IoT BASED FOREST FIRE PREDICTION SYSTEM USING FUZZY LOGIC

A PROJECT REPORT

Submitted by

ARCHAN RAJARAM RAMADEVI (312215205017)

HARSHINI M (312215205031)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

SRI SIVASUBRAMANIYA NADAR COLLEGE OF ENGINEERING, KALAVAKKAM

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2019

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **“IoT BASED FOREST FIRE PREDICTION SYSTEM USING FUZZY LOGIC”** is the bonafide work of **“ARCHAN RAJARAM RAMADEVI (312215205017), HARSHINI M (312215205031)”** who carried out the project under my supervision.

SIGNATURE	**SIGNATURE**
Dr. T. Nagarajan	Mr.V.Sivamurugan
HEAD OF THE DEPARTMENT	**SUPERVISOR**
Department of Information Technology	Department of Information Technology
SSN College of Engineering	SSN College of Engineering
Kalavakkam – 603 110.	Kalavakkam – 603 110.

Submitted for the Examination held on _____________ .

SIGNATURE	**SIGNATURE**
INTERNAL EXAMINER	**EXTERNAL EXAMINER**

ACKNOWLEDGEMENTS

We express our reverence to **Padma Bhushan Dr. Shiv Nadar**, *Chairman*, SSN Institutions for being a source of inspiration and instilling in us the passion to pursue our goals. We would also like to express our gratitude to **Ms. Kala Vijayakumar**, *President*, SSN Institutions, for constantly encouraging us to be inquisitive.We also thank our *Principal*, **Dr. S. Salivahanan** for providing us with all the support we needed. We are grateful to the Management for providing financial support to carry out this project.

We sincerely thank the *Head of the Department of Information Technology*, **Dr. T. Nagarajan** for having given us the opportunity to carry out this project. We are thankful to our Project Coordinator, **Dr. R. Srinivasan**, *Professor*, Department of Information Technology, for providing us with the necessary resources for the completion of our project.

Words would not suffice to thank our beloved guide, **Mr.V.Sivamurugan**,*Associate Professor*, Department of Information Technology, SSN College of Engineering.He has been actively involved in the project and has provided us a great deal of motivation and continuous guidance to enable the successful completion of this project.

We sincerely thank our project panel members and the various teaching and non-teaching staff of SSN College of Engineering for their help during the various stages of our project.

We would like to take this opportunity to thank our parents for their moral support all through and our friends who have been our pillars of strength during times of difficulty. We also extend our regards to all others who supported us in different ways during the course of our project. Above all, we thank the Almighty for showering His blessings upon us and making our lives better.

ABSTRACT

Forest is considered as one of the most important and indispensable resource. However, forest fire, affected by some human uncontrolled behavior in social activities and abnormal natural factors, occurs occasionally. This project's motive is to predict the occurrence of forest fire by continuous monitoring of factors that are responsible for forest fire like temperature, smoke and flame in forests. We use latest technologies like IoT for collecting data frequently through the sensors. All these nodes containing sensors are deployed at various locations of the forest publish the data to an MQTT broker. Further the nodes can be classified as no forest fire,mild,moderate and severe based on the threshold values and fuzzy logic is applied on this data to predict the chance of occurrence of forest fire and is expressed in percentage.The client devices subscribe to MQTT broker and obtain the sensor data and result of prediction. This will help the authorities to take necessary measures to prevent it and to avoid the loss of human life.

Keywords:

IoT; MQTT broker; Fuzzy logic; Forest fire; Temperature; Smoke; Flame; Sensors

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 OBJECTIVE

Several million acres of forest are destroyed every year due to forest fire. Forest fire not only destroys many valuable trees but also destroys the vegetation in that area. The fire will burn the trees and also the soil is burnt and so many acres of land become water repellent. Forest fire is one of the major causes of global warming as tones of greenhouse gases are emitted into the atmosphere. Nowadays the prediction mechanisms used are watching through towers, satellite imaging, long distance video recording, etc. But these do not provide quicker response which is most important in forest fire prediction.

The causes of forest fires are many: natural i.e. rise in temperature, change in wind direction, moisture level etc. and human related, both intentional and unintentional like the burning by grazers and gatherers of forest produce, shifting cultivation, fires to ward off wild animals or by visitors to forests by way of smoking etc.

The cost of such disaster may be millions of trees, in addition to losses of structures, animals (wild and farm), and human life. Forest fires lead to global warming, soil erosion, ozone layer depletion and the loss of livelihood of those dependent on forest products. The only way of protecting forest from wildfires is their early prediction.

This project predicts the occurrence of forest fire by making use of IoT where temperature, smoke and flame sensors deployed in the forest classify it as no, mild, moderate and severe forest fire using the sensor values and the percentage of chance of occurrence of forest fire is calculated using fuzzy logic and necessary measures can be taken based on the predicted output.

1.2 INTERNET OF THINGS

1.2.1 What is IoT?

The internet of things, or IoT, is a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers (UIDs) and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction.

An IoT ecosystem consists of web-enabled smart devices that use embedded processors, sensors and communication hardware to collect, send and act on data they acquire from their environments. IoT devices share the sensor data they collect by connecting to an IoT gateway or other edge device where data is either sent to the cloud to be analyzed or analyzed locally. Sometimes, these devices communicate with other related devices and act on the information they get from one another. The devices do most of the work without human intervention, although people can interact with the devices -- for instance, to set them up, give them instructions or access the data.

The connectivity, networking and communication protocols used with these web-enabled devices largely depend on the specific IoT applications deployed.

1.2.2.Applications of IoT

Consumer Applications

A growing portion of IoT devices are created for consumer use, including connected vehicles, home automation, wearable technology (as part of Internet of Wearable Things (IoWT), connected health, and appliances with remote monitoring capabilities.

Commercial applications

Medicine and health care: IoT devices can be used to enable remote health monitoring and emergency notification systems. These health monitoring devices can range from blood pressure and heart rate monitors to advanced devices capable of monitoring specialized implants, such as pacemakers, Fit bit electronic wristbands, or advanced hearing aids.

Transportation: Application of the IoT extends to all aspects of transportation systems (i.e. the vehicle, the infrastructure, and the driver or user). Dynamic interaction between these components of a transport system enables inter and intra vehicular communication, smart traffic control, smart parking, electronic toll collection systems, logistic and fleet management, vehicle control, safety and road assistance.

Industrial applications

Manufacturing: The IoT intelligent systems enable rapid manufacturing of new products, dynamic response to product demands, and real-time optimization of manufacturing production and supply chain networks, by networking machinery, sensors and control systems together.

Agriculture: There are numerous IoT applications in farming such as collecting data on temperature, rainfall, humidity, wind speed, pest infestation, and soil content. This data can be used to automate farming techniques, take informed decisions to improve quality and quantity, minimize risk and waste, and reduce effort required to manage crops.

1.2.3 Role of IoT in environmental monitoring and disaster management

Environmental monitoring is a broad application for the Internet of Things. It involves everything from monitoring levels of ozone in a meat packing facility to monitoring national forests for smoke. Using IoT environment sensors for these various applications can take an otherwise highly labor-intensive process and make it simple and efficient.

Some of the use cases are:

1. **Monitoring air** for quality, carbon dioxide and smog-like gasses, carbon monoxide in confined areas, and indoor ozone levels.
2. **Monitoring water** for quality, pollutants, thermal contaminants, chemical leakages, the presence of lead, and flood water levels.
3. **Monitoring soil** for moisture and vibration levels in order to detect and prevent landslides.
4. **Monitoring forests** and protected land for forest fires.
5. **Monitoring for natural disasters** like earthquake and tsunami warnings.
6. **Monitoring fisheries** for both animal health and poaching.
7. **Monitoring snowfall levels** at ski resorts and in national forests for weather tracking and avalanche prevention.
8. **Monitoring data centers** for air temperature and humidity.

CHAPTER 2

LITERATURE REVIEW

2.1 FOREST FIRE PREDICTION AND ALERT SYSTEM USING WIRELESS SENSOR NETWORK

In this method the sensor module is deployed in the forest manually or through a helicopter. The sensor module consists of multiple sensors like temperature sensor, humidity sensor, etc. They collect the target environment information and continuously transfer it to the control center where the necessary process is carried out.

It uses Zigbee which is a specification for communication in a wireless personal area network (WPAN). Zigbee is based on an IEEE 802.15 standard. It consumes low power with transmission distance of 10 to 100 meters line of sight. It can transmit data over long distance through intermediate devices such as by forming mesh network. Zigbee has a defined rate of 250 Kbit/s, best suited for intermittent data transmissions from a sensor or input device. It is simple and much less expensive than other WPANs such as Bluetooth and Wi-Fi.

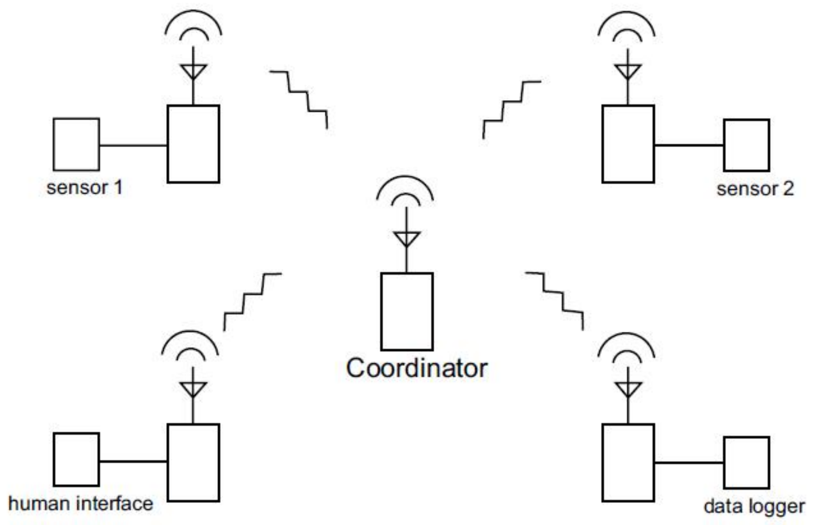


Figure 2.1 Zigbee based sensor network

Sensor nodes are less costly and even if it gets damaged in fire it won't be a great loss. WSN has the property of self configuration and hence need not be organized manually. Using GPS the exact location of the fire can be easily obtained and the nearest fire service can be easily informed using GSM.

When the temperature in a particular node gets increased over a fixed threshold value then the alert is sent to the control center. The threshold value is always fixed above the maximum temperature which is experienced in that particular region to avoid any false alarm due to the increase in the atmospheric temperature.

As soon as the fire is detected in a particular node the alert is sent to the control center and also to the neighboring nodes. Once the nearer nodes get the alert, timer is started and it is run till the nearer node detects the fire. This is to find the rate of spread of the fire in the forest. When the rate of spread is known then the necessary action can be taken quickly. All the nodes are equally spaced in order to easily find the rate of spread of fire. The rate of spread directly depends on the speed of air blowing and also the fire usually spreads upwards in a hilly area. These are taken into considerations while designing the detection system.

$$\text{Rate of spread of fire} = \frac{\text{Distance between two nodes}}{\text{Time interval between reception of alert and fire detection}}$$

2.2 FOREST FIRE DETECTION THROUGH VARIOUS MACHINE LEARNING TECHNIQUES USING MOBILE AGENT IN WSN

This paper suggests that regression works best for forest fire detection with high accuracy by dividing the forest fire dataset. It also presents comparison of various machine learning techniques like as SVM (support vector machines), neural network, decision tree, regression, so on for detection of forest fires and new approach perform better as compared to other machine learning techniques.

Support Vector Machine(SVM)

In machine learning support vector machine is supervised learning model with associated learning algorithms that analyze data and pattern matching. Support Vector Machine (SVM) models are a close cousin to classical alternative training method for polynomial, radial basis function and multi-layer perception classifiers in which the weights of the network are found by solving a quadratic programming problem with linear constraints, rather than by solving a non-convex, unconstrained minimization problem as in standard neural network training.

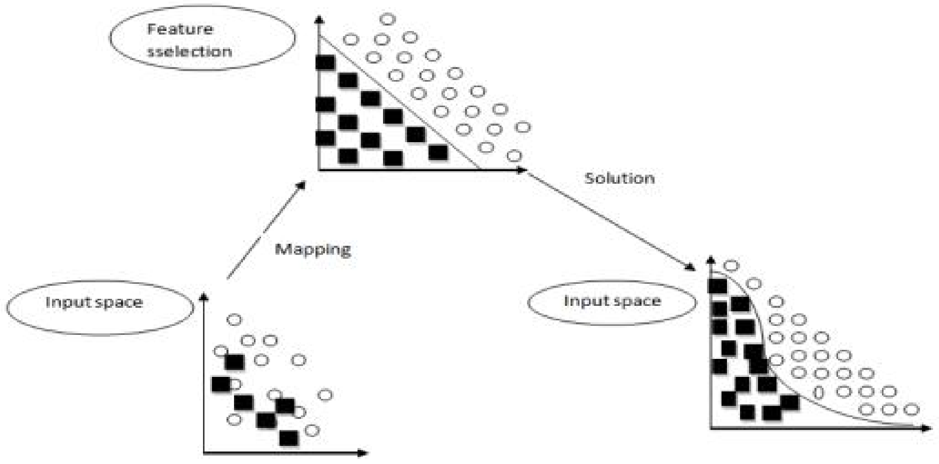


Figure 2.2 SVM algorithm

Artificial Neural Networks(ANN)

ANN is a system that has the ability of learning by itself and learning is an adaptive process. These basically work on principle of neuron. The first model of neuron contained two inputs and one output. Both the inputs should be active for correct output. The weights for both the inputs were equal and output was binary.

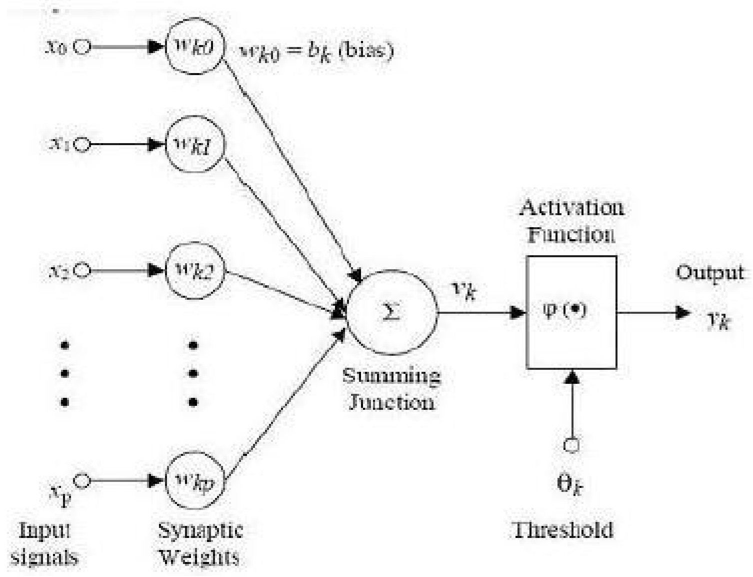


Figure 2.3 ANN model

Decision tree

Decision tree is a form of multiple variable analyses. They allow predicting, explaining, and classifying an outcome. Decision tree classify instances by sorting them down the tree from the root to some leaf node. The final result is a decision tree in which each branch represents a possible scenario of decision and its outcome.

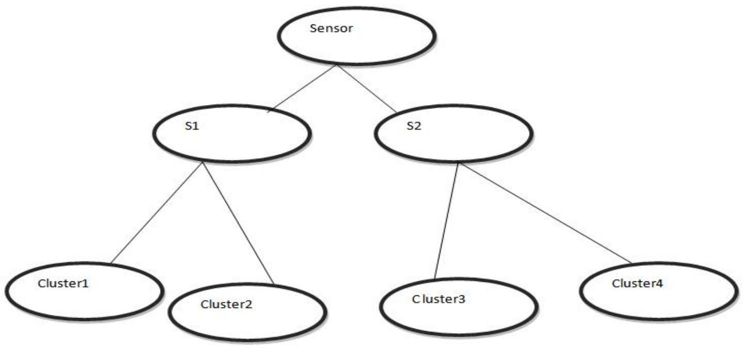


Figure 2.4 Decision tree

Feed Forward Neural Networks(FFNN)

Feed-forward networks can be seen as cascaded squashed linear functions. The inputs feed into a layer of hidden units, which can feed into layers of more hidden units, which eventually feed into the output layer. Each of the hidden units is a squashed linear function of its inputs.

Neural networks of this type can have as inputs any real numbers, and they have a real number as output. For regression, it is typical for the output units to be a linear function of their inputs. For classification it is typical for the output to be a sigmoid function of its inputs (because there is no point in predicting a value outside of [0,1]). For the hidden layers, there is no point in having their output be a linear function of their inputs because a linear function of a linear function is a linear function; adding the extra layers gives no added functionality. The output of each hidden unit is thus a squashed linear function of its inputs.

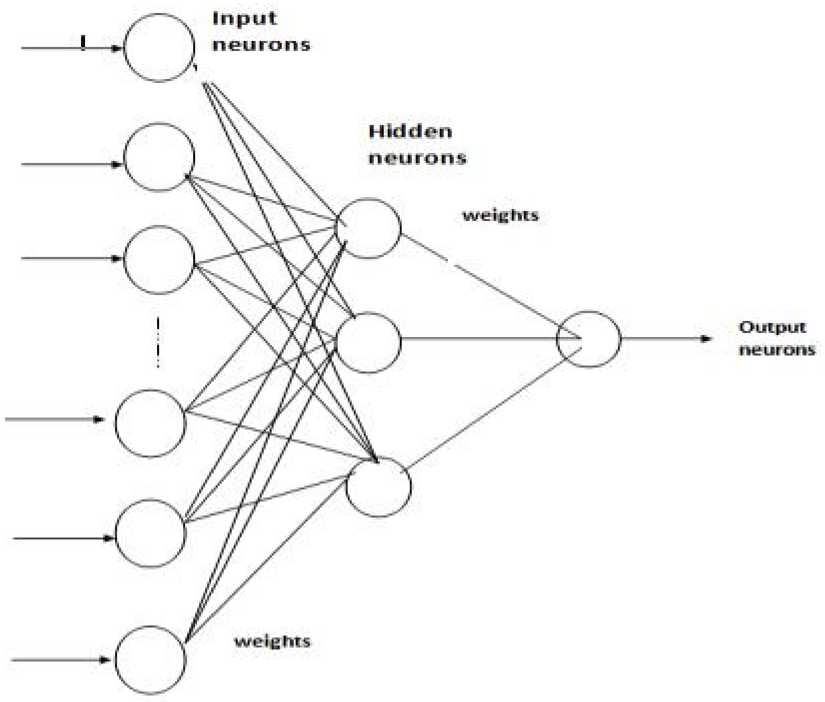


Figure 2.5 Architecture of FFNN

2.3 PREDICTION OF FOREST FIRE USING SVM – A NOVEL APPROACH

In this approach, video cameras are used for monitoring the forest fire. Camera data are sufficient only for the detection of forest fire, but for the prediction some more information is needed.For this purpose sensor nodes are deployed in the forest area. Sensor nodes are capable of measuring the parameters like temperature and humidity. These parameters are important for the prediction of natural forest fire.From there both the sensor reading and the camera data are sent to base station (BS). But for the classification purpose data only before and during the forest fire are taken.

SVM belong to the class of supervised learning algorithms in which the learning machine is given a set of examples (or inputs) with the associated labels (or output values). SVMs construct a hyper plane that separates two classes (this can be extended to multi-class problems). While doing so, SVM algorithm tries to achieve maximum separation between the classes.

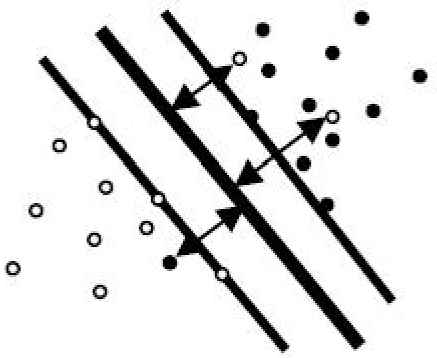


Figure 2.6 SVM linear classification

In support vector classifiers we have a maximally separating hyper plane of the form $w^T x - \gamma = 0$ and two bounding hyper planes of the form $w^T x - \gamma = 1$and$\mathbf{w}^T \boldsymbol{x} - \gamma = -1$. The data points belonging to +1class satisfy the constraint $w^T x - \gamma \geq 1$ and the data point belonging to -1 class satisfy the constraint $w^T x - \gamma \leq -1$. However in our case some of the data points will be deviated from their respective bounding plane, such deviation of data points from their respective bounding planes are called as error. A positive quantity called ξ is added or subtracted to the data points that constitutes to error to satisfy the constraints. The new constraints are $w^T x - \gamma + \xi \geq 1$ and $w^T x - \gamma - \xi \leq -1$. Our aim is to try for maximum margin between the bounding planes and minimize the number of data points contributing to error. Maximum margin can be achieved by minimizing the quantity $1/2 w^T w$ which is the reciprocal of the distance between the two bounding hyper planes from the origin.

Feature values i.e. Humidity and Temperature values prevailing inside the forest will be sent to the base station from the observation spot periodically. These feature values are given to the learned SVM classifier to predict the class labels as ‘YES 'corresponding to a ‘chance for fire occurrence’ and ‘NO’ corresponding to ‘no chance of fire occurrence’.

CHAPTER 3

PROJECT DESCRIPTION

3.1 SYSTEM FLOW

The sensors used for measuring parameters such as temperature, smoke and flame are deployed in the forest as nodes and monitors the environmental conditions continuously.

Then each node is classified into no, mild, moderate and severe forest fire categories and the percentage of chance of occurrence of forest fire is calculated using fuzzy logic.

The sensor data and the classification output is then published to an MQTT broker. The MQTT broker can be either online (cloud) or it can be local(Raspberry PI).

An app which subscribes to these sensor data and the classification output receives them continuously or periodically which helps the person using the app to know if there is any chance of forest fire to occur and take the precautionary measures.

A buzzer is also fixed to the nodes deployed in the forest to alert if there is forest fire so that people or animals approaching towards that location will be made known of the danger.

In this project,

- The nodes deployed in the forest act as publishers
- The online cloud or Raspberry PI act as MQTT broker
- The client devices act as the subscribers

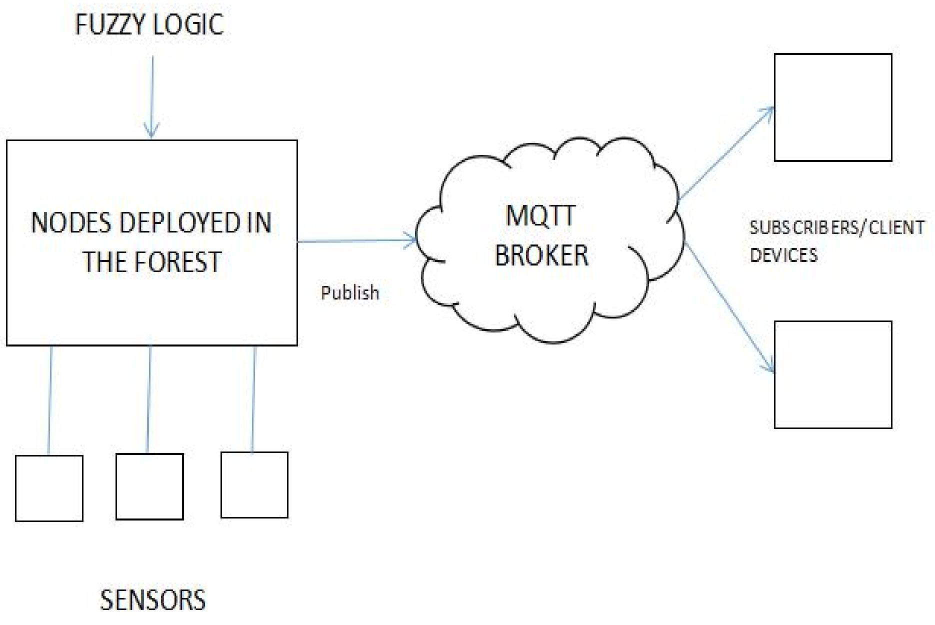


Figure 3.1 System flow

3.2 PREDICTION USING FUZZY LOGIC

3.2.1 BASIC CONCEPTS

3.2.1.1 What is fuzzy logic?

The term **fuzzy** refers to things which are not clear or are vague. In the real world many times we encounter a situation when we can't determine whether the state is true or false, their fuzzy logic provides a very valuable flexibility for reasoning. In this way, we can consider the inaccuracies and uncertainties of any situation.

In Boolean system truth value, 1.0 represents absolute truth value and 0.0 represents absolute false value. But in the fuzzy system, there is no logic for absolute truth and absolute false value. But in fuzzy logic, there is intermediate value too present which is partially true and partially false.

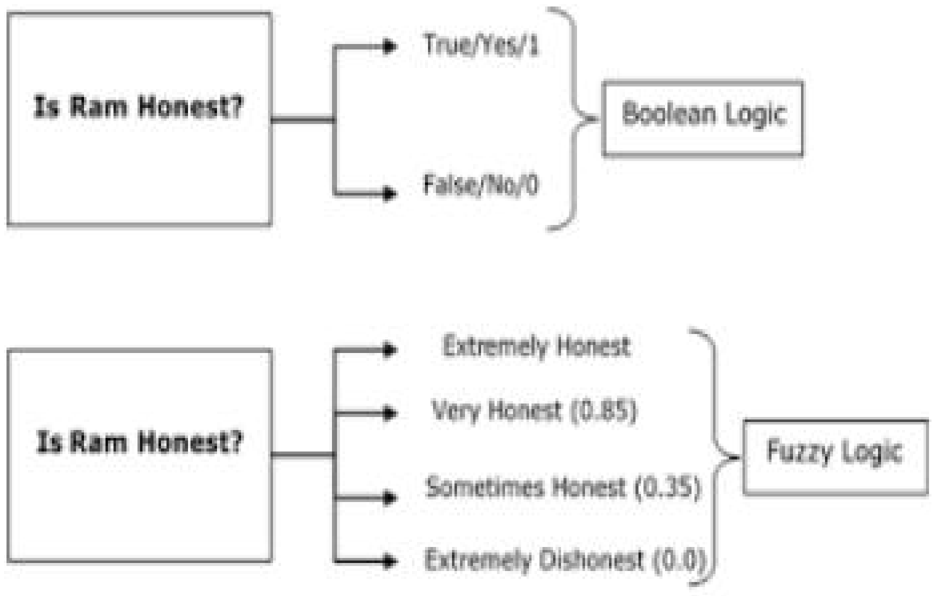


Figure 3.2 Fuzzy logic example

3.2.1.2 Fuzzy logic architecture

Its Architecture contains four parts :

Rule base

It contains the set of rules and the IF-THEN conditions provided by the experts to govern the decision making system, on the basis of linguistic information. Recent developments in fuzzy theory offer several effective methods for the design and tuning of fuzzy controllers. Most of these developments reduce the number of fuzzy rules.

Fuzzification

It is used to convert inputs i.e. crisp numbers into fuzzy sets. Crisp inputs are basically the exact inputs measured by sensors and passed into the control system for processing, such as temperature, pressure, rpm’s, etc.

Inference engine

It determines the matching degree of the current fuzzy input with respect to each rule and decides which rules are to be fired according to the input field.

Next, the fired rules are combined to form the control actions.

Defuzzification

It is used to convert the fuzzy sets obtained by inference engine into a crisp value. There are several defuzzification methods available and the best suited one is used with a specific expert system to reduce the error.

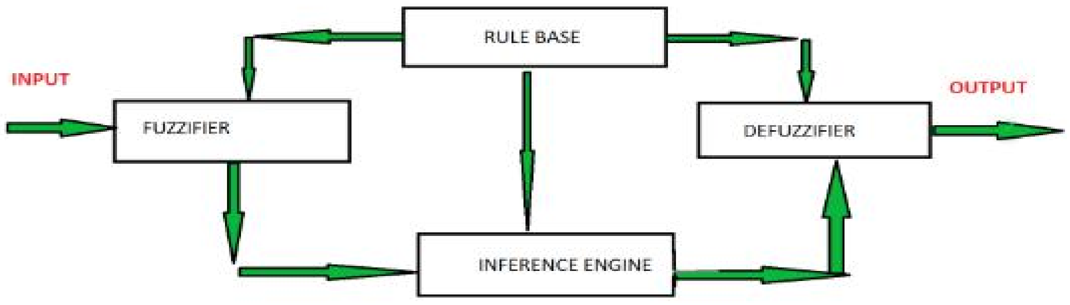


Figure 3.3 Fuzzy logic architecture

3.2.1.3 Membership function

A graph that defines how each point in the input space is mapped to membership value between 0 and 1. Input space is often referred as the universe of discourse or universal set (u), which contain all the possible elements of concern in each particular application.

There are largely three types of fusiliers:

- singleton fuzzifier,
- Gaussian fuzzifier, and
- trapezoidal or triangular fuzzifier

3.2.1.4 Advantages of Fuzzy Logic System

- This system can work with any type of inputs whether it is imprecise, distorted or noisy input information.

- The construction of Fuzzy Logic Systems is easy and understandable.
- Fuzzy logic comes with mathematical concepts of set theory and the reasoning of that is quite simple.
- It provides a very efficient solution to complex problems in all fields of life as it resembles human reasoning and decision making.
- The algorithms can be described with little data, so little memory is required.

3.2.2 FUZZY CONTROL SYSTEMS

3.2.2.1 What is Fuzzy Control?

- It is a technique to embody human-like thinkings into a control system.
- It may not be designed to give accurate reasoning but it is designed to give acceptable reasoning.
- It can emulate human deductive thinking, that is, the process people use to infer conclusions from what they know.
- Any uncertainties can be easily dealt with the help of fuzzy logic.

3.2.2.2 The Tipping problem example

The ‘tipping problem’ is commonly used to illustrate the power of fuzzy logic principles to generate complex behavior from a compact, intuitive set of expert rules.

Let’s create a fuzzy control system which models how you might choose to tip at a restaurant. When tipping, you consider the service and food quality, rated between 0 and 10. You use this to leave a tip of between 0 and 25%.

We would formulate this problem as:

Antecedents (Inputs)

Service

- Universe (ie, crisp value range): How good was the service of the wait staff, on a scale of 0 to 10?
- Fuzzy set (ie, fuzzy value range): poor, acceptable, amazing

Food quality

- Universe: How tasty was the food, on a scale of 0 to 10?
- Fuzzy set: bad, decent, great

Consequents (Outputs)

Tip

- Universe: How much should we tip, on a scale of 0% to 25%
- Fuzzy set: low, medium, high

Rules

- IF the *service* was good *or* the *food quality* was good, THEN the tip will be high.
- IF the *service* was average, THEN the tip will be medium.
- IF the *service* was poor *and* the *food quality* was poor THEN the tip will be low.

Usage

If I tell this controller that I rated:

- the service as 9.8, and
- the quality as 6.5,

it would recommend I leave:

- a 20.2% tip.

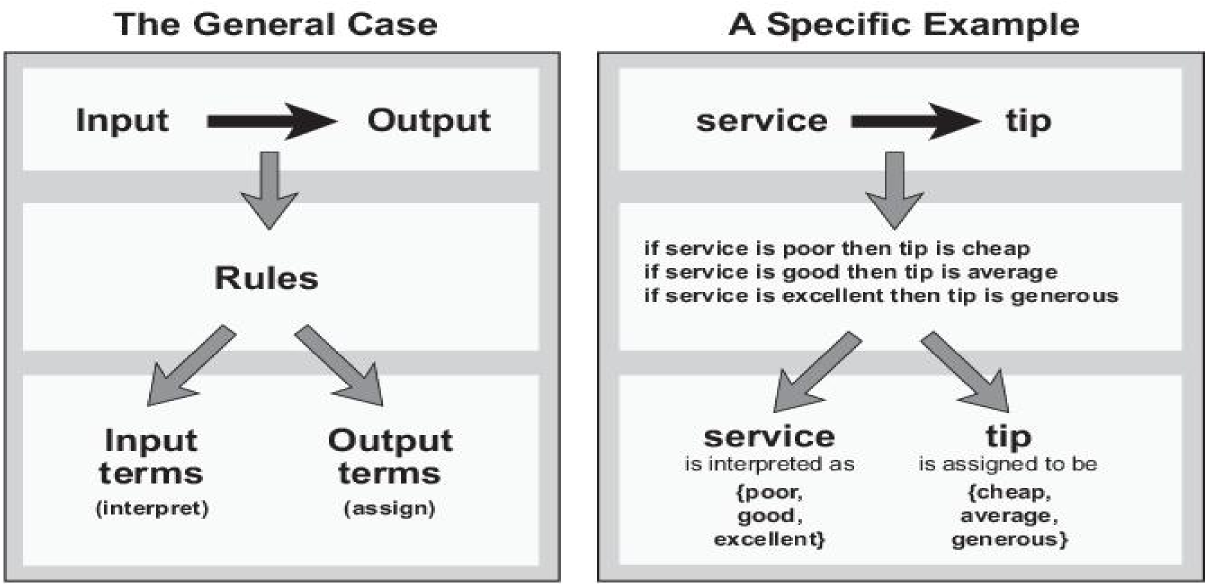

The General Case
Input
Output
Rules
Input terms (interpret)
Output terms (assign)
A Specific Example
service
tip
if service is poor then tip is cheap
if service is good then tip is average
if service is excellent then tip is generous
service is interpreted as {poor, good, excellent}
tip is assigned to be {cheap, average, generous}

Figure 3.4 Fuzzy control system example

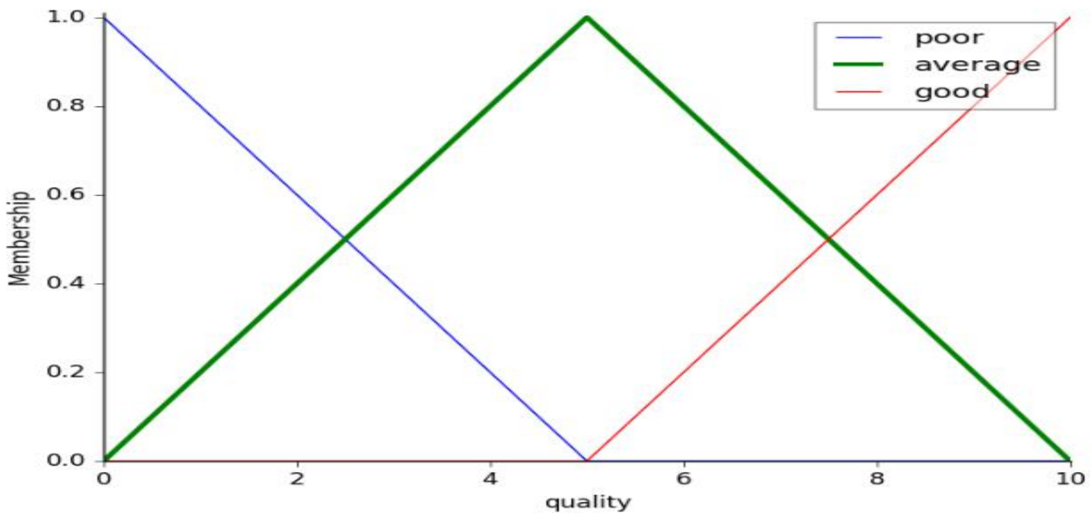


Figure 3.5 Membership function for quality

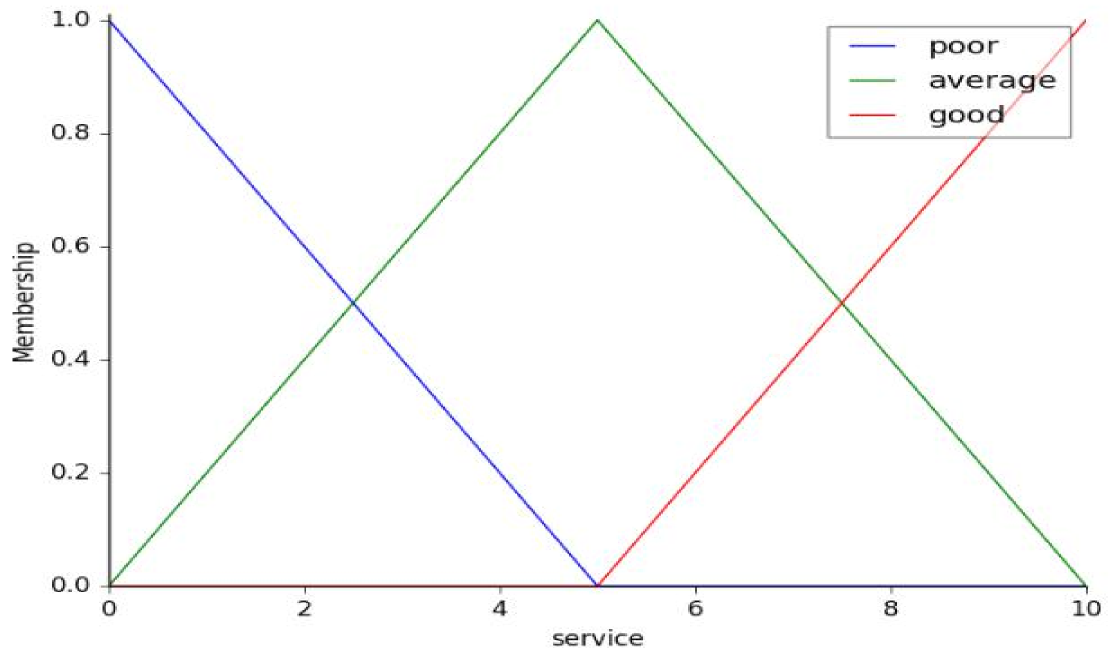


Figure 3.6 Membership function for service

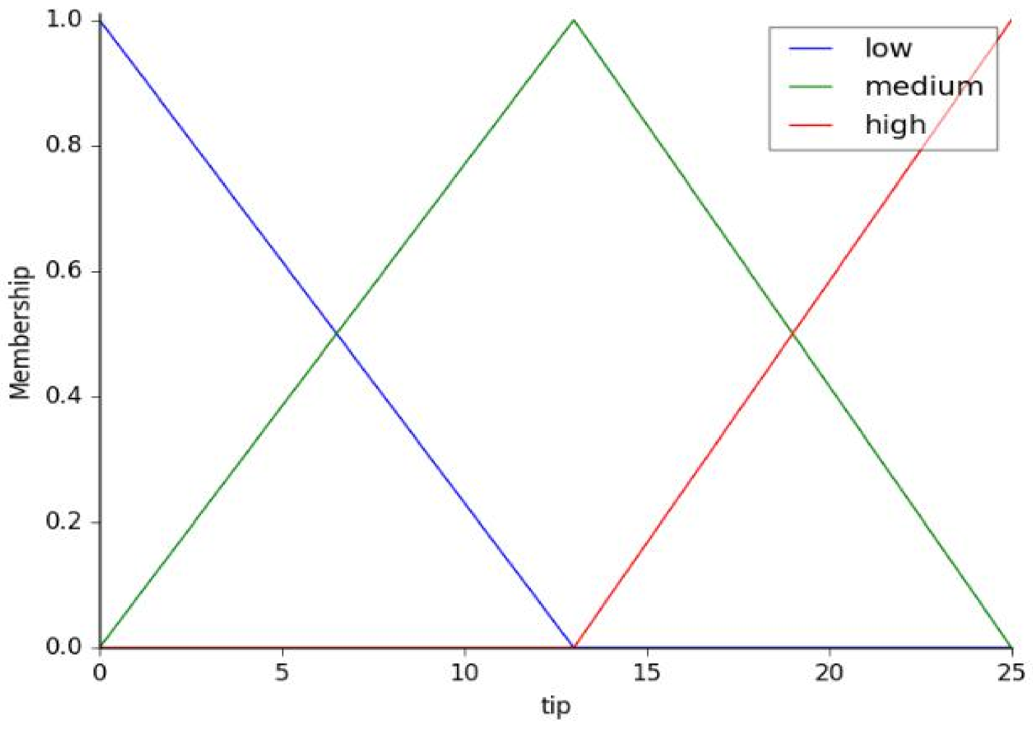


Figure 3.7 Membership function for tip

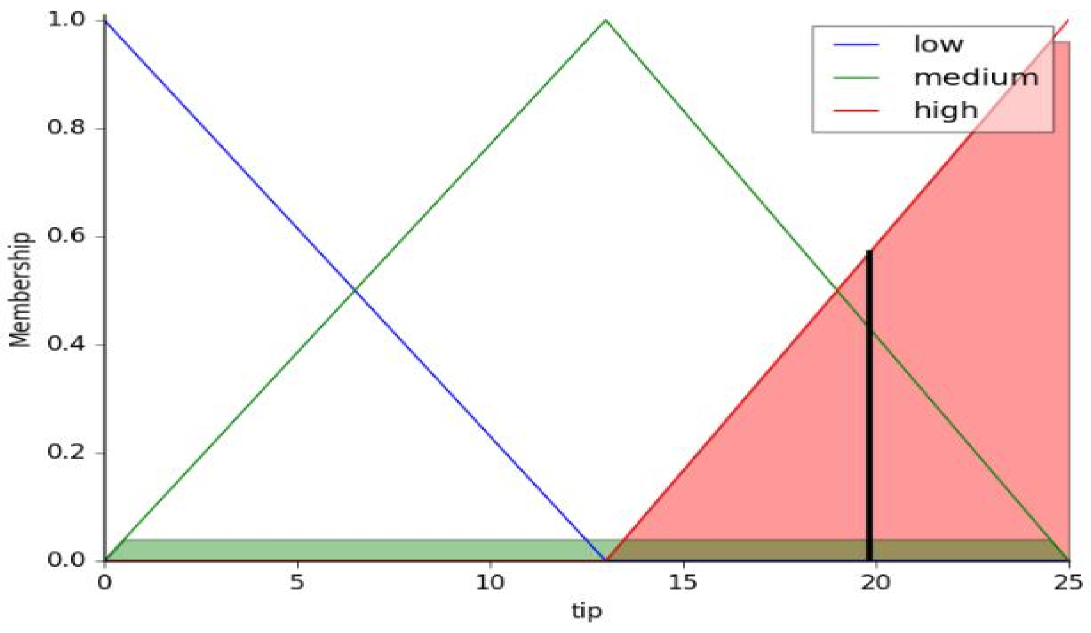


Figure 3.8 Recommended tip

The resulting suggested tip is **20.24%**.

3.3 MQTT PROTOCOL

3.3.1 What is MQTT?

MQTT (MQ Telemetry Transport) is a lightweight messaging protocol that provides resource-constrained network clients with a simple way to distribute telemetry information. The protocol, which uses a publish/subscribe communication pattern, is used for machine-to-machine (M2M) communication and plays an important role in the internet of things (IoT).

MQTT enables resource-constrained IoT devices to send, or publish, information about a given topic to a server that functions as an MQTT messagebroker. The broker then pushes the information out to those clients that have previously subscribed to the client's topic. To a human, a topic looks like a hierarchical file path. Clients can subscribe to a specific level of a topic's hierarchy or use a wild-card character to subscribe to multiple levels.

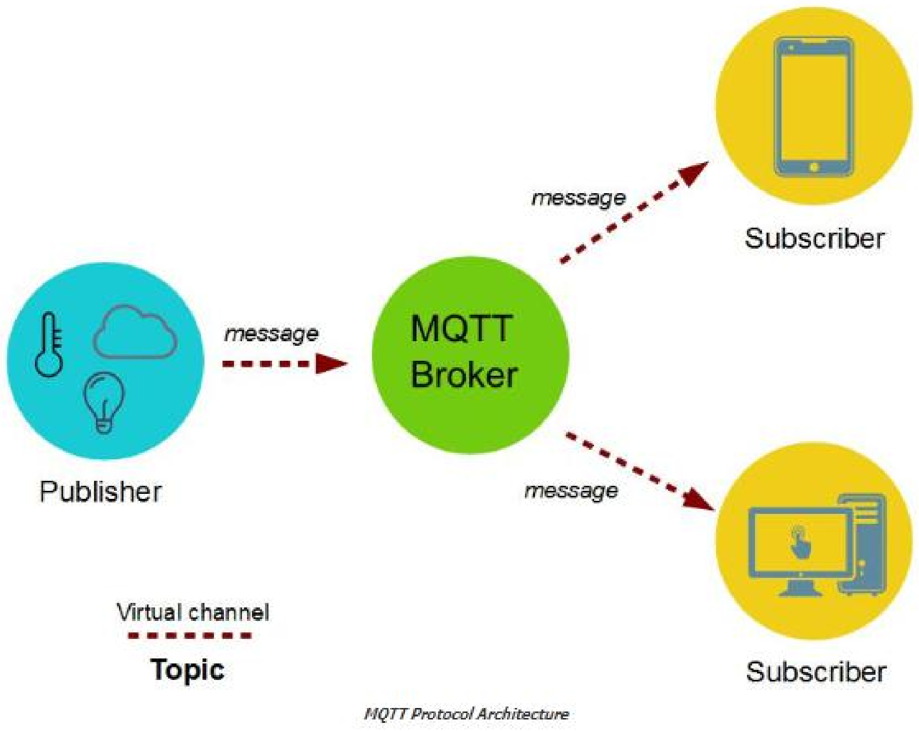


Figure 3.9 MQTT Protocol Architecture

The MQTT protocol is a good choice for wireless networks that experience varying levels of latency due to occasional bandwidth constraints or unreliable connections. Should the connection from a subscribing client to a broker get broken, the broker will buffer messages and push them out to the subscriber when it is back online. Should the connection from the publishing client to the broker be disconnected without notice, the broker can close the connection and send subscribers a cached message with instructions from the publisher.

Other transfer protocols under consideration for IoT devices with constrained resources include the Constrained Application Protocol (CoAP), which uses a request/response communication pattern, and the Advanced

Message Queuing Protocol (AMQP), which, like MQTT, uses a publish/subscribe communication pattern.

3.3.2 Working of MQTT

An MQTT session is divided into four stages: connection, authentication, communication and termination. A client starts by creating a TCP/IP connection to the broker by using either a standard port or a custom port defined by the broker's operators. When creating the connection, it is important to recognize that the server might continue an old session if it is provided with a reused client identity.

The standard ports are 1883 for non-encrypted communication and 8883 for encrypted communication using SSL/TLS. During the SSL/TLS handshake, the client validates the server certificate to authenticate the server. The client may also provide a client certificate to the broker during the handshake, which the broker can use to authenticate the client. While not specifically part of the MQTT specification, it has become customary for brokers to support client authentication with SSL/TLS client-side certificates.

Because the MQTT protocol aims to be a protocol for resource-constrained and IoT devices, SSL/TLS might not always be an option and, in some cases, might not be desired. In such cases, authentication is presented as a clear-text username and password that is sent by the client to the server as part of the CONNECT/CONNACK packet sequence. Some brokers, especially open brokers published on the internet, will accept anonymous clients. In such cases, the username and password are simply left blank.

MQTT is called a lightweight protocol because all its messages have a small code footprint. Each message consists of a fixed header -- 2 bytes -- an optional variable header, a message payload that is limited to 256 MB of information and a quality of service (QoS) level.

During the communication phase, a client can perform publish, subscribe, unsubscribe and ping operations. The publish operation sends a binary block of data -- the content -- to a topic that is defined by the publisher.

MQTT supports message BLOBS up to 256 MB in size. The format of the content is application-specific. Topic subscriptions are made using a SUBSCRIBE/SUBACK packet pair. Unsubscription is similarly performed using an UNSUBSCRIBE/UNSUBACK packet pair.

Topic strings form a natural topic tree with the use of a special delimiter character, the forward slash (/). A client can subscribe to -- and unsubscribe from -- entire branches in the topic tree with the use of special wild-card characters. There are two wild-card characters: a single-level wild-card character, the plus character (+); and a multilevel wild-card character, the hash character (#). A special topic character, the dollar character ($), excludes a topic from any root wild-card subscriptions. Typically, the $ is used to transport server-specific or system messages.

3.3.3 MQTT protocol applications and use cases

Facebook currently uses MQTT for their messenger app, not only because the protocol conserves battery power during mobile phone-to-phone messaging, but also because, in spite of inconsistent internet connections across the globe, the protocol enables messages to be delivered efficiently in milliseconds.

Most major cloud services providers, including AWS, Google Cloud, IBM Bluemix and Microsoft Azure, support MQTT, as do the Carriots, Everything and ThingWorx IoT platforms.

MQTT is well-suited to applications using M2M and IoT devices for real-time analytics, preventative maintenance and monitoring, among other uses, in environments such as smart homes, healthcare, logistics, industry and manufacturing.

CHAPTER 4

SYSTEM REQUIREMENTS

4.1 HARDWARE REQUIREMENTS

4.1.1 Node MCU

NodeMCU is an open source IoT platform.It includes firmware which runs on the ESP8266 Wi-Fi SoC from Espressif Systems, and hardware which is based on the ESP-12 module.

Specifications:

• Type	:Single-board microcontroller
• Operating system	:XTOS
• CPU	:ESP8266
• Memory	:128k bytes
• Storage	:4Mbytes
• Power	:USB

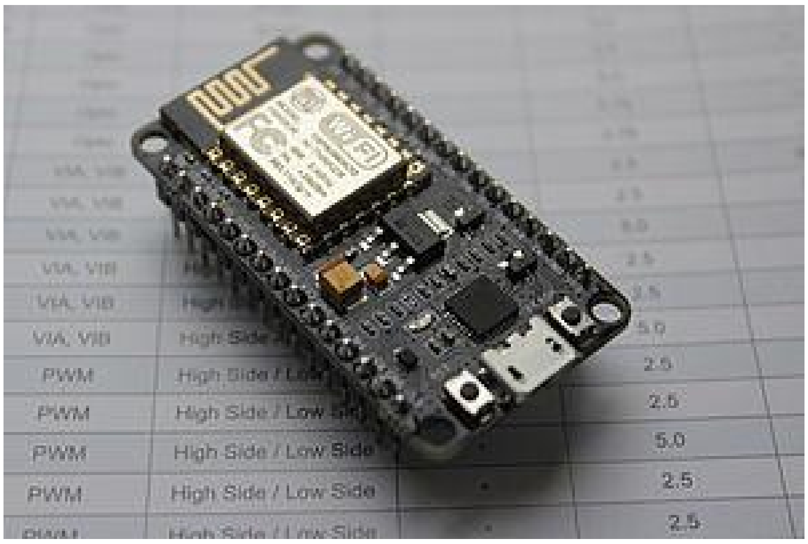


Figure 4.1 Node MCU

4.1.2 Temperature sensor(LM 35)

LM35 is a precision IC temperature sensor with its output proportional to the temperature (in ⁰C). The sensor circuitry is sealed and therefore it is not subjected to oxidation and other processes.

Features:

- Linear + 10-mV/°C Scale Factor
- 0.5°C Ensured Accuracy (at 25°C)
- Rated for Full −55°C to 150°C Range
- Operates From 4 V to 30 V
- Less Than 60-µA Current Drain
- Low Self-Heating, 0.08°C in Still Air
- Non-Linearity Only ±¼°C Typical
- Low-Impedance Output, 0.1 Ω for 1-mA Load

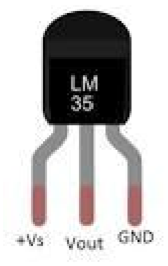


Figure 4.2 LM35 sensor

4.1.3 Smoke sensor(MQ2)

The MQ-2 is a flammable gas and smoke sensor detects the concentrations of combustible gas in the air and outputs its reading as an analog voltage.The sensor can measure concentrations of flammable gas of 300 to 10,000 ppm

Features:

- Operating Voltage is +5V
- Can be used to Measure or detect LPG, Alcohol, Propane, Hydrogen, CO and even methane
- Analog output voltage: 0V to 5V
- Digital Output Voltage: 0V or 5V (TTL Logic)
- Preheat duration 20 seconds
- Can be used as a Digital or analog sensor
- The Sensitivity of Digital pin can be varied using the potentiometer

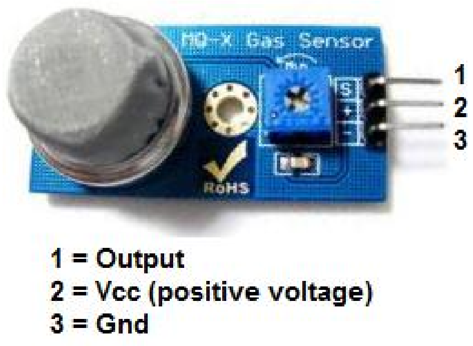


Figure 4.3 MQ-2 sensor

4.1.4 Flame sensor

A flame sensor module consists of a flame sensor (IR receiver), resistor, capacitor, potentiometer, and comparator LM393 in an integrated circuit.

Features:

- Can detect infrared light with a wavelength ranging from 700nm to 1000nm.
- The far-infrared flame probe converts the light detected in the form of infrared light into current changes.

- Sensitivity is adjusted through the onboard variable resistor with a detection angle of 60 degrees.
- Working voltage is between 3.3v and 5.2v DC, with a digital output to indicate the presence of a signal.
- Sensing is conditioned by an LM393 comparator.

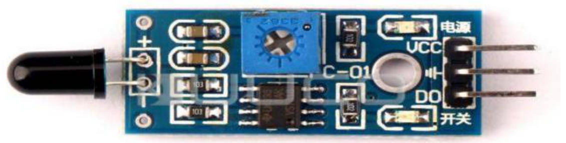


Figure 4.4 Flame sensor

4.1.5 Buzzer(Active)

An active buzzer will generate a tone using an internal oscillator, so all that is needed is a DC voltage.

Specifications:

- Module using S8550 transistor drive
- Operating voltage 3.3V-5V
- With a fixed bolt hole, easy to install
- Small board PCB size: 3.2cm * 1.3cm approx
- When the I/O port input low, the buzzer Sound

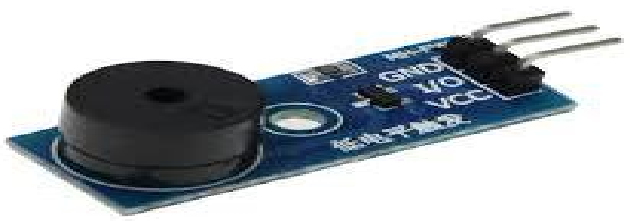


Figure 4.5 Active buzzer

4.2 SOFTWARE REQUIREMENTS:

4.2.1 Arduino IDE

The Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. It connects to the Arduino and Genuino hardware to upload programs and communicate with them.

Programs written using Arduino Software (IDE) are called sketches. These sketches are written in the text editor and are saved with the file extension .ino. The editor has features for cutting/pasting and for searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino Software (IDE), including complete error messages and other information. The bottom right-hand corner of the window displays the configured board and serial port. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.

4.2.2 Android Studio

Android Studio is the official integrated (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development.

Features:

- Gradle based build support.
- Android-specific refactoring and quick fixes.
- Lint tools to catch performance, usability, version compatibility and other problems.

- ProGuard integration and app-signing capabilities.
- Template-based wizards to create common Android designs and components.
- A rich layout editor that allows users to drag-and-drop UI components, option to preview layouts on multiple screen configurations.
- Support for building Android Wear apps.
- Built-in support for Google Cloud Platform, enabling integration with Firebase Cloud Messaging (Earlier 'Google Cloud Messaging') and Google App Engine.
- Android Virtual Device (Emulator) to run and debug apps in the Android studio.

4.2.3 Hive MQ

Hive MQ is a MQTT based messaging platform designed for the fast, efficient and reliable movement of data to and from connected IoT devices. It uses the MQTT protocol for instant, bi-directional push of data between your device and your enterprise systems.

Key features:

- Scalable MQTT Broker. Hive MQ broker instances scale with the underlying hardware.
- Efficient Network Utilization.
- Reliable Data Delivery.
- Enterprise Data Integration.
- Elastic Clustering.
- Real-time Data Monitoring.
- Enterprise-grade security.
- Extension Framework.

CHAPTER 5

IMPLEMENTATION

5.1 Setting up Arduino IDE

- Download Arduino IDE.
- Open your IDE and click on **"File -> Preferences".**
- In "Additional Boards Manager URLs" add this line and click on "OK":

  http://arduino.esp8266.com/stable/package_esp8266com_index.json
- Go to **"Tools -> Board -> Boards Manager"**, type "ESP8266" and install it.
- Go again to **"Tools -> Board"** and select "Generic ESP8266 Module".
- Install necessary libraries like Adafruit MQTT Library ,Arduino Uno WiFi Dev Ed Library,PubSubClient etc.

5.2 Connecting the sensors and reading data

The temperature sensor(LM 35),the smoke sensor(MQ-2) and the flame sensor are connected to the node MCU.

Correct mapping has to be done between NodeMCU and ESP8266 pins. The numbers of the pins in the board don't map to the numbers of the pins on the ESP8266. So, for example, pin D1 of the board doesn't map to GPIO1 of the ESP8266 (it actually maps to GPIO5).

So, the correct pin mapping is the following (NodeMCU on the left and ESP8266 on the right):

- D0 = GPIO16;
- D1 = GPIO5;

- D2 = GPIO4;
- D3 = GPIO0;
- D4 = GPIO2;
- D5 = GPIO14;
- D6 = GPIO12;
- D7 = GPIO13;
- D8 = GPIO15;
- D9 = GPIO3;
- D10 = GPIO1;
- LED_BUILTIN = GPIO16 (auxiliary constant for the board LED, not a board pin);

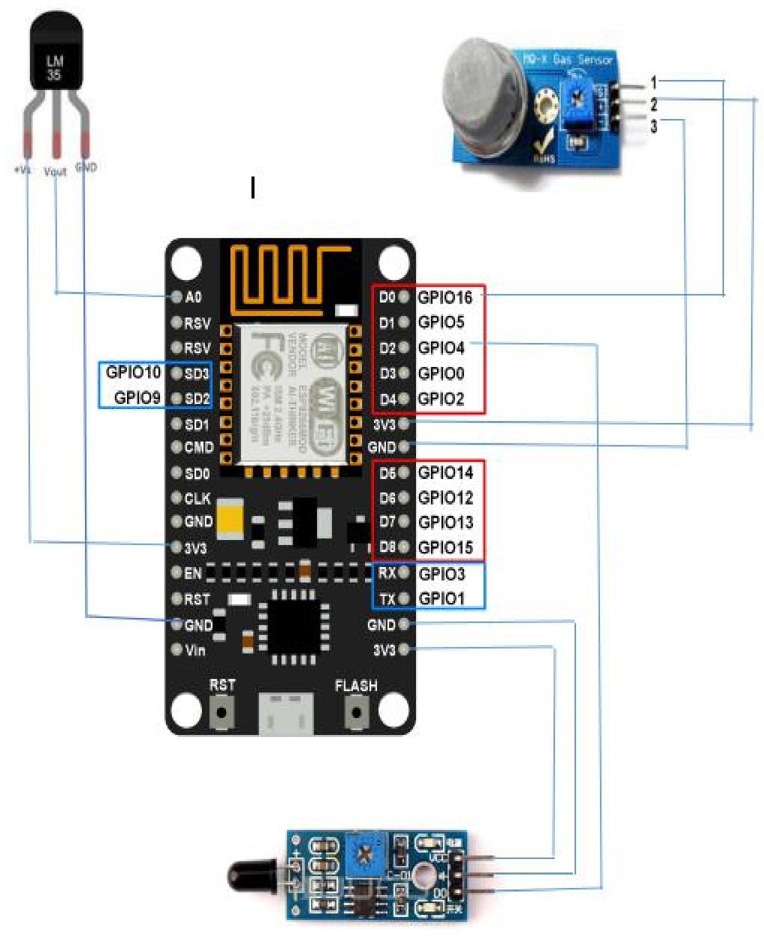


Figure 5.1 NodeMCU connection with sensors

Code snippet:

```
int flamePin=4;//D2 of the Node MCU

void ReadTemp()

{

  t = (analogRead(A0) * 330.0) / 1023.0;

  Serial.print("lms 35 Sensor value:");

  Serial.println(t);

}

void gasSensor()

{

  sensorValue = analogRead(D0); // read analog input pin 0

  Serial.print("smoke Sensor Value: ");

  Serial.print(sensorValue);

}

void flameSensor()

{

  Flame = digitalRead(flamePin);

  Serial.println(Flame);

}
```

5.3 Prediction of occurrence of forest fire

Based on the threshold values of the sensors the nodes are classified into one of the following:no forest fire, mild, moderate and severe.

Code snippet:

```
float t_severe=37.5;

float t_moderate=33.5;

float t_mild=31.0;

float t_curr=0.0;

void classify()

{

  if(Flame==0 || t>t_severe|| sensorValue==1)

  {

    Status1="Severe";

    Buzzerfn();

  }

  else if(t>=t_moderate || (t>t_mild && sensorValue==1))

  {

    Status1="Moderate";

    Buzzerfn();
```

```
          }

          else if(t>=t_mild)

          {

            Status1="Mild";

            Buzzerfn();

          }

          else

          {

            Status1="No Forest fire";

          }
```

Fuzzy logic is applied such that it gives the percentage of occurrence of forest fire.Detailed explanation of fuzzy logic is given in section 3.2.

Code snippet:

```
float predict(float T,float smoke,float flame ){

  float C1 = 1; //severe

  float C2 = 0.66;//moderate

  float C3 = 0.33; //mild
  if(T>t_severe) T=t_severe; // if temperature is above 35°C, set temperature to
maxiimum in our range

  if(flame==0) flame=1;

  else flame=0;

   // Fuzzy rules

  float w1 = trimf(T,t_moderate,t_severe,t_severe);//severe

  float w2 = trimf(T,t_mild,t_moderate,t_severe);//moderate

  float w3 = trimf(T,t_curr,t_mild,t_moderate);//mild

  // Defuzzyfication

  float z = (w1*C1 + w2*C2 + w3*C3)/(w1+w2+w3);

  return z;

}
```

5.4 Publishing data to the MQTT broker

- Define the online MQTT broker (Hive MQ) and the client ID.
- The client id identifies the ESP8266 device
- Initialise the WiFi and MQTT Client objects
- 1883 is the listener port for the Broker.

Code snippet:

```
#include <PubSubClient.h> // Allows us to connect to, and publish to the
MQTT broker
```

```
const char* mqtt_server = "broker.hivemq.com";
 const char* clientID = "client-1";
WiFiClient wifiClient;
PubSubClient client(mqtt_server, 1883, wifiClient);
```

Define the MQTT topics (virtual channels) through which the data will be published to the MQTT broker.

Code snippet:

```
const char* tem="archan/temperature";
const char* gas="archan/gas";
const char* flame="archan/flame";
const char* status1="archan/status";
```

Then the sensor data and the classified output is published through the respective channels to the MQTT broker.

Code snippet:

```
if (client.connect(clientID)) {
  Serial.println("Connected to MQTT Broker!");
  client.publish(tem,String(t).c_str());
  client.publish(gas,String(SmokeStatus).c_str());
   client.publish(flame,String(Flame).c_str());
   client.publish(status1,String(Status1).c_str());
  }
 else {
   Serial.println("Connection to MQTT Broker failed...");
  }
```

5.5 Displaying the output

An app in a client device is used to display the output where one can continuously receive the sensor data the classified output which tells if there is chance of occurrence of the forest fire.

The first step is to connect the client device to the online MQTT broker.

Code snippet:

```
String clientId = MqttClient.generateClientId();
constraints constants=new contraints();


final MqttAndroidClient client = new MqttAndroidClient(getApplication
context(),constants.MQTT_BROKER_URL,clientId);
```

Then the client device subscribes to the various topics such as temperature,gas,flame and status and the MQTT broker sends the data of those topics to the client device.

Code snippet:

```
client.subscribe(topic, 0, null, new IMqttActionListener() {
```

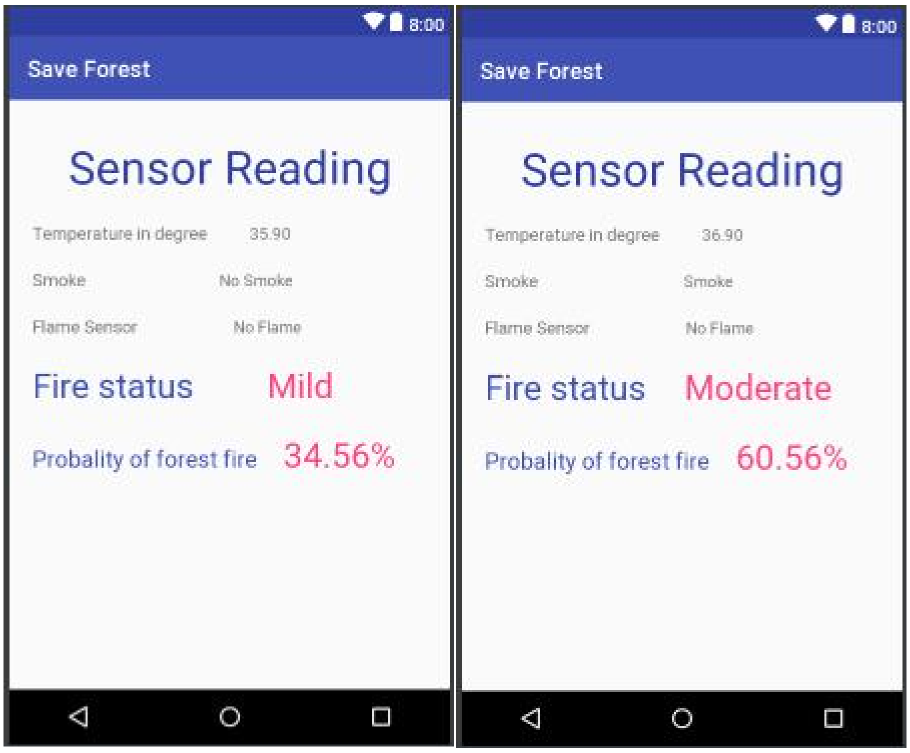
```
            @Override

            public void onSuccess(IMqttToken asyncActionToken) {

              // Log.w("Mqtt", "Subscribed!");

               Toast.makeText(MainActivity.this,"Subscribed!!!!!!",
Toast.LENGTH_SHORT).show();

            }
```



Save Forest	8:00
Sensor Reading	
Temperature in degree	35.90
Smoke	No Smoke
Flame Sensor	No Flame
Fire status	Mild
Probality of forest fire	34.56%

Save Forest	8:00
Sensor Reading	
Temperature in degree	36.90
Smoke	Smoke
Flame Sensor	No Flame
Fire status	Moderate
Probality of forest fire	60.56%

Figure 5.2 Output in the app

Additonal enhancement:

An active buzzer is connected to the NodeMCU which produces a warning alarm sound which will be useful to alert the public or even wildlife to refrain from going into the forest when there is a chance of occurrence of forest fire.

CHAPTER 6

APPENDIX

Three_sensors.ino

```
#include <ESP8266WiFi.h>

#include <PubSubClient.h> // Allows us to connect to, and publish to the MQTT broker

#include <FirebaseArduino.h>

//defining constant

#define FIREBASE_HOST "nodemcu-de561.firebaseio.com"

#define FIREBASE_AUTH "kHRY8t2cmkH0C4lChOj6ro2CfCVATgvoRU0owcVX"

#define WIFI_SSID "Arch_angel"

#define WIFI_PASSWORD "archan1997"

// MQTT

// Make sure to update this for your own MQTT Broker!

const char* mqtt_server = "broker.hivemq.com";

// The client id identifies the ESP8266 device. Think of it a bit like a hostname (Or just a name, like Greg).

const char* clientID = "client-1";

// Initialise the WiFi and MQTT Client objects

WiFiClient wifiClient;
```

```
PubSubClient client(mqtt_server, 1883, wifiClient); // 1883 is the listener port
for the Broker

//MQTT TOPICS

const char* temperature="archan/temperature";

const char* gas="archan/gas";

const char* flame="archan/flame";

const char* status1="archan/status";

const char* Status1="No Forest fire";

const char* SmokeStatus="No Smoke";

const char* prob="archan/prob";

float Prob=0.0;

float t=0.0;

int sensorValue;

int Flame= 0;

int flamePin=4;

int temp=0;

float t_severe=37.5;

float t_moderate=33.5;

float t_mild=31.0;

float t_curr=0.0;

int buzzer=0;
```

```
void setup() {
 // put your setup code here, to run once:
  Serial.begin(9600);
 Serial.setTimeout(2000);
 //pinMode(buzzer, OUTPUT);
  Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
 Firebase.set("flame", 0);
 Firebase.set("temperature", 0.0);
 Firebase.set("gas",0);
}
void loop() {
 if (client.connect(clientID)) {
   Serial.println("Connected to MQTT Broker!");
  client.publish(temperature,String(t).c_str());
   client.publish(gas,String(SmokeStatus).c_str());
   client.publish(flame,String(Flame).c_str());
   client.publish(status1,String(Status1).c_str());
   client.publish(prob,String(Prob).c_str());
 }
 else {
  Serial.println("Connection to MQTT Broker failed...");
 }
 ReadTemp();
 gasSensor();
 flameSensor();
 prediction();
}
void Buzzerfn()
{
 digitalWrite(buzzer, HIGH);
 delay(200);
 digitalWrite(buzzer, LOW);
 delay(200);
 pinMode(2,HIGH);
}
void classify()
{
 if(Flame==0 || t>t_severe|| sensorValue==1)
 {
   Status1="Severe";
   Buzzerfn();
 }
```

```
 else if(t>=t_moderate || (t>t_mild && sensorValue==1))
 {
  Status1="Moderate";
  Buzzerfn();
 }
 else if(t>=t_mild)
 {
  Status1="Mild";
  Buzzerfn();
 }
 else
 {
  Status1="No Forest fire";
 }
 float conf=predict(t,sensorValue,Flame);
Serial.println("confort level=");
Prob=conf;
}
float trimf(float x,float a, float b, float c){
 float f;
 if(x<=a)
```

```
    f=0;

  else if((a<=x)&&(x<=b))

    f=(x-a)/(b-a);

  else if((b<=x)&&(x<=c))

    f=(c-x)/(c-b);

  else if(c<=x)

    f=0;

  return f;

}

// Function for predicting atmospheric comfort from temperature and relative
humidity

float predict(float T,float smoke,float flame ){

  float C1 = 1; //severe

  float C2 = 0.66;//moderate

  float C3 = 0.33; //mild

   if(T>t_severe) T=t_severe; // if temperature is above 35°C, set temperature to
maxiimum in our range

  if(flame==0) flame=1;

  else flame=0;

   // Fuzzy rules

  float w1 = trimf(T,t_moderate,t_severe,t_severe);//severe
  float w2 = trimf(T,t_mild,t_moderate,t_severe);//moderate
  float w3 = trimf(T,t_curr,t_mild,t_moderate);//mild
   // Defuzzyfication
  float z = (w1*C1 + w2*C2 + w3*C3)/(w1+w2+w3);
  return z;
}
void ReadTemp()
{
  if(temp==0)
  {
    t_curr=((analogRead(A0)*330.0)/1023.0);
    t_mild=t_curr+ 2.0 ;
    t_moderate=t_mild+3.0;
    t_severe=t_moderate+2.0;
    Serial.print("Mild temp is");Serial.println(t_mild);
    Serial.print("Moderate temp is");Serial.println(t_moderate);
    Serial.print("severe temp is");Serial.println(t_severe);
    temp=1;
  }
  t = (analogRead(A0) * 330.0) / 1023.0;
  Serial.print("lms 35 Sensor value:");
```

```
 Serial.println(t);
 Firebase.set("temperature", t);
 }
void gasSensor()
{
 sensorValue = digitalRead(D0); // read analog input pin 0
 Serial.print("smoke Sensor Value: ");
 Serial.print(sensorValue);
 Firebase.set("gas", sensorValue);
 if(sensorValue==1)
 {
  SmokeStatus="Critical Smoke Detected";
 }
 else if(sensorValue==0)
 {
  SmokeStatus="No Smoke";
 }
}
void flameSensor()
{
 Flame = digitalRead(flamePin);
```

```
  Firebase.set("flame",Flame);

  if (Flame== 0)

  {

    Serial.println("Fire!!!");

  }

  else if(Flame==1)

  {

    Serial.println("No worries");

  }

}
```

Activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="10dp"
    tools:context="com.example.archan.saveforest.MainActivity">
<TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
    android:textAlignment="center"
    android:text="Sensor Reading"
    android:textIsSelectable="false"
    android:textColor="@color/colorPrimaryDark"
    android:textSize="40dp"
    android:paddingBottom="10dp"
    android:paddingTop="20dp"/>
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:padding="10dp">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="0.5"
        android:textSize="15dp"
        android:text="Temperature in degree"
        android:paddingRight="10dp"
        />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:id="@+id/temp"
        android:text="0.0"
        android:paddingRight="10dp"/>
</LinearLayout>
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:padding="10dp">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:textSize="15dp"
        android:text="Smoke"
        android:paddingRight="10dp"
        />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:id="@+id/gas"
        android:text="0.0"
        android:paddingRight="10dp"/>
</LinearLayout>
```

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:padding="10dp">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1.5"
        android:textSize="15dp"
        android:text="Flame Sensor"
        android:paddingRight="10dp"
        />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:id="@+id/flame"
        android:text="0.0"
        android:paddingRight="10dp"/>
</LinearLayout>
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:padding="10dp">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1.5"
        android:textSize="30dp"
        android:text="Fire status"
        android:textColor="@color/colorPrimary"
        android:paddingRight="10dp"
        />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:id="@+id/status"
        android:textSize="30dp"
        android:text="No Fire"
        android:textColor="@color/colorAccent"
        android:paddingRight="10dp"/>
</LinearLayout>
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:padding="10dp">
```

```
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1.5"
            android:textSize="20dp"
            android:text="Probality of forest fire"
            android:textColor="@color/colorPrimary"
            android:paddingRight="10dp"
            />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="2"
            android:id="@+id/prob"
            android:textSize="30dp"
            android:text="0.0%"
            android:textColor="@color/colorAccent"
            android:paddingRight="10dp"
            />
    </LinearLayout>
</LinearLayout>
```

MainActivity.java:

```
package com.example.archan.saveforest;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;
import android.widget.Toast;
import org.eclipse.paho.android.service.MqttAndroidClient;
import org.eclipse.paho.client.mqttv3.IMqttActionListener;
import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.IMqttMessageListener;
import org.eclipse.paho.client.mqttv3.IMqttToken;
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.MqttCallbackExtended;
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import java.io.UnsupportedEncodingException;
import java.lang.annotation.Annotation;
import static com.example.archan.saveforest.contraints.flame;
import static com.example.archan.saveforest.contraints.gas;
import static com.example.archan.saveforest.contraints.prob;
import static com.example.archan.saveforest.contraints.status;
import static com.example.archan.saveforest.contraints.temp;
public class MainActivity extends AppCompatActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
```

```
        final TextView tempText = (TextView) findViewById (R.id.temp);

        final TextView gasText = (TextView) findViewById (R.id.gas);

        final TextView flameText = (TextView) findViewById (R.id.flame);

        final TextView statusText = (TextView) findViewById (R.id.status);

        final TextView probText = (TextView) findViewById (R.id.prob);

        String clientId = MqttClient.generateClientId();

        contraints constants=new contraints();

        final MqttAndroidClient client = new
MqttAndroidClient(getApplicationContext(),
constants.MQTT_BROKER_URL,clientId);

 client.setCallback(new MqttCallbackExtended() {

            @Override

            public void connectComplete(boolean b, String s) {

                Log.w("mqtt", s);

            }

            @Override

            public void connectionLost(Throwable throwable) {


            }

            @Override

            public void messageArrived(String topic, MqttMessage mqttMessage)
throws Exception {

                // Log.w("Mqtt", mqttMessage.toString());

                 //Toast.makeText(MainActivity.this,
topic+":"+mqttMessage.toString(), Toast.LENGTH_SHORT).show();
```

```
                if (topic.equals("archan/temperature"))
                {
                    //Toast.makeText(MainActivity.this,
topic+":"+mqttMessage.toString(), Toast.LENGTH_SHORT).show();

                    tempText.setText(mqttMessage.toString());
                }
                else if(topic.equals("archan/gas"))
                {
                    //Toast.makeText(MainActivity.this,
topic+":"+mqttMessage.toString(), Toast.LENGTH_SHORT).show();

                    gasText.setText(mqttMessage.toString());
                }
                else if(topic.equalsIgnoreCase("archan/flame"))
                {
                    //Toast.makeText(MainActivity.this,
topic+":"+mqttMessage.toString(), Toast.LENGTH_SHORT).show();

                    if(mqttMessage.toString().equalsIgnoreCase("1"))
                    flameText.setText("NO flame");
                    else
                    {
                        flameText.setText("flame");
                    }
                }
```

```
            else if(topic.equalsIgnoreCase("archan/status"))
            {
                statusText.setText(mqttMessage.toString());
            }
            else if(topic.equals("archan/prob"))
            {
                probText.setText(mqttMessage.toString()+"% chance");
            }
        }
        @Override
        public void deliveryComplete(IMqttDeliveryToken
iMqttDeliveryToken) {


        }
    });
    try {
        client.connect().setActionCallback(new IMqttActionListener() {
            @Override
            public void onSuccess(IMqttToken asyncActionToken) {
                // We are connected
                Toast.makeText(MainActivity.this, "Connected to Mqtt
Broker!!!!!!", Toast.LENGTH_SHORT).show();
                    SubscribeTopic(client, contraints.temp);
                    SubscribeTopic(client,flame);
                    SubscribeTopic(client,gas);
                    SubscribeTopic(client,status);

                    SubscribeTopic(client,prob);

            }

            @Override

            public void onFailure(IMqttToken asyncActionToken, Throwable
exception) {

                // Something went wrong e.g. connection timeout or firewall
problems

                Toast.makeText(MainActivity.this, "Connection to Mqtt Broker
Failed", Toast.LENGTH_SHORT).show();

            }

        });

    } catch (MqttException e1) {

        e1.printStackTrace();

    }

}

public void SubscribeTopic(MqttAndroidClient client, final String  topic) {

    try {

        //Toast.makeText(this, "try to subscribe...........",
Toast.LENGTH_SHORT).show();

        client.subscribe(topic, 0, null, new IMqttActionListener() {

        @Override

        public void onSuccess(IMqttToken asyncActionToken) {

                // Log.w("Mqtt", "Subscribed!");

            Toast.makeText(MainActivity.this, "Subscribed!!!!!!",
Toast.LENGTH_SHORT).show();
```

```
            }
            @Override
            public void onFailure(IMqttToken asyncActionToken, Throwable
exception) {
                //Log.w("Mqtt", "Subscribed fail!");
                Toast.makeText(MainActivity.this, "Subscribed fail!",
Toast.LENGTH_SHORT).show();
            }
        });

    } catch (MqttException ex) {
        System.err.println("Exceptionst subscribing");
        ex.printStackTrace();
    }
  }
}
```

Constraints.java:

```
public class constraints {
   public static final String MQTT_BROKER_URL =
"tcp://broker.hivemq.com:1883";
   public static final String PUBLISH_TOPIC = "androidkt/topic";
   public static final String CLIENT_ID = "androidkt";
   public static final  String  temp="archan/temperature";
   public static final  String  flame="archan/flame";
```

```
  public static final  String  gas="archan/gas";

  public static final  String  status="archan/status";

  public static final  String  prob="archan/prob";

}
```

CHAPTER 7

CONCLUSION AND FUTURE WORK

The aim of this project is to reduce the damage and destruction caused by the forest fire to the life and property of humans and also wild animals. This is the reason to predict the fire in advance with the help of the data obtained from the sensors that are deployed in the forest.

The data collected can be pushed into a database continuously for future use. It can used to refine the rules and make it more accurate. Decisions on fixing threshold will be made a lot more easier. It is an added benefit while performing analysis, as large sets of real-time data gives a more accurate analysis.

The Firebase Real-time Database is a cloud-hosted database. Data is stored as JSON and synchronized in real-time to every connected client. When you build cross-platform apps with our iOS, Android, and JavaScript SDKs, all of your clients share one Real-time Database instance and automatically receive updates with the newest data.This could be used to push data or when using a local MQTT broker(Raspberry PI with Mosquitto MQTT broker installed),a local database like sqlite3 can also be used if preferred.

The better available solution for forest fire prediction is using sensors because it can provide all required information that influences the environment at any point in time precisely.These sensors are small, low cost , fairly densely that can observe and influence the physical changes in the forest around them and gathers all the physical information, transforming it into electrical signals, does all the analysis or prediction, and sends it to a remote location for further precautions which should be taken care by the officials.

REFERENCES

[1] A. Chauhan, S. Semwal and R. Chawhan, "Artificial neural network-based forest fire detection system using wireless sensor network," 2013 Annual IEEE India Conference (INDICON), Mumbai, 2013, pp. 1-6.
doi: 10.1109/INDCON.2013.6725913

[2] Anupam Mittal, Geetika Sharma, Ruchi Aggarwal, "Forest Fire Detection Through Various Machine Learning Techniques using Mobile Agent in WSN"International Research Journal of Engineering and Technology (IRJET) Volume: 03, Issue: 06, June-2016

[3] Dutta M., Bhowmik S., Giri C. (2014) Fuzzy Logic Based Implementation for Forest Fire Detection Using Wireless Sensor Network. In: Kumar Kundu M., Mohapatra D., Konar A., Chakraborty A. (eds) Advanced Computing, Networking and Informatics- Volume 1. Smart Innovation, Systems and Technologies, vol 27. Springer, Cham

[4] D.Vignesh Kirubaharan, A.John Clement Sunder, S.M.Ramesh, P.Dhinakar, "Forest Fire Prediction and Alert System Using Wireless Sensor Network"International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE) Volume 3, Issue 11, November 2014

[5] G. Demin, L. Haifeng, J. Anna and W. Guoxin, "A forest fire prediction system based on rechargeable wireless sensor networks," 2014 4th IEEE International Conference on Network Infrastructure and Digital Content, Beijing, 2014, pp. 405-408.

[6] Liyang Yu, Neng Wang and Xiaoqiao Meng, "Real-time forest fire detection with wireless sensor networks," Proceedings. 2005 International Conference on Wireless Communications, Networking and Mobile Computing, 2005., Wuhan, China, 2005, pp. 1214-1217

[7] Sukumar.K, Joevivek.V, Hemalatha.T, Soman.K.P, "Prediction of forest fire using SVM – a novel approach"10th ESRI India User Conference 2009

[8] V. G. Gasull, D. F. Larios, J. Barbancho, C. León and M. S. Obaidat, "Computational intelligence applied to wildfire prediction using wireless sensor networks," Proceedings of the International Conference on, Seville, Spain, 2011, pp. 1-8.