitu sistem yang berbasis fitur beroperasi lebih cepat daripada sistem berbasis *pixel*.

Fitur sederhana yang digunakan merupakan turunan dari fungsi dasar Haar yang digunakan oleh Papageorgiou et al (Viola dan Jones, 2001). Lebih spesifiknya menggunakan tiga jenis fitur ini. Nilai dari fitur dua persegi panjang merupakan perbedaan antara jumlah keseluruhan *pixel* dalam lingkup dua daerah persegi panjang. Daerah yang mempunyai ukuran dan bentuk yang sama adalah berdekatan secara horizontal atau vertikal. Suatu fitur tiga persegi panjang menghitung jumlah dalam dua persegi panjang luar yang dipisahkan dari jumlah keseluruhan persegi

panjang pusat. Kemudian terakhir, fitur empat persegi panjang menghitung beda antara pasangan diagonal dari persegi panjang.

Dengan diberikan resolusi dasar dari detektor adalah 24*24 pixel, set dari fitur kotak ini berjumlah sangat besar, lebih dari 180.000. Tidak seperti basis Haar, set fitur kotak ini lebih dari lengkap.



Gambar 3.2 Penjumlahan Fitur

Jumlah dari keseluruhan *pixel* dalam persegi panjang D dapat dihitung dengan empat referensi array. Nilai dari citra integral pada lokasi ke-1 adalah jumlah dari pixel dalam persegi panjang A. Nilai dari lokasi ke-2 adalah A+B, pada lokasi ke-3 adalah A+C, dan pada lokasi ke-4 adalah jumlah A+B+C+D. Jumlah pada D saja dapat dihitung sebagai 4+1-(2+3) (Viola dan Jones, 2001).

Banyak metode pembelajaran mesin yang dapat dipakai pada fungsi klasifikasi ini, tetapi yang diterapkan disini adalah varian dari teknik AdaBoost (Adaptive Boost). Pada teknik AdaBoost yang sebenarnya, teknik ini digunakan untuk meningkatkan performa pada klasifikasi dari algoritma pembelajaran sederhana. Sedangkan pada sistem *face-detection* ini, AdaBoost digunakan untuk memilih sejumlah kecil fitur dan juga melatih unit pengklasifikasi. Alasan kenapa teknik AdaBoost digunakan, karena telah dibuktikan bahwa pada pelatihan pengklasifikasi kuat, tidak menghasilkan error, bahkan pada beberapa kali pelatihan. Lebih penting lagi, teknik ini telah terbukti meningkatkan performa generalisasi yang berhubungan pada pelatihan.

Menurut hipotesa Viola-Jones, sejumlah kecil dari fitur-fitur ini dapat dikombinasikan untuk membentuk pengklasifikasi yang efektif, dengan mengambil

contoh dari hasil 180.000 fitur persegi panjang yang diasosiasikan dengan setiap *sub-window* pada citra,yang mana komputasi dari semua ini sangat berat dan lamban.

Untuk mencari fitur yang dapat dikombinasikan ini, algoritma pembelajaran yang lemah digunakan untuk memilih fitur persegi panjang tunggal yang terbaik dalam memisahkan contoh positif dan negatif. Kemudian untuk setiap fitur, pembelajar lemah menentukan ambang batas optimal fungsi klasifikasi, seperti jumlah minimum dari contoh yang telah salah terpilih. Unit pengklasifikasi lemah hj(x) kemudian mengandung fitur fj, ambang batas [teta-j] dan paritas pj, menunjukkan arah dari pertidaksamaan :

$$h_j = \begin{cases} 1 & if\ p_j f_j(x) < p_j \theta_j \\ 0 & otherwise \end{cases} \qquad (3.4)$$

Disini x adalah *sub-window* 24x24 *pixel* dari citra. Namun pada prakteknya tidak ada fitur tunggal yang dapat melakukan tugas klasifikasi dengan error yang rendah, tingkatan errornya berkisar antara 0,1 dan 0,3.

Berikut adalah algoritma AdaBoost yang dipakai (Viola dan Jones, 2001):

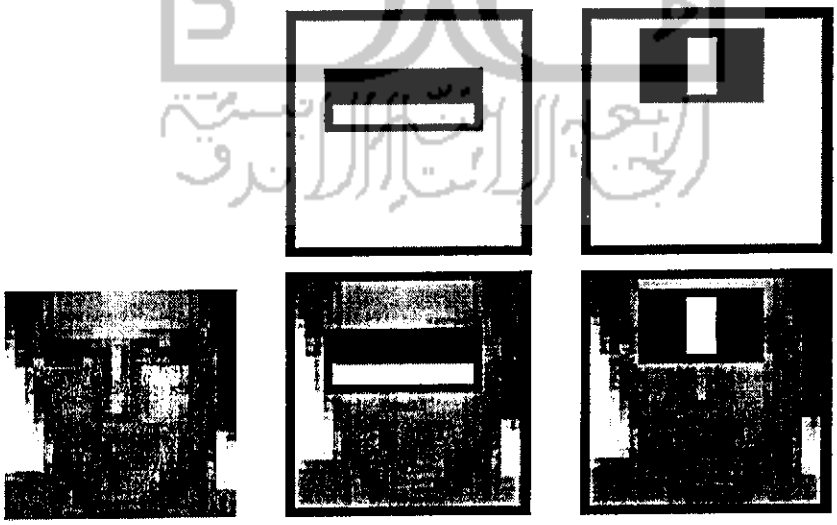
1. Dengan diberikan contoh citra $(x_1,y_1),\ldots,(x_n,y_n)$ dimana $y_i=0,1$ untuk contoh negatif dan positif berturut-turut.
2. Tentukan bobot $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ untuk $y_i = 0,1$ masing-masing, dimana m dan l masing-masing adalah nilai positif dan negatif.
3. Untuk $t=1,\ldots T$:
   a. Normalisasikan bobot,
   $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$
   sehingga w_t merupakan distribusi probabilitas.
   b. Untuk setiap fitur j, latih sejumlah a pengklasifikasi hj untuk satu fitur. Kemudian *error* dievaluasi dengan melalui wt, $E_j = \sum_i w_i \mid h_j(x_i) - y_i \mid$.
   c. Pilih pengklasifikasi, h_t, dengan error terendah e_t.
   d. Perbarui bobot : $w_{t+1,i} = w_{t,i}\beta_t^{1-ei}$
   Dimana $e_i = 0$ jika contoh x_i diklasifikasikan dengan benar, $e_i = 1$ sebaliknya, dan $\beta_t = \frac{\epsilon t}{1-\epsilon t}$
4. Pengklasifikasi kuat yang terakhir adalah :

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

dimana $a_t = \log 1/\beta_t$

Dari algoritma diatas, setiap kali pelatihan memilih satu fitur dari 180.000 fitur yang potensial. Eksperimen awal menunjukkan bahwa pengklasifikasi wajah dari sisi depan yang dibangun dari 200 fitur menghasilkan tingkatan deteksi 95% dengan tingkatan kesalahan positif 1 dari 14084. Hasil ini cukup bagus, tetapi belum cukup untuk diterapkan pada tugas-tugas *real-time*. Pada lingkup komputasi, mungkin pengklasifikasi ini tergolong bekerja dengan cepat daripada sistem lain, dimana teknik ini hanya membutuhkan 0.7 detik untuk memindai suatu citra dengan luas 384x288 pixel. Namun teknik paling tepat untuk meningkatkan performa deteksi yaitu menambahkan fitur kepada pengklasifikasi, secara langsung berakibat pada penambahan waktu komputasi.

Untuk permasalahan deteksi wajah ini, fitur persegi panjang yang dipilih oleh algoritma AdaBoost cukup berarti dan dapat diterjemahkan dengan mudah. Fitur pertama memfokuskan pada daerah mata, yang seringkali lebih gelap daripada daerah hidung dan pipi. Fitur ini tidak begitu terpengaruh dengan ukuran dan lokasi dari citra wajah. Fitur kedua yang dipilih bergantung pada ciri bahwa mata lebih gelap daripada daerah diantara mata dan hidung.



Gambar 3.3 Fitur yang dipilih AdaBoost

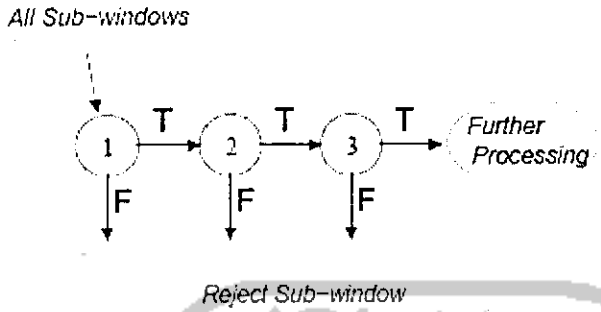
Gambar 3.3 (Viola dan Jones, 2001) menunjukkan fitur pertama dan kedua yang dipilih oleh algoritma AdaBoost. Dua fitur yang ada pada baris atas ditimpakan ke citra wajah yang dilatih pada baris bawah. Fitur pertama mengukur perbadaan

dalam intensitas antara daerah mata dan daerah di sekitar atas pipi. Fitur ini berdasar atas observasi bahwa daerah mata seringkali lebih gelap daripada pipi. Kemudian fitur kedua membandingkan intensitas daerah mata dengan intensitas bagian atas hidung.

Berikutnya adalah penggunaan algoritma untuk membangun suatu susunan *cascade* dari pengklasifikasi yang mencapai performa deteksi yang tinggi, dengan mengurangi waktu komputasi secara signifikan. Untuk dapat mencapai ini, diperlukan pengklasifikasi yang lebih kecil dan lebih efisien, serta telah mengalami *boost* yang dapat dibangun dengan melakukan penolakan pada banyak hasil *sub-window* negatif dan mendeteksi sebagian besar hasil yang positif, yang dapat dicapai dengan mengatur ambang batas dari pengklasifikasi hingga tingkat kesalahan negatifnya mencapai nol. Kemudian pengklasifikasi yang lebih sederhana digunakan untuk menolak sebagian besar *sub-window* sebelum pengklasifikasi yang lebih rumit dipanggil untuk mencapai tingkat kesalahan positif yang rendah.

Bentuk keseluruhan dari proses deteksi adalah pohon keputusan degenerasi, yang selama ini disebut "*cascade*", dapat dilihat pada gambar 3.3. Hasil positif dari pengklasifikasi pertama menyebabkan evaluasi dari pengklasifikasi kedua yang mana telah diatur agar dapat mencapai tingkat deteksi yang tinggi. Hasil positif dari pengklasifikasi kedua ini memanggil pengklasifikasi ketiga, dan begitu seterusnya. Hasil yang negatif akan menyebabkan dilakukan penolakan secara langsung pada *sub-window* yang bersangkutan.

Tingkatan pada *cascade* dibangun dengan melatih pengklasifikasi menggunakan algoritma AdaBoost dan kemudian mengatur ambang batas untuk meminimalisir tingkat kesalahan negatif. AdaBoost sendiri dirancang dengan tingkat error yang rendah atas data pelatihan. Secara umum, ambang batas yang rendah menghasilkan tingkat deteksi yang tinggi dan juga tingkat kesalahan positif yang tinggi.



Gambar 3.4 *Cascade* Deteksi

Gambar 3.4 (Viola dan Jones, 2001) adalah gambaran skematik dari susunan *cascade* deteksi, yang merupakan urutan pengklasifikasi yang diaplikasikan pada setiap *sub-window*. Pengklasifikasi yang sedang berjalan membuang sejumlah besar contoh negatif dengan pemrosesan yang sangat kecil. Kemudian lapisan berikutnya membuang hasil negatif yang lain, tetapi membutuhkan komputasi tambahan dan lebih rumit dari sebelumnya. Setelah beberapa tingkatan dari pemrosesan, *sub-window* yang ada telah berkurang banyak. Pemrosesan lebih lanjut dapat mencapai banyak bentuk, seperti penambahan lapisan dari *cascade* pada sistem deteksi ini.

Suatu unit pengklasifikasi yang bekerja dengan sangat baik dapat dibangun dari pengklasifikasi kuat dua-fitur, dengan cara mengurangi ambang batas untuk meminimalisir kesalahan negatif. Dengan menggunakan suatu set data pelatihan validasi, ambangnya dapat diatur agar tingkat deteksinya 100%, dengan tingkatan kesalahan positifnya 40%. Dijelaskan pada gambar 3, penggunaan dua-fitur ini.

Struktur dari *cascade* mencerminkan fakta bahwa dalam suatu citra, sebagian besar *sub-window* yang bekerja mempunyai hasil negatif. Karenanya, susunan *cascade* ini mencoba melakukan penolakan hasil negatif sebanyak mungkin pada tingkatan *cascade* yang paling awal.

Hampir mirip dengan pohon keputusan, pengklasifikasi berikutnya dilatih dengan menggunakan contoh yang dapat melalui tingkatan klasifikasi sebelumnya. Hasilnya, pengklasifikasi kedua menghadapi tugas yang lebih sulit daripada pengklasifikasi pertama. Contoh yang dapat melalui pengklasifikasi pertama memerlukan komputasi yang lebih sulit daripada contoh biasanya. Contoh-contoh lebih sulit yang dihadapi pengklasifikasi yang lebih dalam, mendorong kurva ROC

(*Receiver Operating Characteristic)* menjadi menurun. Dengan tingkat deteksi yang diberikan, pengklasifikasi yang lebih berikutnya mempunyai tingkatan kesalahan positif yang lebih tinggi.

Viola dan Jones mengorganisasikan setiap grup pengklasifikasi yang telah mengalami boost menjadi suatu node cascade penolakan. Setiap node mengandung keseluruhan cascade yang telah mengalami boost dari beberapa grup dari pohon keputusan yang telah dilatih pada fitur sejenis-Haar dari wajah dan bukan wajah (atau bisa juga objek lain). Umumnya, node-node ini diatur berurutan dari yang paling kompleks, sehingga komputasinya dapat diminimalisir karena yang dicoba adalah node yang paling sederhana komputasinya, ketika melakukan penolakan pada region dari citra. Biasanya, proses boost pada setiap node diatur sehingga mempunyai tingkatan deteksi yang sangat tinggi, yang berakibat banyak hasil kesalahan positif. Ketika dilatih pada wajah contohnya, hampir sebagian besar (99,9%) dari wajah ditemukan, tetapi juga banyak (sekitar 50%) dari yang bukan wajah, secara error, terdeteksi pada setiap node. Tapi ini tidak terlalu masalah, sebab menggunakan sekitar 20 node akan tetap menghasilkan tingkatan deteksi wajah sekitar 98% dengan tingkatan kesalahan positif hanya 0.5% (Bradski dan Kaehler, 2008).

Ketika aplikasi dijalankan, sebuah daerah pencarian dengan ukuran yang berbeda-beda ditelusuri secara keseluruhan citra aslinya. Pada prakteknya, 70-80% dari yang objek bukan wajah ditolak pada dua node pertama dari cascade penolakan, dimana setiap node menggunakan sekitar sepuluh cabang dari pohon keputusan (Bradski dan Kaehler, 2008).

Proses pelatihan *cascade* melibatkan dua tipe *'trade-off'*. Pada sebagian besar kasus, pengklasifikasi dengan banyak fitur akan lebih dapat mencapai tingkatan deteksi yang tinggi dan tingkat kesalahan positif yang rendah. Di sisi lain, hal ini menyebabkan waktu komputasi yang lebih tinggi. Pada prinsipnya dapat didefinisikan framework optimisasi dimana : i)jumlah dari tingkatan pengklasifikasi, ii)jumlah fitur pada setiap tingkatan, iii)ambang batas setiap tingkatan, merupakan *trade-off* yang bertujuan untuk meminimalisir jumlah fitur terevaluasi yang

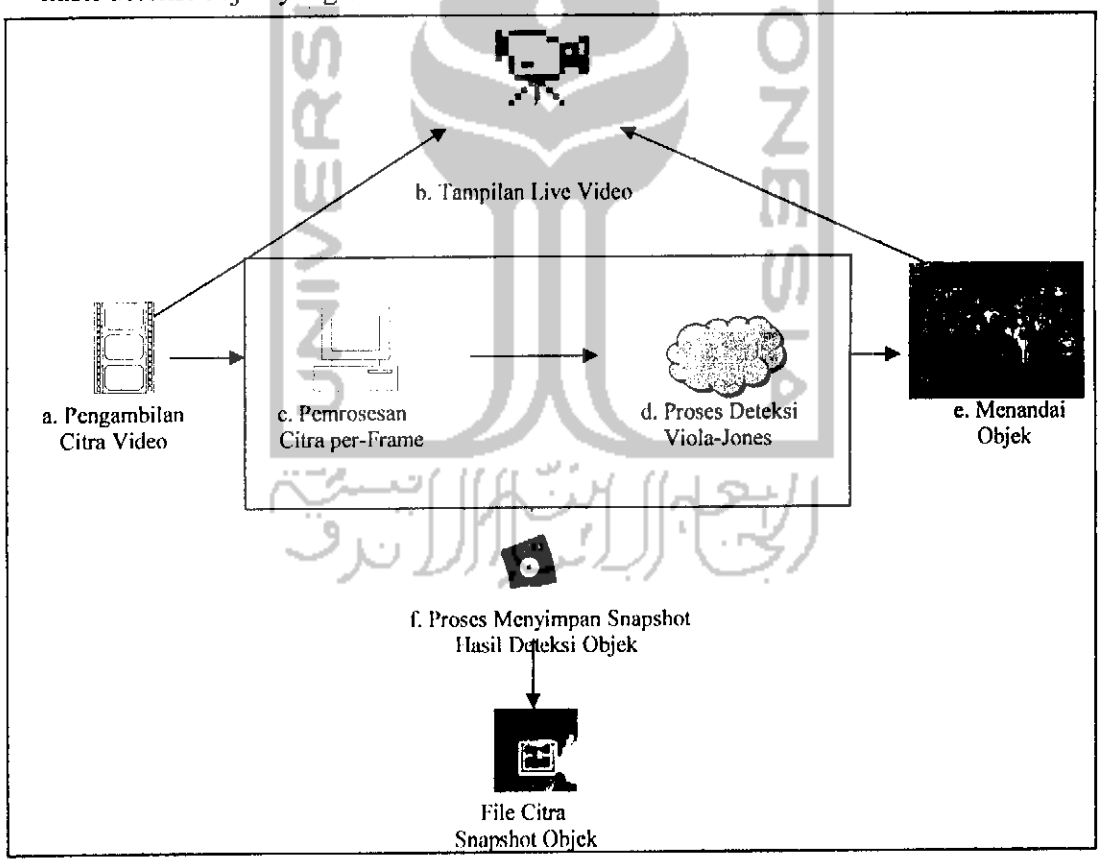
diharapkan. Sayangnya, menemukan nilai optimal ini merupakan hal yang sangat sulit.

Pada prakteknya, *framework* yang sangat sederhana digunakan untuk memproduksi pengklasifikasi yang efektif yang bekerja dengan sangat efisien. Setiap lapisan pada cascade mengurangi tingkat kesalahan positifnya dan menurunkan tingkat deteksi. Suatu target dipilih untuk reduksi kesalahan positif dan penurunan tingkat deteksi. Setiap lapisan dilatih dengan cara menambahkan fitur sampai target deteksi dan tingkat kesalahan positif ditemukan, dan tingkat kesalahan positif ini ditentukan dengan melakukan tes menggunakan set validasi. Lapisan baru ditambahkan sampai keseluruhan target untuk tingkat kesalahan positif dan deteksi dapat tercapai.

BAB IV
MODEL SISTEM

4.1. Model Sistem

Aplikasi yang dibangun adalah aplikasi yang dapat mengolah citra yang diambil dari kamera video, dan menganalisisnya untuk mengetahui keberadaan suatu objek. Ada beberapa proses yang terjadi pada jalannya aplikasi ini, yaitu pengambilan citra video, pemrosesan citra pada setiap *frame*, , pendeteksian objek menggunakan teknik Viola-Jones, dan kemudian menggambar persegi panjang pada objek wajah yang ditemukan dari citra video tersebut serta menyimpan *snapshot* hasil deteksi objek yang ditemukan.



Gambar 4.1 Diagram Proses

Diagram yang tertera pada gambar 4.1 menunjukkan gambaran proses yang terjadi pada aplikasi ini secara umum.

a. *Pengambilan Citra Video*

Proses pengambilan citra adalah proses yang pertama dilakukan pada aplikasi ini dan proses ini dilakukan terus menerus sampai aplikasi dihentikan. Proses ini menggunakan suatu pustaka dari framework C++ OpenFrameworks yang didalamnya sudah tersedia suatu *interface* untuk mengambil data dari *webcam* dan melakukan pengambilan citra video yang bersifat *streaming*. Pada proses ini dilakukan *streaming* video dari *webcam* dengan ukuran 320x240 *pixel* dan kemudian mengirimkan datanya ke proses lain untuk dianalisa.

b. *Tampilan Live Video*

Proses ini adalah proses yang dapat terlihat oleh pengguna sistem. Proses ini menerima data streaming video hasil dari proses **a** diatas dan menampilkannya ke layar. Komponen yang menangani proses ini adalah library *streaming video* dan *image processing* dari OpenFrameworks. Proses ini juga menerima hasil penandaan objek yang telah diketemukan berupa persegi panjang di sekitar objek.

c. *Pemrosesan Citra Per-Frame*

Selama proses pengambilan citra berjalan, *input* citra *streaming* dari *webcam* diambil setiap frame menggunakan library openFrameworks, kemudian dari *input* tersebut frame diproses menjadi suatu variabel OpenCV dan diubah menjadi citra *greyscale* melalui fungsi yang ada pada OpenCV.

Dari citra yang sudah diubah menjadi *greyscale* ini, akan ada fungsi yang bertugas memproses dan menganalisa citra tersebut. Kemudian pada citra *greyscale* ini dilakukan *histogram equalization* untuk menormalkan *brightness* dan menaikkan tingkat kontras pada citra tersebut, sehingga citra yang ada lebih mudah untuk dianalisa kedepannya. Setelah selesai diproses, citra ini diproses lagi oleh fungsi deteksi dengan menggunakan teknik Viola-Jones. Intinya, pemrosesan *real-time* yang terjadi sebenarnya dilakukan per-frame.

d. Proses Deteksi Viola-Jones

Disini dilakukan deteksi pada citra per-frame yang telah diproses sebelumnya dengan berdasar teknik yang telah dikembangkan oleh Viola-Jones dan menggunakan fitur Haar. Untuk membantu beberapa pemrosesan citra disini menggunakan framework *computer vision* dari Intel, yaitu OpenCV. Untuk mengenali objek, fungsi ini juga menggunakan suatu file xml "haarcascade_frontalface_alt.xml" yang berisi hasil pelatihan pembelajaran mesin berupa *cascade* atau *decision tree* yang dapat digunakan untuk mengenali objek.

Pertama-tama fungsi ini mengkonversi input citra *greyscale* yang semula bertipe IplImage yang merupakan suatu tipe variabel OpenCV untuk mendefinisikan suatu citra, menjadi suatu variabel *multi-channel* matrix agar nanti mudah diproses. Sebelum melakukan ini, fungsi menciptakan suatu *hidden cascade classifier* dari input file pelatihan xml yang telah disebutkan sebelumnya, dan bekerja bersama-sama dengan *cascade classifier* sebenarnya. Proses ini kemudian memanggil fungsi lain yang bertugas membuat suatu citra *integral* yang berupa matrix dari input citra, dan dimasukkan ke variabel lain. Dengan penghitungan citra *integral*, bisa dilakukan penghitungan cepat dari suatu *subregion*, membuat suatu ringkasan dari *matrix* citra, atau melakukan penghitungan standar deviasi (Bradski dan Kaehler, 2008). Citra *integral* ini merupakan salah satu tahap dalam proses deteksi Viola-Jones.

Pada proses pendeteksian objek ini juga menggunakan fungsi Canny Pruning dari OpenCV yang bertujuan untuk menemukan garis tepi yang menonjol dari objek dan memudahkan pada pendeteksian (Bradski dan Kaehler, 2008). Setelah dilakukan fungsi *canny pruning*, citra diubah kembali menjadi citra *integral*.

Setelah melakukan inisialisasi variabel-variabel dasar yang diperlukan dan memanggil set *cascade* pengklasifikasi dari file xml dan membuat set *hidden cascade*, akan disiapkan *cascade* untuk mendeteksi objek pada ukuran tertentu dan bagian tertentu dari citra, dan hal ini dilakukan pada suatu iterasi. Disini juga dilakukan penghitungan poin-poin pojok pada *hidden cascade* yang merupakan koordinat window objek original yang telah diskalakan dan hasil penghitungan dari poin-poin *cascade* asli, dan melakukan pengaturan pada citra *input* untuk diproses

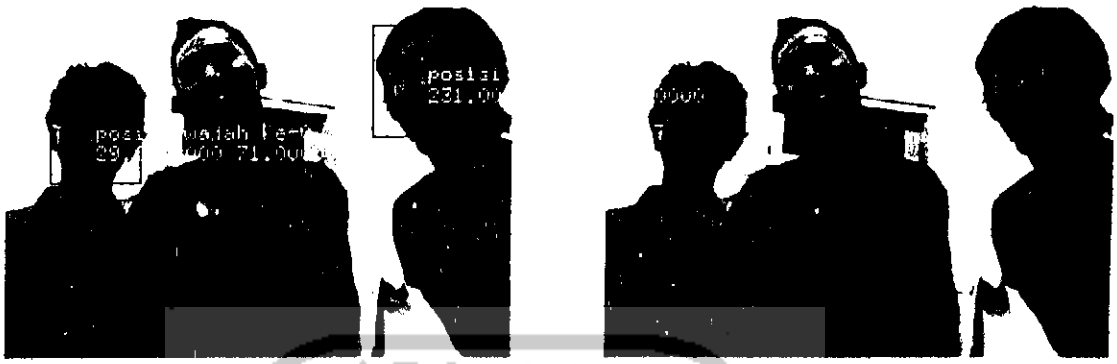
lebih lanjut. Kemudian dilakukan penghitungan pada poin-poin pojok yang digunakan untuk fitur persegi panjang Haar pada *node cascade*.

Kemudian ketika citra yang ada telah disiapkan, fungsi akan melakukan penghitungan antara citra *input* yang dianalisa, dengan fitur persegi panjang Haar yang telah disebutkan sebelumnya dalam suatu iterasi. Jika nilai balikan yang didapat dari penghitungan adalah positif, fungsi akan melanjutkan klasifikasi menuju klasifikasi yang lebih kompleks, sehingga akurasi keberadaan objek cukup mencapai *threshold*. Namun jika memberikan nilai negatif, maka fungsi akan mengabaikan dan maju ke poin koordinat pencarian berikutnya dari citra *input* kemudian melakukan penghitungan lagi, begitu seterusnya sampai mencapai poin koordinat akhir.

Setelah beberapa region yang lolos klasifikasi ditandai koordinat posisi dan ukurannya, region persegi panjang yang saling berdekatan akan disatukan menjadi satu persegi panjang dan dianggap sebagai satu kesatuan objek, kemudian nilai-nilainya akan dimasukkan ke suatu variabel nilai balikan fungsi.

e. Proses Menandai Objek Pada Tampilan

Fungsi dari OpenFrameWorks akan menerima nilai balikan dari pemanggilan fungsi pendeteksian objek yang berisi nilai-nilai tentang objek yang telah berhasil dideteksi. Dari nilai balikan tersebut, didapat koordinat pojok dari region objek, panjang dan lebarnya, dan juga titik tengahnya. Kemudian fungsi pendeteksi akan mengembalikan nilai-nilai tersebut ke aplikasi utama, dan fungsi dari OpenFrameworks akan menggambar persegi panjang pada frame sesuai posisi dan ukuran objek yang telah ditemukan, dan mengirimnya ke tampilan *live-video* pada proses **a**.



Gambar 4.2 Objek yang telah terdeteksi (Rowley et al,1998)

f. Proses menyimpan snapshot hasil deteksi objek

Selain menandai objek pada *live-video*, aplikasi juga akan memanggil fungsi lain yang bertugas melakukan pengambilan snapshot pada objek yang terdeteksi. Properti dari snapshot ini merupakan hasil nilai balikan dari fungsi pendeteksi objek, yang berisi koordinat, panjang, dan lebar dari region persegi panjang objek yang terdeteksi. Kemudian dari hasil snapshot ini akan disimpan sebagai file citra, dan disimpan ke dalam folder tertentu.

Gambar 4.3 Snapshot Hasil Deteksi

BAB V
PERANCANGAN SISTEM

5.1. Metode Perancangan Sistem

Metode perancangan yang dipakai untuk membangun aplikasi pendeteksi wajah ini ini adalah metode perancangan berorientasi obyek. Proses pembuatan aplikasi dimulai dari perancangan sistem menggunakan UML yang akan menghasilkan Use Case diagram, Class diagram , dan Sequence diagram.

Metode ini digunakan karena *framework* openFrameworks dan OpenCV yang digunakan menggunakan konsep berorientasi obyek, dan juga karena metode ini mempunyai banyak kelebihan daripada metode prosedural. Kelebihan dari metode analisis berorientasi obyek adalah :

1. Kode-kode pemrograman dibagi-bagi menjadi unsur yang disebut objek, dan objek ini menangani tugas masing-masing sehingga pengaturan dan perubahan kode serta fungsionalitasnya menjadi lebih mudah.
2. Penggunaan kembali kode yang telah dibuat, karena berupa objek.
3. Pemeliharaan dan perluasan software ke depannya menjadi lebih mudah, dengan menggunakan metode ini.

Diharapkan dengan analisis yang digunakan ini dapat mempermudah dalam pengembangan aplikasi *computer vision* khususnya aplikasi pendeteksi wajah, dan juga pengembangannya ke depan.

5.2. Perancangan Sistem

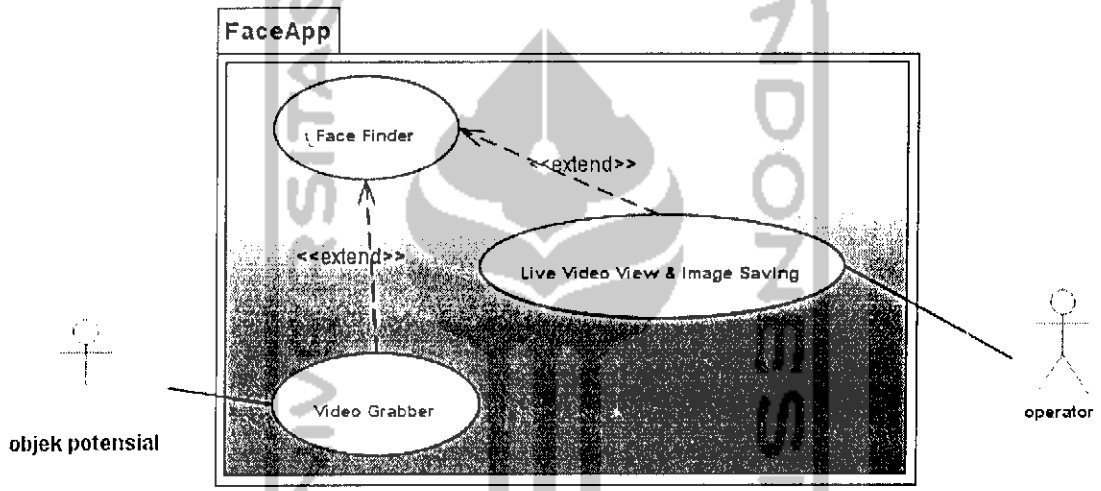
Dari hasil perancangan dengan menggunakan metode UML menghasilkan :

- Use case diagram
- Sequence diagram
- Activity diagram
- Class diagram

Pada perancangan sistem ini juga dibuat suatu *pseudo-code* untuk membantu pada implementasi sistem.

5.2.1 Use Case Diagram

Diagram ini digunakan untuk menggambarkan fungsi-fungsi yang dibutuhkan oleh sebuah sistem dan pada bagian ini diagram akan lebih digunakan untuk merepresentasikan interaksi antara pengguna dan sistem. Secara umum, *use case diagram* menunjukkan hubungan antara sistem dan objek-objek yang ada didalamnya dengan aktor-aktor atau objek di luar sistem.



Gambar 5.1 – Use Case Diagram

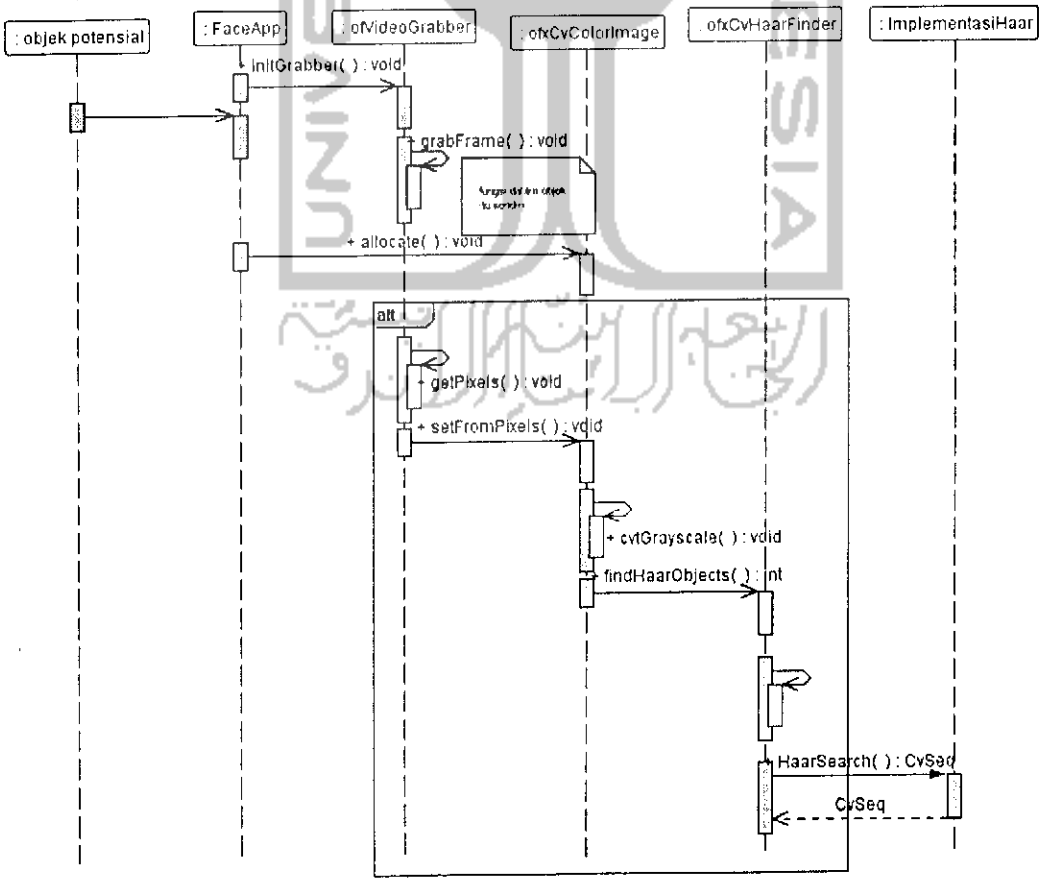
Pada kasus pengembangan aplikasi ini, *use case* diagram yang dibuat menggambarkan proses yang terjadi antara aktor objek potensial deteksi, dengan fungsi-fungsi pada aplikasi, dan juga hubungan aplikasi dengan operator yang menangani aplikasi ini.

5.2.2 Sequence Diagram

Sequence Diagram digunakan untuk menggambarkan perilaku pada suatu skenario. Diagram ini menunjukkan sejumlah obyek dan message (pesan) yang diletakkan diantara obyek-obyek ini dalam *use-case*. Dalam perancangan sistem ini, dibuat banyak *sequence diagram* sesuai skenario yang dimaksud.

5.2.2.1 Sequence Diagram Use Case Video Grabber

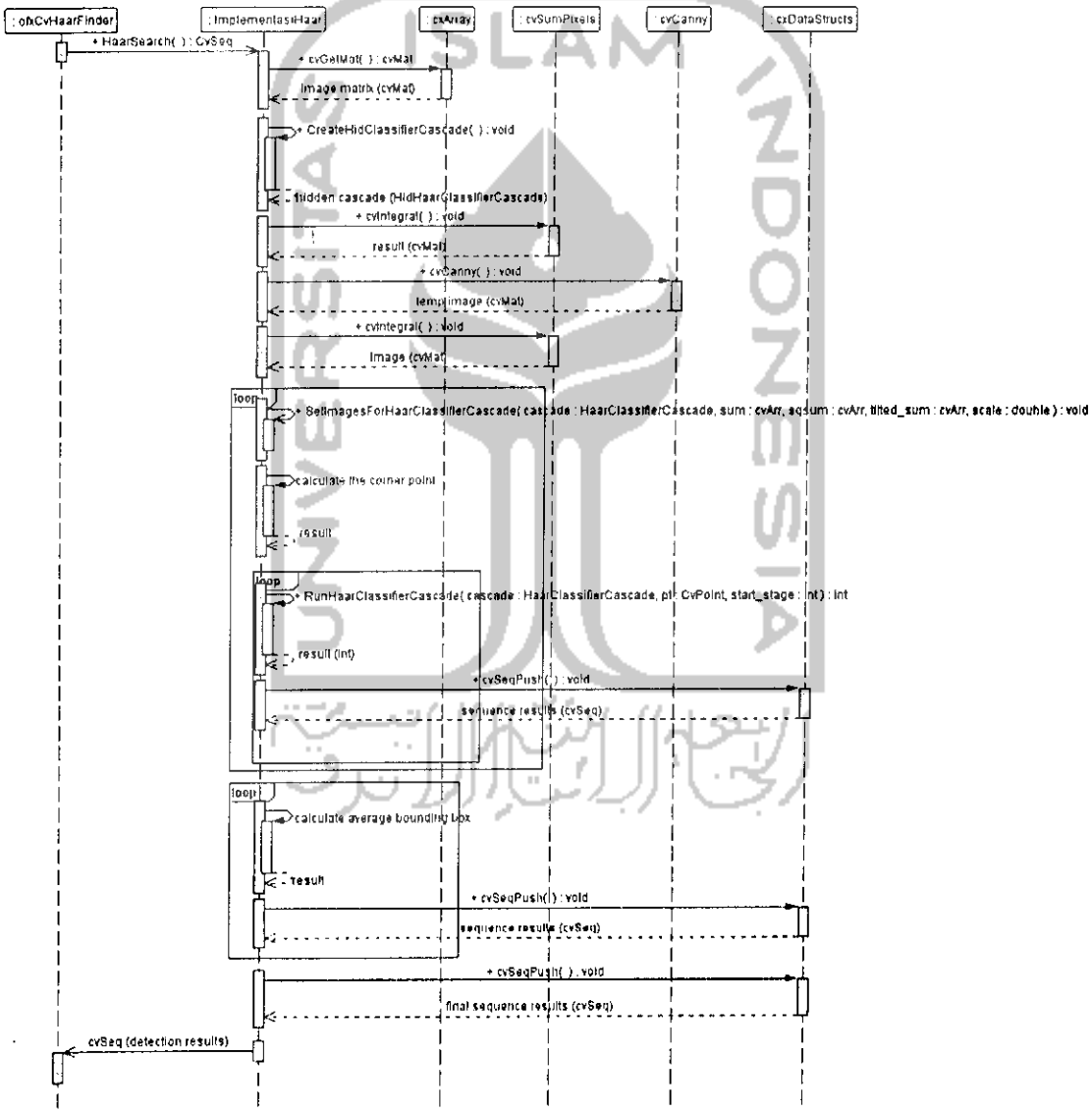
Sequence diagram pada gambar 5.2 menunjukkan proses yang terjadi pada saat pengambilan citra video sampai pemrosesan per-frame sebelum dilakukan pencarian objek wajah, sesuai dengan *use case* Video Grabber.



Gambar 5.2 - Sequence Diagrram Use Case Video Grabber

5.2.2.2 Sequence Diagram Use Case Face Finder

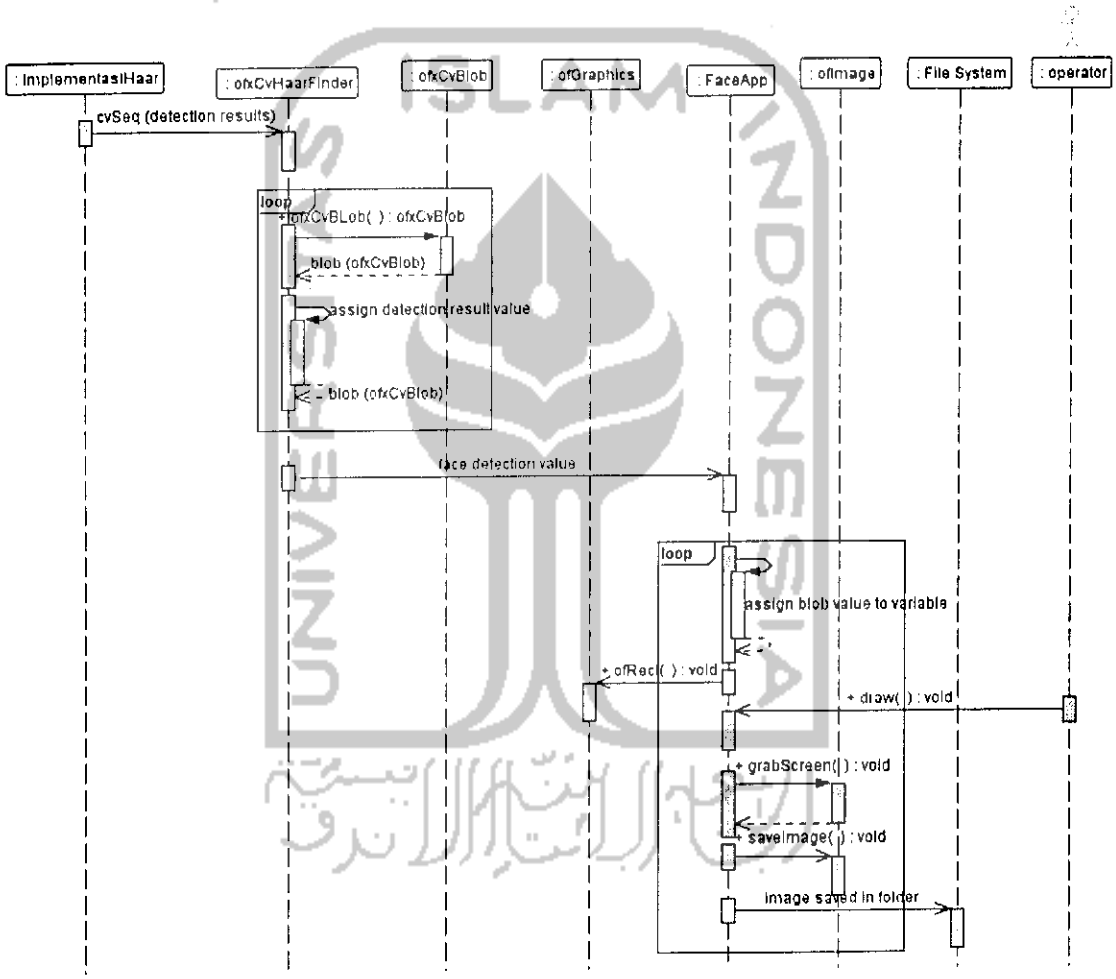
Sequence diagram pada Gambar 5.3 menunjukkan alur yang terjadi pada proses pendeteksian objek wajah, yaitu mulai dari citra per-frame diterima proses pendeteksian objek pada citra tersebut dan kemudian melempar hasilnya ke fungsi yang lain untuk kemudian diproses lebih lanjut, sesuai dengan *use case* Face Finder.



Gambar 5.3 – Sequence Diagram Use Case Face Finder

5.2.2.3 Sequence Diagram Use Case Live Video View dan Image Saving

Sequence diagram pada Gambar 5.4 menunjukkan proses akhir dari pendeteksian objek dan menandainya ke layar. Fungsi yang ada menerima nilai balikan dari kelas yang bertugas mendeteksi objek dari citra per-frame, dan kemudian menggambar persegi panjang pada lokasi ditemukannya objek wajah pada layar *live video*, sesuai dengan *use case* Live Video View dan Image Saving.

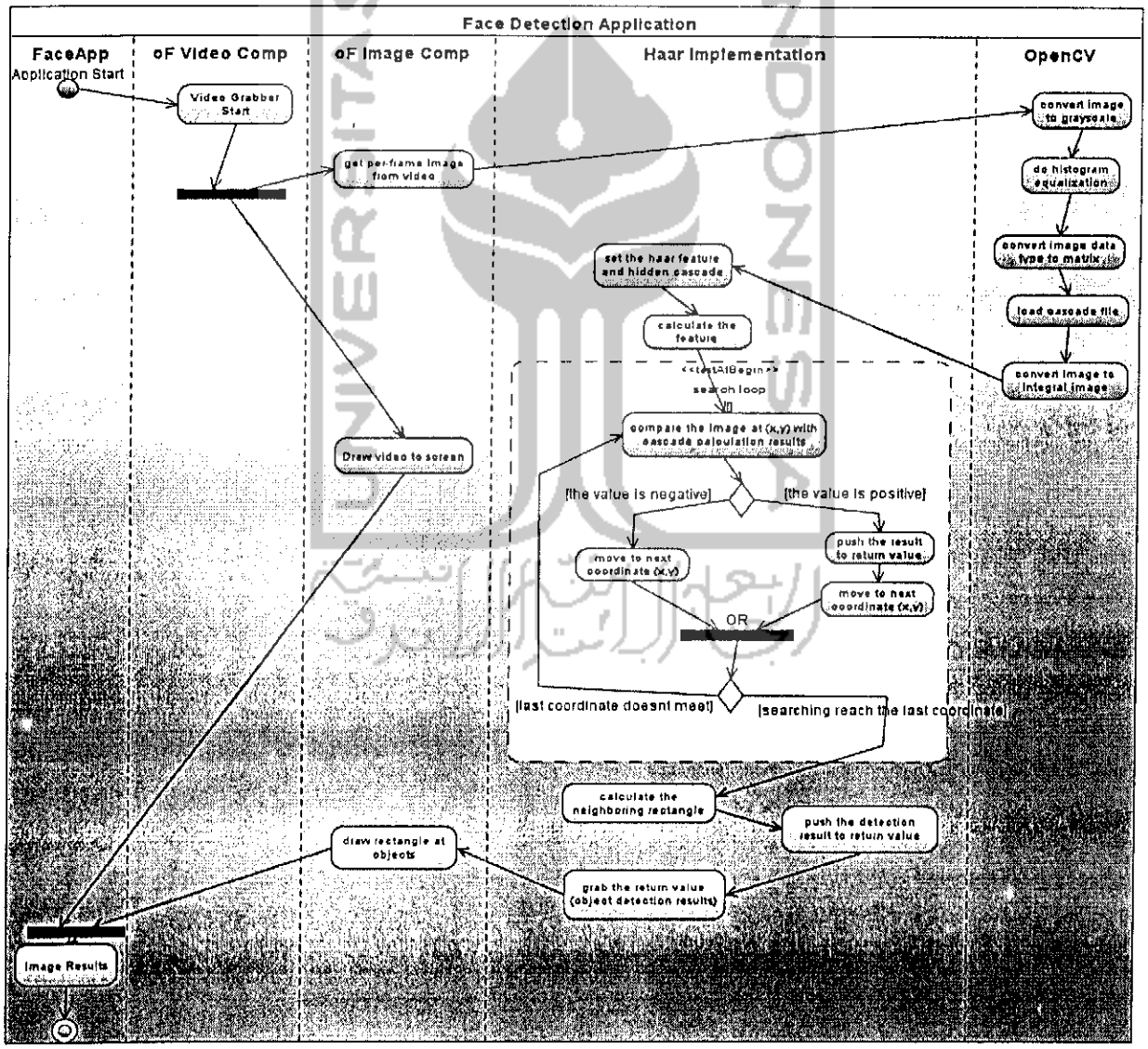


Gambar 5.4 – Sequence Diagram Use Case Live Video View & Image Saving

5.2.3 Activity Diagram

Activity Diagram digunakan ketika perancang ingin menggambarkan alur dari kelas-kelas yang terlibat secara non-teknis dan logika proses. *Activity Diagram* ini mirip dengan *data flow diagram* tradisional, tetapi dapat menggambarkan proses yang terjadi paralel. *Activity diagram* dapat digunakan untuk menunjukkan perilaku

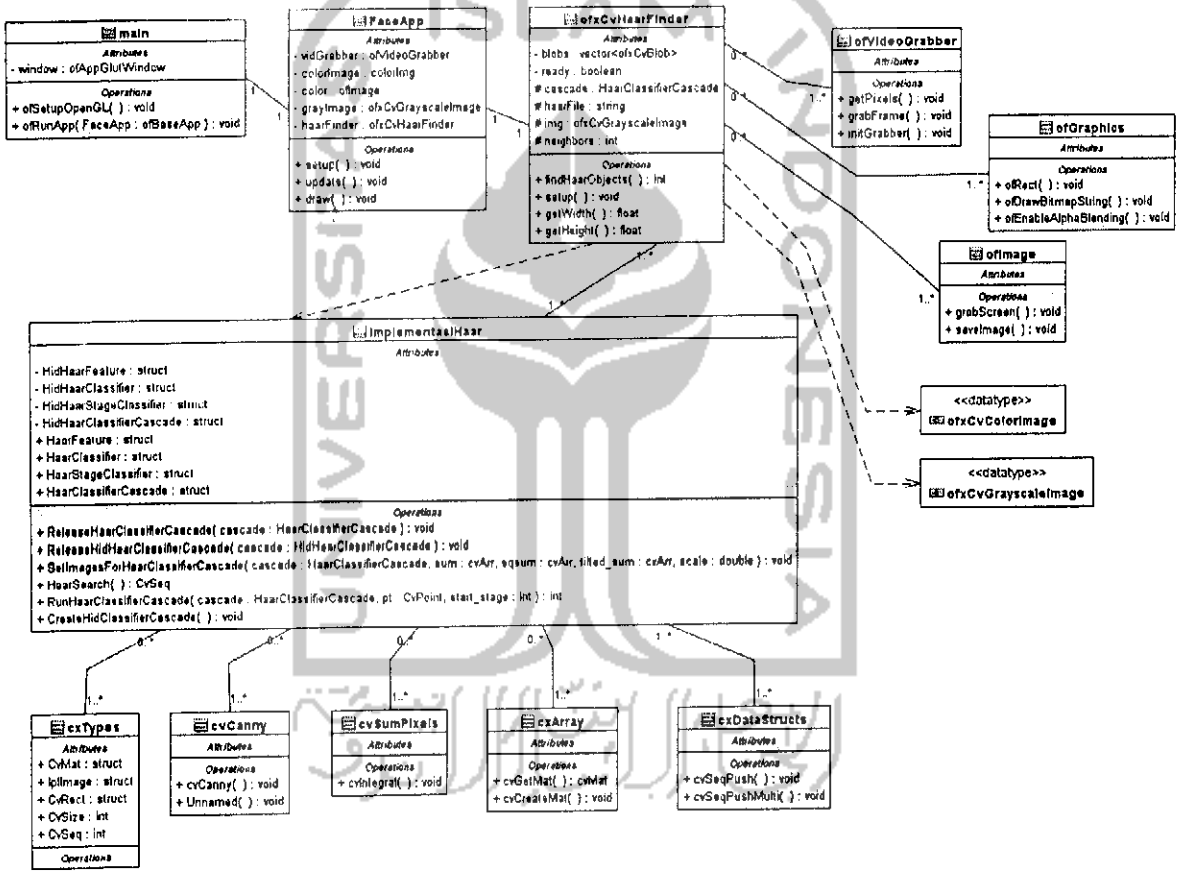
dari banyak kelas dan alir dari objek, data, atau alir kendali dari banyak kelas yang berbeda. Gambar 5.5 dibawah menunjukkan alur yang terjadi pada proses aplikasi deteksi wajah dan objek-objek yang berperan didalamnya.



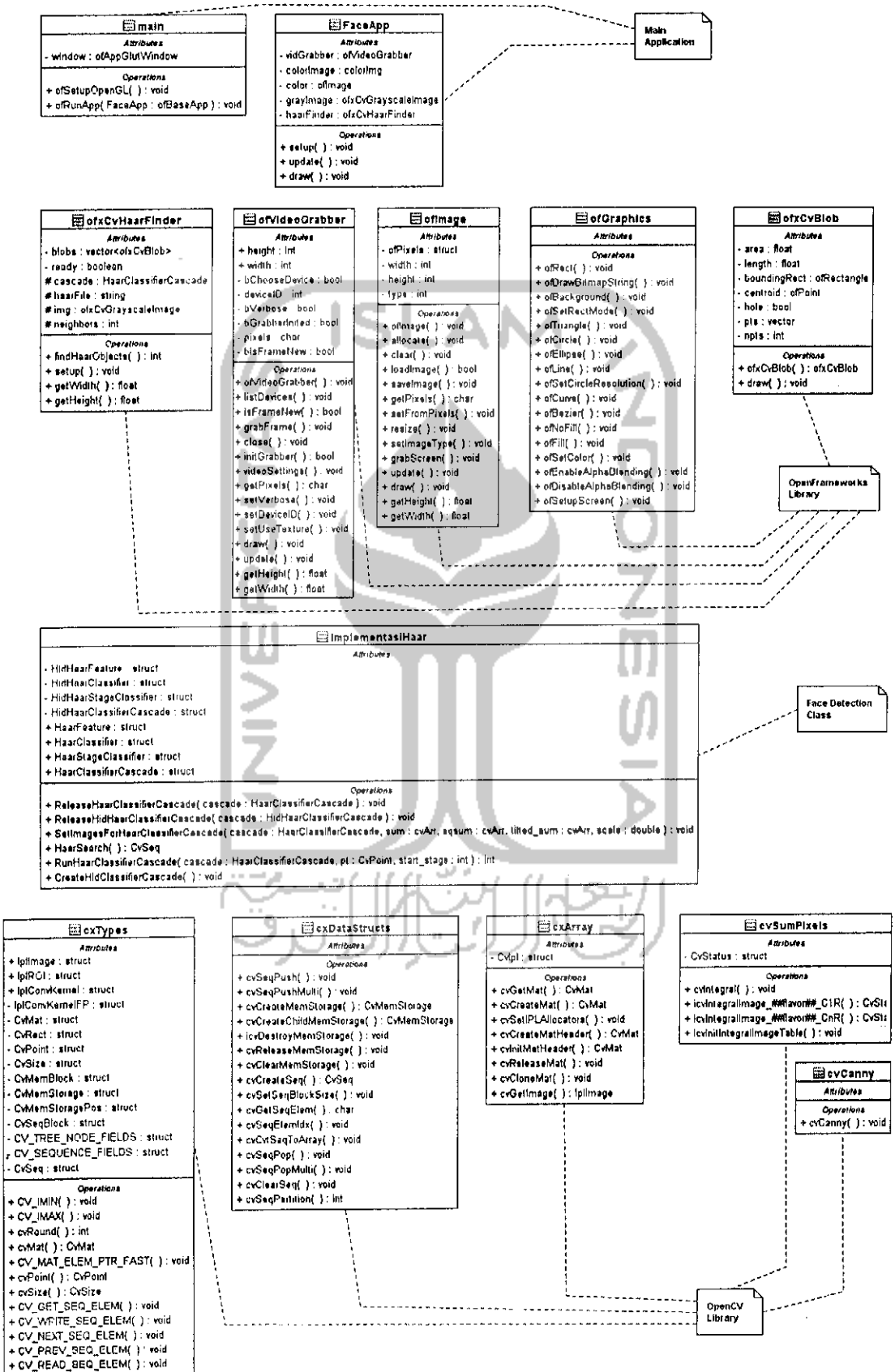
Gambar 5.5 – Activity Diagram

5.2.4 Class Diagram

Class diagram menunjukkan kelas-kelas yang terlibat pada keseluruhan proses secara teknis dan juga atribut-atribut serta operasi yang ada dalam kelas masing-masing. Diagram ini juga menggambarkan *multiplicity* yang ada pada kelas-kelas yang saling berhubungan, seperti tertera pada gambar 5.6. Kemudian gambar 5.7 menunjukkan atribut dan operasi yang ada pada diagram itu secara detail.



Gambar 5.6 – Class Diagram



Gambar 5.7 – Class Diagram Detail

5.2.5 Pseudo Code Fungsi

Pada perancangan ini juga dibuat suatu *pseudo-code* dari fungsi-fungsi yang terdapat pada file kelas `ImplementasiHaar.cpp`, dimana kelas ini adalah kelas yang menangani pendeteksian objek.

5.2.5.1 Fungsi `SetImageForHaarClassifierCascade`

Pseudo-code fungsi :

```
1. Mengubah citra inputan menjadi suatu array matrix dengan variabel cvMat.
2.membuat hidden cascade dengan memanggil fungsi CreateHidClassifierCascade() jika
hidden cascade pada cascade aslinya kosong.
3.mengisikan variabel 'sum' yang merupakan citra input ke dalam cascade asli,di var
'sum' juga.
4.membuat suatu persegi panjang bertipe cvRect. Mengatur ukuran real_window_size dari
cascade dengan disesuaikan skala yang telah didefinisikan di parameter fungsi.
5.mengisi variabel p0,p1,p2,p3 dari hidden cascadenya dan juga pq0,pq1,pq2,pq3.dgn
penghitungan :
        5.1.pastikan nilai penskalaan original window size dari real cascade lebih
kecil dari nilai integral imagenya.
        5.2.kemudian isi nilai dengan jumlah total panjang memory dan data pointer
dari sum dikali ukuran original window size,seperti ini:
                5.2.1.(mat).data.ptr + (size_t)(mat).step*(row) + (pix_size)*(col)
6.isikan elemen p0,p1,p2,p3 dari nilai persegi panjang node fitur hidden cascade
dengan penghitungan :
        6.3.hampir sama dengan langkah 5,parameternya tr(poin x & y rectangle dan
ukuran width & heightnya).
        6.4.Melakukan penghitungan poin x & y persegi panjang region :
                6.4.5.tr.x = r[k].x*scale
                6.4.6.tr.width = r[k].width*scale
                6.4.7.tr.y = r[k].y*scale
                6.4.8.tr.height = r[k].height*scale
7.isikan elemen pq0,pq1,pq2,pq3 dari nilai persegi panjang node fitur hidden cascade.
8.isi weight ke-[k] dari feature hidden node classifier dengan weight sebelumnya
dikalikan ((1/(equ_rect.width*equ_rect.height)) * 1).
9.isi weight ke-[0] dengan -sum0/area0
        9.1.isikan sum0 dengan weight ke-[k] sebelumnya * tr.width * tr.height
        9.2.isikan area0 dengan tr.width * tr.height
        9.3.isi tr.height & tr.width dengan rectangle ke-[k] dari fitur asli * scale.
10.hasil akhirnya adalah poin-poin elemen rectangle hidden cascade.
```

5.2.5.2 Fungsi `RunHaarClassifierCascade`

Pseudo-code fungsi :

```
1.mengecek apakah poin mulai (x,y) dimana fungsi ini berjalan tidak kurang dari 0
```

2.mengecek apakah real window size dari cascade tidak lebih besar dari matrix 'sum' dari hidden cascade.
3.menghitung rerata dengan penghitungan :
(p0(dari hidden cascade)[offset]-p1[offset]-p2[offset]+p3[offset]) * (1/(_cascade->orig_window_size.width-2)*scale)*(_cascade->orig_window_size.height-2)*scale)).
Offset adalah hasil penskalaan.
3.jika cascade adalah tree :

3.1.tentukan nilai sum dari penghitungan empat poin rectangle hidden node dan tentukan apakah melewati node kiri atau kanan.

3.2.kemudian jika nilainya lebih besar atau mencapai threshold,maka lanjutkan ke classifier child.

3.3.Jika tidak mencapai threshold, periksa jika penunjuk ke next-nya null,

3.4.jika null maka isi dengan nilai penunjuk ke stage classifier parent.kemudian isi penunjuk dengan penunjuk ke stage classifier next.

4.jika cascade adalah stump (ujung) :

4.1.isi sum dengan penghitungan poin-poin pojok rectangle hidden node [0] dan [1] dikalikan bobot masing-masing.

4.2.jika ada nilai rectangle ke[2],isikan juga ke sum.isi sum dari hidden stage classifier dengan sum hidden stage classifier sebelumnya ditambah : (nilai alpha ke-[0] hidden classifier jika 'sum' lebih kecil dari threshold,dan alpha ke-[1] jika sebaliknya).

4.3.kemudian jika penghitungan stage sum tersebut lebih kecil daripada threshold hidden stage classifier,maka nilai balikan fungsi ini adalah - (minus).

5.jika bukan tree atau stump : isi stage sum dengan penghitungan seperti langkah 3 dan jika lebih kecil dari threshold hidden stage classifier, maka berikan nilai balikan - (minus) dan keluar fungsi.
6.hasil akhirnya adalah nilai integer positif atau negatif.

5.2.5.3 Fungsi HaarSearch

Pseudo-code fungsi :

1.melakukan konversi ke integral image
2.melakukan canny pruning jika diperlukan.
3.melakukan penskalaan dari citra objek wajah yang telah ditentukan dalam file setting xml,(20x20),
4.sebelum melakukan pencarian, memanggil fungsi SetImagesForHaarClassifierCascade untuk menyiapkan cascade dan hidden cascade yang nantinya digunakan pada pencarian.
5.menentukan titik koordinat awal dimana harus dilakukan deteksi dan juga titik koordinat dimana harus menghentikan pencarian.
6. melakukan penghitungan pada 4 titik poin dari persegi panjang original window dari objek,yang tertera di file xml.
7.menentukan koordinat awal dan koordinat akhir,serta nilai 4 poin referensi array yang akan digunakan untuk penghitungan 'sum' integral image untuk memperoleh penjumlahan area dalam region tersebut.
8.fungsi mulai melakukan perulangan untuk mendeteksi objek :

8.1.memanggil fungsi RunHaarClassifierCascade dengan parameternya adalah cascade yang telah tersedia dan koordinat posisi dimana fungsi ini berjalan.

8.2.jika fungsi RunHaarClassifierCascade memberikan hasil positif, maka region yang ditandai akan disimpan ke suatu variabel sequence, dan akan pencarian akan dilanjutkan lagi dengan perulangan.

9.beberapa region yang lolos klasifikasi ditandai koordinat posisi dan ukurannya.

10.region persegi panjang yang saling berdekatan akan disatukan menjadi satu persegi panjang dan dianggap sebagai satu kesatuan objek.

11.nilai-nilai penghitungan langkah 10 akan dimasukkan ke suatu variabel nilai balikan berupa sequence.

BAB VI
IMPLEMENTASI

6.1. Batasan Implementasi

Implementasi adalah tahap dimana sistem siap diaplikasikan dan digunakan dalam keadaan sesungguhnya. Batasan implementasi dari sistem ini adalah suatu sistem yang dapat digunakan untuk membantu pengoptimalan penggunaan kamera pengawas (*surveillance camera*) pada ruangan-ruangan tertentu dengan mendeteksi objek wajah manusia dan menyimpan wajah yang tertangkap sebagai *file* gambar. Tujuan implementasi ini adalah untuk mengetahui apakah sistem yang telah dirancang sebelumnya telah berjalan dengan benar.

6.2. Spesifikasi Kebutuhan Sistem

6.2.1. Perangkat Lunak Yang Dibutuhkan

Spesifikasi perangkat lunak yang dibutuhkan untuk mengembangkan sistem ini adalah :

1. Code::Blocks sebagai *text editor* yang digunakan untuk menulis program dan GCC Compiler untuk mengcompile program yang telah dibuat.
2. Microsoft Visual C++ 2008 Express Edition untuk sebagai text editor untuk OpenCV.
3. NetBeans 6.5 untuk membuat diagram perancangan UML.
4. Notepad++ sebagai *text editor*.
5. Microsoft Windows Vista Business sebagai sistem operasi yang digunakan.

6.2.2. Perangkat Keras Yang Dibutuhkan

Sedangkan perangkat keras yang dibutuhkan untuk mengembangkan sistem ini adalah :

1. Komputer dengan prosessor Intel Pentium III, sekelasnya atau lebih

tinggi.

2. RAM 128 MB atau lebih tinggi.
3. Hardisk dengan space kosong 1 GB atau lebih.
4. VGA 32 MB atau lebih.
5. Monitor VGA / SVGA.
6. Web Camera Standard.
7. Mouse.
8. Keyboard.

6.3. Implementasi Sistem

6.3.1 Proses Pengambilan Citra Video

Karena keseluruhan sistem ini merupakan penggunaan dari *framework* C++ OpenFrameworks, maka struktur pada sistem ini mengikuti aturan dari pola *framework* tersebut. Aplikasi utama sistem pendeteksi wajah ini memanggil tiga fungsi dari OpenFrameworks yang sangat penting yaitu `setup()`, `update()`, dan `draw()`, seperti tertera pada *source code* berikut :

```
void FaceApp::setup()
{
	//code....
}
void FaceApp::update()
{
	//code....
}
Void FaceApp::draw()
{
	//code....
}
```

Objek `FaceApp` di atas merupakan pewarisan dari kelas dasar *library* OpenFrameworks `ofBaseApp` yang bertugas menangani operasi-operasi pada aplikasi yang akan dibuat. Oleh karena itu, kelas ini adalah kelas yang sangat penting karena merupakan basis dari operasi-operasi yang lebih rumit.

Sebelumnya, kelas-kelas dari *library* yang ada pada OpenFrameworks sudah dipanggil pada file *header*, seperti berikut :

```
void setup();
void update();
```

```
void draw();
ofVideoGrabber vidGrabber;
ofxCvColorImage colorImg;
ofImage color;
ofxCvGrayscaleImage grayImage;
ofImage gray;
ofxCvGrayscaleImage grayBg;
ofxCvGrayscaleImage grayDiff;
ofxCvHaarFinder haarFinder;
```

Kemudian, fungsi `setup()` disini bertugas menginisialisasi semua *library* yang akan digunakan, inisialisasi variabel-variabel, atau memuat file-file *external* yang dibutuhkan.

```
void FaceApp::setup()
{
	vidGrabber.setVerbose(true);
	vidGrabber.initGrabber(320,240);
	colorImg.allocate(320,240);
	grayImage.allocate(320,240);
	haarFinder.setup("haarcascade_frontalface_default.xml");
}
```

Source code di atas menunjukkan apa saja yang dilakukan pada operasi `setup()`. Pertama objek melakukan inisialisasi *library* video `vidGrabber` yang merupakan *instance* dari kelas `ofVideoGrabber`, dengan memanggil fungsi `setVerbose()` dan `initGrabber()`, memanggil komponen yang menangani pengambilan citra video dengan ukuran 320x240 *pixel*. Di dalam *library* `ofVideoGrabber` ini terdapat komponen dari quicktime yang menyediakan akses tingkat rendah kepada kamera video. Khusus di sistem operasi MS Windows, komponen ini menggunakan *library* berbasis *directshow* dan tidak memerlukan instalasi quicktime di sistem operasi. Kemudian aplikasi memanggil fungsi allocate() untuk menciptakan suatu *instance* kosong dengan tipe variabel `ofxCvGrayscaleImage` dan `ofxColorImage` berukuran 320x240 *pixel* yang nantinya akan digunakan untuk menyimpan citra video per-*frame*. Terakhir, aplikasi memanggil fungsi `setup()` dari `ofxCvHaarFinder` untuk memuat file xml yang nantinya akan digunakan pada proses pendeteksian objek.

Tahap berikutnya adalah proses yang terjadi pada fungsi `update()`. Fungsi `update()` ini melakukan operasinya tepat sebelum dilakukan fungsi `draw()`, dan fungsi ini dipanggil berulangkali. Jika ingin dilakukan penghitungan yang

berhubungan dengan sesuatu yang akan ditampilkan ke layar, maka idealnya operasinya diletakkan pada fungsi ini.

```
void FaceApp::update(){
    vidGrabber.grabFrame();
    if(vidGrabber.isFrameNew()){
        colorImg.setFromPixels(vidGrabber.getPixels(), 320, 240);
        grayImage = colorImg;
        haarFinder.findHaarObjects(grayImage);
    }
}
```

Source code di atas adalah detil dari operasi-operasi yang dilakukan fungsi `update()` pada aplikasi ini. Fungsi `grabframe()` digunakan untuk mengambil data *frame* baru dari video *grabber*. Fungsi ini akan menghentikan komponen video *grabber* sementara agar bisa didapat data *frame* baru. Kondisi `isFrameNew()` terpenuhi jika ada *frame* baru yang terdeteksi dari adanya *pixel* yang berbeda dari *pixel* sebelumnya pada *frame*. Kemudian jika kondisi tersebut terpenuhi, maka fungsi akan melakukan beberapa operasi tertentu. Pertama fungsi akan menyalin data dari *pixel-pixel* frame didapat dengan `setFromPixel()` untuk mengambil data dari video *grabber* pada frame tersebut dan disalin ke dalam variabel `colorImg`. Kemudian citra yang telah ditangkap yang sebelumnya berwarna dikonversi menjadi *grayscale* dengan melemparnya ke variabel `grayImage`. Dari citra *grayscale* ini akan dilempar ke fungsi `findHaarObjects` yang nantinya akan memanggil kelas yang bertugas melakukan pendeteksian objek wajah.

6.3.2 Proses Pendeteksian Objek

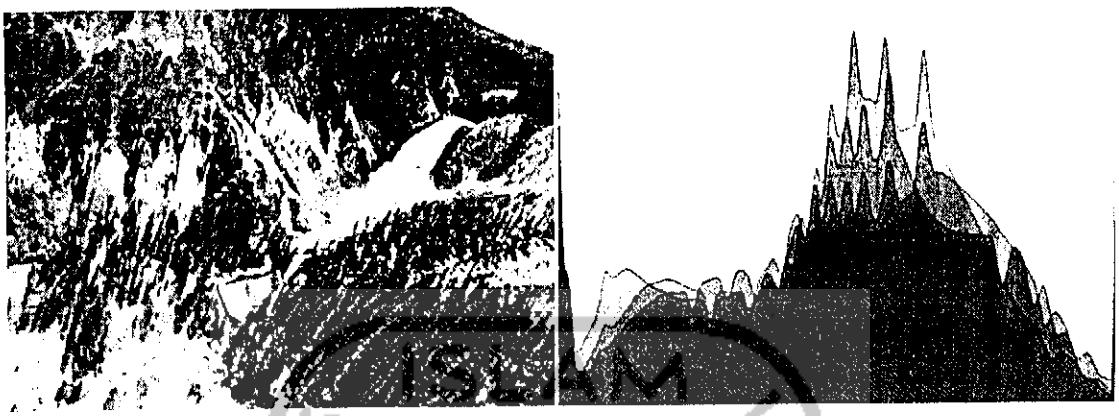
Citra video per-frame yang telah dikonversi menjadi *grayscale* ini diproses oleh fungsi `findHaarObjects()` yang berasal dari objek `haarFinder` yang merupakan instansiasi dari kelas `ofxCvHaarFinder`, dimana kelas ini hanya merupakan penghubung antara aplikasi utama openFrameWorks dengan fungsi yang sebenarnya menangani pendeteksian objek. Berikut adalah bagian dari fungsi

```
int ofxCvHaarFinder::findHaarObjects(
ofxCvGrayscaleImage& input,int x, int y, int w, int h, int minWidth,
int minHeight) {
      int nHaarResults = 0;
      if (cascade) {
            if (!blobs.empty())blobs.clear();
            if (img.width==input.width&&img.height==input.height)
```

```
        {img = input;}
        else
        {img.clear();
         img.allocate(input.width, input.height);
         img = input;
        }
img.setROI(x, y, w, h);
cvEqualizeHist(img.getCvImage(), img.getCvImage());
CvMemStorage* storage = cvCreateMemStorage();

CvSeq* haarResults =
      HaarSearch(img.getCvImage(),      cascade,      storage,
        scaleHaar,neighbors,CV_HAAR_DO_CANNY_PRUNING,
        cvSize(minWidth, minHeight));
//.....
```

Kode di atas menunjukkan proses pada saat sebelum pemanggilan fungsi untuk mendeteksi objek yaitu `HaarSearch`. Sebelumnya proses memeriksa apakah file setting xml yang berisi *cascade* sudah dimuat atau belum, dan kemudian menyalin citra yang akan diperiksa ke dalam suatu variabel lain, karena akan dilakukan *histogram equalization*. *Histogram equalization* adalah proses untuk menyeimbangkan gangguan yang ada pada saat penangkapan citra dan perbedaan kontras pada citra yang kadang terlalu tinggi dan membuat citra semakin sulit untuk dianalisa. Gambar 6.1 berikut menunjukkan contoh proses *histogram equalization*:



Gambar 6.1 – Contoh Histogram Equalization

Setelah dilakukan *histogram equalization*, fungsi akan menciptakan suatu variabel storage dengan tipe data adalah `CvMemStorage` dan memanggil fungsi `cvCreateMemStorage()`. Tipe data ini adalah struktur tingkat rendah yang dapat digunakan untuk menyimpan struktur data yang dapat bertambah besar secara dinamis seperti *sequence*, kontur, dan lain-lain. Tipe data ini merupakan tipe data bawaan dari framework OpenCV. Nantinya tipe data ini akan digunakan pada saat pemrosesan pendeteksian objek.

6.3.2.1 Fungsi `HaarSearch`

Pada aplikasi ini, fungsi `HaarSearch` dan fungsi-fungsi lain yang ada pada file "implementasiHaar.cpp" merupakan fungsi-fungsi yang krusial pada proses pendeteksian objek dari citra. Fungsi `ofxCvHaarFinder` yang disebutkan sebelumnya hanya merupakan penghubung antara frameworks OpenFrameworks dengan fungsi pendeteksian objek dan juga framework OpenCV. Setelah melakukan beberapa hal pada citra, objek `ofxCvHaarFinder` akan memanggil fungsi `HaarSearch()` yang ada pada file "implementasiHaar.cpp".

```
CvSeq* haarResults =
      HaarSearch(img.getCvImage(),      cascade,      storage,
        scaleHaar,neighbors,CV_HAAR_DO_CANNY_PRUNING,
        cvSize(minWidth, minHeight));
```

Fungsi `HaarSearch()` ini mempunyai nilai balikan `CvSeq` yang merupakan hasil dari pendeteksian objek. `CvSeq` ini adalah tipe data *sequence* bawaan dari OpenCV, dan merupakan data struktur yang bersifat dinamis.

Fungsi `HaarSearch` yang berfungsi melakukan pendeteksian objek ini ada di dalam file "implementasiHaar.cpp", dan prosesnya banyak sekali dibantu oleh frameworks OpenCV untuk melakukan beberapa analisa dan penyesuaian pada citra.

```
HaarSearch(const CvArr* _img,HaarClassifierCascade* cascade,
        CvMemStorage* storage, double scale_factor,
        int min_neighbors, int flags, CvSize min_size)
{
    CvMat *temp=0, *sum=0, *tilted=0, *sqsum=0, *norm_img=0,
*sumcanny=0, *img_small=0;
    CvSeq* result_seq = 0;
    CvMemStorage* temp_storage = 0;
    CvAvgComp* comps = 0;
    //.....
```

Untuk input dari citra yang akan diperiksa, parameter pada fungsi di atas menunjukkan variabel `_img` sebenarnya bertipe `IplImage` yang merupakan tipe data primitif OpenCV berbentuk struktur yang digunakan untuk menyimpan citra. Citra ini bisa berupa *grayscale*, berwarna, empat-*channel* (RGB+*alpha*), dan setiap *channel* berupa integer atau float, dan tipe `IplImage` disini merupakan turunan dari tipe variabel `CvMat` yang berfungsi menyimpan *multi-channel matrix*. Intinya, citra input disimpan sebagai *multi-channel matrix* yang pada setiap elemennya mengandung spesifikasi setiap *pixel* pada citra tersebut, seperti RGB+A atau *grayscale*. Pada parameter disebutkan bertipe `CvArr`, hanya untuk menentukan bahwa fungsi ini menerima array yang bisa berupa lebih dari satu tipe, seperti `IplImage`, `CvMat`, atau `CvSeq`; seperti disebutkan pada dokumentasi fungsi OpenCV. Kemudian setelahnya menunjukkan inisialisasi variabel-variabel awal yang dibutuhkan oleh fungsi `HaarSearch` agar dapat berjalan. Variabel-variabel dengan tipe CvMat di atas akan digunakan untuk pemrosesan citra dan penyimpanan sementara dari hasil pemrosesan seperti citra *integral* yang akan dibahas nanti. Berikutnya adalah inisialisasi variabel dengan tipe `CvMemStorage*`. Tanda '*' disini menunjukkan pointer ke *memory*. Tipe data `CvMemStorage` adalah salah satu tipe data bawaan OpenCV yang dapat digunakan untuk menyimpan rangkaian blok memori berupa *linked list*, dan tipe ini menyediakan alokasi dan dealokasi cepat dari

suatu set yang berkesinambungan. Pada fungsi ini variabel dengan tipe `CvMemStorage` digunakan sebagai tempat penyimpanan sementara dari variabel `storage` yang dimasukkan dari parameter fungsi `HaarSearch`, dimana variabel ini nantinya akan terlibat pada proses menyimpan hasil deteksi objek. Kemudian berikutnya inisialisasi variabel `result_seq` yang berupa `CvSeq`, yang nantinya juga akan digunakan untuk menyimpan hasil deteksi objek. `CvSeq` ini adalah tipe data *sequence* yang dapat disimpan pada *memory storage* dengan tipe `CvMemStorage` seperti disebutkan sebelumnya. Sequence ini berupa *linked list* dari stuktur data suatu objek lain, yang dalam kasus ini digunakan untuk menyimpan hasil dari pendeteksian objek. Untuk variabel dengan tipe `CvAvgComp` digunakan untuk menggabungkan region persegi panjang hasil deteksi objek.

Langkah berikutnya, fungsi mengambil *matrix* dari input pada variabel `_img` yang berupa `IplImage` dan melemparnya ke variabel `img`; kemudian menciptakan variabel-variabel sementara berupa *matrix* dengan ukuran baris dan kolom sesuai dengan matrix `img`. Fungsi memeriksa apakah input yang ada sudah berupa citra *grayscale* atau belum, dan jika belum maka akan citra yang ada akan dikonversi menjadi citra *grayscale*. Berikutnya fungsi melakukan penghitungan citra *integral*, dimana penghitungan ini adalah salah satu hal yang cukup penting dalam pendeteksian objek, dan dilakukan dengan memanggil fungsi `CvIntegral` dari OpenCV.

```
cvIntegral( img, sum, sqsum, tilted );
```

Pemanggilan fungsi di atas berfungsi untuk melakukan penghitungan dari *matrix* input menjadi suatu citra *integral*. Misalkan ada suatu citra normal sebelumnya yang berbentuk matrix 3x3 :

$$x_1y_1 \; x_2y_1 \; x_3y_1$$

$$x_1y_2 \; x_2y_2 \; x_3y_2$$

$$x_1y_3 \; x_2y_3 \; x_3y_3$$

Dimana x dan y di atas adalah koordinat dari tiap *pixel*. Maka nilai citra *integral* dari pixel(x, y) adalah penghitungan dari *pixel* di atas dan kiri dari pixel(x, y) tersebut. Integral Image didapatkan dengan menggunakan setiap titik pojok dari persegi

panjang region citra atau fitur Haar. Dengan mengambil contoh citra 3x3 *pixel*, maka penghitungan citra *integral* dari citra normal di atas adalah :

- Baris pertama

$x_1y_1=(x_1y_1)$; $x_2y_1=(x_2y_1+x_1y_1)$; $x_3y_1=(x_3y_1+(x_2y_1'))$

- Baris kedua

$x_1y_2=(x_1y_2+x_1y_1)$; $x_2y_2=((x_1y_2'+x_2y_1')-(x_1y_1-x_2y_2))$; $x_3y_2=((x_2y_2'+x_3y_1')-(x_2y_1'-x_3y_2))$

- Baris ketiga

$x_1y_3=(x_1y_2'+x_1y_3)$; $x_2y_3=((x_1y_3'+x_2y_2')-(x_1y_2'-x_2y_3))$; $x_3y_3=((x_2y_3'+x_3y_2')-(x_2y_2'-x_3y_3))$

1	2	5	1	2
2	20	50	20	5
5	50	100	50	2
2	20	50	20	1
1	5	25	1	2
5	2	25	2	5
2	1	5	2	1

1	3	8	9	11
3	25	80	101	108
8	80	235	306	315
10	102	307	398	408
11	108	338	430	442
16	115	370	464	481
18	118	378	474	492

Gambar 6.2 – Contoh penghitungan citra *integral*

Gambar 6.2 menunjukkan hasil penghitungan dari citra *integral* suatu citra.

Setelah berhasil mendapatkan penghitungan citra *integral* dari citra input, proses melakukan iterasi untuk melakukan pendeteksian objek. Iterasi ini dilakukan berdasarkan faktor penskalaan dari fitur Haar yang telah dispesifikasikan dalam file xml *cascade* dan sampai kondisi tertentu. Didalam iterasi inilah mulai dilakukan tahap-tahap dalam proses pendeteksian objek. Pertama dilakukan inisialisasi koordinat (`x, y`) awal dari *pixel* citra input yang akan dianalisa :

```
int start_x = 0, start_y = 0;
int end_x = cvRound((img->cols - win_size.width) / ystep);
int end_y = cvRound((img->rows - win_size.height) / ystep);
```

Seperti pada *source code* di atas, untuk koordinat awal yaitu `start_x` dan `start_y` dimulai dari 0, dan untuk koordinat akhirnya `end_x` dan `end_y` adalah merupakan ukuran citra dikurangi dengan ukuran region yang telah dispesifikasikan pada cascade dibagi dengan `ystep` yang merupakan skala. Berikut adalah deklarasi dari region variabel `win_size` :

```
CvSize win_size= {cvRound(cascade->orig_window_size.width*factor),
                  cvRound(cascade->orig_window_size.height*factor)};
```

Variabel yang dimaksud pada `cascade->orig_window_size` mengacu pada spesifikasi ukuran region citra yang ada pada file cascade xml. Kemudian didalam proses ini dipanggil fungsi `SetImagesForHaarClassifierCascade` yang secara sederhananya berfungsi melakukan pengaturan pada *cascade* dan *hidden cascade* dengan melakukan penghitungan pada poin-poin pojok dari fitur Haar persegi panjangnya agar nanti mudah ketika dilakukan pembandingan dengan citra input. Setelah melakukan pengaturan ulang pada *cascade* dan *hidden cascade*, fungsi mendeklarasikan iterasi lagi untuk memulai pencarian, kali ini dimulai dengan titik awal `start_y` dan `start_x`.

```
for( int _iy=start_y; _iy<end_y; _iy++ )
{
        //...
        for( _ix=start_x; _ix<end_x; _ix+=_xstep )
            {
            //...
            }
}
```

Kemudian didalam iterasi ini, barulah dilakukan pembandingan antara citra input dengan fitur persegi panjang Haar. Fungsi yang bertugas melakukan pembandingan ini adalah fungsi `RunHaarClassifierCascade()`, dimana fungsi ini hanya melakukan pembandingan pada point tertentu saja, dengan nilai balikan positif atau negatif. Proses pada fungsi `RunHaarClassifierCascade` akan dijelaskan setelah penjelasan tentang *cascade* Haar.

6.3.2.2 Cascade Pengklasifikasi Haar

Fitur haar dihasilkan oleh algoritma pembelajaran AdaBoost. Setiap fitur Haar mempunyai nilai yang dihitung dengan menimpa fitur di atas citra,dan kemudian menentukan perbedaannya dari *threshold*. Dengan mengambil area dari setiap region pada fitur, mengkalikannya dengan bobot masing-masing, dan kemudian menjumlahkan hasilnya, didapat nilai dari fitur. Area dari setiap region persegi panjang tersebut dapat ditemukan dengan mudah melalui fungsi Integral Image.

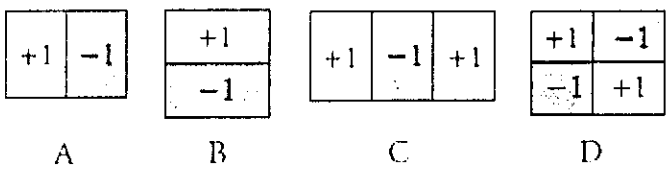
Suatu cascade haar menerjemahkan suatu rangkaian dari operasi deteksi fitur dalam suatu struktur *tree*. Pada setiap *stump-based* Haar *cascade*, decision pointnya

bisa menghasilkan keputusan lolos atau gagal. Karena itu, suatu *stump-based* Haar *cascade* bekerja pada suatu rangkaian langkah-langkah yang hampir sama. Setiap langkah menerapkan suatu grup dari operasi deteksi fitur pada citra dan menjumlahkan hasilnya. Jika jumlahnya melebihi *threshold*, maka klasifikasi Haar maju ke langkah berikutnya, jika suatu langkah gagal, maka citra yang diperiksa akan dilewati dan digolongkan sebagai bukan wajah.

Haar *cascade* diimplementasikan sebagai suatu class bersarang dalam `HaarClassifierCascade` dalam suatu file XML. Level terendah dari pendeteksian fitur diselesaikan dengan menggunakan `HaarFeature`. Pada level terendah, Haar *cascade* terdiri atas pendeteksi fitur yang berupa region persegi-panjang dari citra yang telah dijumlahkan, dikalikan dengan koefisien, dan ditambahkan bersama. Fitur haar selalu mempunyai dua atau tiga region seperti di atas. Region yang tepat untuk dijumlahkan telah diterangkan secara spesifik pada XML file `haarcascade_frontalface_alt.xml` dengan memberikan koordinat kiri-atas dari persegi-panjang, panjang dan lebarnya, dan juga koefisiennya, secara berurutan.

```
<feature>
        <rects>
            <_>3 7 14 4 -1.</_>
            <_>3 9 14 2 2.</_>
        </rects>
        <tilted>0</tilted>
<feature>
```

Spesifikasi fitur haar di atas menjelaskan dua persegi panjang, satu dimulai pada koordinat (3,7) dan mempunyai panjang 14 *pixel* dan tinggi 4 *pixel*, dan dikalikan dengan -1. Persegi panjang kedua dimulai dari koordinat (3,9) dan mempunyai panjang 14 *pixel* dan tinggi 2 *pixel*, dan dikalikan oleh 2. Kedua persegi panjang ini akan membentuk suatu pengklasifikasi haar berbentuk persegi panjang seperti ditunjukkan pada gambar 6.3 fitur B.



Gambar 6.3 – Fitur Persegi Panjang Haar

Ketika file xml dari fitur Haar sudah dimuat oleh aplikasi menggunakan `cvLoad`, maka akan dideklarasikan sebagai variabel dengan bentuk *tree* sebagai berikut :

```
HaarClassifierCascade:
    HaarStageClassifier1:
        HaarClassifier11:
            HaarFeature1
        HaarClassifier12:
            HaarFeature1
        ...
    HaarStageClassifier2:
        HaarClassifier21:
            HaarFeature1
        ...
    ...
```

Sedangkan struktur dari file xml-nya adalah sebagai berikut :

```
<opencv_storage>
<haarcascade_frontalface_alt type_id="opencv-haar-classifier">
  <size>20 20</size>
  <stages>
    <_>
      <!-- stage 0 -->
      <trees>
        <_>
          <!-- tree 0 -->
          <_>
            <!-- root node -->
            <feature>
              <rects>
                <_>3 7 14 4 -1.</_>
                <_>3 9 14 2 2.</_></rects>
              <tilted>0</tilted></feature>
            <threshold>4.0141958743333817e-003</threshold>
            <left_val>0.03379419073462491</left_val>
            <right_val>0.83781069517135620</right_val></_></_>
        <_>
          <!-- tree 1 -->
          <_>
            <!-- root node -->
            <!-- ......... -->
          </_>
        </_>
        <_>
            <!-- tree 2 -->
            <!-- .....  -->
        </_>
      </trees>
      <stage_threshold>0.82268941402435303</stage_threshold>
      <parent>-1</parent>
      <next>-1</next></_>
<!-classifier berlanjut... -->
  </stages>
```

6.3.2.3 Fungsi RunHaarClassifierCascade

Fungsi ini bertugas melakukan pendeteksian menggunakan pengklasifikasi Haar pada region tertentu pada citra input. Parameter `_cascade` menunjukkan susunan *cascade* pengklasifikasi Haar yang dipakai, `pt` menunjukkan poin pojok kanan atas pada region yang akan dianalisa, dan `start_stage` menunjukkan mulai dari *stage* (seperti dijelaskan pada penjelasan tentang cascade pengklasifikasi Haar) berapa *cascade* akan digunakan, tetapi biasanya 0.

```
int RunHaarClassifierCascade(CascadeHaarClassifier* _cascade,
CvPoint pt, int start_stage)
```

Sebelumnya fungsi ini akan memanggil *hidden cascade* yang sebelumnya telah dideklarasikan oleh fungsi `SetImagesForHaarClassifierCascade`, tetapi tetap menggunakan *cascade* aslinya. Kemudian melakukan iterasi sesuai dengan *cascade* dan *hidden cascade* dimulai dari *stage* yang telah ditentukan sebelumnya, didalam iterasi inilah dilakukan pembandingan antara perhitungan *integral* dari citra input dengan perhitungan *integral* dari fitur persegi panjang Haar yang telah dispesifikasikan pada *cascade*. Inti dari perhitungan fitur atau citra inputnya adalah:

$$sum = pointX_1Y_1 - pointX_2Y_2 - pointX_3Y_3 + pointX_4Y_4$$

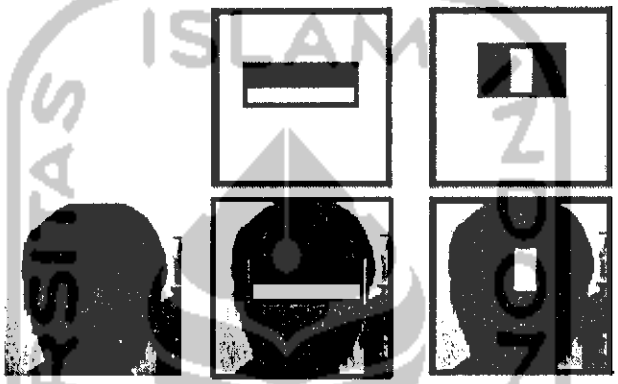
Dari hasil perhitungan di atas, untuk dibandingkan masih dikalikan dengan bobot masing-masing, dan bobot ini telah ditentukan pada spesifikasi *cascade*. Yang dimaksud point di atas adalah point kiri atas, kanan atas, kiri bawah dan kanan bawah dari region citra yang akan dianalisis dan juga fitur Haar yang digunakan. Untuk point-point pojok yang akan dihitung di atas sebelumnya harus diproses menjadi citra *integral* dahulu menggunakan fungsi OpenCV `cvIntegral`. Alasan penggunaan penghitungan *integral* adalah untuk mengetahui kalkulasi dari suatu region, dengan hanya menggunakan empat array referensi poin-poin pojoknya, dan ini dapat mempercepat proses pendeteksian. Baris kode yang melakukan penghitungan adalah:

```
sum=calc_sum(node->feature.rect[0],p_offset)*node->feature.rect[0].weight;
```

Dimana fungsi calc_sum :

```
calc_sum(rect,offset) =
((rect).p0[offset]-(rect).p1[offset]-(rect).p2[offset]+(rect).p3[offset]);
```

Penghitungan ini dilakukan pada setiap bagian dari fitur persegi panjang, dan p0,p1,p2 dan p3 menunjukkan poin-poin pojoknya. Jika fitur terdiri dari 2 persegi panjang berbeda, maka dilakukan dua kali penghitungan dengan index dari `rect` (region persegi panjang) sesuai urutan bagian dari fitur. Jika fitur terdiri dari tiga bagian, maka dilakukan tiga kali penghitungan. Dari hasil penghitungan ini kemudian dikalikan bobot.



Gambar 6.4 – Contoh Pembandingan Citra

Gambaran secara sederhana dapat dilihat pada gambar 6.4. Pada intinya, fungsi ini melakukan penghitungan empat poin pojok dari citra *integral* fitur persegi panjang Haar dengan bobot pada *cascade*, kemudian dibandingkan dengan *threshold*, dimana *threshold* ini adalah hasil penghitungan antara empat poin pojok citra *integral input* yang dianalisa dengan reratanya. Setelah didapatkan nilainya, fungsi melakukan pengecekan apakah penghitungan *threshold* dari citra *input* kurang dari atau lebih dari threshold yang telah dispesifikasikan pada *cascade* :

```
if(stage_sum < cascade->stage_classifier[i].threshold)
{     result = -i;
      goto exit;
}
```

Jika nilainya kurang dari *threshold* dari *cascade*, maka fungsi akan memberikan nilai balikan negatif dan keluar menyelesaikan operasi, tetapi jika tidak maka fungsi akan memberikan nilai positif sebagai nilai balikan.

Kemudian setelah fungsi `HaarSearch` selesai menjalankan fungsi `RunHaarClassifierCascade` pada koordinat tertentu citra input, fungsi melakukan

pengecekan apakah nilainya negatif atau positif, dan jika positif maka hasil pencarian akan dimasukkan ke variabel *sequence*, seperti pada kode berikut :

```
{
      CvRect rect = cvRect(ix,iy,win_size.width,win_size.height);
      cvSeqPush( seq_thread[thread_id], &rect );
}
```

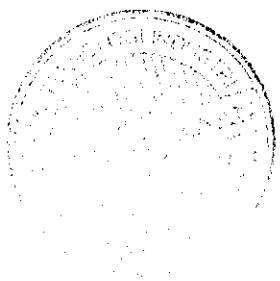
Variabel bertipe `CvRect` di atas adalah variabel primitif bawaan dari OpenCV untuk menggambarkan suatu struktur persegi panjang. Pada kode di atas, `ix` dan `iy` menunjukkan koordinat ditemukannya objek, dan `win_size.width` dan `win_size.height` adalah region pencariannya yang berarti ukuran dari objek. Kemudian kode berikutnya adalah proses memasukkan hasil ke dalam variabel *sequence*. Dari hasil akhir variabel *sequence* ini masih harus diproses lagi untuk menjadi suatu hasil deteksi yang lengkap. Sebelum digabungkan, region persegi panjang yang didapat masih berdekatan dan saling bertumpuk.

Setelah mengumpulkan hasil pencarian yang ditemukan, fungsi `HaarSearch` selanjutnya melakukan operasi untuk menggabungkan hasil pencarian yang berdekatan agar dapat digunakan untuk menandai objek wajah secara keseluruhan.

```
for(i=0; i<ncomp; i++)
{
      int n = comps[i].neighbors;
      if( n>=min_neighbors )
      {
            CvAvgComp comp;
            comp.rect.x = (comps[i].rect.x*2 + n)/(2*n);
            comp.rect.y = (comps[i].rect.y*2 + n)/(2*n);
            comp.rect.width=(comps[i].rect.width*2+n)/(2*n);
            comp.rect.height=(comps[i].rect.height*2 + n)/(2*n);
            comp.neighbors = comps[i].neighbors;
            cvSeqPush(seq2,&comp);
      }
}
```

Kode di atas berguna menghitung persegi panjang dari hasil klasifikasi Haar yang saling berdekatan untuk digabungkan menjadi satu persegi panjang yang menunjukkan region ditemukannya objek. Kemudian dari hasil penghitungan tersebut dimasukkan kembali ke dalam suatu variabel *sequence*, dan dilempar lagi ke dalam variabel balikan fungsi.

```
cvSeqPush(result_seq, &r1);
return result_seq;
```



Ketika semua operasi sudah selesai, maka hasilnya adalah variabel *sequence* yang berisi hasil pencarian objek, dan kemudian diproses kembali oleh aplikasi utama dan framework OpenFrameworks.

6.3.3 Proses Menandai Objek

Setelah didapatkan hasil deteksi objek dari proses yang dilakukan oleh fungsi HaarSearch, maka hasil tersebut akan diproses oleh OpenFrameworks untuk menandainya ke layar.

```
CvSeq* haarResults =
    HaarSearch(    img.getCvImage(), cascade, storage, scaleHaar,
    neighbors, CV_HAAR_DO_CANNY_PRUNING,cvSize(minWidth,
    minHeight));

nHaarResults = haarResults->total;
```

Kode di atas berjalan pada fungsi `findHaarObjects` yang ada di kelas ofxCvHaarFinder, dimana setelah fungsi `HaarSearch` selesai dipanggil, maka fungsi tersebut akan melempar variabel balikannya ke dalam variabel `haarResults` yang bertipe *sequence*. Kemudian variabel `nHaarResults` adalah variabel dengan tipe integer yang akan diisi oleh jumlah total dari isi *sequence*.

```
for (int i = 0; i < nHaarResults; i++ )
{
      ofxCvBlob blob;
```

Iterasi di atas melakukan pengulangan mulai dari i=0 sampai jumlah total isi dari *sequence* hasil deteksi objek yang dilakukan fungsi `HaarSearch()`.

```
      CvRect* r = (CvRect*) cvGetSeqElem(haarResults, i);
      float area = r->width * r->height;
      float length = (r->width * 2) + (r->height * 2);
      float centerx     = (r->x) + (r->width / 2.0);
      float centery     = (r->y) + (r->height / 2.0);
```

Pertama fungsi mengambil elemen tertentu dari *sequence* `haarResults` pada index `i` menggunakan fungsi `cvGetSeqElem`, kemudian mendeklarasikan elemennya masing-masing pada variabel lain, dimana area adalah luas dari region ditemukannya objek, dan length adalah keliling dari region, `centerx` dan `centery` adalah titik tengah dari region.

```
      blob.area = fabs(area);
```

```
        blob.hole = area < 0 ? true : false;
        blob.length = length;
        blob.boundingRect.x = r->x + x;
        blob.boundingRect.y = r->y + y;
        blob.boundingRect.width = r->width;
        blob.boundingRect.height = r->height;
        blob.centroid.x = centerx;
        blob.centroid.y = centery;
```

Blob disini adalah variabel bertipe `ofxCvBlob`, yang merupakan tipe variabel dari OpenFrameworks yang bertipe struktur. Variabel dengan tipe ini biasa digunakan untuk menyimpan spesifikasi suatu area atau objek pada suatu citra. Variabel `area` digunakan untuk menyimpan luas, `hole` untuk menandakan apakah regionnya kosong atau tidak, `boundingRect` berfungsi untuk menyimpan spesifikasi region dengan tipe `ofRect` yang berisi koordinat (`x, y`) dan ukuran (`width, height`), dan `centroid` disini adalah koordinat titik tengah dari *blob*.

```
        blob.pts.push_back(ofPoint(r->x, r->y));//top left corner
        blob.pts.push_back(ofPoint(r->x + r->width, r->y));//top right
corner
        blob.pts.push_back(ofPoint(r->x + r->width, r->y + r-
>height));//bottom right corner
        blob.pts.push_back(ofPoint(r->x, r->y + r->height));//bottom
left corner
        blobs.push_back(blob);
}
```

Kemudian `pts` disini menandakan point atau koordinat, yang biasanya digunakan untuk menyimpan titik-titik suatu kontur, tetapi disini digunakan untuk menyimpan koordinat titik-titik pojok dari persegi panjang region. Karena bertipe vektor, maka digunakan `push_back` untuk memasukkan koordinat-koordinat tersebut ke dalam variabel `pts`. Setelah itu, keseluruhan elemen blob dimasukkan ke dalam variabel `blobs` yang bertipe sama dengan `blob`, tetapi berupa vektor, sehingga bisa menyimpan lebih dari satu grup elemen. Variabel ini akan digunakan untuk menandai objek nantinya, pada aplikasi utama.

Aplikasi utama yang memanggil fungsi `findHaarObjects` dari `ofxCvHaarFinder` pada fungsi `update`, menerima nilai berupa `blobs` yang berisi spesifikasi hasil pendeteksian. Pada aplikasi utama, proses menandai objek terjadi pada fungsi `draw()`. Fungsi `draw` disini melakukan proses menampilkan tampilan

live video ke layar dan juga menandai objek dengan menggunakan persegi panjang secara bersamaan.

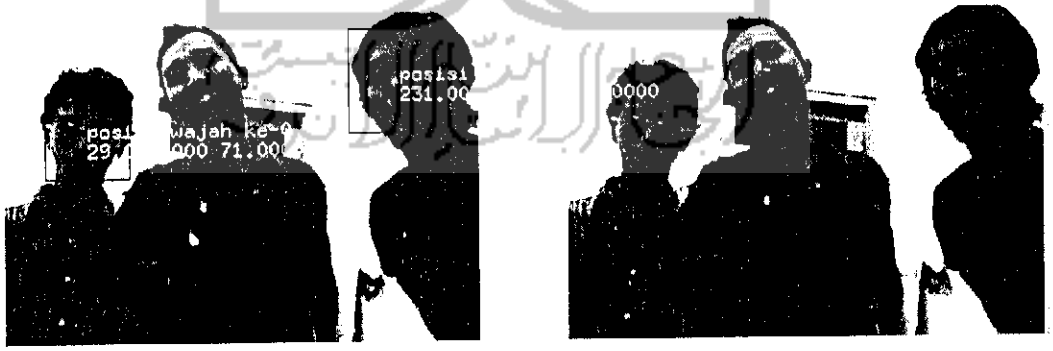
```
void FaceApp::draw()
{
      //....

      colorImg.draw(20,20);
      grayImage.draw(400,20);
```

Kedua fungsi `draw` pada `colorImg` dan `grayImage` di atas berfungsi menampilkan video hasil tangkapan kamera ke layar dengan parameternya adalah posisinya pada window. Fungsi `draw` pada `colorImg` berfungsi menampilkan video berwarna, dan pada `grayImage` berfungsi menampilkan video dengan *grayscale*.

Kemudian proses melakukan iterasi sesuai jumlah ditemukannya objek, dan didalam iterasi tersebut dilakukan proses menandai objek.

```
ofEnableAlphaBlending();
ofNoFill();
for(int i=0; i<numFaces; i++){
  float x = haarFinder.blobs[i].boundingRect.x;
  float y = haarFinder.blobs[i].boundingRect.y;
  float w = haarFinder.blobs[i].boundingRect.width;
  float h = haarFinder.blobs[i].boundingRect.height;
  float cx = haarFinder.blobs[i].centroid.x;
  float cy = haarFinder.blobs[i].centroid.y;
  ofSetColor(0x000000); //rectangle color
  ofRect(x, y, w, h);
```



Gambar 6.5 – Menandai objek

Fungsi `ofRect` berfungsi menggambar persegi panjang dengan koordinat dan posisi sesuai parameter yang dimasukkan ke fungsi tersebut, seperti terlihat pada gambar 6.5. Parameter `x,y,w,h` adalah nilai-nilai sesuai hasil dari pemanggilan fungsi pendeteksian objek yang telah dibahas sebelumnya. Sebelumnya dipanggil fungsi `ofEnableAlphaBlending` dan `ofNoFill` agar persegi panjang yang digambar oleh

`ofRect` transparan dan kosong ditengahnya. Fungsi `ofSetColor` berguna untuk menentukan warna dari persegi panjang.

6.3.4 Proses Menyimpan Objek Yang Ditemukan

Proses menyimpan objek adalah proses yang sederhana, dibandingkan dengan proses-proses lain, karena menggunakan fitur dari framework OpenFrameworks.

```
        int day = ofGetDay();
        int month = ofGetMonth();
        int year = ofGetYear();
        int sec = ofGetSeconds();
        int min = ofGetMinutes();
        int hour = ofGetHours();

        if(haarFinder.blobs[i].boundingRect.x)
        {
            gray.grabScreen(cvRound(x),cvRound(y),cvRound(w+50),cvRo
            und(h+50));

            gray.saveImage("capture"+ofToString(day)+ofToString(mont
            h)+ofToString(year)+"-
            "+ofToString(hour)+ofToString(min)+ofToString(sec)+"-
            "+ofToString(i)+".jpg");
        }

    }
```

Seperti terlihat pada kode di atas, fungsi yang mengambil *screenshot* dari objek yang terdeteksi adalah fungsi `grabScreen` dan `saveImage` yang merupakan inheritance dari `ofGrayScaleImage`. Pertama fungsi ini mengambil data dari *pixel* pada layar video dengan parameternya adalah koordinat pojok kiri atas dan panjang serta lebarnya. Data untuk parameter ini didapat dari hasil deteksi objek, sedangkan untuk panjang dan lebar (`w` dan `h`) ditambah 50 *pixel*, agar dapat menangkap keseluruhan objek yang ditemukan. Karena sebelumnya berupa *float*, masing-masing parameter tersebut diubah menjadi *integer* dengan menggunakan fungsi `cvRound`. Kemudian setelah itu dipanggil fungsi `saveImage` untuk menyimpan *screenshot* tersebut ke dalam *filesystem*, dengan nama filenya adalah sesuai yang tertera pada parameter pemanggilan fungsi tersebut. Untuk nama filenya menggunakan "capture" ditambahkan tanggal dan waktu dari ditemukannya objek, dengan ekstensi file adalah "*.jpg". Parameter waktu yaitu `day`, `month`, `year`, `sec`, `min`, dan `hour` sudah

ditentukan pada saat sebelum pemanggilan fungsi penyimpanan citra. Gambar 6.6 menunjukkan hasil penyimpanan citra deteksi. Nama filenya misalkan "capture1312010-231913-0.jpg".

Gambar 6.6 Hasil deteksi yang tersimpan

BAB VII
PENGUJIAN SISTEM

7.1. Pengujian Sistem

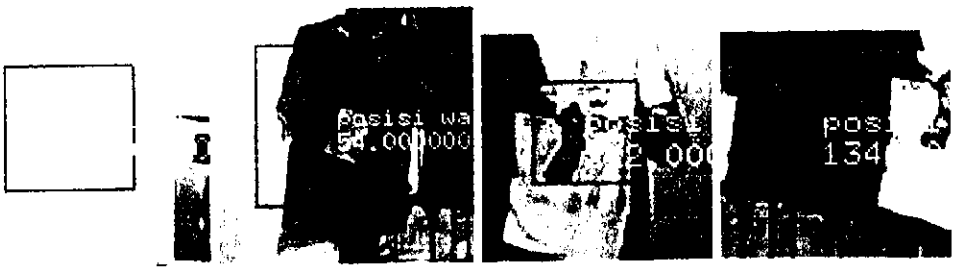
Pengujian sistem disini dimaksudkan untuk mengetahui sejauh mana jalannya aplikasi yang telah dibangun dalam menghadapi kondisi pengujian. Hal ini ditujukan agar dapat diketahui hasil akhir yang diharapkan dari tujuan awal dibangunnya aplikasi ini dan juga kondisi-kondisi dimana aplikasi ini tidak dapat memberikan hasil yang optimal, atau dengan kata lain kelemahan aplikasi ini. Pengujian sistem ini dilakukan beberapa kali, dengan kondisi yang berbeda-beda.

7.1.1 Pengujian Pertama

Pengujian pertama dilakukan dengan tanpa objek wajah pada bidang yang diambil citranya, dan pengujian ini menunjukkan bahwa aplikasi masih salah mendeteksi objek yang bukan wajah sebagai wajah.

Gambar 7.1 – Pengujian pertama

Dapat dilihat pada gambar 7.1, pada pengujian pertama ini, aplikasi melakukan salah deteksi pada objek yang bukan wajah, dan ini terjadi beberapa kali. Hal ini menunjukkan aplikasi ini mempunyai *False Positive Rate* atau Tingkat Kesalahan Positif yang cukup tinggi.

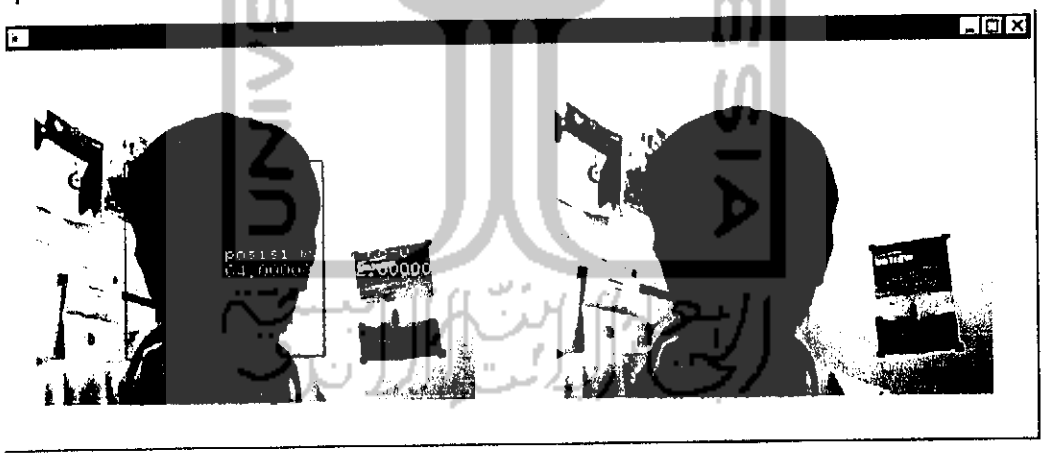


Gambar 7.2 – Screenshot hasil pengujian

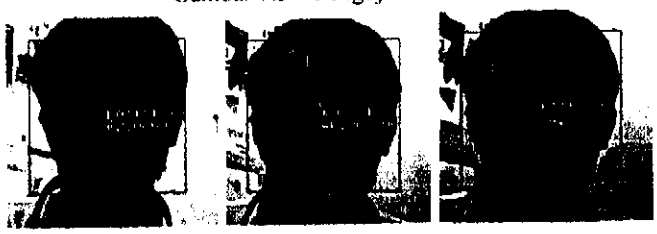
Gambar 7.2 menunjukkan *screenshot* yang didapat hasil dari pengujian pertama, dan dari gambar tersebut menunjukkan aplikasi beberapa kali melakukan salah deteksi pada satu kali pengujian.

7.1.2 Pengujian Kedua

Pengujian kedua ini dilakukan dengan hanya satu objek pada bidang penangkapan citra. Pengujian ini menghasilkan hasil deteksi positif, seperti pada gambar 7.3, dan gambar 7.4 menunjukkan hasil screenshot yang diambil oleh aplikasi.



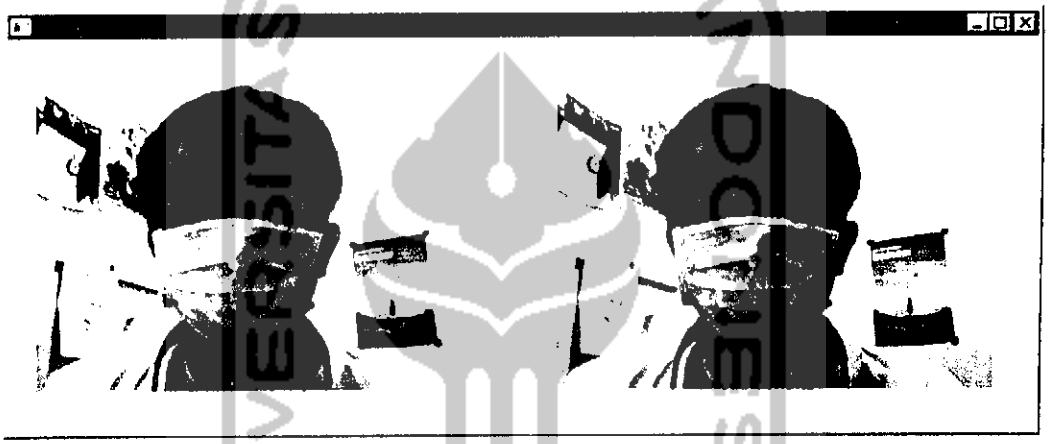
Gambar 7.3 – Pengujian Kedua



Gambar 7.4 – Screenshot hasil pengujian

7.1.3 Pengujian Ketiga

Pengujian ketiga ini dilakukan dengan adanya objek yang sebagian tertutup, dan aplikasi tidak dapat mendeteksinya. Pengujian ini dilakukan dua kali, yang pertama dengan objek tertutup pada bagian mulut, dan kedua tertutup pada bagian mata, seperti pada gambar 7.5 dan 7.6.

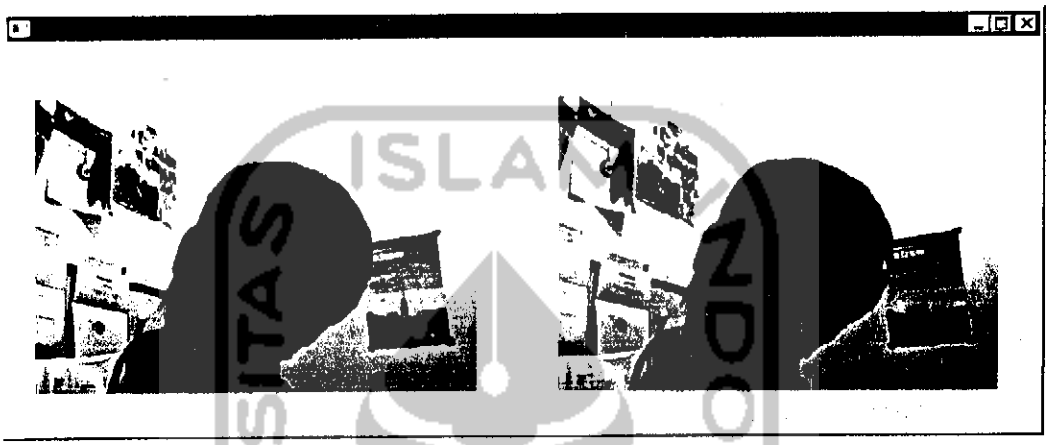


Gambar 7.5 – Pengujian Ketiga, objek tertutup mulutnya

Gambar 7.6 – Pengujian Ketiga, objek tertutup mata

7.1.4 Pengujian Keempat

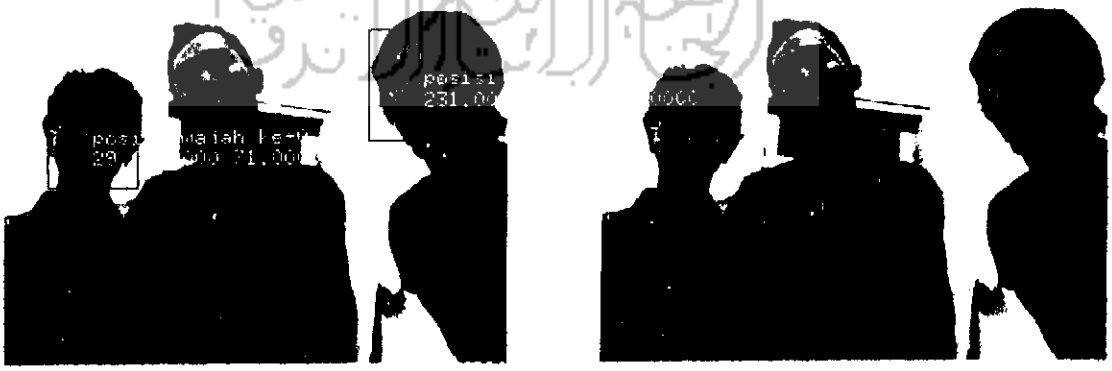
Pengujian keempat menggunakan objek yang sama dan tidak tertutup apa-apa, tetapi kali ini objeknya miring. Pengujian ini menunjukkan bahwa aplikasi tidak dapat mendeteksi objek yang miring, seperti pada gambar 7.7.



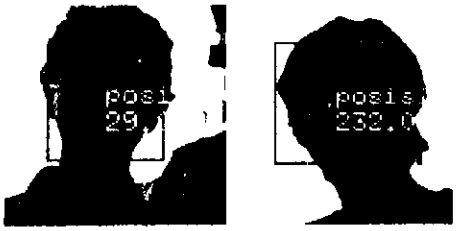
Gambar 7.7 – Pengujian Keempat, objek miring

7.1.5 Pengujian Kelima

Pengujian kelima ini menggunakan objek deteksi lebih dari satu, dan hasilnya menunjukkan aplikasi mampu mendeteksi dua dari tiga objek yang ada. Pada pengujian ini ada objek yang tidak terdeteksi karena objek tersebut miring, seperti ditunjukkan gambar 7.8. Gambar 7.9 menunjukkan *screenshot* yang berhasil diambil oleh aplikasi.



Gambar 7.8 – Pengujian Kelima, objek banyak



Gambar 7.9 – Screenshot hasil deteksi

7.2. Analisa Ukuran File

Seperti disebutkan pada tujuan awal pembangunan sistem ini, yaitu efisiensi dari penggunaan ruang harddisk, sehingga tidak perlu menyimpan keseluruhan rekaman video dan hanya menyimpan data wajah yang terdeteksi. Berikut penulis paparkan analisa ukuran file dari citra objek yang terdeteksi, tertera pada tabel 7.1.

No	Pengujian	File Hasil Deteksi	Ukuran
1	Pengujian Pertama	capture1412010-104821-0.jpg	2.06 kb
		capture1412010-104843-0.jpg	4.41 kb
		capture1412010-104825-0.jpg	2.74 kb
2	Pengujian Kedua	capture1412010-11510-0.jpg	6.99 kb
		capture1412010-11510-1.jpg	8.80 kb
		capture1412010-11511-0.jpg	7.07 kb
3	Pengujian Ketiga	-	-
4	Pengujian Keempat	-	-
5	Pengujian Kelima	capture1312010-23192-0.jpg	3.08 kb
		capture1312010-231856-0.jpg	3.07 kb

7.3. Pembahasan

Berdasarkan pengujian-pengujian tersebut, maka dapat diambil kesimpulan dari hasil pengujian tentang kinerja sistem ini :

1) Pengujian pertama dengan kondisi tidak ada objek pada bidang pengujian, sistem beberapa kali melakukan salah deteksi dengan menandai objek yang

bukan objek wajah. Hal ini menandakan sistem mempunyai Tingkat Kesalahan Positif yang cukup tinggi.

2) Pengujian kedua dengan kondisi normal, adanya satu objek pada bidang pengujian. Sistem telah dapat mendeteksi adanya objek dan melakukan penandaan pada objek.
3) Pengujian ketiga dengan menggunakan objek sebelumnya, tetapi tertutup pada sebagian bidangnya, pada mulut dan wajah. Sistem tidak dapat mendeteksi objek tersebut.
4) Pengujian keempat, dengan satu objek pada bidang pengujian, tetapi objek dimiringkan. Sistem tidak dapat mendeteksi objek tersebut.
5) Pengujian kelima, dengan tiga objek wajah pada bidang pengujian. Sistem hanya dapat mendeteksi dua dari tiga objek yang ada, karena ada satu objek yang miring.

BAB VIII
PENUTUP

8.1. Kesimpulan

Berdasarkan hasil dari penelitian dan pembahasan yang telah dipaparkan pada bab-bab sebelumnya, maka dapat ditarik suatu kesimpulan sebagai berikut :

1. Aplikasi pendeteksi wajah ini dapat memaksimalkan proses pada penggunaan kamera pengawas, dengan hanya menyimpan data yang penting saja sehingga mengurangi spesifikasi *space* untuk menyimpan data yang dibutuhkan.
2. Penggunaan metode Viola-Jones dan fitur Haar untuk implementasi aplikasi pendeteksian wajah adalah metode yang cukup optimal, dilihat dari kecepatan proses dan tingkat akurasi deteksi yang tinggi, namun metode ini memiliki tingkat kesalahan positif yang cukup tinggi pula.
3. Penggunaan framework OpenCV disini telah banyak membantu pada proses analisa citra dan proses-proses lain yang berhubungan dengan citra.
4. Penggunaan framework OpenFrameworks telah mempermudah pembangunan sistem dan mempercepat waktu pembuatan sistem. Framework ini juga membantu interaksi, karena tidak perlu memprogram dari awal komponen untuk interaksi aplikasi dengan *hardware* seperti kamera.

8.2. Saran

Dari kesimpulan diatas, maka dapat dikemukakan beberapa saran sebagai berikut :

1. Untuk kepentingan optimalisasi proses, dapat dicari lagi metode-metode pendeteksian lain yang lebih cepat dan lebih akurat selain metode Viola-Jones.
2. *Interface* aplikasi ke pengguna sebaiknya dapat didesain agar lebih bagus lagi agar lebih mudah digunakan dan dipahami oleh pengguna.

3. Aplikasi ini kedepannya sebaiknya dapat dioptimalkan lagi agar dapat mengenali wajah yang miring dan wajah yang tertutup. Kekurangan-kekurangan yang ada pada aplikasi perlu diperbaiki lagi.
4. Aplikasi yang ada dapat ditambahkan proses pengenalan wajah, agar dapat mengenali wajah milik siapa yang dideteksi oleh aplikasi.
5. Aplikasi ini dapat digunakan pada bidang lain selain bidang keamanan, seperti hiburan, atau dapat dikembangkan lagi menjadi sistem biometrika.

Daftar Pustaka

Bradski, Gary & Kaehler, Adrian. 2008. *Learning OpenCV*. Sebastopol : O'Reilly Media.

Crane, Randy. 1997. *A Simplified Approach to Image Processing*. New Jersey : Prentice-Hall.

Gonzalez, Rafael & Woods, Richard. 2002. *Digital Image Processing*. New Jersey : Prentice-Hall.

Haußecker, Horst. 2000. *Computer Vision and Applications*. London : Academic Press.

Hornegger, Joachim et al. 2000. *Computer Vision and Applications : Probabilistic Modelling in Computer Vision*. London : Academic Press.

Jensen, Helvig. 2008. *Implementing the Viola-Jones Face Detection Algorithm*. Kongens-Lingby: Technical University of Denmark.

Khattab, K., Dubois, J., & Miteran, J. 2009. Cascade Boosting-Based Object Detection from High-Level Description to Hardware Implementation. *EURASIP Journal on Embedded Systems*. Volume 2009.

Noble, Joshua. 2009. *Programming Interactivity*. Sebastopol : O'Reilly Media.

Sebe, N., Cohen, Ira., Garg, Ashutosh., and Huang, Thomas. 2005. *Machine Learning In Computer Vision*. Dordrecht : Springer.

Shapiro, Linda & Stockman, George. 2000. *Computer Vision*. Washington.

Szeliski, Richard. *Computer Vision:Algorithms and Applications*. 2009. Intel Website.

Ritter, G. X. and Wilson, J. N. 1996. *Handbook of Computer Vision Algorithmsin Image Algebra*. Boca Raton: CRC Press.

Viola, Paul & Jones, Michael J. 2004. *Robust Real Time Face Detection*. Netherlands : Kluwer Academic Publishers.