Human Computer Interaction and Painting through gestures using ML Techniques

A Project Report Submitted to

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY

In partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

SUBMITTED BY

Student name	(Roll no)
R.RITHWIK	15BD1A056N
SARAFF VIVEK DATHA	15BD1A056T
SATHWIK PITTALA	15BD1A056V
S R TARUN	15P61A05L0

Under the Guidance of

(Mr. Ajeet Jain Asst/ Associate Professor, CSE Dept.)

DEPARTMENT OF COMPUTER SCIENCE ANDENGINEERING
KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
(Approved by AICTE, Affiliated to JNTUH)
Narayanaguda, Hyderabad

Academic Year: 2018-19

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that this a bonafide record of the project report titled **"Human Computer Interaction and painting through gestures using ML techniques"** which is being presented as the Major Project report by

1. Student Name	Roll Number
R.RITHWIK	15BD1A056N
SARAFF VIVEK DATHA	15BD1A056T
SATHWIK PITTALA	15BD1A056V
S R TARUN	15P61A05L0

In partial fulfilment for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** affiliated to the **Jawaharlal Nehru Technology University, Hyderabad.**

Internal Guide

(Mr. Ajeet Jain)

Head of the Department

External Examiner

DECLARATION

I hereby declare that the results embodied in this dissertation entitled **" Human Computer Interaction and painting through gestures using ML techniques"** has been carried out by us together during the academic year 2018-2019 as a partial fulfilment of the award of the B. Tech degree in Computer Science and Engineering from JNTUH. We have not submitted this report to any other university or organization for award of any other degree.

Student Name	(Roll No)
R.RITHWIK	**15BD1A056N**
SARAFF VIVEK DATHA	**15BD1A056T**
SATHWIK PITTALA	**15BD1A056V**
S R TARUN	**15P61A05L0**

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose constant encouragement and guidance have been a source of inspiration throughout the course of the project work. I am glad to have the opportunity of expressing our gratitude to all of them.

I am thankful to Dr.J.V.S SRINIVAS, Professor and Head of the Department of Computer Science and Engineering, Keshav Memorial Institute of Technology. I thank him for his support and guidance throughout the project. . He is a source of inspiration for innovative ideas and his kind support is well known to all his students and colleagues.

I am thankful to our Guide Mr. Ajeet Jain, Asst.Professor/Asso.Professor of Computer Science and Engineering Dept., Keshav Memorial Institute of Technology, for his support and guidance throughout the project.

I would like to express our sincere gratitude to Technical faculty, Mr.P V N Balarama Murthy, Asst.Professor and Project Co-ordinators for their support, guidelines and encouragement.

I also express my sincere gratitude to the Management and Principal of Keshav Memorial Institute of Technology for their encouragement, facilities provided and support.

Finally, I would like to make a special mention of all our family members and friends who helped us for the successful completion of the project report.

Student Name	(Roll No)
R.RITHWIK	**15BD1A056N**
SARAFF VIVEK DATHA	**15BD1A056T**
SATHWIK PITTALA	**15BD1A056V**
S R TARUN	**15P61A05L0**

ABSTRACT

This HCI (Human-Computer Interaction) application will allow you to control your mouse with your facial movements. This project works with predicting the facial landmarks of a given face as well as capturing hand gestures.

We can accomplish a lot of things using these landmarks and gestures. From detecting eye-blinks in a video to predicting emotions of the subject. The applications, outcomes, and possibilities of facial landmarks are immense and intriguing.

We use Dlib's prebuilt model, which is essentially an implementation of Haar Cascade Algorithm for object recognition, which not only does a fast face-detection but also allows us to accurately predict 68 2D facial landmarks which are found to be very handy in multiple occasions.

This project also uses the approach of treating hand gestures as an input mechanism to do some Realtime painting or writing without touching your device.

Tools and frameworks such as OpenCV, NumPy, pyautogui can be used for recognition, Data Processing and automation respectively.

INDEX

LIST OF FIGURES

CHAPTER-1

1. INTRODUCTION

Human Computer Interaction(HCI) is the academic discipline that most of us think of as UI design. It focuses on the way that human beings and computers interact to ever increasing levels of both complexity and simplicity. With the increasing pace of technological advancements and exponential growth of computer users world wide the interaction between a human and a computer needs to be simplified much further so that a wide range of people can use them to accomplish complex tasks.

We focused on using multiple technologies and incorporate it in an efficient way such that a smooth flow of Human computer interaction occurs. Our application can be considered as combination of 3 major parts Object Detection, Face Detection and Automation.

Object Detection:

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.

Efficient and accurate object detection has been an important topic in the advancement of computer vision systems. With the advent of Computer vision techniques, the accuracy for object detection has increased drastically. In this project we use techniques likes masking, colour segmentation, Finding our region of interest(ROI) by detecting contours.

Initially we use Once we start reading the webcam feed, we constantly look for a blue color object in the frames with the help of inbuilt method and use the Upper and lower bound of the colour variables. Once we find the contour, we do a series of image operations and make it smooth. They just makes our lives easier. If you want to know more about these operations — erode, morph and dilate.

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

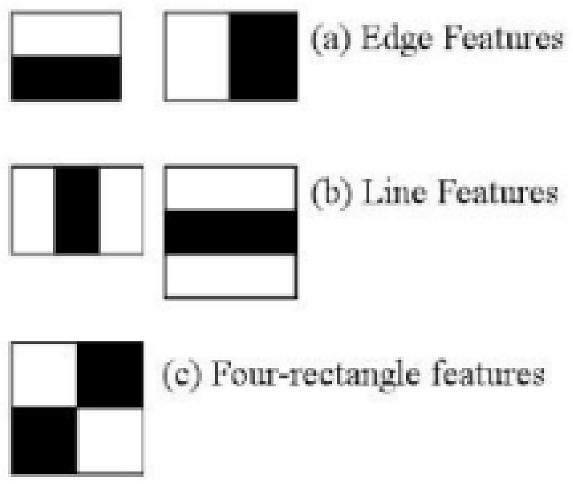


Fig: Edge Features

Face Detection:

This project works with predicting the facial landmarks of a given face from the continuous video stream. We can accomplish a lot of things using these landmarks and gestures. From detecting eye-blinks in a video to predicting emotions of the subject. The applications, outcomes, and possibilities of facial landmarks are immense and intriguing.

We can use models, which are essentially an implementation of Haar Cascade Algorithm for object recognition or we build our model to detect faces and facial landmarks, This not only does a fast face-detection but also allows us to accurately predict 68 2D facial landmarks which are found to be very handy in multiple occasions.

Face detection is also a first step in Human Computer Interaction (HCI) systems. E.g. expression recognition. Unlike traditional HCI device, keyboard, mouse and display, it provides more effective methods to increase user experience with computer used. As a result, it speeds up human’s work. It’s conveyed information from physical world into virtual world to control the computer system.

Automation:

After detecting the faces and objects from the video stream we then make use of some statistical calculations to detect the facial expressions or facial gestures. This can be done using Eye aspect ratio for calculating eyes. We use the python packages for this purpose.

Automation is the creation of technology and its application in order to control and monitor the production and delivery of various goods and services. It performs tasks that were previously performed by humans.Automation is being used in a number of areas such as manufacturing, transport, utilities, defense, facilities,operations and lately, information technology.

PyAutoGUI can simulate moving the mouse, clicking the mouse, dragging with the mouse, pressing keys, pressing and holding keys, and pressing keyboard hotkey combinations.

1.2 Problem with the Existing System

Initially one had to personally use different kinds of tools and applications to interact with the computer. Though gesture recognition is a field which has shown tremendous capability in the last few years , the usage in every day interaction has been limited to a very few functionalities. The reason for this might be poor implementation and complex controls. This has also made the utilization of computers almost impossible for the people without a functional Hand. This project can also be used in Teaching purposes where the professor can explain to the students by writing or drawing the scenarios without even moving from his position

In order to avoid these complexities and issues we are using facial recognition and machine learning techniques. For this we only need an camera to capture the visual frames.

1.3 Proposed System

We propose to use pythons existing package Dlib which is essentially a pre implementation of Haar Cascade Algorithm for object recognition or to build our Facial Landmark prediction model, The model, which not only does a fast face-detection but also allows us to accurately predict 68 2D facial landmarks which are found to be very handy.

This project also uses the approach of Image masking *which is a process of hiding some portions* of an image and to reveal some portions. It is a non-destructive process of image editing. Very often, it is efficient and more creative way of detecting an object and thus track it in a continuous video stream.

Later the predicted facial landmarks can be used to analyze the facial gestures with techniques such as Eye aspect ratio and point movements. The detected object can be tracked along its motion and can be considered as a paintbrush to paint along the path.

1.4 Scope of the Project

Thus this project helps you in a variety of ways right from controlling your personal computer with your facial gestures(Highly useful for a person with nonfunctional hand) to using complex applications such as paint with your hand gestures.

This can be further extended and used as an application where it helps in writing content by using hand gestures. This can be used in smart classes and smart teaching applications.

1.5 Architecture Diagram

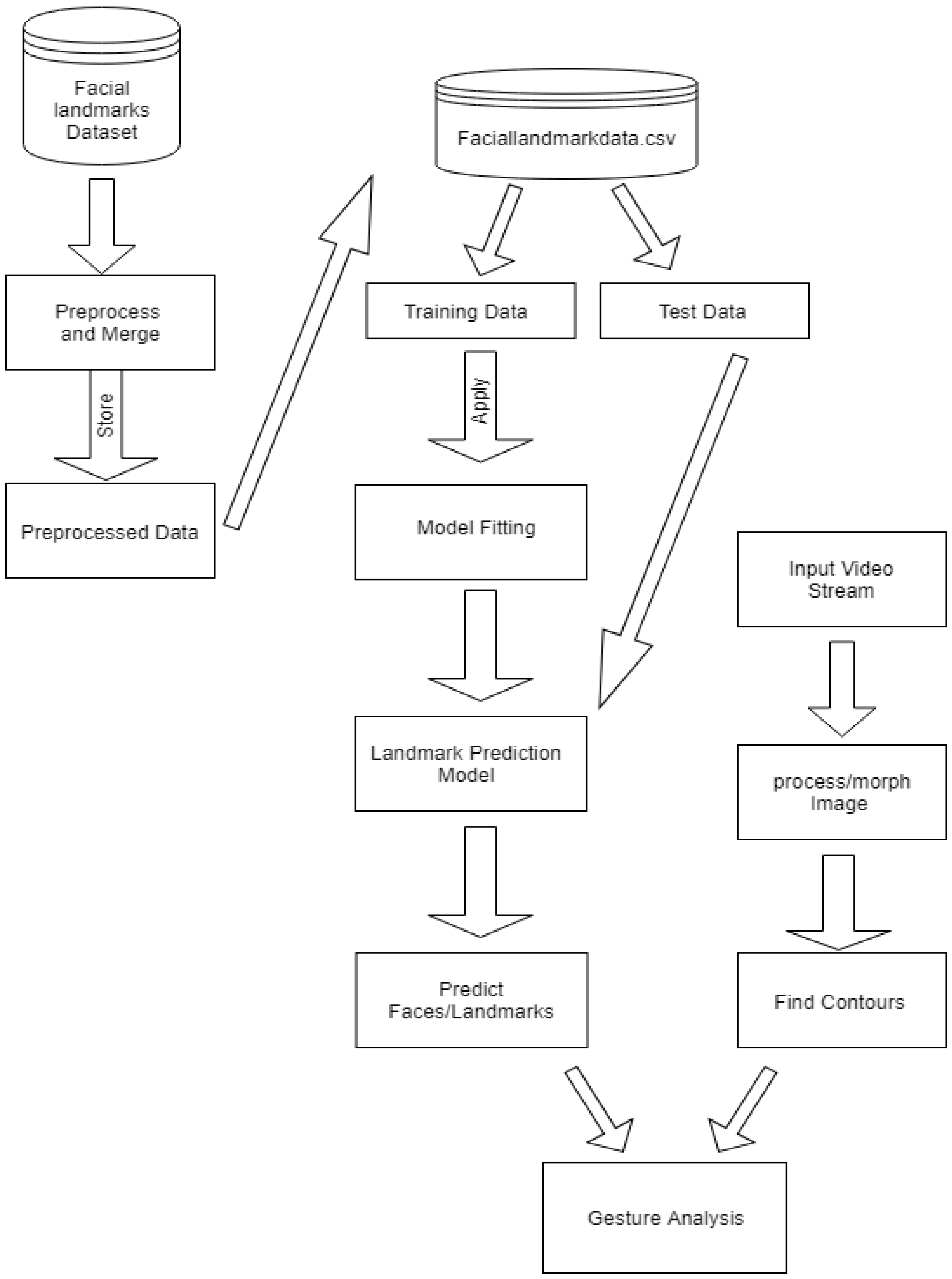
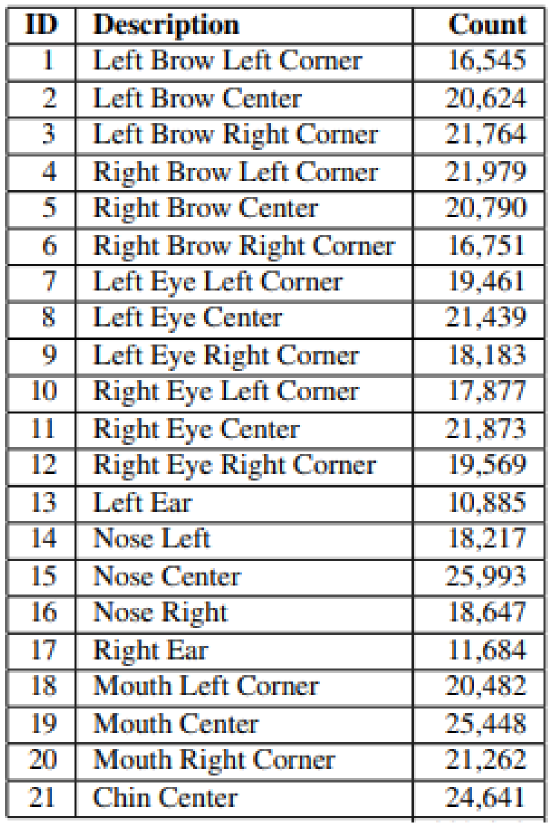


Figure 1: Architecture Diagram

1.6 Data set description

Facial landmarks are standard reference points, such as the inner and outer corner of the eye fissure where the eyelids meet. The landmark positions of AFLW are defined on a rigid 3D face model. It uses a markup of 21 reference landmarks mainly located in the area between eyebrows and chin. Starting at the forehead three landmarks are located at each eyebrow, on the leftmost, rightmost and medial point. Each eye area is covered by further three landmarks. The inner and outer corner of the eye fissure where the eyelids meet and the medial point. On the external nose we specified the left and right point of attachment of the nose cavity with the face and the tip of the nose. On the external ear we mark the lowest point of attachment to the head. On the mouth and lips the landmarks are placed on the left and right intersection point of the lips and the mouth center as medial point. Finally, on the chin the lowest point on the lower border (gnathion) is selected. In the annotation process landmarks are marked upon visibility. So if a landmark is not visible it is simply not annotated. In total 230,189 landmarks have been annotated so far. For individual landmarks the number of annotations ranges from 6,203 (left ear) to 15,677 (nose center). Please see Table for detailed statistics.



ID	Description	Count
1	Left Brow Left Corner	16,545
2	Left Brow Center	20,624
3	Left Brow Right Corner	21,764
4	Right Brow Left Corner	21,979
5	Right Brow Center	20,790
6	Right Brow Right Corner	16,751
7	Left Eye Left Corner	19,461
8	Left Eye Center	21,439
9	Left Eye Right Corner	18,183
10	Right Eye Left Corner	17,877
11	Right Eye Center	21,873
12	Right Eye Right Corner	19,569
13	Left Ear	10,885
14	Nose Left	18,217
15	Nose Center	25,993
16	Nose Right	18,647
17	Right Ear	11,684
18	Mouth Left Corner	20,482
19	Mouth Center	25,448
20	Mouth Right Corner	21,262
21	Chin Center	24,641

Each of the above mentioned data elements are described using 2 columns in the dataset which describes the x and y coordinate of the element. For example Left Eye Centre is represented using 2 columns (LeftEyeCenter_X , LeftEyeCenter_Y).

CHAPTER-2

2. SOFTWARE REQUIREMENTS SPECIFICATION

What is SRS?

Software Requirement Specification (SRS) is the starting point of the software developing activity. As system grew more complex it became evident that the goal of the entire system cannot be easily comprehended. Hence the need for the requirement phase arose. The software project is initiated by the client needs. The SRS is the means of translating the ideas of the minds of clients (the input) into a formal document (the output of the requirement phase.)

The SRS phase consists of two basic activities:

1) Problem/Requirement Analysis:

The process is order and more nebulous of the two, deals with understand the problem, the goal and constraints.

2) Requirement Specification:

Here, the focus is on specifying what has been found giving analysis such as representation, specification languages and tools, and checking the specifications are addressed during this activity.

The Requirement phase terminates with the production of the validate SRS document. Producing the SRS document is the basic goal of this phase.

Role of SRS

The purpose of the Software Requirement Specification is to reduce the communication gap between the clients and the developers. Software Requirement Specification is the medium though which the client and user needs are accurately specified. It forms the basis of software development. A good SRS should satisfy all the parties involved in the system.

2.1 Requirements Specification Document

A Software Requirements Specification (SRS) is a document that describes the nature of a project, software or application. In simple words, SRS document is a manual of a project provided it is prepared before you kick-start a project/application. This document is also known by the names SRS report, software document. A software document is primarily prepared for a project, software or any kind of application.

There are a set of guidelines to be followed while preparing the software requirement specification document. This includes the purpose, scope, functional and nonfunctional requirements, software and hardware requirements of the project. In addition to this, it also contains the information about environmental conditions required, safety and security requirements, software quality attributes of the project etc.

The purpose of SRS (Software Requirement Specification) document is to describe the external behavior of the application developed or software. It defines the operations, performance and interfaces and quality assurance requirement of the application or software. The complete software requirements for the system are captured by the SRS.

This section introduces the requirement specification document for Storm Forecasting using Machine Learning which enlists functional as well as non-functional requirements.

2.1.1 Functional Requirements

For documenting the functional requirements, the set of functionalities supported by the system are to be specified. A function can be specified by identifying the state at which the data is to be input to the system, its input data domain, the output domain, and the type of processing to be carried on the input data to obtain the output data

Functional requirements define specific behavior or function of the application. Following are the functional requirements:

Training the machine using the given dataset

Detection of the face in the photograph using haar cascade algorithm

Analyzing the input image to be tested

Recognition of the face using LBPH algorithm

Displaying the detected face

2.1.2 Non-Functional Requirements

A non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. Especially these are the constraints the system must work within. Following are the non-functional requirements:

Performance:

The performance of the developed applications can be calculated by using following methods:

Measuring enables you to identify how the performance of your application stands in relation to your defined performance goals and helps you to identify the bottlenecks that affect your application performance. It helps you identify whether your application is moving toward or away from your performance goals. Defining what you will measure, that is, your metrics, and defining the objectives for each metric is a critical part of your testing plan.

Performance objectives include the following:

- Response time or latency
- Throughput
- Resource utilization

2.2 Software Requirements

2.2.1 Operating System	:	Windows 7/8/10
2.2.2 Platform	:	PyCharm Community Version
2.2.3 Programming Language	:	Python
2.2.4 External files	:	Dataset(images) & an input testing image

2.3 Hardware Requirements

2.3.1 Processor	:	Intel I3/I5
2.3.2 Hard Disk	:	64GB
2.3.3 RAM	:	4GB or more

CHAPTER-3

3. LITERATURE SURVEY

The survey done on HCI in the year 2003 contributed to bringing the topic of HCI as an issue on the agenda in many conferences and organizations. It reviewed 102 mobile HCI publications from 2000-02. Inspired by a similar study in Information Systems. Each paper was classified in terms of its research method and purpose, extracted from with supplementary input from.

TRENDS AND ASSUMPTIONS

The literature survey revealed a strong bias towards applied research for engineering and laboratory experiments for evaluation, as shown in figure 13. Put simply, mobile interaction design research in the early 2000s was dominated by building new systems in a trial-and-error manner, and evaluating them in laboratory settings – if evaluating them at all. There was very little going on in terms of trying to understand the phenomenon of mobility itself in relation to interaction design and technology use, and to use such insight when designing and building actual interactive systems. Nor was much attention given to the role of real world context in relation to understanding, building or evaluating interactive mobile systems. In essence this echoed a fundamental segregation between use- and technology-centeredness depending on whether the involved researchers were primarily interested in people or systems. On a more general level, it became apparent that methodology seemingly played a very small role. The approaches taken often remained unexplained, their suitability unchallenged, and their limitations and alternatives not discussed.

In 2010 the research field had developed in terms of approaches and focus. The second literature review took its offset in research papers on the topic of mobile HCI published in ten top outlets in 2009. These papers constitute all publications related to mobile human-computer interaction in 2009 in the top-level conference proceeding series and journals.

While other conferences and journals exist in which interesting research on mobile human-computer interaction is presented, few conferences and journals provided a solid and adequately representative base for this study given the number of publications on the topic and the general level of the reviewing processes for these conferences and journals..

3.1 Technologies Used

3.1.1 Python

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Importance of Python

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Features of Python

- **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** − Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.
- **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** − Python provides interfaces to all major commercial databases.
- **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** − Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below −

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.

- It provides very high-level dynamic data types and supports dynamic type checking.
- IT supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Libraries used in python:

- numpy - mainly useful for its N-dimensional array objects.
- pandas - Python data analysis library, including structures such as dataframes.
- matplotlib - 2D plotting library producing publication quality figures.
- scikit-learn - the machine learning algorithms used for data analysis and data mining tasks.

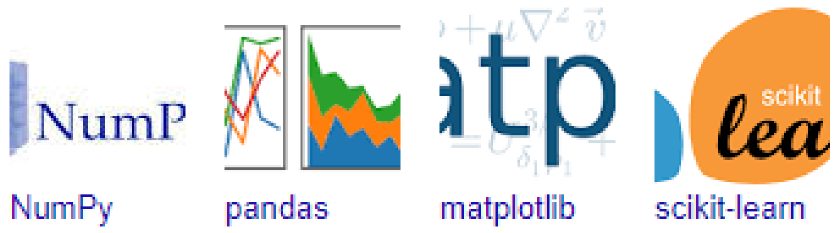


Figure 3: NumPy, Pandas, Matplotlib, Scikit-learn

3.1.2 OpenCV

OpenCV (Open source computer vision) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD licence.

OpenCV supports the deep learning frameworks TensorFlow, Torch/Pytorch and Caffe.

History of OpenCV

Officially launched in 1999, the OpenCV project was initially an Intel Research initiative to advance CPU-intensive applications, part of a series of projects including real-time ray tracing and 3D display walls. The main contributors to the project included a number of optimization experts in Intel Russia, as well as Intel's Performance Library Team. In the early days of OpenCV, the goals of the project were described as:

- Advance vision research by providing not only open but also optimized code for basic vision infrastructure. No more reinventing the wheel.
- Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.
- Advance vision-based commercial applications by making portable, performance-optimized code available for free – with a license that did not require code to be open or free itself.

The first alpha version of OpenCV was released to the public at the IEEE Conference on Computer Vision and Pattern Recognition in 2000, and five betas were released between 2001 and 2005. The first 1.0 version was released in 2006. A version 1.1 "pre-release" was released in October 2008.

The second major release of the OpenCV was in October 2009. OpenCV 2 includes major changes to the C++ interface, aiming at easier, more type-safe patterns, new functions, and better implementations for existing ones in terms of performance (especially on multi-core systems). Official releases now occur every six months[7] and development is now done by an independent Russian team supported by commercial corporations.

In August 2012, support for OpenCV was taken over by a non-profit foundation OpenCV.org, which maintains a developer and user site.

On May 2016, Intel signed an agreement to acquire Itseez, a leading developer of OpenCV.

Applications:

OpenCV's application areas include:

- 2D and 3D feature toolkits
- Egomotion estimation
- Facial recognition system
- Gesture recognition
- Human–computer interaction (HCI)
- Mobile robotics
- Motion understanding

- Object identification
- Segmentation and recognition
- Stereopsis stereo vision: depth perception from 2 cameras
- Structure from motion (SFM)
- Motion tracking
- Augmented reality

To support some of the above areas, OpenCV includes a statistical machine learning library that contains:

- Boosting
- Decision tree learning
- Gradient boosting trees
- Expectation-maximization algorithm
- k-nearest neighbor algorithm
- Naive Bayes classifier
- Artificial neural networks
- Random forest
- Support vector machine (SVM)
- Deep neural networks (DNN)

Programming Language

OpenCV is written in C++ and its primary interface is in C++, but it still retains a less comprehensive though extensive older C interface. There are bindings in Python, Java and MATLAB/OCTAVE. The API for these interfaces can be found in the online documentation. Wrappers in other languages such as C#, Perl, Ch, Haskell, and Ruby have been developed to encourage adoption by a wider audience.

Since version 3.4, OpenCV.js is a JavaScript binding for selected subset of OpenCV functions for the web platform.

All of the new developments and algorithms in OpenCV are now developed in the C++ interface.

Hardware acceleration

If the library finds Intel's Integrated Performance Primitives on the system, it will use these proprietary optimized routines to accelerate itself.

A CUDA-based GPU interface has been in progress since September 2010.

An OpenCL-based GPU interface has been in progress since October 2012, documentation for version 2.4.13.3 can be found at docs.opencv.org.

OS Support

OpenCV runs on the following desktop operating systems: Windows, Linux, macOS, FreeBSD, NetBSD, OpenBSD. OpenCV runs on the following mobile operating systems: Android, iOS, Maemo,BlackBerry 10. The user can get official releases from SourceForge or take the latest sources from GitHub. OpenCV uses CMake.

NumPy

NumPy is the fundamental package for all the scientific computing in Python. It is a Python library that provides all the fundamental or multidimensional array object, various derived objects in the (such asmasked in arrays and matrices), and an assortments for all fast operation on arrays,including mathematical operations, logical, shape manipulation, selecting, I/O, discrete Fourierstransforms, basic linear algebra, basic statistical operation, random simulations etc.

- NumPy arrays have a fixed size at creation itself, unlike the Python lists which can grow dynamically. Changing the size of an *ndarray* will create the new array and delete the existing.
- The elements in a NumPy array are required to be of the same data type, and thus will be the same size in the memory. The exception: one can have the arrays of Python, including NumPy objects, thereby allowing for arrays of various sized elements to be supported.
- NumPy arrays support advanced mathematical and other types of operations on a large number of data. Typically, such operations are executed more and more efficiently and with less code that is possible using Python’s built-in sequences.

A growing plethora of scientific and mathematical Python based packages are using NumPy arrays though these typically support Python-sequence input, these convert such input to NumPy arrays before processing, and they often output NumPy arrays only. In other words, in order to use much of the perhaps even most of the today's scientific and mathematical Python based software, just knowing how to use Python's built-in sequence types is insufficient and also one also needs to know how to use NumPy arrays

CHAPTER-4

4. SYSTEM DESIGN

4.1 Introduction to UML

The Unified Modeling Language allows the software engineer to express an analysis model using the modeling notation that is governed by a set of syntactic, semantic and pragmatic rules.

A UML system is represented using five different views that describe the system from distinctly different perspective. Each view is defined by a set of diagram, which is as follows:

1. User Model View

   i. This view represents the system from the users' perspective.
   ii. The analysis representation describes a usage scenario from the end-users' perspective.

2. Structural Model View

   i. In this model, the data and functionality are arrived from inside the system.
   ii. This model view models the static structures.

3. Behavioral Model View

It represents the dynamic of behavioral as parts of the system, depicting he interactions of collection between various structural elements described in the user model and structural model view.

4. Implementation Model View

In this view, the structural and behavioral as parts of the system are represented as they are to be built.

5. Environmental Model View

In this view, the structural and behavioral aspects of the environment in which the system is to be implemented are represented.

4.2 UML Diagrams

4.2.1 Use Case Diagram

To model a system, the most important aspect is to capture the dynamic behavior. To clarify a bit in details, dynamic behavior means the behavior of the system when it is running /operating.

So only static behavior is not sufficient to model a system rather dynamic behavior is more important than static behavior. In UML there are five diagrams available to model dynamic nature and use case diagram is one of them. Now as we have to discuss that the use case diagram is dynamic in nature there should be some internal or external factors for making the interaction.

These internal and external agents are known as actors. So use case diagrams are consists of actors, use cases and their relationships. The diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system. So to model the entire system numbers of use case diagrams are used.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So when a system is analyzed to gather its functionalities use cases are prepared and actors are identified.

In brief, the purposes of use case diagrams can be as follows:

a. Used to gather requirements of a system.

b. Used to get an outside view of a system.

c. Identify external and internal factors influencing the system.

d. Show the interacting among the requirements are actors.

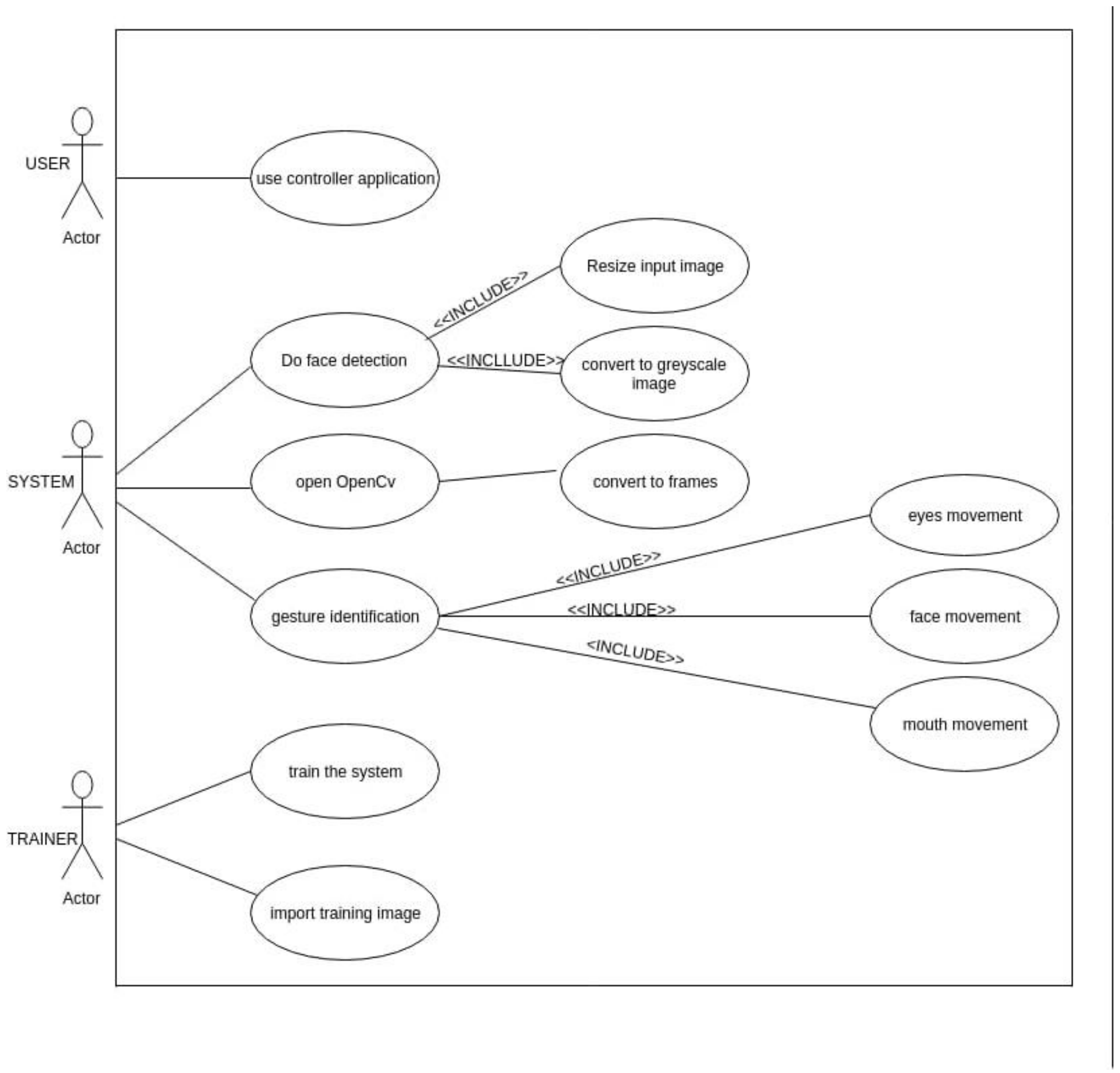


Figure 4: Use case Diagram for entire application functionality

4.2.2 Sequence Diagram

Sequence diagrams describe interactions among classes in terms of an exchange of messages over time. They're also called event diagrams. A sequence diagram is a good way to visualize and validate various runtime scenarios. These can help to predict how a system will behave and to discover responsibilities a class may need to have in the process of modeling a new system.

The aim of a sequence diagram is to define event sequences, which would have a desired outcome. The focus is more on the order in which messages occur than on the message per se. However, the majority of sequence diagrams will communicate what messages are sent and the order in which they tend to occur.

Basic Sequence Diagram Notations

- **Class Roles or Participants**

Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.

- **Activation or Execution Occurrence**

Activation boxes represent the time an object needs to complete a task. When an object is busy executing a process or waiting for a reply message, use a thin gray rectangle placed vertically on its lifeline.

- **Messages**

Messages are arrows that represent communication between objects. Use half-arrowed lines to represent asynchronous messages.

Asynchronous messages are sent from an object that will not wait for a response from the receiver before continuing its tasks.

- **Lifelines**

Lifelines are vertical dashed lines that indicate the object's presence over time.

- **Destroying Objects**

Objects can be terminated early using an arrow labeled "<< destroy >>" that points to an X. This object is removed from memory. When that object's lifeline ends, you can place an X at the end of its lifeline to denote a destruction occurrence.

- **Loops**

A repetition or loop within a sequence diagram is depicted as a rectangle. Place the condition for exiting the loop at the bottom left corner in square brackets [ ].

- **Guards**

When modeling object interactions, there will be times when a condition must be met for a message to be sent to an object. Guards are conditions that need to be used throughout UML diagrams to control flow.

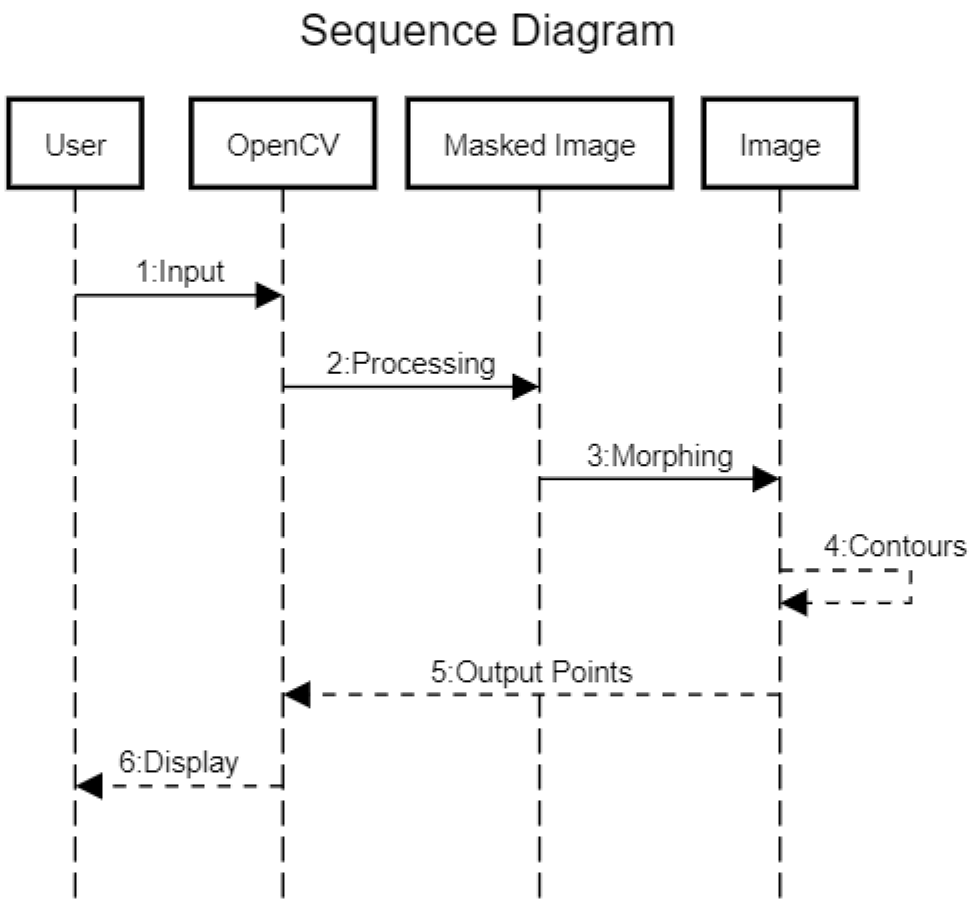


Figure 5: Sequence Diagram

4.2.3 Component Diagram

Component diagrams are different in terms of nature and behavior. Component diagrams are used to model the physical aspects of a system. Now the question is, what are these physical aspects? Physical aspects are the elements such as executables, libraries, files, documents, etc. which reside in a node.

Component diagrams are used to visualize the organization and relationships among components in a system. These diagrams are also used to make executable systems.

Component diagram is a special kind of diagram in UML. The purpose is also different from all other diagrams discussed so far. It does not describe the functionality of the system but it describes the components used to make those functionalities.

Thus from that point of view, component diagrams are used to visualize the physical components in a system. These components are libraries, packages, files, etc.

Component diagrams can also be described as a static implementation view of a system. Static implementation represents the organization of the components at a particular moment.

A single component diagram cannot represent the entire system but a collection of diagrams is used to represent the whole.

The purpose of the component diagram can be summarized as −

- Visualize the components of a system.
- Construct executables by using forward and reverse engineering.
- Describe the organization and relationships of the components.

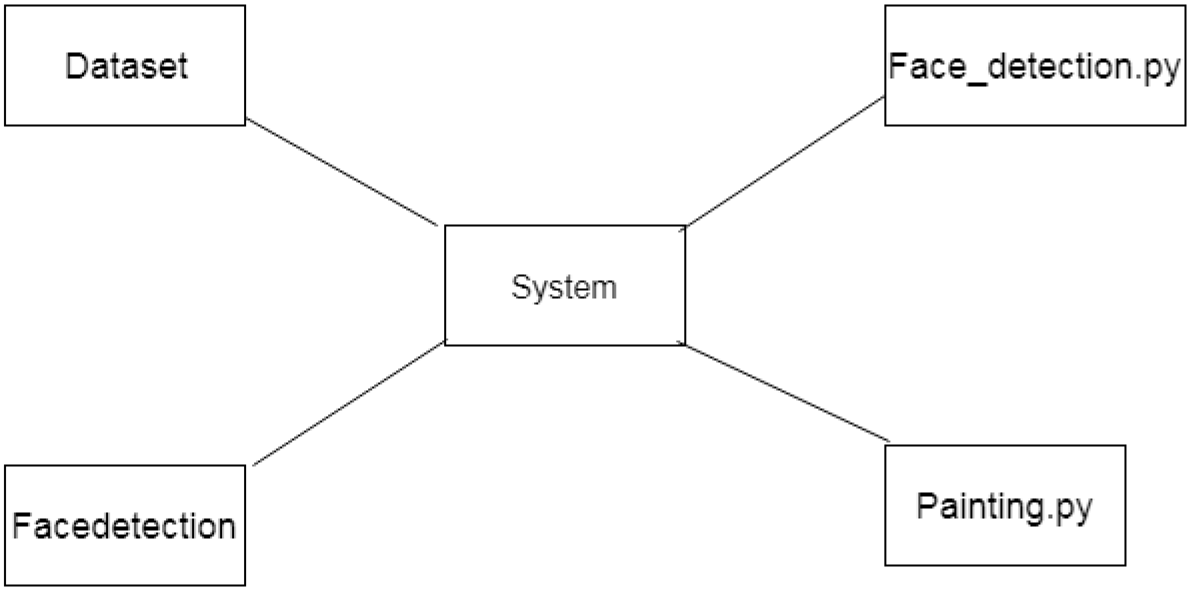


Figure : Component Diagram

4.2.4 Collaboration Diagram

A collaboration diagram, also called a communication diagram or interaction diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML). The concept is more than a decade old although it has been refined as modeling paradigms have evolved.

A collaboration diagram resembles a flowchart that portrays the roles, functionality and behavior of individual objects as well as the overall operation of the system in real time. Objects are shown as rectangles with naming labels inside. These labels are preceded by colons and may be underlined. The relationships between the objects are shown as lines connecting the rectangles. The messages between objects are shown as arrows connecting the relevant rectangles along with labels that define the message sequencing.

Collaboration diagrams are best suited to the portrayal of simple interactions among relatively small numbers of objects. As the number of objects and messages grows, a collaboration diagram can become difficult to read. Several vendors offer software for creating and editing collaboration diagrams.

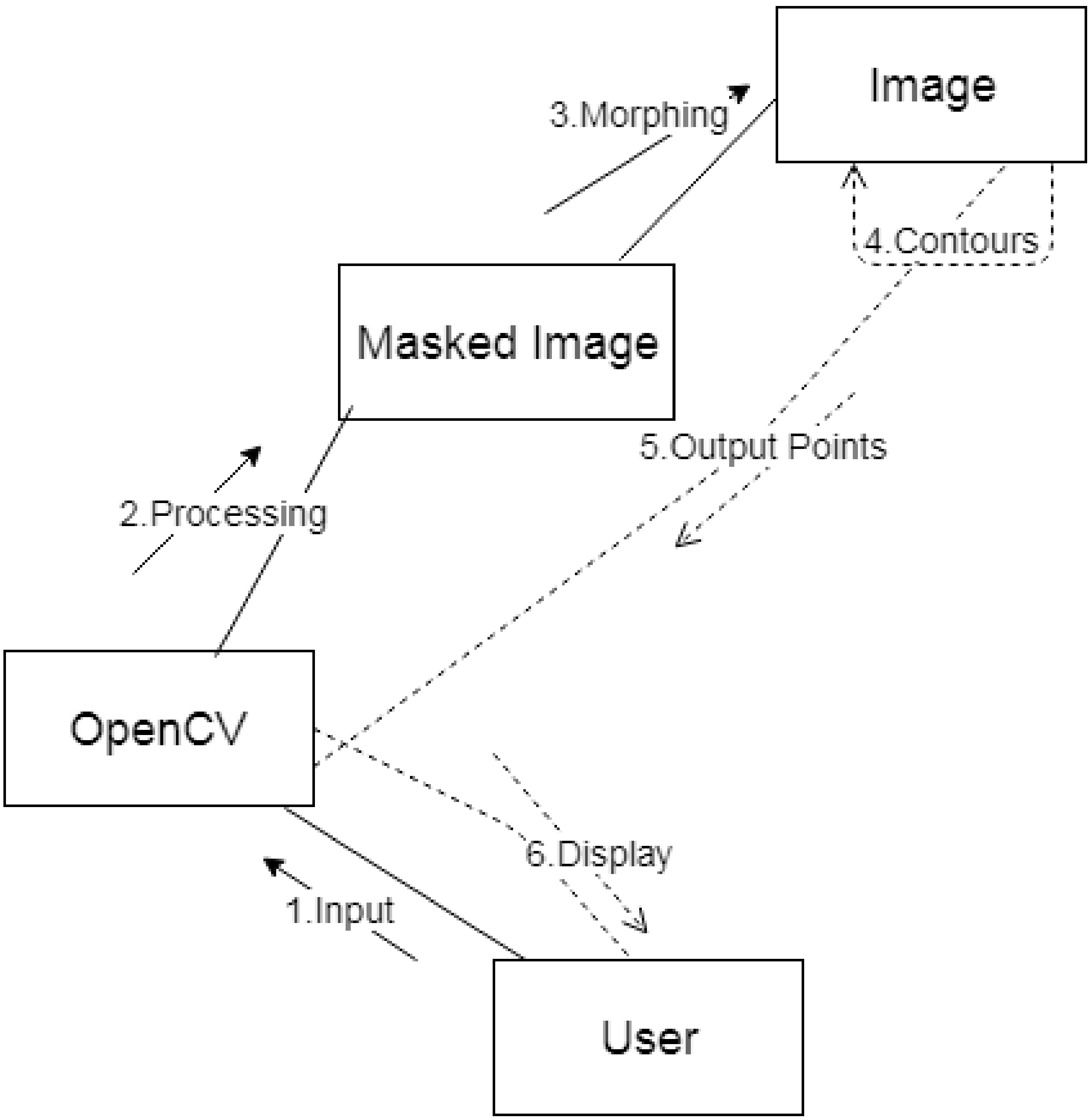


Figure: Collaboration Diagram

4.2.5 Class Diagram

A class diagram in the Unified Modeling Language is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations, and the relationships among objects.

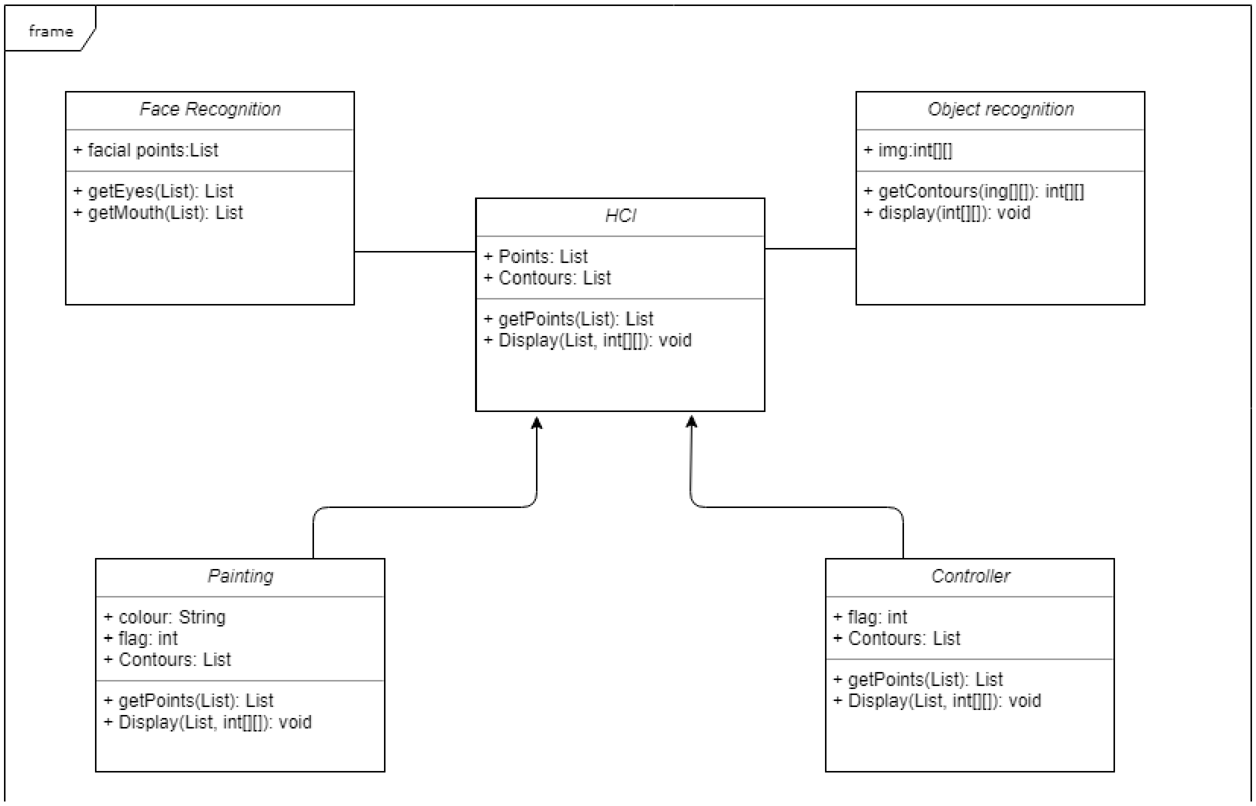


Figure: Class Diagram

4.2.6 Activity Diagram

Activity diagram is another important diagram in UML to describe the dynamic aspects of the system. The activity can be described as an operation of the system.The control flow is drawn from one operation to another.

The basic purposes of activity diagrams is similar to other four diagrams. It captures the dynamic behavior of the system.Activity is a particular operation of the system. Activity diagrams are not only used for visualizing the dynamic nature of a system, but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in the activity diagram is the message part.

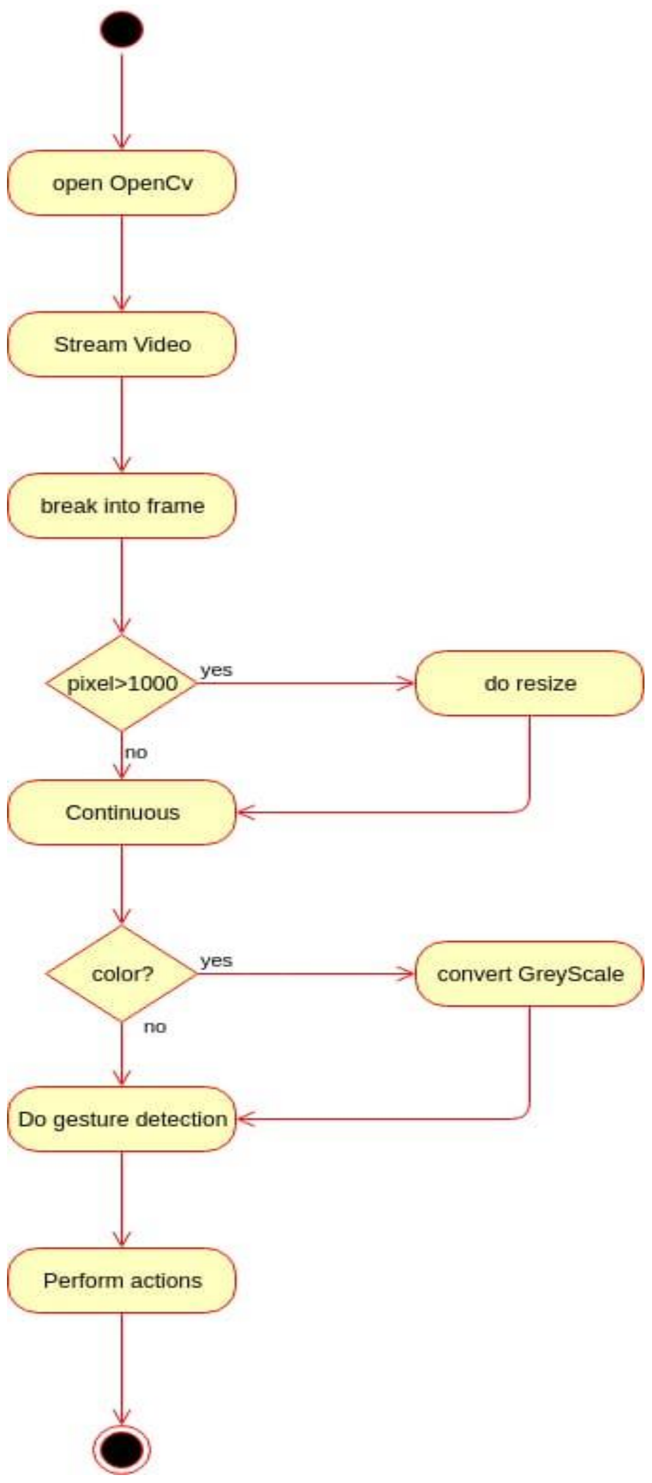


Figure: Activity Diagram

CHAPTER-5

5. IMPLEMENTATION

Read an image

Use the function cv2.imread() to read an image. The image should be in the working directory or a full path of image should be given.

Second argument is a flag which specifies the way image should be read.

cv2.IMREAD_COLOR : Loads a color image. Any transparency of image will be neglected. It is the default flag.

cv2.IMREAD_GRAYSCALE : Loads image in grayscale mode

cv2.IMREAD_UNCHANGED : Loads image as such including alpha channel

```
import cv2

Load an color image in grayscale img = cv2.imread('messi5.jpg',0)
```

Display an image

Use the function cv2.imshow() to display an image in a window. The window automatically fits to the image size.

First argument is a window name which is a string. second argument is our image. You can create as many windows as you wish, but with different window names.

```
cv2.imshow('image',img)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

cv2.waitKey() is a keyboard binding function. Its argument is the time in milliseconds. The function waits for specified milliseconds for any keyboard event. If you press any key in that time, the program continues. If 0 is passed, it waits indefinitely for a key stroke. It can also be set to detect specific key strokes like, if key a is pressed etc which we will discuss below.

cv2.destroyAllWindows() simply destroys all the windows we created. If you want to destroy any specific window, use the function cv2.destroyWindow() where you pass the exact window name as the argument.

See the code below:

```
cv2.namedWindow('image', cv2.WINDOW_NORMAL)

cv2.imshow('image',img)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

Capture Video from Camera

Often, we have to capture live stream with camera. OpenCV provides a very simple interface to this. Let's capture a video from the camera (I am using the in-built webcam of my laptop), convert it into grayscale video and display it. Just a simple task to get started.

To capture a video, you need to create a VideoCapture object. Its argument can be either the device index or the name of a video file. Device index is just the number to specify which camera. Normally one camera will be connected (as in my case). So I simply pass 0 (or -1). You can select the second camera by passing 1 and so on. After that, you can capture frame-by-frame. But at the end, don't forget to release the capture.

```
import numpy as np

import cv2

cap = cv2.VideoCapture(0)

while(True):

    Capture frame-by-frame
    ret, frame = cap.read()

      Our operations on the frame come here

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    cv2.imshow('frame',gray)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

When everything done, release the capture
cap.release() cv2.destroyAllWindows()
```

cap.read() returns a bool (True/False). If frame is read correctly, it will be True. So you can check end of the video by checking this return value.

Sometimes, cap may not have initialized the capture. In that case, this code shows error. You can check whether it is initialized or not by the method cap.isOpened(). If it is True, OK. Otherwise open it using cap.open().

You can also access some of the features of this video using cap.get(propId) method where propId is a number from 0 to 18. Each number denotes a property of the video (if it is applicable to that video) and full details can be seen here: Property Identifier. Some of these values can be modified using cap.set(propId, value). Value is the new value you want.

For example, I can check the frame width and height by cap.get(3) and cap.get(4). It gives me 640x480 by default. But I want to modify it to 320x240. Just use ret = cap.set(3,320) and ret = cap.set(4,240).

Drawing Functions in OpenCV

In all the above functions, you will see some common arguments as given below:

img : The image where you want to draw the shapes

color : Color of the shape. for BGR, pass it as a tuple, eg: (255,0,0) for blue. For grayscale, just pass the scalar value.

thickness : Thickness of the line or circle etc. If -1 is passed for closed figures like circles, it will fill the shape. default thickness = 1

lineType : Type of line, whether 8-connected, anti-aliased line etc. By default, it is 8-connected. cv2.LINE_AA gives anti-aliased line which looks great for curves.

Drawing Line

To draw a line, you need to pass starting and ending coordinates of line. We will create a black image and draw a blue line on it from top-left to bottom-right corners.

import numpy as np

import cv2

img = np.zeros((512,512,3), np.uint8)

Draw a diagonal blue line with thickness of 5 px
img = cv2.line(img,(0,0),(511,511),(255,0,0),5)

Drawing Rectangle

To draw a rectangle, you need top-left corner and bottom-right corner of rectangle. This time we will draw a green rectangle at the top-right corner of image.

img = cv2.rectangle(img,(384,0),(510,128),(0,255,0),3)

Drawing Circle

To draw a circle, you need its center coordinates and radius. We will draw a circle inside the rectangle drawn above.

img = cv2.circle(img,(447,63), 63, (0,0,255), -1)

Drawing Ellipse

To draw the ellipse, we need to pass several arguments. One argument is the center location (x,y). Next argument is axes lengths (major axis length, minor axis length). angle is the angle of rotation of ellipse in anti-clockwise direc-tion. startAngle and endAngle denotes the starting and ending of ellipse arc measured in clockwise direction from major axis. i.e. giving values 0 and 360 gives the full ellipse. For more details, check the documentation of cv2.ellipse(). Below example draws a half ellipse at the center of the image.

img = cv2.ellipse(img,(256,256),(100,50),0,0,180,255,-1)

Drawing Polygon

To draw a polygon, first you need coordinates of vertices. Make those points into an array of shape ROWSx1x2 where ROWS are number of vertices and it should be of type int32. Here we draw a small polygon of with four vertices in yellow color.

pts = np.array([[10,5],[20,30],[70,20],[50,10]], np.int32)

pts = pts.reshape((-1,1,2))

img = cv2.polylines(img,[pts],**True**,(0,255,255))

Mouse as a Paint-Brush

Here, we create a simple application which draws a circle on an image wherever we double-click on it.

First we create a mouse callback function which is executed when a mouse event take place. Mouse event can be anything related to mouse like left-button down, left-button up, left-button double-click etc. It gives us the coordinates (x,y) for every mouse event. With this event and location, we can do whatever we like. To list all available events available, run the following code in Python terminal:

```
    import cv2

    events = [i for i in dir(cv2) if 'EVENT' in i]

    print events
```

Creating mouse callback function has a specific format which is same everywhere. It differs only in what the function does. So our mouse callback function does one thing, it draws a circle where we double-click. So see the code below. Code is self-explanatory from comments :

```
import cv2

import numpy as np

def draw_circle(event,x,y,flags,param):

    if event == cv2.EVENT_LBUTTONDBLCLK:

        cv2.circle(img,(x,y),100,(255,0,0),-1)
```

Create a black image, a window and bind the function to window

```
img = np.zeros((512,512,3), np.uint8) cv2.namedWindow('image')
cv2.setMouseCallback('image',draw_circle)
while(1):

    cv2.imshow('image',img)

    if cv2.waitKey(20) & 0xFF == 27:

        break

        cv2.destroyAllWindows()
```

Advanced

Now we go for much more better application. In this, we draw either rectangles or circles (depending on the mode we select) by dragging the mouse like we do in Paint application. So our mouse callback function has two parts, one to draw rectangle and other to draw the circles. This specific example will be really helpful in creating and understanding some interactive applications like object tracking, image segmentation etc.

```
import cv2

import numpy as np
drawing = False # true if mouse is pressed

mode = True # if True, draw rectangle. Press 'm' to toggle to curve ix,iy = -
1,-1
# mouse callback function

def draw_circle(event,x,y,flags,param):

    global ix,iy,drawing,mode
    ix,iy = x,y
```

Next we have to bind this mouse callback function to OpenCV window. In the main loop, we should set a keyboard binding for key ‘m’ to toggle between rectangle and circle.

```
img = np.zeros((512,512,3), np.uint8)

cv2.namedWindow('image')

cv2.setMouseCallback('image',draw_circle)
while(1):

    cv2.imshow('image',img)

    k = cv2.waitKey(1) & 0xFF

    if k == ord('m'):

        mode = not mode

    elif k == 27:
        break
cv2.destroyAllWindows()
```

Basic Operations on Images

Accessing and Modifying pixel values

Let's load a color image first:

```
import cv2

import numpy as np
img = cv2.imread('messi5.jpg')
```

You can access a pixel value by its row and column coordinates. For BGR image, it returns an array of Blue, Green, Red values. For grayscale image, just corresponding intensity is returned.

```
px = img[100,100]

print px [157 166 200]
blue = img[100,100,0]
```

You can modify the pixel values the same way.

```
img[100,100] = [255,255,255]

print img[100,100]

[255 255 255]
```

Image ROI

Sometimes, you will have to play with certain region of images. For eye detection in images, first perform face detection over the image until the face is found, then search within the face region for eyes. This approach improves accuracy (because eyes are always on faces :D) and performance (because we search for a small area).

ROI is again obtained using Numpy indexing. Here I am selecting the ball and copying it to another region in the image:

```
ball = img[280:340, 330:390]

img[273:333, 100:160] = ball
```

Scaling

Scaling is just resizing of the image. OpenCV comes with a function cv2.resize() for this purpose. The size of the image can be specified manually, or you can specify the scaling factor. Different interpolation methods are used. Preferable interpolation methods are cv2.INTER_AREA for shrinking and cv2.INTER_CUBIC (slow) & cv2.INTER_LINEAR for zooming. By default, interpolation method used is cv2.INTER_LINEAR for all resizing purposes. You can resize an input image either of following methods:

import cv2

import numpy as np

img = cv2.imread('messi5.jpg')

res = cv2.resize(img,**None**,fx=2, fy=2, interpolation = cv2.INTER_CUBIC)

#OR

height, width = img.shape[:2]

res = cv2.resize(img,(2*width, 2*height), interpolation = cv2.INTER_CUBIC)

What are contours?

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

For better accuracy, use binary images. So before finding contours, apply threshold or canny edge detection.

findContours function modifies the source image. So if you want source image even after finding contours, already store it to some other variables.

In OpenCV, finding contours is like finding white object from black background. So remember, object to be found should be white and background should be black.

Let’s see how to find contours of a binary image:

```
import numpy as np

import cv2

im = cv2.imread('test.jpg')

imgray = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)

ret,thresh = cv2.threshold(imgray,127,255,0)

image, contours, hierarchy =
 cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_ ‹→SIMPLE)
```

See, there are three arguments in cv2.findContours() function, first one is source image, second is contour retrieval mode, third is contour approximation method. And it outputs the image, contours and hierarchy. contours is a Python list of all the contours in the image. Each individual contour is a Numpy array of (x,y) coordinates of boundary points of the object.

Drawing the contours

To draw the contours, cv2.drawContours function is used. It can also be used to draw any shape provided you have its boundary points. Its first argument is source image, second argument is the contours which should be passed as a Python list, third argument is index of contours (useful when drawing individual contour. To draw all contours, pass -1) and remaining arguments are color, thickness etc.

To draw all the contours in an image:

```
img = cv2.drawContours(img, contours, -1, (0,255,0), 3)
```

To draw an individual contour, say 4th contour:

But most of the time, below method will be useful:

```
cnt = contours[4]

img = cv2.drawContours(img, [cnt], 0, (0,255,0), 3)
```

1. Contour Area

Contour area is given by the function cv2.contourArea() or from moments, M['m00'].

```
area = cv2.contourArea(cnt)
```

2. Contour Perimeter

It is also called arc length. It can be found out using cv2.arcLength() function. Second argument specify whether shape is a closed contour (if passed True), or just a curve.

perimeter = cv2.arcLength(cnt,**True**)

3. Convex Hull

Convex Hull will look similar to contour approximation, but it is not (Both may provide same results in some cases). Here, cv2.convexHull() function checks a curve for convexity defects and corrects it. Generally speaking, convex curves are the curves which are always bulged out, or at-least flat. And if it is bulged inside, it is called convexity defects. For example, check the below image of hand. Red line shows the convex hull of hand. The double-sided arrow marks shows the convexity defects, which are the local maximum deviations of hull from con

There is a little bit things to discuss about it its syntax:

hull = cv2.convexHull(points[, hull[, clockwise[, returnPoints]]

Arguments details:

points are the contours we pass into.

hull is the output, normally we avoid it.
clockwise : Orientation flag. If it is True, the output convex hull is oriented clockwise. Otherwise, it is oriented counter-clockwise.

returnPoints : By default, True. Then it returns the coordinates of the hull points. If False, it returns the indices of contour points corresponding to the hull points.

So to get a convex hull as in above image, following is sufficient:

hull = cv2.convexHull(cnt)

But if you want to find convexity defects, you need to pass returnPoints = False. To understand it, we will take the rectangle image above. First I found its contour as cnt. Now I found its convex hull with returnPoints

CHAPTER-6

6. TECHNIQUES

6.1 Changing Colorspaces

We start by converting our video stream from BGR to HSV.There are more than 150 color-space conversion methods available in OpenCV. But we will look into only two which are most widely used ones, BGR ↔ Gray and BGR ↔ HSV.

For color conversion, we use the function cv2.cvtColor(input_image, flag) where flag determines the type of conversion.

For BGR → Gray conversion we use the flags cv2.COLOR_BGR2GRAY. Similarly for BGR → HSV, we use the flag cv2.COLOR_BGR2HSV.

we use the flag cv2.COLOR_BGR2HSV. To get other flags, just run following commands in your Python terminal

```
import cv2

flags = [i for i in dir(cv2) if i.startswith('COLOR_')]

print flags
```

For HSV, Hue range is [0,179], Saturation range is [0,255] and Value range is [0,255]. Different softwares use different scales. So if you are comparing OpenCV values with them, you need to normalize these ranges.

6.2 Object Tracking

Now we know how to convert BGR image to HSV, we can use this to extract a colored object. In HSV, it is more easier to represent a color than RGB color-space. In our application, we will try to extract a blue colored object. So here is the method:

Take each frame of the video

Convert from BGR to HSV color-space

We threshold the HSV image for a range of blue color

Now extract the blue object alone, we can do whatever on that image we want.

Code Sample :

```
import cv2

import numpy as np

cap = cv2.VideoCapture(0)

while(1):

    # Take each frame

    _, frame = cap.read()

    # Convert BGR to HSV

    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    define range of blue color in HSV
    lower_blue = np.array([110,50,50])
    upper_blue = np.array([130,255,255])

    Threshold the HSV image to get only blue colors mask
    = cv2.inRange(hsv, lower_blue, upper_blue)

   Bitwise-AND mask and original image

   res = cv2.bitwise_and(frame,frame, mask= mask)

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

    cv2.imshow('res',res)

    k = cv2.waitKey(5) & 0xFF

    if k == 27:

        break

cv2.destroyAllWindows()
```

Below image shows tracking of the blue object:

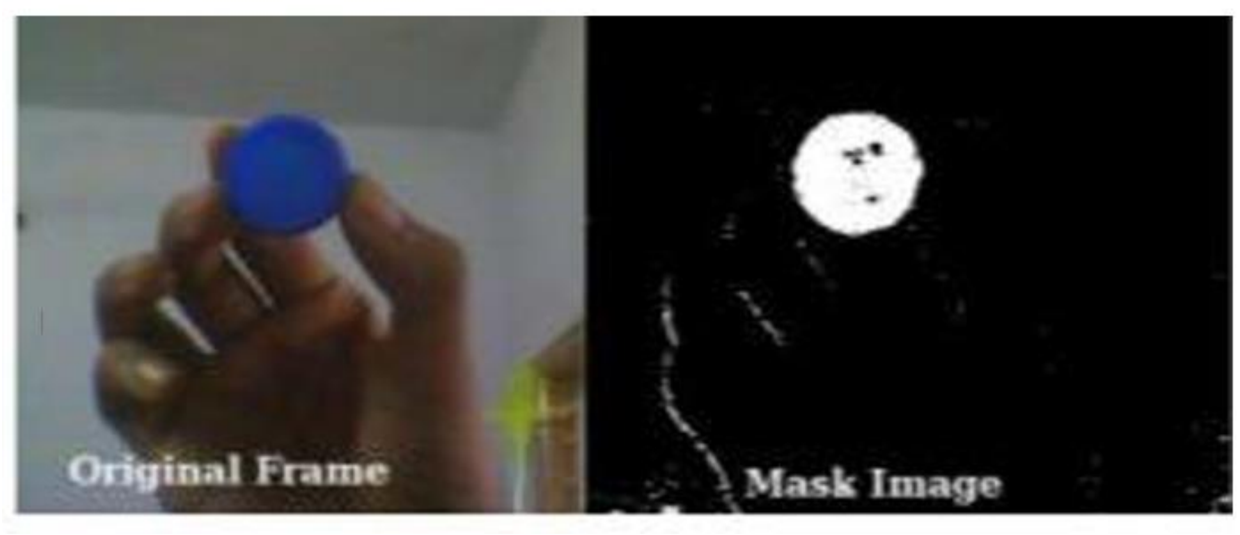


Figure: Object Mask

Tracking HSV Values

It is very simple and you can use the same function, cv2.cvtColor(). Instead of passing an image, you just pass the BGR values you want.

HSV value of Green, try following commands in Python terminal:

```
    green = np.uint8([[[0,255,0 ]]])

    hsv_green = cv2.cvtColor(green,cv2.COLOR_BGR2HSV)

    print hsv_green

[[[ 60 255 255]]]
```

Now you take [H-10, 100,100] and [H+10, 255, 255] as lower bound and upper bound respectively. Apart from this method, you can use any image editing tools like GIMP or any online converters to find these values, but don’t forget to adjust the HSV ranges.

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. For this, haar features shown in below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle.

Now all possible sizes and locations of each kernel is used to calculate plenty of features. (Just imagine how much computation it needs? Even a 24x24 window results over 160000 features). For each feature calculation, we need to find sum of pixels under white and black rectangles. To solve this, they introduced the integral images. It simplifies calculation of sum of pixels, how large may be the number of pixels, to an operation involving just four pixels. Nice, isn’t it? It makes things super-fast.

But among all these features we calculated, most of them are irrelevant. For example, consider the image below. Top row shows two good features. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. But the same windows applying on cheeks or any other place is irrelevant. So how do we select the best features out of 160000+ features? It is achieved by Adaboost.

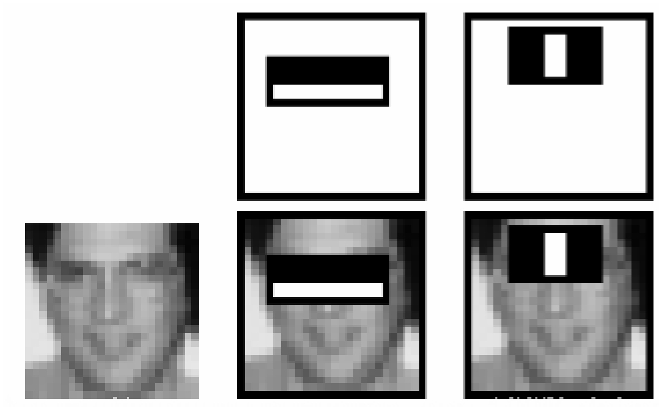


Figure: Tracking HSV

For this, we apply each and every feature on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. But obviously, there will be errors or misclassifications. We select the features with minimum error rate, which means they are the features that best classifies the face and non-face images. (The process is not as simple as this. Each image is given an equal weight in the beginning. After each classification, weights of misclassified images are increased. Then again same process is done. New error rates are calculated. Also new weights. The process is continued until required accuracy or error rate is achieved or required number of features are found).

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

So now you take an image. Take each 24x24 window. Apply 6000 features to it. Check if it is face or not. Wow.. Wow.. Isn't it a little inefficient and time consuming? Yes, it is. Authors have a good solution for that.

In an image, most of the image region is non-face region. So it is a better idea to have a simple method to check if a window is not a face region. If it is not, discard it in a single shot. Don't process it again. Instead focus on region where there can be a face. This way, we can find more time to check a possible face region.

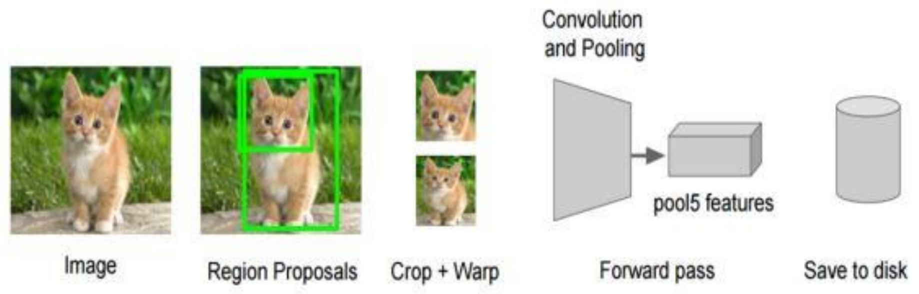
For this they introduced the concept of Cascade of Classifiers. Instead of applying all the 6000 features on a window, group the features into different stages of classifiers and apply one-by-one.

(Normally first few stages will contain very less number of features). If a window fails the first stage, discard it. We don't consider remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region. How is the plan !!!

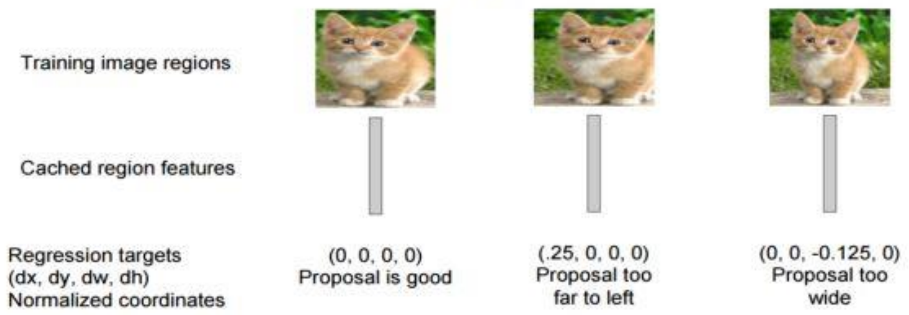
Authors' detector had 6000+ features with 38 stages with 1, 10, 25, 25 and 50 features in first five stages. (Two features in the above image is actually obtained as the best two features from Adaboost). According to authors, on an average, 10 features out of 6000+ are evaluated per sub-window.

So this is a simple intuitive explanation of how Viola-Jones face detection works. Read paper for more details or check out the references in Additional Resources section.

Two-stage Method In this case, the proposals are extracted using some other computer vision technique and then resized to fixed input for the classification network, which acts as a feature extractor. Then an SVM is trained to classify between object and background (one SVM for each class). Also a bounding box regressor is trained that outputs some some correction (offsets) for proposal boxes. The overall idea is shown in Fig. 5 These methods are very accurate but are computationally intensive (low fps).



(a) Stage 1


Training image regions
Cached region features
Regression targets (dx, dy, dw, dh) Normalized coordinates
(0, 0, 0, 0) Proposal is good
(.25, 0, 0, 0) Proposal too far to left
(0, 0, -0.125, 0) Proposal too wide

(b) Stage 2

Figure 5: Two stage method

Unified Method

The difference here is that instead of producing proposals, pre-define a set of boxes to look for objects. Using convolutional feature maps from later layers of the network, run another network over these feature maps to predict class scores and bounding box offsets. The steps are mentioned below:

1. Train a CNN with regression and classification objective.

2. Gather activation from later layers to infer classification and location with a fully connected or convolutional layers.

3. During training, use jaccard distance to relate predictions with the ground truth.

4. During inference, use non-maxima suppression to filter multiple boxes around the same object.

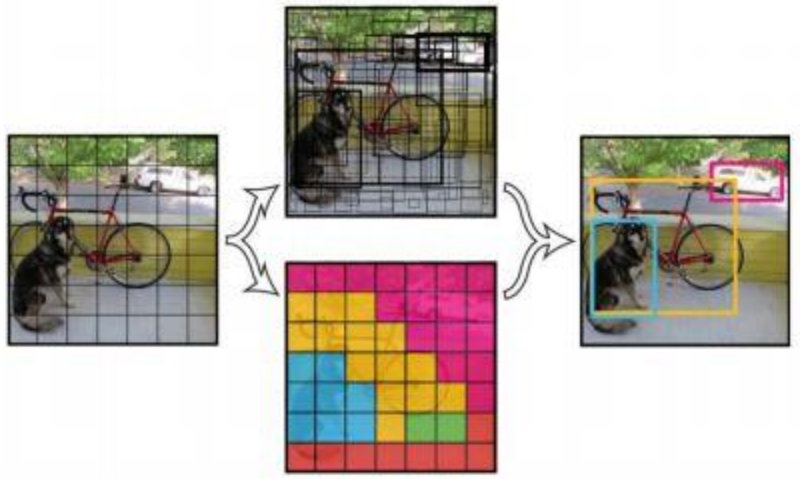


Figure 6: Unified Method

The major techniques that follow this strategy are: SSD (uses different activation maps (multiple-scales) for prediction of classes and bounding boxes) and Yolo (uses a single activation map for prediction of classes and bounding boxes). Using multiple scales helps to achieve a higher mAP(mean average precision) by being able to detect objects with different sizes on the image better. Thus the technique used in this project is SSD.

6.3 Image Thresholding

Simple Thresholding

Here, the matter is straight forward. If pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black). The function used is cv2.threshold. First argument is the source image, which should be a grayscale image. Second argument is the threshold value which is used to classify the pixel values. Third argument is the maxVal which represents the value to be given if pixel value is more than (sometimes less than) the threshold value. OpenCV provides different styles of thresholding and it is decided by the fourth parameter of the function. Different types are:

cv2.THRESH_BINARY

cv2.THRESH_BINARY_INV

cv2.THRESH_TRUNC
cv2.THRESH_TOZERO

cv2.THRESH_TOZERO_INV

Sample Code :

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('gradient.png',0)
ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)

ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)

titles                    =                    ['Original
Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']  images  =  [img,
thresh1, thresh2, thresh3, thresh4, thresh5]

for i in xrange(6):
plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
plt.title(titles[i])
plt.xticks([]),plt.yticks([])

plt.show()
```

To plot multiple images, we have used plt.subplot() function. Please checkout Matplotlib docs for more details.

Result is given below :

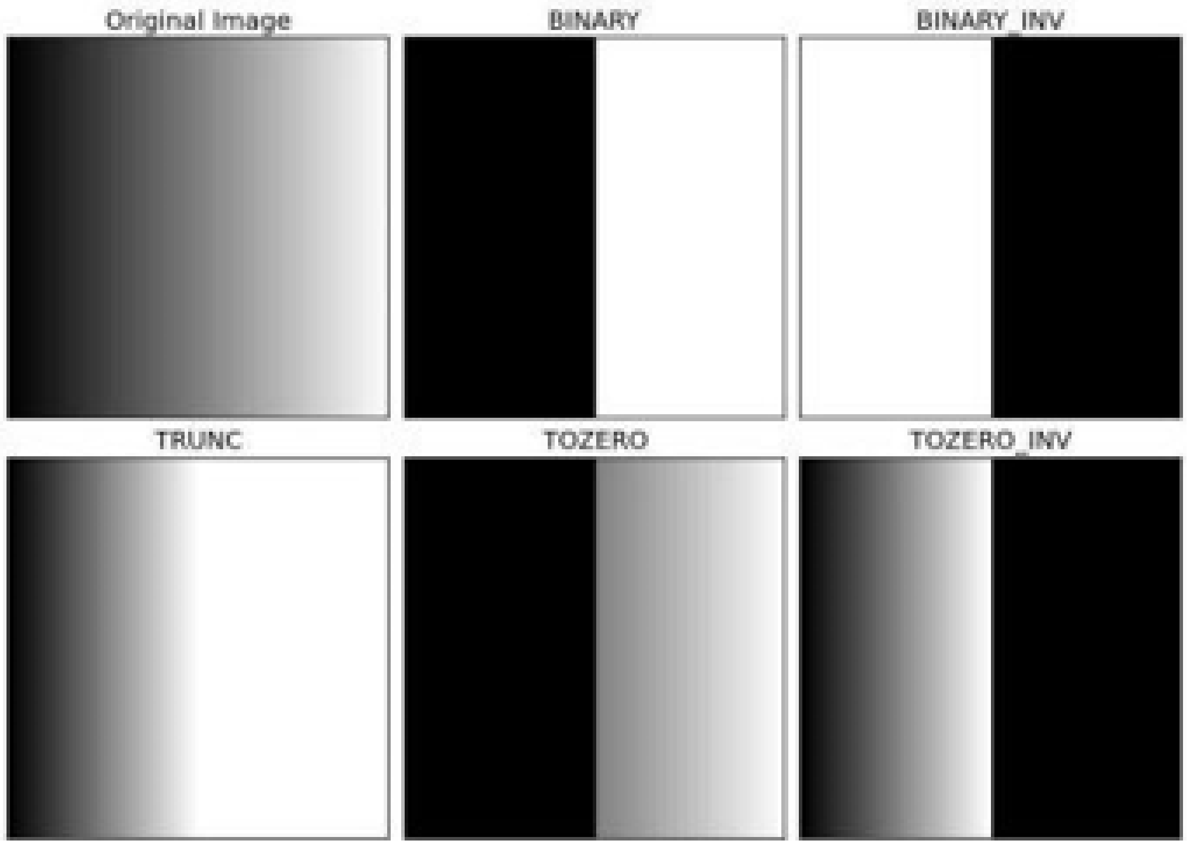


Figure: Thresholding

Canny Edge Detection:

Canny Edge Detection is a popular edge detection algorithm. It was developed by John F. Canny in 1986. It is a multi-stage algorithm and we will go through each stages.

Noise Reduction**:**

Since edge detection is susceptible to noise in the image, first step is to remove the noise in the image with a 5x5 Gaussian filter. We have already seen this in previous chapters.

Finding Intensity Gradient of the Image

Gradient direction is always perpendicular to edges. It is rounded to one of four angles representing vertical, horizontal and two diagonal directions.

Non-maximum Suppression

After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient. Check the image below:

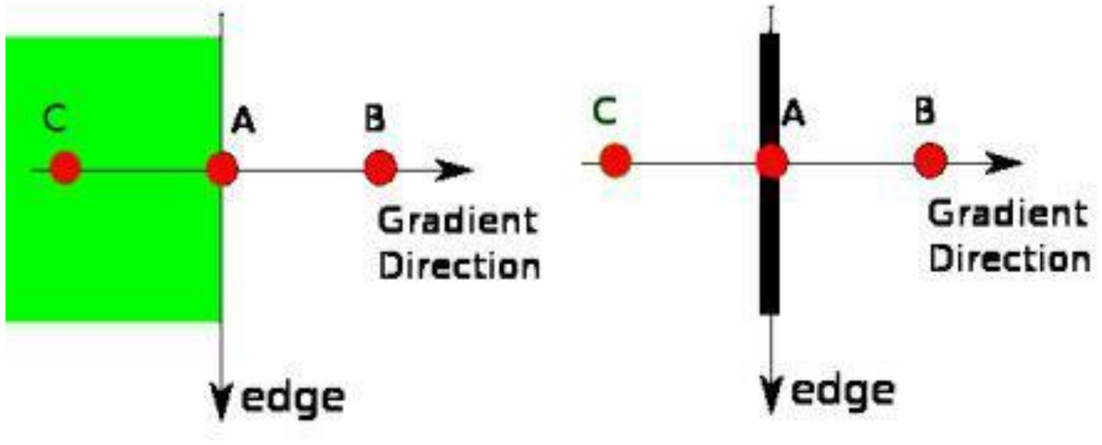


Figure: Gradient Direction

Point A is on the edge (in vertical direction). Gradient direction is normal to the edge. Point B and C are in gradient directions. So point A is checked with point B and C to see if it forms a local maximum. If so, it is considered for next stage, otherwise, it is suppressed (put to zero).

In short, the result you get is a binary image with "thin edges".

Hysteresis Thresholding

This stage decides which are all edges are really edges and which are not. For this, we need two threshold values, minVal and maxVal. Any edges with intensity gradient more than maxVal are sure to be edges and those below minVal are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to "sure-edge" pixels, they are considered to be part of edges. Otherwise, they are also discarded. See the image below:

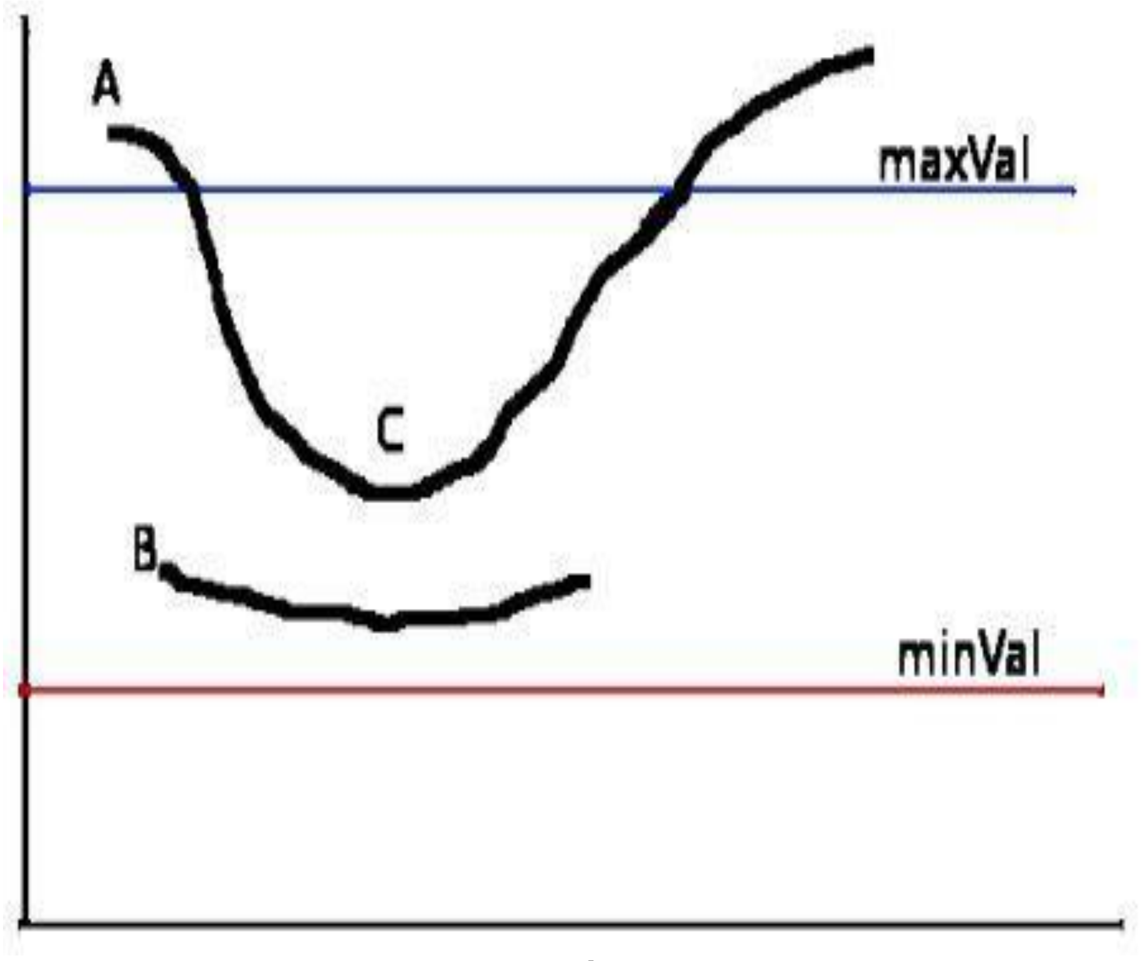


Figure: Sample Drawing

The edge A is above the maxVal, so considered as “sure-edge”. Although edge C is below maxVal, it is connected to edge A, so that also considered as valid edge and we get that full curve. But edge B, although it is above minVal and is in same region as that of edge C, it is not connected to any “sure-edge”, so that is discarded. So it is very important that we have to select minVal and maxVal accordingly to get the correct result.

This stage also removes small pixels noises on the assumption that edges are long lines.

So what we finally get is strong edges in the image.

Canny Edge Detection in OpenCV

OpenCV puts all the above in single function, cv2.Canny(). We will see how to use it. First argument is our input image. Second and third arguments are our minVal and maxVal respectively. Third argument is aperture_size. It is the size of Sobel kernel used for find image gradients. By default it is 3. Last argument is L2gradient which specifies the equation for finding gradient magnitude. If it is True, it uses the equation mentioned above which is more accurate, otherwise it uses this function: _ ( ) = | | + | |. By default, it is False.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('messi5.jpg',0)
edges = cv2.Canny(img,100,200)
plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
plt.show()
```

See the result below:

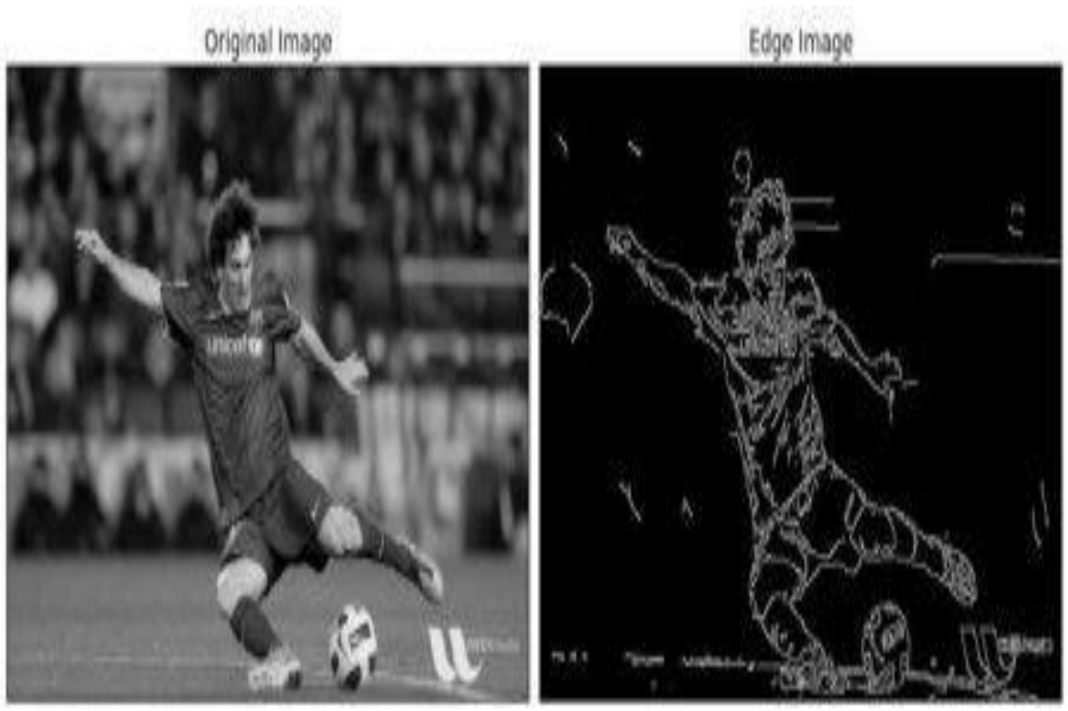


Figure: Edge Detection

Machine learning algorithms used for natural language processing (NLP) currently take too long to complete their learning function. This slow learning performance tends to make the model ineffective for an increasing requirement for real time applications such as voice transcription, language translation, text summarization topic extraction and sentiment analysis. Moreover, current implementations are run in an offline batch-mode operation and are unfit for real time needs. Newer machine learning algorithms are being designed that make better use of sampling and distributed methods to speed up the learning performance.

6.4 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

Geometrical features and Template matching.

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.

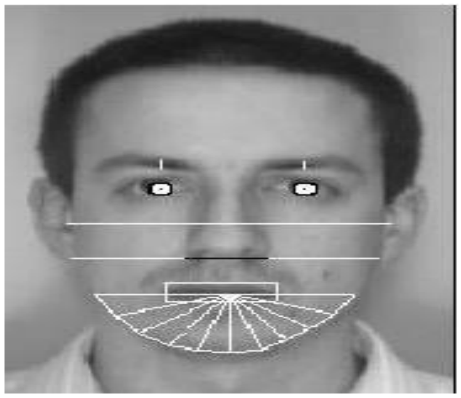


Figure 6.1 Geometrical features (white) which could be used for face recognition

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.

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.

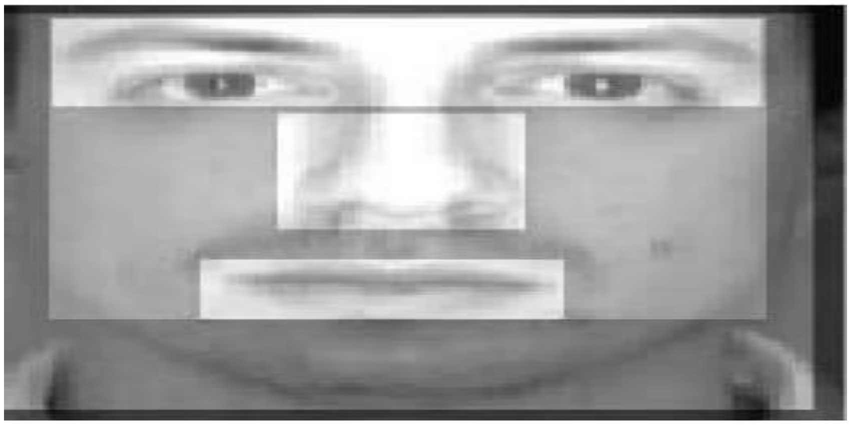


Figure: Template Matching

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.

PROBLEM SCOPE AND SYSTEM SPECIFICATION

The following problem scope for this project was arrived at after reviewing the literature on face detection and face recognition, and determining possible real-world situations where such systems would be of use. The following system(s) requirements were identified

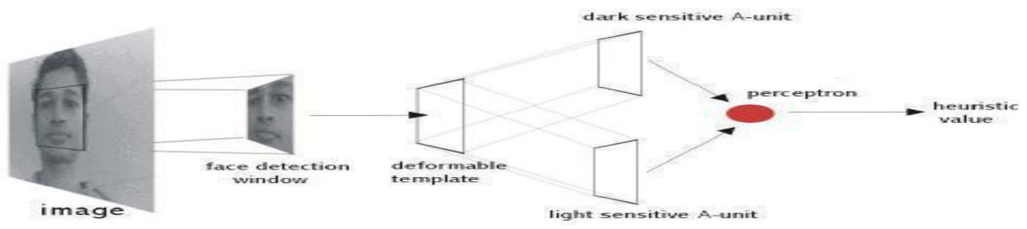
A system to detect frontal view faces in static images
A system to recognize a given frontal view face
Only expressionless, frontal view faces will be presented to the face detection&recognition
All implemented systems must display a high degree of lighting invariency.
All systems must posses near real-time performance.
Both fully automated and manual face detection must be supported
Frontal view face recognition will be realised using only a single known image

Automated face detection and recognition systems should be combined into a fully automated face detection and recognition system. The face recognition sub-system must display a slight degree of invariency to scaling and rotation errors in the segmented image extracted by the face detection sub-system.
The frontal view face recognition system should be extended to a pose invariant face recognition system.
Unfortunately although we may specify constricting conditions to our problem domain, it may not be possible to strictly adhere to these conditions when implementing a system in the real-world.

BRIEF OUT LINE OF THE 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



A template matching based technique was implemented for face recognition. This was because of its increased recognition accuracy when compared to geometrical features based techniques and the fact that an automated geometrical features based technique would have required complex feature detection pre-processing.

Of the many possible template matching techniques, Principal Component Analysis was chosen because it has proved to be a highly robust in pattern recognition tasks and because it is relatively simple to implement. The author would also liked to have implemented a technique based on Elastic Graphs but could not find sufficient literature about the model to implement such a system during the limited time available for this project.

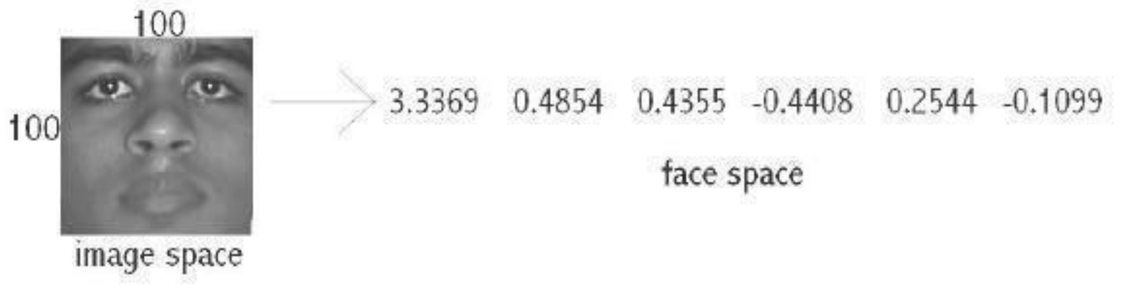


Figure :Principal Component Analysis transform from 'image space' to 'face space'.

Using Principal Component Analysis, the segmented frontal view face image is transformed from what is sometimes called 'image space' to 'face space'. All faces in the face database are transformed into face space. Then face recognition is achieved by transforming any given test image into face space and comparing it with the training set vectors. The closest matching training set vector should belong to the same individual as the test image.Principal Component Analysis is of special interest because the transformation to face space is based on the variation of human faces (in the training set). The values of the 'face space' vector correspond to the amount certain 'variations' are present in the test 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.

FACE RECOGNITION DIFFICULTIES

1. Identify similar faces (inter-class similarity)
2. Accommodate intra-class variability

   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 brightness

POSE INVARIANT FACE RECOGNITION

Extending the frontal view face recognition system to a pose-invariant recognition system is quite simple if one of the proposed specifications of the face recognition system is relaxed. Successful pose-invariant recognition will be possible if many images of a known individual are in the face database. Nine images from each known individual can be taken as shown below. Then if an image of the same individual is submitted within a 30^o angle from the frontal view he or she can be identified. Nine images in face database from a single known individual Unknown image from same individual to be identified

Fig: 6.8 Pose invariant face recognition.

Pose invariant face recognition highlights the generalisation ability of PCA. For example, when an individual's frontal view and 30^{o} left view known, even the individual's 15^{o} left view can be recognised.

CHAPTER-7

7. SCREENSHOTS

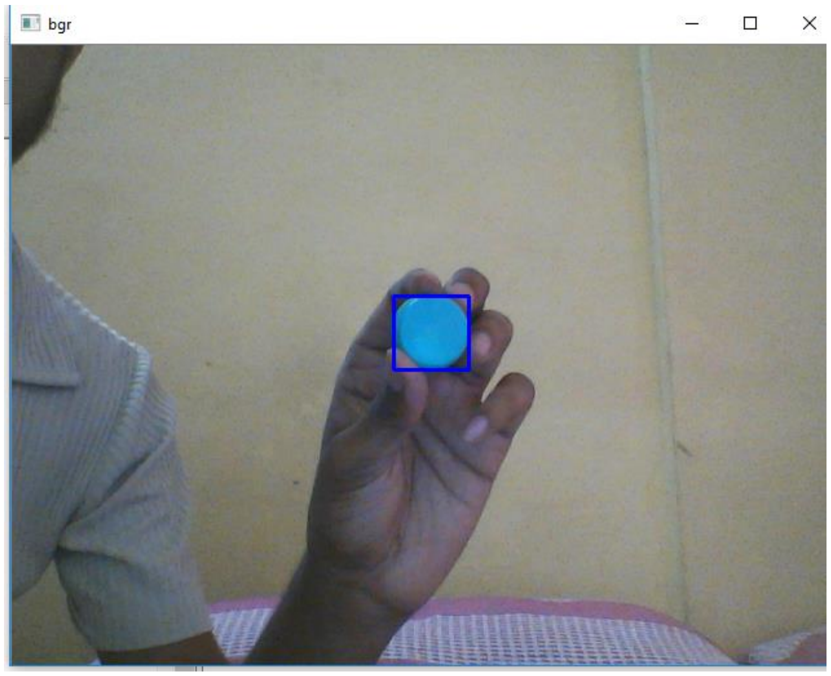


Figure : object detection

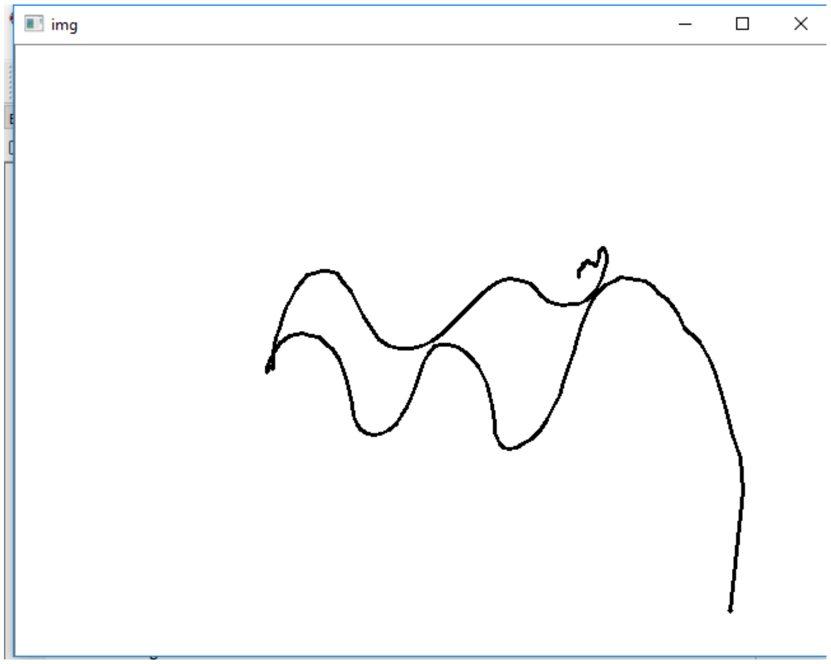


Figure : line drawing

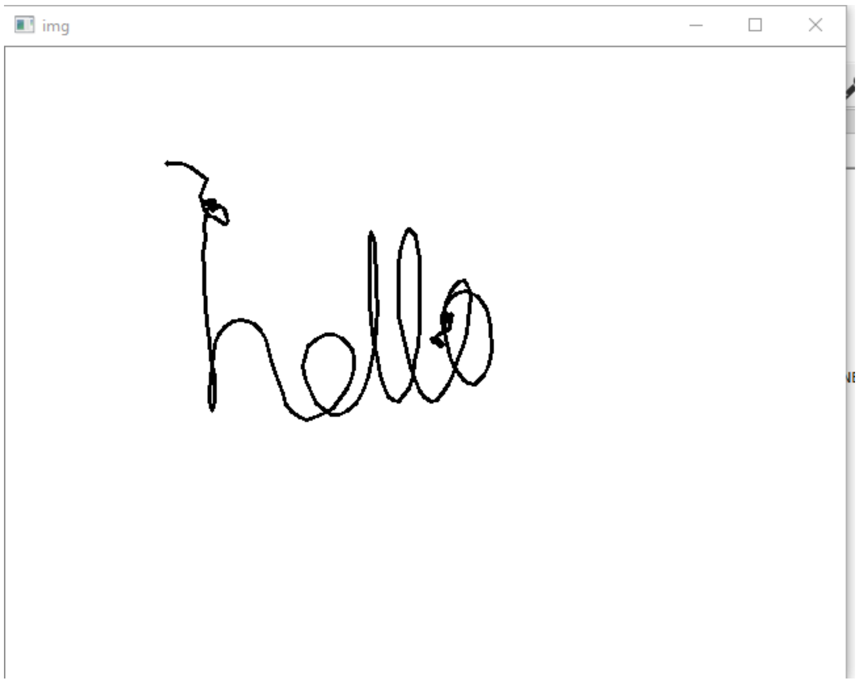


Figure : Gesture writing

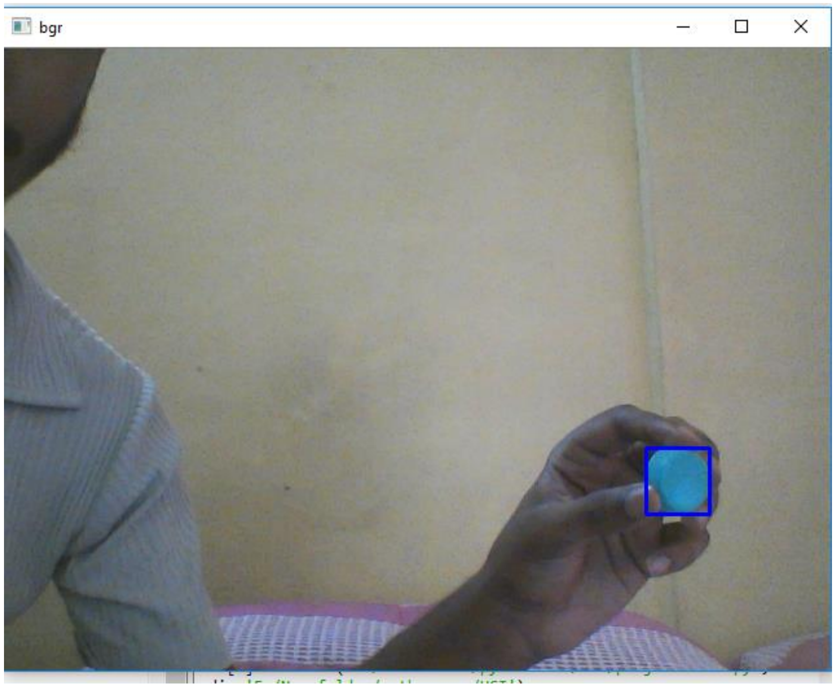


Figure : object detection

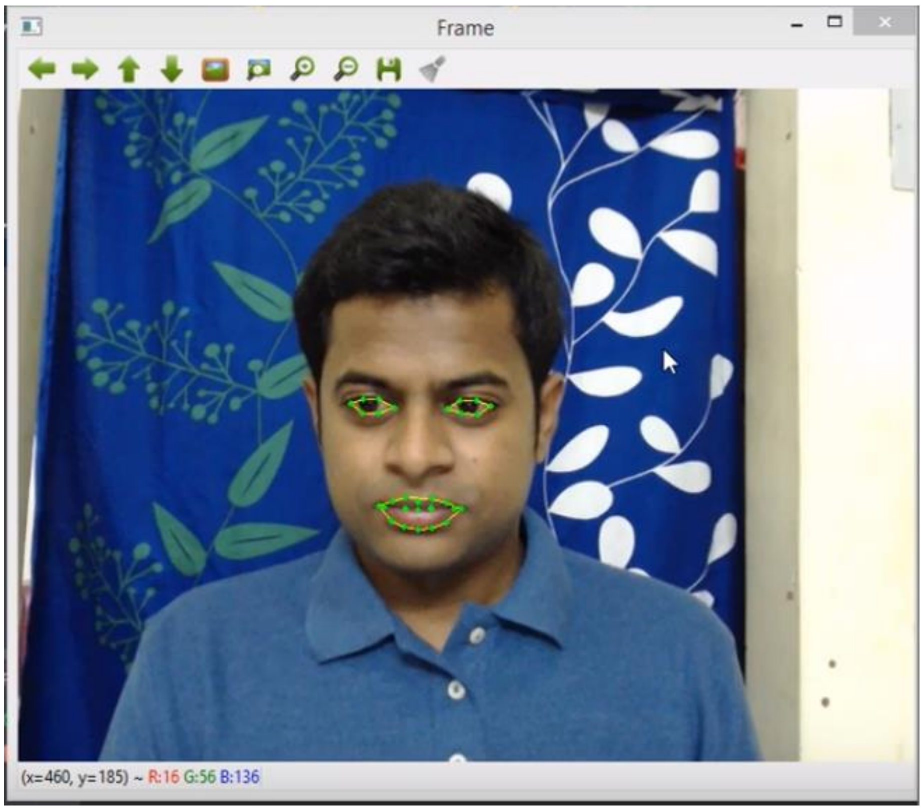


Figure : Facial Landmarks

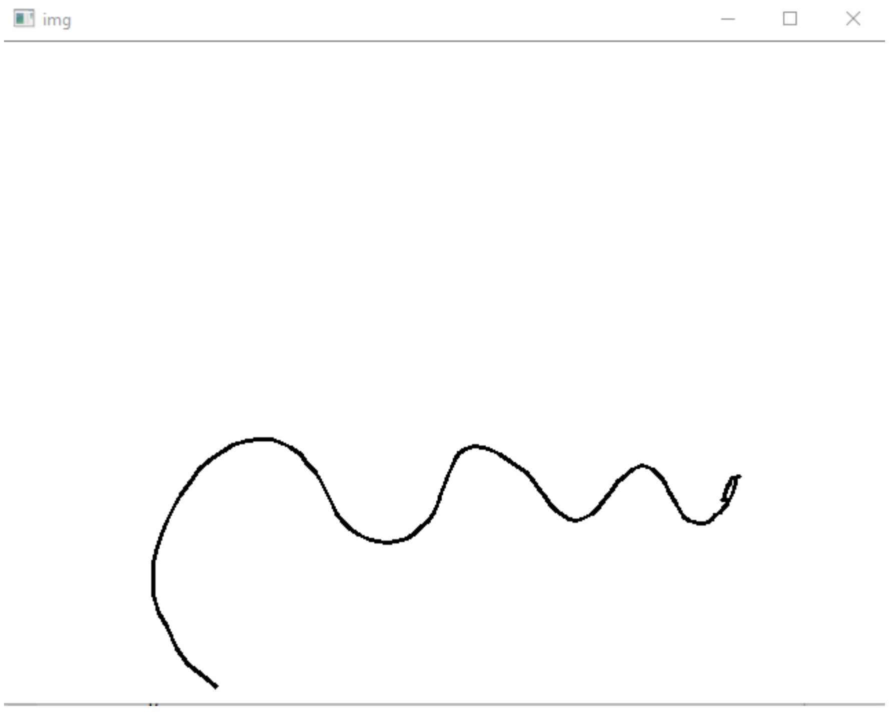


Figure : Sample line

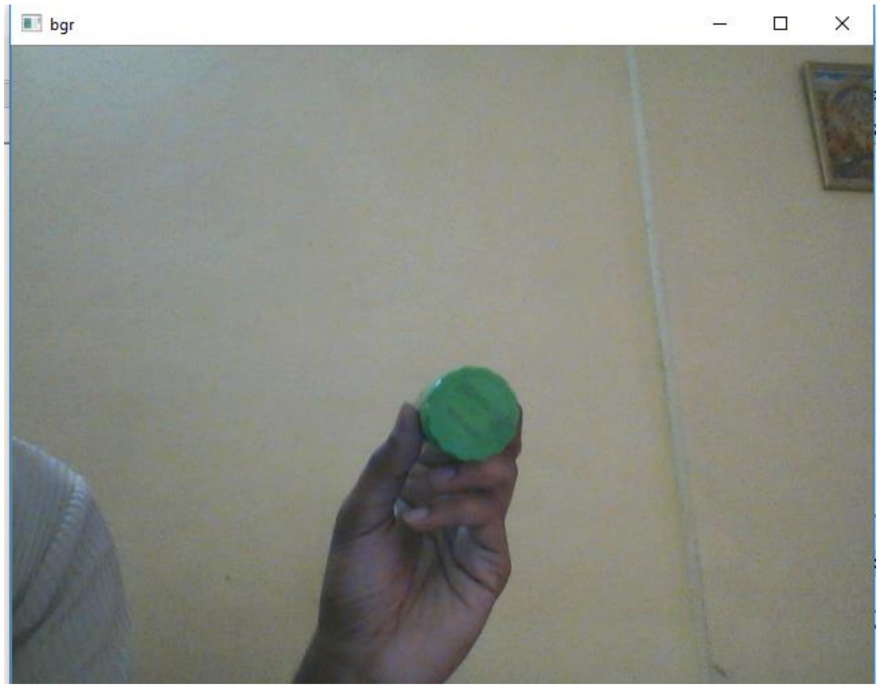


Figure : object with similar shade

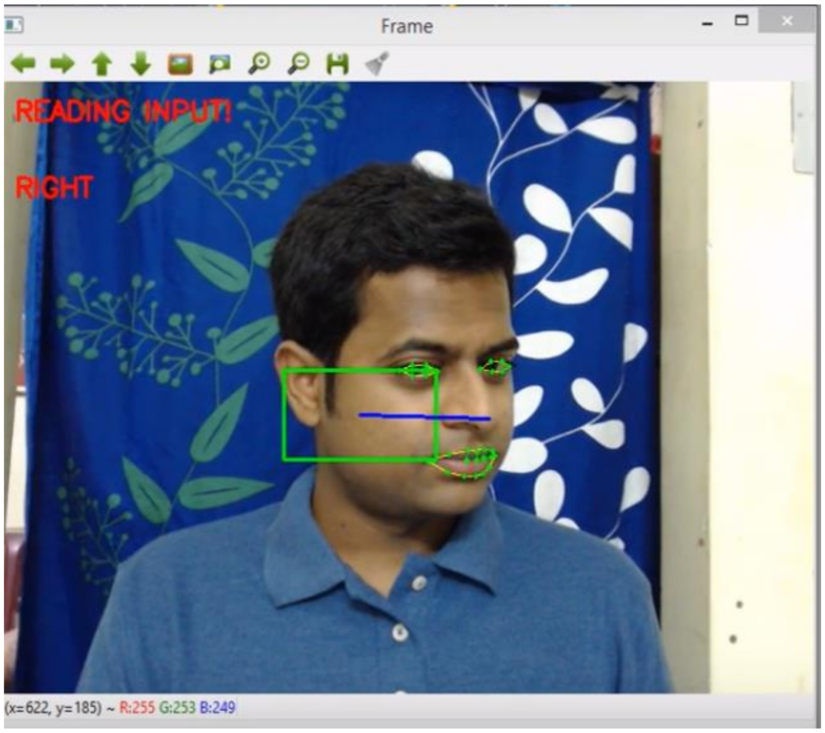
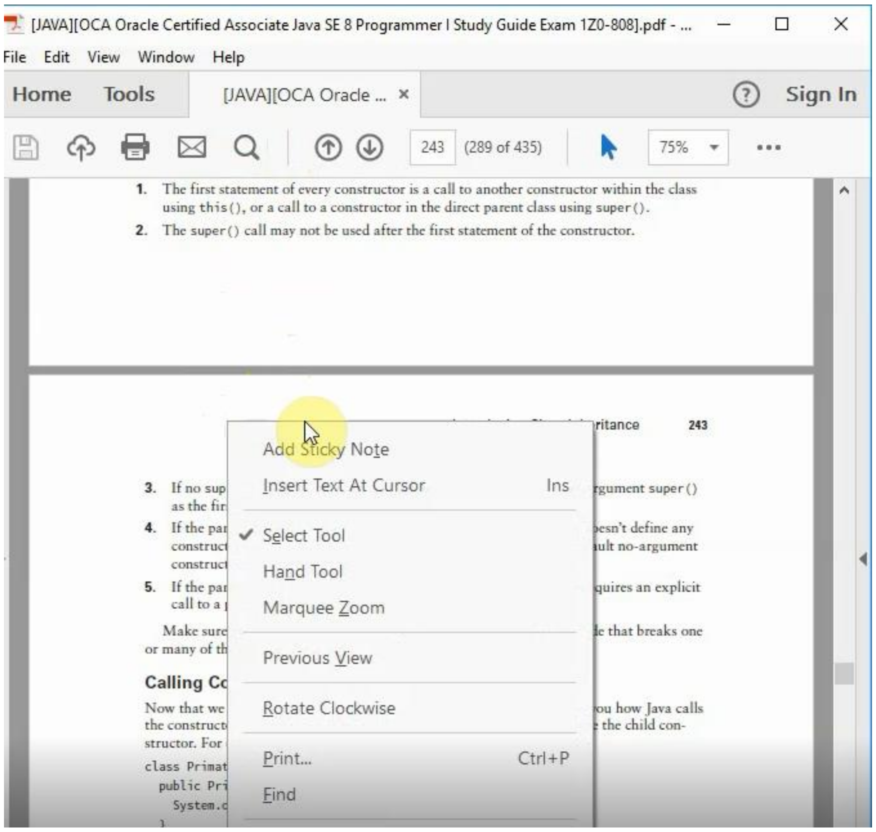
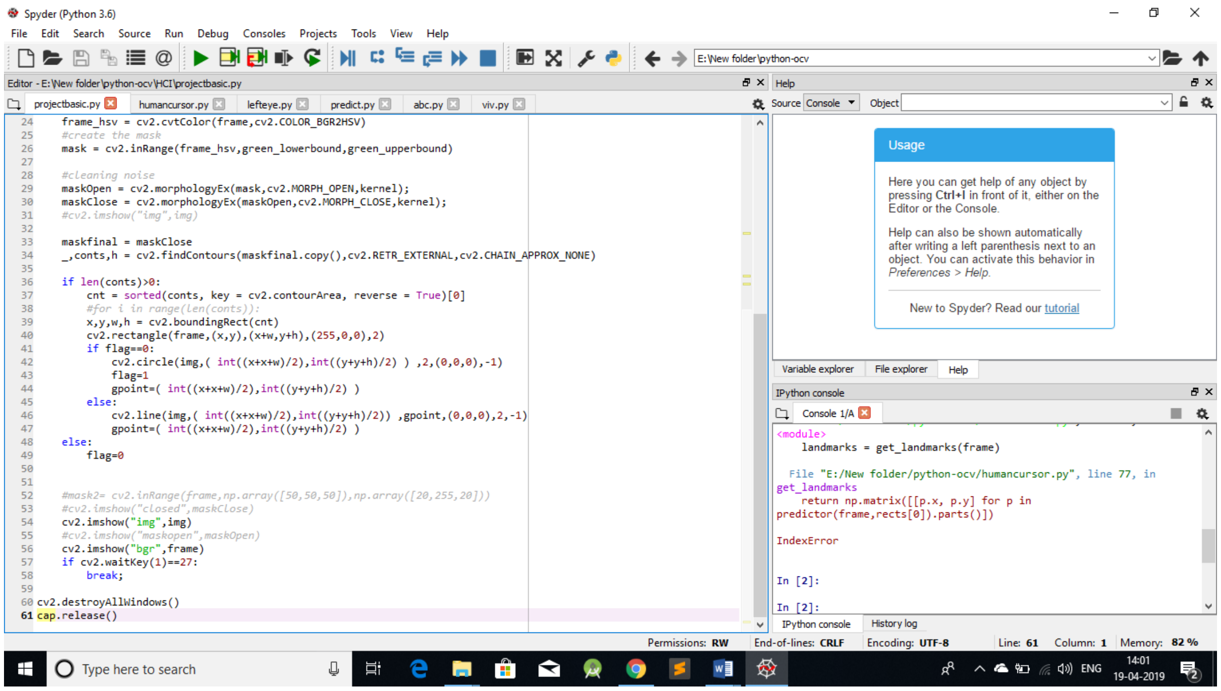


Figure : Reading Input


[JAVA][OCA Oracle Certified Associate Java SE 8 Programmer I Study Guide Exam 1Z0-808].pdf - ...
File Edit View Window Help
Home Tools [JAVA][OCA Oracle ...
Sign In
243 (289 of 435)
75%
Add Sticky Note
Insert Text At Cursor Ins
Select Tool
Hand Tool
Marquee Zoom
Previous View
Rotate Clockwise
Print... Ctrl+P
Find

1. The first statement of every constructor is a call to another constructor within the class using this(), or a call to a constructor in the direct parent class using super().
2. The super() call may not be used after the first statement of the constructor.

Figure : Automation


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
E:\New folder\python-ocv
Editor - E:\New folder\python-ocv\HCI\projectbasic.py
projectbasic.py humancursor.py lefteye.py predict.py abc.py viv.py
Help
Source Console Object
Usage
Variable explorer File explorer Help
IPython console
Console 1/A
IPython console History log
Type here to search

```
    frame_hsv = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
    #create the mask
    mask = cv2.inRange(frame_hsv,green_lowerbound,green_upperbound)

    #cleaning noise
    maskOpen = cv2.morphologyEx(mask,cv2.MORPH_OPEN,kernel);
    maskClose = cv2.morphologyEx(maskOpen,cv2.MORPH_CLOSE,kernel);
    #cv2.imshow("img",img)

    maskfinal = maskClose
    _,conts,h = cv2.findContours(maskfinal.copy(),cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_NONE)

    if len(conts)>0:
        cnt = sorted(conts, key = cv2.contourArea, reverse = True)[0]
        #for i in range(len(conts)):
        x,y,w,h = cv2.boundingRect(cnt)
        cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)
        if flag==0:
            cv2.circle(img,( int((x+x+w)/2),int((y+y+h)/2) ) ,2,(0,0,0),-1)
            flag=1
            gpoint=( int((x+x+w)/2),int((y+y+h)/2) )
        else:
            cv2.line(img,( int((x+x+w)/2),int((y+y+h)/2)) ,gpoint,(0,0,0),2,-1)
            gpoint=( int((x+x+w)/2),int((y+y+h)/2) )
    else:
        flag=0


    #mask2= cv2.inRange(frame,np.array([50,50,50]),np.array([20,255,20]))
    #cv2.imshow("closed",maskClose)
    cv2.imshow("img",img)
    #cv2.imshow("maskopen",maskOpen)
    cv2.imshow("bgr",frame)
    if cv2.waitKey(1)==27:
        break;

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

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

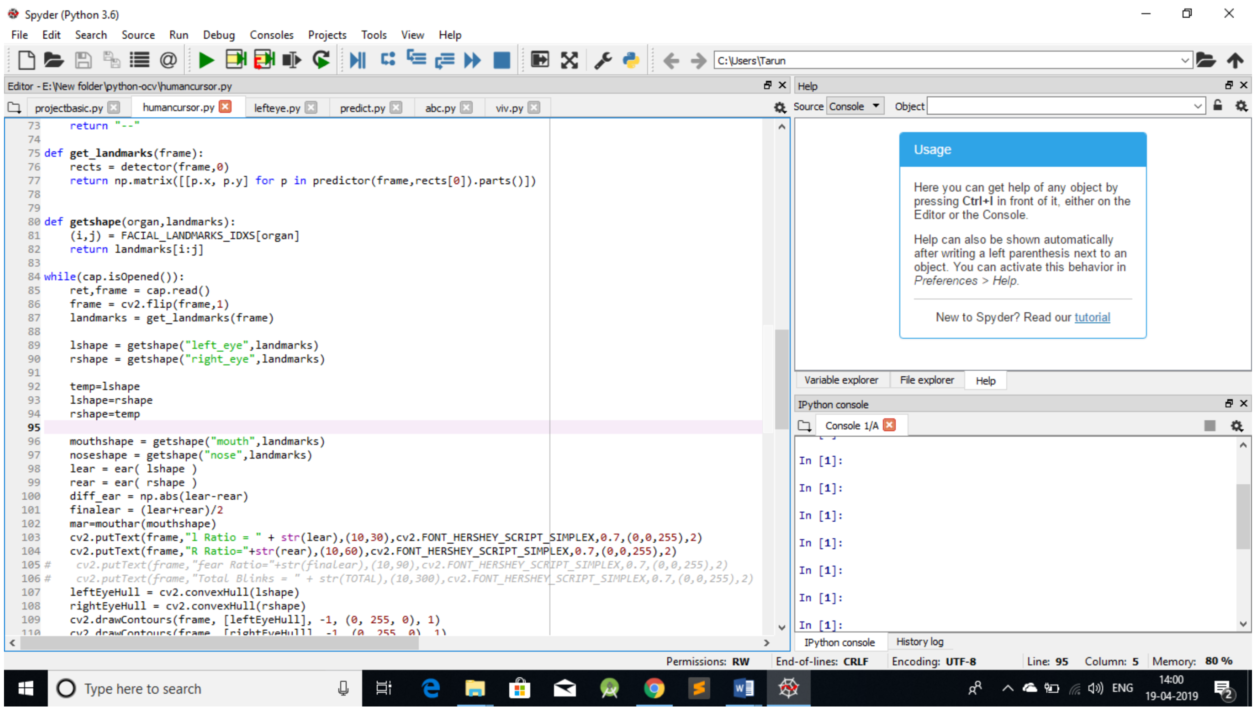
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

```
<module>
    landmarks = get_landmarks(frame)

  File "E:/New folder/python-ocv/humancursor.py", line 77, in
get_landmarks
    return np.matrix([[p.x, p.y] for p in
predictor(frame,rects[0]).parts()])

IndexError


In [2]:

In [2]:
```

Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 61 Column: 1 Memory: 82 %

Figure : spyder snippet 1



```
    return "--"

def get_landmarks(frame):
    rects = detector(frame,0)
    return np.matrix([[p.x, p.y] for p in predictor(frame,rects[0]).parts()])


def getshape(organ,landmarks):
    (i,j) = FACIAL_LANDMARKS_IDXS[organ]
    return landmarks[i:j]

while(cap.isOpened()):
    ret,frame = cap.read()
    frame = cv2.flip(frame,1)
    landmarks = get_landmarks(frame)

    lshape = getshape("left_eye",landmarks)
    rshape = getshape("right_eye",landmarks)

    temp=lshape
    lshape=rshape
    rshape=temp

    mouthshape = getshape("mouth",landmarks)
    noseshape = getshape("nose",landmarks)
    lear = ear( lshape )
    rear = ear( rshape )
    diff_ear = np.abs(lear-rear)
    finalear = (lear+rear)/2
    mar=mouthar(mouthshape)
    cv2.putText(frame,"l Ratio = " + str(lear),(10,30),cv2.FONT_HERSHEY_SCRIPT_SIMPLEX,0.7,(0,0,255),2)
    cv2.putText(frame,"R Ratio="+str(rear),(10,60),cv2.FONT_HERSHEY_SCRIPT_SIMPLEX,0.7,(0,0,255),2)
#    cv2.putText(frame,"fear Ratio="+str(finalear),(10,90),cv2.FONT_HERSHEY_SCRIPT_SIMPLEX,0.7,(0,0,255),2)
#    cv2.putText(frame,"Total Blinks = " + str(TOTAL),(10,300),cv2.FONT_HERSHEY_SCRIPT_SIMPLEX,0.7,(0,0,255),2)
    leftEyeHull = cv2.convexHull(lshape)
    rightEyeHull = cv2.convexHull(rshape)
    cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)
    cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)
```

```
In [1]:

In [1]:

In [1]:

In [1]:

In [1]:

In [1]:

In [1]:
```

Usage

Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

Figure : spyder snippet 2

FUTURE ENHANCEMENTS

The accuracy of the existing system will be made approximately up to 100% to detect the gestures and objects more accurately.

The anomalies with respect to noise in the background and negative input values can be eliminated to improve the accuracy and efficiency.

Providing brief data about specific objects or faces to further enhance the face detection to face recognition and hence minimize the unauthorized access to the application.

CONCLUSION

Various techniques were used to predict the Interaction between human and a computer with respect to using a physically wired device or wireless handsfree device. These methods were not completely handsfree as we always have to use that specific device to interact.

However, in recent years, with the advancement in technology, it has been possible to communicate with the computer using techniques like gesture recognition and voice recognition using various computer vision and machine learning algorithms.

In our project, we implemented these algorithms to detect the object,detect the face and thus perform various functions including gesture tracking/object tracking, facial landmark detection and finally the automation based on the given gestures.

REFERENCES

1. ***https://docs.opencv.org/2.4***
2. ***https://www.kaggle.com/faciallandmarks/data***
3. ***https://machinelearningmastery.com/***
4. ***http://en.wikipedia.org/wiki/haar-like_feature***
5. ***http://www.numpy.org***

BIBLIOGRAPHY

1. Learning OpenCV by Gary Bradski
2. Programming with Python by David Ascher and Mark Lutz
3. Data Preprocessing by Jun Du
4. Introduction to Machine Learning with Python: A guide for Data Scientist by Andreas