SMART ELEVATOR SYSTEMS USING EMBEDDED MACHINE LEARNING AND FIRE SAFETY MECHANISMS

Micro-Project Report

by

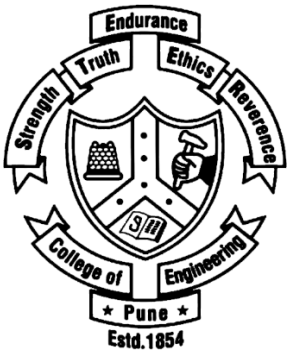
Anway Pimpalkar **111907066**

Esha Dorle **111907070**

Sayali Gadre **111907079**

Under the guidance of

Prof. Deeplakshi Niture



DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION ENGINEERING,

COLLEGE OF ENGINEERING, PUNE

2020 – 2021

Abstract

This report explores the possibilities of elevator control systems using Arduino. With tremendous developments in architectural engineering, the installation of elevators has become an integral part of the infrastructure for the vertical movement. Hence for more efficient performance and maintenance, more importance is given to the design of an elevator control system. We have made the use of various components for upgrading the elevator which enables the elevator.

Using an approach of Embedded Machine Learning, we built a multi-tenant architecture to facilitate a contactless elevator system. It consists of two parts – person detection and speech recognition. To create an effective pipeline, we used multiple signal processing techniques to extract necessary features from the input signals. Further, we developed a fire safety system using basic yet functional sensors in two methods – using a microcontroller and using an operational amplifier. These systems are deployed outside and inside an elevator, respectively. Lastly, we built a PCB to house all these components and make a holistically deployable product.

Contents

List of Figures

List of Tables

Abbreviations

CNN: Convolutional Neural Network

KWS: Keyword Spotting

VWW: Visual Wake Word

PCB: Printed Circuit Board

1. Introduction

Elevator, also called lift, car that moves in a vertical shaft to carry passengers or freight between the levels of a multistory building. It is a platform that could either be open or closed and is used for lifting or lowering both people and goods to upper and lower floors.

Structurally, elevators have not changed much since the 1800s. However, their control systems have been altered for modern elevators in ways that improve on speed and safety. Modern elevator systems have added technology installed in them. Some have phones that allow the occupant to call for help in the event of an emergency. Others are fitted with a trap door located at the ceiling that make escape possible in emergency situations.

Even though elevators still maintain their original purpose of transporting people across floors, these upgrades that exist in the status quo have agreeably increased the quality and the general user experience of these elevators.

1.1 Literature Review

Studies have shown that a significant part of the population believes elevators are unreliable and hold certain levels of uncertainty especially during emergencies. Additionally, elevators are not recommended during fire emergencies, which puts the people inside and their safety at a huge risk, the lack of automation is also a big risk factor, particularly during a time of a public health emergency as we are experiencing. To conclude, the current elevator technologies are not intuitive and cognizant of the risk that they may cause to the users and therefore, certain aspects need to be further developed.

The improvements and upgrades we propose are two-fold. One aspect is incorporating a smoke detector system inside and outside the elevator. This subsystem curbs the risk of people getting stuck inside an elevator to a large extent since it informs the users about smoke being in immediate

vicinity and gives them two things based on where the user is:

1. Ample time to get out of the elevator safely if the user is inside.
2. Get help for the people inside if the user is outside.

The second part is a person and speech detector in the elevator system. This subsystem allows a person to verbally communicate to the elevator instead of pressing buttons. This feature is exceptionally useful in the current paradigm, where social distancing and sanitization are quintessential for one's safety.

Therefore, our system not only maximizes sanitization, but it also ensures user protection to its apex. Our project is a simple but a significant change to the way things work in the status quo by the increasing the efficiency and safety of the current elevators in use.

1.2 Aim

To ideate and model a safer and smarter elevator system using a device which can be deployed to pre-existing elevators.

1.3 Objectives

1. To ensure a contactless and thus hygienic elevator operation.
2. To ensure safety from fire hazards by using smoke detectors and temperature sensors inside and outside the elevator
3. To understand, analyze and apply the working of different types of sensors
4. To learn effective and functional circuit designing.
5. To understand the crucial memory constraints of a given microcontroller and development environment and attain maximum benefits from the limited architecture.
6. To understand the difference of using pure electronic circuits to achieve a given output versus using a hybrid hardware/software development platform such as Arduino to achieve the same.
7. To understand the types of PCBs and develop our own PCB for the given project.
8. To understand how to apply the gained knowledge to create a holistic deployable product.

2. Theoretical Background

To form a good basis for our project, we thoroughly studied the concepts and theoretical background required to efficiently execute our project. On a broad scale, these concepts include the functioning of various sensors, understanding of some basic concepts in electronics, usage of various signal processing techniques and a high level understanding of embedded machine learning. In this section, we will briefly go through all the key concepts used in this project.

2.1 Peripheral Components

In this project, we have used two main sensors for executing the fire alarm systems – a smoke detector and a LM-35 temperature sensor. The workings of these sensors are briefed.

2.1.1 MQ-2 Smoke and Flammable Gas Detector

MQ2 is one of the commonly used gas sensors in MQ sensor series. It is a Metal Oxide Semiconductor (MOS) type Gas Sensor also known as Chemiresistors as the detection is based upon change of resistance of the sensing material when the Gas comes in contact with the material. Using a simple voltage divider network, concentrations of gas can be detected. It can be used for sensing the concentration of gases in the air such as LPG, propane, methane, hydrogen, alcohol, smoke, and carbon monoxide.

Figure 2.1: Outer build of MQ-2 Gas Sensor (Courtesy: www.lastminuteengineers.com)

It also provides protection for the sensor and filters out suspended particles so that only gaseous elements are able to pass inside the chamber. The mesh is bound to rest of the body via a copper plated clamping ring.

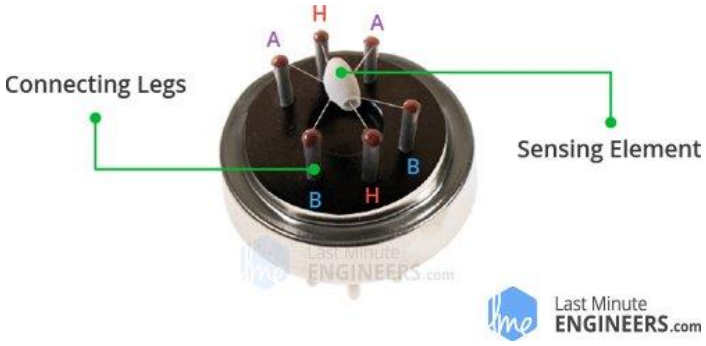


Figure 2.2: Sensor build with outer mesh removed (Courtesy: www.lastminuteengineers.com).

The tubular sensing element is made up of Aluminum Oxide (AL2O3) based ceramic and has a coating of Tin Dioxide (SnO2). The Tin Dioxide is the most important material being sensitive towards combustible gases. However, the ceramic substrate merely increases heating efficiency and ensures the sensor area is heated to a working temperature constantly.

When tin dioxide (semiconductor particles) is heated in air at high temperature, oxygen is adsorbed on the surface. In clean air, donor electrons in tin dioxide are attracted toward oxygen which is adsorbed on the surface of the sensing material. This prevents electric current flow.

In the presence of reducing gases, the surface density of adsorbed oxygen decreases as it reacts with the reducing gases. Electrons are then released into the tin dioxide, allowing current to flow freely through the sensor.

The analog output voltage provided by the sensor changes in proportional to the concentration of smoke/gas. The greater the gas concentration, the higher is the output voltage, while lesser gas concentration results in low output voltage.

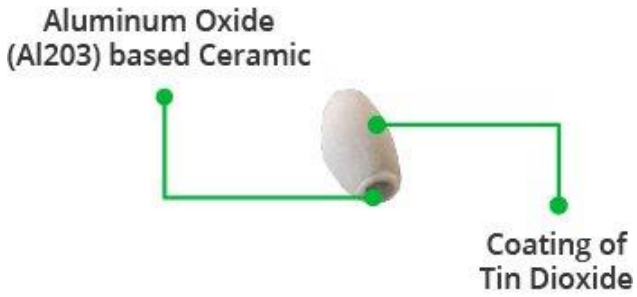


Figure 2.3: Tubular Sensing element inside MQ-2 (Courtesy: www.lastminuteengineers.com).

2.1.2 LM-35 Temperature Sensor

LM35 is a temperature sensor that outputs an analog signal which is proportional to the instantaneous temperature. The output voltage can easily be interpreted to obtain a temperature reading in Celsius. The advantage of lm35 over thermistor is it does not require any external calibration. The coating also protects it from self-heating. LM35 can measure from -55 degrees centigrade to 150-degree centigrade. The accuracy level is very high if operated at optimal temperature and humidity levels. The conversion of the output voltage to centigrade is also easy and straight forward.

LM-35's output is 10mV/^{o}C which means for every degree rise in temperature the output of LM-35 will rise by 10mV. LM35 can be operated from a 5V supply and the stand by current is less than 60uA. 6.The working principle of the temperature sensor high temperatures-Analog temperature sensor Temperature sensor high temperatures adopt the digital temperature sensor produced by silicon process, which adopts PTAT structure, which has the precision and temperature related good output characteristics. As the temperature increase the ADC value will also increase.

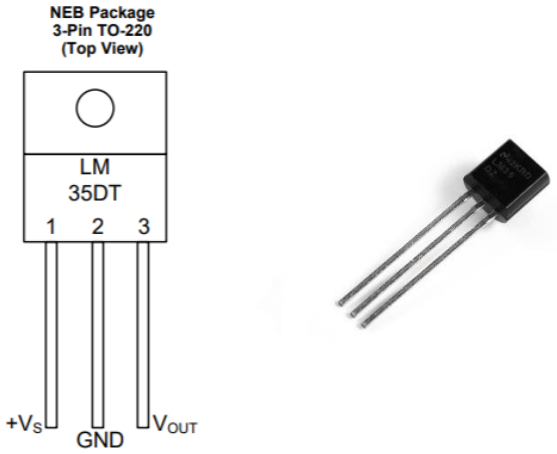


Figure 2.4 NEB Package of LM-35 Temperature sensor (Courtesy: Texas Instruments).

2.1.3 Shift Registers

Shift Registers are used for data storage or for the movement of data. This sequential device loads the data present on its inputs and then moves or “shifts” it to its output once every clock cycle, hence the name Shift Register.

Shift registers hold the data in their memory which is moved or “shifted” to their required positions on each clock pulse. Each clock pulse shifts the contents of the register one bit position to either the left or the right.

Data bits may be fed in or out of a shift register serially, that is one after the other from either the left or the right direction, or all together at the same time in a parallel configuration.

Let's assume that all the flip-flops (FFA to FFD) have just been RESET (CLEAR input) and that all the outputs Q_A to Q_D are at logic level "0" i.e., no parallel data output.

If a logic "1" is connected to the DATA input pin of FFA then on the first clock pulse the output of FFA and therefore the resulting Q_A will be set HIGH to logic "1" with all the other outputs still remaining LOW at logic "0". Assume now that the DATA input pin of FFA has returned LOW again to logic "0" giving us one data pulse or 0-1-0.

The second clock pulse will change the output of FFA to logic "0" and the output of FFB and Q_B HIGH to logic "1" as its input D has the logic "1" level on it from Q_A. The logic "1" has now moved or been "shifted" one place along the register to the right as it is now at Q_A.

When the third clock pulse arrives this logic "1" value moves to the output of FFC (Q_C) and so on until the arrival of the fifth clock pulse which sets all the outputs Q_A to Q_D back again to logic level "0" because the input to FFA has remained constant at logic level "0".

The effect of each clock pulse is to shift the data contents of each stage one place to the right, and this is shown in the following table until the complete data value of 0-0-0-1 is stored in the register. This data value can now be read directly from the outputs of Q_A to Q_D.

74HC595 IC is a 16-pin shift register IC consisting of a D-type latch along with a shift register inside the chip. It receives serial input data and then sends out this data through parallel pins. In addition to parallel outputs, it also provides a serial output. It has independent clock inputs for shift register and D latch. This IC belongs to the HC family of logic devices which is designed for use in CMOS applications.

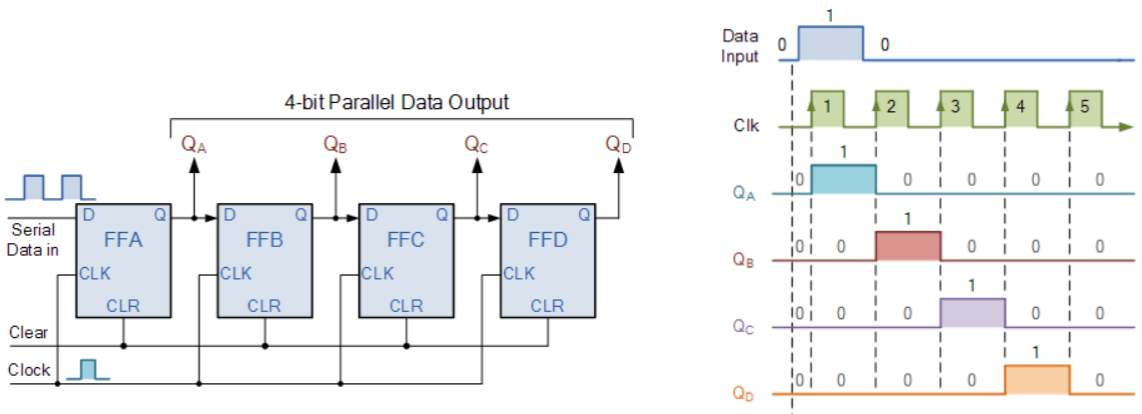


Figure 2.5: Working of Shift Registers

2.1.4 OV7675 Camera Module

The OV7675 is a low-voltage color CMOS image sensor that supports the full functionality of a single chip VGA (640x480) camera in a small footprint package. It provides full-frame, sub-sampled, windowed images in VGA, QVGA and QQVGA formats via the control of the serial camera control bus (SCCB) interface. Its image array is capable of operating at up to 30 frames per second (FPS) in full VGA resolution with complete user control over image quality, formatting and output data transfer. All required image processing functions, including exposure control, gamma, white balance, color saturation, hue control, defective pixel canceling, noise canceling are programmable through the SCCB interface.

Figure 2.6: OV7675 Camera Module (Courtesy: AskElectronics).

2.2 Comparator Circuit

The comparator is an electronic decision making circuit that makes use of an operational amplifiers very high gain in its open-loop state, that is, there is no feedback resistor.

The Op-Amp comparator compares one analog voltage level with another analog voltage level, or some preset reference voltage, V_{ref} and produces an output signal based on this voltage comparison. In other words, the op-amp voltage comparator compares the magnitudes of two voltage inputs and determines which is the largest of the two.

Op-Amp in open loop mode acts as a comparator which gives output $\pm V_{sat}$.

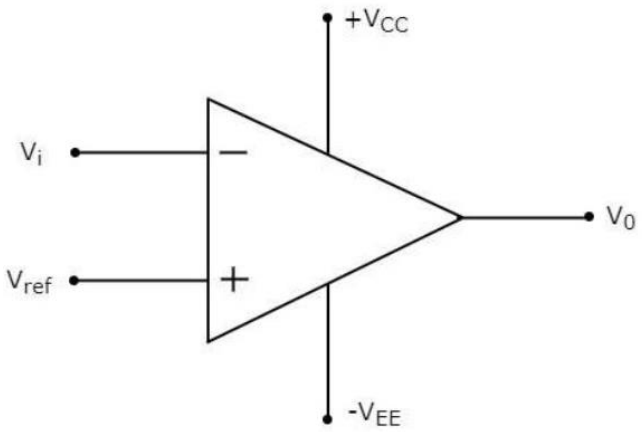


Figure 2.7: Skeleton circuit of Comparator using Operational Amplifier.

Specific to our project, MQ-2 sensor contains a sensing material whose resistance changes when it encounters the gas. This change in the value of resistance is used for the detection of gas. MQ-2 is a metal oxide semiconductor type gas sensor. Concentrations of gas in the gas is measured using a voltage divider network present in the sensor. This output given by MQ-2 sensor is given to the comparator at its noninverting terminal.

In this case, V_- is given to a potentiometer which acts as V_{ref} and V_+ is the output of the gas sensor MQ-2. Vo of the op amp is connected to an led which will glow if the output is high i.e., $+V_{sat}$.

Now, if the output of MQ-2 sensor is higher than the set reference voltage, then , $V_+ > V_-$ and the output of comparator will be high so the LED glows, indicating a warning. On the other hand, when the output of smoke detector is lower than the reference value that we have set, V_+ will be less than V_- , $V_o = -V_{sat}$ and the LED will not glow.

This is how the comparator compares the output of MQ-2 sensor with that of the reference voltage and gives a signal when gas sensor output has exceeded a set value.

2.3 Feature Extraction in Signal Processing

In our project, to implement speech recognition on our Arduino, we have used some key concepts of signal processing to extract features from a microphone signal and feed to our Machine Learning model.

2.3.1 Fast Fourier Transform (FFT)

Fast Fourier Transform is an algorithm that computes the Discrete Fourier Transform of a given signal. Fourier analysis converts a signal from the time domain to a representation in the frequency domain. The general equation for the Discrete Fourier Transform is given as:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{j2\pi kn}{N}} \tag{2.1}$$

This operation is useful in many fields but computing it directly from the definition is often too slow to be practical. Hence, we implement the Cooley-Turkey algorithm for faster computations. It re-expresses the discrete Fourier transform (DFT) of an arbitrary composite size $N = N_1N_2$ in terms of N_1 smaller DFTs of sizes N_2, recursively, to reduce the computation time to *O(N log N)* for highly composite N.

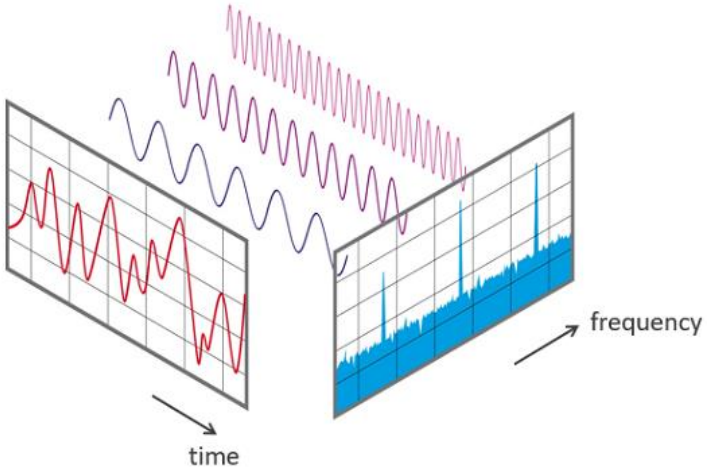


Figure 2.8: Fast Fourier Transform (Courtesy: www.nti-audio.com).

2.3.2 Spectrogram Analysis

A spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time. They are a plot of the intensity of the frequency content of the signal as time progresses. At a high-level understanding, a spectrogram is a 3-dimensional representation of a signal, the three dimensions representing time, frequency, and intensity. The frequency and amplitude axes can be either linear or logarithmic, depending on what the graph is being used for.

A spectrogram uses a color scale to depict the amplitude of a given frequency at a given point in time. This color scale is not fixed and can vary for different applications.

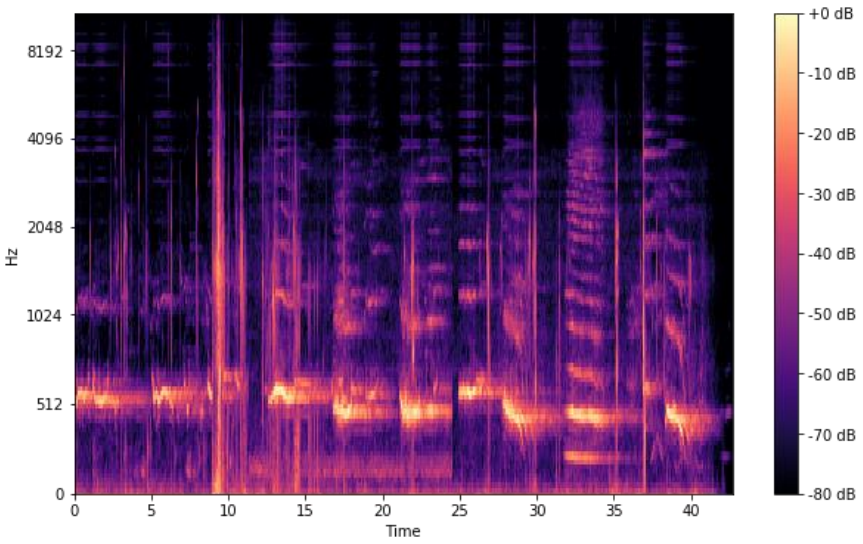


Figure 2.9: Sample spectrogram with a color legend (Courtesy: www.towardsdatascience.com)

2.3.3 Mel-Scale and MFCCs

When working on applications such as speech recognition, we must be extremely efficient in terms of the data we train with to attain a good accuracy with our model. The main point to understand about speech is that the sounds generated by a human are filtered by the shape of the vocal tract including tongue, teeth etc. This shape determines what sound comes out. If we can determine the factors accurately, this should give us an accurate representation of the phoneme being produced. Hence, through evolution, humans are much better at discerning small changes in pitch at low frequencies than they are at high frequencies. Incorporating this scale makes our features match more closely what humans hear. A frequency measured in Hertz (f) can be converted to the Mel scale using the following formula :

$$Mel(f) = 2595 \log\left(1 + \frac{f}{700}\right) \tag{2.2}$$

Mel scale is a scale that relates the perceived frequency of a tone to the actual measured frequency. It scales the frequency to match more closely what the human ear can hear.

MFCCs or Mel Frequency Cepstral Coefficients considers human perception for sensitivity at appropriate frequencies by converting the conventional frequency to Mel Scale, and are thus suitable for speech recognition tasks.

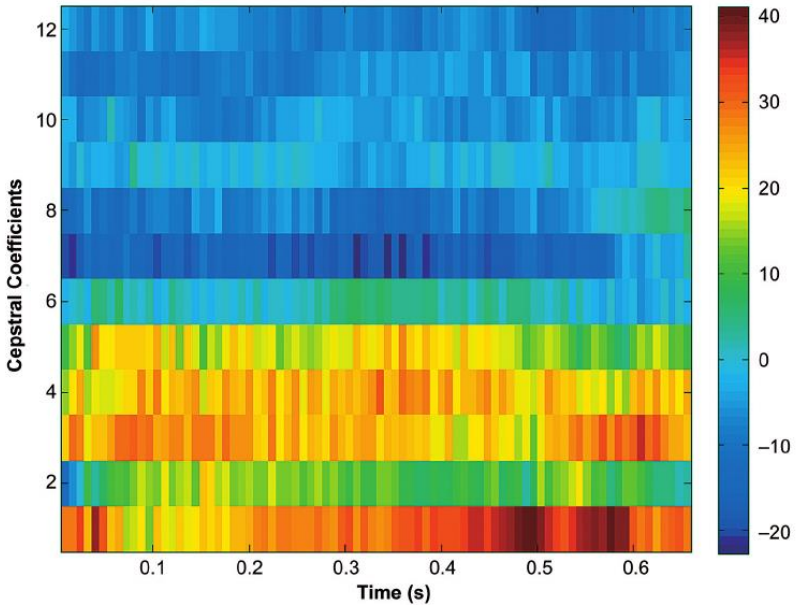


Figure 2.10: Sample MFCC (Courtesy: A Vibroacoustic Model of Selected Human Larynx Diseases)

2.4 Embedded Machine Learning

Machine learning is the science of getting computers to act without being explicitly programmed. These processes require a lot of computing power and memory to be present in the system on which it runs. A CNN model once compiled is in the order of MBs if not GBs. A typical GPU consumes power in the order of Watts. Deploying such models to embedded microcontrollers is a challenge because of the severe power and memory constraints.

The solution is to make use of various quantization methods on the models to shrink down the size and power requirements to be deployable to embedded microcontrollers. This is done by using various methods such as quantization, pruning, etc.

Some accuracy is lost in the process due to the reduces computes. The key is to find the balance between the size, latency, and accuracy for any given application.

TensorFlow provides multiple APIs to streamline this process of shrinking down models to make them deployable to embedded systems. For this reason, TensorFlow has been used extensively throughout this project for building and deploying neural networks to Arduino.

3. System Architecture

This section elaborates the methodologies employed in our work for increasing the safety of elevator systems as well as introducing smart elements such as VWW and KWS to pre-existing elevators. A brief overview of this section is as follows. We begin by introducing the basic structure of the project we chose to implement. Then we briefly describe the approach to implement the KWS and VWW models for automation of the system. Further we move on to take an overview of the fire safety system in place as well as the peripherals utilized. Then we look at the implementation of a fire safety system inside the elevator using a simple Operational Amplifier circuit and weigh the advantages and disadvantages of using a microcontroller to achieve the same goal.

3.1 System Overview

To increase the functionality of a pre-existing elevator, the developed technology in our project must be compatible with the current typical interface in current use. In the currently employed technology, elevators use a PLC which works in synchronization with multiple elements such as sensors, brakes, indicators, display modules, cabin peripherals, etc. Hence any additional hardware or software which are to be implemented should be able to fulfill all the current functions as well as efficiently introduce new ones. A generic block diagram of the entire proposed system is represented in Figure 3.1.

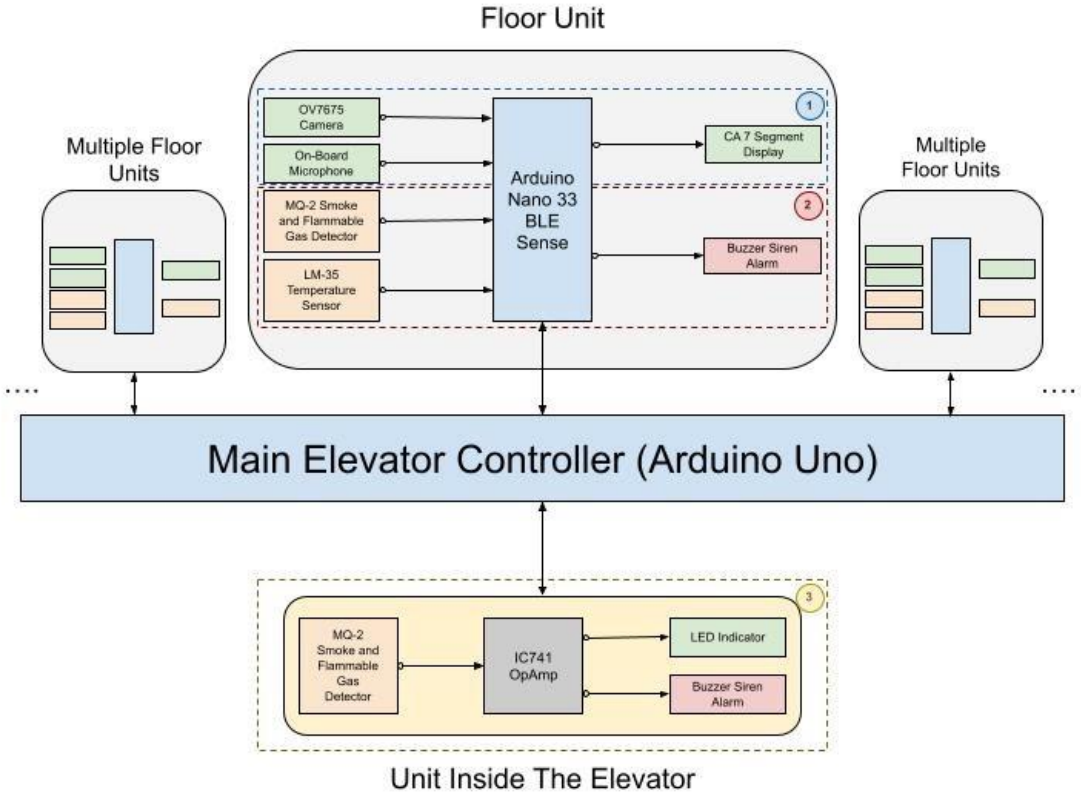


Figure 3.1: Generic Block Diagram for Implementation

The project a three-part system encompassing the following subsystems:

i. Person recognition (VWW Detection) and voice recognition (KWS) to operate the elevator.
ii. Fire safety mechanism inside the elevator to drop installed oxygen masks and signal the main elevator interface.
iii. Fire safety and detection system outside the elevator on different floors bundled with the features mentioned in point (1).

3.2 Hardware and Software Overview

To implement the project, we selected components and controllers based on their availability, suitability, feasibility, cost, and ease of use.

For our primary microcontroller development board, we selected the Arduino Nano 33 BLE Sense. It is a microcontroller board based on the Nordic nRF52840 processor that contains a Cortex M4F processor. The board features a rich set of sensors that allow for creating innovative and highly interactive designs. For the keyword spotting application of this project, we used the on-board microphone available. An overview of the key specs of the Arduino Nano

33 BLE Sense is given in Table 3.1.

The key factors we considered while choosing this board:

i. It comes loaded with an operating system (MBED OS) as compared to other boards available in the market. The Mbed OS is a platform and operating system for internet-connected devices based on 32-bit ARM Cortex-M microcontrollers. The presence of this OS allows for TensorFlow Lite for Microcontrollers to be deployed on the device due to the support for high level API calls and integration.

ii. The system is highly power efficient. While using a USB to power it, it consumes approximately 18.5mA.

iii. The board is robust and compact, hence highly deployable to real world applications.

iv. It is loaded with on-board sensors including 9 axis inertial sensor, humidity, and temperature sensor, barometric sensor, microphone, gesture, proximity, light color and light intensity sensor. Of these, we used the microphone to lead our Keyword Spotting Model.

Table 3.1: Arduino Nano 33 BLE Sense Specifications.

Feature	**Specification**
Microcontroller	nRF52840, 32-bit ARM Cortex M4 CPU, 64 MHz
Operating Voltage	3.3V
RAM	256KB
Flash Memory	1MB
Power	5V / 3.3V
BLE	Bluetooth Low Energy enabled
Sensors	LSM9DS1-9 Axis IMU, HTS221 Humidity & Temperature Sensor, APDS9960 gesture detection, proximity detection, digital ambient Light Sense (ALS), and Color Sense (RGBC), LPS22HB Barometer, MP34DT05-A Microphone
Digital Pins	14 (All PWM)
Analog Pins	8
Length	45 mm
Width	18 mm
Weight	5 grams (with headers)

A complete list of hardware and software tools used to build this project is listed in Table 3.2. The components were carefully chosen based on the characteristic features they offer. The reasons behind the choices are thoroughly explained in the chapter over various sections.

Table 3.2 List of Hardware and Software components used.

Type	Component	Purpose
Hardware	Arduino Nano 33 BLE Sense	Main development board, KWS Model
	OV7675 Camera Module	VWW Model
	MQ-02 Smoke/Gas Sensors	Fire safety systems
	LM-35 Temperature Sensor	
	74HC595 Shift Register	Interfacing a 7 Segment Display
	7 Segment Display	Displaying received floor number
	IC741	Operational Amplifier for Comparator
	Resistors, Capacitors, LEDs, etc.	Circuit accessories
Software	Arduino IDE	Programming, compiling and uploading C++ sketches to Arduino
	TensorFlow (Lite Micro)	Open-source software library for machine learning and neural networks
	Netron	Open-source structure viewer and analyzer for CNNs
	TinkerCAD	Online electronic circuit simulation software
	Proteus Design Suite	Simulation software for advanced electronic circuits
	KiCAD EDA	Facilitates the design of schematics for electronic circuits and their conversion to PCB designs
Programming Languages	C++	Programming the Arduino and interfacing sensors and modules to it
	Python	For TensorFlow; Creating CNN models and Machine Learning Architecture

3.3 Contactless Operation using KWS and VWW CNN Models

Our approach to the contactless elevator system problem statement is summarized in two steps:

1. Recognizing whether a person is standing in front of the elevator.
2. If there is a person standing, start listening for commands given by the person.

To implement this, we worked with the TensorFlow. TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular

focus on training and inference of deep neural networks. We used the concepts of Visual Wake Words (VWW) to implement person recognition and the concept of Keyword Spotting (KWS) to implement the voice recognition feature.

3.3.1 Datasets

The VWW and KWS models are trained on open-source datasets *COCO (Common Objects in Context)* and *Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition*, respectively.

i. 'COCO (Common Objects in Context)' Dataset

The 'COCO' dataset contains 2.5 million labeled instances in roughly 328,000 images openly crowdsourced for category detection, instance spotting and instance segmentation. The dataset contains photos of 91 object types that would be easily recognizable for any human. For our project we utilized a subset of the main dataset and used images containing only human subjects. This filtering led to a total training image pool size of 66808 images. For the training of the model, we As the number of images and the size of each image are both less, this dataset can be used as a preliminary database for computationally heavy algorithms.

Figure 3.2: Sample images from the COCO dataset with identified human subjects.

ii. ‘Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition’

The ‘Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition’ is a dataset of spoken words designed to help train and evaluate keyword spotting systems. It is a crowdsourced dataset assembled by Pete Warden which lies in the public domain. The dataset contains a lot of the common speech commands which are required by common KWS applications. These included the digits zero to nine, words that would be useful as commands in IoT or robotics applications such as "Yes", "No", "Up", "Down", "Left", "Right", "On", "Off", "Stop", and "Go" and many more. For the elevator application chosen, the requirement was of digit utterances, and hence the chosen numbers were “One”, “Two”, “Three” and “Four”. Table 3.3 shows the number of utterances which were present in the dataset for each of the keywords.

Table 3.2: Utterances of the chosen keywords in the KWS dataset.

Word	Number of Utterances
One	3,890
Two	3,880
Three	3,727
Four	3,728

3.3.2 Generic Methodology for the CNN Models

The KWS and VWW models are combined in a multi-tenant architecture. In such a structure, only one of the two models runs inference on the input data at a time. This allows for fast inferences and efficient memory allocation when and wherever required. As our development board of choice - *Arduino Nano 33 BLE Sense* has limited resources available on board (1MB CPU Flash Memory, 256KB SRAM), the neural network on the system had to be memory and energy efficient compared to a whole scale neural network model. Further post-training quantization methods were implemented to reduce the size of the model (KWS model: 26,720 bytes; VWW model: 3,00,568 bytes; Total size after compilation: 207712 bytes 79% of dynamic memory) and latency of the inferences in real time. Each model is divided into multiple steps. The model structure is detailed in the following subsections.

3.3.2.1 VWW Model for Person Detection

The Visual Wake Word model used for our elevator use case is based on the existing MobileNets model. Training a VWW model from scratch takes days of time because of the massively cascaded architecture and large image sizes. This is the reason we chose to build our model based on this preexisting architecture and quantize it to suit our needs. MobileNets is a family of architectures designed to provide good accuracy for as few weight parameters and arithmetic operations as possible. There are multiple versions available, but in our case, we are using the original version since it required the smallest amount of RAM at runtime. The core concept behind the architecture is depthwise separable convolution. This is a variant of classical two-dimensional convolutions that works in a much more efficient way, without compromising on accuracy too much. The model has various steps, as shown in Figure 3.3. Namely, these steps are Initialization, Pre-Processing, Model Inference, and Response.

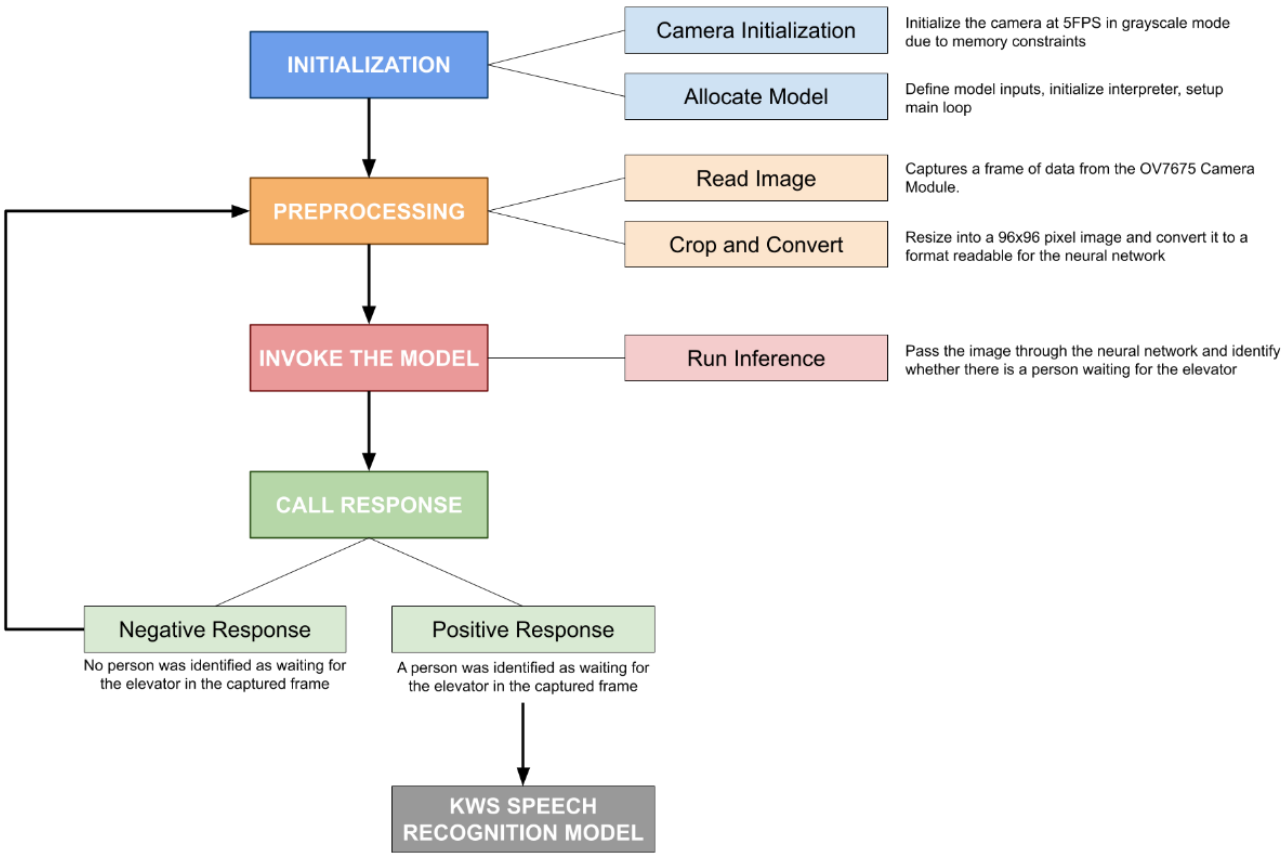


Figure 3.3: VWW block diagram.

i. Initialization of the Model:

In the first step of the cycle, the OV7675 camera module is initialized. For this specific application we initialized the camera at 5FPS grayscale mode due to the lesser memory requirements. The model is also allocated memory to run on the device. Further, we defined the model inputs, initialized the interpreter and setup the main loop of the model.

ii. Image Capture and Pre-Processing:

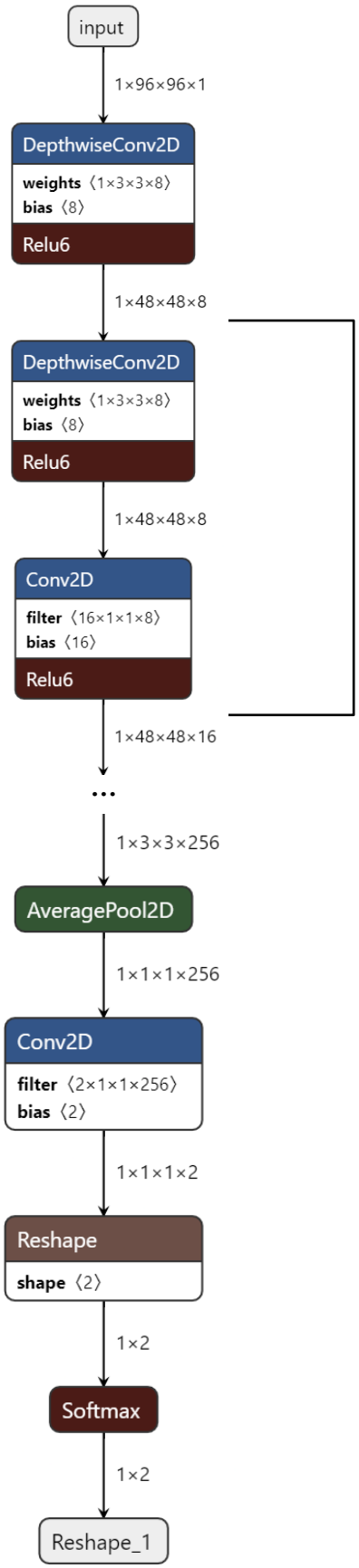
Once the model is initialized, we are ready to start the inference process. We capture a 176 x 144 pixel image in QCIF (Quarter Common Intermediate Format) from the camera module and parse the frame data. The captured frame of data is then resized into a 96 x 96 pixel image and is converted to format which is readable for the CNN to run inference on.

iii. CNN Model and Inference:

The preprocessed frame is fed to the CNN model for inference. The time taken for learning is greatly improved as well as the volume of data required for training the model is reduced significantly when fewer parameters are considered, and this is its practical advantage. The CNN is based on the pre-existing MobileNets model which is adequate for our purpose as discussed in the introduction of this section. The model has a 31-layer structure, including an iterating cascade of multiple DepthwiseConv2D and Conv2D layers. All these layers are equipped with ReLU activation layers to enable learning of non-linear mappings by the network. Earlier layers act more like edge recognition filters, spotting low-level structure in the image, and later layers synthesize that information into more abstract patterns that help with the final object classification. After the cascade, the data is pooled, fed through one final Conv2D layer, reshaped and passed through a SoftMax layer to normalize the output. The architecture summary of the model is shown in Figure 3.4.

While training the model, the data was augmented in multiple dimensions to introduce various methods for the recognition of distorted scenarios and images. The data was augmented in the following manners: horizontal shift of 10%, vertical shift of 10%, rotation range of 90^{o}, horizontal flip and vertical flip.

The training of the model takes a couple of days on a single-GPU v100 instance to complete one-million steps, but we were able to build a fairly accurate model after a few hours to experiment early.



x 13 cascades (with varying weights, filters, and biases)

Figure 3.4: VWW model architecture as shown using Netron.

The accuracy of the model at a checkpoint we examined after 5179 training iterations gave an accuracy of approximately 72%. After the completion of one million training steps, we can expect the accuracy to rise to 84% with a loss of around 0.4 as listed by TensorFlow in the usage guide of the MobileNets framework.

Once the CNN model was trained using the TensorFlow and Keras APIs, the model was exported to TensorFlow Lite by freezing and quantizing the weights of the CNN. All the weights were converted to 8-bit integer values for low memory performance on the Arduino Nano 33 BLE Sense at the loss of some accuracy. We used the TFLiteConverter class to handle the quantization and conversion into the TensorFlow Lite flatbuffer file that we need for the inference engine. The flatbuffer file was further converted into a C-source data array file to be directly compiled into the Arduino hex file as the model source.

During inference, the preprocessed 96 x 96 pixel image is fed to the on-board model which was compiled. The inference checks for the presence of a human waiting in front of the elevator and represents the output as a "No person score" and a "Person score" on the scale of -127 to +127, where negative values represent a low probability of the positive outcome and positive scores represent a higher probability of a positive outcome.

iv. Call Response:

If a person is detected in the given frame, the Arduino starts listening for audio commands which the person is giving to select a particular floor number by initializing the KWS model. If no command is found, then the system loops back to the Preprocessing step and captures another frame of data and reiterates the entire process for the next frame.

3.3.2.2 KWS Model for Speech Recognition

To enable a completely contactless experience for the elevator interface, we decided to go with a speech recognition approach to choose the floor number that a given user would like to go to. We went on to implement a KWS (Keyword Spotting) model to satisfy this requirement. The chosen keywords were floor numbers in the lower order "One", "Two", "Three" and "Four". Like the VWW model, the KWS model has 4 high level steps in the system as shown in Figure 3.5. Compared to the prior, this model is relatively lightweight and takes lesser time for training and inference.

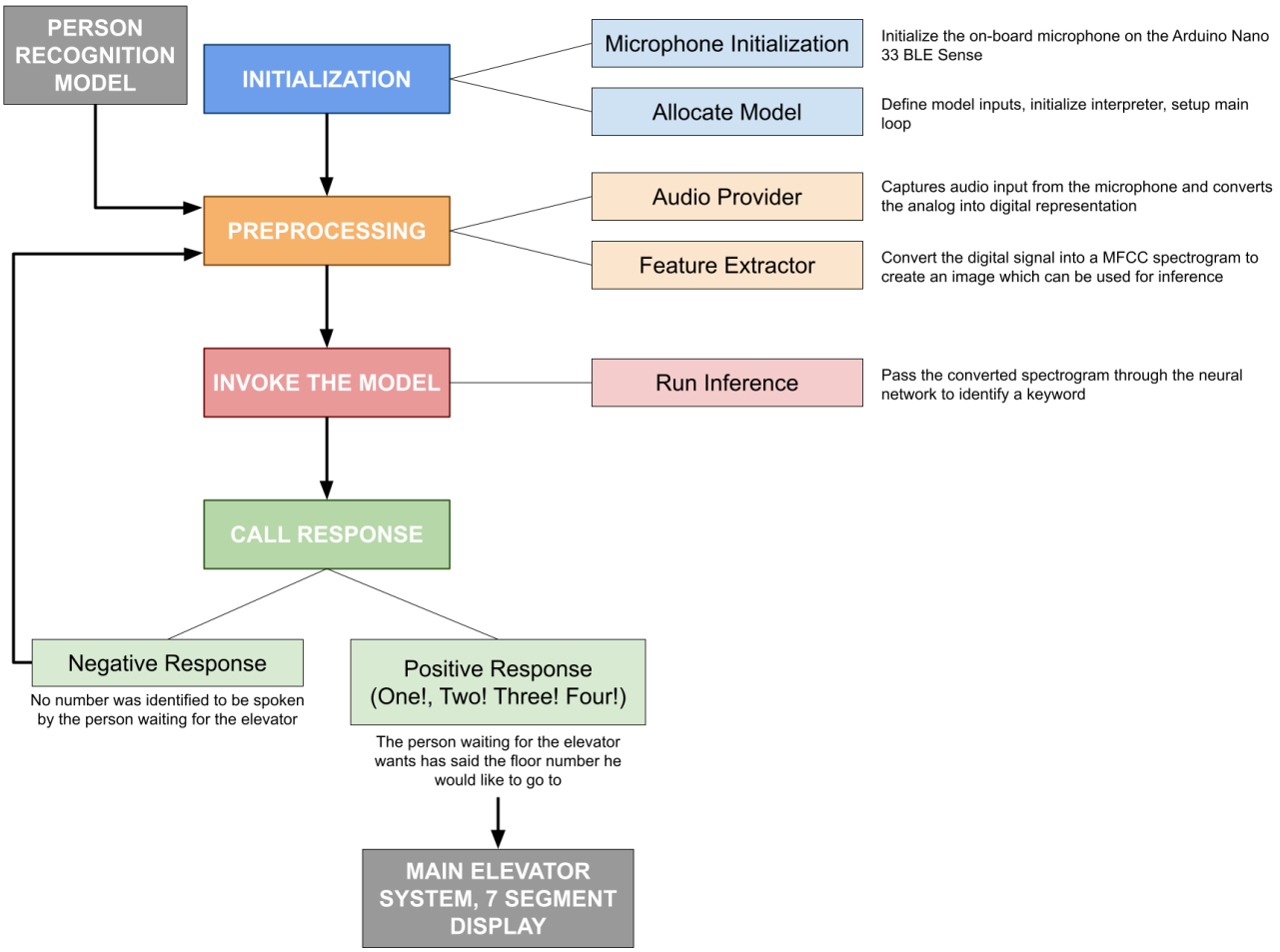


Figure 3.5 KWS block diagram.

i. Initialization:

During the initialization phase of the KWS model, the peripheral microphone required is setup and interfaced. Further, the model weights and inputs are defined, interpreter is initialized, and the main loop is setup. The required memory is allocated, and the required operators are resolved.

ii. Preprocessing:

The preprocessing stage of the KWS model consists of two stages:

a. Audio Provider
b. Feature Extractor

These two systems collectively transform the raw input microphone data in to data which is interpretable and classifiable for the CNN. Both stages consist of various intermediate steps.

a. Audio Provider:

The audio provider stage sets up the microphone, reads a timeframe of data input and converts the incoming analog data into a digital representation. We sample the data at a rate of 16 kHz. The Analog to Digital conversion in the Arduino is handled by pre-existing libraries built into the Arduino. The digital representation of the data is loaded into an audio buffer continuously to be able to identify when a user says a particular keyword. The values are stored as 16 bit values in the buffer for the feature extractor to process. The input analog signals are visualized and shown in Figure 3.6 below.

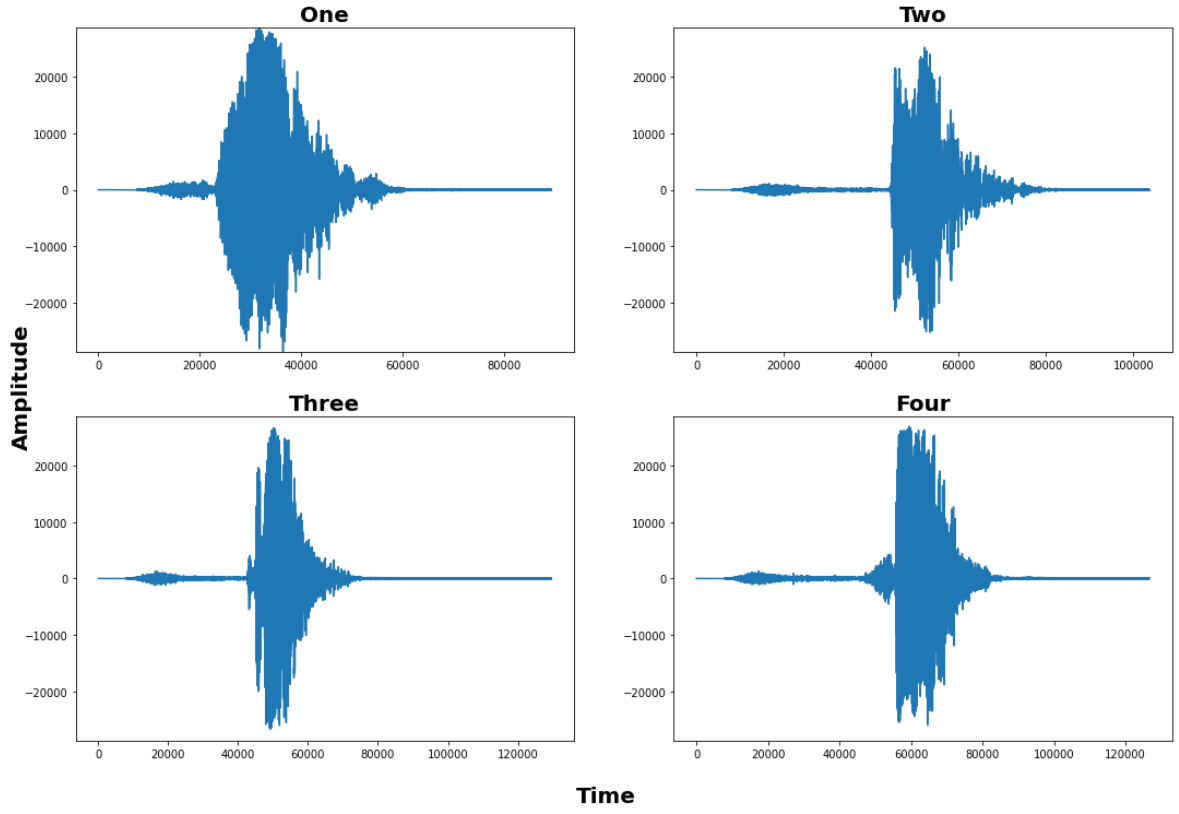


Figure 3.6: Analog visualization of raw audio signals of the keywords.
(Note: These figures are recorded using our own voices and plotted using a Python Script.)

b. Feature Extractor:

To make it easier for the CNN to process and classify the audio signals into different categories, we extract the features of the buffer audio sample before passing it into an inference pipeline. First, we take a 30 millisecond snapshot of the audio buffer and generate a FFT (Fast Fourier Transform) of it in 256 values. To make the FFT more robust, we average the 256 values of the FFT into 43 values. The FFT visualizations of the analog

visualizations in Figure 3.6 are shown in Figure 3.7. These 43 values represent a single slice of the spectrogram we aim to generate. Each FFT represents a single column in the complete spectrogram of the audio buffer. Once one second's worth of spectrogram data is obtained, a complete spectrogram is generated. The keyword representations as spectrograms are shown in Figure 3.8. Hence, the audio signal we currently possess is a time series signal, which we have converted into a frequency domain signal. In the frequency domain, we can do further optimizations and generate a MFCC (Mel Frequency Cepstral Coefficients) to understand features influenced by factors which are not comprehended by human ears. Our hearing mechanism naturally pick up low-frequency signals much better than higher-frequency signals. And, therefore, if we focus our effort or energy more on the low-frequency signals, then we are likely to pick up the words better. The MFCC emphasizes the lower frequencies much more cleanly and that allows our machine learning model to pick up the signals for the keywords much more clearly. The generated MFCCs of our chosen keywords are shown in Figure 3.9. These MFCCs are then passed as inputs to our CNN model for inference.

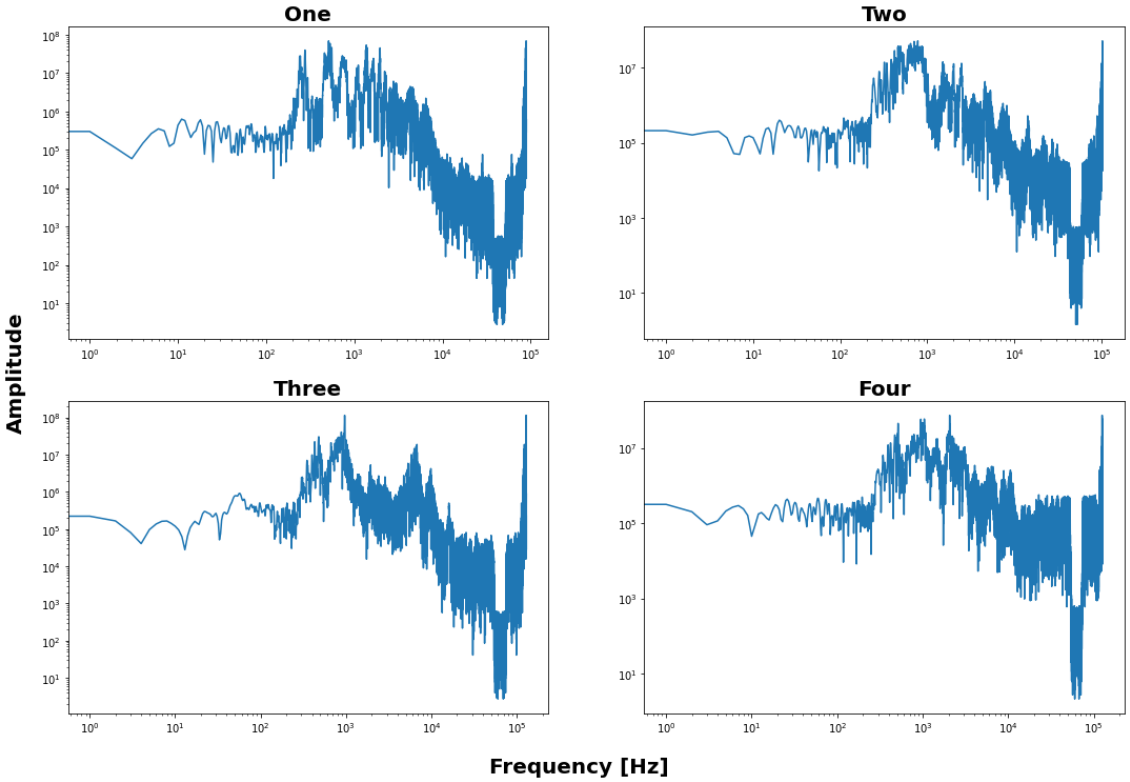


Figure 3.7: Generated FFTs of the keyword audio signals.
(Note: These figures are recorded using our own voices and plotted using a Python Script.)

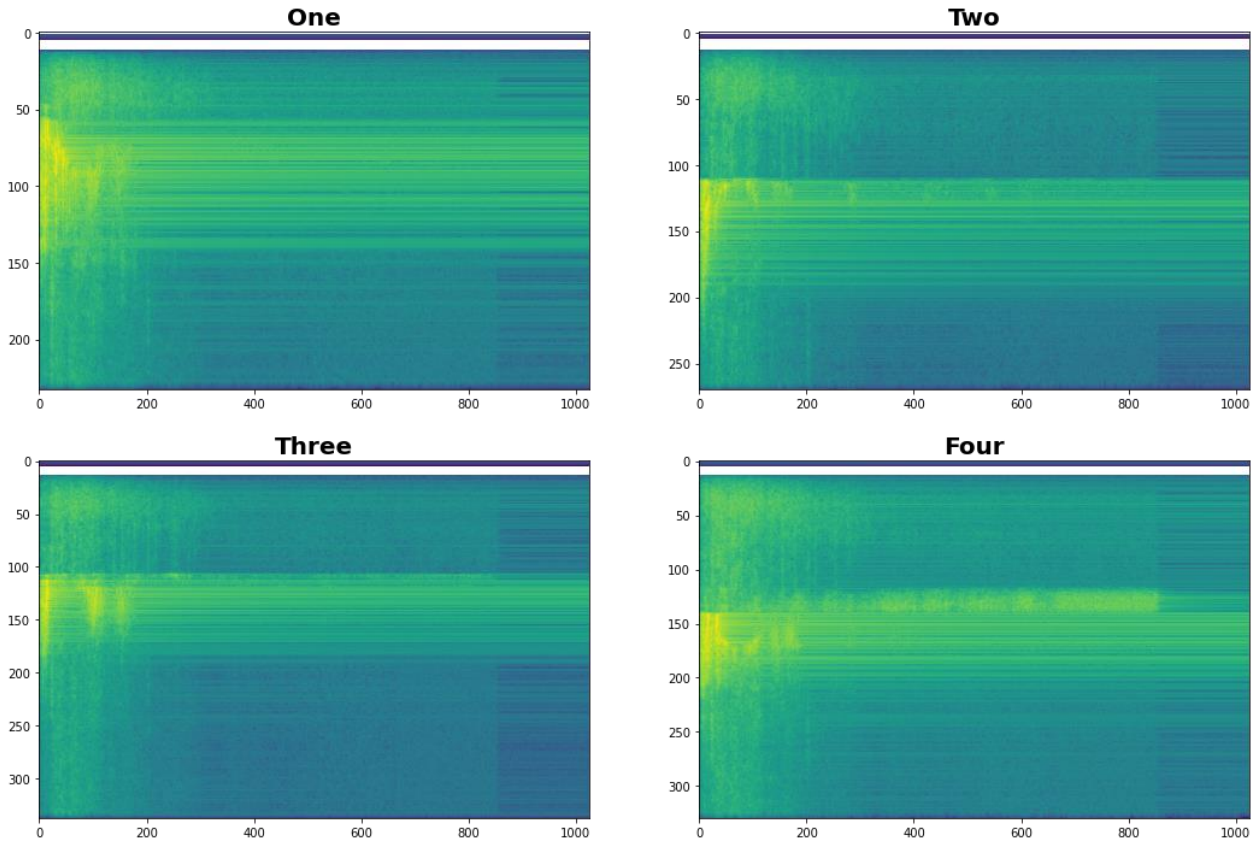


Figure 3.8: Generated frequency domain spectrograms of the keyword audio signals.
(Note: These figures are recorded using our voices and plotted using a Python Script.)

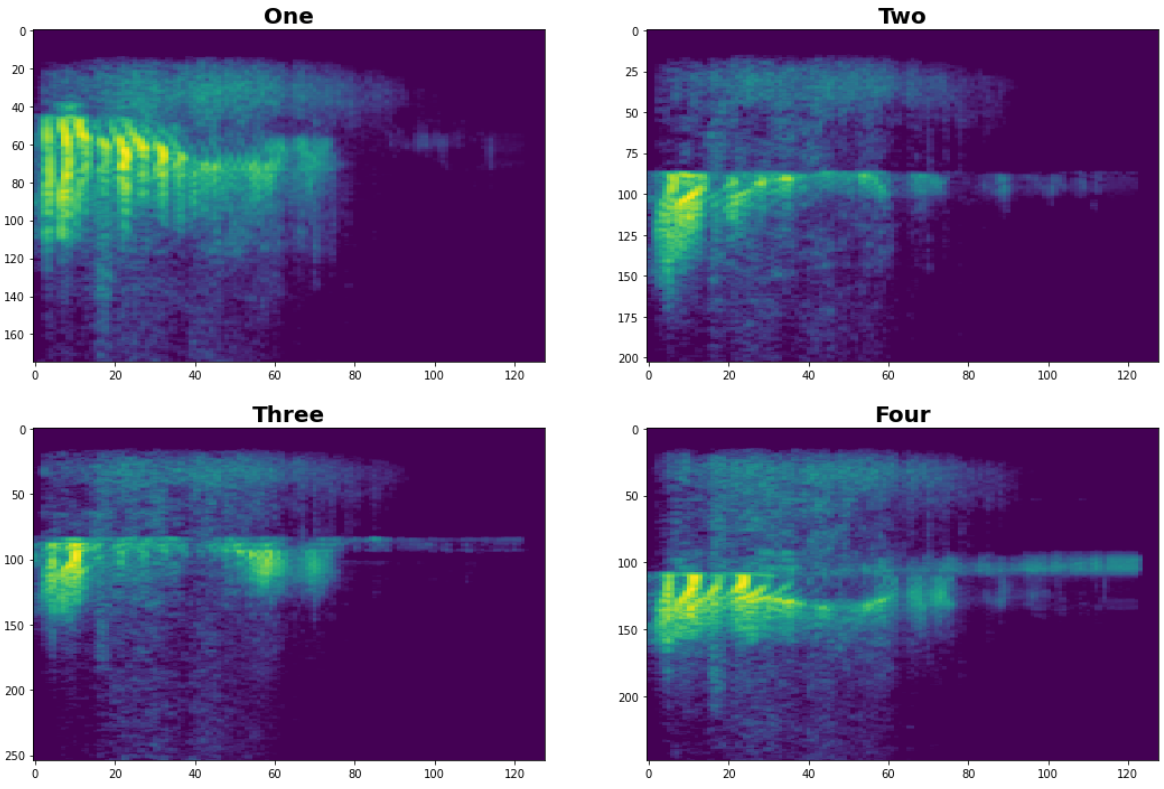


Figure 3.9: Generated MFCCs of the keyword audio signals.
(Note: These figures are recorded using our own voices and plotted using a Python Script.)

iii. KWS CNN Model and Inference:

To identify a spoken keyword, we have built a basic CNN model based on the "tiny_conv" model architecture. The chosen neural network is built on this model to leverage its pre-optimized architecture for deployment to embedded microcontrollers. The tiny_conv architecture is lightweight and consists of a single DepthwiseConv2D and a FullyConnected followed by a Softmax layer for normalization. The architecture of the CNN is as shown in Figure 3.10. A summary of the training parameters is given in Table 3.4. The final training result along with the confusion matrix is shown in Figure 3.11.

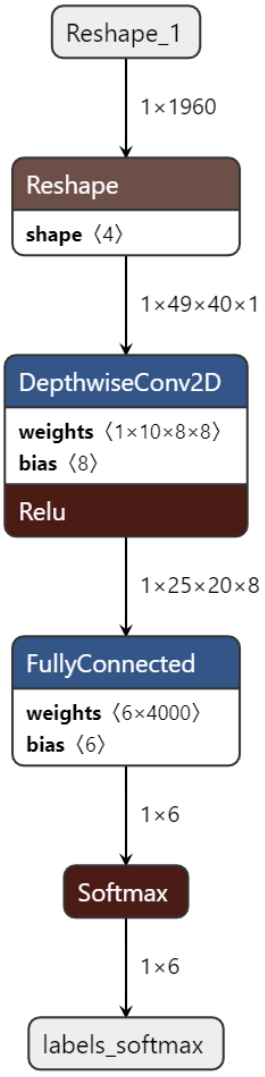


Figure 3.10 Architecture of KWS Model.

Table 3.4 KWS model summary specifications.

Parameter	Description
Model Architecture	tiny_conv
Total Training Steps	12,000 + 3,000 = 15,000
Learning Rate For 12,000 Steps	0.001
Learning Rate For 3,000 Steps	0.0001
Sampling Rate	16 kHz
Clip Duration Size	1000 milliseconds
Window Size	30 milliseconds
Validation Accuracy	85.5%
Final Test Accuracy	83.5%

```
INFO:tensorflow:Confusion Matrix:
 [[222   1   0   3   1   1]
 [  1 130  36  18  25  18]
 [  4  18 309   2   6  12]
 [  2  22   0 308   4   9]
 [  0  20   2   6 326   2]
 [  1  21  13  20   4 314]]
I0412 15:55:53.936519 139994649655168 train.py:287] Confusion Matrix:
 [[222   1   0   3   1   1]
 [  1 130  36  18  25  18]
 [  4  18 309   2   6  12]
 [  2  22   0 308   4   9]
 [  0  20   2   6 326   2]
 [  1  21  13  20   4 314]]
INFO:tensorflow:Step 15000: Validation accuracy = 85.5% (N=1881)
I0412 15:55:53.936759 139994649655168 train.py:289] Step 15000: Validation accuracy = 85.5% (N=1881)
INFO:tensorflow:Saving to "train/tiny_conv.ckpt-15000"
I0412 15:55:53.936888 139994649655168 train.py:297] Saving to "train/tiny_conv.ckpt-15000"
INFO:tensorflow:set_size=2150
I0412 15:55:53.963052 139994649655168 train.py:301] set_size=2150
WARNING:tensorflow:Confusion Matrix:
 [[254   1   2   3   1   0]
 [  2 144  39  34  24  18]
 [  2  14 356   5   4  18]
 [  0  33   9 365   3  14]
 [  0  17   6   8 367   7]
 [  2  29  16  32  11 310]]
W0412 15:56:06.869505 139994649655168 train.py:320] Confusion Matrix:
 [[254   1   2   3   1   0]
 [  2 144  39  34  24  18]
 [  2  14 356   5   4  18]
 [  0  33   9 365   3  14]
 [  0  17   6   8 367   7]
 [  2  29  16  32  11 310]]
WARNING:tensorflow:Final test accuracy = 83.5% (N=2150)
W0412 15:56:06.869749 139994649655168 train.py:322] Final test accuracy = 83.5% (N=2150)
```

Figure 3.11 Final Training Results of KWS Model.

After the CNN model was trained using the TensorFlow and Keras APIs, the model was exported to TensorFlow Lite by freezing and quantizing the weights of the CNN, a common process for both the VWW and KWS model. All the weights were converted to 8-bit integer values for low memory performance on the Arduino Nano 33 BLE Sense at the loss of some accuracy. We used the TFLiteConverter class to handle the quantization and conversion into the TensorFlow Lite flatbuffer file that we need for the inference engine. The flatbuffer file was further converted into a C-source data array file to be directly compiled into the Arduino hex file as the model source, just as the VWW model.

During inference, we copy the feature buffer data to the input tensor. Then we pass the MFCC data through the CNN and attempt to classify the incoming data as one of the following categories:

a. "One"
b. "Two"
c. "Three"
d. "Four"
e. Unknown – unrecognized word input.
f. Silence – no word input detected.

iv. Call Response:

Based on the outcome of the inference, the floor number (if detected) and sent to the main elevator interface and displayed on a Seven Segment Display. If there is no output found (i.e., Unknown or Silence category), then the system loops back at the preprocessing phase of the KWS model. If no data is found for 5 seconds of time, then the KWS loop is exited and the VWW model is executed from the preprocessing stage.

3.3.3 Multitenant Architecture for Running KWS and VWW Simultaneously

Due to the memory and power constraints on any embedded microcontroller, running two CNN inference engines simultaneously is a challenge. Our approach to this problem was to implement a multi-tenant architecture for combining the KWS and VWW model. In such a structure, only one of the two models runs inference on the input data at a time. Both models share the TensorArena and use the same memory allocations. This allowed for fast inferences and efficient memory allocation when and wherever required. There is also no requirement for the synchronization of data streams from the camera module and the microphone, both of which are running at different sampling rates.

3.4 Fire Safety Unit Outside The Elevator

Elevator shafts may be exposed to smoke and that smoke could reach occupants not just inside the elevator but on other floors as well. In such situations, the elevator is advised to return to the ground floor of the building immediately. The most efficient way to develop a fire safety system which protects the occupants of an elevator is to design a system which works in tandem with it the smoke/flammable gas detectors and temperature sensors directly. To maximize the efficiency of such a system, we developed a pipeline in which the main elevator interface has direct inputs from the smoke detection units. Depending on the levels obtained from the sensors, the elevator interface can further be commanded to execute a specific set of instructions, for example taking the elevator to the lowest floor immediately in the case of a fire.

To implement our approach, we chose to use a MQ-2 Smoke/Flammable Gas Detector and a LM-35 Temperature sensor.

3.4.1 MQ-2 Smoke/Flammable Gas Sensor

MQ-2 is one of the commonly used gas sensors in real world applications. It is a Metal Oxide Semiconductor (MOS) type Gas Sensor (also known as chemiresistors) as the detection is based upon change of resistance of the sensing material when the gas comes into contact with the material.

The key factors we considered while choosing this sensor:

i. Using a simple voltage divider network, concentrations of gas can be detected.
ii. It can detect LPG, Smoke, Alcohol, Propane, Hydrogen, Methane and Carbon Monoxide concentrations anywhere from 200 to 10000 ppm.

3.4.2 LM-35 Temperature Sensor

The LM35 series are precision integrated-circuit temperature devices with an output voltage linearly proportional to the centigrade temperature. This is an industrial standard sensor which is readily available for use.

The key factors we considered while choosing this sensor:

i. Highly accurate across a variance of temperatures which our use case required.
ii. Small module and easy for deployment.
iii. Using a simple voltage divider network, accurate temperature readings can be obtained.

3.4.3 System Working

The MQ-2 Detector and the LM-35 Temperature sensor are connected to the microcontroller as per the specifications given in their respective datasheets. The provided connections for both the sensors are shown in Figure 3.12 and Figure 3.13, respectively.

The output voltages are given to analog input/output terminals of the Arduino Nano 33 BLE Sense. Using the *analogRead()* API provided by Arduino which accesses the on-board 10-bit analog to digital converter. The microcontroller maps input voltages between 0 and the operating voltage(5V) into integer values between 0 and 1023. This process yields a resolution between readings of 5 volts / 1024 units or, 0.0049 volts (4.9 mV) per unit. On our chosen microcontroller, it takes about 100 microseconds (0.0001 s) to read an analog input, so the maximum reading rate is about 10,000 times a second.

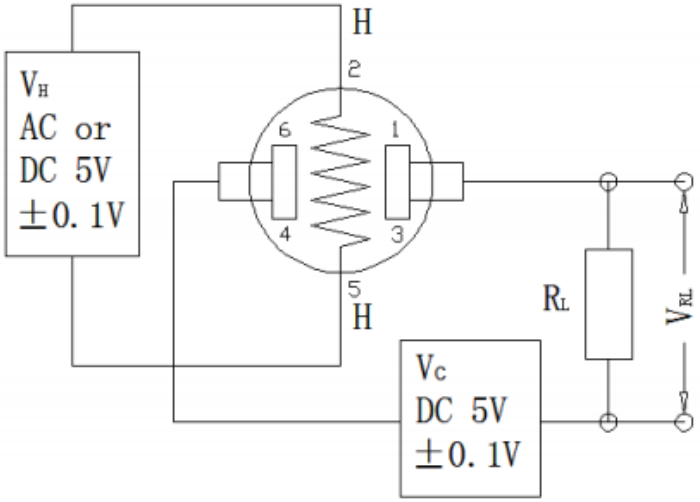


Figure 3.12 Circuit for MQ-2 Smoke Detector.

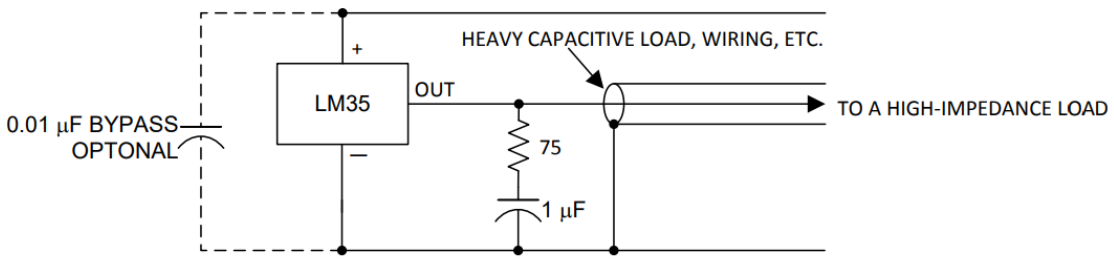


Figure 3.13 Circuit for LM-35 Temperature Sensor.

Due to the limitations of our simulation, there was no on-board ADC available and hence, we programmed the Arduino to do the conversion manually for the LM-35 Temperature sensor using the following formula:

$$Temperature\ (Celsius) = V_{out}\ \cdot \frac{1100}{1024*10} \tag{3.1}$$

For the detection threshold, we chose values which are abnormally high enough to indicate that the building has a fire present. For the temperature sensor, an alarm system was triggered once the temperature crossed 45^{o}C. The smoke detector we used in our simulation software did not have an analog output capability and hence we triggered it as a binary device indicating presence of smoke/no smoke.

We placed a condition on the system such that if either the smoke detector output is positive or the temperature reading is above 45^{o}C, then an alarm system is triggered which delivers an output to a logical buzzer circuit as well as conveys the status to the central elevator system.

3.4.4 Fire Alarm System and Connection to Main Elevator Interface

In the situation of a fire, the entire building as well as the central elevator interface must be notified at once. To create a robust system for handling this, we created a logic table as shown in Table 3.6. The system is based on the presence and absence of a digital HIGH signal from the Arduino Nano 33 BLE Sense.

Comparing the outputs required for the execution of the logic, we decided to use an Exclusive-OR Gate to execute the logic. By this, if a fire is present in a given situation, then the alarm circuit is triggered and the connection from the main elevator interface is lost. If there is no signal from a given floor, the main elevator interface flags it and immediately grounds the elevator. In the absence of a fire, the elevator resumes normal operation. The circuit for this alarm mechanism is shown in Figure 3.14.

Table 3.5: Logic Table for Alarm System.

Arduino Signal (Input)	**Main Elevator Interface (Output)**	**Fire Safety Alarm (Output)**
1 (No Fire)	1 (Signal present, hence, no fire)	0 (Not triggered)
0 (Fire)	0 (Signal absent, hence presence of anomaly)	1 (Triggered)

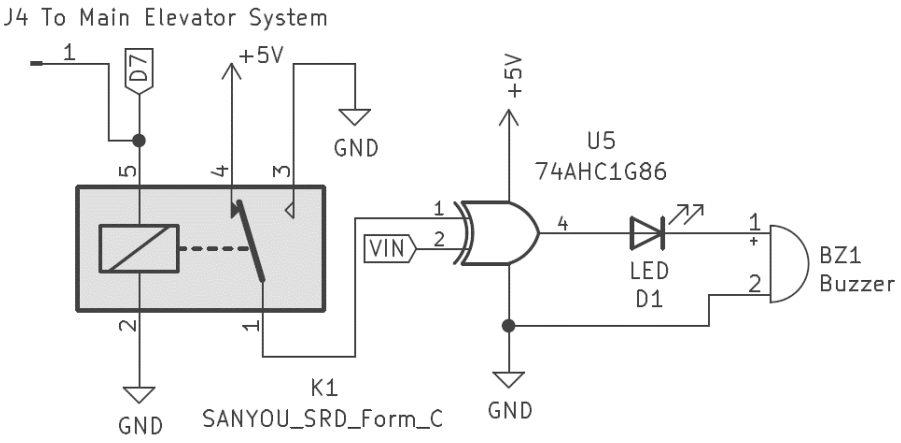


Figure 3.14:Circuit diagram for Fire Safety Alarm Outside the Elevator.

For this, we employed the 74AHC1G86 EX-OR Gate IC. It is a two-input single EX-OR gate IC. The key factor we considered while choosing this element was the simplicity required for the execution of the purpose. This was used in the fire detection alarm system buzzer and LED.

3.5 Fire Safety Unit Inside The Elevator

In the case of an electrical malfunction or fire hazard inside the elevator, the users must be immediately evacuated and taken to safety. For such reasons, an independent fire safety alarm system is necessary. We implemented this using a simple comparator circuit based on an IC741 Operational Amplifier. An operational amplifier is a high-gain electronic voltage amplifier with a differential input and a single-ended output. The key factors we considered while choosing this element:

i. The IC741 is a broadly used single input Op-Amp IC which aptly fulfilled our requirements.

ii. It is a reliable system with ease of use.

The details of a comparator circuit as were discussed in the Theoretical Background chapter serve as the basis for the smoke detector circuit we have assembled here. The circuit diagram of the circuit is as shown in Figure 3.15 and is in accordance with the circuit diagram as obtained for the MQ-2 in Figure 3.12.

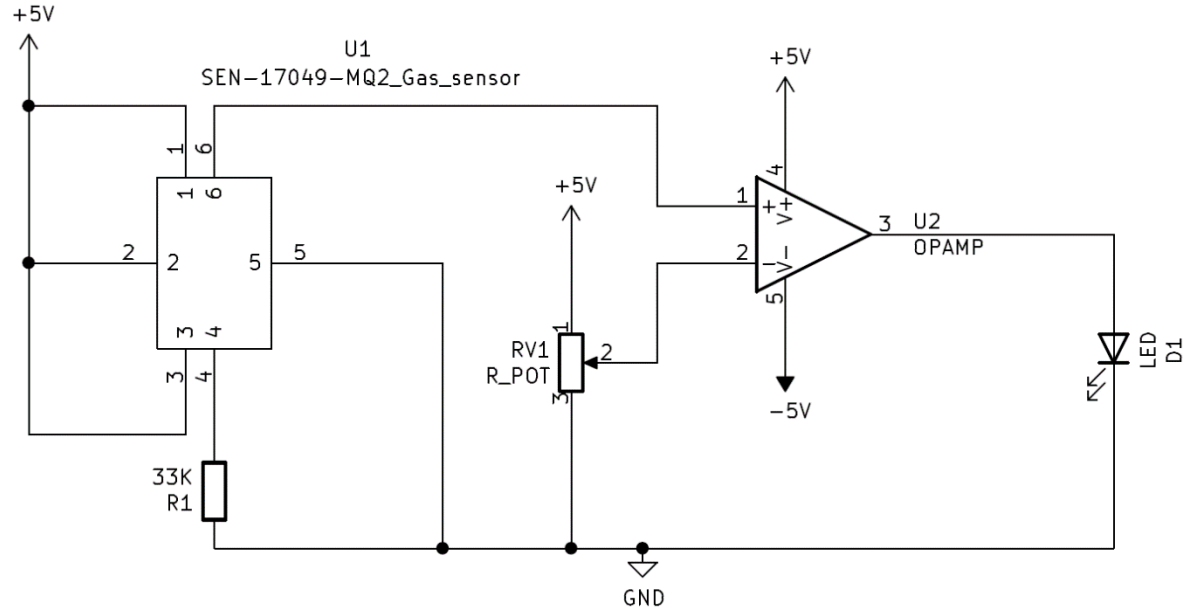


Figure 3.15:Circuit diagram for Fire Safety Alarm Inside the Elevator.

By adjusting the reference voltage on the potentiometer, we can increase or decrease the sensitivity of the circuit as required. The output of the comparator circuit is an LED in the given circuit but can also be a sound-based alarm system such a buzzer or a siren. The output of the circuit is also given to the main elevator interface as shown in the general block diagram Figure 3.1. The practical values of the circuit are discussed in the Results and Conclusion chapter.

4. Results, Analysis and Conclusion

The implementation of this project was carried out in two stages based on the availability of parts and components. The VWW and KWS models for the contactless elevator system were implemented using hardware on an Arduino Nano 33 BLE Sense module. The rest of the subsystems were implemented on simulation softwares including Proteus Design Suite and TinkerCAD based on the necessities and additional features which are required.

4.1. Hardware Implementation

The hardware required for the implementation of the contactless elevator system, i.e., the VWW and KWS models was readily available with us, namely the Arduino Nano 33 BLE Sense and OV7675 Camera Module. They are mounted on a shield for the scope of this experiment. Further into the implementation, we designed our own shield to house the OV7675 and Arduino module along with other peripherals.

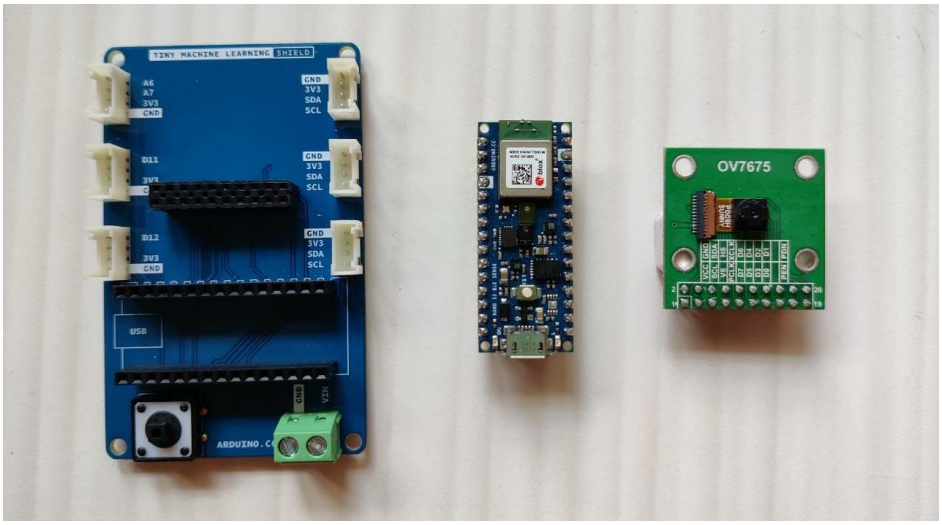


Figure 4.1: Individual components used for the hardware setup.

4.1.1. Working and Results Achieved

Once the created KWS and VWW models were deployed onto the Arduino, we were able to run inference in particular test cases.

4.1.1.1. Setting Up

The field of view of the camera module is moderately narrow. The best resolution is obtained when the camera is at least roughly an arm's length from the human subject for the VWW model to run efficiently. We found the best results when the shield is propped up against a vertical surface. The microphone feature occasionally struggles to be accurate from such distances, but the efficiency can be increased using directional microphones instead of on-board default microphones in the future.

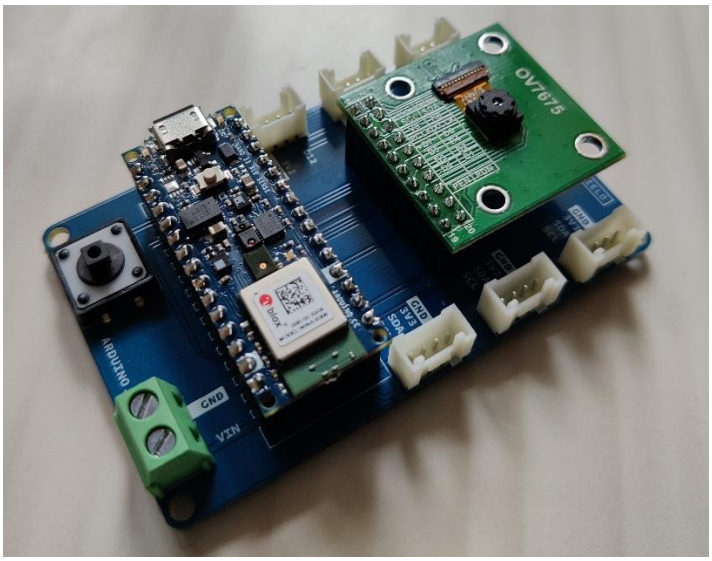


Figure 4.2: Hardware setup for the Arduino Nano 33 BLE Sense and OV7675 Camera

4.1.1.2. Running Inference

Once the shield was properly positioned, we ran inference by standing inside and outside the frame of the camera. As discussed in the System Architecture chapter, each inference returns a probability value between -127 and +127, where higher numbers indicate a higher probability of a human subject being present in the frame. Sample greyscale QCIF images of size 144 x 176 pixels with human subject are shown in Figure 4.3. These images when fed to the model algorithms on-board give a positive output after inference. Although these images are distorted and incomplete, the algorithm is able to recognize adequate features to correctly identify a human subject because of the vast MobileNets training parameters and dataset.

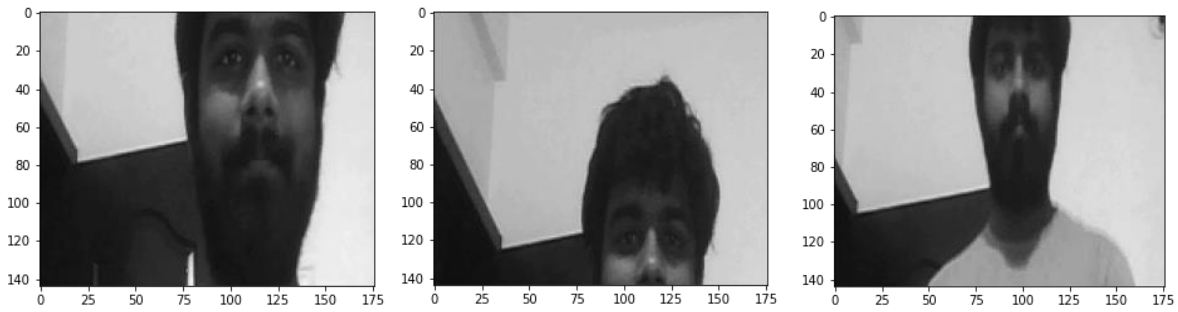


Figure 4.3: Sample greyscale QCIF images of size 144 x 176 pixels with human subject.

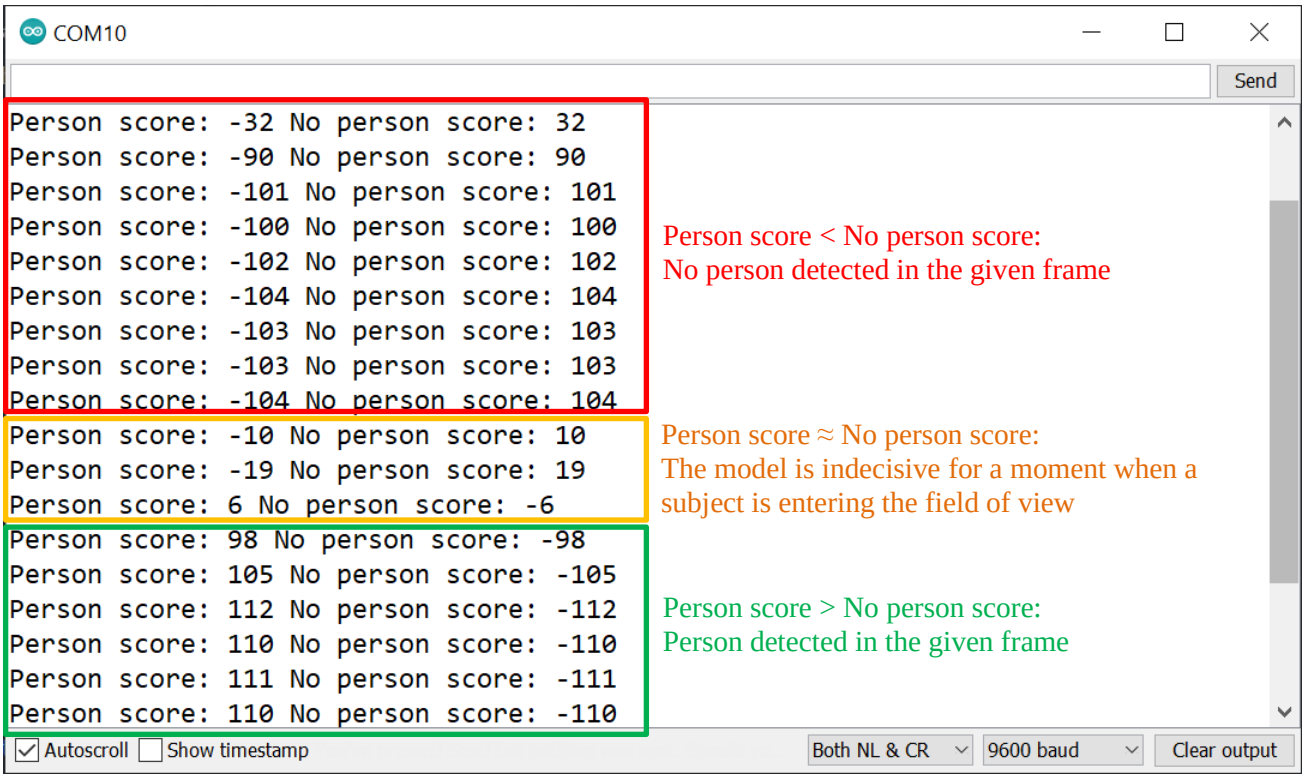


Figure 4.4: Output of the VWW model inferences.

Once the inference engine is run, the probability values are generated. These values are printed on the serial monitor of the Arduino IDE as shown in Figure 4.4. When a person is entering the frame or in motion, the camera is unable to pick up the fast movement due to the low frame rate. At such moments, the model is decisive of whether a person is the frame or not. Hence, while classifying as a human detected in the field of view, we chose to keep a minimum Person score of 25. This threshold value reduces the number of false positives. If a person is detected in the frame, the green RGB light on the Arduino is lit up, whereas when there is no person detected, the red RGB light is lit up as shown in Figure 4.5.

Once a person has been detected in the frame, the KWS model inference engine is triggered, and the system starts listening for keywords spoken by the user to communicate the floor number which they would like to go to. These numbers are chosen to be “One”, “Two”, “Three”, and “Four”. The incoming audio is sampled at 16 kHz and ran inference on every one second. Whenever the Arduino is listening for a keyword, the yellow light rapidly blinks, and upon positive inference, the blue RGB light is lit up.

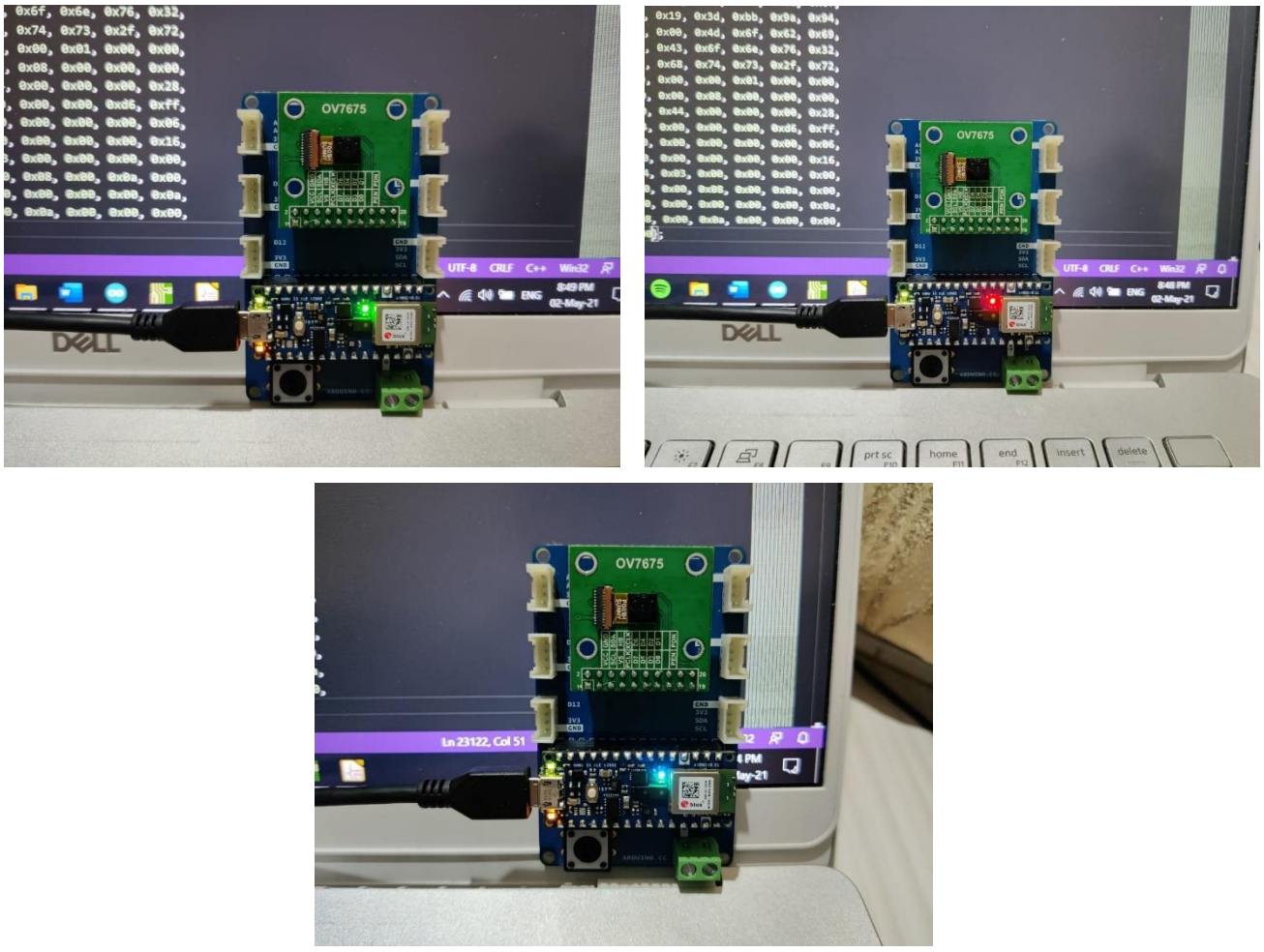


Figure 4.5: LED Indicators of the VWW and KWS model inferences.

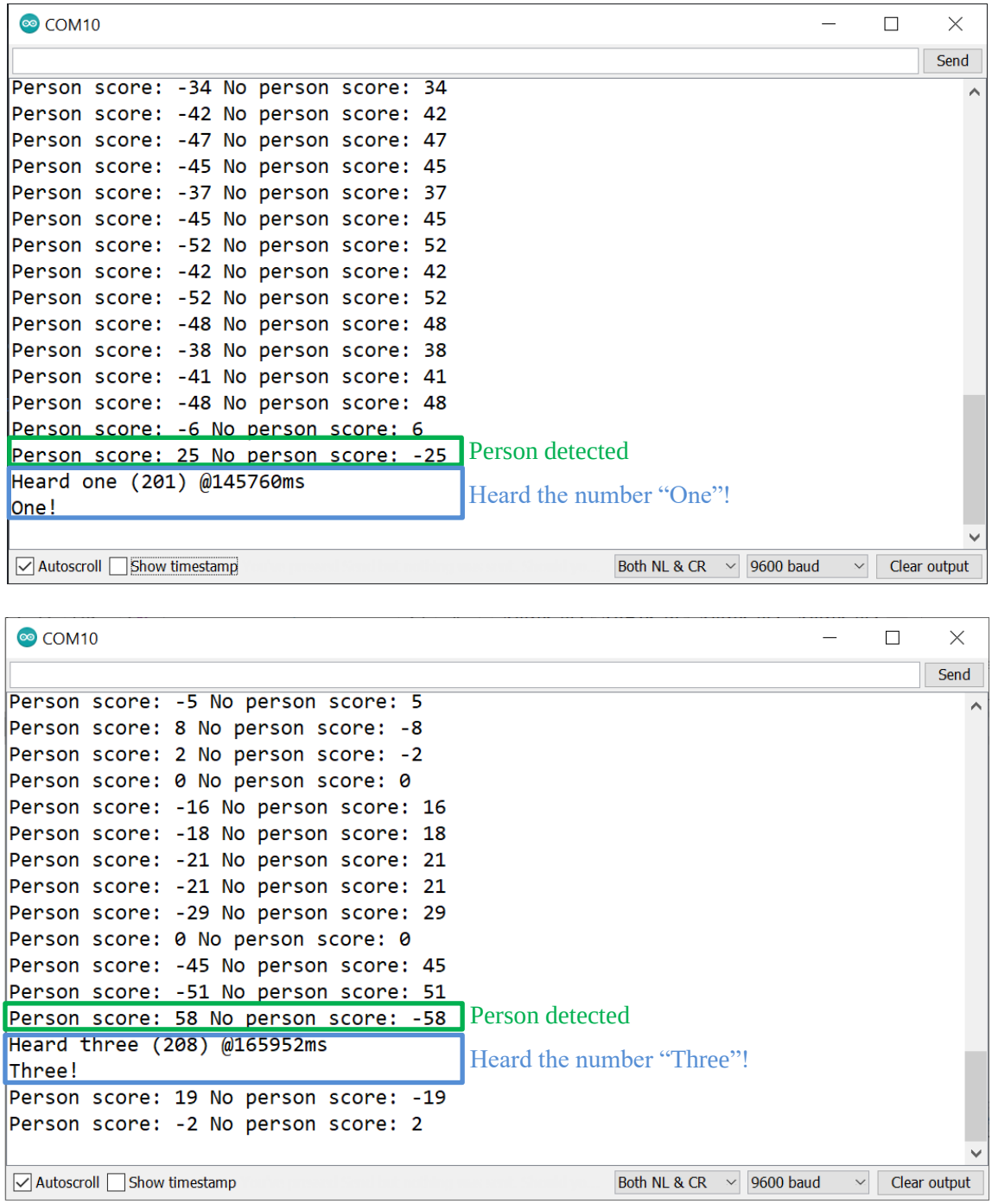


Figure 4.6: Output of the KWS model inferences.

The output hence obtained from the KWS inference is then displayed on the 7 Segment Display by using a 74LS595 Shift Register, as well as sent to the main elevator interface for controlling the elevator.

4.1.1.3. Accuracy and Latency Observations

During the testing phase of the hardware implementation, the model was robust and quick to respond to keywords and wake words. The accuracy so obtained while training the model of 83.5% and 84% for the VWW and KWS model is adequate and optimal.

The model shows a latency low enough for it to be feasible as a practical product. The inference of the person detection model takes approximately 740 milliseconds, and an inference of the speech recognition model takes about 30 milliseconds.

4.1.2. Bridging the Gaps Between The Hardware and The Simulations

Since the further parts of the results are conducted on simulation software such as Proteus Design Suite and TinkerCAD, there is a continuity break between the hardware and simulation pipeline.

To address this, we created the following system:

- The output of the KWS speech recognition model is printed on the serial monitor of the Arduino IDE as a number between one to four.
- The same input will be fed through the virtual monitor of the Proteus simulation to continue the further steps of the pipeline.
- In ideal implementation, the two systems will take inputs directly from each other without any physical human bridging.

4.2. Software Based Simulations

The fire safety system inside and outside the elevator were implemented using simulation softwares, namely Proteus Design Suite and TinkerCAD. The simulations work in continuation of the hardware based person and speech detection systems, linked together using manual inputs from the user to correspond between the two.

4.2.1. Display and Fire Safety System of the Outside (Floor) Unit on Proteus

The fire safety system outside the elevator bundled with the deployed unit was simulated on Proteus. The system consisted of the following components:

1. MQ-2 Smoke Detector System
2. LM-35 Temperature Sensor
3. 7 Segment Display using Shift Register
4. Alarm and Connection to Main Elevator Interface

Since the Arduino Nano 33 BLE Sense was not available as a library for simulation, we used the Arduino Nano baseline model for it. The architecture is similar and hence is adequate for the purposes of the project.

For the sake of completeness in the project, we chose to go create a pseudo-main elevator interface using an Arduino Uno in the simulation. A 4 x 16 LCD Display was also added for us to keep track of the system status.

Figure 4.7 shows the circuit schematic for the project on Proteus. The Arduino Nano is interfaced with a LM-35 and MQ-2 sensor, a 7 Segment Display and an alarm system which is triggered when the system is simulating a fire scenario.

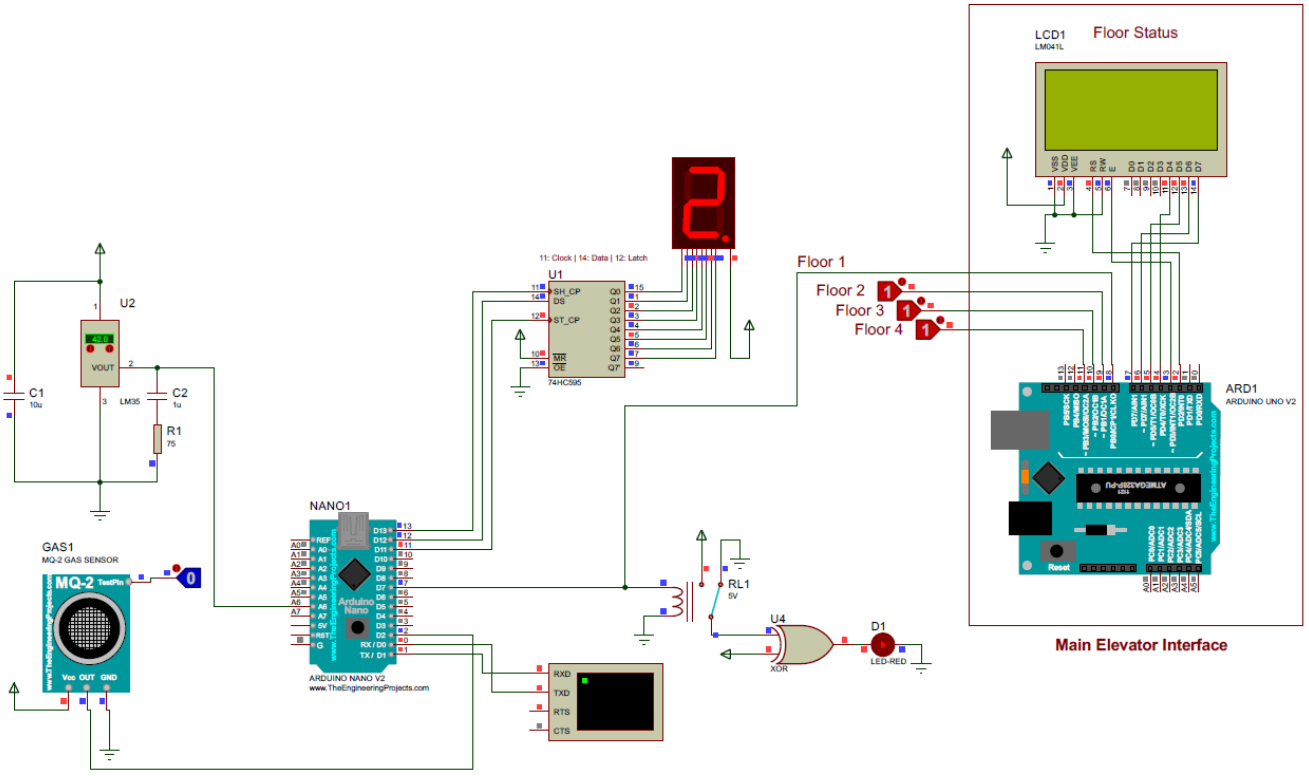


Figure 4.7: Outside Unit Simulation on Proteus

4.2.1.1. Working Overview

To maintain continuity between the hardware and the simulation, the user can input a floor number into the virtual terminal of the Arduino Nano on the Proteus Simulation. This floor number is displayed on the 7 Segment Display by using the *Serial.read()* method. This is shown in Figure 4.8.

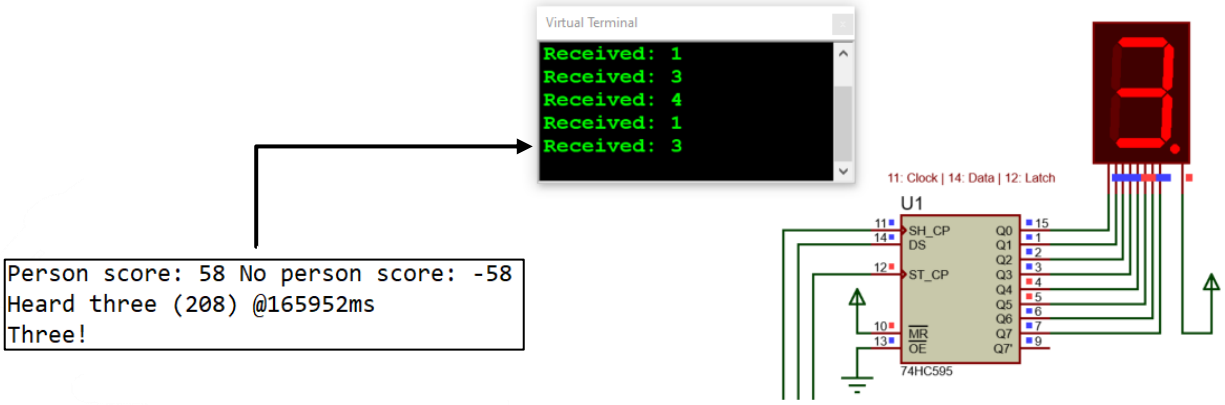


Figure 4.8: Displaying the hardware output on the 7 Segment Display of the simulation.

To simulate the situation of a fire, there are two options available in the simulation. These are:

i. Toggling of the MQ-2 Sensor state to indicate smoke/no-smoke.
ii. Controlling the termperaure input to the LM-35 sensor to indicate high/low temperatures of a given room/floor.

In the simulation, the MQ-2 library gives only a digital output for the presence of smoke. Hence, for the scope of this simulation, we connected the sensor to a digital pin instead of an analog pin. In the actual implementation of the project, the sensor will be interfaces to an analog pin.

The threshold value for the LM-35 is 45^{o}C, beyond which the alarm system will be triggered. Figure 4.9 (a) shows the state of the fire safety system when all parameters are optimal. Figure 4.9 (b) shows the state of the system when the temperature is above the threshold value. Figure 4.9 (c) displays the state of the system when the smoke detector has detected smoke (1 – smoke absent / 0 - smoke present).

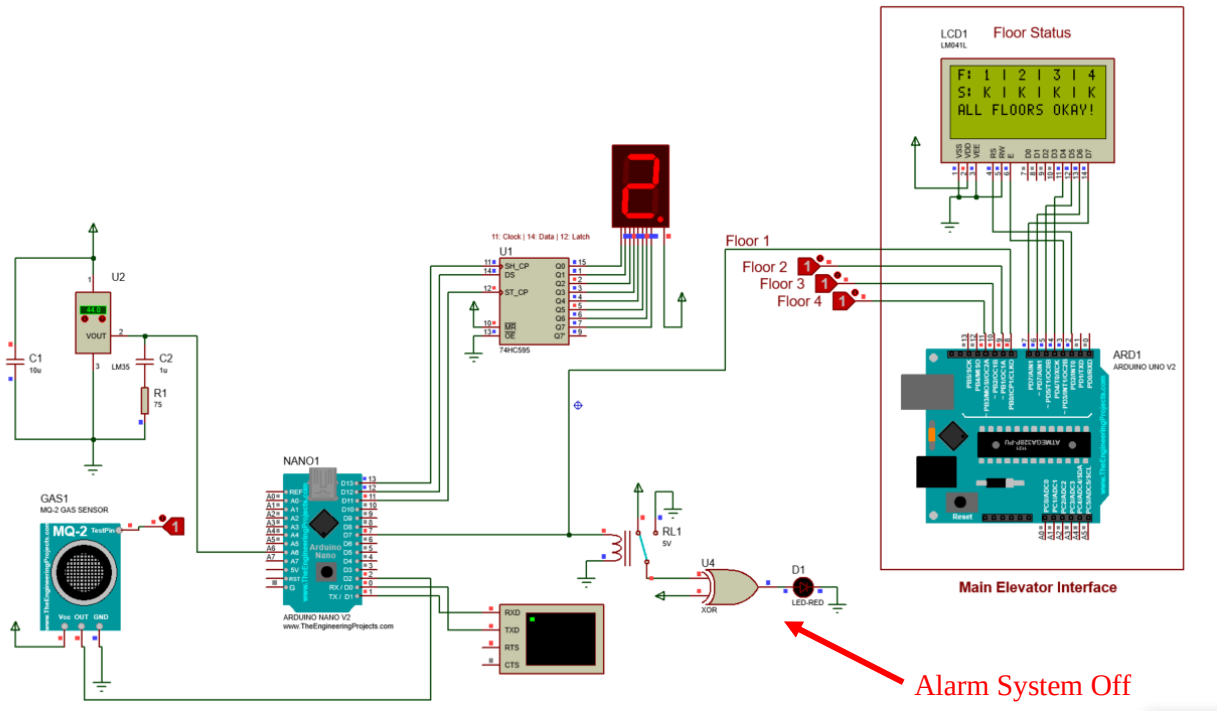


Figure 4.9 (a): Simulation when all parameters are optimal.

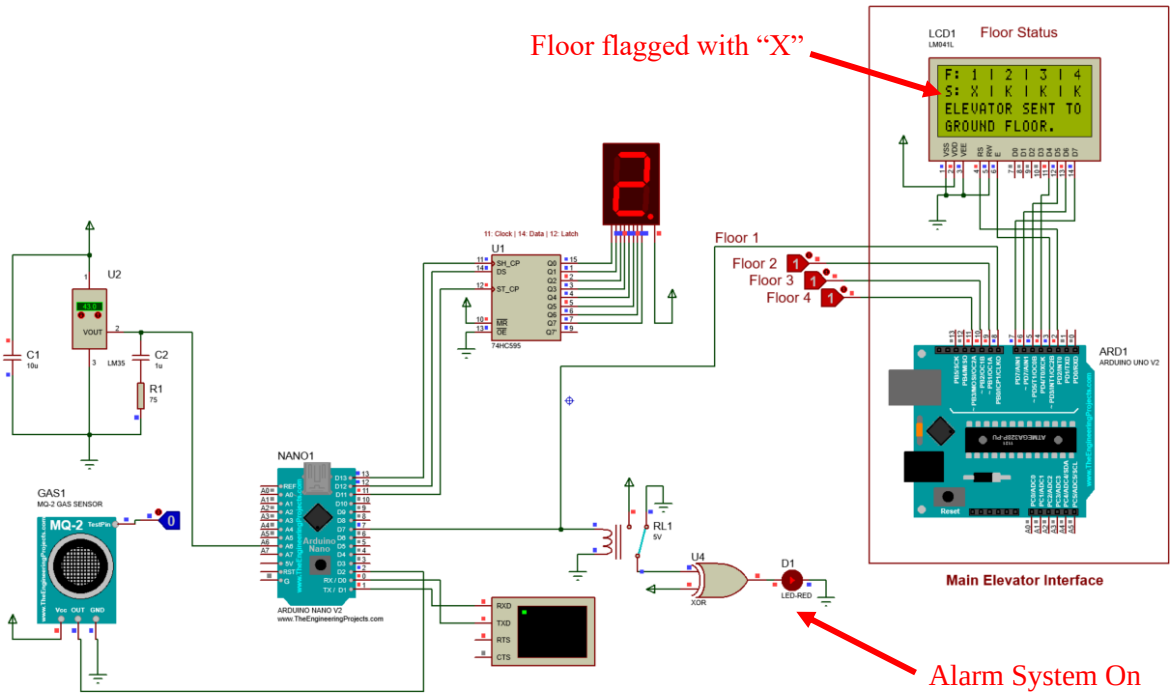


Figure 4.9 (b): Simulation when smoke detector is set to 0 (smoke present).

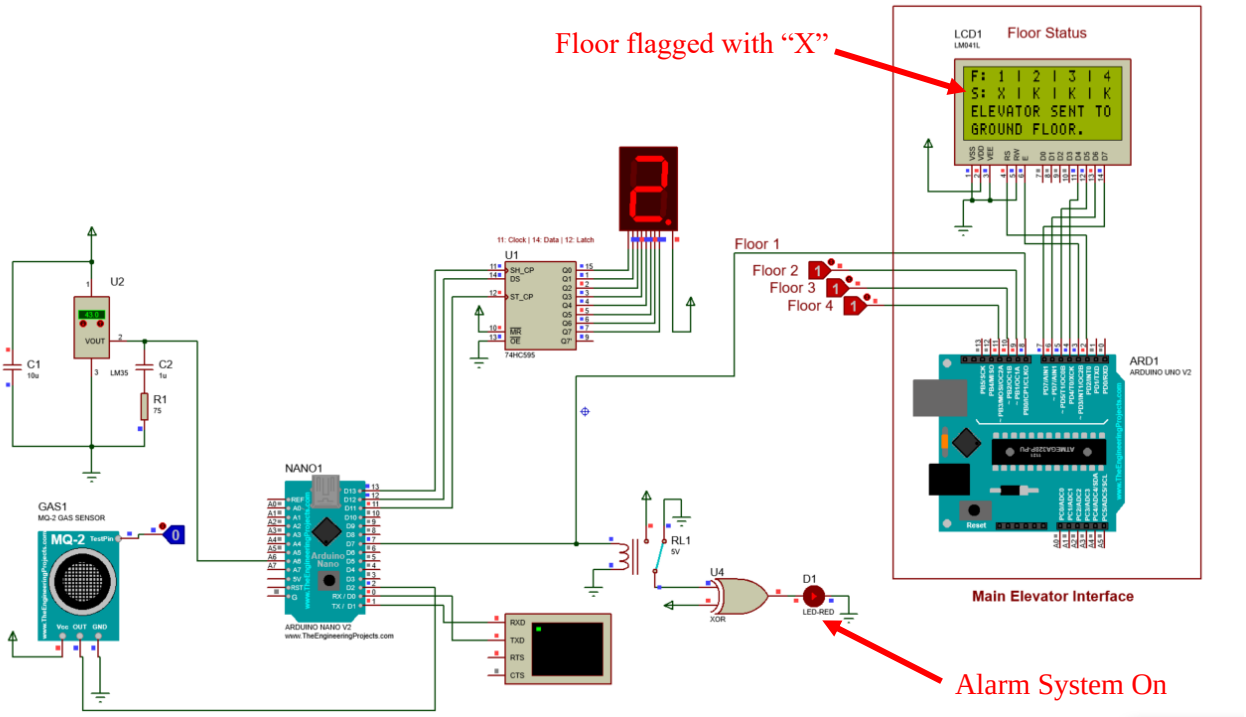


Figure 4.9 (c): Simulation when temperature is above 45°C and smoke detector is set to 0.

Such units will be deployed on all floors of a given building. To simulate the output of multiple such floors, we used logic toggles which represent the state of the floor as a 1 (optimal) and 0 (danger). Figure 4.10 shows the usage of the toggle of floor 3 for example, and the change which is reflected in the LCD display.

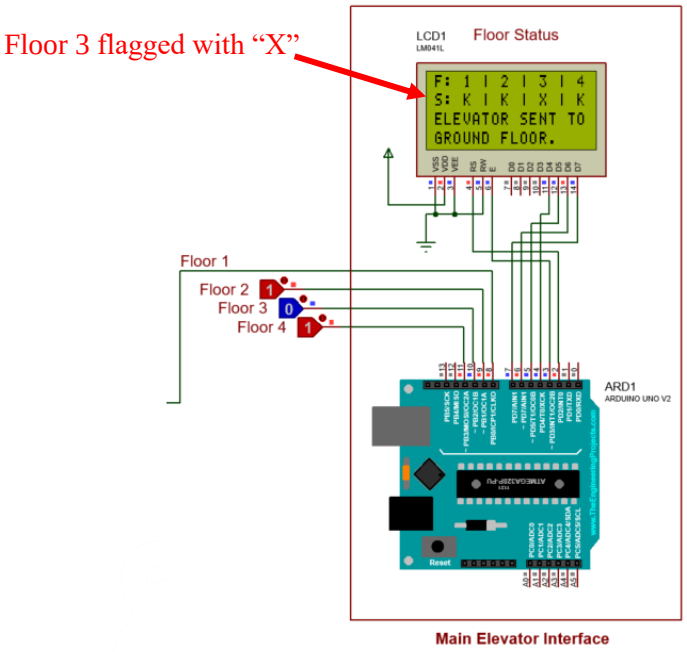


Figure 4.10: Simulation state when an anomaly is detected on other floors (Example: Floor 3).

4.2.1.2. Limitations of the Simulation

In the simulation, the MQ-2 library gives only a digital output for the presence of smoke. Hence, for the scope of this simulation, we connected the sensor to a digital pin instead of an analog pin. In the actual implementation of the project, the sensor will be interfaces to an analog pin.

4.2.2. Fire Safety System Inside The Elevator Unit on TinkerCAD

For a system to be complete, we proposed a comparator circuit within the elevator car which can notify the main interface about the status of smoke within it. We implemented this based on an IC741 Operational Amplifier circuit on TinkerCAD.

We went ahead with TinkerCAD against the choice of more advanced Proteus simulation software due to the analog capability of simulating smoke which was not possible in the latter. TinkerCAD serves as a great platform for basic circuitry such as the smoke detector circuit.

The schematic on a breadboard is shown in Figure 4.11. This circuit was made with reference to the datasheets of IC741 and the MQ-2 Smoke Detector.

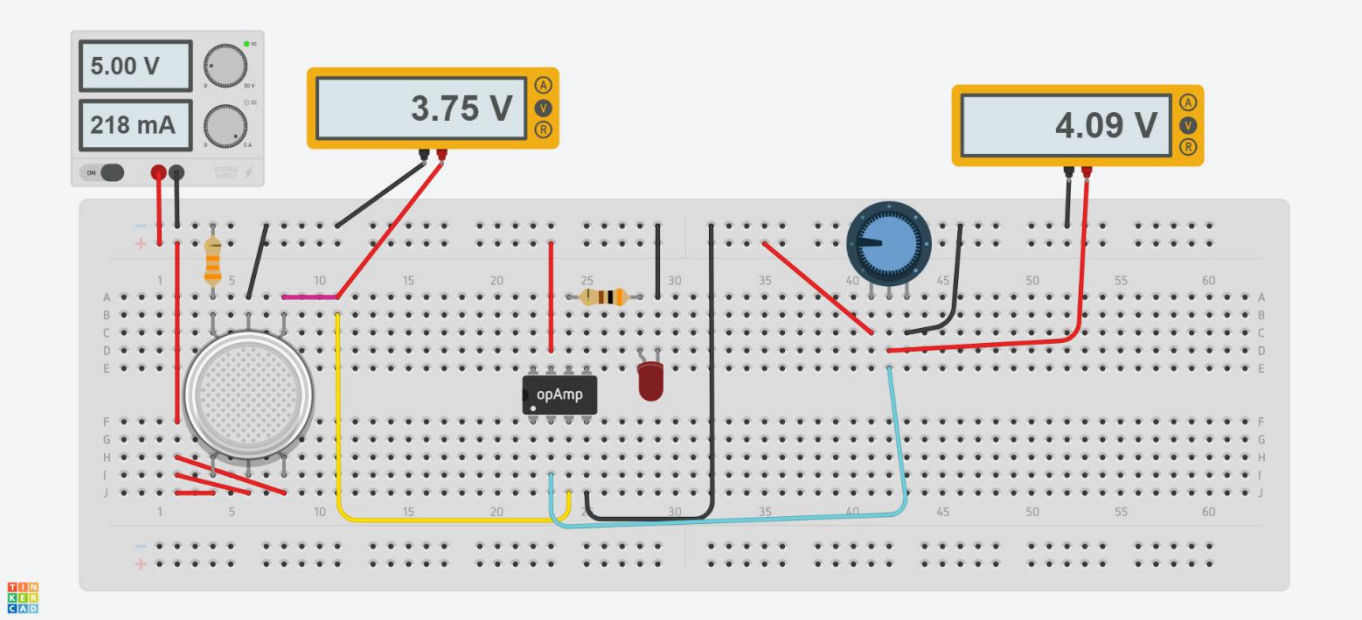


Figure 4.11: Smoke detector circuit using IC741

Through experimentation, the reference voltage of the circuit was set to be +4.09V by adjusting the potentiometer. Figure 4.12 shows the output of the circuit with low levels and high levels of smoke.

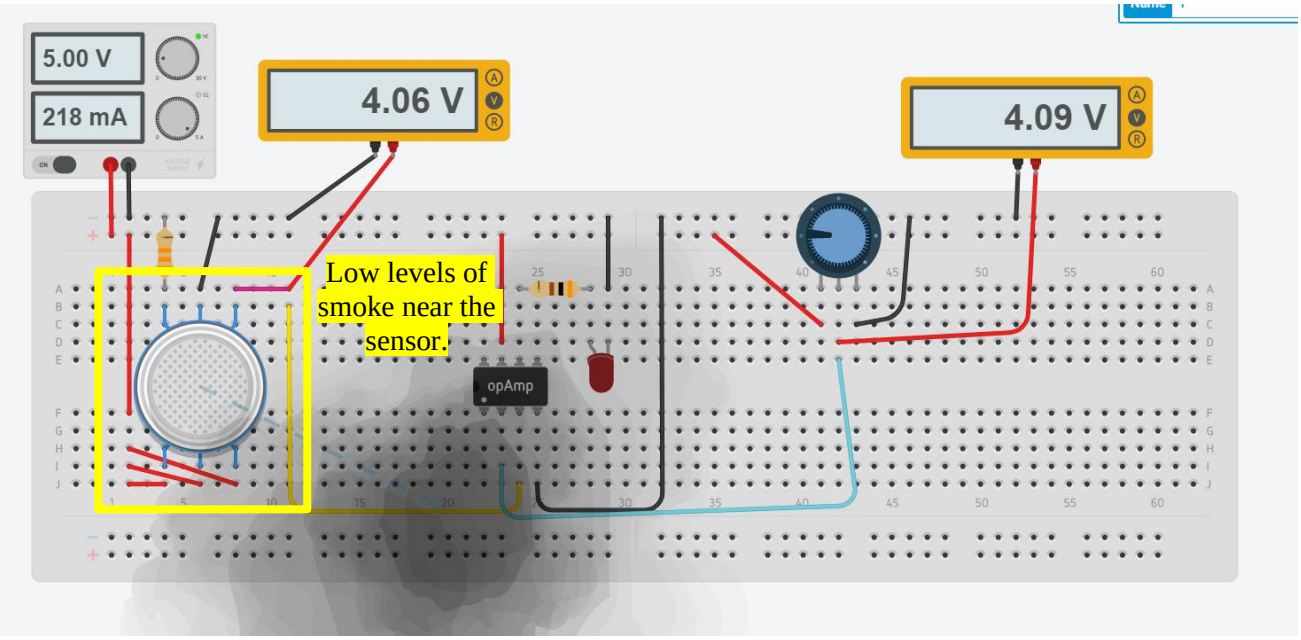


Figure 4.12 (a) Simulation when smoke is detected inside the elevator below threshold.

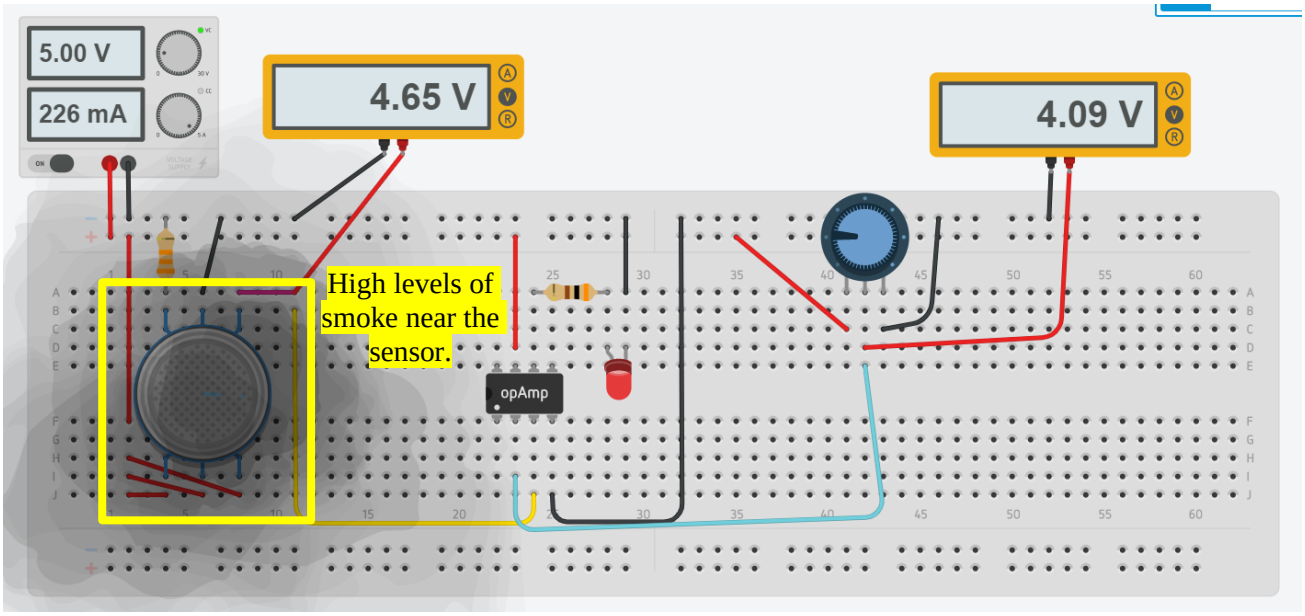


Figure 4.12 (b) Simulation when smoke is detected inside the elevator above threshold.

4.3. PCB Design

To house the components of this project, we created a Printed Circuit Board design using KiCAD, an open-source EDA software. Some key specifications of the PCB are:

1. The PCB consists of 4 Copper layers.
2. Front and Back Copper Layers are signal layers.
3. Inner1 Copper layer contains a Power Plane.
4. Inner2 Copper layer contains a Ground Plane.
5. All components used utilize Through Hole Technology (THT) except for 74AHC1G86 Dual Input Single Output EX-OR Gate which uses SMT.

4.3.1. Schematic

The Eeschema file of the KiCAD build is shown in Figure 4.13. The complete detailed version of the same is also listed as Appendix 1.

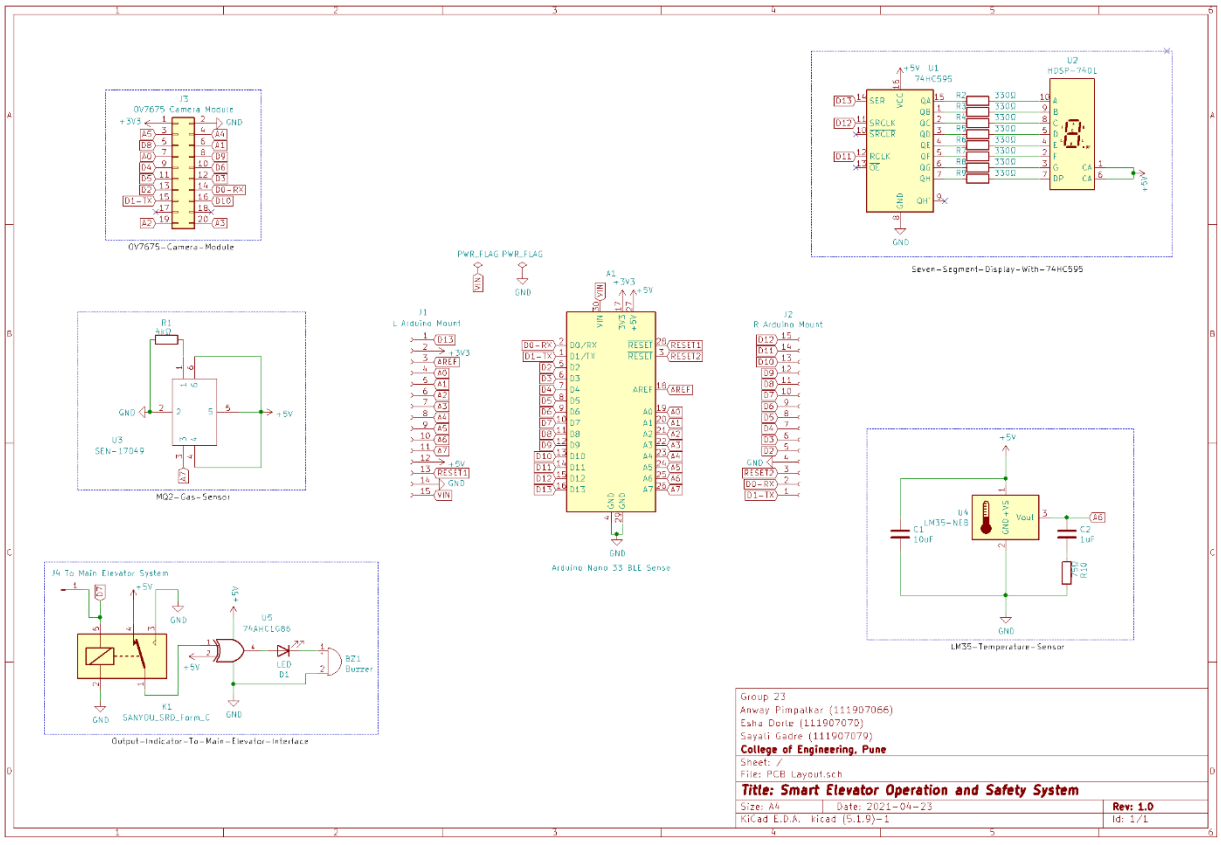


Figure 4.13: PCB Schematic on KiCAD EDA.

4.3.2. Bill of Materials

With reference to building a PCB, a Bill of Materials (BOM) is a comprehensive inventory of the raw materials, assemblies, subassemblies, parts, and components, as well as the quantities of each, needed to manufacture the board. The BOM for the designed PCB is given in Figure 4.14.

Id	Designator	Package	Quantity	Designation
1	K1	Relay_SPDT_SANYOU_SRD_Series_Form_C	1	SANYOU_SRD_Form_C
2	U3	SEN17049	1	SEN-17049
3	U4	TO-220-3_Vertical	1	LM35-NEB
4	U1	DIP-16_W7.62mm	1	74HC595
5	J3	PinSocket_2x10_P2.54mm_Vertical	1	OV7675 Camera Module
6	J1	PinSocket_1x15_P2.54mm_Vertical	1	L Arduino Mount
7	J2	PinSocket_1x15_P2.54mm_Vertical	1	R Arduino Mount
8	U2	HDSP-7401	1	HDSP-7401
9	BZ1	Buzzer_15x7.5RM7.6	1	Buzzer
10	U5	SOT-23-5	1	74AHC1G86
11	J4	PinHeader_1x01_P2.54mm_Vertical	1	To Main Elevator System
12	R1	R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal	1	4kΩ
13	R2,R3,R4,R5,R6,R7,R8,R9	R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal	8	330Ω
14	C1	C_Disc_D3.0mm_W1.6mm_P2.50mm	1	10uF
15	C2	C_Disc_D3.0mm_W1.6mm_P2.50mm	1	1uF
16	R10	R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal	1	75Ω
17	D1	LED_D5.0mm_FlatTop	1	LED

Figure 4.4: Bill of Materials for the PCB

4.3.3. PCB Layout

As mentioned, the PCB is built of 4 copper layers. The layers from front to back are:

1. Front Cu Signal Layer
2. Power Plane Layer
3. Ground Plane Layer
4. Back Cu Signal Layer

The layer schematics are as shown in Figure 4.15.

.

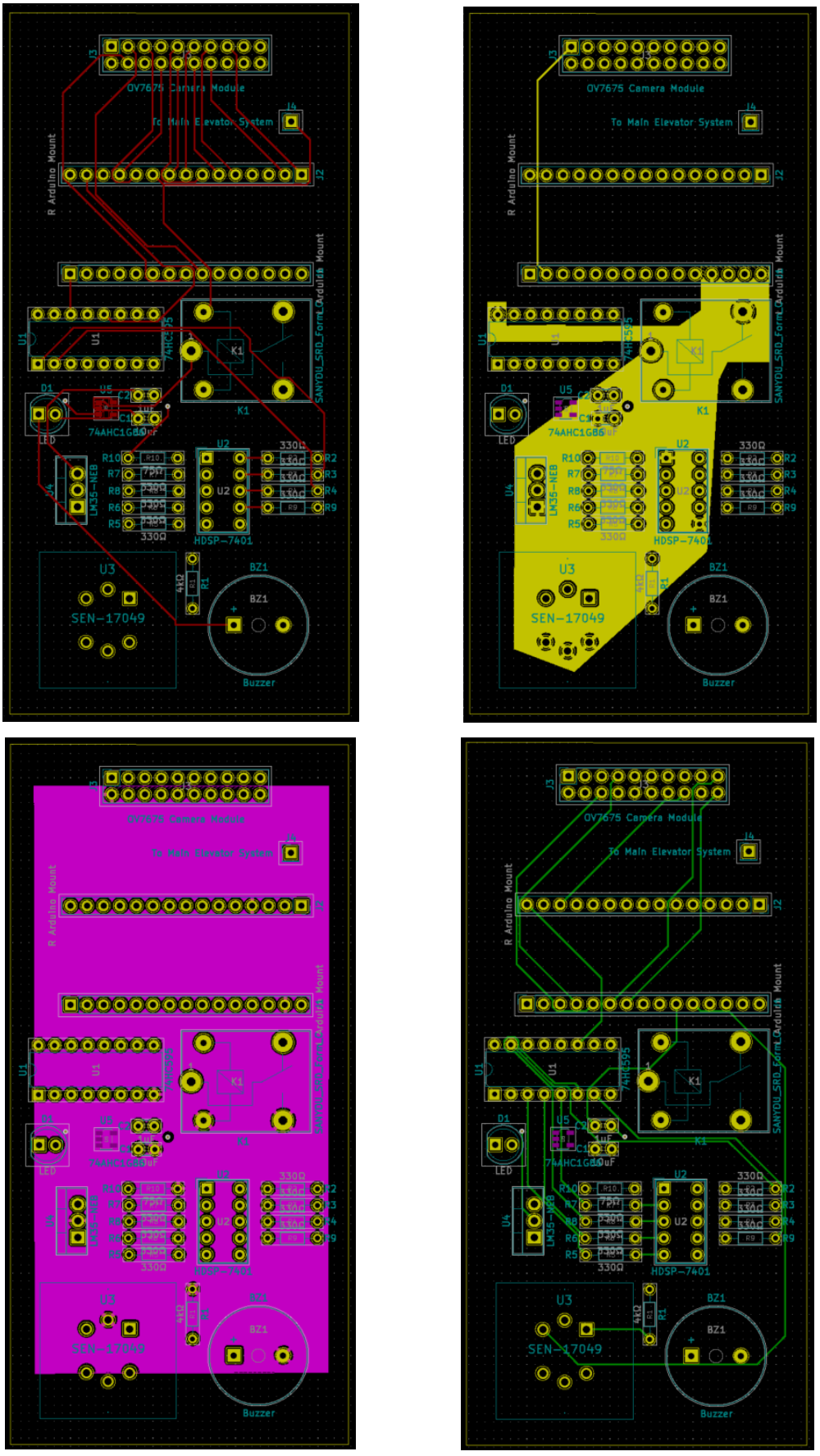


Figure 4.15: Four Copper layers of designed PCB.

4.3.4. 3D Renders of PCB

Multiple angles of the 3D render generated by KiCAD are shown in Figure 4.16.

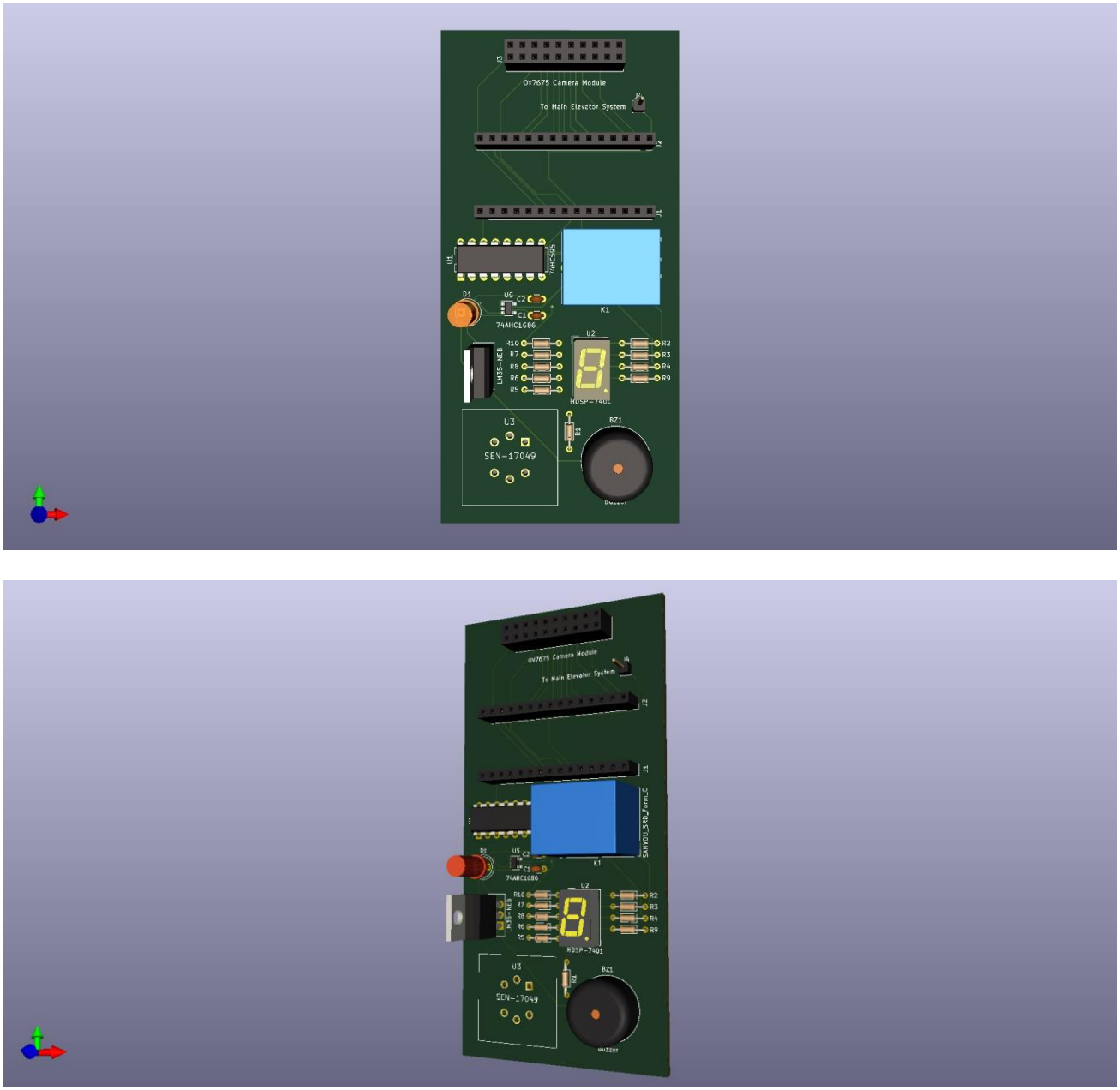


Figure 4.16: 3D View of the designed PCB.

4.4 Conclusion

Smart Elevator Systems Using Embedded Machine Learning and Fire Safety Mechanisms contributes to the domain of conventional elevators to improve current safety systems on various levels. The standard ways of implementing safety mechanisms include trap doors, phone systems inside the elevator for emergencies. Our project upgrades these systems in accordance to current needs. We have implemented machine learning to cater to the tasks based on keyword spotting and computer vision. Taking advantage of the TinyML system we used, we could do more with less cost incurred. We built a VWW model for the purpose of person detection. To create a speech recognition model, we had to extract certain features from the microphone signals, for which we employed a couple of ways, namely FFTs, Spectrograms and MFCCs. Once the user speaks, the raw data is captured and its FFT is generated, The Discrete Fourier Transform of the data sampled over in a small unit of time makes up a small slice of the spectrogram therefore after taking multiple such Fourier Transforms of different timestamps of the incoming signal, we are able to form a single spectrogram which represents the variation of volume and frequency over time much better. We then generate a MFCC of the audio signal, and feed this to our neural network to get a much higher accuracy because of the extracted features than we would have using just the raw audio data. Thus, our model with its increased efficiency enables more automation for the comfort and safety of the users, allowing them to experience a touch-free elevator ride. Studies have shown that elevators elicit a form of fear in the population despite being normalized to a huge extent. This fear is caused because of the possible safety issues that they may compromise on in the status quo. In our model, we aimed to change that. We have also inculcated a smoke detector within the elevator which detects smoke and alerts the people inside the elevator, giving them buffer time to move out of the elevator and our fire safety system outside the elevator, which detects smoke and safety hazards outside the elevators and alerts the people. We therefore provide a holistic solution to everyday elevator-related problems faced.

References

[1] Prof. Omkar M. Shete, Ms. Amrita A. Vitekar, Ms. Ashwini N. Patil, "Design & Control of an Elevator Control System using PLC," *International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering*, vol. 5, 2017.

[2] Max T. Kinateder, Hidemi Omori, Erica D. Kuligowski "The Use of Elevators for Evacuation in Fire Emergencies in International Buildings", July 2014

[3] Arduino Nano 33 BLE Sense Schematics:
https://content.arduino.cc/assets/NANO33BLE_V2.0_sch.pdf

[4] Arduino Nano 33 BLE Sense Datasheet:
https://content.arduino.cc/assets/Nano_BLE_MCU-nRF52840_PS_v1.1.pdf

[5] OV7675 Camera Module Datasheet:
https://www.uctronics.com/download/Image_Sensor/OV7675_DS.pdf

[6] MQ-2 Gas Sensor Datasheet:
https://cdn.sparkfun.com/assets/3/b/0/6/d/MQ-2.pdf

[7] LM-35 Temperature Sensor Datasheet:
https://www.ti.com/lit/ds/symlink/lm35.pdf

[8] Common Anode Seven Segment Display:
https://docs.broadcom.com/docs/AV02-2553EN

[9] 74HC595 Shift Register Datasheet:
https://www.ti.com/lit/ds/symlink/sn74hc595.pdf

[10] 74AHC1G86 EX-OR Gate IC Datasheet:
https://www.diodes.com/assets/Datasheets/74AHC1G86.pdf

[11] IC741 Operational Amplifier Datasheet:
https://www.ti.com/lit/ds/symlink/lm741.pdf

[12] IC741 Operational Amplifier Datasheet:
https://www.ti.com/lit/ds/symlink/lm741.pdf

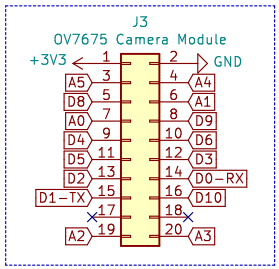
[13] TensorFlow and Embedded Machine Learning:
https://www.edx.org/professional-certificate/harvardx-tiny-machine-learning

[14] Daniel Situnayake, Pete Warden *"TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers"*, December 2019

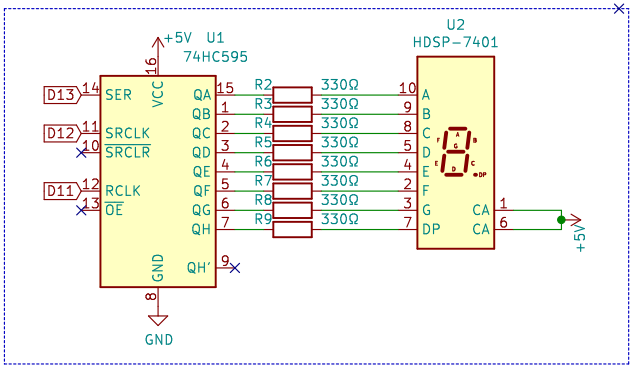
Appendix

[A] PCB Design Schematic on KiCAD EDA

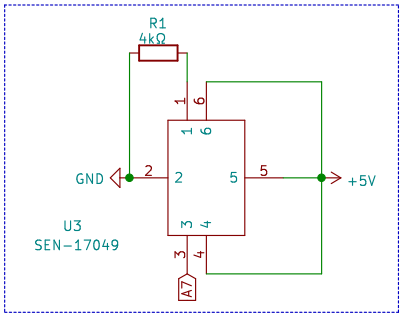
[B] PCB pcbnew_ Design on KiCAD EDA



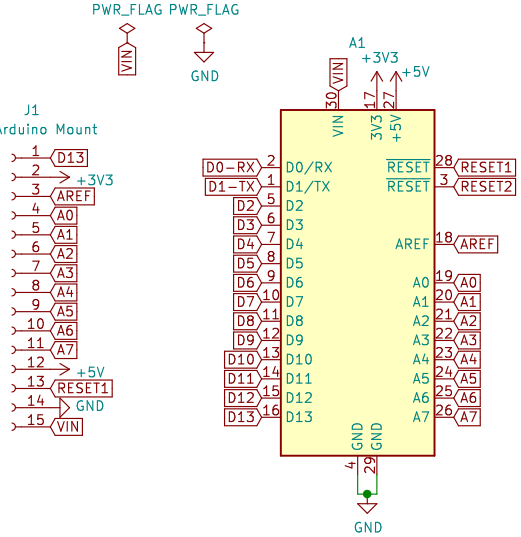
OV7675–Camera–Module

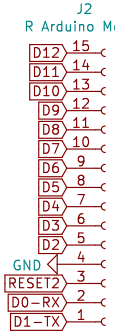


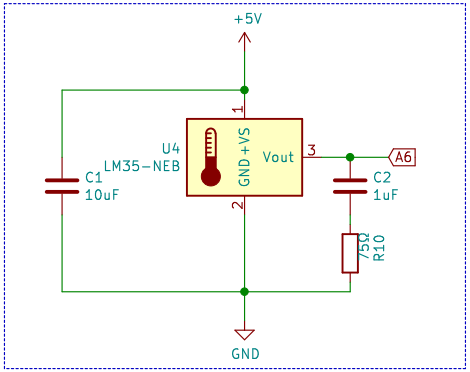
Seven–Segment–Display–With–74HC595



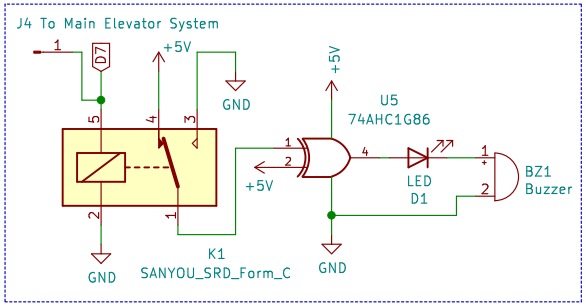
MQ2–Gas–Sensor







LM35–Temperature–Sensor



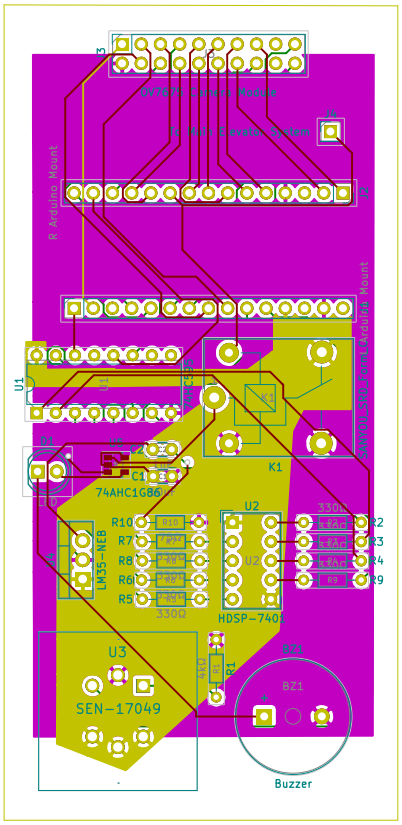
Output–Indicator–To–Main–Elevator–Interface

Group 23
Anway Pimpalkar (111907066)
Esha Dorle (111907070)
Sayali Gadre (111907079)
College of Engineering, Pune

Sheet: /
File: PCB Layout.sch

Title: Smart Elevator System

Size: A4	Date: 2021–04–23	**Rev: 1.0**
KiCad E.D.A. kicad (5.1.9)–1		Id: 1/1



Group 23 Anway Pimpalkar (111907066) Esha Dorle (111907070) Sayali Gadre (111907079) **College of Engineering, Pune**		
Sheet: File: PCB Layout.kicad_pcb		
Title: Smart Elevator Systems		
Size: A4	Date: 2021-04-23	**Rev: 1.0**
KiCad E.D.A. kicad (5.1.9)-1		Id: 1/1