Home Automation System Using IoT

Thesis Submitted

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY

in

Electronics and communication engineering

Submitted by
Kartik Saini (19JE0423)
and
Keshav Raj (19JE0434)

IV yr, Bachelor of Technology in Electronics and Communication engineering
INDIAN INSTITUTE OF TECHNOLOGY (ISM) DHANBAD

Under the Supervision of
Prof. Nirupama Mandal
Associate Professor, Department of Electronics Engineering
Indian Institute of Technology (ISM) Dhanbad

DEPARTMENT OF ELECTRONICS ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY (INDIAN SCHOOL OF MINES)
DHANBAD-826004

INDIAN INSTITUTE OF TECHNOLOGY (ISM) DHANBAD
DEPARTMENT OF ELECTRONICS ENGINEERING DHANBAD 826004

STUDENT'S DECLARATION

I hereby submit the project entitled **"Home Automation System using IoT"** to the Department of Electronics Engineering, Indian Institute of Technology (ISM) Dhanbad under the guidance of Prof. **Nirupama Mandal**, Associate Professor, Department of Electronics Engineering, Indian Institute of Technology (ISM) Dhanbad.

KARTIK SAINI
(Admission no: 19JE0423)

KESHAV RAJ
(Admission no: 19JE0434)

इलेक्ट्रॉनिक्स अभियांत्रिकी विभाग
DEPARTMENT OF ELECTRONICS ENGINEERING
भारतीय प्रौद्योगिकी संस्थान (भारतीय खनि विद्यापीठ), धनबाद
INDIAN INSTITUTE OF TECHNOLOGY (INDIAN SCHOOL OF MINES), DHANBAD
धनबाद-826004, झारखण्ड, भारत / DHANBAD-826004, JHARKHAND, INDIA

CERTIFICATE

This is to certify that **Kartik Saini** (Admission No. 19JE0423), **Keshav Raj** (Admission No. 19JE0434), students of B.Tech. (ECE), Department of Electronics Engineering, Indian Institute of Technology (Indian School of Mines), Dhanbad have worked under my supervision and completed their project work entitled **"Home Automation System Using IoT"** in partial fulfillment of the requirement for award of degree of B.Tech. ECE from Indian Institute of Technology (Indian School of Mines), Dhanbad.

This work has not been submitted for any other degree, award, or distinction else whereto the best of my knowledge and belief. They are solely responsible for technical data and information provided in this work.

Prof. Nirupama Mandal
Supervisor, Associate Professor
Department of Electronics Engineering
Indian Institute of Technology
(Indian School of Mines) Dhanbad

FORWARDED BY:

Prof. Ravi Kumar Gangwar
Head of the Department
Department of Electronics Engineering
Indian Institute of Technology
(Indian School of Mines) Dhanbad

ABSTRACT

In today's world, technology has evolved a lot and is still evolving by each passing day. People are using technology in various ways, one of them is to interact with the environment they are living in, like home or office. The rise of automation technologies has made life easier in every way. In the modern world, automatic systems are favored over manual ones.

Internet of Things or IoT is the newest and most cutting-edge internet technology. The exponential growth in internet users over the past ten years has made the internet a necessary component of daily life. The internet of things is an expanding network of commonplace items, including consumer goods and industrial machinery, that can exchange information and carry out tasks while you are engaged in other activities.

Home automation is the automatic control of household appliances. The technology offers the ability to effectively regulate household appliances using IoT sensors and other communication devices. Anywhere in the world, we can operate household appliances using a mobile device, a laptop, or the internet.

The driving force behind automating your house is not the novelty of "total phone control", rather, it is to replace inefficient interfaces (home/environment interaction) with more efficient ones. Home automation aims to minimize our interactions with our surroundings. Automations with clever design can eliminate interfaces. It is intended to conserve both human and electric energy. Moreover, it will greatly improve the lifestyle of individuals in this fast and stressful world.

List of Contents

List of Figures

1. Introduction

The network of physical objects—"things"—that are integrated with sensors, software, and other technologies for the purpose of communicating and sharing data with other devices and systems over the internet is referred to as the Internet of Things (IoT). These gadgets include anything from common domestic items to high-tech industrial gear. By connecting to an IoT gateway or other edge device, which either sends data to the cloud for analysis or analyzes it locally, IoT devices exchange the sensor data they collect. Today, there are more than 7 billion connected IoT devices, and according to analysts, there will be 10 billion by 2020 and 22 billion by 2025.

Figure 1: Idea of Home Automation

Home automation utilizes a network of gadgets linked to the Internet via various protocols, such as Wi-Fi, Bluetooth, ZigBee, and others. The devices can be remotely controlled by controllers via electronic connections. Many of these Internet of Things (IoT) gadgets feature sensors that keep track of changes in motion, temperature, and light so the user may learn more about the environment around the device. The user activates actuators, which are the actual physical components like motors, motorized valves, and smart light switches that enable remote control of a device.

Home automation works on three levels:

- **Monitoring:** Monitoring means that users can check in on their devices remotely through an app. An individual might be able to see their live feed from a smart security camera, for instance.
- **Control:** Control means that the user can control these devices remotely, like panning a security camera to see more of a living space.
- **Automation:** Finally, automation means setting up devices to trigger one another, like having a smart siren go off whenever an armed security camera detects motion.

2. Objective

To create the blueprints for a network of linked home appliances that will be incorporated into the Home Automation System and further integrate and automate various appliances, control them, and keep track of every one of them.

- To operate household appliances using an app, the wall switches, and our voice
- To be able to check the status of appliances through the app
- To integrate an alarming system in case of LPG leakage

3. Methodology

3.1. Introduction

After undertaking the project we started with the idea of controlling the appliances using a smartphone and will still be able to control them using the already mounted wall switches. The main aim is to control various appliances, but along with that we also thought of adding a safety alarm system for LPG leakage. Beyond the basic task of controlling the appliances wirelessly, we thought of providing an extra functionality to control the appliances. For that, we started with two different approaches, one is to detect hand gestures or sign language and the other is using voice commands.

3.2. Sensors and Microcontroller

For LPG leakage detection we decided to use the MQ-5 gas detection sensor which can detect many different types of gas. For the main processing unit, we considered using NodeMCU or ESP32 microcontrollers as they have a built-in wifi module which makes it easier to connect to the internet and can send and receive messages or commands over the internet. Also they can easily be programmed using the Arduino Framework. Finally we decided to use ESP32 microcontroller as it is faster than NodeMCU and will be more useful when working with machine learning. The appliances will be connected to a relay module which will act as a programmable switch to control the appliances.

3.3. Smartphone Application

For controlling the appliances using a smartphone, we thought of designing our own application. We tried to learn and create android applications using java in Android Studio. But later on chose to use Blynk IoT platform, as we encountered a lot of problems when trying to create the android application completely from scratch. Blynk IoT is a digital dashboard where we can design our own custom interface with button

and sensor readings along with an android application and a library to integrate everything using the Arduino Framework. We still need to program all the logic by ourselves, it only creates an interface to connect the smartphone and the microcontroller.

3.4. Hand Gesture Approach

In this approach, we thought of using a camera to detect hand gestures. Different gestures can be mapped to different operations. We can train a machine learning model to detect gestures and then can interpret the detected gesture to perform the corresponding task. We started with a CNN model to detect the hand gesture from the image frame sent by the camera, but training the model on a complete image of millions of pixels which will require a lot of processing time, we shifted to landmark detection based on CNN which involves identifying some landmark points from the image and then using those landmarks for training and prediction.

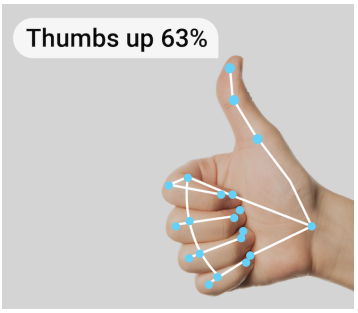

Figure 2: Landmark Detection

This way the input space for the model will be reduced a lot. Of course it will not be as efficient as the previous model but for our task it is more than enough. Also we first do some preprocessing on the image using OpenCV to decrease the input size further by converting the input image to grayscale and apply a filter to detect edges from the grayscale image. The filtered image with edges is sufficient to detect landmarks and it will also result in faster performance.

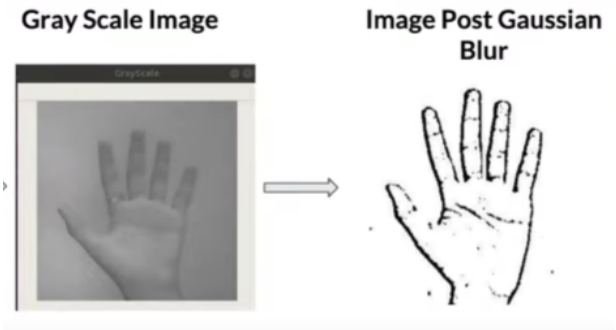

Figure 3: Image Preprocessing

But this approach had various drawbacks like

- Will not work if room is dark, as image will be completely black
- Where to mount the camera in the room
- Camera needed to be ON all the which consumes a lot of power
- Gesture must be performed in front of the camera
- The range will be limited by the camera placement

So, looking at the above drawbacks we decided to drop this approach and move on with the voice command approach.

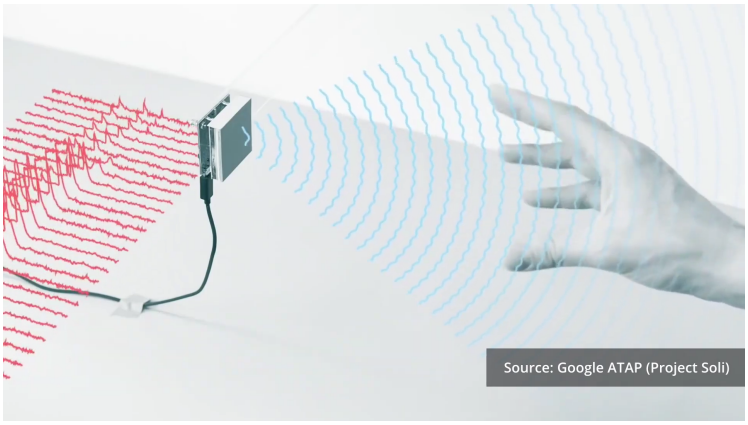

Figure 4: Radar based gesture detection

An alternative to using a camera is that we can use a radar like 3D sensor which can detect the gestures even in the dark and the range will be a lot better but then their can be a lot of false positives that will be detected as the sensor will be covering most of the area of the room and might detect normal roaming or waving motion as some kind of gesture.

3.5. Voice Command Approach

In this approach, the main aim is to take the voice input into the microcontroller using a microphone and perform some analysis to detect what needs to be done. First task is to take the voice input such that it can be recognisable and there is not much noise in the input. For this we decided to use the INMP441 microphone which works on I^2S protocol and can be directly used with ESP32 microcontroller and also provides very low noise so the input signal can be used with very small amounts of audio processing.

After taking the voice input, it will be passed to a machine learning model for further processing. Now we divide our system in two parts, first is to detect a Wake Word, i.e. a trigger word which will make our system go in active state. The second is the Intent processing task, that is whatever the mic takes as input after detecting the wake word is

supposed to be a command. We have to process the audio to detect what the command says and do that accordingly. We should not consider everything which is coming from the mic as a possible command, as it will utilize a lot of resources and will make the system slow and inefficient. So we will define a wake word after which we will have a window of 3 seconds to take the mic's input and process it as a command.

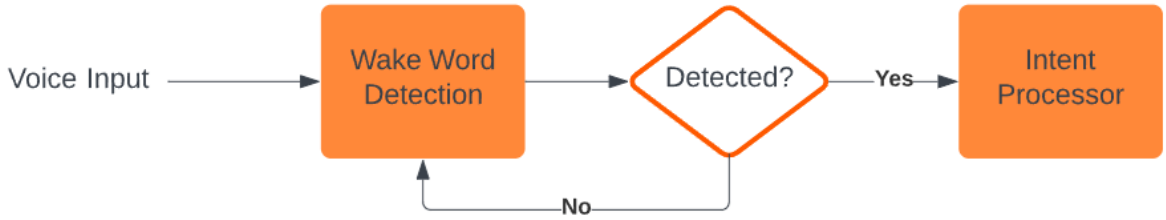

Figure 5: Flow Diagram of voice model

For the wake word detection, we tried to train a tensorflow machine learning model to detect the word HOUSE, which we considered as the wake word. The complete process of using the model with our microcontroller is as follows: We, first, train the model to detect HOUSE and not detect any other word. Then this model needs to be converted to tensorflow lite (tflite) model so that it can run on ESP32. After generating the tflite model, it needs to be integrated with the Arduino Framework which we are using to program the ESP32.

The very first problem that arose during this was to get a dataset to train the model. We faced a lot of problems with finding and creating our own dataset and then with the accuracy of the model. After generating the model we faced a lot of errors on converting it to tflite but were unable to resolve those errors in time. We also tried integrating a different tflite model with the Arduino Framework and again encountered a lot of problems. So, currently we are unable to complete the wake word detection task but to make the system working, we connected a switch, which will act as a manual override. On pressing the switch, the system will go in active state and take a 3 second input from the microphone and then pass it on to perform the intent processing task.

To figure out what needs to be done from the voice input, we can again use a machine learning model to detect the user's intent directly from the voice or first convert the voice into text and then detect the intent from the text. As we were unable to integrate our own model with the ESP32, we decided to use wit.ai, which is a Facebook's service where you can train a model to detect the intent from text or voice and it also provides an API which makes it easier to integrate with our microcontroller. So, we trained a model on wit.ai to detect an action of turning ON and OFF our three appliances which we named as "red", "green", and "blue". They can be named more meaningful like

bedroom lights, kitchen lights, bedroom fan, etc, but we kept it simple based on the color of the LED.

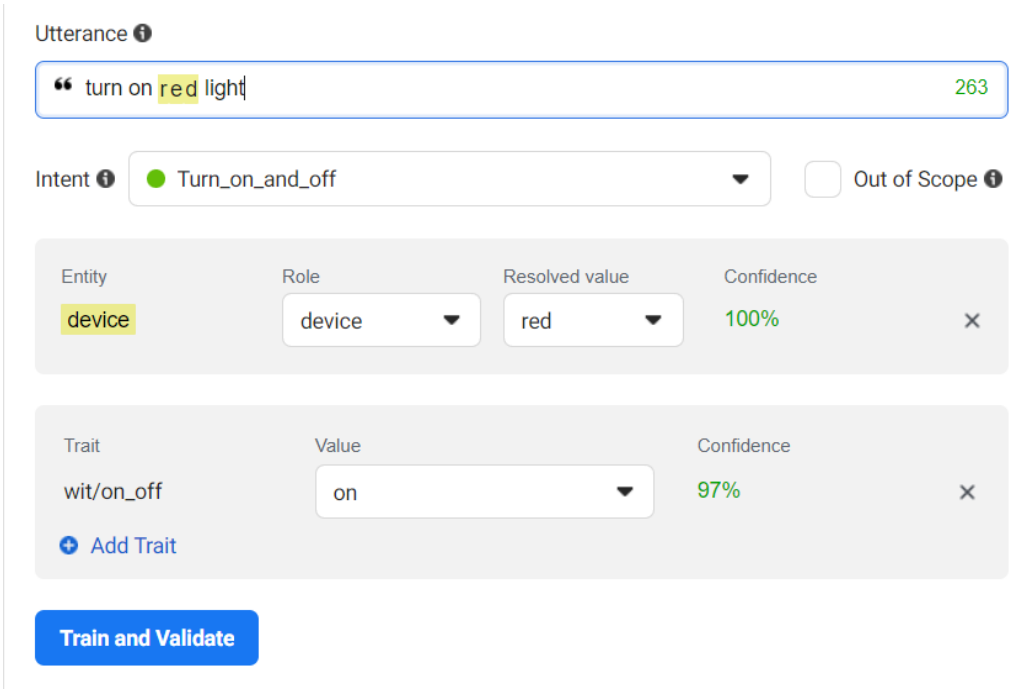

Figure 6: wit.ai training example for red light

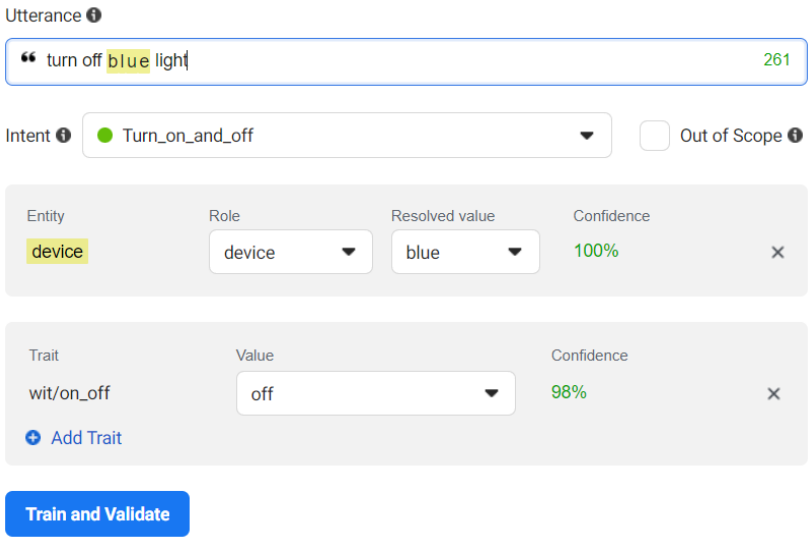

Figure 7: wit.ai training example for blue light

We created a NodeJs server which will receive the audio input from the ESP32 as a .wav file and then send this file to wit.ai to process the intent. On receiving the response from the wit.ai the server will then send the necessary parameters back to ESP32. Now from the ESP32 we will check for any errors in the detection based on the receiving

parameters. If there is some error, then we won't do anything and the system will return back to the initial state. If there is no error, then we perform the task corresponding to the parameters, and after that the system will return back to the initial state.

Parallely, the ESP32 will keep on interacting with other sensors and the android app. This is achieved by utilizing the dual core functionality of ESP32. We have pinned the voice command system to core 0 and the manual switch and android application task to core 1 of the ESP32. This way both tasks can be performed in parallel and will not block or interfere with each other.

4. Implementation

4.1. Components Required

- Microcontroller (ESP32)
- Relay Module (5 Volt)
- Power Adapter (5 Volt)
- Gas Sensor (MQ-5)
- Microphone (INMP441)
- LEDs (as indicator and appliance)
- Connecting Wires
- Smartphone

Figure 8: Components Required

4.2. System Design

In our system, we have the ESP32 microcontroller which is connected to the smartphone through the internet. On the smartphone, we have the Blynk IoT app from which we can control different appliances. ESP32 is connected to a relay module which is further connected to the appliances, so we can control the relay using ESP32 which in turn will control the appliances. The wall switches are also connected to the GPIO pins of the microcontroller to detect if it's ON or OFF. The MQ-5 gas sensor sends the analog data to the microcontroller which checks if it's value is greater than a particular threshold value, in that case turn on the red LED which acts as an alarm. If the gas sensor value is below the threshold keep the LED off.

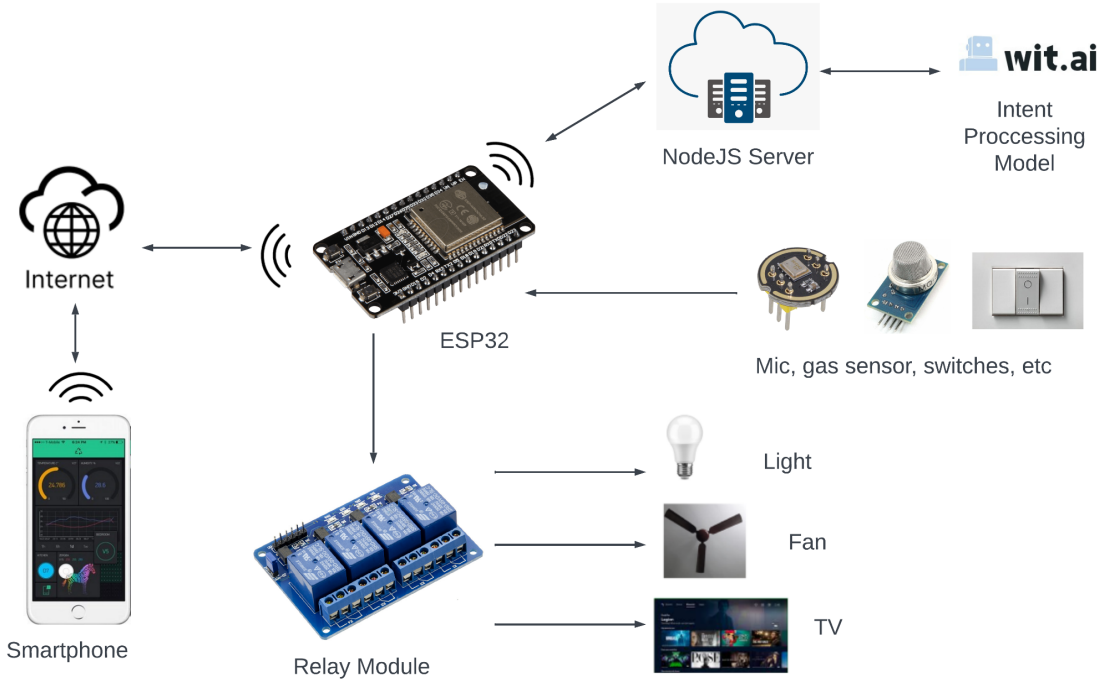

Figure 9: System Design

The INMP441 microphone is also connected to the microcontroller and a switch is there, which acts as manual override to go into active state. In the active state, the microcontroller records a .wav file of 3 seconds and sends the file to our NodeJS server which then calls the wit.ai API with the .wav file and receives the response. Then sends the required parameters like device name, on/off intent, etc, back to the microcontroller.

Then the microcontroller performs the error handling as if the intent is not recognised, or the confidence is low on the device name or the trait/intent value. Then if no error is present, the corresponding appliance is turned ON or OFF and the system returns back to initial state which can again be triggered by the switch and the same procedure is followed.

In the Blynk IoT application, we designed the interface as shown below to turn ON and OFF the three LEDs. It also shows the reading from the MQ-5 gas sensor on a scale of 0 to 100. The alarming LED will turn on if the reading value crosses 30 and will turn off again when the reading drops below 30. In the code, we are constantly reading the input from the Blynk server and turning the appliance accordingly. Along with this, the microcontroller is monitoring the wall switches to turn the appliance manually which is handled by the manualControl() function.

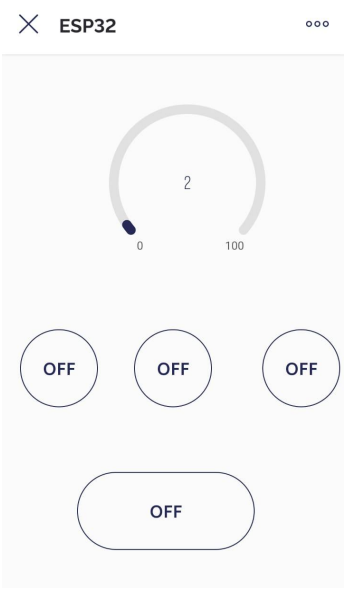

Figure 10: Blynk IoT App Interface

4.3. Circuit Diagram

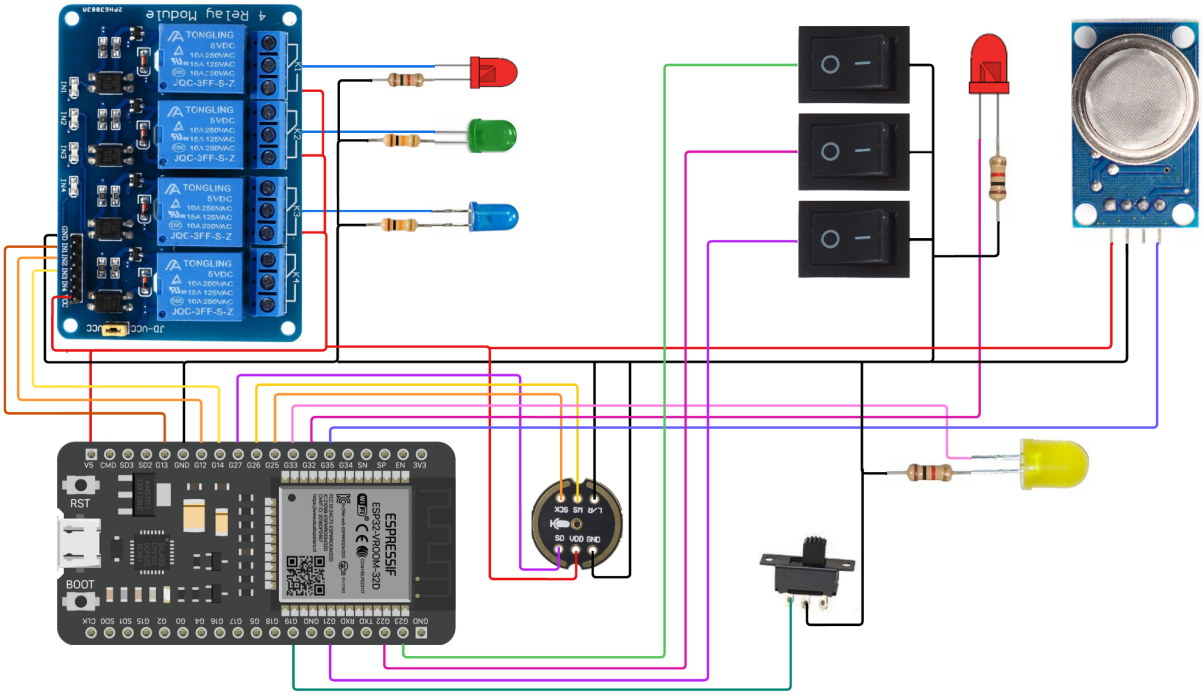

Figure 11: Circuit Diagram

In the circuit diagram, the relay module, microphone, switches, gas sensor, and the indicator leds, all are connected to the ESP32 through its general purpose input output (GPIO) pins. The three leds, red, green and blue represent the appliances and are

connected to the relay module. The relay, gas sensor, and the microphone, all are getting the power supply of 5V from the microcontroller's VIN pin. The microcontroller will get the power through its microUSB port using a 5V power adapter.

4.4. Code Implementation

- **Including all the libraries**

```
#include <Arduino.h>
#include <WiFi.h>
#include <WiFiClient.h>
#include <HTTPClient.h>
#include <BlynkSimpleEsp32.h>
#include <driver/i2s.h>
#include <esp_task_wdt.h>
#include <SPIFFS.h>
#include <ArduinoJson.h>
#include "config.h"
```

- **Defining the configuration constants in config.h file**

```
// WiFi credentials
#define WIFI_SSID "OnePlus"
#define WIFI_PASS "111222777"

// define Blynk parameters
#define BLYNK_TEMPLATE_ID "TMPL3tLoeBaNP"
#define BLYNK_TEMPLATE_NAME "Home Automation"
#define BLYNK_AUTH_TOKEN "j_rrIUE312N_s1mcsOxCS4d-OFhoqF1A"
#define BLYNK_PRINT Serial

// define the GPIOs connected with Relays and switches
#define RELAY_PIN_1    14  //D14
#define RELAY_PIN_2    12  //D12
#define RELAY_PIN_3    13  //D13

#define SWITCH_PIN_1   23  //D23
#define SWITCH_PIN_2   22  //D22
#define SWITCH_PIN_3   21  //D21
```

```
#define WIFI_LED_PIN   2   //D2
#define GAS_LED_PIN    32  //D32 led indicator for gas
#define GAS_SENSOR_PIN 35  //D35 pin for gas sensor analog output
#define VOICE_BTN_PIN  19  //D19 voice input button
#define MIC_LED_PIN    33  //D33 mic indicator led

// define virtual pins for Blynk
#define VPIN_BUTTON_1    V0
#define VPIN_BUTTON_2    V1
#define VPIN_BUTTON_3    V2
#define VPIN_BUTTON_ALL  V3
#define VPIN_GAS_SENSOR  V4

// I2S configuration settings
#define I2S_WS            26
#define I2S_SD            27
#define I2S_SCK           25
#define I2S_PORT          I2S_NUM_0
#define I2S_SAMPLE_RATE   (16000)
#define I2S_SAMPLE_BITS   (16)
#define I2S_READ_LEN      (16 * 1024)
#define I2S_CHANNEL_NUM   (1)
#define FLASH_RECORD_SIZE (I2S_CHANNEL_NUM * I2S_SAMPLE_RATE *
I2S_SAMPLE_BITS / 8 * 3)

// command recognition settings
#define COMMAND_RECOGNITION_ACCESS_KEY "PRCVAZ4OSQM2ZSQUWJNU6UV5DZTQ7H2O"
```

- **Initializing global constants and variables**

```
// Relay State - To remember the toggle state for relay
bool relayState1 = LOW;
bool relayState2 = LOW;
bool relayState3 = LOW;

// Switch State
bool switchState1 = LOW;
bool switchState2 = LOW;
bool switchState3 = LOW;
```

```
int gasSensorVal;
bool blinkLed = false;

// Voice command
bool toBeUpdated = false;
const char *device_name;
const char *trait_value;

BlynkTimer timer;

File file;
const char filename[] = "/recording.wav";
const int headerSize = 44;
```

- **Function to create .wav file header**

```
void wavHeader(byte* header, int wavSize)
{
  header[0] = 'R';
  header[1] = 'I';
  header[2] = 'F';
  header[3] = 'F';
  unsigned int fileSize = wavSize + headerSize - 8;
  header[4] = (byte)(fileSize & 0xFF);
  header[5] = (byte)((fileSize >> 8) & 0xFF);
  header[6] = (byte)((fileSize >> 16) & 0xFF);
  header[7] = (byte)((fileSize >> 24) & 0xFF);
  header[8] = 'W';
  header[9] = 'A';
  header[10] = 'V';
  header[11] = 'E';
  header[12] = 'f';
  header[13] = 'm';
  header[14] = 't';
  header[15] = ' ';
  header[16] = 0x10;
  header[17] = 0x00;
  header[18] = 0x00;
  header[19] = 0x00;
  header[20] = 0x01;
```

```
  header[21] = 0x00;
  header[22] = 0x01;
  header[23] = 0x00;
  header[24] = 0x80;
  header[25] = 0x3E;
  header[26] = 0x00;
  header[27] = 0x00;
  header[28] = 0x00;
  header[29] = 0x7D;
  header[30] = 0x00;
  header[31] = 0x00;
  header[32] = 0x02;
  header[33] = 0x00;
  header[34] = 0x10;
  header[35] = 0x00;
  header[36] = 'd';
  header[37] = 'a';
  header[38] = 't';
  header[39] = 'a';
  header[40] = (byte)(wavSize & 0xFF);
  header[41] = (byte)((wavSize >> 8) & 0xFF);
  header[42] = (byte)((wavSize >> 16) & 0xFF);
  header[43] = (byte)((wavSize >> 24) & 0xFF);
}
```

- **Function to initialize the microphone**

```
void SPIFFSInit()
{
  if(!SPIFFS.begin(true)){
    Serial.println("SPIFFS initialisation failed!");
    while(1) yield();
  }
}

void i2sInit()
{
  i2s_config_t i2s_config = {
    .mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_RX),
    .sample_rate = I2S_SAMPLE_RATE,
```

```
    .bits_per_sample = i2s_bits_per_sample_t(I2S_SAMPLE_BITS),
    .channel_format = I2S_CHANNEL_FMT_ONLY_LEFT,
    .communication_format = i2s_comm_format_t(I2S_COMM_FORMAT_I2S |
I2S_COMM_FORMAT_I2S_MSB),
    .intr_alloc_flags = 0,
    .dma_buf_count = 64,
    .dma_buf_len = 1024,
    .use_apll = 1
  };

  i2s_driver_install(I2S_PORT, &i2s_config, 0, NULL);

  const i2s_pin_config_t pin_config = {
    .bck_io_num = I2S_SCK,
    .ws_io_num = I2S_WS,
    .data_out_num = -1,
    .data_in_num = I2S_SD
  };

  i2s_set_pin(I2S_PORT, &pin_config);
}
```

- **Setup function for the Arduino Framework**

```
void setup()
{
  Serial.begin(115200);

  pinMode(RELAY_PIN_1, OUTPUT);
  pinMode(RELAY_PIN_2, OUTPUT);
  pinMode(RELAY_PIN_3, OUTPUT);

  pinMode(SWITCH_PIN_1, INPUT_PULLUP);
  pinMode(SWITCH_PIN_2, INPUT_PULLUP);
  pinMode(SWITCH_PIN_3, INPUT_PULLUP);

  pinMode(WIFI_LED_PIN, OUTPUT);
  pinMode(MIC_LED_PIN, OUTPUT);
  pinMode(GAS_LED_PIN, OUTPUT);
  pinMode(GAS_SENSOR_PIN, INPUT);
```

```
  pinMode(VOICE_BTN_PIN, INPUT_PULLUP);

  //During Starting all Relays should TURN OFF
  digitalWrite(RELAY_PIN_1, HIGH);
  digitalWrite(RELAY_PIN_2, HIGH);
  digitalWrite(RELAY_PIN_3, HIGH);

  Blynk.begin(BLYNK_AUTH_TOKEN, WIFI_SSID, WIFI_PASS);

  // Setup a function to be called every 2 seconds
  timer.setInterval(2000L, checkBlynkStatus);

  SPIFFSInit();
  i2sInit();

  // make sure we don't get killed for our long running tasks
  esp_task_wdt_init(1000, false);

  xTaskCreatePinnedToCore(i2s_adc, "i2s_adc", 1024 * 3, NULL, 1, NULL, 0);
}
```

- **Loop function for the Arduino Framework**

```
void loop()
{
  Blynk.run();
  manualController();
  timer.run();
  if (blinkLed)
  {
    digitalWrite(MIC_LED_PIN, HIGH);
    delay(100);
    digitalWrite(MIC_LED_PIN, LOW);
    delay(100);
  }
}
```

- **Functions to listen to the Blynk server**

```
BLYNK_WRITE(VPIN_BUTTON_1)
{
  relayState1 = param.asInt();
  digitalWrite(RELAY_PIN_1, !relayState1);
}

BLYNK_WRITE(VPIN_BUTTON_2)
{
  relayState2 = param.asInt();
  digitalWrite(RELAY_PIN_2, !relayState2);
}

BLYNK_WRITE(VPIN_BUTTON_3)
{
  relayState3 = param.asInt();
  digitalWrite(RELAY_PIN_3, !relayState3);
}

BLYNK_WRITE(VPIN_BUTTON_ALL)
{
  relayState1 = 0; digitalWrite(RELAY_PIN_1, HIGH);
Blynk.virtualWrite(VPIN_BUTTON_1, relayState1); delay(100);
  relayState2 = 0; digitalWrite(RELAY_PIN_2, HIGH);
Blynk.virtualWrite(VPIN_BUTTON_2, relayState2); delay(100);
  relayState3 = 0; digitalWrite(RELAY_PIN_3, HIGH);
Blynk.virtualWrite(VPIN_BUTTON_3, relayState3); delay(100);
}
```

- **Function for manually controlling using wall switches**

```
void manualController()
{
  if (digitalRead(SWITCH_PIN_1) == LOW && switchState1 == LOW)
  {
    digitalWrite(RELAY_PIN_1, LOW);
    Blynk.virtualWrite(VPIN_BUTTON_1, HIGH);
    relayState1 = HIGH;
    switchState1 = HIGH;
```

```
  Serial.println("Switch-1 on");
}
if (digitalRead(SWITCH_PIN_1) == HIGH && switchState1 == HIGH)
{
  digitalWrite(RELAY_PIN_1, HIGH);
  Blynk.virtualWrite(VPIN_BUTTON_1, LOW);
  relayState1 = LOW;
  switchState1 = LOW;
  Serial.println("Switch-1 off");
}
if (digitalRead(SWITCH_PIN_2) == LOW && switchState2 == LOW)
{
  digitalWrite(RELAY_PIN_2, LOW);
  Blynk.virtualWrite(VPIN_BUTTON_2, HIGH);
  relayState2 = HIGH;
  switchState2 = HIGH;
  Serial.println("Switch-2 on");
}
if (digitalRead(SWITCH_PIN_2) == HIGH && switchState2 == HIGH)
{
  digitalWrite(RELAY_PIN_2, HIGH);
  Blynk.virtualWrite(VPIN_BUTTON_2, LOW);
  relayState2 = LOW;
  switchState2 = LOW;
  Serial.println("Switch-2 off");
}
if (digitalRead(SWITCH_PIN_3) == LOW && switchState3 == LOW)
{
  digitalWrite(RELAY_PIN_3, LOW);
  Blynk.virtualWrite(VPIN_BUTTON_3, HIGH);
  relayState3 = HIGH;
  switchState3 = HIGH;
  Serial.println("Switch-3 on");
}
if (digitalRead(SWITCH_PIN_3) == HIGH && switchState3 == HIGH)
{
  digitalWrite(RELAY_PIN_3, HIGH);
  Blynk.virtualWrite(VPIN_BUTTON_3, LOW);
  relayState3 = LOW;
  switchState3 = LOW;
```

```
    Serial.println("Switch-3 off");
  }
}
```

- **Function to read the Gas sensor data and update it in Blynk server**

```
// reads gas sensor data
void sendSensorData()
{
  gasSensorVal = map(analogRead(GAS_SENSOR_PIN), 0, 4095, 0, 100);

  if (gasSensorVal > 30)
  {
    digitalWrite(GAS_LED_PIN, HIGH);
  }
  else
  {
    digitalWrite(GAS_LED_PIN, LOW);
  }

  Blynk.virtualWrite(VPIN_GAS_SENSOR, gasSensorVal);
}
```

- **Function to take the 3 sec MIC input and save as .wav file**

```
void i2s_adc(void *arg)
{
  bool voiceBtnState = HIGH;
  while(true)
  {
    // Enter active/recording state if button is switched
    if (digitalRead(VOICE_BTN_PIN) != voiceBtnState)
    {
      voiceBtnState = !voiceBtnState;
      blinkLed = true;
      SPIFFS.remove(filename);
      file = SPIFFS.open(filename, FILE_WRITE);
      if(!file){
        Serial.println("File is not available!");
      }
```

```
      byte header[headerSize];
      wavHeader(header, FLASH_RECORD_SIZE);

      file.write(header, headerSize);
      listSPIFFS();
      blinkLed = false;
      digitalWrite(MIC_LED_PIN, HIGH);

      int i2s_read_len = I2S_READ_LEN;
      int flash_wr_size = 0;
      size_t bytes_read;

      char* i2s_read_buff = (char*) calloc(i2s_read_len, sizeof(char));
      uint8_t* flash_write_buff = (uint8_t*) calloc(i2s_read_len,
sizeof(char));

      i2s_read(I2S_PORT, (void*) i2s_read_buff, i2s_read_len, &bytes_read,
portMAX_DELAY);
      i2s_read(I2S_PORT, (void*) i2s_read_buff, i2s_read_len, &bytes_read,
portMAX_DELAY);

      Serial.println(" *** Recording Start *** ");
      while (flash_wr_size < FLASH_RECORD_SIZE) {
          //read data from I2S bus, in this case, from ADC.
          i2s_read(I2S_PORT, (void*) i2s_read_buff, i2s_read_len,
&bytes_read, portMAX_DELAY);
          //example_disp_buf((uint8_t*) i2s_read_buff, 64);
          //save original data from I2S(ADC) into flash.
          i2s_adc_data_scale(flash_write_buff, (uint8_t*)i2s_read_buff,
i2s_read_len);
          file.write((const byte*) flash_write_buff, i2s_read_len);
          flash_wr_size += i2s_read_len;
          ets_printf("Sound recording %u%%\n", flash_wr_size * 100 /
FLASH_RECORD_SIZE);
          ets_printf("Never Used Stack Size: %u\n",
uxTaskGetStackHighWaterMark(NULL));
      }

      digitalWrite(MIC_LED_PIN, LOW);
```

```
      file.close();

      free(i2s_read_buff);
      i2s_read_buff = NULL;
      free(flash_write_buff);
      flash_write_buff = NULL;

      listSPIFFS();

      if(Blynk.connected()){
        uploadFile();
      }
    }
  }
}
```

- **Function for audio processing**

```
void i2s_adc_data_scale(uint8_t * d_buff, uint8_t* s_buff, uint32_t len)
{
    uint32_t j = 0;
    uint32_t dac_value = 0;
    for (int i = 0; i < len; i += 2) {
        dac_value = ((((uint16_t) (s_buff[i + 1] & 0xf) << 8) | ((s_buff[i
+ 0])))));
        d_buff[j++] = 0;
        d_buff[j++] = dac_value * 256 / 2048;
    }
}
```

- **Function to upload the .wav file to our NodeJS server**

```
void uploadFile(){
  file = SPIFFS.open(filename, FILE_READ);
  if(!file){
    Serial.println("FILE IS NOT AVAILABLE!");
    return;
  }
```

```
  Serial.println("===> Upload FILE to Node.js Server");

  HTTPClient client;
  client.begin("http://192.168.110.103:8888/uploadCommand");
  client.addHeader("Content-Type", "audio/wav");
  int httpResponseCode = client.sendRequest("POST", &file, file.size());
  Serial.print("httpResponseCode : ");
  Serial.println(httpResponseCode);

  if(httpResponseCode == 200){
    String response = client.getString();
    StaticJsonDocument<500> doc;
    deserializeJson(doc, response);

    const char* text = doc["text"];
    const char *intent_name = doc["intent_name"];
    float intent_confidence = doc["intent_confidence"];
    device_name = doc["device_name"];
    float device_confidence = doc["device_confidence"];
    trait_value = doc["trait_value"];
    float trait_confidence = doc["trait_confidence"];

    Serial.println("==================== Transcription
====================");
    Serial.println(text);
    Serial.print("Intent: ");
    Serial.println(intent_name);
    Serial.print("Intent Confidence: ");
    Serial.println(intent_confidence);
    Serial.print("Device Name: ");
    Serial.println(device_name);
    Serial.print("Device Confidence: ");
    Serial.println(device_confidence);
    Serial.print("Trait Value: ");
    Serial.println(trait_value);
    Serial.print("Trait Confidence: ");
    Serial.println(trait_confidence);
    Serial.println("====================      End
====================");
```

```
    voiceController(intent_confidence, device_name, device_confidence,
trait_value, trait_confidence);
  }else{
    Serial.println("Error: Failed to detect voice... Please Try
Again!!!");
  }
  file.close();
  client.end();
}
```

- **Function for error handling and triggering the voice command task**

```
void voiceController(float& intent_confidence, const char *device_name,
float& device_confidence, const char *trait_value, float&
trait_confidence) {
    if (intent_confidence < 0.7)
    {
        Serial.printf("Only %.f%% certain on intent\n", 100 *
intent_confidence);
        return;
    }
    if (strcmp(device_name, "red") != 0 && strcmp(device_name, "green") !=
0 && strcmp(device_name, "blue") != 0)
    {
        Serial.println("Device not found");
        return;
    }
    if (device_confidence < 0.7)
    {
        Serial.printf("Only %.f%% certain on device\n", 100 *
device_confidence);
        return;
    }
    if (strcmp(trait_value, "on") != 0 && strcmp(trait_value, "off") != 0)
    {
        Serial.println("Can't work out the intent action");
        return;
    }
    if (trait_confidence < 0.7)
    {
```

```
        Serial.printf("Only %.f%% certain on trait\n", 100 *
trait_confidence);
        return;
    }

    toBeUpdated = true;
}
```

- **Function to perform the voice command**

```
void checkBlynkStatus() // called every 2 seconds by timer
{
  if (Blynk.connected())
  {
    digitalWrite(WIFI_LED_PIN, HIGH);
    sendSensorData();

    if (toBeUpdated)
    {
      if (strcmp(device_name, "red") && strcmp(trait_value, "on"))
      {
        digitalWrite(RELAY_PIN_1, LOW);
        Blynk.virtualWrite(VPIN_BUTTON_1, HIGH);
        relayState1 = HIGH;
        Serial.println("Switch-1 on");
      }
      if (strcmp(device_name, "red") && strcmp(trait_value, "off"))
      {
        digitalWrite(RELAY_PIN_1, HIGH);
        Blynk.virtualWrite(VPIN_BUTTON_1, LOW);
        relayState1 = LOW;
        Serial.println("Switch-1 off");
      }
      if (strcmp(device_name, "green") && strcmp(trait_value, "on"))
      {
        digitalWrite(RELAY_PIN_2, LOW);
        Blynk.virtualWrite(VPIN_BUTTON_2, HIGH);
        relayState2 = HIGH;
        Serial.println("Switch-2 on");
      }
```

```
      if (strcmp(device_name, "green") && strcmp(trait_value, "off"))
      {
        digitalWrite(RELAY_PIN_2, HIGH);
        Blynk.virtualWrite(VPIN_BUTTON_2, LOW);
        relayState2 = LOW;
        Serial.println("Switch-2 off");
      }
      if (strcmp(device_name, "blue") && strcmp(trait_value, "on"))
      {
        digitalWrite(RELAY_PIN_3, LOW);
        Blynk.virtualWrite(VPIN_BUTTON_3, HIGH);
        relayState3 = HIGH;
        Serial.println("Switch-3 on");
      }
      if (strcmp(device_name, "blue") && strcmp(trait_value, "off"))
      {
        digitalWrite(RELAY_PIN_3, HIGH);
        Blynk.virtualWrite(VPIN_BUTTON_3, LOW);
        relayState3 = LOW;
        Serial.println("Switch-3 off");
      }

      toBeUpdated = false;
    }
  }
  else
  {
    digitalWrite(WIFI_LED_PIN, LOW);
    Serial.println("Blynk Not Connected");
  }
}
```

- **Code for the NodeJS server**

```
const fs = require("fs");
const express = require("express");
const axios = require("axios");
const cors = require("cors");

const app = express();
```

```
const port = 8888;

app.use(express.urlencoded({ extended: true }));
app.use(express.json());
app.use(cors());

const fileName = "./resources/recording.wav";

app.get("/", cors(), async (req, res) => {
  res.status(200).send("This is the server home");
});

app.post("/uploadCommand", cors(), async (req, res) => {
  var recordingFile = fs.createWriteStream(fileName, { encoding: "utf8"
});
  req.on("data", function (data) {
    recordingFile.write(data);
  });

  req.on("end", async function () {
    recordingFile.end();

    const file = fs.readFileSync(fileName);
    await axios
      .post("https://api.wit.ai/speech?v=20230506", file, {
        headers: {
          Authorization: "Bearer PRCVAZ4OSQM2ZSQUWJNU6UV5DZTQ7H2O",
          "Content-Type": "audio/wav",
        },
      })
      .then((respo) => {
        console.log("Response from wit.ai: ")
        console.log(respo.status, respo.statusText);

        const data = JSON.parse(
          "[" + respo.data.replace(/}\r\n{/g, "},{") + "]"
        ).slice(-1)[0];

        const text = data.text;
        const intent_name = data.intents[0].name;
```

```
        const intent_confidence = data.intents[0].confidence;
        const device_name = data.entities["device:device"][0].value;
        const device_confidence =
data.entities["device:device"][0].confidence;
        const trait_value = data.traits["wit$on_off"][0].value;
        const trait_confidence = data.traits["wit$on_off"][0].confidence;
        console.log(
          text,
          intent_name,
          intent_confidence,
          device_name,
          device_confidence,
          trait_value,
          trait_confidence
        );
        res.status(200).json({
          text,
          intent_name,
          intent_confidence,
          device_name,
          device_confidence,
          trait_value,
          trait_confidence,
        });
      })
      .catch((err) => {
        console.log(err.response);
      });
  });
});

app.listen(port, () => {
  console.log(`Listening at http://192.168.110.103:${port}`);
});
```

5. Future Scope

- First of all, we can integrate our own model into the microcontroller itself.
- We can have multiple models each for their specific task like wake word detection, speech to text, or intent recognition
- The system can be made modular to easily add or remove devices
- A speaker can be added to give real time feedback
- A complete interactive voice assistant can also be integrated to work with natural language processing
- Homes can be interfaced with multiple sensors including motion sensors, light sensors and temperature sensors and provide automated toggling of devices based on conditions
- The system can be enhanced to include weather stations or energy monitoring. This type of system with the appropriate modifications can be used in companies that cannot or should not allow human intrusion, in hospitals for the disabled, and for environmental monitoring
- Multiple layers of security and monitoring can also be integrated

6. References

1. https://en.wikipedia.org/wiki/Home_automation
2. https://www.security.org/home-automation/
3. https://www.atomic14.com/2020/09/12/esp32-audio-input.html
4. https://www.tensorflow.org/lite
5. https://www.spokestack.io/docs/machine-learning/wakeword-models
6. https://eloquentarduino.com/tensorflow-lite-esp32/
7. https://medium.com/wit-ai/save-time-building-with-wit-ai-built-in-intents-261be59b4db1
8. https://techtutorialsx.com/2019/02/11/esp32-http-2-get-request-to-node-js-server/
9. https://dev.to/miku86/nodejs-how-to-create-a-simple-server-using-express-1n9d
10. Shahzad Ahmed,Karam Dad Kallu,Sarfaraz Ahmed, and Sung Ho Cho - Hand Gestures Recognition Using Radar Sensors for Human Computer Interaction

11. Rizwan Majeed, Nurul Azma Abdullah, Imran Ashraf, Yousaf Bin Zikria, Muhammad Faheem Mushtaq, and Muhammad Umer - An Intelligent, Secure, and Smart Home Automation System: Rizwan Majeed 2020
12. Sameer Alani, Sarah Zaeead Mahmood, Haneen Sameer Attaallah, Zeena Abdulsattar Mhmood, Azzam Amer Khudhur, Sarmad Nozad Mahmood - “IoT based implemented comparison analysis of two well-known network platforms for smart home automation”
13. Jerry John, Bismin V. Sherif - “Hand Landmark-Based Sign Language Recognition Using Deep Learning”, In book: Machine Learning and Autonomous Systems, Proceedings of ICMLAS 2021 (pp.147-157)
14. Zhengyu Peng, Changzhi Li, José-María Muñoz-Ferreras, Roberto Gómez-García - “An FMCW radar sensor for human gesture recognition in the presence of multiple targets”, 2017, First IEEE MTT-S International Microwave Bio Conference (IMBIOC), doi: 10.1109/IMBIOC.2017.7965798