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

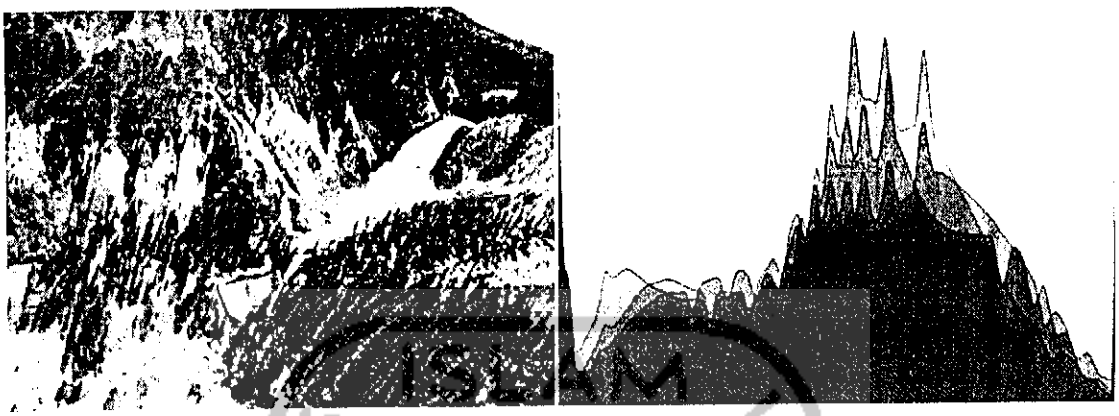
Citra video per-frame yang telah dikonversi menjadi *grayscale* ini diproses oleh fungsi `findHaarObjects()` yang berasal dari objek `haarFinder` yang merupakan instansiasi dari kelas `ofxCvHaarFinder`, dimana kelas ini hanya merupakan penghubung antara aplikasi utama openFrameWorks dengan fungsi yang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

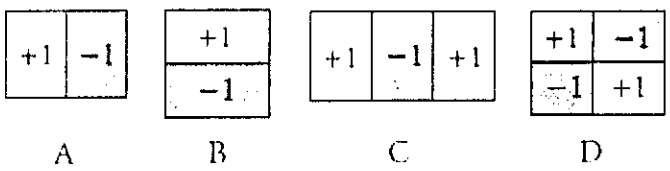
Suatu cascade haar menerjemahkan suatu rangkaian dari operasi deteksi fitur dalam suatu struktur *tree*. Pada setiap *stump-based* Haar *cascade*, decision pointnya

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:

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

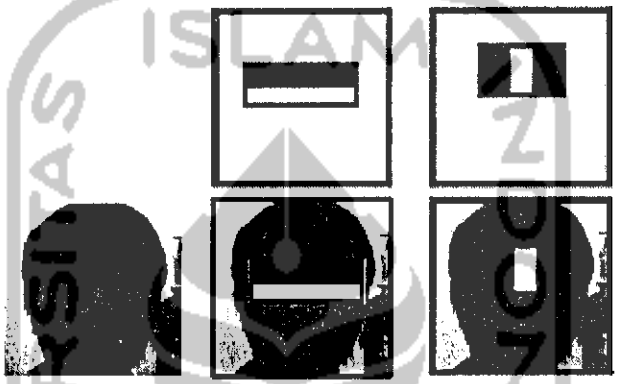
Dari hasil perhitungan di atas, untuk dibandingkan masih dikalikan dengan bobot masing-masing, dan bobot ini telah ditentukan pada spesifikasi *cascade*. Yang dimaksud point di atas adalah point kiri atas, kanan atas, kiri bawah dan kanan bawah dari region citra yang akan dianalisis dan juga fitur Haar 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 );
}
```

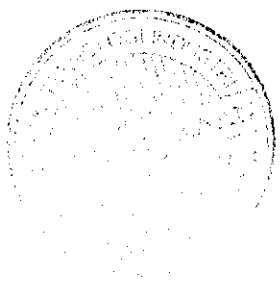
Variabel bertipe `CvRect` di atas adalah variabel primitif bawaan dari OpenCV untuk menggambarkan suatu struktur persegi panjang. Pada kode di atas, `ix` dan `iy` menunjukkan koordinat ditemukannya objek, dan `win_size.width` dan `win_size.height` adalah region pencariannya yang berarti ukuran dari objek. 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.

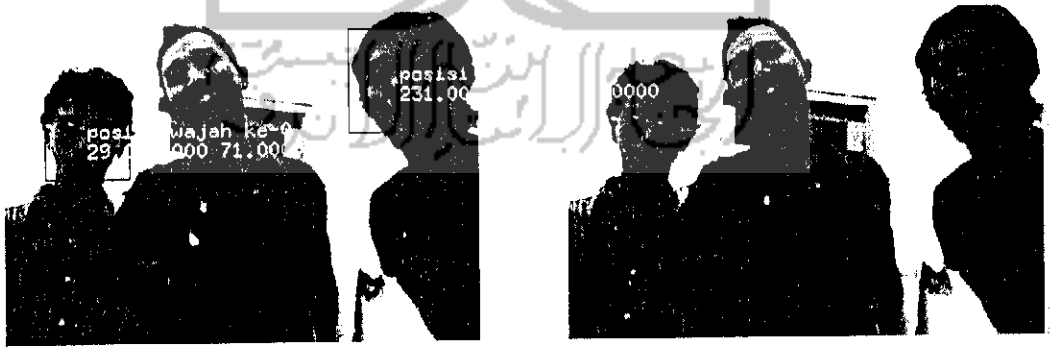
```
void FaceApp::draw()
{
      //....

      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