A PROJECT ON

“Real-Time Face Recognition Attendence System And HandTracking And Gesture Control”

BY

AKASH SURENDRA BEHARA

Seat No: TCS20011

UNDER GUIDANCE OF

HoD. Mr. JIVAN ROTHE

IN PARTIAL FULFILLMENT OF

B.Sc.(COMPUTER SCIENCE)

DEGREE OF

UNIVERSITY OF MUMBAI

2020-2021

DEPARTMENT OF COMPUTER SCIENCE

S.I.C.E.S COLLEGE OF ARTS, SCIENCE AND COMMERCE, AMBERNATH

S.I.C.E.S COLLEGE OF ARTS,SCIENCE AND COMMERCE

AMBERNATH

CERTIFICATE

This is to certify that Mr. Akash Surendra Behara has satisfactorily completed the project work on **REAL TIME FACE RECOGNITION ATTENDENCE SYSTEM and HANDTRACKING GESTURE CONTROL** for partial fulfillment of the 3 year full time cource Bachelor of computer science of university of mumbai for the year 2019-2020.

Place:

Date:

_____ _____ _____

Head of department **Signature of External** **Sub in-charger**

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

They were the sources of continuous encouragement as such milestone was crossed. I like to extend our gratitude to HoD Mr.Jivan Rothe of S.I.C.E.S college Ambernath, who extended moral support, conductive work environment and the much needed inspiration to conclude this project in time.

I take this opportunity to thank **Prof. Mrs. Vaishali Khachane**, Head of the department and all professors of Department and all professors of the Department Computer Science of S.I.C.S College Ambernath, for giving us an opportunity to study in the Institute and the must needed guidance throughout the duration of the course.

Prof. Mrs. Ashwini Shirsat provided the guidance and necessary support during each phase of the project.

Thanking You.

ABSTRACT

The face is one of the easiest ways to distinguish the individual identity of each other. Face recognition is a personal identification system that uses personal characteristics of a person to identify the person's identity. Human face recognition procedure basically consists of two phases, namely face detection, where this process takes place very rapidly in humans, except under conditions where the object is located at a short distance away, the next is the introduction, which recognize a face as individuals. Stage is then replicated and developed as a model for facial image recognition (face recognition) is one of the much-studied biometrics technology and developed by experts. There are two kinds of methods that are currently popular in developed face recognition pattern namely, Eigenface method and Fisherface method. Facial image recognition Eigenface method is based on the reduction of face-dimensional space using Principal Component Analysis (PCA) for facial features. The main purpose of the use of PCA on face recognition using Eigen faces was formed (face space) by finding the eigenvector corresponding to the largest eigenvalue of the face image. The area of this project face detection system with face recognition is Image processing. The software requirements for this project is matlab software.
Keywords: face detection, Eigen face, PCA, matlab
Extension: There are vast number of applications from this face detection project, this project can be extended that the various parts in the face can be detect which are in various directions and shapes.

INDEX

INTRODUCTION

Face recognition is the task of identifying an already detected object as a known or unknown face. Often the problem of face recognition is confused with the problem of face detection Face Recognition on the other hand is to decide if the "face" is someone known, or unknown, using for this purpose a database of faces in order to validate this input face.

Computer vision and Digital Image Processing

The sense of sight is arguably the most important of man's five senses. It provides a huge amount of information about the world that is rich in detail and delivered at the speed of light. However, human vision is not without its limitations, both physical and psychological. Through digital imaging technology and computers, man has transcending many visual limitations. He can see intofar galaxies, the microscopic world, the sub-atomic world, and even “observe” infra-red, x-ray, ultraviolet and other spectra for medical diagnosis, meteorology,

surveillance, and military uses, allwith great success.

While computers have been central to this success, for the most part man is the sole interpreter of all the digital data. For a long time, the central question has been whether computers can be designed to analyze and acquire information from images autonomously in the same natural way humans can. According to Gonzales and Woods [2], this is the province of computer vision, whichis that branch of artificial intelligence that ultimately aims to "use computers to emulate human vision, including learning and being able to make inferences and tak[ing] actions based on visual inputs."

The main difficulty for computer vision as a relatively young discipline is the current lack of a finalscientific paradigm or model for human intelligence and human vision itself on which to build a infrastructure for computer or machine learning [3]. The use of images has an obvious drawback. Humans perceive the world in 3D, but current visual sensors like cameras capture the world in 2D images. The result is the natural loss of a good deal of information in the captured images. Withouta proper paradigm to explain the mystery of human vision and perception,

the recovery of lost information (reconstruction of the world) from 2D images represents a difficult hurdle for machine vision [4]. However, despite this limitation, computer vision has progressed, riding mainly on the remarkable advancement of decades-old digital image processing techniques, using the science and methods contributed by other disciplines such as optics, neurobiology, psychology, physics, mathematics, electronics, computer science, artificial intelligence and others.

Computer vision techniques and digital image processing methods both draw the proverbial water
from the same pool, which is the digital image, and therefore necessarily overlap. Image processing takes a digital image and subjects it to processes, such as noise reduction, detail enhancement, or filtering, for the purpose of producing another desired image as the end result. For example, the blurred image of a car registration plate might be enhanced by imaging techniques to produce a clear photo of the same so the police might identify the owner of the car. On the other hand, computer vision takes a digital image and subjects it to the same digital imaging techniques but for the purpose of analyzing and understanding what the image depicts. For example, the image of a building can be fed to a computer and thereafter be identified by the computer

as a residential house, a stadium, high-rise office tower, shopping mall, or a farm barn. [5]

Russell and Norvig [6] identified three broad approaches used in computer vision to distill useful information from the raw data provided by images. The first is the feature extraction approach, which focuses on simple computations applied directly to digital images to measure some useable characteristic, such as size. This relies on generally known image processing algorithms for noise reduction, filtering, object detection, edge detection, texture analysis, computation of optical flow, and segmentation, which techniques are commonly used to pre-process images for subsequent image analysis. This is also considered an "uninformed" approach.

The second is the recognition approach, where the focus is on distinguishing and labelling objects based on knowledge of characteristics that sets of similar objects have in common, such as shape or appearance or patterns of elements, sufficient to form classes. Here computer vision uses the techniques of artificial intelligence in knowledge representation to enable a "classifier" to match classes to objects based on the pattern of

their features or structural descriptions. A classifier has to "learn" the patterns by being fed a training set of objects and their classes and achieving the goal ofminimizing mistakes and maximizing successes through a step-by-step process of improvement.

There are many techniques in artificial intelligence that can be used for object or pattern recognition, including statistical pattern recognition, neural nets, genetic algorithms and fuzzy systems.

The third is the reconstruction approach, where the focus is on building a geometric model of the world suggested by the image or images and which is used as a basis for action. This corresponds tothe stage of image understanding, which represents the highest and most complex level of computervision processing. Here the emphasis is on enabling the computer vision system to construct internal models based on the data supplied by the images and to discard or update these internalmodels as they are verified against the real world or some other criteria. If the internal model is consistent with the real world, then image understanding takes place. Thus, image understandingrequires the construction, manipulation and control of models and at the

moment relies heavily upon the science and technology of artificial intelligence.

OpenCV

OpenCv is a widely used tool in computer vision. It is a computer vision library for real-time applications, written in C and C++, which works with the Windows, Linux and Mac platforms. It is freely available as open source software from http://sourceforge.net/projects/opencvlibrary/.

OpenCv was started by Gary Bradsky at Intel in 1999 to encourage computer vision research and commercial applications and, side-by-side with these, promote the use of ever faster processors from Intel [7]. OpenCV contains optimised code for a basic computer vision infrastructure so developers do not have to re-invent the proverbial wheel. The reference documentation for OpenCVis at http://opencv.willowgarage.com/documentation/index.html. The basic tutorial documentation isprovided by Bradsky and Kaehler

[6]. According to its website, OpenCV has been downloaded more than two million times and has a user group of more than 40,000 members. This attests to its popularity.

A digital image is generally understood as a discrete number of light intensities captured by a devicesuch as a camera and organized into a two-dimentional matrix of picture elements or pixels, each of which may be represented by number and all of which may be stored in a particular file format (such as jpg or gif) [8]. OpenCV goes beyond representing an image as an array of pixels. It represents an image as a data structure called an IplImagethat makes immediately accessible useful image data or fields, such as:

- width– an integer showing the width of the image in pixels
- height– an integer showing the height of the image in pixels
- imageData– a pointer to an array of pixel values
- nChannels– an integer showing the number of colors per pixel
- depth– an integer showing the number of bits per pixel
- widthStep– an integer showing the number of bytes per image row
- imageSize– an integer showing the size of in bytes
- roi– a pointer to a structure that defines a region of interest within the image [9].

OpenCV has a module containing basic image processing and computer vision algorithms. These include:

- smoothing (blurring) functions to reduce noise,
- dilation and erosion functions for isolation of individual elements,
- floodfill functions to isolate certain portions of the image for further processing,
- filter functions, including Sobel, Laplace and Canny for edge detection,
- Hough transform functions for finding lines and circles,
- Affine transform functions to stretch, shrink, warp and rotate images,
- Integral image function for summing subregions (computing Haar wavelets),
- Histogram equalization function for uniform distribution of intensity values,
- Contour functions to connect edges into curves,
- Bounding boxes, circles and ellipses,
- Moments functions to compute Hu's moment invariants,
- Optical flow functions (Lucas-Kanade method),
- Motion tracking functions (Kalman filters), and
- Face detection/ Haar classifier.

OpenCV also has an ML (machine learning) module containing well known statistical classifiers and clustering tools. These include:

- Normal/ naïve Bayes classifier,

- Decision trees classifier,
- Boosting group of classifiers,
- Neural networks algorithm, and
- Support vector machine classifier.

Pattern Recognition and Classifiers

In computer vision a physical object maps to a particular segmented region in the image from which object descriptors or features may be derived. A *feature* is any characteristic of an image, or any region within it, that can be measured. Objects with common features may be grouped into classes, where the combination of features may be considered a *pattern*. Object recognition may be understood to be the assignment of classes to objects based on their respective patterns. The program that does this assignment is called a *classifier*. [10]

The general steps in pattern recognition may be summarized in Figure 1 below:

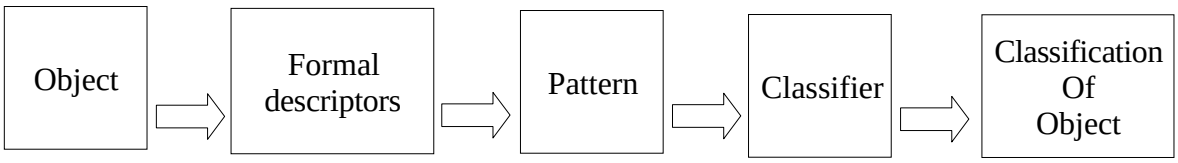


Figure 1. General pattern recognition steps. [3]

The most important step is the design of the formal descriptors because choices have to be made on which characteristics, quantitative or qualitative, would best suit the target object and in turn determines the success of the classifier.

In statistical pattern recognition, quantitative descriptions called features are used. The set of features constitutes the pattern vector or feature vector, and the set of all possible patterns for the object form the *pattern space X* (also known as *feature space*). Quantitatively, similar objects in each class will be located near each other in the feature space forming clusters, which may ideally be separated from dissimilar objects by lines or curves called *discrimination functions*. Determining the most suitable discrimination function or *discriminant* to use is part of classifier

design.

A statistical classifier accepts *n* features as inputs and gives 1 output, which is the classification or decision about the class of the object. The relationship between the inputs and the output is a *decision rule*, which is a function that puts in one space or subset those feature vectors that are associated with a particular output. The decision rule is based on the particular discrimination function used for separating the subsets from each other.

The ability of a classifier to classify objects based on its decision rule may be understood as *classifier learning*, and the set of the feature vectors (objects) inputs and corresponding outputs of classifications (both positive and negative results) is called the *training set.* It is expected that awell-designed classifier should get 100% correct answers on its training set. A large training set is generally desirable to optimize the training of the classifier, so that it may be tested on objects it has not encountered before, which constitutes its *test set*. If the classifier does not perform well on the test set, modifications to the design of the recognition system may be needed.

1.1 FACE RECOGNIZATION:

DIFFERENT APPROACHES OF FACE RECOGNITION:

There are two predominant approaches to the face recognition problem: Geometric (feature based) and photometric (view based). As researcher interest in face recognition continued, many different algorithms were developed, three of which have been well studied in face recognition literature.

Recognition algorithms can be divided into two main approaches:

1. **Geometric:** Is based on geometrical relationship between facial landmarks, or in other words the spatial configuration of facial features. That means that the main geometrical features of the face such as the eyes, nose and mouth are first located and then faces are classified on the basis of various geometrical distances and angles between features.

2. **Photometric stereo:** Used to recover the shape of an object from a number of images taken under different lighting conditions. The shape of the recovered object is defined by a gradient map, which is made up of an array of surface normals (Zhao and Chellappa, 2006)

Popular recognition algorithms include:

1. Principal Component Analysis using Eigenfaces, (PCA)

2. Linear Discriminate Analysis,

3. Elastic Bunch Graph Matching using the Fisherface algorithm,

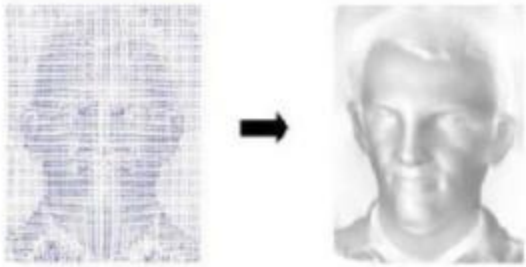


Figure 2 -Photometric stereo image.

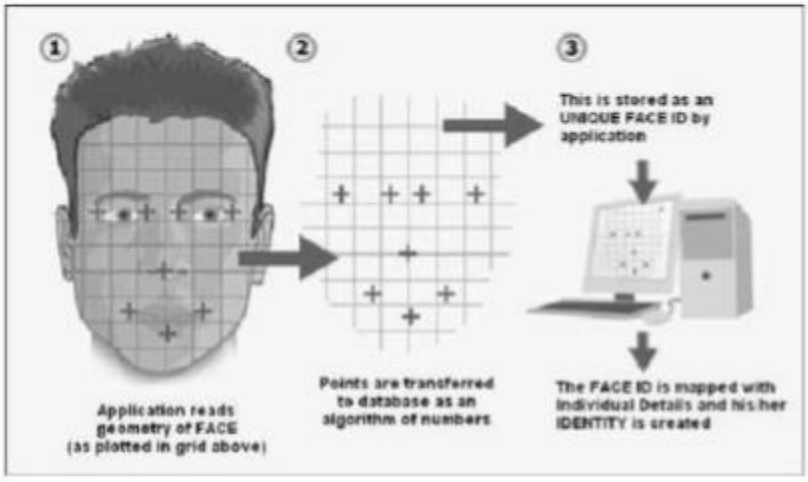


Figure 3 - Geometric facial recognition.

1.2 FACE DETECTION:

Face detection involves separating image windows into two classes; one containing faces (tarning the background (clutter). It is difficult because although commonalities exist between faces, they can vary considerably in terms of age, skin colour and facial expression. The problem is further complicated by differing lighting conditions, image qualities and geometries, as well as the possibility of partial occlusion and disguise. An ideal face detector would therefore be able to detect the presence of any face under any set of lighting conditions, upon any background. The face detection task can be broken down into two steps. The first step is a classification task that takes some arbitrary image as input and outputs a binary value of yes or no, indicating whether there are any faces present in the image. The second step is the face localization task that aims to take an image as input and output the location of any face or faces within that image as some bounding box with (x, y, width, height).

The face detection system can be divided into the following steps:-

1. Pre-Processing: To reduce the variability in the faces, the images are processed before they are fed into the network. All positive examples that is the face images are obtained by cropping
images with frontal faces to include only the front view. All the cropped images are then corrected for lighting through standard algorithms.
2. Classification: Neural networks are implemented to classify the images as faces or nonfaces by training on these examples. We use both our

implementation of the neural network and the Matlab neural network toolbox for this task. Different network configurations are experimented with to optimize the results.

3. Localization: The trained neural network is then used to search for faces in an image and if present localize them in a bounding box. Various Feature of Face on which the work has done on:- Position Scale Orientation Illumination

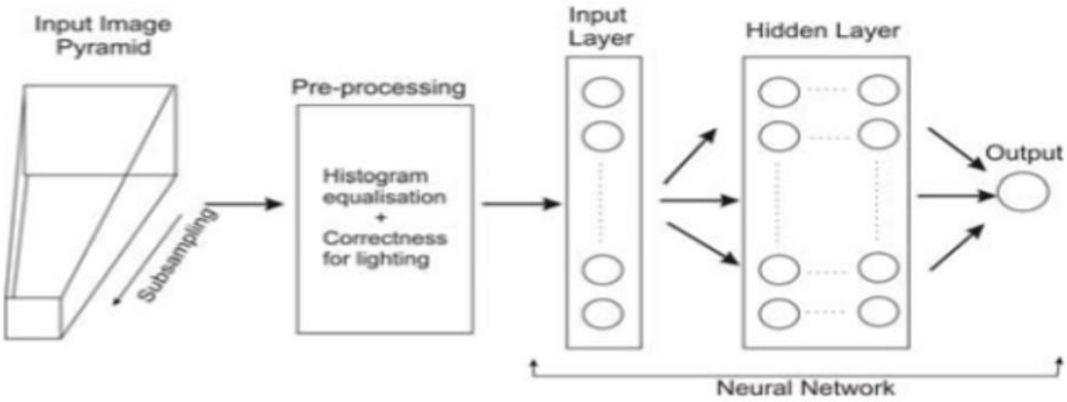


Fig: Face detection algorithm

LITERATURE SURVEY

Face detection is a computer technology that determines the location and size of human face in arbitrary (digital) image. The facial features are detected and any other objects like trees, buildings and bodies etc are ignored from the digital image. It can be regarded as a specific case of object-class detection, where the task is finding the location and sizes of all objects in an image that belong to a given class. Face detection, can be regarded as a more general case of face localization. In face localization, the task is to find the locations and sizes of a known number of faces (usually one). Basically there are two types of approaches to detect facial part in the given image i.e. feature base and image base approach. Feature base approach tries to extract features of the image and match it against the knowledge of the face features. While image base approach tries to get best match between training and testing images.

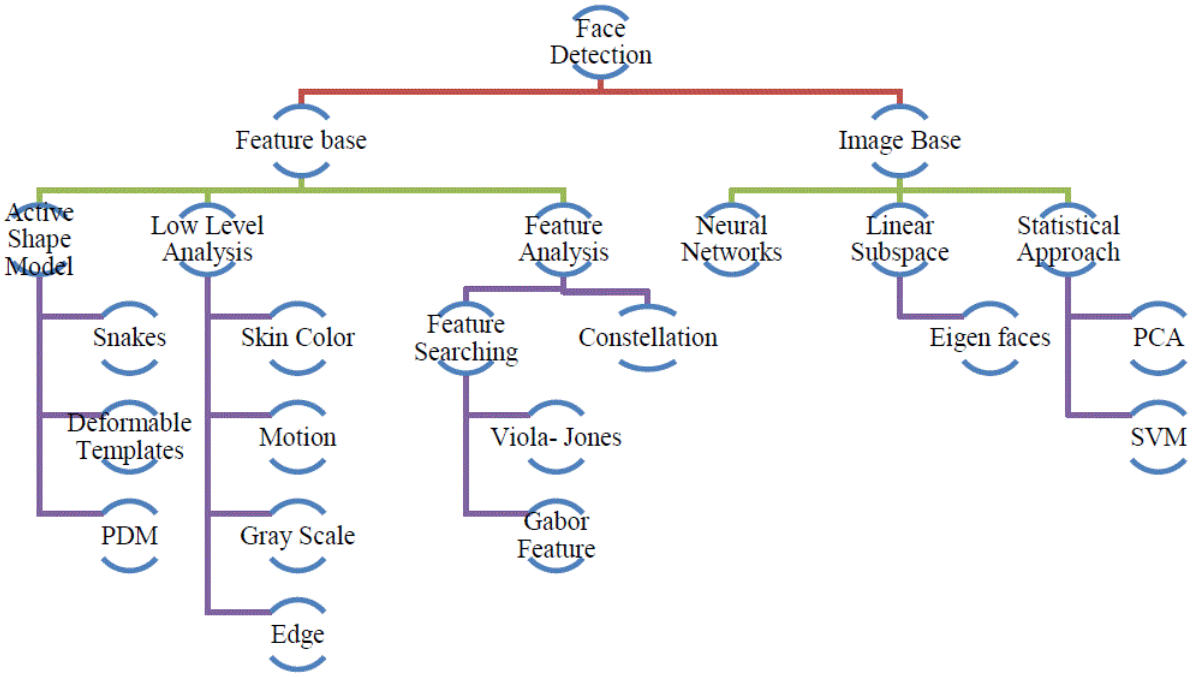


Fig: detection methods

2.1 FEATURE BASE APPROCH:

Active Shape ModelActive shape models focus on complex non-rigid features like actual physical and higher level appearance of features Means that Active Shape Models (ASMs) are aimed at automatically locating landmark points that define the shape of any statistically modeled object in an image. When of facial features such as the eyes, lips, nose, mouth and eyebrows. The training stage of an ASM involves the building of a statistical.

a) facial model from a training set containing images with manually annotated landmarks. ASMs is classified into three groups i.e. snakes, PDM, Deformable templates

b) Snakes:The first type uses a generic active contour called snakes, first introduced by Kass et al. in 1987 Snakes are used to identify head boundaries [8,9,10,11,12]. In order to achieve the task, a snake is first initialized at the proximity around a head boundary. It then locks onto nearby edges and subsequently assume the shape of the head. The evolution of a snake is achieved by minimizing an energy function, Esnake (analogy

with physical systems), denoted as Esnake = Einternal + EExternal WhereEinternal and EExternal are internal and external energy functions. Internal energy is the part that depends on the intrinsic properties of the snake and defines its natural evolution. The typical natural evolution in snakes is shrinking or expanding. The external energy counteracts the internal energy and enables the contours to deviate from the natural evolution and eventually assume the shape of nearby features—the head boundary at a state of equilibria.Two main consideration for forming snakes i.e. selection of energy terms and energy minimization. Elastic energy is used commonly as internal energy is vary with the distance between control points on the snake, through which we get contour an elastic-band characteristic that causes it to shrink or expand. On other side external energy relay on image features. Energy minimization process is done by optimization techniques such as the steepest gradient descent. Which needs highest computations. Huang and Chen and Lam and Yan both employ fast iteration methods by greedy algorithms. Snakes have some demerits like contour often becomes trapped onto false image features and another one is that snakes are not suitable in extracting non convex features

2.1.1 Deformable Templates:

Deformable templates were then introduced by Yuille et al. to take into account the a priori of facial features and to better the performance of snakes. Locating a facial feature boundary is not an easy task because the local evidence of facial edges is difficult to organize into a sensible global entity using generic contours. The low brightness contrast around some of these features also makes the edge detection process.Yuille et al. took the concept of snakes a step further by incorporating global information of the eye to improve the reliability of the extraction processDeformable templates approaches are developed to solve this problem. Deformation is based on local valley, edge, peak, and brightness .Other than face boundary, salient feature (eyes, nose, mouth and eyebrows) extraction is a great challenge of face recognition.E = Ev + Ee + Ep + Ei + Einternal ; where Ev , Ee , Ep , Ei , Einternal are external energy due to valley, edges, peak and image brightness and internal energy

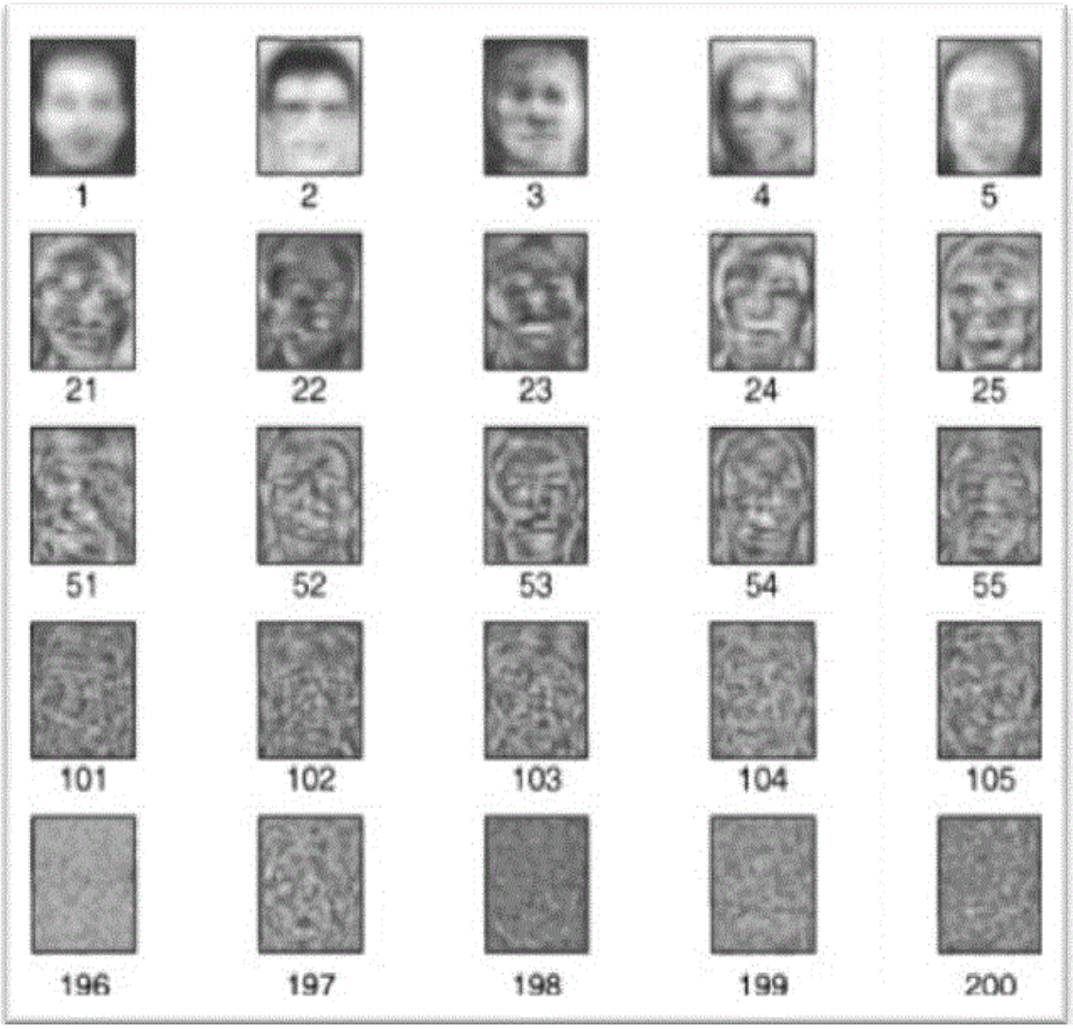
2.1.2 PDM (Point Distribution Model)**:**

Independently of computerized image analysis, and before ASMs were developed, researchersdeveloped statistical models of shape . The idea is that once you represent shapes asvectors, you can apply standard statistical methods to them just like any other multivariateobject. These models learn allowable constellations of shape points from training examplesand use principal components to build what is called a Point Distribution Model. These havebeen used in diverse ways, for example for categorizing Iron Age broaches.Ideal Point Distribution Models can only deform in ways that are characteristic of the object. Cootes and his colleagues were seeking models which do exactly that so if a beard, say, covers the chin, the shape model can \override the image" to approximate the position of the chin under the beard. It was therefore natural (but perhaps only in retrospect) to adopt Point Distribution Models. This synthesis of ideas from image processing and statistical shape modelling led to the Active Shape Model.The first parametric statistical shape model for image analysis based on principal components of inter-landmark distances was presented by Cootes and Taylor in. On this approach, Cootes, Taylor, and their colleagues, then released a series of papers that cumulated in what we call the classical Active Shape Model.

2.2) LOW LEVEL ANALYSIS:

Based on low level visual features like color, intensity, edges, motion etc. Skin Color BaseColor is avital feature of human faces. Using skin-color as a feature for tracking a face has several advantages. Color processing is much

faster than processing other facial features. Under certain lighting conditions, color is orientation invariant. This property makes motion estimation much easier because only a translation model is needed for motion estimation.Tracking human faces using color as a feature has several problems like the color representation of a face obtained by a camera is influenced by many factors (ambient light, object movement, etc



Majorly three different face detection algorithms are available based on RGB, YCbCr, and HIS color space models.In the implementation of the algorithms there are three main steps viz.
(1) Classify the skin region in the color space,
(2) Apply threshold to mask the skin region and
(3) Draw bounding box to extract the face image.

Crowley and Coutaz suggested simplest skin color algorithms for detecting skin pixels. The perceived human color varies as a function of the relative direction to the illumination.

The pixels for skin region can be detected using a normalized color histogram, and can be normalized for changes in intensity on dividing by luminance. Converted an [R, G, B] vector is converted into an [r, g] vector of normalized color which provides a fast meanskin detection. This algorithm fails when there are some more skin region like legs, arms, etc.Cahi and Ngan [27] suggested skin color classification algorithm with YCbCr color space.Research found that pixels belonging to skin region having similar Cb and Cr values. So that the thresholds be chosen as [Cr1, Cr2] and [Cb1, Cb2], a pixel is classified to have skin tone if the values [Cr, Cb] fall within the thresholds. The skin color distribution gives the face portion in the color image. This algorithm is also having the constraint that the image should be having only face as the skin region. Kjeldson and Kender defined a color predicatein HSV color space to separate skin regionsfrom background. Skin color classification inHSI color space is the same as YCbCr color spacebut here the responsible values are hue (H) andsaturation (S). Similar to above the threshold be chosen as [H1, S1] and [H2, S2], and a pixel isclassified to have skin tone if the values [H,S] fallwithin the threshold and this distribution gives thelocalized face image. Similar to above twoalgorithm this algorithm is also having the same constraint

2.3) MOTION BASE:

When useof video sequence is available, motion informationcan be used to locate moving objects. Movingsilhouettes like face and body parts can be extractedby simply thresholding accumulated framedifferences . Besides face regions, facial featurescan be located by frame differences .

2.3.1 Gray Scale Base:

Gray information within a face canalso be treat as important features. Facial features such as eyebrows, pupils, and lips appear generallydarker than their surrounding facial regions. Various recent feature extraction algorithms searchfor local gray minima within segmented facial regions. In these algorithms, the input images are first enhanced by contrast-stretching and gray-scale morphological routines to improve the quality of local dark patches and thereby make detection easier. The extraction of darkpatches is achieved by low-level gray-scale thresholding. Based method and consist three levels. Yang and huang presented new approach i.e. faces gray scale behaviour in pyramid (mosaic) images. This system utilizes hierarchical Face location consist three levels. Higher two level based on mosaic images at different resolution. In the lower level, edge detection method is proposed. Moreover this algorithms gives fine response in complex background where size of the face is unknown

2.3.2 Edge Base:

Face detection based on edges was introduced by Sakai et al. This workwas based on analysing line drawings of the faces from photographs, aiming to locate facialfeatures. Than later Craw et al. proposed a hierarchical framework based on Sakai et al.'swork to trace a human head outline. Then after remarkable works were carried out by many researchers in this specific area. Method suggested by Anila and Devarajan was very simple and fast. They proposed frame work which consist three steps i.e. initially the images are enhanced by applying median filter for noise removal and histogram

equalization for contrast adjustment. In the second step the edge images constructed from the enhanced image by applying sobel operator. Then a novel edge tracking algorithm is applied to extract the sub windows from the enhanced image based on edges. Further they used Back propagation Neural Network (BPN) algorithm to classify the sub-window as either face or non-face.

2.4 FEATURE ANALYSIS

These algorithms aimto find structural features that exist even when thepose, viewpoint, or lighting conditions vary, andthen use these to locate faces. These methods aredesigned mainly for face localization

2.4.1 Feature Searching
Viola Jones Method:

Paul Viola and Michael Jones presented an approach for object detection which minimizes computation time while achieving high detection accuracy. Paul Viola and Michael Jones [39] proposed a fast and robust method for face detection which is 15 times quicker than any technique at the time of release with 95% accuracy at around 17 fps.The technique relies on the use of simple Haar-like features that are evaluated quickly through the use of a new image representation. Based on the concept of an —Integral Image‖ it generates a large set of features and uses the boosting algorithm AdaBoost to reduce the overcomplete set and the introduction of a degenerative tree of the boosted classifiers provides for robust and fast interferences. The detector is applied in a scanning fashion and used on gray-scale images, the

scanned window that is applied can also be scaled, as well as the features evaluated.

Gabor Feature Method:

Sharif et al proposed an Elastic Bunch Graph Map (EBGM) algorithm that successfully implements face detection using Gabor filters. The proposed system applies 40 different Gabor filters on an image. As aresult of which 40 images with different angles and orientationare received. Next, maximum intensity points in each filtered image are calculated and mark them as fiducial points. The system reduces these points in accordance to distance between them. The next step is calculating the distances between the reduced points

using distance formula. At last, the distances arecompared with database. If match occurs, it means that thefaces in the image are detected. Equation of Gabor filter [40] is shown below

$$\psi_{u,v}(z) = \frac{\| k_{u,v} \|^2}{\sigma^2} e^{\left(\frac{\|k_{u,v}\|^2 \|z\|^2}{2\sigma^2} \right)} \left[e^{i\vec{k}_{u,v} z} - e^{-\frac{\sigma^2}{2}} \right]$$

Where

$\phi_u = \frac{u\pi}{8}$, $\phi_u \in [0, \pi)$ gives the frequency,

DIGITAL IMAGE PROCESSING

3.1 DIGITAL IMAGE PROCESSING

Interest in digital image processing methods stems from two principal application areas:

1. Improvement of pictorial information for human interpretation

2. Processing of scene data for autonomous machine perception

In this second application area, interest focuses on procedures for extracting image information in a form suitable for computer processing.
Examples includes automatic character recognition, industrial machine vision for product assembly and inspection, military recognizance, automatic processing of fingerprints etc.

Image:

Am image refers a 2D light intensity function f(x, y), where(x, y) denotes spatial coordinates and the value of f at any point (x, y) is proportional to the brightness or gray levels of the image at that point. A digital image is an image f(x, y) that has been discretized both in spatial coordinates and brightness. The elements of such a digital array are called image elements or pixels.

A simple image model:

To be suitable for computer processing, an image f(x, y) must be digitalized both spatially and in amplitude. Digitization of the spatial coordinates (x, y) is called image sampling. Amplitude digitization is called gray-level quantization.
The storage and processing requirements increase rapidly with the spatial resolution and the number of gray levels.
Example: A 256 gray-level image of size 256x256 occupies 64k bytes of memory.
Types of image processing

- • Low level processing
- • Medium level processing
- • High level processing
- Low level processing means performing basic qperations on images such as reading an image resize, resize, image rotate, RGB to gray level conversion, histogram equalization etc…, The output image obtained after low level processing is raw image.Medium level processing means extracting regions of interest from output of low level processed image. Medium level processing deals with identification of boundaries i.e edges .This process is called segmentation.High level processing deals with adding of artificial intelligence to medium level processed signal.

• 3.2 FUNDAMENTAL STEPS IN IMAGE PROCESSING

- **Fundamental steps in image processing are**
- 1. Image acquisition: to acquire a digital image
- 2. Image pre-processing: to improve the image in ways that increases the chances for success of the other processes.
- 3. Image segmentation: to partitions an input image into its constituent parts of objects.
- 4. Image segmentation: to convert the input data to a from suitable for computer processing.
- 5. Image description: to extract the features that result in some quantitative information of interest of features that are basic for differentiating one class of objects from another.

- 6. Image recognition: to assign a label to an object based on the information provided by its description.

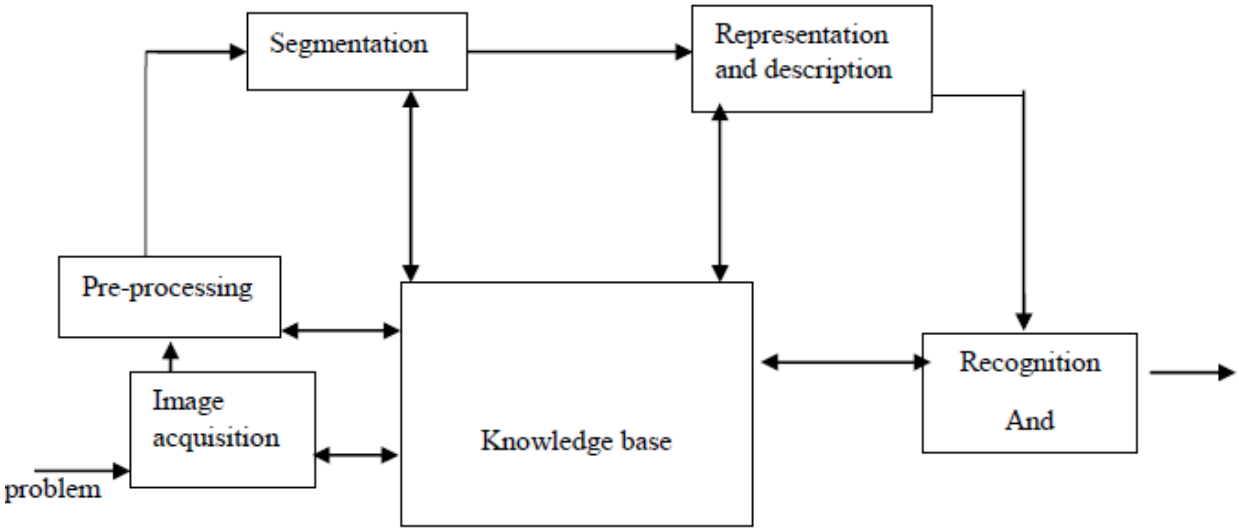


fig.3.1. Fundamental steps in digital image processing

3.3 ELEMENTS OF DIGITAL IMAGE PROCESSING SYSTEMS

A digital image processing system contains the following blocks as shown in the figure

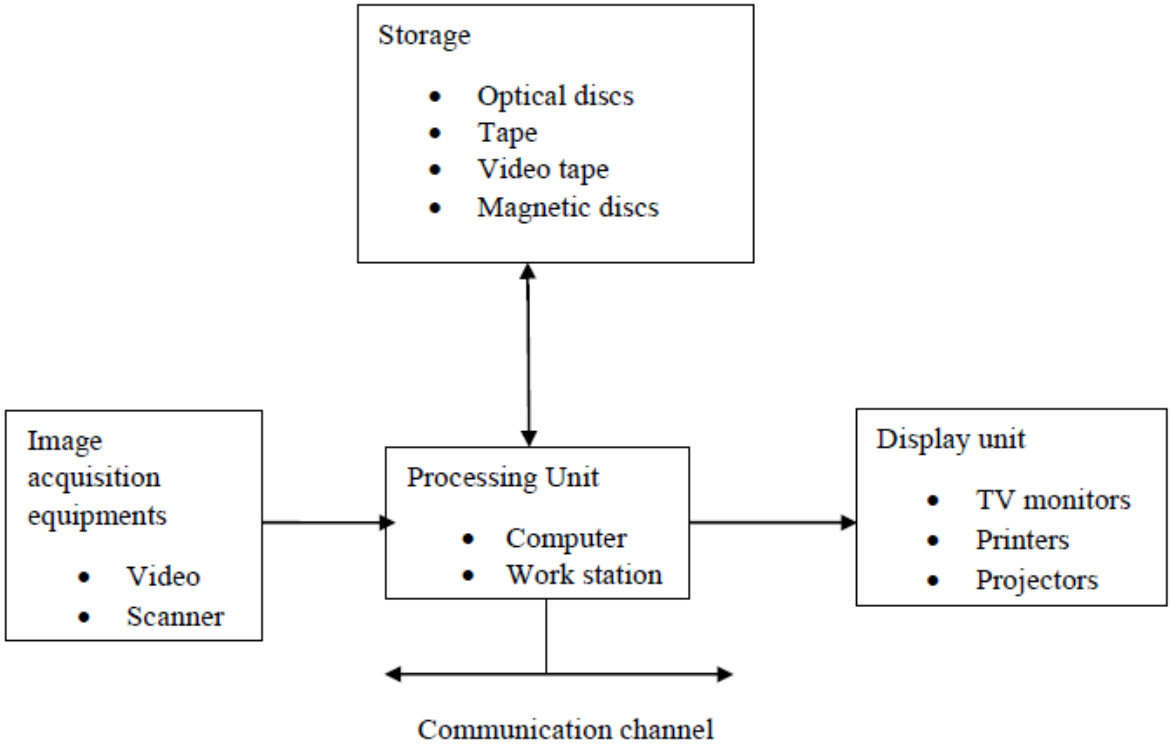


Fig.3.3. Elements of digital image processing systems

The basic operations performed in a digital image processing system include

1. Acquisition
2. Storage
3. Processing
4. Communication
5. Display

3.3.1 A simple image formation model

Image are denoted by two-dimensional function f(x, y).f(x, y) may be characterized by 2 components:

1. The amount of source illumination i(x, y) incident on the scene

2. The amount of illumination reflected r(x, y) by the objects of the scene

3. f(x, y) = i(x, y)r(x, y), where 0 < i(x,y) < and 0 < r(x, y) < 1

Typical values of reflectance r(x, y):

- 0.01 for black velvet
- 0.65 for stainless steel
- 0.8 for flat white wall paint
- 0.9 for silver-plated metal

• 0.93 for snow Example of typical ranges of illumination i(x, y) for visible light (average values)

- • Sun on a clear day: ~90,000 lm/m^2,down to 10,000lm/m^2 on a cloudy day
- • Full moon on a clear evening :-0.1 lm/m^2
- • Typical illumination level in a commercial office. ~1000lm/m^2

FACE DETECTION

The problem of face recognition is all about face detection. This is a fact that seems quite bizarre to new researchers in this area. However, before face recognition is possible, one must be able to reliably find a face and its landmarks. This is essentially a segmentation problem and in practical systems, most of the effort goes into solving this task. In fact the actual recognition based on features extracted from these facial landmarks is only a minor last step.

There are two types of face detection problems:
1) Face detection in images and

2) Real-time face detection

4.1 FACE DETECTION IN IMAGES

Most face detection systems attempt to extract a fraction of the whole face, thereby eliminating most of the background and other areas of an individual's head such as hair that are not necessary for the face recognition task. With static images, this is often done by running a across the image. The face detection system then judges if a face is present inside the window (Brunelli and Poggio, 1993). Unfortunately, with static images there is a very large search space of possible locations of a face in an image. Most face detection systems use an example based learning approach to decide whether or not a face is present in the *window* at that given instant (Sung and Poggio,1994 and Sung,1995). A neural network or some other classifier is trained using supervised learning with 'face' and 'non-face' examples, thereby enabling it to classify an image (*window* in face detection system) as a 'face' or 'non-face'.. Unfortunately, while it is relatively easy to find face examples, how would one find a representative sample of images which represent non-faces (Rowley et al., 1996)? Therefore, face detection systems using example based learning need thousands of 'face' and 'non-face' images for effective training. Rowley, Baluja, and Kanade (Rowley et al.,1996) used 1025 face images and 8000 non-face images (generated from 146,212,178 sub-images) for their training set!

There is another technique for determining whether there is a face inside the face detection system's *window* - using Template Matching. The difference between a fixed target pattern (face) and the window is computed and thresholded. If the window contains a pattern which is close to the target pattern(face) then the window is judged as containing a face. An implementation of template matching called Correlation Templates uses a whole bank of fixed sized templates to detect facial features in an image (Bichsel, 1991 & Brunelli and Poggio, 1993). By using several templates of different (fixed) sizes, faces of different scales (sizes) are detected. The other implementation of template matching is using a deformable template (Yuille, 1992). Instead of using several fixed size templates, we use a deformable template (which is non-rigid) and there by change the size of the template hoping to detect a face in an image.

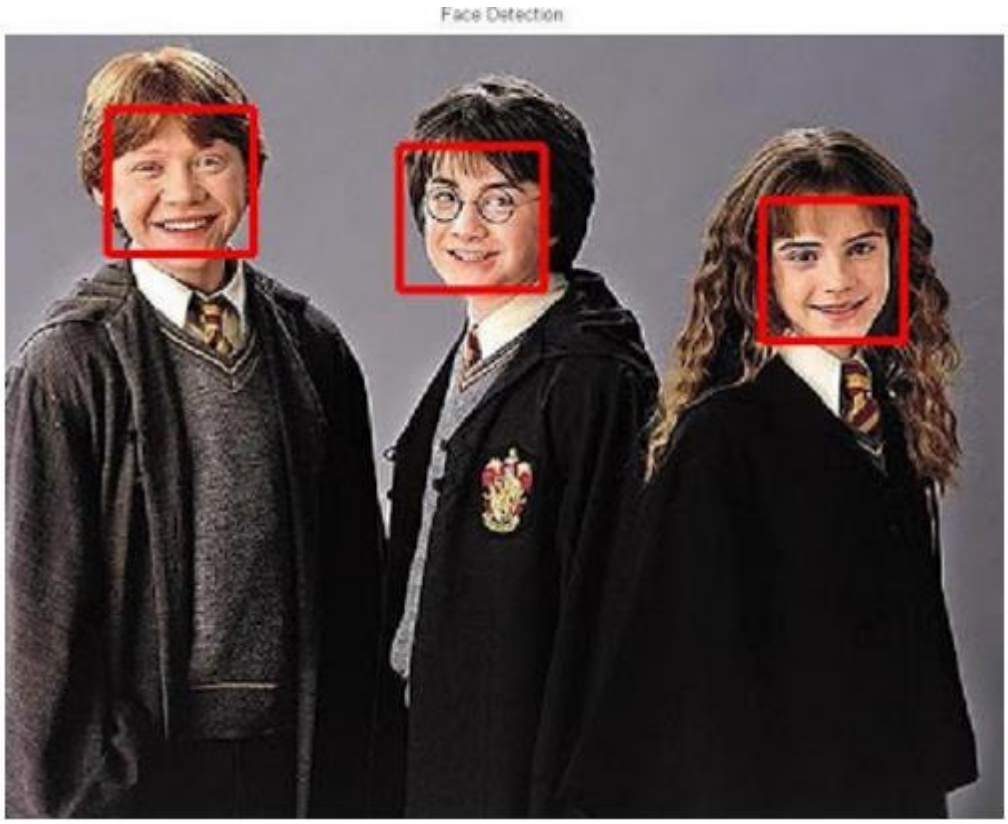
4.2 REAL-TIME FACE DETECTION

Real-time face detection involves detection of a face from a series of frames from a video-capturing device. While the hardware requirements for such a system are far more stringent, from a computer vision stand point, real-time face detection is actually a far simpler process thandetecting a face in a static image. This is because unlike most of our surrounding environment, people are continually moving. We walk around, blink, fidget, wave our hands about, etc.

Since in real-time face detection, the system is presented with a series of frames in which to detect a face, by using spatio-temperal filtering (finding the difference between subsequent frames), the area of the frame that has changed can be identified and the individual detected (Wang and Adelson, 1994 and Adelson and Bergen 1986).Further more as seen in Figure exact face locations can be easily identified by using a few simple rules, such as,

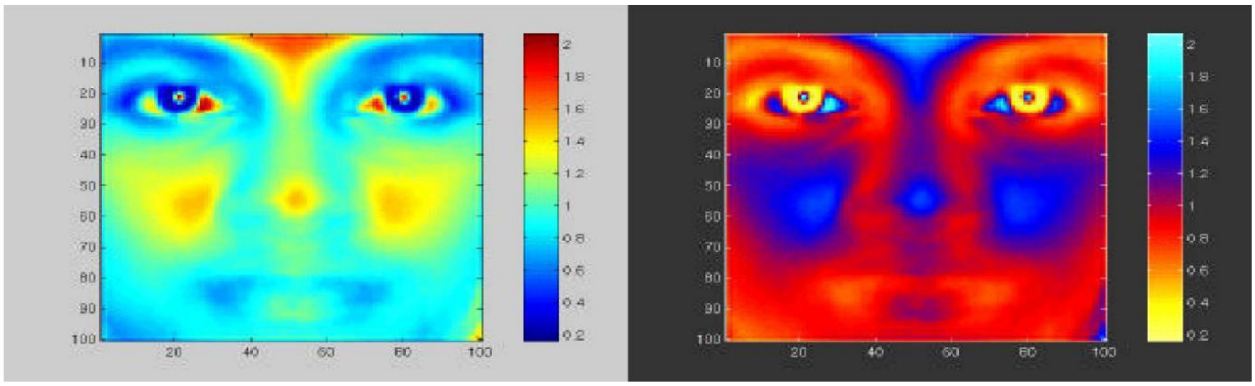
1)the head is the small blob above a larger blob -the body
2)head motion must be reasonably slow and contiguous -heads won't jump around erratically (Turk and Pentland 1991a, 1991b).
Real-time face detection has therefore become a relatively simple problem and is possible even in unstructured and uncontrolled environments using these very simple image processing techniques and reasoning rules.

4.3 FACE DETECTION PROCESS



It is process of identifying different parts of human faces like eyes, nose, mouth, etc… this process can be achieved by using MATLAB codeIn this

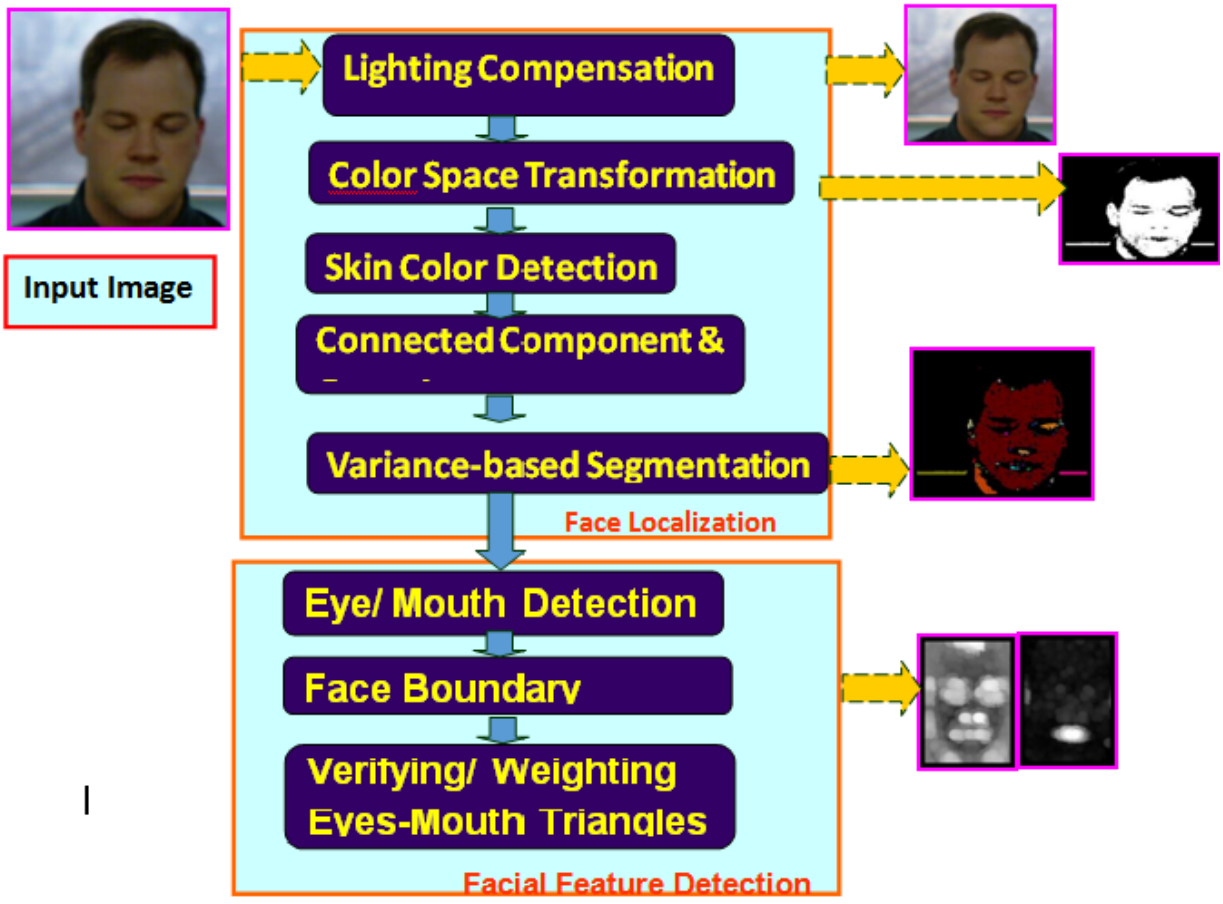
project the author will attempt to detect faces in still images by using image invariants. To do this it would be useful to study the grey-scale intensity distribution of an average human face. The following 'average human face' was constructed from a sample of 30 frontal view human faces, of which 12 were from females and 18 from males. A suitably scaled colormap has been used to highlight grey-scale intensity differences.

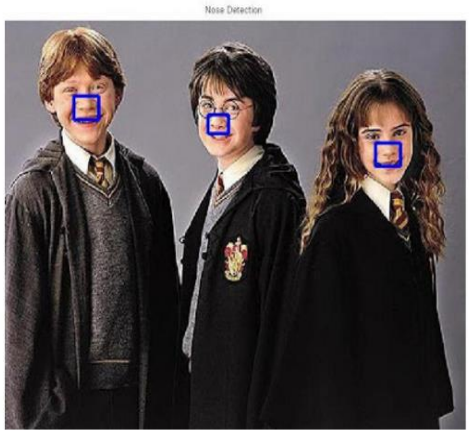


The grey-scale differences, which are invariant across all the sample faces are strikingly apparent. The eye-eyebrow area seem to always contain dark intensity (low) gray-levels while nose forehead and cheeks contain bright intensity (high) grey-levels. After a great deal of experimentation, the researcher found that the following areas of the human face were suitable for a face detection system based on image invariants and a deformable template. The above facial area performs well as a basis for a face template, probably because of the clear divisions of the bright intensity invariant area by the dark intensity invariant regions. Once this pixel area is located by the face detection system, any particular area required can be segmented based on the proportions of the average human face After studying the above images it was subjectively decided by the author to use the following as a basis for dark intensity sensitive and bright intensity sensitive

templates. Once these are located in a subject's face, a pixel area 33.3% (of the width of the square window) below this.

4.4 FACE DETECTION ALGORITHM





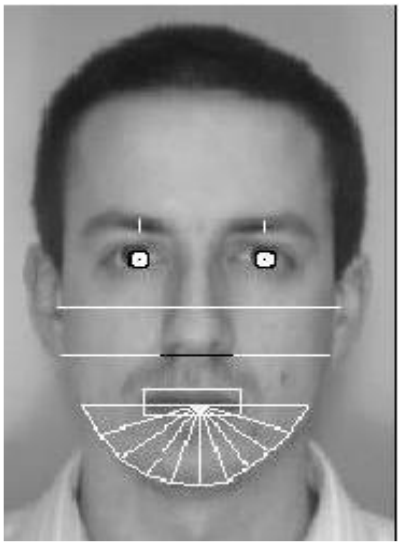
FACE RECOGNITION

Over the last few decades many techniques have been proposed for face recognition. Many of the techniques proposed during the early stages of computer vision cannot be considered successful, but almost all of the recent approaches to the face recognition problem have been creditable. According to the research by Brunelli and Poggio (1993) all approaches to human face recognition can be divided into two strategies:
(1) Geometrical features and (2) Template matching.

5.1 FACE RECOGNITION USING GEOMETRICAL FEATURES

This technique involves computation of a set of geometrical features such as nose width and length, mouth position and chin shape, etc. from the

picture of the face we want to recognize. This set of features is then matched with the features of known individuals. A suitable metric such as Euclidean distance (finding the closest vector) can be used to find the closest match. Most pioneering work in face recognition was done using geometric features (Kanade, 1973), although Craw et al. (1987) did relatively recent work in this area.



The advantage of using geometrical features as a basis for face recognition is that recognition is possible even at very low resolutions and with noisy images (images with many disorderly pixel intensities). Although the face cannot be viewed in detail its overall geometrical configuration can be extracted for face recognition. The technique's main disadvantage is that automated extraction of the facial geometrical features is very hard. Automated geometrical feature extraction based recognition is also very sensitive to the scaling and rotation of a face in the image plane (Brunelli and Poggio, 1993). This is apparent when we examine Kanade's(1973)

results where he reported a recognition rate of between 45-75 % with a database of only 20 people. However if these features are extracted manually as in Goldstein et al. (1971), and Kaya and Kobayashi (1972) satisfactory results may be obtained.

5.1.1 Face recognition using template matching

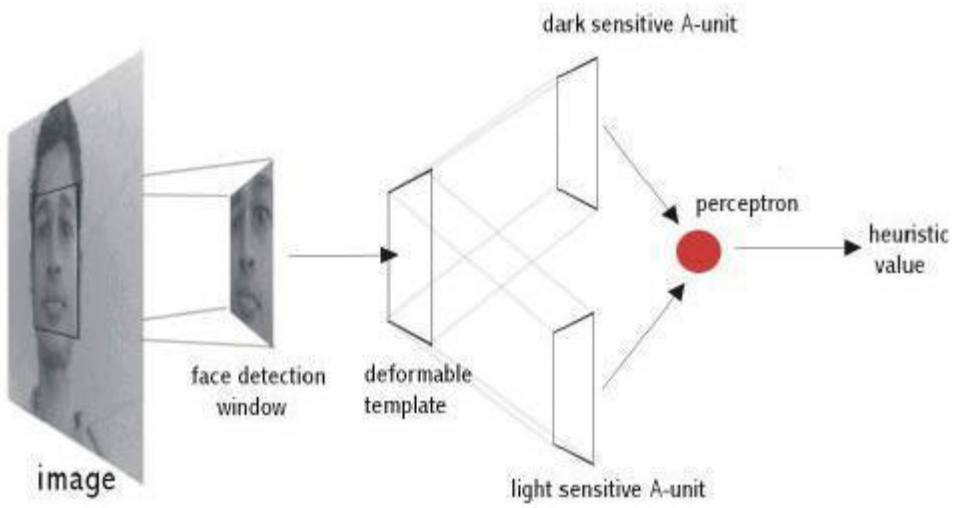
This is similar the template matching technique used in face detection, except here we are not trying to classify an image as a 'face' or 'non-face' but are trying to recognize a face.

Whole face, eyes, nose and mouth regions which could be used in a template matching strategy.The basis of the template matching strategy is to extract whole facial regions (matrix of pixels) and compare these with the stored images of known individuals. Once again Euclidean distance can be used to find the closest match. The simple technique of comparing grey-scale intensity values for face recognition was used by Baron (1981). However there are far more sophisticated methods of template matching for face recognition. These involve extensive pre-processing and transformation of the extracted grey-level intensity values. For example, Turk and Pentland (1991a) used Principal Component Analysis, sometimes known as the eigenfaces approach, to pre-process the gray-levels and Wiskott et al. (1997) used Elastic Graphs encoded using Gabor filters to pre-process the extracted regions. An investigation of geometrical features versus template matching for face recognition by Brunelli and Poggio (1993) came to the conclusion that although a feature based strategy may offer higher recognition speed and smaller memory requirements, template based techniques offer superior recognition accuracy.

5.3 BRIEF OUT LINE OF THE IMPLEMENTED SYSTEM

Fully automated face detection of frontal view faces is implemented using a deformable template algorithm relying on the image invariants of human faces. This was chosen because a similar neural-network based face detection model would have needed far too much training data to be

implemented and would have used a great deal of computing time. The main difficulties in implementing a deformable template based technique were the creation of the bright and dark intensity sensitive templates and designing an efficient implementation of the detection algorithm



A manual face detection system was realised by measuring the facial proportions of the average face, calculated from 30 test subjects. To detect a face, a human operator would identify the locations of the subject's eyes in an image and using the proportions of the average face, the system would segment an area from the image.

Face recognition and detection system is a pattern recognition approach for personal identification purposes in addition to other biometric approaches

such as fingerprint recognition, signature, retina and so forth. Face is the most common biometric used by humans applications ranges from static, mug-shot verification in a cluttered background.

5.4 FACE RECOGNITION DIFFICULTIES

1. Identify similar faces (inter-class similarity)
2. Accommodate intra-class variability due to
2.1 head pose
2.2 illumination conditions
2.3 expressions
2.4 facial accessories
2.5 aging effects

3. Cartoon faces

Inter - class similarity:

Different persons may have very similar appearance

Face recognition and detection system is a pattern recognition approach for personal identification purposes in addition to other biometric approaches such as fingerprint recognition, signature, retina and so forth. The variability in the faces, the images are processed before they are fed into the network. All positive examples that is the face images are obtained by cropping images with frontal faces to include only the front view. All the cropped images are then corrected for lighting through standard algorithms.

Inter – class variability

Faces with intra-subject variations in pose, illumination, expression, accessories, color, occlusions, and brightnes

PRINCIPAL COMPONENT ANALYSIS (PCA)

Principal Component Analysis (or Karhunen-Loeve expansion) is a suitable strategy for face recognition because it identifies variability between human faces, which may not be immediately obvious. Principal Component Analysis (hereafter PCA) does not attempt to categorise faces using familiar geometrical differences, such as nose length or eyebrow width. Instead, a set of human faces is analysed using PCA to determine which 'variables' account for the variance of faces. In face recognition, these variables are called eigen faces because when plotted they display an eerie resemblance to human faces. Although PCA is used extensively in statistical analysis, the pattern recognition community started to use PCA for classification only relatively recently. As described by Johnson and Wichern (1992), 'principal component analysis is concerned with explaining the variance- covariance structure through a few linear combinations of the original variables.'

Perhaps PCA's greatest strengths are in its ability for data reduction and interpretation. For example a 100x100 pixel area containing a face can be very accurately represented by just 40 eigen values. Each eigen value describes the magnitude of each eigen face in each image. Furthermore, all interpretation (i.e. recognition) operations can now be done using just the 40 eigen values to represent a face instead of the manipulating the 10000 values contained in a 100x100 image. Not only is this computationally less demanding but the fact that the recognition information of several thousand.

UNDERSTANDING EIGENFACES

Any grey scale face image I(x,y) consisting of a NxN array of intensity values may also be consider as a vector of N2. For example, a typical 100x100 image used in this thesis will have to be transformed into a 10000 dimension vector!

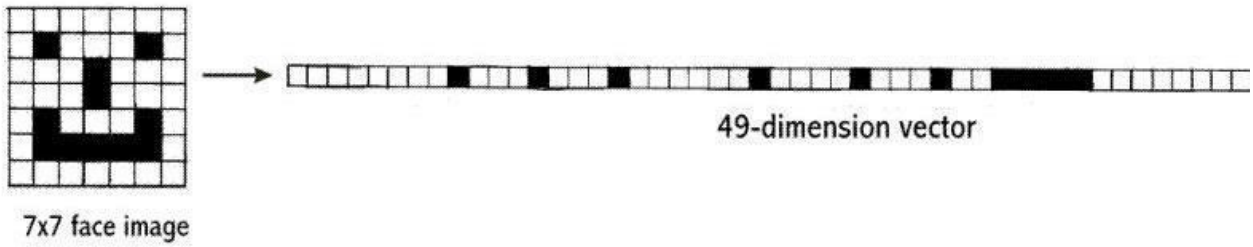
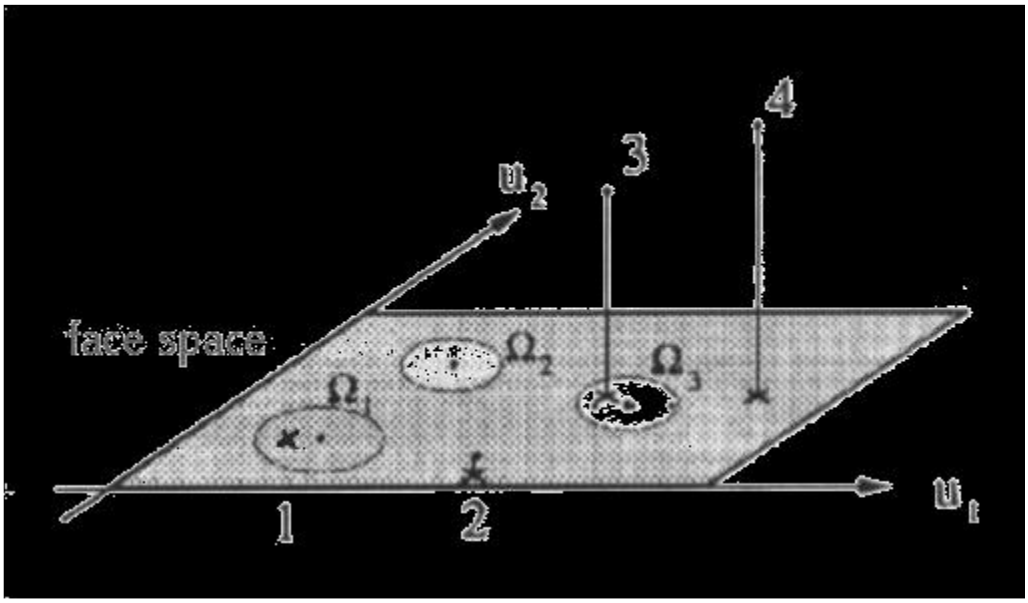
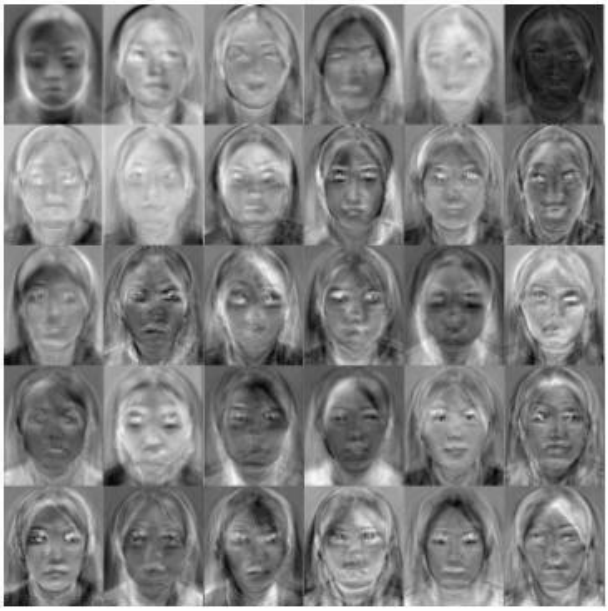


Figure 6.6.0 A 7x7 face image transformed into a 49 dimension vector

This vector can also be regarded as a point in 10000 dimension space. Therefore, all the images of subjects' whose faces are to be recognized can be regarded as points in 10000 dimension space. Face recognition using

these images is doomed to failure because all human face images are quite similar to one another so all associated vectors are very close to each other in the 10000-dimension space.





The transformation of a face from image space (I) to face space (f) involves just a simple matrix multiplication. If the average face image is A and U contains the (previously calculated) eigenfaces,
$f = U * (I - A)$

This is done to all the face images in the face database (database with known faces) and to the image (face of the subject) which must be recognized. The possible results when projecting a face into face space are given in the following figure.
There are four possibilities:
1. Projected image is a face and is transformed near a face in the face database
2.Projected image is a face and is not transformed near a face in the face database
3.Projected image is not a face and is transformed near a face in the face database

4. Projected image is not a face and is not transformed near a face in the face database
While it is possible to find the closest known face to the transformed image face by calculating the Euclidean distance to the other vectors, how does one know whether the image that is being transformed actually contains a face? Since PCA is a many-to-one transform, several vectors in the image space (images) will map to a point in face space (the problem is that even non-face images may transform near a known face image's faces space vector). Turk and Pentland (1991a), described a simple way of checking whether an image is actually of a face. This is by transforming an image into face space and then transforming it back (reconstructing) into image space. Using the previous notation,

$I' = U^T * U * (I - A)$

APPENDIX

FACE DETECTION:

```
import numpy as np
import cv2

# multiple cascades:
https://github.com/Itseez/opencv/tree/master/data/haarcascades

#https://github.com/Itseez/opencv/blob/master/data/haarcascades/haarcas
cade_frontalface_default.xml
face_cascade =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
#https://github.com/Itseez/opencv/blob/master/data/haarcascades/haarcas
cade_eye.xml
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')

cap = cv2.VideoCapture(0)

while 1:
    ret, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

    for (x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]

        eyes = eye_cascade.detectMultiScale(roi_gray)
        for (ex,ey,ew,eh) in eyes:
            cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)

    cv2.imshow('img',img)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

NOSE DETECTION:

```
%To detect Nose
NoseDetect =
vision.CascadeObjectDetector('Nose','MergeThreshold',16);
BB=step(NoseDetect,I);
figure,
imshow(I); hold on
for i = 1:size(BB,1)
```

```
rectangle('Position',BB(i,:),'LineWidth',4,'LineStyle','-
','EdgeColor','b');
end
title('Nose Detection');
hold off;
```

EXPLANATION:

To denote the object of interest as 'nose', the argument 'Nose' is passed.
vision.CascadeObjectDetector('Nose','MergeThreshold',16);
The default syntax for Nose detection :
vision.CascadeObjectDetector('Nose');
Based on the input image, we can modify the default values of the parameters passed to **vision.CascaseObjectDetector.** Here the default value for 'MergeThreshold' is 4.
When default value for 'MergeThreshold' is used, the result is not correct.
Here there are more than one detection on Hermione.
To avoid multiple detection around an object, the 'MergeThreshold' value can be overridden.

MOUTH DETECTION:

```
%To detect Mouth
MouthDetect =
vision.CascadeObjectDetector('Mouth','MergeThreshold',16);
BB=step(MouthDetect,I);
figure,
imshow(I); hold on
for i = 1:size(BB,1)
rectangle('Position',BB(i,:),'LineWidth',4,'LineStyle','-
','EdgeColor','r');
end
title('Mouth Detection');
hold off;
```

6. Introduction to HAAR Cascade

This project was done with this fantastic “Open Source Computer Vision Library”, the OpenCV. On this, we will be focusing on OpenCV and Python,

OpenCV was designed for computational efficiency and with a strong focus on real-time applications. So, it’s perfect for real-time face recognition using a camera.

A Haar wavelet is a mathematical fiction that produces square-shaped waves with a beginning and an end and used to create box shaped patterns to recognise signals with sudden transformations. By combining several wavelets, a cascade can be created that can identify edges, lines and circles with different colour intensities. These sets are used in Viola Jones face detection technique in 2001 and since then more patterns are introduced for object detection. To analyze an image using Haar cascades, a scale is selected smaller than the target image. It is then placed on the image, and the average of the values of pixels in each section is taken. If the difference between two values pass a given threshold, it is considered a match. Face detection on a human face is performed by matching a combination of different Haar-like-features. For example, forehead, eyebrows and eyes contrast as well as the nose with eyes. A single classifier is not accurate enough. Several classifiers are combined as to provide an accurate face detection system.

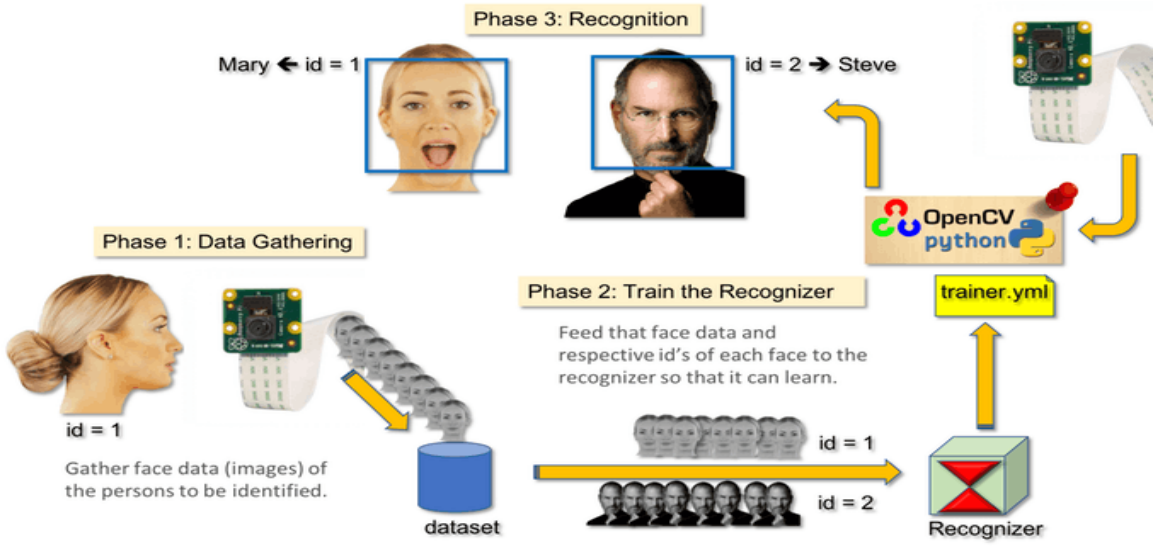
The 3 Phases

To create a complete project on Face Recognition, we must work on 3 very distinct phases:

- Face Detection and Data Gathering
- Train the Recognizer

- Face Recognition

The below block diagram resumes those phases:



6.1. Installing OpenCV Package

To use OpenCV, first you have to install the Opencv package and numpy package.

Open “cmd” and type the following code:

>>Pip install opencv-python

This will install the OpenCV package and also install numpy if your system doesn’t have it.

.

Once you finished, you should have an OpenCV virtual environment ready to run our experiments on your Pi.

Let's go to our virtual environment and confirm that OpenCV is correctly installed.

Run the command "import cv2" in Python shell.

The ***cv Python virtual environment*** is entirely independent and sequestered from the default Python version included in the download. So, any Python packages in the global site-packages directory will not be available to the cv virtual environment. Similarly, any Python packages installed in site-packages of cv will not be available to the global install of Python.

Now, enter in your Python interpreter:

```
python
```

and confirm that you are running the 3.5 (or above) version.

Inside the interpreter (the ">>>" will appear), import the OpenCV library:

```
import cv2
```

If no error messages appear, the OpenCV is correctly installed ON YOUR PYTHON VIRTUAL ENVIRONMENT.

You can also check the OpenCV version installed:

```
cv2.__version__
```

The 3.3.0 should appear (or a superior version that can be released in future).

Testing Your Camera

Once you have OpenCV installed in your system let's test to confirm that your camera is working properly.

I am assuming that you have a Cam already installed and enabled on your system.

Enter the below Python code on your IDE:

```
import numpy as np
import cv2cap = cv2.VideoCapture(0)
cap.set(3,640) # set Width
cap.set(4,480) # set Heightwhile(True):
    ret, frame = cap.read()
    frame = cv2.flip(frame, -1) # Flip camera vertically
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    cv2.imshow('frame', frame)
    cv2.imshow('gray', gray)

    k = cv2.waitKey(30) & 0xff
    if k == 27: # press 'ESC' to quit
        breakcap.release()
cv2.destroyAllWindows()
```

The above code will capture the video stream that will be generated by your Cam, displaying both, in BGR color and Gray mode..

You can alternatively download the code from my GitHub: simpleCamTest.py

To finish the program, you must press the key [ESC] on your keyboard. Click your mouse on the video window, before pressing [ESC].

Some people found issues when trying to open the camera and got "Assertion failed" error messages. That could happen if the camera was not enabled during OpenCv installation and so, camera drivers did not install correctly. To correct, use the command:

```
sudo modprobe bcm2835-v4l2
```

You can also add bcm2835-v4l2 to the last line of the /etc/modules file so the driver loads on boot.

6.2 Face Detection

The most basic task on Face Recognition is of course, "Face Detecting". Before anything, you must "capture" a face (Phase 1) in order to recognize it, when compared with a new face captured on future (Phase 3).

The most common way to detect a face (or any objects), is using the "Haar Cascade classifier"

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. 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. The good news is that OpenCV comes with a trainer as well as a detector. If you want to train your own classifier for any object like car, planes etc. you can use OpenCV to create one. Its full details are given here: Cascade Classifier Training.

If you do not want to create your own classifier, OpenCV already contains many pre-trained classifiers for face, eyes, smile, etc. Those XML files can be download from haarcascades directory.

Enough theory, let's create a face detector with OpenCV!

Download the file: faceDetection.py from my GitHub.

```
import numpy as np
import cv2faceCascade =
cv2.CascadeClassifier('Cascades/haarcascade_frontalface_default.xml')
cap = cv2.VideoCapture(0)
cap.set(3,640) # set Width
cap.set(4,480) # set Heightwhile True:
    ret, img = cap.read()
    img = cv2.flip(img, -1)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.2,
        minNeighbors=5,
        minSize=(20, 20)
    )
    for (x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]
    cv2.imshow('video',img)
    k = cv2.waitKey(30) & 0xff
    if k == 27: # press 'ESC' to quit
        breakcap.release()
cv2.destroyAllWindows()
```

Believe it or not, the above few lines of code are all you need to detect a face, using Python and OpenCV.

When you compare with the last code used to test the camera, you will realize that few parts were added to it. Note the line below:

```
faceCascade =
cv2.CascadeClassifier('Cascades/haarcascade_frontalface_default.xml')
```

This is the line that loads the “classifier” (that must be in a directory named “Cascades/”, under your project directory).

Then, we will set our camera and inside the loop, load our input video in grayscale mode (same we saw before).

Now we must call our classifier function, passing it some very important parameters, as scale factor, number of neighbors and minimum size of the detected face.

```
faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.2,
        minNeighbors=5,
        minSize=(20, 20)
        )
```

Where,

- **gray** is the input grayscale image.
- **scaleFactor** is the parameter specifying how much the image size is reduced at each image scale. It is used to create the scale pyramid.

- **minNeighbors** is a parameter specifying how many neighbors each candidate rectangle should have, to retain it. A higher number gives lower false positives.
- **minSize** is the minimum rectangle size to be considered a face.

The function will detect faces on the image. Next, we must "mark" the faces in the image, using, for example, a blue rectangle. This is done with this portion of the code:

```
for (x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
```

If faces are found, it returns the positions of detected faces as a rectangle with the left up corner (x,y) and having "w" as its Width and "h" as its Height ==> (x,y,w,h). Please see the picture.

Once we get these locations, we can create an "ROI" (drawn rectangle) for the face and present the result with *imshow()* function.

Run the above python Script on your python environment, using l:

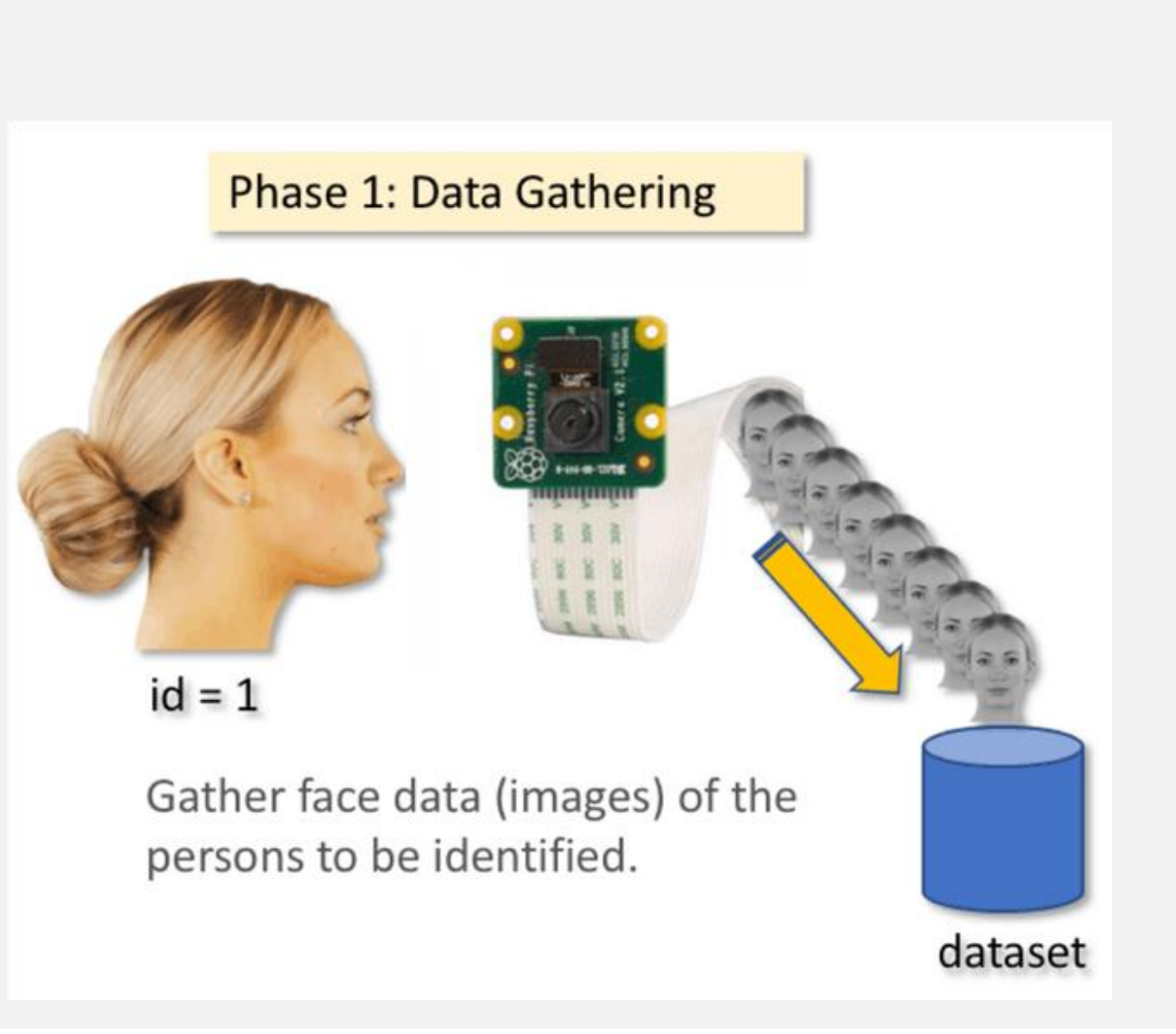
```
python faceDetection.py
```

You can also include classifiers for "eyes detection" or even "smile detection". On those cases, you will include the classifier function and rectangle draw inside the face loop, because would be no sense to detect an eye or a smile outside of a face.

6.3 Data Gathering

FACE RECOGNITION — 3 parts.

Let's start the first phase of our project. What we will do here, is starting from last step (Face Detecting), we will simply create a dataset, where we will store for each id, a group of photos in gray with the portion that was used for face detecting.



First, create a directory where you develop your project, for example, FacialRecognitionProject:

```
mkdir FacialRecognitionProject
```

In this directory, besides the 3 python scripts that we will create for our project, we must have saved on it the Facial Classifier. You can download it from my GitHub: haarcascade_frontalface_default.xml

Next, create a subdirectory where we will store our facial samples and name it “dataset”:

```
mkdir dataset
```

And download the code from my GitHub: 01_face_dataset.py

```
import cv2
import oscam = cv2.VideoCapture(0)
cam.set(3, 640) # set video width
cam.set(4, 480) # set video height
face_detector =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')# For each
person, enter one numeric face id
face_id = input('\n enter user id end press <return> ==>  ')
print("\n [INFO] Initializing face capture. Look the camera and wait
...")# Initialize individual sampling face count
count = 0
while(True):
    ret, img = cam.read()
    img = cv2.flip(img, -1) # flip video image vertically
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_detector.detectMultiScale(gray, 1.3, 5)
    for (x,y,w,h) in faces:
        cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)
        count += 1
        # Save the captured image into the datasets folder
        cv2.imwrite("dataset/User." + str(face_id) + '.' +
                    str(count) + ".jpg", gray[y:y+h,x:x+w])
        cv2.imshow('image', img)
    k = cv2.waitKey(100) & 0xff # Press 'ESC' for exiting video
    if k == 27:
        break
    elif count >= 30: # Take 30 face sample and stop video
         break# Do a bit of cleanup
print("\n [INFO] Exiting Program and cleanup stuff")
cam.release()
cv2.destroyAllWindows()
```

The code is very similar to the code that we saw for face detection. What we added, was an “input command” to capture a user id, that should be an integer number (1, 2, 3, etc)

```
face_id = input('\n enter user id end press  ==>  ')
```

And for each one of the captured frames, we should save it as a file on a “dataset” directory:

```
cv2.imwrite("dataset/User." + str(face_id) + '.' + str(count) + ".jpg",
gray[y:y+h,x:x+w])
```

Note that for saving the above file, you must have imported the library “os”. Each file’s name will follow the structure:

```
User.face_id.count.jpg
```

For example, for a user with a face_id = 1, the 4th sample file on dataset/ directory will be something like:

```
User.1.4.jpg
```

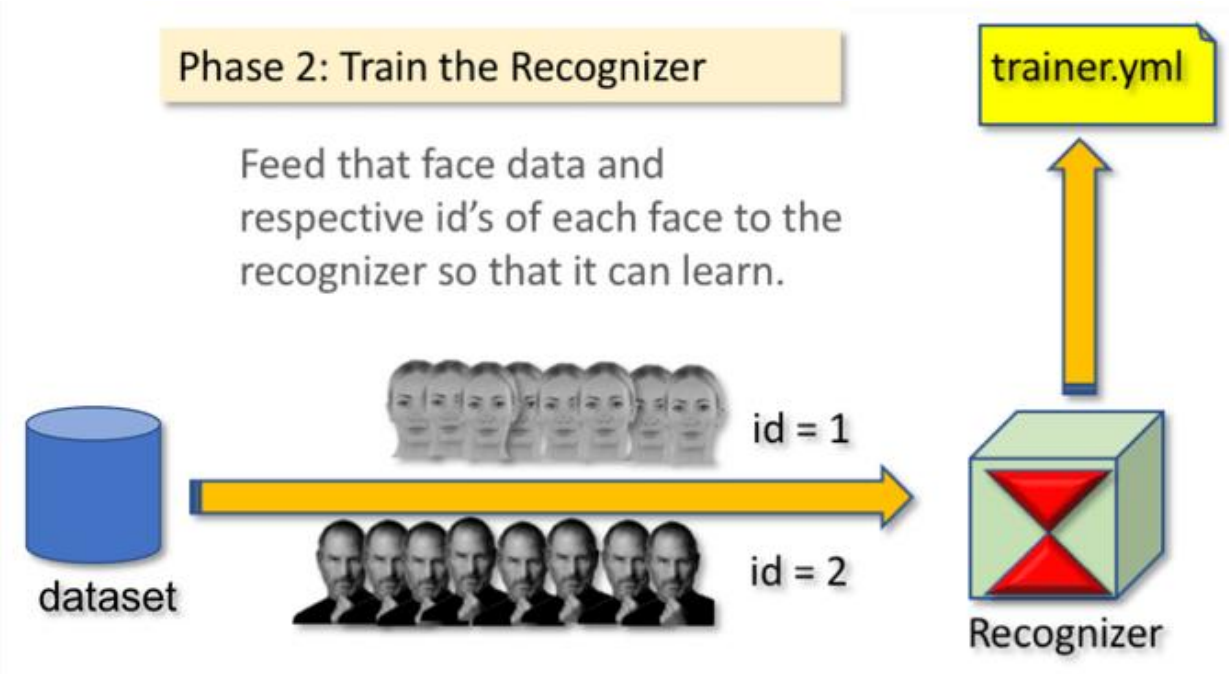
On my code, I am capturing 3 samples from each id. You can change it on the last “elif”. The number of samples is used to break the loop where the face samples are captured.

Run the Python script and capture a few Ids. You must run the script each time that you want to aggregate a new user (or to change the photos for one that already exists).

6.4 Trainer

On this second phase, we must take all user data from our dataset and “trainer” the OpenCV Recognizer. This is done directly by a specific

OpenCV function. The result will be a .yml file that will be saved on a "trainer/" directory.



So, let's start creating a subdirectory where we will store the trained data:

```
mkdir trainer
```

Download from my GitHub the second python script: [02_face_training.py]

```
import cv2
import numpy as np
from PIL import Image
import os# Path for face image database
path = 'dataset'
recognizer = cv2.face.LBPHFaceRecognizer_create()
detector =
cv2.CascadeClassifier("haarcascade_frontalface_default.xml");# function
to get the images and label data
```

```
def getImagesAndLabels(path):
    imagePaths = [os.path.join(path,f) for f in os.listdir(path)]
    faceSamples=[]
    ids = []
    for imagePath in imagePaths:
        PIL_img = Image.open(imagePath).convert('L') # grayscale
        img_numpy = np.array(PIL_img,'uint8')
        id = int(os.path.split(imagePath)[-1].split(".")[1])
        faces = detector.detectMultiScale(img_numpy)
        for (x,y,w,h) in faces:
            faceSamples.append(img_numpy[y:y+h,x:x+w])
            ids.append(id)
    return faceSamples,idsprint ("\n [INFO] Training faces. It will
take a few seconds. Wait ...")faces,ids = getImagesAndLabels(path)
recognizer.train(faces, np.array(ids))# Save the model into
trainer/trainer.yml
recognizer.write('trainer/trainer.yml') # Print the numer of faces
trained and end program
print("\n [INFO] {0} faces trained. Exiting
Program".format(len(np.unique(ids))))
```

```
# recognizer.save() worked on Mac, but not on Pi
```

Confirm if you have the PIL library installed on your python. If not, run the below command in Terminal:

```
pip install pillow
```

We will use as a recognizer, the LBPH (LOCAL BINARY PATTERNS HISTOGRAMS) Face Recognizer, included on OpenCV package. We do this in the following line:

```
recognizer = cv2.face.LBPHFaceRecognizer_create()
```

The function “getImagesAndLabels (path)”, will take all photos on directory: “dataset/”, returning 2 arrays: “Ids” and “faces”. With those arrays as input, we will “train our recognizer”:

```
recognizer.train(faces, ids)
```

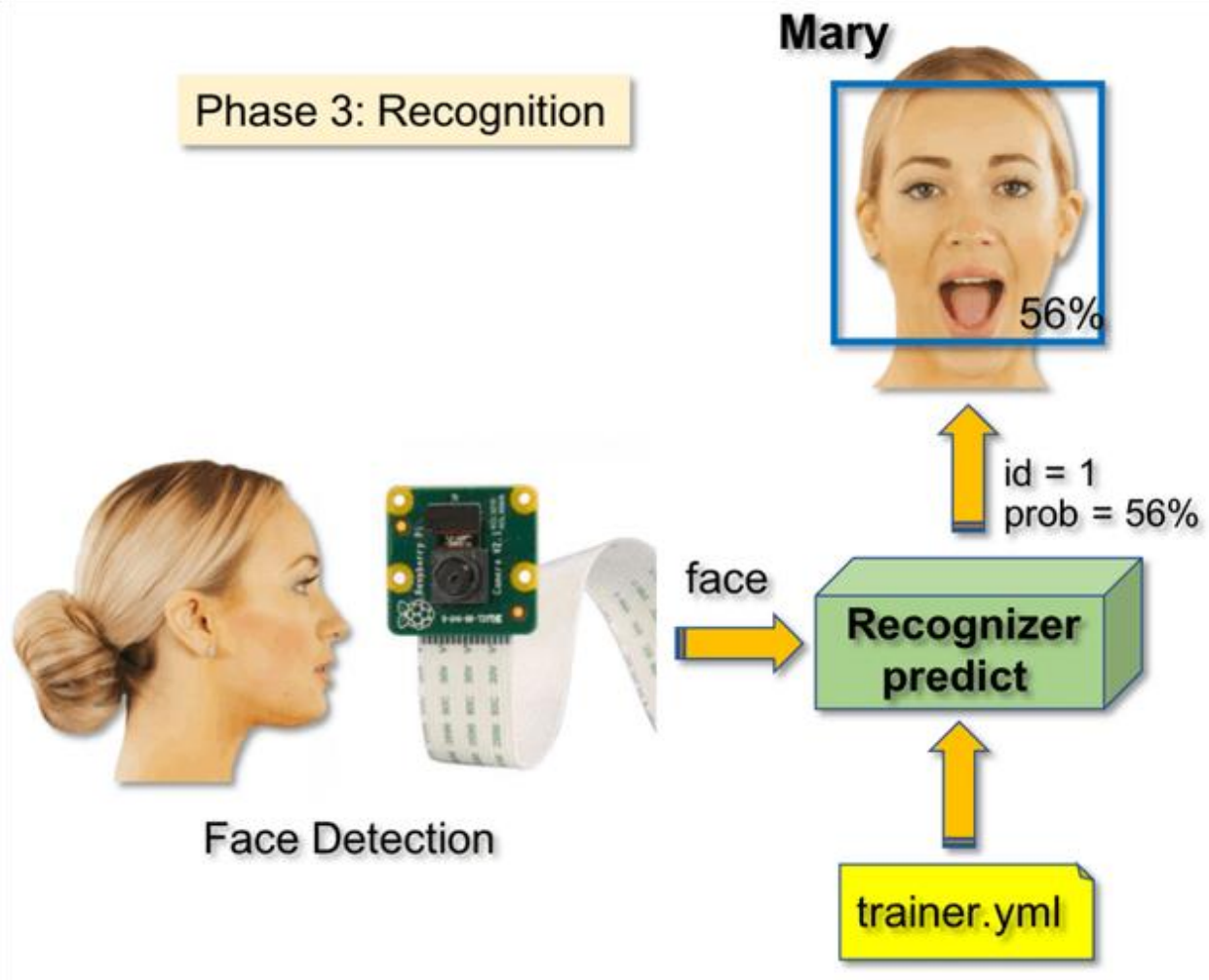
As a result, a file named “trainer.yml” will be saved in the trainer directory that was previously created by us.

That's it! I included the last print statement where I displayed for confirmation, the number of User's faces we have trained.

Every time that you perform Phase 1, Phase 2 must also be run.

6.5 Recognizer

Now, we reached the final phase of our project. Here, we will capture a fresh face on our camera and if this person had his face captured and trained before, our recognizer will make a "prediction" returning its id and an index, shown how confident the recognizer is with this match.



```
import cv2
import numpy as np
import os recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('trainer/trainer.yml')
cascadePath = "haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascadePath);
font = cv2.FONT_HERSHEY_SIMPLEX#iniciate id counter
id = 0# names related to ids: example ==> Marcelo: id=1,  etc
names = ['None', 'Marcelo', 'Paula', 'Ilza', 'Z', 'W'] # Initialize and
start realtime video capture
cam = cv2.VideoCapture(0)
cam.set(3, 640) # set video widht
cam.set(4, 480) # set video height# Define min window size to be
recognized as a face
minW = 0.1*cam.get(3)
minH = 0.1*cam.get(4)while True:
    ret, img =cam.read()
    img = cv2.flip(img, -1) # Flip vertically
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor = 1.2,
        minNeighbors = 5,
        minSize = (int(minW), int(minH)),
       )
    for(x,y,w,h) in faces:
        cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2)
        id, confidence = recognizer.predict(gray[y:y+h,x:x+w])
                # If confidence is less them 100 ==> "0" : perfect
match
        if (confidence < 100):
            id = names[id]
            confidence = "  {0}%".format(round(100 - confidence))
        else:
            id = "unknown"
            confidence = "  {0}%".format(round(100 - confidence))

        cv2.putText(
                    img,
                    str(id),
                    (x+5,y-5),
                    font,
                    1,
                    (255,255,255),
                    2
                   )
        cv2.putText(
                    img,
                    str(confidence),
                    (x+5,y+h-5),
```

```
                    font,
                    1,
                    (255,255,0),
                    1
                   )

    cv2.imshow('camera',img)
    k = cv2.waitKey(10) & 0xff # Press 'ESC' for exiting video
    if k == 27:
        break# Do a bit of cleanup
print("\n [INFO] Exiting Program and cleanup stuff")
cam.release()
cv2.destroyAllWindows()
```

We are including here a new array, so we will display “names”, instead of numbered ids:

```
names = ['None', 'paul', 'Cortana', 'GHOST', 'Z', 'W']
```

So, for example: Cortana will be the user with id = 2; Paul: id=1, etc.

Next, we will detect a face, same we did before with the haasCascade classifier. Having a detected face we can call the most important function in the above code:

```
id, confidence = recognizer.predict(gray portion of the face)
```

The recognizer.predict (), will take as a parameter a captured portion of the face to be analyzed and will return its probable owner, indicating its id and how much confidence the recognizer is in relation with this match.

Note that the confidence index will return “zero” if it will be cosidered a perfect match

And at last, if the recognizer could predict a face, we put a text over the image with the probable id and how much is the “probability” in % that the match is correct (“probability” = 100 — confidence index). If not, an “unknow” label is put on the face.

DATABASE CREATION:

The first step in the Attendance System is the creation of a database of faces that will be used. Different individuals are considered and a camera is used for the detection of faces and The images are saved in gray scale after being recorded by a camera. The LBPH recognizer is employed to coach these faces because the coaching sets the resolution and therefore the recognized face resolutions are completely variant. A part of the image is taken as the centre and the neighbours are thresholded against it. If the intensity of the centre part is greater or equal than it neighbour then it is denoted as 1 and 0 if not. This will result in binary patterns generally known as LBP codes.

I. FLOW CHART

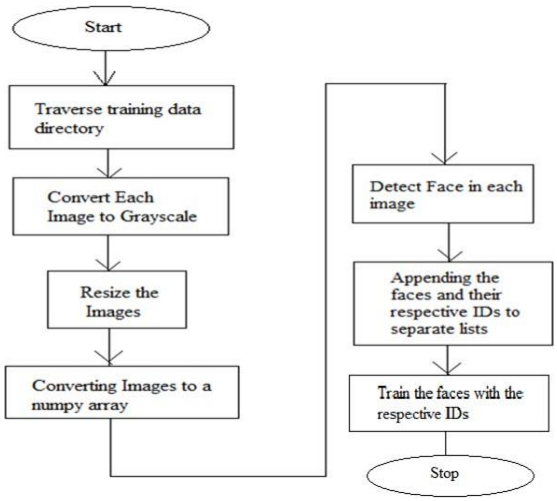


Fig 2. Flow-chart of the methodology used for Training Process

The training process starts with traversing of the training data

directory. Each image in the training date is converted into gray scale. A part of the image is taken as center and threshold its neighbours against it. If the intensity of the middle part is more or equal than its neighbour then denote it with 1 and 0 if not. After this the images are resized. Then the images are converted into a numpy array which is the central data structure of the numpy library. Each face in the image is detected. Creation of separate lists of each face is done and the faces are appended into them along with their respective IDs. The faces are then trained with their respective IDs.

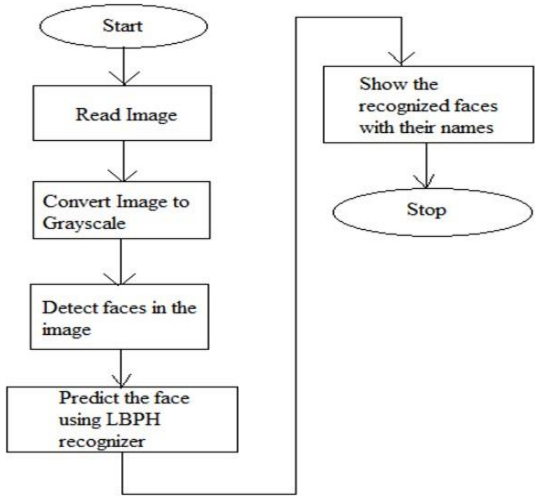


Fig 3. Flow-chart of the methodology used for Face Detection and Recognition

The input image is read by the camera of the phone. After the image is read it is converted into gray scale. The faces in the image are detected using the Haar Cascade frontal face module. Using the LBPH algorithm, the faces in the image are predicted. After the images are predicted, the recognized faces are shown in a green box along with their names.

1. Pandas

Pandas is an open source Python package that caters diverse tools for data analysis. The package contains various data structures that can be used for many diverse data manipulation tasks. It also includes a range of methods that can be invoked for data analysis, which becomes feasible when working on data science and machine learning problems in Python.

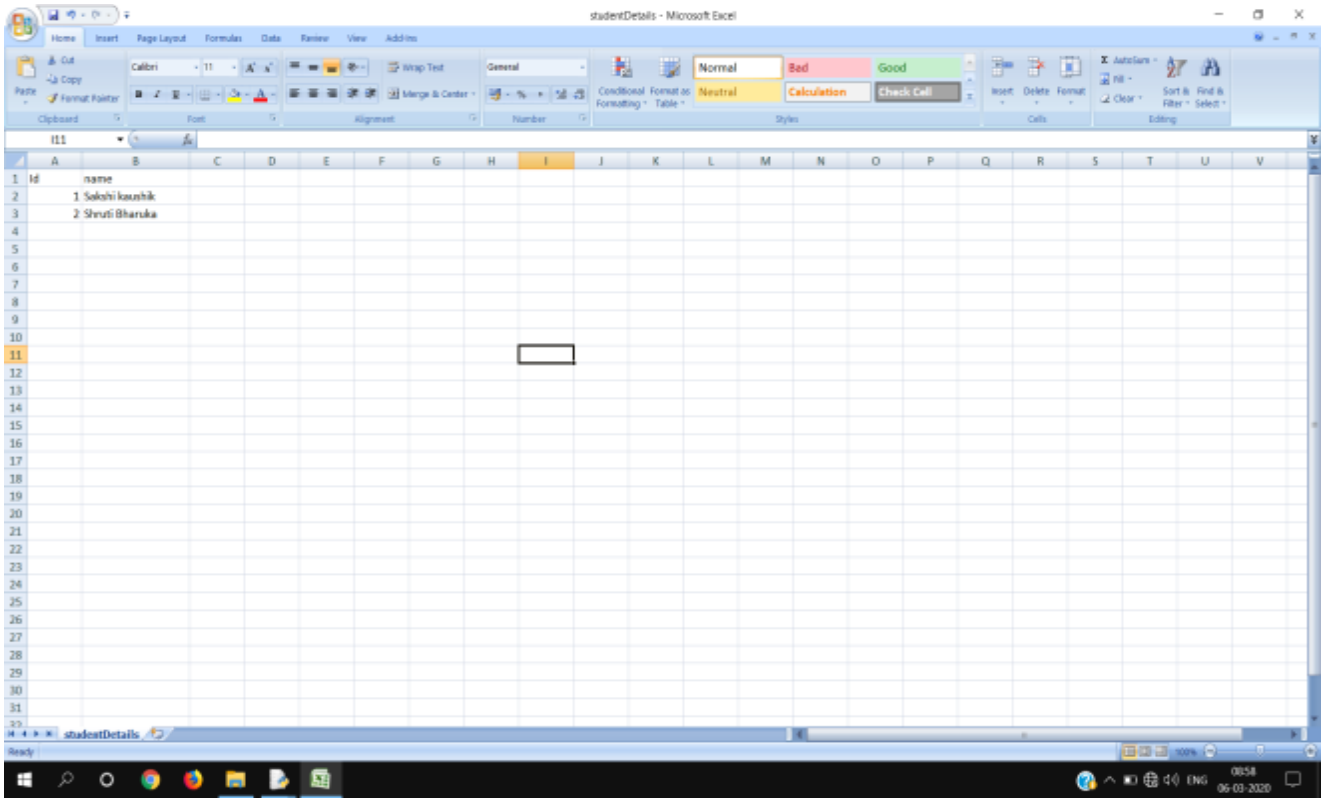
2. Idle

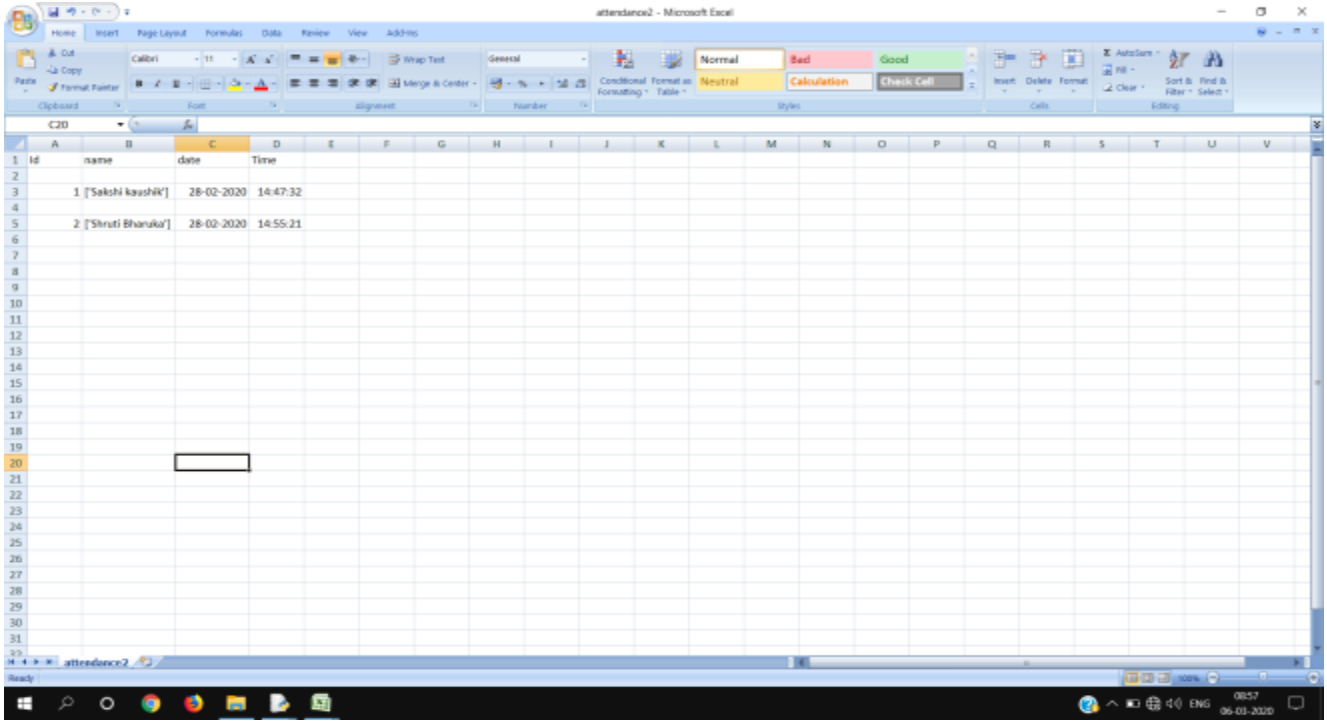
IDLE is Python's Integrated Development and Learning Environment. IDLE is completely coded in Python, using the tkinter GUI toolkit. It works mostly uniformly on Windows, Unix and macOS. It has a Python shell window (interactive interpreter) with colorizing of error messages, code input and code output. There is a multi-window text editor with multiple undo, Python colorizing, smart indent, call tips, auto completion, and other features. Searching within any window, replacing within editor windows and searching through multiple files is possible. It also has configuration, browsers and other dialogs as well.

3. Microsoft Excel

Microsoft Excel is a spreadsheet program incorporated in Microsoft Office suite of applications. Spreadsheets prompt tables of values arranged in rows and columns that can be mathematically manipulated using both basic and complex arithmetic functions and operations. Apart from its standard spreadsheet features, Excel also extends programming support via Microsoft's Visual Basic for Applications (VBA), the capacity to

access data from external sources via Microsoft's Dynamic Data Exchange (DDE) and extensive graphing and charting abilities. Excel being electronic spreadsheet program can be used to store, organize and manipulate the data. Electronic spreadsheet programs were formerly based on paper spreadsheets used for accounting purpose. The basic layout of computerized spreadsheets is more or less same as the paper ones. Related data can be stored in *tables* - which are a group of small rectangular boxes or cells that are standardized into rows and columns.





Algorithm for Colour Segmentation Using Thresholding

Segmentation is the process of identifying regions within an image [10]. Colour can be used to helpin segmentation. In this project, the hand on the image was the region of interest. To isolate the image pixels of the hand from the background, the range of the HSV values for skin colour was determined for use as

the threshold values. Segmentation could then proceed after the conversion ofall pixels falling within those threshold values to white and those without to black.

The algorithm used for the colour segmentation using thresholding is shown below:

1. Capture an image of the gesture from the camera.
2. Determine the range of HSV values for skin colour for use as threshold values.
3. Convert the image from RGB colour space to HSV colour space.
4. Convert all the pixels falling within the threshold values to white.
5. Convert all other pixels to black.
6. Save the segmented image in an image file.

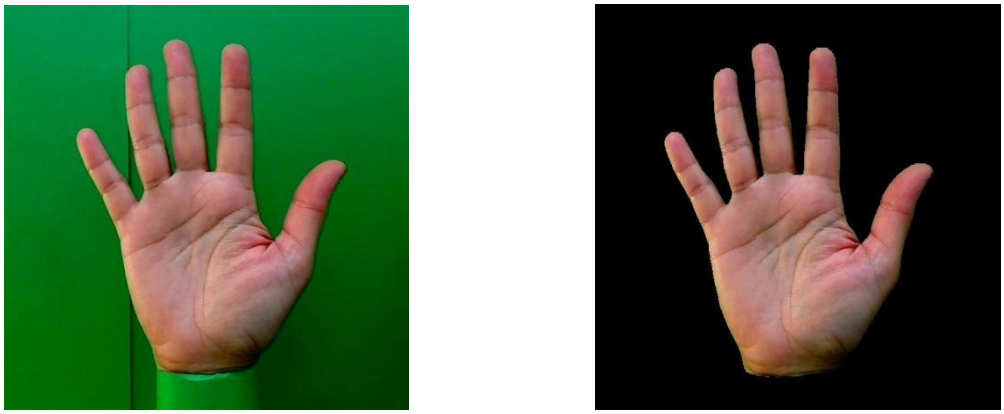
The pseudocode for the segmentation function is as follows:

```
//initialize threshold values for
skin colourmaxH = maximum
value for Hue
minH = minimum value for Hue
maxS = maximum value for
SaturationminS = minimum
```

```
value for Saturation maxV =
maximum value for Value
minV = minimum value for
Value

//initialize values for RGB components
pixelR = red
component pixelG =
green component
pixelB = blue
component

//the function uses two identical 3-channel colour images and
//saves the segmented pixels in
imageA segmentationfunction
(imageA, imageB)
{
        //convert imageB to HSV, using cvCvtColor
        function of OpenCVcvCvtColor(imageA, imageB,
        CV_RGB2HSV);

        //access all the image pixels
        for ( x=0; x < width of imageB; x++)
        {
                for ( y=0; y < height of imageB; y++)
                {
```

```
                    if(     //imageB pixels are pixels outside the
                            threshold rangepixelR of imageB < minV
                            ||

                            pixelR of imageB >
                            maxV || pixelG of
                            imageB < minS ||
                            pixelG of imageB >
                            maxS || pixelB of
                            imageB < minH ||
                            pixelB of imageB >
                            maxH  ){}

                    else {


                    }

            }

}//convert pixels to colour blackpixelR of imageA =0;pixelG of imageA =0;pixelB of
                                    imageA 0;

//convert pixels to colour whitepixelR of imageA =255; pixelG of imageA =255; pixelB
                                    of imageA =255;}
```

Figure 3 shows a sample raw image and the resulting image after colour segmentation bythresholding.



(a) (b)

Figure 3: Sample images. (a) Original image. (b) Image after colour segmentation.

Algorithm for Labeling and Blob Detection

The gesture in the colour segmented image should be recognized as one object before it can be interpreted. This can be done through the process of labeling and blob detection.

Labeling is the process of giving each region a unique integer number or label for the purpose of regional identification [3]. In effect, while no two neighboring regions should have the same

label,the pixels within one region should have the same label or description so that the region could be interpreted as one object or blob.

For the purpose of determining whether pixels might belong to the same region, their adjacencyrelationships can be examined. The two most common adjacency relationships are:

- 4-adjacency and
- 8-adjacency [8]

In 4-adjacency, a pixel is considered connected to its neighboring pixels if they occupy the left-most, top-most, right-most, and bottom positions with respect to the pixel. Using the (x,y) coordinate descriptions, a 4-adjacency relationship for pixel (x, y) is shown in Figure 4.

	(x, y-1)	
(x-1, y)	(x, y)	(x+1, y)
	(x, y+1)	

Figure 4. 4-Adjacency connectivity model.

In 8-adjacency, the neighboring pixels also include the top-left-most, top-right-most, bottom-left- most, and the bottom-right-most positions. Using the (x,y) coordinate descriptions, a 8-adjacencyrelationship for pixel (x, y) is shown in Figure 5.

(x-1, y-1)	(x, y-1)	(x+1, y-1)

(x-1, y)	(x, y)	(x+1, y)
(x-1, y+1)	(x, y+1)	(x+1, y+1)

Figure 5. 8-Adjacency connectivity model.

In labelling algorithms, pixels are examined one by one, row by row, and moving from left to right. Therefore, for the practical implementation of the algorithm, only the pixels that may be considered as existing at each point in time with respect to the pixel under scrutiny are considered. For the 4- adjacency model, these pixels would be the top-most and the left-most, as shown in Figure 6.

	(x, y-1)
(x-1, y)	(x, y)

Figure 6. Significant pixels in 4-adjacency model.

In the 8-adjacency model, these pixels would be the top-left-most, the top-most, the top-right-most,and the left-most, as shown in Figure 7.

(x-1, y-1)	(x, y-1)	(x+1, y-1)
(x-1, y)	(x,y)	

Figure 7. Significant pixel in 8-adjacency model.

Usually, images that undergo labelling are given preliminary processing to become binary images orgrayscale images. This will make it easier to identify the pixels of interest because the background pixels would either be zero-valued (coloured black) or measured against a threshold value in grayscale.

The labelling procedure looks at each pixel in sequence, checks it against the threshold value, and identifies its neighboring pixels based on the adjacency relationship model being used. If the neighboring pixels belong to a set, the pixel under consideration will be placed in that set. If the pixel has no neighbors, then it will be placed in a new set. Thereafter, all sets with connecting pixels

will be merged together into one set or blob and be considered as one object.

The algorithm for labelling and blob detection using an 8-adjacency relationship and a threshold value for a grayscale image is as follows:

1. Select the threshold value for the grayscale image.
2. For each non-zero pixel in the image that is above the threshold value:
   (a) If the pixel has non-zero labeled pixels in its immediate 8-adjacency neighborhood (namely, the top-right, top, top-left and left pixels), then give it a non-zero label (for example, assign the minimum value of the pixels in this neighborhood to the pixel as its label).
   (b) If the pixel has no neighboring pixels, then give it a new unused label and include it as part of a new set.
3. After all the pixels have been labeled and placed in sets, merge together the sets with connected pixels into one blob or object. Objects may be given different colours to distinguish them visually.

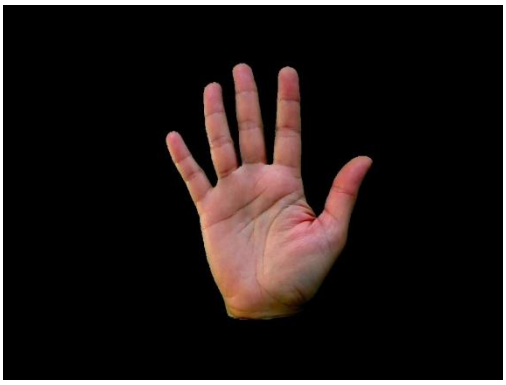
In pseudocode, the algorithm may be shown as follows:

Require: Set of sets, counters, threshold, input image I = i(x,y)

```
for (n=0; n < width of image; n++)
{
    for (m=0; m < height of image; m++)
    {
        if ( i(x,y) != 0 && i(x,y) > threshold)
        // not a background pixel and above the threshold
        {
            if ( i(x-1, y-1) != 0 || //it has at least one
                labeled neighbori(x-1, y) != 0 ||
                i(x, y-1) != 0 ||
                i(x+1, y-1) != 0 )
            {//give it a label
                i(x,y) → set(i(a,b)) {where (a,b) ∈
                    {(x-1, y-1), (x-1, y), (x, y-1), (x+1, y-1)}}
                //merge sets, if possible
                if ( set(i(x-1, y-1)) != set(i( x-1, y)) )
                {
                    set(i(x-1, y-1))  ○  set(i( x-1, y))
                }
```

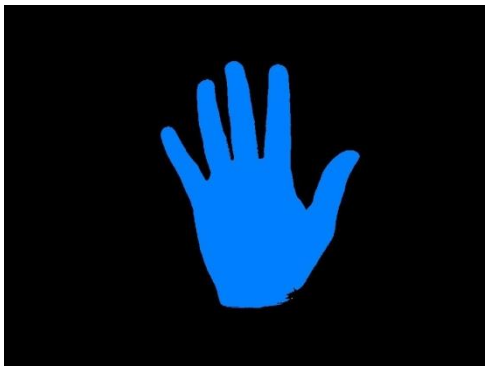
```
                    if ( set(i(x-1, y)) != set(i( x-1, y)) )
                    {
                        set(i(x-1, y))  ○  set(i(x-1, y))
                    }

                    if ( set(i(x, y-1)) != set(i( x, y-1)) )
                    {
                        set(i(x, y-1))  ○  set(i(x, y-1))
                    }

                    if ( set(i(x+1, y-1)) != set(i( x+1, y-1)) ){
                        set(i(x+1, y-1))  ○  set(i(x+1, y-1))
                    }
                }
                else  //make new set for pixels without neighbors
                {
                    makeNewSet( set(i(x, y)))
                    i(x,y) → set(i(x, y))
                }
            }
        }
    }
```

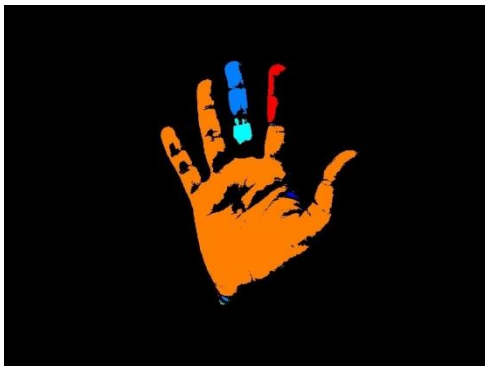
Figure 8 shows a sample original segmented image of hand with skin colour that is converted to a grayscale image, and the finally into two separate images, one showing a single blob using a lower lower threshold and another showing multiple blob using a higher threshold value.



(a)

(b)





(c) (d)

Figure 8: Sample images showing the results of labelling and blob detection. (a) Original segmented image. (b) Image in grayscale. (c) Image with a single blue-coloured blob, using of a threshold pixel value = 10. (d) Image showing multiple blobs with individual colours, using a higher threshold pixel value = 120.

Algorithm for Feature Extraction (Moment Invariants)

A digital approximation of finding the geometric moments of an image involves the product of pixel values and pixel positions, as shown by the formulas below with respect to an M x N image *i(x, y)* for the moment of order (*p* + *q*):

$$m_{pq} = \sum_{x=0}^{M} \sum_{y=0}^{N} x^p y^q i(x, y)$$

The moment of order zero (m_{00}) is equivalent to the total intensity of the image and for a blob the total geometrical area. The first-order functions m_{10} *and* m_{01} give the intensity moments about the y-axis and x-axis respectively.

The intensity centroid ($\bar{x}$, $\bar{y}$) gives the geometric center of the image and the formula to obtain it is:

$$\bar{x}=\frac{m_{10}}{m_{00}} \qquad \bar{y}=\frac{m_{01}}{m_{00}}$$

The central moments are moments computed with respect to the centroid. The central moment

μ_{pq} is calculated as follows:

$$\mu_{pq}=\sum_{x=0}^{M}\sum_{y=0}^{N}(x-\bar{x})^{p}(y-\bar{y})^{q}\, i(x,y)$$

The normalized central moment η_{pq} is computed as:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}} \quad \text{where} \quad \gamma = \frac{p+q+2}{2}$$

The first of Hu's moment invariants is given by:

$$\eta_1 = \eta_{20} + \eta_{02}$$

As an illustrative example, a C++ code fragment implementing the computation of the first Hu's moment invariant using an arbitrary 3x3 image is shown below.

```
int i, j; double sum = 0.0;
double image[3][3] = {{4,5,6},{7,8,9},{10,11,12}};
double m00 = 0.0; //representing
m00double m10 = 0.0; //representing
```

m_{10}double m01 = 0.0; //representing

m_{01}33

//computation of $m_{00}=\sum_{x=0}\sum_{y=0} x^0 y^0 i(x, y)$

```
for(i=0; i<3; i++)
{
    for(j=0; j<3; j++)
    {
        cout<< setw(5)<< pow(i,0) * pow(j,0) * image[i][j] << " ";m00 =
        m00 + pow(i,0) * pow(j,0) * image[i][j];
    }
        cout<<endl;
}3    3
```

// computation of $m_{10}=\sum_{x=0}\sum_{y=0} x^1 y^0 i(x, y)$

```
for(i=0; i<3; i++)
{
    for(j=0; j<3; j++)
    {
        cout<< setw(5)<< pow(i,1) * pow(j,0) * image[i][j] << " ";m10 =
        m10 + pow(i,1) * pow(j,0) * image[i][j];
    }
        cout<<endl;
}3    3
```

//computation of $m_{01}=\sum_{x=0}\sum_{y=0} x^0 y^1 i(x, y)$

```
        for(i=0; i<3; i++)
        {
            for(j=0; j<3; j++)
            {
                cout<< setw(5)<< pow(i,0) * pow(j,1) * image[i][j] << " ";m01 =
                m01 + pow(i,0) * pow(j,1) * image[i][j];
            }
                cout<<endl;
        }

    //computation of average x and average y;

    double aveX=0.0, aveY=0.0;
    aveX = m10/m00;      //for
    m10/m00;cout<<endl<<"aveX=
    "<<aveX<<endl;

    aveY = m01/m00;         //for m01/m00;
     cout<<endl<<"aveY=
     "<<aveY<<endl<<endl;

  // computation of the central moment :   upq00 for
              M N
                   // y00
```

$$= \sum_{x=0} \sum_{y=0} (x - x)^0 (y - y)^0 i(x, y)$$

```
double upq00 = 0.0, difx=0.0, dify=0.0;
    cout<<endl<<"upq00"<<endl;
for(i=0; i<3; i++)
    {
        for(j=0; j<3; j++)
        {
            difx = aveX - i;
    dify = aveY - j;
    cout<< setw(10)<< pow(difx,0) * pow(dify,0) * image[i][j] << " ";upq00 =
    upq00 + pow(difx,0) * pow(dify,0) * image[i][j];
        }
        cout<<endl;
    }
    cout<<endl<<"upq00= "<< upq00 <<endl;
```

//normalized central moment : npq00 for $\eta_{00} = \frac{y_{00}}{y_{00}}$

```
double uxpo=0.0; int p=0, q=0;
uxpo = (p + q + 2)/2;
cout<<endl<<"uxpo= " << uxpo <<endl;

double npq00 =0.0;
```

```
npq00 = upq00/pow(upq00,uxpo);
cout<<endl<<"npq00= "<< npq00 << endl;

//computation of Hu's first invariant      ?1=?20+?02
double invar1=0.0;
double upq20 = 0.0;
```

//upq20 for $y_{20} = \sum_{x=0}^{M}\sum_{y=0}^{N} (x-\bar{x})^2 (y-\bar{y})^0 i(x, y)$

```
difx=0.0; dify=0.0;
```

// npq20 for $?_{20} = \frac{y_{20}}{y_{00}}$?

```
cout<<endl<<"upq20"<<en
dl;for(i=0; i<3; i++)
    {
        for(j=0; j<3; j++)
        {
            difx = aveX - i;
    dify = aveY - j;
```

```
        cout<< setw(10)<< pow(difx,2) * pow(dify,0) * image[i][j] << " ";upq20 =
        upq20 + pow(difx,2) * pow(dify,0) * image[i][j];
                }
                cout<<endl;
        }
        cout<<endl<<"upq20= "<< upq20 <<endl;


uxpo=0.0; p=2, q=0;
uxpo = (p + q + 2)/2;
cout<<endl<<"uxpo= " << uxpo <<endl;


double npq20 =0.0;
npq20 = upq20/pow(upq00,uxpo);
cout<<endl<<"npq20= "<< npq20 << endl;


                        y02
// npq02 for      ?02= ——?
                        y
                        00

double upq02 = 0.0;
difx=0.0; dify=0.0;
cout<<endl<<"upq02"<<en
dl;
        for(i=0; i<3; i++)
        {
                for(j=0; j<3; j++)
```

```
        {
                difx = aveX - i;
    dify = aveY - j;
    cout<< setw(10)<< pow(difx,0) * pow(dify,2) * image[i][j] << " ";upq02 =
    upq02 + pow(difx,0) * pow(dify,2) * image[i][j];
        }
        cout<<endl;
    }
    cout<<endl<<"upq02= "<< upq02 <<endl;

uxpo=0.0; p=0, q=2;
uxpo = (p + q + 2)/2;
cout<<endl<<"uxpo= " << uxpo <<endl;

double npq02 =0.0;          // η02=y02/y00

npq02 = upq02/pow(upq00,uxpo);
cout<<endl<<"npq02= "<< npq02 << endl;

invar1 = npq20 + npq02;  // η1=η20+η02
cout<<endl<<"invar1 = npq20 + npq02
"<<endl;cout<<endl<<"invar1= "<<
invar1<<endl;
```

Algorithm for Feature Extraction (Center of Mass)

The center of mass is a simple feature of the image or object that is easy to compute. It is simple average or mean of the x and y coordinates, computed individually, to determine the *x* and *y* coordinates of the center of the image or object.

$$center_x = \frac{\sum_{i=0}^{n} x_i}{n} \qquad center_y = \frac{\sum_{i=0}^{n} y_i}{n}$$

where *n* is the total number of points of the image or object.

The pseudocode for computing the center of mass appears below:

```
//getting the center of mass of the blob
int centerx=0, centery=0; int totalpts = total
points in blob;for (int w=0; w < totalpts; w++)
{
```

```
        //access each point in
        the blob eachpt =
        getNextPoint(blob, w);

         //add total x's and total y's
                centerx=centerx+eac
                hpt->x;
                centery=centery+eac
                hpt->y;
}
//get the center of mass
centerx=centerx/totalpts;
centery=centery/totalpts;
```

Figure 9 shows the results of the computation of the center of mass of a blob object. The center ofmass is represented by a red cross.

Figure 9. Screenshot of blob with the center of mass marked with a red cross.

Implement the Algorithms Using C/C++ and OpenCV

The next step was to implement the needed algorithms to process the images and to recognize thegestures, using C/C++ and OpenCV. The resulting program should do the following procedure:

- capture the gestures through a web camera
- set-up an interface to convert from RGB to HSV colour space and determine the HSV valuesfor skin colour
- use the HSV values to reduce the image to black and white
- produce a grayscale image in preparation for labelling and blob detection
- produce a blob of the hand in preparation for feature extraction
- extract features from the blob for gesture recognition
- output the recognized gesture/ command for robot navigation.

Capture of Images Through a Webcam

If the webcam is properly installed, OpenCV would easily detect it and OpenCV capture functions can be used to get the images from the camera. The cvCaptureFromCAM() function initializes capturing video from the camera and stores it in the CvCapture structure, which is the video capturing structure. The cvCaptureFromCAM() function may be passed an index number to identifywhich camera is to be the source of images. In case there is only 1 camera, the parameter (0) is used. Thereafter, the CvGrabFrame() function may grab the frame (image) from the CvCapture structure, and from there the frame may be retrieved through the CvRetrieveFrame function and stored in an IplImage structure [9]. From there, the image can be displayed on the screen by the cvShowImage() function in a window created through the cvNamedWindow() function. For video streaming the process of capture and display may be placed inside a loop.

Below is the code fragment for capturing and displaying images through the webcam:

```
IplImage *imagecaptured;
CvCapture* capture = 0;
capture =
cvCaptureFromCAM(0);for ( ; ; )
{
        if( cvGrabFrame( capture )) {
                imagecaptured = cvRetrieveFrame( capture );
        }
        cvNamedWindow( "captured image", 0 );
        cvShowImage( "captured image", imagecaptured
        );
}
```

Interface for Adjustment of HSV Values for Skin Colour

The range of HSV values for skin colour had to be determined in preparation for the reduction ofthe image into two colours for eventual segmentation. OpenCV trackbars were used for this purpose. The trackbars or sliders represented the minimum and

maximum range of the Hue, Saturation and Value of the HSV colour space. Skin colour was successfully isolated using the following combination of slider values:

```
const int
upH=155;
const int
loH=109;const
int upS=112;
const int
loS=35; const
int upV=255;
const int
loV=208;
```

However, the combination did not work all of the time. The HSV values for skin colour had to be adjusted each time, depending on the type of camera used and the amount of illumination present. Fortunately, making the periodic adjustments was facilitated by the use of the trackbars.

The cvCreateTrackbar() function used to create the trackbars

have the following parameters: thetrackbar name, window name, pointer to integer value showing the trackbar position, maximum position of the trackbar, and the pointer to the function to be called when the trackbar changes position. The cvSetTrackbarPos() function sets the new position of a trackbar [9].

The code fragment to set up the trackbars is shown below, using the combination of HSV values represented by the constant variables mentioned above:

```
int marker_upH = upH;

int marker_loH =loH;

int marker_upS = upS;

int marker_loS = loS;

int marker_upV = upV;

int marker_loV = loV;


cvCreateTrackbar( "upH", "result", &marker_upH, 255,  NULL);

cvSetTrackbarPos("upH","result",upH);

cvCreateTrackbar( "loH", "result", &marker_loH,  255,  NULL);

cvSetTrackbarPos("loH","result",loH);
```

```
cvCreateTrackbar( "upS", "result", &marker_upS,  255,  NULL);
cvSetTrackbarPos("upS","result",upS);

cvCreateTrackbar( "loS", "result", &marker_loS,  255,  NULL);
cvSetTrackbarPos("loS","result",loS);

cvCreateTrackbar( "upV", "result", &marker_upV,  255,  NULL);
cvSetTrackbarPos("upV","result",upV);

cvCreateTrackbar( "loV", "result", &marker_loV,  255,  NULL);
cvSetTrackbarPos("loV","result",loV);
```

Figure 11 shows a sample window with trackbars for colour segmentation.

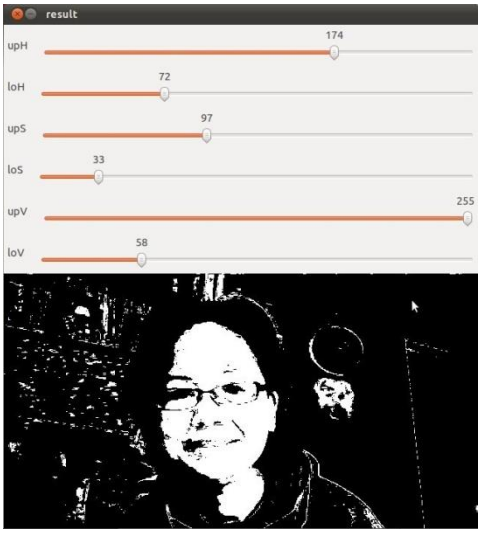


Figure 11. Sample window with trackbars for HSV values

Production of a Black and White Image

After the HSV skin colour range was determined, it was used as the threshold range to convert the image into two colours: black for the background and white for the hand. It was easier to implement the convertion through a separate function. Here the Find_markers() function first converts the source RGB image to an HSV image (using cvCvtColor()), and then loops through each pixel in the HSV image to pick out the skin-coloured pixels, turns the corresponding pixels in the original source image to white, and finally turns all other pixels in the original source image to black. This converts the original image into black and white. This step facilitates the isolation of the hand as the object of interest.

The code fragment to convert the image into black and white using the HSV values is shown below:

```
//skin colour
setupconst
int upH=155;
const int
loH=109;
```

```
const int
upS=112;
const int
loS=35; const
int upV=255;
const int
loV=208;

int
marker_upH
=upH;

int
marker_loH=
loH;

int
marker_upS
=upS;       int
marker_loS=l
oS;

int
marker_upV
=upV;

int
marker_loV=
loV;
```

```
void Find_markers(IplImage* image, IplImage *imageHSV){
//convert to HSV, the order is the same
        as BGR
        cvCvtColor(image,imageHSV,
        CV_RGB2HSV);
//segment markers
        for (int x=0; x < imageHSV->width; x++){
            for (int y=0; y < imageHSV->height; y++){
                    if(     pixelR(imageHSV,x,y) < marker_loV ||
                            pixelR(imageHSV,x,y) >
                            marker_upV ||
                            pixelG(imageHSV,x,y) <
                            marker_loS ||
                            pixelG(imageHSV,x,y) >
                            marker_upS ||
                            pixelB(imageHSV,x,y) <
                            marker_loH ||
                            pixelB(imageHSV,x,y) >
                            marker_upH){
                            pixelR(image,x,y)=0;
                            pixelG(image,x,y)=0;
                            pixelB(image,x,y)=0;
                    }
                    else {
                            pixelR(image,x,y
                            )=255;
                            pixelG(image,x,
```

```
                    y)=255;
                    pixelB(image,x,y
                    )=255;
                }
            }
        }
    }
```

HAND GESTURE METHOD

Hand gestures are an aspect of body language that can be conveyed through the center of the palm, the finger position and the shape constructed by the hand. Hand gestures can be classified into static and dynamic. As its name implies, the static gesture refers to the stable shape of the hand, whereas the dynamic gesture comprises a series of hand movements such as waving. There are a variety of hand movements within a gesture; for example, a handshake varies from one person to another and changes according to time and place. The main difference between posture and gesture is that posture focuses more on the shape of the hand whereas gesture focuses on the hand movement. The main approaches to hand gesture research can be classified into the wearable glove-based sensor approach and the camera vision-based sensor approach [1,2].

Hand gestures offer an inspiring field of research because they can facilitate communication and provide a natural means of interaction that can be used across a variety of applications. Previously, hand gesture recognition was achieved with wearable sensors attached directly to the hand with gloves. These sensors detected a physical response according to hand movements or finger bending. The data collected were then processed using a computer connected to the glove with wire. This system of glove-based sensor could be made portable by using a sensor attached to a microcontroller.

As illustrated in Figure 1, hand gestures for human–computer interaction (HCI) started with the invention of the data glove sensor.

It offered simple commands for a computer interface. The gloves used different sensor types to capture hand motion and position by detecting the correct coordinates of

the location of the palm and fingers [3]. Various sensors using the same technique based on the angle of bending were the curvature sensor [4], angular displacement sensor [5], optical fiber transducer [6], flex sensors [7] and accelerometer sensor [8]. These sensors exploit different physical principles according to their type.

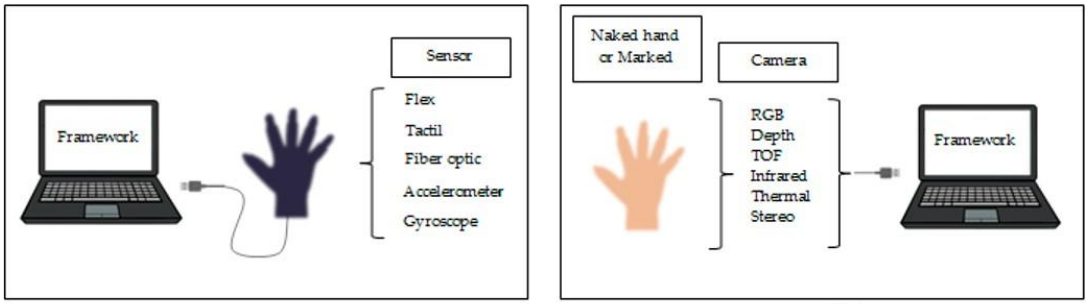


Figure 1. Different techniques for hand gestures. (**a**) Glove-based attached sensor either connected to the computer or portable; (**b**) computer vision–based camera using a marked glove or just a naked hand.

Although the techniques mentioned above have provided good outcomes, they have various limitations that make them unsuitable for the elderly, who may experience discomfort and confusion due to wire connection problems. In addition, elderly people suffering from chronic disease conditions that result in loss of muscle function may be unable to wear and take off gloves, causing them discomfort and constraining them if used for long periods. These sensors may also cause skin damage, infection or adverse reactions in people with sensitive skin or those suffering burns. Moreover, some sensors are quite expensive. Some of these problems were addressed in a study by Lamberti and Camastra [9], who developed a computer vision system based on colored marked gloves. Although this study did not require the attachment of sensors, it still required colored gloves to be worn.

These drawbacks led to the development of promising and cost-effective techniques that did not require cumbersome gloves to be worn. These techniques are called camera vision-based sensor technologies. With the evolution of open-source software libraries, it is easier than ever to detect hand gestures that can be used under a wide range of applications like clinical operations [10], sign

language [11], robot control [12], virtual environments [13], home automation [14], personal computer and tablet [15], gaming [16]. These techniques essentially involve replacement of the instrumented glove with a camera. Different types of camera are used for this purpose, such as RGB camera, time of flight (TOF) camera, thermal cameras or night vision cameras.

Algorithms have been developed based on computer vision methods to detect hands using these different types of cameras. The algorithms attempt to segment and detect hand features such as skin color, appearance, motion, skeleton, depth, 3D model, deep learn detection and more. These methods involve several challenges, which are discussed in this paper in the following sections.

Several studies based on computer vision techniques were published in the past decade. A study by Murthy et al. [17] covered the role and fundamental technique of HCI in terms of the recognition approach, classification and applications, describing computer vision limitations under various conditions. Another study by Khan et al. [18] presented a recognition system concerned with the issue of feature extraction, gesture classification, and considered the application area of the studies. Suriya et al. [19] provided a specific survey on hand gesture recognition for mouse control applications, including methodologies and algorithms used for human–machine interaction. In addition, they provided a brief review of the hidden Markov model (HMM). A study by Sonkusare et al. [20] reported various techniques and made comparisons between them according to hand segmentation methodology, tracking, feature extraction, recognition techniques, and concluded that the recognition rate was a tradeoff with temporal rate limited by computing power. Finally, Kaur et al. [16] reviewed

several methods, both sensor-based and vision-based, for hand gesture recognition to improve the precision of algorithms through integrating current techniques.

The studies above give insight into some gesture recognition systems under various scenarios, and address issues such as scene background limitations, illumination conditions, algorithm accuracy for feature extraction, dataset type, classification algorithm used and application. However, no review paper mentions camera type, distance limitations or recognition rate. Therefore, the objective of this study is to provide a comparative review of recent studies concerning computer vision techniques with regard to hand gesture detection and classification supported by different technologies. The current paper discusses the seven most reported approaches to the problem such as skin color, appearance, motion, skeleton, depth, 3D-model, deep-learning. This paper also discusses these approaches in detail and summarizes some modern research under different considerations (type of camera used, resolution of the processed image or video, type of segmentation technique, classification algorithm used, recognition rate, type of region of interest processing, number of gestures, application area, limitation or invariant factor, and detection range achieved and in some cases data set use, runtime speed, hardware run, type of error). In addition, the review presents the most popular applications associated with this topic.

The remainder of this paper is summarized as follows. Section 2 explains hand gesture methods and take consideration and focus on computer vision techniques, where describe seven most common techniques such as skin color, appearance, motion, skeleton, depth, 3D-module, deep learn and support that with tables. Section 3 illustrates in detail seven application areas that deal with hand gesture recognition systems. Section 4 briefly discusses research gaps and challenges. Finally, Section 5 presents our conclusions. Figure 2 below clarify the classification methods conducted by this review.

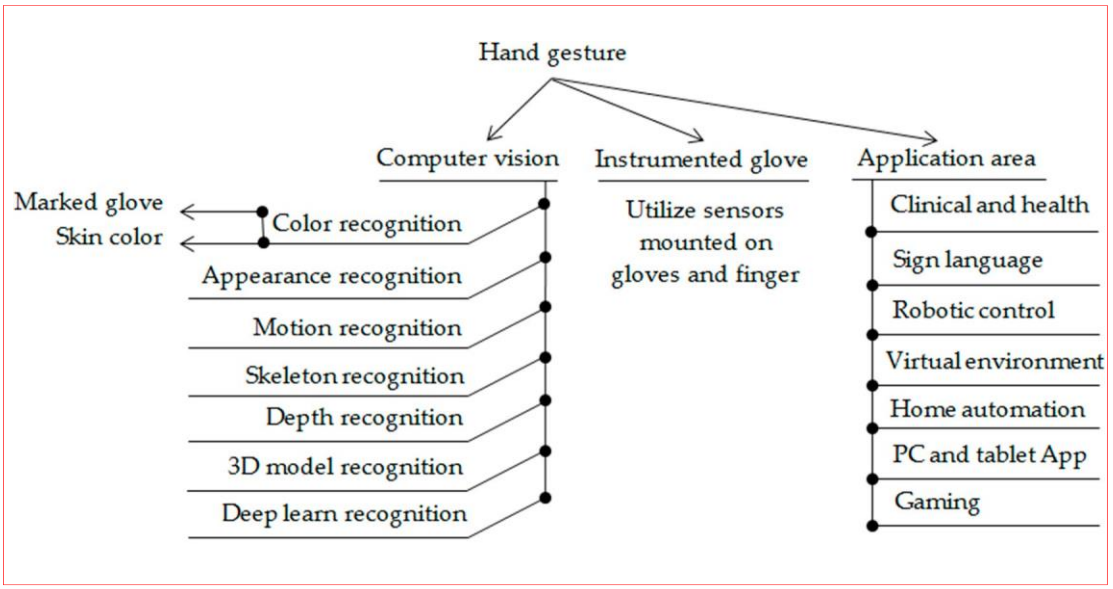


Figure 2. Classifications method conducted by this review.

1. Hand Gesture Methods

The primary goal in studying gesture recognition is to introduce a system that can detect specific human gestures and use them to convey information or for command and control purposes. Therefore, it includes not only tracking of human movement, but also the interpretation of that movement as significant commands. Two approaches are generally used to interpret gestures for HCI applications. The first approach is based on data gloves (wearable or direct contact) and the second approach is based on computer vision without the need to wear any sensors.

Hand Gestures Based on Instrumented Glove Approach

The wearable glove-based sensors can be used to capture hand motion and position. In addition, they can easily provide the exact coordinates of palm and finger locations, orientation and configurations by using sensors attached to the gloves [21–23]. However, this approach requires the user to be connected to the computer physically [23], which blocks the ease of interaction between user and computer. In addition, the price of these devices is quite high [23,24]. However, the modern glove based approach uses the technology of touch, which more promising technology and it is considered Industrial-grade haptic technology. Where the glove gives haptic feedback that makes user sense the shape, texture, movement and weight of a virtual object by using microfluidic technology. Figure 3 shows an example of a sensor glove used in sign language.

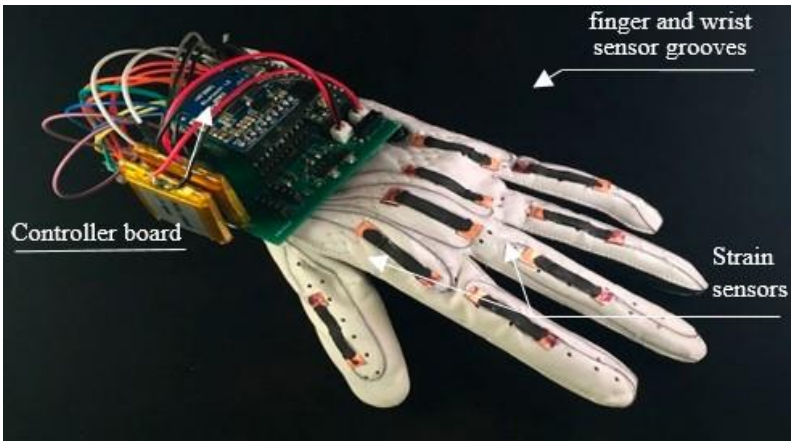


Figure 3. Sensor-based data glove (adapted from website: https://physicsworld.com/a/smart-glove- translates-sign-language-into-digital-text/).

Hand Gestures Based on Computer Vision Approach

The camera vision based sensor is a common, suitable and applicable technique because it provides contactless communication between humans and computers [16]. Different configurations of cameras can be utilized, such as monocular, fisheye, TOF and IR [20]. However, this technique involves several challenges, including lighting variation, background issues, the effect of occlusions, complex background, processing time traded against resolution and frame rate and foreground or background objects presenting the same

skin color tone or otherwise appearing as hands [17,21]. These challenges will be discussed in the following sections. A simple diagram of the camera vision-based sensor for extracting and identifying hand gestures is presented in Figure 4.

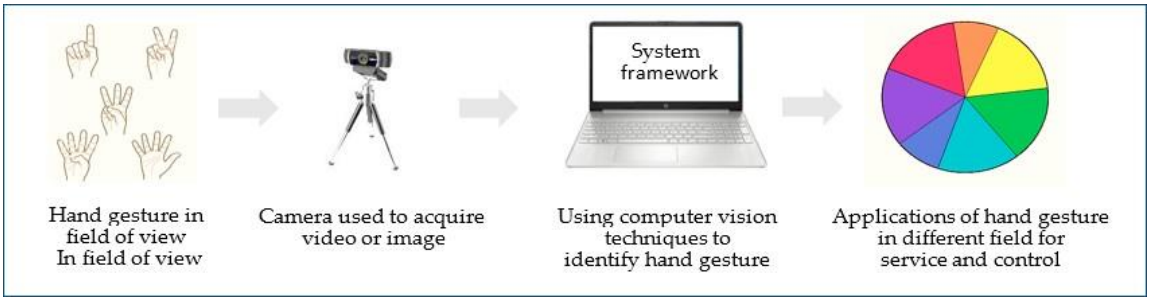


Figure 4. Using computer vision techniques to identify gestures. Where the user perform specific gesture by single or both hand in front of camera which connect with system framework that involve different possible techniques to extract feature and classify hand gesture to be able control some possible application.

Color-Based Recognition:

Color-Based Recognition Using Glove Marker

This method uses a camera to track the movement of the hand using a glove with different color marks, as shown in Figure 4. This method has been used for interaction with 3D models, permitting some processing, such as zooming, moving, drawing and writing using a virtual keyboard with good flexibility [9]. The colors on the glove enable the camera sensor to track and detect the location of the palm and fingers, which allows for the extraction of geometric model of the shape of the hand [13,25]. The advantages of this method are its simplicity of use and low price compared with the sensor data glove [9]. However, it still requires the wearing of colored gloves and limits the degree of natural and spontaneous interaction with the HCI [25]. The color-based glove marker is shown in Figure 5 [13].

Figure 5. Color-based recognition using glove marker [13].

Color-Based Recognition of Skin Color

Skin color detection is one of the most popular methods for hand segmentation and is used in a wide range of applications, such as object classification, degraded photograph recovery, person movement tracking, video observation, HCI applications, facial recognition, hand segmentation and gesture identification. Skin color detection has been achieved using two methods. The first method is pixel based skin detection, in which each pixel in an image is classified into skin or not, individually from its neighbor. The second method is region skin detection, in which the skin pixels are spatially processed based on information such as intensity and texture.

Color space can be used as a mathematical model to represent image color information. Several color spaces can be used according

to the application type such as digital graphics, image process applications, TV transmission and application of computer vision techniques [26,27]. Figure 6 shows an example of skin color detection using YUV color space.

Figure 6. Example of skin color detection. (**a**) Apply threshold to the channels of YUV color space in order to extract only skin color then assign 1 value for the skin and 0 to non-skin color; (**b**) detected and tracked hand using resulted binary image.

A several formats of color space are obtained for skin segmentation, as itemized below:

- red, green, blue (R–G–B and RGB-normalized);
- hue and saturation (H–S–V, H–S–I and H–S–L);

- luminance (YIQ, Y–Cb–Cr and YUV).

More detailed discussion of skin color detection based on RGB channels can be found in [28,29]. However, it is not preferred for skin segmentation purposes because the mixture of the color channel and intensity information of an image has irregular characteristics [26]. Skin color can detect the threshold value of three channels (red, green and blue). In the case of normalized-RGB, the color information is simply separated from the luminance. However, under lighting variation, it cannot be relied on for segmentation or detection purposes, as shown in the studies [30,31].

The characteristics of color space such as hue/saturation family and luminance family are good under lighting variations. The transformation of format RGB to HSI or HSV takes time in case of substantial variation in color value (hue and saturation). Therefore, a pixel within a range of intensity is chosen. The RGB to HSV transformation may consume time because of the transformation from Cartesian to polar coordinates. Thus, HSV space is useful for detection in simple images.

Transforming and splitting channels of Y–Cb–Cr color space is simple if compared with the HSV color family in regard to skin color detection and segmentation, as illustrated in [32,33]. Skin tone detection based Y–Cb–Cr is demonstrated in detail in [34,35].

The image is processed to convert RGB color space to another color space in order to detect the region of interest, normally a hand. This method can be used to detect the region through the range of possible colors, such as red, orange, pink and brown. The training sample of skin regions is studied to obtain the likely range of skin pixels with the band values for R, G and B pixels. To detect skin regions, the pixel color should compare the colors in the region with the predetermined sample color. If similar, then the region can be labeled as skin [36]. Table 1 presents a set of research papers that use different techniques to detect skin color.

The skin color method involves various challenges, such as illumination variation, background issues and other types of noise. A study by Perimal et al. [37] provided 14 gestures under controlled-conditions room lighting using an HD camera at short distance (0.15 to 0.20 m) and, the gestures were tested with three parameters, noise, light intensity and size of hand, which directly affect recognition rate. Another study by Sulyman et al. [38] observed that

using Y–Cb–Cr color space is beneficial for eliminating illumination effects, although bright light during capture reduces the accuracy. A study by Pansare et al. [11] used RGB to normalize and detect skin and applied a median filter to the red channel to reduce noise on the captured image. The Euclidian distance algorithm was used for feature matching based on a comprehensive dataset. A study by Rajesh et al. [15] used HSI to segment the skin color region under controlled environmental conditions, to enable proper illumination and reduce the error.

Another challenge with the skin color method is that the background must not contain any elements that match skin color. Choudhury et al. [39] suggested a novel hand segmentation based on combining the frame differencing technique and skin color segmentation, which recorded good results, but this method is still sensitive to scenes that contain moving objects in the background, such as moving curtains and waving trees. Stergiopoulou et al. [40] combined motion-based segmentation (a hybrid of image differencing and background subtraction) with skin color and morphology features to obtain a robust result that overcomes illumination and complex background problems. Another study by Khandade et al. [41] used a cross-correlation method to match hand segmentation with a dataset to achieve better recognition. Karabasi et al. [42] proposed hand gestures for deaf-mute communication based on mobile phones, which can translate sign language using HSV color space. Zeng et al. [43] presented a hand gesture method to assist wheelchair users indoors and outdoors using red channel thresholding with a fixed background to overcome the illumination change. A study by Hsieh et al. [44] used face skin detection to define skin color. This system can correctly detect skin pixels under low lighting conditions, and even when the face color is not in the normal range of skin chromaticity. Another study, by Bergh et al. [45], proposed a hybrid method based on a combination of the histogram and a pre-trained Gaussian mixture model to overcome lighting conditions. Pansare et al. [46] aligned

two cameras (RGB and TOF) together to improve skin color detection with the help of the depth property of the TOF camera to enhance detection and face background limitations.

Appearance-Based Recognition

This method depends on extracting the image features in order to model visual appearance such as hand and comparing these parameters with feature extracted from the input image frames. Where the features are directly calculated by the pixel intensities without a previous segmentation process. The method is executed in real time due to the easy 2D image features extracted and is considered easier to implement than the 3D model method. In addition, this method can detect various skin tones. Utilizing the AdaBoost learning algorithm, which maintains fixed feature such as key points for a portion of a hand, which can solve the occlusion issue [47,48], it can separate into two models: a motion model and a 2D static model. Table 2 presents a set of research papers that use different segmentation techniques based on appearance recognition to detect region of interest (ROI). A study by Chen et al. [49] proposed two approaches for hand recognition. The first approach focused on posture recognition using Haar-like features, which can describe the hand posture pattern effectively used the AdaBoost learning algorithm to speed up the performance and thus rate of classification. The second approach focused on gesture recognition using context-free grammar to analyze the syntactic structure based on the detected postures. Another study by Kulkarni and Lokhande [50] used three feature extraction method such as a histogram technique to segment and observe images that contained a large number of gestures, then suggested using edge detection such as Canny, Sobel and Prewitt operators to detect the edges with a different threshold. The classification gesture performed using feed forward back propagation artificial neural network with supervision learns. Some of the limitation reported by the author where conclude when use histogram technique the system gets misclassified result because histogram can only be used for the small number of gesture which completely different from each other. Fang et al. [51] used an extended AdaBoost method for hand detection and combined optical flow with the color cue for tracking. They also collected hand color from the neighborhood of features' mean position using a single

Gaussian model to describe hand color in HSV color space. Where multi feature extracted and gesture recognition using palm and finger decomposition, then utilizing scale-space feature detection where integrated into gesture recognition in order to encounter the limitation of aspect ratio which facing most of the learning of hand gesture methods. Licsa'r et al. [52] used a simple background subtraction method for hand segmentation and extended it to handle background changes in order to face some challenges such as skin like color and complex and dynamic background then used boundary-based method to classify hand gesture. Finally, Zhou et al. [53] proposed a novel method to directly extract the fingers where the edges were extracted from the gesture images, and then the finger central area was obtained from the obtained edges. Fingers were then obtained from the parallel edge characteristics. The proposed system cannot recognize the side view of hand pose. Figure 7 below show simple example on appearance recognition.

Figure 7. Example on appearance recognition using foreground extraction in order to segment only ROI, where the object features can be extracted using different techniques such as pattern or image subtraction and foreground and background segmentation algorithms.

According to information mentioned in Table 2. The first row indicates Haar-like feature which consider a good for analyze ROI pattern efficiently. Haar-like features can efficiently analyze the contrast between dark and bright object within a kernel, which can operate faster compared with pixel based system. In addition, it is immune for noise and lighting variation because they calculate the gray value difference between the white and black rectangles. The result of first row is 90%, but if compared with single gaussian model which used to describe hand color in HSV color space in the third row the result of recognition rate is 93%. Although both proposed system used the Adaboost algorithm to speed up the system and classification.

Motion-Based Recognition

Motion-based recognition can be utilized for detection purposes; it can be extracts the object through a series of image frames. The AdaBoost algorithm utilized for object detection, characterization, movement modeling, and pattern recognition is needed to recognize the gesture [16]. The main issue encounter motion recognition is this is an occasion if one more gesture is active at the recognition process and also dynamic background has a negative effect. In addition, the loss of gesture may be caused by occlusion among tracked hand gesture or error in region extraction from tracked gesture and effect long-distance on the region appearance Table 3 presents a set of research papers that used different segmentation techniques based on motion recognition to detect ROI.

Two stages for efficient hand detection were proposed in [54]. First, the hand detected for each frame and center point is used for tracking the hand. Then, the second stage matching model applying to each type of gesture using a set of features is extracted from the motion tracking in order to provide better classification where the main drawback of the skin color is affected by lighting variations which lead to detect non-skin color. A standard face detection algorithm and optical flow computation was used by [55] to give a user-centric coordinate frame in which motion features were used to recognize gestures for classification purposes using the multiclass boosting algorithm. A real-time dynamic hand gesture recognition system based on TOF was offered in [56], in which motion patterns were detected based on hand gestures received as input depth

images. These motion patterns were compared with the hand motion classifications computed from the real dataset videos which do not require the use of a segmentation algorithm. Where the system provides good result except the depth rang limitation of TOF camera. In [57], YUV color space was used, with the help of the CAMShift algorithm, to distinguish between background and skin color, and the naïve Bayes classifier was implemented to assist with gesture recognition. The proposed system faces some challenges such as illumination variation where light changes affect the result of the skin segment. Other challenges are the degree of gesture freedom which affect directly on the output result by change rotation. Next, hand position capture problem, if hand appears in the corner of the frame and the dots which must cover the hand does not lie on hand that may led to failing captured user gesture. In addition, the hand size quite differs between humans and maybe causes a problem with the interaction system. However, the major still challenging problem is the skin-like color which affects overall system and can abort the result. Figure 8 gives simple example on hand motion recognition.

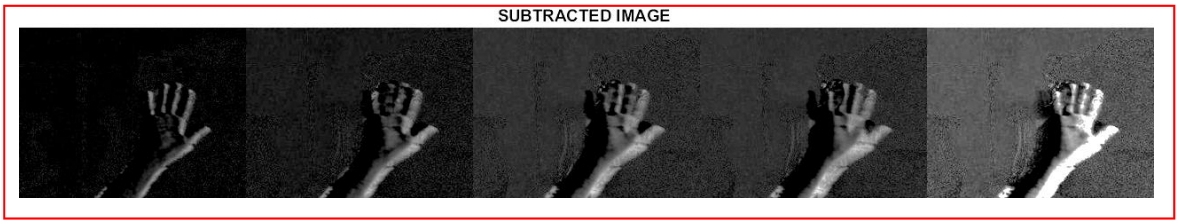


Figure 8. Example on motion recognition using frame difference subtraction to extract hand feature, where the moving object such as hand extracted from the fixed background.

According to information mentioned in Table 3. The first row recognition rate of system is 97%, where the hybrid system based on skin detect and motion detection is more reliable for gesture recognition, where the motion hand can track using multiple track candidates depend on stand derivation calculation for both skin and motion approach. Where every single gesture encoded as chain-code in order to model every single gesture which considers a simple model compared with (HMM) and classified gesture using a model of the histogram distribution. The proposed system in the third row use depth camera based on (TOF) where the motion pattern of the arm model for human utilized to define motion patterns, were the authors confirm that using the depth information for hand trajectories estimation is to improve gesture recognition rate. Moreover, the proposed system no need for the segmentation algorithm, where the system is examined using 2D and 2.5D approaches, were 2.5D performs better than 2D and gives recognition rate 95%.

Skeleton-Based Recognition

The skeleton-based recognition specifies model parameters which can improve the detection of complex features [16]. Where the various representations of skeleton data for the hand model can be used for classification, it describes geometric attributes and constraint and easy translates features and correlations of data, in order to focus on geometric and statistic features. The most common feature used is the joint orientation, the space between joints, the skeletal joint location and degree of angle between joints and trajectories and curvature of the joints. Table 4 presents a set of research papers that use different segmentation techniques based on skeletal recognition to detect ROI.

Hand segmentation using the depth sensor of the Kinect camera, followed by location of the fingertips using 3D connections, Euclidean distance, and geodesic distance over hand skeleton pixels to provide increased accuracy was proposed in [58]. A new 3D hand gesture recognition approach based on a deep learning model using parallel convolutional neural networks (CNN) to process hand skeleton joints' positions was introduced in [59], the proposed system has a limitation where it works only with complete sequence. The optimal viewpoint was estimated and the point cloud of gesture transformed using a curve skeleton to specify topology, then Laplacian-based contraction was applied to specify the skeleton points

in [60]. Where the Hungarian algorithm was applied to calculate the match scores of the skeleton point set, but the joint tracking information acquired by Kinect is not accurate enough which give a result with constant vibration. A novel method based on skeletal features extracted from RGB recorded video of sign language, which presents difficulties to extracting accurate skeletal data because of occlusions, was offered in [61]. A dynamic hand gesture using depth and skeletal dataset for a skeleton-based approach was presented in [62], where supervised learning (SVM) used for classification with a linear kernel. Another dynamic hand gesture recognition using Kinect sensor depth metadata for acquisition and segmentation which used to extract orientation feature, where the support vector machine (SVM) algorithm and HMM was utilized for classification and recognition to evaluate system performance where the SVM bring a good result than HMM in some specification such elapsed time, average recognition rate, was proposed in [63]. A hybrid method for hand segmentation based on depth and color data acquired by the Kinect sensor with the help of skeletal data were proposed in [64]. In this method, the image threshold is applied to the depth frame and the super-pixel segmentation method is used to extract the hand from the color frame, then the two results are combined for robust segmentation. Figure 9 show an example on skeleton recognition.

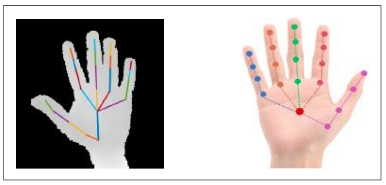


Figure 9. Example of skeleton recognition using depth and skeleton dataset to representation hand skeleton model [62].

According to information mentioned in Table 4. The depth camera provides good accuracy for segmentation, because not affected by lightening variations and cluttered background. However, the main issue is in the range of detection. The Kinect V1 sensor has an embedded system in which gives feedback information received by depth sensor as a metadata, which gives information about human body joint coordinate. The Kinect V1 provides information used to track skeletal joint up to 20 joints, that's help to module the hand skeleton. While Kinect V2 sensor can tracking joint as 25 joints and up to six people at the same time with full joints tracking. With a range of detection between (0.5–4.5) meter.

Depth-Based Recognition

Approaches have proposed for solving hand gesture recognition using different types of cameras. A depth camera provides 3D geometric information about the object [65]. Previously, both major approximations were utilized: TOF precepts and light coding. The 3D data from a depth camera directly reflects the depth field if compared with a color image which contains only a projection [66]. Using this approach, the lighting, shade, and color did not affect the result image. However, the cost, size and availability of the depth camera will limit its use [67]. Table 5 presents a set of research papers that use different segmentation techniques based on depth recognition to detect ROI.

The finger earth mover's distance (FEMD) approach was evaluated in terms of speed and precision, and then compared with the shape-matching algorithm using the depth map and color image acquired by the Kinect camera [65]. Improved depth threshold segmentation was offered in [68], by combining depth and color information using the hierarchical scan method, then hand segmentation by the local neighbor method; this approach gives a result over a range of up to two meters. A new method was proposed in [69], based on a near depth range of less than 0.5 m where skeletal data were not provided by Kinect. This method was implemented using two image frames, depth and infrared. A depth threshold was used in order to segment the hand, then a K-mean algorithm was applied to obtain both user's hand pixels [70]. Next, Graham's scan algorithm was used to detect the convex hulls of the hand in order to merge with the result of the contour tracing algorithm to detect the fingertip. The depth image

frame was analyzed to extract 3D hand gestures in real time, which were executed using frame differences to detect moving objects [71]. The foremost region was utilized and classified using an automatic state machine algorithm. The skin–motion detection technique was used to detect the hand, then Hu moments were applied to feature extraction, after which HMM was used for gesture recognition [72]. Depth range was utilized for hand segmentation, then Otsu's method was used for applying threshold value to the color frame after it was converted into a gray frame [14]. A kNN classifier was then used to classify gestures. In [73], where the hand was segmented based on depth information using a distance method, background subtraction and iterative techniques were applied to remove the depth image shadow and decrease noise. In [74], the segmentation used 3D depth data selected using a threshold range. In [75], the proposed algorithm used an RGB color frame, which converted to a binary frame using Otsu's global threshold. After that, a depth range was selected for hand segmentation and then the two methods were aligned. Finally, the kNN algorithm was used with Euclidian distance for finger classification. Depth data and an RGB frame were used together for robust hand segmentation and the segmented hand matched with the dataset classifier to identify the fingertip [76]. This framework was based on distance from the device and shape based matching. The fingertips selected using depth threshold and the K-curvature algorithm based on depth data were presented in [77]. A novel segmentation method was implemented in [78] by integrating RGB and depth data, and classification was offered using speeded up robust features (SURF). Depth information with skeletal and color data were used in [79], to detect the hand, then the segmented hand was matched with the dataset using SVM and artificial neural networks (ANN) for recognition. The authors concluded that ANN was more accurate than SVM. Figure 10 shows an example of segmentation using Kinect depth sensor.

3D Model-Based Recognition

The 3D model essentially depends on 3D Kinematic hand model which has a large degree of freedom, where hand parameter estimation obtained by comparing the input image with the two-dimensional appearance projected by three-dimensional hand model. In addition, the 3D model introduces human hand feature as pose estimation by forming volumetric or skeletal or 3D model that identical to the user's hand. Where the 3D model parameter updated through the matching process. Where the depth parameter is added to

the model to increase accuracy. Table 6 presents a set of research papers based on 3D model.

A study by Tekin et al. [80] proposed a new model to understand interactions between 3D hands and object using single RGB image, where single image is trained end-to-end using neural network, and show jointly estimation of the hand and object poses in 3D. Wan et al. [81] proposed 3D hand pose estimation from single depth map using self-supervision neural network by approximating the hand surface with a set of spheres. A novel of estimating full 3D hand shape and pose presented by Ge et al. [82] based on single RGB image. Where Graph Convolutional Neural Network (Graph CNN) utilized to reconstruct full 3D mesh for hand surface. Another study by Taylor et al. [83] proposed a new system tracking human hand by combine surface model with new energy function which continuously optimized jointly over pose and correspondences, which can track the hand for several meter from the camera. Malik et al. [84] proposed a novel CNN-based algorithm which automatically learns in order to segment hand from a raw depth image and estimate 3D hand pose estimation including the structural constraints of hand skeleton. Tsoli et al. [85] presented a novel method to track a complex deformable object in interaction with a hand. Chen et al. [86] proposed self-organizing hand network SO—Hand Net—which achieved 3D hand pose estimation via semi-supervised learning. Where end-to-end regression method utilized for single depth image to estimation 3D hand pose. Another study by Ge et al. [87] proposed a point-to-point regression method for 3D hand pose estimation in single depth images. Wu et al. [88] proposed novel hand pose estimation from a single depth image by combine detection based method and regression-based method to improve accuracy. Cai et al. [89] present one-way to adapt a weakly labeled real-world dataset from a fully annotated synthetic dataset with the aid of low-cost depth images and take only RGB inputs for 3D joint predictions. Figure 11 shows an example of a 3D hand model interaction with virtual system.

Figure 11. 3D hand model interaction with virtual system [83].

There are some reported limitations, such as 3D hand required a large dataset of images to formulate the characteristic shapes of the hand in case multi-view. Moreover, the matching process considers time consumption, also computation costly and less ability to treat unclear views.

Deep-Learning Based Recognition

The artificial intelligence offers a good and reliable technique used in a wide range of modern applications because of using a learning role principle. The deep learning used multilayers for learning data and gives a good prediction out result. The most challenges facing this technique is required dataset to learn algorithm which may affect time processing. Table 7 presents a set of research papers that use different techniques based on deep-learning recognition to detect ROI.

Authors proposed seven popular hand gestures which captured by mobile camera and generate 24,698 image frames. The feature extraction and adapted deep convolutional neural network (ADCNN) utilized for hand classification. The experiment evaluates result for the training data 100% and testing data 99%, with execution time 15,598 s [90]. While other proposed systems used webcam in order to track hand. Then used skin color (Y–Cb–Cr color space) technique and morphology to remove the background. In addition, kernel correlation filters (KCF) used to track ROI. The resulted image enters into a deep convolutional neural network (CNN). Where the CNN model used to compare performance of two modified from Alex Net and VGG Net. The recognition rate for training data and testing data, respectively 99.90% and 95.61% in [91]. A new method based on deep convolutional neural network, where the resized image directly feds into the network ignoring segmentation and detection stages in orders to classify hand gestures directly. The system works in real time and gives a result with simple background 97.1% and with complex background 85.3% in [92]. The depth image produced by Kinect sensor used to segment color

image then skin color modeling combined with convolution neural network, where error back propagation algorithm applied to modify the threshold and weights for the neural network. The SVM classification algorithm added to the network to enhance result in [93]. Other research study used Gaussian Mixture model (GMM) to filter out non-skin colors of an image which used to train the CNN in order to recognize seven hand gestures, where the average recognition rate 95.96 % in [94]. The next proposed system used long-term recurrent convolutional network-based action classifier, where multiple frames sampled from the video sequence recorded is fed to the network. In order to extract the representative frames, the semantic segmentation-based de-convolutional neural network is used. The tiled image patterns and tiled binary patterns are utilized to train the de-convolutional network in [95]. A double-channel convolutional neural network (DC-CNN) is proposed by [96] where the original image preprocessed to detect the edge of the hand before fed to the network. The each of two-channel CNN has a separate weight and softmax classifier used to classify output results. The proposed system gives recognition rate of 98.02%. Finally, a new neural network based on SPD manifold learning for skeleton-based hand gesture recognition proposed by [97]. Figure 12 below shown example on deep learn convolution neural network.

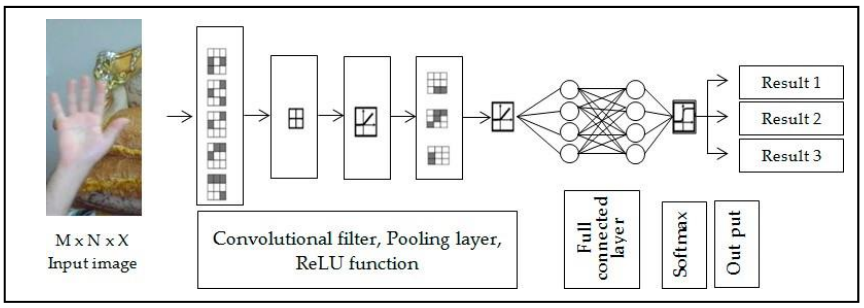


Figure 12. Simple example on deep learning convolutional neural network architecture.

2. Application Areas of Hand Gesture Recognition Systems

Research into hand gestures has become an exciting and relevant field; it offers a means of natural interaction and reduces the cost of using sensors in terms of data gloves. Conventional interactive methods depend on different devices such as a mouse, keyboard, touch screen, joystick for gaming and consoles for machine controls. The following sections describe some popular applications of hand gestures. Figure 13 shows the most common application area deal with hand gesture recognition techniques.

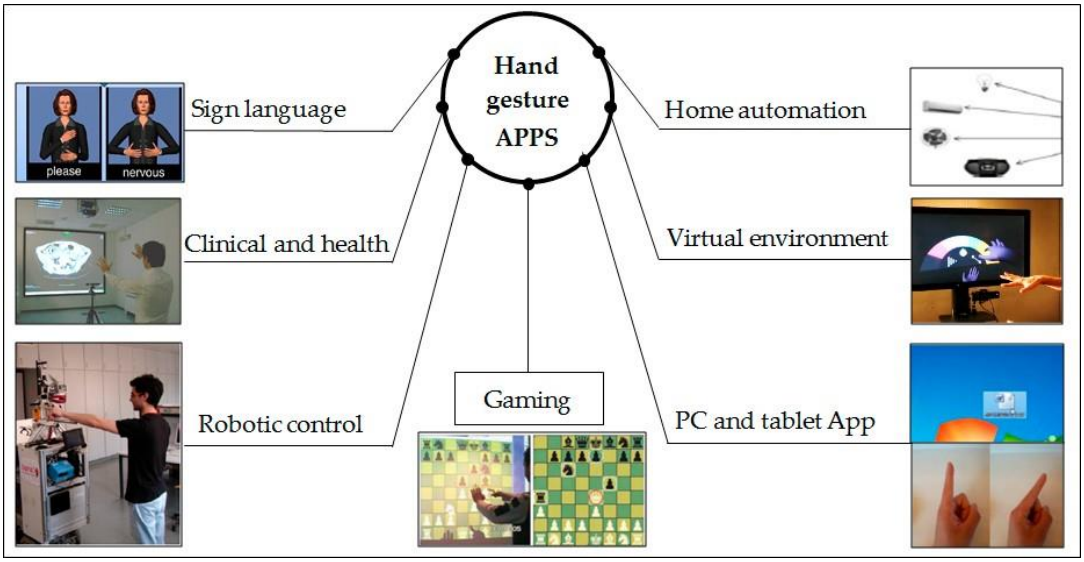


Figure 13. Most common application area of hand gesture interaction system (the image of Figure 13 is adapted from [12,14,42,76,83,98,99]).

Clinical and Health

During clinical operations, a surgeon may need details about the patient's entire body structure or a detailed organ model in order to shorten the operating time or increase the accuracy of the result. This is achieved by using a medical imaging system such as MRI, CT or X-ray system [10,99], which collects data from the patient's body and

displays them on the screen as a detailed image. The surgeon can facilitate interaction with the viewed images by performing hand gestures in front of the camera using a computer vision technique. These gestures can enable some operations such as zooming, rotating, image cropping and going to the next or previous slide without using any peripheral device such as a mouse, keyboard or touch screen. Any additional equipment requires sterilization, which can be difficult in the case of keyboards and touch screen. In addition, hand gestures can be used for assistive purpose such as wheelchair control [43].

Sign Language Recognition

Sign language is an alternative method used by people who are unable to communicate with others by speech. It consists of a set of gestures wherein every gesture represents one letter, number or expression. Many research papers have proposed recognition of sign language for deaf-mute people, using a glove-attached sensor worn on the hand that gives responses according to hand movement. Alternatively, it may involve uncovered hand interaction with the camera, using computer vision techniques to identify the gesture. For both approaches mentioned above, the dataset used for classification of gestures matches a real-time gesture made by the user [11,42,50].

Robot Control

Robot technology is used in many application fields such as industry, assistive services [100], stores, sports and entertainment. Robotic control systems use machine learning techniques, artificial intelligence and complex algorithms to execute a specific task, which lets the robotic system, interact naturally with the environment and make an independent decision. Some research proposes computer vision technology with a robot to build assistive systems for elderly people. Other research uses computer vision to enable a robot to ask a human for a proper path inside a specific building [12].

Virtual Environment

Virtual environments are based on a 3D model that needs a 3D gesture recognition system in order to interact in real time as a HCI. These gestures may be used for modification and viewing or for recreational purposes, such as playing a virtual piano. The gesture recognition system utilizes a dataset to match it with an acquired gesture in real time [13,78,83].

Home Automation

Hand gestures can be used efficiently for home automation. Shaking a hand or performing some gesture can easily enable control of lighting, fans, television, radio, etc. They can be used to improve older people's quality of life [14].

Personal Computer and Tablet

Hand gestures can be used as an alternative input device that enables interaction with a computer without a mouse or keyboard, such as dragging, dropping and moving files through the desktop environment, as well as cut and paste operations [19,69,76]. Moreover, they can be used to control slide show presentations [15]. In addition, they are used with a tablet to permit deaf-mute people to interact with other people by moving their hand in front of tablet's camera. This requires the installation of an application that translates sign language to text, which is displayed on the screen. This is analogous to the conversion of acquired voice to text.

Gestures for Gaming

The best example of gesture interaction for gaming purposes is the Microsoft Kinect Xbox, which has a camera placed over the screen and connects with the Xbox device through the cable port. The user can interact with the game by using hand motions and body movements that are tracked by the Kinect camera sensor [16,98].

3. Research Gaps and Challenges

From the previous sections, it is easy to identify the research gap,

since most research studies focus on computer applications, sign language and interaction with a 3D object through a virtual environment. However, many research papers deal with enhancing frameworks for hand gesture recognition or developing new algorithms rather than executing a practical application with regard to health care. The biggest challenge encountered by the researcher is in designing a robust framework that overcomes the most common issues with fewer limitations and gives an accurate and reliable result. Most proposed hand gesture systems can be divided into two categories of computer vision techniques. First, a simple approach is to use image processing techniques via Open-NI library or OpenCV library and possibly other tools to provide interaction in real time, which considers time consumption because of real-time processing. This has some limitations, such as background issues, illumination variation, distance limit and multi-object or multi-gesture problems. A second approach uses dataset gestures to match against the input gesture, where considerably more complex patterns require complex algorithm. Deep learning technique and artificial intelligence techniques to match the interaction gesture in real time with dataset gestures already containing specific postures or gestures.

CONCLUSION

The computational models, which were implemented in this project, were chosen after extensive research, and the successful testing results confirm that the choices made by the researcher were reliable. The system with manual face detection and automatic face recognition did not have a recognition accuracy over 90%, due to the limited number of Eigenfaces that were used for the PCA transform. This system was tested under very robust conditions in this experimental study and it is envisaged that real-world performance will be far more accurate. The fully automated frontal view face detection system displayed virtually perfect accuracy and in the researcher's opinion further work need not be conducted in this area.

The fully automated face detection and recognition system was not robust enough to achieve a high recognition accuracy. The only reason for this was the face recognition subsystem did not display even a slight degree of invariance to scale, rotation or shift errors of the segmented face image. However, if some sort of further processing, such as an eye detection technique, was implemented to further normalise the segmented face image, performance will increase to levels comparable to the manual face detection and recognition system. Implementing an eye detection technique would be a minor extension to the implemented system and would not require a great deal of additional research. All other implemented systems displayed commendable results and reflect well on the deformable template and Principal Component Analysis strategies.The most suitable real-world applications for face detection

and recognition systems are for mugshot matching and surveillance. There are better techniques such as iris or retina recognition and face recognition using the thermal spectrum for user access and user verification applications since these need a very high degree of accuracy. The real-time automated pose invariant face detection and recognition system proposed would be ideal for crowd surveillance applications. If such a system were widely implemented its potential for locating and tracking suspects for law enforcement agencies is immense.

The implemented fully automated face detection and recognition system (with an eye detection system) could be used for simple surveillance applications such as ATM user security, while the implemented manual face detection and automated recognition system is ideal of mugshot matching. Since controlled conditions are present when mugshots are gathered, the frontal view face recognition scheme should display a recognition accuracy far better than the results, which were obtained in this study, which was conducted under adverse conditions.

The results also showed that the gesture recognition application was quite robust for static images. However, the video version was enormously affected by the amount of illumination, such that is was necessary to check and adjust the HSV values for skin colour when starting the program to get the proper output. Sometimes the adjustment was difficult to do because of the lighting conditions and the amount of objects in the background.

The application was very susceptible to noise on the video stream. Slight hand movements could affect gesture recognition.

Nevertheless, if the hand is steady enough for long enough, the program outputs the correct command.

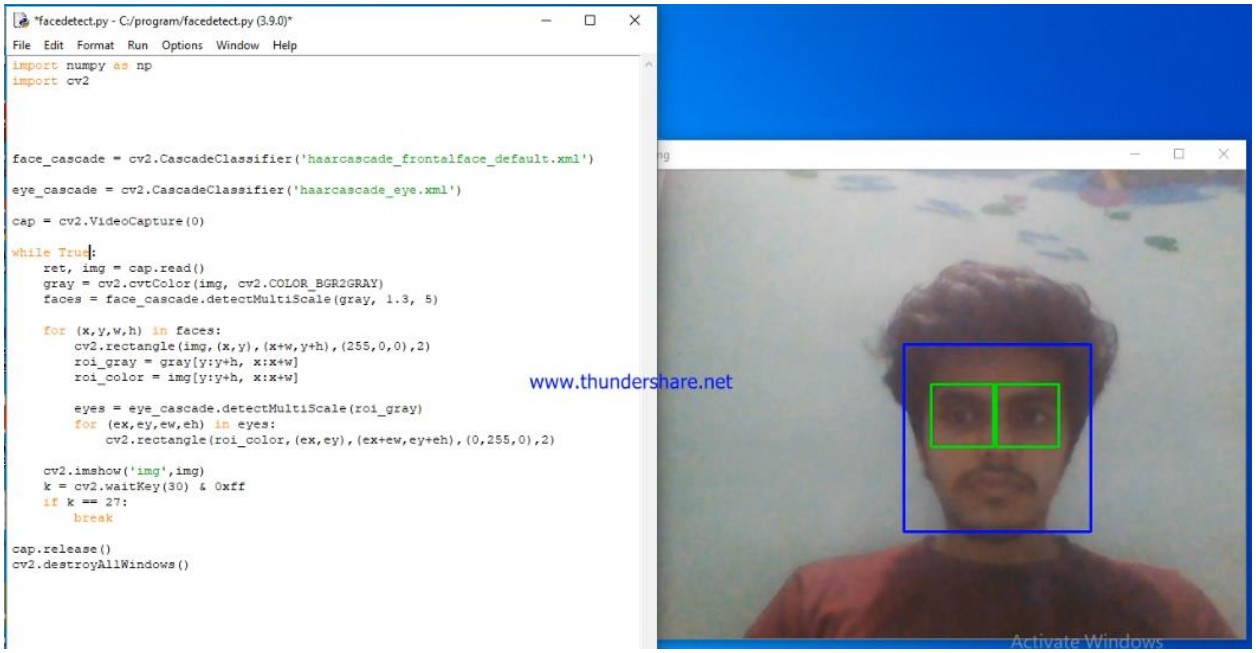
It was also observed that while the program was executing there were memory leaks. Attempts to remedy the problem were made by using the OpenCV functions to release memory. Despite this, the leaks continued. Perhaps the leaks were due to the implementation of OpenCV functions for the sequences behind the scenes.

For integrating the program with the robot in the future, it would be necessarily to consider other output such as speed or velocity as part of the navigational control commands.

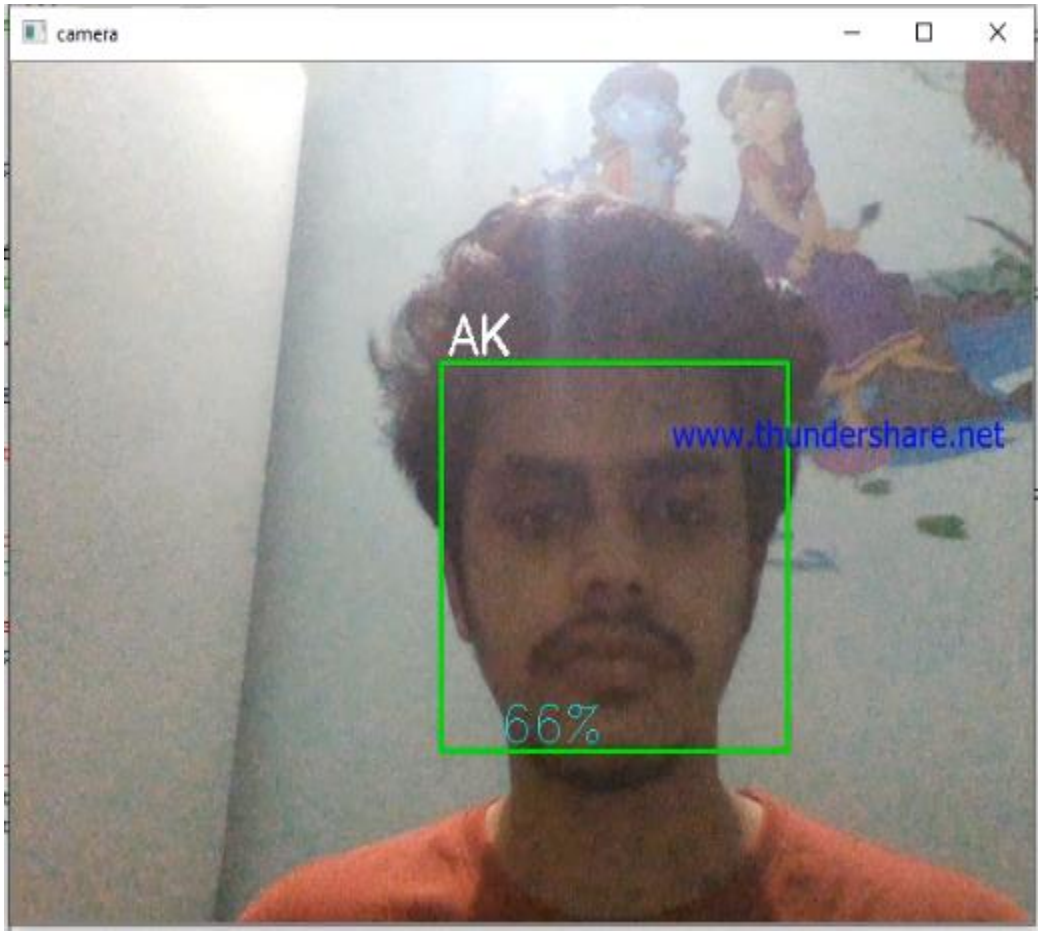
Based on the results, a computer vision application could detect and recognize simple hand gestures for robot navigation using simple heuristic rules. While the use of moment invariants was not considered suitable because the same gestures could be used pointing in opposite directions, other learning algorithms like AdaBoost could be explored to make the program more robust and less affected by extraneous objects and noise.

RESULT:

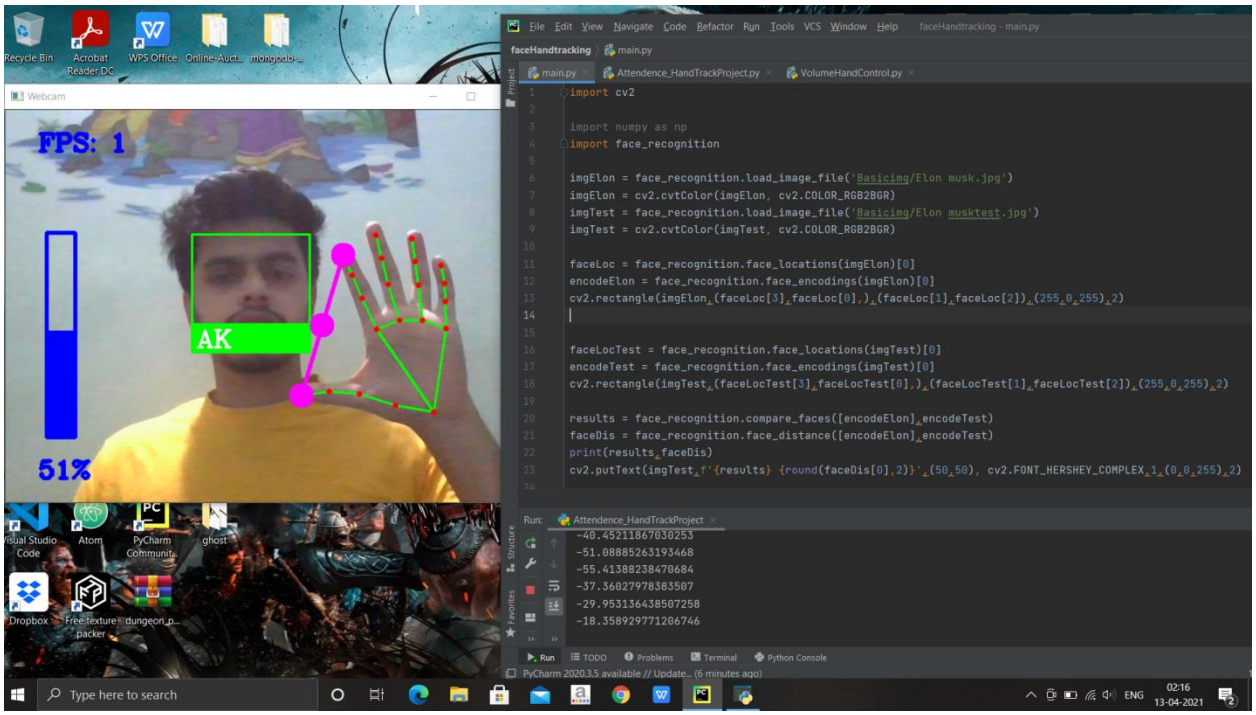
Face Detection



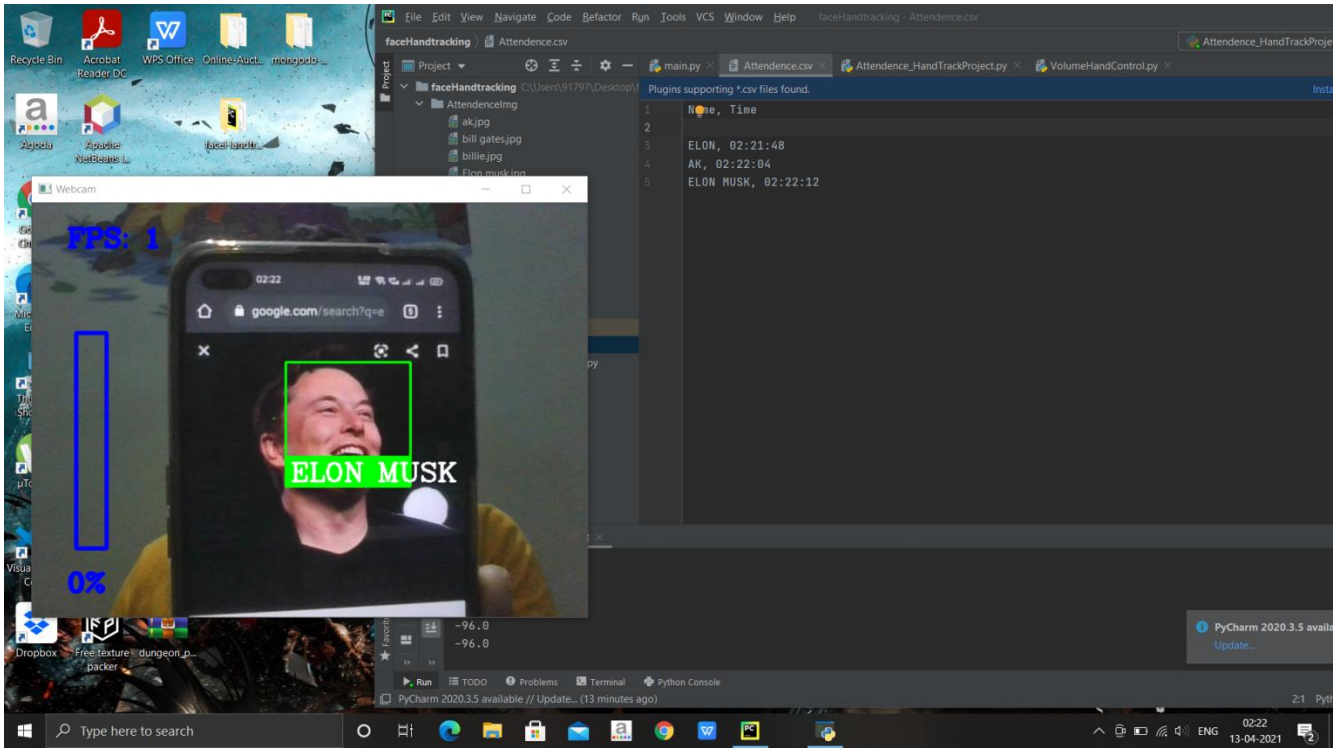
Face Recognition:



Face Recognition and HandTracking:



Attendence:


Webcam
FPS: 1
0%
ELON MUSK
None, Time
ELON, 02:21:48
AK, 02:22:04
ELON MUSK, 02:22:12

HandTracking Module:

```
import numpy as np

import cv2

import mediapipe as mp

import time


class handDetector():

   def __init__(self, mode= False, maxHands = 2, detectionCon = 0.5, trackCon = 0.5):

      self.mode = mode

      self.maxHands = maxHands

      self.detectionCon = detectionCon

      self.trackCon = trackCon


      self.mpHands = mp.solutions.hands

      self.hands = self.mpHands.Hands(self.mode, self.maxHands, self.detectionCon, self.trackCon )

      self.mpDraw = mp.solutions.drawing_utils


   def findHands(self, img, draw=True):


      imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

      self.results = self.hands.process(imgRGB)

      #print(results.multi_hand_landmarks)


      if self.results.multi_hand_landmarks:

         for handLms in self.results.multi_hand_landmarks:

            if draw:

               self.mpDraw.draw_landmarks(img, handLms, self.mpHands.HAND_CONNECTIONS)
```

```
        return img


    def findPosition(self, img, handNo=0, draw=True):


        lmList = []
        if self.results.multi_hand_landmarks:
            myHand = self.results.multi_hand_landmarks[handNo]


            for id, lm in enumerate(myHand.landmark):
                #print(id,lm)
                h, w, c= img.shape
                cx, cy = int(lm.x*w), int(lm.y*h)
                #print(id, cx, cy)
                lmList.append([id, cx, cy])
                if draw:
                    cv2.circle(img, (cx,cy), 5, (255,0,255), cv2.FILLED)


        return lmList


def main():
    pTime = 0
    cTime = 0
    cap = cv2.VideoCapture(0)
    detector = handDetector()


    while True:
        success, img = cap.read()
```

```
        img = detector.findHands(img)

        lmList = detector.findPosition(img)

        if len(lmList) !=0:

            print(lmList[4])


        cTime = time.time()

        fps = 1 / (cTime - pTime)

        pTime = cTime


        cv2.putText(img, str(int(fps)), (10, 70), cv2.FONT_HERSHEY_PLAIN, 3, (255, 0, 255), 3)


        cv2.imshow('Image', img)

        cv2.waitKey(1)
if __name__ == "__main__":

    main()
```

Main:

```
import cv2

import numpy as np

import face_recognition


imgElon = face_recognition.load_image_file('Basicimg/Elon musk.jpg')

imgElon = cv2.cvtColor(imgElon, cv2.COLOR_RGB2BGR)

imgTest = face_recognition.load_image_file('Basicimg/Elon musktest.jpg')

imgTest = cv2.cvtColor(imgTest, cv2.COLOR_RGB2BGR)


faceLoc = face_recognition.face_locations(imgElon)[0]

encodeElon = face_recognition.face_encodings(imgElon)[0]

cv2.rectangle(imgElon,(faceLoc[3],faceLoc[0],),(faceLoc[1],faceLoc[2]),(255,0,255),2)



faceLocTest = face_recognition.face_locations(imgTest)[0]

encodeTest = face_recognition.face_encodings(imgTest)[0]

cv2.rectangle(imgTest,(faceLocTest[3],faceLocTest[0],),(faceLocTest[1],faceLocTest[2]),(255,0,255),2)


results = face_recognition.compare_faces([encodeElon],encodeTest)

faceDis = face_recognition.face_distance([encodeElon],encodeTest)

print(results,faceDis)

cv2.putText(imgTest,f'{results} {round(faceDis[0],2)}',(50,50), cv2.FONT_HERSHEY_COMPLEX,1,(0,0,255),2)


cv2.imshow('Elon Musk', imgElon)

cv2.imshow('Elon test', imgTest)

cv2.waitKey(0)
```

VolumeControl through HandTracking:

```
import cv2

import time

import numpy as np

import HandTrackinModule as htm

import math

from ctypes import cast, POINTER

from comtypes import CLSCTX_ALL

from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume


#########

wCam, hCam = 640, 480

########


cap = cv2.VideoCapture(0)

cap.set(3, wCam)

cap.set(4, hCam)

pTime = 0

detector = htm.handDetector(detectionCon=0.5)

devices = AudioUtilities.GetSpeakers()

interface = devices.Activate(

   IAudioEndpointVolume._iid_, CLSCTX_ALL, None)

volume = cast(interface, POINTER(IAudioEndpointVolume))

#volume.GetMute()

#volume.GetMasterVolumeLevel()


volRange = volume.GetVolumeRange()
```

```
minVol = volRange[0]
maxVol = volRange[1]
vol = 0
volBar = 400
volPer = 0
while True:
    success, img = cap.read()
    img = detector.findHands(img)
    lmList = detector.findPosition(img, draw=False)
    if len(lmList) != 0:
        #print(lmList[4], lmList[8])

        x1, y1 = lmList[4][1], lmList[4][2]
        x2, y2 = lmList[8][1], lmList[8][2]
        cx, cy = (x1+x2)//2, (y1+y2)//2

        cv2.circle(img, (x1, y1), 15, (255, 0, 255), cv2.FILLED)
        cv2.circle(img, (x2, y2), 15, (255, 0, 255), cv2.FILLED)
        cv2.line(img, (x1, y1), (x2, y2), (255, 0, 255), 3)
        cv2.circle(img, (cx, cy), 15, (255, 0, 255), cv2.FILLED)

        length = math.hypot(x2-x1, y2-y1)
        #print(length)

        # Hand range 50 - 300
        # volume Range -65 - 0
```

```
    vol = np.interp(length, [50, 210], [minVol, maxVol])
    volBar = np.interp(length, [50, 300], [400, 150])
    volPer = np.interp(length, [50, 300], [0, 100])
    print(vol)
    volume.SetMasterVolumeLevel(vol, None)

    if length < 50:
        cv2.circle(img, (cx, cy), 15, (0, 255, 0), cv2.FILLED)

cv2.rectangle(img, (50, 150), (85, 400), (255, 0, 0), 3)
cv2.rectangle(img, (50, int(volBar)), (85, 400), (255, 0, 0), cv2.FILLED)
cv2.putText(img, f'{int(volPer)}%', (40, 450), cv2.FONT_HERSHEY_COMPLEX, 1, (255, 0, 0), 3)

cTime = time.time()
fps = 1/(cTime-pTime)
pTime = cTime

cv2.putText(img, f'FPS: {int(fps)}', (40, 50), cv2.FONT_HERSHEY_COMPLEX, 1, (255, 0, 0), 3)

cv2.imshow("Img", img)
cv2.waitKey(1)
```

Main_Hand.py:

```
import numpy as np

import cv2

import mediapipe as mp

import time

cap = cv2.VideoCapture(0)

mpHands = mp.solutions.hands

hands = mpHands.Hands()

mpDraw = mp.solutions.drawing_utils

pTime = 0

cTime = 0

while True:

   success, img = cap.read()

   imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

   results = hands.process(imgRGB)

   #print(results.multi_hand_landmarks)

   if results.multi_hand_landmarks:

      for handLms in results.multi_hand_landmarks:

         for id, lm in enumerate(handLms.landmark):

            print(id,lm)

            h, w, c= img.shape

            cx, cy = int(lm.x*w), int(lm.y*h)

            print(id, cx, cy)

            if id == 0:

               cv2.circle(img, (cx,cy), 5, (255,0,255), cv2.FILLED)

         mpDraw.draw_landmarks(img, handLms, mpHands.HAND_CONNECTIONS)
```

```
cTime = time.time()

fps = 1/(cTime-pTime)

pTime = cTime

cv2.putText(img, str(int(fps)),(10,70), cv2.FONT_HERSHEY_PLAIN, 3, (255,0,255), 3)


cv2.imshow('Image', img)

cv2.waitKey(1)
```

FaceRecognition Attendence_VolumeControl:

```
import cv2

import dlib

import numpy as np

import face_recognition

import os

from datetime import datetime

import cv2

import time

import numpy as np

import HandTrackinModule as htm

import math

from ctypes import cast, POINTER

from comtypes import CLSCTX_ALL

from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume

pTime = 0

detector = htm.handDetector(detectionCon=0.5)

devices = AudioUtilities.GetSpeakers()

interface = devices.Activate(

   IAudioEndpointVolume._iid_, CLSCTX_ALL, None)

volume = cast(interface, POINTER(IAudioEndpointVolume))

#volume.GetMute()

#volume.GetMasterVolumeLevel()

volRange = volume.GetVolumeRange()

minVol = volRange[0]

maxVol = volRange[1]

vol = 0
```

```
volBar = 400

volPer = 0


path = 'AttendenceImg'

images = []

classNames = []

myList = os.listdir(path)

print(myList)

for cl in myList:

   curImg = cv2.imread(f'{path}/{cl}')

   images.append(curImg)

   classNames.append(os.path.splitext(cl)[0])

print(classNames)


def findEncodings(images):

   encodeList = []

   for img in images:

      img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

      encode = face_recognition.face_encodings(img)[0]

      encodeList.append(encode)

   return encodeList


def markAttendence(name):

   with open('Attendence.csv','r+')as f:

      myDataList= f.readlines()

      nameList = []

      for line in myDataList:

         entry = line.split(', ')
```

```
            nameList.append(entry[0])
        if name not in nameList:
            now = datetime.now()
            dtString = now.strftime('%H:%M:%S')
            f.writelines(f'\n{name}, {dtString}')
markAttendence('ELON')
encodeListKnown = findEncodings(images)
print('encoding complete')


cap = cv2.VideoCapture(0)
while True:
    success, img = cap.read()
    imgS =cv2.resize(img,(0,0),None,0.25,0.25)
    imgS = cv2.cvtColor(imgS, cv2.COLOR_BGR2RGB)
    faceCurFrame = face_recognition.face_locations(imgS)
    encodeCurFrame = face_recognition.face_encodings(imgS,faceCurFrame)
    for encodeFace,faceLoc in zip(encodeCurFrame,faceCurFrame):
        matches = face_recognition.compare_faces(encodeListKnown,encodeFace)
        faceDis = face_recognition.face_distance(encodeListKnown,encodeFace)
        #print(faceDis)
        matchIndex = np.argmin(faceDis)


        if matches[matchIndex]:
            name = classNames[matchIndex].upper()
            #print(name)
            y1,x2,y2,x1=faceLoc
            y1, x2, y2, x1= y1*4,x2*4,y2*4,x1*4
            cv2.rectangle(img,(x1,y1),(x2,y2),(0,255,0),2)
```

```
        cv2.rectangle(img,(x1,y2-35),(x2,y2),(0,255,0),cv2.FILLED)
        cv2.putText(img,name,(x1+6,y2-6),cv2.FONT_HERSHEY_COMPLEX,1,(255,255,255),2)
        markAttendence(name)

img = detector.findHands(img)
lmList = detector.findPosition(img, draw=False)
if len(lmList) != 0:
    # print(lmList[4], lmList[8])

    x1, y1 = lmList[4][1], lmList[4][2]
    x2, y2 = lmList[8][1], lmList[8][2]
    cx, cy = (x1 + x2) // 2, (y1 + y2) // 2

    cv2.circle(img, (x1, y1), 15, (255, 0, 255), cv2.FILLED)
    cv2.circle(img, (x2, y2), 15, (255, 0, 255), cv2.FILLED)
    cv2.line(img, (x1, y1), (x2, y2), (255, 0, 255), 3)
    cv2.circle(img, (cx, cy), 15, (255, 0, 255), cv2.FILLED)

    length = math.hypot(x2 - x1, y2 - y1)
    # print(length)

    # Hand range 50 - 300
    # volume Range -65 - 0

    vol = np.interp(length, [50, 210], [minVol, maxVol])
    volBar = np.interp(length, [50, 300], [400, 150])
    volPer = np.interp(length, [50, 300], [0, 100])
    print(vol)
```

```
    volume.SetMasterVolumeLevel(vol, None)


    if length < 50:
        cv2.circle(img, (cx, cy), 15, (0, 255, 0), cv2.FILLED)


cv2.rectangle(img, (50, 150), (85, 400), (255, 0, 0), 3)
cv2.rectangle(img, (50, int(volBar)), (85, 400), (255, 0, 0), cv2.FILLED)
cv2.putText(img, f'{int(volPer)}%', (40, 450), cv2.FONT_HERSHEY_COMPLEX, 1, (255, 0, 0), 3)


cTime = time.time()
fps = 1 / (cTime - pTime)
pTime = cTime
cv2.putText(img, f'FPS: {int(fps)}', (40, 50), cv2.FONT_HERSHEY_COMPLEX, 1, (255, 0, 0), 3)



cv2.imshow('Webcam',img)
cv2.waitKey(1)
```