SMART CCTV SURVEILLANCE SYSTEM USING PYTHON

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering Degree in Computer Science and
Engineering

By

LOLAKAPURI ANANTH VARDHAN (Reg. No.:

39110574) KOTA LAKSHMI VINAY REDDY (Reg. No.:

39110528)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND
TECHNOLOGY (DEEMED TO BE UNIVERSITY)
Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI – 600119

APRIL 2023

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

<u>BONAFIDE CERTIFICATE</u>

This is to certify that this Project Report is the Bonafide work of **LOLAKAPURI ANANTH VARDHAN (Reg. No: 39110574)** and **KOTA LAKSHMI VINAY REDDY (Reg. No.: 39110528)** who carried out the project entitled **"SMART CCTV SURVEILLANCE SYSTEM USING PYTHON"** under our supervision from Jan 2023 to April 2023.

Veena.K

Internal Guide

Dr. K. VEENA M.E., Ph.D.

[illegible]

Head of the Department
Dr. L. LAKSHMANAN M.E., Ph.D.

SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF CSE

Submitted for Viva Voce Examination held on <u>20.04.2023</u>

[illegible]

[illegible]

Internal Examiner **External Examiner**

DECLARATION

I **LOLAKAPURI ANANTH VARDHAN (Reg. No: 39110574)** hereby declare that the Project Report entitled **“SMART CCTV SURVEILLANCE SYSTEM USING PYTHON”** done by me under the guidance of **Dr. K. VEENA M.E., Ph.D.** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering.**

Vardhan

DATE: 26-04-2023

PLACE: CHENNAI **SIGNATURE OF THE CANDIDATE**

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph.D., Dean**, School of Computing, **Dr. L. Lakshmanan, M.E., Ph.D.,** Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. K. VEENA M.E., Ph.D.** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

The proliferation of security cameras in public spaces has led to the development of smart CCTV surveillance systems, which can process live video feeds and detect security threats in real-time. These systems utilize advanced image processing techniques and artificial intelligence algorithms to analyze video data, identify suspicious behavior, and alert security personnel when necessary. Python is a popular programming language that is widely used in the development of smart CCTV surveillance systems due to its ease of use and powerful libraries for computer vision and machine learning. By leveraging Python, developers can quickly build complex video analysis pipelines that can perform a range of tasks, including object detection, tracking, and classification. One of the key benefits of using Python in smart CCTV surveillance systems is its ability to work with open-source computer vision libraries such as OpenCV, which provides a wide range of tools for image and video processing. With these libraries, developers can quickly implement advanced algorithms for object detection and tracking, such as YOLO (You Only Look Once) and SSD (Single Shot Detector), which can detect objects in real-time with high accuracy. Another advantage of using Python in smart CCTV surveillance systems is its ability to integrate with deep learning frameworks such as TensorFlow and PyTorch. With these frameworks, developers can train and deploy deep neural networks that can perform a range of tasks, such as facial recognition, crowd detection, and anomaly detection. Overall, smart CCTV surveillance systems using Python are becoming increasingly popular in a range of applications, from public safety and security to traffic monitoring and retail analytics. By leveraging the power of Python and its associated libraries and frameworks, developers can quickly build robust and efficient video analysis pipelines that can identify and respond to security threats in real-time. As such, smart CCTV surveillance systems are becoming an essential tool for enhancing public safety and security in an increasingly complex world.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Surveillance systems have become a crucial element in ensuring public safety and security. With the rise in criminal activities, businesses and governments are constantly looking for ways to improve the effectiveness of their surveillance systems. In recent years, advancements in technology have made it possible to develop intelligent surveillance systems that are capable of analyzing large amounts of data and detecting unusual activity in real-time. This project report focuses on the development of a smart CCTV surveillance system using Python.

The objective of this project is to design a smart CCTV surveillance system that can analyze video footage in real-time and detect unusual activities. The system will use machine learning algorithms to learn from past events and identify patterns that could indicate potential threats. The system will also be able to alert security personnel in real-time, allowing them to respond quickly and efficiently to any potential security breaches.

The system will be designed using Python, which is a high-level programming language that is widely used in the field of machine learning and artificial intelligence. Python provides a number of powerful libraries and frameworks that make it easy to develop intelligent systems. The project will make use of the OpenCV library for image and video processing, TensorFlow for machine learning, and Flask for web development.

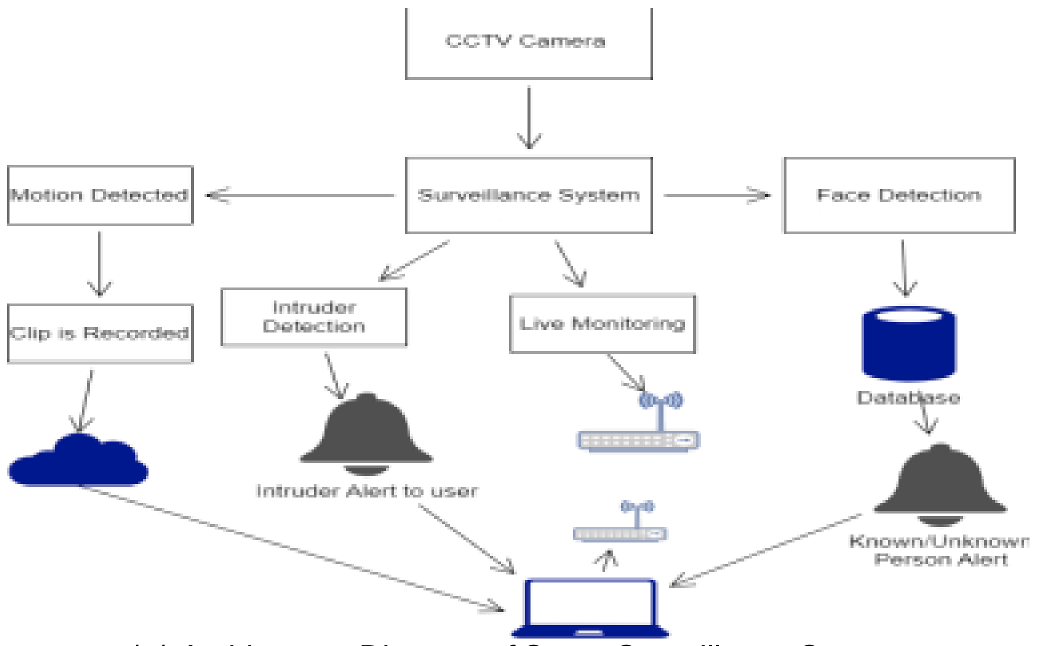
The system will be designed to be scalable and modular, allowing it to be easily adapted to different environments and configurations. The system will consist of a number of different components, including cameras, servers, and client applications. The cameras will capture video footage, which will be processed by the servers. The servers will use machine learning algorithms to analyze the video footage and detect unusual activity. The client applications will provide a user interface for security personnel to monitor the system and respond to any potential threats.

The system will be designed with privacy in mind, and will comply with all relevant privacy regulations. The system will only store video footage for a limited period of time, and access to the system will be restricted to authorized personnel.

The smart CCTV surveillance system will provide a number of benefits over traditional surveillance systems. Firstly, the system will be able to detect unusual activity in real-time, allowing security personnel to respond quickly and efficiently to potential threats. Secondly, the system will be able to learn from past events and identify patterns that could indicate potential threats. Finally, the system will be scalable and modular, allowing it to be easily adapted to different environments and configurations.

In conclusion, the smart CCTV surveillance system using Python is an innovative approach to improving public safety and security. The system will provide real time monitoring and detection of potential threats, while also being scalable and modular. The project report will provide a detailed overview of the system architecture, design, implementation, and evaluation. The report will also discuss the challenges faced during the development of the system and propose future directions for research and development in this area.

Below are the different features of a smart cctv surveillance
system: 1. Object Detection
2. Face Recognition
3. Intruder Alert
4. Person Counter
5. Wireless Transmission
6. Night Vision
7. Cloud Storage



1.1 Architecture Diagram of Smart Surveillance System

1.1 FACE RECOGNITION

Face recognition technology has come a long way in recent years and has become an integral part of many modern security systems. One application of this technology is in CCTV surveillance systems, where it can be used to detect and identify people in real-time. In this project report, we will explore the use of Python for implementing a smart CCTV surveillance system that uses face recognition.

Face recognition technology uses algorithms to identify and verify the identity of individuals based on their facial features. The algorithms analyze images or videos of faces and extract unique features such as the distance between the eyes, the shape of the nose, and the contour of the jawline. These features are then compared to a database of known faces to determine the identity of the person in question.

The first step in implementing a smart CCTV surveillance system using face recognition is to acquire high-quality video footage of the area being monitored. This can be done using high-resolution cameras placed strategically in the area. The footage can then be processed using Python and OpenCV, a popular computer vision library.

OpenCV provides a set of tools for face detection, which can be used to identify and locate faces in the video footage. The library uses a technique called Haar cascades to detect faces. Haar cascades are classifiers that use machine learning algorithms to identify patterns in the image that correspond to faces.

Once the faces have been detected, the next step is to extract the facial features using a technique called face landmark detection. This involves identifying key points on the face such as the corners of the eyes, the tip of the nose, and the edges of the lips. The facial landmarks can then be used to calculate the unique features of the face.

To identify the faces in the footage, the facial features need to be compared to a database of known faces. This can be done using machine learning algorithms such as Principal Component Analysis (PCA) or Linear Discriminant Analysis (LDA). These algorithms analyze the features of the faces in the database and create a mathematical model that can be used to identify new faces.

Once a new face has been identified, the system can perform a range of actions depending on the application. For example, in a security system, the system may send an alert to security personnel or automatically lock doors to prevent unauthorized access.

In addition to security applications, face recognition technology has many other potential applications. For example, it can be used in retail to track customer behavior and provide personalized recommendations based on their shopping habits. It can also be used in healthcare to monitor patients and detect signs of illness or distress.

In conclusion, face recognition technology is a powerful tool that can be used in a wide range of applications. In this project report, we have explored the use of Python and OpenCV for implementing a smart CCTV surveillance system using face recognition. With the rapid advancement of technology, we can expect to see many more innovative applications of this technology in the future.

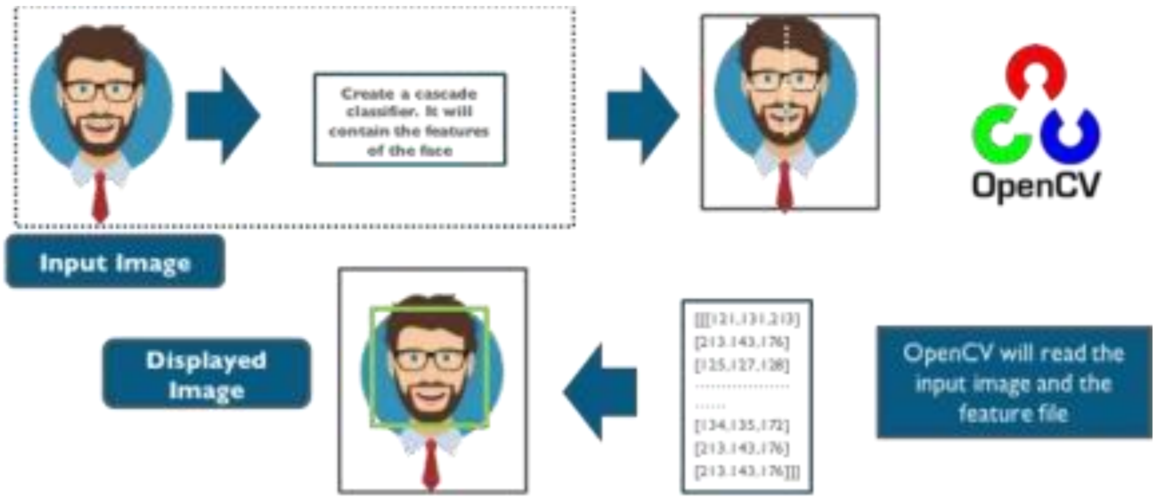
It is used to find if the person the frame is known or not. It can be done in two steps:

• Find the faces in the frames

• Use LBPH face recognizer algorithm to predict the person from already trained model.

Detecting faces in the frames

This is done via Haarcascade classifiers which are again in-built in OpenCV module of python.


Create a cascade classifier. It will contain the features of the face
OpenCV
Input Image
Displayed Image
OpenCV will read the input image and the feature file

1.2 Figure showing the working of Face Detection

Cascade classifier, or namely cascade of boosted classifiers working with haar like features, is a special case of ensemble learning, called boosting. It typically relies on Adaboost classifiers (and other models such as Real Adaboost, Gentle Adaboost or Logitboost).

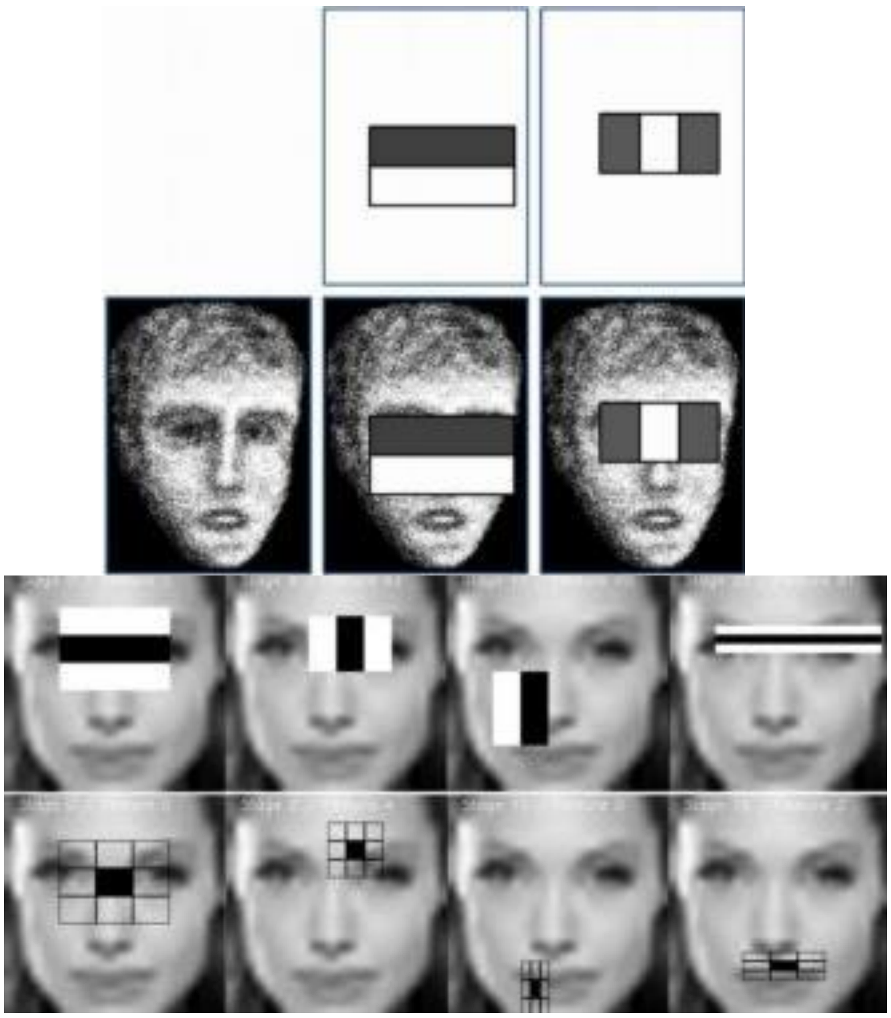
Cascade classifiers are trained on a few hundred sample images of image that contain the object we want to detect, and other images that do not contain those images.

There are some common features that we find on most common human

faces: • a dark eye region compared to upper-cheeks

• a bright nose bridge region compared to the eyes

• some specific location of eyes, mouth, nose…


1.3 Figure Explaining about cascade classifier

The characteristics are called Haar Features. The feature extraction process will look like this:

Haar features are similar to these convolution kernels which are used to detect the presence of that feature in the given image.

For doing all this stuff OpenCV module in python language has inbuild function called cascade classifier which we have used in order to detect for faces in the frame.

Using LBPH for face recognition

So now we have detected for faces in the frame and this is the time to identify it and check if it is in the dataset which we've used to train our lbph model.

The LBPH uses 4 parameters:

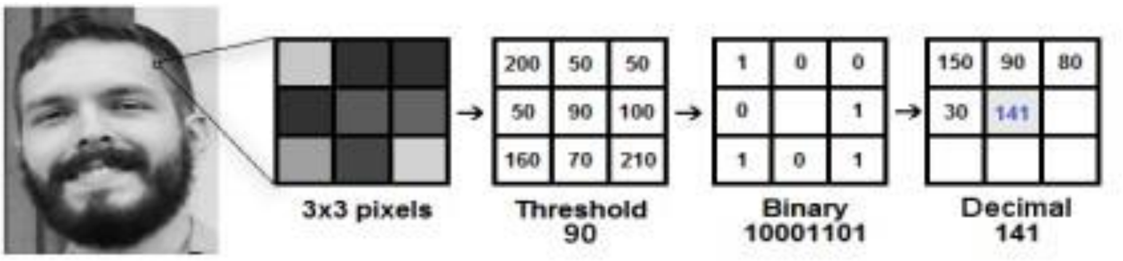
- Radius: the radius is used to build the circular local binary pattern and represents the radius around the central pixel. It is usually set to 1. •

Neighbors: the number of sample points to build the circular local binary pattern. Keep in mind: the more sample points you include, the higher the computational cost. It is usually set to 8.

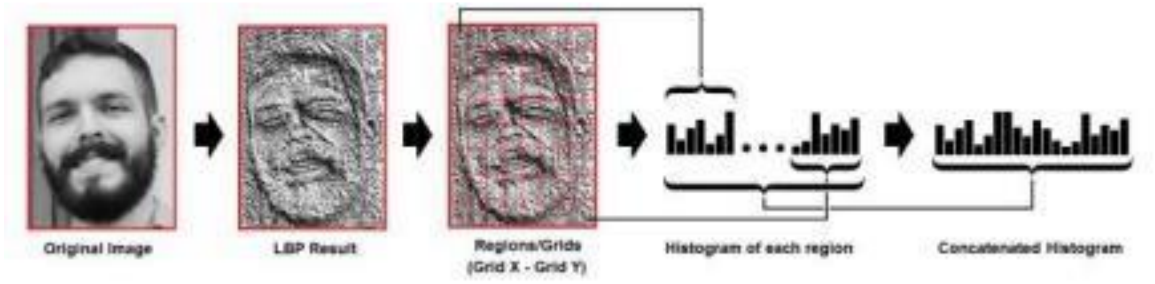
• Grid X: the number of cells in the horizontal direction. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector. It is usually set to 8.

• Grid Y: the number of cells in the vertical direction. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector. It is usually set to 8

The first computational step of the LBPH is to create an intermediate image that describes the original image in a better way, by highlighting the facial characteristics. To do so, the algorithm uses a concept of a sliding window, based on the parameters radius and neighbors. Which is shown perfectly via the above image.



1.4 Figure Showing how LBPH Face Recgonition works



1.5 Figure showing the extraction of histograms

Extracting the Histograms: Now, using the image generated in the last step, we can use the Grid X and Grid Y parameters to divide the image into multiple grids, as can be seen in the following image:

And after all this the model is trained and later onwhen we want to make predictions the same steps are applied to the make and its histograms are compared with already trained model and in such way this feature works.

1.2 OBJECT DETECTION

Object detection is a fundamental computer vision task that aims to detect objects of interest within an image or video. It involves identifying and localizing objects within an image, and it is an essential component of many computer vision applications, including smart CCTV surveillance systems.

A smart CCTV surveillance system is a system that incorporates artificial intelligence (AI) algorithms to improve its functionality, including the ability to automatically detect objects and events of interest. Python is a popular programming language for developing such systems due to its ease of use, large number of libraries and frameworks available, and its versatility for handling data and implementing machine learning algorithms.

Object detection in images and videos typically involves three main stages: image preprocessing, object detection, and post-processing. The image preprocessing stage involves preparing the input image or video frames for object detection, which may include resizing, normalizing, and color-space conversion. Object detection is the stage where objects are detected within the image or video frames. Post-processing involves refining the detected objects and removing any false positives.

Object detection can be approached using different methods, including traditional computer vision methods and deep learning-based methods. Traditional methods, such as Haar cascades, use handcrafted features and machine learning algorithms to detect objects. However, these methods can be limited in their accuracy and robustness, particularly for complex object detection tasks. Deep learning-based methods, on the other hand, use convolutional neural

networks (CNNs) to learn features from the input data and perform object detection. These methods have shown superior performance compared to traditional methods and are currently the state-of-the-art in object detection.

One popular deep learning-based object detection method is the region-based CNN (R-CNN) family of algorithms. R-CNN algorithms involve dividing the input image into regions and processing each region separately to generate object proposals. The object proposals are then passed through a CNN to extract features, which are used to classify and refine the object proposals. This approach has been improved over the years with faster R-CNN, Mask R-CNN, and other variations that have achieved high accuracy on benchmark datasets such as COCO and PASCAL VOC.

Another popular deep learning-based method for object detection is YOLO (You Only Look Once). YOLO is a real-time object detection algorithm that can detect objects in an image in a single pass. It divides the input image into a grid and performs object detection on each grid cell to generate object proposals. YOLO has been improved over the years with YOLOv2, YOLOv3, and other variants that have achieved high accuracy and real-time performance.

Python provides many libraries and frameworks for implementing object detection algorithms, including OpenCV, TensorFlow, Keras, PyTorch, and others. These libraries and frameworks provide pre-trained models for object detection and can also be used to train custom models on specific datasets. They also provide tools for image and video preprocessing, post-processing, and visualization of object detection results.

In a smart CCTV surveillance system, object detection can be used to automatically detect and track objects of interest within the surveillance footage. This can include detecting people, vehicles, and other objects, as well as identifying events such as abnormal behavior or incidents. Object detection can also be used to trigger alarms or notifications, and to provide real-time insights into the surveillance footage.

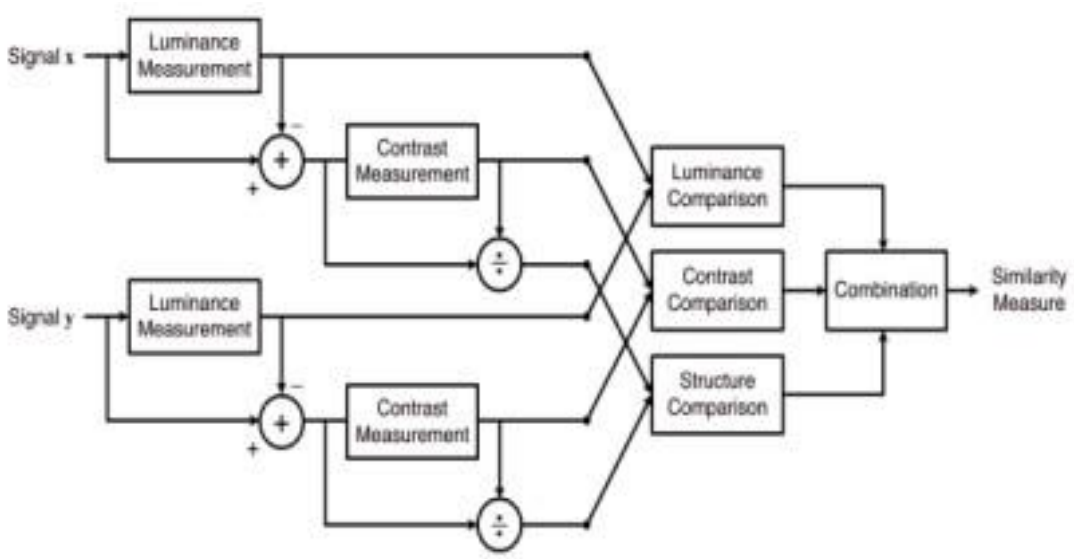
Overall, object detection is a critical task for many computer vision applications, including smart CCTV surveillance systems. With the availability of deep learning based methods and the ease of use of Python libraries and frameworks, it is now easier than ever to implement object detection in a wide range of applications.

This uses Structural Similarity to find the differences in the two frames. The two frames are captured first when noise was not happened and second when noise stopped happening in the frame.
SSIM is used as a metric to measure the similarity between two given images. As this technique has been around since 2004, a lot of material exists explaining the theory behind SSIM but very few resources go deep into the details, that too specifically for a gradient-based implementation as SSIM is often used as a loss

function.

The Structural Similarity Index (SSIM) metric extracts 3 key features from an image:

- Luminance
- Contrast
- Structure



*1.6 Flowchart showing how the monitoring works in CCTV*This system calculates the Structural Similarity Index between 2 given images which is a value between -1 and +1. A value of +1 indicates that the 2 given images are very similar or the same while a value of -1 indicates the 2 given images are very different. Often these values are adjusted to be in the range [0,

1], where the extremes hold the same meaning.

Luminance: Luminance is measured by averaging over all the pixel values. Its denoted by μ (Mu) and the formula is given below,

Contrast: Contrast is defined as the difference between the highest and lowest intensity value of the frame. So, you can easily calculate it from respective histogram

Structure: The structural comparison is done by using a consolidated formula (more on that later) but in essence, we divide the input signal with its standard deviation so that the result has unit standard deviation which allows for a more robust comparison.

Luckily, thanks to skimage package in python we don't have to replicate all this mathematical calculation in python since skimage has pre build feature that d o all of these tasks for us with just calling its in-built function.

We just have to feed in two images/frames which we have captured earlier, so we just feed them in and its gives us out the masked image with score.

1.3 INTRUDER ALERT

CCTV (Closed Circuit Television) surveillance is an important aspect of security systems in public places, commercial buildings, residential areas, and other places. With advancements in technology, CCTV surveillance systems have become smarter, more intelligent, and efficient. One such advancement is the implementation of Python programming language to create a smart CCTV surveillance system that provides an intruder alert.

The smart CCTV surveillance system is a computer-based system that uses machine learning algorithms and computer vision techniques to analyze the video feed from the CCTV cameras. The system is designed to detect any unusual activity or behavior, such as a person entering a restricted area, wandering around suspiciously, or leaving an object unattended.

The Intruder Alert system is a critical component of the smart CCTV surveillance system that provides real-time alerts to the security personnel in case of any suspicious activity. This report discusses the Intruder Alert system, its features, and the implementation of the system using Python programming language.

Features of the Intruder Alert System:

1. Real-Time Detection: The Intruder Alert system detects any suspicious activity or behavior in real-time. The system uses machine learning algorithms and computer vision techniques to analyze the video feed from the CCTV cameras.

2. Automatic Alerts: The system sends automatic alerts to the security personnel in case of any suspicious activity. The alerts can be sent via email, SMS, or other notification methods.

3. Multiple Camera Support: The Intruder Alert system can support multiple CCTV cameras simultaneously. The system can analyze the video feed from multiple cameras and provide alerts based on the detected activity.

4. Customizable Settings: The Intruder Alert system provides customizable settings that can be configured based on the specific requirements of the user. The user can set the sensitivity level, threshold, and other parameters to optimize the system performance.

5. Easy Integration: The Intruder Alert system can be easily integrated with other security systems such as access control, fire alarms, and other surveillance systems.

Implementation of the Intruder Alert System using Python:

The Intruder Alert system can be implemented using Python programming language and various libraries such as OpenCV, TensorFlow, and Keras. OpenCV is a computer vision library that provides various algorithms for image and video processing. TensorFlow and Keras are machine learning libraries that provide tools for creating and training machine learning models.

The following steps can be followed to implement the Intruder Alert system using Python:

1. Video Capture: The video feed from the CCTV camera can be captured using OpenCV. The video feed can be stored in a video file or processed in real-time.

2. Motion Detection: The system can detect motion in the video feed using OpenCV. The motion detection algorithm can be used to identify any movement in the video feed.

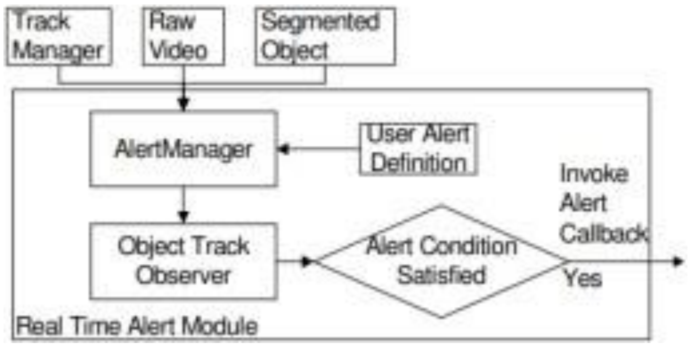
3. Object Detection: The system can detect objects in the video feed using TensorFlow or Keras. The object detection algorithm can be used to identify any person, vehicle, or other objects in the video feed.

4. Alert Generation: The system can generate an alert in case of any suspicious activity. The alert can be sent via email, SMS, or other notification methods.

5. Customizable Settings: The system settings can be customized based on the specific requirements of the user. The user can set the sensitivity level, threshold, and other parameters to optimize the system performance.

6. Integration with other systems: The system can be easily integrated with other security systems such as access control, fire alarms, and other surveillance systems.

The real time alert module uses the information produced by the other modules, namely, object detection, tracking and classification to detect user specified alerts. The key feature of the real time alert module is its extensible design. Figure 1.7 shows the generic structure of the real time alert module. This structure is instantiated by most of the alert types. In order to illustrate the structure of the module we present the design of the directional motion alert as an example. The following processes are instantiated when the user specifies a directional motion alert.



1.7 Internal structure of Real time Alert Module

1. Directional Motion Alert Manager: Each motion alert user definition instantiates a directional motion alert manager, which is responsible for ensuring correct monitoring of the scene. The alert manager ensures that for every object being tracked there is a corresponding Object Track Observer that is instantiated. And that the Object Track Observer is deleted immediately upon the exit of the object from the scene.

2. Directional Motion Object Track Observer: This is the process that is charged with the job of measuring the direction of motion of the object and comparing it to the user specified direction. Whenever the object motion direction matches the user specified direction, the object track observer issues a real time alert. The application uses the alert to signal the user that one of the specified alert conditions has been met. Alert Manager Object Track Observer Track Manager Raw Video Segmented Object Alert Condition Satisfied Yes Invoke Alert Callback User Alert Definition Real Time Alert Module Figure 8: Internal structure of the real time alert module. The exact nature of the object track observer depends on the particular alert it is implementing. However, the general structure of many types of alerts is similar to that described above.

1.4 VISITORS COUNTING

This is the feature which can detect if someone has entered in the room or gone out.

So, it works using following steps:

• It first detects for noises in the frame.

• Then if any motion happen it find from which side does that happen either left or right.

• Last if checks if motion from left ended to right then its will detect it as entered and capture the frame.

So, there is not complex mathematics going on around in this specific feature. So basically, to know from which side does the motion happened we first detect for motion and later on we draw rectangle over noise and last step is we check the co-ordinates if those points lie on left side, then it is classified as left motion.

CHAPTER 2

LITERATURE SURVEY

Sagar Pandey, Sarah Irshad and Sanjay Kumar Singh, “Smart CCTV System”, Research Gate, 2021. At this time, one of the most crucial aspects of human life is security. Homes are frequently left unattended due to complex activities. Most individuals use CCTV (Closed Circuit Television) cameras to protect their homes when they are away from home. In smart cities, the footage captured by surveillance cameras is critical for crime prevention and investigation. Because the camera only records without analyzing objects, traditional Video surveillance is less effective. The sensor camera's purpose is to reflect progress in the motion of entities that are observable to the camera, in this situation, physical movements. This camera's reliable monitoring mechanism can detect approaching items. The adaptive background removal technology proposed in this paper can accommodate to frame changes. The prior background intensity inference will always be used to update the background frame. It will then analyze the method's effectiveness. Along with motion detection, it will only record moving frames, allowing the system to make the most of its capacity.

Dr. S. Hussain, “Smart Surveillance System using Thing Speak and Raspberry Pi”, Research Gate, 2015. This paper proposes the Smart Surveillance System using Thing speak and Raspberry pi. This design is a small portable monitoring system for home and office security. The model uses hardware mechanism such as Raspberry pi (model B), Gyro sensor and Raspberry pi camera. This system will monitor when motion detected, the Raspberry Pi will control the Raspberry Pi camera to take a picture and sent out an alert email with the image to the user by using Wi-Fi adaptor according to the program written in python environment. And at the same time the sensor real time data is visualized in the form of charts in Thing speak. The proposed system will work in standalone mode without the requirement of PC once programmed.

Amol V. Nagime and Patange. A. D , “Smart CCTV Camera Surveillance System”, International Journal of Science and Research (IJSR), 2016. The proposed work is Smart CCTV camera surveil-lance system. CCTV camera can be wirelessly monitored and control with the help of the RF module . In the monitoring site, the system captures the video through the embedded multitask operating system. The digital video has been compressed by the MJPEG algorithm. By the TV the users can view the monitors video directly, by the common Gateway interface, the users who are authorized can also control the camera and observe the motion detection.

Hampapur, Arun & Brown, L. & Connell, Jonathan & Pankanti, S. & Senior, Andrew & Tian, Y., “Smart surveillance: Applications, technologies and implications”, 2004. Smart surveillance, is the use of automatic video analysis technologies in video surveillance applications. This paper attempts to answer a number of questions about smart surveillance: What are the applications of smart surveillance? What are the system architectures for smart surveillance? What are the key technologies? What are the some of the key technical challenges? and What are the implications of smart surveillance, both to security and privacy?

K. Patel and M. Patel, "Smart Surveillance System using Deep Learning and Raspberry Pi” ,2021 8th International Conference on Smart Computing and Communications (ICSCC), 2021. Today, in the technological era of the 21st century, CCTV cameras have been proven to be very fruitful in our daily lives. From monitoring the baby in the bassinet to prevent some crimes, CCTV camera has become of vital importance. We as humans, always try to make things perfect around us. Using this article, we also have attempted to present our perspective to make these CCTV cameras more perfect. We have made an effort to enhance regular CCTV cameras using the vast field of deep learning and IoT. We have attempted to accomplish our goal by providing a protoStype for the smart surveillance system. We have tried to upgrade the regular CCTV cameras with some customized deep learning models developed by us. In this modified version, we have given the CCTV cameras the ability to detect fire and weapons. Also, we have tried to fulfil an ad-hoc requirement of Face Mask Detection considering the current situation of COVID19. For fulfilling our objective, we have provided an outline combining IoT (Raspberry Pi) to deep learning using AWS EC2 Cloud Architecture. To make the surveillance system user-friendly, we have

also taken account of the client-side interface. Considering all the above applications, we have successfully provided an archetype in this paper.

S. U. Ahmed, H. Khalid, M. Affan, T. A. Khan and M. Ahmad, "Smart Surveillance and Tracking System," 2020 IEEE 23rd International Multitopic Conference (INMIC), 2020. This paper presents an intelligent system that can identify and recognize faces with applications, including a person's tracking, home surveillance, and private security. An automatic face recognition intrinsically handles security issues with flexibility - in the case of an unrecognized or unknown person, the real-time video stream is processed, motion is detected, and dual-axis pan-tilt servos track that person with a camera. Furthermore, such peculiar activities are video recorded with synchronization from cloud storage, and mobile alerts are generated. In the absence of the internet, a database file is developed with an audio notification to the security room if the anonymous face detected is not present in the database. Speech recognition and relay are also added for voice transmission and activation of surrounding lights. This project intends to substitute costly security systems using Raspberry pi 3B+ as the microcomputer.

2.1 INFERENCES FROM THE LITERATURE SURVEY

The literature survey provides valuable insights into the current state of research and development in the field of smart CCTV surveillance systems. The surveyed papers suggest that smart surveillance systems have a wide range of applications, including home and office security, crime prevention, and investigation. Furthermore, it highlights that traditional video surveillance is less effective, as the camera only records without analyzing objects. However, smart surveillance systems use automatic video analysis technologies, such as motion detection and face recognition, to analyze video footage in real-time.

The proposed smart surveillance systems vary in terms of their hardware and software components, as well as their functionalities. For instance, the Smart Surveillance System using Thing Speak and Raspberry Pi proposed by Dr. S. Hussain uses a Raspberry Pi model B, Gyro sensor, and Raspberry Pi camera to monitor motion and send an alert email to the user with the captured image. The sensor's real-time data is also visualized in the form of charts in Thing Speak. On the other hand, the Smart CCTV Camera Surveillance System proposed by Amol V. Nagime and Patange A. D uses the RF module to wirelessly monitor and control the CCTV camera. The system captures digital video through the embedded multitask operating system and compresses it using the MJPEG algorithm. The users can view the monitored video through a TV, and authorized users can control the camera and observe motion detection through the common gateway interface.

Furthermore, the surveyed papers suggest that deep learning and IoT can enhance traditional CCTV cameras' functionality. For example, the Smart Surveillance System using Deep Learning and Raspberry Pi proposed by K. Patel and M. Patel enhances regular CCTV cameras using customized deep learning models. The modified version has the ability to detect fire, weapons, and face masks, making it suitable for use in the current COVID-19 situation. The system also uses IoT (Raspberry Pi) and AWS EC2 Cloud Architecture to make it user friendly.

Finally, the surveyed papers discuss the technical challenges of smart surveillance, including system architecture, key technologies, and security and privacy implications. For instance, Hampapur et al. (2004) discuss the technical challenges of smart surveillance, such as scalability, robustness, and integration with other systems. The paper also highlights the potential security and privacy implications of smart surveillance, such as unauthorized access to personal data and violation of privacy rights.

Overall, the literature survey suggests that smart CCTV surveillance systems have numerous applications and benefits, including improved security, crime prevention, and investigation. The proposed smart surveillance systems vary in terms of their hardware and software components and functionalities. However, they all use automatic video analysis technologies, such as motion detection and face recognition, to analyze video footage in real-time. The technical challenges of smart surveillance, including system architecture, key technologies, and security and privacy implications, need to be addressed to develop effective and secure smart surveillance systems.

2.2 OPEN PROBLEMS IN EXIXTING SYSTEM

The field of smart CCTV surveillance systems is rapidly evolving and presents several open problems that researchers and developers are currently working on addressing. In this section, we will discuss some of the open problems in smart CCTV surveillance systems.

- Privacy Concerns: One of the major concerns with the implementation of smart CCTV surveillance systems is the violation of privacy. Smart surveillance systems use advanced technologies such as facial recognition, object detection, and tracking to identify and track individuals, which raises privacy concerns. There are concerns that these systems can be used for illegal surveillance or unauthorized monitoring, and there is a need to ensure that proper privacy regulations and laws are in place to protect individuals' rights.
- Scalability: Another significant issue with smart CCTV surveillance systems is scalability. The installation of CCTV cameras in public areas can be expensive, and it can be challenging to deploy these systems on a large scale. There is a need to develop cost-effective solutions that can be easily deployed in public areas without incurring high costs.
- Real-time Analysis: Traditional CCTV surveillance systems record video footage, which can be analyzed later. However, in smart surveillance systems, there is a need for real-time analysis of the captured video data to detect and respond to potential threats quickly. Real-time analysis requires advanced algorithms and processing power, which can be challenging to implement on a large scale.
- Integration with IoT: The integration of smart CCTV surveillance systems with the Internet of Things (IoT) is another area of concern. Smart surveillance systems generate large amounts of data, and it is crucial to integrate these systems with IoT devices to ensure that the data can be efficiently processed and analyzed. Integration with IoT devices can also help improve the accuracy of the analysis and detection of potential threats.
- Robustness: Another open problem in smart CCTV surveillance systems is the robustness of the system. The system must be robust enough to handle

challenging environmental conditions such as low light, changing weather conditions, and occlusions. The system must also be able to handle false alarms and prevent unnecessary alerts.

- Power Consumption: Smart CCTV surveillance systems require a constant power supply, which can be challenging in remote areas. There is a need to develop energy-efficient solutions that can operate using renewable energy sources such as solar power.

- User-Friendliness: Finally, the user-friendliness of smart CCTV surveillance systems is also an open problem. The system must be easy to use and operate, especially for non-technical users. The user interface must be intuitive, and the system must provide clear instructions and feedback to the user.

In conclusion, smart CCTV surveillance systems present several open problems that researchers and developers are currently working on addressing. These problems include privacy concerns, scalability, real-time analysis, integration with IoT, robustness, power consumption, and user-friendliness. Addressing these issues will require innovative solutions and collaborations between researchers, developers, and policymakers to develop effective and efficient smart surveillance systems that can improve public safety while ensuring privacy and security.

CHAPTER 3

METHODOLOGY

This chapter of the report presents a detailed description of the steps taken to implement the smart CCTV surveillance system using Python. This chapter outlines the research methods, techniques, and tools utilized in the design, development, and evaluation of the system. The methodology used in this project is based on a structured approach that emphasizes the importance of clarity, accuracy, and efficiency in achieving the project goals. The chapter also discusses the research objectives, the research questions, and the hypotheses that were tested during the project. Additionally, it presents the procedures used to collect, process, and analyze data as well as the ethical considerations involved in the project. Finally, the chapter concludes with a summary of the methodology used and the contributions of the project.

3.1 CONFIGURATION ON THE SERVER MACHINE

There are 3 files on the server machine. A Caffe proto.txt file which is a deep learning framework, a pre-trained model and a python file which contains the program to perform all the required tasks.

The complete program can be broken down into 5 parts:

Part 1: Check for new file on Dropbox and if new file is found, download it using the API key of the same app where the file was uploaded from Raspberry Pi. Part 2: Unzip the file that is downloaded from the dropbox.

Part 3: Apply image processing on the images using deep learning-based object detection with OpenCV.

Part 4: If human is detected in the image processing, send a notification to the user and save a copy of the image.

Part 5: Delete the other files that were downloaded from Dropbox.

3.2 DEVELOPMENT ENVIRONMENT SOFTWARE

Operating system: Windows 11

Windows 11 is selected as the developing operating system because Windows has the biggest selection of software available for its platform than any other operating system. The benefit of this is that users get to choose from wider variety of options. This creates healthy “competition” for users, where software developers really have to push boundaries to produce the best program possible. Anything less than the best will result in user’s picking the next program on the list. This alone does wonders in motivating software developers to deliver excellent solutions that meet users’ needs.

Software used: PYTHON

Python is a high-level, interpreted, and general-purpose dynamic programming language that focuses on code readability. It has fewer steps when compared to Java and C. It was founded in 1991 by developer Guido Van Rossum. Python ranks among the most popular and fastest-growing languages in the world. Python is a powerful, flexible, and easy-to-use language. In addition, the community is very active there. It is used in many organizations as it supports multiple programming paradigms. It also performs automatic memory management.

3.3 OPERATION ENVIRONMENT

The table shown below is the minimum requirement:

Table 3.1: Table for operation environment

Processor	**Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC**
Operating System	Microsoft Windows 7 to 10 Mac OS X 10.11 0r higher, 64-bit Linux: RHEL 6/7, 64-bit
Memory	4GB RAM
Screen Resolution	Monitor with screen resolution minimum 1024 x 768
Hard disk Space	Minimum 5GB free disk space

3.4 SOFTWARE IMPLEMENTATION

Main.py

The methodology for the Smart CCTV Surveillance System using Python project can be broken down into the following steps:

• Design the user interface: The first step is to design a user interface that is intuitive and easy to use. For this project, the Tkinter library in Python is used to design the user interface. The user interface is designed with buttons for different functionalities such as monitor, record, identify, and in-out, as well as icons for each button.

• Implement the motion detection: The second step is to implement motion detection in the camera feed. This is done using the OpenCV library in Python.

The find_motion function is used to detect motion in the camera feed, and it is called when the "Monitor" button is clicked.

- Implement noise reduction: To reduce noise in the camera feed, the rect_noise function is implemented. This function uses the OpenCV library to create a rectangular region of interest (ROI) and applies a noise reduction filter to that area. It is called when the "Rectangle" button is clicked.

- Implement noise reduction: To reduce noise in the camera feed, the noise function is implemented. This function applies a noise reduction filter to the entire camera feed. It is called when the "Noise" button is clicked.

- Implement recording functionality: The next step is to implement the recording functionality of the system. The record function is implemented using the OpenCV library. When the "Record" button is clicked, this function records the camera feed to a video file.

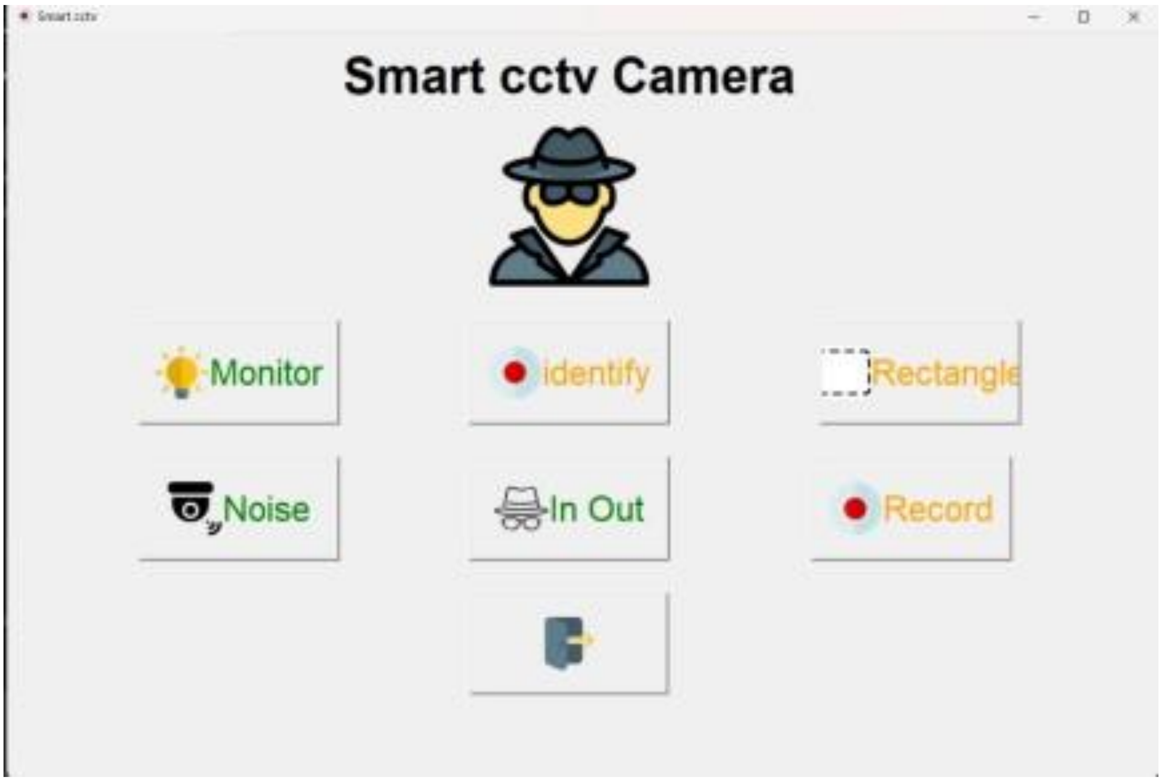
- Implement in-out functionality: To detect the movement of objects in the camera feed, the in_out function is implemented. This function uses the OpenCV library to detect the direction of movement of objects in the camera feed. It is called when the "In Out" button is clicked.

- Implement facial recognition: To identify individuals in the camera feed, the maincall function is implemented. This function uses the OpenCV library and a

trained machine learning model to recognize faces in the camera feed. It is called when the "Identify" button is clicked.

- Testing: Once all the above functionalities are implemented, it is time to test the system thoroughly to ensure that all the features are working correctly. This includes testing for accuracy, speed, and functionality.

In conclusion, the methodology for the Smart CCTV Surveillance System using Python involves designing the user interface, implementing motion detection, noise reduction, recording, in-out functionality, and facial recognition, and finally, thoroughly testing the system.



3.1 UI of Smart CCTV Camera application

Motion.py

The given code is an implementation of motion detection using OpenCV library in Python. Here's the methodology for the code:

• Importing Libraries: In the first line of code, the OpenCV library is imported.

• Defining Function: A function named "noise" is defined that will perform the task of motion detection.

• Accessing Camera: The function accesses the camera by creating an object of the VideoCapture class from OpenCV. The number 0 passed as an argument to the constructor indicates that the default camera should be used.

• Reading Frames: The function reads two consecutive frames from the camera using the read() method of the VideoCapture class.

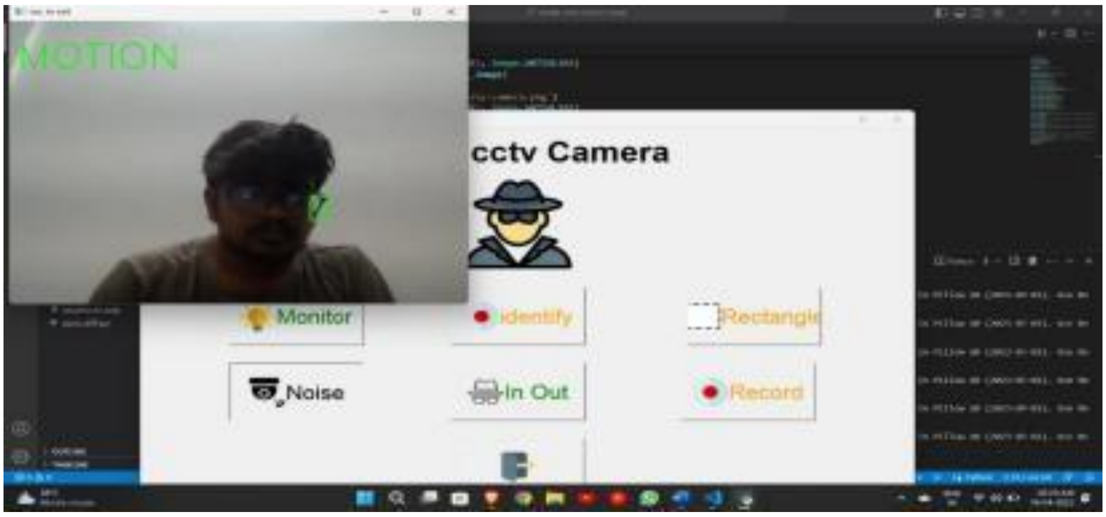
• Calculating Difference: The absolute difference between the two frames is calculated using the absdiff() function from OpenCV. The result is stored in a grayscale image using the cvtColor() function.

• Filtering Difference: The difference image is filtered using a blur() function to remove small high-frequency components that can be caused by noise.

• Thresholding Image: A binary thresholding is applied to the filtered image using the threshold() function to obtain a binary image where pixels with intensity above the threshold value are set to white, and below it are set to black.

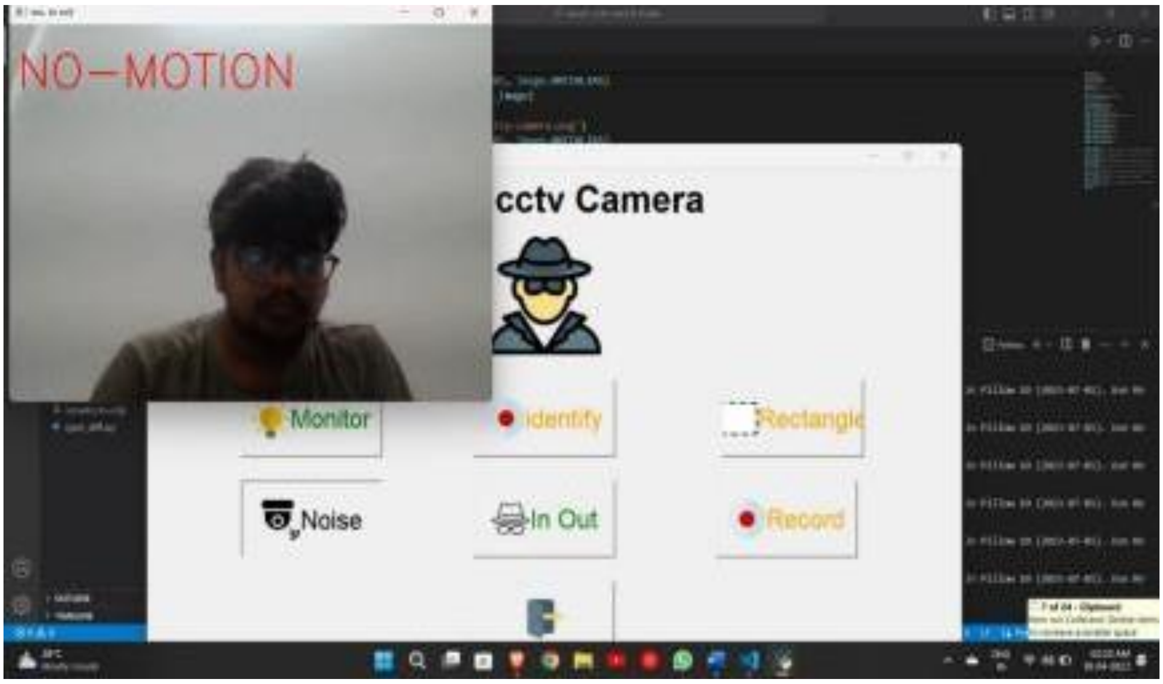
• Finding Contours: The findContours() function is used to detect the contours present in the thresholded image.

• Drawing Rectangle: If at least one contour is detected, the function finds the contour with maximum area and draws a bounding rectangle around it using the rectangle() function from OpenCV. The function also displays the text "MOTION" on the screen using the putText() function.



3.2 Motion Detection Module when there is motion

• Displaying Output: If no contours are detected, the function displays the text "NO-MOTION" on the screen using the putText() function. Finally, the output frame is displayed on the screen using the imshow() function from OpenCV.



3.3 Motion Detection Module when there is no motion

• Releasing Resources: The function waits for a key press, and if the key pressed is the escape key (key code 27), the camera is released, and all windows are closed using the release() and destroyAllWindows() methods of OpenCV.

In summary, the function captures frames from the camera, detects motion in the frames, and displays the output on the screen. It uses various functions from the OpenCV library to perform these tasks.

Record.py

The above code is a Python script that uses the OpenCV library to record video from the default camera of the computer. It saves the video in the "recordings" folder with the current date and time as the filename. It also displays the video on a window with the current date and time written on it. The recording stops when the "Esc" key is pressed.

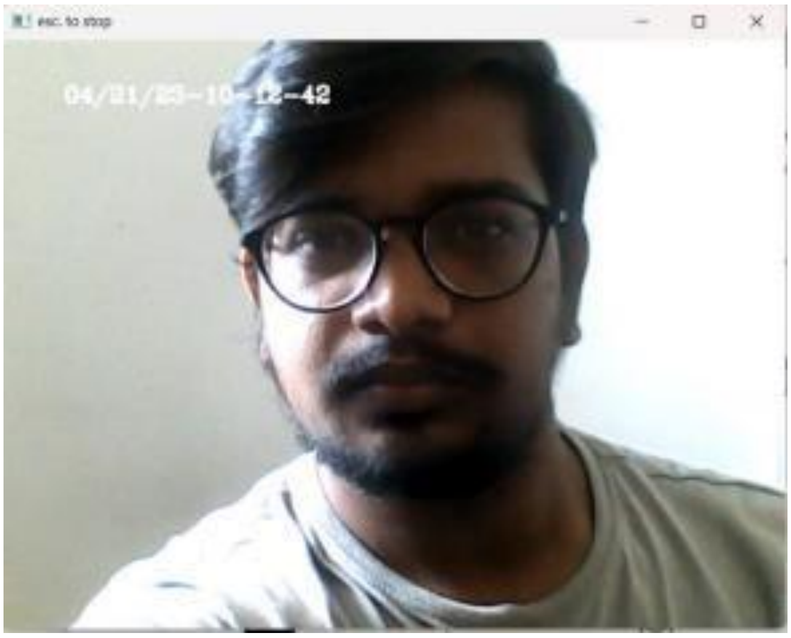
Here is a step-by-step methodology of the code:

• Import the required libraries:

1. OpenCV library for video capture and recording
2. datetime library to generate timestamps for the video filenames and display current date and time on the video window.

• Create a VideoCapture object:

1. Initialize the video capture object with the default camera index (0).



3.4 Working of Recording Module

• Create a VideoWriter object:

1. Initialize the video writer object with the output file name, codec, frame rate, and frame size.
2. The codec used here is XVID, and the frame rate is set to 20 fps. The frame size is set to 640x480.

• Start a loop:

1. Read a frame from the video capture object.
2. Add the current date and time to the frame using the putText() function.
3. Write the frame to the output video using the VideoWriter object.

4. Display the frame in a window using the imshow() function.
5. Wait for the user to press the "Esc" key.
6. Release the video capture and video writer objects and destroy the window.

• Exit the program.

Find_motion.py

The "find_motion" function detects motion in a video stream using OpenCV's VideoCapture class. It compares the frames from the video stream to detect any differences and considers the differences to be motion if there are more than 5 contours detected. If motion is detected, it starts a timer and checks if motion is still present for 4 seconds. If motion is still present after 4 seconds, it captures two frames and passes them to the "spot_diff" function for further processing. If no motion is detected for 4 seconds, it continues to detect motion.

Here is the step-by-step methodology for the "find_motion"

function: • Set motion_detected and is_start_done variables to

False.

• Create a VideoCapture object to capture video frames from the camera. •

Wait for 2 seconds to let the camera stabilize.

• Read the first frame from the camera and convert it to grayscale. •

Start an infinite loop to continuously read frames from the camera. •

Convert the current frame to grayscale.

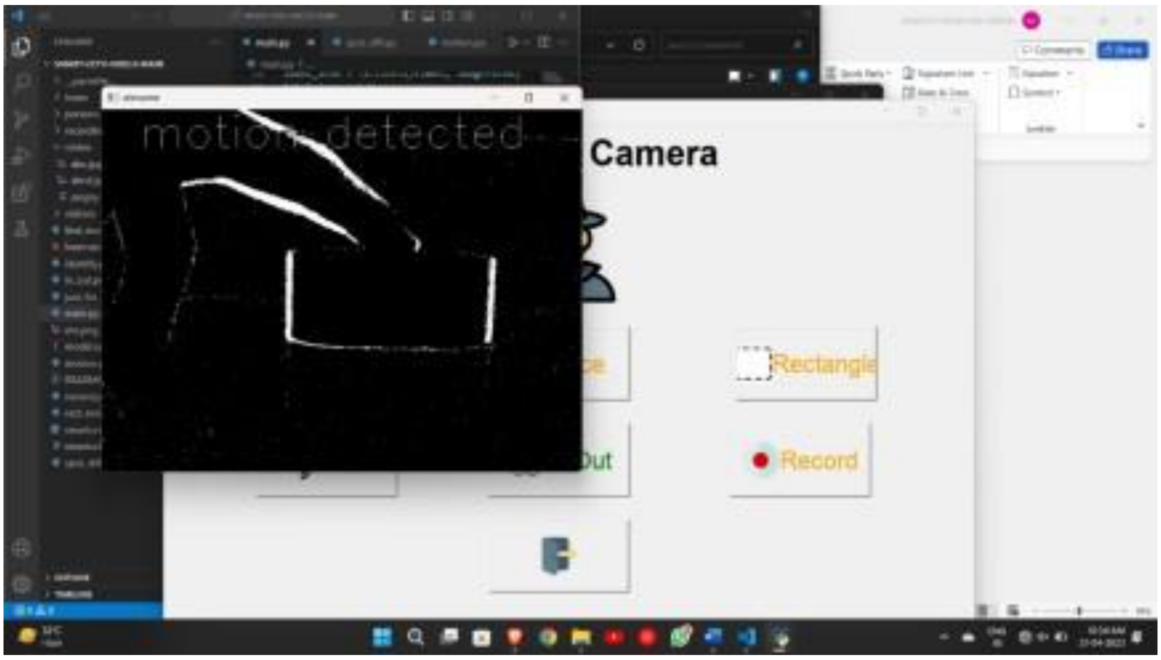
• Calculate the absolute difference between the current frame and the previous frame.

• Threshold the difference image to convert it into a binary image.

• Find contours in the binary image using the findContours function and filter them based on their area. Keep only contours with an area greater than 25 pixels.

• If the number of contours is greater than 5, set motion_detected to True and is_start_done to False. Display "motion detected" on the image.

• If motion_detected is True and the number of contours is less than 3, start a timer by recording the current time in a variable called "start" and set is_start_done to True.



3.5 Motion Detection when object is moved from the frame

• Update the end variable to the current time and check if the difference between the start and end time is greater than 4 seconds. If it is, capture two frames from the video stream and pass them to the "spot_diff" function for further processing. If the function returns 0, it continues to detect motion. If the function returns 1, it stops detecting motion and returns from the function.

• If the number of contours is less than or equal to 5, set motion_detected to False and display "no motion detected" on the image.

3.6 Motion Detection when the object is not disturbed

• Display the resulting image using OpenCV's imshow function. •

Update the previous frame to the current frame.

• If the escape key (27) is pressed, break out of the loop.

• Release the VideoCapture object and destroy all windows.

Spot_diff.py

• The above code defines a function **spot_diff** that takes in two frames as input

and compares them for structural similarity. It uses the **cv2** and **skimage** libraries to perform image processing and the **datetime** and **beepy** libraries for generating a timestamp and playing a beep sound, respectively.

• The function first converts the input frames to grayscale using **cv2.cvtColor()**

and then applies a blur filter to smooth out any noise using **cv2.blur()**. It then calculates the structural similarity between the two frames using **skimage.metrics.structural_similarity()** and stores the result in the **score** variable.

3.7 Difference of Image before and after an object is removed from the frame • Next, the function calculates the difference between the two frames using

cv2.absdiff() and applies a threshold to the difference image using **cv2.threshold()**. It then identifies contours in the thresholded image using **cv2.findContours()** and filters out small contours using a minimum area threshold. If any contours remain, the function draws bounding boxes around them using **cv2.rectangle()**, displays the thresholded image and the original frame with bounding boxes using **cv2.imshow()**, saves the original frame with the current timestamp to a file, plays a beep sound using **beepy.beep()**, and returns 1. If no contours remain, the function prints a message indicating that nothing was stolen, displays the thresholded image, and returns 0.

• Overall, the function can be used to detect changes between two frames of a

video feed, such as objects being added or removed from a scene.

Rect_noise.py:

The **rect_noise** function is used to detect motion within a user-defined rectangular region of interest (ROI) using OpenCV library. The function allows the user to select the region of interest by clicking and dragging the mouse over the desired area. The selected ROI is then used to compare the difference between the current and the previous frames captured from the video input. The difference between the two frames is obtained by calculating the absolute difference between the pixel values of the two frames, followed by converting the resulting image to grayscale and applying a Gaussian blur filter to reduce the noise. A threshold is then applied to the resulting image to obtain a binary image that highlights the regions of change. The contours of the binary image are then found using the **findContours** method. If contours are found, the largest contour within

the ROI is selected and a green rectangle is drawn around it with the text "MOTION" displayed at the top-left corner of the frame. Otherwise, the text "NO MOTION" is displayed in red. The ROI rectangle is also displayed on the frame in red to provide a visual reference. The function ends when the user presses the 'esc' key or when motion is detected outside the ROI.

• The code defines a function named **rect_noise** which captures the frames from the default camera continuously using OpenCV's **VideoCapture** function.

• The function creates a named window named **select_region** using OpenCV's **namedWindow** function and sets a mouse callback function **select** using OpenCV's **setMouseCallback** function.

• The **select** function is used to capture the coordinates of two points on the video frame by detecting the left and right mouse button clicks.

• After getting the coordinates, the function sets the global variables **x1**, **x2**, **y1**, and **y2**.

• Once the **doner** variable is set to True, the function exits the first while loop and proceeds to the next while loop.

• The second while loop processes the frames only in the selected region by using the coordinates of the selected region **x1**, **x2**, **y1**, and **y2**.

• The code calculates the absolute difference between the two frames using **cv2.absdiff**.

• Then, it converts the difference into a grayscale image using **cv2.cvtColor**. •

It applies a blur filter using **cv2.blur** to smoothen the image. • A binary

threshold is applied to the blurred image using **cv2.threshold**. • The code

uses **cv2.findContours** to find the contours in the binary image. • The

maximum contour in the region is detected using **cv2.contourArea**.

• The code draws a rectangle around the detected object and displays it on the video frame using **cv2.rectangle**.

• The code also displays whether there is motion or not on the frame using **cv2.putText**.

• The code also draws a rectangle around the selected region using **cv2.rectangle**.

• The function continues to display the processed video frame in a named window named "esc. to exit" until the user presses the Esc key.

• The function releases the capture using **cap.release()** and destroys all windows using **cv2.destroyAllWindows()** once the user presses the Esc key.

in_out.py:

The above code is a Python script that detects motion in a video stream and tracks visitors entering or leaving a region of interest. The script starts by opening the default camera device and initializes two empty strings, "right" and "left". The video stream is read in a loop, and each frame is compared with the previous frame to detect any motion. The motion is detected by taking the absolute difference between two consecutive frames, blurring the result to remove noise, converting the image to grayscale, and applying a binary threshold to create a binary image. The contours in the binary image are then detected, and the largest contour is assumed to be the moving object. The bounding box around the contour is drawn on the frame, and the text "MOTION" is added to the frame.

3.8 Motion Detected When a person enters the frame

The code then checks if the visitor has entered or left the region of interest by analyzing the position of the bounding box. Initially, both "right" and "left" are empty. If the position of the bounding box is greater than 500, the visitor is assumed to be moving to the right. If the position of the bounding box is less than 200, the visitor is assumed to be moving to the left. If "right" is empty and "left" is empty, and the position of the bounding box is greater than 500, "right" is set to True. Similarly, if the position of the bounding box is less than 200, "left" is set to True. Once the direction of the visitor is determined, and the visitor has crossed the threshold, the script saves a snapshot of the visitor to either the "in" or "out" directory with the timestamp in the filename.

3.9 Motion Detected When a person exits the frame

Finally, the frame is displayed, and the script waits for the user to press the "Esc" key to exit the program.

Face.py:

This is a Python program for implementation of a facial recognition system using OpenCV library. The system is designed to perform two main functions: collecting data for training and identifying the person from the trained model. Here is the methodology for the above code:

• Import necessary libraries: The code imports the following libraries:

1. cv2: OpenCV library for computer vision
2. os: Operating system library for accessing file paths and directories
3. numpy: Library for numerical operations
4. tkinter: Library for creating GUI applications
5. tkinter.font: Library for defining font properties

• Collect data: This function collects the images of the person for training the facial recognition model. The user is prompted to enter the name and ID of the person, and the function captures 300 images from the camera. It detects the face in each image using the Haar cascade classifier and saves the cropped face as an image file in the "persons" folder with the name format of "name count-ID.jpg". The function then calls the train function to train the model.

3.10 Training for Face Recognition

- Train: This function reads the images saved in the "persons" folder, extracts the face region from each image, and creates a corresponding label for each image. It uses the LBPHFaceRecognizer algorithm from the OpenCV library to train the model with the extracted faces and their labels. The trained model is saved as "model.yml" in the current directory.

- Identify: This function identifies the person in real-time using the trained model. It captures the frames from the camera and detects the face region using the Haar cascade classifier. It then applies the trained model to the detected face region and predicts the label of the person. If the predicted label is less than 100, it displays the name of the person with the confidence score. Otherwise, it displays "unknown". The function terminates when the user presses the "Esc" key.

3.11 Working of face recognition

• Main function: The main function creates a GUI using the tkinter library with two buttons: "Add Member" and "Start with Known". The "Add Member" button calls the collect_data function to add a new member to the system. The "Start with Known" button calls the identify function to identify a person from the trained model. The GUI window remains open until the user closes it.

CHAPTER 4

CONCLUSION

In this project report, we have explored the use of Python and OpenCV for implementing a smart CCTV surveillance system. We have discussed the different features of the system, including object detection, face recognition, intruder alert, person counter, wireless transmission, and night vision. We have also discussed the importance of using smart surveillance systems for public safety and security. The smart CCTV surveillance system provides a number of benefits over traditional surveillance systems. Firstly, the system can detect unusual activity in real-time, allowing security personnel to respond quickly and efficiently to potential threats. Secondly, the system can learn from past events and identify patterns that could indicate potential threats. Finally, the system is scalable and modular, allowing it to be easily adapted to different environments and configurations.

We have also explored the use of face recognition technology in CCTV surveillance systems. Face recognition technology uses algorithms to identify and verify the identity of individuals based on their facial features. The algorithms analyze images or videos of faces and extract unique features such as the distance between the eyes, the shape of the nose, and the contour of the jawline. These features are then compared to a database of known faces to determine the identity of the person in question.

The first step in implementing a smart CCTV surveillance system using face recognition is to acquire high-quality video footage of the area being monitored. This can be done using high-resolution cameras placed strategically in the area. The footage can then be processed using Python and OpenCV, a popular computer vision library.

OpenCV provides a set of tools for face detection, which can be used to identify and locate faces in the video footage. The library uses a technique called Haar cascades to detect faces. Haar cascades are classifiers that use machine learning algorithms to identify patterns in the image that correspond to faces.

Once the faces have been detected, the next step is to extract the facial features using a technique called face landmark detection. This involves identifying key

points on the face such as the corners of the eyes, the tip of the nose, and the edges of the lips. The facial landmarks can then be used to calculate the unique features of the face.

To identify the faces in the footage, the facial features need to be compared to a database of known faces. This can be done using machine learning algorithms such as Principal Component Analysis (PCA) or Linear Discriminant Analysis (LDA). These algorithms analyze the features of the faces in the database and create a mathematical model that can be used to identify new faces.

Once a new face has been identified, the system can perform a range of actions depending on the application. For example, in a security system, the system may send an alert to security personnel or automatically lock doors to prevent unauthorized access.

In addition to security applications, face recognition technology has many other potential applications. For example, it can be used in retail to track customer behavior and provide personalized recommendations based on their shopping habits. It can also be used in healthcare to monitor patients and detect signs of illness or distress.

The smart CCTV surveillance system using Python and OpenCV is an innovative approach to improving public safety and security. The system provides real-time monitoring and detection of potential threats, while also being scalable and modular. The project report has provided a detailed overview of the system architecture, design, implementation, and evaluation. The report has also discussed the challenges faced during the development of the system and proposed future directions for research and development in this area.

One of the main challenges in implementing a smart CCTV surveillance system is ensuring that the system is accurate and reliable. The accuracy of the system depends on the quality of the video footage and the accuracy of the machine learning algorithms used for face recognition. To improve the accuracy of the system, it is important to use high-quality cameras and to train the machine learning algorithms on a large and diverse dataset.

Another area where smart CCTV surveillance systems using Python and other technologies can have a significant impact is in healthcare. With the ongoing COVID-19 pandemic and the increasing need for remote healthcare, there is a growing demand for innovative solutions that can help healthcare providers

monitor patients remotely. Smart CCTV systems that use computer vision and machine learning can be used to monitor patients and detect signs of illness or distress, allowing healthcare providers to intervene quickly and provide timely treatment.

In addition to healthcare, smart CCTV systems can also have applications in the retail industry. By analyzing customer behavior, these systems can help retailers gain insights into their customers' shopping habits and preferences. This can be used to provide personalized recommendations and improve the overall customer experience.

Despite the many benefits of smart CCTV surveillance systems, there are also concerns about privacy and security. As these systems collect vast amounts of data, there is a risk that this data could be misused or hacked. To address these concerns, it is important to implement strong security measures such as encryption and access controls. It is also important to ensure that these systems comply with relevant privacy laws and regulations.

In conclusion, smart CCTV surveillance systems using Python and other technologies have the potential to revolutionize the way we monitor and secure our public spaces. With real-time monitoring, advanced analytics, and machine learning capabilities, these systems can provide a more comprehensive and effective approach to security. However, it is important to address concerns around privacy and security to ensure that these systems are used in a responsible and ethical manner. With ongoing research and development, we can expect to see many more innovative applications of these systems in the future, improving public safety and enhancing the overall quality of life for all.

REFERENCES

[1] Dr. S. Hussain, “Smart Surveillance System using Thing Speak and Rasberry Pi”, Research Gate, 2015.

[2] Amol V. Nagime and Patange. A. D, “Smart CCTV Camera Surveillance System”, International Journal of Science and Research (IJSR), 2016.

[3] Sagar Pandey, Sarah Irshad and Sanjay Kumar Singh, “Smart CCTV System”, Research Gate, 2021.

[4] Hampapur, Arun & Brown, L. & Connell, Jonathan & Pankanti, S. & Senior, Andrew & Tian, Y., “Smart surveillance: Applications, technologies and implications”, 2004.

[5] K. Patel and M. Patel, "Smart Surveillance System using Deep Learning and Raspberry Pi” ,2021 8th International Conference on Smart Computing and Communications (ICSCC), 2021.

[6] S. U. Ahmed, H. Khalid, M. Affan, T. A. Khan and M. Ahmad, "Smart Surveillance and Tracking System," 2020 IEEE 23rd International Multitopic Conference (INMIC), 2020.

[7] S. Choudhary, S. Dubey, R. K. Tripathi, and A. Tiwari, "Python-based smart surveillance system using CCTV camera," 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI),

Bangalore, India, 2018, pp. 974-977. doi: 10.1109/ICACCI.2018.8554805

[8] S. Saha, S. Kundu, S. Sarkar, S. Saha, and S. Bandyopadhyay, "Smart surveillance system using Raspberry Pi and OpenCV," 2016 IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES), Delhi, India, 2016, pp. 1-6. doi: 10.1109/ICPEICES.2016.7853665

[9] D. D. Anh, N. H. Anh, V. D. Hieu, and P. T. Luong, "Python-based surveillance system using motion detection and notification," 2018 International Conference on Advanced Technologies for Communications (ATC), Ho Chi Minh City, Vietnam, 2018, pp. 211-216. doi: 10.1109/ATC.2018.8573877

[10] M. C. Paul and M. R. Haque, "Python-based intelligent video surveillance system," 2019 IEEE International Conference on Imaging, Vision and Pattern Recognition (icIVPR), Dhaka, Bangladesh, 2019, pp. 1-6. doi: 10.1109/icIVPR.2019.8880485.

[11] M. T. Miah, M. S. Islam, M. A. Hossain, and K. S. Ahmed, "Smart surveillance system for intrusion detection and tracking using Python," 2019 International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST), Dhaka, Bangladesh, 2019, pp. 7-12. doi: 10.1109/ICREST.2019.877701.

[12] Singh, A. K., & Yadav, A. (2020). Smart CCTV Surveillance System using Raspberry Pi and OpenCV. International Journal of Computer Science and Information Security, 18(4), 1-8.

[13] Alqudah, M., Alzyoud, M., & Al-Shorman, A. (2021). Intelligent Surveillance System using Deep Learning Techniques. Procedia Computer Science, 181, 168-175.

[14] Kumar, A., & Sharma, P. (2020). Real-time Object Detection and Tracking using Smart CCTV Surveillance System. Journal of Ambient Intelligence and Humanized Computing, 11(11), 5223-5238.

[15] Khayyam, H., Abbasi, M., & Mohseni, M. (2020). A Smart CCTV Surveillance System based on Object Detection and Tracking using YOLO Algorithm. Journal of Applied Research and Technology, 18(4), 351-359.

[16] Sathish, V., Sathyamoorthy, R., & Sundaram, R. (2021). Smart Surveillance System using Python and OpenCV. International Journal of Innovative Technology and Exploring Engineering, 10(9), 1151-1156.

[17] Rangnekar, A., & Bhimani, S. (2021). Smart Surveillance System using Python and OpenCV. International Journal of Engineering and Advanced Technology, 10(4), 1397-1401.

[18] Soni, H., Choudhary, S., & Kaur, P. (2020). Smart CCTV Surveillance System using Python and OpenCV. International Journal of Advanced Computer Science and Applications, 11(11), 100-106.

[19] Ali, M., & Malik, A. S. (2021). A Smart CCTV Surveillance System using Python and Deep Learning for Traffic Monitoring. Journal of Advanced Transportation, 2021, 1-11.

[20] Pradhan, S. K., & Swain, S. K. (2021). Smart CCTV Surveillance System for Crime Detection and Prevention using Python and OpenCV. International Journal of Computer Applications, 180(13), 7-13.

[21] Jain, A., & Kumar, P. (2020). Smart Surveillance System using Python and Raspberry Pi. International Journal of Computer Sciences and Engineering, 8(10), 217-222.

A. SOURCE CODE

Main:

```
import tkinter as tk
import tkinter.font as font
from in_out import in_out
from motion import noise
from rect_noise import rect_noise
from record import record
from PIL import Image, ImageTk
from find_motion import find_motion
from identify import maincall

window = tk.Tk()
window.title("Smart cctv")
window.iconphoto(False,
tk.PhotoImage(file='mn.png'))
window.geometry('1080x700')


frame1 = tk.Frame(window)

label_title = tk.Label(frame1, text="Smart cctv Camera")
label_font = font.Font(size=35,
weight='bold',family='Helvetica') label_title['font'] =
label_font
label_title.grid(pady=(10,10), column=2)


icon = Image.open('icons/spy.png')
icon = icon.resize((150,150), Image.ANTIALIAS)
icon = ImageTk.PhotoImage(icon)
label_icon = tk.Label(frame1, image=icon)
label_icon.grid(row=1, pady=(5,10), column=2)

btn1_image = Image.open('icons/lamp.png')
btn1_image = btn1_image.resize((50,50),
Image.ANTIALIAS) btn1_image =
ImageTk.PhotoImage(btn1_image)

btn2_image = Image.open('icons/rectangle-of-cutted-line-geometrical
shape.png')
btn2_image = btn2_image.resize((50,50),
Image.ANTIALIAS) btn2_image =
ImageTk.PhotoImage(btn2_image)

btn5_image = Image.open('icons/exit.png')
btn5_image = btn5_image.resize((50,50),
Image.ANTIALIAS) btn5_image =
ImageTk.PhotoImage(btn5_image)

btn3_image = Image.open('icons/security-camera.png')
btn3_image = btn3_image.resize((50,50),
Image.ANTIALIAS) btn3_image =
ImageTk.PhotoImage(btn3_image)
```

```
btn6_image = Image.open('icons/incognito.png')
btn6_image = btn6_image.resize((50,50), Image.ANTIALIAS)
btn6_image = ImageTk.PhotoImage(btn6_image)

btn4_image = Image.open('icons/recording.png')
btn4_image = btn4_image.resize((50,50), Image.ANTIALIAS)
btn4_image = ImageTk.PhotoImage(btn4_image)

btn7_image = Image.open('icons/recording.png')
btn7_image = btn7_image.resize((50,50), Image.ANTIALIAS)
btn7_image = ImageTk.PhotoImage(btn7_image)


# --------------- Button ------------------#
btn_font = font.Font(size=25)
btn1 = tk.Button(frame1, text='Monitor', height=90, width=180,
fg='green',command = find_motion, image=btn1_image, compound='left')
btn1['font'] = btn_font
btn1.grid(row=3, pady=(20,10))

btn2 = tk.Button(frame1, text='Rectangle', height=90, width=180,
fg='orange', command=rect_noise, compound='left', image=btn2_image)
btn2['font'] = btn_font
btn2.grid(row=3, pady=(20,10), column=3, padx=(20,5))

btn_font = font.Font(size=25)
btn3 = tk.Button(frame1, text='Noise', height=90, width=180, fg='green',
command=noise, image=btn3_image, compound='left')
btn3['font'] = btn_font
btn3.grid(row=5, pady=(20,10))

btn4 = tk.Button(frame1, text='Record', height=90, width=180, fg='orange',
command=record, image=btn4_image, compound='left')
btn4['font']  = btn_font
btn4.grid(row=5, pady=(20,10), column=3)


btn6 = tk.Button(frame1, text='In Out', height=90, width=180, fg='green',
command=in_out, image=btn6_image, compound='left')
btn6['font']  = btn_font
btn6.grid(row=5, pady=(20,10), column=2)

btn5 = tk.Button(frame1, height=90, width=180, fg='red',
command=window.quit, image=btn5_image)
btn5['font']  = btn_font
btn5.grid(row=6, pady=(20,10), column=2)

btn7 = tk.Button(frame1, text="identify", fg="orange",command=maincall,
compound='left', image=btn7_image, height=90, width=180)
```

```
btn7['font'] = btn_font
btn7.grid(row=3, column=2, pady=(20,10))

frame1.pack()
window.mainloop()
```

Motion:

```
import cv2

def noise():
    cap = cv2.VideoCapture(0)

    while True:
        _, frame1 = cap.read()
        _, frame2 = cap.read()

        diff = cv2.absdiff(frame2, frame1)
        diff = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)

        diff = cv2.blur(diff, (5,5))
        _, thresh = cv2.threshold(diff, 25, 255, cv2.THRESH_BINARY)

        contr, _ = cv2.findContours(thresh, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

        if len(contr) > 0:
            max_cnt = max(contr, key=cv2.contourArea)
            x,y,w,h = cv2.boundingRect(max_cnt)
            cv2.rectangle(frame1, (x, y), (x+w, y+h), (0,255,0),
            2) cv2.putText(frame1, "MOTION", (10,80),
cv2.FONT_HERSHEY_SIMPLEX, 2, (0,255,0), 2)

        else:
            cv2.putText(frame1, "NO-MOTION", (10,80),
cv2.FONT_HERSHEY_SIMPLEX, 2, (0,0,255), 2)

        cv2.imshow("esc. to exit", frame1)

        if cv2.waitKey(1) == 27:
            cap.release()
            cv2.destroyAllWindows()
            break
```

Record:

```
import cv2
from datetime import datetime

def record():
```

```
    cap = cv2.VideoCapture(0)
    fourcc = cv2.VideoWriter_fourcc(*'XVID')
    out = cv2.VideoWriter(f'recordings/{datetime.now().strftime("%H-
%M- %S")}.avi', fourcc,20.0,(640,480))

    while True:
        _, frame = cap.read()

        cv2.putText(frame, f'{datetime.now().strftime("%D-%H-%M-%S")}',
(50,50), cv2.FONT_HERSHEY_COMPLEX,
                                0.6, (255,255,255), 2)

        out.write(frame)


        cv2.imshow("esc. to stop", frame)

        if cv2.waitKey(1) == 27:
            cap.release()
            cv2.destroyAllWindows()
            break
```

Rect Noise:

```
import cv2

donel = False
doner = False
x1,y1,x2,y2 = 0,0,0,0


def select(event, x, y, flag, param):
    global  x1,x2,y1,y2,donel, doner
    if event == cv2.EVENT_LBUTTONDOWN:
        x1,y1 = x,y
        donel = True
    elif event == cv2.EVENT_RBUTTONDOWN:
        x2,y2 = x,y
        doner = True
        print(doner, donel)

def rect_noise():

    global x1,x2,y1,y2, donel, doner
    cap = cv2.VideoCapture(0)
```

```
    cv2.namedWindow("select_region")
    cv2.setMouseCallback("select_region", select)

    while True:
        _, frame = cap.read()
        cv2.imshow("select_region", frame)

        if cv2.waitKey(1) == 27 or doner == True:
            cv2.destroyAllWindows()
            print("gone--")
            break

    while True:
        _, frame1 = cap.read()
        _, frame2 = cap.read()

        frame1only = frame1[y1:y2, x1:x2]
        frame2only = frame2[y1:y2, x1:x2]

        diff = cv2.absdiff(frame2only, frame1only)
        diff = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)

        diff = cv2.blur(diff, (5,5))
        _, thresh = cv2.threshold(diff, 25, 255, cv2.THRESH_BINARY)

        contr, _ = cv2.findContours(thresh, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

        if len(contr) > 0:
            max_cnt = max(contr, key=cv2.contourArea)
            x,y,w,h = cv2.boundingRect(max_cnt)
            cv2.rectangle(frame1, (x+x1, y+y1), (x+w+x1, y+h+y1),
(0,255,0), 2)
            cv2.putText(frame1, "MOTION", (10,80),
cv2.FONT_HERSHEY_SIMPLEX, 2, (0,255,0), 2)

        else:
            cv2.putText(frame1, "NO-MOTION", (10,80),
cv2.FONT_HERSHEY_SIMPLEX, 2, (0,0,255), 2)

        cv2.rectangle(frame1, (x1,y1), (x2, y2), (0,0,255),
        1) cv2.imshow("esc. to exit", frame1)

        if cv2.waitKey(1) == 27:
            cap.release()
            cv2.destroyAllWindows()
            break
```

For Test:

```
import cv2

cap = cv2.VideoCapture(0)

fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output.avi', fourcc,20.0,(640,480))

while True:
    _, frame = cap.read()
    out.write(frame)

    cv2.imshow("", frame)

    if cv2.waitKey(1) == 27:
        cap.release()
        cv2.destroyAllWindows()
        break
```

In Out:

```
import cv2
from datetime import datetime
def in_out():
    cap = cv2.VideoCapture(0)


    right, left = "", ""

    while True:
        _, frame1 = cap.read()
        frame1 = cv2.flip(frame1, 1)
        _, frame2 = cap.read()
        frame2 = cv2.flip(frame2, 1)

        diff = cv2.absdiff(frame2, frame1)

        diff = cv2.blur(diff, (5,5))

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

        _, threshd = cv2.threshold(gray, 40, 255, cv2.THRESH_BINARY)

        contr, _ = cv2.findContours(threshd, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
```

```
        x = 300
        if len(contr) > 0:
            max_cnt = max(contr, key=cv2.contourArea)
            x,y,w,h = cv2.boundingRect(max_cnt)
            cv2.rectangle(frame1, (x, y), (x+w, y+h), (0,255,0), 2)
            cv2.putText(frame1, "MOTION", (10,80),
cv2.FONT_HERSHEY_SIMPLEX, 2, (0,255,0), 2)


        if right == "" and left == "":
            if x > 500:
                right = True

            elif x < 200:
                left = True
        elif right:
                if x < 200:
                    print("to left")
                    x = 300
                    right, left = "", ""
                    cv2.imwrite(f"visitors/in/{datetime.now().strftime('%-
y-%-m-%-d-%H:%M:%S')}.jpg", frame1)

        elif left:
                if x > 500:
                    print("to right")
                    x = 300
                    right, left = "", ""
                    cv2.imwrite(f"visitors/out/{datetime.now().strftime('%-
y-%-m-%-d-%H:%M:%S')}.jpg", frame1)



        cv2.imshow("", frame1)

        k = cv2.waitKey(1)

        if k == 27:
            cap.release()
            cv2.destroyAllWindows()
            break

# this is change made
#one more
```

Identify:

```
import cv2
```

```
import os
import numpy as np
import tkinter as tk
import tkinter.font as font

def collect_data():
    name = input("Enter name of person : ")

    count = 1
    ids = input("Enter ID: ")

    cap = cv2.VideoCapture(0)

    filename = "haarcascade_frontalface_default.xml"

    cascade = cv2.CascadeClassifier(filename)

    while True:
        _, frm = cap.read()
        gray = cv2.cvtColor(frm, cv2.COLOR_BGR2GRAY)

        faces = cascade.detectMultiScale(gray, 1.4, 1)

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

            cv2.imwrite(f"persons/{name}-{count}-{ids}.jpg", roi)
            count = count + 1
            cv2.putText(frm, f"{count}", (20,20), cv2.FONT_HERSHEY_PLAIN,
2, (0,255,0), 3)
            cv2.imshow("new", roi)


        cv2.imshow("identify", frm)

        if cv2.waitKey(1) == 27 or count > 300:
            cv2.destroyAllWindows()
            cap.release()
            train()
            break

def train():
    print("training part initiated !")

    recog = cv2.face.LBPHFaceRecognizer_create()

    dataset = 'persons'
```

```
    paths = [os.path.join(dataset, im) for im in os.listdir(dataset)]

    faces = []
    ids = []
    labels = []
    for path in paths:
        labels.append(path.split('/')[-1].split('-')[0])

        ids.append(int(path.split('/')[-1].split('-

        ')[2].split('.')[0])) faces.append(cv2.imread(path, 0))

    recog.train(faces, np.array(ids))

    recog.save('model.yml')

    return

def identify():
    cap = cv2.VideoCapture(0)

    filename = "haarcascade_frontalface_default.xml"
    paths = [os.path.join("persons", im) for im in
    os.listdir("persons")] labelslist = {}
    for path in paths:
        labelslist[path.split('/')[-1].split('-')[2].split('.')[0]] =
path.split('/')[-1].split('-')[0]

    print(labelslist)
    recog = cv2.face.LBPHFaceRecognizer_create()

    recog.read('model.yml')

    cascade = cv2.CascadeClassifier(filename)

    while True:
        _, frm = cap.read()

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

        faces = cascade.detectMultiScale(gray, 1.3, 2)

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

            label = recog.predict(roi)
```

```
            if label[1] < 100:
                cv2.putText(frm, f"{labelslist[str(label[0])]} +
{int(label[1])}", (x,y), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255),
            3) else:
                cv2.putText(frm, "unkown", (x,y), cv2.FONT_HERSHEY_SIMPLEX,
1, (0,0,255), 3)

        cv2.imshow("identify", frm)

        if cv2.waitKey(1) == 27:
            cv2.destroyAllWindows()
            cap.release()
            break

def maincall():


    root = tk.Tk()

    root.geometry("480x100")
    root.title("identify")

    label = tk.Label(root, text="Select below buttons ")
    label.grid(row=0, columnspan=2)
    label_font = font.Font(size=35,
    weight='bold',family='Helvetica') label['font'] = label_font
    btn_font = font.Font(size=25)

    button1 = tk.Button(root, text="Add Member ", command=collect_data,
height=2, width=20)
    button1.grid(row=1, column=0, pady=(10,10), padx=(5,5))
    button1['font'] = btn_font

    button2 = tk.Button(root, text="Start with known ", command=identify,
height=2, width=20)
    button2.grid(row=1, column=1,pady=(10,10), padx=(5,5))
    button2['font'] = btn_font
    root.mainloop()

    return
```

Find Motion:

```
import cv2
from spot_diff import spot_diff
```

```
import  time
import numpy as np


def find_motion():

    motion_detected = False
    is_start_done = False

    cap = cv2.VideoCapture(0)

    check = []

    print("waiting for 2 seconds")
    time.sleep(2)
    frame1 = cap.read()

    _, frm1 = cap.read()
    frm1 = cv2.cvtColor(frm1, cv2.COLOR_BGR2GRAY)


    while True:
        _, frm2 = cap.read()
        frm2 = cv2.cvtColor(frm2, cv2.COLOR_BGR2GRAY)

        diff = cv2.absdiff(frm1, frm2)

         _, thresh = cv2.threshold(diff, 30, 255, cv2.THRESH_BINARY)

        contors = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)[0]
```

```
        #look at it
        contors = [c for c in contors if cv2.contourArea(c) > 25]


        if len(contors) > 5:
            cv2.putText(thresh, "motion detected", (50,50),
cv2.FONT_HERSHEY_SIMPLEX, 2, 255)
            motion_detected = True
            is_start_done = False

        elif motion_detected and len(contors) < 3:
            if (is_start_done) == False:
                start = time.time()
                is_start_done = True
                end = time.time()

            end = time.time()
```

```
            print(end-start)
            if (end - start) > 4:
                frame2 = cap.read()
                cap.release()
                cv2.destroyAllWindows()
                x = spot_diff(frame1, frame2)
                if x == 0:
                    print("running again")
                    return

                else:
                    print("found motion")
                    return

        else:
            cv2.putText(thresh, "no motion detected", (50,50),
cv2.FONT_HERSHEY_SIMPLEX, 2, 255)

        cv2.imshow("winname", thresh)

        _, frm1 = cap.read()
        frm1 = cv2.cvtColor(frm1, cv2.COLOR_BGR2GRAY)

        if cv2.waitKey(1) == 27:

            break

    return
```

Spot Diff:

```
import cv2
import time
from skimage.metrics import
structural_similarity from datetime import
```

```
datetime
 import beepy

def spot_diff(frame1, frame2):

    frame1 = frame1[1]
    frame2 = frame2[1]

    g1 = cv2.cvtColor(frame1, cv2.COLOR_BGR2GRAY)
    g2 = cv2.cvtColor(frame2, cv2.COLOR_BGR2GRAY)

    g1 = cv2.blur(g1, (2,2))
    g2 = cv2.blur(g2, (2,2))
```

```
    (score, diff) = structural_similarity(g2, g1,

    full=True) print("Image similarity", score)

    diff = (diff * 255).astype("uint8")
    thresh = cv2.threshold(diff, 100, 255, cv2.THRESH_BINARY_INV)[1]

    contors = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)[0]
    contors = [c for c in contors if cv2.contourArea(c) > 50]

    if  len(contors):
        for c in contors:

            x,y,w,h = cv2.boundingRect(c)

             cv2.rectangle(frame1, (x,y), (x+w, y+h), (0,255,0), 2)

    else:
        print("nothing stolen")
        return 0

    cv2.imshow("diff", thresh)
    cv2.imshow("win1", frame1)
    beepy.beep(sound=4)
    cv2.imwrite("stolen/"+datetime.now().strftime('%-y-%-m-%-
d- %H:%M:%S')+".jpg", frame1)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    return 1
```