

Edge-Based Intelligent Anomaly Detection for Industrial Dc Motor Using Esp32

G. Varshitha¹, G. Revanth Sai², G. Sai Avinash³

^{1,2,3}Students, Electronics And Communication Engineering, Vignana Bharathi Institute of Technology, Hyderabad, India

Abstract

This Paper proposes a simple and efficient system to monitor a DC motor and protect it from faults. Electric motors in the industries may experience abrupt failures caused by overheating, vibrations, voltage, current imbalances, and overloading. A cost-effective system was designed and implemented using ESP32 to monitor five crucial parameters, namely, vibration, current, voltage, speed, and temperature. The system can turn off the motor once any of these parameters experience an abnormal condition. The user can be notified of the fault via an LED indicator and a mobile phone using BLE. The data can also be uploaded to Thing Speak for storage. All processing take place on the ESP32, without relying on internet connectivity.

This project is a simple hardware system that is used to monitor and protect a DC motor in industries, laboratories, or as a prototype for more significant implementations.

Keywords: DC motor protection, ESP32, anomaly detection, multi-sensor monitoring, edge computing, BLE alerts.

INTRODUCTION

Electric motors are widely used in the modern industrial sector. They are used in various applications such as pumps, fans, and conveyors. The abrupt failure of these motors may cause significant losses to the industries. Most of these motors experience faults such as overheating, vibration, and overloading. The traditional methods of monitoring are inadequate and do not offer a fast response, which is critical in protecting the motors.

To address these limitations, this work presents a real-time monitoring and protection system for a DC motor based on edge computing. This system operates on the concept of edge computing. Edge computing enables sensor data to be processed locally at the source before selected information is transmitted to a remote server. This concept is essential in systems that require a quick response since it does not rely on internet connectivity.

This system was designed and implemented using ESP32 microcontroller as the central processing unit. The microcontroller was used to acquire data from five sensors. The system was able to detect an abnormal condition and consequently turn off the motor. The system was designed to offer an alert to the user through an LED indicator and a mobile phone using Bluetooth Low Energy (BLE). The system was developed to be a prototype for future implementations in the industrial field.

The motivation behind this project was to develop a working prototype that monitors a DC motor, detects faults, and offer protection to the motor. This project was inspired by the concept of anomaly detection

using edge computing.

LITERATURE SURVEY

The research article "Real-Time Anomaly Detection in Industrial Cyber-Physical Systems Using Unsupervised Machine Learning" authored by Zakea I lAgure and Shawulu Hunira Nggada suggests a method that leverages unsupervised machine learning for anomaly detection at the Cyber-Physical Systems' edge nodes. The authors discuss that the conventional supervised machine learning methods need a large volume of fault-labeled data, which is quite a challenge in the industrial environment.

This hurdle was overcome by creating a stream processing pipeline that combines three distinct unsupervised machine learning algorithms such as Autoencoders (AE), Isolation Forest (IF), and K-means and DBSCAN clustering algorithms. The proposed solution is a modular stream processing system composing of seven stages.

At first, multivariate sensor and actuator data are recorded from the CPS at the speed of one record per second. Then, data pre-processing occurs which consists of time synchronization, missing values treatment and robust scaling. Sliding windows of 60-seconds duration with 5-seconds overlap are used for partitioning the data for enhancing the effectiveness of the data analysis process. In the feature engineering step, features such as statistical features, temporal features, frequency domain features, variable correlation and actuator duty cycle are generated.

SYSTEM ARCHITECTURE

The system has five major modules which are power supply, sensors, processing unit, motor control and alert system. The system's central processor is the ESP32 microcontroller.

The power supply is a 6V-7.4V battery which powers the motor through the ACS712 current sensor and the VM pin of the DRV8833 driver. Power supply (to ESP32) is separate from laptop to avoid voltage droops. The motor driver's power pins have two capacitors: 100 μ F and 0.1 μ F.

The sensor layer consists of the ADXL345 accelerometer connected to the ESP32 via I²C interface at GPIO 21 (SDA) and GPIO 22 (SCL), ACS712 current sensor connected to the motor's power supply, a voltage sensor connected in parallel to the battery, a Hall sensor connected to GPIO 26 for RPM calculation, and DS18B20 temperature sensor connected to GPIO 4 with a 4.7k pull-up resistor. All sensors share the same ground as the ESP32 and the motor driver.

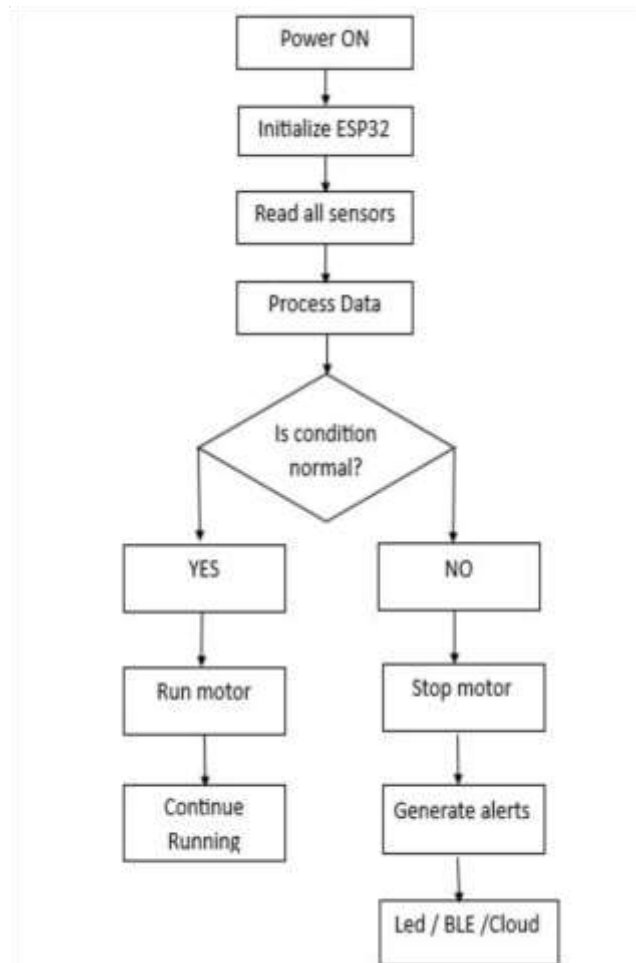
The processing layer is composed of the ESP32 microcontroller programmed to periodically read the sensor data and convert it to motor current, voltage, speed, temperature and vibration values. Then the values are compared with pre-defined thresholds, and in case of anomaly, the ESP32 turns off the motor and sends an alert to the user.

The motor control layer is made of the DRV8833 driver, connected to the ESP32 on OUT3 (GPIO25) and OUT4 (GPIO33). The driver normally provides power to the motor, but in case of an anomaly, it is turned off by the ESP32.

The alert and monitoring layer consists of three parts: an onboard LED connected to GPIO27 to indicate the system's status, a BLE module connected to GPIO 32 that sends notifications to the user's mobile phone in case of an anomaly, and Wi-fi connected to ThingSpeak to store the sensor data for later analysis. The system was designed using a breadboard and jumper wires, and the modular approach allowed individual modules to be tested while also ensuring minimal latency between the modules. The system's simplicity and low cost made it easy to build and maintain, while the processing layer's ability to make

decisions locally allowed the system to function without an internet connection. The system's main design feature is its ability to provide instantaneous protection to the motor by making decisions locally on the microcontroller.

The data flows from sensors to the processing layer of the ESP32 and then control signals are sent to the motor driver and alert system. The simple system design made it possible to build the system on breadboard. This makes the system easily portable and testable. This also allows for the replacement or upgrading of individual modules, such as the sensor layer or alert system, without affecting the rest of the system.



METHODOLOGY

We have built this project step by step instead of trying to put everything together at once. Working on one module at a time, testing it, and only then moving to the next made the whole process a lot easier to manage and a lot easier to debug when something didn't work.

The overall approach follows how embedded systems are usually built: start with the requirements, then work outward toward the full system.

We began by figuring out what the system actually needed to do. It had to monitor five things on the DC motor: vibration, current, voltage, speed, and temperature. Once we had that settled, we picked the components. The main controller is the ESP32 it has enough power to do the job, has both Wi-Fi and BLE built in and has plenty of GPIO pins. I used a DRV8833 module to drive the motor for sensors we used an

ADXL345 for vibration, an ACS712 for current, a voltage sensor module, a Hall effect sensor for RPM and a DS18B20 for temperature.

Next is the hardware. We put the whole circuit together on a regular breadboard with jumper wires, wiring each sensor to the ESP32. The voltage sensor is in parallel and the ACS712 is in series with the power line of the motor so it reads current accurately. The IC sensors are connected to GPIO 21 and 22. The Hall sensor and the temperature sensor have dedicated pins. Everything shares a common ground. We also added 100 μ F and 0.1 μ F capacitors near the motor driver to keep the power supply stable without them, the readings were noisier than we wanted then actually.

Then we moved to the code part after the hardware was done. For the main program we used the Arduino IDE to read all the sensors on a regular cycle and convert the raw values into something usable current in amps, voltage, RPM, temperature and vibration levels. Fault detection runs on simple thresholds: if any reading crosses its safe limit, the program cuts power to the motor immediately through the DRV8833. The software also handles the LED blinking at different speeds depending on system state, sends alerts over BLE to a phone, and uploads data to ThingSpeak whenever Wi-Fi is available. We used the `millis()` function throughout so none of these tasks block each other sensor reads, motor control, LED blinking, and wireless communication all run side by side instead of waiting in line.

With the individual modules working, we brought everything together on the breadboard and went through the connections between the sensors, the ESP32, the motor driver, and the alert system to make sure they all worked as one unit.

Building it this way, module by module, kept the project manageable at every stage, and made it a lot clearer how each piece actually contributes to the finished system.

EXISTING SYSTEM

Mostly the people in the industries check motors manually. Workers would listen to the sound, touch the motor to feel if it was getting hot, or use basic tools to measure current and temperature. This method was slow and faced many problems. These problems were noticed after the motor had already stopped or got damaged.

Later, some systems started using sensors and sending all the data to a cloud server for analysis, it had big problems. There was always a delay because the information had to travel to the cloud and come back. If the internet was slow or not working, the whole system would stop. These systems were expensive and not feasible for the small industries or labs.

We have studied the base paper “Proactive Anomaly Detection Framework Utilizing Edge-Driven Ubiquitous IoT for Industrial Automation Systems” by Glenn John Om Fernando. The proposed framework of this paper is EAD-IoT Net. It used edge computing and deep learning models such as CNN and LSTM to detect anomalies. The paper has shown very good accuracy in computer simulations using Kaggle datasets. It also mentioned context awareness and federated learning to reduce false alarms. However, the paper was exclusively based on simulation. It didn't create any physical hardware with a physical DC motor, multiple sensors, or automatic motor shutdown. So it provided good theoretical ideas but it did not provide a practical working device that people can actually use in real life.

PROPOSED SYSTEM

The proposed system was designed to overcome the short comings of the existing systems. The system was designed to be a simple and lowcost solution to be used in small scale industries and laboratories. The

system was designed to monitor a DC motor with multiple sensors and to protect it from damage through local decision making on the ESP32 microcontroller. This method is much faster than cloud computing and very reliable as it does not depend on internet connectivity. It was built from available parts and a bread board.

The system includes power supply, sensors, processing layer, motor control and alert system. The processing takes place in ESP32 microcontroller and it is programmed to read the data from the sensors and make decisions based on the thresholds. If an anomaly is detected, the ESP32 tells the motor driver to stop the motor and alerts the user through an LED on the board and a BLE notification. The sensor data is then uploaded to Thing Speak for storage and analysis.

The system is designed on a breadboard with jumper wires to connect the different components. The battery powered the motor through the current sensor and motor driver, while the ESP32 was powered by a separate laptop power supply. The system's simplicity and low cost allowed it to be built on a breadboard while still being able to provide instantaneous protection to the motor. The system can be used as a prototype for more complex motor monitoring systems

COMPARISON TABLE

Parameters	Base Pape	Current project
Working type	Only computer simulation using datasets	Real working hardware prototype with real DC motor.
Implementation	Software-based (Python, CNN+LSTM models).	Complete hardware using ESP32, sensor, and motor driver.
Hardware Used	No real hardware, no physical motor	ESP32, DRV8833 driver, 5real sensors, DC motor.
Fault Detection	Advanced deep learning in PC.	Simple threshold based detection on ESP32
Motor Protection	No automatic shutdown or motor protection.	Automatic motor shutdown when fault is detected.
Alerts	Only simulation results, no real alerts.	LED blinking+BLE mobile notification + ThingSpeak cloud
Response Time	Depends on cloud or computer.	Very fast realtime response (edge computing)
Cost	Expensive setup.	Very low cost, easy to build on breadboard

RESULT AND TESTING

When the hardware and software were finished, we tested the entire system many times to ensure that it all worked properly. The tests were performed in two main scenarios, which were normal operation and fault conditions.

Under normal conditions running on a 7.4V battery the motor started up and ran at a safe, low speed. All sensor readings were clean on the Serial Monitor: temperature holding around 25-30°C, current between

0.8 and 2.0 A, voltage around 6.5-7V, RPM reading correctly, vibration staying low. The LED flashed slowly, so all was well, and the motor kept running, no problems.

We also ran the system for 30-40 minutes to check its stability over time. The readings were steady and the motor never stopped by itself. With the Wi-Fi connected the data reliably made it to ThingSpeak and We could check the values from my phone.

The fault testing was carried out by using non-destructive techniques. To cause a current fault we would lightly put my finger on the motor shaft for a second or two to add load. The current reading immediately hit the threshold, the system flagged the fault, the motor stopped, the LED became a fast blink and a BLE message of “FAULT! Motor Stopped” reached my phone. We have did this multiple times, always with the same result.

we had a voltage fault, so we quickly popped the jumper wire off the positive terminal of the battery and put it back on. The voltage reading dropped and the system detected the fault, shut down the motor and issued the same alerts confirming that both the voltage sensor and the fault logic were working properly. We also tried pulling the temperature sensor completely. The system returned to the default value of 25°C and continued to run normally without producing a false fault. The fallback worked exactly as planned. Every time, the BLE notifications came through on my phone within a few seconds and we could easily see at a glance the difference between the normal and fault LED patterns. Every time a fault tripped the motor off immediately and it stayed off until we restarted the system. The auto protection worked as it should. The prototype was providing stable and reliable results for each test case. Response times were still fast, it stood up to sustained use, and it met all the major objectives I’d established for it.



Fig.1. shows the output circuit of the project.

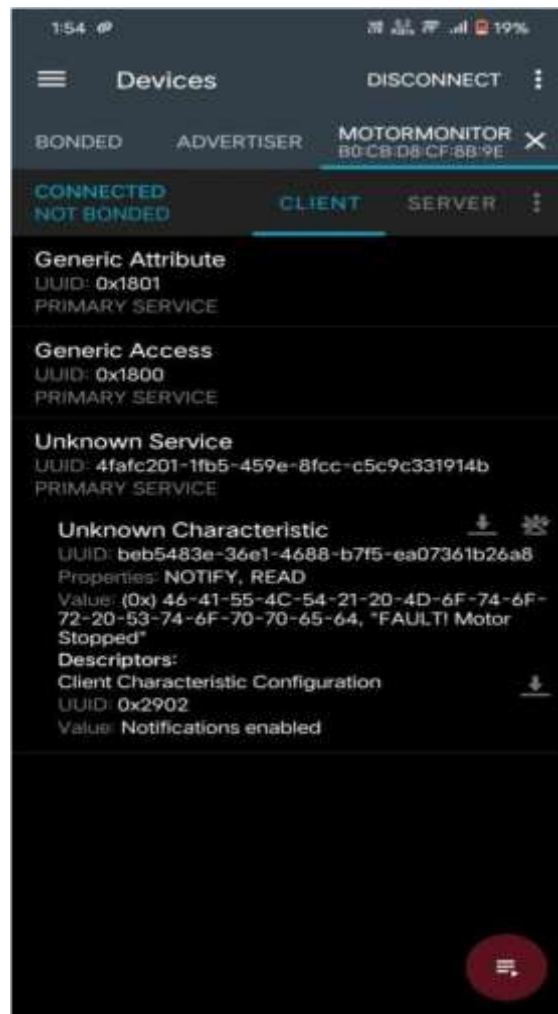


Fig.2. shows the BLE alerts in mobile phone when it detects the fault.

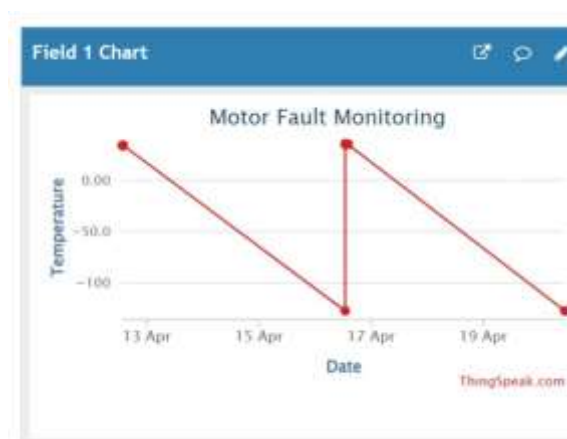


Fig.3. shows the Motor fault monitoring showing temperature variations in ThingSpeak.

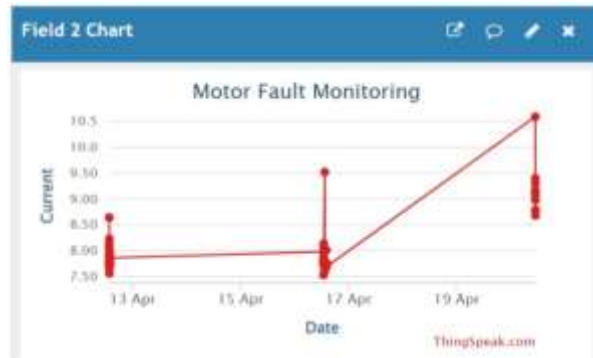


Fig.4. shows the Motor fault monitoring showing current variations in Thingspeak.

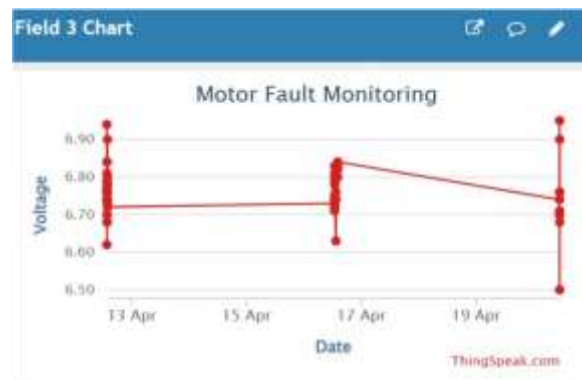


Fig.5. shows the Motor fault monitoring showing voltage variations in Thingspeak.



Fig.6. shows the Motor fault monitoring showing vibration variations in ThingSpeak.

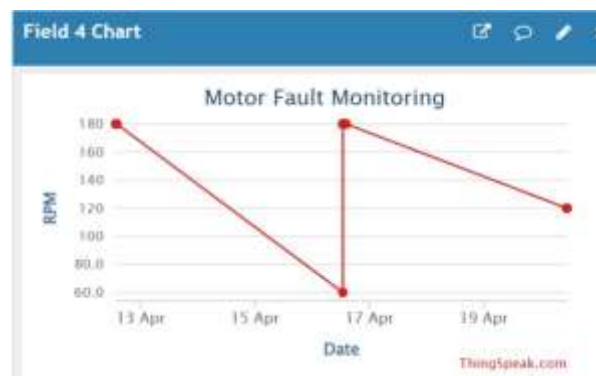


Fig.7. shows the Motor fault monitoring showing RPM variations in Thingspeak.

CONCLUSION

We have done a project for building a system to detect if a DC motor is not working well and solve the problem at the right time. The system is used to monitor the motors vibration, current, voltage, speed and temperature all the time by using 5 sensors and Esp32. If something is wrong it disables the motor and notifies the user with a light and a message on his/her phone..

We have tried to keep the system simple and not too expensive. It does not need to be connected to the internet all the time to work. This is one of the example of how edge computing can help solve problems and create a safer environment for industrial motor operations. This system was a huge learning experience for our wiring hardware, calibrating sensors, programming in real time, troubleshooting issues when things don't go as expected.

How to improve this system. If we add TinyML the system can get better at predicting problems before they happen. We could also use LoRa to monitor the motor from away. The same basic design could be used to watch over motors in a factory. The way it is now this project shows that we can make a smart system without using really expensive parts. It is a starting point, for making something that can be used in industries.

REFERENCE

1. Fernando, G J O. 2023. Proactive Anomaly Detection Framework Utilizing Edge-Driven Ubiquitous IoT for Industrial Automation Systems. International Journal of Advanced Research in Engineering and Technology. (Base Paper)
2. Espressif Systems. (2022). ESP32 Series Datasheet. Espressif Systems Inc.
3. Adafruit Industries. (2021). ADXL345 Triple Axis Accelerometer Breakout Board Adafruit Learning System.
4. Allegro Micro Systems. (2013). ACS712 Fully Integrated, Hall Effect-Based Linear Current Sensor with 2.1 KVRMS Voltage Isolation. Datasheet.
5. Maxim Integrated. (2019). DS18B20 Programmable Resolution 1-Wire Digital Thermometer.
6. Datasheet.
7. Texas Instruments. (2020). DRV8833 Dual H-Bridge Motor Driver Data Sheet .
8. Smith, J.,&Lee, K. (2022). Multi-sensors fusion and edge computing for real time fault detection in industrial motors IEEE Transactions on Industrial Informatics, 18(4), 2456–2467.
9. Zhang L., Wang H.&Chen Y., 2021. Sensors and Actuators, 15(3), 112-125. IoT Based Predictive Maintenance System Vibration and Current Analysis for DC Motors
10. Kumar R., Sharma S. (2023). Edge-based machine learning approach for anomaly detection in industrial equipment using ESP32 International Journal of Embedded Systems, 12(2), 89-102.
11. Works for math. (2022, January 1) Predictive Maintenance Using MATLAB and Simulink [Online]. Available.