A PROJECT REPORT ON

"OBJECT DETECTION"

Submitted

In partial fulfillment of the
requirements for the Degree of

MASTERS OF COMPUTER APPLICATIONS

By

AKHIL GUSAIN

(Enrollment Number: PV-21010727)

Under the Supervision of

Dr. NISHA CHANDRAN

(Professor, GEHU)

SCHOOL OF COMPUTING

GRAPHIC ERA HILL UNIVERSITY, DEHRADUN

(2020-2022)

CERTIFICATE

I hereby declare that the work which is being presented in the project entitled, “Object Detection” has been carried out by **AKHIL GUSAIN** for the partial fulfilment of the requirements for the award of the MCA, submitted in the School of Computing, **GRAPHIC ERA HILL UNIVERSITY** is an authentic record of our own **Dr. Nisha chadran ,**. I further declare that the matter embodied in this project has not been submitted by us for the award of any other degree.

PROJECT IN CHARGE

Dr. NISHA CHANDRAN

(Assistant Professor)

ACKNOWLEDGEMENT

This is a great opportunity to acknowledge and to thanks all those persons without whose support and help this project would have been impossible. I would like to add a few heartfelt words for the people who were part of this project in numerous ways.

I am obliged and thankful to my project SUPERVISOR, **Dr. NISHA CHANDRAN**, for her continuous encouragement, motivation and professional guidance during the work of this project which has proven to be an integral part of it. Without her valuable support and guidance, this project could not elevate up this level of development from our point of view. I would like to thank all the Faculty members, School Of Computing, Graphic Era Hill University for their valuable time spent in requirements analysis and evaluation of the project work.

I would like to express my sincere and cordial gratitude to the people those who have supported me directly, purveyed mental encouragement, evaluated and criticized my work in several phases during the development of this project and for preparing this dissertation indirectly.

ABSTRACT

Efficient and accurate Object Detection has been an important topic in the advancement of computer vision systems. With the advent of machine learning and deep learning techniques,the accuracy for Object Detection has increased drastically. The project aims to in corporate state-of-the-art technique for Object Detection with the goal of achieving high accuracy with areal-time performance. In this project, we use a completely machine learning with open cv and deep learning based approach to solve the problem of Object Detection in an end-to-end fashion.The network is trained on the most challenging publicly available data set, on which a Object Detection challenge is conducted annually. The resulting system is fast and accurate, thus aiding those applications which require Object Detection

The main objective of this project is:

· To save the time of the user

· To provide best information within one place

· Simple to navigate

· To provide user friendly and interactive platform

•

INDEX

CHAPTER 1: INTRODUCTION

• INTRODUCTION

Object Detection program is a project through which you will able to get the name of object correctly and it could tell you the difference between overflowing garbage or not, if overflowing ,a serious garbage management is required. It uses the python programming language and its different libraries like open cv that is open computer vision,numpy ,os etc.

It uses computer's camera as its eye to detect objects and its name, it uses different algorithms to do so this looks like stepping into the future(A.I) .

• AIM

Object Detection is something everybody deals with in general life ,a human can detect objects and its features because we have our own intelligence while a computer or a machine can't until we feed it with prior information and data(supervised) and create a Artificial intelligence.

Undoubtedly, Object Detection with open cv has come a long way, helping people to know about Object Detection with computer's camera. So,if you want machines not only be developed enough to undertake tasks that humans perform, but to do so in a way that humans do involving 'thought' and action derived from intelligence.

• EXISTING SYSTEM

Previously built Object Detection project are so vast in nature that it can detect 80+ objects with it but it uses a coco model it is basically a list of objects compatible with system and every time users uses this it only detect this objects only , they have to bound with this system only. Due to complex coding, system responding time was high and require more memory to get start up. The concept of feature detection and description for objects was not implemented in this coco model version. Dynamic concept was not implemented under the existing system.

• PROPOSED SYSTEM

Under Object Detection project we don’t get fixated with 80 objects but also it determines the name for it based on the features it has detected from its supervised data. Its pattern recognition system will able to detect features with image processing and it uses ORB system of open cv(oriented fast and rotated brief) instead of coco model.

• FEASIBILITY STUDY

A feasibility study is a high-level capsule version of the entire System analysis and Design Process. The study begins by classifying the problem definition. Feasibility is to determine if it's worth doing. Once an acceptance problem definition has been generated, the analyst develops a logical model of the system. A search for alternatives is analysed carefully. There are 3 parts in feasibility study.

● Operational Feasibility

● Technical Feasibility

● Economical Feasibility

• OPERATIONAL FEASIBILITY

Operational feasibility is the measure of how well a proposed system solves the problems, and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development. The operational feasibility assessment focuses on the degree to which the proposed development projects fits in with the existing business environment and objectives with regard to development schedule, delivery date, corporate culture and existing business processes. To ensure success, desired operational outcomes must be imparted during design and development. These include such design-dependent parameters as reliability, maintainability, supportability , usability, productability, disposability, sustainability, affordability and others. These parameters are required to be considered at the early stages of design if desired operational behaviour are to be released. A system design and development requires appropriate and timely application of engineering and management efforts to meet the previously mentioned parameters. A system may serve its intended purpose most effectively when its technical and operating characteristics are engineered into the design. Therefore, operational feasibility is a critical aspect of systems engineering that needs to be an integral part of the early design phases.

• TECHNICAL FEASIBILITY

This involves questions such as whether the technology needed for the system exists, how difficult it will be to build, and whether the firm has enough experience using that technology. The assessment is based on outline design of system requirements in terms of input, processes, output, fields, programs and procedures. This can be qualified in terms of volume of data, trends, frequency of updating in order to give an introduction to the technical system. The application is the fact that it has been developed on windows XP platform and a high configuration of 1GB RAM on Intel Pentium Dual core processor. This is technically feasible .The technical feasibility assessment is focused on gaining an understanding of the present technical resources of the organization and their applicability to the expected needs of the proposed system. It is an evaluation of the hardware and software and how it meets the need of the proposed system.

• ECONOMICAL FEASIBILITY

Establishing the cost-effectiveness of the proposed system i.e. if the benefits do not outweigh the costs then it is not worth going ahead. In the fast paced world today there is a great need of online social networking facilities. Thus the benefits of this project in the current scenario make it economically feasible. The purpose of the economic feasibility assessment is to determine the positive economic benefits to the organization that the proposed system will provide. It includes quantification and identification of all the benefits expected. This assessment typically involves a cost/benefits analysis.

ORGANISATION OF REPORT

- ## INTRODUCTION

This section includes the overall view of the project i.e. the basic problem definition and the general overview of the problem which describes the problem in layman terms. It also specifies the software used and the proposed solution strategy.

- ## SOFTWARE REQUIREMENTS SPECIFICATION

This section includes the Software and hardware requirements for the smooth running of the application.

- ## DESIGN & PLANNING

This section consists of the Software Development Life Cycle model. It also contains technical diagrams like the Data Flow Diagram and the Entity Relationship diagram.

- ## IMPLEMENTATION DETAILS

This section describes the different technologies used for the entire development process of the Front-end as well as the Back-end development of the application.

- ## RESULTS AND DISCUSSION

This section has screenshots of all the implementation i.e. user interface and their description.

• SUMMARY AND CONCLUSION

This section has screenshots of all the implementation i.e. user interface and their description.

CHAPTER 2: SOFTWARE REQUIREMENTS SPECIFICATION

- **HARDWARE REQUIREMENTS**

Number	**Description**
1	PC with 250 GB or more of HDD.
2	PC with 2 GB RAM.
3	PC with Pentium 1 or Above.

- **SOFTWARE REQRUIREMENTS**

Number	**Description**	**Type**
1	Operating System	Windows / Linux / Mac
2	Language	Python
4	IDE	Visual Studio Code
5	Browser	Google Chrome

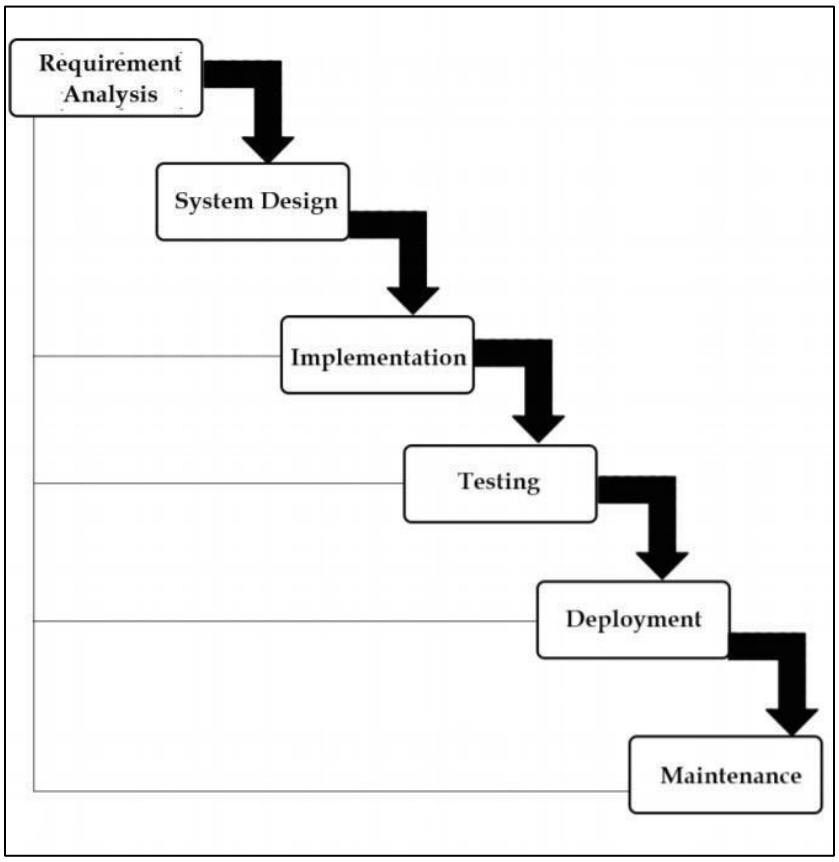
CHAPTER 3: DESIGNING AND PLANNING

● SOFTEWARE DEVELOPMENT LIFE CYCLE MODEL

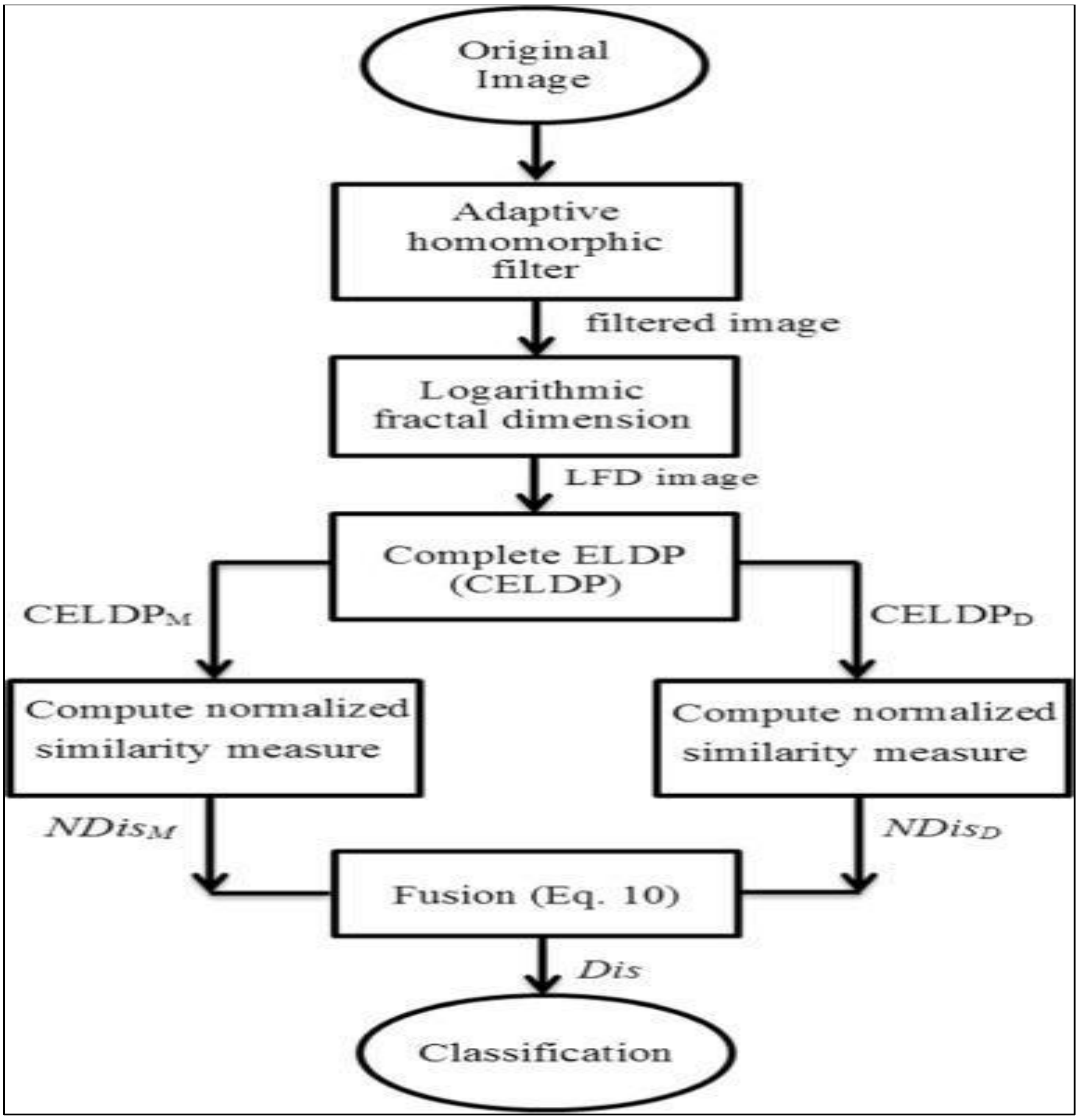
• WATERFALL MODEL

The waterfall model was selected as the SDLC model due to the following reasons:

- Requirements were very well documented, clear and fixed.
- Technology was adequately understood.
- Simple and easy to understand and use.
- There were no ambiguous requirements.
- Easy to manage due to the rigidity of the model.
- Each phase has specific delivery and a review process.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks


Requirement Analysis
System Design
Implementation
Testing
Deployment
Maintenance

- **DFD DIAGRAM**


Original Image
Adaptive homomorphic filter
filtered image
Logarithmic fractal dimension
LFD image
Complete ELDP (CELDP)
$CELDP_M$
$CELDP_D$
Compute normalized similarity measure
Compute normalized similarity measure
$NDis_M$
$NDis_D$
Fusion (Eq. 10)
Dis
Classification

CHAPTER 4: IMPLENENTATION DETAILS

PYTHON

Python is an interpreted high-level general-purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

Guido van Rossum began working on Python in the late 1980s, as a successor to the ABC programming language, and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features, such as list comprehensions and a garbage collection system using reference counting. Python 3.0 was released in 2008 and was a major revision of the language that is not completely backward-compatible. Python 2 was discontinued with version 2.7.18 in 2020.

OPEN CV

OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as Numpy which is a highly optimized library for numerical operations, then the number of weapons increases in your Arsenal i.e whatever operations one can do in Numpy can be combined with OpenCV.

This OpenCV tutorial will help you learn the Image-processing from Basics to Advance, like operations on Images, Videos using a huge set of Opencv-programs and projects.

Applications of OpenCV: There are lots of applications which are solved using OpenCV, some of them are listed below

- face recognition
- Automated inspection and surveillance
- number of people – count (foot traffic in a mall, etc)
- Vehicle counting on highways along with their speeds
- Interactive art installations
- Anamoly (defect) detection in the manufacturing process (the odd defective products)
- Street view image stitching
- Video/image search and retrieval
- Robot and driver-less car navigation and control
- object recognition
- Medical image analysis
- Movies – 3D structure from motion
- TV Channels advertisement recognition

NumPy :

NumPy is a Python module. The name NumPy represents :Numerical Python and it is utilized. It is an expansion module for Python, generally written in C. This guarantees the precompiled logical and numerical limits and functionalities of Numpy guarantee remarkable execution speed.
Numpy is mostly used for performing calculations using certain functions it provides like multiply, divide, power etc. It is basically used for performing complex calculations with ease like dot product, summation etc. It is a very efficient module and makes the work easier.

CHAPTER 5: TESTING

5.1 UNIT TESTING

5.1.1 INTRODUCTION

In computer programming, unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application. In procedural programming, a unit could be an entire module, but it is more commonly an individual function or procedure. In object-oriented programming, a unit is often an entire interface, such as a class, but could be an individual method. Unit tests are short code fragments created by programmers or occasionally by white box testers during the development process. It forms the basis for component testing. Ideally, each test case is independent from the others. Substitutes such as method stubs, mock objects, fakes, and test harnesses can be used to assist testing a module in isolation. Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended.

5.1.2 BENEFITS

The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. A unit test provides a strict, written contract that the piece of code must satisfy. As a result, it affords several benefits

4 **Find problems early:** Unit testing finds problems early in the development cycle. In test-driven development (TDD), which is frequently used in both extreme programming and scrum, unit tests are created before the code itself is written. When the tests pass, that code is considered complete. The same unit tests are run against that function frequently as the larger code base is developed either as the code is changed or via an automated process with the build. If the unit tests fail, it is considered to be a bug either in the changed code or the tests themselves. The unit tests then allow the location of the fault or failure to be easily traced. Since the unit tests alert the development JGVDDteam of the problem before handing the code off to testers or clients, it is still early in the development process.

5 **Facilitates Change:** Unit testing allows the programmer to refactor code or upgrade system libraries at a later date, and make sure the module still works correctly (e.g., in regression testing). The procedure is to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified. Unit tests detect changes which may break a design contract.

6 **Simplifies Integration:** Unit testing may reduce uncertainty in the units themselves and can be used in a bottom-up testing style approach. By testing the parts of a program first and then testing the sum of its parts, integration testing becomes much easier.

7 **Documentation:** Unit testing provides a sort of living documentation of the system. Developers looking to learn what functionality is provided by a unit, and how to use it, can look at the unit tests to gain a basic understanding of the unit's interface (API).Unit test cases embody characteristics that are critical to the success of the unit. These characteristics can indicate appropriate/inappropriate use of a unit as well as negative behaviours that are to be trapped by the unit.

5.2: INTEGRATION TESTING

Integration testing (sometimes called integration and testing, abbreviated I&T) is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

5.2.1 PURPOSE

The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items. These "design items", i.e., assemblages (or groups of units), are exercised through their interfaces using black-box testing, success and error cases being simulated via appropriate parameter and data inputs. Simulated usage of shared data areas and inter-process communication is tested and individual subsystems are exercised through their input interface. Test cases are constructed to test whether all the components within assemblages interact correctly, for example across procedure calls or process activations, and this is done after testing individual modules, i.e., unit testing. The overall idea is a "building block" approach, in which verified assemblages are added to a verified base which is then used to support the integration testing of further assemblages. Software integration testing is performed according to the software development life cycle (SDLC) after module and functional tests. The cross dependencies for software integration testing are: schedule for integration testing, strategy and selection of the tools used for integration, define the cyclomatic complexity of the software and software architecture, reusability of modules and life- cycle and versioning management. Some different types of integration testing are big-bang, top-down, and bottom-up, mixed (sandwich) and risky-hardest. Other Integration Patterns are: collaboration integration, backbone integration, layer integration, client-server integration, distributed services integration and high- frequency integration.

5.2.1.1 Big Bang

In the big-bang approach, most of the developed modules are coupled together to form a complete software system or major part of the system and then used for integration testing. This method is very effective for saving time in the integration testing process. However, if the test cases and their results are not recorded properly, the entire integration process will be more complicated and may prevent the testing team from achieving the goal of integration testing. A type of big-bang integration testing is called "usage model testing" which can be used in both software and hardware integration testing. The basis behind this type of integration testing is to run user- like workloads in integrated user-like environments. In doing the testing in this manner, the environment is proofed, while the individual components are proofed indirectly through their use. Usage Model testing takes an optimistic approach to testing, because it expects to have few problems with the individual components. The strategy relies heavily on the component developers to do the isolated unit testing for their product. The goal of the strategy is to avoid redoing the testing done by the developers, and instead flesh-out problems caused by the interaction of the components in the environment.

5.2.1.2 Top-down and Bottom-up

Bottom-up testing is an approach to integrated testing where the lowest level components are tested first, then used to facilitate the testing of higher level components. The process is repeated until the component at the top of the hierarchy is tested. All the bottom or low-level modules, procedures or functions are integrated and then tested. After the integration testing of lower level integrated modules, the next level of modules will be formed and can be used for integration testing. This approach is helpful only when all or most of the modules of the same development level are ready. This method also helps to determine the levels of software developed and makes it easier to report testing progress in the form of a percentage. Top-down testing is an approach to integrated testing where the top integrated modules are tested and the branch of the module is tested step by step until the end of the related module. Sandwich testing is an approach to combine top down testing with bottom up testing.

5.3: SOFTWARE VERIFICATION AND VALIDATION

5.3.1 Introduction

In software project management, software testing, and software engineering, verification and validation (V&V) is the process of checking that a software system meets specifications and that it fulfils its intended purpose. It may also be referred to as software quality control. It is normally the responsibility of software testers as part of the software development lifecycle. Validation checks that the product design satisfies or fits the intended use (high-level checking), i.e., the software meets the user requirements. This is done through dynamic testing and other forms of review. Verification and validation are not the same thing, although they are often confused. Boehm succinctly expressed the difference between

3 Validation: Are we building the right product?

4 Verification: Are we building the product right?

According to the Capability Maturity Model (CMMI-SW v1.1)

Software Verification: The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

Software Validation: The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements.

In other words, software verification is ensuring that the product has been built according to the requirements and design specifications, while software validation ensures that the product meets the user's needs, and that the specifications were correct in the first place. Software verification ensures that "you built it right". Software validation ensures that "you built the right thing". Software validation confirms that the product, as provided, will fulfil its intended use.

From Testing Perspective

5 Fault – wrong or missing function in the code.

6 Failure – the manifestation of a fault during execution.

7 Malfunction – according to its specification the system does not meet its specified functionality

Both verification and validation are related to the concepts of quality and of software quality assurance. By themselves, verification and validation do not guarantee software quality; planning, traceability, configuration management and other aspects of software engineering are required. Within the modelling and simulation (M&S) community, the definitions of verification, validation and accreditation are similar:

7.1 M&S Verification is the process of determining that a • computer model, simulation, or federation of models and simulations implementations and their associated data accurately represent the developer's conceptual description and specifications.

7.2 M&S Validation is the process of determining the degree to which a model, simulation, or federation of models and simulations, and their associated data are accurate representations of the real world from the perspective of the intended use(s)

5.3.2 Classification of Methods

In mission-critical software systems, where flawless performance is absolutely necessary, formal methods may be used to ensure the correct operation of a system. However, often for non-mission critical software systems, formal methods prove to be very costly and an alternative method of software V&V must be sought out. In such cases, syntactic methods are often used.

5.3.3 Test Cases

A test case is a tool used in the process. Test cases may be prepared for software verification and software validation to determine if the product was built according to the requirements of the user. Other methods, such as reviews, may be used early in the life cycle to provide for software validation

5.4: Black-Box Testing

Black-box testing is a method of software testing that examines the functionality of an application without peering into its internal structures or workings. This method of test can be applied virtually to every level of software testing: unit, integration, system and acceptance. It typically comprises most if not all higher level testing, but can also dominate unit testing as well.

5.4.1 Test Procedures

Specific knowledge of the application's code/internal structure and programming knowledge in general is not required. The tester is aware of what the software is supposed to do but is not aware of how it does it. For instance, the tester is aware that a particular input returns a certain, invariable output but is not aware of how the software produces the output in the first place.

5.4.2 Test Cases

Test cases are built around specifications and requirements, i.e., what the application is supposed to do. Test cases are generally derived from external descriptions of the software, including specifications, requirements and design parameters. Although the tests used are primarily functional in nature, non-functional tests may also be used. The test designer selects both valid and invalid inputs and determines the correct output, often with the help of an oracle or a previous result that is known to be good, without any knowledge of the test object's internal structure.

5.5: White-Box Testing

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing). In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g. in-circuit testing (ICT). White-box testing can be applied at the unit, integration and system levels of the software testing process. Although traditional testers tended to think of white-box testing as being done at the unit level, it is used for integration and system testing more frequently today. It can test paths within a unit, paths between units during integration, and between subsystems during a system–level test. Though this method of test design can uncover many errors or problems, it has the potential to miss unimplemented parts of the specification or missing requirements.

5.5.1 Levels

2 **Unit testing:** White-box testing is done during unit testing to ensure that the code is working as intended, before any integration happens with previously tested code. White-box testing during unit testing catches any defects early on and aids in any defects that happen later on after the code is integrated with the rest of the application and therefore prevents any type of errors later on.

3 **Integration testing:** White-box testing at this level are written to test the interactions of each interface with each other. The Unit level testing made sure that each code was tested and working accordingly in an isolated environment and integration examines the correctness of the behaviour in an open environment through the use of white-box testing for any interactions of interfaces that are known to the programmer.

4 **Regression testing:** White-box testing during regression testing is the use of recycled white-box test cases at the unit and integration testing levels.

5.5.2 Procedures

White-box testing's basic procedures involves the tester having a deep level of understanding of the source code being tested. The programmer must have a deep understanding of the application to know what kinds of test cases to create so that every visible path is exercised for testing. Once the source code is understood then the source code can be analysed for test cases to be created. These are the three basic steps that white-box testing takes in order to create test cases:

1) Input involves different types of requirements, functional specifications, detailed designing of documents, proper source code, security specifications. This is the preparation stage of white box testing to layout all of the basic information. Processing involves performing risk analysis to guide whole testing process, proper test plan, execute test cases and communicate results. This is the phase of building test cases to make sure they thoroughly test the application the given results are recorded accordingly.
2) Output involves preparing final report that encompasses all of the above preparations and results.

5.6: SYSTEM TESTING

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black-box testing, and as such, should require no knowledge of the inner design of the code or logic. As a rule, system testing takes, as its input, all of the "integrated" software components that have passed integration testing and also the software system itself integrated with any applicable hardware system(s). The purpose of integration testing is to detect any inconsistencies between the software units that are integrated together (called assemblages) or between any of the assemblages and the hardware. System testing is a more limited type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole.

System testing is performed on the entire system in the context of a Functional Requirement Specification(s) (FRS) and/or a System Requirement Specification (SRS). System testing tests not only the design, but also the behaviour and even the believed expectations of the customer. It is also intended to test up to and beyond the bounds defined in the software/hardware requirements specification(s)

CHAPTER 6: RESULTS

CAPTURE CAMERA INPUT

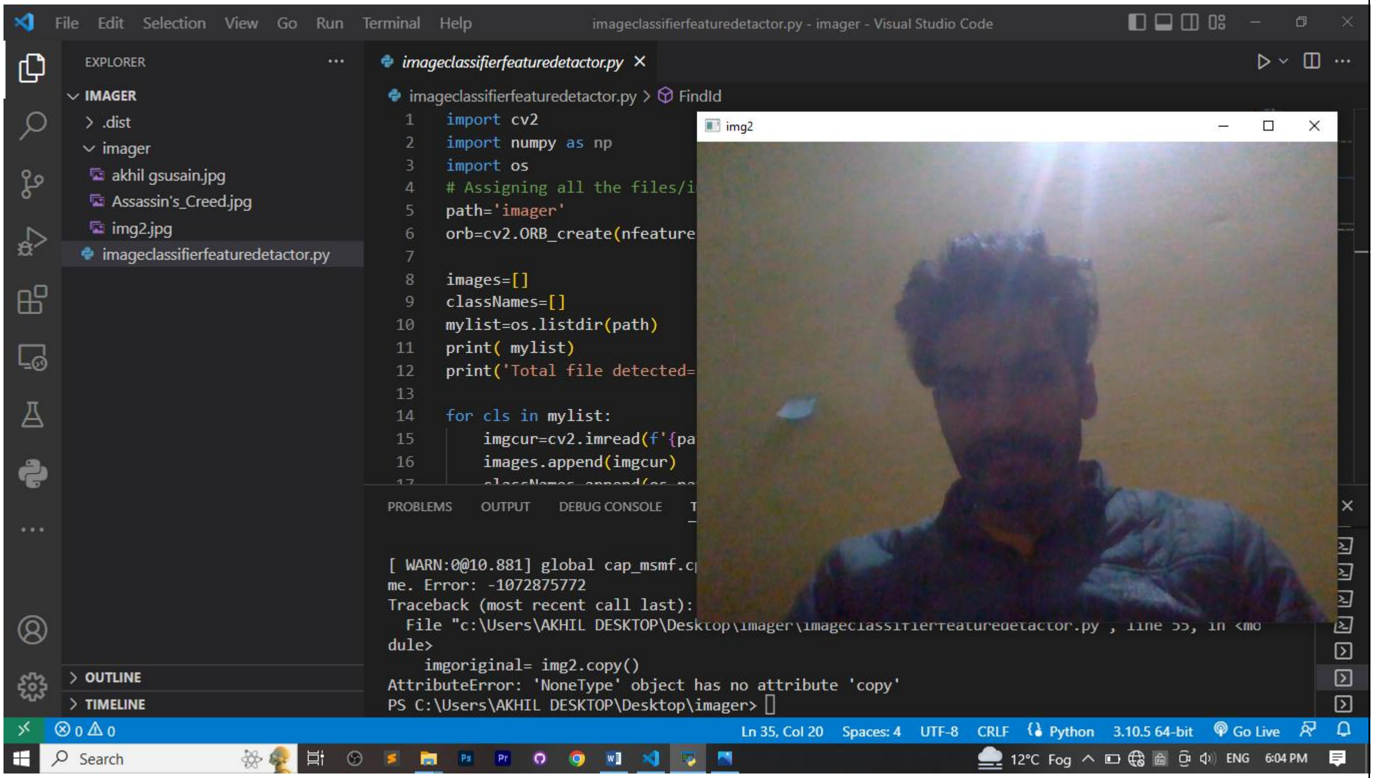
To capture the input camera, we have used the following code.

```
cv2.imshow('img2',imgoriginal)




cv2.waitKey(1)
```

The program will first detect the background, which will avoid the detection of any objects kept at rest. The objects around the hand histogram are avoided. To apply histogram, the client needs to evacuate their hand or any one part from the ideal box and ten needs to tap the necessary key on console to identify any undesirable items. To apply histogram the client at that point puts the deliver the histogram box and afterward press the necessary key



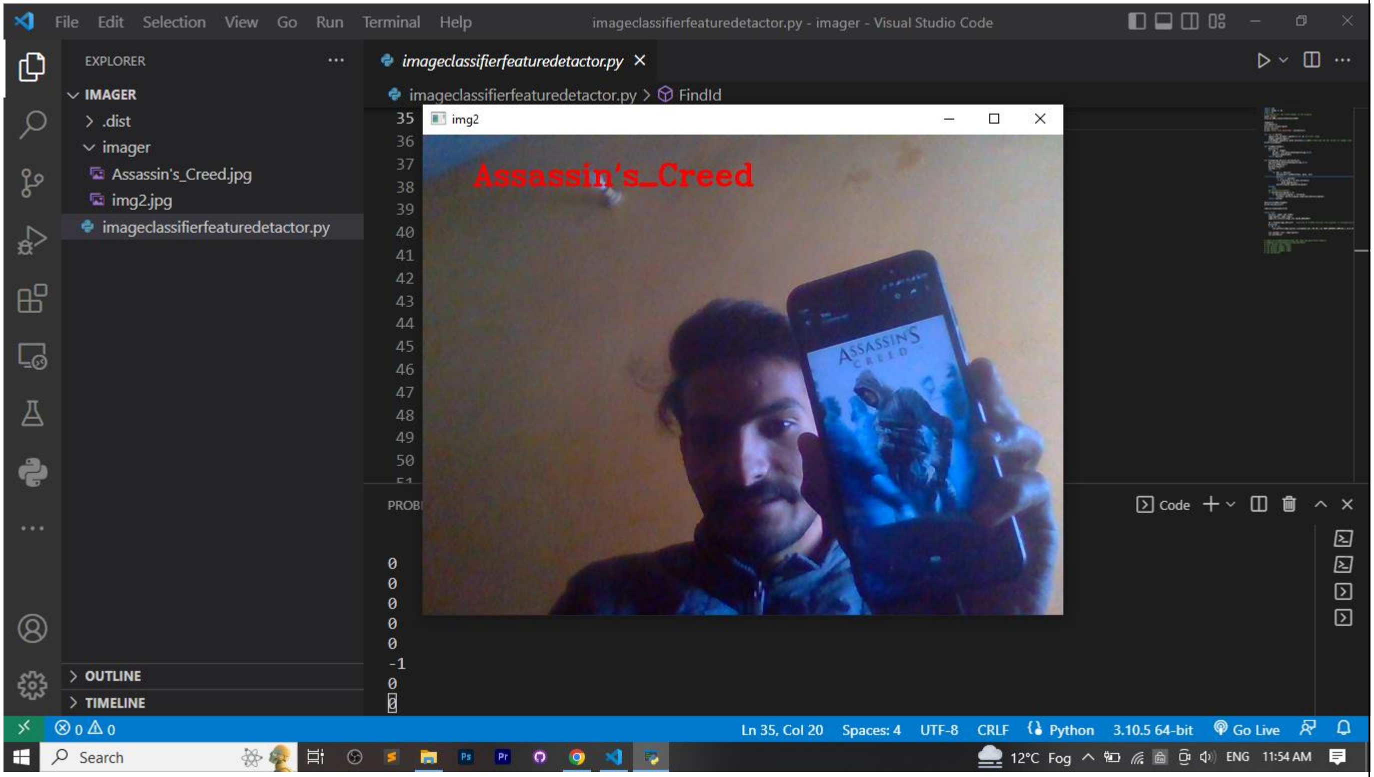
```
import cv2
import numpy as np
import os
# Assigning all the files/i
path='imager'
orb=cv2.ORB_create(nfeature

images=[]
classNames=[]
mylist=os.listdir(path)
print( mylist)
print('Total file detected=

for cls in mylist:
    imgcur=cv2.imread(f'{pa
    images.append(imgcur)
```

```
[ WARN:0@10.881] global cap_msmf.c
me. Error: -1072875772
Traceback (most recent call last):
  File "c:\Users\AKHIL DESKTOP\Desktop\imager\imageclassifierfeaturedetactor.py", line 55, in <mo
dule>
    imgoriginal= img2.copy()
AttributeError: 'NoneType' object has no attribute 'copy'
PS C:\Users\AKHIL DESKTOP\Desktop\imager>
```

Computer vision result without any object



```
0
0
0
0
0
-1
0
0
```

Showing the result's name as output

CODE:

```
import cv2
import numpy as np
import os
# Assigning all the files/images to the program
path='imager'
orb=cv2.ORB_create(nfeatures=1000)

images=[]
classNames=[]
mylist=os.listdir(path)
print( mylist)
print('Total file detected=',len(mylist))
```

```
for cls in mylist:
    imgcur=cv2.imread(f'{path}/{cls}',0) #current image
    images.append(imgcur)
    classNames.append(os.path.splitext(cls)[0]) #removing the dot format of images like .jpg
print(classNames)
```

```
def findDes(images):
    desList=[]
    for img in images:
        kp,des = orb.detectAndCompute(img,None)
        desList.append(des)
    return desList
```

```
def FindId(img,desList,thrshold=12):
    kp2,des2=orb.detectAndCompute(img,None)
    brf=cv2.BFMatcher()
    matchlistgood=[]
    lastVal = -1
    try:
        for des in desList:
            matches=brf.knnMatch(des, des2, k=2)
            good=[]
            for m,n in matches:
             if m.distance < 0.75*n.distance:
                good.append([m])
            matchlistgood.append(len(good))
    except:
        pass
    #print(matchlistgood)
    if len(matchlistgood)!= 0:
        if max(matchlistgood) > thrshold:
            lastVal= matchlistgood.index(max(matchlistgood))
    return lastVal
```

```
desList=findDes(images)
print(len(desList))
```

```
cam=cv2.VideoCapture(0)
```

```
while True:
    success, img2= cam.read()
    imgoriginal= img2.copy()
    img2=cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)

    id = FindId(img2,desList)   #calling of FindId function 3rd argument is automatically i.e threshold=12
    print(id)
    if id != -1:
        cv2.putText(imgoriginal,classNames[id],(50,50),cv2.FONT_HERSHEY_COMPLEX,1,(0,0,255),2)

    cv2.imshow('img2',imgoriginal)
    cv2.waitKey(1)
```

```
# img3=cv2.drawMatchesKnn(img1,kp1,img2,kp2,good,None,flags=2)
# #imgkp1=cv2.drawKeypoints(img,kp1,None)
# #cv2.imshow('kp1',imgkp1)
# cv2.imshow('image1',img1)
# cv2.imshow('image2',img2)
# cv2.imshow('image3',img3)
# cv2.waitKey(0)
```

ADVANTAGES

1 This system will help in many places like detecting a book’s name in a library.

2 Easy to use.

3 You’ll have better insight into AI.

4 Provides a better overview and comprehensive analysis.

5 Helps you to create your own virtual assistant using somehow AI.

CONCLUSION

After making this application we assure that this application will help its users to manage the library system and much more which can be detected through camera but with some intelligence. It will guide them and aware them about how a simple camera can be transformed into an eye of an AI. It will prove to be helpful for the people who works in the field of camera surveillance , irritated because of amount of expenses and wishes to manage money and to preserve the record of their daily cost which may be useful to change their way of spending money. In short, this application will help its users to overcome the wastage of money.

REFRENCES

- **Introduction to Python available at:**
  https://www.geeksforgeeks.org/python-language-introduction/
- **Introduction to Open CV available at:**
  https://www.geeksforgeeks.org/introduction-to-opencv/
- **Introduction to Machine Learning available at:**
  https://www.geeksforgeeks.org/introduction-machine-learning/