MOTION AND FIRE DETECTION SYSTEM USING CAMERAS

Final Year Project Report

Submitted By

Suryansh Choudhary (BA034-21)

MBA-BA 2021-23

Guided By

Prof. Pradip Kumar Bala

Information Systems & Business Analytics

INDIAN INSTITUTE OF MANAGEMENT RANCHI

March 2023

Acknowledgment

I would like to extend my heartfelt gratitude to my esteemed guide, Prof. Pradip Kumar Bala, for his unwavering support and guidance throughout my MBA program at the Indian Institute of Management Ranchi. Prof. Pradip Kumar Bala’s vast knowledge and expertise in the field of Information Systems & Business Analytics has been a constant source of inspiration for me and has helped me immensely in enhancing my understanding of the subject.

I am also immensely grateful to the institute’s faculty members for their exceptional teaching and mentorship. Their unwavering commitment to providing us with a comprehensive and insightful education has been instrumental in shaping my academic journey.

Furthermore, I would like to extend my heartfelt thanks to my classmates for their camaraderie, support, and encouragement throughout my time at the institute. They have been an integral part of my academic journey, and their unwavering support has helped me achieve my academic goals.

Finally, I would like to thank my family and friends for their endless love, support, and encouragement throughout my academic journey. Their unwavering support has been a constant source of strength for me and has helped me to overcome every obstacle that came my way.

Abstract

This project is a fusion of two separate projects: one that detects motion and another that detects fire. OpenCV and a webcam were used to construct both projects. The system’s purpose is to showcase computer vision technology’s capabilities in the context of the safety and security industries.

In the first project, motion detection is accomplished through image processing techniques. When motion is detected for the first time, a message is sent to the user using a telegram bot. This notification includes a timestamp as well as an image of the motion that was identified. In addition, a CSV file is produced in order to log the date, in addition to the beginning and ending times of the move.

In the second project, fire detection is accomplished using HAAR Cascade files that have been pre-installed. This causes a notification to be sent to the user through a telegram bot on the first instance of a fire. The notification contains a timestamp and an image of the fire that has been detected. Following then, the user receives a notification once every ten minutes in the event that fresh fires occur.

Integrating these two separate initiatives results in an elevated level of protection and assurance the system provides. Because the integration of the Telegram bot enables real-time monitoring and reaction to potential threats, it is now possible to take prompt and appropriate action. The incorporation of photos into the alerts offers visual confirmation of the events that have been identified. This enables the user to determine the location of the occurrence and respond appropriately. The CSV file that is part of the initial project makes it possible to generate a log of occurrences, which, after the fact, may be investigated further for purposes such as trend analysis or additional research.

The project, on its whole, shows the need for real-time monitoring and responding to possible threats and the practical uses of computer vision technology in security and safety sectors.

Table of Contents

List of Figures

List of Abbreviations

API	Application Programming Interface
AVG	Average
CNN	Convolution Neural Network
CSV	Comma Separated Values
HOG	Histogram of Oriented Gradients
MATLAB	Matrix Laboratory
OPENCV	Open-Source Computer Vision Library
OS	Operating System
PC	Personal Computer
ROI	Region of Interest
SSD	Single Shot Detection
YOLO	You Only Look Once

Motion Detection Surveillance System

Introduction

Motion detection is a job that is very necessary for applications involving computer vision and surveillance. It entails examining a series of photographs to determine whether there have been any shifts in the location, orientation, or appearance of the items in the scene. It is helpful in various domains, including crowd analysis, determining the gender of individuals, and recognizing anomalous human behavior in public settings. Using a camera in conjunction with OpenCV, a very effective open-source toolkit for computer vision and image processing is a common approach to putting motion detection into practice.

OpenCV is a freely accessible software library for machine learning, computer vision, and other applications. It is used for real-time vision-based applications and has a vast collection of algorithms that can be used to solve various computer vision problems. OpenCV supports various programming languages, including C++, Python, Java, and MATLAB.

The first step in determining whether or not the motion is present using a camera and OpenCV is to capture several frames from the webcam. After that, these frames are processed and compared to look for differences in the scene. Developing a backdrop model that depicts the immobile components of the scene, such as the walls, floors, and furniture, is one method that can be utilized.

A different picture is created by subtracting the active frame from the backdrop model. This allows for the detection of motion. The difference image draws attention to any alterations that have taken place in the scene, such as the movement of items or alterations in the lighting conditions. After that, a threshold is applied to this different picture to produce a binary image, in which each pixel represents either the presence or absence of motion and can be either black or white.

The generated binary picture can be subjected to further processing to achieve higher precision in motion detection. The use of morphological procedures, such as erosion and dilation, is a typical method that may be used to clean up the binary picture by removing noise and filling in gaps.

If motion has been identified, the system can do various activities, including capturing video, activating an alarm, and sending a warning. When a user receives a notice, a Telegram bot may send a message to the user's phone or computer, providing the user with information on the time and location of the move.

Motion detection using a camera and OpenCV is a handy tool for various applications, including surveillance, security, and home automation. It provides a strong and flexible solution for real-time monitoring and alerting because of its ability to detect and respond to environmental changes. There are now cameras that have been created explicitly for surveillance in nearly every area of modern life. There are many other applications for cameras outside just surveillance. In addition, it makes it much simpler for individuals to zero in on fascinating destinations and goals, as well as carry out the responsibilities that are expected of them. The ability to locate, identify, position, and follow targets is critical in computer vision.

Literature Review

Detecting moving items in a scene is essential to computer vision's motion detection process, including tracking those moving things. This activity has many potential uses, including surveillance, home automation, and security.

In recent years, several research have concentrated on the development of camera and OpenCV-based motion detection systems. One of the field's earliest investigations was undertaken by Kaehler and Bradski (2017), who developed a method for motion recognition employing background subtraction and image differencing.

The system used a stationary background image to detect changes in the scene and extracted the moving objects by subtracting the current image from the background. The resulting image was then thresholded to obtain a binary image, where each pixel was either black or white, indicating the presence or absence of motion.

Subsequently, several studies have focused on improving the accuracy and robustness of motion detection systems using webcams and OpenCV. For example, Navaux et al. (2012) developed a method to identify the presence of individuals in a room by combining motion detection with facial recognition. The system used background subtraction and morphological operations to extract the moving objects and then applied face recognition algorithms to identify the presence of people in the scene. The system achieved an accuracy of up to 98%.

In addition, additional research has investigated the use of background removal in conjunction with feature-based motion detection algorithms. For instance, Mohamed et al. (2016) created a feature extraction and matching system to identify and track moving objects in a video sequence. This system was able to recognize and track the objects. To determine whether or not there was motion, it first extracted features from each frame, such as corners and edges, and then compared those features to the features in the frame before it.

Recently, techniques from deep learning have been applied to motion detection using cameras and OpenCV. In the research carried out by Han et al. (2020), an open-source machine learning framework called TensorFlow was used in conjunction with a camera in order to construct a deep learning-based motion detection system. In order to identify motion in real time, the system used a convolutional neural network (CNN), which enabled it to achieve an accuracy of up to 95%.

Furthermore, various applications have been developed for motion detection using webcams and OpenCV. For example, a home security system was developed by Patel et al. (2017), which used a webcam and OpenCV to detect and notify users of any motion in the scene. The system also used a Raspberry Pi to send notifications and store images in the cloud.

In conclusion, motion detection using webcams and OpenCV is valuable for surveillance, security, and home automation applications. Detecting and responding to environmental changes offers a robust, flexible real-time monitoring and notification solution. Future research in this field may explore further integrating deep learning and neural networks to improve motion detection systems' accuracy and effectiveness.

Methodology

The project uses the webcam of the system to capture the video stream. The video frames are processed using OpenCV. The principle of motion detection is based on the difference between the most recent frame and the average of the recent frames. The process is summarized in the following steps:

1. **Capture the webcam's live video feed.**
   The project uses the OpenCV VideoCapture() function to capture the webcam's video feed. The function inputs the device index or the video file's name. The video stream is then read frame by frame using the read() function.
2. **Convert the video frames to grayscale**
   The video frames are converted to grayscale using OpenCV's cvtColor() function. The function converts the color image to grayscale using various color conversion algorithms. The grayscale image is easier to process and requires less computation.
3. **Average out the recent frames**
   The recent frames are averaged out using the accumulateWeighted() function in OpenCV. The function takes the input frame, the output frame, and the weight of the input frame as input. The input frame's

weight determines the input frame's contribution to the output frame. The function is used to create a running average of the recent frames.

4. **Subtract the recent frame from the average**
   The current frame is subtracted from the average of the recent frames using the absdiff() function in OpenCV. The function calculates the absolute difference between the two frames. The result is a grayscale image with pixel values representing the difference between the frames.
5. **Threshold the difference image**
   The difference image is thresholded using the threshold() function in OpenCV. The function takes the input image and a threshold value as input. The function sets the pixel value to 255 if the value is above the minimum threshold and 0 otherwise. The threshold value is set based on the sensitivity of the motion detection.
6. **Find the contours in the thresholded image**
   The contours in the thresholded image are found using OpenCV's findContours() function. The function takes the input image and the contour retrieval mode as input. The function returns a list of contours detected in the image.
7. **Send notification to the Telegram bot**
   A Telegram bot is notified if the project detects motion in the video stream. The notification comprises the time and date of motion and an image. The Telegram bot is created using the Telegram API.
8. **Transfer the information to a CSV file**
   The project uses the CSV module available in Python to write the motion's date and start and end time to a CSV file. The CSV file works as a log for the motion detection events.
9. **Set a timer for subsequent notifications**
   If a motion is detected in the video stream, the project sets a timer for 10 minutes. After this period, a notification is delivered to the Telegram bot in the event of a new motion.

Hardware and Software Requirements

The following hardware and software specifications are required for implementing motion detection using a webcam and OpenCV:

Hardware

- Webcam
- Computer

Software

- Python 3
- OpenCV 4
- Telegram Bot API
- Pandas Library

Implementation

To understand the working of the model, we have divided the whole model into various steps, which are explained below with the codes, and the complete source code can be found in **Exhibit A**.

Setting up the Environment

Installing the required libraries is vital in preparing the environment for working with OpenCV; nevertheless, this step must be completed before we can begin. For this particular project, we will be utilizing Python 3. The Python package manager, pip, may be utilized to install OpenCV successfully.

pip install opencv-python

In addition to that, we are going to make use of the Telegram API in order to send alerts to a Telegram bot. Installing the telepot library is necessary for using the Telegram application programming interface (API).

pip install telepot

Before starting the motion detection process, we must configure our webcam correctly. The webcam should be connected to the computer, and the drivers should be installed. We can check whether our webcam is working correctly by opening a webcam application. We can proceed with the following steps if the webcam is working correctly.

OpenCV is a computer vision library that provides tools for processing images and computer vision tasks. It is a freely available library available for download from the official website. We must download and install the most recent version of OpenCV on our machine. We can install OpenCV using the pip package manager.

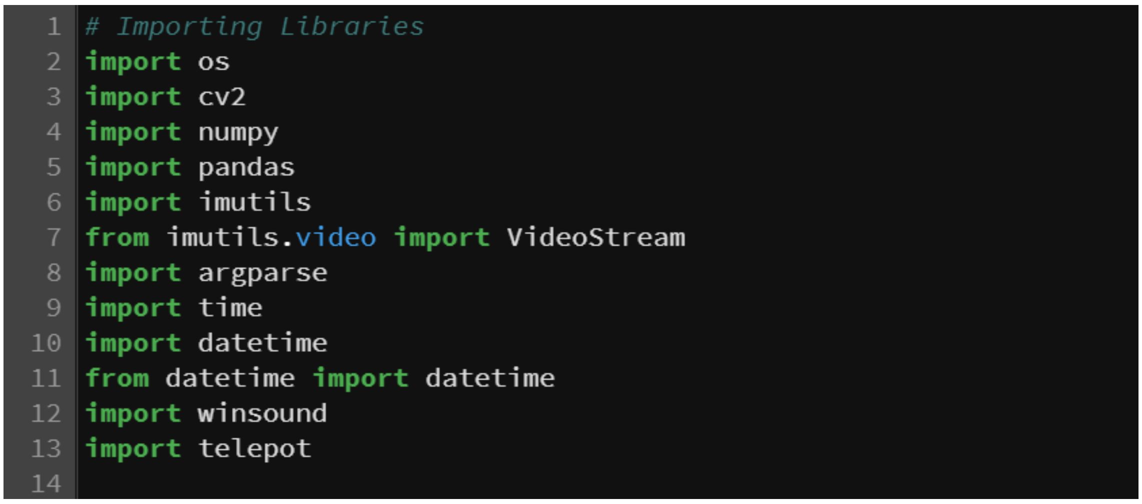
pip install opencv-python

Importing Libraries

The first step in a python code is importing the necessary packages and libraries. We have used the following libraries in our code:

- **os**
  The OS library in Python offers methods for operating system interaction. OS is included in the standard Python utility modules. This module offers an adaptable method for utilizing capabilities depending on the operating system. It includes many functions to interact with the file system.
- **cv2**
  OpenCV is a Python package that enables the processing of images and computer vision operations. It is a freely available library that captures frames from the webcam, which can be further used for processing and detecting motion.
- **NumPy**
  NumPy is a Python package for manipulating arrays. It also incorporates mathematical principles, the Fourier series, and matrix operations.
- **pandas**
  The pandas is a Python package used for data set manipulation. It provides capabilities for data analysis, cleansing, exploration, and manipulation.
- **imutils**
  A collection of OpenCV convenience functions that ease core image processing operations such as translations, rotations, scaling, segmentation, displaying Matplotlib images, classifying contours, and recognizing edges.
- **argparse**
  The argparse makes intuitive command-line interactions easy. The application specifies the parameters, and argparse extracts them from sys.argv. Argparse automatically produces assistance and usage messages. Incorrect program parameters also cause module faults.
- **time**
  This library provides various time-related functions.
- **datetime**
  The datetime module provides date and time manipulation classes. While date and time arithmetic are available, the integration emphasizes attribute extraction for effective output structuring and handling.

- **winsound**
  The winsound package gives access to the Windows platform's fundamental sound-playing infrastructure. It contains functions and constants.
- **telepot**
  Telepot helps to build applications for Telegram Bot API. It is being used to send notifications to Telegram.

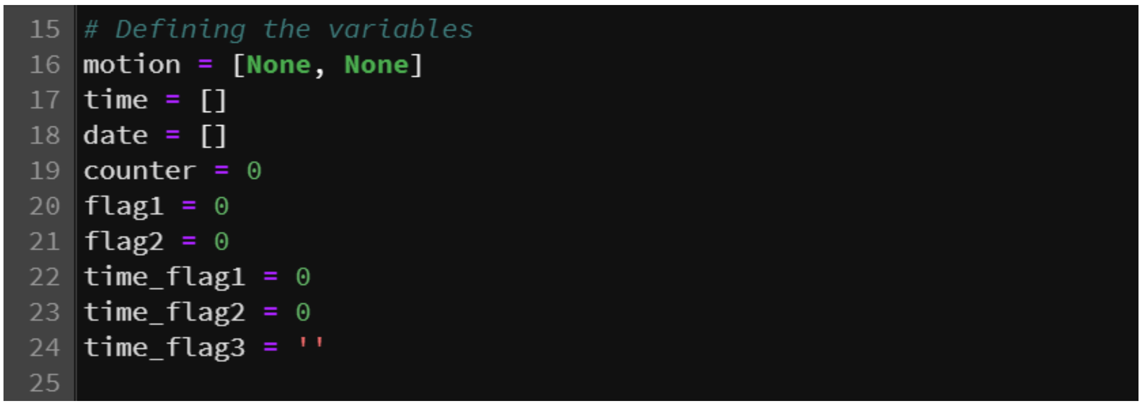


```python
# Importing Libraries
import os
import cv2
import numpy
import pandas
import imutils
from imutils.video import VideoStream
import argparse
import time
import datetime
from datetime import datetime
import winsound
import telepot

```

Fig 1: Screenshot of Code for Importing Libraries

Initializing Variables

We use the **"motion"** list to store all the instances of motion. Initially, we assign a None value to it as there is no motion at the start of the model. The **"time"** and **"date"** lists are used to store the date and time instances of motion, which would be used later to create the security log file. The model later uses the "**counter"** variable as a flag for the alarm system. The **"flag1"** is used as a flag for motion instances which would be used later to store start and end dates and time of motion, and **"flag2"** is used to count the number of instances of motion and perform certain functions using it. The **"time_flag1" and "time_flag2"** are being used to store the instances of time, and **"time_flag3"** are being used to the timestamps as a string.

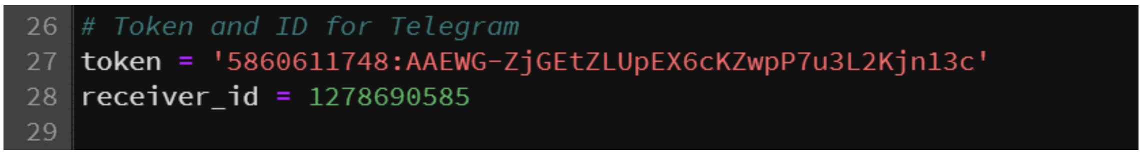


```python
# Defining the variables
motion = [None, None]
time = []
date = []
counter = 0
flag1 = 0
flag2 = 0
time_flag1 = 0
time_flag2 = 0
time_flag3 = ''

```

Fig 2: Screenshot of Code for Initializing Variables

Initializing Telegram API

We would use the Telegram API service to send notifications to the telegram bot we have created whenever we have an instance of motion. However, before sending notifications, we must access our telegram bot's **"token"** and **"receiver_id**."

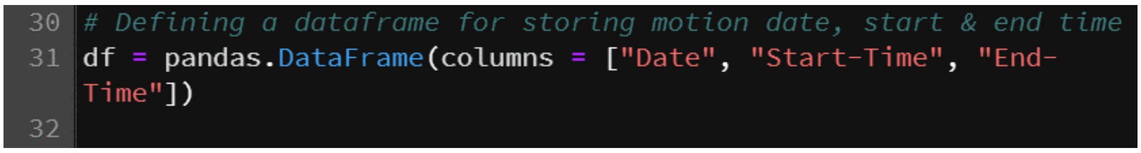


```python
# Token and ID for Telegram
token = '5860611748:AAEWG-ZjGEtZLUpEX6cKZwpP7u3L2Kjn13c'
receiver_id = 1278690585
```

Fig 3: Screenshot of Code for Initializing Telegram API

Initializing Data Frame

We create a data frame **“df”** to save it as a security log file later in the model. It has three columns – **“Date**,**”** **“Start-Time**,**”** and **“End-Time**.**”**



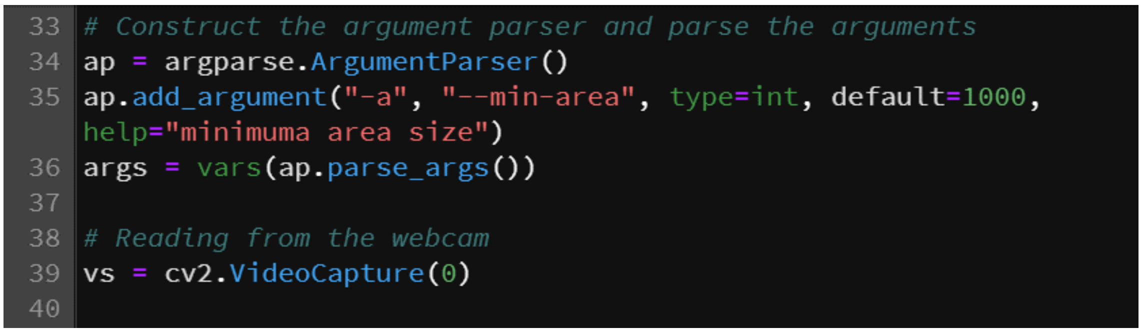
```python
# Defining a dataframe for storing motion date, start & end time
df = pandas.DataFrame(columns = ["Date", "Start-Time", "End-Time"])
```

Fig 4: Screenshot of Code for Initializing Data Frame

Parsing the Arguments

Instead of creating two switches in the model, where one would be used as a path for the video file and the other as a video camera, we directly go for a webcam as the video source. The OpenCV will use the webcam of the laptop to detect motion. We leave the video file as we are going for a live motion detection system.

We will also define **“--min-area**,**”** which is the minimum size (in pixels) for an image region to be deemed “motion.” Frequently, we observe little parts of the frame that have changed significantly, most likely due to noise or altered lighting circumstances. In actuality, these little zones do not represent natural motion. Thus we will establish a minimum region size to find and filter out these false positives.

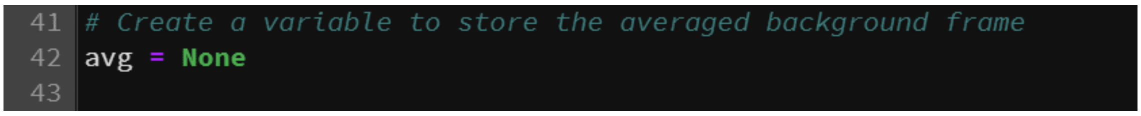


```python
# Construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-a", "--min-area", type=int, default=1000, help="minimuma area size")
args = vars(ap.parse_args())

# Reading from the webcam
vs = cv2.VideoCapture(0)
```

Fig 5: Screenshot of Code for Parsing the Arguments

Initializing the Average Frame

We initialize the variable **“avg”** as None, which would be used later to store the average frame of our webcam feed. The average frame would be a reference frame to account for any changes in the scene.



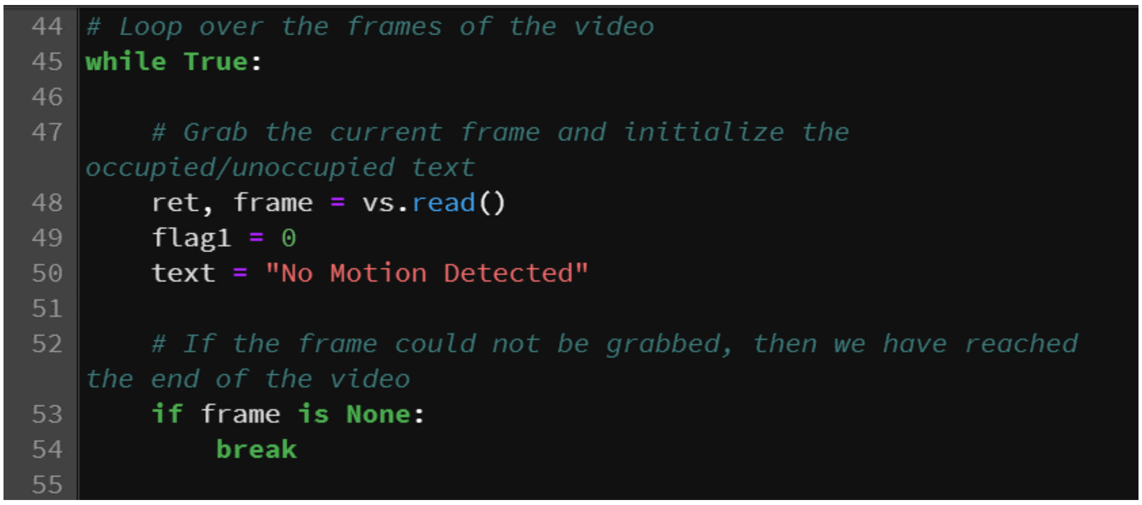
```python
# Create a variable to store the averaged background frame
avg = None
```

Fig 6: Screenshot of Code for Initializing the Average Frame

Grabbing the Frames

We initiate a loop and begin iterating through each frame. The **vs.read()** function returns a webcam frame. Moreover, we exit the loop if a frame cannot be read correctly from the video file.

We initialize the **“flag1”** variable because the first frame is considered motionless. Additionally, we construct a **“text”** string and initialize it to show that the monitored room is **“No Motion Detected**.**”** This string can be updated if there is any activity in the room.



```
# Loop over the frames of the video
while True:

    # Grab the current frame and initialize the
occupied/unoccupied text
    ret, frame = vs.read()
    flag1 = 0
    text = "No Motion Detected"

    # If the frame could not be grabbed, then we have reached
the end of the video
    if frame is None:
        break

```

Fig 7: Screenshot of Code for Grabbing the Frames

Processing the Frames

We begin processing the frames and prepping them for motion assessment. There is no need to handle the vast, unprocessed photos directly from the video feed; we will first resize the image to 1000 pixels wide. Since color does not influence our motion detection technique, we will also convert the color of the frame to grayscale. Finally, we will apply Gaussian blurring to our photos to smooth them.

Due to minute differences in camera sensors, no two frames will be identical in every way; some pixels will undoubtedly have different intensities. We use Gaussian smoothing to average the pixel intensities over a 21×21 rectangle. This helps to level out high-frequency noise that may throw off our system for motion recognition.

We need to simulate the image’s backdrop. The average of the recent frames is calculated to create a reference frame. The reference frame represents the background of the scene without any motion. The reference frame is updated periodically to account for any changes in the scene, such as changes in lighting conditions or objects in the scene. We initialize the average frame as the gray frame but in float format.

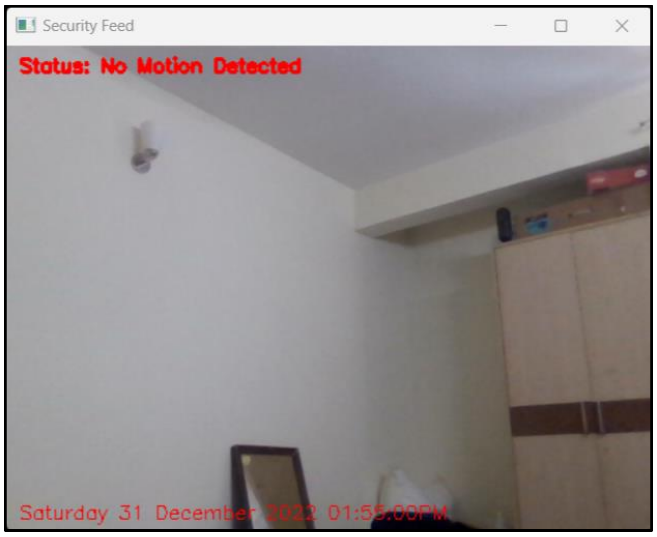
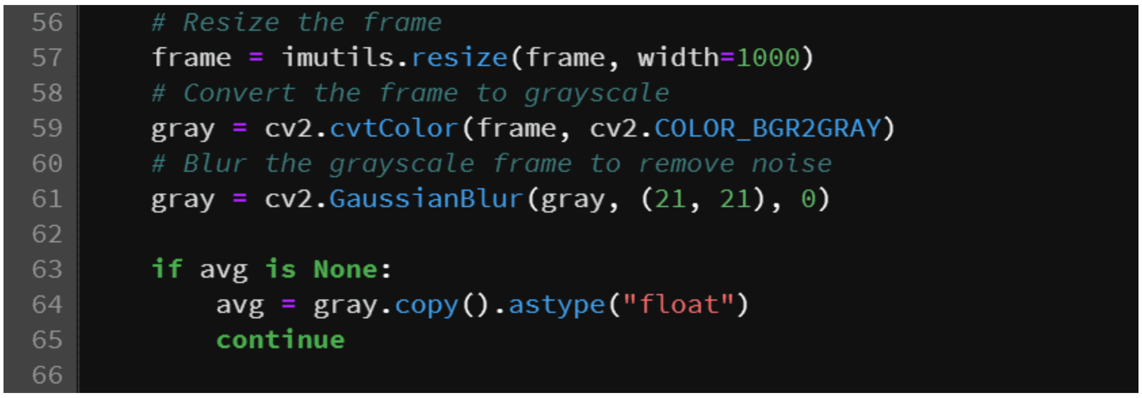

Security Feed
Status: No Motion Detected
Saturday 31 December 2022 01:55:00PM

Fig 8: Screenshot of First Frame



```python
    # Resize the frame
    frame = imutils.resize(frame, width=1000)
    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Blur the grayscale frame to remove noise
    gray = cv2.GaussianBlur(gray, (21, 21), 0)

    if avg is None:
        avg = gray.copy().astype("float")
        continue

```

Fig 9: Screenshot of Code for Processing the Frames

Weighted Average, Absolute Differencing and Thresholding

We must subtract the current frame from the background to detect motion. The background frame can be obtained by averaging out recent frames. We can use the accumulateWeighted() function in OpenCV to perform this averaging. The average is calculated using the current and recent frames allowing the background image to keep updating. The updating rate is determined by the value of Alpha, which represents the weight of the source image. If we select a smaller value for this factor, the running average will be conducted over a more significant number of previous frames. If we set a higher value, the running average will be executed over fewer number of previous frames.

Now that our backdrop has been modeled using the **"avg"** variable, we can use it to calculate the difference between the recent frame and subsequent frames in the video stream. Calculating the difference between two frames is a straightforward subtraction involving the absolute value of the pixel intensity changes.

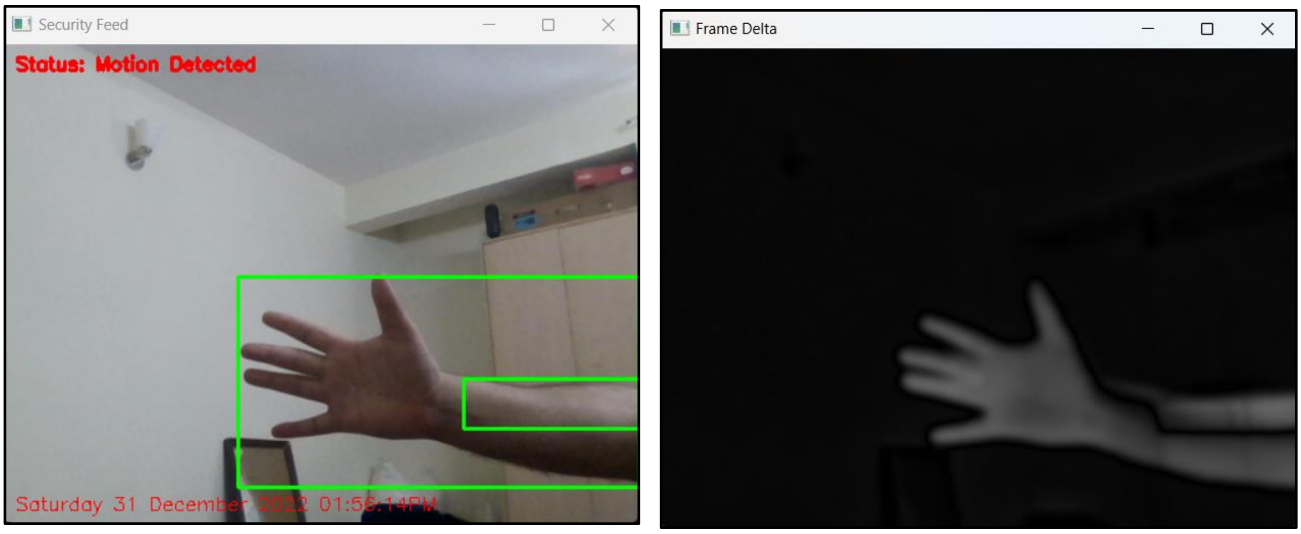


Fig 10: Example of the Frame Delta

Then, we will threshold the **"frameDelta"** to identify parts of the image where only the pixel intensity values have changed significantly. If the frame delta is less than 25, the pixel is discarded and turned to black or the background. If the frame delta is over 25, the background will be white.

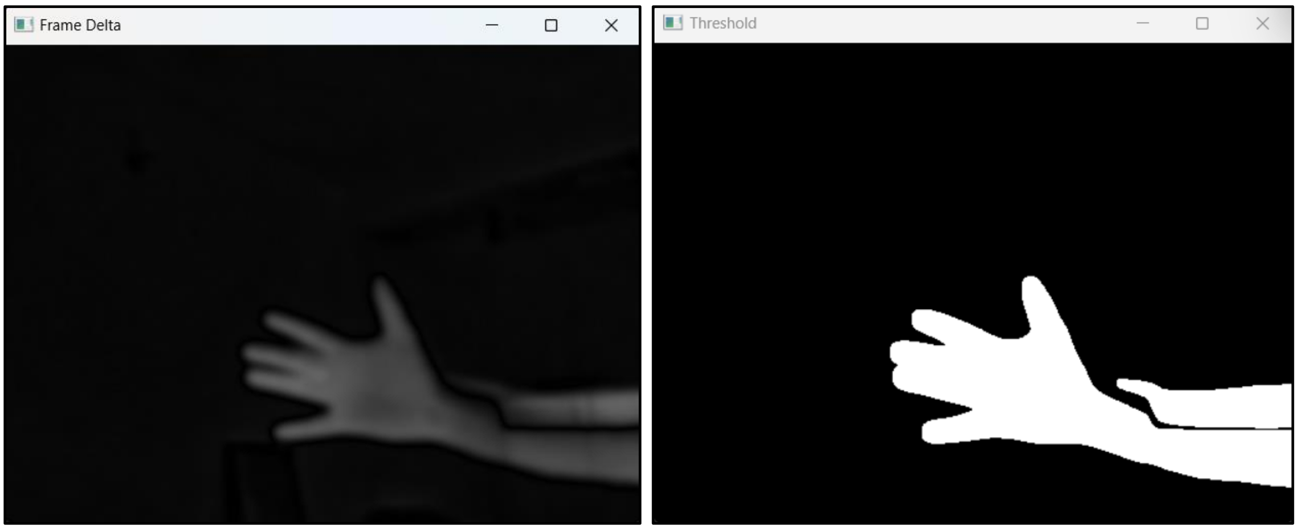
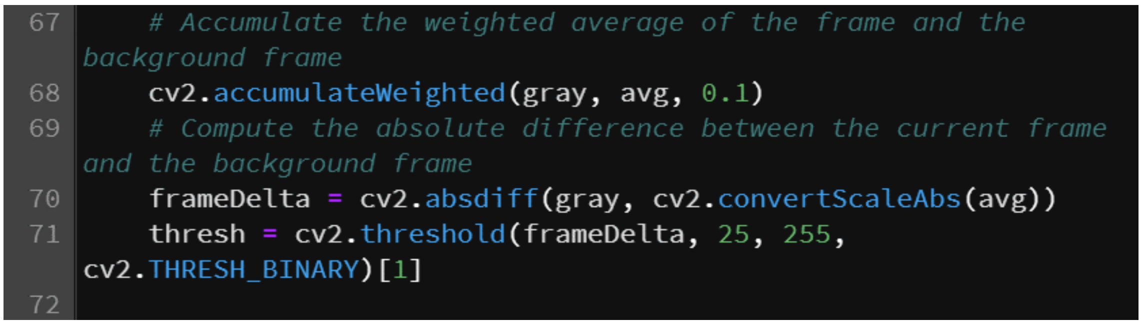


Fig 11: Thresholding the Frame Delta Image

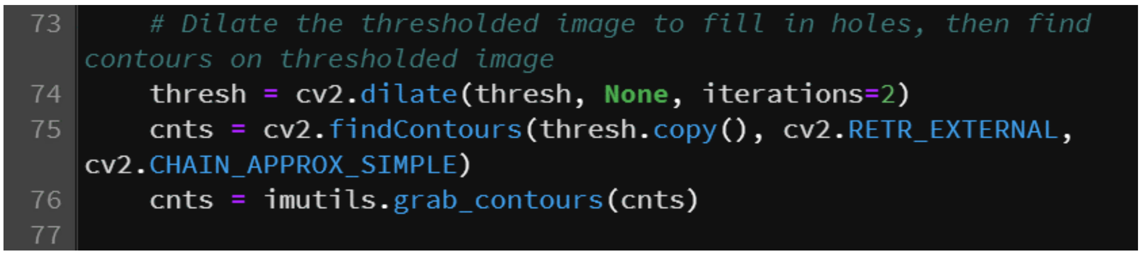


```
    # Accumulate the weighted average of the frame and the background frame
    cv2.accumulateWeighted(gray, avg, 0.1)
    # Compute the absolute difference between the current frame and the background frame
    frameDelta = cv2.absdiff(gray, cv2.convertScaleAbs(avg))
    thresh = cv2.threshold(frameDelta, 25, 255, cv2.THRESH_BINARY)[1]

```

Fig 12: Screenshot of Code for Weighted Average Absolute Differencing and Thresholding

Contour Detection

Given the threshold picture, it is straightforward to use contour detection to identify the boundaries of these white sections. Using contour detection, we can quickly recognize and locate the edges of objects in a picture. It is frequently the initial stage in several fascinating applications, such as image-foreground extraction, simple-image segmentation, detection, and identification.



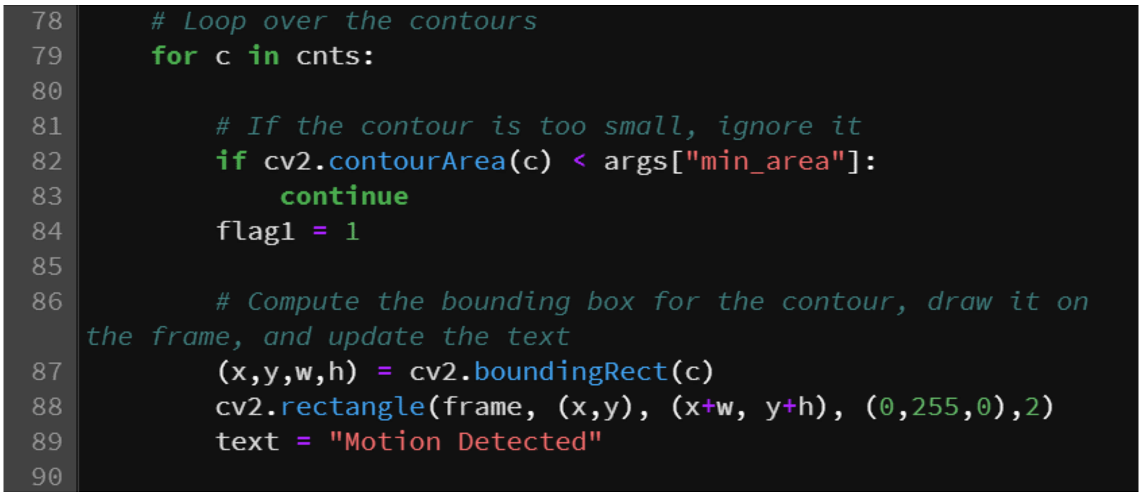
```
    # Dilate the thresholded image to fill in holes, then find contours on thresholded image
    thresh = cv2.dilate(thresh, None, iterations=2)
    cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)

```

Fig 13: Screenshot of Code for Contour Detection

Filtering Irrelevant Contours and Drawing Bounding Boxes

We begin iterating over each contour, and we filter out the little, unimportant contours. We change the status of our **“flag1”** variable from 0 to 1 to signify the motion.

If the contour area exceeds our specified **“--min-area**,” the foreground and motion region’s bounding box will be drawn. We will also change our text status string to **“Motion Detected”** to show there is motion in the room.

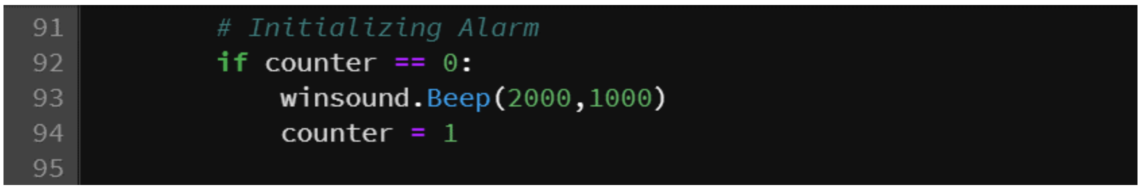


```
    # Loop over the contours
    for c in cnts:

        # If the contour is too small, ignore it
        if cv2.contourArea(c) < args["min_area"]:
            continue
        flag1 = 1

        # Compute the bounding box for the contour, draw it on
the frame, and update the text
        (x,y,w,h) = cv2.boundingRect(c)
        cv2.rectangle(frame, (x,y), (x+w, y+h), (0,255,0),2)
        text = "Motion Detected"

```

Fig 14: Screenshot of Code for Filtering Irrelevant Contours and Drawing Bounding Boxes

Initializing Alarm

We play a buzzer when motion has occurred using the windows platform that is the speaker in the system. We use **winsound.Beep(frequency, duration)** to beep the system's speaker. The frequency parameter sets the sound's frequency in hertz and must fall between 37 and 32,768. The duration parameter sets the duration of the sound in milliseconds. We set the **frequency to 2000 Hertz** and the **duration to 1000 milliseconds**. We change the assigned value of the counter from 0 to 1 to ensure that the buzzer only plays once and not the whole instance of motion.



```
        # Initializing Alarm
        if counter == 0:
            winsound.Beep(2000,1000)
            counter = 1

```

Fig 15: Screenshot of Code for Initializing Alarm

Saving the Motion Image and Sending Notification

We use the **"flag2"** variable to keep the count of the number of instances of motion that occurred. If the instance of motion is one, then we store the timestamp and use **"time_flag5"** to store a string which would be used as the name of the image to be stored in the system and later used to send out the notification to the telegram bot. The name of the image will be in the following format:

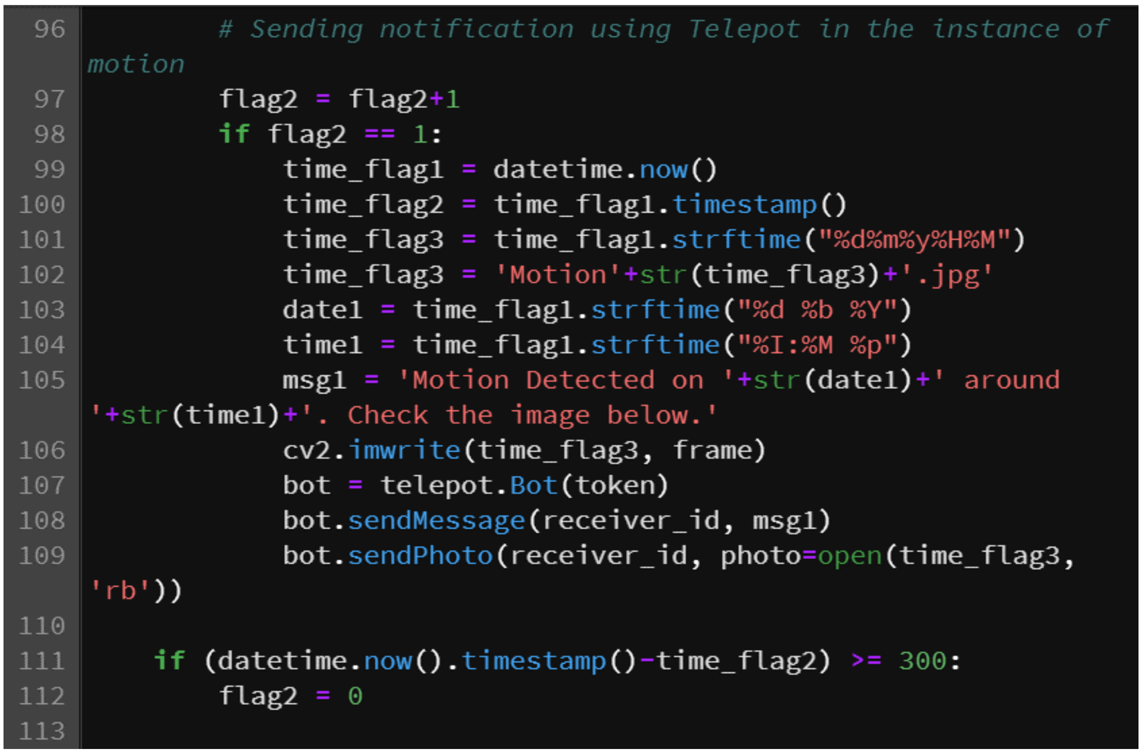
Motion202232206.jpg

In the file name format, 25 is the date, 02 is the month, 23 is the year, 22 is the hour, and 06 is minute. The file is stored in jpg format. We now initialize a notification message in the **"msg1"** variable. The message is structured in the following format.

Motion Detected on "date1" around "time1". Check the image below.

Where **"date1"** and **"time1"** are the variables that store the date and time of the instance of motion. We then store the frame of the instance of motion using the **cv2.imwrite()** function. The image file stored and the message constructed are then sent as a notification to the telegram bot.

We also create a condition where a new notification can only be sent out after 5 minutes or 300 seconds, even if an instance of motion occurs in those 5 minutes. We have done this to prevent to many notifications, which could lead to slowing up of the motion detection system.



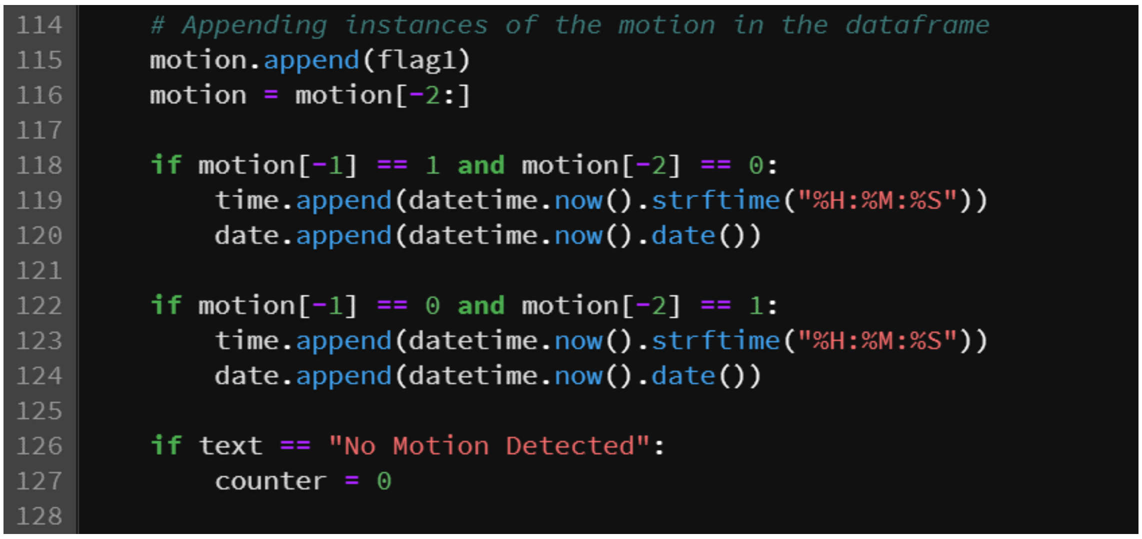
```
        # Sending notification using Telepot in the instance of 
motion
        flag2 = flag2+1
        if flag2 == 1:
            time_flag1 = datetime.now()
            time_flag2 = time_flag1.timestamp()
            time_flag3 = time_flag1.strftime("%d%m%y%H%M")
            time_flag3 = 'Motion'+str(time_flag3)+'.jpg'
            date1 = time_flag1.strftime("%d %b %Y")
            time1 = time_flag1.strftime("%I:%M %p")
            msg1 = 'Motion Detected on '+str(date1)+' around 
'+str(time1)+'. Check the image below.'
            cv2.imwrite(time_flag3, frame)
            bot = telepot.Bot(token)
            bot.sendMessage(receiver_id, msg1)
            bot.sendPhoto(receiver_id, photo=open(time_flag3, 
'rb'))

    if (datetime.now().timestamp()-time_flag2) >= 300:
        flag2 = 0

```

Fig 16: Screenshot of Code for Saving the Motion Image and Sending Notification

Appending Start and End Date & Time for Security Log

We append the motion status in the **"motion"** list, followed by appending the start time and date in the time and date list. We also append the end time and date in the time and date list. We use the **strftime()** function to format the time in the "**H:M:S**" format.

We assign the **"counter"** variable the value 0 if the model's status is no motion to ensure that the buzzer plays the next time there is any motion.

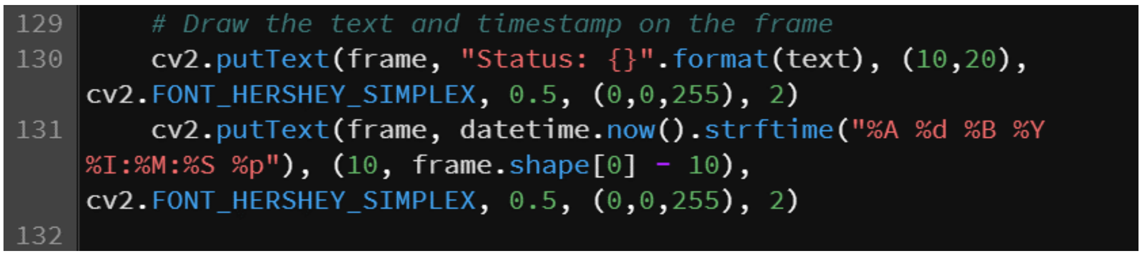


```
    # Appending instances of the motion in the dataframe
    motion.append(flag1)
    motion = motion[-2:]

    if motion[-1] == 1 and motion[-2] == 0:
        time.append(datetime.now().strftime("%H:%M:%S"))
        date.append(datetime.now().date())

    if motion[-1] == 0 and motion[-2] == 1:
        time.append(datetime.now().strftime("%H:%M:%S"))
        date.append(datetime.now().date())

    if text == "No Motion Detected":
        counter = 0

```

Fig 17: Screenshot of Code for Appending Start and End Date & Time for Security Log

Displaying the Status and Time on the Security Feed

We draw the room status, **"Motion Detected"** or **"No Motion Detected,"** in the upper-left corner of the picture, followed by a timestamp in the lower-left corner. The timestamp is in **"Saturday 25 February 2023 22:25:30 PM"** format.

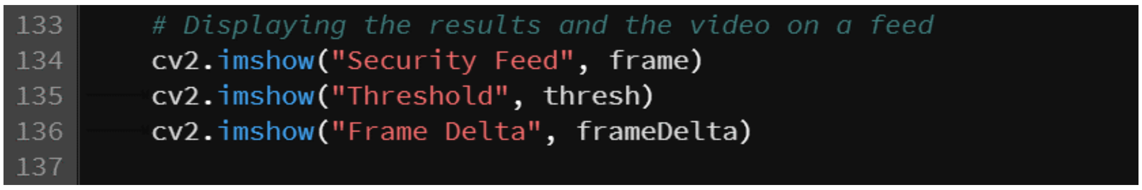


```
    # Draw the text and timestamp on the frame
    cv2.putText(frame, "Status: {}".format(text), (10,20),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 2)
    cv2.putText(frame, datetime.now().strftime("%A %d %B %Y
%I:%M:%S %p"), (10, frame.shape[0] - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 2)

```

Fig 18: Screenshot of Code for Displaying the Status and Time on the Security Feed

Displaying the Results

Displaying the results of our effort, enabling us to see if a motion was identified in our video, as well as the **frame delta and threshold picture** for debugging purposes.

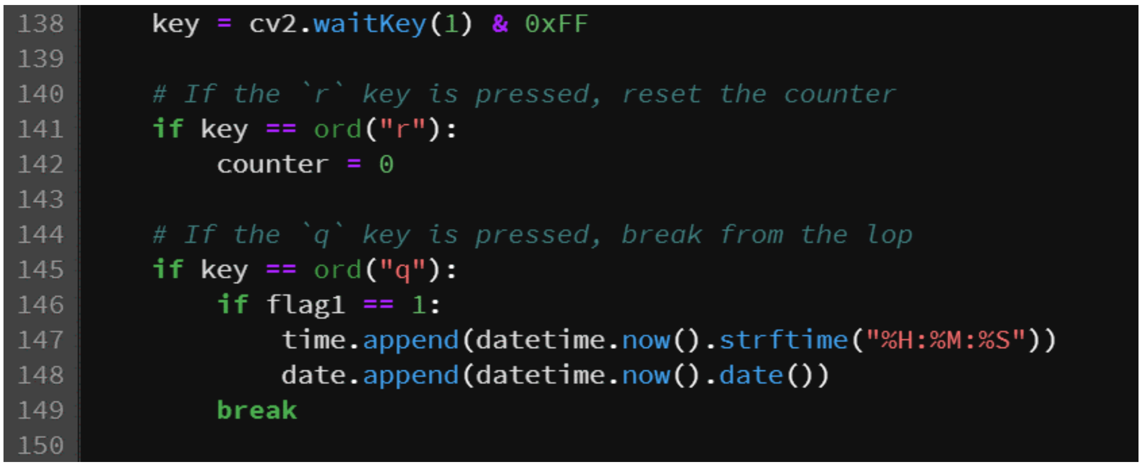


```
    # Displaying the results and the video on a feed
    cv2.imshow("Security Feed", frame)
    cv2.imshow("Threshold", thresh)
    cv2.imshow("Frame Delta", frameDelta)

```

Fig 19: Screenshot of Code for Displaying the Results

Initializing Keys

We initialize a variable **"key"** to be used as input by the user to perform specific tasks at the stroke of the particular keys. **Key "r"** is used to reinitialize the buzzer**,** and **key "q"** is used to stop the execution of the model. Also, if the status of the room is motion, then the press of **key "q"** will also append the current time and date to their respective lists as the end time and date for the particular instance of motion.



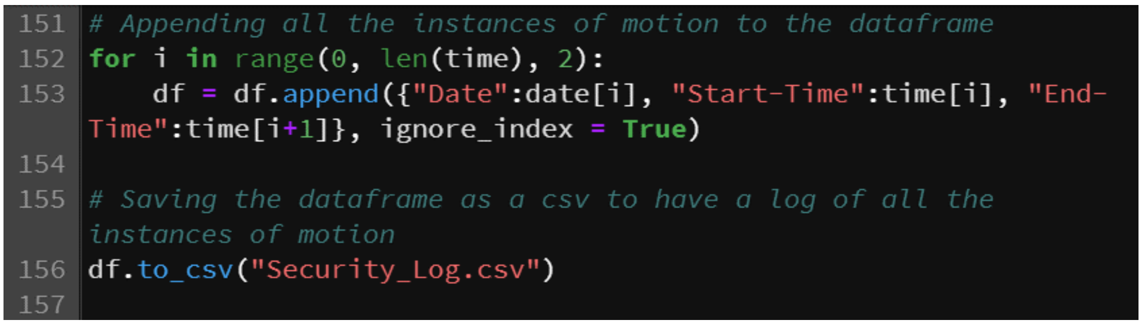
```
    key = cv2.waitKey(1) & 0xFF

    # If the `r` key is pressed, reset the counter
    if key == ord("r"):
        counter = 0

    # If the `q` key is pressed, break from the lop
    if key == ord("q"):
        if flag1 == 1:
            time.append(datetime.now().strftime("%H:%M:%S"))
            date.append(datetime.now().date())
        break

```

Fig 20: Screenshot of Code for Initializing Keys

Saving the Security Log as a CSV File

We start a loop to store the values from the date and time list in the dataframe. We assign the value at **date[i]** to the date column, **time[i]** to the Start-Time column, and **time[i+1]** to the End-Time column.

We also **convert the dataframe to a CSV file** and save it in the current folder, the same as the file where our python model is saved. The name of the CSV file would be **"Security_Log.csv**."

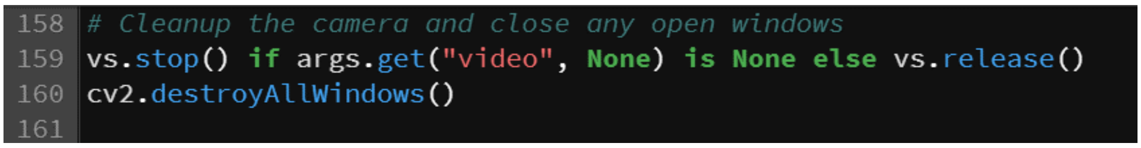


```python
# Appending all the instances of motion to the dataframe
for i in range(0, len(time), 2):
    df = df.append({"Date":date[i], "Start-Time":time[i], "End-Time":time[i+1]}, ignore_index = True)

# Saving the dataframe as a csv to have a log of all the instances of motion
df.to_csv("Security_Log.csv")
```

Fig 21: Screenshot of Code for Saving the Security Log as a CSV File

Cleanup and Release of the Video Stream Pointer

Finally, we clean up and release our video stream pointer. The Python OpenCV **destroyAllWindows()** method enables script-exiting users to delete or shut all windows at any moment.



```python
# Cleanup the camera and close any open windows
vs.stop() if args.get("video", None) is None else vs.release()
cv2.destroyAllWindows()
```

Fig 22: Screenshot of Code for Cleanup and Release of the Video Stream Pointer

Camera Placement

The placement of the camera within the room plays a significant role in determining how well the model will work. The people in the room make up the image's Region of Interest (ROI). Calibration of the camera's perspective helps simplify the process of object detection. OpenCV's most straightforward camera calibration requires selecting four points in the perspective view and translating them into a two-dimensional rectangle's corners. This allows the camera to detect activity better and monitor its surroundings by allowing it to watch everyone in the room. Therefore, the upper corner of the room is the ideal spot for placing the camera within a room since, from this vantage point; the camera will be able to see all of the entrances to the room and every other location within the room.

In general, placing the camera at a high angle is recommended, pointing downwards towards the area of interest. This provides a broader field of view and reduces the likelihood of obstructions in the camera's line of sight. Additionally, it is essential to consider the lighting conditions in the room and adjust the camera settings accordingly, such as exposure and white balance. Finally, ensuring the camera is securely mounted and protected from damage or tampering is vital. This may involve using a mounting bracket or enclosure or positioning the camera out of reach or sight of potential intruders.

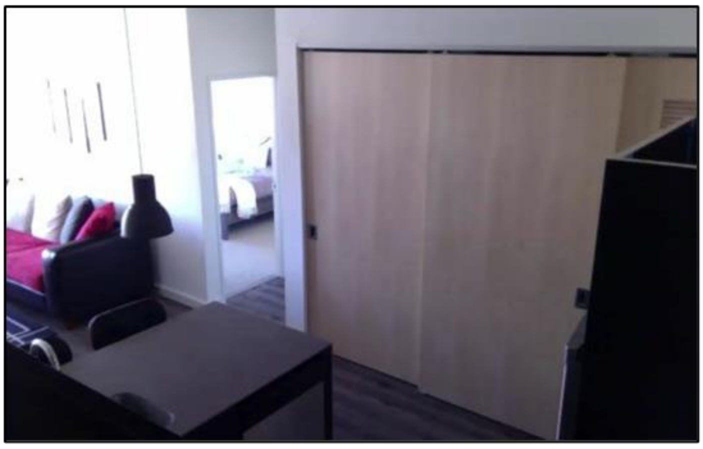


Fig 23: Example of Camera Placement

Result

Motion detection in real-time may be accomplished through a camera and OpenCV in a straightforward and very effective way. Motion detection with a camera and OpenCV works by subtracting the most recent frame from all the frames that came before it. This is the fundamental idea behind the method. By subtracting the most recent frame from all the frames before it, one may determine the difference between them, a sign that motion is occurring.

When attempting to detect motion using a camera and OpenCV, one of the most important steps is to ensure that alerts are sent out immediately upon the first instance of motion and after 10 minutes if more motion has occurred. The user may then respond appropriately after being notified of any motion in the scene, which is made possible by the notifications.

Using a camera with OpenCV to detect motion involves critical steps, including generating a CSV file to record motion activity. Recording motion activity with a CSV file is a straightforward and efficient method. The CSV file includes the date, as well as the beginning and ending times of the move. The CSV file has a variety of applications, including report generation and the analysis of motion activity, to name just a couple of them.

As a consequence of the model, we can observe, as depicted in **Figure 24**, that the model can **determine whether or not there is motion in the room**. Additionally, the model produces a beeping buzzer sound at each incidence of the move.

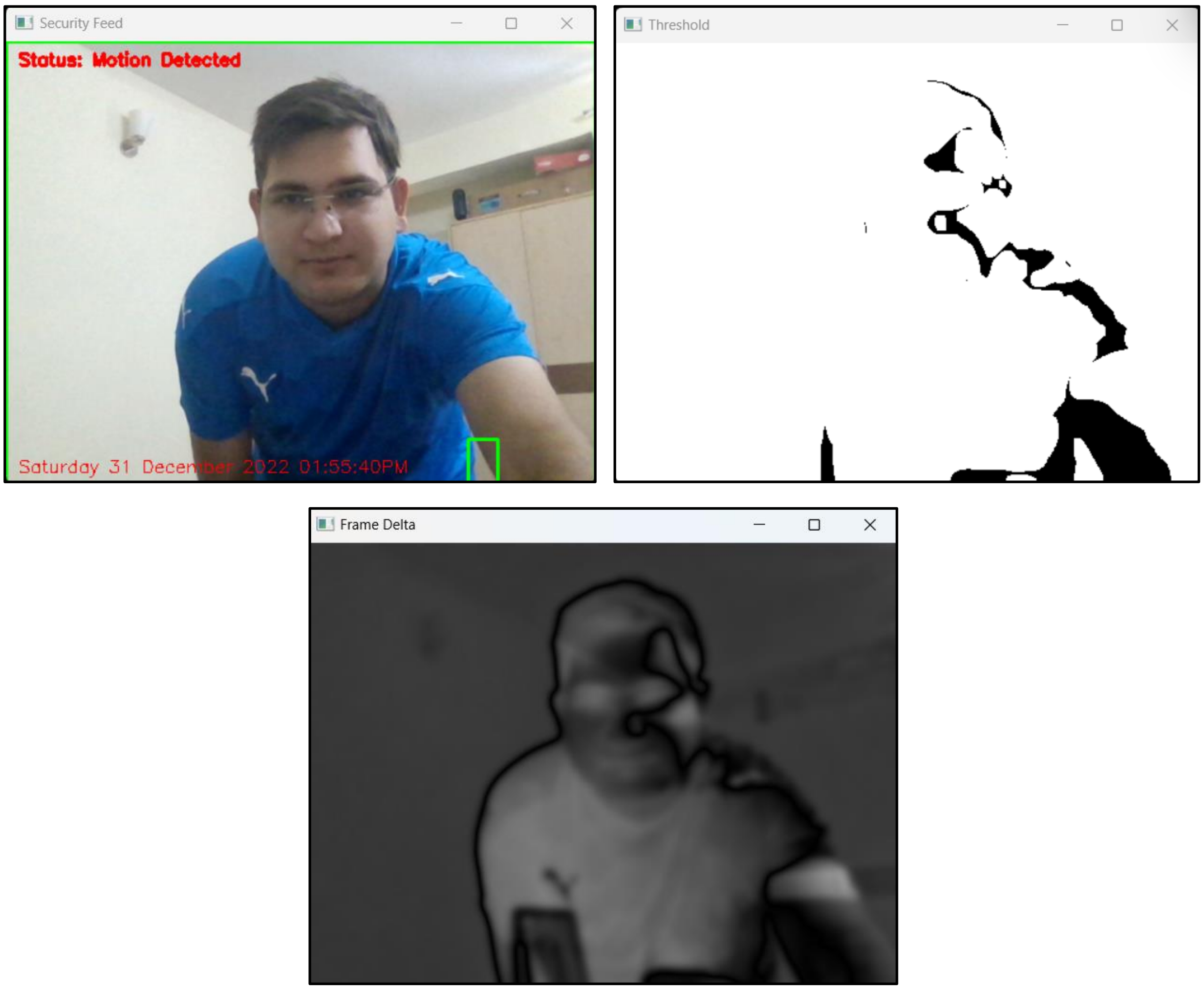
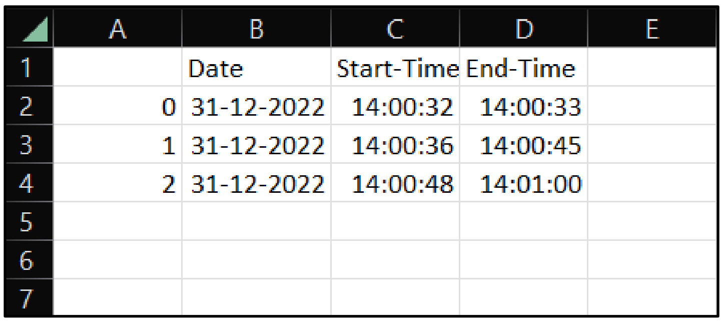


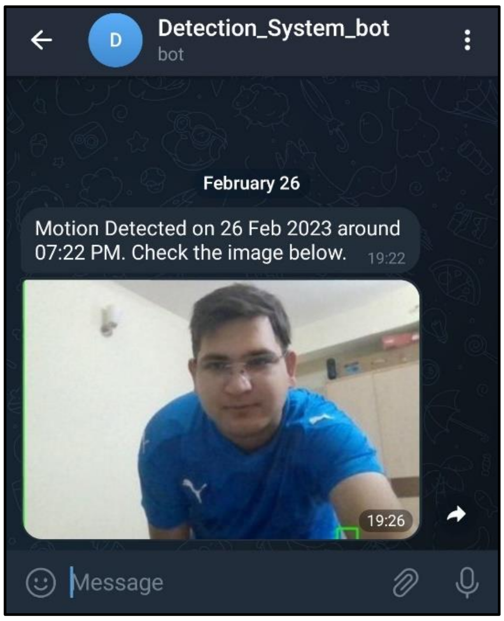
Fig 24: Screenshot of the Final Result of the Model

After the execution of the model has been finished, there will be a **CSV file with the name** “**Security_Log**” located in the same directory as our Python module. As can be seen **in Figure 25**, the CSV file **possesses the dates, as well as the start and end times**, of the motion instances.



	A	B	C	D	E
1		Date	Start-Time	End-Time	
2	0	31-12-2022	14:00:32	14:00:33	
3	1	31-12-2022	14:00:36	14:00:45	
4	2	31-12-2022	14:00:48	14:01:00	
5					
6					
7					

Fig 25: Screenshot of the Security Log CSV File



Detection_System_bot
bot

February 26

Motion Detected on 26 Feb 2023 around 07:22 PM. Check the image below. 19:22

19:26

Fig 26: Screenshot of the Telegram Notification

Motion detection may be utilized in a diverse set of contexts, including but not limited to security systems, surveillance video, and traffic monitoring. We can produce a low-cost surveillance system that is simple to install and can be utilized in various contexts if we use motion detection by integrating a camera and OpenCV into our workflow.

A simple application of motion detection utilizing OpenCV and a camera is provided as an example in this report’s accompanying code. This code may be enhanced and expanded in several different ways. For instance, we could make motion detection more accurate by utilizing machine learning methods, or we could add the option to capture movies in addition to still photographs. Both of these improvements would be welcome additions.

The code might also be improved by making it more performance-friendly, which is another option. We are currently processing each frame in memory, which may provide a challenge for computer systems with a finite amount of memory. We can store only the required frames by utilizing a circular buffer.

In general, motion detection through a camera and OpenCV is a method that is not only straightforward but also very accurate. It can be used in a wide variety of applications. We can generate a surveillance system that is not only low-cost but also simple to personalize and set up if we put the code that is detailed in this paper into action.

Way Forward

Using a camera with OpenCV to detect motion more accurately may be accomplished in the following ways moving forward:

- **Advanced Motion Detection Algorithms**
  Motion detection with a camera and OpenCV works by subtracting the most recent frame from all the frames that came before it. This is the fundamental idea behind the method. While it is true that this theory works, it also has the potential to produce false positives and false negatives. The accuracy of motion detection can be improved by the development of more sophisticated motion detection algorithms. Techniques like background removal, optical flow, and machine learning are a few examples of what these algorithms may employ.
- **Use of Multiple Cameras**
  When many cameras are used, a complete scene picture can be obtained, and the precision with which motion is detected may also be improved. It is possible to record the scene from a variety of views using many cameras that have been positioned at various angles.
- **Integration with Other Systems**
  Motion detection with a camera and OpenCV may be incorporated into various systems, including security alarms, access control, and lighting systems. Integrating with other systems can increase the overall security of the location and give a more effective reaction to motion occurrences. Integration can also make the environment more comfortable for occupants.
- **Improved Notification System**
  The existing notification system sends alerts when there is a motion for the first time and then again after 10 minutes if there is further motion. However, this system has the potential to be enhanced so that it can supply more information, such as the number of persons who have been identified, the direction in which motion is occurring, and the total amount of time motion has been detected.
- **Use of Cloud-Based Systems**
  Cloud-based systems can offer more significant storage space for motion activity data and enable remote access to the data. Cloud-based systems can also give more storage space for other types of data. Cloud-based solutions are also capable of providing more sophisticated reporting and analytics capabilities.
- **Use of High-Resolution Cameras**
  A higher-resolution camera can provide images with a higher level of detail and increase motion detection accuracy. It can catch more information, including facial features, car license plates, and other identifying characteristics.
- **Integration with Other Sensors**
  Motion detection using a camera and OpenCV may be combined with other sensors, such as temperature and humidity sensors. Motion detection can also be incorporated using additional sensors. Integration with additional sensors has the potential to offer more detailed information about the surrounding environment and increase the reliability of motion detection.

In conclusion, motion detection using a camera and OpenCV is a helpful tool that can be applied to various applications. Using a camera and OpenCV, motion detection may be made more accurate and efficient by applying the strategies outlined in this study. In many different types of businesses, the ongoing development of improved motion detection systems may enhance security, safety, and productivity.

Fire Detection Surveillance System

Introduction

Detecting fire is crucial in various applications, including fire safety, surveillance, and industrial control. Sensors that detect heat or smoke are traditionally utilized in conventional fire detection systems. However, there is a possibility that these systems may not be able to effectively identify fires in their early phases, which may result in significant property damage as well as the loss of lives.

Using computer vision techniques with an OpenCV library and a camera might solve this issue. These approaches would allow for the detection of flames in real-time. OpenCV is a computer vision package that is freely available to the public and offers various image processing and computer vision methods. Because of its flexibility, dependability, and user-friendliness, it has quickly emerged as a leading candidate for developing computer vision applications.

As part of this endeavor, we will create a system for detecting fires utilizing OpenCV and a HAAR Cascade file that has already been deployed. The HAAR Cascade file is a pre-trained classifier that can identify characteristics present in digital photographs or motion pictures. We will use the HAAR Cascade file to recognize the telltale signs of a fire in the video collected in real-time by a camera.

If there is a fire, the system will act as a detector and communicate this information to a telegram bot. The message will also include an image of the blaze as obtained by the camera and the time and date of the fire. Moreover, the system will continue to monitor the region and send additional warnings if it finds any new fire occurrences after 10 minutes.

In general, this project aims to demonstrate the usefulness of computer vision techniques in detecting fires and the potential for such systems to enhance the performance of fire safety and surveillance applications.

Literature Review

Fire detection is a critical task that has received significant attention in recent years due to its importance in ensuring public safety and preventing property damage. Many traditional fire detection systems use sensors that detect heat or smoke, but these systems may not be effective in detecting fires in their early stages. Computer vision techniques have emerged as a promising solution to this problem, as they can detect fires in real-time using cameras and image processing algorithms.

In this literature review, we will discuss the current state-of-the-art fire detection using computer vision techniques, focusing on using OpenCV and HAAR Cascade classifiers.

OpenCV

OpenCV is a freely available computer vision package providing various image processing and algorithms. It has become famous for developing computer vision applications due to its versatility, reliability, and ease of use. OpenCV provides a variety of functions for processing images, including image segmentation, edge detection, and object recognition. These functions can identify and track objects in real-time video streams.

HAAR Cascade classifiers

A HAAR Cascade classifier is a pre-trained classifier that detects features in images or videos. It uses a set of Haar-like features to identify regions of an image that are likely to contain the object of interest. These features are simple rectangular patterns that can be used to identify edges, lines, and corners in an image. The classifier then combines these features to determine the presence of the object in the image. HAAR Cascade classifiers have been used in various applications, including face detection, object recognition, and pedestrian detection.

Fire detection using computer vision

Recent computer vision research has detected fires. These experiments mainly employed cameras to detect flames in real-time video feeds. Flames, smoke, and heat are the biggest challenges in fire detection. Computer vision technologies for fire detection include color-based, texture-based, and motion-based approaches.

Color-based methods employ flames' wavelengths to differentiate them from other light sources. These approaches detect flames using color thresholds. Color-based approaches may falsely alert due to illumination changes.

Texture-based methods make use of flame's unique texture. These approaches employ Gabor filters or wavelet transformations to find fire in images. Texture-based approaches are computationally demanding but more resilient than color-based methods.

Motion-based techniques, such as flashing flames or smoke ascending, exploit fire's visual motion. These approaches employ optical flow or background removal to find fire in images. Motion-based fire detection may falsely warn if other moving items are present.

Fire detection using HAAR Cascade classifiers

Several studies have used HAAR Cascade classifiers for fire detection in real-time video streams. These studies have focused on detecting flames or smoke using the characteristic features of fire. Most of these studies have used color-based approaches to detect flames, such as using the red or yellow color of flames to identify them in the image.

Fire Detection Using Video Analysis

Video analysis is a famous fire detection method. Video analysis can identify fire smoke, flames, and heat patterns. Software analyses video data from cameras to identify fires. Video analysis techniques for fire detection might cause false alarms. Video analysis tools may misinterpret movement or illumination changes for smoke or fire. Algorithms scan video footage to distinguish genuine fires from false alarms.

Video analysis for fire detection has numerous methods. Infrared cameras detect fire heat, whereas visible light cameras detect smoke and flames. Some systems use infrared and visible light cameras for better fire detection. It allows real-time surveillance of enormous areas. This is important in warehouses, factories, and other big buildings where flames spread fast and inflict severe damage. Video analysis systems may be linked with smoke detectors and sprinkler systems for a complete fire detection and suppression solution.

Fire Detection Using Machine Learning

Machine learning teaches computers to learn without being programmed. Fire patterns may be detected using machine learning systems. It can distinguish distinct types of flames based on their visual and thermal properties. Machine learning algorithms can distinguish electrical fires from arson fires.

Machine learning algorithms can detect fire trends in sensor and monitoring system data. Machine learning systems can detect fire trends in the smoke detector, temperature, and other sensor data. Machine learning for fire detection requires lots of training data. Machine learning techniques need massive datasets when real-world fire data is scarce in fire detection.

Fire Detection Using Deep Learning

Deep learning uses data to train neural networks. Deep learning systems can find fire-related data patterns. Deep learning for fire detection may be trained to recognize complicated data patterns, including invisible ones. Deep learning systems can identify fire heat patterns even without flames or smoke.

Deep learning algorithms can detect fire trends in sensor and monitoring system data. Deep learning systems can predict fire trends in the smoke detector, temperature, and other sensor data. Deep learning fire detection requires many training data. Deep learning systems need big datasets when real-world fire data is scarce in fire detection.

Fire Detection using Thermal Imaging

Thermal imaging has been used for fire detection for many years. In this method, thermal cameras are used to capture images of the environment, and then the images are analyzed to detect temperature anomalies that may indicate the presence of fire. Thermal cameras are sensitive to infrared radiation and can detect temperature differences as slight as 0.1°C. This makes them ideal for detecting fires in their early stages.

In a study by Wang et al. (2019), a camera and OpenCV-based fire detection system was constructed, and a machine learning model was employed to increase fire detection accuracy. The system identified a 97.3% accuracy rate and a 2.6% false alert rate.

In research by Prasetyo et al. (2020), a camera and OpenCV-based fire detection system was created, and its performance was examined under different illumination circumstances. The study indicated that lighting circumstances influenced the system’s performance, with the system’s accuracy decreasing under low-light situations.

In a paper by Chen et al. (2018), a webcam and OpenCV-based fire detection system was constructed, and its performance was compared to that of a typical smoke sensor-based fire detection system. The study determined that the camera and OpenCV-based system could identify fires more quickly than the conventional smoke sensor-based system.

In work by Zhou et al. (2019), a camera and OpenCV-based fire detection system was constructed, and a deep learning-based algorithm was employed to increase fire detection accuracy. The study indicated that the method based on deep learning was capable of 99.7% detection accuracy.

Overall, these studies illustrate the potential of webcam and OpenCV for fire detection, and including machine learning, and deep learning approaches can increase the system’s accuracy and dependability. Nevertheless, environmental factors like lighting conditions, smoke, or other airborne particles may impact the system’s effectiveness. Consequently, further study is required to solve these obstacles and enhance the system’s performance in real-world environments.

Methodology

The project captures the video stream using the system’s camera. With OpenCV, the video frames are processed. The idea of fire detection is based on using the HAAR Cascade file to detect fire. The following steps outline the process:

1. **Hardware Setup**
   A webcam is connected to a computer. The webcam should be positioned to have an unobstructed view of the monitored area. Ensuring that the webcam is of sufficient quality and has an appropriate frame rate for the application is vital.
2. **Software Installation**
   The OpenCV and other required libraries are installed on the computer. This may involve downloading the necessary software packages and running the installation commands.
3. **Preparing the HAAR Cascade file**
   A pre-installed HAAR Cascade file is used for fire detection. The HAAR Cascade file is a set of classifiers that can be trained to detect specific features in an image. The pre-installed file is trained to detect fire. However, if needed, a new HAAR Cascade file can be trained using machine learning techniques and labeled datasets.

4. **Image Capture and Processing**
   The webcam captures images from the monitored area, which are then processed using OpenCV. The images are converted to grayscale and then filtered to remove noise. The HAAR Cascade file is then applied to the images to detect fire. If a fire is detected, the coordinates of the detected object are obtained, and a bounding box is drawn around the object.
5. **Notification System**
   A telegram bot is created, which sends a notification when a fire is detected. The bot sends a message with the date and time of the fire along with an image showing the location of the fire. The notification system can be configured to send a notification on the first instant of fire and then send subsequent notifications every 5 minutes if a new instance of fire is detected.
6. **Testing and Evaluation**
   The system is tested to ensure that it performs as expected. The system is tested under different lighting conditions and with smoke or other particles in the air to ensure that it can detect fire accurately. The system's accuracy is evaluated by comparing the system's output with actual fires.
7. **Improvement of System**
   The system is improved to enhance its performance based on the evaluation results. Improvements may include adjusting the camera position, optimizing the HAAR Cascade file, or refining the notification system.
8. **Deployment**
   Once the system is deemed reliable and accurate, it can be deployed in the area to be monitored. The system should be continuously monitored to ensure it functions as expected. Any issues that arise should be addressed promptly to maintain the system's reliability.

In summary, the methodology involves setting up the hardware and software, preparing the HAAR Cascade file, capturing and processing images, implementing the notification system, testing and evaluating the system, improving the system, and deploying the final system.

Hardware and Software Requirements

The following hardware and software specifications are required for implementing motion detection using a webcam and OpenCV:

Hardware

- Webcam
- Computer

Software

- Python 3
- OpenCV 4
- Telegram Bot API
- Pandas Library

Implementation

To facilitate comprehension of the model's operation, we have broken it down into a series of phases outlined and coded below; the source code may be found in **Exhibit B**.

Setting up the Environment

The installation of the necessary libraries is an essential part of getting the environment ready for working with OpenCV; nevertheless, this step needs to be finished before we can go on to the next one. Python 3 is going to be the programming language that we use for this particular project. Installing OpenCV successfully requires using the pip package manager, which can be found in Python.

pip install opencv-python

In addition, we will use the Telegram Application Programming Interface (API) to deliver notifications to a Telegram bot. In order to use the Telegram application programming interface, you will need to install the telepot library (API).

pip install telepot

Before beginning the motion detection process, we must ensure our webcam is configured correctly. Establishing a connection between the computer and the camera is necessary before installing the necessary drivers. Launching a program specifically designed to operate with webcams allows us to validate the functionality of our device. If the webcam functions normally, we may go on to the following stages.

OpenCV is a library for computer vision that offers tools for processing images and other computer vision-related activities. It is a library with open-source code that can be downloaded from the official website anytime. We must obtain the most recent version of OpenCV and then install it on our workstation. With the pip package manager, we can successfully install OpenCV.

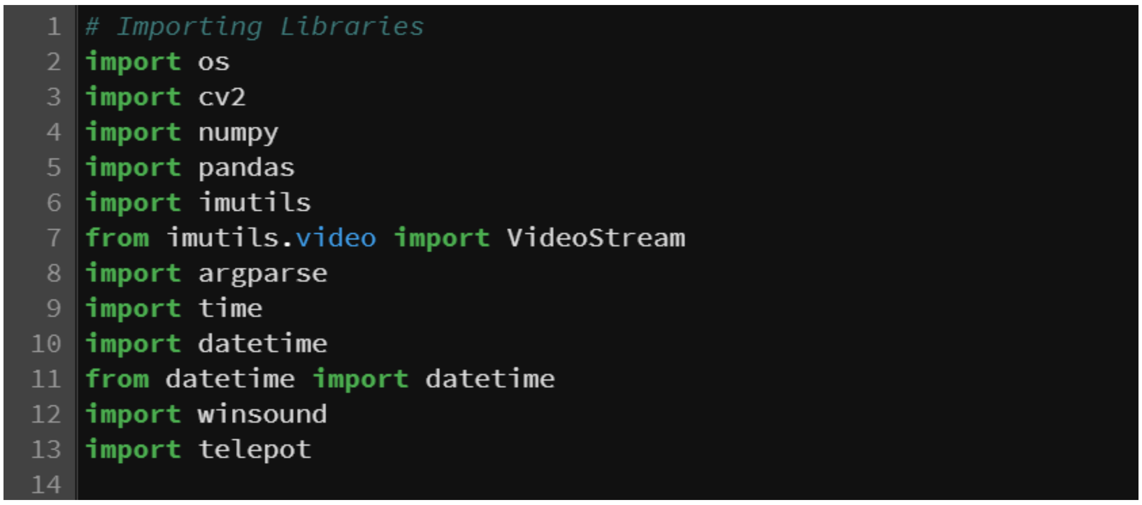
pip install opencv-python

Importing Libraries

Importing the required packages and libraries is the initial step in writing Python code. The following libraries were utilized in the development of our code:

- **os**
  Python's OS library provides a number of different interfaces for interacting with operating systems. Python's basic utility modules include OS. This module provides an adaptive technique for using capabilities, with the specifics of the machine dependent on the operating system. It has various functionalities allowing users to communicate with the file system.
- **cv2**
  OpenCV is a Python module that supports various computer vision and image processing functions.
- **NumPy**
  A Python package called NumPy is used to manipulate arrays. The Fourier transform, matrices, and procedures for linear algebra are also included.
- **pandas**
  A Python library called pandas is used to manipulate data sets. It offers features for data analysis, purification, exploration, and modification.
- **imutils**
  A set of convenience functions for OpenCV to ease the essential image processing operations such as translations, rotations, scaling, skeletonization, displaying Matplotlib images, sorting contours, and recognizing edges. These methods may be found in convenience collection.
- **argparse**
  The argparse makes creating command-line interfaces that are pleasant to users simple. The application defines the arguments, and argparse pulls them out of sys.argv. Argparse generates automatic help and use messages. Improper program parameters also cause module problems.
- **time**
  This library offers a variety of functions that are connected to time.

- **datetime**
  Date and time manipulation classes are accessible in the datetime module. Although the date and time arithmetic is accessible, the implementation concentrates on attribute extraction for effective output formatting and processing.
- **winsound**
  The winsound package provides access to the core sound-playing infrastructure that is included with the Windows operating system. It has constants and functions.
- **telepot**
  Telepot assists in the development of Telegram Bot API apps. It is being utilized to transmit alerts to Telegram at the moment.

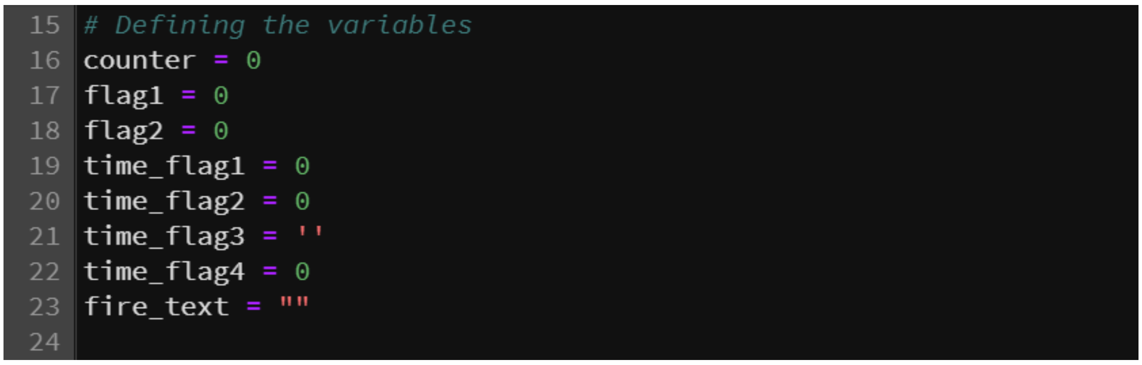


```python
# Importing Libraries
import os
import cv2
import numpy
import pandas
import imutils
from imutils.video import VideoStream
import argparse
import time
import datetime
from datetime import datetime
import winsound
import telepot

```

Fig 27: Screenshot of Code for Importing Libraries

Initializing Variables

The model uses the "counter" variable as an alert system flag. The **"flag1"** variable counts the number of fire instances that would subsequently be used to send notifications, and "flag2" controls the display of fire warnings and reinitializes the alarm. The **"time_flag1"**, **"time_flag2"**, and **"time_flag4"** are used to hold instances of time, whereas **"time_flag3"** save time stamps as strings. The **"fire_text"** string variable will display the fire warning text on the security feed.

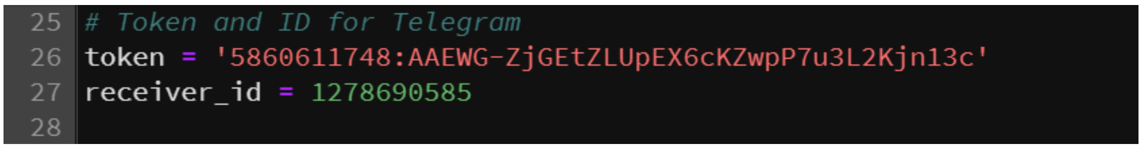


```python
# Defining the variables
counter = 0
flag1 = 0
flag2 = 0
time_flag1 = 0
time_flag2 = 0
time_flag3 = ''
time_flag4 = 0
fire_text = ""

```

Fig 28: Screenshot of Code for Initializing Variables

Initializing Telegram API

When we detect fire, we will utilize the API function provided by Telegram to send notifications to the Telegram bot we have developed. However, we must first access the **"token"** and **"receiver_id"** associated with our Telegram bot to send alerts.

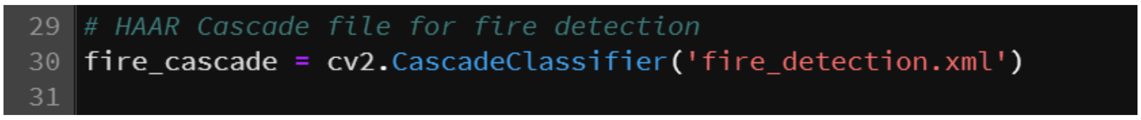


```python
# Token and ID for Telegram
token = '5860611748:AAEWG-ZjGEtZLUpEX6cKZwpP7u3L2Kjn13c'
receiver_id = 1278690585
```

Fig 29: Screenshot of Code for Initializing Telegram API

HAAR Cascade File

Haar Cascades is a machine-learning algorithm used for object detection. The algorithm uses a set of Haar-like features to detect objects. Haar-like features are rectangular regions with different intensity values. The algorithm uses these features to create a classifier to distinguish between the object and the background. We import the pre-trained HAAR Cascade file to detect fires. The HAAR Cascade file is **"fire_detection.xml"** and stored in the **"fire_cascade"** variable.

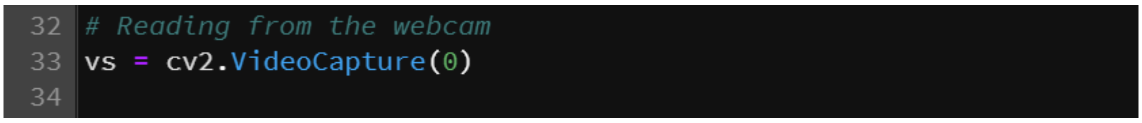


```python
# HAAR Cascade file for fire detection
fire_cascade = cv2.CascadeClassifier('fire_detection.xml')
```

Fig 30: Screenshot of Code for HAAR Cascade File

Reading from the Webcam

We choose to utilize a webcam as the video source rather than constructing two switches in the model, one of which would be used as a route for the video file and the other as a video camera. OpenCV will use the laptop's camera to detect fire. We are looking for a live motion detection system. Thus we leave the video clip alone.



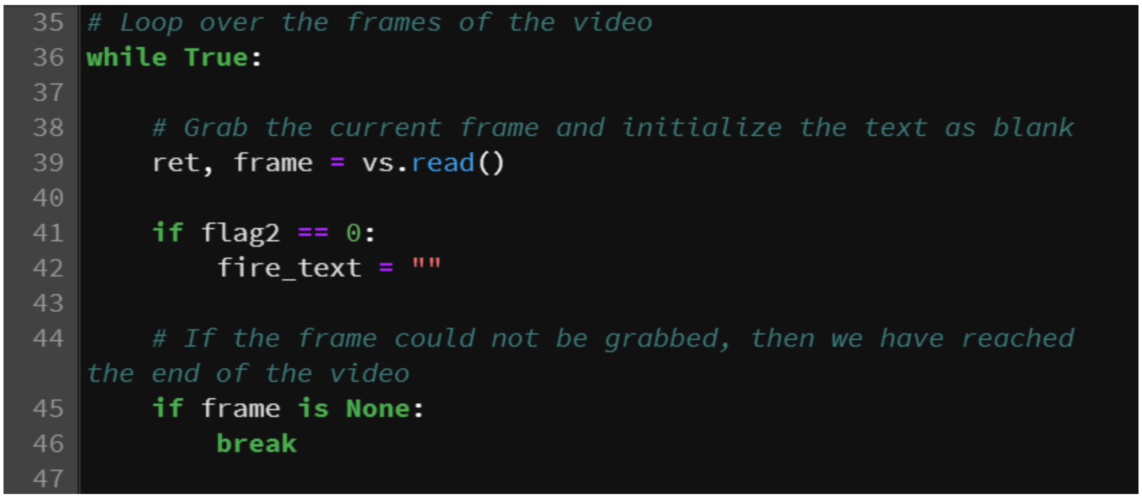
```python
# Reading from the webcam
vs = cv2.VideoCapture(0)
```

Fig 31: Screenshot of Code for Reading from the Webcam

Grabbing the Frames

We start a loop and iterate over each frame. The **vs.read()** function returns a webcam frame. Also, we quit the loop if a frame from the video file cannot be read successfully.

We create a string with the name **"fire_text"** and set its initial value to nothing, which indicates that there has been no indication of a fire in the monitored room. This string can be modified if a fire event occurs in the room. The loop is terminated if the frame cannot be obtained.

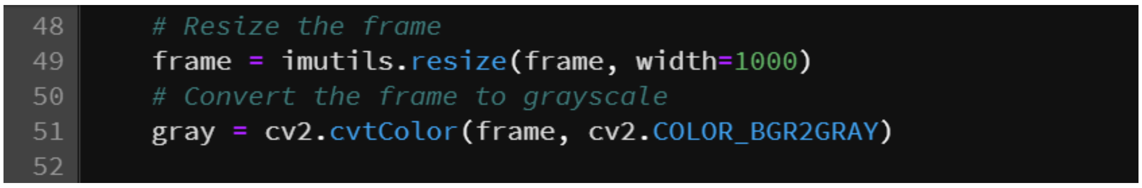


```python
# Loop over the frames of the video
while True:

    # Grab the current frame and initialize the text as blank
    ret, frame = vs.read()

    if flag2 == 0:
        fire_text = ""

    # If the frame could not be grabbed, then we have reached 
the end of the video
    if frame is None:
        break

```

Fig 32: Screenshot of Code for Grabbing the Frames

Processing the Frames

We begin processing and preparing the frame for fire analysis. We will first shrink the image to 1000 pixels wide using the **imutils.resize()** method, as handling the vast, raw images directly from the video stream is unnecessary. As color does not affect our fire detection method, we will also transform the image to grayscale. Using the **cv2.cvtColor()** method, we transform the color frame to a grayscale frame.



```
    # Resize the frame
    frame = imutils.resize(frame, width=1000)
    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

```

Fig 33: Screenshot of Code for Processing the Frames

Detecting Fire

Instead of applying all 6,000 characteristics to a single window, split the features into distinct stages of classifiers and apply them individually. If a window fails the first test, it should be discarded. We do not consider its remaining characteristics. Apply the second stage of features and continue the procedure if it passes. The window through which all phases pass is an area of the fire.

To detect fires, we utilize **cv2.CascadeClassifier.detectMultiScale()**, which is specified as follows:

cv2.CascadeClassifier.detectMultiScale(image[, scaleFactor[, minNeighbors[, flags[, minSize[, maxSize]]]]])

We only use the **image**, **scaleFactor**, and **minNeighbors** to detect fire in the frame.

- **image:** We pass the CV_8U matrix comprising an image in which objects are detected. So, we provide the **“frame”** variable to the classifier as the picture.
- **scaleFactor:** This scaling factor is used to construct the pyramid in the image. Imagine the scale factor is 1.02, which indicates that we are taking a modest step for resizing, i.e., reducing the size by 2%. By doing so, we enhance the likelihood that a matching size with the model for detection is discovered, even though the process is costly.

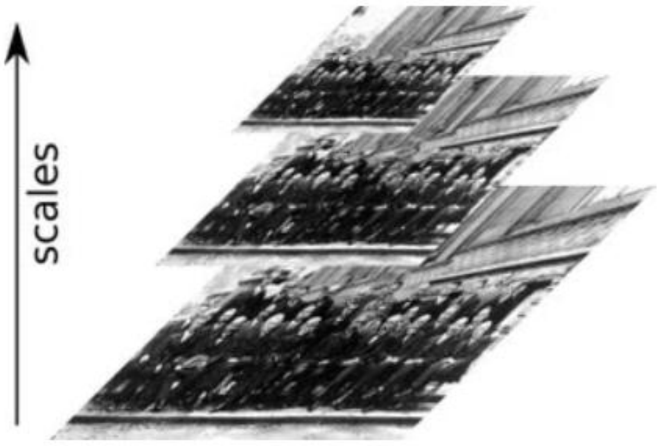
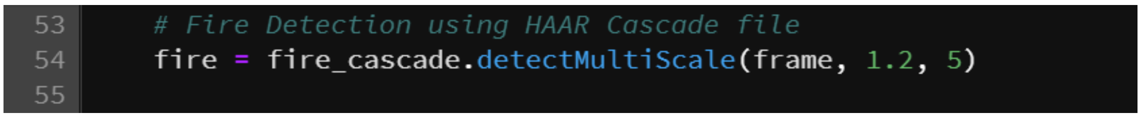


Fig 34: Scale Pyramid

- **minNeighbors:** Sets the minimum number of neighbors required for each candidate rectangle to be preserved. This parameter influences the quality of the recognized faces, with a higher value resulting in fewer detections but superior quality.

If flames are detected, the function returns the coordinates of detected fires as **Rect(x,y,w,h)**. The coordinates are stored in the **“fire”** variable.

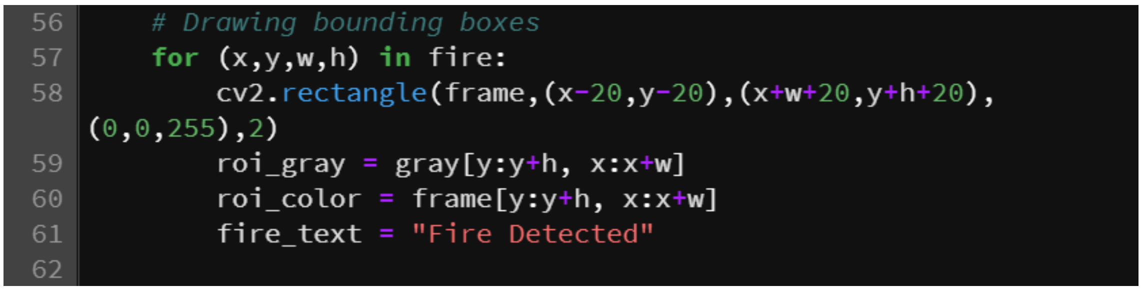


```python
    # Fire Detection using HAAR Cascade file
    fire = fire_cascade.detectMultiScale(frame, 1.2, 5)

```

Fig 35: Screenshot of Code for Detecting Fire

Drawing Bounding Boxes

Once the fire is detected through the HAAR Cascade Classifier, we extract the coordinates from the **"fire"** variable and start a loop to draw the bounding boxes in the region of the fire detected. The **cv2.rectangle()** function creates a bounding box in the frame using the x, y, w, and h coordinated of the fire. We highlight the region of interest or ROI to inform the user regarding the fire. Once we have drawn the bounding boxes, we need to change the status to be displayed in the security feed to **"Fire Detected**."

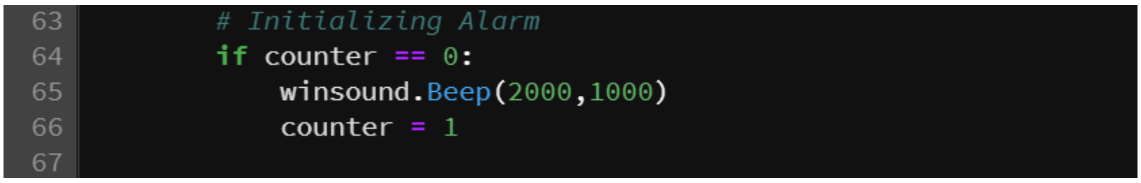


```python
    # Drawing bounding boxes
    for (x,y,w,h) in fire:
        cv2.rectangle(frame,(x-20,y-20),(x+w+20,y+h+20),
(0,0,255),2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = frame[y:y+h, x:x+w]
        fire_text = "Fire Detected"

```

Fig 36: Screenshot of Code for Drawing Bounding Boxes

Initializing Alarm

The Windows platform serves as the system's speaker and plays a buzzer when a fire occurs. **Winsound** library is used to start the alarm sound. The **Beep(frequency, duration)** function beeps the computer's speaker. The frequency parameter determines the sound's frequency in hertz and must lie within the 37 to 32,768 Hz range. The duration parameter specifies the length in milliseconds of the sound. We set the **frequency to 2000 Hertz** and the **duration to 1000 ms**. We update the allocated value of the counter from 0 to 1 so that the buzzer only sounds once and not for the duration of the fire event.



```python
        # Initializing Alarm
        if counter == 0:
            winsound.Beep(2000,1000)
            counter = 1

```

Fig 37: Screenshot of Code for Initializing Alarm

Saving the Fire Image and Sending Notification

We utilize the **"flag1"** variable to keep track of the number of fire-related occurrences. If the instance of fire is one, we record the date and use **"time_flag3"** to store a string that will be used as the name of the picture to be stored in the system and then sent as a notice to the telegram bot. The image's name will be formatted as follows:

Fire202232206.jpg

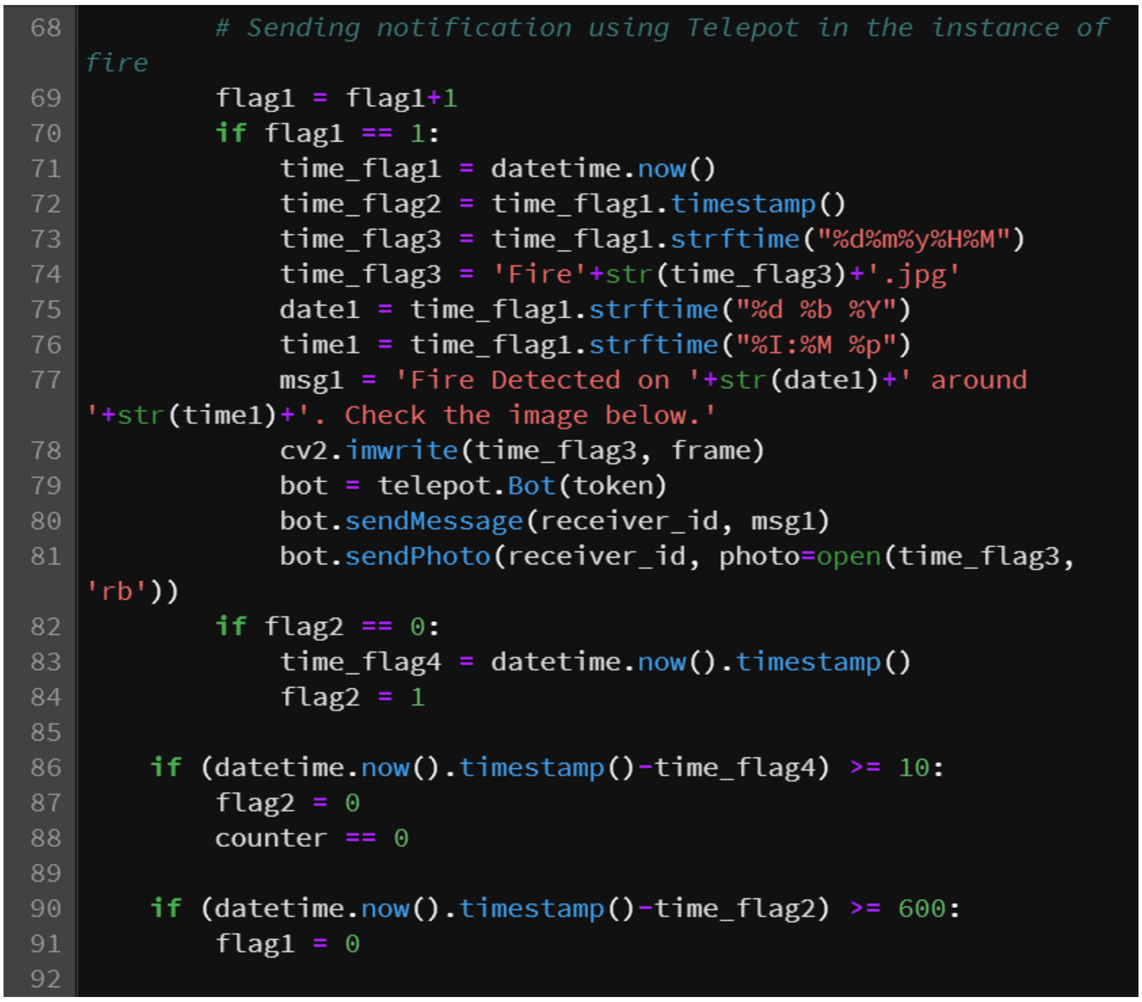
25 is the date, 02 is the month, 23 is the year, 22 is the hour, and 06 is the minute in the file name format. The file has been saved in jpg format. We now initialize the **"msg1"** variable with a notification message. The message is formatted as indicated below.

Fire Detected on "date1" around "time1". Check the image below.

Where **“date1”** and **“time1”** are variables that record the instance of the fire’s date and time. The frame of the instance of fire is then saved using the cv2.imwrite() method. The saved picture file and the created notice are transmitted to the Telegram bot.

We establish a condition to set the variables **“flag2”** and **“counter”** to 0. Something has been done to ensure the fire warning is displayed for at least 10 seconds.

In addition, we define a condition in which a fresh notice can only be delivered after 10 minutes or 600 seconds, even if a fire happens within 10 minutes. This was done to prevent too many notifications, which might cause the fire detection system to slow down.

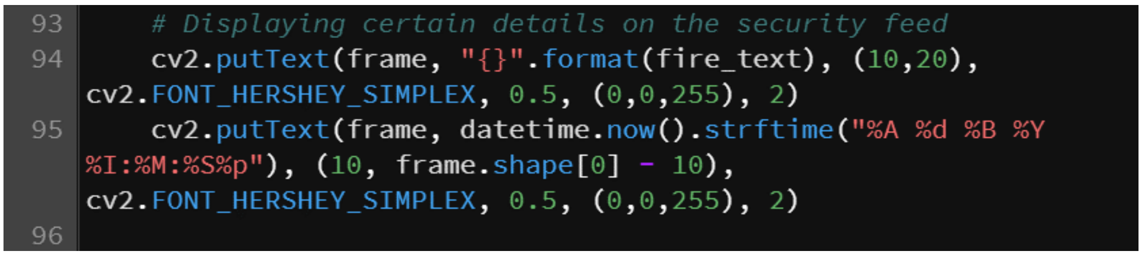


```
        # Sending notification using Telepot in the instance of fire
        flag1 = flag1+1
        if flag1 == 1:
            time_flag1 = datetime.now()
            time_flag2 = time_flag1.timestamp()
            time_flag3 = time_flag1.strftime("%d%m%y%H%M")
            time_flag3 = 'Fire'+str(time_flag3)+'.jpg'
            date1 = time_flag1.strftime("%d %b %Y")
            time1 = time_flag1.strftime("%I:%M %p")
            msg1 = 'Fire Detected on '+str(date1)+' around '+str(time1)+'. Check the image below.'
            cv2.imwrite(time_flag3, frame)
            bot = telepot.Bot(token)
            bot.sendMessage(receiver_id, msg1)
            bot.sendPhoto(receiver_id, photo=open(time_flag3, 'rb'))
        if flag2 == 0:
            time_flag4 = datetime.now().timestamp()
            flag2 = 1

    if (datetime.now().timestamp()-time_flag4) >= 10:
        flag2 = 0
        counter == 0

    if (datetime.now().timestamp()-time_flag2) >= 600:
        flag1 = 0

```

Fig 38: Screenshot of Code for Saving the Fire Image and Sending Notification

Displaying the Status and Time on the Security Feed

In the top left corner of the photo, we depict the state of the room as either a blank text if there is no fire or the phrase **“Fire Detected”** if there is a fire, followed by a timestamp in the bottom left corner of the picture. The timestamp has the format **“Saturday 25 February 2023 22:25:30 PM**.”

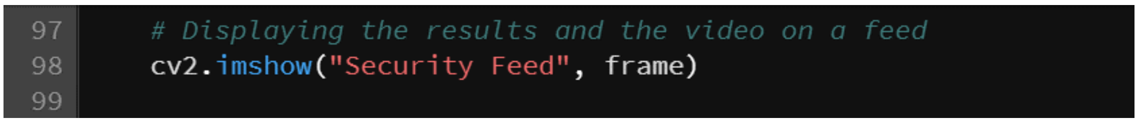


```
    # Displaying certain details on the security feed
    cv2.putText(frame, "{}".format(fire_text), (10,20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 2)
    cv2.putText(frame, datetime.now().strftime("%A %d %B %Y %I:%M:%S%p"), (10, frame.shape[0] - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 2)

```

Fig 39: Screenshot of Code for Displaying the Status and Time on the Security Feed

Displaying the Results

Displaying the results helps present the outcomes of our efforts and lets us check whether a fire was located in the footage. The footage being shown has been labeled **"Security Feed,"** and users can view it to maintain constant surveillance over the region.

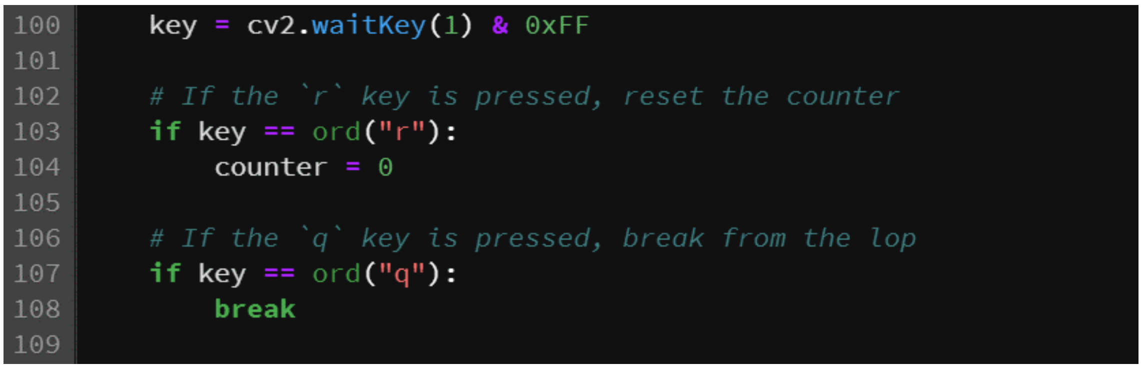


```
    # Displaying the results and the video on a feed
    cv2.imshow("Security Feed", frame)

```

Fig 40: Screenshot of Code for Displaying the Results

Initializing Keys

We provide the variable **"key"** its initial value so that it may be used as input by the user to perform various actions in response to pressing particular keys. To reset the buzzer, use the **"r" key** on your keyboard, and press the **"q" key** to halt the process of executing the model.

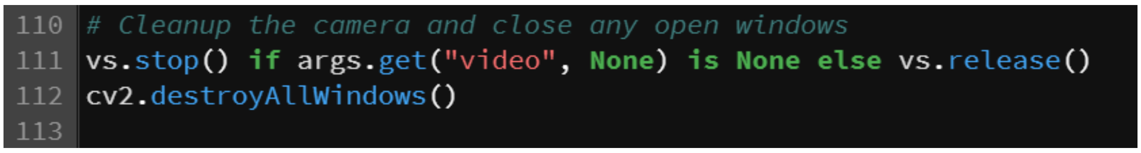


```
    key = cv2.waitKey(1) & 0xFF

    # If the `r` key is pressed, reset the counter
    if key == ord("r"):
        counter = 0

    # If the `q` key is pressed, break from the lop
    if key == ord("q"):
        break

```

Fig 41: Screenshot of Code for Initializing Keys

Cleanup and Release the Video Stream Pointer

At this point, everything is cleaned up, and the pointer to the video stream is released. Script-exiting users can delete or close all windows at any time by utilizing the Python OpenCV destroyAllWindows() function.



```
# Cleanup the camera and close any open windows
vs.stop() if args.get("video", None) is None else vs.release()
cv2.destroyAllWindows()

```

Fig 42: Screenshot of Code for Cleanup and Release of the Video Stream Pointer

Camera Placement

When attempting to implement fire detection using a webcam and OpenCV effectively, the positioning of the camera inside the room is a crucial aspect that must be considered. The camera must be positioned to have an unobstructed view of the whole room. There should be nothing in the way of the vision. In addition, it should be situated in an area that is easily accessible for maintenance and cleaning.

The camera should ideally be positioned on the ceiling or high on a wall, and a wide-angle lens should be used so that a significant portion of the room may be captured in the image. This enables a more comprehensive coverage and more precise detection of flames. It is essential to place the camera so that it is not exposed to intense light sources or direct sunlight, which can negatively impact the image quality and the system's overall performance.

In addition, the positioning of the camera should be such that it results in a minimum of false alarms being triggered. This may be accomplished by avoiding regions with much foot traffic, such as the areas close to the doors, windows, and vents. The camera should also be placed in a location far from potential sources of heat or smoke, such as fireplaces or cooking equipment, as these might cause false alarms to be triggered.

When successfully implementing fire detection using a webcam and OpenCV, it is essential to consider the camera's positioning inside the room. A more accurate and reliable system may lead to more effective fire detection and prevention if properly placed, which can be helped by careful placement.

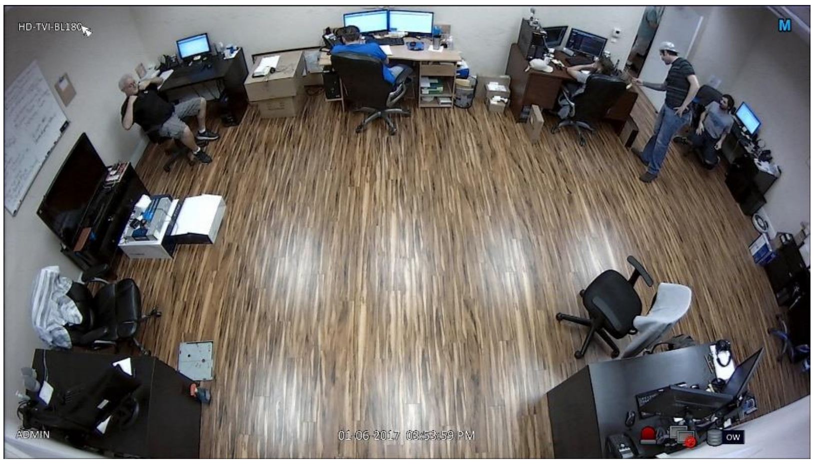


Fig 43: Example of Camera Placement

Result

In recent years, there has been an increasing interest in the application of computer vision technology in fire detection and prevention. Specifically, this interest has been driven by advancements in the field. It has been determined that using cameras and OpenCV for fire detection is a promising technology that can give early identification of fire breakouts, which in turn can permit rapid reaction and mitigation.

During the completion of this project, the possibility of performing fire detection using a camera and OpenCV was proved. Setting up the hardware and software for the project, preparing the HAAR Cascade file, collecting and refining images, incorporating the push notification, testing and evaluating the system, improving the system, and finally deploying the final version of the system are all steps that are included in the methodology of the project.

As a result of the model, we can see, as shown in **Figure 43**, that the **model can detect whether or not there is a fire in the room**. This is a consequence of the model. In addition to that, whenever there is a fire, the model will make a buzzing beeping sound.

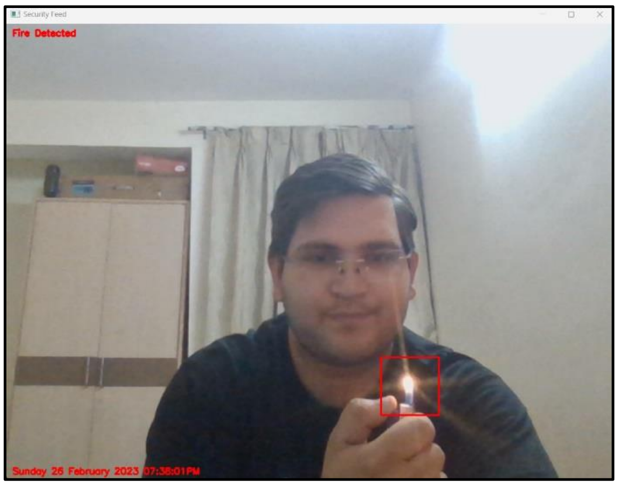
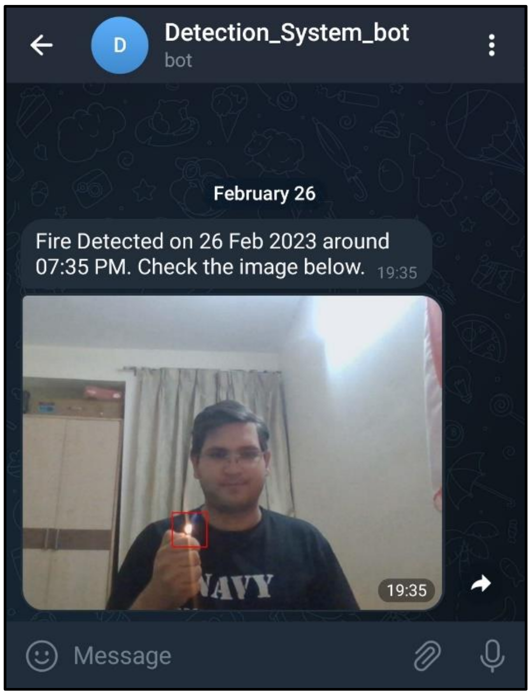


Fig 44: Screenshot of the Final Result of the Model

The notification system that the project has in place is an essential component since it enables rapid responses to fire breakouts. The notification system will send notifications to a telegram bot with the fire's date, time, and position. This will provide crucial information for reaction and mitigation efforts. The notification system may be modified to cater to a user's particular requirements and can be incorporated with additional notification systems. We can see the model sending a **fire detection notification to Telegram** in **Figure 44**.



February 26

Fire Detected on 26 Feb 2023 around 07:35 PM. Check the image below. 19:35

Fig 45: Screenshot of the Telegram Notification

The performance of the HAAR Cascade file and the camera both have a role in determining how accurate the fire detection system is. As a result, it is essential to check that the camera is appropriately positioned and that the HAAR Cascade file has been appropriately adjusted to detect fire properly. In addition, more computer vision algorithms and machine learning strategies might be included in the system to bring further improvements.

Utilizing a camera and OpenCV to implement a fire detection system has several benefits, including cost-effectiveness, the convenience of deployment, and low maintenance needs. The system may find use in various contexts, including residential and commercial structures, woods, and other environments prone to fire breakouts.

In conclusion, the experiment has proven that computer vision technology has the potential to improve fire detection and prevention. Further research and development in this area might result in the creation of fire detection systems that are both more advanced and complex, which would ultimately contribute to the protection of lives and property from the dangers posed by fire.

Way Forward

Implementing fire detection using a webcam and OpenCV is a significant step toward improving building fire safety. However, there are several areas where further research and development can be done to enhance the system's accuracy, reliability, and efficiency.

- **Optimization of the Detection Algorithm**
  While the current system uses a pre-installed Haar Cascade file for fire detection, other algorithms such as HOG, YOLO, and SSD can be explored. These algorithms are more robust and can handle

lighting conditions, camera angles, and object size variations. Additionally, the performance of the detection algorithm can be improved by training the model on a larger dataset of fire images.

- **Integration with Smart Home**
  When a fire is detected, the device may be connected to a home automation system and programmed to switch off all electrical appliances. This will lessen the likelihood of fires caused by electrical sources and stop the spread of existing fires.
- **Interoperability with Other Communication Channels**
  The system can deliver notifications to email addresses or mobile phones through text messages and send alerts to a Telegram bot. In addition, the notification system can be programmed to send an automated alarm to the fire department or any other appropriate authorities.
- **Connection with Safety Systems**
  Integration of fire detection with other safety systems, such as sprinkler and fire suppression systems. Integration of fire detection with other safety systems. For instance, the system may be programmed to activate a sprinkler system once a fire is detected. This helps to limit the fire and prevent it from spreading further throughout the building.
- **Camera Placement**
  It is possible to record the scene from various angles and points of view using several different cameras that may be spread out over the room in various locations. Integrating and analyzing the pictures captured by each camera to get higher precision in fire detection is possible.

To summarise, constructing an efficient fire detection system starts with implementing fire detection using a camera and OpenCV. This is only the beginning of the process. The next step entails conducting more research and development work to enhance the system’s precision, dependability, and efficacy. In the end, this will help to the protection of people as well as property from the dangers that are posed by fire.

Conclusion

Both projects demonstrate how powerful computer vision techniques can be in real-world settings, particularly for safety and protection. The projects show how image processing techniques may be successfully implemented by utilizing OpenCV and a camera to detect motion and fire.

The Motion Detection System subtracts the most recent frame from the previous frames' average to determine whether or not the motion is occurring. This method enables the detection of moving objects inside the field of vision, and on the first occurrence of motion, it sends a notice to the user via a telegram bot. A timestamp and a picture of the identified motion are also included in the notice. In addition, a CSV file is produced in order to log the date, in addition to the beginning and ending times of the move.

In the Fire Detection System, fire identification is accomplished by utilizing HAAR Cascade files that have been pre-installed. This enables the recognition of particular patterns within the image that is suggestive of a fire, which in turn causes a notification to be sent to the user via a telegram bot the first time a fire occurs. In addition, the message contains a timestamp and an image of the discovered fire. Following then, the user receives a notification once every ten minutes in the event that fresh fires occur.

Integrating the Telegram bot into both projects enables quick notifications to be sent to the user, providing real-time monitoring and the ability to respond immediately to any possible dangers. The incorporation of photos into the alerts offers visual confirmation of the events that have been identified. This enables the user to determine the location of the occurrence and respond appropriately. The Motion Detection System includes a CSV file that enables the development of an event log, which can then be studied later for trend analysis or additional study.

In general, these projects show the various ways computer vision technology may be practically applied in the sectors of safety and security. They demonstrate the possibility of real-time monitoring and response to potential dangers, which makes it possible to take action at the appropriate moment. These initiatives have the potential to be further expanded and incorporated into pre-existing security systems, which would allow for improved monitoring and safety measures.

Bibliography

- Kaehler, A., & Bradski, G. (2017). Learning OpenCV 3: computer vision in C++ with the OpenCV library. Packt Publishing Ltd.
- Navaux, P. O., Schulter, A. D., & de Assis Angeloni, M. (2012). A surveillance system based on face recognition and motion detection. In 2012 International Conference on Multimedia Computing and Systems (pp. 1119-1124). IEEE.
- Mohamed, H., Ibrahim, M., & Atia, A. (2016). Motion detection and tracking using feature extraction and matching techniques. International Journal of Computer Applications, 148(7), 17-23.
- Han, J., Li, J., Chen, Y., & Wang, Y. (2020). Real-time motion detection based on deep learning. Journal of Physics: Conference Series, 1569(3), 032012.
- Patel, R., Patel, J., Patel, P., & Patel, B. (2017). Real time security system using Raspberry pi and OpenCV. International Journal of Computer Applications, 169(11), 8-12.
- Wang, Z., Zou, L., & Wang, J. (2019). Fire detection based on image processing and machine learning. Journal of Physics: Conference Series, 1290(1), 012038. https://doi.org/10.1088/1742-6596/1290/1/012038
- Prasetyo, H., Rahmatullah, M., & Darma, A. (2020). Fire detection system based on OpenCV algorithm in various lighting conditions. Journal of Physics: Conference Series, 1467(1), 012031.
- Chen, C., Li, Q., & Xie, Y. (2018). Fire detection system based on computer vision technology. In 2018 International Conference on Computational Science and Computational Intelligence (CSCI) (pp. 1365-1368).
- Zhou, X., Hu, Y., Chen, L., Liu, H., Wang, Z., & Zhao, Y. (2019). Fire detection based on deep learning algorithm. In 2019 IEEE International Conference on Image Processing (ICIP) (pp. 1410-1414). IEEE.
- GeeksforGeeks. (2020). Webcam Motion Detector using Python. Retrieved from https://www.geeksforgeeks.org/webcam-motion-detector-python
- Towards Data Science. (2020). Understanding Motion Analysis in Machine Learning. Retrieved from https://towardsdatascience.com/understanding-motion-analysis-in-machine-learning-f504e9987413
- H Pranamurti1, A Murti1, & C Setianingsih1. (2019). Fire Detection Use CCTV with Image Processing Based Raspberry Pi. Journal of Physics: Conference Series, 1201(1), 012015.
- GeeksforGeeks. (2021). Extract Video Frames from Webcam and Save to Images using Python. Retrieved from https://www.geeksforgeeks.org/extract-video-frames-from-webcam-and-save-to-images-using-python
- GeeksforGeeks. (2021). Background Subtraction in an Image using Concept of Running Average. Retrieved from https://www.geeksforgeeks.org/background-subtraction-in-an-image-using-concept-of-running-average
- Bogotobogo. OpenCV-Python - How to install OpenCV-Python package to Anaconda. Retrieved from https://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_Object_Detection_Face_Detection_Haar_Cascade_Classifiers.php

Exhibits

Exhibit A: Motion Detection System Code

```
 # Importing Libraries
import os
import cv2
import numpy
import pandas
import imutils
from imutils.video import VideoStream
import argparse
import time
import datetime
from datetime import datetime
import winsound
import telepot

# Defining the variables
motion = [None, None]
time = []
date = []
counter = 0
flag1 = 0
flag2 = 0
time_flag1 = 0
time_flag2 = 0
time_flag3 = ''

# Token and ID for Telegram
token = '5860611748:AAEWG-ZjGEtZLUpEX6cKZwpP7u3L2Kjn13c'
receiver_id = 1278690585

# Defining a dataframe for storing motion date, start & end time
df = pandas.DataFrame(columns = [“Date”, “Start-Time”, “End-Time”])
```

```
# Construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-a", "--min-area", type=int, default=1000, help="minimum area size")
args = vars(ap.parse_args())

# Reading from the webcam
vs = cv2.VideoCapture(0)

# Create a variable to store the averaged background frame
avg = None

# Loop over the frames of the video
while True:

    # Grab the current frame and initialize the occupied/unoccupied text
    ret, frame = vs.read()
    flag1 = 0
    text = "No Motion Detected"

    # If the frame could not be grabbed, then we have reached the end of the video
    if frame is None:
        break

    # Resize the frame
    frame = imutils.resize(frame, width=1000)
    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Blur the grayscale frame to remove noise
    gray = cv2.GaussianBlur(gray, (21, 21), 0)

    if avg is None:
        avg = gray.copy().astype("float")
        continue

    # Accumulate the weighted average of the frame and the background frame
```

```
cv2.accumulateWeighted(gray, avg, 0.1)
# Compute the absolute difference between the current frame and the background frame
frameDelta = cv2.absdiff(gray, cv2.convertScaleAbs(avg))
thresh = cv2.threshold(frameDelta, 25, 255, cv2.THRESH_BINARY)[1]

# Dilate the thresholded image to fill in holes, then find contours on the thresholded image
thresh = cv2.dilate(thresh, None, iterations=2)
cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)

# Loop over the contours
for c in cnts:

    # If the contour is too small, ignore it
    if cv2.contourArea(c) < args["min_area"]:
        continue
    flag1 = 1

    # Compute the bounding box for the contour, draw it on the frame, and update the text
    (x,y,w,h) = cv2.boundingRect(c)
    cv2.rectangle(frame, (x,y), (x+w, y+h), (0,255,0),2)
    text = "Motion Detected"

    # Initializing Alarm
    if counter == 0:
        winsound.Beep(2000,1000)
        counter = 1

    # Sending notification using Telepot in the instance of motion
    flag2 = flag2+1
    if flag2 == 1:
        time_flag1 = datetime.now()
        time_flag2 = time_flag1.timestamp()
        time_flag3 = time_flag1.strftime("%d%m%y%H%M")
        time_flag3 = 'Motion'+str(time_flag3)+'.jpg'
```

```
            date1 = time_flag1.strftime("%d %b %Y")
            time1 = time_flag1.strftime("%I:%M %p")
            msg1 = 'Motion Detected on '+str(date1)+' around '+str(time1)+'. Check the image below.’
            cv2.imwrite(time_flag3, frame)
            bot = telepot.Bot(token)
            bot.sendMessage(receiver_id, msg1)
            bot.sendPhoto(receiver_id, photo=open(time_flag3, 'rb'))

    if (datetime.now().timestamp()-time_flag2) >= 300:
        flag2 = 0

    # Appending instances of the motion in the dataframe
    motion.append(flag1)
    motion = motion[-2:]

    if motion[-1] == 1 and motion[-2] == 0:
        time.append(datetime.now().strftime("%H:%M:%S"))
        date.append(datetime.now().date())

    if motion[-1] == 0 and motion[-2] == 1:
        time.append(datetime.now().strftime("%H:%M:%S"))
        date.append(datetime.now().date())

    if text == "No Motion Detected":
        counter = 0

    # Draw the text and timestamp on the frame
    cv2.putText(frame, "Status: {}".format(text), (10,20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255),
2)
    cv2.putText(frame, datetime.now().strftime("%A %d %B %Y %I:%M:%S %p"), (10, frame.shape[0] - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 2)

    # Displaying the results and the video on a feed
    cv2.imshow("Security Feed", frame)
            cv2.imshow(“Threshold”, thresh)
        cv2.imshow("Frame Delta", frameDelta)

    key = cv2.waitKey(1) & 0xFF

    # If the `r` key is pressed, reset the counter
    if key == ord("r"):
        counter = 0

    # If the `q` key is pressed, break from the loop
    if key == ord(“q” ):
        if flag1 == 1:
            time.append(datetime.now().strftime("%H:%M:%S"))
            date.append(datetime.now().date())
        break

# Appending all the instances of motion to the dataframe
for i in range(0, len(time), 2):
    df = df.append({"Date":date[i], "Start-Time":time[i], "End-Time":time[i+1]}, ignore_index = True)

# Saving the dataframe as a CSV to have a log of all the instances of motion
df.to_csv("Security_Log.csv")

# Clean up the camera and close any open windows
vs.stop() if args.get(“video”, None) is None else vs.release()
cv2.destroyAllWindows()
```

Exhibit B: Fire Detection System Code

```
# Importing Libraries
import os
import cv2
import numpy
import pandas
import imutils
from imutils.video import VideoStream
import argparse
import time
import datetime
from datetime import datetime
import winsound
import telepot

# Defining the variables
counter = 0
flag1 = 0
flag2 = 0
time_flag1 = 0
time_flag2 = 0
time_flag3 = ''
time_flag4 = 0
fire_text = ""

# Token and ID for Telegram
token = '5860611748:AAEWG-ZjGEtZLUpEX6cKZwpP7u3L2Kjn13c'
receiver_id = 1278690585

# HAAR Cascade file for fire detection
fire_cascade = cv2.CascadeClassifier('fire_detection.xml')

# Reading from the webcam
vs = cv2.VideoCapture(0)
```

```
# Loop over the frames of the video
while True:

    # Grab the current frame and initialize the text as blank
    ret, frame = vs.read()

    if flag2 == 0:
        fire_text = ""

    # If the frame could not be grabbed, then we have reached the end of the video
    if frame is None:
        break

    # Resize the frame
    frame = imutils.resize(frame, width=1000)
    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Fire Detection using HAAR Cascade file
    fire = fire_cascade.detectMultiScale(frame, 1.2, 5)

    # Drawing bounding boxes
    for (x,y,w,h) in fire:
        cv2.rectangle(frame,(x-20,y-20),(x+w+20,y+h+20),(0,0,255),2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = frame[y:y+h, x:x+w]
        fire_text = "Fire Detected"

        # Initializing Alarm
        if counter == 0:
            winsound.Beep(2000,1000)
            counter = 1

        # Sending notification using Telepot in the instance of fire
        flag1 = flag1+1
        if flag1 == 1:
            time_flag1 = datetime.now()
            time_flag2 = time_flag1.timestamp()
            time_flag3 = time_flag1.strftime("%d%m%y%H%M")
            time_flag3 = 'Fire'+str(time_flag3)+'.jpg'
            date1 = time_flag1.strftime("%d %b %Y")
            time1 = time_flag1.strftime("%I:%M %p")
            msg1 = 'Fire Detected on '+str(date1)+' around '+str(time1)+'. Check the image below.’
            cv2.imwrite(time_flag3, frame)
            bot = telepot.Bot(token)
            bot.sendMessage(receiver_id, msg1)
            bot.sendPhoto(receiver_id, photo=open(time_flag3, 'rb'))
        if flag2 == 0:
            time_flag4 = datetime.now().timestamp()
            flag2 = 1

    if (datetime.now().timestamp()-time_flag4) >= 10:
        flag2 = 0
        counter == 0

    if (datetime.now().timestamp()-time_flag2) >= 600:
        flag1 = 0

    # Displaying certain details on the security feed
    cv2.putText(frame, "{}".format(fire_text), (10,20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 2)
    cv2.putText(frame, datetime.now().strftime("%A %d %B %Y %I:%M:%S%p"), (10, frame.shape[0] - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 2)

    # Displaying the results and the video on a feed
    cv2.imshow("Security Feed", frame)

    key = cv2.waitKey(1) & 0xFF

    # If the `r` key is pressed, reset the counter
    if key == ord("r"):
        counter = 0

    # If the `q` key is pressed, break from the loop
    if key == ord("q"):
        break

# Clean up the camera and close any open windows
vs.stop() if args.get(“video”, None) is None else vs.release()
cv2.destroyAllWindows()
```