EEL 7150: Embedded System Design

Project Report

On-board Word Recognition and Sensor Monitoring System for Relay Control

Submitted By

Arashdeep Singh (M23IRM003)

Chaitanya Patil (M23IRM004)

Parag Chourasia (M23IRM010)

Contents:-

Introduction

Problem Statement

In this project, we are trying to implement a word recognition system to control appliances.The main purpose of the project is to use a low-power microcontroller with more computational power and capability to run machine learning models, which will help to control many devices. The major benefit of this project will be that a low power device can control high power devices. So that they donot have to continuously look for user commands. Even a small device can help them to work smarter.

Here is the main segemenrts on which our project relies.:-

- **Intelligent Voice Control**: Integrate a speech recognition engine optimized for microcontrollers, allowing users to control appliances through spoken commands.
- **Sensor-Driven Automation**: Equip the system with relevant sensors (temperature, light, etc.) to gather real-time environmental data. This data can be used independently to trigger actions (e.g., light turns on with motion) or combined with voice commands for more sophisticated control.
- **Relay Power Management**: Utilize relays to bridge the gap between the microcontroller's low-power signals and the higher power requirements of appliances. Ensure chosen relays are properly sized for safe operation.

Benefits:-

- **Low-Power Efficiency**: The project capitalizes on a low-power microcontroller, promoting energy-conscious smart home automation.
- **Scalability and Versatility**: This system can be adapted to control various appliances by incorporating different sensors and tailoring the machine learning model for specific use cases.
- **Enhanced User Experience**: Voice control and sensor-driven automation offer a convenient and intuitive way to manage your home environment.

Device chosen :-

Arduino Nano 33 BLE Sense Rev 2

The Arduino Nano 33 BLE Sense Rev2 combines a tiny form factor, different environment sensors and the possibility to run AI using TinyML and TensorFlow™ Lite. Whether you are looking at creating your first embedded ML application or you want to use Bluetooth® Low Energy to connect your project to your phone, the Nano 33 BLE Sense Rev2 will make that journey easy.

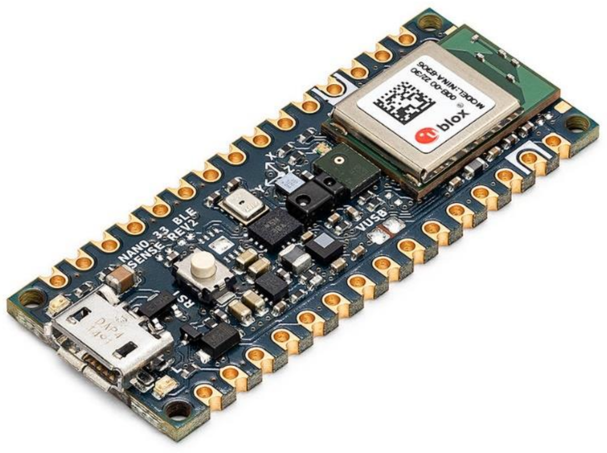


Fig Arduino Nano 33 BLE Sense Rev 2

Why we use this device?

1. Low-Power Efficiency:
   - The Nano 33 BLE Sense is designed to be power-efficient, making it suitable for battery-powered or energy-conscious applications. It features a low-power ARM Cortex-M4 processor and power management capabilities that help minimize energy consumption.
2. Computational Power:
   - Despite its compact size, the Nano 33 BLE Sense packs a punch in terms of computational power. It is equipped with a powerful ARM Cortex-M4 processor running at 64 MHz, providing sufficient processing power to execute machine learning models for speech recognition and sensor data processing.
3. Sensor Integration:
   - The Nano 33 BLE Sense comes with built-in sensors that are relevant to your project, including an accelerometer, gyroscope, temperature sensor, humidity sensor, pressure sensor, and microphone. These sensors enable the device to gather real-time environmental data necessary for sensor-driven automation.
4. Bluetooth Connectivity:
   - As indicated by its name, the Nano 33 BLE Sense features Bluetooth Low Energy (BLE) connectivity. This allows the device to communicate wirelessly with other BLE-enabled devices, such as smartphones or smart home hubs, expanding its capabilities and enabling remote control and monitoring of appliances.

5. Machine Learning Support:

   - The Nano 33 BLE Sense supports machine learning applications through the integration of the TensorFlow Lite for Microcontrollers library. This allows you to deploy machine learning models directly onto the device, enabling intelligent voice control and other AI-driven functionalities without relying on external processing.

6. Relay Power Management:

   - While the Nano 33 BLE Sense itself does not include relay modules, it can interface with external relay modules or transistors to control appliances with higher power requirements. Its GPIO pins can be used to send control signals to relays, allowing safe operation and management of appliances.

We have decided to divide our project in four segments. So it becomes easy, and all of us can work on different parts simultaneously.

Work Flow:-

1. Data Collection and Formatting
2. Model Training Testing
3. Model Deployment
4. Hardware Integration

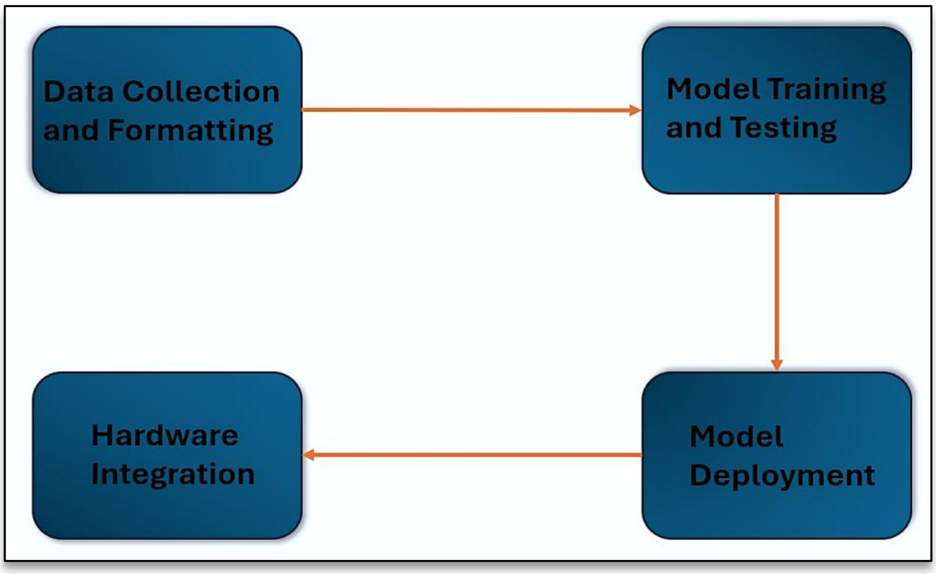


Fig Workflow of the Project

1. Data Collection and Formatting

A. Data Collection

- Voice samples from our classmates for the following keywords.:-
  - Light On, Light Off, Temperature, Pressure, Jalao(जालाओ),Bujao(बुजाओ) is collected.

- Simply phone recorder is used to collect samples. Here, participants were asked to speak keywords in different ways so that a robust model can be prepared by training on various forms of same command.

B. DataSet Making

We have used audacity software in this process. First, we have the collected samples in 1-second frames, for each time. The file format after this we got is .wav. After that we have got roughly 100 files for each keyword.

- Adding Noise
  - To make model robust so that it can sustain noisy situations we have mixed noise with collected samples so that it can work more robustly.
- Multiplexing dataset
  - As to get good performance from model it is advisable to train model on large data and data should be repeated. Going by this logic we have multiplexed it by 15. So we have got 1500 files for each segmentated audio file.
- We have used a Python tool to implement this thinking. The python code is attached with report.
  - Using noiseandmultiplexing.py

Here are the steps of how it works:

1. It takes a list of target words, directories containing audio samples of those words, a directory containing background noise files, and other options as input.

2. It creates a number of subdirectories in the output directory, one for each target word and one for unknown words.

3. It mixes audio samples from the background noise directory with audio samples from the target words directories. The mixed audio samples are then saved in the corresponding subdirectory in the output directory.

4. It also creates a subdirectory for unknown words and mixes audio samples from the background noise directory with audio samples from unknown words directories.

The reason for doing this is to create a dataset of audio samples that are more representative of the real world, where spoken words are often heard in the presence of background noise. This can help to improve the accuracy of keyword spotting models.

2. Model Training Testing

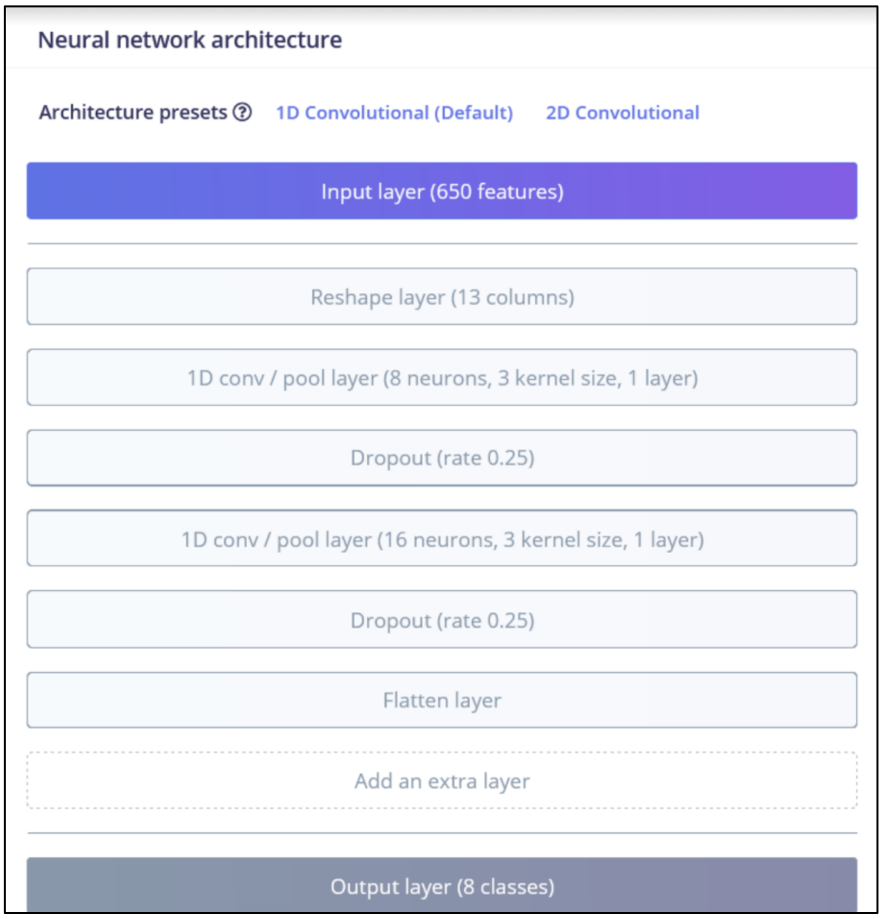
A. Model Making

The model takes in one second’s worth of data at a time. It outputs four probability scores, one for each of the chosen classes, predicting how likely it is that the data represents one of them. However, the model does not take in raw audio sample data. Instead, it works with log power spectrums.

Used signal processing to extract feature.

Window size means duration of audio file which is taken as 1 sec.

Frequency of each audio file is 16000Hz.


Neural network architecture

Architecture presets: 1D Convolutional (Default) | 2D Convolutional

Input layer (650 features)

Reshape layer (13 columns)

1D conv / pool layer (8 neurons, 3 kernel size, 1 layer)

Dropout (rate 0.25)

1D conv / pool layer (16 neurons, 3 kernel size, 1 layer)

Dropout (rate 0.25)

Flatten layer

Add an extra layer

Output layer (8 classes)

Fig Neural Network Architecture

The layers of model are :-

- **Reshape Layer:**
  - Reshapes the input data into a suitable format for the convolutional layers.
- **Conv1D Layer (with 8 filters):**
  - Applies convolution operation with 8 filters and a kernel size of 3.
- **MaxPooling1D Layer**:
  - Performs max pooling operation with a pool size of 2 and strides of 2.
- **Dropout Layer**:
  - Applies dropout regularization to prevent overfitting (dropout rate of 0.25).
- **Conv1D Layer (with 16 filters):**
  - Another convolutional layer with 16 filters and a kernel size of 3.
- **MaxPooling1D Layer:**
  - Another max pooling operation with the same parameters as before.
- **Dropout Layer**:
  - Another dropout layer with the same dropout rate.
- **Flatten Layer**:
  - Flattens the output of the previous layers into a one-dimensional vector.
- **Dense Layer (Output Layer):**
  - Fully connected layer with 'classes' number of neurons, using softmax activation for multi-class classification.

So, the network architecture has a total of 8 layers:

1) Conv1D layers (2)
2) MaxPooling1D layers (2)
3) Dropout layers (2)
4) Flatten layer (1)
5) Dense (Output) layer (1)

The number of neurons in each layer is as follows:

- Input Layer: Depends on the length of the input data, determined by input_length.
- Conv1D Layer 1: 8 filters
- MaxPooling1D Layer 1: No neurons, as it performs pooling.
- Dropout Layer 1: No neurons, as it performs regularization.
- Conv1D Layer 2: 16 filters
- MaxPooling1D Layer 2: No neurons, as it performs pooling.
- Dropout Layer 2: No neurons, as it performs regularization.
- Flatten Layer: Flattens the output of the previous layer.
- Dense (Output) Layer: 'classes' number of neurons, which is the number of output classes for classification.

How model will work?

It takes audio samples and converts them to MFCC files. So when this model is deployed it converts the given audio input to MFCC. Now, input converted MFCC files will be compared with already MFCC files with which the model is trained. Here is very simplified flow describing working of model:-

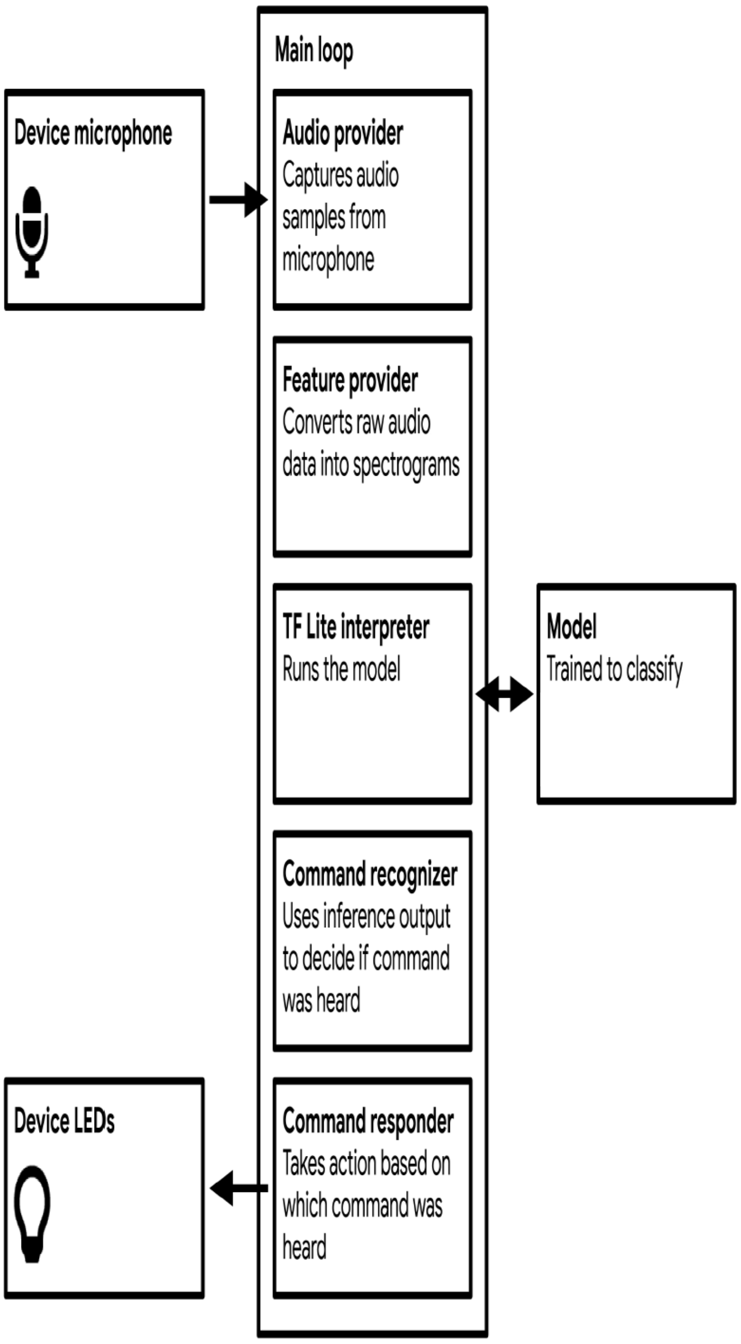

Main loop
Device microphone
Audio provider
Captures audio samples from microphone
Feature provider
Converts raw audio data into spectrograms
TF Lite interpreter
Runs the model
Model
Trained to classify
Command recognizer
Uses inference output to decide if command was heard
Device LEDs
Command responder
Takes action based on which command was heard

Fig Block Diagram of the Audio Recognition

How audio is converted to MFCC files?

MFCC (Mel-Frequency Cepstral Coefficients) is a feature extraction technique widely used in speech and audio processing. It captures the spectral characteristics of sound in a way that is well-suited for various machine-learning tasks, such as speech recognition and music analysis. In simpler terms, MFCCs are a set of coefficients that capture the shape of the power spectrum of a sound signal. MFCC stands for it is an acronym for **Mel Frequency Cepstral Co-efficients** which are the coefficients that collectively make up an MFC. **MFC** is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency. This is similar to JPG format for images.

Here is a step-by-step breakdown of the MFCC technique :

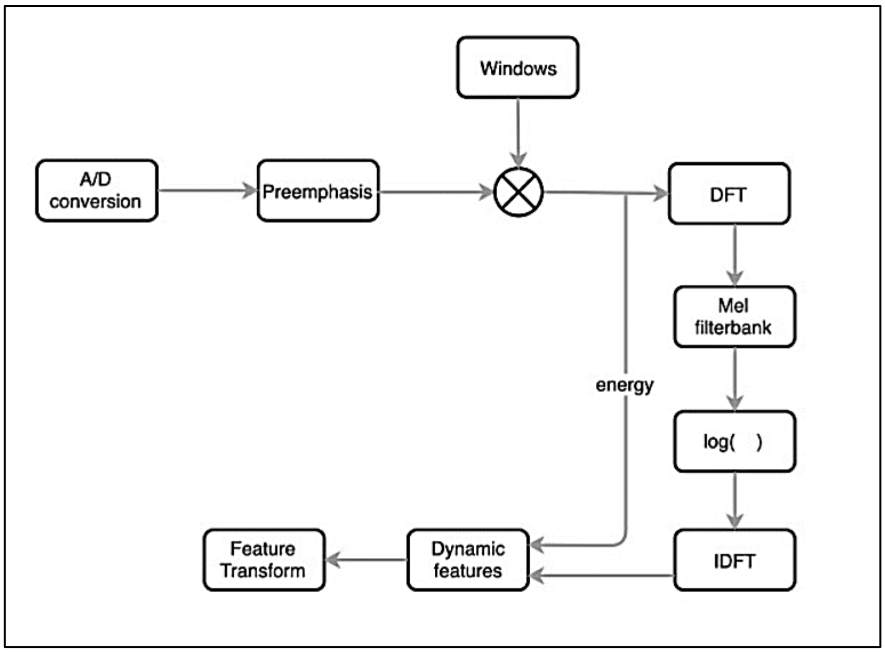


Fig Flowchart of the MFCC technique

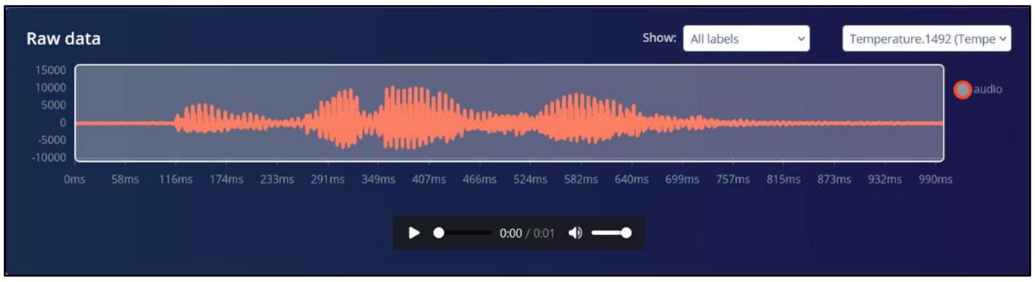


Fig Raw Audio file

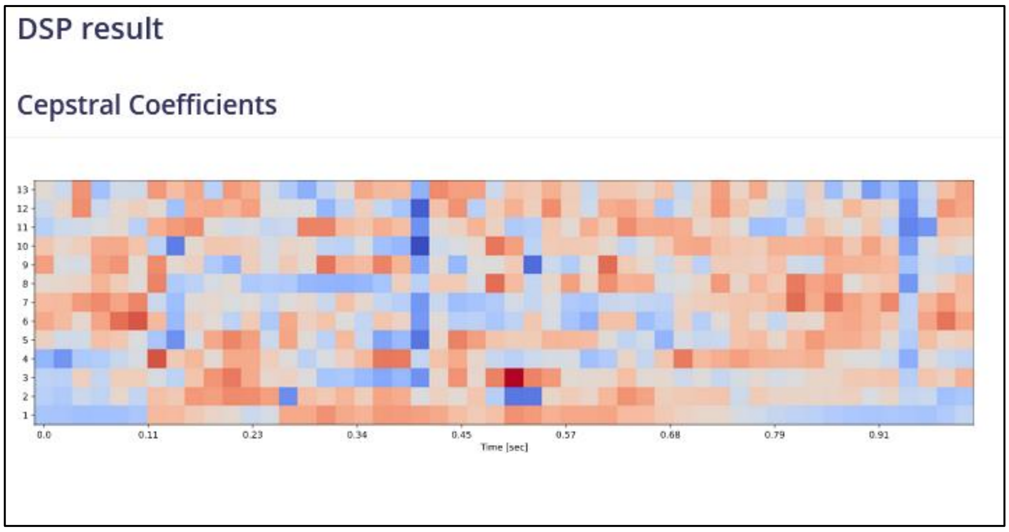


Fig MFCC Final Output

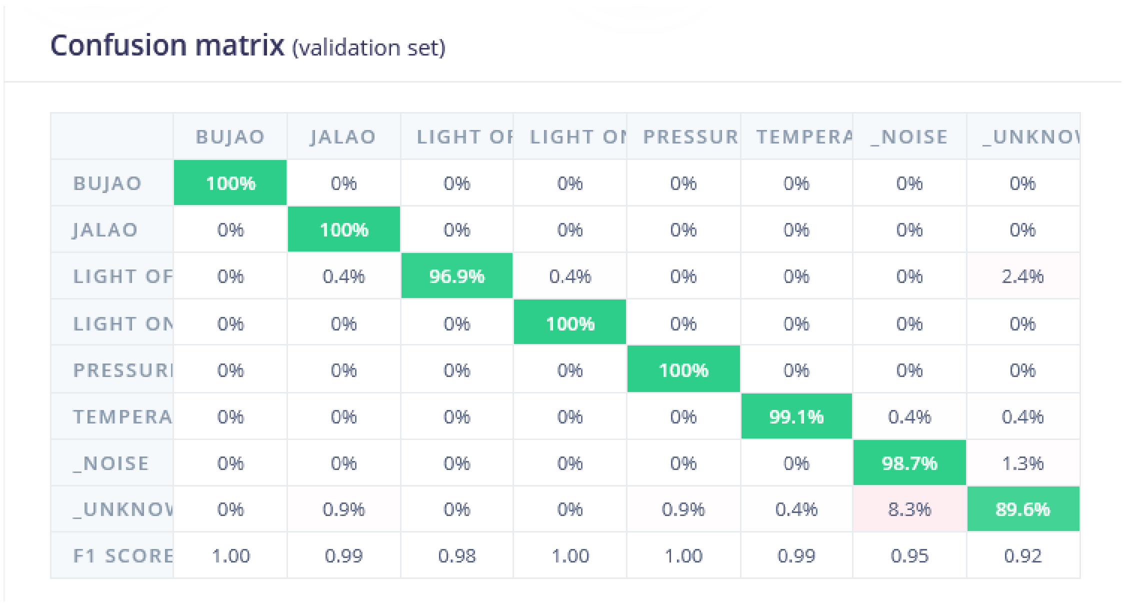
B. Model Training

Here basically collected data set is fed to created model. 80% of all keyword data will be fed at this stage.

The model adjusts its internal parameters to minimize the difference between its predictions and the actual values (the labels of the training data). Classification of keyword file using supervised learning. Edge impulse platform with tensor flow lite is used for this purpose.

A code for model training is provided in this link.

This code is a Python script designed to train a convolutional neural network (CNN) using TensorFlow and Keras. It begins by importing the necessary libraries, including TensorFlow and modules from Keras for building and training neural networks. Key constants such as the number of epochs, learning rate, batch size, and a flag for ensuring deterministic behavior during training are defined. Following this, the script processes the training and validation datasets, shuffling them if non-deterministic behavior is allowed and batching them according to the specified batch size. The model architecture is then defined using a Sequential model from Keras, comprising convolutional layers with max pooling and dropout for feature extraction, followed by a dense layer with softmax activation for classification. An Adam optimizer is employed for training, and a custom callback, likely for logging training progress, is appended. The model is compiled with categorical cross-entropy loss and trained using the fit() method, with validation data provided for monitoring performance. Additionally, there's a flag to enable or disable per-channel quantization for the model, potentially impacting memory usage and accuracy. Overall, this script encapsulates the process of building, training, and evaluating a CNN for classification tasks using TensorFlow and Keras.



Confusion matrix (validation set)

	BUJAO	JALAO	LIGHT OF	LIGHT OI	PRESSUR	TEMPERA	_NOISE	_UNKNOV
BUJAO	100%	0%	0%	0%	0%	0%	0%	0%
JALAO	0%	100%	0%	0%	0%	0%	0%	0%
LIGHT OF	0%	0.4%	96.9%	0.4%	0%	0%	0%	2.4%
LIGHT ON	0%	0%	0%	100%	0%	0%	0%	0%
PRESSURI	0%	0%	0%	0%	100%	0%	0%	0%
TEMPERA	0%	0%	0%	0%	0%	99.1%	0.4%	0.4%
_NOISE	0%	0%	0%	0%	0%	0%	98.7%	1.3%
_UNKNOV	0%	0.9%	0%	0%	0.9%	0.4%	8.3%	89.6%
F1 SCORE	1.00	0.99	0.98	1.00	1.00	0.99	0.95	0.92

Fig Confusion Matrix for Validation Set

A confusion matrix is a performance measurement tool for machine learning classification algorithms. It is a table that is used to evaluate the performance of a classification model by summarizing the number of correct and incorrect classifications for each class.

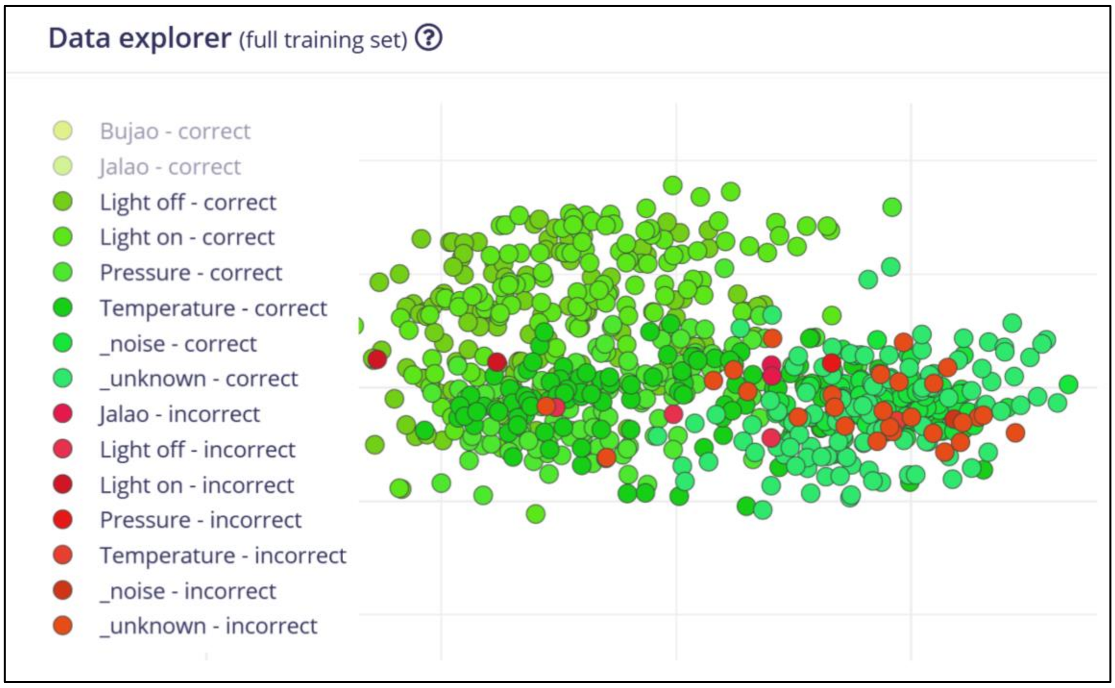
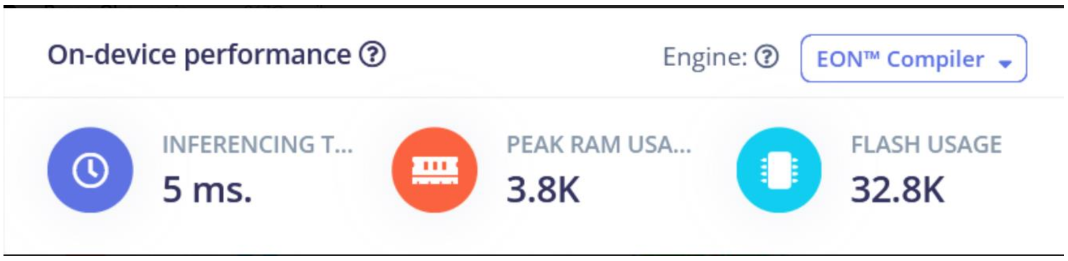


Fig Point chart showing the data of Full Training Set.



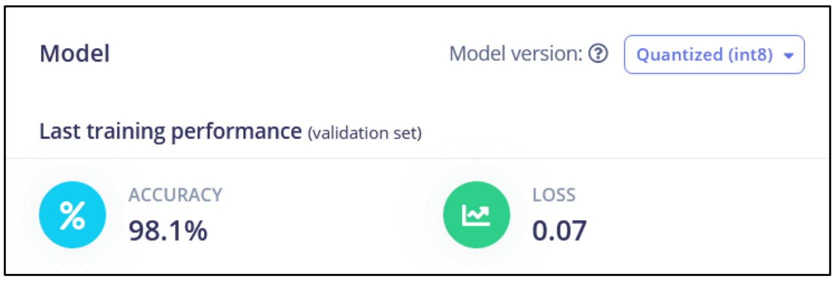
On-device performance

Engine: EON™ Compiler

INFERENCING T... 5 ms.

PEAK RAM USA... 3.8K

FLASH USAGE 32.8K



Model

Model version: Quantized (int8)

Last training performance (validation set)

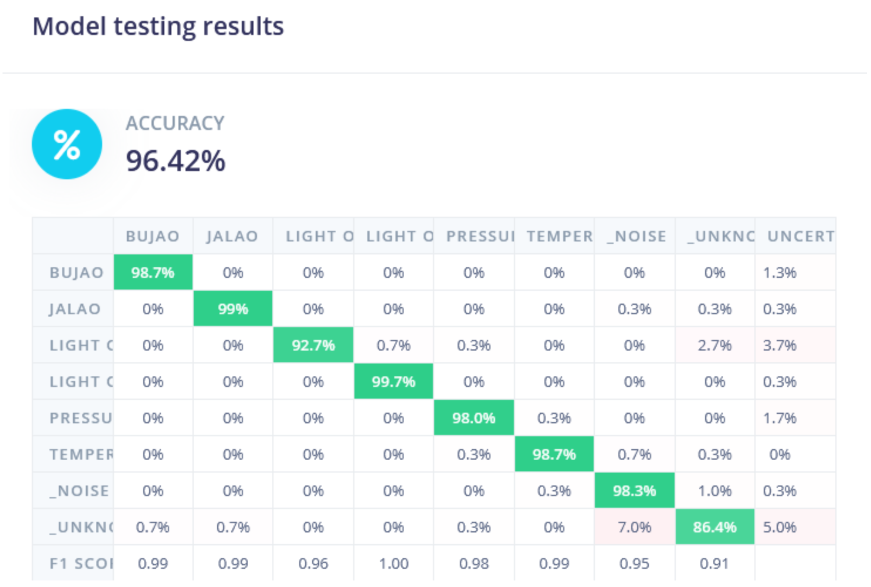
ACCURACY 98.1%

LOSS 0.07

Fig Model Accuracy and On-device Performance

C. Model Testing

Rest 20% keyword data will be fed to check accuracy and result as giving a test to created system.This data is not used during training, so it provides a measure of how well the model can generalize to new, unseen data.



Model testing results

ACCURACY 96.42%

	BUJAO	JALAO	LIGHT O	LIGHT O	PRESSUI	TEMPER	_NOISE	_UNKNC	UNCERT
BUJAO	98.7%	0%	0%	0%	0%	0%	0%	0%	1.3%
JALAO	0%	99%	0%	0%	0%	0%	0.3%	0.3%	0.3%
LIGHT (	0%	0%	92.7%	0.7%	0.3%	0%	0%	2.7%	3.7%
LIGHT (	0%	0%	0%	99.7%	0%	0%	0%	0%	0.3%
PRESSU	0%	0%	0%	0%	98.0%	0.3%	0%	0%	1.7%
TEMPER	0%	0%	0%	0%	0.3%	98.7%	0.7%	0.3%	0%
_NOISE	0%	0%	0%	0%	0%	0.3%	98.3%	1.0%	0.3%
_UNKN(	0.7%	0.7%	0%	0%	0.3%	0%	7.0%	86.4%	5.0%
F1 SCOI	0.99	0.99	0.96	1.00	0.98	0.99	0.95	0.91	

Fig Confusion matrix showing Overall Accuracy

3. <u>Model Deployment</u>

To deploy the model on an Arduino Nano 33 BLE Sense board, you'll need to convert it into a format compatible with the board's hardware constraints. This may involve optimizing the model architecture and parameters for efficient execution on the microcontroller.

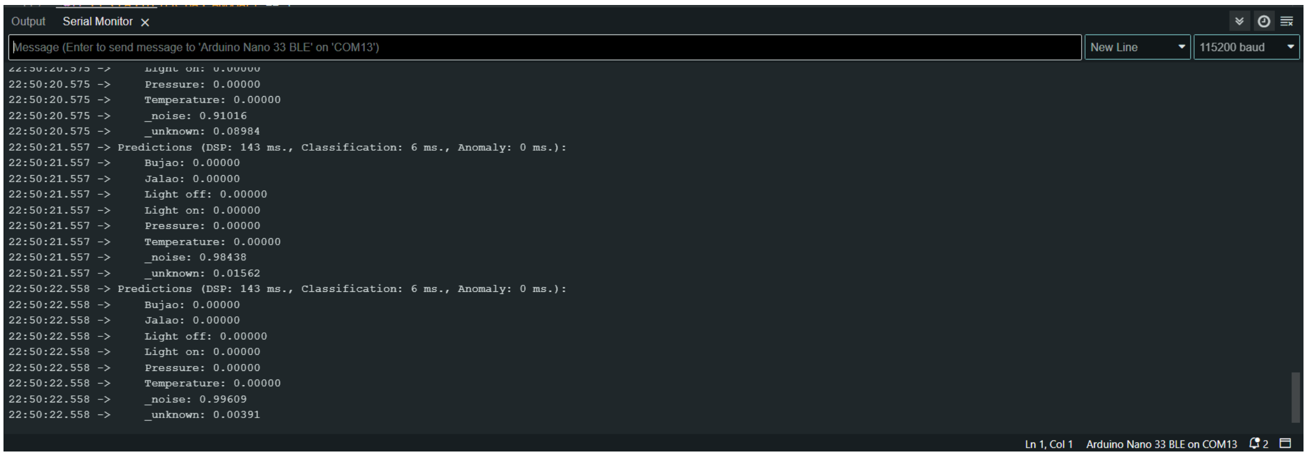
Inside the microcontroller, the deployed model will process incoming audio samples captured by the onboard microphone.

Word detection involves continuously monitoring the audio input for specific keywords ("Light On," "Light Off," etc.). When a match is found, the microcontroller can trigger relevant actions, such as turning on a light or adjusting temperature.

<u>A typical model deployment needs to consider these things:-</u>

- Model Conversion
- Library Setup
- Initialization
- Audio Input
- Preprocessing
- Model Inference
- Wake Word Detection
- Action Trigger
- Edge impulse takes care of many things automatically.

<u>Probabilty based Thresholding</u> is used to identify the given input. There is possibility that a given input may have matches with a few trained keywords. Then this thresholding helps to decide which one is dominating. The value of threshold varies according to the environment. In silent environment it should be kept high. We have chosen 0.7 as threshold. If match is more than 0.7 then result will be declared that user has given this command. We have also created a category as unknown. If the user says something that the model does not understand, then it will be kept in the category of the unknown in the keywords.


Output
Serial Monitor
Message (Enter to send message to 'Arduino Nano 33 BLE' on 'COM13')
New Line
115200 baud
Ln 1, Col 1
Arduino Nano 33 BLE on COM13
2

```
22:50:20.575 ->     Pressure: 0.00000
22:50:20.575 ->     Temperature: 0.00000
22:50:20.575 ->     _noise: 0.91016
22:50:20.575 ->     _unknown: 0.08984
22:50:21.557 -> Predictions (DSP: 143 ms., Classification: 6 ms., Anomaly: 0 ms.):
22:50:21.557 ->     Bujao: 0.00000
22:50:21.557 ->     Jalao: 0.00000
22:50:21.557 ->     Light off: 0.00000
22:50:21.557 ->     Light on: 0.00000
22:50:21.557 ->     Pressure: 0.00000
22:50:21.557 ->     Temperature: 0.00000
22:50:21.557 ->     _noise: 0.98438
22:50:21.557 ->     _unknown: 0.01562
22:50:22.558 -> Predictions (DSP: 143 ms., Classification: 6 ms., Anomaly: 0 ms.):
22:50:22.558 ->     Bujao: 0.00000
22:50:22.558 ->     Jalao: 0.00000
22:50:22.558 ->     Light off: 0.00000
22:50:22.558 ->     Light on: 0.00000
22:50:22.558 ->     Pressure: 0.00000
22:50:22.558 ->     Temperature: 0.00000
22:50:22.558 ->     _noise: 0.99609
22:50:22.558 ->     _unknown: 0.00391
```

Fig Serial Monitor of Arduino IDE showing the showing the word and its probability

4. Hardware Integration

- SENSOR INTERFACING

Microphone

MP34DT05-A (MEMS audio sensor omnidirectional digital microphone)

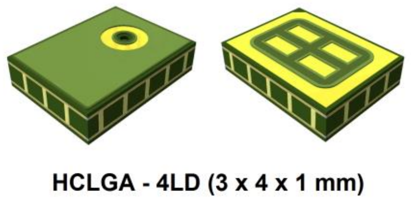


Fig Microphone Sensor

The MP34DT05-A is an ultra-compact, low-power, omnidirectional, digital MEMS microphone built with a capacitive sensing element and an IC interface. The sensing element, capable of detecting acoustic waves, is manufactured using a specialized silicon micromachining process dedicated to producing audio sensors. The IC interface is manufactured using a CMOS process that allows designing a dedicated circuit able to provide a digital signal externally in PDM format.
The sensing element shall mean the acoustic sensor consisting of a conductive movable plate and a fixed plate placed in a tiny silicon chip. This sensor transduces the sound pressure into the changes of coupled capacity between those two plates.

Pulse Density Modulation (PDM) is a form of modulation used to represent an analog signal with a binary signal. In a PDM signal, specific amplitude values are not encoded into codewords of pulses of different weight as they would be in pulse-code modulation (PCM); rather, the relative density of the pulses corresponds to the analog signal's amplitude. The output of a 1-bit DAC is the same as the PDM encoding of the signal.

In a PDM bitstream, a 1 corresponds to a pulse of positive polarity (+A), and a 0 corresponds to a pulse of negative polarity (-A). A run consisting of all 1s would correspond to the maximum (positive) amplitude value, all 0s would correspond to the minimum (negative) amplitude value, and alternating 1s and 0s would correspond to a zero amplitude value. The continuous amplitude waveform is recovered by low-pass filtering the bipolar PDM bitstream.

Temperature and Humidity Sensor

HTS221 (Capacitive digital sensor for relative humidity and temperature). The HTS221 is an ultra-compact sensor for relative humidity and temperature. It includes a sensing element and a mixed signal ASIC to provide the measurement information through digital serial interfaces.

The sensing element consists of a polymer dielectric planar capacitor structure capable of detecting relative humidity variations and is manufactured using a dedicated ST process.

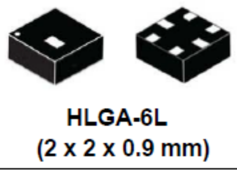


Fig Temperature and Humidity Sensor.

Register which are being used for reading.

- **WHO_AM_I (0Fh)**
  Device Identification

7	6	5	4	3	2	1	0
1	0	1	1	1	1	0	0

This register holds a value which holds the value of device address. Before starting anything it is confirmed that it is the right device.

```
_wire->begin();
if (i2cRead(HTS221_WHO_AM_I_REG) != 0xbc) {
        end();
```

HTS221_CTRL1_REG

One shot mode is enabled using this by configuring bit 0 and 1. The ODR1 and ODR0 bits permit changes to the output data rates of humidity and temperature samples.The default value corresponds to a “one-shot” configuration for both humidity and temperature output. As per datasheet bot ODR bits need to be set to zero to enable one-shot mode.

One-shot mode is triggered and a new acquisition starts when it is required. Enabling this mode is possible only if the device was previously in power-down mode. Once the acquisition is completed and the output registers updated, the device automatically enters in power-down mode. ONE_SHOT bit self-clears itself.

HTS221_CTRL2_REG

One shot mode is enabled using bit 0.

HTS221_STATUS_REG

Bit 0 tells about the data availability of temperature. Bit 1 tells about the data availability of humidity. Whenever a new sample is available, these bits are set to high by the hardware. After reading values these bits are cleared.

HTS221_TEMP_OUT_L
It holds output data LSB of temperature measured. It contains direct digital value automatically converted by ADC.

HTS221_TEMP_OUT_H
It holds temperaturte data measured MSB.

INT16_T TOUT = i2cRead16(HTS221_TEMP_OUT_L_REG);

Final Temperature RTeading.
 float reading = (tout * _hts221TemperatureSlope + _hts221TemperatureZero)

HTS221_HUMIDITY_OUT_L
It holds output data LSB of Humidity measured. It contains direct digital value automatically converted by ADC.

HTS221_HUMIDITY_OUT_H
It holds humidity data MSB.

Hout from output register

int16_t hout = i2cRead16(HTS221_HUMIDITY_OUT_L_REG);

  float reading =hout * _hts221HumiditySlope + _hts221HumidityZero)

Pressure Sensor

Arduino nano 33 ble sense has few inbuilt sensors. It has world's smallest pressure sensor.

It has LPS22HB(MEMS nano pressure sensor: 260-1260 hPa absolute digital output barometer) by STMicroelectronics.

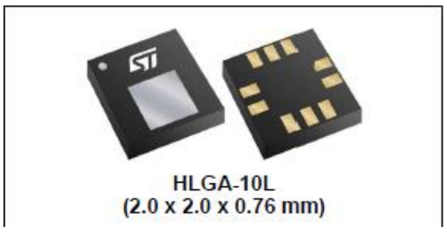


Fig Pressure Sensor

We are not discussing much about working of the sensor, rather we are focusing on register and other parts related to our course.

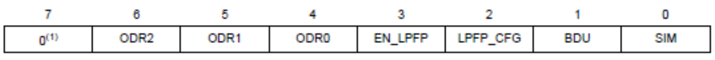
Important registers we were used in our application.

- **WHO_AM_I(0Fh)**
  This register stores the address of the sensor. If I want to use the sensor and sensor is integrated in device. Then First I will find the address of the sensor from the microcontroller datasheet. And before doing anything I will compare that value with the value stored in WHO_AM_I register.
  Here is the code justifying this statement:-

```
{
_wire->begin();
if (i2cRead(LPS22HB_WHO_AM_I_REG) != 0xb1) {
end();
return 0;
}
_initialized = true;
return 1;
}
```

  *0Fh is the address of the register in the sensor.

  CTRL_REG1(10h)
  This register helps us to select output data rate, Low-pass configuration,SPI Serial Interface Mode selection.



7	6	5	4	3	2	1	0
0(1)	ODR2	ODR1	ODR0	EN_LPFP	LPFP_CFG	BDU	SIM

Here, we will control three bits, 4,5,6 to select the output data rate. In this case we have used one shot mode. According to the datasheet, the sensor needs to set all three bits to zero to enable one-shot mode.
One Shot Mode,
If the ONE_SHOT bit in CTRL_REG2 (11h) is set to '1', One-shot mode is triggered and a new acquisition starts when it is required. Enabling this mode is possible only if the device was previously in power-down mode (ODR bits set to '000'). Once the acquisition is completed and the output registers updated, the device automatically enters in power-down mode. ONE_SHOT bit self-clears itself.

CTRL_REG2(11h)
Bit 0 of this register represents one_shot mode by default.

Output Value:-
As pressure is analog but we are measuring it digital. So we need to convert analog quantity to digital. But this sensor directly gives us digital value. We do not have to care about ADC conversion. It gives us 24-bit output. As it has 8-bit registers only. It uses 3 registers to display output. Here is a detailed explanation of these registers.

- **PRESS_OUT_XL(28h)**
  It contains the low part of the pressure output value.
- **PRESS_OUT_L(29h)**
  It contains the mid-part of the pressure output value.
- **PRESS_OUT_H(2Ah)**
  It contains the high part of the pressure output value.

```
    float reading = (i2cRead(LPS22HB_PRESS_OUT_XL_REG) |
     (i2cRead(LPS22HB_PRESS_OUT_L_REG) << 8) |
              (i2cRead(LPS22HB_PRESS_OUT_H_REG) << 16)) / 40960.0;
    if (units == MILLIBAR) { // 1 kPa = 10 millibar
    return reading * 10;
    } else if (units == PSI) {  // 1 kPa = 0.145038 PSI
     return reading * 0.145038;
    } else {
return reading;
    }
    }
```

How to get value from these registers:-

Pressure Value (LSB) = PRESS_OUT_H (2Ah) & PRESS_OUT_L (29h) & PRESS_OUT_XL (28h)
= 3FF58Dh = 4191629 LSB (decimal signed)

$$\text{Pressure (hPa)} = \frac{\text{Pressure Value (LSB)}}{\text{Scaling Factor}} = \frac{4191629\ \text{LSB}}{4096\ \text{LSB/hPa}} = 1023.3\text{hPa}$$

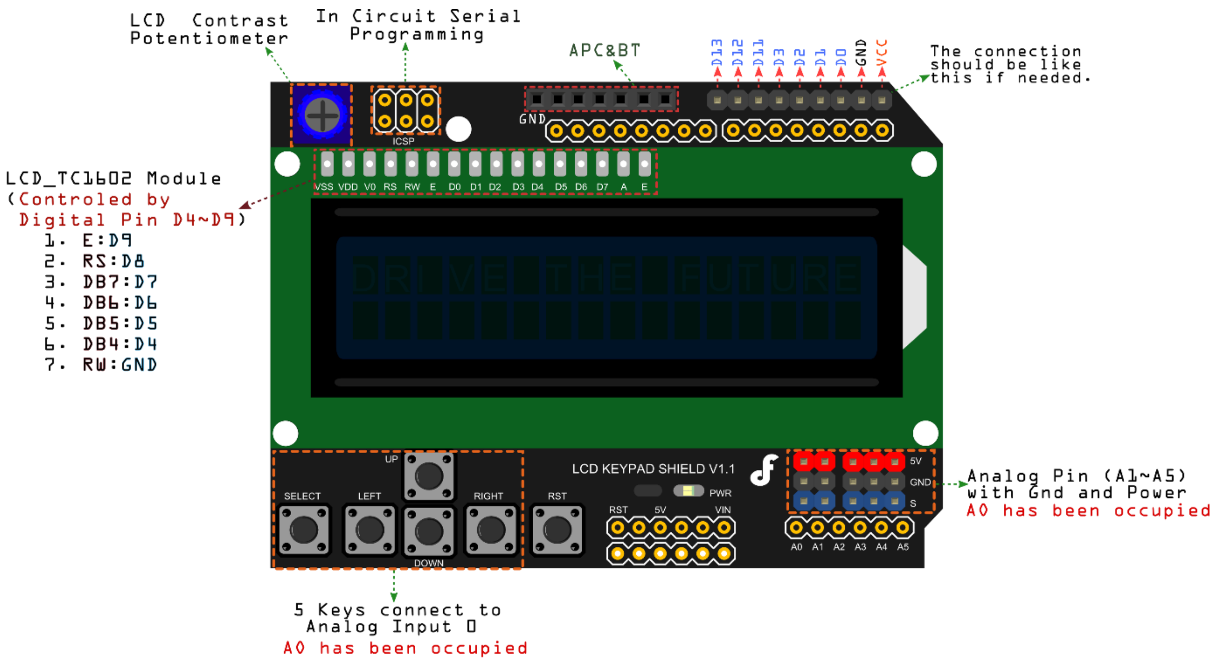
<u>DISPLAY</u>

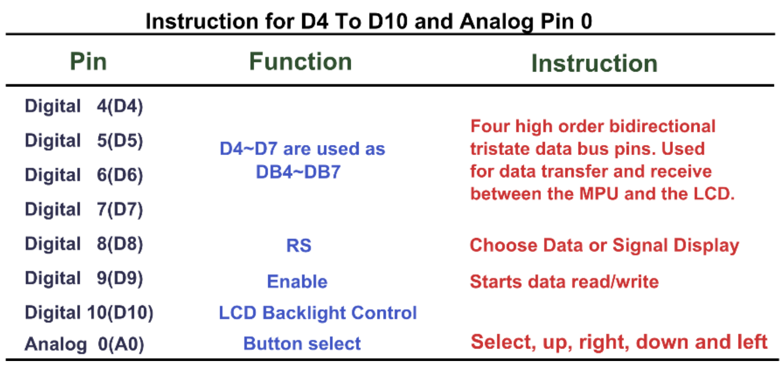
LCD Keypad Shield

This is a very popular LCD Keypad shield for Arduino. It includes a 2x16 LCD display and 6 momentary push buttons. Pins 4, 5, 6, 7, 8, 9 and 10 are used to interface with the LCD. The LCD shield supports contrast adjustment and backlit on/off functions. It also expands analog pins for easy analog sensor reading and display.

Specification

- Operating Voltage:5V
- 5 Push buttons to supply a custom menu control panel
- RST button for resetting Arduino program
- Integrate a potentiometer for adjusting the backlight
- Expanded available I/O pins
- Expanded Analog Pinout with standard DFRobot configuration for fast sensor extension
- Dimension: 80 x 58 mm


LCD Contrast Potentiometer
In Circuit Serial Programming
APC&BT
D13 D12 D11 D3 D2 D1 D0 GND VCC
The connection should be like this if needed.
ICSP
GND
LCD_TC1602 Module (Controled by Digital Pin D4~D9)
1. E:D9
2. RS:D8
3. DB7:D7
4. DB6:D6
5. DB5:D5
6. DB4:D4
7. RW:GND
VSS VDD V0 RS RW E D0 D1 D2 D3 D4 D5 D6 D7 A E
DRIVE THE FUTURE
UP
SELECT
LEFT
RIGHT
DOWN
RST
LCD KEYPAD SHIELD V1.1
PWR
RST 5V VIN
5V GND S
A0 A1 A2 A3 A4 A5
Analog Pin (A1~A5) with Gnd and Power
A0 has been occupied
5 Keys connect to Analog Input 0
A0 has been occupied


Instruction for D4 To D10 and Analog Pin 0

Pin	Function	Instruction
Digital 4(D4)	D4~D7 are used as DB4~DB7	Four high order bidirectional tristate data bus pins. Used for data transfer and receive between the MPU and the LCD.
Digital 5(D5)		
Digital 6(D6)		
Digital 7(D7)		
Digital 8(D8)	RS	Choose Data or Signal Display
Digital 9(D9)	Enable	Starts data read/write
Digital 10(D10)	LCD Backlight Control	
Analog 0(A0)	Button select	Select, up, right, down and left

Function Explanation

- LiquidCrystal(rs, enable, d4, d5, d6, d7)
    - Creates a variable of type LiquidCrystal. The display can be controlled using 4 or 8 data lines. If the former, omit the pin numbers for d0 to d3 and leave those lines unconnected. The RW pin can be tied to ground instead of connected to a pin on the Arduino; if so, omit it from this function's parameters. for example:
        - LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

- lcd.begin(cols, rows)
    - Initializes the interface to the LCD screen, and specifies the dimensions (width and height) of the display. begin() needs to be called before any other LCD library commands. for example:
        - lcd.begin(16, 2);

- lcd.setCursor(col,row)
    - Set the location at which subsequent text written to the LCD will be displayed. for example:
        - lcd.setCursor(0,0);

- lcd.print(data)
    - Prints text to the LCD.for example:
        - lcd.print("hello, world!");

- lcd.write(data)
    - Write a character to the LCD

RELAY

- This is a LOW Level 5V 2-channel relay interface board, and each channel needs a 15 -20mA driver current.
- It can be used to control various appliances and equipment with large current.
- It is equipped with high-current relays that work under AC 250V 10A or DC 30V 10A.
- It has a standard interface that can be controlled directly by microcontroller.

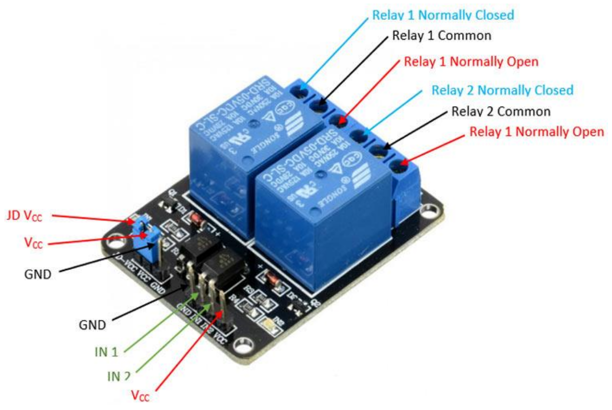
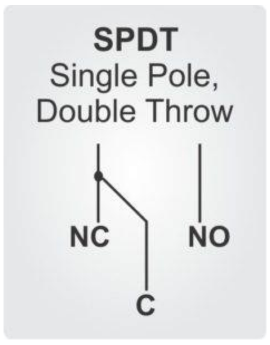


Fig Relay Module

Feature:

- Relay Maximum output: DC 30V/10A, AC 250V/10A .
- 2 Channel Relay Module with Optocoupler LOW Level Triger expansion board, which is compatible with Arduino.
- Standard interface that can be controlled directly by microcontroller (8051, AVR, *PIC, DSP, ARM, ARM, MSP430, TTL logic) .
- Relay of high quality loose music relays SPDT (Single Pole Double Throw) . A common terminal, a normally open, one normally closed terminal optocoupler isolation, good anti-jamming.
- SPDT

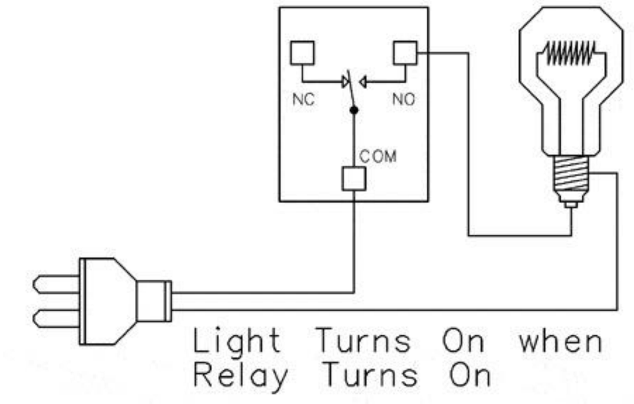


Input:

- VCC : Connected to positive supply voltage (supply power according to relay voltage) ; input for directly powering the relay coils .
- JD-VCC : Input for the isolated power supply for relay coils . (VCC AND JD-VCC are shorted)
- GND : Connected to negative supply voltage ; input ground reference
- IN1: Signal triggering terminal 1 of relay module
- IN2: Signal triggering terminal 2 of relay module

Output:

Each submodular of the relay has one NC(nomalclose), one NO(nomalopen) and one COM(Common). So there are 2 NC, 2 NO and 2 COM of the channel relay in total. NC stands for the normal close port contact and the state without power; No stands for the normal open port contact and the state with power. COM means the common port. You can choose NC port or NO port according to whether power or not.



```
{
	relay_SetStatus(ON, OFF);//turn on RELAY_1

	delay(2000);//delay 2s

	relay_SetStatus(OFF, ON);//turn on RELAY_2

	delay(2000);//delay 2s

}
```

Wiring Diagram

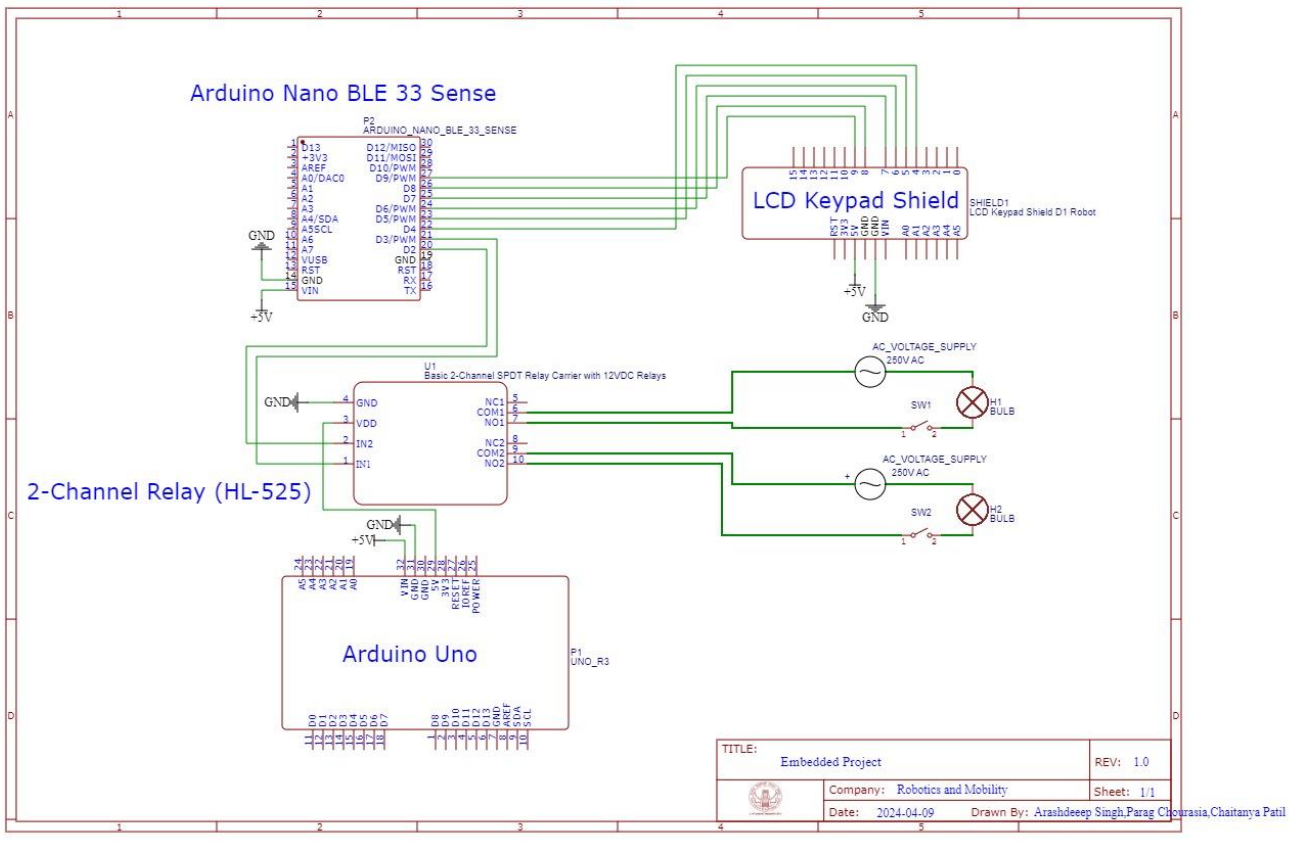

Arduino Nano BLE 33 Sense
P2
ARDUINO_NANO_BLE_33_SENSE
LCD Keypad Shield
SHIELD1
LCD Keypad Shield D1 Robot
2-Channel Relay (HL-525)
U1
Basic 2-Channel SPDT Relay Carrier with 12VDC Relays
AC_VOLTAGE_SUPPLY
250V AC
SW1
H1 BULB
SW2
H2 BULB
Arduino Uno
P1
UNO_R3
GND
+5V
TITLE: Embedded Project
REV: 1.0
Company: Robotics and Mobility
Sheet: 1/1
Date: 2024-04-09
Drawn By: Arashdeeep Singh,Parag Chourasia,Chaitanya Patil

Fig Wiring Diagram

Main Script

We have attached the code as separate files. Here are explaining main segments which are responsible for model deployment.

- Initialising the libraries of Sensors and LCD

```
// Library Setup
#include <PDM.h>
#include <speech_recognition_2_inferencing.h>
#include <Arduino_LPS22HB.h>
#include <Arduino_HS300x.h>
#include <LiquidCrystal.h>

// Initialization
void setup() {
    Serial.begin(115200);

    HS300x.begin();

    BARO.begin();

    pinMode(relay1, OUTPUT);

    pinMode(relay2, OUTPUT);

    lcd.begin(16, 2);

    // Other initializations...

}


// Audio Input (part of Initialization)

static bool microphone_inference_start(uint32_t n_samples) {

    // PDM initialization...

}


// Model Inference

void loop() {

    bool m = microphone_inference_record();

    signal_t signal;

    signal.total_length = EI_CLASSIFIER_SLICE_SIZE;

    signal.get_data = &microphone_audio_signal_get_data;

    ei_impulse_result_t result = {0};

    EI_IMPULSE_ERROR r = run_classifier_continuous(&signal, &result, debug_nn);

    // Actions based on classification results...

}
```

- Condition of if-else for showing the result, by relay and LCD Module

```
    if(result.classification[0].value > thresh){ // ............// "BUJHAO"
     //digitalWrite(led_red,LOW);   //  BLE Sense is ON when the pin  is LOW
     digitalWrite(relay1,HIGH);  // Turning off the relay
     lcd.print("BUJHAO  ");
     delay(500);
     lcd.clear();

    }else if(result.classification[1].value > thresh){ // .......// "JALAO"
            //  BLE Sense is ON when the pin  is LOW
     // Turning on the relay
     digitalWrite(relay1,LOW);
     lcd.print("JALAO  ");
     delay(500);
     lcd.clear();

    }else if(result.classification[2].value > thresh){ // .......// "LIGHT OFF"
     digitalWrite(relay2,HIGH); // Turning off the relay
```

Hardware Demonstration

We attaching a link of video demonstrating our work. Here we are attaching pictures to give you more idea about project.

https://drive.google.com/drive/folders/1zmc6pTd-Wkw5olhbmeGIsCwXqpTlRqMD?usp=sharing

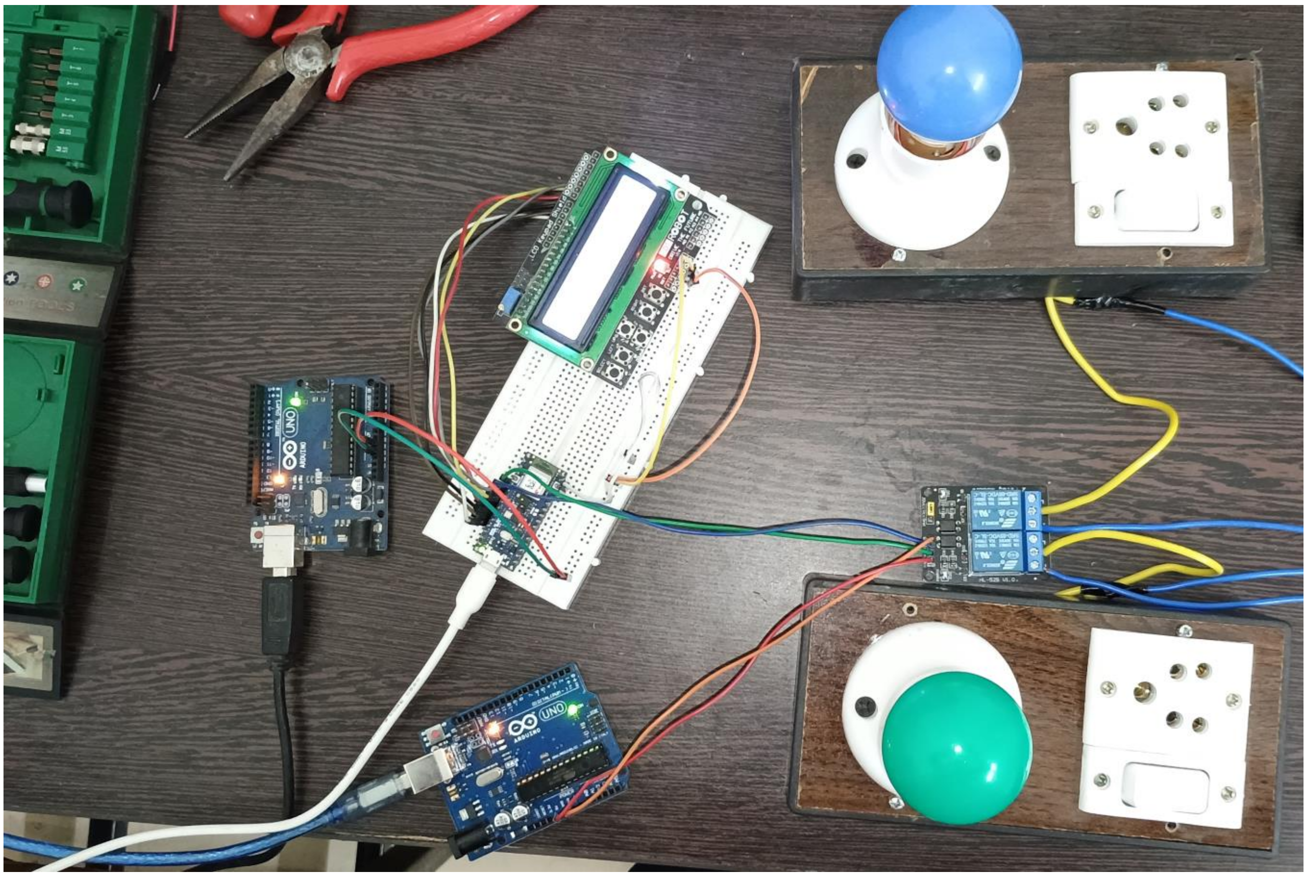


Fig Actual Circuit

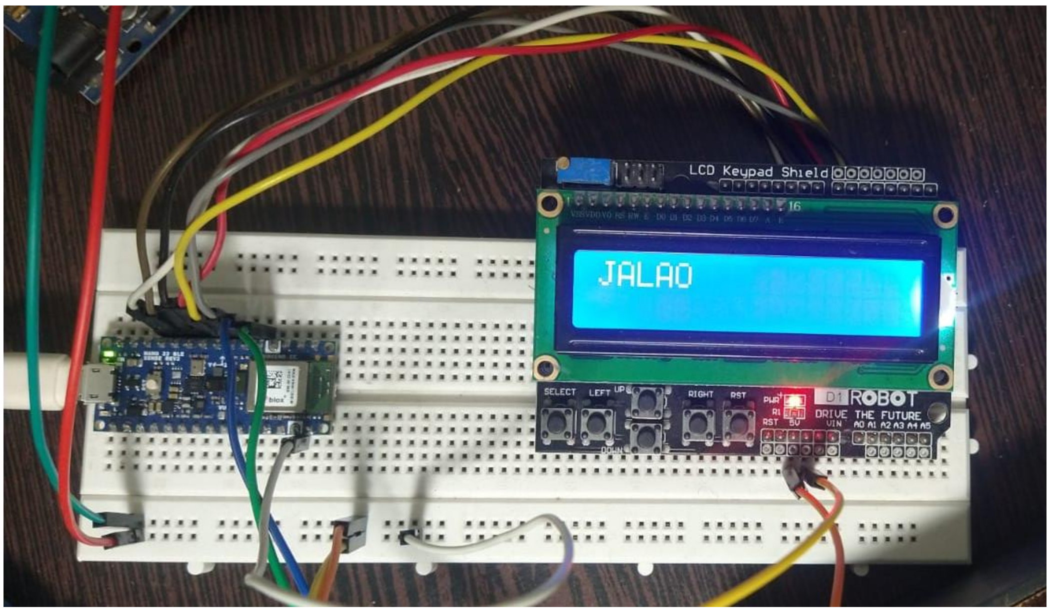


Fig Screen Displaying the result for keyword ‘JALAO’

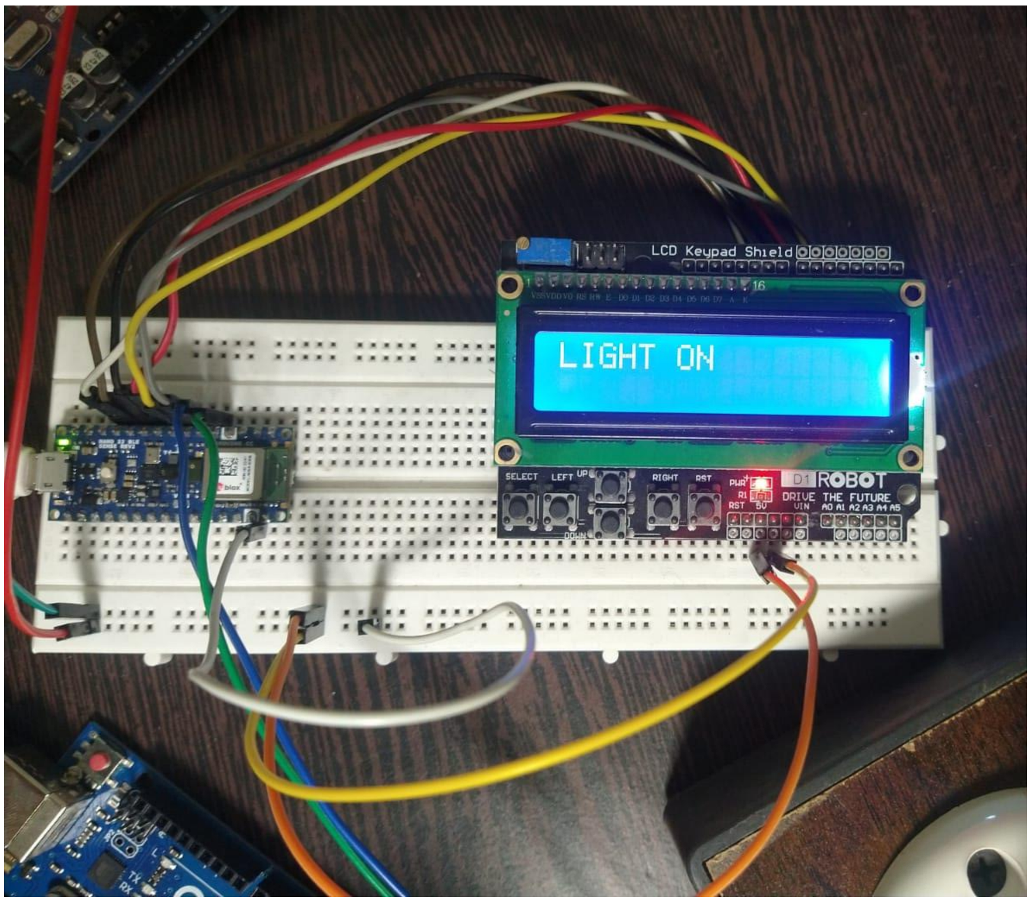


Fig Screen Displaying the result for keyword ‘LIGHT ON’

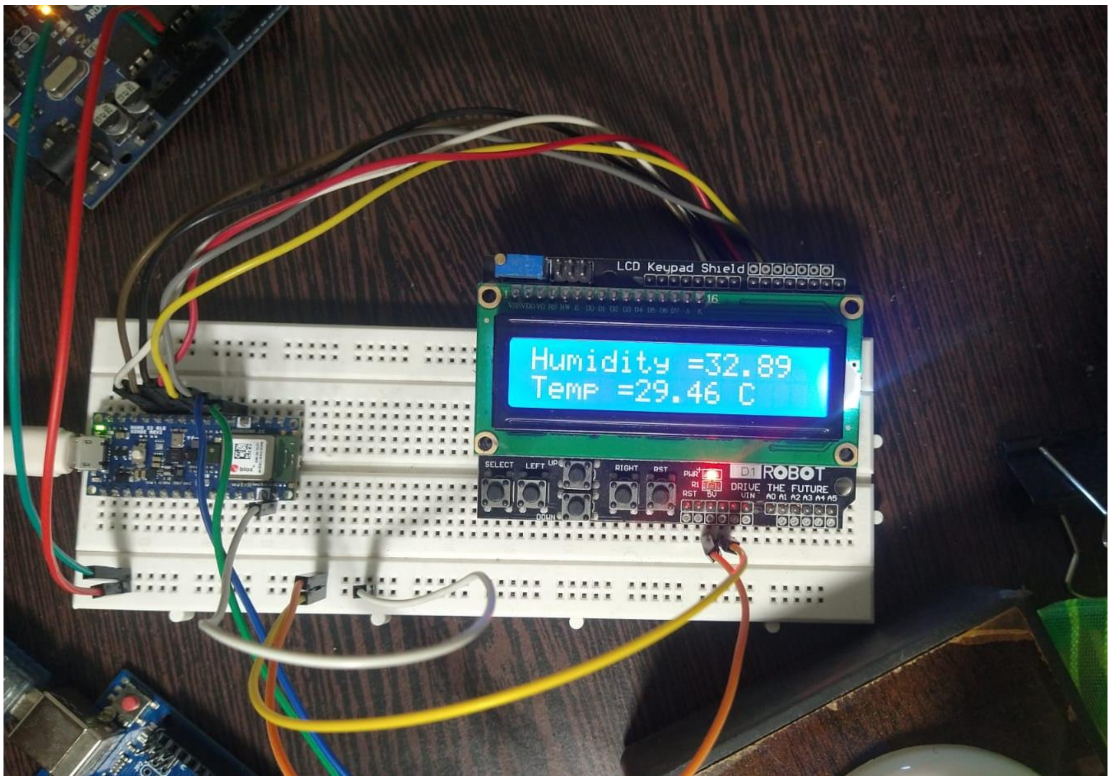


Fig Screen Displaying the result for keyword ‘TEMPERATURE’

WORK DISTRIBUTION

ARASHDEEP SINGH (M23IRM003)	CHAITAYNA PATIL (M23IRM004)	PARAG CHOURASIA (M23IRM010)
Data Collection Model Deployment(Sensors) Documentation	Data Collection Model Deployment(Peripheral) Hardware	Data Set Model Training Model Testing

References:-

Seeing is Believing: Converting Audio Data into Images | by Tony Chen | Towards Data Science

Arduino 4 Relays Shield Basics | Arduino Documentation

MFCC Technique for Speech Recognition - Analytics Vidhya

https://components101.com/switches/5v-dual-channel-relay-module-pinout-features-applications-working-datasheet

https://cdn-reichelt.de/documents/datenblatt/B300/ME114.pdf

MP34DT05-A - MEMS audio sensor omnidirectional stereo digital microphone - STMicroelectronics

Pulse-density modulation - Wikipedia

MFCC (Mel Frequency Cepstral Coefficients) for Audio format (opengenus.org)

Audacity ® | Free Audio editor, recorder, music making and more! (audacityteam.org)

Edge Impulse - The Leading edge AI platform