CSE1901 - Technical Answers to Real World Problems (TARP)

Project Report

AUTOMATIC EARLY FIRE DETECTION USING DEEP LEARNING AND SENDING ALARM ALERT AND SMS

By

20BAI1061	Mandla Sheshi Kiran Reddy
20BAI1191	Amaravadi Dheeraj
20BAI1289	Uppanapalli Lakshmi Sowjanya

B. Tech Computer Science and Engineering

Submitted to

Dr. Rukmani P

School of Computer Science and Engineering

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

April 2022

DECLARATION

I hereby declare that the report titled **"AUTOMATIC EARLY FIRE DETECTION USING DEEP LEARNING AND SENDING ALARM ALERT AND SMS"** submitted by me to VIT Chennai is a record of bona-fide work undertaken by me under the supervision of **Dr. Rukmani P**, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai.

Mandla Sheshi Kiran Reddy
20BAI1061

Amaravadi Dheeraj
20BAI1191

Uppanapalli Lakshmi Sowjanya
20BAI1289

CERTIFICATE

Certified that this project report entitled **"AUTOMATIC EARLY FIRE DETECTION USING DEEP LEARNING AND SENDING ALARM ALERT AND SMS"** is a bonafide work of Mandla Sheshi Kiran Reddy (20BAI1061), Amaravadi Dheeraj (20BAI1191), Uppanapalli Lakshmi Sowjanya (20BAI1289) and they carried out the Project work under my supervision and guidance for CSE1901 - Technical Answers to Real World Problems (TARP).

Dr. Rukmani p
SCOPE, VIT Chennai

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. Rukmani P**, Senior Professor, School of Computer Science Engineering, for her consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We express our thanks to our **Head of The Dept Dr. Sweetlin Hemaltha C (for B.Tech Cse with AI and ML)** for her support throughout the course of this project.

We also take this opportunity to thank all the faculty of the school for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

Mandla Sheshi Kiran Reddy
20BAI1061

Amaravadi Dheeraj
20BAI1191

Uppanapalli Lakshmi Sowjanya
20BAI1289

ABSTRACT

The safety of people and property is paramount, and reliable fire detection systems are critical in preventing loss of life and damage to property. Wireless fire detection systems have gained popularity due to their ease of use and convenience. One such system is the Wireless Fire Detection System Using Deep Learning & Python with Raspberry Pi Pico - GSM Call / SMS Alert.

This system uses a combination of Deep Learning and Python programming language to detect fires in real-time. The Raspberry Pi Pico serves as the microcontroller to control the entire system, and it is a wireless system that is easy to install and maintain.

The system includes a camera that captures real-time video footage, which is analyzed by the Deep Learning algorithm. The algorithm has been trained to detect fire by analyzing various features such as color, texture, and motion of flames. Once the system detects a fire, it sends an alert message to the GSM module, which can either make a call or send a text message to a predetermined phone number. This feature ensures that people can be alerted promptly, even if they are not present on the premises.

The Wireless Fire Detection System Using Deep Learning & Python with Raspberry Pi Pico - GSM Call / SMS Alert is an efficient and reliable solution for detecting fires. By using advanced technologies such as Deep Learning, the system can detect fires quickly and accurately. The wireless design of the system makes it easy to install and maintain, which makes it suitable for both residential and commercial buildings. With this system in place, people can have peace of mind knowing that their property is protected from fires, and they will be alerted promptly in case of an emergency.

In conclusion, the Wireless Fire Detection System Using Deep Learning & Python with Raspberry Pi Pico - GSM Call / SMS Alert is a powerful system that leverages Deep Learning technology to provide fast and accurate fire detection. The system's wireless design, coupled with its ability to send alerts via GSM calls or SMS, ensures that people can take immediate action to prevent loss of life and property.

CONTENTS

1. Introduction

In recent years, technological advancements have revolutionized the way we approach fire safety in our homes and workplaces. Wireless fire detection systems have become increasingly popular due to their ability to detect fires quickly and efficiently without the need for complex wiring or infrastructure.

One of the latest innovations in wireless fire detection systems is the use of deep learning and Python with Raspberry Pi Pico to create an intelligent and efficient system that can detect fires and alert authorities using GSM call or SMS. The system utilizes machine learning algorithms to analyses data from various sensors, including temperature sensors, smoke detectors, and motion sensors.

The Raspberry Pi Pico is a powerful microcontroller board that is compatible with Python programming language, making it an ideal platform for developing a wireless fire detection system. Its compact size and low power consumption make it ideal for use in small spaces, making it an ideal choice for both residential and commercial applications.

Deep learning algorithms allow the system to analyses data from sensors in real-time, making it possible to detect fires quickly and accurately. The system can also learn from past data, allowing it to improve its accuracy over time.

The GSM call and SMS alert feature is a critical component of the system, ensuring that authorities are alerted to fires quickly, allowing them to respond promptly and minimize damage. This feature is particularly useful in cases where the building is unoccupied, such as during weekends or holidays, ensuring that fires are detected and reported regardless of whether anyone is present.

In summary, the wireless fire detection system using deep learning and Python with Raspberry Pi Pico is an innovative and intelligent solution for fire safety. Its ability to detect fires quickly and accurately, and its GSM call and SMS alert features make it an essential tool for protecting homes and businesses from the devastating effects of fires.

1.1 **Objective and goal of the project**

The core objective of our project is to build a wireless fire detection system using deep learning and python with Raspberry Pi Pico as an intelligent solution for fire safety. The outcome of our project must have GSM call and SMS alert features along with the ability to detect fire quickly and accurately which can help in protecting homes and businesses from the devastating effects of fire.

1.2 **Problem Statement**

Our project "AUTOMATIC EARLY FIRE DETECTION USING DEEP LEARNING AND SENDING ALARM ALERT AND SMS" is all about designing a fire detection system which comes with additional features like sending an alarm and SMS to the fire stations nearby along with some given numbers like police stations, ambulance etc... Fire detection is done using a booming computer science technology i.e., Deep Learning. We first developed a deep learning and then incorporated some hardware elements like GSM to send alarms and alerts. This project can be used in forests to detect forest fires, in old age homes for fast response, near transformers and some sensitive official places like data servers. Fire in these places has to be recognized soon and action has to be taken. The cameras we setup keep monitoring the locations and if there is a trace of fire, alarms will be on to help nearby people to move out to safer place and SMS will be sent to centers like fire stations, police stations, hospitals etc... for further action.

1.3 **Motivation**

There are several reasons to use deep learning in automated fire alarm systems:

- **Early detection of fires:** Fires can spread quickly and cause devastating property damage and serious risk to life safety. Automated fire detection using deep learning algorithms can provide early warnings that enable rapid response and

implementation of mitigation measures, potentially saving lives and minimizing property damage.

- **Real-time monitoring:** Deep learning-based fire detection systems can continuously monitor large areas such as forests, industrial facilities, or buildings in real time without human intervention. This enables rapid detection of fires, even in remote or inaccessible locations, as they develop and enables immediate response and intervention.
- **High Accuracy:** Deep learning algorithms can achieve high accuracy in fire detection by analyzing large amounts of visual and/or thermal data that human operators may find difficult to process. The ability to detect fires accurately and quickly can significantly reduce false alarms and ensure that firefighting resources are efficiently allocated and disruptions to normal operations are minimized.
- **Cost effective:** Automated fire detection using deep learning can potentially reduce costs associated with manual fire monitoring and inspection. It can also help prevent costly damage that can result from late detection of fire, such as: B. Loss of property, equipment, and loss of production.
- **Scalability:** Deep learning-based fire detection systems can be easily adapted to different environments, such as: B. Indoor, outdoor, and even large-scale forest fire detection. This makes them versatile and adaptable to various applications, from industrial facilities and transportation hubs to residential and commercial buildings.
- **Safety:** The use of deep learning-based fire detection systems can reduce the need for human firefighters to enter hazardous or hazardous environments for fire detection and minimize their exposure to risk such as toxic smoke, extreme heat and structural collapse.

- **Integration with other technologies:** Deep learning-based fire detection systems can be integrated with other technologies such as drones, IoT sensors and emergency response systems to provide a solution comprehensive fire detection and management. This can improve situational awareness, reduce response times, and enable coordinated firefighting efforts.

1.4 **Challenges**

Automated fire detection using deep learning models has received significant attention in recent years due to its potential for improving fire safety and prevention. However, several challenges need to be addressed in order to develop accurate and reliable automated fire detection systems using deep learning models. Some of the main challenges are:

- **Data scarcity:** Deep learning models require large amounts of labeled data for training. However, due to the rarity of fires and associated safety risks, it can be difficult to obtain a large set of fire-related image or video data for training purposes. This can lead to limited data availability, which can lead to overfitting or poor generalization performance of the trained model. Variability in fire types and conditions: Fires can occur in a variety of forms, such as forest fires, building fires, and industrial fires, and can exhibit different behaviors based on factors such as fuel type, flame size, and the environmental conditions. This variability makes it difficult to develop a single deep learning model that can accurately detect fires in different types of fires and under different conditions. This may require robust model architectures and several training data.
- **Class Imbalance:** In fire detection datasets, the number of non-fire-related images or videos usually significantly exceeds the number of fire-related samples, resulting in a class imbalance. This can affect the model's ability to accurately detect fires, as the model may lean towards the majority (non-fire) class during training, resulting in reduced sensitivity and false negatives in detecting fires.

- **Real-time processing:** Fire detection systems often need to work in real-time to respond and take action in a timely manner. However, deep learning models can be computationally intensive and achieving real-time processing can be difficult, especially in resource-limited environments such as embedded systems or Internet of Things devices. (IoT). Balancing accuracy and real-time processing requirements is a major challenge in developing practical automated fire detection systems.
- **Environmental Factors:** Environmental factors such as lighting, smoke, and occlusion can significantly affect the accuracy of fire detection systems. Smoke, in particular, can obscure the visibility of flames, making it difficult for deep learning models to accurately detect fires in smoky environments. Developing robust models capable of handling these environmental factors and maintaining high detection accuracy is a challenge.
- **Safety Issues:** Fire detection systems must operate in safety-critical environments, and false positives or false negatives in fire detection can have serious consequences. Ensuring the reliability, safety, and interpretability of deep learning fire detection models is a key challenge that must be addressed to gain trust and acceptance in practical applications.
- **Generalize across environments:** Deep learning models trained on specific datasets may have limited generalization capabilities when applied to different settings or domains. For example, a model trained on indoor fires may not perform well when applied to outdoor wildfire scenarios. Ensuring the generalizability of fire detection deep learning models across different environments and scenarios is a challenge that requires careful consideration of the training data and model architecture.

2. Literature Survey

[1] ResNet50 is a reliable model for early-time monitoring of forest fires, based on VGG16, ResNet50, and DenseNet121. It classifies images into two classes, fire, and no fire, and uses high train and test accuracy. Fine-tuning helps the model avoid overfitting and the model can be applied in real-time to avoid forest fires.

[2] The proposed system for forest fire detection using wireless sensor networks and machine learning was found to be an effective method with accurate results. It is able to be mounted at any place in the forest and is easily implementable as a standalone system for prolonged periods. It also alerted authorities with lower latency than existing systems during test trials.

[3] This paper presents a smart fire detection system that will notify authorities in the early stages of their early stages and even before the fire breakout. It uses a signal-processing unit, an image processing unit, and a GSM module unit. An integrated sensor system is proposed instead of using a single detector. A machine learning approach is adapted and compared with the output to get more accurate detection. The proposed system encompasses the entire safety and security of the property as well as reduces the expense and time while designing and securing modern properties in respect of fire danger. The adapted soft computing approach can open a scope for promoting further research work.

[4] The developed prototype is designed to be used by a user to control the fire alarm system remotely. It is more appropriate than the use of only one of the modules and its portability can be improved by an efficient assimilation of the different modules. It can be applied in smart cities due to its flexibility and simplicity in handling.

[5] This proposed a fire detection approach based on computer vision methods and deep learning technology. It was successfully deployed based on C++ pro-gamming language and videos from surveillance cameras were processed frame by frame through a background subtraction method to extract moving region images. The VSD algorithm was used to extract significant regions from the area containing moving objects. To reduce the fire false alarm, a motion detection method based on background superagency was added. The proposed method was developed for practical application in forests and other high-fire risk scenarios.

[6] This paper discusses a new low-cost drone equipped with image processing and object detection abilities for smoke and fire recognition tasks in forests. The ssdlite nanobilenet model, a sub-model of the supervised deep learning model called Model Zoo, is used as an example. The proposed system can be improved by increasing flight time and using convertible energy sources such as solar energy. It can also be trained to detect flame and fire visuals, and a downward fire/flame scan can be performed on the vehicle suspended above the smoke detection zone. The study aims to shed light on similar scientific studies.

[7] This work presents on Recurrent Trend Predictive Neural Network (rTPNN) for multi sensor fire detection. It is based on trend prediction, level prediction, and fusion of sensor readings. It outperforms other models with 96% accuracy and achieves high True Positive and True Negative rates at the same time. It also triggers an alarm in 11s from the start of the fire, making it acceptable for real-time applications. They used several models like LSTM, SVM, MLP and compared with rTPNN. The rTPNN model outperforms all of the machine learning models in terms of the prediction performance with high generalization ability. This paper interprets that the rTPNN model will have an impact on the various time series problems.

[8] FireNet is a lightweight neural network built from scratch and trained on a diverse dataset to develop an internet of things (IoT) capable fire detection unit that can replace physical sensor-based detectors and reduce false and delayed triggering.

The introduced neural network can smoothly run on a low-cost embedded device like Raspberry Pi 3B at a frame rate of 24 frames per second. The performance obtained by the model on a standard fire dataset and a self-made test dataset, the dataset consists challenging real-world fire and non-fire images with image quality that is similar to the images captured by the camera attached to Raspberry Pi. The performance is in terms of accuracy, precision, recall, and F-measure is encouraging.

[9] This paper mainly focuses on alerting the surrounding people and also inform the firemen regarding the incident. The proposed system contains the fire alarm, in addition to that it sends a notification to our mobile and mail can be sent to the attached mail id which will be having the information of the accident-prone area, and also the information needed to alert the fire station about the incident. This implemented fire detection system will be more efficient and provide more safety when compared to the conventional fire alarm systems available earlier.

[10] In this paper, the main purpose is to build fire detection system that is able to detect the conditions of the environment and the results are informed quickly on the incident area and fire department. This study uses Arduino Uno, GPS Ublox Neo 6MV2, SIM900A and three sensors to build an early fire detection system with an average GPS error of 1.6%, an accuracy of reading shifts, and an average distance of ± 4 meters. The average data transmission speed between Providers is 1 second because the processing time and sending speed are in line with the network conditions and the capabilities of the GSM module used.

3 Requirements Specification

3.1 Hardware Requirements

Micro Controller

A microcontroller is a compact hardware device containing a central processing unit (CPU), memory, and input/output peripherals, commonly used in fire detection systems to perform control and monitoring functions.

Power Supply

A power supply is a hardware device used in fire detection systems to convert incoming AC voltage to the appropriate DC voltage required to power the system's components, such as sensors, control panels, and alarms.

GSM module

A GSM module is a hardware device used in fire detection systems to enable wireless communication via cellular networks, allowing the system to transmit alarm signals and notifications to remote monitoring centers or mobile devices.

Laptop

A laptop is a portable hardware device commonly used in fire detection systems for programming and configuring control panels, analyzing system data, and displaying alerts and notifications in real-time.

USB cable

A USB cable is a hardware device used in fire detection systems to establish a wired connection between the system's components and a computer or other peripheral device, allowing for programming, data transfer, and firmware updates.

LCD

An LCD (Liquid Crystal Display) is a hardware device used in fire detection systems to provide visual feedback on system status, such as sensor readings, alarm conditions, and menu-based user interfaces, using low power consumption and compact form factor.

LED’s

LEDs (Light Emitting Diodes) are hardware devices used in fire detection systems to provide visual indication of system status, such as power, fault, and alarm conditions, using low power consumption and long service life.

ALARM

A alarm is a hardware device used in fire detection systems to automatically gives alarm sound or other fire-suppression agents when triggered by a fire alarm, helping to extinguish or contain fires in their early stages.

3.2 Software Requirements

Embedded C

Embedded C is a programming language used to develop software for microcontrollers and other embedded systems. It is commonly used in fire detection systems to control and monitor sensor inputs, alarms, and other system functions.

ARDUINO IDE

Arduino IDE is an integrated development environment used to program Arduino microcontrollers. It can be used in fire detection systems to develop and upload software for controlling and monitoring sensor inputs, alarms, and other system functions.

Uc-Flash

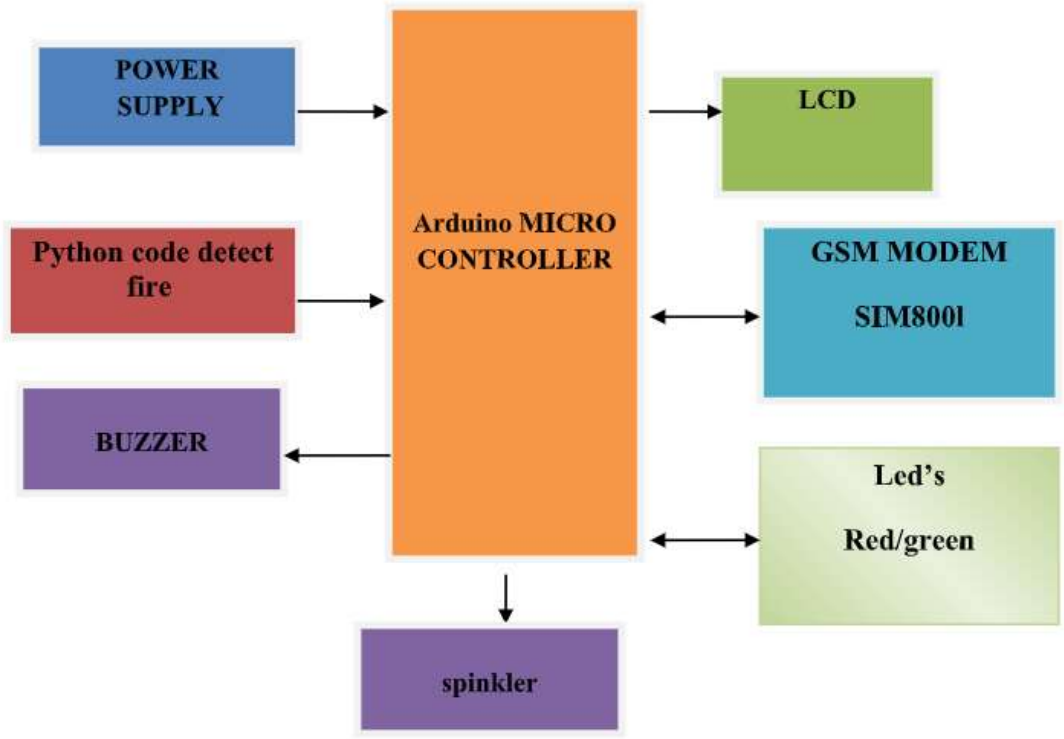
UC-Flash is a software utility used for programming and updating firmware on microcontrollers. It can be used in fire detection systems to update or change the software running on the microcontroller controlling and monitoring sensor inputs, alarms, and other system functions.

Express PCB

Express PCB is a computer-aided design software used to create printed circuit board (PCB) layouts. It can be used in fire detection systems to design and manufacture custom PCBs for controlling and monitoring sensor inputs, alarms, and other system functions.

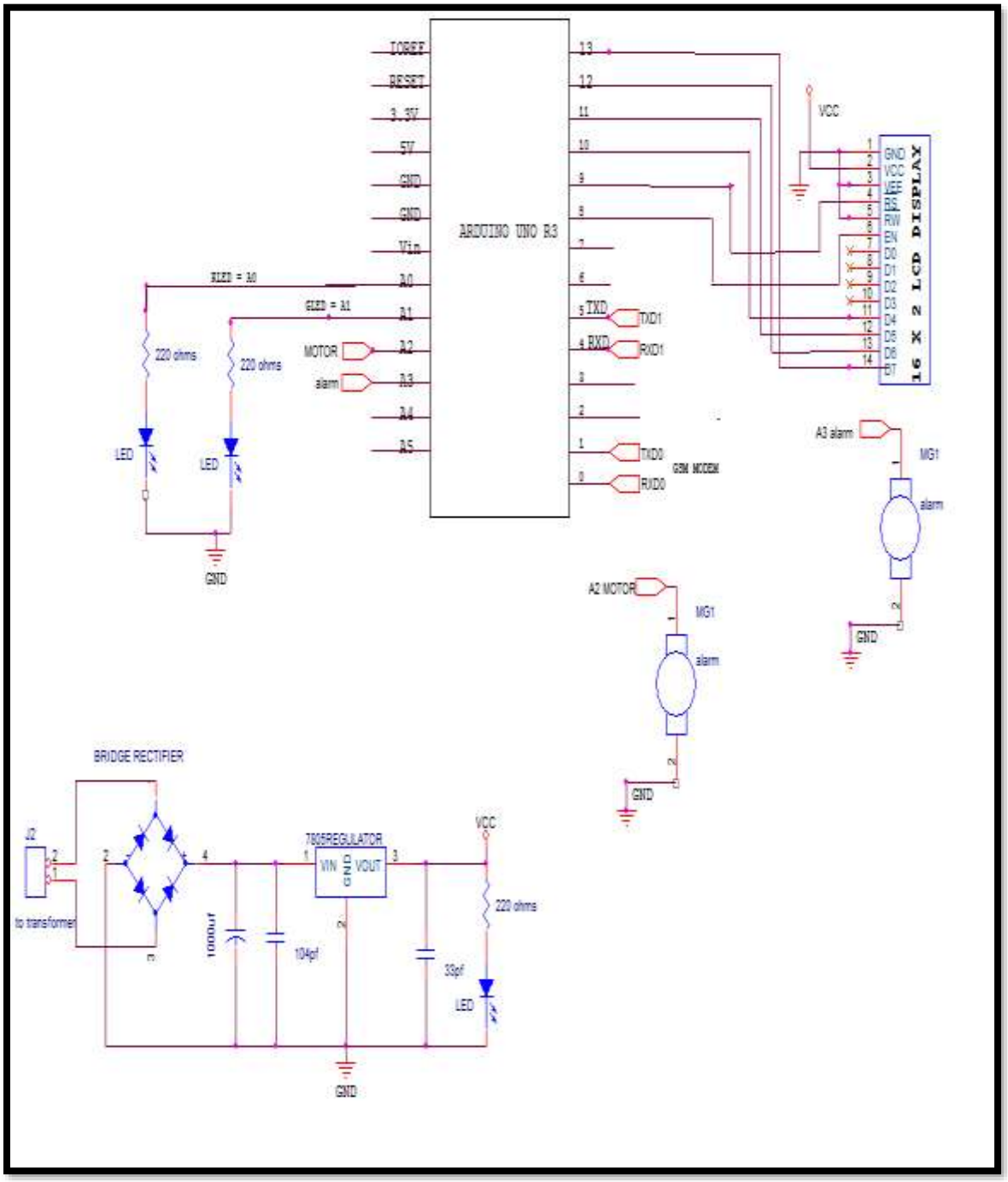
4 System Design

This section should normally cover design / programming methodologies



5 Implementation of System

(a) Hardware implementation:



Step 1: Connect the Arduino board.

- Connect the Arduino board to your laptop using a USB cable.
- Make sure you have the Arduino IDE or a compatible IDE installed on your laptop.

Step 2: Connect the GSM module.

- Connect the GSM module to the Arduino board using the appropriate pins.
- Connect the Rx pin of the GSM module to a digital pin on the Arduino board (e.g. pin 2).
- Connect the Tx pin of the GSM module to a digital pin on the Arduino board (e.g. pin 3).
- Connect the GND pin of the GSM module to the GND pin of the Arduino board.
- Connect the GSM module's VCC pin to a 5V pin on the Arduino board.

Step 3: Connect the LCD module

- Connect the LCD module to the Arduino board using the appropriate pins.
- Connect the RS pin of the LCD module to a digital pin on the Arduino board (eg pin 4). Connect the LCD module's EN pin to a digital pin on the Arduino board (eg pin 5).
- Connect the D4, D5, D6 and D7 pins of the LCD module to the digital pins of the Arduino board (eg pins 6, 7, 8, 9).
- Connect the GND pin of the LCD module to the GND pin of the Arduino board.
- Connect the VCC pin of the LCD module to a 5V pin on the Arduino board.

Step 4: Connect the LEDs

- Connect the Fire Alarm LED to a digital pin on the Arduino board (eg pin 10).
- Connect the system status LED to a digital pin on the Arduino board (eg pin 11).
- Connect the positive lead of both LEDs to a 220 ohm resistor and connect the other end of the resistor to a 5V pin on the Arduino board.
- Connect the negative pole of the two LEDs directly to the GND pin of the Arduino board.

Step 5: Connect the fire alarm

- Connect the fire alarm or fire extinguisher to a digital pin on the Arduino board (eg pin 12).
- Connect the other end of the fire alarm to a separate power supply, eg. a battery or a relay, depending on the needs of your specific watering mechanism.

Step 6: Turn on the Arduino board

- Power the Arduino board with a 9V battery connected to the power socket or via the USB cable connected to your laptop.

Step 7: Install and configure the required software

- Install OpenCV and Python on your laptop if they are not already installed.
- Upload the supplied Arduino code to interface with the GSM module, LCD module, LEDs and fire alarm onto the Arduino board using the Arduino IDE or any other compatible IDE.
- Upload the Arduino fire detection code using OpenCV to the Arduino board.

Step 8: Implement the fire detection system

- Connect a camera to the laptop and make sure it is recognized by OpenCV.
- We have to develop the python script with required Deep Learning Algorithms that captures video frames from the camera along with OpenCV.
- Process video frames using OpenCV to detect fire using appropriate image processing techniques (for example, color-based fire detection).
- If a fire is detected, send a signal to the Arduino board via a USB cable to activate the fire alarm LED and send an SMS notification using the GSM module.
- Displays system status and fire alarms on the LCD module.
- Based on the output, activate the alarm or fire mechanism to extinguish the fire using the Arduino board.

Step 9: Test and troubleshoot

- Run the Python script and test the fire detection system by exposing it to fire or similar conditions.
- Verify that the Fire Alarm LED activates, an SMS notification is sent and the LCD module displays the correct messages.
- Verify that the alarm or fire suppression mechanism is activated correctly.

(b) Software stimulation:

The python script which we develop using the concepts of the data augmentation, deep learning algorithms such as the Resnet, InceptionV3, YOLO, MobilenetV2 and also, we use the hyper parameter tuning such as the Adam, Adagrad & the RMSProp etc.

DATA AUGMENTATION:

Data augmentation is a common technique used in deep learning to artificially increase the size of the training dataset by creating new training examples from the original data through various transformations. In automated fire detection using deep learning, data augmentation can be particularly useful for improving model performance and robustness. Here are some data augmentation techniques which are used are applied specifically for fire detection:

- **Image Rotation:** By rotating the images by different degrees, the model can learn to see the fires from different orientations. Fires can appear from different angles and orientations in real-life scenarios, and by rotating the images during training, the model can learn to see fires from different perspectives.

- **Image Flip:** Flipping images horizontally or vertically can create additional training examples and help the model learn to detect fires in mirror or inverted orientation. This can be useful when working with images captured from different sources or cameras that may have different orientations.

- **Image Scaling:** By enlarging or reducing the images, the model can learn to recognize fires of different sizes. Fires can vary in size, and by applying scale transformations, the model can learn to recognize fires at different scales, from small flames to large infernos.

- **Image Translation:** Horizontal or vertical translation of images can be used to simulate changes in camera viewpoints or camera angles. This can help the model learn to spot fires from different perspectives and positions in the image.

- **Image Brightness and Contrast Adjustment:** Adjusting image brightness and contrast can simulate changes in lighting conditions, such as different times of day or weather conditions. This can help the model learn to detect fires under varying lighting conditions and make it more resilient to changes in lighting.

- **Adding Random Noise:** Adding random noise to images can simulate image distortions or sensor noise, which can be common in realistic fire detection scenarios. This can help the model become more robust against noise or distorted images.

- **Data fusion:** Combining images from different sources or modalities, such as infrared and visible images, can provide additional information and improve the model's ability to accurately detect fires. This can be done by overlaying the images, stitching them together or using them as separate channels in a multi-channel input network.

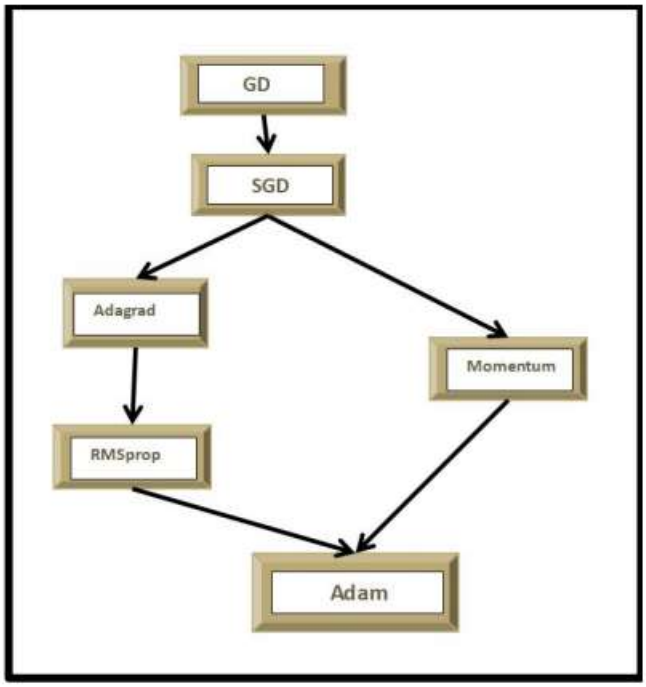
DATASET CREATED:

We created the dataset using the data augmentation of nearly 3000 images with the train data having 2700 images and the test data with 300 images. The dataset is totally divided into 3 classes that is fire, smoke and neutral as it is also important for us to take the consideration of the distinction between the smoke and the fire and consideration of non-fire scenarios.

ALGORITHMS USED:

ALGORITHMS	DESCRIPTION
ALGORITHM-1	**Implementation OF InceptionV3:** InceptionV3 is a deep convolutional neural network architecture that has been used for image classification and object detection tasks. It consists of multiple layers of convolutional and pooling operations, followed by fully connected layers. InceptionV3 has shown high accuracy in detecting flames and smoke in fire scenarios.
ALGORITHM-2	**Implementation OF ResNet:** ResNet (short for Residual Network) is a deep neural network architecture that has won several image classification competitions. It uses skip connections to enable the training of much deeper neural networks without vanishing gradients. ResNet has been used for early fire detection and has shown high accuracy.
ALGORITHM-3	**Implementation OF YOLO:** YOLO is an object detection algorithm that can identify and locate objects in real-time video. YOLO has been used to detect flames and smoke in video feeds from cameras and sensors. YOLO-based approaches for early fire detection can achieve high accuracy and real-time performance.
ALGORITHM-4	**Implementation OF MobileNetV2:** MobileNetV2 is a lightweight convolutional neural network architecture that has been optimized for mobile and embedded devices. It uses depthwise separable convolutions to reduce the number of parameters and increase the speed of the model. MobileNetV2 has been used for real-time fire detection on low-power devices and has shown high accuracy.

OPTIMIZERS USED:



SGD:

SGD is Stochastic Gradient Descent (SGD) optimizer is a popular optimization algorithm used in machine learning and deep learning for training models. It is a widely used iterative optimization algorithm to minimize the loss function during the training process.

- The basic idea behind SGD is to update the model parameters in small steps based on the gradient of the loss function with respect to the parameters. The gradient is the vector of partial derivatives of the loss function with respect to each parameter and indicates the direction of the strongest increase in the loss function.
- SGD uses a small subset of the training data, usually called a mini-batch, to compute the gradient and update the parameters at each iteration, making it computationally efficient and suitable for large datasets.
- SGD has several hyperparameters to set, such as learning rate, momentum, and batch size. The learning rate determines the step size of parameter updates and can affect the speed of convergence and the stability of the optimization process.
- The momentum term adds a fraction of the previous parameter update to the current update, helping to accelerate convergence and reduce wobble in the optimization path. The batch size determines the number of training samples used to compute the gradient at each iteration and can affect gradient estimation noise and the computational efficiency of the algorithm.
- SGD is a widely used optimizer due to its simplicity and efficiency, and is often used as a core optimizer in deep learning experiments.
- However, it also has some limitations, such as sensitivity to hyperparameter settings, sensitivity to getting stuck in local optima, and slow convergence in some cases.
- Consequently, researchers have proposed several modifications and variants of SGD, such as mini-batch SGD, momentum SGD, Nesterov accelerated gradient (NAG), and adaptive frequency learning methods such as AdaGrad, RMSprop, and

Adam, to address some of these limitations. and improve optimization performance in different scenarios.

Adam:

Adam is another optimization algorithm that can be used in CNN models to efficiently train them and improve their performance in predicting outcomes from a dataset where every attribute is equally important. Here are some key points about Adam's uniqueness and how it works:

- This means that it not only uses a moving average of past gradients like momentum, but also adapts the learning rates of each weight parameter based on the statistics of the gradients.
- The learning rates in Adam are adapted by computing a separate adaptive learning rate for each weight parameter based on estimates of the first and second moments of the gradients.
- The first moment estimate is a moving average of the gradients, which serves as an estimate of the gradient mean.
- The second moment estimate is a moving average of the squared gradients, which serves as an estimate of the gradient variance.
- By using these estimates, Adam is able to automatically adapt the learning rates for each weight parameter in a way that is sensitive to the magnitude and direction of the gradients.
- Overall, Adam is a powerful optimization algorithm that can be used to train CNN models more efficiently and effectively, especially when every attribute of the input data is equally important.
- Its ability to adapt the learning rates of each weight parameter based on estimates of the gradients can help the model to converge more quickly and produce more accurate results.

Adagrad:

Adagrad is an optimization algorithm that can be used with CNN models to train them more effectively and efficiently. By predicting results from a dataset where each attribute is equally important, Adagrad can help ensure that the model learns the most important features of the input data and converges to a good solution. Here are some key points about the uniqueness of Adagrad and how it works:

- Adagrad is a gradient-based optimization algorithm, i.e. it works by calculating the gradients of the loss function with respect to the model parameters and adjusting them in a direction that minimizes the loss.
- This means that parameters with large gradients have a lower learning capacity, while parameters with small gradients have a higher learning capacity.
- Learning rate matching in Adagrad is done using a parameter specific learning rate, calculated by dividing the initial learning rate by the sum of squares of the historical gradients for that parameter.
- This means that parameters with large historical gradients have a slower learning rate, while parameters with small historical gradients have a higher learning rate.
- This is because the parameter-specific learning rate can be adjusted separately for each feature, so the model learns the most important features faster and more accurately.

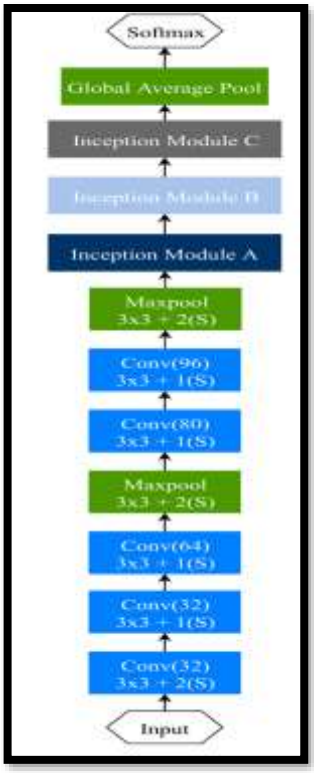
- Instead, the learning rate is automatically adjusted for each parameter based on its historical gradients.

RMSProp

RMSProp is an optimization algorithm that can be used in CNN models to improve the efficiency and effectiveness of the training process. When predicting outcomes from a dataset where each attribute is equally important, RMSProp can help the model learn the most important features and converge to a good solution. Here are some key points about RMSProp and its uniqueness:

- Unlike Adagrad, RMSProp uses a moving average of squared gradients to normalize the learning rate, which helps prevent the learning rate from becoming too large or too small as training progresses.
- One of the main differences between RMSProp and other optimization algorithms is that it uses a parameter- specific learning rate that adapts based on the moving average of the squared gradients of each parameter.
- This is because parametric learning rates can help the model avoid getting stuck in narrow valleys of the loss function.
- One potential disadvantage of RMSProp is that it can be sensitive to the choice of hyper-parameters, such as the initial learning rate and the decay rate of the moving average.
- Overall, RMSProp is a powerful optimization algorithm that helps CNN models learn more efficiently and effectively, especially when dealing with large datasets or datasets with high-dimensional inputs.
- Its parametric learning rate adaptation, moving average of squared slopes, and robustness to local optima make it a popular choice for many deep learning applications.

INCEPTION-V3:



A key component of fire safety is early fire detection. Fires can be disastrous, resulting in the loss of life, destruction of property, and harm to the environment. Early fire detection can help avoid these disastrous effects by enabling timely action to put out the fire and reduce damage. Deep learning models like Inception Net v3 have demonstrated particularly promising outcomes in early fire detection using artificial intelligence (AI).
A deep convolutional neural network called Inception Net v3 classifies images by combining convolutional layers, pooling layers, and fully connected layers. It has been trained on a sizable dataset, and it has attained great accuracy rates, making it a popular choice for picture classification tasks.

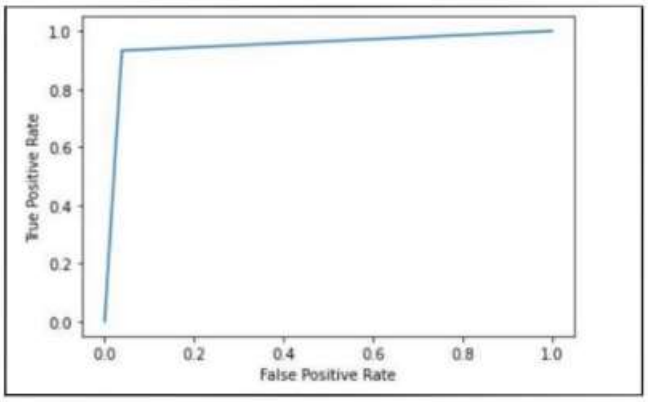
Because of its ability to identify photographs of fires swiftly and reliably, it is perfect for early fire detection.
Installing cameras in strategic positions and putting the video feeds into the neural network are common procedures for early fire detection systems that use Inception Net v3. The neural network then does a real-time analysis of the images to look for fire-related indicators. When a fire is discovered, the neural network can alert a central monitoring system, which can then notify the emergency services.
The capability of Inception Net v3 to detect fires in a range of conditions is one advantage of employing it for early fire detection.
It can find flames in both indoor and outdoor settings, as well as in various weather situations. It can also find fires of different sizes, from tiny fires to massive firestorm. It is therefore a flexible instrument for early fire detection in a variety of circumstances.

For fire safety, early fire detection is essential, and artificial intelligence has shown promise in this regard. Due to its excellent accuracy rates and adaptability, Inception Net v3 is a particularly effective deep learning model for early fire detection. Early fire detection systems are going to become even more important as AI technology develops.

6. Results and Discussion

The highest accuracy we got is nearly 0.95 for the Inceptionnet-V3 model with SGD optimizer and the ROC CURVE is as follows:

	precision	recall	f1-score	support
0	0.92	0.96	0.94	518
1	0.97	0.93	0.95	656
micro avg	0.95	0.95	0.95	1174
macro avg	0.94	0.95	0.94	1174
weighted avg	0.95	0.95	0.95	1174
samples avg	0.95	0.95	0.95	1174



The reaction time is very fast compared to other algorithms and the reaction time is nearly 1.7 seconds. The use of Inception-netV3 is very less based on our survey and the accuracy and the reaction time is pretty good and also the OpenCV is helping us to to capture the frames properly.

7. Conclusion and Future Work

In summary, deep learning has shown great potential for fire detection due to its ability to automatically learn complex patterns from large amounts of data. Using deep learning models and Open-Cv demonstrated high accuracy in detecting fires in various environments, including indoor and outdoor environments, and through different imaging and video modalities.

However, there are still challenges and future directions for deep learning fire detection that can be explored further. Some potential areas for future work include:

- Real-time fire detection: Real-time fire detection is essential for rapid response and effective firefighting. Further research could focus on developing real-time deep learning models that provide instantaneous fire detection capabilities, enabling faster response and mitigation of fire incidents.

- Robustness to different environmental conditions: Deep learning fire detection models can face challenges when dealing with different environmental conditions such as lighting changes, weather conditions, and smoke. Future research could explore techniques to improve the robustness of deep learning models to different environmental conditions and make them more reliable in real-world scenarios.

- Multimodal fire detection: Deep learning models can benefit from integrating multiple modalities such as image, video, audio, and sensor data for fire detection. Future work could explore multimodal deep learning approaches that can leverage different data sources to improve the accuracy and robustness of fire detection systems.

- Edge computing for fire detection: Edge computing, where processing occurs locally on devices at the edge of the network, can significantly reduce latency and bandwidth requirements. Future research could focus on developing effective deep learning models that can be deployed on edge devices such as drones, surveillance cameras, and IoT devices to detect fires in real-time in remote or remote environments. limited resources.

- Large-scale fire detection: Deep learning models can be trained on large datasets to further improve their performance. Future research could focus on the collection and annotation of large-scale fire datasets, which can help train more accurate and robust deep learning fire detection models.

- Explainable AI for fire detection: Deep learning models are often regarded as black boxes because they are not interpretable. Future research could focus on developing explainable AI techniques that can inform the decision-making process of deep learning fire detection models, making them more transparent and verifiable.

- We have to also overcome the false positives here which is major concern

In summary, while deep learning has shown promise for fire detection, there are several areas that can be further explored to improve the accuracy, real-time capabilities, robustness, and interpretability of fire detection models. of deep learning. Continuing research and development in these areas can contribute to the advancement of fire detection technology leading to more effective strategies for fire prevention, early detection and mitigation.

8. REFERENCES

[1]https://www.researchgate.net/publication/347869434_Early_Detection_of_Forest_Fire_using_Deep_Learning

[2] https://www.nature.com/articles/s41598-021-03882-9#Sec12

[3] https://www.worldscientific.com/doi/abs/10.1142/S1469026817500092

[4] https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3724291

[5] https://www.tandfonline.com/doi/full/10.1080/22797254.2022.2133745

[6] https://arxiv.org/ftp/arxiv/papers/2101/2101.09362.pdf

[7] https://ieeexplore.ieee.org/document/9451553

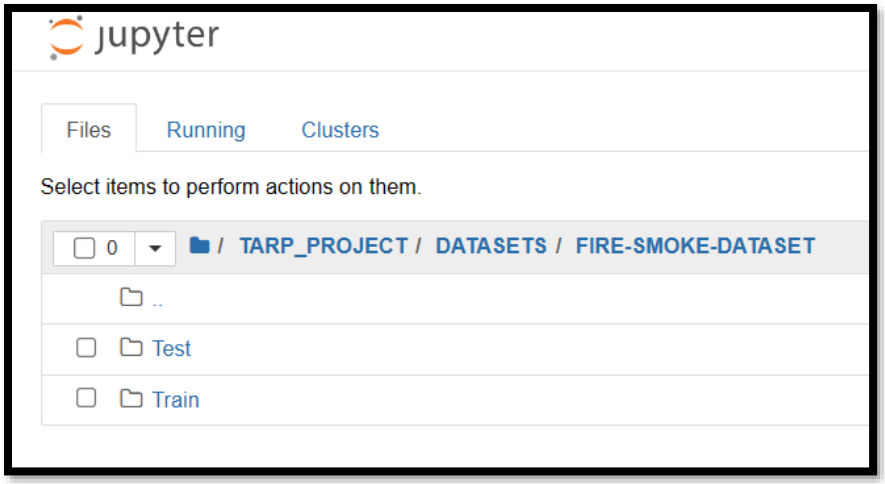
[8] https://arxiv.org/abs/1905.11922v2

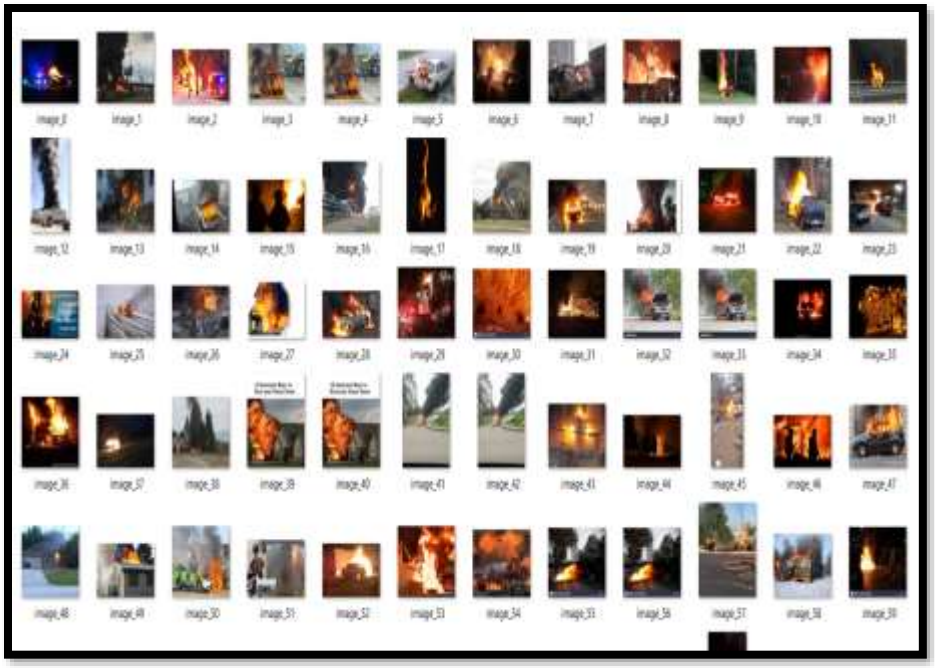
[9]https://www.researchgate.net/publication/341522138_Automatic_Fire_Detection_and_Alert_System

[10] https://iopscience.iop.org/article/10.1088/1757-899X/732/1/012056

APPENDIX

DATASET IMAGES:





DEEP LEARNING MODEL:

In [1]:

```
import tensorflow as tf
import keras_preprocessing
from keras_preprocessing import image
from keras_preprocessing.image import ImageDataGenerator
TRAINING_DIR = r"C:\Users\laksh\TARP PROJECT\FIRE-SMOKE-DATASET\FIRE-SMOKE-DATASET\Train"
training_datagen = ImageDataGenerator(rescale=1./255,zoom_range=0.15,horizontal_flip=True,fill_mode='nearest')
VALIDATION_DIR = r"C:\Users\laksh\TARP PROJECT\FIRE-SMOKE-DATASET\FIRE-SMOKE-DATASET\Test"
validation_datagen = ImageDataGenerator(rescale = 1./255)
train_generator = training_datagen.flow_from_directory(TRAINING_DIR,target_size=(224,224),shuffle = True,class_m
validation_generator = validation_datagen.flow_from_directory(VALIDATION_DIR,target_size=(224,224),class_mode='ca
```

```
Found 2700 images belonging to 3 classes.
Found 300 images belonging to 3 classes.
```

In [5]:

```
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Input, Dropout
input_tensor = Input(shape=(224, 224, 3))
base_model = InceptionV3(input_tensor=input_tensor, weights='imagenet', include_top=False)
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(2048, activation='relu')(x)
x = Dropout(0.25)(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.2)(x)
predictions = Dense(3, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)
for layer in base_model.layers:
  layer.trainable = False
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['acc'])
history = model.fit(train_generator,steps_per_epoch = 14,epochs = 20,validation_data = validation_generator,vali
```

```
Epoch 1/20
14/14 [==============================] - 172s 11s/step - loss: 9.1838 - acc: 0.5555 - val_loss: 0.7642 - val_acc:
0.7857
Epoch 2/20
14/14 [==============================] - 368s 27s/step - loss: 0.5086 - acc: 0.8052 - val_loss: 0.7949 - val_acc:
0.6820
Epoch 3/20
14/14 [==============================] - 322s 22s/step - loss: 0.5569 - acc: 0.7868 - val_loss: 0.4081 - val_acc:
0.8265
Epoch 4/20
14/14 [==============================] - 370s 28s/step - loss: 0.4868 - acc: 0.8175 - val_loss: 0.3065 - val_acc:
0.8724
Epoch 5/20
14/14 [==============================] - 147s 10s/step - loss: 0.4645 - acc: 0.8305 - val_loss: 0.3128 - val_acc:
0.8980
Epoch 6/20
14/14 [==============================] - 154s 11s/step - loss: 0.4330 - acc: 0.8465 - val_loss: 0.2852 - val_acc:
0.8929
Epoch 7/20
14/14 [==============================] - 151s 11s/step - loss: 0.5632 - acc: 0.8150 - val_loss: 0.3104 - val_acc:
0.8929
Epoch 8/20
14/14 [==============================] - 161s 11s/step - loss: 0.3166 - acc: 0.8789 - val_loss: 0.4799 - val_acc:
0.8265
Epoch 9/20
14/14 [==============================] - 228s 14s/step - loss: 0.3592 - acc: 0.8605 - val_loss: 0.7447 - val_acc:
0.7602
Epoch 10/20
14/14 [==============================] - 146s 10s/step - loss: 0.2982 - acc: 0.8813 - val_loss: 0.2621 - val_acc:
0.8980
Epoch 11/20
14/14 [==============================] - 146s 10s/step - loss: 0.4104 - acc: 0.8711 - val_loss: 0.3899 - val_acc:
```

In [6]:

```
#To train the top 2 inception blocks, freeze the first 249 layers and unfreeze the rest.
for layer in model.layers[:249]:
  layer.trainable = False
for layer in model.layers[249:]:
  layer.trainable = True
#Recompile the model for these modifications to take effect
from tensorflow.keras.optimizers import SGD
model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='categorical_crossentropy', metrics=['acc'])
history = model.fit(train_generator,steps_per_epoch = 14,epochs = 10,validation_data = validation_generator,vali
```

```
F:\Users\laksh\anaconda3\lib\site-packages\keras\optimizers\optimizer_v2\gradient_descent.py:108: UserWarning: Th
e `lr` argument is deprecated, use `learning_rate` instead.
  super(SGD, self).__init__(name, **kwargs)
```

```
Epoch 1/10
14/14 [==============================] - 157s 11s/step - loss: 0.6928 - acc: 0.6856 - val_loss: 0.2758 - val_acc:
0.9082
Epoch 2/10
14/14 [==============================] - 103s 7s/step - loss: 0.6428 - acc: 0.7132 - val_loss: 0.3017 - val_acc:
0.8929
Epoch 3/10
14/14 [==============================] - 97s 7s/step - loss: 0.5222 - acc: 0.7822 - val_loss: 0.2248 - val_acc: 0
.9031
Epoch 4/10
14/14 [==============================] - 129s 9s/step - loss: 0.4650 - acc: 0.7971 - val_loss: 0.2980 - val_acc:
0.8878
Epoch 5/10
14/14 [==============================] - 117s 8s/step - loss: 0.3919 - acc: 0.8490 - val_loss: 0.3074 - val_acc:
0.8724
Epoch 6/10
14/14 [==============================] - 118s 8s/step - loss: 0.3806 - acc: 0.9111 - val_loss: 0.3064 - val_acc:
0.9027
Epoch 7/10
14/14 [==============================] - 108s 8s/step - loss: 0.3231 - acc: 0.9041 - val_loss: 0.3213 - val_acc:
0.9076
Epoch 8/10
14/14 [==============================] - 114s 8s/step - loss: 0.3149 - acc: 0.9289 - val_loss: 0.3014 - val_acc:
0.9327
Epoch 9/10
14/14 [==============================] - 106s 7s/step - loss: 0.3213 - acc: 0.9289 - val_loss: 0.3354 - val_acc:
0.9322
Epoch 10/10
14/14 [==============================] - 98s 7s/step - loss: 0.2976 - acc: 0.9398 - val_loss: 0.2802 - val_acc: 0
.9429
```

In [16]:

```
model.summary()
model.save(r'C:\Users\laksh\TARP PROJECT\InceptionV3.h5')
```

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
=================================================================
 input_4 (InputLayer)        [(None, 224, 224, 3       0         []
                             )]

 conv2d_282 (Conv2D)         (None, 111, 111, 32       864       ['input_4[0][0]']
                             )

 batch_normalization_282 (Batch  (None, 111, 111, 32   96        ['conv2d_282[0][0]']
 Normalization)              )

 activation_282 (Activation)  (None, 111, 111, 32      0         ['batch_normalization_282[0][0]']
                             )

 conv2d_283 (Conv2D)         (None, 109, 109, 32       9216      ['activation_282[0][0]']
                             )

 batch_normalization_283 (Batch  (None, 109, 109, 32   96        ['conv2d_283[0][0]']
 Normalization)              )
```

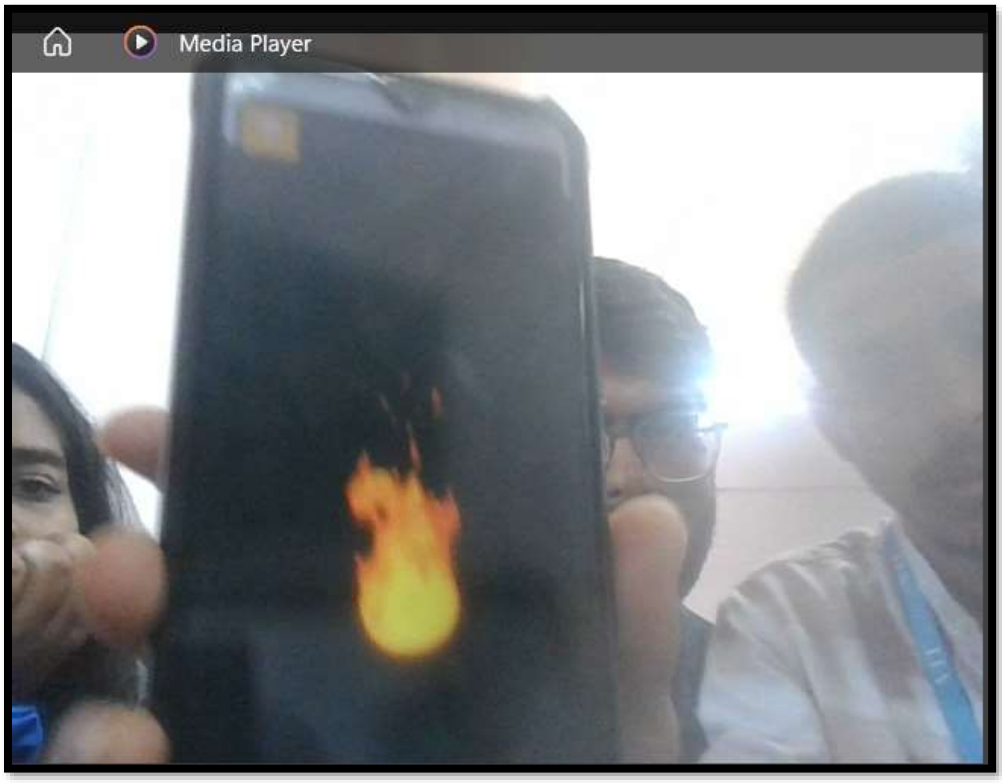
File Edit View Code Window Help app_tarp.py - ...\laksh\TARP PROJECT

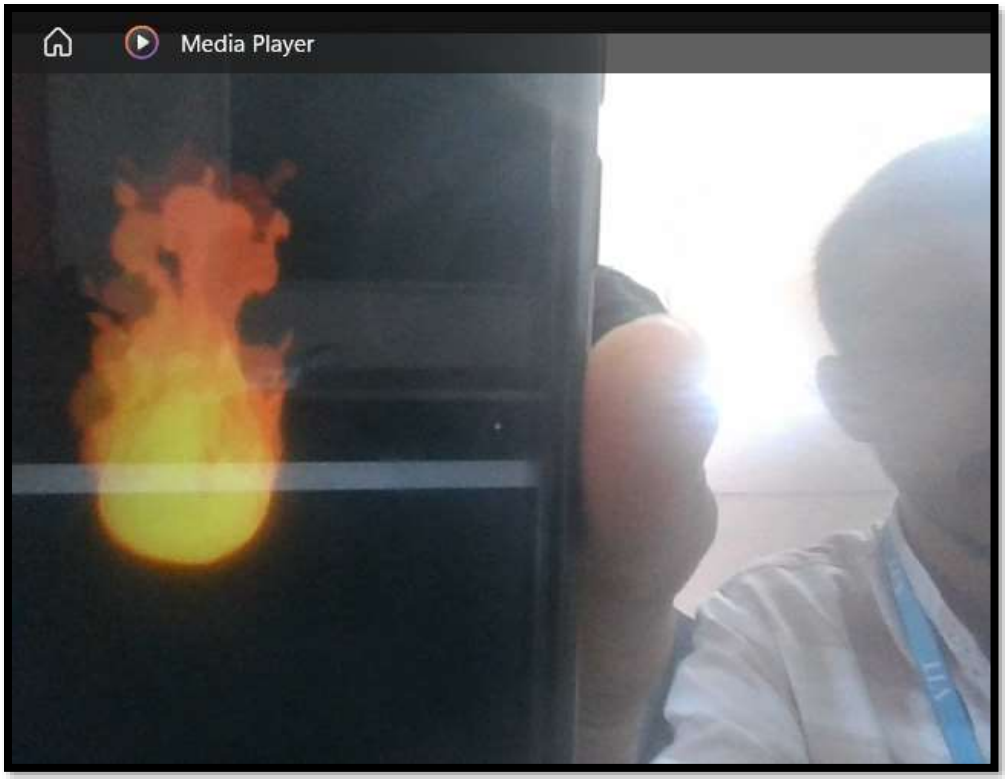
app_tarp.py

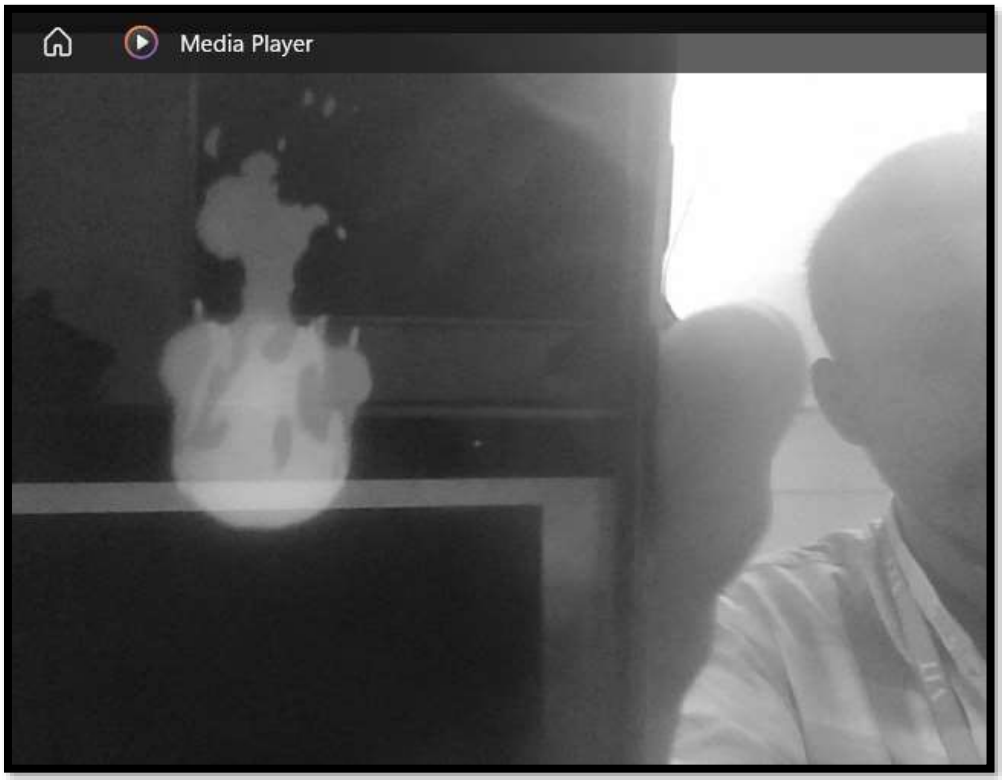
```
import cv2
import numpy as np
from PIL import Image
import tensorflow as tf
from keras.preprocessing import image
#Load the saved model
model = tf.keras.models.load_model(r'C:\Users\laksh\TARP PROJECT\InceptionV3.h5')
video = cv2.VideoCapture(0)
while True:
        _, frame = video.read()
#Convert the captured frame into RGB
        im = Image.fromarray(frame, 'RGB')
#Resizing into 224x224 because we trained the model with this image size.
        im = im.resize((224,224))
        img_array = tf.keras.utils.img_to_array(im)
        img_array = np.expand_dims(img_array, axis=0) / 255
        probabilities = model.predict(img_array)[0]
        #Calling the predict method on model to predict 'fire' on the image
        prediction = np.argmax(probabilities)
        #If prediction is 0, which means there is fire in the frame.
        if prediction == 0:
                frame = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
                print(probabilities[prediction])
        cv2.imshow("Capturing", frame)
        key=cv2.waitKey(1)
        if key == ord('q'):
            break
video.release()
cv2.destroyAllWindows()
```

INITIAL FINDINGS:

Name	Date modified	Type	Size
.ipynb_checkpoints	21-02-2023 09:40	File folder	
FIRE-SMOKE-DATASET	21-02-2023 09:33	File folder	
app_tarp.py	21-02-2023 12:22	PY File	2 KB
InceptionV3.h5	21-02-2023 11:39	H5 File	1,78,762 KB
tarp_inceptionnet.ipynb	21-02-2023 11:55	IPYNB File	282 KB
tarp_inceptionnet.pdf	21-02-2023 12:59	Brave HTML Docu...	272 KB
tarp_main.ipynb	21-02-2023 12:04	IPYNB File	153 KB




Media Player


Media Player

PYTHON FINAL CODE:

```
import cv2          # Library for openCV
import threading   # Library for threading -- which allows code to run in backend
import playsound   # Library for alarm sound
import smtplib     # Library for email sending
import serial
import time

serialPort = serial.Serial(port = "COM4", baudrate=9600, bytesize=8, timeout=2, stopbits=serial.STOPBITS_ONE)
time.sleep(3)



fire_cascade = cv2.CascadeClassifier('fire_detection_cascade_model.xml') # To access xml file which includes positive and negative images of fire. (Trained images)

# File is also provided with the code.

vid = cv2.VideoCapture(0) # To start camera this command is used "0" for laptop inbuilt camera and "1" for USB attahed camera
runOnce = False # created boolean

def play_alarm_sound_function(): # defined function to play alarm post fire detection using threading
```

```
    playsound.playsound('fire_alarm.mp3',True) # to play alarm #
mp3 audio file is also provided with the code.
    print("Fire alarm end") # to print in consol
    serialPort.write(b'S')

def send_mail_function():

    recipientEmail = "Enter_Recipient_Email"
    recipientEmail = recipientEmail.lower()

    try:
        server = smtplib.SMTP('smtp.gmail.com', 587)
        server.ehlo()
        server.starttls()
        server.login("Enter_Your_Email      (System      Email)",
'Enter_Your_Email_Password (System Email')
        server.sendmail('Enter_Your_Email     (System     Email)',
recipientEmail, "Warning A Fire Accident has been reported on ABC
Company")
        print("sent to {}".format(recipientEmail))
        server.close()
    except Exception as e:
      print(e)

while(True):
    Alarm_Status = False
    ret, frame = vid.read() # Value in ret is True # To read video
frame
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) # To convert
frame into gray color
```

```
    fire = fire_cascade.detectMultiScale(frame, 1.2, 5) # to
provide frame resolution

    ## to highlight fire with square
    for (x,y,w,h) in fire:
        cv2.rectangle(frame,(x-20,y-
20),(x+w+20,y+h+20),(255,0,0),2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = frame[y:y+h, x:x+w]

        print("Fire alarm initiated")
        serialPort.write(b'F')      #transmit   'A'   (8bit)   to
micro/Arduino

threading.Thread(target=play_alarm_sound_function).start()   # To
call alarm thread

        if runOnce == False:
            print("Mail send initiated")
            threading.Thread(target=send_mail_function).start() #
To call alarm thread
            runOnce = True
        if runOnce == True:
            print("Mail is already sent once")
            runOnce = True

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

ARUDINO CODE:

```
#include<LiquidCrystal.h>
LiquidCrystal lcd(8, 9, 10, 11, 12, 13);

int BUZZ = A3; //Connect BUZZ  To Pin #7 ///////buzzer
int RLED = A0; //Connect LED  To Pin #A0 ///////led
int GLED = A1; //Connect LED  To Pin #A1 ///////led
int relay = A2; //Connect relay To Pin #6 ///////relay


char data = 0;

void setup()
{
lcd.begin(16, 2);
Serial.begin(9600);
pinMode(BUZZ, OUTPUT);
pinMode(RLED, OUTPUT);
pinMode(GLED, OUTPUT);
pinMode(relay, OUTPUT);

digitalWrite(BUZZ,LOW);
digitalWrite(GLED,LOW);
digitalWrite(relay,LOW);
digitalWrite(RLED,LOW);

lcd.clear();
lcd.setCursor(0,0);lcd.print("Fire Detection");
```

```
lcd.setCursor(0,1);lcd.print("and Alerting");
delay(5000);lcd.clear();
lcd.setCursor(0,0);lcd.print("System Using");
lcd.setCursor(0,1);lcd.print("Arduino- GSM");
delay(5000);lcd.clear();
digitalWrite(GLED,LOW);delay(100);
digitalWrite(BUZZ,LOW);delay(100);
digitalWrite(GLED,HIGH);
lcd.setCursor(0,0);
lcd.print("  NO FIRE    ");
lcd.setCursor(0,1);
lcd.print("  ALAERT     ");
digitalWrite(BUZZ,LOW);delay(100);
digitalWrite(RLED,LOW);delay(100);
digitalWrite(GLED,HIGH);delay(100);
digitalWrite(relay,LOW);delay(100);

}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void loop()
{
st:

if (Serial.available() > 0)
{
data = Serial.read();
////////////////////////////////////
if(data == 'S')
{
lcd.setCursor(0,0);
lcd.print("  NO FIRE    ");
```

```
lcd.setCursor(0,1);
lcd.print("  ALAERT     ");
digitalWrite(BUZZ,LOW);delay(100);
digitalWrite(RLED,LOW);delay(100);
digitalWrite(GLED,HIGH);delay(100);
digitalWrite(relay,LOW);delay(100);
}

if(data == 'F')
{
lcd.setCursor(0,0);
lcd.print("     Fire  ");
lcd.setCursor(0,1);
lcd.print("  Detected  ");
digitalWrite(BUZZ,HIGH);delay(100);
digitalWrite(RLED,HIGH);delay(100);
digitalWrite(GLED,LOW);delay(100);
digitalWrite(relay,HIGH);delay(100);
lcd.clear();lcd.print("Sending SMS ");
delay(500);
Send();
delay(500);
digitalWrite(BUZZ,LOW);digitalWrite(RLED,LOW);
digitalWrite(relay,LOW);delay(100);
lcd.clear();lcd.print("CALLING------1 ");
Serial.print("ATD+91");
Serial.print("7095906007");
Serial.print(";\r\n");delay(12000);
Serial.println("ATH");
delay(2000);
lcd.setCursor(0,0);
```

```
lcd.print("  NO FIRE    ");
lcd.setCursor(0,1);
lcd.print("  ALAERT     ");
digitalWrite(BUZZ,LOW);delay(100);
digitalWrite(RLED,LOW);delay(100);
digitalWrite(GLED,HIGH);delay(100);
digitalWrite(relay,LOW);delay(100);
goto st;
}

}
}


void init_sms1()
{Serial.println("AT+CMGF=1");delay(400);Serial.println("AT+CMGS=\"7095906
007\""); delay(400);}

void Send_sms()
{
Serial.println("Fire Detected");delay(500);
Serial.println("Plz Rescue ");delay(500);
Serial.println("https://www.google.com/maps/place/17.3572096,78.5580032");delay(
500);
Serial.write(26);delay(3000);lcd.clear();
lcd.print("Message Sent ");
delay(1000); lcd.clear();
Serial.print("AT\r\n");delay(500);
Serial.print("AT\r\n");delay(500);
Serial.println("AT+CMGF=1");delay(200);
lcd.clear();
```

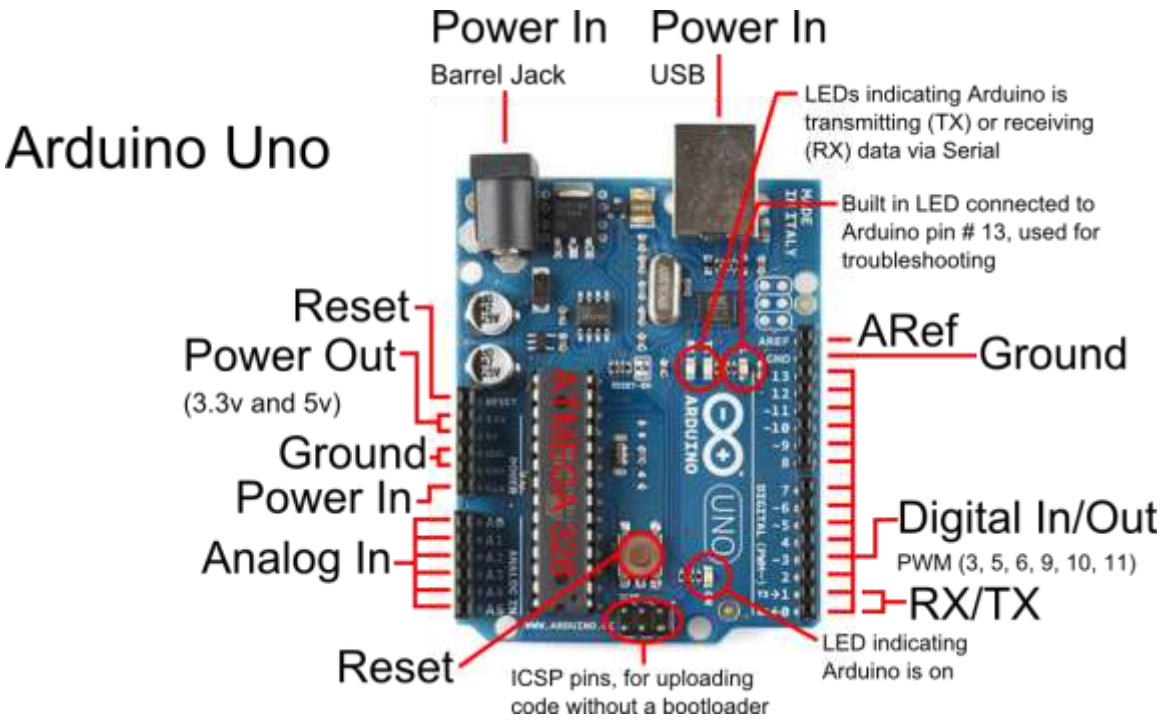
}

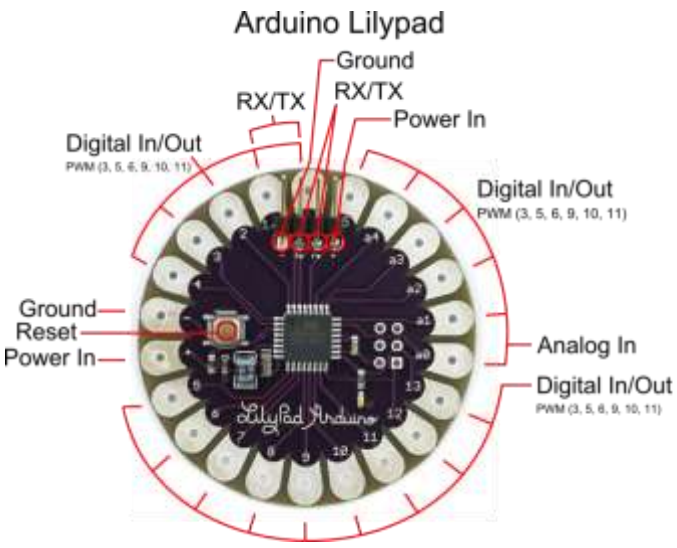
void Send()

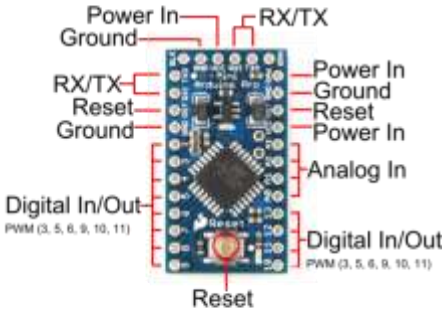
{

init_sms1();delay(1000);lcd.clear();lcd.print("Sending SMS ");Send_sms();

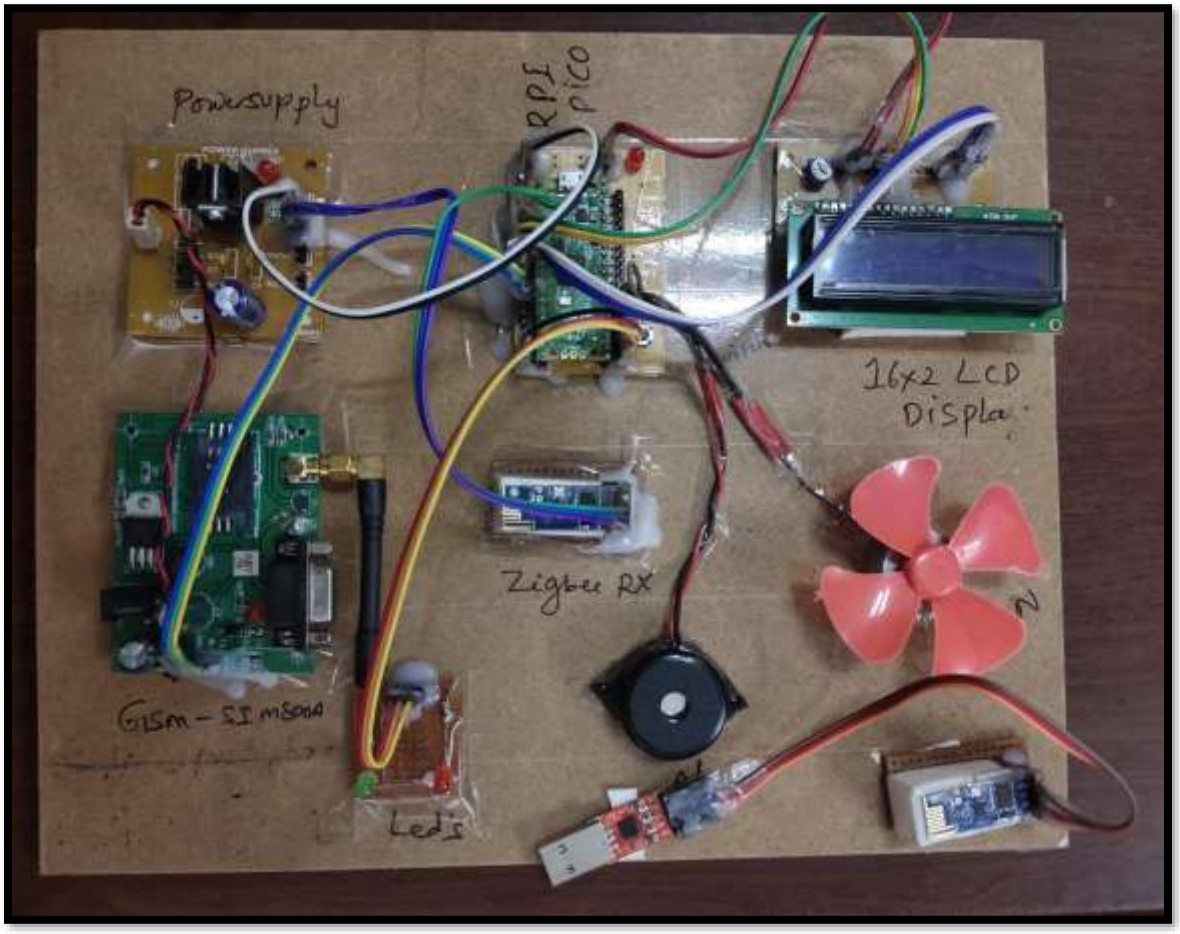
}

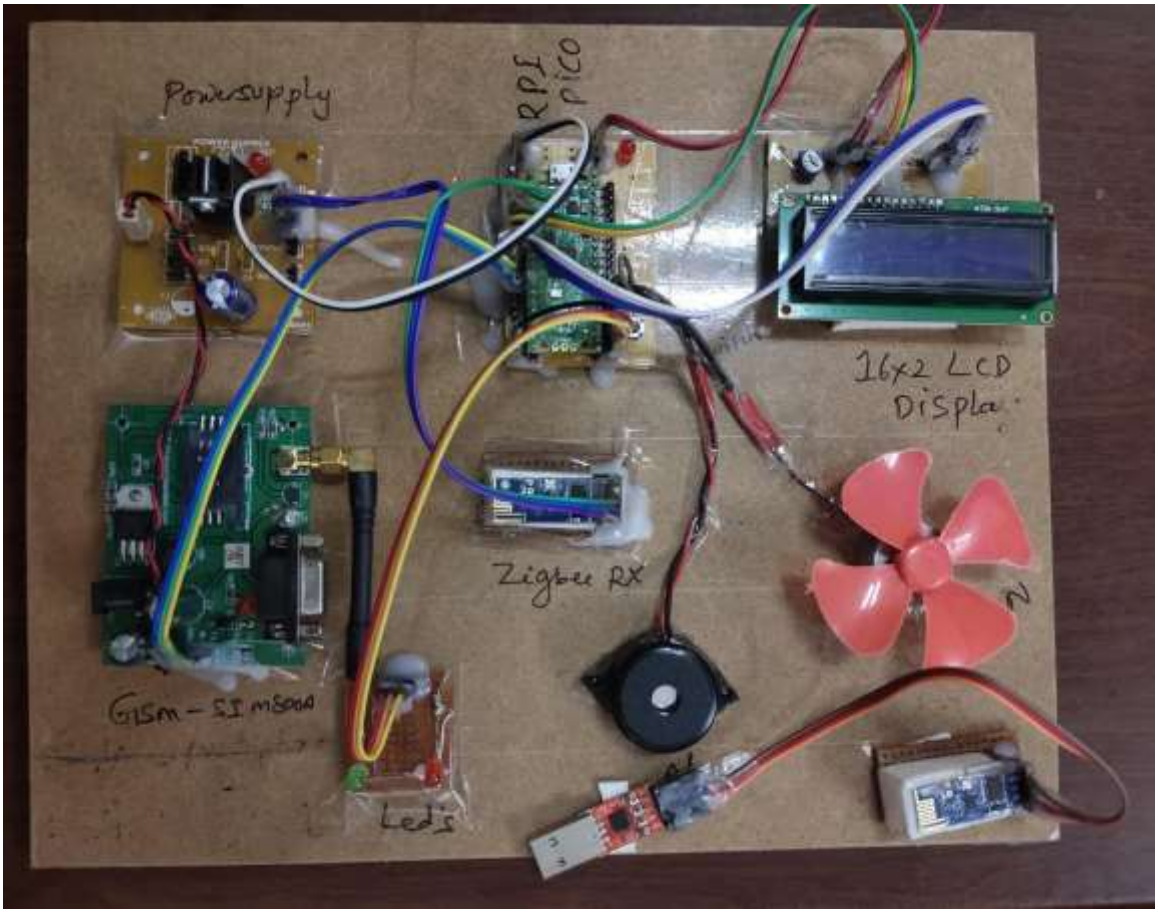

Arduino Uno
Power In
Barrel Jack
Power In
USB
LEDs indicating Arduino is transmitting (TX) or receiving (RX) data via Serial
Built in LED connected to Arduino pin # 13, used for troubleshooting
Reset
Power Out
(3.3v and 5v)
Ground
Power In
Analog In
AREF
Ground
Digital In/Out
PWM (3, 5, 6, 9, 10, 11)
RX/TX
Reset
ICSP pins, for uploading code without a bootloader
LED indicating Arduino is on

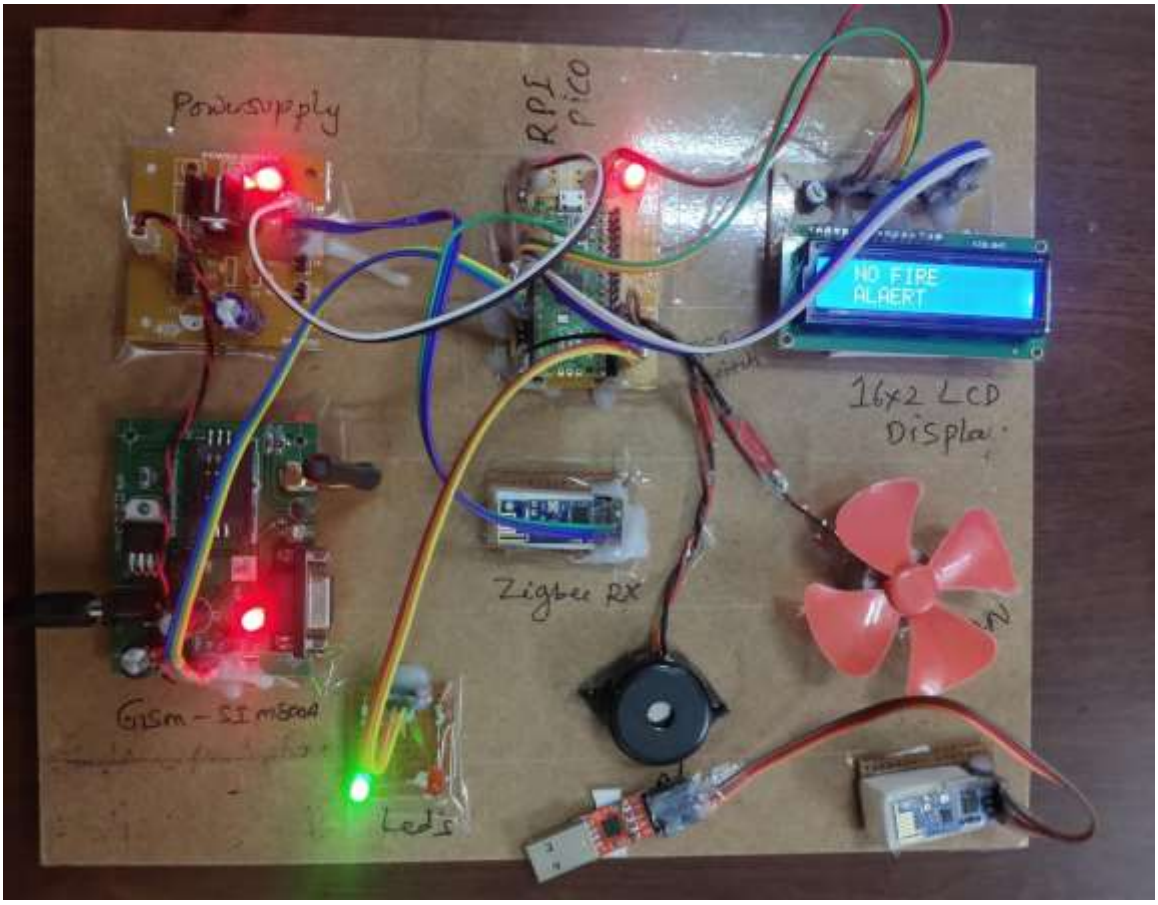

Arduino Lilypad
Ground
RX/TX
RX/TX
Power In
Digital In/Out
PWM (3, 5, 6, 9, 10, 11)
Digital In/Out
PWM (3, 5, 6, 9, 10, 11)
Ground
Reset
Power In
Analog In
Digital In/Out
PWM (3, 5, 6, 9, 10, 11)

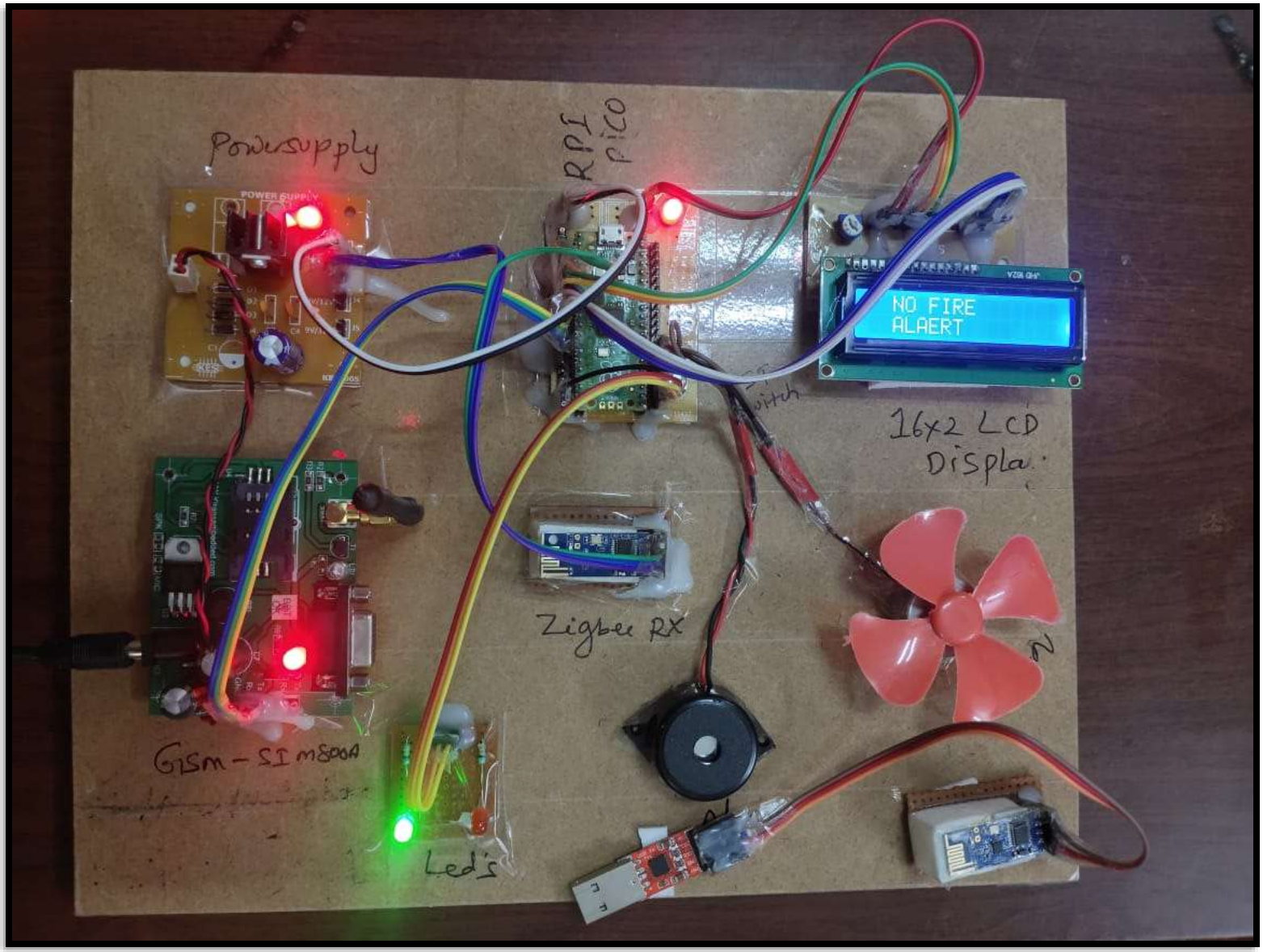

Arduino Mini
Power In
Ground
RX/TX
RX/TX
Reset
Ground
Power In
Ground
Reset
Power In
Analog In
Digital In/Out
PWM (3, 5, 6, 9, 10, 11)
Digital In/Out
PWM (3, 5, 6, 9, 10, 11)
Reset

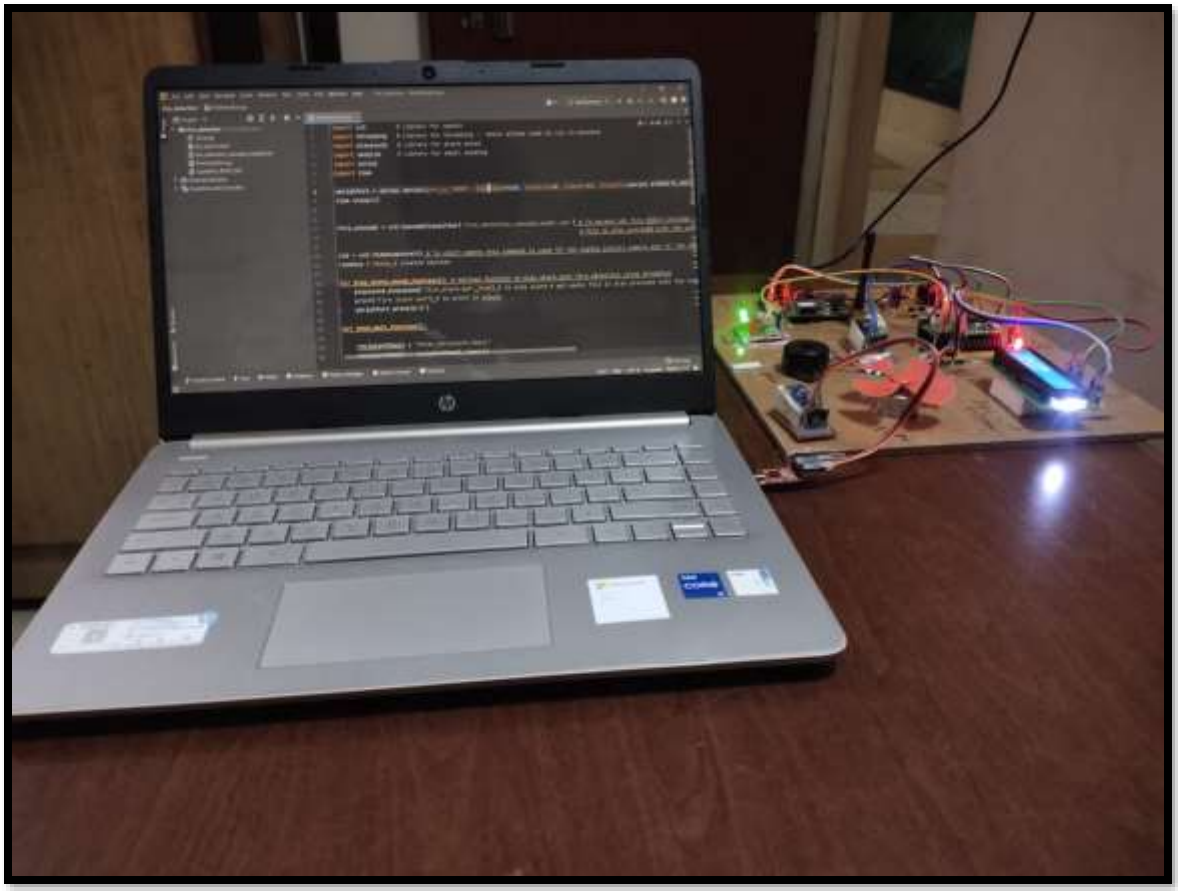
RESULTS SCREENSHOTS ALONG WITH IMPEMENTATION:

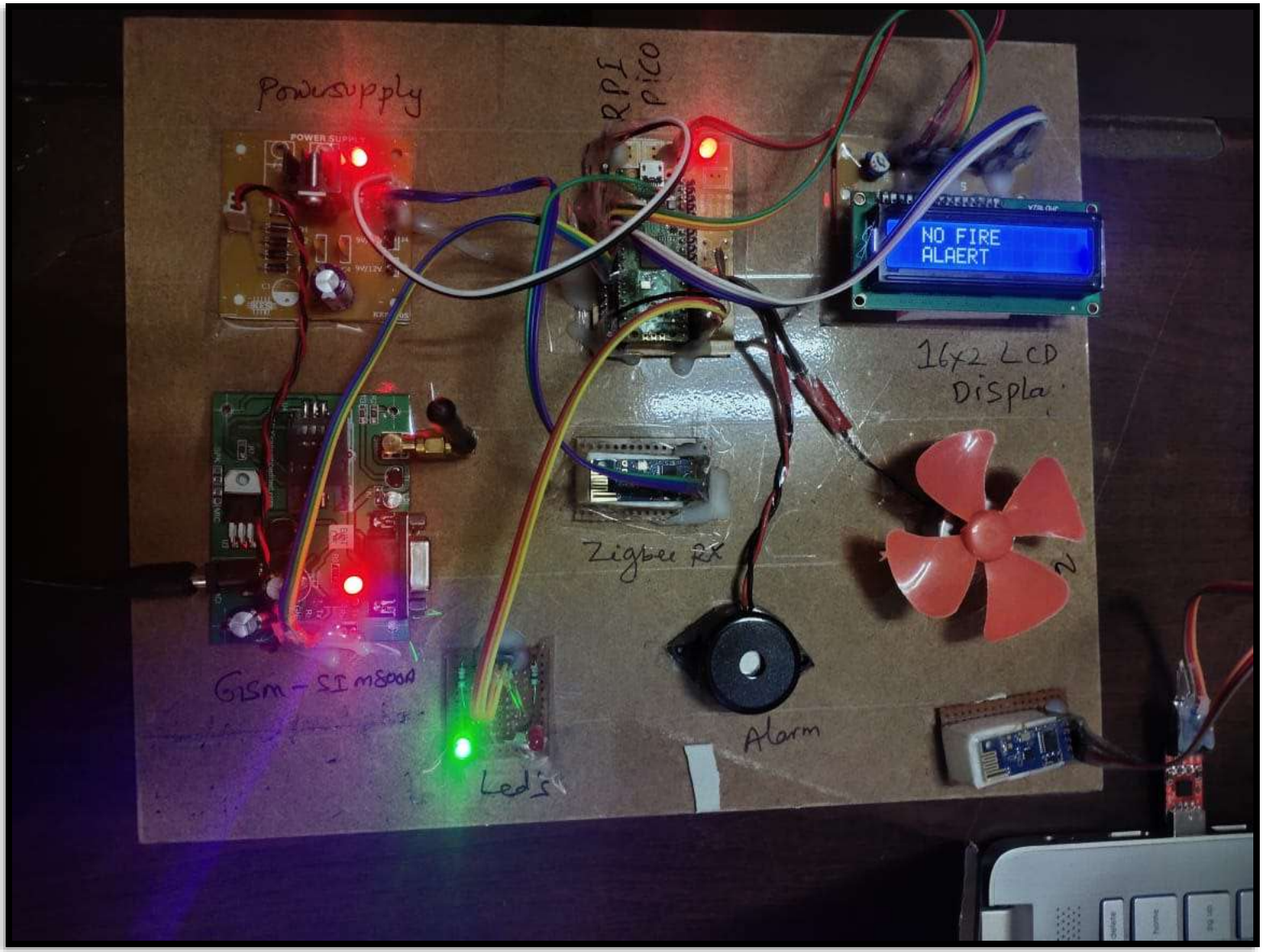


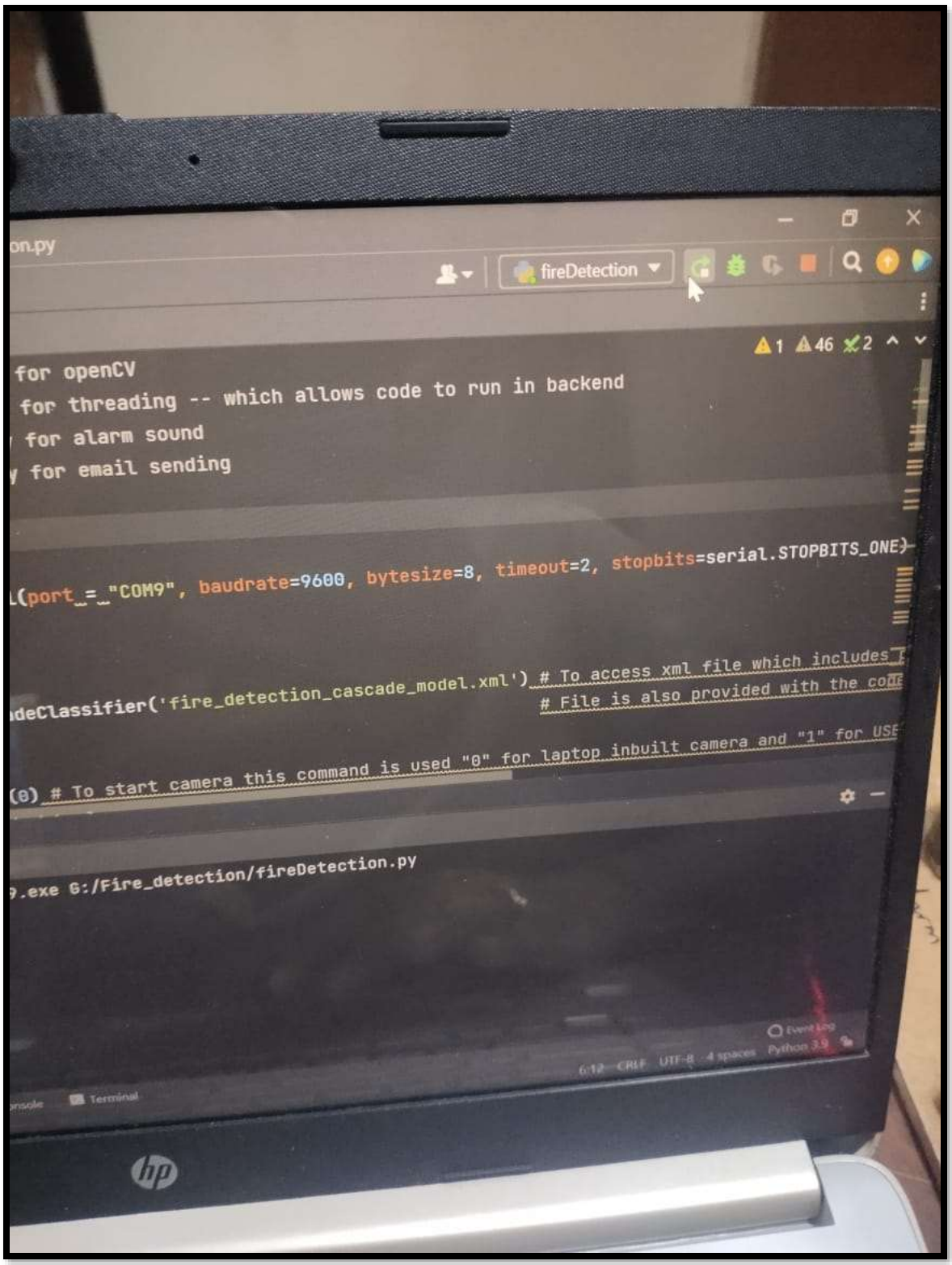

Powersupply
RPI pico
16x2 LCD Displa
Zigbee RX
GSM - SIM800A
Leds

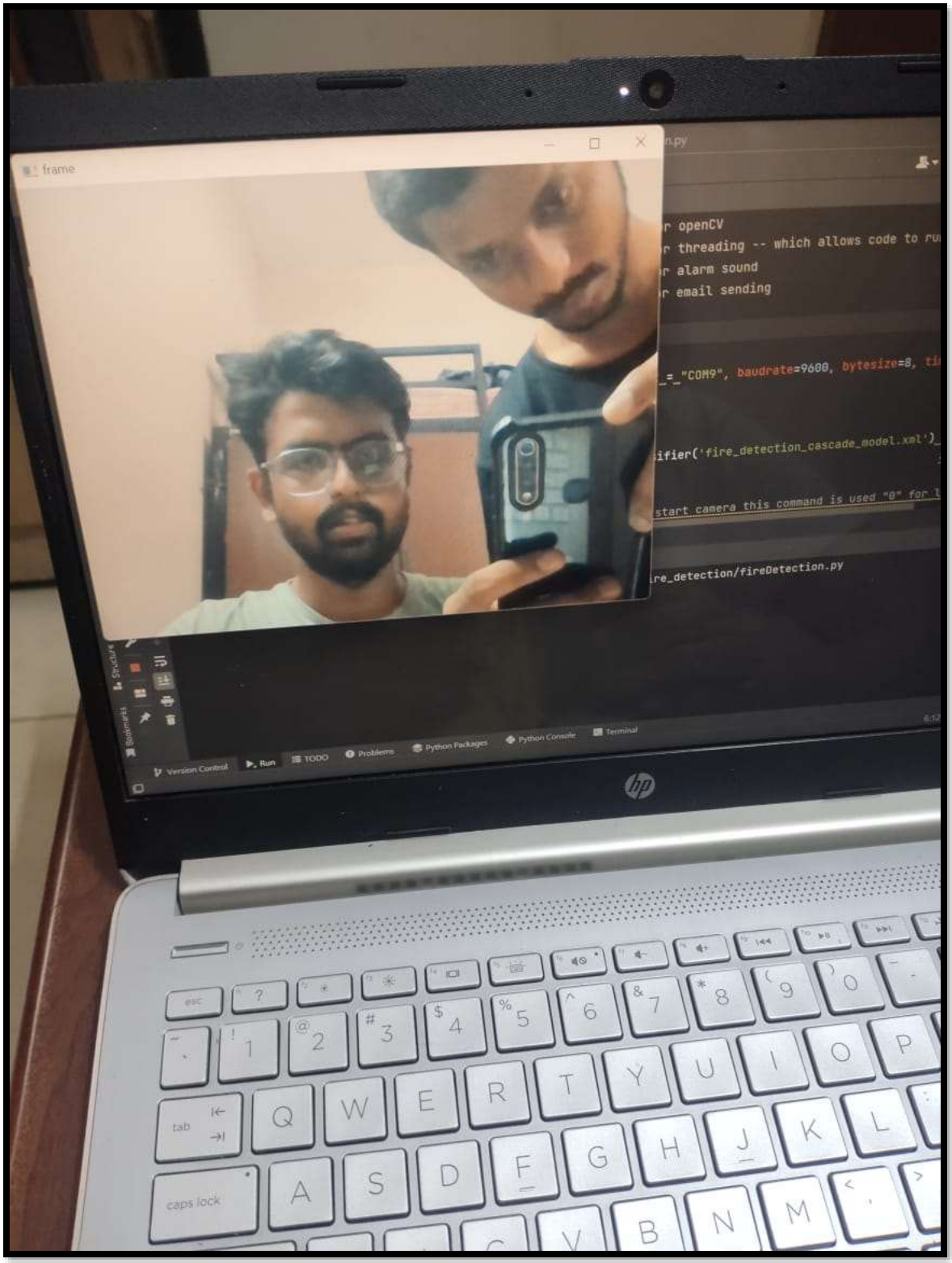

Powersupply
RPI Pico
NO FIRE
ALAERT
16x2 LCD
Displa
Zigbee RX
GSm - SI m800A
Leds

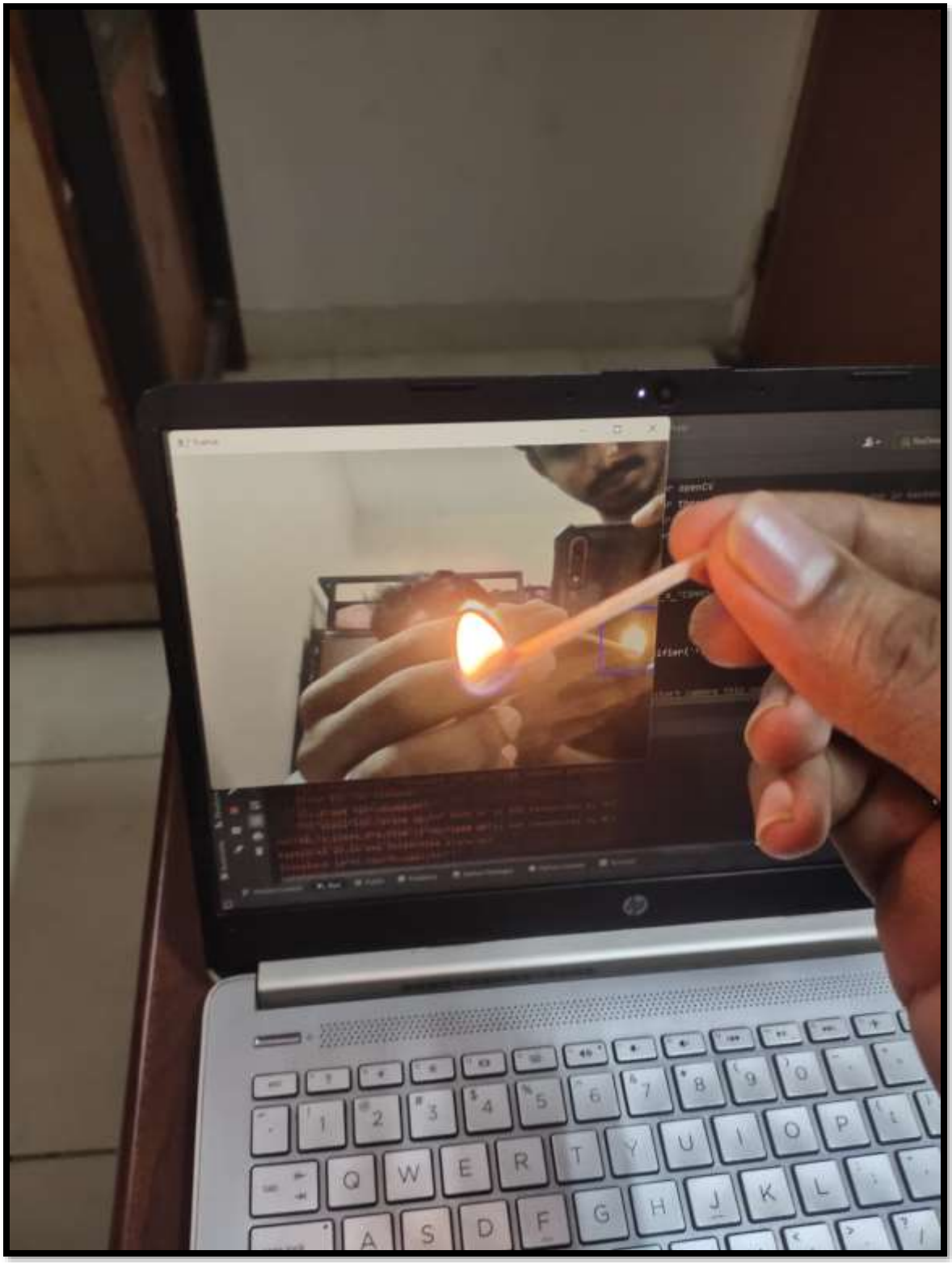

Powersupply
RPI pico
NO FIRE
ALAERT
16x2 LCD
Displa
Zigbee RX
Gsm - SIM800A
Led's

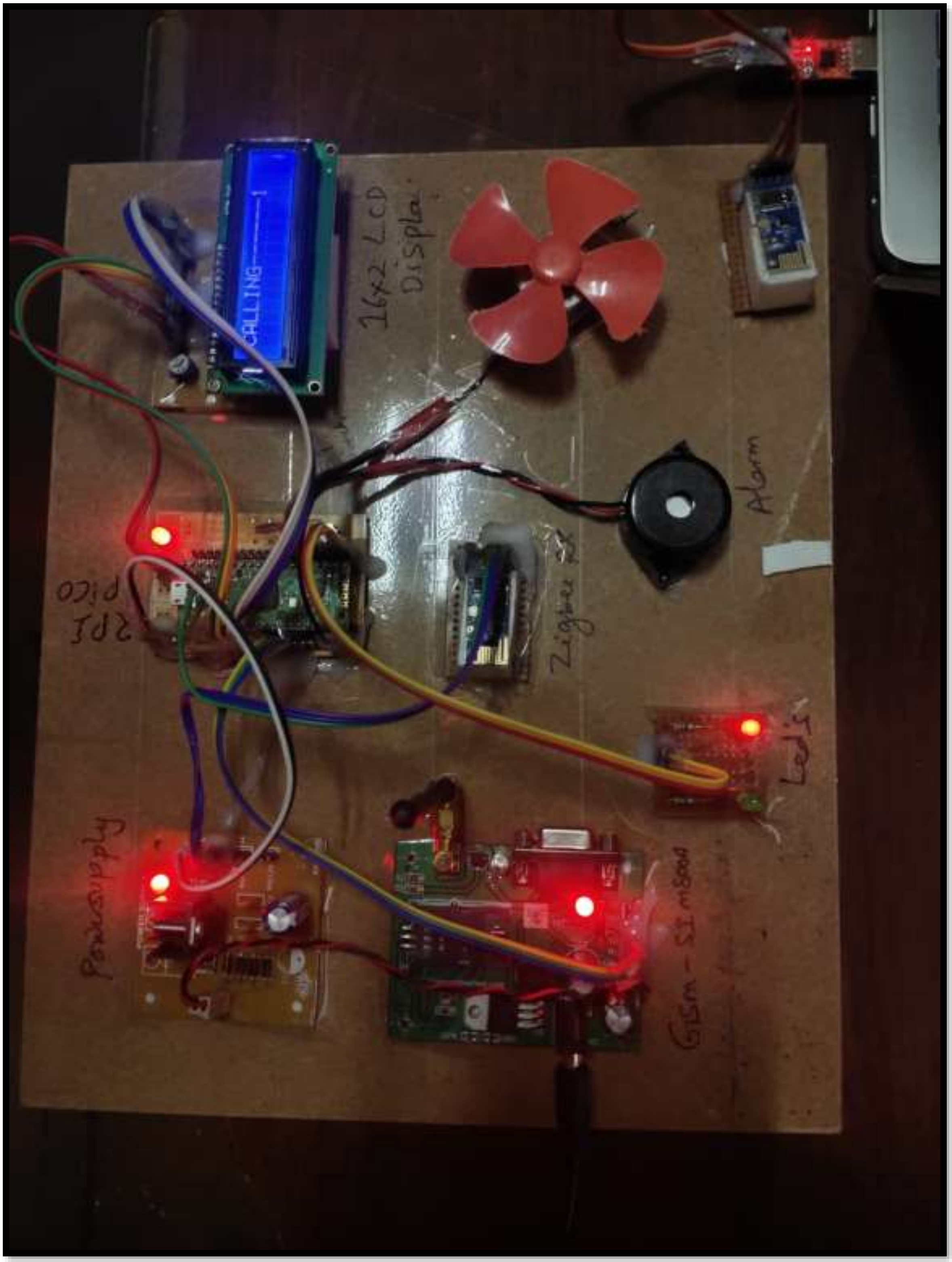


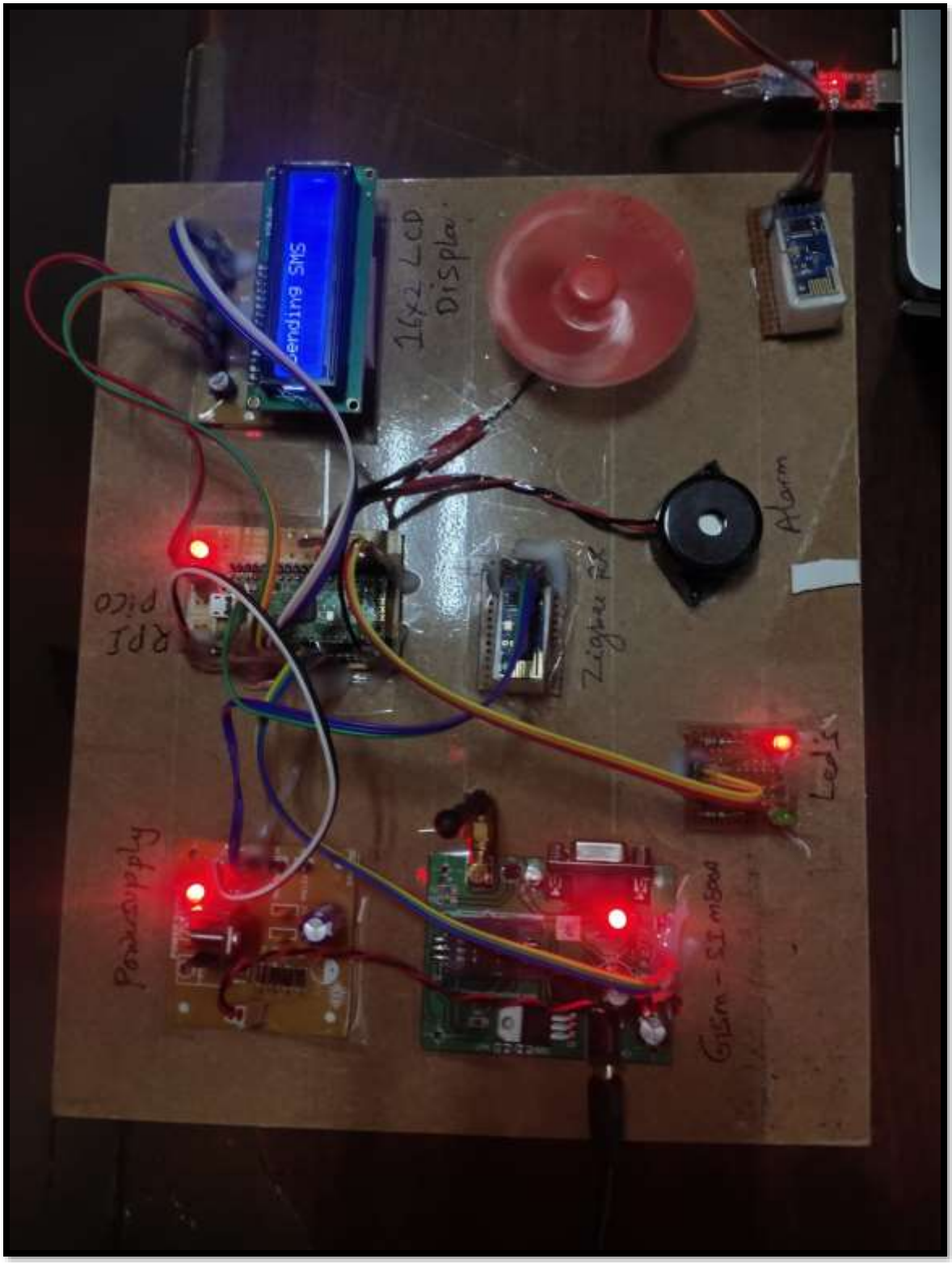

Powersupply
RPI pico
NO FIRE
ALAERT
16x2 LCD
Displa
Zigbee
Gsm - SIm800A
Alarm
Led's

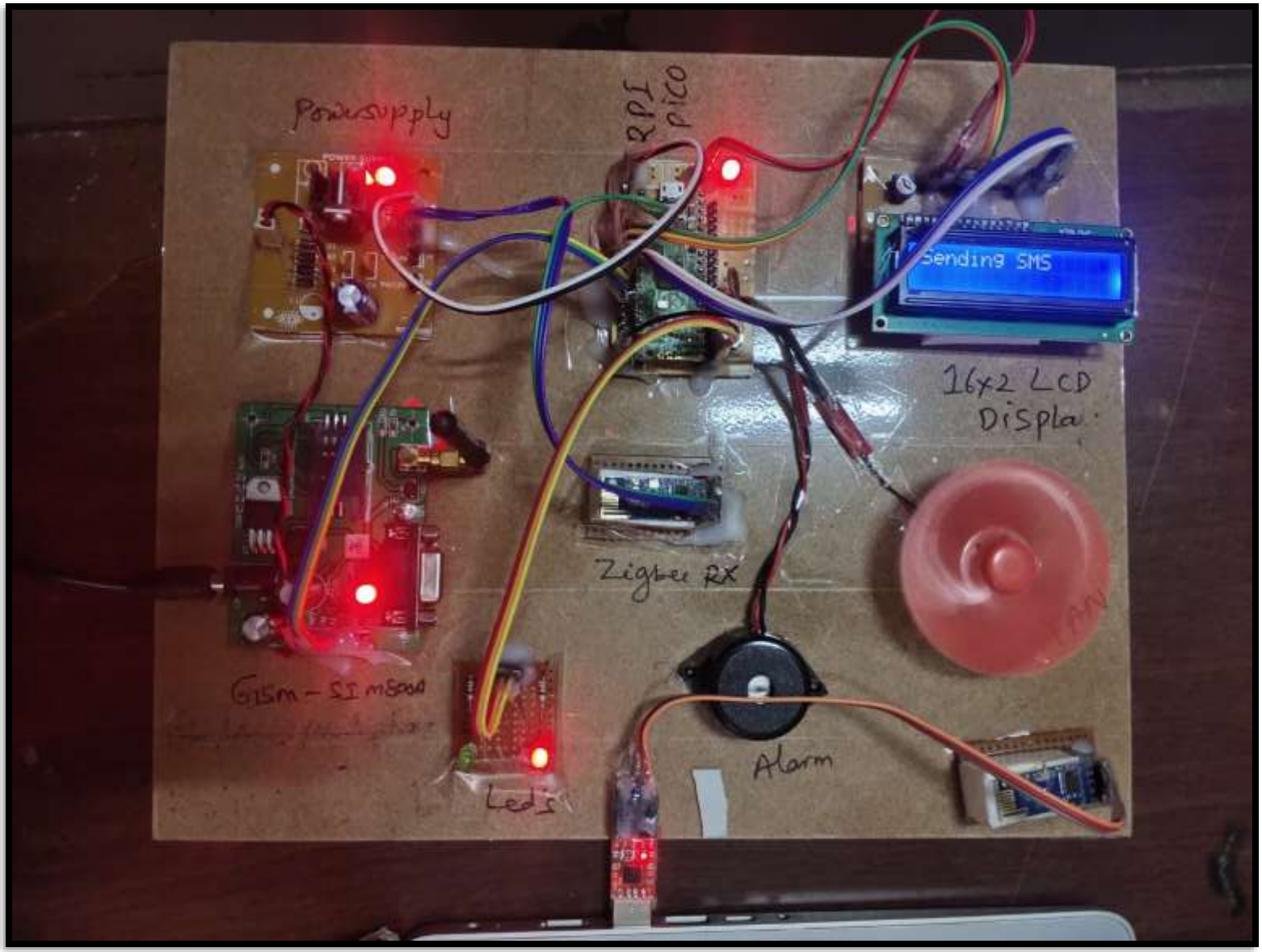

on.py
fireDetection
for openCV
for threading -- which allows code to run in backend
for alarm sound
for email sending
(port_=_"COM9", baudrate=9600, bytesize=8, timeout=2, stopbits=serial.STOPBITS_ONE)
deClassifier('fire_detection_cascade_model.xml') # To access xml file which includes
File is also provided with the co
(0) # To start camera this command is used "0" for laptop inbuilt camera and "1" for USE
.exe G:/Fire_detection/fireDetection.py
Terminal
Python 3.9

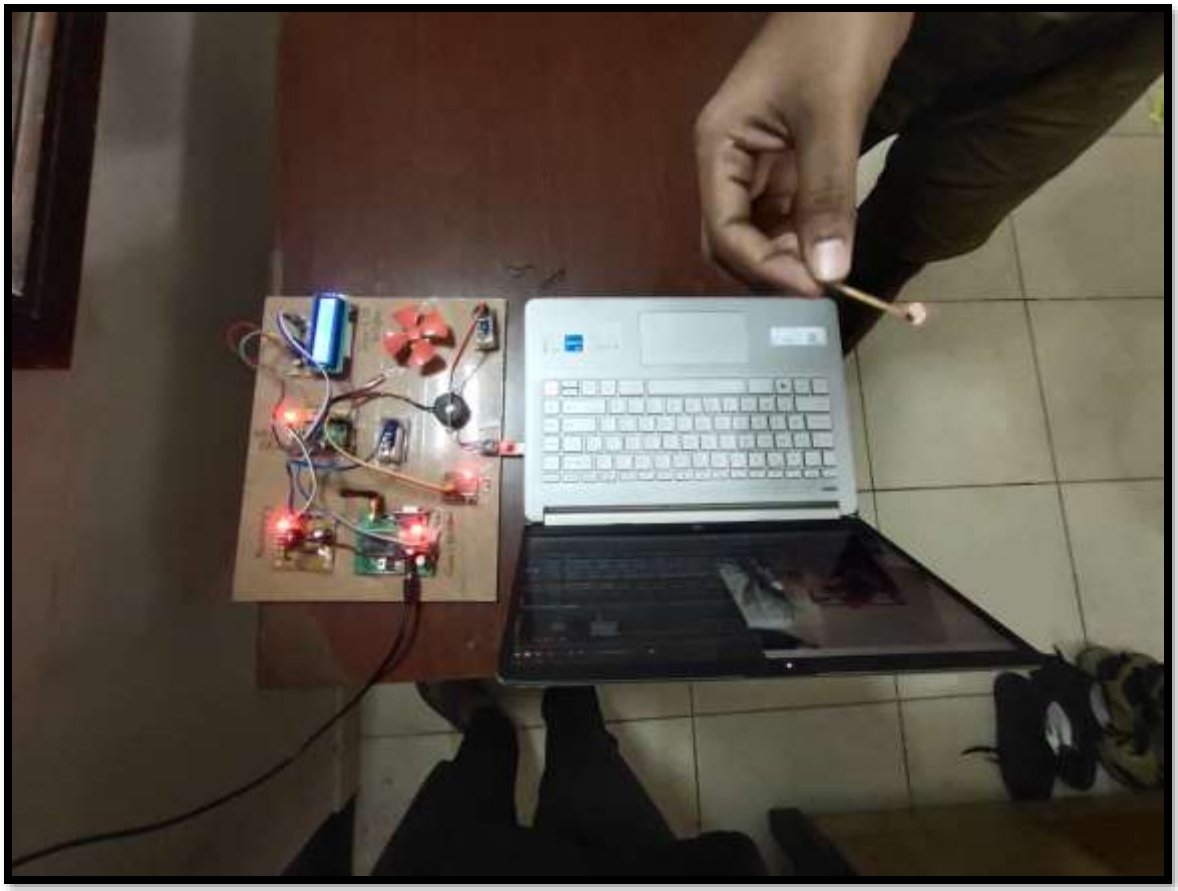

frame
openCV
threading -- which allows code to ru
alarm sound
email sending
"COM9", baudrate=9600, bytesize=8, ti
ifier('fire_detection_cascade_model.xml')
start camera this command is used "0" for l
ire_detection/fireDetection.py
Version Control
Run
TODO
Problems
Python Packages
Python Console
Terminal
Structure
Bookmarks

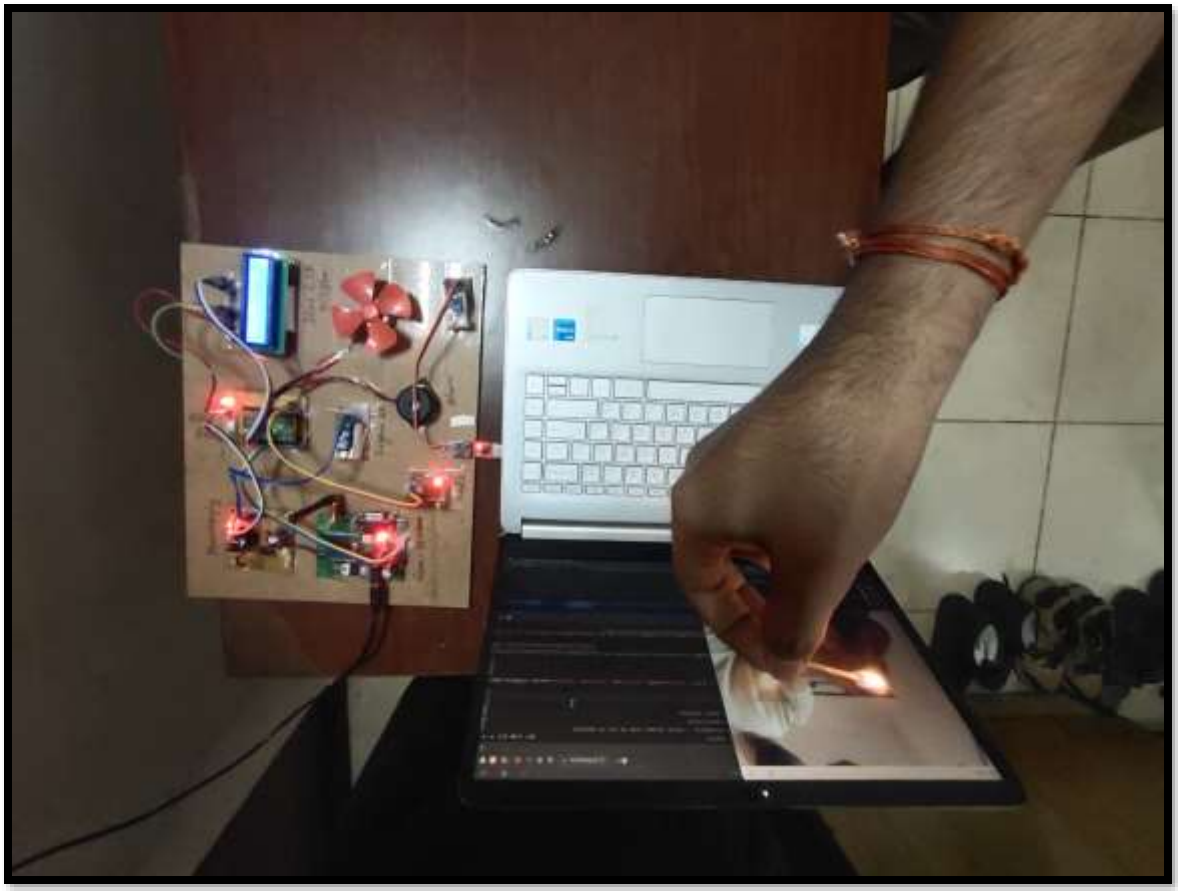


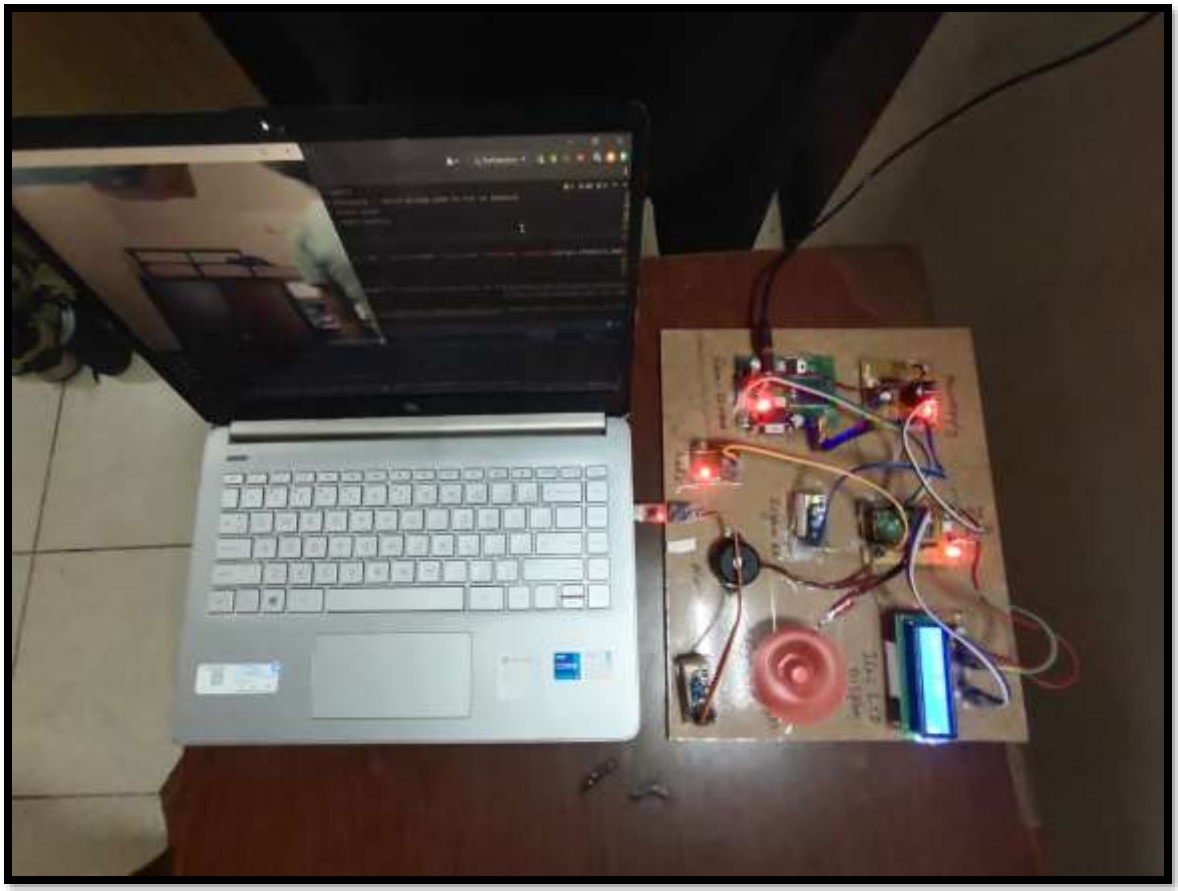

CALLING
16x2 LCD Display
Alarm
Powersupply

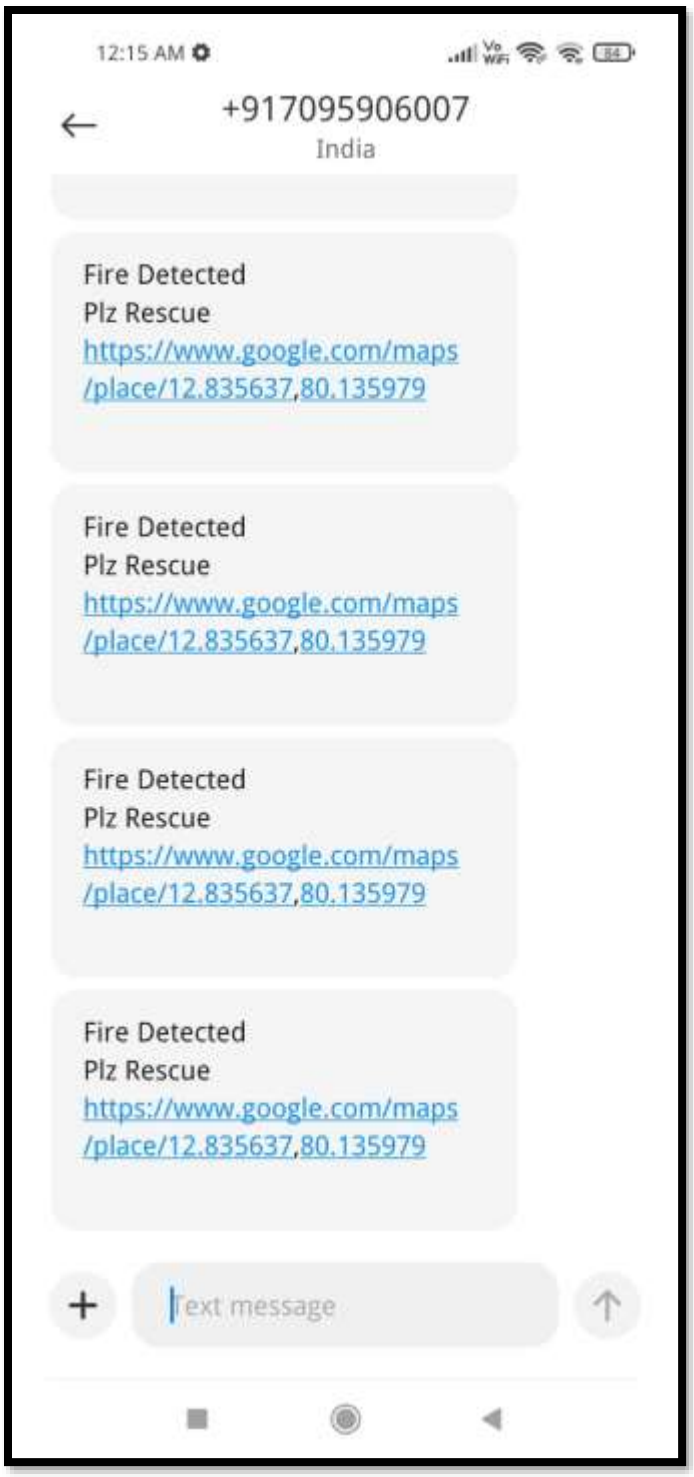

Sending SMS
16x2 LCD
Display
Alarm
RPI
pico
Zigbee
Leds
Powersupply
GSM - SIM800


Powersupply
RPI
pico
Sending SMS
16x2 LCD
Displa
Zigbee RX
GSM - SIM800
Leds
Alarm








12:15 AM
+917095906007
India
Fire Detected
Plz Rescue
https://www.google.com/maps
/place/12.835637,80.135979
Fire Detected
Plz Rescue
https://www.google.com/maps
/place/12.835637,80.135979
Fire Detected
Plz Rescue
https://www.google.com/maps
/place/12.835637,80.135979
Fire Detected
Plz Rescue
https://www.google.com/maps
/place/12.835637,80.135979
Text message

THE END