# IBM SMARTBRIDGE

INTELLIGENT CARGO MANAGEMENT SYSTEM USING IOT

GROUP NAME (VSSR)

P.RUPESH

B.VISHNU

S.V.PRADEEP

S.SIVA SANKAR

IOT (INTERNET OF THINGS)

DEF : The things around us are connected and communicating with each other.

“Everything that can be connected will be connected”

**Abstract:**

This paper presents an agent based intelligent cargo tracking system including an agent structure and system architecture which is based on the Internet of Things. The concept and methods of Cargo Information Modeling are introduced, together with the cargo information contents and structure are explained prior to the implementation of an intelligent cargo tracking system. In this system, a model that includes all information needed in logistics chain management is used in the tracking of a product lifecycle. In this paper, has been described.

**INTRODUCTION :**Today, **Internet application development demand is very high**. So **IoT is a major technology** by which we can produce various useful internet applications.

Basically, **IoT is a network in which all physical objects are connected to the internet** through network devices or routers and exchange data. IoT allows objects to be controlled remotely across existing network infrastructure. IoT is a very good and intelligent technique which reduces human effort as well as easy access to physical devices. This technique also has autonomous control feature by which any device can control without any human interaction.



The **above figure shows the connectivity of various devices of different fields with Internet and exchange data between them**. So above figure represent the connectivity of world through various existing technologies.

**“Things” in the IoT sense, is the mixture of hardware, software, data, and services**. “Things” can refer to a wide variety of devices such as DNA analysis devices for environmental monitoring, electric clamps in coastal waters, Arduino chips in home automation and many other. **These devices gather useful data with the help of various existing technologies and share that data between other devices.** Examples include Home Automation System which uses Wi-Fi or Bluetooth for exchange data between various devices of home.

**HISTORICAL BACKGROUND**

**In early 1982 the concept of the network of smart devices was discussed**, **with a modified Coke machine**. This coke machine is modified at “Carnegie Mellon University” and becoming the first Internet-connected appliance. This machine was able to report its inventory and whether newly loaded drinks were cold.

**In 1994 Reza Raji explained the idea of IoT as “small packets of data to a large set of nodes, so as to integrate and automate everything from home appliances to entire factories”**. After that many companies proposed various solutions like Microsoft’s at Work or Novell’s Nest. Bill Joy proposed Device to Device (D2D) communication as a part of his “Six Webs” frameworks at the World Economic Forum at Davos in 1999.

**The thought of Internet of Things first became popular in 1999**. British entrepreneur Kevin Ashton first used the term Internet of Things in 1999 while working at Auto-ID labs. **PRACTICAL APPLICATIONS**

The versatility of IoT has become very popular in recent years. There are many advantages to having a device based on IoT. **Mckinsey Global Institute reports that IoT business will reach 6.2 trillion in revenue by 2025**. There are lots of applications are available in the market in different areas.

**1) Personal Home Automation System:** Home Automation system is the major example in this area.

**Wemo Switch Smart Plug:** It is the most useful devices which connected home devices in the Switch, a smart plug. **It plugs into a regular outlet, accepts the power cable from any device, and can be used to turn it on and off on hit a button on your smart phone.**

**2) Enterprise:** In the enterprise area many applications are there like environmental monitoring system, smart environment etc.

**Nest Smart Thermostat:** It is connected to the internet. **The Nest learns automatically your family’s routines and will automatically adjust the temperature based on your activities, to make your house more efficient.** There is also a mobile app which allows the user to edit temperature and schedules.

**3) Utilities:** **smart metering, smart grid, and water monitoring system** are the most useful applications in the various utility areas.

**4) Energy Management:** **Advanced Metering Infrastructure** is the major example in this area.

**5) Medical and Health Care:** **Remote health monitoring and emergency notification system** are examples of IOT in the medical field.

**Health patch  Health Monitor:** It can be used for the patient who can’t go to doctors, letting them get ECG, heart rate, respiratory rate, skin temperature, body posture, fall detection, and activity readings remotely.

**6) Transportation:** **Electronic toll collection system** is the most useful example in this area.

**7) Large scale deployment:** There are various large projects ongoing in the world. **Songdo (South Korea), the first of its kind fully wired Smart City, is near completion**. Everything in this city is planned to be wired, connected and turned into a data stream that would be monitored by an array of computers without any human interaction.

**Another example is the Sino-Singapore Guangzhou Knowledge City work on improving air and water quality.** French company Sigfox commenced building an ultra-narrowband wireless data network in the San Francisco Bay area in 2014.

**Another example is the one completed by New York Waterways in New York City to connect all their vessels and being able to monitor them 24/7.**

So these are large applications are present in the market which is based on IoT. This world is going to become a better place to live with more communication with everyone. In near future large number of devices connected to the internet and provide great facilities to the world.

**FUTURE SCOPE OF IOT**

**According to Gartner (an information technology research and advisory firm), consumer applications will drive the number of connected things, while Enterprise will account for most of the revenue. Gartner estimated that 2.9 billion connected things are in use in the consumer sector in 2015 and would increase to over 13 billion till 2020.**

**The UK Government allocated £40,000,000 towards research into the Internet of Things in their 2015 budget.**

**IOT -  the new revolution**

When we look at today’s state of technologies, we get a clear indication of how IoT will be implemented on a global level in near future. Use of the internet is increasing day-by-day. Commute and connectivity became easier in the present scenario. In near future, the number of internet connected devices would increase exponentially.

Although there are some issues in IoT. These issues can be removed in near future. **With such a rapid growth, the day is not too far that we can decide our dinner even before reaching home on the way.**

**COMPONENTS REQUIRED:**

**HARDWARE:**

* **MQ – 135 GAS SENSORS**
* **HC – 05 BLUETOOTH MODULE**
* **DHT – 11 MODULE**
* **NODEMCU**

**SOFTWARE:**

* **ARDUINO**
* **MIT APP INVENTOR**

MQ SENSORS:

The MQ series of gas sensors utilizes a small heater inside with an electro chemical sensor these sensors are sensitive to a range of gasses are used at room temperature. MQ135 alcohol sensor is a Sno2 with a lower conductivity of clean air. When the target explosive gas exists, then [the sensor’s](https://www.elprocus.com/sensors-types-applications/) conductivity increases more increasing more along with the gas concentration rising levels. By using [simple electronic circuits](https://www.elprocus.com/build-your-own-electronic-circuits-for-simple-applications/), it convert the charge of conductivity to correspond output signal of gas concentration

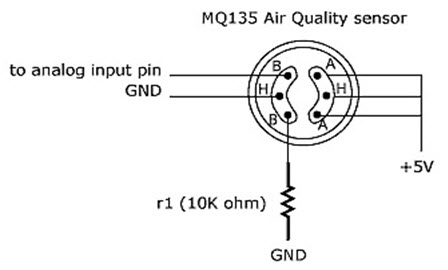
The MQ135 gas sensor has high sensitivity in ammonia, sulfide, benze steam, smoke and in other harm full gas. It is low cost and suitable for different applications. There are different types of alcohol sensors like MQ-2, MQ-3, MQ-4, MQ-5, MQ-6, etc.

MQ – 135 GAS SENSOR :

The MQ-135 gas sensor senses the gases like ammonia nitrogen, oxygen, alcohols, aromatic compounds, sulfide and smoke. The [boost converter](https://www.elprocus.com/dc-dc-converter-types/) of the chip MQ-3 gas sensor is PT1301. The operating voltage of this gas sensor is from 2.5V to 5.0V. The MQ-3 gas sensor has a lower conductivity to clean the air as a gas sensing material. In the atmosphere we can find polluting gases, but the conductivity of gas sensor increases as the concentration of polluting gas increases. MQ-135 gas sensor can be implementation to detect the smoke, benzene, steam and other harmful gases. It has potential to detect different harmful gases. The MQ-135 gas sensor is low cost to purchase. The basic image of the MQ-135 sensor is shown in the below figure.

The air quality sensor is also a MQ-135 sensor for detecting venomous gases that are present in the air in homes and offices. The gas sensor layer of the sensor unit is made up of tin dioxide (SnO2); it has lower conductivity compare to clean hair and due to air pollution the conductivity is increases. The air quality sensor detects ammonia, nitrogen oxide, smoke, CO2 and other harmful gases. The air quality sensor has a small potentiometer that permits the adjustment of the load resistance of the sensor circuit. The 5V [power supply](https://www.elprocus.com/types-power-supplies/)is used for air quality sensor.





The air quality sensor is a signal output indicator instruction. It has two outputs: analog output and [TTL output](https://www.elprocus.com/transistor-transistor-logic-ttl/). The TTL output is low signal light which can be accessed through the IO ports on the Microcontroller. The analog output is an concentration, i.e. increasing voltage is directly proportional to increasing concentration. This sensor has a long life and reliable stability as well.

#### Characteristics Of MQ -135

* Good sensitivity to harmful gases in wide range.
* It has long life and low cost.
* Possesses high sensitivity to ammonia, benzene, sulfide gases.

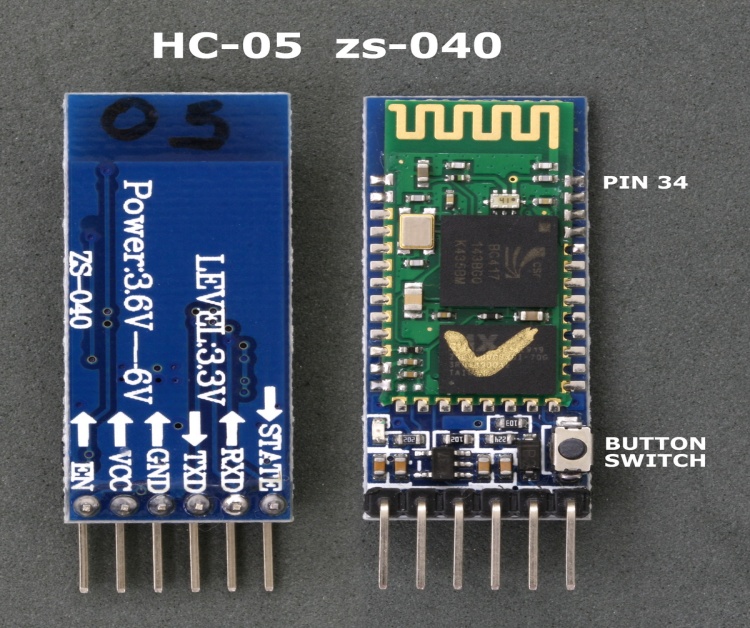
It is a simple drive circuit.

#### Applications Of MQ 135 Gas Sensor

* Air quality monitor
* Detection of harmful gases
* Domestic air pollution detection
* Industrial pollution detection
* Portable air pollution detection

**HC-05 Bluetooth Module :**

**HC‐05 module** is an easy to use **Bluetooth SPP (Serial Port Protocol) module**, designed for transparent wireless serial connection setup. The HC-05 Bluetooth Module can be used in a Master or Slave configuration, making it a great solution for wireless communication. This serial port bluetooth module is fully qualified **Bluetooth V2.0+EDR (Enhanced Data Rate)** 3Mbps Modulation with complete 2.4GHz radio transceiver and baseband. It uses **CSR Blue core 04**‐External single chip Bluetooth system with CMOS technology and with AFH (Adaptive Frequency Hopping Feature).



The Bluetooth module HC-05 is a MASTER/SLAVE module. By default the factory setting is SLAVE. The Role of the module (Master or Slave) can be configured only by AT COMMANDS. The slave modules cannot initiate a connection to another Bluetooth device, but can accept connections. Master module can initiate a connection to other devices. The user can use it simply for a serial port replacement to establish connection between MCU and GPS, PC to your embedded project, etc. Just go through the datasheet for more details

### Hardware Features :

* Typical ‐80dBm sensitivity.
* Up to +4dBm RF transmit power.
* 3.3 to 5 V I/O.
* PIO (Programmable Input/Output) control.
* UART interface with programmable baud rate.
* With integrated antenna.
* With edge connector.

PIN DESCRIPTION :

**ENABLE:**

When enable is pulled **LOW**, the module is disabled which means the module will **not turn on** and it **fails to communicate**. When enable is **left open or connected to 3.3V**, the module is enabled i.e. the module **remains on** and **communication also takes place**.

**Vcc:**

Supply Voltage 3.3V to 5V

**GND:**

Ground pin

**TXD & RXD:**

These two pins acts as an UART interface for communication

**STATE:**

It acts as a status indicator. When the module is **not connected to / paired** with any other Bluetooth device, signal goes **Low**. At this **low state**, the **led flashes continuously** which denotes that the module is **not paired** with other device. When this module is **connected to/paired** with any other Bluetooth device, the signal goes **High**. At this **high state**, the **led blinks with a constant delay** say for example 2s delay which indicates that the module is **paired**.

**BUTTON SWITCH:**

This is used to switch the module into AT command mode. To enable AT command mode, press the button switch for a second. With the help of AT commands, the user can change the parameters of this module but only when the module is not paired with any other BT device. If the module is connected to any other Bluetooth device, it starts to communicate with that device and fails to work in AT command mode.

**DHT11 Module :**

The **DHT11** is a basic, ultra low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor (NTC temperature sensor) to measure the surrounding air

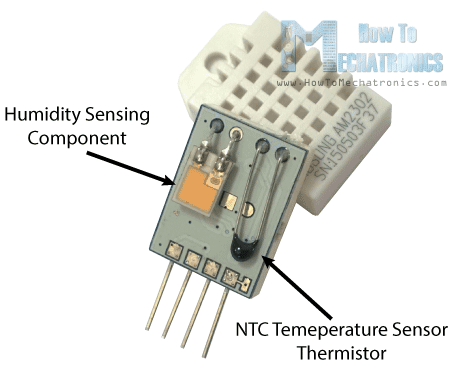
(NTC = negative temperature coefficient)



This DHT11 Temperature and Humidity Sensor features a calibrated digital signal output with the temperature and humidity sensor capability. It is integrated  with a high-performance 8-bit microcontroller. Its technology ensures the high reliability and excellent long-term stability.  This sensor includes a resistive element and a sensor for wet NTC temperature measuring devices. It has excellent quality, fast response; anti-interference ability and high performance.DHT11 can be interface with any microcontroller like Arduino, Raspberry Pi, etc. and get instantaneous results.

TECHNICAL DETAILS :

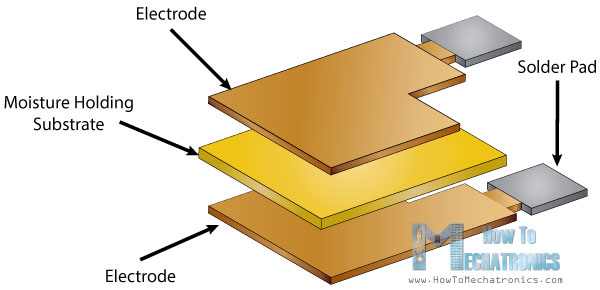
* Low cost
* 3 to 5V power and I/O
* Its operating point is 3.3v.
* 2.5mA max current use during conversion (while requesting data)
* Good for 20-80% humidity readings with 5% accuracy
* Good for 0-50°C temperature readings ±2°C accuracy
* No more than 1 Hz sampling rate (once every second)
* Body size 15.5mm x 12mm x 5.5mm
* 4 pins with 0.1" spacing



Working process :

In dht11, humidity sensing component, a NTC temperature sensor (or thermistor) and an IC on the back side of the sensor.

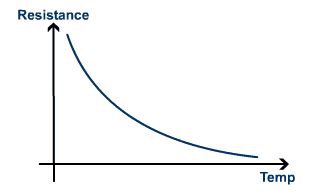
HUMIDITY SENSOR :



humidity sensing component which has two electrodes with moisture holding substrate between them. So as the humidity changes, the conductivity of the substrate changes or the resistance between these electrodes changes. This change in resistance is measured and processed by the IC which makes it ready to be read by a microcontroller.

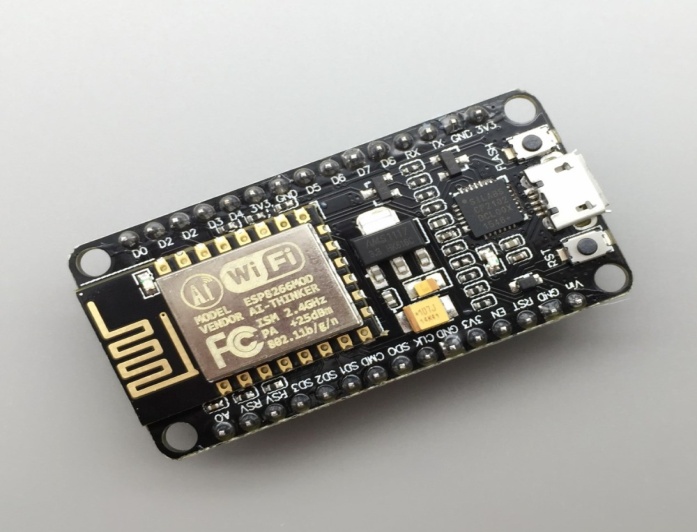
NTC temperature (thermistor):

A thermistor is actually a variable resistor that changes its resistance with change of the temperature. These sensors are made by sintering of semi conductive materials such as ceramics or polymers in order to provide larger changes in the resistance with just small changes in temperature. The term “NTC” means “Negative Temperature Coefficient”, which means that the resistance decreases with increase of the temperature.



NODE MCU :

* **NodeMCU** is an open source [IoT](https://en.wikipedia.org/wiki/Internet_of_Things) platform. It includes [firmware](https://en.wikipedia.org/wiki/Firmware) which runs on the [ESP8266](https://en.wikipedia.org/wiki/ESP8266) [Wi-Fi](https://en.wikipedia.org/wiki/Wi-Fi) [SoC](https://en.wikipedia.org/wiki/System_on_a_chip) from Espressif Systems, and hardware which is based on the ESP-12 module.
* NodeMCU was created shortly after the [ESP8266](https://en.wikipedia.org/wiki/ESP8266) came out. On December 30, 2013, The ESP8266 is a Wi-Fi SoC integrated with a [Tensilica](https://en.wikipedia.org/wiki/Tensilica) Xtensa LX106 core

****

Nodemcu consist of ESP8266 wifi enabled chip. The **ESP8266** is a low-cost [Wi-Fi](https://en.wikipedia.org/wiki/Wi-Fi) chip developed by Espressif Systems with TCP/IP protocol. For more information about ESP8266

Its basically an SoC (System on Chip)

A System on a Chip or System on Chip (SoC) is an integrated circuit that integrates all components of a computer or other electronic systems.

It has operates at **80 to 160 MHz** adjustable clock frequency and supports **RTOS**.

There’s also **128 KB RAM and 4MB of Flash memory**.

The ESP8266 Integrates **802.11b/g/n HT40 Wi-Fi transceiver**, so it can not only connect to a WiFi network and interact with the Internet, but it can also set up a network of its own, allowing other devices to connect directly to it. This makes the ESP8266 NodeMCU even more versatile.

## Power Requirement

As the operating voltage range of ESP8266 is **3V to 3.6V**, the board comes with a LDO voltage regulator to keep the voltage steady at 3.3V. It can reliably supply up to 600mA, which should be more than enough when ESP8266 pulls as much as **80mA during RF transmissions**. The output of the regulator is also broken out to one of the sides of the board and labeled as 3V3. This pin can be used to supply power to external components.

**Power to the ESP8266 NodeMCU**is supplied via the **on-board MicroB USB connector**. Alternatively, if you have a regulated 5V voltage source, the **VIN pin** can be used to directly supply the ESP8266 and its peripherals.

## Peripherals and I/O :

The ESP8266 NodeMCU has total **17 GPIO pins** broken out to the pin headers on both sides of the development board. These pins can be assigned to all sorts of peripheral duties, including:

* **ADC channel** – A 10-bit ADC channel.
* **UART interface** – UART interface is used to load code serially.
* **PWM outputs** – PWM pins for dimming LEDs or controlling motors.
* **SPI, I2C & I2S interface** – SPI and I2C interface to hook up all sorts of sensors and peripherals.

## ****I2S interface**** – I2S interface if you want to add sound to your project.

## On-board Switches & LED Indicator :

The ESP8266 NodeMCU features two buttons. One marked as **RST** located on the top left corner is the Reset button, used of course to reset the ESP8266 chip. The other **FLASH** button on the bottom left corner is the download button used while upgrading firmware.

The board also has a**LED indicator** which is user programmable and is connected to the D0 pin of the board.

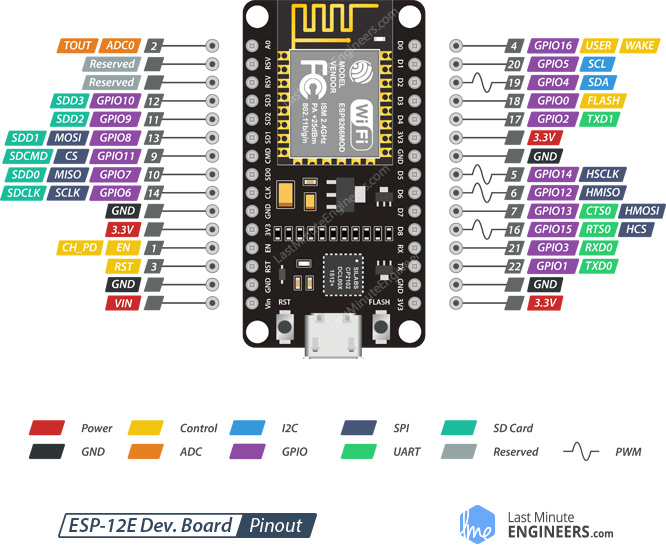
## Serial Communication :

* The board includes CP2102 USB-to-UART Bridge Controller from [Silicon Labs](http://www.silabs.com/), which converts USB signal to serial and allows your computer to program and communicate with the ESP8266 chip. Is has 4.5 mbps communication speed.

## ESP8266 NodeMCU Pinout :

The ESP8266 NodeMCU has total 30 pins that interface it to the outside world. The connections are as follows:

* 4.5 Mbps communication speed

4.5 Mbps communication speed

* 4.5 Mbps communication
* **Power pins: 4There** are four power pins viz. one VIN pin & three 3.3V pins. The VIN pin can be used to directly supply the ESP8266 and its peripherals, if you have a regulated 5V voltage source. The 3.3V pins are the output of an on-board voltage regulator. These pins can be used to supply power to external components.
* **GND :** is a ground pin of ESP8266 NodeMCU development board.

**12C PINS :** are used to hook up all sorts of I2C sensors and peripherals in your project. Both I2C Master and I2C Slave are supported. I2C interface functionality can be realized programmatically, and the clock frequency is 100 kHz at a maximum. It should be noted that I2C clock frequency should be higher than the slowest clock frequency of the slave devices.

**GPIO PINS :** ESP8266 NodeMCU has 17 GPIO pins which can be assigned to various functions such as I2C, I2S, UART, PWM, IR Remote Control, LED Light and Button programmatically. Each digital enabled GPIO can be configured to internal pull-up or pull-down, or set to high impedance. When configured as an input, it can also be set to edge-trigger or level-trigger to generate CPU interrupts.

**ADC CHANNEL :** The NodeMCU is embedded with a 10-bit precision SAR ADC. The two functions can be implemented using ADC viz. Testing power supply voltage of VDD3P3 pin and testing input voltage of TOUT pin. However, they cannot be implemented at the same time.

**UART PINS :**  ESP8266 NodeMCU has 2 UART interfaces, i.e. UART0 and UART1, which provide asynchronous communication (RS232 and RS485), and can communicate at up to 4.5 Mbps. UART0 (TXD0, RXD0, RST0 & CTS0 pins) can be used for communication. It supports fluid control. However, UART1 (TXD1 pin) features only data transmit signal so, it is usually used for printing log.

**SPI PINS :** ESP8266 features two SPIs (SPI and HSPI) in slave and master modes. These SPIs also support the following general-purpose SPI features:

* 4 timing modes of the SPI format transfer
* Up to 80 MHz and the divided clocks of 80 MHz
* Up to 64-Byte FIFO

**SDIO PIN :**  ESP8266 features Secure Digital Input/output Interface (SDIO) which is used to directly interface SD cards. 4-bit 25 MHz SDIO v1.1 and 4-bit 50 MHz SDIO v2.0 are supported.

**PWM PINS :**  The board has 4 channels of Pulse Width Modulation (PWM). The PWM output can be implemented programmatically and used for driving digital motors and LEDs. PWM frequency range is adjustable from 1000 μs to 10000 μs, i.e., between 100 Hz and 1 kHz.

**CONTROL PINS :** are used to control ESP8266. These pins include Chip Enable pin (EN), Reset pin (RST) and WAKE pin.

* EN pin – The ESP8266 chip is enabled when EN pin is pulled HIGH. When pulled LOW the chip works at minimum power.
* RST pin – RST pin is used to reset the ESP8266 chip.
* WAKE pin – Wake pin is used to wake the chip from deep-sleep.

SOTFWARE

ARDUINO :

* Arduino IDE is an open source software that is mainly used for writing and compiling the code into the Arduino Module.
* It is an official Arduino software, making code compilation too easy that even a common person with no prior technical knowledge can get their feet wet with the learning process.
* It is easily available for operating systems like MAC, Windows, Linux and runs on the Java Platform that comes with inbuilt functions and commands that play a vital role for debugging, editing and compiling the code in the environment.
* A range of Arduino modules available including Arduino Uno, Arduino Mega, Arduino Leonardo, [Arduino Micro](https://www.theengineeringprojects.com/2018/09/introduction-to-arduino-micro.html) and many more.
* Each of them contains a microcontroller on the board that is actually programmed and accepts the information in the form of code.
* The main code, also known as a sketch, created on the IDE platform will ultimately generate a Hex File which is then transferred and uploaded in the controller on the board.
* The IDE environment mainly contains two basic parts: Editor and Compiler where former is used for writing the required code and later is used for compiling and uploading the code into the given Arduino Module.

This

* Environment supports both C and C++ languages.
* **Arduino** consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or **IDE** (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board.
* The **Arduino** board is connected to a computer via USB, where it connects with the **Arduino** development environment (**IDE**). The user writes the **Arduino** code in the **IDE**, then uploads it to the microcontroller which executes the code, interacting with inputs and outputs such as sensors, motors, and lights
* It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments.

MIT APP INVENTOR :

Hour of Code with **MIT App Inventor**. **App Inventor** is a free, cloud-based service that allows you to make your own mobile apps using a blocks based programming language.

 App Inventor lets you develop applications for Android phones using a **web browser** and either a connected phone or emulator. The App Inventor servers store your work and help you keep track of your projects. You build apps by working with: The App Inventor Designer, where you select the components for your app.

(In the first version of App Inventor, the **blocks editor** was implemented in **Java** on top of the **Open Blocks** blocks framework.) The Designer window is implemented in **Java** using the Google Web Toolkit (GWT), which is translated to JavaScript so that it, too, runs in the browser

**MIT App Inventor** is an intuitive, visual programming environment that allows everyone – even children – to build fully functional apps for smart phones and tablets. Those new to **MIT App Inventor** can have a simple first **app** up and running in less than 30 minutes.

**App Inventor** lets you develop applications for Android phones using a web browser and either a connected phone or emulator. The **App Inventor** servers store your work and help you keep track of your projects. You build apps by working with: The **App Inventor** Designer, where you select the components for your **app**.

**MIT App Inventor** is basically for creating Android Apps in your browser where you design how the app will look and function. Like fitting together puzzle pieces, you set how your app will behave to different events by simply signing in with your Gmail Account, so that the App Inventor server can store your work and help you keep track of projects.  
  
It basically consists of two parts:

1. **App Inventor Designer**
2. **App Inventor Block Editor**

In **App Inventor Designer** you select the components for your app while in **App Inventor Block Editor** you assemble program blocks that specify how the components should behave visually, fitting pieces together like pieces of a puzzle. After completing the above phases you may run your app directly in your Android phone by connecting it to your computer or may run it on Android Emulator if you don't have an Android phone.

ANALYSIS :

**Satellites :**

Satellites are basically equipment platforms located in space. Each satellite has a 'payload' which describes its purpose. For example a satellite fitted with radio equipment is a 'communication satellite'. On the other hand a satellite fitted with weather measuring instruments is a 'weather satellite'.

Communication satellites send and receive data from one point on the Earth to another point on the Earth.

**Types of orbits**

* Geostationary transfer **orbit**. This is an elliptical Earth **orbit** used to transfer a spacecraft from a low altitude **orbit** or flight trajectory to geostationary **orbit**. ...
* Low Earth **orbits**. ...
* Medium low Earth **orbit**. ...
* Polar **orbits**. ...
* Sun synchronous **orbits**.

**Orbits - Low Earth Orbit**

Equally important to the design of a satellite is its orbit. This is the path it takes around the Earth.

There are two very popular paths. Polar orbit - where the satellite travels over the North and South Poles and the other is 'Equatorial' orbit where the satellite moves around the equator. A satellite in polar orbit will be able to scan the entire planet, as the Earth rotates beneath it.

* Many satellites are only a few hundred kilometers above the planet. These are called 'Low Earth Orbit' (LEO) satellites.

From the ground they appear to move rapidly from one horizon to the other. They have some advantages over satellites further away.

**Weather satellites**

Weather satellites are designed to monitor the Earth and so they are fitted with a whole array of specialised sensors, such as Infra-Red, radar and optical sensors.

Weather satellites monitor

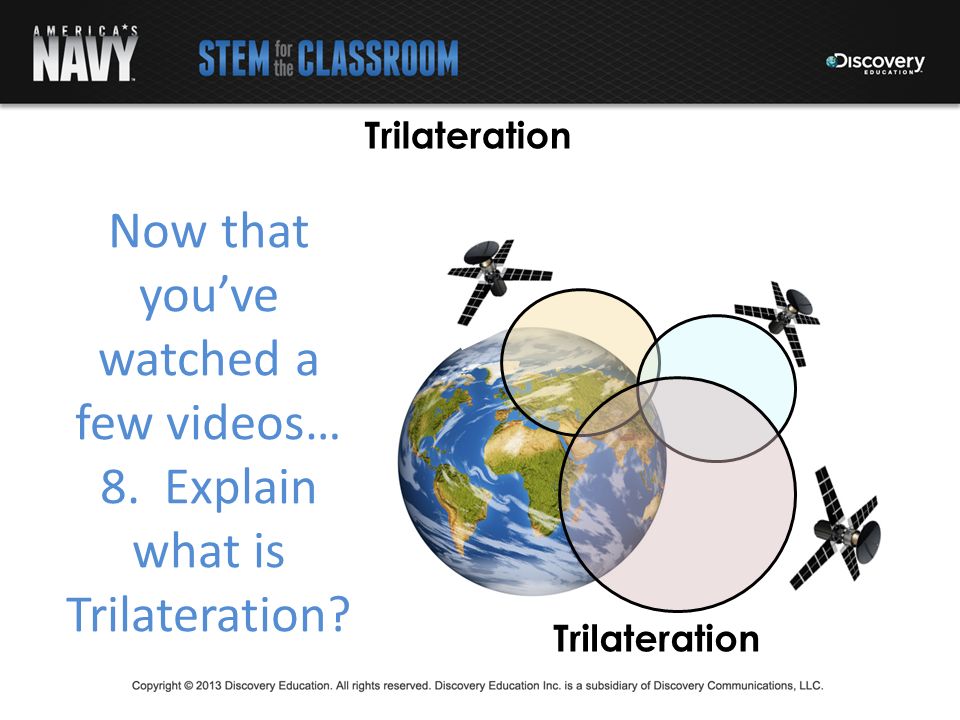
* Cloud cover
* Storms, Hurricanes and Typhoons
* Surface temperatures
* Rainfall
* Ice and snow cover

They are an essential tool to help warn people of impending storms and weather extremes. They are critical for monitoring and measuring the effects of global warming.

For 'whole world' monitoring a weather satellite is placed in geostationary equatorial orbit. But for close up mapping of the weather, they are placed in polar orbit only a few hundred kilometers up..

**Navigation satellites - how they work**

There is a constellation of 24 low earth orbit satellites covering the entire earth called NAVSTAR. This gives enough coverage to have 3 or 4 satellites visible from anywhere. Accuracy is good to about a foot or even better with more sophisticated equipment.



Here's how GPS works in six steps:

1. GPS works by using a method called "triangulation" or "trilateration".
2. It needs to get a message from at least three, preferably four satellites
3. To "triangulate", a GPS **receiver** measures the distance between itself and each satellite. It can measure distance because it works out exactly **how long** it took for each satellite's message to arrive. (distance = time of arrival \* speed of light)
4. To measure travel time, GPS needs very accurate timing which it achieves with **atomic clocks** on board each satellite.
5. Along with distance, the device needs to know exactly **where** the satellites are in space at any given time. This information is held inside the GPS receiver itself.
6. Finally, because it knows exactly where the satellites are at that instant, by using some very clever mathematics, it can work out where it is on the ground.

Triangulation, is the key idea behind GPS. It makes use of satellites in space as reference points for locations here on earth. By very, very accurately measuring our distance from at least three satellites we can "triangulate" our position anywhere on earth.

PROGRAM FOR DHT11 SENSOR AND GPRS :

#include <TinyGPS++.h>

#include <SoftwareSerial.h>

#include <ESP8266WiFi.h>

#include <PubSubClient.h>

TinyGPSPlus gps; // The TinyGPS++ object

SoftwareSerial ss(D2, D3); // The serial connection to the GPS device

const char\* ssid = "Rupesh";

const char\* password = "Rupesh@123";

#include "DHT.h"

#define DHTPIN D1 // what pin we're connected to

#define DHTTYPE DHT11 // define type of sensor DHT 11

DHT dht (DHTPIN, DHTTYPE);

float latitude , longitude;

int year , month , date, hour , minute , second;

String date\_str , time\_str , lat\_str , lng\_str;

int pm;

#define ORG "vsz0ie"

#define DEVICE\_TYPE "rupesh"

#define DEVICE\_ID "asvr"

#define TOKEN "rupeshkumar"

char server[] = ORG ".messaging.internetofthings.ibmcloud.com";

char topic[] = "iot-2/evt/Data/fmt/json";

char authMethod[] = "use-token-auth";

char token[] = TOKEN;

char clientId[] = "d:" ORG ":" DEVICE\_TYPE ":" DEVICE\_ID;

WiFiClient wifiClient;

PubSubClient client(server, 1883,wifiClient);

//WiFiServer server1(80);

void setup()

{

Serial.begin(115200);

ss.begin(9600);

Serial.println();

Serial.print("Connecting to ");

Serial.println(ssid);

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED)

{

delay(500);

Serial.print(".");

}

Serial.println("");

Serial.println("WiFi connected");

// server.begin();

// Serial.println("Server started");

// Print the IP address

Serial.println(WiFi.localIP());

dht.begin();

}

void loop()

{

float h = dht.readHumidity();

float t = dht.readTemperature();

//float l = dht.readlight();

if (isnan(h) || isnan(t))

{

Serial.println("Failed to read from DHT sensor!");

delay(1000);

return;

}

while (ss.available() > 0)

if (gps.encode(ss.read()))

{

if (gps.location.isValid())

{

latitude = gps.location.lat();

lat\_str = String(latitude , 6);

Serial.println(lat\_str);

longitude = gps.location.lng();

lng\_str = String(longitude , 6);

Serial.println(lng\_str);

}

if (gps.date.isValid())

{

date\_str = "";

date = gps.date.day();

month = gps.date.month();

year = gps.date.year();

if (date < 10)

date\_str = '0';

date\_str += String(date);

date\_str += " / ";

if (month < 10)

date\_str += '0';

date\_str += String(month);

date\_str += " / ";

if (year < 10)

date\_str += '0';

date\_str += String(year);

}

if (gps.time.isValid())

{

time\_str = "";

hour = gps.time.hour();

minute = gps.time.minute();

second = gps.time.second();

minute = (minute + 30);

if (minute > 59)

{

minute = minute - 60;

hour = hour + 1;

}

hour = (hour + 5) ;

if (hour > 23)

hour = hour - 24;

if (hour >= 12)

pm = 1;

else

pm = 0;

hour = hour % 12;

if (hour < 10)

time\_str = '0';

time\_str += String(hour);

time\_str += " : ";

if (minute < 10)

time\_str += '0';

time\_str += String(minute);

time\_str += " : ";

if (second < 10)

time\_str += '0';

time\_str += String(second);

if (pm == 1)

time\_str += " PM ";

else

time\_str += " AM ";

}

}

// Check if a client has connected

//WiFiClient client = server.available();

int air=10;

PublishData(lat\_str,lng\_str,t,h,air);

if (!client.loop())

{

mqttConnect();

return;

}

// Prepare the response

String s = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n <!DOCTYPE html> <html> <head> <title>GPS Interfacing with NodeMCU</title> <style>";

s += "a:link {background-color: YELLOW;text-decoration: none;}";

s += "table, th, td {border: 1px solid black;} </style> </head> <body> <h1 style=";

s += "font-size:300%;";

s += " ALIGN=CENTER> GPS Interfacing with NodeMCU</h1>";

s += "<p ALIGN=CENTER style=""font-size:150%;""";

s += "> <b>Location Details</b></p> <table ALIGN=CENTER style=";

s += "width:50%";

s += "> <tr> <th>Latitude</th>";

s += "<td ALIGN=CENTER >";

s += lat\_str;

s += "</td> </tr> <tr> <th>Longitude</th> <td ALIGN=CENTER >";

s += lng\_str;

s += "</td> </tr> <tr> <th>Date</th> <td ALIGN=CENTER >";

s += date\_str;

s += "</td></tr> <tr> <th>Time</th> <td ALIGN=CENTER >";

s += time\_str;

s += "</td> </tr> </table> ";

if (gps.location.isValid())

{

s += "<p align=center><a style=""color:RED;font-size:125%;"" href=""http://maps.google.com/maps?&z=15&mrt=yp&t=k&q=";

s += lat\_str;

s += "+";

s += lng\_str;

s += """ target=""\_top"">Click here!</a> To check the location in Google maps.</p>";

}

s += "</body> </html> \n";

// client.print(s);

delay(100);

}

void mqttConnect() {

if (!client.connected()) {

Serial.print("Reconnecting MQTT client to "); Serial.println(server);

while (!client.connect(clientId, authMethod, token)) {

Serial.print(".");

delay(500);

}

initManagedDevice();

Serial.println();

}

}

void initManagedDevice() {

if (client.subscribe(topic)) {

Serial.println("subscribe to cmd OK");

} else {

Serial.println("subscribe to cmd FAILED");

}

}

void PublishData(String lat\_str, String lng\_str,int t,int h,int air){

if (!!!client.connected()) {

Serial.print("Reconnecting client to ");

Serial.println(server);

while (!!!client.connect(clientId, authMethod, token)) {

Serial.print(".");

delay(500);

}

Serial.println();

}

String payload = "{\"d\":{\"latitude\":";

payload += lat\_str;

payload+="," "\"longitude\":";

payload += lng\_str;

payload+="," "\"temperature\":";

payload += t;

payload+="," "\"humidity\":";

payload += h;

payload+="," "\"airqulity\":";

payload += air;

payload += "}}";

Serial.print("Sending payload: ");

Serial.println(payload);

if (client.publish(topic, (char\*) payload.c\_str())) {

Serial.println("Publish ok");

} else {

Serial.println("Publish failed");

}

}

OUTPUT OF THE PROJECT:

