Tank Water Level Meter (Group-18)

An IoT device for monitoring water level

Submitted To:
Prof. D.V.Gadre

Submitted By:

Arsh Kohli-2020UEC2509

Priyanshu-2020UEC2502

Gurkirat Singh-2020UEC2552

Electronics Design and Workshop(EDW) Project
Netaji Subhas University of Technology
ECE-2024

Acknowledgement

We would like to thank a few people without whom this project would certainly not have been possible. Gadre sir for his constant support and guidance. Thank you for sparking our interest in building something real, we had a ton of fun while putting this project together and are grateful for the hands-on approach that you took while teaching this course. Naman Puri bhaiyya for always being there to help us with anything related to our project. Thank you for patiently solving even the silliest doubts that we had. Rohan Deswal bhaiyya for helping us with the 3D printing process.

Synopsis

We regularly come across a situation where we see an overflowing water tank, and often hear this colloquial phrase, *"motor band kardo paani bhar gaya"*. So, we decided to develop a Tank Water Level System using components like NodeMCU ESP8266 and an ultrasonic sensor- *HC-SR04* with a primary aim to activate a buzzer whenever water tank is full. In addtion to that real time status of the tank is sent to the user using a WebApp, also it would provide the data regarding total water consumption of a household and would detect any kind of leakage in the tank.

Motivation

Our motive behind making this project is primarily to reduce wastage of water due to overflowing and leakages and also to calculate the daily water consumption. Although there are several other products in the market but they are not cost effective. So, we decided to make a very cost effective and efficient tank water meter.

Project Description

- Ultrasonic sensor-*HC-SR04*

  It is a sensor that can measure distance using echolocation. It emits ultrasound at 40 000 Hz (40kHz). A pulse is transmitted from the sensor and distance-to-water-level is determined by measuring the time required for the echo return from the water surface.

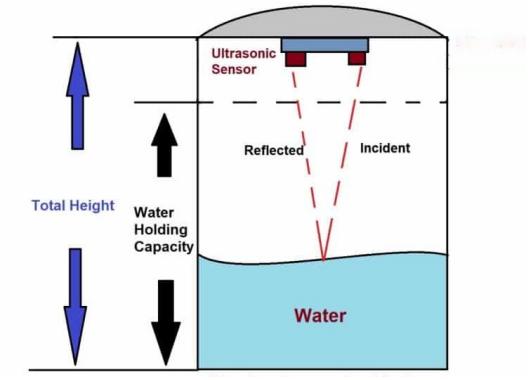

Figure 1: Working of Ultrasonic sensor

- NodeMCU ESP8266

  The brain behind our whole project, NodeMCU ESP8266 is one type of microcontroller board, designed by Espressif Systems. It is a small size board which is also flexible with a wide variety of applications. It performs all the functions ranging from calculating the tank size to sending information/status about the tank to the web app in real time. It has the capability to connect to WiFi inbuilt. Some technical specifications of the board are as follows:

  Microcontroller: Tensilica 32-bit RISC CPU Xtensa LX106
  Operating voltage: 3.3 volts
  Input voltage: 7-12 volts
  Digital I/O pins: 16
  Analog input pins: 1
  Flash memory: 4 MB
  SRAM: 64 KB
  Clock speed: 80 MHz

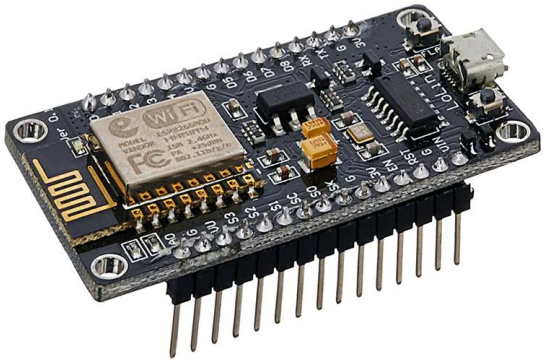

Figure 2: ESP8266 NodeMCU

- Buzzer

  It is mainly used to prompt an alarm signal whenever water level goes above a certain threshold.

Figure 3: Buzzer

Block Diagrams

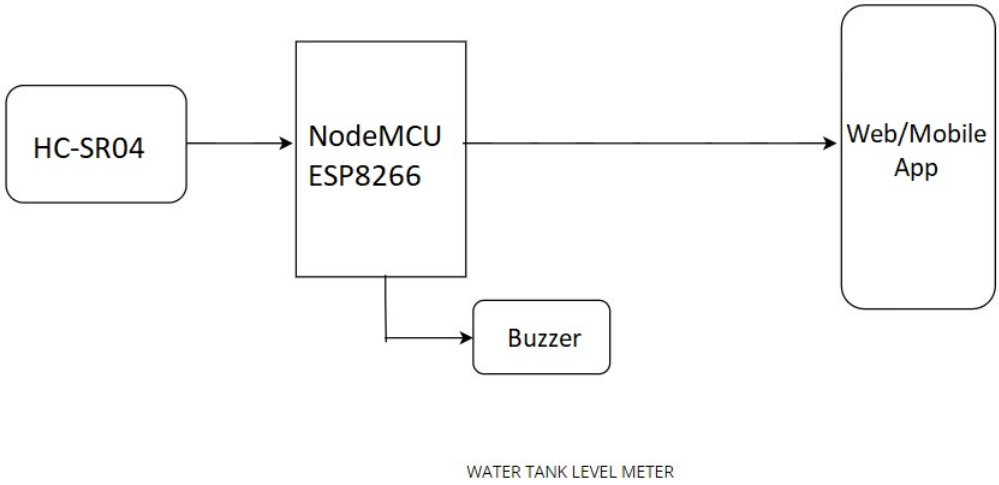

Figure 4: Block diagram

Flowcharts

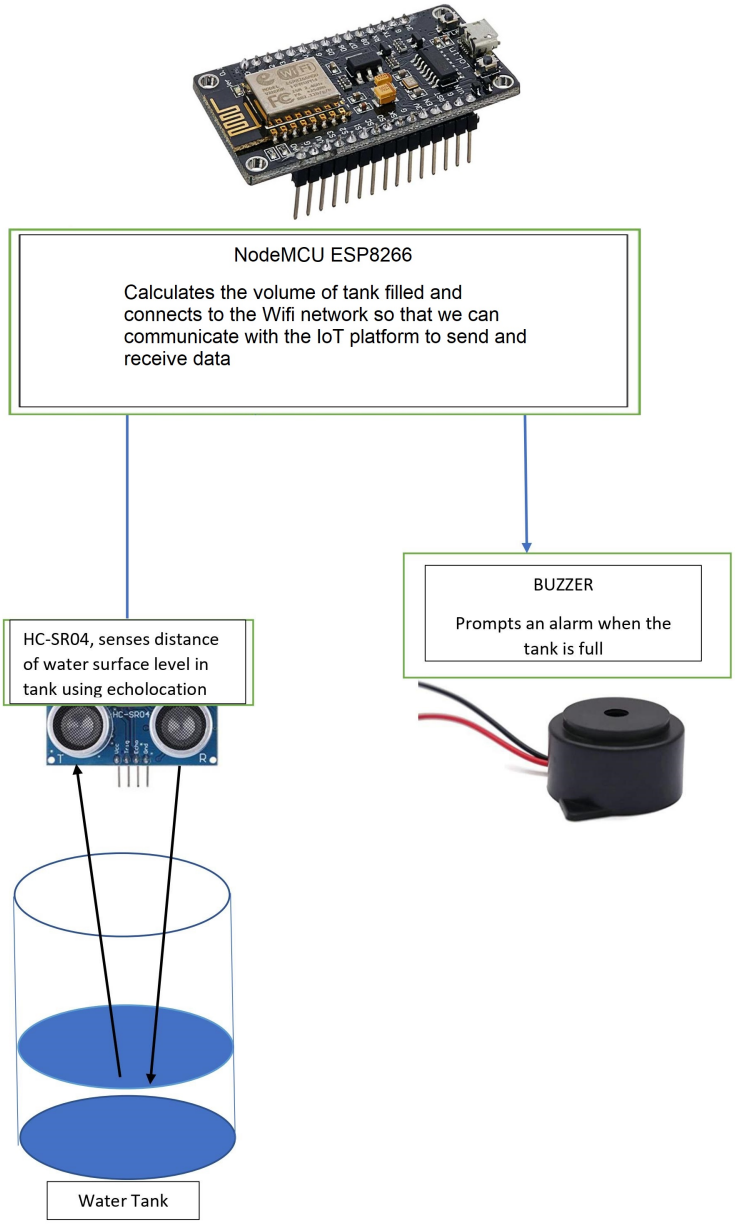

Figure 5: Flow chart-1

Bill Of Materials

S.No	Component Name	Quantity	Price
1	NodeMCU ESP8266	1	300-600
2	HC-SR406 ULTRASONIC SENSOR	1	110
3	BUZZER	1	50
4	ENCLOSING CASE	1	200-500
5	MISCELLANEOUS	-	400

Gantt Chart

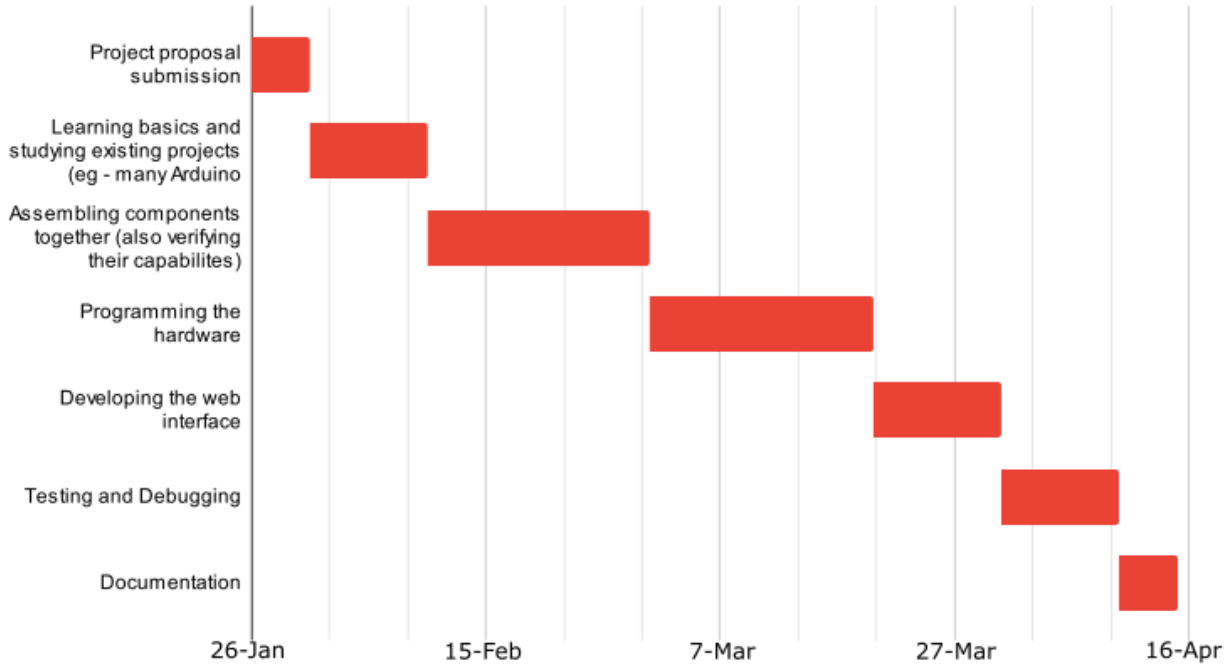

Figure 6: Gantt Chart

Difficulties faced

- Programming the ESP8266 - We had initially started with an Arduino Nano as our main microcontroller. We had planned to use it to operate the ESP8266 module however after a bit of tinkering, we found out that the ESP8266 module cannot be operated by another microcontroller, it can only be programmed to operate a certain way such that its code is separated from the code that runs on the Arduino. This way, the Arduino essentially became an inefficient UART bridge that is used to program the ESP8266 chip. We solved this problem by switching to a NodeMCU ESP8266 (aka ESP-12) microcontroller.

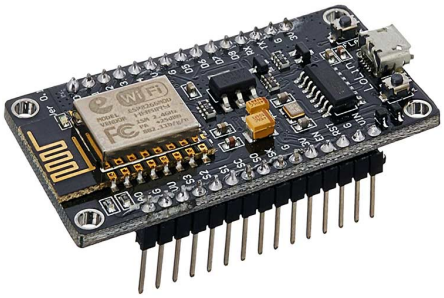

Figure 7: ESP8266 NodeMCU

- Power Issues - With the arduino, we often ran into issues where the ESP8266 module would randomly start outputting gibberish to the Serial Monitor. After hours of scouring through dozens of forums online, we figured out that the problem arose due to power issues. The Arduino could not supply enough power to the ESP8266 module, This was also solved when we switched to the NodeMCU ESP8266 microcontroller.

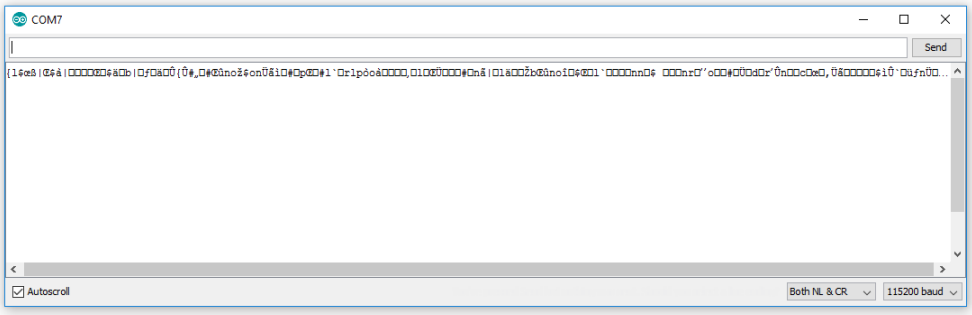

Figure 8: Gibberish on the Serial Monitor

- Driver Issues - The microcontroller would not respond to Serial AT (ATtention) commands in the beginning. After installing the relevant drivers and experimenting with different baud rates, we got it working.

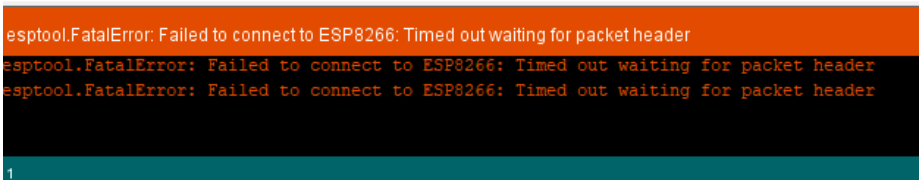

Figure 9: Driver Issues while uploading the sketch

- Complexities on the software side - Our plan initially was to send HTTP requests through the microcontroller and use it to trigger actions on the server side using an IoT service such as IFTTT or Thingspeak. The problem with these platforms was two-fold. First, they did not store past data which made plotting water level on a chart impossible without using another service (and increasing complexity). Secondly, they used HTTP to talk to the microcontroller which was inefficient. After some reasearch, we found a platform (Blynk) that kept logs of data and had a convenient library which abstracted the MQTT protocol for the ESP8266. The code had to be rewritten to use it instead of HTTP requests.

Figure 10: Block diagram of our IoT stack

Code Explanation

- On the software side of things, we are using Blynk, an IoT platform that makes it easier to integrate IoT devices with the cloud. Blynk has both a web interface and a mobile dashboard to monitor our devices outputs. We've set the outputs as virtual pins on the Blynk platform and our microcontroller (NodeMCU ESP8266) utilizes these to receive and send data to the Blynk server using the MQTT protocol.

- The setup function runs at startup and sets the Ultrasonic sensor's trigger pin as output and its echo pin as input. The buzzer is set as an output pin. Then, it attempts to connect to our Wifi network.

```
void setup() {
  delay(2000);
  Serial.begin(115200);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(buzzerPin, OUTPUT);
  Blynk.begin(auth, ssid, pass, "blynk.cloud", 80);
}
```

Figure 11: Setup function which runs on startup

- The loop function first initiates a handshake to the Blynk server. In order to refresh the water level every 4 seconds, we keep checking if 4000 milliseconds have elapsed since the last time we checked and run the code under the if statement if the condition is satisfied. This delay period change be adjusted and is in the variable named timerDelay.

- Next, we trigger an ultrasonic wave by setting the trigger pin of the Ultrasonic sensor to HIGH for a very short period of time (10 milliseconds). Now, we record the time before this wave returns to the Ultrasonic sensor. When it returns, after hitting the water level, our echo pin is set to HIGH. This time is recorded in a variable named "duration."

```
void loop() {
  Blynk.run();
  // Send an HTTP POST request every 4 seconds
  if ((millis() - lastTime) > timerDelay) {
    //Check WiFi connection status


        digitalWrite(trigPin, LOW);
        delayMicroseconds(2);
        digitalWrite(trigPin, HIGH);
        delayMicroseconds(10);
        digitalWrite(trigPin, LOW);
        duration = pulseIn(echoPin, HIGH);
```

Figure 12: First part of the loop function

- Now, we calculate the distance of the water surface using the duration. Since we have the distance of the water surface when there is no water in the tank (in a variable name minLevel), we can use this to calculate the percentage of water filled in the tank. We trigger the buzzer if the tank is more than 50% filled and buzzer alerts are turned on. After this, we update the percentage level through a virtual pin so we can track it on the Blynk platform's mobile app.

```
  distance = duration/58.2;
  Serial.println(distance);
  float perc= (float)(minLevel-distance)/minLevel*100;
  Serial.println("percenatge Filled ");
  Serial.println(perc);
  digitalWrite(buzzerPin, LOW);
  if(perc < 20)
  Serial.println("Low Level");
  if(perc > 50 && buzzerOn) {
    digitalWrite(buzzerPin, HIGH);
    Blynk.virtualWrite(V2, 1);
  }
  if(perc>90)
  Serial.println("Almost Filled");


Blynk.virtualWrite(V1, perc);
```

Figure 13: Second part of the loop function

Gallery

Figure 14: Testing Ultrasonic sensor

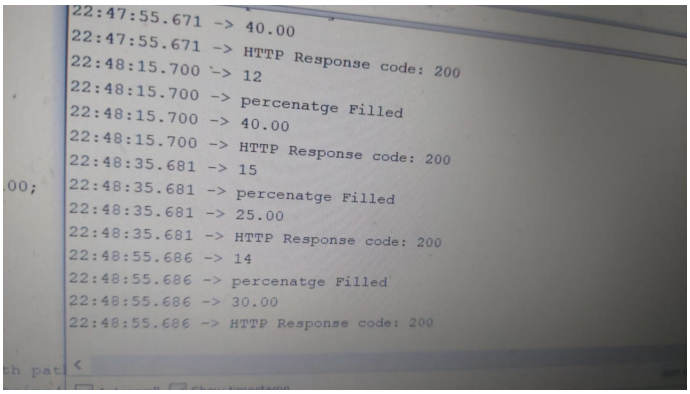

Figure 15: Fetching data from the sensor to the cloud

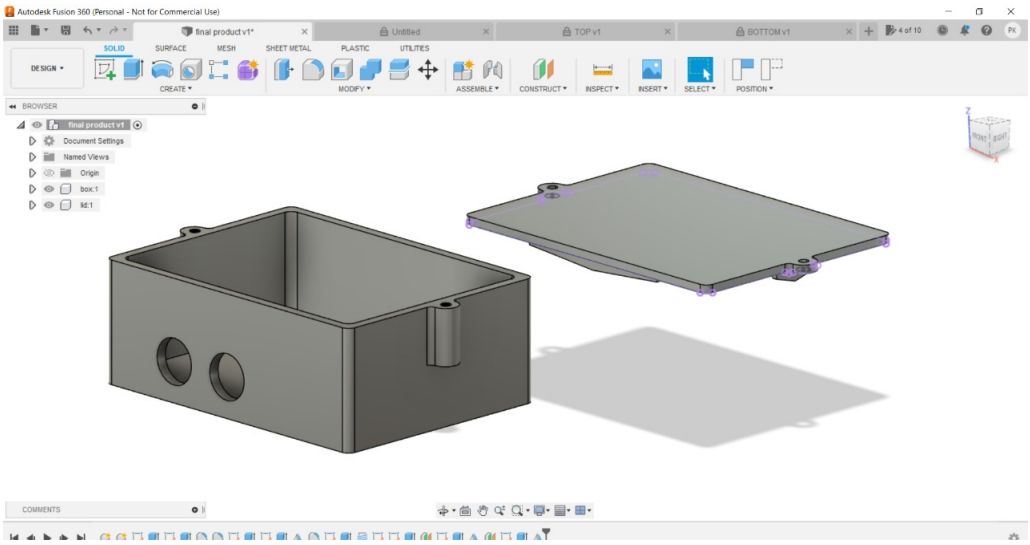

Figure 16: 3-D model of the case

Figure 17: Final Assembled Project

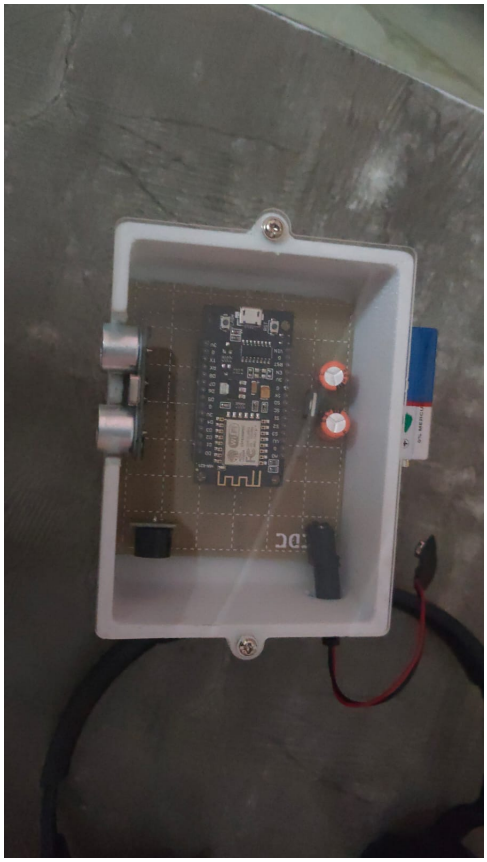

Figure 18: Final Assembled Project