CHAPTER THREE

ANALYSIS AND DESIGN

3.1 Introduction

A face Detector has to tell whether an image of arbitrary size contains a human face and if so, where it is. Face detection can be performed based on several cues: skin color (for faces in color images and videos, motion (for faces in videos), facial/head shape, facial appearance or a combination of these parameters. Most face detection algorithms are appearance based without using other cues.

An input image is scanned at all possible locations and scales by a sub window. Face detection is posed as classifying the pattern in the sub window either as a face or a non-face. The face/non-face classifier is learned from face and non-face training examples using statistical learning methods. Most modern algorithms are based on the Viola Jones object detection framework, which is based on Haar Cascades.

3.2 Viola jones object detection framework

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001[3]. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

Here we will work with face detection. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, Haar features shown in the below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle.

Now, all possible sizes and locations of each kernel are used to calculate lots of features. (Just imagine how much computation it needs? Even a 24x24 window results over 160000 features). For each feature calculation, we need to find the sum of the pixels under white and black

rectangles. To solve this, they introduced the integral image. However large your image, it reduces the calculations for a given pixel to an operation involving just four pixels. Nice, isn't it? It makes things super-fast.

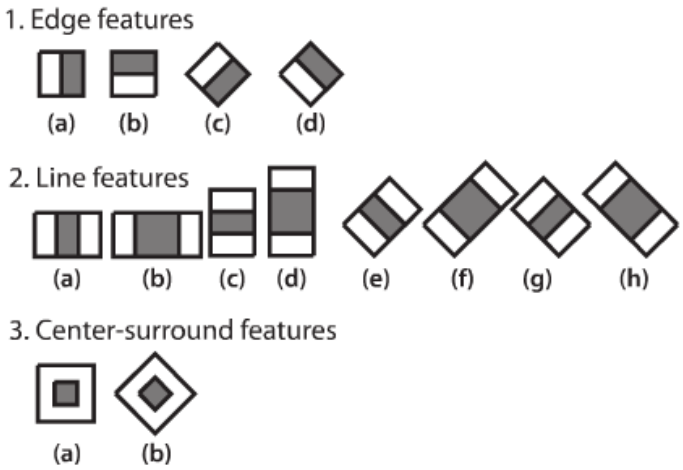

Figure 3.1: Haar-like features with different sizes and orientation. [2]

But among all these features we calculated, most of them are irrelevant. For example, consider the image below. The top row shows two good features. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. But the same windows applied to cheeks or any other place is irrelevant.

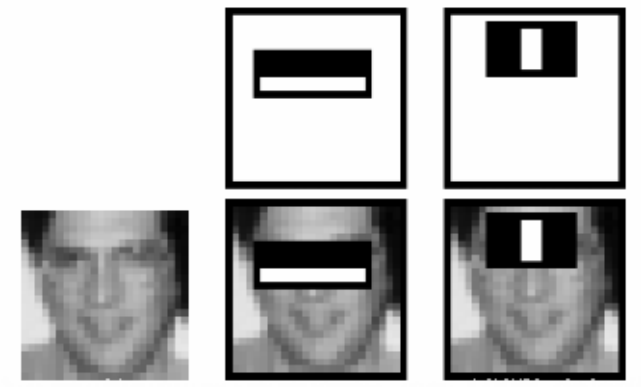

Figure 3.2: How the Haar-like feature of figure 3.1 can be used to scale the eyes. [2]

For this, we apply each and every feature on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. Obviously, there will be errors or misclassifications. We select the features with minimum error rate, which means they are the

features that most accurately classify the face and non-face images. (The process is not as simple as this. Each image is given an equal weight in the beginning. After each classification, weights of misclassified images are increased. Then the same process is done. New error rates are calculated. Also new weights. The process is continued until the required accuracy or error rate is achieved or the required number of features are found).

The final classifier is a weighted sum of these weak classifiers. It is called weak because it alone can't classify the image, but together with others forms a strong classifier. The paper says even 200 features provide detection with 95% accuracy. Their final setup had around 6000 features. (Imagine a reduction from 160000+ features to 6000 features).

So now you take an image. Take each 24x24 window. Apply 6000 features to it. Check if it is face or not. The authors have a good solution for that. In an image, most of the image is non-face region. So it is a better idea to have a simple method to check if a window is not a face region. If it is not, discard it in a single shot, and don't process it again. Instead, focus on regions where there can be a face. This way, we spend more time checking possible face regions.

For this they introduced the concept of Cascade of Classifiers. Instead of applying all 6000 features on a window, the features are grouped into different stages of classifiers and applied one-by-one. (Normally the first few stages will contain very many fewer features). If a window fails the first stage, discard it. We don't consider the remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region. How is that plan!

The authors' detector had 6000+ features with 38 stages with 1, 10, 25, 25 and 50 features in the first five stages. (The two features in the above image are actually obtained as the best two features from Adaboost). According to the authors, on average 10 features out of 6000+ are evaluated per sub-window. The characteristics of Viola–Jones algorithm which make it a good detection algorithm are:

- Robust – very high detection rate (true-positive rate) & very low false-positive rate always.
- Real time – For practical applications at least 2 frames per second must be processed.
- Face detection only (not recognition) - The goal is to distinguish faces from non-faces (detection is the first step in the recognition process).

3.2.1 Haar-like features

All human faces share some similar properties. These regularities may be matched using Haar Features. A few properties common to human faces:

- The eye region is darker than the upper-cheeks.
- The nose bridge region is brighter than the eyes.

Composition of properties forming match able facial features:

- Location and size: eyes, mouth, bridge of nose
- Value: oriented gradients of pixel intensities

The four features matched by this algorithm are then sought in the image of a face (shown at right). Rectangle features:

- Value = Σ (pixels in black area) - Σ (pixels in white area)
- Three types: two-, three-, four-rectangles, Viola & Jones used two-rectangle features
- For example: the difference in brightness between the white &black rectangles over a specific area
- Each feature is related to a special location in the sub-window

3.2.2 Cascaded classifier

From figure 3-3, A 1 feature classifier achieves 100% face detection rate and about 50% false positive rate. A 5 feature classifier achieves 100% detection rate and 40% false positive rate (20% cumulative). A 20 feature classifier achieves 100% detection rate with 10% false positive rate (2% cumulative). Combining several weak classifiers improves the accuracy of detection.

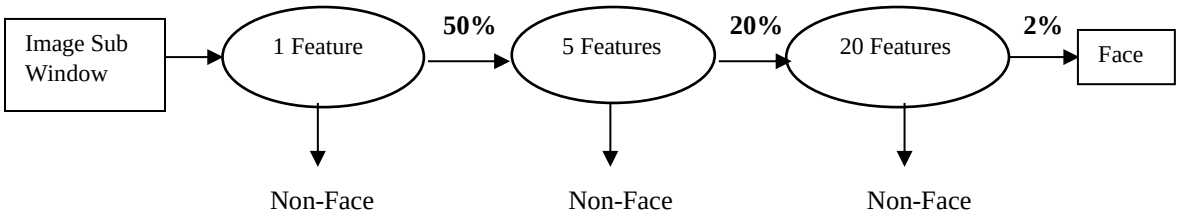

Figure 3.3: Several classifiers combined to enhance face detection.

A training algorithm called Adaboost, short for adaptive boosting, which had no application before Haar cascades, was utilized to combine a series of weak classifiers in to a strong classifier. Adaboost tries out multiple weak classifiers over several rounds, selecting the best weak classifier in each round and combining the best weak classifier to create a strong classifier. Adaboost can use classifiers that are consistently wrong by reversing their decision. In the design and development, it can take weeks of processing time to determine the final cascade sequence.

3.3 Process of face detection

As can be assumed, detecting a face is simpler than recognizing a face of a specific person. In order to be able to determine that a certain picture contains a face (or several) we need to be able to define the general structure of a face. Luckily human faces do not greatly differ from each other; we all have noses, eyes, foreheads, chins and mouths; and all of these compose the general structure of a face. Consider the following 5 figures:

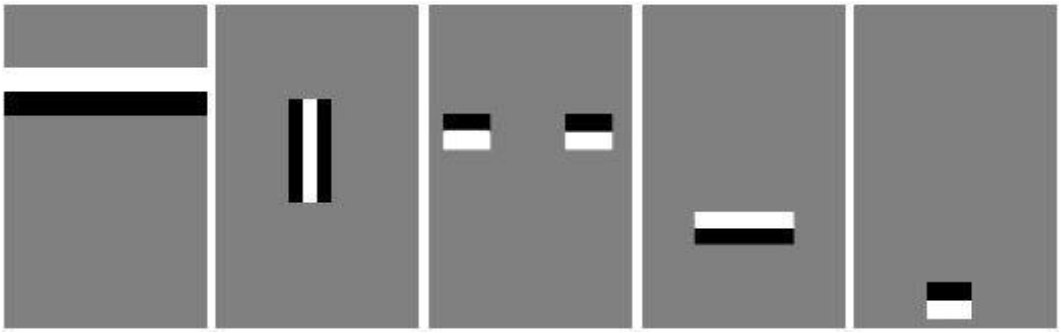

Figure 3.4: Several figures represents a general feature of a human face.

Each of these figures represents a general feature of a human face. Combining all the features together we, indeed, receive something that resembles a face.

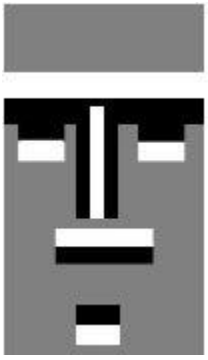

Figure 3.5: Combine figures represents a general feature of a human face.

By determining if each of these features is similar to some part of our picture, we can conclude if the picture contains a face or not. Notice that this does not have to be an accurate match; we just need to know if, roughly, each of these features corresponds to some part of the image. The technique used for this purpose is Template Matching.

Let's see an example. See in the figures below how the above features can be used to detect a face (namely, the face of President Barack Obama).

Figure 3.6: Combine features used to detect a face. [9]

In order for this process be quick, we design it in such a way that we first check the coarse features which represent the coarse structure of a face; and only if these features match, we continue to the next iteration and use finer features. In each such iteration we can quickly reject areas of the picture which do not match a face, and keep checking those which we are not sure about. In every iteration we increase the certainty that the checked area is indeed a face, until finally we stop and make our determination.

3.4 Tools

3.4.1. Python

Python is an open source programming language that was made to be easy-to-read and powerful. A Dutch programmer named Guido van Rossum made Python in 1991. He named it after the television show Monty Python's Flying Circus. Many Python examples and tutorials include jokes from the show. Python is a great object-oriented, interpreted, and interactive programming language.

Python is an interpreted language. Interpreted languages do not need to be compiled to run. A program called an interpreter runs Python code on almost any kind of computer. This means that a programmer can change the code and quickly see the results. This also means Python is slower than a compiled language like C, because it is not running machine code directly.

3.4.2. OpenCV

OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage and is now maintained by Itseez.

OpenCV is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

It has C, C++, python, Java, MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a template interface that works seamlessly with STL containers [10].

3.5 System specification

3.5.1. Hardware specification

This system has been run and tested on following configuration. In order to get better performance, recommended to use high resolution camera.

Processor	Inter Core i5 2.50 GHz
Motherboard	HP
RAM	4 GB
Hard Disk	1 TB
Monitor	HP Monitor
Graphic Card	2 GB
Camera	HP Webcam

Table 3.1: Hardware specification.

3.5.2. Software specification

This system is developed by cross-platform component E.g. Python and OpenCV. It means that face detection system can be operated in windows-platform, mac system, Linux-based system. We used those below software tools. Python is used to logic implementation. OpenCV is used to take and manipulate the image. Other python package is used to calculate many operation. Numpy is used to manipulate with number. Pynput is used to manipulate the mouse. And wxpython is used to get window maximum size.

- Python
- OpenCV
- Numpy package
- Pynput package
- wxpython package

3.6 General overview/Flowchart

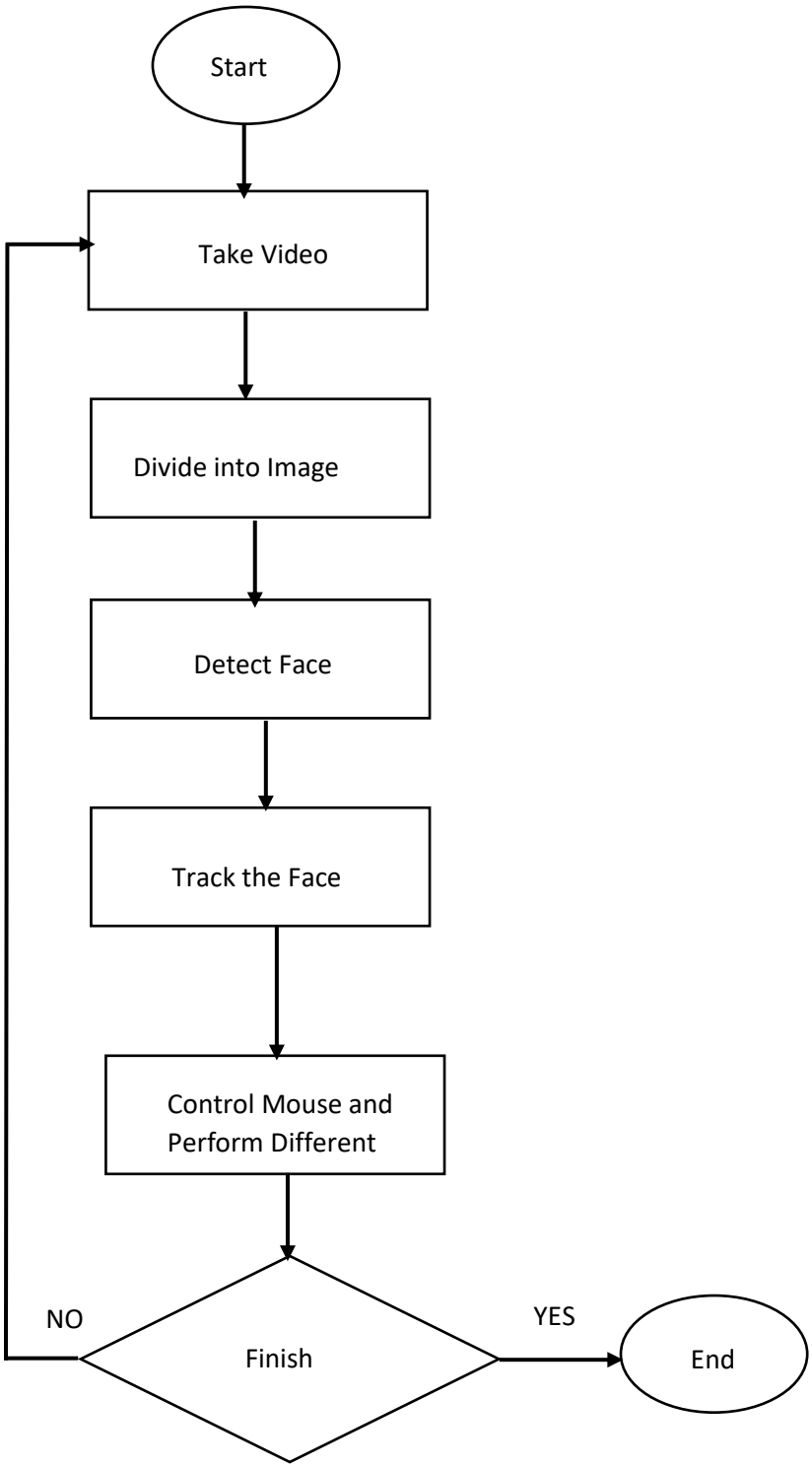

Figure 3.7: Flowchart of the system.

3.7 The proposed procedure

Our System work on those procedure or step:

Step 1. Take video from web camera.
Step 2. Divide this video into image frame.
Step 3. Detect face on this image using haar Cascaded feature.
Step 4. Track the face into different image frame.
Step 5. Control mouse position and movement with this face co-ordinate.
Step 6. Perform different operation (like left click, right click, double click).

3.8 Chapter summary

This chapter is about our system analysis and design. We broadly discuss about our system development process. Firstly de explain the Viola Jones face detection method. This include the haar like feature and cascade classifier. Then we explain overall face detection method. We discuss about our tools, hardware and software specification. Later we explain our whole process in step by step with proper algorithm. At the end of the chapter we draw a flowchart for our developed system.