A

MAJOR PROJECT REPORT

on

Intelligent Bus Monitoring and Alert System Based on IoT

Submitted in partial fulfillment of the Requirements for the award of

Degree

of

Bachelor of Technology

in

ELECTRONICS & COMMUNICATION ENGINEERING

By

K.RAJU	**-**	**22681A0433**
M. MANOJ	**-**	**23685A0408**
S. MANOJ KUMAR	**-**	**22681A0452**
S. MANU	**-**	**22681A0450**

under the esteemed guidance of

Mr. K. Amarender
M.TECH
Assistant professor

CHRISTU JYOTHI INSTITUTE OF TECHNOLOGY AND SCIENCE

Colombo Nagar, Yeshwanthpur, Jangaon, TS 506167, Telangana

Department of Electronics & Communication Engineering

(Accredited by National Board of Accreditation)

2025-2026

CHRISTU JYOTHI INSTITUTE OF TECHNOLOGY & SCIENCE

(Affiliated to JNTU, Hyderabad)

Colombonagar, Yeshwanthapur, Jangaon Dist. 506167, Telangana.

Dept. Electronics & Communication Engineering

2025-2026

CERTIFICATE

This is to certify that the work which is being presented in the B. Tech. Mini Project Reportentitled **"Intelligent Bus Monitoring and Alert System Based on IoT"** being submitted by **22681A0433 (K. RAJU)**, impartial fulfillment of the requirements for the award of the Bachelor of Technology in **"Electronics & Communication Engineering"** and submitted to the Department of Electronics & Communication Engineering of Christu Jyothi Institute of Technology and Science, Jangaon.

This is to certify that the above statement made by the candidate is correct to the best of my knowledge

Signature of Guide	**Signature of HOD**
Mr. K. Amarender	**Mr. ALLANKI SANYASI RAO**
ASSISTANT PROFESSOR	**ASSOCIATE PROFESSOR**

Signature of External Examine

CHRISTU JYOTHI INSTITUTE OF TECHNOLOGY & SCIENCE

(Affiliated to JNTU, Hyderabad)

Colombonagar, Yeshwanthapur Jangaon Dist

Institute Vision and Mission

Vision

To admit and groom students from rural background and be a truly rural technical institution, benefiting society and nation as a whole institute.

Mission

- The mission of the institution is to create, deliver and refine knowledge. Being a rural technical institute, our mission is to.
- Enhance our position to one of the best technical institutions and to measure our performance against the highest defined standards.
- Provide highest quality learning environment to our students for their greater well-being so as to equip them with highest technical and professional ethics.
- Produce engineering graduates fully equipped to meet the ever-growing needs of industry and society

CHRISTU JYOTHI INSTITUTE OF TECHNOLOGY & SCIENCE

(Affiliated to JNTU, Hyderabad)

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

VISION

To be an established center of excellence in Electronics and Communication Engineering facilitating youth towards professional, leadership and industrial needs.

MISSION

- Impart theoretical and practical technical education of high standard with quality resources and collaborations.
- Organize training's and activities towards Overall personality development in time with industrial need.
- Promote innovation towards sustainable solutions with multi discipline team work with ethics.

HEAD OF THE DEPARTMENT

DECLARATION

I'm hereby declare that the project entitled--**" Intelligent Bus Monitoring and Alert System Based on IoT ",** which us being submitted as Mini Project in Electronics and Communication Engineering to Christu Jyothi Institute of Technology & Science, is an authentic record of our genuine work done under the guidance of K. Amarender, Assistant Professor of ECE Dept.

PROJECTMEMBER:

K.RAJU - 22681A0433
M. MANOJ - 23685A0408
S. MANOJ KUMAR - 22681A0452
S. MANU - 22681A0450

ACKNOWLEDGMENT

We hereby express our sincere gratitude to the **Management of Christu Jyothi Institute of Technology & Science** for their kind encouragement bestowed up on us to do this Mini project.

We earnestly take the responsibility to acknowledge the following distinguished personalities who graciously allowed our project work successfully.

We express our sincere thanks to our director **Rev.Fr. D. Vijay Paul Reddy**, Principal **Mr. Dr. S. Chandrashekhar Reddy** for his encouragement, which has motivated us to strive hard to excel in our discipline of engineering.

We are greatly indebted to the professor and Head of the Department **Mr. Allanki Sanyasi Rao, Associate Professor** for his motivation and guidance through the course of this project work. He has been responsible for providing us with lot of splendid opportunities, which has shaped our career. His advice ideas and constant support has engaged us on and helped us get through in difficult time.

We express our profound sense of appreciation and gratitude to our guide **K. Amarender**, **Assistant Professor** for providing generous assistance, and spending many hours of valuable time with us. This excellent guidance has made the timely completion of this mini project.

Last but not the least, we express our gratitude to the Teaching and Non-Teaching Staff ofthe Department of Electronics and communication for their needy and continuous support in technical assistance.

ABSTRACT

In the era of smart transportation, ensuring the safety, efficiency, and accountability of public transit systems has become a critical challenge. This project introduces an Intelligent Bus Monitoring and Alert System that leverages advanced Internet of Things (IoT) technologies to provide real-time surveillance, anomaly detection, and automated alerting for public buses. The system is designed to address key issues such as unauthorized access, route deviations, overcrowding, and emergency response delays—common problems in conventional bus operations.

The proposed solution integrates multiple sensors including GPS modules for location tracking, infrared (IR) sensors for passenger movement detection, and vibration sensors for accident or tampering detection. These sensors are interfaced with a Raspberry Pi microcontroller, which performs edge-level data processing and communicates with a cloud-based dashboard for centralized monitoring. In case of any irregular activity—such as a sudden stop, excessive vibration, or deviation from the assigned route—the system triggers instant alerts via buzzers, SMS notifications, and cloud updates to transport authorities and emergency responders.

The architecture supports scalability, low power consumption, and modular integration with existing fleet management systems. It also enables remote diagnostics, data logging, and predictive analytics for maintenance and operational optimization. Experimental simulations demonstrate the system's ability to detect anomalies with high accuracy, minimize false positives, and maintain robust performance under varied environmental and traffic conditions.

This project contributes to the field of intelligent transportation by offering a cost-effective, responsive, and adaptable solution for enhancing public safety, improving operational transparency, and supporting smart city initiatives. Future enhancements may include facial recognition for passenger verification, voice-based alert systems, and AI-driven route optimization to further elevate the capabilities of urban mobility networks.

TABLE OF CONTENTS

LIST OF DIAGRAM

ACRONYMS

Acronym	Full Form
IoT	Internet of Things
GPS	Global Positioning System
GSM	Global System for Mobile Communication
RFID	Radio Frequency Identification
UART	Universal Asynchronous Receiver-Transmitter
I2C	Inter-Integrated Circuit
LCD	Liquid Crystal Display
Wi-Fi	Wireless Fidelity
SMS	Short Message Service
RAM	Random Access Memory
ROM	Read-Only Memory
USB	Universal Serial Bus
IC	Integrated Circuit
PCB	Printed Circuit Board
LED	Light Emitting Diode
DC	Direct Current
ZB	ZigBee
VAS	Vehicle Alert System
AVAS	Accident Vehicle Alert System
ZAVAS	ZigBee Based Accident Vehicle Alert System
ZAS	ZigBee Alert System
ZARA	ZigBee based Accident Response Alert
NAVAS	Nearby Accident Alert System
NZAS	Nearby ZigBee Alert System

CHAPTER-1

INTRODUCTION

1.1 Overview

In recent years, public transportation has become a fundamental part of modern life, playing a vital role in connecting people and communities. However, with the increasing demand for safe and efficient transport, monitoring the movement, security, and condition of buses has become a major concern. Traditional bus tracking and management systems rely on manual operations or outdated technologies, which are often inefficient and fail to provide real-time data.

With the emergence of the Internet of Things (IoT), it is now possible to interconnect physical devices such as sensors, GPS modules, and cameras with cloud-based systems. IoT technology enables real-time monitoring, automation, and intelligent decision-making in transportation systems.

The "Intelligent Bus Monitoring and Alert System based on IoT using Raspberry Pi" is designed to modernize public bus transportation by integrating multiple IoT components into a single intelligent framework. The system continuously tracks the bus using GPS, authenticates passengers through RFID, detects abnormal vibrations or accidents using sensors, and captures security footage using a Raspberry Pi camera. The data is processed and transmitted to the IoT cloud platform, where authorities and passengers can view live information and receive real-time alerts.

This project not only enhances safety and reliability but also provides transparency in bus operations, ensuring timely communication between the bus, control room, and passengers.

1.2 Background

Public transportation systems in most cities still depend on traditional methods for bus tracking and passenger management. Manual monitoring of buses results in errors, delayed responses, and inefficiencies in communication. In emergencies such as accidents or security threats, the lack of real-time alerts can lead to delayed rescue operations.

To overcome these issues, an automated system is required that can detect, report, and display information in real time. The integration of IoT technology with microcontrollers like Raspberry Pi provides an ideal solution. The Raspberry Pi acts as the central processing unit that collects data from multiple sensors, processes it, and uploads it to cloud platforms such as Firebase, ThingSpeak, or Blynk.

1.3 Need for the System

With the increasing population and growing dependence on public transportation, the need for a smart monitoring system has become essential.
Some of the major challenges faced by traditional bus systems include:

Lack of real-time information about bus location and arrival time.

Absence of passenger verification or attendance tracking.

No automated alerts during accidents or route deviations.

Poor communication between bus drivers, authorities, and passengers.

Difficulty in ensuring safety and security of passengers.

To address these problems, a smart IoT-based monitoring system is proposed that provides live updates, automatic alerts, and efficient data logging.

- The proposed system ensures:
- Real-time visibility of bus movement.
- Instant notifications during accidents or security events.
- Passenger authentication for safety and attendance.
- Centralized cloud monitoring for authorities.

1.4 Objectives of the Project

The primary objective of this project is to design and implement an IoT-based intelligent bus monitoring and alert system that ensures passenger safety and efficient fleet management.

Specific Objectives:

To track the **real-time location** of the bus using a GPS module.

To provide **passenger authentication** through RFID technology.

To monitor the **bus interior and surroundings** using a Raspberry Pi camera.

To detect **accidents or abnormal vibrations** using sensors.

To send **automatic notifications** through GSM or IoT cloud services.

To provide **live data visualization** on mobile or web platforms.

1.5 Scope of the Project

This project is designed for public and institutional transportation systems, including city buses, school buses, and private fleets. It ensures real-time tracking, passenger safety, and automated alerts in a single system.

The proposed model can be extended to:

School Bus Tracking Systems – Parents can receive alerts when students board or leave the bus.

Public Transport Systems – Commuters can track buses and receive delay notifications.

Company Fleet Management – For monitoring official vehicles and employee transportation.

CHAPTER-2

Problem Statement

Public transportation plays a vital role in daily commuting, but most conventional bus systems lack real-time monitoring, security, and communication mechanisms. As a result, passengers, transport authorities, and family members face several challenges related to safety, location tracking, and emergency response.

In existing bus systems, there is no effective way to track the exact location of a bus, no automated alert mechanism in case of accidents, and no proper monitoring of passenger entry or exit. When accidents or route deviations occur, information rarely reaches authorities or passengers on time, leading to delayed assistance and reduced safety.

Furthermore, manual attendance or ticketing systems are inefficient, time-consuming, and prone to errors. Lack of intelligent surveillance also increases the risk of theft, unauthorized entry, and other safety issues.

Hence, there is an urgent need for an automated, IoT-based intelligent bus monitoring system that can:

1. Track the real-time location of buses using GPS.
2. Detect accidents or abnormal conditions using sensors.
3. Send instant alerts and notifications to authorities and passengers through IoT or GSM networks.
4. Use RFID technology for secure passenger identification or attendance.
5. Employ a camera system for onboard surveillance and evidence collection.

By integrating IoT with Raspberry Pi, GPS, RFID, and camera modules, the system can provide real-time monitoring, enhanced passenger security, and efficient fleet management.

This system aims to make public transport smarter, safer, and more reliable by combining hardware sensors, communication modules, and cloud-based data services into one intelligent platform.

CHAPTER-3

Literature Review

3.1 GPS and GSM-Based Vehicle Tracking System (2017 – IEEE)

This paper presented a vehicle tracking system that used a **GPS receiver** to obtain real-time location and a **GSM module** to transmit coordinates via SMS to a control center. The data could be visualized using Google Maps.

Advantages: Reliable and cost-effective for basic tracking.
Limitation: Lacked automation, cloud integration, and accident detection; no IoT data visualization.

3.2 IoT-Based Bus Location Tracking and Passenger Information System (2018 – IJETT)

The authors implemented a system where the **bus location was updated to a cloud server**, and passengers could check arrival times through a web app.

Advantages: Improved passenger convenience.
Limitation: System depended on network availability and did not include **security or alert features**.

3. 3 Smart Accident Detection and Notification System using IoT (2019 – IJAREEIE)

This project used an accelerometer sensor and GPS module to detect vehicle collisions and automatically send alert messages with the location to emergency contacts using GSM.

Advantages: Automatic emergency communication.
Limitation: Standalone system, no cloud data storage, and no passenger verification module.

3.4 Real-Time School Bus Tracking and Monitoring System using RFID and GPS (2020 – IRJET)

A system designed to ensure student safety by using RFID tags for each student and GPS tracking for the bus. Parents were notified when their child boarded or exited the bus.
Advantages: Enhanced parental awareness and student tracking.
Limitation: Used **Arduino**, limiting multitasking and camera integration.

3.5 IoT-Based Transportation Safety System using Raspberry Pi (2021 – IJRASET)

This research utilized Raspberry Pi, GPS, GSM, and sensors for accident detection and vehicle monitoring. The data was uploaded to a ThingSpeak cloud for visualization.

Advantages: Internet-enabled, real-time tracking with data logging.
Limitation: No passenger identification or camera support.

3.6 Smart Bus Monitoring System using IoT and RFID (2022 – Springer)

This paper proposed an IoT-based smart bus system that combined RFID authentication with GPS tracking to maintain passenger logs and location updates.

Advantages: Improved passenger safety and route management.
Limitation: Lacked multimedia evidence collection (camera integration) and real-time alert system.

3.7 Intelligent Vehicle Accident Detection and Notification System using Raspberry Pi (2022 – IEEE Access)

The system used Raspberry Pi 4, ADXL335 accelerometer, and Pi Camera to detect accidents and capture evidence. Data was uploaded to a Firebase database and sent via email/SMS.

Advantages: Advanced Raspberry Pi processing and multimedia support.
Limitation: Focused only on accidents, not full bus monitoring.

CHAPTER-4

INTERNET OF THINGS (IoT)

4.1 Introduction to IoT

The **Internet of Things (IoT)** is a network of interconnected physical devices that communicate and exchange data through the internet without direct human intervention. These devices, often equipped with sensors, actuators, and communication modules, can collect, process, and transmit data to cloud platforms for analysis and control. IoT enables real-time monitoring, automation, and intelligent decision-making across various fields such as smart homes, healthcare, agriculture, transportation, and industry.

In the **Intelligent Bus Monitoring and Alert System**, IoT plays a key role by connecting sensors like GPS, RFID, vibration sensors, and the Raspberry Pi controller to a centralized cloud platform. This allows the continuous monitoring of bus location, passenger information, and safety conditions in real time.

4.2 Components of IoT System

An IoT-based system consists of the following main components:

1. Sensors and Devices

These are the data-generating units in an IoT system. Sensors detect changes in the physical environment such as temperature, motion, vibration, or location.

> **Examples in this project:** GPS module (for location), RFID (for passenger ID), and Vibration sensor (for accident detection).

2. Connectivity / Network

The communication between devices and servers is achieved using wired or wireless networks. Common technologies include Wi-Fi, GSM, Bluetooth, and LoRa.

> **In this project:** The Raspberry Pi connects to the internet via **Wi-Fi**, and the **GSM module** is used for backup communication in case of network failure.

3. Data Processing / Edge Device

This stage involves the processing of collected data. The Raspberry Pi acts as the **edge computing device** that filters, analyzes, and prepares data before sending it to the cloud.

> **Example**: Detecting abnormal vibration values before sending an accident alert.

4. Cloud Platform

The processed data is stored and analyzed on cloud platforms such as **Blynk, ThingSpeak, or Firebase**, which allow visualization, notification, and decision-making.

> **Example**: The bus location and passenger information are uploaded to the cloud and displayed in real time.

5. User Interface (Application Layer)

The final output is shown to users through dashboards, mobile applications, or web interfaces.

> **Example**: A bus tracking app or web page showing live location, speed, and alert notifications.

4.3 Working of IoT in the Project

In the **Intelligent Bus Monitoring and Alert System**, IoT technology connects all hardware and software components to provide real-time monitoring and alert features.

1. The **Raspberry Pi** collects data from sensors — GPS (location), RFID (passenger identification), Vibration (accident detection), and Camera (visual monitoring).
2. The collected data is processed locally on the Raspberry Pi to identify conditions such as accidents, route deviations, or unauthorized access.
3. The processed information is then transmitted to **cloud servers** via Wi-Fi or GSM network.
4. On the cloud platform, the data is stored, analyzed, and displayed on the **mobile or web application**.
5. If any emergency is detected, **alerts are sent instantly** to the driver, control center, or passengers through notifications, buzzer alarms, and SMS messages.

This enables efficient bus tracking, passenger safety, and real-time management using IoT connectivity.

4.4 Advantages of IoT in the Project

Real-Time Monitoring: Enables continuous tracking of bus location and passenger activity.

Automation: Reduces the need for human intervention by automatically sending alerts during accidents or emergencies.

Improved Safety: Detects vibration and impact levels to identify accidents and instantly notify authorities.

Data Storage & Analysis: Cloud platforms store data for future analysis, helping improve transport management.

Remote Access: Bus information can be accessed anywhere through mobile or web applications.

4.5 IoT Platforms Used

Blynk: Provides a mobile interface for viewing real-time bus status and notifications.

ThingSpeak: Used for data visualization and analysis through graphs and charts.

Firebase: Provides a reliable cloud database for storing real-time sensor data and enabling instant communication.

4.6 Architecture of IoT System

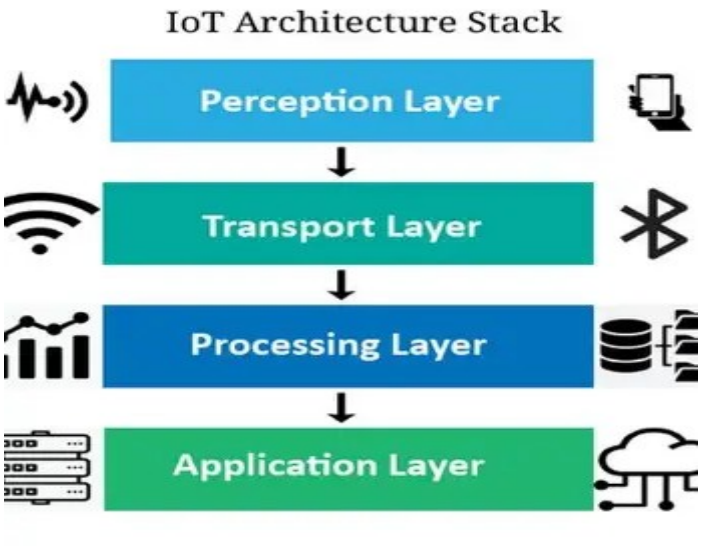


Fig no :- Architecture of IoT System

An IoT system consists of multiple layers that work together to collect, transmit, process, and display data. The architecture of the IoT system in this project is divided into **five main layers**:

`1. Perception Layer (Sensors & Devices)

This layer includes the hardware components that sense or collect data from the physical environment.

GPS Module: Captures real-time bus location (latitude, longitude, speed).

RFID Module: Detects and records passengers using RFID tags or cards.

Vibration Sensor: Detects sudden impacts or accidents.

Camera Module: Captures live images for monitoring.

2. Network Layer (Communication)

This layer transmits the collected data to the processing unit or the cloud.Communication technologies include **Wi-Fi, GSM, and Serial Communication (UART/SPI/I2C)**.

The Raspberry Pi acts as a gateway, sending sensor data to the **IoT cloud** for analysis.

3. Processing Layer (Edge Computing)

The **Raspberry Pi** performs local data processing to filter and analyze sensor inputs.
For example, if vibration values exceed a certain limit, the system detects an accident and immediately sends an alert.

4. Cloud Layer (Storage & Analytics)

Cloud platforms like **ThingSpeak**, **Blynk**, or **Firebase** store the data for visualization and historical analysis.
The data can be represented graphically to monitor trends such as bus routes, accident records, and passenger attendance.

5. Application Layer (User Interface)

This layer provides end users — such as passengers, drivers, or administrators — with access to real-time data.
The interface can be a **mobile application**, **web dashboard**, or **notification system** that displays:

- Live bus location
- Passenger status
- Alerts in case of emergencies

4.7 IoT Communication Protocols Used

IoT systems rely on specific communication protocols to transmit data efficiently and securely.
The major protocols used in this project are:

Wi-Fi: For real-time cloud connectivity and data transfer.

MQTT (Message Queuing Telemetry Transport): Used by platforms like ThingSpeak for lightweight data communication.

HTTP/HTTPS: For sending or receiving REST API requests from IoT platforms.

UART/SPI/I2C: For serial communication between Raspberry Pi and sensors.

GSM (SMS Communication): For emergency alerts when Wi-Fi is not available.

Protocol	Purpose	Advantages	Use in Project
Wi-Fi	Wireless internet connection	Fast data transfer and easy setup	Sends live bus data to cloud platforms
GSM	Cellular communication (SMS/GPRS)	Works even without Wi-Fi	Sends SMS alerts during accidents
UART	Serial communication	Simple and reliable	Used between Raspberry Pi and GPS/GSM modules
SPI	Communication between controller and sensors	Fast data exchange	Used for RFID module
I2C	Communication using two wires	Connects multiple sensors easily	Used for vibration and other sensors

CHAPTER-5

COMPARATIVE STUDY OF IoT AND EMBEDDED SYSTEMS

5.1 Introduction

In modern electronics and automation, both Embedded Systems and the Internet of Things (IoT) play crucial roles. While embedded systems are designed for dedicated control tasks within devices, IoT systems extend these capabilities by enabling connectivity, remote access, and data analytics. This chapter presents a detailed comparison between these two technologies, highlighting their architecture, functionality, applications, and integration in real-world systems.

5.2 Definition and Scope

- **Embedded System**: A microcontroller-based computing system designed to perform a specific task. It operates independently or within a larger mechanical/electrical system. Examples include washing machines, microwave ovens, and vehicle ECUs.
- **Internet of Things (IoT)**: A network of interconnected devices embedded with sensors, software, and communication modules that collect and exchange data over the internet or local networks. Examples include smart homes, wearable health monitors, and fleet tracking systems.

5.3 Technical Comparison

Feature	Embedded System	IoT System
Purpose	Dedicated task execution	Remote monitoring and control
Connectivity	Local or standalone	Internet/cloud-enabled
Hardware	Microcontroller, sensors, actuators	Embedded system + communication module
Software	Firmware, real-time OS	Firmware + cloud APIs + mobile/web apps
Communication Protocols	Serial, SPI, I2C	MQTT, HTTP, CoAP, LoRaWAN
Data Handling	Local processing	Cloud-based storage and analytics
Security	Basic authentication	Encryption, identity management

5.4 Real-World Applications

- **Embedded Systems**

    1. Automotive: Engine control units, airbag systems
    2. Home Appliances: Washing machines, microwave ovens
    3. Industrial Automation: PLCs, robotic arms
    4. Medical Devices: Pacemakers, infusion pumps

- **IoT Systems**

    1. Smart Agriculture: Soil moisture sensors, automated irrigation
    2. Smart Cities: Traffic monitoring, waste management
    3. Healthcare: Remote patient monitoring, wearable devices
    4. Transportation: Fleet tracking, predictive maintenance

5.5 Integration in Intelligent Bus Monitoring System

In the Intelligent Bus Monitoring and Alert System, embedded systems handle sensor data acquisition and local control (e.g., gas detection via MQ-6 sensor). IoT components, such as LoRa modules, enable long-range communication between buses and the control room. This integration allows real-time alerts, remote monitoring, and scalable deployment across multiple vehicles.

5.6 Advantages and Limitations

- **Embedded Systems**
  Advantages:

    1. Low cost and power consumption
    2. Real-time performance
    3. Simple architecture

  Limitations:

    1. No remote access
    2. Limited scalability
    3. Difficult to update remotely

- **IoT Systems**
  Advantages:

    1. Remote monitoring and control
    2. Scalable and flexible
    3. Cloud integration and analytics

  Limitations:

  Higher power consumption

CHAPTER-6

HARDWARE COMPONENTS

6.1 Introduction

The Intelligent Bus Monitoring and Alert System is composed of several hardware and software components that collectively perform data acquisition, processing, transmission, and alerting tasks. Each component plays a crucial role in sensing environmental parameters, communication, and system control.
This chapter provides a detailed description of all components used in the design and implementation of the system, including their specifications and working principles.

6.2 ESP8266 Wi-Fi Module

The **ESP8266** is a low-cost Wi-Fi microchip developed by Espressif Systems. It has become a popular choice for IoT-based applications due to its integrated TCP/IP protocol stack and microcontroller capabilities. It allows microcontrollers to connect to a Wi-Fi network and make simple TCP/IP connections using AT commands or custom firmware.

It features a 32-bit RISC CPU (Tensilica L106) operating at 80 MHz or 160 MHz. The module supports multiple communication protocols such as UART, SPI, and I2C, making it easy to interface with sensors and other devices. It comes with built-in flash memory for program storage and provides stable Wi-Fi connectivity for data transfer between the system and the server.

Technical Specifications:

- Operating Voltage: 3.3V DC
- Current Consumption: 70–200 mA
- Flash Memory: 4 MB
- CPU: 32-bit Tensilica L106
- Wi-Fi Standard: IEEE 802.11 b/g/n
- Communication Interfaces: UART, SPI, I2C
- Antenna: PCB Antenna (onboard)

Fig No :-ESP8266 Wi-Fi Module

Working Principle:
The ESP8266 connects to a Wi-Fi network and establishes communication with a remote server. The data collected from sensors is transmitted over the network using HTTP or MQTT protocols. It can also act as an access point, allowing other devices to connect directly to it. The module processes serial commands and sends data packets wirelessly to the designated host system.

6.3 Raspberry Pi

The **Raspberry Pi** is a compact, single-board computer capable of performing tasks similar to a desktop PC. It is equipped with a microprocessor, memory, and various communication ports. It operates under the Linux-based Raspberry Pi OS and supports programming languages such as Python, C, and Java.

It provides General Purpose Input/Output (GPIO) pins that allow connection to sensors, actuators, and other peripherals. It also includes interfaces for HDMI display, Ethernet networking, USB devices, and wireless connectivity.

Technical Specifications:

- Processor: Broadcom BCM2837 ARM Cortex-A53, 1.2 GHz
- RAM: 1 GB (for Raspberry Pi 3 Model B)
- Operating Voltage: 5V DC
- GPIO Pins: 40
- Communication: UART, I2C, SPI
- Storage: MicroSD Card
- Operating System: Raspberry Pi OS (Linux)
- Connectivity: Wi-Fi, Bluetooth, Ethernet

Working Principle:
The Raspberry Pi serves as the main control unit that processes and manages data obtained from connected sensors and devices. It reads sensor data through GPIO pins or serial interfaces, performs computations, and executes logic as per the program code. It can display data locally on a connected screen or transmit it to cloud servers via Wi-Fi. It also supports camera interfacing and multimedia handling

Pin Configuration:

Pin No	Pin Name	Description
1	3.3V	Power Supply
2	5V	Power Supply
6	GND	Ground
3	GPIO2	I2C SDA
5	GPIO3	I2C SCL
8	GPIO14	UART TX
10	GPIO15	UART RX
11	GPIO17	General I/O
13	GPIO27	General I/O
19	GPIO10	SPI MOSI
21	GPIO9	SPI MISO
23	GPIO11	SPI SCLK

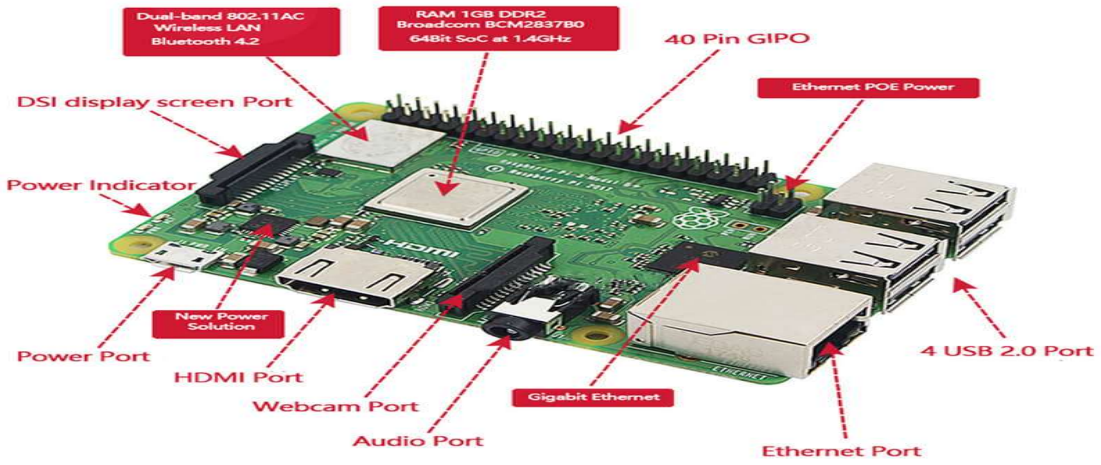


Fig No :− Raspberry Pi

6.4 Raspberry Pi Camera Module

The **Raspberry Pi Camera Module** is a high-resolution camera designed specifically for Raspberry Pi boards. It connects through the CSI (Camera Serial Interface) port and can capture still images and video. It uses the OmniVision sensor for high-quality imaging and supports real-time video streaming.

Technical Specifications:

1. Resolution: 8 MP or 12MP
2. Image Sensor: OmniVision OV5647 or IMX219
3. Interface: CSI (Camera Serial Interface)
4. Power Supply: 3.3V DC
5. Frame Rates: 30 fps for 1080p, 60 fps for 720p
6. Lens: Fixed Focus

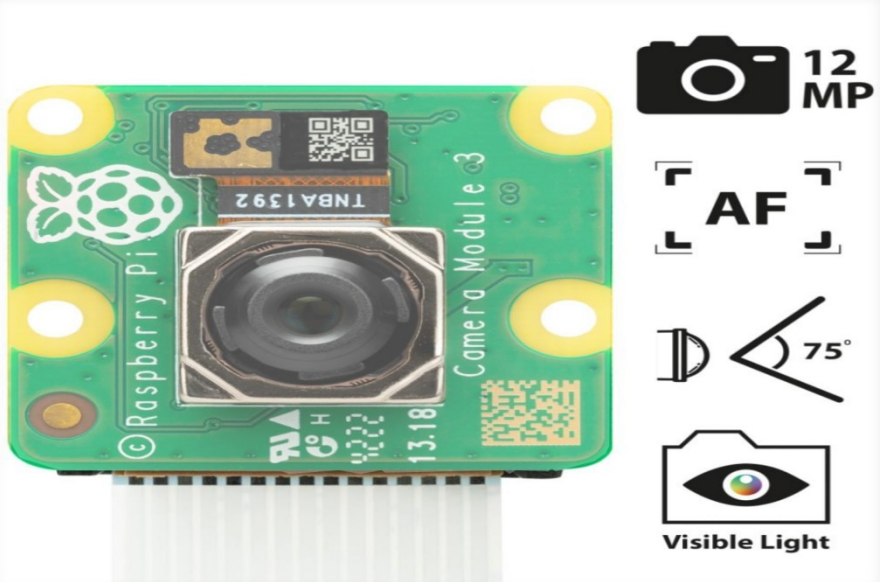


Fig No :− Raspberry Pi Camera Module

Working Principle:
The camera module captures light through the lens, which is converted into electronic signals by the image sensor. These signals are then processed into digital image data by the onboard processor. The Raspberry Pi controls the camera through the CSI interface and can display or store the video stream locally or on a cloud server.

6.5 GPS Module (Neo-6M)

The **GPS module** (Global Positioning System) is used to determine the geographical location of the system. It operates by receiving signals from multiple GPS satellites and calculating the position using triangulation.

Technical Specifications:

1. Model: Neo-6M
2. Operating Voltage: 3.3V to 5V DC
3. Communication Interface: UART
4. Data Format: NMEA 0183
5. Update Rate: 1 Hz (configurable up to 10 Hz)
6. Position Accuracy: ±2.5 meters
7. Built-in EEPROM and Antenna

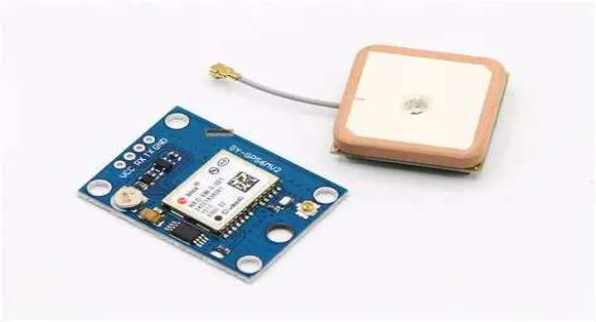


Fig No :−GPS Module

Working Principle:
The GPS module continuously receives signals from at least four satellites. It measures the time it takes for each satellite's signal to reach the receiver and uses this data to calculate latitude, longitude, altitude, and speed. The data is output in NMEA sentence format through serial communication, which can be read by a microcontroller or Raspberry Pi.

6.6 DHT Sensor (DHT11 / DHT22)

The **DHT sensor** is used to measure both temperature and humidity levels in the surrounding environment. It provides a calibrated digital output, making it simple to interface with microcontrollers.

Technical Specifications:

- Operating Voltage: 3.3V–5V DC
- Temperature Range: 0°C–50°C (DHT11), -40°C–80°C (DHT22)
- Humidity Range: 20%–90% RH (DHT11), 0–100% RH (DHT22)
- Accuracy: ±2°C (Temp), ±5% RH (Humidity)
- Interface: Single Digital Line

Fig No :−DHT Sensor

Working Principle:
The sensor has two main components: a capacitive humidity sensor and a thermistor. The humidity sensor measures the electrical resistance variation with the change in humidity, while the thermistor measures the change in resistance with temperature. The internal microcontroller converts these analog signals into digital data and transmits it serially.

6.7 RFID Module (RC522)

The **RC522 RFID Module** is designed for contactless data communication using radio frequency identification technology. It operates at 13.56 MHz and supports reading and writing of RFID tags.

Technical Specifications:

- Operating Voltage: 3.3V DC
- Operating Frequency: 13.56 MHz
- Communication Interface: SPI
- Reading Distance: 2–5 cm
- Data Transfer Rate: Up to 424 kbps
- Supported Tags: MIFARE 1K, 4K, Ultralight

Fig No :−RFID Module

Working Principle:
The RFID reader generates a radio frequency field that powers the RFID tag when it enters the field range. The tag responds by sending its unique identification code back to the reader. The reader then communicates this information to the microcontroller using SPI communication for further processing.

Pin Configuration:

Pin	Name	Description
1	VCC	Power Supply
2	RST	Reset
3	GND	Ground
4	MISO	Master In Slave Out
5	MOSI	Master Out Slave In
6	SCK	Clock Signal
7	NSS	Chip Select
8	IRQ	Interrupt Output (Optional)

6.8 Infrared (IR) Sensor

The **Infrared Sensor** is an electronic device that senses infrared light emitted or reflected from an object. It is mainly used for detecting the presence or movement of objects.

The Infrared (IR) sensor is an electronic device that detects infrared radiation emitted by objects. It is used in this project to detect obstacles, passengers, or bus entry and exit actions. IR sensors work by emitting infrared light through an IR LED and detecting the reflected signal with a photodiode. When an object comes close to the sensor, the reflected IR light is received by the photodiode, indicating the presence of an obstacle or movement.

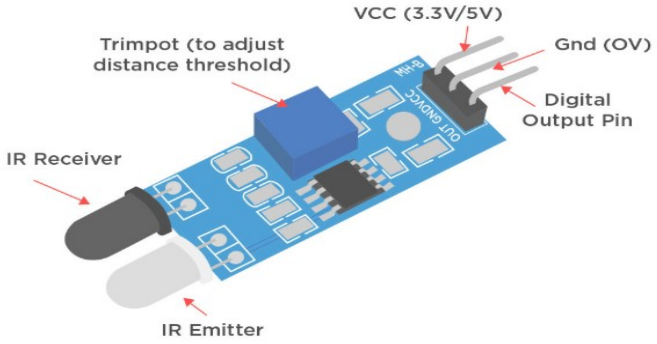


Fig No :⁻Infrared (IR) Sensor

In the **Intelligent Bus Monitoring and Alert System**, the IR sensor helps in automatic door control and passenger detection. When the sensor detects a person near the bus door, it sends a signal to the controller (Raspberry Pi or Arduino) to perform specific actions, such as counting the number of passengers or triggering a safety alert.

Working Principle:
The IR sensor consists of an infrared LED and a photodiode. The LED continuously emits infrared light. When this light hits an obstacle, it gets reflected and detected by the photodiode. The amount of reflected light determines the output state of the sensor. If reflection is detected, the sensor outputs a LOW signal; otherwise, it remains HIGH.

6.9 Buzzer

A **buzzer** is an audio signaling device that produces a buzzing sound when an electric voltage is applied. It is widely used in electronic systems for audible alerts and notifications.

A buzzer is an audio signaling device that converts electrical energy into sound. It is commonly used for alerts, notifications, and alarms in embedded systems. In this project, the buzzer is used to produce sound alerts during critical events, such as when a bus encounters an obstacle, accident detection occurs, or an emergency situation is identified.

The buzzer operates on the principle of electromagnetic induction. When current passes through the coil, it generates a magnetic field that moves a diaphragm, producing a sound or beep. Depending on the signal given from the controller, the buzzer can generate different sound patterns to indicate various system alerts.

Fig No :-Buzzer

Working Principle:
The buzzer operates on the piezoelectric effect. When an alternating voltage is applied to the piezoelectric material inside the buzzer, it vibrates rapidly, producing sound waves. The frequency of these vibrations determines the pitch of the sound emitted.

6.10 Power Supply Unit

The **Power Supply Unit (PSU)** is responsible for converting AC voltage from the mains to DC voltage suitable for electronic circuits. It provides regulated and filtered voltage to ensure the stable operation of all components.

The power supply is one of the most essential components in any electronic system, as it provides the necessary electrical energy to operate all the modules and sensors in the project. In the *Intelligent Bus Monitoring and Alert System*, the power supply ensures that devices such as the Raspberry Pi, sensors, GPS module, camera, and buzzer receive stable and regulated voltage for proper functioning.

Fig No :−Power Supply Unit

A typical power supply system converts the alternating current (AC) from the mains into direct current (DC), which is suitable for operating electronic components. The output voltage levels (usually 3.3V, 5V, or 12V) depend on the requirements of the connected components.

Working Principle

The power supply section consists of the following stages:

- **Step-Down Transformer** –
  Converts the high-voltage AC (230V) from the mains into a lower AC voltage (e.g., 12V AC).
- **Rectifier Circuit** –
  Converts the AC voltage into pulsating DC. A bridge rectifier made up of four diodes is commonly used for this purpose.
- **Filter Circuit** –
  Removes ripples from the rectified DC signal using capacitors, producing a smoother DC output.
- **Voltage Regulator** –
  Maintains a constant output voltage despite variations in input voltage or load.

CHAPTER-7

Working Principle

The **Intelligent Bus Monitoring and Alert System** is an Internet of Things (IoT)–based project designed to improve passenger safety, provide real-time bus tracking, and send alerts during emergency or unusual conditions. The system integrates multiple sensors and communication modules to monitor the bus environment and transmit data automatically to the server or control unit.

The working of the system is based on continuous sensing, data processing, and communication between the bus and the monitoring station.

7.1 Block Diagram

A **block diagram** is a simple representation of the system showing all major components and how they are connected. It helps in understanding the overall structure and functioning of the project without going into detailed circuits.

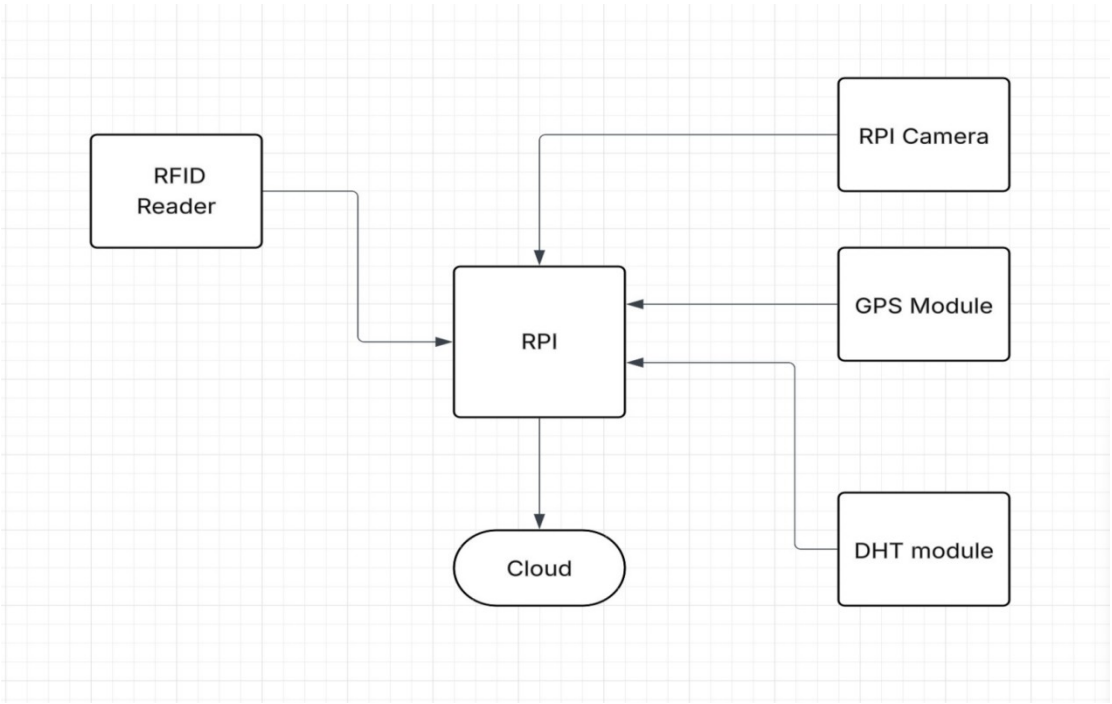


Fig No :-Block Diagram

7.2 Flow Diagram

A **flow diagram** represents the logical flow of operations in the system. It explains step-by-step how the system works from power-on to alert generation.

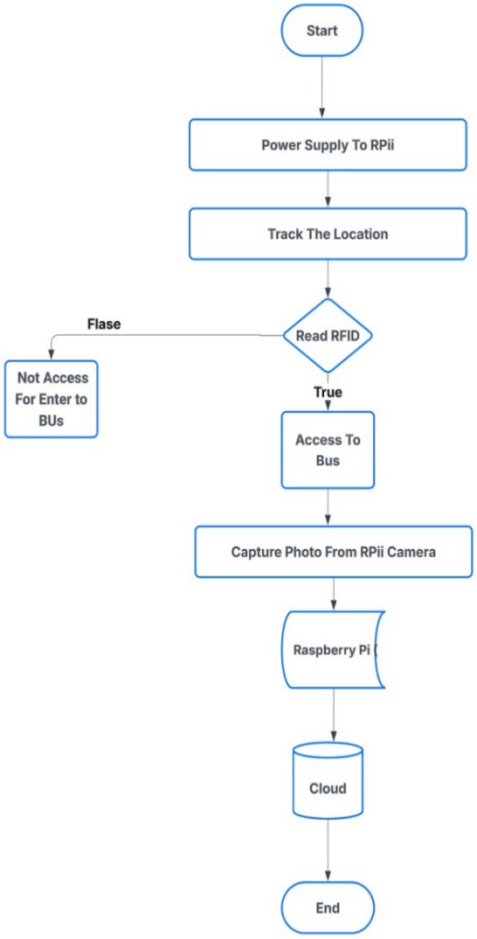


Fig No :-Flow Diagram

7.3 circuit diagram

The circuit diagram shows how the smart bus monitoring system uses both Raspberry Pi and NodeMCU to manage different tasks. Raspberry Pi handles passenger-related components like the RFID reader, IR sensor, buzzer, and camera for identification and security. NodeMCU is used for environmental monitoring and location tracking, connecting to the DHT11 sensor, GPS module, and an I2C LCD display. Each controller powers its own devices and communicates through GPIO pins. This setup ensures efficient data collection, alert generation, and real-time display, making the system reliable and scalable for smart transportation.

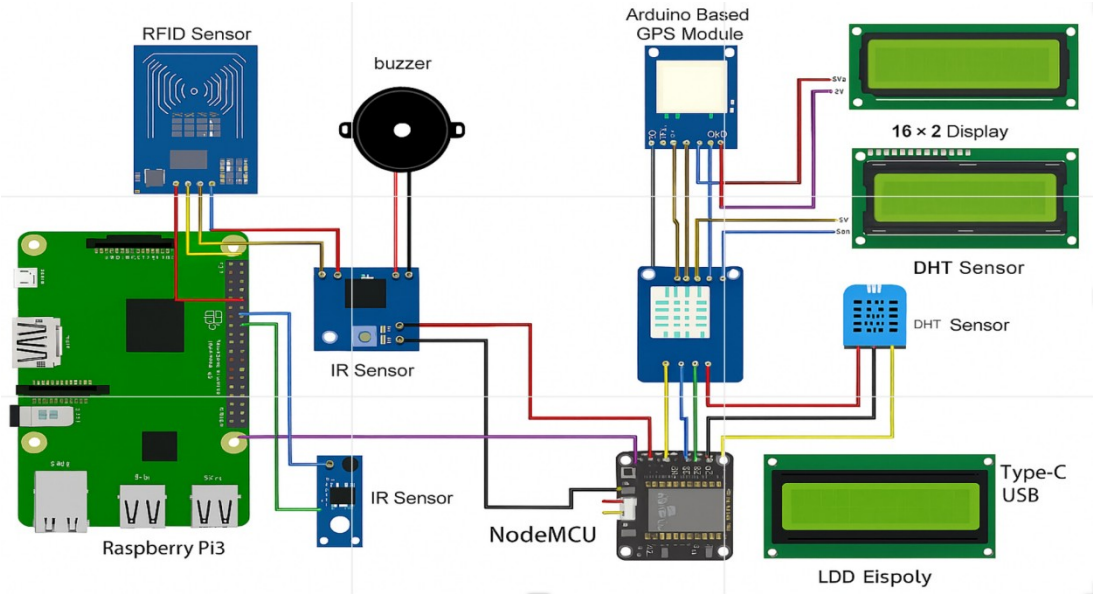


Fig No:- circuit diagram

CHAPTER-7

SOFTWARE

7.1 Raspberry Pi:

7.1.1 Introduction to Raspberry Pi :

The Raspberry Pi is a series of compact, affordable single-board computers (SBCs) developed in the United Kingdom by the Raspberry Pi Foundation, in collaboration with Broadcom. Originally designed to promote computer science education in schools, the Raspberry Pi has evolved into a versatile platform widely adopted across industries including IoT, robotics, home automation, and industrial control systems.

To meet growing global demand, the Foundation established a commercial arm known as Raspberry Pi Holdings, which oversees product development and distribution.

Raspberry Pi devices range from basic microcontrollers to fully functional computers capable of general-purpose computing. Key features include:

- HDMI video/audio output
- USB ports (including USB 3.0 on newer models)
- Wireless networking (Wi-Fi and Bluetooth)
- GPIO (General Purpose Input/Output) pins for hardware interfacing
- RAM options up to 16 GB
- Storage via microSD cards

Evolution and Milestones

- In 2015, Raspberry Pi became the best-selling British computer, surpassing the ZX Spectrum.
- As of March 2025, over 68 million units have been sold globally.
- The Raspberry Pi 4, launched in June 2019, introduced significant upgrades including faster processors, dual-monitor support, and up to 8 GB RAM.
- The Compute Module 4 (CM4) followed in October 2020, targeting embedded applications.
- The Raspberry Pi 400, released in November 2020, integrated a computer into a keyboard for ease of use.
- The Raspberry Pi Pico, launched in January 2021, marked the company's entry into the microcontroller market, powered by the custom RP2040 chip.
- The Raspberry Pi Zero 2 W, introduced in 2021, offered improved performance while maintaining a compact and low-cost form factor.

Market Challenges

The global chip shortage beginning in 2020, coupled with rising demand in 2021, impacted Raspberry Pi availability. In response, the company prioritized supply for business and industrial customers, ensuring continuity for mission-critical applications due

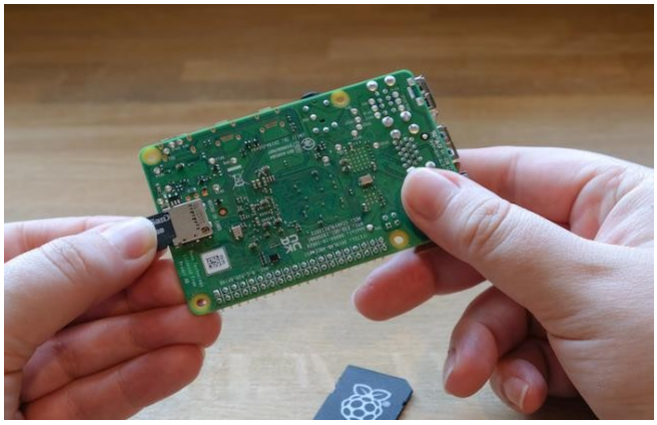
to limitations in the MBR. As with any other boot media, you'll see improved performance on SD cards with faster read and write speeds.

If you're unsure which SD card to buy, consider Raspberry Pi's official SD cards.

Because of a hardware limitation, the following devices will only boot from a boot partition of 256GB or less:

- Raspberry Pi Zero
- Raspberry Pi 1
- early Raspberry Pi 2 models with the BCM2836 SoC

Other operating systems have different requirements. Check the documentation for your operating system for capacity requirements.



Step 1: Installing Raspberry Pi

To use your Raspberry Pi, you'll need an operating system. By default, Raspberry Pis check for an operating system on any SD card inserted in the SD card slot.

Depending on your Raspberry Pi model, you can also boot an operating system from other storage devices, including USB drives, storage connected via a HAT, and network storage.

To install an operating system on a storage device for your Raspberry Pi, you'll need:

- a computer you can use to image the storage device into a boot device
- a way to plug your storage device into that computer.

Most Raspberry Pi users choose microSD cards as their boot device.

We recommend installing an operating system using Raspberry Pi Imager.

Step 2: Installing Raspberry Pi Imager

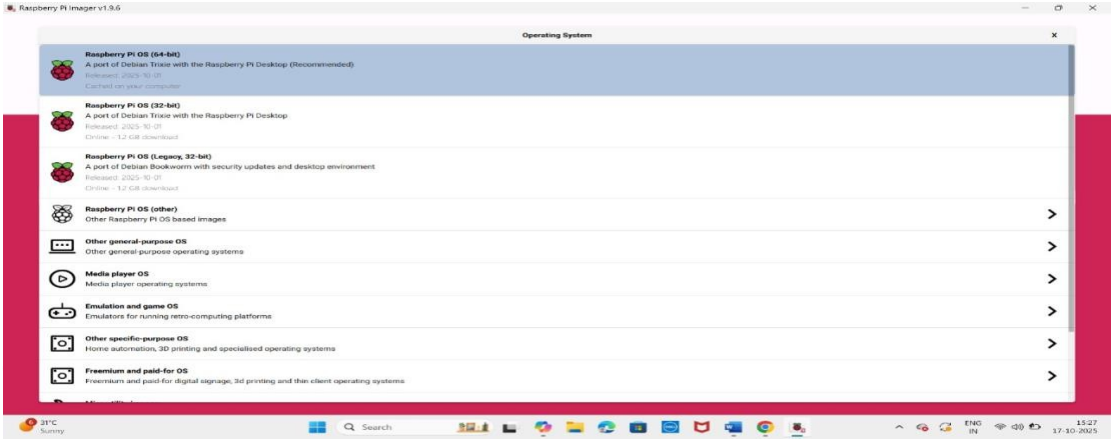
Raspberry Pi Imager is a tool that helps you download and write images on macOS, Windows, and Linux. Imager includes many popular operating system images for Raspberry Pi. Imager also supports loading images downloaded directly from Raspberry Pi or third-party vendors such as Ubuntu. You can use Imager to preconfigure credentials and remote access settings for your Raspberry Pi.

Imager supports images packaged in the .img format as well as container formats like .zip.

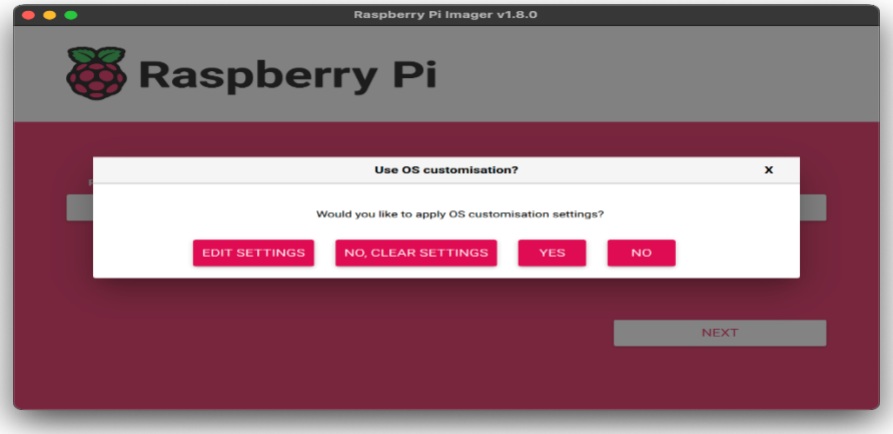
If you have no other computer to write an image to a boot device, you may be able to install an operating system directly on your Raspberry Pi from the internet.

Step 3: Click **Choose device** and select your Raspberry Pi model from the list.

Step 4: Click **Choose OS** and select an operating system to install. Imager always shows the recommended version of Raspberry Pi OS for your model at the top .



Step 5: Connect your preferred storage device to your computer. For example, plug a microSD card in using an external or built-in SD card reader. Then, click **Choose storage** and select your storage device.



In a popup, Imager will ask you to apply OS customisation. We strongly recommend configuring your Raspberry Pi via the OS customisation settings. Click the **Edit Settings** button to open OS customisation.

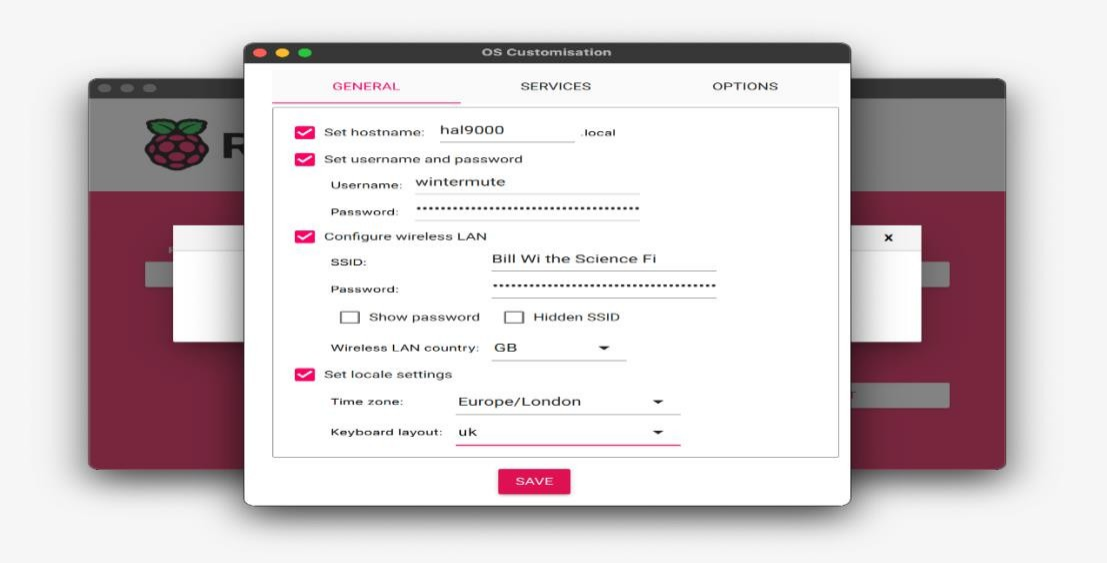
If you don't configure your Raspberry Pi via OS customisation settings, Raspberry Pi OS will ask you for the same information at first boot during the configuration wizard. You can click the **No** button to skip OS customisation.

Step 6: The hostname option defines the hostname your Raspberry Pi broadcasts to the network using mDNS. When you connect your Raspberry Pi to your network, other devices on the network can communicate with your computer using <hostname>.local or <hostname>.lan.

The username and password option defines the username and password of the admin user account on your Raspberry Pi.

The wireless LAN option allows you to enter an SSID (name) and password for your wireless network. If your network does not broadcast an SSID publicly, you should enable the "Hidden SSID" setting. By default, Imager uses the country you're currently in as the "Wireless LAN country". This setting controls the Wi-Fi broadcast frequencies used by your Raspberry Pi. Enter credentials for the wireless LAN option if you plan to run a headless Raspberry Pi.

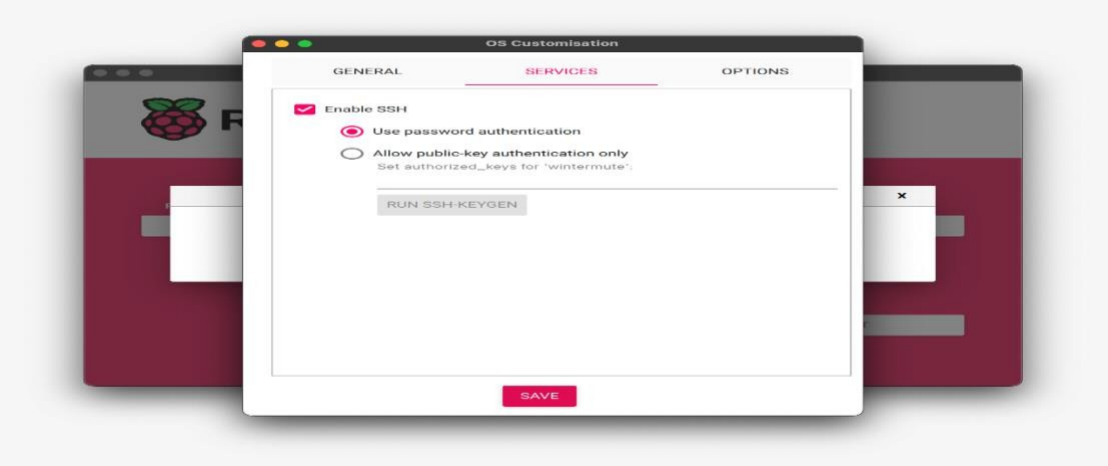
The locale settings option allows you to define the time zone and default keyboard layout for your Pi.

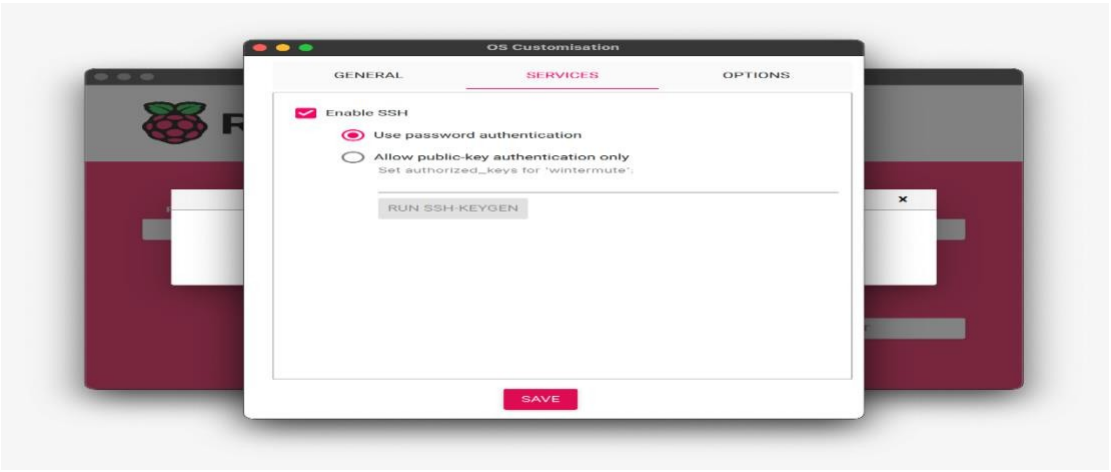


The Services tab includes settings to help you connect to your Raspberry Pi remotely.

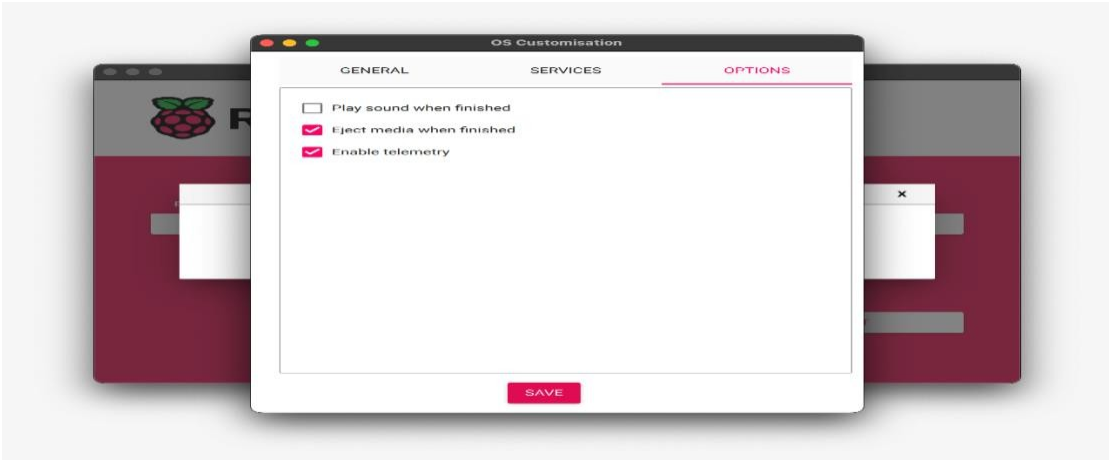
If you plan to use your Raspberry Pi remotely over your network, check the box next to Enable SSH. You should enable this option if you plan to run a headless Raspberry Pi.

- Choose the password authentication option to SSH into your Raspberry Pi over the network using the username and password you provided in the general tab of OS customisation.
- Choose Allow public-key authentication only to preconfigure your Raspberry Pi for password less public-key SSH authentication using a private key from the computer you're currently using. If already have an RSA key in your SSH configuration, Imager uses that public key. If you don't, you can click Run SSH-keygen to generate a public/private key pair. Imager will use the newly-generated public key.





OS customization also includes an Options menu that allows you to configure the behaviour of Imager during a write. These options allow you to play a noise when Imager finishes verifying an image, to automatically unmount storage media after verification, and to disable telemetry.

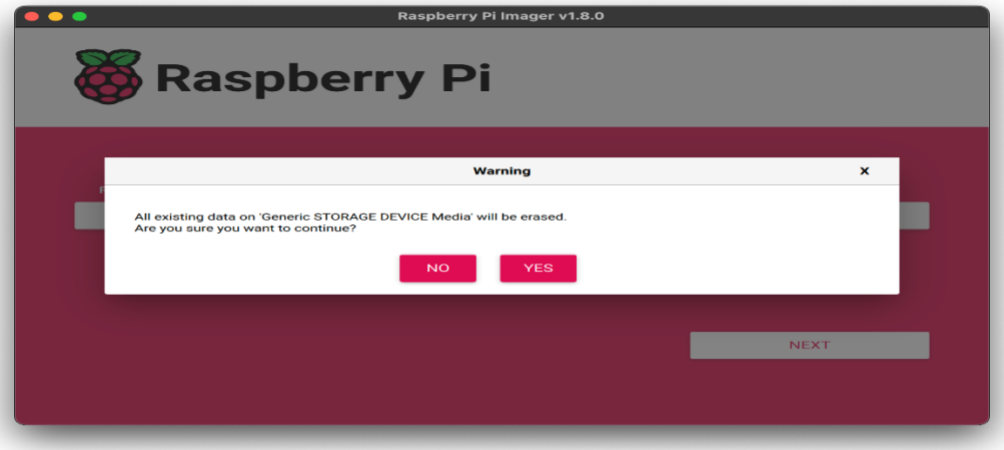


Step 7: WRITE

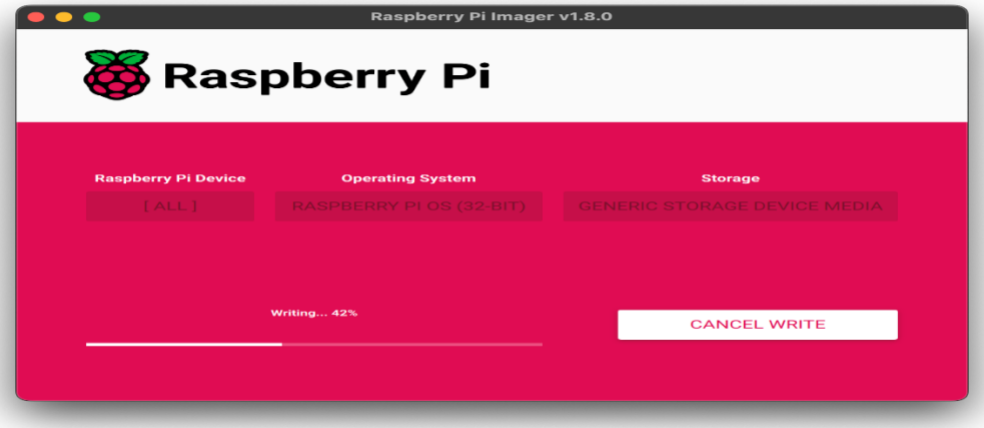
When you've finished entering OS customisation settings, click **Save** to save your customisation.

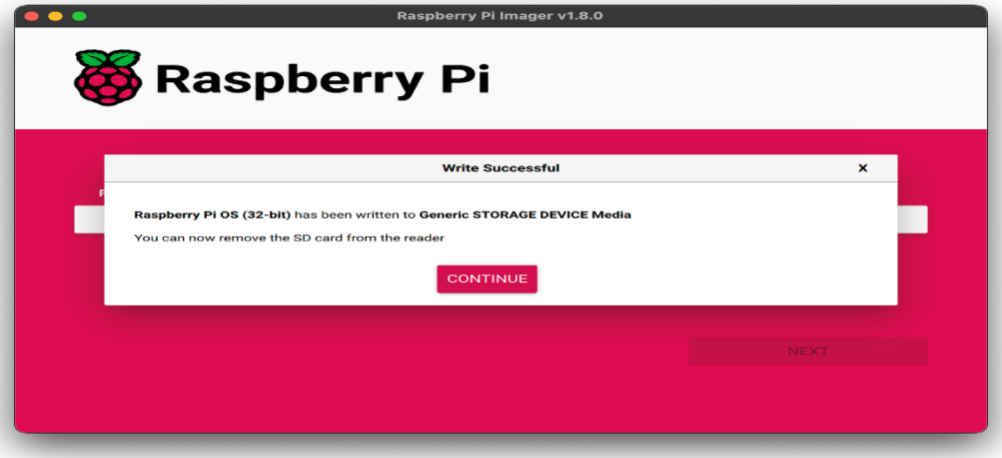
Then, click **Yes** to apply OS customisation settings when you write the image to the storage device.

Finally, respond **Yes** to the "Are you sure you want to continue?" popup to begin writing data to the storage device.



If you see an admin prompt asking for permissions to read and write to your storage medium, grant Imager the permissions to proceed.





Step 8: Your Raspberry Pi then boots up into a graphical desktop.

Step 9: open terminal in the VNC Viewer. Then run output by giving the input code.

Arduino IDE:

7.2 Introduction to Arduino IDE:

Arduino is a prototype platform (open-source) based on an easy-to-use hardware and software. It consists of a circuit board, which can be programmed (referred to as a microcontroller) and a ready-made software called Arduino IDE (Integrated Development Environment), which is used to write and upload the computer code to the physical board.

The key features are:

- Arduino boards are able to read analog or digital input signals from different sensors and turn it into an output such as activating a motor, turning LED on/off, connect to the cloud and many other actions.
- You can control your board functions by sending a set of instructions to the microcontroller on the board via Arduino IDE (referred to as uploading software).
- Unlike most previous programmable circuit boards, Arduino does not need an extra piece of hardware (called a programmer) in order to load a new code onto the board. You can simply use a USB cable.
- Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program.
- Finally, Arduino provides a standard form factor that breaks the functions of the micro-controller into a more accessible package.
- After learning about the main parts of the Arduino UNO board, we are ready to learn how to set up the Arduino IDE. Once we learn this, we will be ready to upload our program on the Arduino board.

Arduino data types:

Data types in C refers to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in the storage and how the bit pattern stored is interpreted.

The following table provides all the data types that you will use during

Arduino programming.

Void:

The void keyword is used only in function declarations. It indicates that the function is expected to return no information to the function from which it was called.

Example:

```
Void Loop ( )
{
// rest of the code
}
```

Boolean:

A Boolean holds one of two values, true or false. Each Boolean variable occupies one byte of memory.

Char: A data type that takes up one byte of memory that stores a character value. Character literals are written in single quotes like this: 'A' and for multiple characters, strings use double quotes: "ABC".

However, characters are stored as numbers. You can see the specific encoding in the ASCII chart. This means that it is possible to do arithmetic operations on characters, in which the ASCII value of the character is used. For example, 'A' + 1 has the value 66, since the ASCII value of the capital letter A is 65.

Unsigned char:

Unsigned char is an unsigned data type that occupies one byte of memory. The unsigned char data type encodes numbers from 0 to 255.

Byte:

A byte stores an 8-bit unsigned number, from 0 to 255.

int:

Integers are the primary data-type for number storage. **int** stores a 16-bit (2-byte) value. This yields a range of -32,768 to 32,767 (minimum value of -2^15 and a maximum value of (2^15) - 1).

The **int** size varies from board to board. On the Arduino Due, for example, an **int** stores a 32-bit (4-byte) value.

Unsigned int:

Unsigned ints (unsigned integers) are the same as int in the way that they store a 2 byte value. Instead of storing negative numbers, however, they only store positive values, yielding a useful range of 0 to 65,535 (2^16) - 1).

Word:

On the Uno and other ATMEGA based boards, a word stores a 16-bit unsigned number. On the Due and Zero, it stores a 32-bit unsigned number.

Long:

Long variables are extended size variables for number storage, and store 32 bits (4 bytes), from 2,147,483,648 to 2,147,483,647.

Unsigned long: Unsigned long variables are extended size variables for number storage and store 32 bits (4 bytes). Unlike standard longs, unsigned longs will not store negative numbers, making their range from 0 to 4,294,967,295 (2^32 - 1).

Short:

A short is a 16-bit data-type. On all Arduinos (ATMega and ARM based), a short stores a 16-bit (2-byte) value. This yields a range of -32,768 to 32,767 (minimum value of -2^15 and a maximum value of (2^15) - 1).

Float:

Data type for floating-point number is a number that has a decimal point. Floating-point numbers are often used to approximate the analog and continuous values because they have greater resolution than integers.

Floating-point numbers can be as large as 3.4028235E+38 and as low as

3.4028235E+38. They are stored as 32 bits (4 bytes) of information.

Double:

On the Uno and other ATMEGA based boards, Double precision floating-point number occupies four bytes. That is, the double implementation is exactly the same as the float, with no gain in precision. On the Arduino Due, doubles have 8-byte (64 bit) precision.

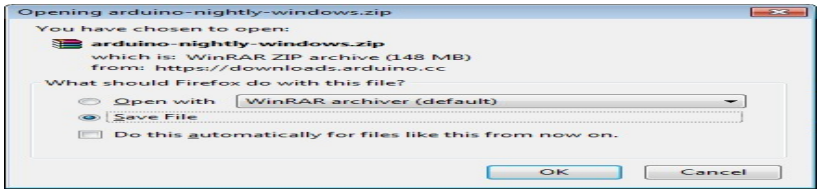
In this section, we will learn in easy steps, how to set up the Arduino IDE on our computer and prepare the board to receive the program via USB cable.

Step 1: First you must have your Arduino board (you can choose your favorite board) and a USB cable. In case you use Arduino UNO, Arduino Duemilanove, Nano, Arduino Mega2560, or Diecimila, you will need a standard USB cable (A plug to B plug), the kind you would connect to a USB printer as shown in the following image.

Figure 7.1: USB Cable

Step 2: Download Arduino IDE Software.

You can get different versions of Arduino IDE from the Download page on the Arduino Official website. You must select your software, which is compatible with your operating system (Windows, IOS, or Linux). After your file download is complete, unzip the file.



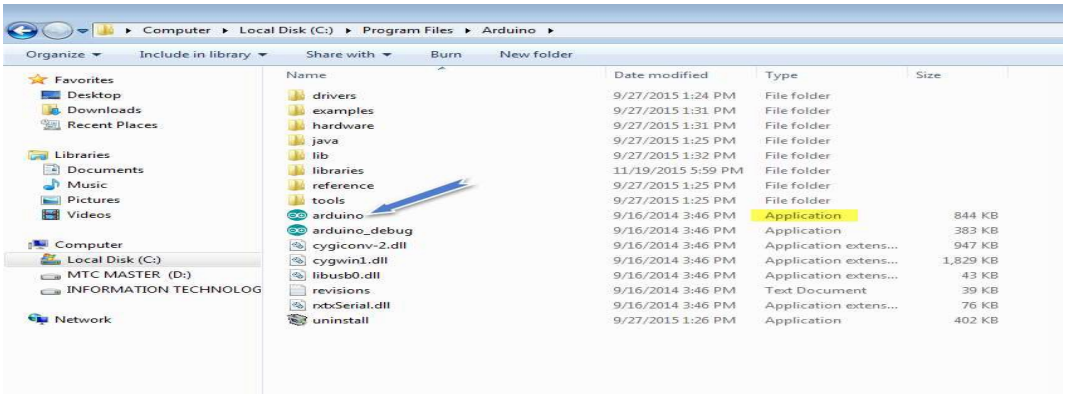
Step 3: Power up your board.

The Arduino Uno, Mega, Duemilanove and Arduino Nano automatically draw power from either, the USB connection to the computer or an external power supply. If you are using an Arduino Diecimila, you have to make sure that the board is configured to draw power from the USB

connection. The power source is selected with a jumper, a small piece of plastic that fits onto two of the three pins between the USB and power jacks. Check that it is on the two pins closest to the USB port. Connect the Arduino board to your computer using the USB cable. The green power LED (labeled PWR) should glow.

Step 4: Launch Arduino IDE.

After your Arduino IDE software is downloaded, you need to unzip the folder. Inside the folder, you can find the application icon with an infinity label (application.exe). Double click the icon to start the IDE.



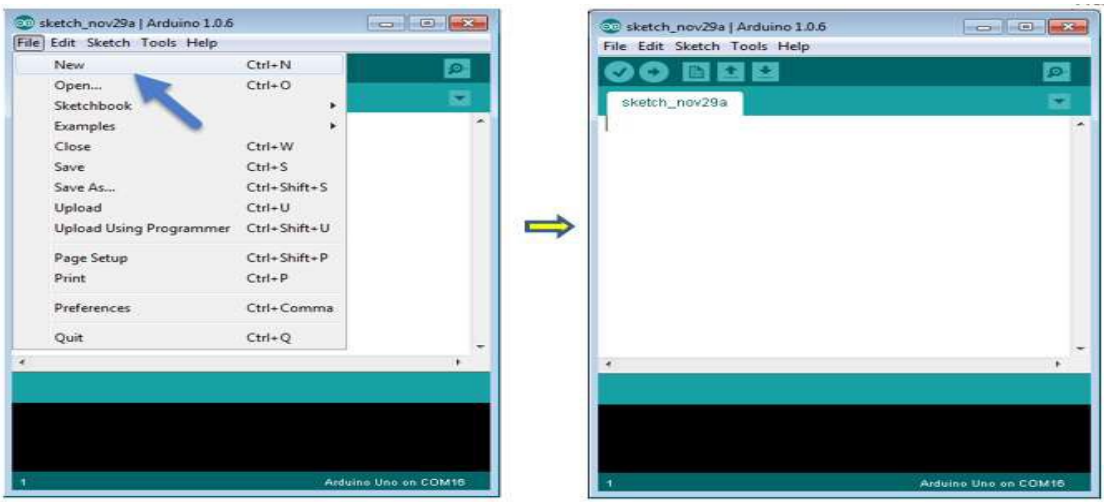
Step 5: Open your first project.

Once the software starts, you have two options:

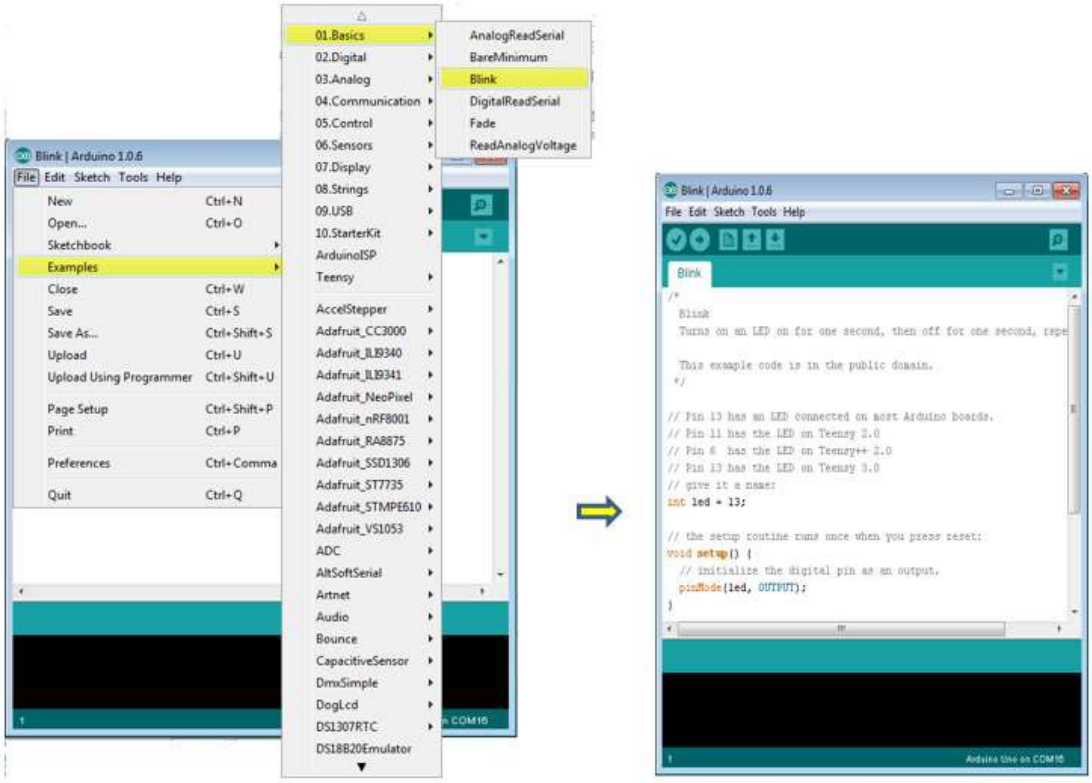
- Create a new project.

- Open an existing project example.

To create a new project, select File --> New. To open



To open an existing project example, select File -> Example -> Basics -> Blink.



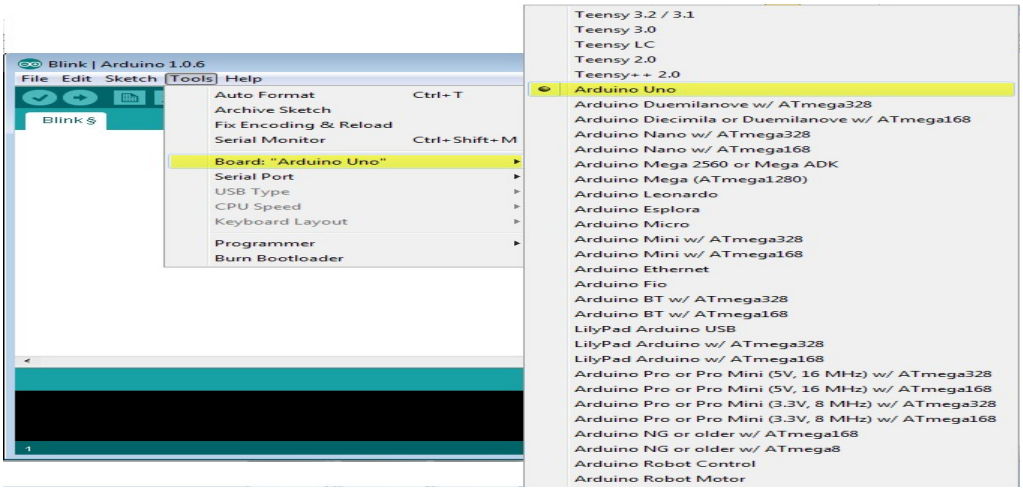
Here, we are selecting just one of the examples with the name **Blink**. It turns the LED on and off with some time delay. You can select any other example from the list.

Step 6: Select your Arduino board.

To avoid any error while uploading your program to the board, you must select the correct Arduino board

name, which matches with the board connected to your computer.

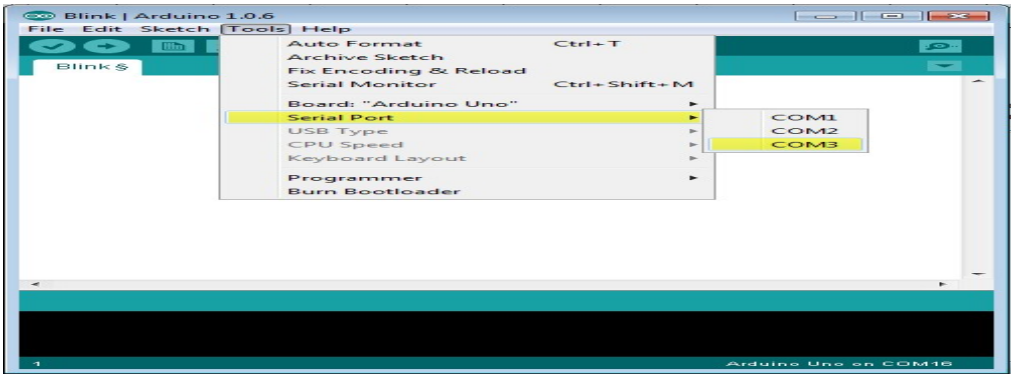
Go to Tools -> Board and select your board



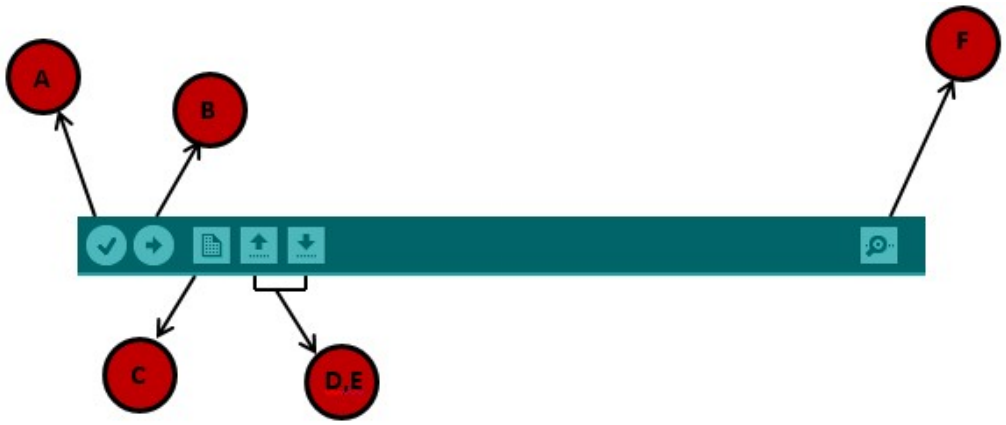
Here, we have selected Arduino Uno board according to our tutorial, but you must select the name matching the board that you are using

Step 7: Select your serial port:

Select the serial device of the Arduino board. Go to **Tools** ->**Serial Port** menu. This is likely to be COM3 or higher (COM1 and COM2 are usually reserved for hardware serial ports). To find out, you can disconnect your Arduino board and re-open the menu, the entry that disappears should be of the Arduino board. Reconnect the board and select that serial port.



Step 8: Upload the program to your board: Before explaining how we can upload our program to the board, we must demonstrate the function of each symbol appearing in the Arduino IDE toolbar.



A- Used to check if there is any compilation error.

B- Used to upload a program to the Arduino board.

C- Shortcut used to create a new sketch.

D- Used to directly open one of the example sketch.

E- Used to save your sketch.

F- Serial monitor used to receive serial data from the board and send the serial data to the board.

Now, simply click the "Upload" button in the environment. Wait a few seconds; you will see the RX and TX LEDs on the board, flashing. If the upload is successful, the message "Done uploading" will appear in the status bar.

Note: If you have an Arduino Mini, NG, or other board, you need to press the reset button physically on the board, immediately before clicking the upload button on the Arduino Software.

Arduino programming structure

In this chapter, we will study in depth, the Arduino program structure and we will learn more new terminologies used in the Arduino world. The Arduino software is open-source. The source code for the Java environment is released under the GPL and the C/C++ microcontroller libraries are under the LGPL.

Sketch: The first new terminology is the Arduino program called "**sketch**".

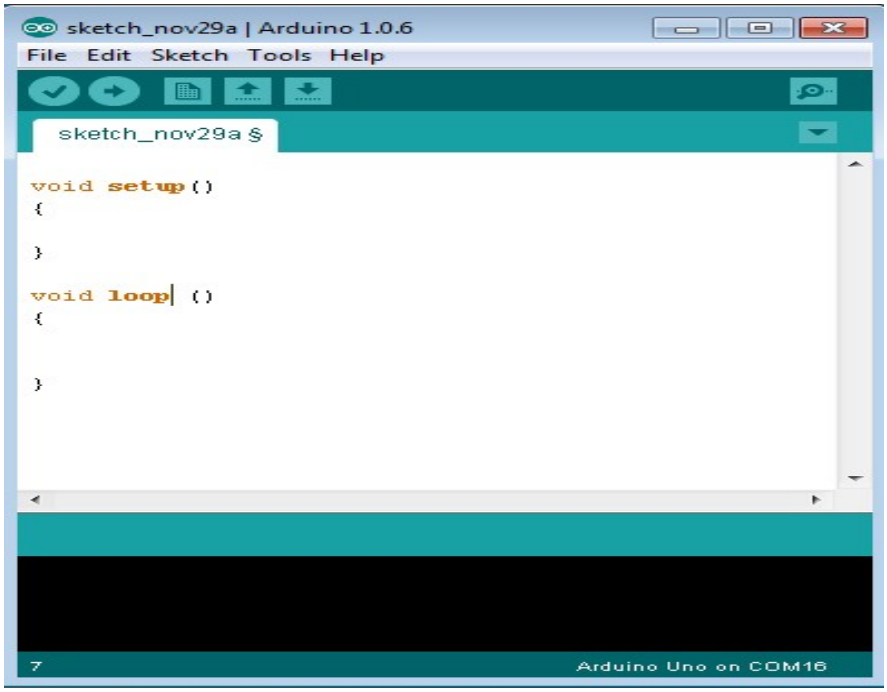
Structure

Arduino programs can be divided in three main parts: **Structure**, **Values**

(variables and constants), and **Functions**. In this tutorial, we will learn about the Arduino software program, step by step, and how we can write the program without any syntax or compilation error.

Let us start with the **Structure**. Software structure consist of two main functions:

- Setup( ) function
- Loop( ) function



Void setup ()

{

}

PURPOSE:

The **setup()** function is called when a sketch starts. Use it to initialize the variables, pin modes, start using libraries, etc. The setup function will only run once, after each power up or reset of the Arduino board.

INPUT

OUTPUT

RETURN

```
Void Loop ()
{


}
```

CHAPTER-8

RESULT

CHAPTER -9

ADVANTAGES & APPLICATIONS

Advantages

- **Real-Time Monitoring:**
  The system continuously tracks the location of buses using GPS and provides live updates to passengers and administrators.
- **Enhanced Passenger Safety:**
  Accident detection and alert mechanisms ensure quick response in emergencies, reducing the risk of serious injuries.
- **Efficient Fleet Management:**
  Authorities can monitor bus routes, speeds, and stops remotely, improving scheduling and reducing fuel costs.
- **Instant Notifications:**
  The GSM and IoT cloud modules send automatic alerts and notifications to passengers, drivers, and control centers during abnormal situations.
- **Reduced Waiting Time:**
  Passengers can check the exact arrival time of buses, minimizing waiting periods at stops.
- **Data Storage and Analysis:**
  All data is stored in the cloud for future analysis, helping in improving transport services.
- **Cost-Effective and Scalable:**
  The system uses affordable IoT components and can easily be scaled to multiple buses and routes.
- **Improved Transparency:**
  Both management and passengers can access live data, ensuring accountability and reliability in public transport.

Applications

- **Public Transportation Systems:**
  Can be implemented in city bus networks for live tracking and passenger safety monitoring.
- **School and College Buses:**
  Ensures safety of students by sending alerts to parents and school authorities.
- **Private Bus Services:**
  Travel agencies can use it to manage their fleet efficiently and provide real-time updates to customers.
- **Government Fleet Management:**
  Useful for monitoring and maintaining vehicles used in public services or emergency response.

- **Tourism Buses:**
  Enhances travel experience by providing route information and timely alerts during the journey.
- **Logistics and Goods Transportation:**
  The same system concept can be applied for monitoring goods vehicles for security and delivery tracking.

CHAPTER-10

CONCLUSION AND FUTURE SCOPE

Conclusion

The **Intelligent Bus Monitoring and Alert System based on IoT** successfully demonstrates how modern technology can enhance the efficiency, safety, and reliability of public transportation. By integrating sensors, GPS, GSM, and cloud platforms, the system provides real-time tracking, accident detection, and instant notifications to both passengers and authorities.

The implementation of Raspberry Pi as the central controller ensures seamless data processing and communication with all connected modules. Through the IoT platform, users can access live data such as bus location, speed, and route status from anywhere, enabling better decision-making and improved service management.

Overall, the system offers a smart, cost-effective, and scalable solution to reduce waiting times, enhance safety, and promote the development of intelligent public transport infrastructure.

Future Scope

1. **Integration with AI and Machine Learning:**
   Future versions can use AI algorithms to predict bus arrival times, detect unusual driving patterns, and suggest route optimizations.
2. **Mobile Application Enhancement:**
   A dedicated mobile app can be developed to provide real-time bus tracking, seat availability, and fare payment options.
3. **Camera-Based Surveillance and Face Recognition:**
   Raspberry Pi Camera can be used for live video monitoring and automatic detection of unauthorized persons or suspicious activities.
4. **Voice Alerts and Driver Assistance:**
   The system can include voice-based alerts for drivers regarding speed limits, route changes, or obstacles.
5. **Cloud Data Analytics:**
   Historical data can be analyzed to identify peak hours, route efficiency, and vehicle performance trends.
6. **Integration with Smart City Infrastructure:**
   The system can be connected with city-wide IoT networks to form part of a comprehensive smart transport ecosystem.
7. **Enhanced Security Features:**
   Additional modules like alcohol sensors or fire detectors can be added to improve passenger safety further

References

1. Rajalakshmi, P. and Devi, S., "Real-Time Bus Monitoring and Passenger Information System using GSM and GPS," International Journal of Advanced Research in Computer Engineering & Technology (IJARCET), Vol. 3, Issue 2, 2014.
2. Gokulnath, C. and Ganesan, M., "Intelligent Public Transport System using IoT," International Journal of Innovative Research in Science, Engineering and Technology, Vol. 5, Issue 10, 2016.
3. Priyanka, P., "Smart Vehicle Tracking System using GPS and GSM Modem," International Journal of Computer Applications, Vol. 119, No. 2, 2015.
4. R. Ranjith and K. Vijayakumar, "IoT Based Smart Vehicle Monitoring System," International Journal of Engineering and Technology (IJET), Vol. 7, No. 4, 2018.
5. J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions," Future Generation Computer Systems, Vol. 29, Issue 7, 2013.
6. Raspberry Pi Foundation – Official Documentation, https://www.raspberrypi.org/documentation/
7. Blynk IoT Platform – Developer Documentation, https://docs.blynk.io
8. NEO-6M GPS Module Datasheet – u-blox Official Documentation, https://www.u-blox.com
9. SIM800L GSM Module AT Command Set – SIMCom Wireless Solutions, https://simcom.ee/documents/
10. ThingSpeak Cloud Platform – MATLAB IoT Analytics, https://thingspeak.com

CODE For Raspberry Pi :

For IR Sensor :

```
from gpiozero import InputDevice
from time import sleep
from gpiozero import LED

#import RPi.GPIO as GPIO
# Initialize the sensor as a digital input device on GPIO 27
sensor = InputDevice(27) #Pin 13
#GPIO.setup(18, GPIO.OUT, initial=GPIO.LOW)
buzzer = LED(22) #Pin 15, GPIO22

while True:
  if sensor.is_active:
    print("No obstacle detected")  # Prints when no obstacle is detected
#    GPIO.output(18, GPIO.LOW)
    buzzer.off()
#     buzzer.value = 1
#     sleep(2)
  else:
    print("Obstacle detected")     # Prints when an obstacle is detected
#    GPIO.output(18, GPIO.HIGH)
    buzzer.on()
    buzzer.value = 1
  sleep(1.0)
```

For RIFD :

//normal_read.py

```
 import RPi.GPIO as GPIO
from mfrc522 import SimpleMFRC522

rfid = SimpleMFRC522()

while True:
      id, text = rfid.read()
      print(id)
      print(text)
```

```
// for read.py

import RPi.GPIO as GPIO
from mfrc522 import SimpleMFRC522
import time

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT)

rfid= SimpleMFRC522()
channel = 17

def relay_on(pin):
   GPIO.output(pin,GPIO.LOW)

def relay_off(pin):
   GPIO.output(pin,GPIO.HIGH)

while True:
     id, text = rfid.read()
     print(id)

     if id == 180009633196:
       relay_on(channel)
       print(text+":Access granted")
       time.sleep(5)
       relay_off(channel)

     else:
       relay_off(channel)
       print("Not allowed...")

//for write.py

import RPi.GPIO as GPIO
from mfrc522 import SimpleMFRC522

rfid = SimpleMFRC522()

try:
       print("Hold tag near the module...")
       rfid.write("Circuit basics")
       print("Written")
finally:
       GPIO.cleanup()
```

CODE For Arduino IDE :

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <TinyGPS++.h>
#include <SoftwareSerial.h>

// -------------------- OLED Setup --------------------
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
-1);

// -------------------- DHT22 Setup --------------------
#define DHTPIN 14     // D5 on NodeMCU
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);

// -------------------- GPS Setup --------------------
// Connect GPS module:
//   GPS TX -> D7 (NodeMCU RX)
//   GPS RX -> D8 (NodeMCU TX)
static const int RXPin = D7, TXPin = D8;
static const uint32_t GPSBaud = 9600;

SoftwareSerial gpsSerial(RXPin, TXPin);
TinyGPSPlus gps;

// -------------------- Setup --------------------
void setup() {
  Serial.begin(115200);
  gpsSerial.begin(GPSBaud);
  dht.begin();

  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;);
  }
  delay(2000);
  display.clearDisplay();
  display.setTextColor(WHITE);
  Serial.println("System Initialized: DHT22 + GPS + OLED");
}
```

```
// -------------------- Main Loop --------------------
void loop() {
  // Update GPS data
  while (gpsSerial.available() > 0) {
    gps.encode(gpsSerial.read());
  }

  // Read DHT22
  float t = dht.readTemperature();
  float h = dht.readHumidity();

  // Clear OLED
  display.clearDisplay();

  // Display DHT22 Data
  display.setTextSize(1);
  display.setCursor(0, 0);
  if (isnan(t) || isnan(h)) {
    display.print("DHT Read Error");
  } else {
    display.print("Temp: ");
    display.print(t, 1);
    display.cp437(true);
    display.write(167);
    display.print("C  ");
    display.print("Hum: ");
    display.print(h, 1);
    display.print("%");
  }

  // Display GPS Data
  display.setTextSize(1);
  display.setCursor(0, 20);
  if (gps.location.isValid()) {
    display.print("Lat: ");
    display.print(gps.location.lat(), 4);
    display.setCursor(0, 30);
    display.print("Lng: ");
    display.print(gps.location.lng(), 4);
  } else {
    display.print("GPS: Searching...");
  }

  display.setCursor(0, 45);
  if (gps.satellites.isValid()) {
    display.print("Sats: ");
    display.print(gps.satellites.value());
  }
```

```
  display.display();
  delay(2000);
}
```