

5-1-2009

Real Time Driver Safety System

Gyuchoon Cho

Western Kentucky University, gyuchoon.cho350@wku.edu

Follow this and additional works at: <http://digitalcommons.wku.edu/theses>



Part of the [Graphics and Human Computer Interfaces Commons](#)

Recommended Citation

Cho, Gyuchoon, "Real Time Driver Safety System" (2009). *Masters Theses & Specialist Projects*. Paper 63.
<http://digitalcommons.wku.edu/theses/63>

This Thesis is brought to you for free and open access by TopSCHOLAR®. It has been accepted for inclusion in Masters Theses & Specialist Projects by an authorized administrator of TopSCHOLAR®. For more information, please contact connie.foster@wku.edu.

REAL TIME DRIVER SAFETY SYSTEM

A Thesis
Presented to
The Faculty of the Department of Mathematics and Computer Science
Western Kentucky University
Bowling Green, Kentucky

In Partial Fulfillment
Of the Requirements for the Degree
Master of Computer Science

By
Gyuchoon Cho

May 2009

REAL TIME DRIVER SAFETY SYSTEM

Date Recommended April 27 / 2009

Ahmed Emam
Director of Thesis

Art Shindhelm

Mustafa Atici

Dean, Graduate Studies and Research Date

ACKNOWLEDGEMENTS

The first person who I would like to thank is my advisor, Dr. Ahmed Emam. He gave me confidence when I doubted whether I could finish this project and thesis. Because he trusted and advised me, I created a new algorithm to find a driver's drowsy eyes. He has an ability that persuades a student to do the impossible and make it possible. I would like to thank my wife, Jungmee Myung, who has supported me to do this thesis by taking care of our lovely children, Minseo and Mini, without me. She helped me focused on my thesis. I also thank many WKU students who participated in this experiment. I needed to collect many eye images when I changed the algorithm. I remember that nobody complained and everybody happily helped me to do this experiment. I would like to thank my committee members, Dr. Art Shindhlem and Dr. Mustafa Atici. They gave me pleasant memories at WKU. I would like to thank Pengpeng Lin for being my best friend. Special thanks to my parents and siblings in Korea.

TABLE OF CONTENTS

ABSTRACT	V
CHAPTER 1: INTRODUCTION	3
<u>1.1 INTRODUCTION TO DRIVER SAFETY SYSTEM</u>	3
<u>1.2 INTRODUCTION TO WEB-CAM PROGRAMMING</u>	7
1.2.1 FUNCTIONS OF VFW	8
<u>1.3 INTRODUCTION TO FACE DETECTION TECHNOLOGY</u>	22
CHAPTER 2: RESEARCHES ON FACE AND EYE DETECTION	30
<u>2.1 INTRODUCTION</u>	30
<u>2.2 PRVIOUS WORKS</u>	30
2.2.1 BASIC UNDERSTANDING	32
2.2.2 HAUSDORFF DISTANCE	34
2.2.3 DEFORMABLE TEMPLATE MATCHING	36
2.2.4 GEOMETRIC PROPERTIES OF TOPOGRAPHIC MANIFOLD	40
2.2.5 EYE LOCATION METHOD USING ORDINAL FEATURES	42
2.2.6 SUPPORT VECTOR MACHINES	43
<u>2.3 FACE DETECTION SOFTWARE DEVELOPMENT KITS</u>	45

2.3.1 OPENCV	45
2.3.2 VELILOOK SDK	45
2.3.3 LUXAND FACE SDK	51
<u>2.4 SUMMARY</u>	53
 CHAPTER 3: FINDING DROWSY EYES	31
<u>3.1 INTRODUCTION</u>	31
<u>3.2 REAL TIME DRIVER SAFETY SYSTEM</u>	31
3.2.1 SYSTEM ENVIRONMENT	31
3.2.2 SYSTEM ARCHITECTURE	56
3.2.3 FACE DETECTION	57
3.2.4 EXTRACT MODULE	62
3.2.5 PROCESS MODULE	64
3.2.6 SYSTEM INTEGRATION	68
 CHAPTER 4: EXPERIMENTAL RESULTS	55
<u>4.1 RESULTS</u>	55
4.1.1 TIME CONSUMPTION	55
4.1.2 LIGHT INFLUENCE	73

4.1.3 GLASSES INFLUENCE	76
4.1.4 SKIN COLORS	79
4.1.5 ERROR ON THE SYSTEM	82
 CHAPTER 5: CONCLUSION AND FUTURE WORK.....	71
<u>5.1 CONCLUSION</u>	71
<u>5.2 FUTURE WORK</u>	85
5.2.1 LIGHT SENSOR	86
5.2.2 PRE-CALCULATION OF THRESHOLD	87

REAL TIME DRIVER SAFETY SYSTEM

Gyuchoon Cho

May, 2009

93 Pages

Directed by: Dr. Ahmed Emam

Department of Computer Science

Western Kentucky University

The technology for driver safety has been developed in many fields such as airbag system, Anti-lock Braking System or ABS, ultrasonic warning system, and others.

Recently, some of the automobile companies have introduced a new feature of driver safety systems. This new system is to make the car slower if it finds a driver's drowsy eyes. For instance, Toyota Motor Corporation announced that it has given its pre-crash safety system the ability to determine whether a driver's eyes are properly open with an eye monitor. This paper is focusing on finding a driver's drowsy eyes by using face detection technology.

The human face is a dynamic object and has a high degree of variability; that is why face detection is considered a difficult problem in computer vision. Even with the difficulty of this problem, scientists and computer programmers have developed and improved the face detection technologies. This paper also introduces some algorithms to find faces or eyes and compares algorithm's characteristics.

Once we find a face in a sequence of images, the matter is to find drowsy eyes in the driver safety system. This system can slow a car or alert the user not to sleep; that is the purpose of the pre-crash safety system. This paper introduces the VeriLook SDK, which is used for finding a driver's face in the real time driver safety system. With several experiments, this paper also introduces a new way to find drowsy eyes by AOI,

Area of Interest. This algorithm improves the speed of finding drowsy eyes and the consumption of memory use without using any object classification methods or matching eye templates. Moreover, this system has a higher accuracy of classification than others.

Chapter 1: INTRODUCTION

1.1 Introduction to Driver Safety System

The technology for driver safety has been developed in many fields such as airbag system, Anti-lock Braking System or ABS, ultrasonic warning system, and others.

Recently, some of automobile companies have introduced another feature of driver safety system. This new system is to find a driver's drowsy eyes and to make the car slower. For instance, Toyota Motor Corporation announced that it has given its pre-crash safety system the ability to determine whether a driver's eyes are properly open with an eye monitor. This paper is focusing on finding a driver's drowsy eyes by using face detection technology.

1.1.1 Driver Safety Technologies

Since the car was invented, it brings both advantages and disadvantages. Not only do people travel to another city for a short time but also many companies ship their products using an automobile. As the functions of a car are improved, the progress of civilization is elevated and it consequently raises an industrial growth rate as well. However, people are placed in danger by using the car. The more speed the car has, the more risk comes with it. For this reason, scientists have developed driver safety systems. The technology for driver safety has been developed in many fields such as airbag system, ABS, Tire Pressure Monitoring System or TPMS, and others. In this section, several driver safety technologies will be introduced.

1.1.1.1 Anti-lock Braking System (ABS):

An ABS is a safety system which prevents the wheels on a motor vehicle from locking while braking.

A rotating wheel allows the driver to maintain steering control under heavy braking by preventing a skid and allowing the wheel to continue interacting with the road surface as directed by driver steering inputs. While ABS offers improved vehicle control in some circumstances, it can also present disadvantages including increased braking distance on slippery surfaces such as ice, packed snow, gravel, steel plates and bridges, or anything other than dry pavement. ABS has also been demonstrated to create a false sense of security in drivers, who may drive more aggressively as a result.

Since initial widespread use in production cars, anti-lock braking systems have evolved considerably. Recent versions not only prevent wheel lock under braking, but also electronically control the front-to-rear brake bias.

1.1.1.2 Air-Bag System:

An airbag is a vehicle safety device. It is an occupant restraint consisting of a flexible envelope designed to inflate rapidly in an automobile collision to prevent vehicle occupants from striking hard interior objects such as steering wheels.

The design is conceptually simple; a central ACU or Airbag Control Unit monitors a number of related sensors within the vehicle, including accelerometers, impact sensors, side pressure sensors, wheel speed sensors, gyroscopes, brake pressure sensors, and seat occupancy sensors. When the requisite “threshold” has been reached or exceeded, the ACU

will trigger the ignition of a gas generator propellant to rapidly inflate a nylon fabric bag. As the vehicle occupant collides with and squeezes the bag, the gas escapes in a controlled manner through small vent holes. The airbag's volume and the size of the vents in the bag are tailored to each vehicle type to spread out the deceleration of the occupant over time and over the occupant's body compared to a seat belt alone.

1.1.1.3 Tire Pressure Monitoring System (TPMS):

A Tire Pressure Monitoring System (TPMS) is generally an electronic system designed to monitor the air pressure inside all the pneumatic tires on automobiles, airplane undercarriages, straddle lift carriers, forklifts, and other vehicles. The system is also sometimes referred to as a Tire Pressure Indication System or TPIS. These systems report real time tire pressure information to the driver of the vehicle -- either via a gauge, a pictograms display, or a simple low pressure warning light.

TPMS helps to improve vehicle safety and aids drivers in maintaining their vehicle's tire pressures. Properly maintained tires help with vehicle safety, performance, and economy. In the US, the National Highway Traffic Safety Administration or NHTSA has estimated that every year 533 fatalities are caused by tire defects in road accidents. Adding TPMS to all vehicles could avoid 120 of the 533 yearly victims and spare as many as 8,400 injuries every year.

The French Road Safety organization estimates that 9% of all road accidents involving fatalities are attributable to tire under inflation, and the DEKRA, Germany's vehicle certification body, estimated that 41% of accidents with physical injuries are linked to tire problems.

On the maintenance side, it is important to realize that fuel efficiency and tire wear are severely affected by under inflation. In the US, NHTSA data relates that tires leak air naturally and over a year a typical new tire can lose between 20 and 60 kPa.

If we also consider that over 40% of vehicle owners in Europe and North America check their tires less than once a year, it is conceivable that 40% or more of vehicles currently in use in those areas are running with underinflated tires.

The European Union reports that an average under inflation of 40 kPa produces an increase of fuel consumption of 2% and a decrease of tire life of 25%. The EU concludes that tire under inflation today is responsible for over 20 million liters of unnecessary burned fuel, dumping over 2 million tons of CO₂ in the atmosphere, and 200 million tires prematurely wasted in the world.

For these safety and environmental reasons, the US Federal Government has mandated the use of TPMS, and other countries should follow closely. The TPMS mandated by the US law must warn the driver when a tire is under inflated by as much as 25% according to US DOT NHTSA Docket No 2005-20586. However, since the recommended tire pressures for most vehicles are more than 160 kPa, a deflation of 40 kPa would be within the 25% allowance and would not trigger the TPMS warning mandated by the US law. Therefore, the mandated TPMS is mainly designed for safety and is unlikely to deliver the above benefits. Drivers are still advised to manually check their tire pressure often to maintain optimal performance.

1.2 Introduction to Web-Cam Programming

This section describes how to get a sequence of images before the explanation of face detection technology. In 1992, Microsoft introduced a multimedia framework called Video for Windows, VFW. According to MSDN, Micro Soft Developer Network, VFW provides functions that enable an application to process video data. This framework covers some multimedia techniques such as AVIFile Functions and Macros, Video Compression Manager, Custom File and Stream Handlers, DrawDib, and Video Capture. A programmer can implement video capture into an application very easily by using the AVICap window class. AVICap is the class in form of DLL, Dynamic Link Library, which is a module that contains functions for the Windows APIs used to capture video from web-cam or other devices. When Windows operating system is installed, AVICap32.dll file is located in c:\Windows\System32 folder. The main functions of AVICap class are following:

- Capture audio and video streams to an audio-video interleaved file.
- Connect and disconnect video and audio input devices dynamically.
- View a live incoming video signal by using the overlay or preview methods.
- Specify a file to use when capturing and copy the contents of the capture file to another file.
- Set the capture rate.
- Display dialog boxes that control the video source and format.
- Create, save, and load palettes.
- Copy images and palettes to the clipboard.
- Capture and save a single image as a device-independent bitmap (DIB).

1.2.1 Functions of VFW

In this section, several functions from VFW API will be demonstrated and segments of code will be shown in the C++ programming language.

To create capture windows, `capCreateCaptureWindow` API is used as in the following example showing below in segment of code.

```
// example to create capture window
// Returns a handle of the capture window if successful or NULL otherwise.
hWndC = capCreateCaptureWindow (
    (LPSTR) "My Capture Window",    // window name if pop-up
    WS_CHILD | WS_VISIBLE,          // window style
    0, 0, 160, 120,                 // window position and dimensions
    (HWND) hwndParent,              // Handle to the parent window
    (int) nID /* child ID */);      // Window identifier
```

After the application creates a capture window of the `AVICap` window class and connects it to a video driver, the capture window is ready to grasp data. At this point, the application can send the `WM_CAP_SEQUENCE` message or the `capCaptureSequence` macro to begin capturing.

Using default settings, `WM_CAP_SEQUENCE` initiates the capture of video and audio input to a file named `CAPTURE.AVI`. Capture continues until one of the following events occurs:

- The user presses the ESC key or a mouse button.
- An application stops or aborts the capture operation.

- The disk becomes full.

Once getting a windows handle by using `capCreateCaptureWindow`, it needs to connect to the capture device. When an application does not need to capture the video any more, it needs to disconnect the device from the other application.

```
// Connect to the MSVIDEO driver
// Returns TRUE if successful or FALSE if the specified capture driver cannot be connected to
// the capture window.
fOK = capDriverConnect(hWndC, 0);
    // hWndC : handle to a capture window
    // Index of the capture driver. The index can range from 0 through 9.

// Disconnect from the MSVIDEO driver
capDriverDisconnect(hWndC);
```

The next step is to find a list of capturing devices so that a user can select one of the capturing devices. To get the list of devices, `capGetDriverDescription` can be used.

```
char szDeviceName[80];           // to store device name
char szDeviceVersion[80];       // to store device's version

for (wIndex = 0; wIndex < 10; wIndex++)
{
    // Returns TRUE if successful or FALSE otherwise.
    if (capGetDriverDescription(
        wIndex,
        // Index of the capture driver. The index can range from 0 through 9.
        // Plug-and-Play capture driver are enumerated first, followed by
        // capture drivers listed in the registry, which are then followed by
        // capture drivers listed in SYSTEM.INI.
        szDeviceName,
        // Pointer to a buffer containing a null-terminated string corresponding
        // to the capture driver name.
```



```

        sizeof(szDeviceName),
            // Length, in bytes, of the buffer pointed to by device name
        szDeviceVersion,
            // Pointer to a buffer containing a null-terminated string
        sizeof(szDeviceVersion)
            // Length, in bytes, of the buffer pointed to by device version.
    ))
{
    // Append name to list of installed capture drivers
    // and then let the user select a driver to use.
}
}

```

capDriverGetCaps is to retrieve the driver capabilities.

```

// Returns TRUE if successful or FALSE if the capture window is not connected to a capture
// driver.
capDriverGetCaps(hWndDC,    // Handle to a capture window.
    &CapDrvCaps,           // Pointer to the CAPDRIVERCAPS structure to contain
                            // the hardware capabilities.
    sizeof(CAPDRIVERCAPS)); // Size, in bytes, of the structure referenced by
                             // psCaps.

// The capabilities returned in CAPDRIVERCAPS are constant for a given capture driver.
// Applications need to retrieve this information once when the capture driver is first connected
// to a capture window.

```

The following example uses the SetWindowPos function to set the size of the capture window to the overall dimensions of the incoming video stream based on information returned by the capGetStatus macro in the CAPSTATUS structure.

```

CAPSTATUS CapStatus;    // Structure defines the current state of the capture window.

// Returns TRUE if successful or FALSE if the capture window is not connected to a capture

```

```

device.
capGetStatus(hWndC, // Handle to a capture window.
    &CapStatus, // Pointer to the CAPDRIVERCAPS structure to obtain the hardware
                // capabilities.
    sizeof(CAPSTATUS)); // Size, in bytes, of the structure referenced by psCaps.

SetWindowPos(hWndC, NULL, 0, 0, CapStatus.uiImageWidth,
    CapStatus.uiImageHeight, SWP_NOZORDER | SWP_NOMOVE);

```

Each capture driver can provide up to four dialog boxes to control aspects of the video digitization and capture process and to define compression attributes used in reducing the size of the video data. The contents of these dialog boxes are defined by the video capture driver.

The Video Source dialog box controls the selection of video input channels and parameters affecting the video image being digitized in the frame buffer. This dialog box enumerates the types of signals that connect the video source to the capture card, typically SVHS and composite inputs, and provides controls to change hue, contrast, or saturation. If the dialog box is supported by a video capture driver, a programmer can display and update it by using the WM_CAP_DLG_VIDEOSOURCE message or the capDlgVideoSource macro.

The Video Format dialog box controls the selection of the digitized video frame dimensions and image-depth and compression options of the captured video. If the dialog box is supported by a video capture driver, a programmer can display and update it by using the WM_CAP_DLG_VIDEOFORMAT message or the capDlgVideoFormat macro.

The Video Display dialog box controls the appearance of the video on the monitor during capture. The controls in this dialog box have no effect on the digitized video data, but they might affect the presentation of the digitized signal. For example, capture devices that support overlay might allow altering hue and saturation, key color, or alignment of the overlay. If the dialog box is supported by a video capture driver, the programmer can display and update it by using the WM_CAP_DLG_VIDEODISPLAY message or the capDlgVideoDisplay macro. The following example shows how to display these dialog boxes and their screen shots.

```
// Defines the structure of the capabilities of the capture driver.
CAPDRIVERCAPS CapDrvCaps;

capDriverGetCaps(hWndC,      // Handle to a capture window.
                &CapDrvCaps, // Pointer to the CapdriverCap structure
                // Size, in bytes, of the structure referenced by the pointer.
                sizeof (CAPDRIVERCAPS));

// Video source dialog box.
if (CapDriverCaps.fHasDlgVideoSource)
    capDlgVideoSource(hWndC);

// Video format dialog box.
if (CapDriverCaps.fHasDlgVideoFormat)
{
    capDlgVideoFormat(hWndC);
    // Are there new image dimensions?
    capGetStatus(hWndC, &CapStatus, sizeof (CAPSTATUS));
    // If so, notify the parent of a size change.
}

// Video display dialog box.
if (CapDriverCaps.fHasDlgVideoDisplay)
    capDlgVideoDisplay(hWndC);
```

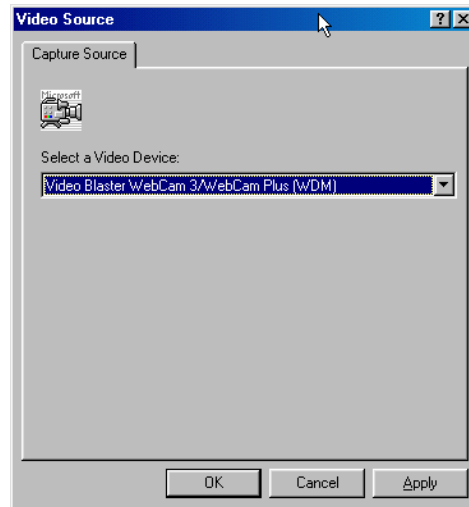


Figure 1: Video source dialog box.

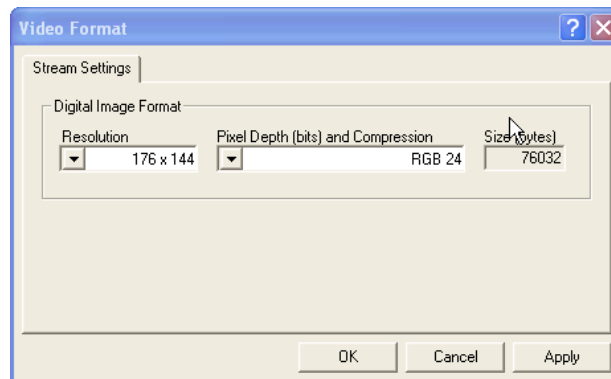


Figure 2: Video format dialog box.

The following example uses the `capFileSetCaptureFile` macro to specify an alternate filename, `MYCAP.AVI`, for the capture file and the `capFileAlloc` macro to preallocate a file of 5 MB.

```
char szCaptureFile[] = "MYCAP.AVI";           // Setting the file name.

// Return TRUE if successful or FALSE if the filename is invalid or if streaming or
```

```
// single-frame capture is in progress.
capFileSetCaptureFile( hWndC, // Handle to a capture window.
    szCaptureFile);          // Pointer to the null-terminated string that contains the
                                // name of the capture file to uses.
capFileAlloc( hWndC,         // Handle to a capture window.
    (1024L * 1024L * 5));     // Size, in bytes, to create the capture file.
```

The programmer can retrieve the current capture format for audio data or the size of the audio format structure by sending the WM_CAP_GET_AUDIOFORMAT message or the capGetAudioFormat and capGetAudioFormatSize macros to a capture window. The default audio capture format is mono, 8-bit, 11 kHz PCM, Pulse Code Modulation. When a programmer retrieves the format by using WM_CAP_GET_AUDIOFORMAT always use the WAVEFORMATEX structure.

The programmer also can set the capture format for audio data by sending the WM_CAP_SET_AUDIOFORMAT message or the capSetAudioFormat macro to a capture window. When setting the audio format, a programmer can pass a pointer to a WAVEFORMAT, WAVEFORMATEX, or PCMWAVEFORMAT structure, depending on the specified audio format. The following example uses capSetAudioFormat to set the audio format to 11-kHz PCM 8-bit, stereo.

```
WAVEFORMATEX wfex;          // Defines structure of the format of waveform-audio data.

wfex.wFormatTag = WAVE_FORMAT_PCM;    // Waveform-audio format type
wfex.nChannels = 2;                  // 2: Use stereo, 1: Use monaural
wfex.nSamplesPerSec = 11025; // Sample rate, in sample per second (hertz)
                                // Common values: 8.0, 11.025, 22.05 and 44.1 kHz
wfex.nAvgBytesPerSec = 22050; // Required average data-transfer rate, in bytes per second.
wfex.nBlockAlign = 2;              // Block alignment, in bytes.
wfex.wBitsPerSample = 8;           // Bits per sample for the wFormatTag format type.
```

```

wfex.cbSize = 0; // Size, in bytes, of extra information appended to
the
// end of the WAVEFORMATEX structure.
// Returns TRUE if successful or FALSE otherwise.
capSetAudioFormat(hWndC, // Handle to a capture window
    &wfex, // Pointer to a WAVEFORMATEX structure that defines
// audio format.
    sizeof(WAVEFORMATEX)); // Size, in bytes, of the structure referenced by wfex.

```

The CAPTUREPARMS structure contains the control parameters for streaming video capture. This structure controls several aspects of the capture process and allows the programmer to perform the following tasks:

- Specify the frame rate.
- Specify the number of allocated video buffers.
- Disable and enable audio capture.
- Specify the time interval for the capture.
- Specify whether an MCI device (VCR or videodisc) is used during capture.
- Specify keyboard or mouse control for ending streaming.
- Specify the type of video averaging applied during capture.

This can be retrieved by the current capture settings within the CAPTUREPARMS structure by sending the WM_CAP_GET_SEQUENCE_SETUP message or the capCaptureGetSetup macro to a capture window. It can set one or more current capture settings by updating the appropriate members of the CAPTUREPARMS structure and then sending the WM_CAP_SET_SEQUENCE_SETUP message or the capCaptureSetSetup macro and CAPTUREPARMS to a capture window.

The following example uses the `capCaptureGetSetup` and `capCaptureSetSetup` macros to change the capture rate from the default value, 15 frames per second to 10 frames per second.

```
CAPTUREPARMS CaptureParms;           // Structure contains parameter that
controls the streaming

// video capture process. This structure is used to get and
// set parameters that affect the capture rate, the number of
// buffers to use while capturing, and how
// capture is terminated.

float FramesPerSec = 10.0;

capCaptureGetSetup(hWndC,             // Handle to a capture window.
    &CaptureParms,                    // Pointer to a CAPTUREPARMS structure.
    sizeof(CAPTUREPARMS));           // Size, in bytes, of the structure referenced by
                                     // CAPTUREPARMS structure

// Requested frame rate in microseconds. The default value is 66667, which corresponds to
// 15 frames per second.
CaptureParms.dwRequestMicroSecPerFrame = (DWORD) (1.0e6 /
    FramesPerSec);

capCaptureSetSetup(hWndC,             // Handle to a capture window
    &CaptureParms,                    // Pointer to a CAPTUREPARMS structure.
    sizeof(CAPTUREPARMS));           // Size, in bytes, of the structure referenced by
                                     // CAPTUREPARMS structure
```

Now, the important part in this section is how to save a file. `AVICap`, by default, routes video and audio stream data from a capture window to a file named `CAPTURE.AVI` in the root directory of the current drive. It can be specified by an alternate filename by sending the `WM_CAP_FILE_SET_CAPTURE_FILE` message or the `capFileSetCaptureFile` macro to a capture window. This message specifies the filename; it

does not create, allocate, or open the file. This can be retrieved by the current capture filename by sending the WM_CAP_FILE_GET_CAPTURE_FILE message or the capFileGetCaptureFile macro to a capture window. The following example uses capCaptureSequence macro to start video capture and the capFileSaveAs macro to copy the captured data from the capture file to the file NEWFILE.AVI.

```
char szNewName[] = "NEWFILE.AVI"; // Setting the file name.

// Set up the capture operation.
capCaptureSequence(hWndC);

// Capture.
capFileSaveAs(      hWndC,      // Handle to a capture window.
               szNewName); // Pointer to the null-terminated string that contains
                           // the name of the destination file used to copy the file.
```

For the face detection purpose, every image in sequence needs to process as a single frame. If a programmer wants to capture a single frame as a still image, he or she can use the WM_CAP_GRAB_FRAME_NOSTOP or WM_CAP_GRAB_FRAME message or the capGrabFrameNoStop or capGrabFrame macro to capture the digitized image in an internal frame buffer. The programmer can freeze the display on the captured image by using WM_CAP_GRAB_FRAME. Otherwise, use WM_CAP_GRAB_FRAME_NOSTOP.

Once captured, the programmer can copy the image for use by other applications. The programmer can copy an image from the frame buffer to the clipboard by using the WM_CAP_EDIT_COPY message or the capEditCopy macro. The programmer can also

copy the image from the frame buffer to a device-independent bitmap, DIB, by using the WM_CAP_FILE_SAVEDIB message or the capFileSaveDIB macro.

The application can also use the two single-frame capture messages to edit a sequence frame by frame or to create a time-lapse photography sequence.

A capture window can send messages to the status callback function while capturing video to disk or during other lengthy operations to notify the application of the progress of an operation. The status information includes a message identifier and a formatted text string ready for display. The application can use the message identifier to filter the notifications and to limit the messages to present to the user. During capture operations, the first message sent to the callback function is always IDS_CAP_BEGIN and the last is always IDS_CAP_END. A message identifier of zero indicates a new operation is starting and the callback function should clear the current status.

The following example is a simple status callback function.

```
TCHAR gachAppName[] = TEXT("Application Name"); // Application name.
TCHAR gachBuffer[100]; // Global buffer.

// StatusCallbackProc: status callback function.
// hWnd: capture window handle.
// nID: status code for the current status.
// lpStatusText: status text string for the current status.

LRESULT PASCAL StatusCallbackProc(HWND hWnd, int nID,
    LPTSTR lpStatusText)
{
    if (!hWnd)
        return FALSE;
    if (nID == 0) {
```

```

        // Clear old status messages.
        SetWindowText(hWnd, gachAppName);
        return (LRESULT) TRUE;
    }

    // Show the status ID and status text.
    _sprintf_s(gachBuffer, TEXT("Status# %d: %s"), nID, lpStatusText);

    SetWindowText(hWnd, gachBuffer);
    return (LRESULT) TRUE;
}

```

A capture window uses error notification messages to notify your application of AVICap errors, such as running out of disk space, attempting to write to a read-only file, failing to access hardware, or dropping too many frames. The content of an error notification includes a message identifier and a formatted text string ready for display. The application can use the message identifier to filter the notifications and to limit the messages to present to the user. A message identifier of zero indicates a new operation is starting and the callback function should clear any displayed error messages. The following example is a simple error callback function.

```

TCHAR gachBuffer[100]; // Global buffer.

// ErrorCallbackProc: error callback function.
// hWnd: capture window handle.
// nErrID: error code for the encountered error.
// lpErrorText: error text string for the encountered error.
LRESULT PASCAL ErrorCallbackProc(HWND hWnd, int nErrID,
    LPTSTR lpErrorText)
{

    if (!hWnd)
        return FALSE;
}

```

```

if (nErrID == 0)    // Starting a new major function.
    return TRUE;    // Clear out old errors.
// Show the error identifier and text.
_sprintf_s(gachBuffer, TEXT("Error# %d"), nErrID);

MessageBox(hWnd, lpErrorText, gachBuffer,
    MB_OK | MB_ICONEXCLAMATION);
return (LRESULT) TRUE;
}

```

A capture window uses frame callback notification messages to notify the application when a new video frame is available. The capture window enables these callback notifications only if the preview rate is nonzero and the streaming capture is not in progress. The following example is a simple frame callback function.

```

TCHAR gachBuffer[100]; // Global buffer.

DWORD gdwFrameNum = 0;

// FrameCallbackProc: frame callback function.
// hWnd:                capture window handle.
// lpVHdr:              pointer to structure containing captured
//                      frame information.
//
LRESULT PASCAL FrameCallbackProc(HWND hWnd, LPVIDEOHDR lpVHdr)
{
    if (!hWnd)
        return FALSE;
    _sprintf_s(gachBuffer, TEXT("Preview frame# %ld "), gdwFrameNum++);
    SetWindowText(hWnd, gachBuffer);
    return (LRESULT) TRUE ;
}

```

As was already mentioned above, recording a video file will be stopped by pressing the ESC key, aborting capture operation, or when the disk becomes full. However, if an application supports the estimated size of uncompressed video file, the user may select recording time. Simply, it is easy once a programmer sets up the frame size and frame per second. If the resolution of video is 640 X 480, a frame takes 0.5 Megabyte and fps, frame per second, which is 29. The size of one second video needs $0.5 * 29 = 14.5$ MB. If it records 1 hour, it requires $60 \text{ minutes} * 60 \text{ seconds} * 14.5 \text{ MB} = 52200 \text{ MB}$ that is almost 51 GB. Here is the formula to calculate the video size in bytes.

$$[\text{Height} * \text{Width} * \text{fps} * \text{bit rate} * \text{sec}] / 8 = \text{video size in byte}$$

If a user adds audio data to captured video the total size of video file will be increased. In case of VFW, it usually displays the audio data rate in kilobytes per second. For example, uncompressed stereo 16-bit 44.1 KHz audio has a data rate of 172 kilobytes/second. The formula for audio data in byte is presented in the table below.

8bit mono	$(\text{sec} * \text{KHz}) / 1024$
8bit stereo	$((\text{sec} * \text{KHz}) / 1024) * 2 = 8\text{bit mono} * 2$
16bit mono	same as 8bit stereo
16bit stereo	$((\text{sec} * \text{KHz}) / 1024) * 2 * 2 = 8\text{bit stereo} * 2$

Table 1: Size of audio data with different format.

Total file size with audio data can be expressed below.

$$\text{video size} + \text{audio size} = \text{total file size}$$

If the video is compressed, the total file size can be changed like below.

$$(\text{video size} / \text{compression ratio}) + \text{audio size} = \text{total file size}$$

Here are some compression ratio tables.

	Predict left	Predict gradient	Predict median
320 x 240	2.18:1	2.27:1	2.34:1
640 x 480	2.66:1	2.66:1	2.76:1

Table 2: Huffyuv compression ratio for YUY2 video.

	size	Quality 16	Quality 17	Quality 18	Quality 19	Quality 20
RGB24	320 x 240	19.96:1	16.37:1	12.66:1	8.14:1	3.03:1
	640 x 480	22.85:1	18.80:1	14.61:1	9.51:1	3.47:1
YUY2	320 x 240	14.47:1	11.78:1	9.12:1	5.80:1	2.11:1
	640 x 480	16.57:1	13.74:1	10.65:1	6.79:1	2.44:1

Table 3: PICVideo MJPEG compression for each video type.

1.3 Introduction to Face Detection Technology

1.3.1 Introduction

The definition of the face detection is a computer technology that determines the locations and size of human faces in arbitrary images. This technology was not introduced for a long time. Minority Report, a movie that was released in 2002, is one example. In this movie, a pre-crime system in the year 2054 catches future murders. A suspect must go below the radar of the state-of-the art automated city, where every step he or she takes is

monitored. Everybody has to run because people cannot hide. This movie shows that in the future people's identities are recognizable by a system. But, this is not fiction because this system is becoming true.

CCTV, Closed Circuit Television, has been improved not only in field of quality of image but also by adding many features such as motion detection or face detection. This kind of CCTV is now called smart CCTV or digital surveillance system. This advanced system can capture high definition images and store them digitally. By this technology, it saves time to find data quicker and easier and needs less storage space because it can only record when someone is detected. Like the movie, when a suspect is detected, it records video streaming on the hard drive, and it sends alerts to the user via e-mail, SMS, voice message, etc. The police in several countries including Japan and Korea have already adapted this system to find suspects. On January 24, 2009, one homicidal maniac was caught by police in Korea. This suspect killed at least 7 women. The main key to catching this suspect was CCTV. According to the news, police registered a picture of the suspect into the system and received alerts from a digital surveillance system later. Thanks to this technology, it may save others' lives.

Recently, the rate of CCTV installing has increased. After the 2001 terrorist attack in New York and the London bombings in 2005, CCTV has become a part of our society. People are caught on video in every place such as airports, buildings, ATMs, parking lots, and even vending machines. It means that the world has been threatened by the terrorist attacks. For this reason many commercial companies and researchers have concentrated on this area. One company, Ex-sight, has created a system called FRS Suspect Detection that is an advanced surveillance system. It can automatically detect faces on-line or off-line using

fast acquisition methods. As face images are detected, FRS Suspect Detection automatically alerts for any identified suspects using a variety of interfaces such as SMS, Email, and Web.

This system working scenario is similar to the ones below.

- The cameras record and watch online for points of interests.
- The operator selects the image or video and run it through the system in order to enroll or to match a suspect.
- The system detects if the targets are suspects according to existing watch list.
- The watch list is located on a shared network database and can be shared between several operators or stations.
- Every operator can work on a different area or camera.

Key facial recognition and detection features are similar to below.

- 4 Channels of Face Detection
- 2 Channels of Face Recognition
- 4 Simultaneous Detection/Recognition per channel, maximum of 16
- Two separate relays for Facial Detection & Recognition related applications (e.g. Access Control)
- Face Detection Search & Playback
- Face Recognition Search & Playback
- Face Event Recording & Playback
- Maximum of 1 Billion faces in the Database
- Easy integration with third party devices such as Access Control, Biometrics.

The following image is an example of a suspect detection program

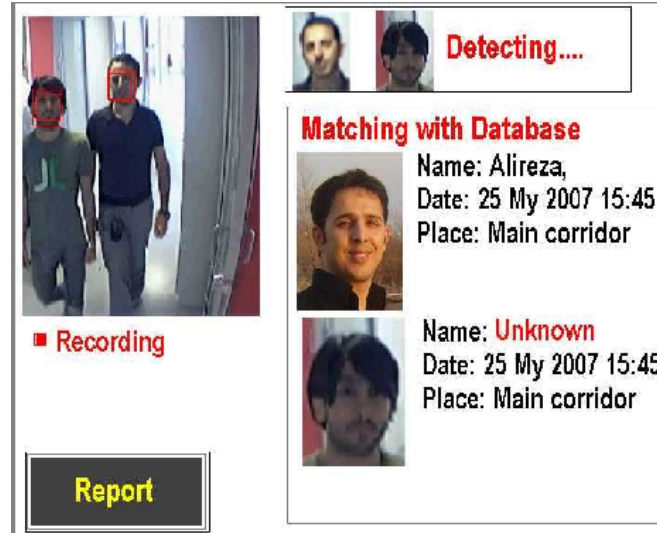


Figure 3: Surveillance camera with suspect detection.

But, this face detection is not only for professional users any more. Recently, many products use face detection technology and it spreads out into our lives. Many digital cameras and camcorders are now installed with this technology. In 2004, Fujifilm first announced that this company released face-finding cameras. Nowadays, most camera manufacturers sell face detection installed cameras in the market. In the case of Fuji films, this company still put new features into cameras including:

- Auto focusing

Face detection optimizes people pictures by identifying faces and adjusting focus and exposure to ensure bright, clear results [3].

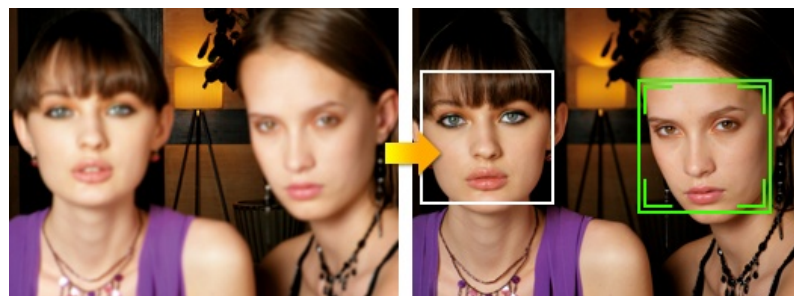


Figure 4: An example image from Fuji.

- Red Eye Correction

Automatically removes red eye effects often spoiled by flash reflections [3].

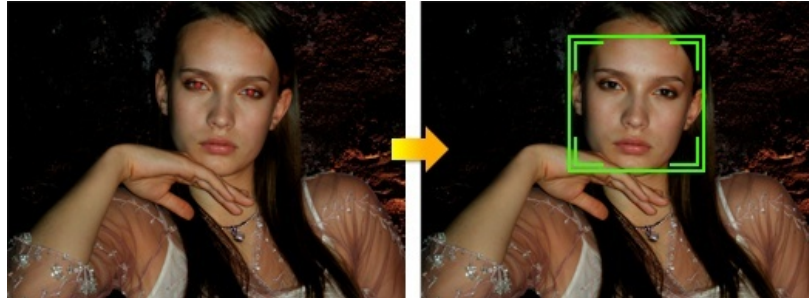


Figure 5: An example image from Fuji.

- Instantly detect up to 10 faces

The advanced Face Detection can simultaneously recognize up to 10 faces in a photo.

Each priceless expression can be cropped and saved in clear high resolution results [3].

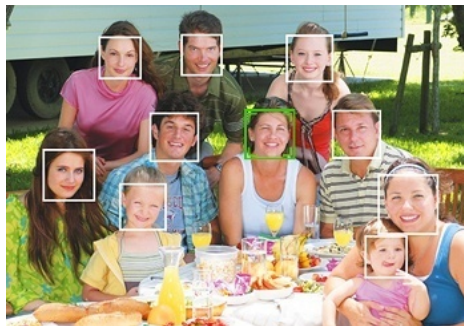


Figure 6: An example image from Fuji.

- Couple Timer

Couple Timer waits until two faces are close together in the frame [3].

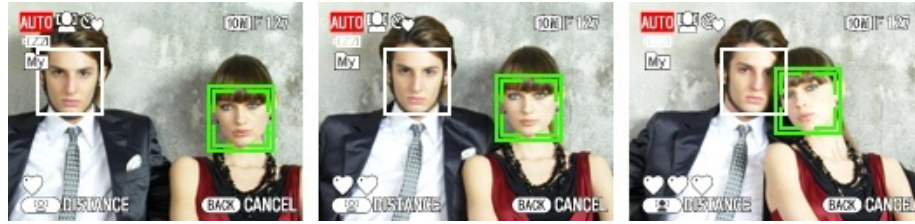


Figure 7: An example image from Fuji.

Although Sony included face detection in the camera or camcorder later than others, Sony's technology about face detection is remarkable. Like other companies it contains face detection technology functions such as automatically optimizing flash, focus, exposure, and white balance. Sony adds one more function that captures a person's emotion with Smile Shutter.

Sony's Smile Shutter mode takes Face Detection technology one step further. Smile Shutter technology helps a user capture those natural smiles, spontaneous reactions, and laughs that are often missed because they only last a few seconds. Simply by selecting the subject and the expression and desired smile sensitivity, Smile Shutter technology does the rest. It's like letting the camera say "cheese" for user. In the case of camcorders, during recording movies and when a person is smiling, they save smile faces into the memory card as a still image [4].

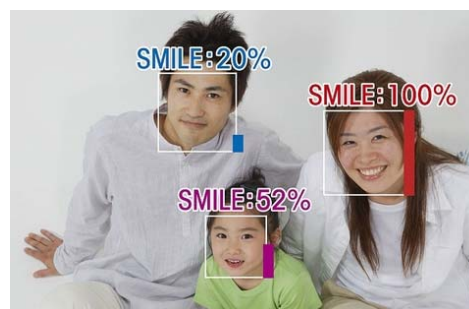


Figure 8: An example image from Sony.

The power of Face Detection and Smile Shutter technologies are especially evident when the user uses their Adult/Child Priority control to fine-tune whether the camera concentrates on grown-ups, children, or both [4].



Figure 9: An example image from Sony.

Another example of using face detection technology is Toyota's pre-crash safety system. Toyota developed an intelligent pre-crash safety system for the automobile industry. Most of the traffic accidents were contributed by the driver's driving attitude. Toyota has put much effort in developing a new vehicle safety system. A sensitive eye motion camera installed in front of the driver seat, in order to detect the upper and lower eyelids, calculate how open the eyes are. This system is able to detect the eyelids movement accurately [5].

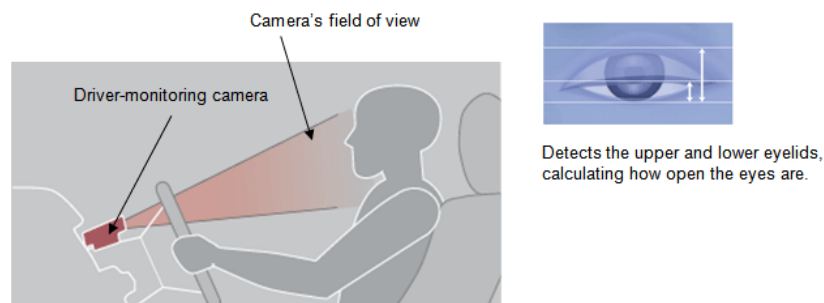


Figure 10: An example image from Toyota.

This pre-crash safety system consist of a camera, millimetre-wave radar, pre-crash brake assist, suspension control and pre-crash seatbelt. The camera and the millimetre wave radar detect the position, distance and speed of any obstacle in front of the car even in bad weather with poor visibility, working in combination with the Pre-Crash Safety computer which monitors vehicle speed, steering angle, and Yaw rate (a vehicle's angular velocity around its vertical axis) inputs to determine if a crash is imminent. If it is, the system gives an audible and visible warning to the driver, retracts the front seatbelts, activates the pre-crash brake assist for maximum braking force and applies the brakes to reduce vehicle speed. Toyota's Advanced Pre-Crash Safety system is able to provide early warning of an impending collision by detecting when a driver is not looking straight ahead. This new feature – a response to data that suggests that most vehicle accidents are caused by lack of driver awareness - uses a camera mounted on the steering column and an image-processing computer to detect the orientation of the driver's face. If the system reads that the driver is not facing forward when it determines that a collision is likely to occur, it will issue a warning to help the driver avoid the accident or lessen collision injuries. This system also offers enhanced Pedestrian Detection capabilities to protect vulnerable road users [5].

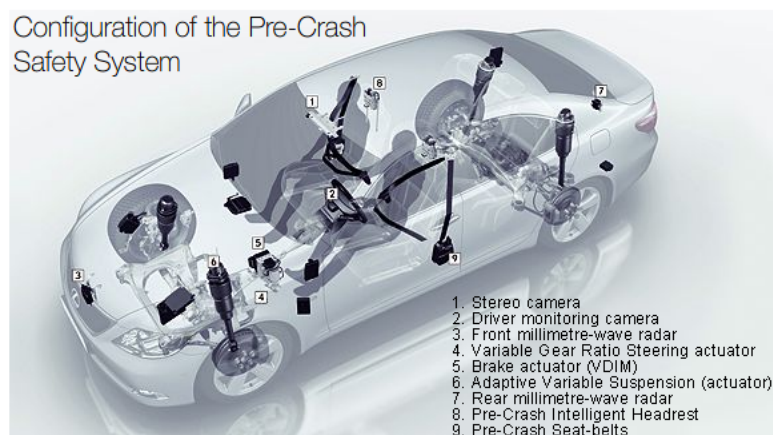


Figure 11: An example image from Toyota.

Chapter 2: Researches on Face and Eye Detection

2.1 Introduction

The human face is a dynamic object and has a high degree of variability that is why face detection is considered a difficult problem in computer vision. Even with the difficulty of this problem, scientists and computer programmers have developed and improved various face detection technologies. In this section, this paper introduces several algorithms and some researches in the field of face and eye detection.

2.2 Previous Works

Face detection is a necessary first step in all of the face processing systems and its performance can severely influence the overall performance of recognition. Three main approaches are proposed for face detection: feature based, image based, and template matching.

Feature based approaches attempt to utilize some prior knowledge of human face characteristics and detect those representative features such as edges, texture, color, or motion. Edge features have been applied in face detection from the beginning (Colmenarez & Huang 1996), and several variations have been developed (Fan, Yau, Elmagarmid & Aref 2001; Froba & Kublbeck 2002; Suzuki & Shibata 2004). Edge detection is a necessary first step for edge representation. Two edge operators that are commonly used are the Sobel Operator and Marr-Hildreth Operator. Edge features can be easily detected in a very short

time, but are not robust for face detection in complex environments. Others have proposed texture-based approaches by detecting local facial features such as pupils, lips, and eyebrows based on an observation that they are normally darker than the regions around them (Huang & Mariani 2000; Hao & Wang 2002). Color feature based face detection is derived from the fact that the skin color of different humans, even from different races, cluster very closely. Several color models are normally used, including RGB (Satoh, Nakamura & Kanade 1999), normalized RGB (Sun, Huang & Wu 1998), HSI (Lee, Kim & Park 1996), YIQ (Wei & Sethi 1999), YES (Saber & Tekalp 1996), and YUV (Marques & Vilaplana 2000). In these color models, HSI is shown to be very suitable when there is a large variation in feature colors in facial areas such as the eyes, eyebrows, and lips. Motion information is appropriate to detect faces or heads when video sequences are available (Espinosa-Duro, Faundez-Zanuy & Ortega 2004; Deng, Su, Zhou & Fu 2005). Normally frame difference analysis, or moving image contour estimation, is applied for face region segmentation. Recently, researchers tend to focus more on multiple feature methods which combine shape analysis, color segmentation, and motion information to locate or detect faces (Qian & Li 2000; Widjojo & Yow 2002).

The Template matching approach can be further divided into two classes: feature searching and face models. Feature searching techniques first detect the prominent facial features, such as eyes, nose, and mouth, then use knowledge of face geometry to verify the existence of a face by searching for less prominent facial features (Jeng, Liao, Liu & Chern 1996). Deformable templates are generally used as face models for face detection. Yuille et al. (1989) extend the snake technique to describe features such as eyes and the mouth by a parameterized template. The snake energy comprises a combination of valley, edge, image

brightness, peak, and internal energy. In Cootes and Taylor's work (1996), a point distributed model is described by a set of labeled points. Principal Component Analysis is used to define a deformable model. Image-based approaches treat face detection as a two class pattern recognition problem and avoids using prior face knowledge.

It uses positive and negative samples to train a face/non-face classifier. Various pattern classification methods are used, including Eigenfaces (Wong, Lam, Siu, & Tse 2001), Neural Network (Tivive & Bouzerdoum 2004), Support Vector Machine (Shih & Liu 2005), and Adaboost (Hayashi & Hasegawa 2006).

2.2.1 Basic understanding

The human face is a dynamic object and has a high degree of variability that is why face detection is considered a difficult problem in computer vision. Several techniques were proposed, varying from simple edge-based algorithms to composite high-level approaches. Another way for algorithms to be classified is feature-based or image-based algorithms. Erik stated that "Many of the current face recognition techniques assume the availability of frontal faces of similar sizes," but in reality this assumption may not hold due to the varied nature of face appearance and environment conditions such as background. The author defined, in general, the face detection problem as given a still or video image and the main goal is to detect and localize an unknown number of faces, the common solutions can be segmentation, extraction, verification of faces, and facial features. The general classification for face detection algorithms is represented in Figure 1.

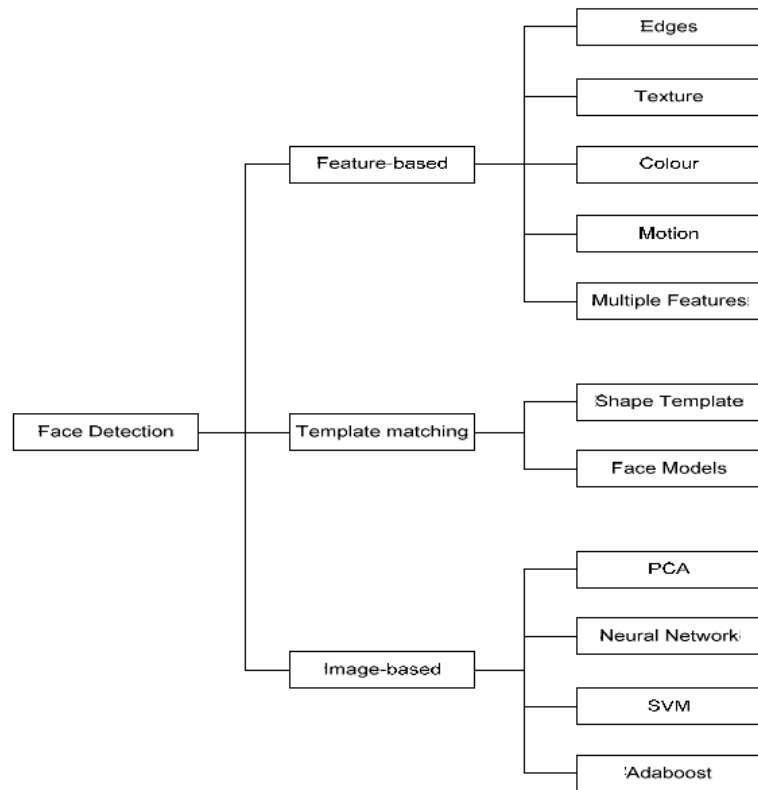


Figure 1: Face detection techniques.

Our research is mainly focused on the feature-based approach in Figure 1. The feature-based approach can be further divided into three categories. Low-level Analysis deals with the segmentation of visual features using pixel properties such as gray-scale and color. Feature analysis is that the locations of the face and facial features are determined. Active shape model is complex and non-rigid feature extraction such as eye pupil and lip tracking.

The author mentioned a generalized measuring technique, such as symmetry measure, that assigns a magnitude at every pixel location in an image based on the contribution of surrounding pixels. The symmetry magnitude map clearly shows the locations of facial features, such as the eyes and the mouth, and can produce a success rate of up to 95% in detecting the eyes.

Jeng et al. (S. H. Jeng, H. Y. M. Liao, C. C. Han, M. Y. Chern, and Y. T. Liu, Facial feature detection using geometrical face model: An efficient approach, Pattern Recog. 31, 1998.) proposed a system for face and facial feature detection which is also based on anthropometric measures. In this system, to detect the eye, the algorithm searches for a nose, a mouth, and eyebrows. They used the evaluation function, E , to determine the final most likely face candidate weighted by their facial importance with coefficients as shown in the following equation

$$E = 0.5 * Eye + 0.2 * Mouth + 0.1 * Right_Eyebrow + 0.1 * Left_Eyebrow + 0.1 * Nose.$$

Finally the author concludes that the face detection is a preprocessor in face recognition, and offline processing for face detection technology has reached a saturation point. However, accurate detection of facial features, such as the corners of the eyes and mouth, is more difficult, and this is still a hard problem to solve.

2.2.2 Hausdorff Distance

One of the famous algorithms for face detection is the Hausdorff distance. It calculates how much two given objects are different from each other. If all points of one set are close to the points of the target set, it matches in the Hausdorff distance. In computer vision, the Hausdorff distance can be used to find a given template in an arbitrary target image. By using an edge detector, a set of images are often pre-processed. After processing, each activated point in the binary image of the template is treated as a point in a set; the shape of the template as well as an area of the binary target image is treated as a set of points. The next step is to minimize the Hausdorff distance between the template and some area of the target image. The area in the target image with the minimum value of Hausdorff distance is

considered the best candidate for locating the template in the target. This algorithm is used for object detection in given images. [6]

- Definition

Given two finite point sets, $A = \{a_1, \dots, a_m\}$ and $B = \{b_1, \dots, b_n\}$, then the Hausdorff distance is defined as

$$H(A, B) = \max(h(A, B), h(B, A)), \text{ where } h(A, B) = \max_{a \in A} \min_{b \in B} \|a - b\|.$$

Hereby $h(A, B)$ is called the directed Hausdorff distance, HD, from set A to B with some underlying norm $\|\cdot\|$ on the points of A and B .

Let the two-dimensional point sets A and B denote representations of the image and the object. Hereby, each point of the set stands for a certain feature in the image, e.g. an edge point. The goal is to find the transformation parameters $p \in P$ such that the HD between the transformed model $T_p(B)$ and A is minimized (see Fig 2).

The detection problem can be formulated as

$$d_{\hat{p}} = \min_{p \in P} H(A, T_p(B))$$

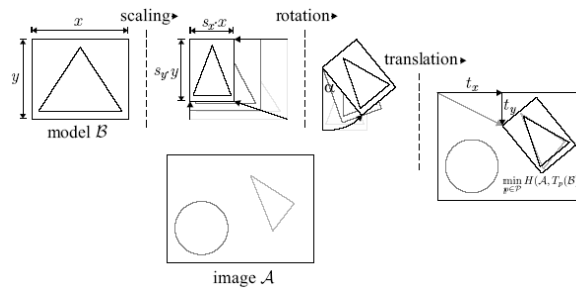


Figure 2: Model fitting by scaling, translation and rotation.

2.2.3 Deformable Template Matching

One of the algorithms that extract the eyes in the given image is known as Deformable Template Matching. This new method overcomes the shortcomings of traditional algorithms, such as unexpected shrinking of the template and the complexity of the updating procedure, while offering higher flexibility and accuracy. According to Paul Kuo[8], the eye features are fitted in a pre-set order to reduce the complexity of the updating procedure and increase the flexibility.

Accurate eye extraction is a key step for many applications such as model-based human face coding, facial expression recognition, human machine interface, biometric identification, and our driver safety system. In this algorithm, it finds and locates one or more of the following eye features: pupil, eye contour, upper and lower eyelids, and iris rather than simple eye location.

Yuille et al. [9] developed a technique using deformable templates to fit the eye features. In order to deform their template to properly fit an eye, they employed 11 parameters and 10 weights to construct a cost function which was then minimized using a steepest descent algorithm, combined with a staged updating procedure. This optimization procedure is complex and requires careful initialization. Including Yuille et al, many researchers found the intersections of the upper and lower eyelids and the intercept points of the iris circumference and the upper eyelid. By forcing the eyelid parabolas to pass through these points they claim faster and more accurate eye fitting. Intercept points of the iris circumference and the upper eyelid may not exist in the case of wide open eyes.



Figure 3: Examples of iris fitting.

The pictures above show a 40 x 20 pixel rectangular search region, centred at the PCA estimated eye centre. A circle of variable size is scanned across the search region to find the best fit. The fitting process uses the Intensity Field, the Edge Field and the Radius of the Iris.

The Intensity Field for color images can be expressed like below.

$$I(x, y) = \frac{1}{3} \times [R(x, y) + G(x, y) + B(x, y)]$$

Where $R(x, y)$, $G(x, y)$, $B(x, y)$ are the intensity values in each R, G, and B color channel. Paul used a Sobel edge operator instead of the Edge Field (x, y) from Yuille's for extracting only the vertical parts of edges. The reason is that the upper and lower parts of the iris are frequently occluded by the eyelids so that only the sides of the iris may be visible. Moreover, only the vertical parts of edges on the sides of the iris may be visible, which is the iris boundary. To avoid the template shrinking to the darkest spot inside the iris, an additional size term can be used. This is the expected Radius of Iris, $R_{expected}$. The preprocessing PCA stage provides the approximate distance between the eyes and Paul sets $R_{expected}$ to be one-tenth of the eye separation, on the basis of experimental observation.

To fit the deformable circle to the iris, this needs to maximize the following function

$$\frac{W_I}{A_{cir}} \sum_{(x,y) \in A_{ctr}} [255 - I(x,y)] + \frac{W_E}{L_{cir}} \sum_{(x,y) \in L_{ctr}} \emptyset(x,y) + W_s \times 255 \left[1 - \frac{|R_{expected} - R_{deform}|}{R_{expected}} \right]$$

A_{cir} and L_{cir} are the area and circumference of the deformable circle. W_I , W_E and W_s are the weighting coefficients associated with intensity, edge and size terms respectively. Because 24 bit color image are used in this function, 255 is the maximum value in Intensity and Edge Field. For the best fit, the area of the circle should be dark. A large intensity contrast should be present along the circumference and the size of the circle should be close to the expected value. The R_{deform} is allowed to vary from $0 \sim 2 \times R_{expected}$.

A vector, with a length equal to three times of the radius of the iris, and an origin at the centre of iris are utilized. This vector is fitted with an arrow head which subtends a 30° angle and has a length equal to the diameter of the iris. The vector rotates through $\pm 20^\circ$ with respect to the line between the two eye centers. The arrow head is curved along the iris boundary. If the vector is pointing towards the eye corner, the area enclosed by the arrow head should contain only white sclera. The vector can be represented like below.

$$\begin{bmatrix} -3 & -2 & -1 & 0 \\ -2 & -1 & 0 & 1 \\ -1 & 0 & 1 & 2 \\ 0 & 1 & 2 & 3 \end{bmatrix}$$

The corner templates are incorporated in the calculation of where the vector arrow head finds the maximum of the function given below.

$$\frac{W_s}{A_{arrow}} \sum_{(x,y) \in A_{arrow}} P[255 - S(x, y)] + \frac{W_c}{20} \times \text{MAX}\{E^+ + E^-\}$$

S(x, y)	≥ 100	75~100	50~75	40~50	< 40
P	0.5	1	2	3	4

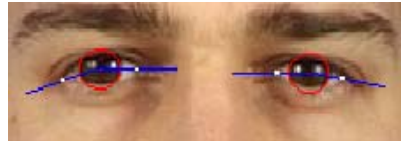
$S(x, y)$ denotes the saturation value at pixel coordinate (x, y) . The first term in this equation computes the average inverted saturation. Coefficient P is inserted to encourage lower saturation values while penalizing higher saturation values. The second term searches for the maximum value after performing a 2-dimensional convolution, along the vector, with the predefined templates. E^+ is the convolved result using the positive gradient template and E^- is the counterpart. A scale of $\frac{1}{20}$ is used to reduce the dynamic range to $0 \sim 255$.



(A) Pointer 00011



(B) Pointer 02011



(C) Corner 00011



(D) Corner 02011

Figure 4: Example of finding eye corners.

(A) and (B) show rotating vectors and arrow heads used to find the eye corners. (C) and (D) show the vectors and corners that are marked as white dots.

2.2.4 Geometric Properties of Topographic Manifold

Wang [10] proposes a novel approach to detect and track eyes using geometric surface features on topographic manifolds of eye images. The first step using Geometric Properties of Topographic Manifolds is making the face an image as a 3D terrain surface in the joint spatial-intensity domain. Most of all, eye areas are formed intrinsic geometric traits on this topographic manifold and hillside-like surround regions. To apply a terrain classification procedure, each location of the manifold can be labeled to generate a terrain map. Wang used the distribution of terrain labels to represent the eye terrain pattern. To measure the distribution similarity between two topographic manifolds, Wang introduced Bhattacharyya affinity. According to the Bhattacharyya kernel, a support vector machine or SVM is applied for selecting proper eye pairs from the pit-labeled candidates.

After labeling each location that creates a terrain map, a mutual-information-based fitting function is defined to describe the similarity between two terrain surfaces of neighboring frames, and eye locations are updated for subsequent frames by optimizing the fitting function.

Using the topographic manifold, Wang said that the appearance variables are eventually linked to the surface characteristics of faces in the 3D scene. Identifying the surface type of a certain area of the topographic manifold can help infer the corresponding face region. Fig 2-5 shows an example of a face image and the topographic manifold of the eye region and the geometric property of the topographic manifold reflects the texture and shape appearance of the 2D image.

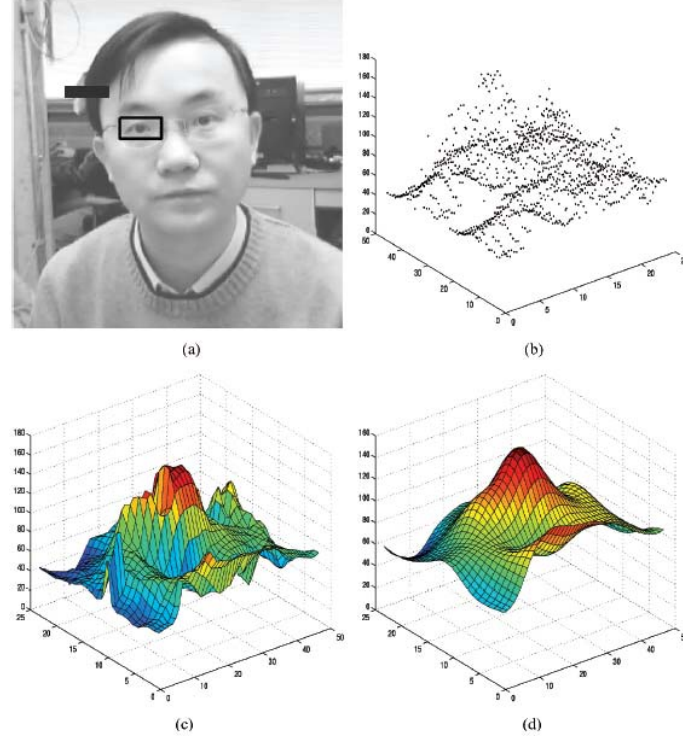


Figure 5: A face image and corresponding 3D terrain surface of the eye region.

In this figure, the surface is reversed for better visualization, so the peak denotes the pit in the real surface: (a) original face image, marked out by an eye patch with a size of 24×48 pixels; (b) corresponding distribution in the joint spatial-range space with 1152 points; (c) continuous terrain surface of the eye patch in the original image; (d) smoothed terrain surface of the eye patch using a Gaussian filter with a kernel size of 15×15 and $\sigma = 2.5$.

Wang introduced the Hessian matrix that can be expressed like below.

$$H(x, y) = \begin{bmatrix} \frac{\partial^2 f(x, y)}{\partial x^2} & \frac{\partial^2 f(x, y)}{\partial x \partial y} \\ \frac{\partial^2 f(x, y)}{\partial x \partial y} & \frac{\partial^2 f(x, y)}{\partial y^2} \end{bmatrix}$$

After applying Eigenvalue decomposition to the Hessian matrix, Wang got

$$H = UDU^T = [u_1 \ u_2] \cdot \text{diag}(\lambda_1 \lambda_2) \cdot [u_1 \ u_2]^T,$$

where λ_1 and λ_2 are the Eigenvalues and u_1, u_2 are the orthogonal Eigenvectors. A pit pixel can be detected when a local minimum gradient $\|\nabla f(x, y)\|$ is found in the local region. In other words, the following conditions must be satisfied.

$$\|\nabla f(x, y)\| = 0, \lambda_1 > 0, \lambda_2 > 0$$

2.2.5 Eye Location Method Using Ordinal Features

Feng [11] proposes a new eye location method using ordinal features. According to Feng, the advantage of ordinal features is they are very robust to the change of illumination and noise, and they can be calculated very fast by the integral image. It has two steps; first, it finds each eye center position using AdaBoost algorithm and Haar-like features. Then, the ordinal features helps to decide the final result.

Using AdaBoost detector, it can create a two dimension array $R(x, y)$ like below

$$R(x, y) = \sum_{i=1}^M I_i(x, y),$$

where M is the number of candidate points, and I_i is defined like below

$$I_i(x, y) = \begin{cases} 1 & (x-x_i)^2 + (y-y_i)^2 \leq C^2 \\ 0 & \text{otherwise} \end{cases}$$

where (x_i, y_i) is the coordinate of each candidate point, and C is the radius of the response area. Then, Feng added evaluation function h_k for the ordinal feature detector.

$$h_k(x, y) = \begin{cases} 1 & p_k C_{k1}(x, y) < p_k C_{k2}(x, y) \\ 0 & \text{otherwise} \end{cases}$$

Where C_{k1} and C_{k2} are mean value of the pixels in two sub-windows, p_k is the direction of the inequality sign and (x, y) is the coordination of center of the eye region.

Feng tested this with 2,800 frontal face images by comparing three methods: probability-based method, AAM method proposed by Cootes, and Using Ordinal Features. The average locating time can be shown in the table below.

Method	Using Ordinal Features	Probability-Based	AAM
Time (ms)	12	25	35

The author insists that using ordinal features has a similar performance with the probability-based algorithm in bad illumination condition and too many noises. However, the eye location method using ordinal features is much faster than other two algorithms. He concludes that proper intermediate features built based on the simple ordinal features can achieve better results.

2.2.6 Support Vector Machines

Support Vector Machines, SVM, is an imaged-based face detection technology. This algorithm gets input data while building a model of training phase. It predicts future data by building output a hypothesis function. According to Michel [12], given a set of labeled training examples

$$\mathbf{S} = ((\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_l, \mathbf{y}_l)), \mathbf{y}_i \in \{-1, 1\}$$

learning systems will find a decision function of the form

$$h(x) = \text{sgn}(< w \cdot x > + b)$$

that yields a label $\in \{-1, 1\}$ for a previously unseen example x . Learning machines gather input data into a high dimensional feature space. To separate embedded input data, linear algebra and geometry are used, and they are formulated to make use of kernel functions. This process allows efficient computation of inner products in feature space without explicit embedding data. Kernel function can be expressed like below

$$K(x, z) = < \Phi(x) \cdot \Phi(z) >$$

where Φ is a non-linear mapping. This kernel function in SVM algorithms can find the hyper-plane of maximal margin, defined as the sum of the distances. The non-linear functions of the SVM algorithm has a formula like below

$$f(x) = \text{sgn}\left(\sum_{i=1}^l a_i y_i K(x_i \cdot x) + b\right)$$

where the a_i are Lagrange multipliers of a dual optimization problem.

According to Michel [12], the training set is interactively created by the user and hence limited in magnitude and that the individual training examples are of constant and small size; overhead is low for typical training runs. This is also aided by the sparseness of the SVM solution, manifested by the fact that the numbers of support vectors which define the decision surface only increases sub-linearly as more examples are added to the training data.

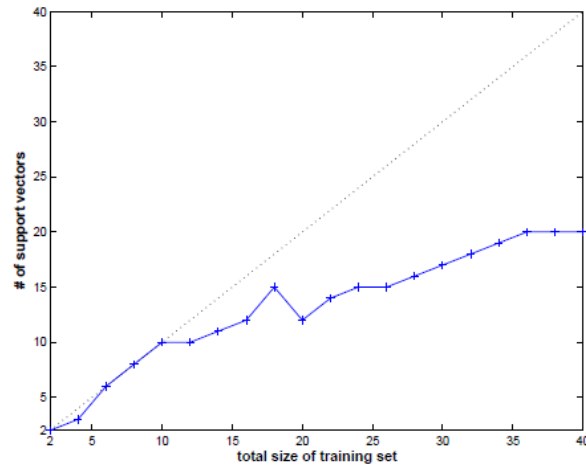


Figure 6: Number of support vectors defining the decision surface as a training set size is increased.

For this reason, Figure 6 shows that evaluation of an SVM decision function on unseen input essentially amounts to checking which of the two subspaces defined by a separating hyperplane a point lays in; classification overhead is negligible.

2.3-Face Detection Software Development Kits

There are several libraries or SDKs for helping to develop face detection. These SDKs are mostly support any development environments such as MATLAB, Java, C, C++, C#, Pascal, Delphi and even Flash Action script. In this section, the most famous commercial and non-commercial libraries or SDKs will be introduced.

2.3.1 OpenCV

OpenCV is a library of programming functions mainly aimed at real time computer vision originally developed by Intel. It is free for commercial and research use. Officially launched in 1999, the OpenCV project was initially an Intel Research initiative to advance

CPU-intensive applications, part of a series of projects including real-time ray tracing and 3D display walls. The main contributors to the project included Intel's Performance Library Team, as well as a number of optimization experts in Intel Russia [14].

The creators of OpenCV wrote that "Learning OpenCV puts you right in the middle of the rapidly expanding field of computer vision. The widely used free open-source library, this book introduces you to computer vision and demonstrates how you can quickly build applications that enable computers to see and make decisions based on the data." OpenCV provides an easy-to-use computer vision infrastructure along with a comprehensive library containing more than 500 functions that can run vision code in real time. With Learning OpenCV, any developer or programmer can get up and running with the framework quickly, whether it should build in simple or sophisticated application. Below is a simple algorithm in C++ for face detection.

```
// Function to detect and draw any faces that is present in an image
void detect_and_draw( IplImage* img )
{
    // Create memory for calculations
    static CvMemStorage* storage = 0;
    // Create a new Haar classifier
    static CvHaarClassifierCascade* cascade = 0;
    int scale = 1;
    // Create a new image based on the input image
    IplImage* temp = cvCreateImage( cvSize(img->width/scale,img->height/scale),
8, 3 );
    // Create two points to represent the face locations
    CvPoint pt1, pt2;
    int i;
    // Load the HaarClassifierCascade
    cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name, 0, 0, 0 );
    // Check whether the cascade has loaded successfully. Else report an error and
```

```

quit

if( !cascade )
{
    fprintf( stderr, "ERROR: Could not load classifier cascade\n" );
    return;
}

// Allocate the memory storage
storage = cvCreateMemStorage(0);

// Create a new named window with title: result
cvNamedWindow( "result", 1 );

// Clear the memory storage which was used before
cvClearMemStorage( storage );

// Find whether the cascade is loaded, to find the faces. If yes, then:
if( cascade )
{

    // There can be more than one face in an image. So create a growable
sequence of faces.

    // Detect the objects and store them in the sequence
    CvSeq* faces = cvHaarDetectObjects( img, cascade, storage, 1.1, 2,
CV_HAAR_DO_CANNY_PRUNING,

    cvSize(40, 40) );

    // Loop the number of faces found.
    for( i = 0; i < (faces ? faces->total : 0); i++ )
    {
        // Create a new rectangle for drawing the face
        CvRect* r = (CvRect*)cvGetSeqElem( faces, i );

        // Find the dimensions of the face, and scale it if necessary
        pt1.x = r->x*scale;
        pt2.x = (r->x+r->width)*scale;

```

```

        pt1.y = r->y*scale;
        pt2.y = (r->y+r->height)*scale;

        // Draw the rectangle in the input image
        cvRectangle( img, pt1, pt2, CV_RGB(255,0,0), 3, 8, 0 );
    }
}

// Show the image in the window named "result"
cvShowImage( "result", img );

// Release the temp image created.
cvReleaseImage( &temp );
}

```



Figure 7. Demo of face extraction and finding eye using OpenCV

The Haar cascade file is one of the sample scripts that come with OpenCV. It provides functions which can be used to train classifiers for the face detection system. It is called HaarTraining. It makes use of the Haar classifier feature detection and extracts human face like regions and passes them back to the Processing program as an array of rectangle information.

2.3.2 VeriLook SDK

VeriLook SDK is released by NeuroTechnology. This SDK is intended for biometric systems developers and integrators. It allows rapid development of biometric applications

- **Face image quality determination.** A quality threshold can be used during face enrollment to ensure that only the best quality face template will be stored into the database.
- **Tolerance to face posture.** VeriLook has certain tolerance to face posture that assures face enrollment convenience: rotation of a head can be up to 10 degrees from frontal in each direction.
- **Multiple samples of the same face.** Biometric template record can contain multiple face samples belonging to the same person. These samples can be enrolled with different face postures and expressions, from different sources and in different time thus allowing improving matching quality. For example a person could be enrolled with and without eyeglasses or with different eyeglasses, with and without a beard or a moustache, with different face expressions like smiling and non-smiling, etc.
- **Identification capability.** VeriLook functions can be used in 1-to-1 matching verification, as well as 1-to-many mode (identification).
- **Fast face matching.** The VeriLook 3.2 face template matching algorithm compares 100,000 faces per second.
- **Compact face features template.** A face features template occupies only 2.3 Kilobytes, thus VeriLook-based applications can handle large face databases.

Recommended minimal image size	640 × 480 pixels
Multiple faces detection time	0.07 second
Single face processing time	0.13 second
Matching speed	100,000 faces / second
Size of one record in the database	2.3 Kbytes

Table 1: VeriLook SDK technical specifications.

2.3.3 Luxand Face SDK

Luxand FaceSDK is a face detection and recognition library that can easily be integrated into a customer's applications. FaceSDK offers the API, Application Programming Interface, to detect a face and facial features, and to match faces 1:1 and 1:N matching is supported. Following face detection, the SDK provides the coordinates of 40 facial feature points such as eyes, eye corners, eyebrows, mouth corners, and nose tip like Figure 2-9 below.

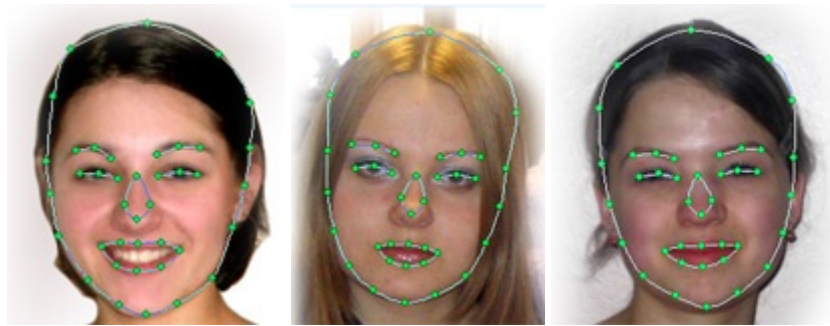


Figure 9: Example of finding 40 facial feature points.

This SDK is supplied as a DLL, a Win32 Dynamic Link Library, and can be used in the number of compilers on the Win32 platform. FaceSDK contains interface header files and sample applications for Microsoft Visual C++ 6.0/7.0/2005/2008, Microsoft .NET and Borland Delphi 6.0/7.0.

The FaceSDK library has the following technical specifications

- Robust frontal face detection
- Detection of multiple faces in a photo
- Detection of 40 facial feature points such as eyes, eyebrows, mouth, nose, and face contour
- Detection time: 0.9 seconds

- Allowed head rotation: –30~30 degrees of in-plane rotation, –10~10 degrees out-of-plane rotation
- Enrollment time: 0.3 seconds
- Template Size: 90 kb
- Matching speed: up to 5,000 faces per second

This SDK introduces the data type, TFacePosition, which stores the information about the position of the face. The following example source code shows how to declare data types. The xc and yc fields specifies the X and Y coordinates of the center of the face, w holds the width of the face, and angle stores the rotations angle of the face in degrees.

C++	Delphi
<pre>typedef struct { int xc, yc, w; double angle; } TFacePosition;</pre>	<pre>TFacePosition = record xc, yc, w: integer; angle: double; end; PFacePosition = ^TFacePosition;</pre>

FSDK_DetectFace function is used to detect a frontal face in an image and stores information about the face position into the data type.

int FSDK_DetectFace(HImage Image, TFacePosition* FacePosition);

// Image: handle of the image to detect the face in.

// FacePosition: pointer to the TFacePosition structure to store face information.

This function returns pre-defined integer value, FSDK_OK, if it finds a face. Otherwise, it returns other integer value to indicate what problems happen such as not found image and too small image. Following C++ source code is an example of how to use this function.

```

int img1;                                // set integer variable
TFacePosition FacePosition;              // declare data type
FSDK_Initialize("");                     // initialized face SDK
FSDK_LoadImageFromFile(&img1, "test.jpg"); // load image file
FSDK_DetectFace(img1, &FacePosition);     // find face and store face information
                                           // print out face information
printf("face position: %d %d %d", FacePosition.xc, FacePosition.yc, FacePosition.angle);

```

2.4 Summary

In this section, some algorithms and SDKs were introduced. As mentioned above, there are many ways to create and develop face detection applications. Mostly, non-commercial SDKs such as OpenCV need a lot of works to build system than commercial SDKs. Also, there are many differences among commercial SDKs. For instance, in the case of developing faster performances; VeriLook SDK is better than Face SDK because VeriLook can match more faces in a given time. However, if a programmer needs more functions to find facial features such as mouth, eyes, eye corners, eyebrows, mouth corners, nose tip and etc. But, none of these SDKs support to find drowsy eyes. This means that SDK helps to find a face and some features but other works should be done by a developer.

Chapter 3: FINDING DROWSY EYES

3.1 Introduction

Once we find a face in a sequence of an image, the matter is to find drowsy eyes in a driver safety system. The pre-crash safety system, introduced by Toyota, is focused on finding how much eye-lids are opened or not. This system can slow down a car or alert the user not to sleep; that is purpose of the pre-crash safety system. This chapter introduces the VeriLook SDK, which is used for finding faces for a driver safety system, and finding drowsy eyes using pixel difference algorithm in AOI, Area of Interest.

3.2 Real Time Driver Safety System

3.2.1 System Environment

In this thesis, VeriLook SDK is selected to detect faces because of its speed. Moreover, IDE, integrated development environment, for this thesis is CodeGear RAD studio that includes C++ builder and Delphi 2009. Delphi is a software development environment for Microsoft Windows applications. It is usually used for the development of desktop and enterprise database applications, but it is a general-purpose software development tool suitable for most software projects. Web applications are also possible due to the inclusion of several libraries. The language is suitable for RAD, Rapid Application Development and comes with an integrated IDE. The Delphi products all ship with a large framework called VCL, Visual Component Library, including most of its source code.

Third-party components are available on the market as well as tools to enhance the IDE or for other Delphi related development tasks such as VeriLook SDK.

To get the image, QuickCam Pro by Logitech is used. For the first time, many sample eye pictures from the old versions of webcam could not get a better result because of its bad quality. The following figure shows the bad qualities the samples have.

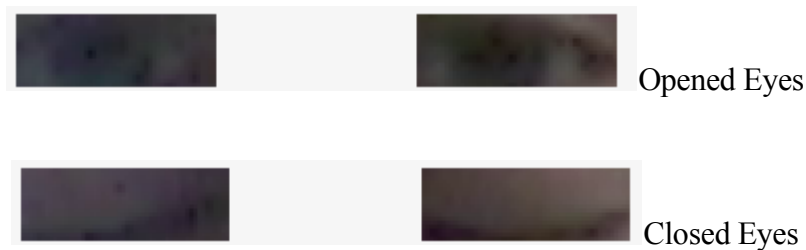


Figure 1: Eye extracted image from old web-cam.

Like the picture above, it is too poor to recognize the image itself. With this quality, the system fails most of time because there are too many digital noises in the picture. Recently, the quality of the webcam image has been improved in the market, such as advanced optics Lenz by Carl Zeiss, and the autofocus keep images razor-sharp. Thanks to this technology for webcam, it supports HD quality images by using the true two-megapixel sensor. Even these small devices also include face detection technology that keeps the user's face in the middle of the action.



Figure 2: Enhanced quality of web-cam image.

3.2.2 System Architecture

The system architecture for driver safety system is almost the same as a pre-crash safety system by Toyota. The system work flow is below in Figure 3.

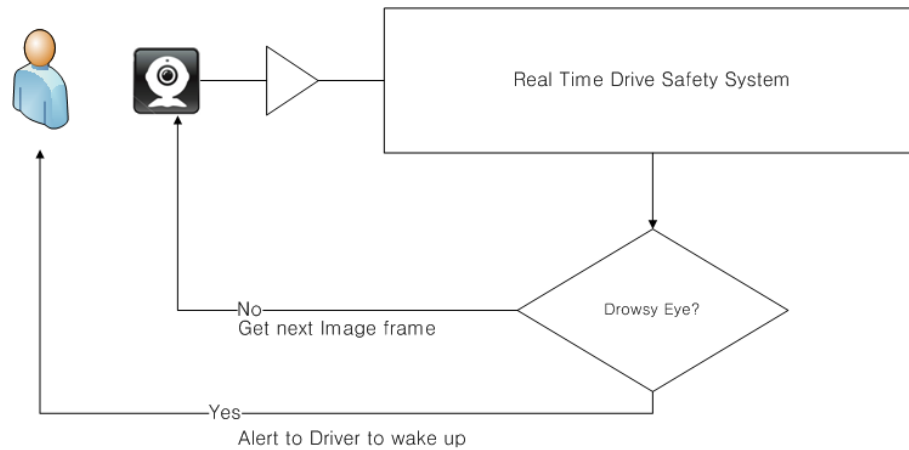


Figure 3: System work flow.

The web-cam mounted in front of a driver captures a sequence of images. The real time driver safety system processes a sequence of images to extract face and eye and runs the algorithm to find out the status of eyes. Inside of the real time driver safety system, there are two modules, extract and process module, to handle a given sequence of images. In the Figure 4, this diagram shows the hierarchy of real time driver safety system. For the first time inside of the system, it needs to grab a single image frame to find a face because no face detection works to movie clip itself. This means that movie clips or streaming videos must be processed to get single image frames. If a face is not found in the single frame, it will grab the next single frame again until the system is terminated or a face is found. Once it finds a face, the next procedure is to execute the extract module that consists of face and eye extractions with storing information such as timestamp, extracted both eye images, and RGB value of a single image frame. The process module is the next step to determine the

status of each eye. If this module finds that it is a drowsy eye, it will send an alert to the driver; otherwise, it will repeat the steps from the beginning.

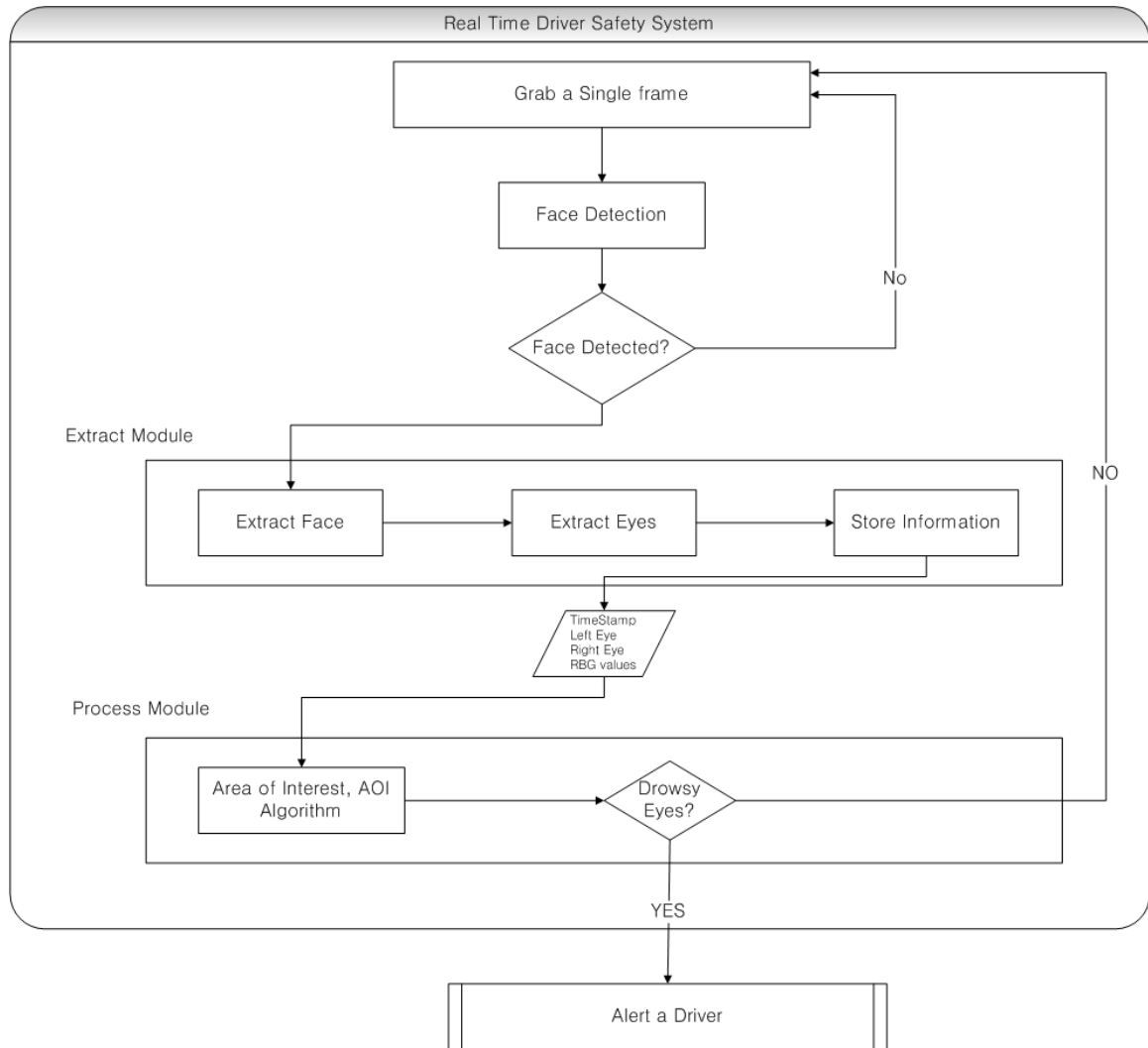


Figure 4: The hierarchy of Real Time Driver Safety System.

3.2.3 Face detection

Before extracting eyes and finding the status of the eyes, the system must find a face first. In this section, this paper will introduce how to program using VeriLook SDK in Delphi. Main functions and their descriptions are in Table 1.

Functions	Description
CameraManager	Show usage of Camera Manager component.
ConcurrentLicensing	Demonstrates usage of concurrent licensing functionality.
CreateMultiFaceTemplate	Creates NTemplate that contains multiple face templates.
EnrollImage	Demonstrates how to write codes for face enrollment to database from a single image.
EnrollStream	Demonstrates how to write code for face enrollment to database from stream.
FindEyes	Demonstrates how to write code to find eyes in face image.
FindFaces	Demonstrates how to write code to find all faces in image.
Identify	Demonstrates how to write code for face identification against database.
MultipleFaceMatching	Demonstrates how to write code for face identification against image containing multiple faces.
TemplateInfo	Demonstrates how to write code to get all information from biometrical templates.
Verify	Demonstrate how to match two face templates.
VerifyImages	Demonstrates how to match two faces images.

Table 1: Main functions and description.

For the first time, the system must find the available number of web-cam devices to be used in this system. In Delphi, there is called OnCreate event that is executed when an

application is about to run. Usually, this event is used to set initial value for the program.

Here is a sample to get the list of webcams in OnCreate event.

```

procedure TMainForm.FormCreate(Sender: TObject); // OnCreate Event
var i: Integer; // for-loop variable
    camera: TCamera; // TCamera Object
begin
    gCameraMan := TCameraMan.Create; // creating camera object

    // display result
    AddToLog('Connected cameras count: ' + IntToStr(gCameraMan.CameraCount));
    AddToLog('Found devices:');
    for i := 0 to gCameraMan.CameraCount - 1 do
    begin
        camera := gCameraMan.GetCameraByIndex(i);
        AddToLog(camera.GetID, false);
        FreeAndNil(camera);
    end;
end;

```

Moreover, the list of web-cams including activated web-cam driver should be released from the memory so that other applications can use the web-cam. OnDestroy event in Delphi happens when a program exits by user or malfunction.

```

procedure TMainForm.FormDestroy(Sender: TObject); // OnDestroy event
var camera: TCamera;
    i: Integer;
begin
    for i := 0 to gCameraMan.CameraCount - 1 do
    begin
        camera := gCameraMan.GetCameraByIndex(i);
        if (camera <> nil) then camera.StopCapturing;
        FreeAndNil(camera);
    end;
    FreeAndNil(gCameraMan);

    Outputfile.Free;
end;

```

Once the list of web-cams was provided, it needs to be a converted image into 8-bits grayscale image to have face detection performed. This function can be done by Extractor's `detecFace` methods consisting of three parameters: image, faces, and face counts.

- Image: handle to the source image
- Faces: pointer to an array of `NleFace` structures of found faces.
- FaceCount: integer variable that is set to the number of found faces in the image.

`NleFace` structures have two fields that store structure with face rectangle information and how confidently the face region was found. If this system finds at least one face, it will call the method, `DetectFacialFeature`. This method finds locations of facial feature points, and the main purpose of this method is to save the position of each eye. It has two parameters to save the information.

- Image: handle to the source image.
- Faces: pointer to the `NleFace` structure of a face found in the image.

When this method is successfully finished, it will return `NleDetectionDetails` structure that brings structure with information about face detection results in a face detection routine. When these methods find face and facial features, the system calls the `DrawMultiFaces` and `DrawEyes` method. `DrawMultiFaces` method draws rectangle lines at the face area as much as the face counts. `DrawEyes` method draws lines between the eyes. The following example is how to use these functions

```

procedure TMainForm.PaintCameraImage(image, rgbImage: Pointer);
var Bitmap: TBitmap;
    faces: TArrayOfTNleFace;
    facesCount: Integer;
    details: TNleDetectionDetails;
begin
    gExtractor.DetectFaces(image, facesCount, faces);
    if (facesCount > 0) then
        details := gExtractor.DetectFacialFeatures(image, faces[0]);
    Bitmap := TBitmap.Create;
    PaintImageToTBitmap(rgbImage, Bitmap);
    // Detecting Eyes
    if facesCount > 0 then
        begin
            DrawMultiFaces(Bitmap, faces, facesCount, true);
            if (details.EyesAvailable = 1) Then
                DrawEyes(Bitmap, details.Eyes, true);
        end
    else
        begin
            isfaceDetected := False;
        end;
    PaintBitmapToTImage(image, Bitmap, imgLeft);
    // Free Bitmap as we don't need it
    FreeAndNil(Bitmap)
end;

```

If a face and the position of the eye are found using these, it will display its position and the application looks like Figure 5.

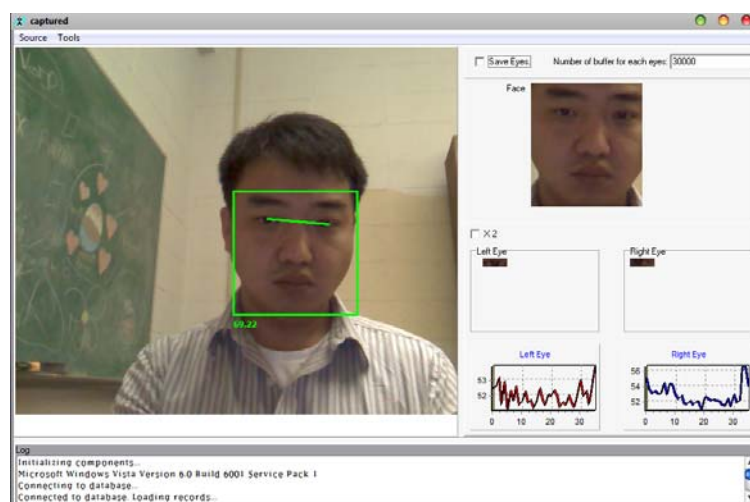


Figure 5: Screen shot of an application.

3.2.4 Extract Module

Next step after finding a face is to execute the extract module. When VeriLook SDK finds a face, it only returns faces with a rectangular area and eye position. Before the process module, the system should extract the left and right eye. Extracting a face is simply cropping the rectangular area that is returned by the SDK. In Delphi, it calls Rect function to create a TRect type that represents the rectangle with the specified coordinates. Face extraction uses Rect function to construct parameters for functions that require TRect, rather than setting up local variables for each parameter. Extracting each eye area was determined based on experience. More than 40 people were tested to extract their eyes many times and finally the best size of their eyes was determined. The ratio can be expressed below and system workflow as Figure 6.

$$H1) \text{Eye.width} = \text{Face.width} / 5$$

$$H2) \text{Eye.height} = \text{Face.height} / 14$$



Figure 6: Extracting Face and Eye.

Since the system knows the position of each eye, it calls the Rect function again to extract eyes. Below the formula shows how to apply the rectangular area.

$$R_{leftEye} = (P_{leftEye} - (Eye.width / 2), P_{leftEye} - (Eye.height / 2), Eye.width, Eye.Height)$$

$$R_{rightEye} = (P_{rightEye} - (Eye.width / 2), P_{rightEye} - (Eye.height / 2), Eye.width, Eye.Height)$$

The Rect stores 4 parameters, left, top, bottom and right. These 4 parameters can represent the coordination of the rectangle. $P_{leftEye}$ and $P_{rightEye}$ are the coordinate points of the left and right eye respectively. The following image is the example of how the image is extracted using the formula above.



Figure 7: Example of extracted left eye.

After the extraction of face and eyes, the system begins to store information such as timestamp, filename for each eye, and RGB values. Figure 8 shows the records of information in the text file.

Timestamp	Filename	RGB Values
11/4/2008 11:54:20 PM	2008_11_4_23_54_20_279	125.345536086775 128.758865248227
11/4/2008 11:54:20 PM	2008_11_4_23_54_20_670	123.475398936171 128.55895390071
11/4/2008 11:54:21 PM	2008_11_4_23_54_21_76	123.234681372548 128.424428104575
11/4/2008 11:54:21 PM	2008_11_4_23_54_21_467	123.81658496732 128.953635620914
11/4/2008 11:54:21 PM	2008_11_4_23_54_21_842	124.565217391304 128.451539855073
11/4/2008 11:54:22 PM	2008_11_4_23_54_22_232	168.390292553191 174.367021276596
11/4/2008 11:54:22 PM	2008_11_4_23_54_22_685	152.269607843137 154.071026282854
11/4/2008 11:54:22 PM	2008_11_4_23_54_22_92	154.025700210858 163.462101063820

Figure 8: The records of information.

The “|” character is a delimiter to separate the information. The first column is the timestamp when the extracted eye image is saved. The next column is a file name that was

made by this format, Year_Month_Day_Hour_Minute_Second_Millisecond, so that file can have a unique name. With this file name, it can store 2 eye image files. For instance, adding the prefix “LE_” and postfix “.bmp” is an image for the left eye. Instead of using the prefix “LE_”, the prefix “RE_” is represented for the right eye image file. Third column and Fourth column are the average RGB values for the left eye and the right eye respectively.

3.2.5 Process Module

Since SDK does not support to find blinking eyes and once the system gets the eyes, the matter is to find whether the driver’s eyes are closed or not. For the first time, this system uses pixel differences of the eyes extract image. There are two ways to find out. One is to get the whole pixel’s average that was saved in text file, and the other is only to calculate pixel value of expected cornea positions which is called AOI, Area of Interest.

For both algorithms, the basic idea is that when eyes are open, there are a variety of pixel values in each eye such as cornea, pupil, iris, and sclera. However, in the eye area, there are only color for skin and eyebrows when the eyes are closed.

3.2.5.1 Pixel Different Algorithm:

Given a picture of the eye area, the system gets red, green, and blue for each pixel, and the total value is divided by the number of pixels. This average value of RGB value for each eye is stored in text file. These have different values each time based on the status of the eyes. Figure 9 shows the sample data with the average of pixel values.

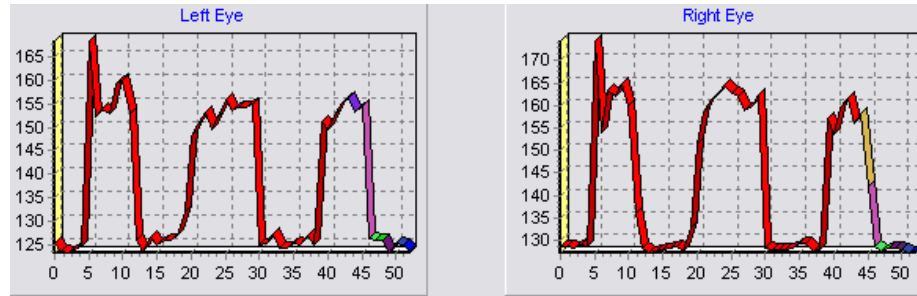


Figure 9: Average pixel value of Sample data.

In Figure 9, the X-axis is the frame number and Y-axis is the average value of pixels. In this experiment, if the value is higher than 130, the eyes are closed. The reason is that when a person closes his or her eyes, usually there is not much different color distributed. This graph indicates that he or she closed the eyes 3 times during 50 frames. However, this way to finding is not accurate because the value is not constant. The value depends on the conditions such as light, skin color, and quality of image. So, the threshold value 130 in the experiment can be changed due to the environment. One more disadvantage is that it takes so much memory space because the system gets RGB value of whole pixels. For this reason, it also takes so much time to consume.

The pseudo code for PDA algorithm is below.

```

Pixel-Difference(Images LeftEye, RightEye)
0  LeftSum, RightSum = 0
1  for i = 0 to LeftEye.width
2      for j = 0 to LeftEye.height
3          LeftSum = LeftSum + (LeftEye[i,j].R + LeftEye[i,j].G + LeftEye[i,j].B) / 3
4  for i = 0 to RightEye.width
5      for j = 0 to RightEye.height
6          RightSum = RightSum + (RightEye[i,j].R + RightEye[i,j].G + RightEye[i,j].B) / 3

```



```

7  avgL = LeftSum / (LeftEye.width * LeftEye.height)
8  avgR = RightSum / (RightEye.width * RightEye.height)
9  display avgL, avgR

```

In this case, the expected running time of the algorithm is $\Theta(N^2)$.

3.2.5.2 Area of Interest (AOI) Algorithm:

In this algorithm, we shift our concentration to become focusing on pupil position. The idea behind this algorithm is to calculate the expected pupil's position. Given a picture of the eye area, get the three points of each eye like figure10.

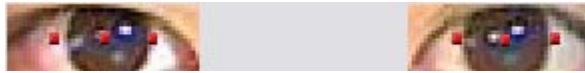


Figure 10: AOI, Three points of each eye.

When a driver looks anywhere, his or her pupil is located in at least one of three points. However, the pixel RGB values of the three points do not have much difference when a driver closes his or her eyes because three points get only the skin's pixel value. Calculating can be expressed like below.

- 1) $Max(p) - Min(p) > threshold$ Eyes are opened
- 2) *Otherwise*, Eyes are closed

The pseudo code for AOI algorithm is below.

```

AreaOfInterest(EyeImage LeftEye, RightEye)
0  for LeftEye and RightEye do
1  position X2 = EyeImage.width / 2
2  position X1 = positionX2 / 2

```

```

3  positionX3 = positionX1 + positionX2
4  positionY = EyeImage.height / 2
5  SumL1, SumL2, SumL3 = 0
6  create a block for every P1, P2, and P3 as a center and 5 pixel length
7  numStep = 0
8  for i = -blockSize to blockSize
9      for j = -blockSize to blockSize
10         ++numStep
11         L1 = LeftEye[positionX1+i, positionY+j].R
              + LeftEye[positionX1+i, positionY+j].G
              + LeftEye[positionX1+i, positionY+j].B
12         L2 = LeftEye[positionX2+i, positionY+j].R
              + LeftEye[positionX2+i, positionY+j].G
              + LeftEye[positionX2+i, positionY+j].B
13         L3 = LeftEye[positionX3+i, positionY+j].R
              + LeftEye[positionX3+i, positionY+j].G
              + LeftEye[positionX3+i, positionY+j].B
14         sumL1 = sumL1 + L1 / 3
15         sumL2 = sumL2 + L2 / 3
16         sumL3 = sumL3 + L3 / 3
17  avgL1 = sumL1 / numStep
18  avgL2 = sumL2 / numStep
19  avgL3 = sumL3 / numStep
20  DiffL = max(avgL1, avgL2, avgL3) - min(avgL1, avgL2, avgL3)
21  if (DiffL < threshold)
22      then display "Close"
23  else display "Open"

```

The AOI algorithm expected running time of the complexity is $\Theta((N/16)^2)$ where N is the number of pixels in the image. In the comparison with the PDA algorithm, this AOI

algorithm scans only expected pupil's position, while PDA algorithm scans whole entire image. For example, given an eye image, the image will be divided by two vertically then be divided by four horizontally to get the three points.

From the set of experiments we conducted, it successfully returned the status of the eyes when the threshold value is 70. This means that one of the points in AOI has 70 higher value than the other two values, and that higher value is located in his or her pupil. Figure 11 shows that it detected the status of the eyes successfully.

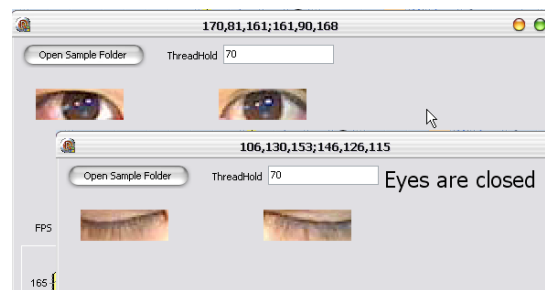


Figure 11: Screen shot of process module.

3.2.6 System Integration

To make this system in real-time, extract and process modules are needed for integration. A new version of Real-Time Driver Safety System has more options to alert to the user in Figure 12.

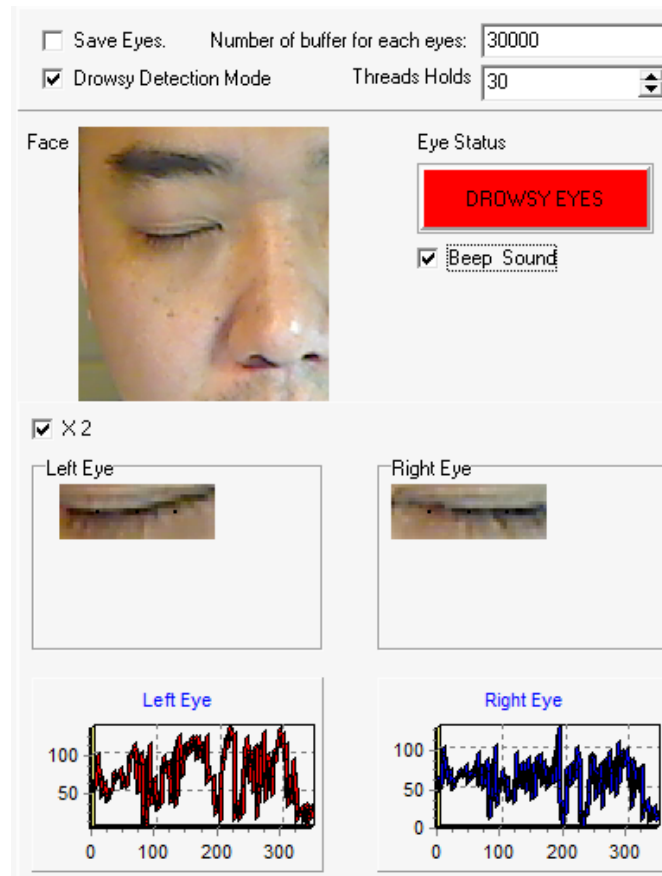


Figure 12: Real-Time Driver Safety System.

When this system saves data, the maximum number of the lines can be defined at the “Number of buffer for each eye.” Because the data is saved as a text file, the system needs to limit the record. Text file can hold up to 64Kb or 65536 Byte, which means that the text file can have up to 65,536 characters. For instance, like above, this system will clear data and rewrite data from the beginning when the line of data is met 30,000 lines. There is a check box called “Drowsy Detection mode” that is for selecting the drowsy eyes detection mode. When this is checked, the system will execute the process module in real time. To alert a driver, there are two ways, visual display and audio.

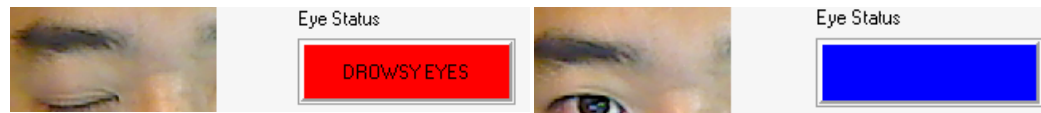


Figure 15: Visual alert.

If the “Eye Status” has a blue color, the driver’s eyes are opened. Otherwise, it displays the message “DROWSY EYES” with red color. But, this visual alert is not helpful to a programmer because the programmer must close his or her eyes to test the system, and he or she could not see this message.

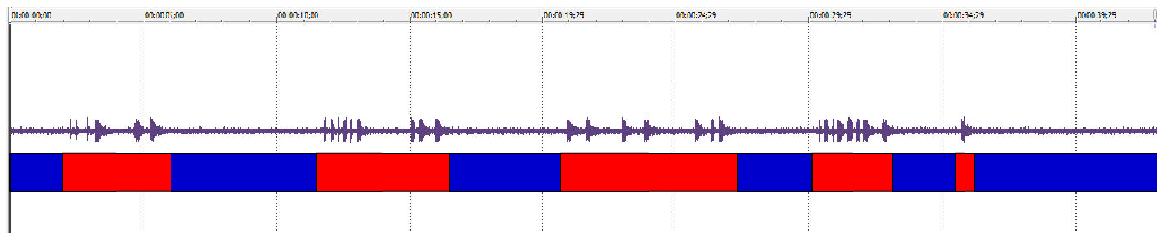


Figure 16: Sound alert.

When a user checks the “Beep Sound,” this system emits a message beep when it finds driver’s drowsy eyes. The beep is a single tone sound that is made by a PC-speaker. In the Figure 16, it is tested for about 40 seconds. The vertically broad width of the waves shows that the system emits a beep sound when it finds the driver’s drowsy eyes.

Chapter 4: Experimental Results

4.1 Results

During developing this system, about 100 people were tested. At the beginning, the biggest problem this system had was that the sample eye images had bad image quality. After we replaced the web-cam, the system could have better accuracy. In this section, several results are introduced by the experiments with time consumption, light, skin color, and glasses effect.

4.1.1 Time consumption

When the extract and process module is executed, the system creates a data file. This file contains timestamps that represent how much time was needed to get the result through the whole process.

In this experiment, there are 51 records during 20 seconds with 800×600 resolution of the web-cam. Based on the result in Figure 1, this system takes 2.55 frames per second. The time, T , field can be calculated using the formula below:

$$T(n) = \text{TimeStamp}(n) - \text{TimeStamp}(n-1) \text{ where } n \geq 2$$

	A	B	C	D	E	F
1	Frame #	TimeStamp	File name	Left Eye Avg	Right Eye Avg	Time
2	1	11/4/2008 23:54	2008114235420270	125.3455361	128.7588652	
3	2	11/4/2008 23:54	2008114235420670	123.4753989	128.5589539	400
4	3	11/4/2008 23:54	2008114235421070	123.2346814	128.4244281	400
5	4	11/4/2008 23:54	2008114235421460	123.816585	128.9536356	390
6	5	11/4/2008 23:54	2008114235421840	124.5652174	128.4515399	380
7	6	11/4/2008 23:54	2008114235422230	168.3902926	174.3670213	390
8	7	11/4/2008 23:54	2008114235422680	152.2696078	154.0710263	450
9	8	11/4/2008 23:54	2008114235423090	154.0257092	163.4621011	410
10	9	11/4/2008 23:54	2008114235423480	152.8597074	162.0066489	390
11	10	11/4/2008 23:54	2008114235423870	159.3182624	164.8922872	390
12	11	11/4/2008 23:54	2008114235424260	160.0163043	159.7930254	390
13	12	11/4/2008 23:54	2008114235424700	152.7629328	136.18398	440
14	13	11/4/2008 23:54	2008114235425100	125.9246324	128.2397876	400
15	14	11/4/2008 23:54	2008114235425490	123.7242386	127.3527326	390
16	15	11/4/2008 23:54	2008114235425870	126.5777013	127.5135586	380
17	16	11/4/2008 23:54	2008114235426270	124.7963079	128.0080309	400
18	17	11/4/2008 23:54	2008114235426650	125.8224638	128.3618659	380
19	18	11/4/2008 23:54	2008114235427120	125.8352107	128.7187109	470
20	19	11/4/2008 23:54	2008114235427490	126.7814977	127.6058615	370
21	20	11/4/2008 23:54	2008114235427870	131.912912	133.0150188	380
22	21	11/4/2008 23:54	2008114235428260	148.0776144	151.3298611	390
23	22	11/4/2008 23:54	2008114235428670	150.6839539	158.5232713	410
24	23	11/4/2008 23:54	2008114235429040	153.2688387	161.4000443	370
25	24	11/4/2008 23:54	2008114235429480	149.5652174	162.2413949	440
26	25	11/4/2008 23:54	2008114235429870	152.169837	164.3976449	390
27	26	11/4/2008 23:54	2008114235430230	155.9560688	162.8011775	360
28	27	11/4/2008 23:54	2008114235430630	153.445922	162.0647163	400
29	28	11/4/2008 23:54	2008114235431020	154.2661791	158.8286791	390
30	29	11/4/2008 23:54	2008114235431420	154.3550725	160.1000906	400
31	30	11/4/2008 23:54	2008114235431850	154.9945652	162.4189312	430

Figure 1: Sample data.

The average time that stores as single frame and process through Real Time Driver Safety System to find the drowsy eyes is 401 milliseconds. This system sends alert to the driver who closed eye for three seconds after this system checks about 12 frames. However, a driver can travel 88 feet per second when he or she drives a car 60 mph. For the three seconds, it will travel 264 feet.

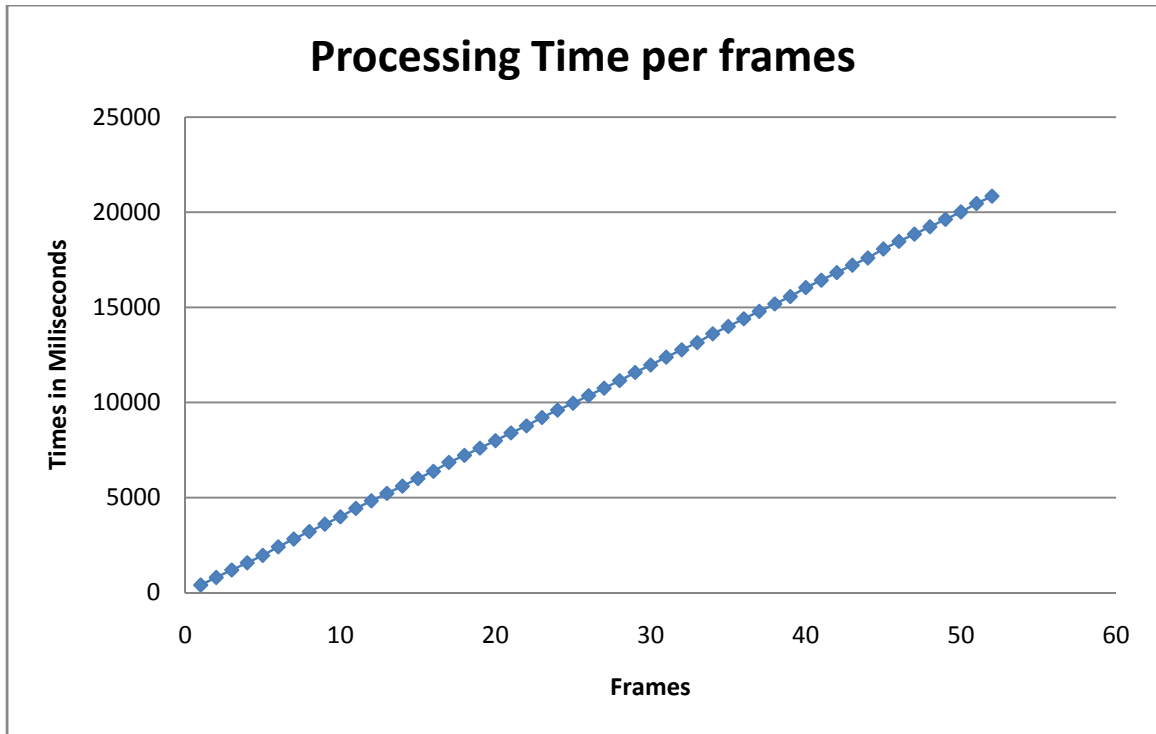


Figure 2: Processing Time per Frames.

4.1.2 Light influence

With more than 40 people tested, the system worked fine when it was given by a proper threshold value. This value can be changed depending on the light.

Figure 3 and 4 show the influence of the light that can determine proper threshold value. In Figure 3, it was tested under the light directly. With a 70 threshold value, this system brings 100 percent accuracy to find drowsy eyes. However, the second figure was tested in an office, which LUX value is usually between 200 and 300. In this case, 20 is the best option for finding drowsy eyes with 97.2 percent accuracy.









































































	LeftEye	RightEye	10	20	30	40	50	60	70	80
1			OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
2			OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
3			OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
4			OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
5			OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE
6			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
7			OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE
8			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
9			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
10			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
11			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
12			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
13			OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
14			OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
15			OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
16			OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
17			OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
18			OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
19			OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
20			OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE
21			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
22			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
23			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
24			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
25			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
26			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
27			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
28			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
29			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
30			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
31			OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
32			OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
33			OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
34			OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
35			OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
36			OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
Sucess %			52.8	55.6	55.6	55.6	61.1	97.2	100	94.4

Figure 3: Result with bright light.









































































	LeftEye	RightEye	10	20	30	40	50	60	70	80
1			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
2			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
3			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
4			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
5			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
6			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
7			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
8			CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
9			CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
10			CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
11			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
12			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
13			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
14			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
15			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
16			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
17			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
18			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
19			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
20			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
21			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
22			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
23			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
24			CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
25			CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
26			CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
27			CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
28			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
29			CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
30			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
31			CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
32			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
33			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
34			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
35			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
36			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
Success %			80.6	97.2	88.9	63.9	44.4	44.4	44.4	44.4

Figure 4: Result with poor light.

4.1.3 Glasses influence

One of the experiments shows how this system works with a person who wears glasses. In Figure 5 and 6, it was tested with the same person in an office and the result indicates that the glasses did not affect the accuracy of our system too much. With a person wearing glasses, the system determines a 30 threshold value with 97.2 percent accuracy and 20 threshold with same accuracy without glasses. When a person wears glasses, the system needs a higher threshold value because the glasses reflect the light as shown in Figure 6.









































































	LeftEye	RightEye	10	20	30	40	50	60	70	80
1			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
2			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
3			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
4			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
5			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
6			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
7			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
8			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
9			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
10			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
11			CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
12			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
13			CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
14			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
15			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
16			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
17			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
18			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
19			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
20			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
21			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
22			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
23			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
24			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
25			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
26			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
27			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
28			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
29			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
30			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
31			CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
32			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
33			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
34			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
35			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
36			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
Sucess %			55.6	97.2	91.6	66.7	52.8	52.8	52.8	52.8

Figure 5: Experiment without glasses.









































































	LeftEye	RightEye	10	20	30	40	50	60	70	80
1			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
2			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
3			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
4			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
5			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
6			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
7			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
8			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
9			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
10			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
11			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
12			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
13			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
14			CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
15			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
16			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
17			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
18			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
19			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
20			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
21			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
22			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
23			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
24			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
25			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
26			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
27			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
28			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
29			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
30			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
31			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
32			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
33			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
34			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
35			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
36			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
Sucess %			66.7	91.7	97.2	55.6	36.1	36.1	36.1	36.1

Figure 6: Experiment with glasses.

4.1.4 Skin Colors

Given the same environment, these results show how skin color can affect the real time driver safety system.









































































	LeftEye	RightEye	10	20	30	40	50	60	70	80
1			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
2			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
3			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
4			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
5			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
6			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
7			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
8			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
9			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
10			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
11			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
12			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
13			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
14			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
15			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
16			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
17			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
18			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
19			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
20			OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE
21			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
22			OPEN	OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE
23			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
24			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
25			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
26			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
27			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
28			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
29			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
30			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
31			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
32			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
33			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
34			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
35			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
36			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
Success %			61.1	72.2	86.1	72.2	69.4	66.7	61.1	61.1

Figure 7: Effect of white male skin color in system performance.









































































	LeftEye	RightEye	10	20	30	40	50	60	70	80
1			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
2			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
3			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
4			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
5			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
6			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
7			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
8			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
9			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
10			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
11			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
12			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
13			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
14			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
15			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
16			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
17			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
18			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
19			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
20			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
21			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
22			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
23			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
24			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
25			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
26			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
27			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
28			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
29			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
30			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
31			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
32			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
33			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
34			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
35			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
36			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
Success %			55.6	55.6	88.9	91.7	52.8	44.4	44.4	44.4

Figure 8: Effect of Indian female skin color in system performance.









































































	LeftEye	RightEye	10	20	30	40	50	60	70	80
1			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
2			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
3			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
4			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
5			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
6			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
7			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
8			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
9			CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
10			CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
11			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
12			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
13			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
14			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
15			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
16			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
17			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
18			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
19			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
20			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
21			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
22			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
23			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
24			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
25			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
26			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
27			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
28			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
29			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
30			OPEN	OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE
31			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
32			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
33			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
34			CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
35			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
36			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
Success %			63.9	91.6	100	97.2	94.4	44.4	44.4	44.4

Figure 9: Effect of white male skin color in system performance.

As the three examples of experiments show, skin color is not an important element to affect real time driver safety system. The AOI algorithm in this system has most optimized technique with at least 85 percent accuracy if it is given by the proper threshold.

4.1.5 Error on the System

The following figure shows that the AOI algorithm cannot find the drowsy eyes in some cases.









































































	LeftEye	RightEye	10	20	30	40	50	60	70	80
1			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
2			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
3			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
4			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
5			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
6			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
7			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
8			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
9			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
10			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
11			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
12			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
13			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
14			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
15			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
16			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
17			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
18			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
19			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
20			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
21			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
22			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
23			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
24			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
25			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
26			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
27			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
28			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
29			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
30			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
31			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
32			OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
33			OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
34			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
35			OPEN	OPEN	OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE
36			OPEN	OPEN	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE	CLOSE
Sucess %										

Figure 10. Error on the system.

In Figure 10, the AOI algorithm has a problem finding drowsy eyes. Until the 15th frame, the system worked properly. However, from the next frame, this system cannot determine not only the best threshold but also the drowsy eyes. The reason is the Face SDK. The SDK should return the center of eyes area, but it returns the wrong position of the eyes depending on a person who cannot easily match the face template.

When we observed the results in evaluating a sample against its population, the RTDSS uses Type I error, also known as a false positive: the error of rejecting a null hypothesis (open eyes counted as closed) when it is actually true. From the above results, the errors are representing “Close” when a subject opens his or her eyes.

The false positive rate of the threshold value 40, which returns the highest accuracy of detecting the drowsy eyes, in Figure 8, is 15 percents by using the formula below.

$$\begin{aligned} \text{false positive rate} &= \frac{\text{number of false positives}}{\text{total number of negative instances}} \\ &= \frac{\text{number of false positives (close)}}{\text{total number of opened eyes}} = \frac{3}{20} = 0.15 \end{aligned}$$

Chapter 5: Conclusion and Future Work

5.1 Conclusion

In this thesis, we have focused on finding a better approach to real time eye tracing and detection. Web-cam resolution was the first challenge and several experiments were performed to find out the minimum resolution to handle with web-cam programming. To guarantee good quality, the minimum web-cam resolution selected was 800x600 with 15fps. Such web-cams are affordable and inexpensive.

Three different software development kits were tested and several algorithms were implemented using those SDK's. One SDK was free and open source code (OpenCV) and two are commercial (VeriLook, Luxand Face). VeriLook was selected because of its ease of use, it accepts different input formats, and provides a higher performance computation than the other.

There are tons of face detection algorithms and techniques developed and a feature based approach was the most promised one for our work. The scientific pervious work in the area recommend to use texture based face detection technique to extract other features from the face. This means that SDK helps to find a face and some features but other feature extraction such as eye or nose or mouth needs to be done by a developer.

In this work, a novel approach to extract the eye area from the rectangular face area is developed using the empirical results from several experiments were achieved through over

40 subjects. The ratio between length and width of rectangular face from Verilook SDK was the main idea behind the extraction eye area algorithm.

Pixel Different Algorithms for detecting drowsy eyes did not perform well and did not achieve reasonable accuracy. A novel approach for eye detection and feature extraction is developed called Area of Interest (AOI) algorithm. AOI algorithm achieved high accuracy of detecting drowsy eye, at least 85%, and it is up to 99% depending on a tester and the surrounded environment. It also is not affected by skin color, light, and obstacles such as glasses. In case of process speed, it is faster than other algorithms because AOI algorithm does not use any template matching or object classification technique.

There were almost 100 people tested with this system. During the test, we set several threshold values to decide proper value. The experiments showed that, the best threshold value occur between 20 and 40, in the closed indoor area and 70 and 90 outside door area.

5.2 Future Work

This system should be installed in the car for the driver's safety. One of future works is loading this system into the embedded system, which can be also cooperated with the field of automobile mechanics. So, it can finally slow the car or alert to the driver when he or she is sleeping.

This system must have proper threshold value to find the driver's drowsy eyes. There are two ways to find the proper threshold value in the future. One solution is using a light

sensor to get the current LUX value. The other way is calculating the threshold value with previous frames.

5.2.1 Light Sensor

In this thesis, there is an assumption that this system pre-installed the light sensor so that it will return proper threshold value. The threshold value will be high if the driver drives a car in the sunny day. To determine the proper threshold value automatically, this system must be installed with a light sensor.



Figure 1: LEGO MINDSTORMS light sensor.

The LEGO MINDSTORMS light sensor detects ambient light that is better to get the LUX value of surroundings. Frank Angeli[13] describes the LEGO light sensor below.

“There is a common operation on LEGO light sensors that involves removal of the LED. The belief is; If the LED is removed, then the sensor works better as an ambient light sensor. I have simulated the LEGO light sensor circuit with and without the LED to quantify the effect. The plot below shows the Light reading that would be made by the RCX over a wide range of ambient light levels. Notice that with the LED the Light reading never goes to 0. Not because the LED shines on the phototransistor, but simply the way the

LED biases the circuit. When the LED is removed it takes a little more light to make the sensor start to read anything but 0, but then it operates over the same light level range as before. Conclusion: Removal of the LED creates a sensor with more resolution (0 to 100) vs (20 to 100) while slightly losing low light level sensitivity and losing the ability to use it as a reflective sensor.”

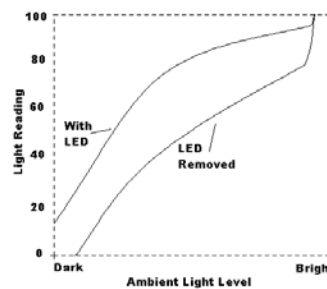


Figure 2: Ambient Light Level vs. Light Reading.

The short term goal for this project is to add the light sensor so that it is not necessary to change threshold value manually. Finally, the long term goal is to build this system into the embedded system.

5.2.2 Pre-Calculation of Threshold

When a driver starts the engine and drives the car for a short time, this system will keep calculating best threshold value. Usually, a driver begins to sleep when he or she drives the car for a long time. This system will store the eye status of each threshold values in order to get its proper value. For instance in Figure 8, during 36 frames, if threshold value is 20, it

always returns “OPEN.” Moreover, it returns all “CLOSE” when threshold value is 60.

The median value of all “OPEN” and “CLOSE” can be a proper threshold value.

In this example,

$$\frac{\text{All Open threshold} + \text{All Close threshold}}{2} = \frac{20 + 60}{2} = 40$$

In the Figure 8 in Chapter 4, if threshold value is 40, the accuracy of this system is 91.7 percent, which is the highest value than other values.

Here is the scenario for future work. When a time system finds drowsy eyes, it will store the result with several threshold values. The system then decides the proper threshold value using the formula above. In the future, this system needs to be implemented using this scenario and tested with samples in real-time. If this scenario works, the system does not need the extra cost of light sensors. It will also save memory space for the sensor.

APPENDIX

1. Source Code of Pixel Different Algorithm (Object Pascal-Delphi)

```
function TMainForm.getPDA(img: TImage): double;
var
  i, j : integer;
  tmpcolor : TColor;
  r, g, b : word;
  tmpdouble1 : double;
  tmpint : integer;
begin
  tmpint := 0;
  tmpdouble1 := 0;
  for i := 0 to img.Width-1 do
  begin
    for j := 0 to img.Height -1 do
    begin
      inc(tmpint);
      tmpColor := img.Canvas.Pixels[i,j];
      r := GetRValue(tmpColor);
      g := GetGValue(tmpcolor);
      b := GetBValue(tmpcolor);
      tmpdouble1 := tmpdouble1 +((r+g+b) / 3);
    end;
  end;

  result := (tmpdouble1 / tmpint);
end;
```


2. Source code of Area of Interest (Object Pascal-Delphi)

```

procedure TMainForm.runAOI;
var
  x, y : Integer;
  posX1, posX2, posX3 : Integer;
  posY : Integer;
  I, J : Integer;
  tmpColor : TColor;
  r, g, b : word;
  diffL, diffR : integer;
  sumL1, sumR1, sumL2, sumR2, sumL3, sumR3 : Integer;
  numOfStep : integer;
begin
  posX2 := ImgLeftEye.Width div 2;
  posY := ImgLeftEye.Height div 2;
  posX1 := posX2 div 2;
  posX3 := posX1 + posX2;

  sumL1 := 0;
  sumR1 := 0;
  sumL2 := 0;
  sumR2 := 0;
  sumL3 := 0;
  sumR3 := 0;
  numOfStep := 0;
  for I := -5 to 5 do
  begin
    for J := -5 to 5 do
    begin
      inc(numOfStep);
      tmpColor := ImgLeftEye.Canvas.Pixels[posX1+I, posY+J];
      r := GetRValue(tmpColor);
      g := GetGValue(tmpColor);
      b := GetBValue(tmpColor);
      sumL1 := sumL1 + ((r+g+b) div 3);

      tmpColor := ImgRightEye.Canvas.Pixels[posX1+I, posY+J];
      r := GetRValue(tmpColor);
      g := GetGValue(tmpColor);
      b := GetBValue(tmpColor);
      sumR1 := sumR1 + ((r+g+b) div 3);

      tmpColor := ImgLeftEye.Canvas.Pixels[posX2+I, posY+J];
      r := GetRValue(tmpColor);
      g := GetGValue(tmpColor);
      b := GetBValue(tmpColor);
      sumL2 := sumL2 + ((r+g+b) div 3);

      tmpColor := ImgRightEye.Canvas.Pixels[posX2+I, posY+J];
      r := GetRValue(tmpColor);
      g := GetGValue(tmpColor);
      b := GetBValue(tmpColor);
      sumR2 := sumR2 + ((r+g+b) div 3);
    end
  end
end

```

```

    tmpColor := ImgLeftEye.Canvas.Pixels[posX3+I, posY+J];
    r := GetRValue(tmpColor);
    g := GetGValue(tmpColor);
    b := GetBValue(tmpColor);
    sumL3 := sumL3 + ((r+g+b) div 3);

    tmpColor := ImgRightEye.Canvas.Pixels[posX3+I, posY+J];
    r := GetRValue(tmpColor);
    g := GetGValue(tmpColor);
    b := GetBValue(tmpColor);
    sumR3 := sumR3 + ((r+g+b) div 3);
  end;
end;

sumL1 := sumL1 div numOfStep;
sumL2 := sumL2 div numOfStep;
sumL3 := sumL3 div numOfStep;
sumR1 := sumR1 div numOfStep;
sumR2 := sumR2 div numOfStep;
sumR3 := sumR3 div numOfStep;

diffL := Max(Max(sumL1, sumL2), sumL3) - Min(Min(sumL1, sumL2), sumL3);
diffR := Max(Max(sumR1, sumR2), sumR3) - Min(Min(sumR1, sumR2), sumR3);

if cbDetectDrowsy.Checked then
begin
  if (diffL < StrToInt(spThreshold.Text)) and (diffR < StrToInt(spThreshold.Text)) then
  begin
    if cbBeep.Checked then Beep;
    eyeStatus.Color := clRed;
    eyeStatus.Caption := 'DROWSY EYES';
  end
  else
  begin
    eyeStatus.Color := clBlue;
    eyeStatus.Caption := ' ';
  end;
end;

Chart1.Series[0].Add(diffL, '', clRed);
Chart2.Series[0].Add(diffR, '', clBlue);

ImgLeftEye.Canvas.Rectangle(PosX1-1, PosY-1, posX1+1, posY+1);
ImgLeftEye.Canvas.Rectangle(PosX2-1, PosY-1, posX2+1, posY+1);
ImgLeftEye.Canvas.Rectangle(PosX3-1, PosY-1, posX3+1, posY+1);
ImgRightEye.Canvas.Rectangle(PosX1-1, PosY-1, posX1+1, posY+1);
ImgRightEye.Canvas.Rectangle(PosX2-1, PosY-1, posX2+1, posY+1);
ImgRightEye.Canvas.Rectangle(PosX3-1, PosY-1, posX3+1, posY+1);
end;

```

REFERENCES

- [1] Brian C. Lovell, Shaokang Chen, & Ting Shan (2009). Real-Time Face Detection and Classification for ICCTV. *Security and Surveillance Research Group, The University of Queensland*.
- [2] Video for Windows (2009). MSDN or Micro Soft Developer Network. *Microsoft Corporation*, [http://msdn.microsoft.com/en-us/library/ms713492\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms713492(VS.85).aspx).
- [3] FUJIFILM Global | FinePix Z200fd : Features - Face Detection with New Features. (2009).
http://www.fujifilmusa.com/products/digital_cameras/z/finepix_z200fd/features/page_04.html.
- [4] Gizmodo, the Gadget Guide – Sony Smile Shutter Images References. (2007).
gizmodo.com/assets/resources/2007/09/OMRON_1.jpg.
- [5] Toyota Enhances Pre-crash Safety System with Eye Monitor. (2008).
<http://www.theautochannel.com/news/2008/01/23/075668.html>.
- [6] Oliver jesorsky, Klaus J. Kirchberg, & Robert W. Frischholz (2001). Robust Face Detection Using the Hausdorff Distance. *Springer, Lecture Notes in Computer Science*, pp 90-95.
- [7] Erik Hjelm & Boon Kee Low (2001). Face Detection: A Survey. *Computer Vision and Image Understanding Volume 83*, pp 236-274.
- [8] Paul Kuo & John Hannah (2005). An Improved Eye Feature Extraction Algorithm Based on Deformable Templates. *Institute for Digital Communications, University of Edingurgh*.

- [9] A. L. Yuille, P. W. Hallinan, & D. S. Cohen (1992), Feature Extraction from Faces using Deformable Templates, *International Journal of Computer Vision*, vol. 8, no. 2, pp. 99–111.
- [10] Jun Wang, Lijun Yin, & Jason Moore (2007), Using Geometric Properties of Topographic Manifold to Detect and Track Eyes for Human-Computer Interaction, *ACM Trans. Multimedia Comput. Comm. Appl.* 3, 4, Article 21, pp.20.
- [11] Xuetao Feng, Yangsheng Wang, & Bai Li (2006), A Fast Eye Location Method Using Ordinal Features, *ACM* 1-59593-380-8.
- [12] Philipp Michel, & Rana El Kaliouby (2003), Real Time Facial Expression Recognition in Video using Support Vector Machines, *ACM* 1-58113-621-8.
- [13] Frank Angeli (2009), LEGO Light Sensor, <http://www.extremenxt.com/light.htm>.
- [14] Gady Agam (2006), Introduction to Programming with OpenCV, *Depart of Computer Science, Illinois Institute of Technology*.