GERMAN - MONGOLIAN INSTITUTE FOR RESOURCES AND TECHNOLOGY

The present work was submitted to

the German-Mongolian Institute of Resources and Technology

Design and Implementation of an IoT-Based DC Motor Control System via Web or Mobile Application

Bachelor's Thesis

By

SUVD Enkhjargal

Study program: Mechatronics Engineer

Student ID: B2200508

1st Supervisor/Examiner: Ph.D. Young Suk Kim

2nd Supervisor/Examiner: M.Eng. Myagmarjav Bold

Ulaanbaatar/Nalaikh

2026

Statutory Declaration

Enkhjargal Suvd	B2200508
Last Name, First Name	Student ID Number

I hereby affirm in lieu of an oath that I provided the submitted bachelor thesis

DESIGN AND IMPLEMENTATION OF AN IOT-BASED DC MOTOR CONTROL SYSTEM VIA WEB OR MOBILE APPLICATION

I did not use any sources other than those stated. In case that the work is additionally submitted on a data medium, I declare that the written and the electronic form are completely identical. The work was not submitted in the same or similar form to any examination authority.

Ulaanbaatar/Nalaikh 2026.05.27	
Place, Date	Signature

Acknowledgement

I would like to express my sincere gratitude to my supervisors Prof. Kim Young Suk and Mr. Myagmarjav Bold for their guidance, support, and valuable feedback throughout the development of this thesis. Their insights greatly contributed to improving both the technical and analytical aspects of this work.

I would also like to thank the instructors, staff and faculty members of the German Mongolian Institute for Resources and Technology (GMIT) for providing me with the knowledge and skills necessary to complete this project. Special thanks go to my friends for their encouragement and assistance during the challenging stages of this research, and to my family for their continuous support and motivation throughout my academic journey.

Finally, I acknowledge all the usefulness and tools that made this project possible, including the hardware platforms and software libraries used in the development of the feedback-controlled motor system.

Table of Contents

List of Figures 7

List of Tables 7

Abstract 8

1. Introduction 9

1.1 Background 9

1.2 Problem statement 10

1.3 Objectives of the study 11

2. State of the Art 12

2.1 DC Motor Operation and PWM Control 12

2.2 Traditional DC Motor Control Methods 14

2.3 IoT-Based Motor Monitoring and Control Systems 15

2.4 Energy Consumption and Efficiency in Motor Systems 16

2.5 IoT Platforms, User Interfaces, and Data Management 18

2.6 Research Gap and Justification of the Study 19

3. Materials and Methods 20

3.1 System Architecture 20

3.1.1 Hardware Layer 21

3.1.2 Communication and cloud layer 22

3.1.3 User Interface Layer 23

3.2 Hardware Components 23

3.3 Software and firmware development 25

3.3.1 Microcontroller Programming 25

3.3.2 IoT Integration and Cloud Connectivity 26

3.4 System Logic and Feedback Mechanisms 28

3.4.1 Speed and Direction Control Logic 28
3.4.2 Stall Detection Algorithm 28
3.4.3 Energy Calculation and Management 29
3.5 User Interface Design 29
3.5.1 Blynk Data stream 29
3.5.2 Dashboard 30
3.6 Experimental and Testing Procedures 31
3.6.1 Speed and Direction Control Verification 31
3.6.2 Fault Simulation and Stall Detection Test 32
3.6.3 Energy Monitoring and Conservation Validation 32
4. Results and Analysis 33
4.1 Motor Speed and Direction Control Performance 33
4.1.1 PWM to Speed Mapping 33
4.1.2 Directional Control Response 34
4.2 Stall Detection and System Safety Response 35
4.3 Energy Monitoring Accuracy 37
4.3.1 Sensor Validation 37
4.3.2 Cumulative Energy Tracking 38
4.4 Automated Energy Management Performance 38
4.4.1 Energy-Saving Mode Activation 39
4.4.2 Maximum Limit Shutdown and Timer 40
5. Discussion 41
5.1 Interpretation of Results 41
5.1.1 Control Responsiveness 41
5.1.2 Effectiveness of Stall Detection 41

5.1.3 Accuracy of Energy Management 41
5.2 Comparison with State of the Art 42
5.3 Practical Relevance and Applications 43
5.4 Future Research and Improvements 43
6. Conclusion 44
7. References 46
8. Appendices 48
Appendix A. Data Stream of Blynk IoT 48
Appendix B. Measured Data 49
Appendix C. Arduino Code 50

List of Figures

Figure 1. Equivalent Electromechanical Model of a DC Motor 12
Figure 2. . Pulse Width Modulation (PWM) signal 13
Figure 3. H-bridge motor driver circuit connected to a DC motor 13
Figure 4. IoT system architecture [10] 16
Figure 5. A flow chart of the Energy-Aware Feedback Loop [11] 17
Figure 6. Control system architecture of the IoT-based DC motor control platform 21
Figure 7. Circuit diagram of simulation circuit illustrated by Fritzing 21
Figure 8. Simulation schematic of ESP32 motor control system using Fritzing 22
Figure 9. Voltage and Current Sensor calibrations on Arduino IDE 25
Figure 10. Measurement of current and power in Arduino IDE 26
Figure 11. Energy saving mode, feedback system in Arduino IDE 26
Figure 12. Connection of ESP32 with Blynk IoT 27
Figure 13. Flowchart of the main execution loop of the firmware 27
Figure 14. Design of the data stream on Blynk IoT 30
Figure 15. Design of the dashboard on Blynk IoT 31
Figure 16. The user's remote input percentage vs Measured Voltage 34
Figure 17 (a) (b). Side by side comparison of direction control 34
Figure 18. Blynk Interface before and after motor stall 36
Figure 19. Experimental setup showing DC motor stall condition 36
Figure 20. Photograph of implemented ESP32 motor control system 37
Figure 21. Cumulative energy consumption (Wh) tracked over time 38
Figure 22. System response showing the automated reduction of the PWM duty cycle from 100% to 50% immediately upon crossing the energy warning threshold.. 39
Figure 23 (a) (b). A screenshot of the Blynk dashboard during the energy save mode 40

List of Tables

Table 1. Hardware components 23
Table 2. DC motor control logic 28
Table 3. Comparison of Sensor Measurements and Reference Values for Voltage, Current, and Power 37

Abstract

This dissertation presents the design and implementation of an Internet of Things (IoT) based feedback-controlled DC motor system using an ESP32 microcontroller. The system integrates real-time sensing, control, and data analysis to evaluate the performance of the motor and improve operational reliability under varying conditions.

The motor is controlled through a motor driver using pulse width modulation (PWM), allowing adjustable speed and direction. A custom web-based interface has been developed to provide visualization of the recorded data and control of the motor parameters including the state of operation, voltage, current, power, speed, and energy consumption. The interface was intended to replace the third-party platforms and demonstrate an independent Internet of Things (IoT) monitoring solution.

A feedback mechanism is then implemented to detect the abnormal conditions, specifically motor stalling caused by external mechanical resistance. By analyzing the relationship between the current and voltage, the system identifies a stall condition when current exceeds a certain threshold while the voltage remains high. Upon detection of the anomalies, the system generates a response signal and can optionally stop the motor from preventing damage.

When the energy usage exceeds a warning level, the system automatically reduces PWM duty cycle to conserve energy. If the maximum energy limit is reached, motor stops or slows down to prevent unnecessary power consumption. Moreover, a time-based control feature is implemented, allowing the motor to operate for a specified duration before automatically stopping.

The results demonstrate the relationship between PWM input and motor performance parameters, as well as the effectiveness of the proposed feedback mechanisms in regulating energy consumption and preventing abnormal operation. The developed system provides a low-cost and efficient solution for intelligent motor monitoring and control, with potential applications in automation and smart systems. Future work may include the integration of advanced control algorithms and more precise sensing techniques.

1. Introduction

1.1 Background

Direct current (DC) motors are widely used in engineering applications, robotics, automation systems, and small-scale industrial equipment due to their simple construction and cost effectiveness. Their speed and direction can be conveniently controlled using pulse width modulation (PWM), where the effective voltage applied to the motor is adjusted by the duty cycle of a digital signal. This switching approach allows for the efficient control of motor performance without significant energy losses [2].

In practical applications, however, the behavior of the DC motor is influenced by several other factors that are non-ideal. The motor driver module L298N used in this study introduces voltage drops due to internal switching components, resulting in a lower effective voltage at the motor in comparison to the supply voltage [6]. Furthermore, variation in mechanical load and supply conditions can significantly affect motor performance, making it difficult to predict its behavior without direct measurement.

To address these challenges, modern motor control systems increasingly incorporate sensing and feedback mechanisms. By measuring electrical parameters such as voltage and current, it's possible to estimate power consumption and gain insight into the operating condition of the motor [8]. In this work, voltage across the motor terminals is measured using voltage sensor modules connected to both sides of the motor, while current is measured using a current sensor module placed in series with the supply. These measurements are processed by an ESP32 microcontroller, enabling real-time evaluation of motor performance [5].

Moreover, feedback control plays a crucial role in improving system reliability. By monitoring real-time values and comparing them with expected behavior, the system can respond to disturbances such as load changes or external forces [9]. The ESP32 microcontroller is used as the central control unit in this study due to its analog input capability and wireless communication features, allowing both real-time processing and remote monitoring [5, 10].

1.2 Problem statement

Despite their widespread use, many low-cost DC motor control systems operate with limited monitoring and feedback capabilities. In such systems, control is typically restricted to adjusting motor speed or direction, without providing information about the motor's actual operating condition. As a result, these systems often function without awareness of performance, efficiency, or potential faults.

In practical applications, DC motors are frequently subjected to varying load conditions, which directly influence both electrical and mechanical behavior. An increase in load leads to higher current draw, resulting in increased power consumption and potential overheating. In extreme cases, such as a stall condition, the back electromotive force (back-EMF) collapses, causing a significant rise in current. Without real-time monitoring of voltage and current, these conditions may remain undetected, increasing the risk of damage to the motor or driver [9]. Additionally, voltage drops across components such as the L298N motor driver can reduce the effective voltage supplied to the motor, further degrading performance in ways that are not immediately observable.

Another key limitation of basic motor control systems is the absence of energy awareness. While instantaneous power provides a snapshot of operating conditions, it does not capture the total energy consumed over time. Consequently, such systems are unable to evaluate efficiency or optimize energy usage during operation. Moreover, many conventional implementations lack the ability to respond autonomously to changing conditions, relying instead on fixed user inputs [11].

The lack of remote monitoring and data logging further limits system functionality. In modern engineering applications, particularly in automation and distributed environments, it is essential to monitor and control systems remotely while maintaining access to historical performance data for analysis and diagnostics [12]. Systems without IoT integration require direct physical interaction and offer no means of long-term data tracking.

To address these limitations, there is a need for an integrated solution that combines real-time sensing, data processing, and intelligent control. The system proposed in this study measures voltage and current continuously, calculates both instantaneous power and cumulative energy consumption, and enables remote monitoring through an IoT platform. In

addition, it implements feedback mechanisms that adjust motor operation based on predefined conditions, thereby improving efficiency, enhancing reliability, and providing deeper insight into motor behavior under varying operating conditions.

1.3 Objectives of the study

The main objective of this study is to design and implement a DC motor monitoring and feedback control system using an ESP32 microcontroller, integrating real-time sensing, analyzing, and control to improve system awareness and reliability.

To achieve this, the study focuses on measuring key electrical parameters of the motor during operation. Voltage across the motor terminals is measured using dual voltage sensor modules, while current is measured using a hall-effect-based or shunt-based current sensor. The measurements are used to calculate instantaneous power and reflect the motor's operating condition under different input and load scenarios. Tracking its energy consumption allows for a more comprehensive assessment of motor performance, particularly in applications where efficiency and power usage are critical [11]. The collected data is also used to analyze the PWM input and motor behavior, including its effect on voltage, current, and power [8].

A key focus of the system is the implementation of feedback mechanisms. The system is designed to detect abnormal conditions such as excessive energy consumption, mechanical stalling, or unexpected behavior, and respond accordingly [9]. This involves automatically adjusting PWM signals or stopping the motor when predefined conditions are met, enabling safer and more efficient operation. It is also used to incorporate time-based control, allowing the motor to operate a specified duration before automatically stopping, demonstrating the system's ability to function autonomously based on user-defined conditions.

Furthermore, the system integrates IoT-based monitoring capabilities using the ESP32 platform, enabling real-time visualization of the parameters through a user interface [10]. The system activity, user interactions, and motor states are also recorded to provide basic logging functionality for analysis and tracking. Overall, the objective of the study is to develop a flexible and intelligent motor monitoring system that combines measurement, analysis, feedback, and communication within a single platform, demonstrating its potential use in modern automation and smart system applications.

2. State of the Art

2.1 DC Motor Operation and PWM Control

A Direct Current (DC) motor is an electromechanical device that converts electrical energy into continuous rotational motion through the interaction between the stator's magnetic field and the current flowing through the rotor armature. The electrical characteristics of a brushed DC motor can be modeled using an equivalent circuit consisting of an armature resistance, armature inductance, and an induced back electromotive force (back-EMF). In steady-state operation, the relationship between the applied terminal voltage and the motor's electrical parameters is expressed as:

$$V_t = E_b + I_a R_a$$

where V_t is the applied terminal voltage, I_a is the armature current, R_a is the armature resistance, and E_b is the back-EMF [1]. The back-EMF is generated as the armature rotates through the magnetic field and is directly proportional to the motor's angular velocity:

$$E_b = K_e \omega$$

where K_e is the motor's voltage constant and ω is the rotational speed [1]. From these equations, it is evident that the rotational speed of a DC motor is highly dependent on the applied terminal voltage V_t and inversely affected by the mechanical load, which draws more current I_a and causes a larger voltage drop across the armature resistance.

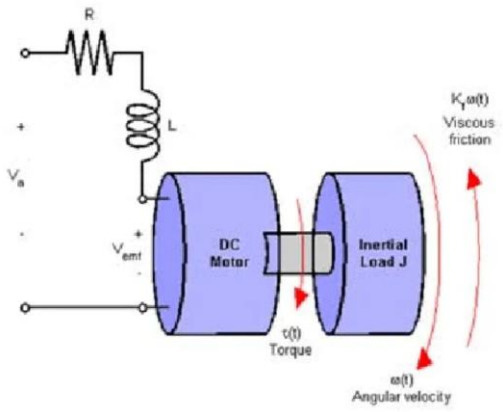


Figure 1. Equivalent Electromechanical Model of a DC Motor

To regulate the speed of a DC motor electronically, Pulse Width Modulation (PWM) has become the industry standard. Instead of using inefficient linear regulators to step down the DC voltage, PWM generates a high-frequency square wave that rapidly switches the power supply to the motor on and off [2]. The effective voltage delivered to the motor is controlled

by altering the duty cycle, which is defined as the ratio of the "on" time to the total period of the signal:

$$D = \frac{t_{ON}}{T} * 100\%$$

where D is the duty cycle, t_{ON} is the duration the signal is high, and T is the total period of the switching cycle. The average voltage V_{avg} applied to the motor is directly proportional to the duty cycle:

$$V_{avg} = D * V_s$$

where V_s is the source voltage. By adjusting the duty cycle, the system can smoothly control the motor speed with minimal power dissipation in the switching components [4].

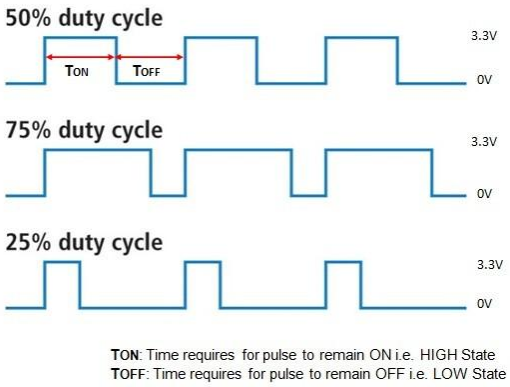


Figure 2. . Pulse Width Modulation (PWM) signal

In modern architectures, these PWM signals are generated digitally by hardware timers on microcontrollers such as the ESP32 [5]. Because a microcontroller cannot supply the high

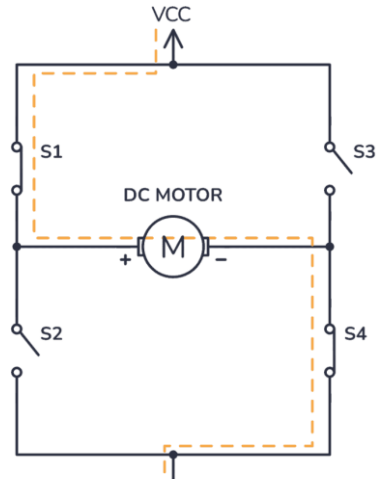


Figure 3. H-bridge motor driver circuit connected to a DC motor

currents required to drive a motor directly, the low-voltage PWM logic signals are fed into a motor driver, typically utilizing an H-bridge configuration like the L298N [6]. The H-bridge consists of four solid-state switches that not only amplify the current but also allow the system to reverse the motor's rotational direction by alternating the active diagonal pairs of switches, effectively reversing the voltage polarity across the motor terminals [7].

The relationship between the PWM voltage, current, and speed also provides critical diagnostic data under varying mechanical loads. During normal operation, the current drawn is proportional to the required torque. However, if the motor encounters severe mechanical resistance and stalls, the rotational speed drops to zero (ω=0). Consequently, the back-EMF also drops to zero (Eb=0). Under this stall condition, the equation simplifies to:

$$I_{stall} = \frac{V_t}{R_a}$$

Because the armature resistanceR_a is typically very low, a stall condition causes a severe and sudden spike in the current draw, even while the applied PWM voltage remains constant [8]. By continuously monitoring this voltage and current relationship, advanced IoT-based control systems can accurately estimate performance, track energy consumption, and implement critical safety feedback loops to cut power before the excessive stall current damages the motor or driver circuit [8].

2.2 Traditional DC Motor Control Methods

Traditional DC motor control systems relied primarily on localized analog and early digital control methods. In such systems, motor speed and direction were adjusted using components such as switches, relays, and potentiometers, or through basic microcontroller-based PWM control. While these approaches were effective and widely used, they were typically limited to local operation and lacked integrated communication capabilities [1].

Although automated control systems, such as those used in household appliances and industrial machinery could operate without continuous human intervention, they generally functioned as closed, standalone systems. These systems did not provide remote accessibility or centralized monitoring, and data such as power consumption and system performance was often not available in real time [8].

Another limitation was the lack of advanced monitoring and adaptive protection. In the event of a mechanical stall, the reduction of back-EMF would lead to a significant increase in

current draw. While some systems included basic protection mechanisms such as fuses or circuit breakers, they lacked intelligent feedback capable of dynamically responding to operating conditions [9].

The integration of Internet of Things (IoT) technologies enhances these existing control systems by enabling connectivity, real-time data acquisition, and remote interaction. Microcontrollers such as the ESP32 [5] can generate PWM signals while simultaneously transmitting operational data to cloud platforms.

This evolution introduces several key advantages. First, remote accessibility allows users to monitor and control motor operation from mobile or web-based interfaces. Second, real-time telemetry enables continuous tracking of voltage, current, and energy consumption, supporting more efficient system operation. Finally, intelligent feedback mechanisms allow the system to respond dynamically to abnormal conditions, such as reducing speed or shutting down during a stall, thereby improving reliability and protection [6].

2.3 IoT-Based Motor Monitoring and Control Systems

The integration of the Internet of Things (IoT) has fundamentally transformed DC motor control from isolated, physical interfaces into globally accessible, data-driven platforms. Modern microcontrollers equipped with embedded Wi-Fi, such as the ESP32, serve as the foundation for this shift, providing both the hardware-level capabilities necessary to generate PWM signals and the network connectivity required to communicate with cloud servers [5].

In a typical IoT motor control architecture, the microcontroller bridges the physical and digital domains. On the physical side, it drives the motor through an H-bridge driver [6] and collects data from localized sensors, such as current and voltage monitors or rotary encoders [8]. On the digital side, it transmits this localized telemetry over the internet to a centralized platform, allowing for continuous remote observation of the motor's state.

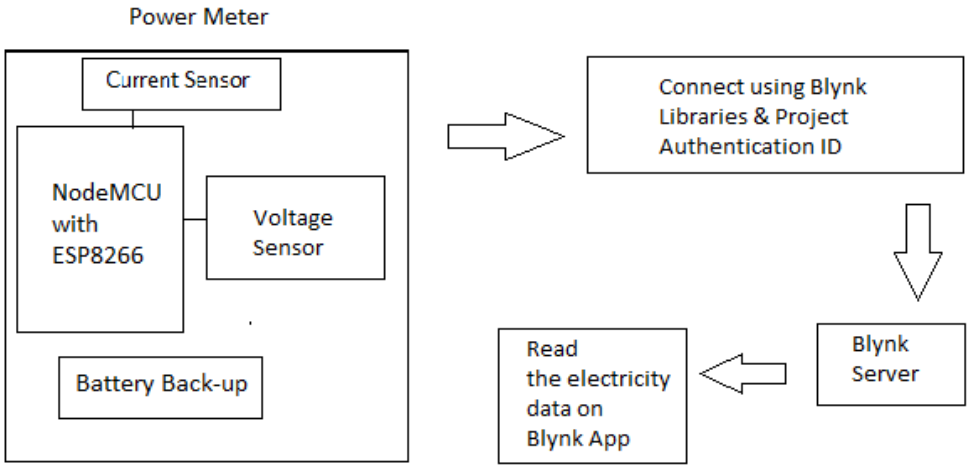


Figure 4. IoT system architecture [10]

The use of cloud-based platforms and mobile applications, such as Blynk IoT, has become a highly effective approach for user interaction in these systems. These platforms act as intermediaries, receiving sensor data from the microcontroller and instantly reflecting it on customizable mobile dashboards. Conversely, when a user inputs a command via the app, such as adjusting a slider to change the motor's speed or toggling its direction, the platform transmits that command back to the microcontroller, which updates the PWM duty cycle accordingly [10]. This bidirectional communication allows for sophisticated, real-time control without requiring the operator to be physically near the machine.

2.4 Energy Consumption and Efficiency in Motor Systems

As automated systems become more reliant on limited power sources, energy efficiency has emerged as a critical concern in electromechanical design. While Pulse Width Modulation (PWM) offers high theoretical efficiency by minimizing heat dissipation in the switching components, the total energy consumed by a DC motor remains highly dependent on its operating speed and mechanical load [3].

The integration of IoT monitoring has introduced the capability for real-time energy profiling, which was previously unavailable in basic open-loop systems. By integrating voltage and current sensors, modern systems can continuously measure the instantaneous electrical power and calculate cumulative energy consumption over time. The electrical power P at any given moment is calculated as:

$$P(t) = V(t) \times I(t)$$

where V(t) is the instantaneous voltage and I(t) is the instantaneous current. The total energy E consumed over a specific time period is the integral of power over time:

$$E = \int P\, dt$$

In a microcontroller environment like the ESP32, this continuous integration is typically approximated discretely by summing the power samples taken at regular intervals [8]. This level of monitoring allows operators to visualize exactly how the motor's power draw fluctuates under different loads or PWM duty cycles.

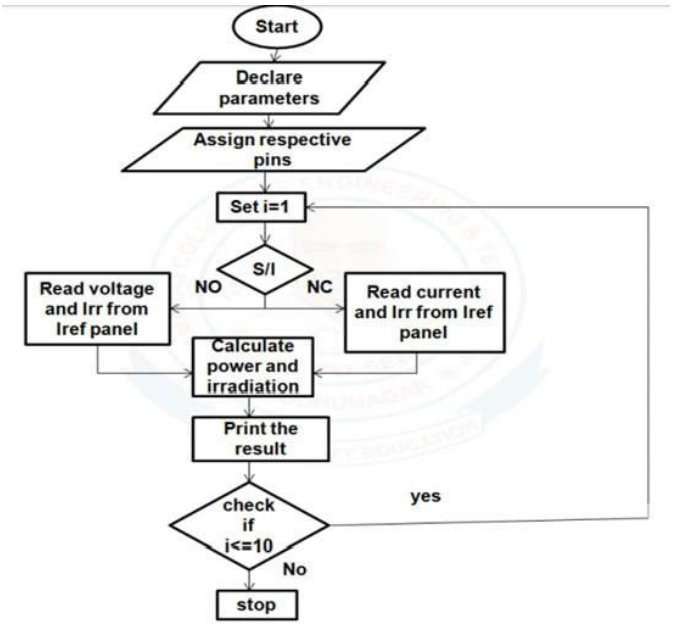


Figure 5. A flow chart of the Energy-Aware Feedback Loop [11]

More importantly, real-time energy telemetry enables active, intelligent feedback loops. Recent research into IoT systems highlights the use of microcontrollers to enforce load-shedding and power-saving algorithms based on live consumption data [11]. In the context of DC motor control, this means the system can be programmed to autonomously respond when energy usage exceeds a predetermined threshold. For example, the system can automatically reduce the PWM duty cycle to lower the motor speed and thereby reduce the current draw or execute a scheduled shutdown using an integrated timer.

Additionally, feedback mechanisms play a vital role in protecting the system from damaging operating states. As established earlier, if a motor stalls, the back-EMF drops to zero and the current spikes. By continuously comparing the drawn current against a safe threshold while the voltage remains high, the controller can detect this stall anomaly and immediately halt the motor, preventing thermal damage to the coils and the driver circuit [9].

By incorporating energy calculation feedback, a timer, and an automated energy-saving mode, the system developed in this thesis aligns with the current literature's emphasis on intelligent, self-regulating energy management. This transition from passive operation to active, energy-aware control represents a significant advancement in the reliability and sustainability of IoT-based electromechanical systems.

2.5 IoT Platforms, User Interfaces, and Data Management

The true value of an IoT-enabled motor control system is realized through the user interface and the underlying data management infrastructure. By shifting control from local, physical switches to digital platforms, IoT architectures make data both visible and actionable in real-time.

In a typical implementation, the data collected by the microcontroller (such as current, voltage, speed, and power) must be securely formatted and transmitted to a cloud server. While several protocols exist, MQTT (Message Queuing Telemetry Transport) and HTTP are the most prevalent for low-power IoT devices. Once the data reaches the cloud, it must be displayed through a Graphical User Interface (GUI) that allows the operator to understand the system state at a glance.

Platforms like Blynk have become increasingly popular for this purpose because they provide a highly customizable dashboard that bridges the gap between hardware and software. Using Blynk, developers can create virtual switches, sliders, and gauges on a mobile or web interface without writing extensive backend code. When a user interacts with the Blynk dashboard—for instance, sliding a bar to change the motor speed—the platform sends an immediate API call back to the ESP32, which then adjusts the PWM duty cycle accordingly [12].

In the context of the system proposed in this thesis, the interface is designed to replace third-party dependence by demonstrating an independent, custom web-based monitoring

solution. This dashboard is specifically built to visualize critical parameters, the state of operation, voltage, current, power, speed, and energy consumption.

Furthermore, data management in these systems is not just about real-time display; it is about historical tracking. By logging energy consumption data over time, the system can identify usage patterns and trigger automated interventions, such as the energy-saving mode discussed previously [11]. This combination of a responsive web interface and intelligent data management allows the operator to transition from simply running a motor to actively managing its performance and energy footprint.

2.6 Research Gap and Justification of the Study

While the existing literature demonstrates substantial progress in both DC motor control and IoT applications, these fields have largely evolved along parallel but separate tracks. A review of current research reveals that most studies focus on either optimizing motor control algorithms (such as advanced PID tuning or predictive control) in isolation, or they focus on basic IoT connectivity, allowing users to remotely turn devices on and off using platforms like Blynk [10].

A significant gap remains in the integration of these two domains. Specifically, there is a lack of compact, low-cost systems that combine real-time motor control, independent web-based telemetry, active stall protection, and autonomous energy management into a single, cohesive architecture. Many existing IoT motor applications are open-loop systems; they allow remote speed adjustment via PWM but fail to monitor the electrical consequences of those adjustments [8]. If the motor encounters a physical obstruction and stalls, these basic systems cannot detect the resulting current spike, potentially leading to thermal damage or excessive power drain [9].

Furthermore, while smart grid literature heavily emphasizes energy tracking and load shedding, this level of energy-aware control is rarely applied to individual DC motor systems at the device level. Standard prototypes rarely calculate instantaneous power or cumulative energy consumption, and they do not feature automated energy-saving modes that react when consumption limits are breached.

This thesis directly addresses this gap by presenting a fully integrated, closed-loop IoT motor control system. By utilizing an ESP32 microcontroller, the proposed system not only enables

remote PWM control via a custom web interface but also continuously monitors voltage and current. This allows the system to calculate energy consumption dynamically and implement intelligent feedback protocols. Specifically, the system can autonomously detect stall conditions by analyzing current-voltage anomalies and can enforce energy-saving modes or time-based shutdowns when predefined limits are reached.

Therefore, the development of this system is highly justified. It provides a comprehensive, low-cost solution for intelligent motor monitoring, demonstrating that advanced fault protection and energy management can be successfully embedded into independent IoT control architectures. This research contributes a practical framework that enhances the operational reliability and sustainability of electromechanical systems in smart applications.

3. Materials and Methods

3.1 System Architecture

The proposed IoT-based DC motor control system is designed as a closed-loop architecture that integrates physical hardware, wireless communication, and remote user interfaces. The architecture ensures that user commands are executed seamlessly while real-time electrical telemetry is continuously monitored and processed for safety and energy management.

The system is structured into three distinct layers: the Hardware Layer, the Communication/Cloud Layer, and the User Interface Layer.

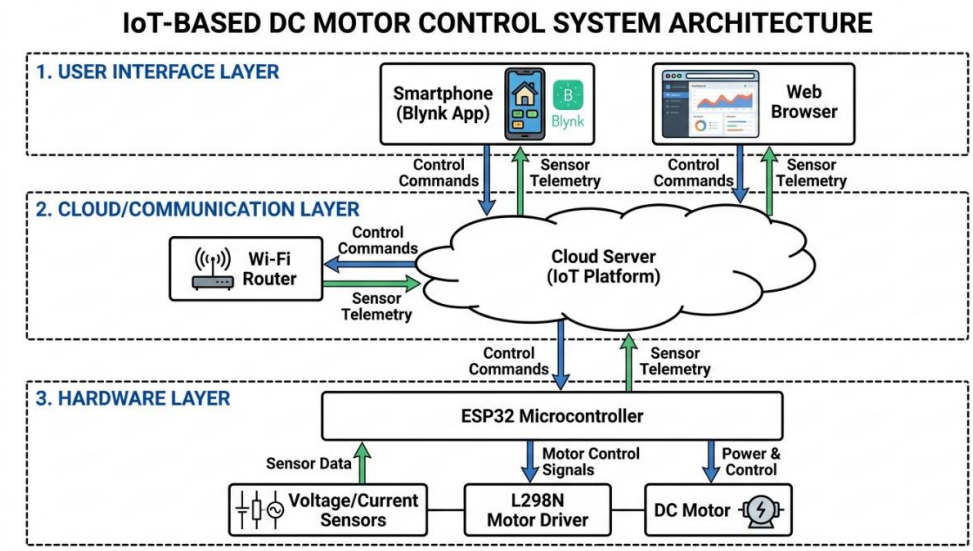


Figure 6. Control system architecture of the IoT-based DC motor control platform [13]

3.1.1 Hardware Layer

The hardware layer is responsible for the physical execution of commands and the collection of operational data. At the center of this layer is the ESP32 microcontroller, which acts as the primary data processing and control unit.

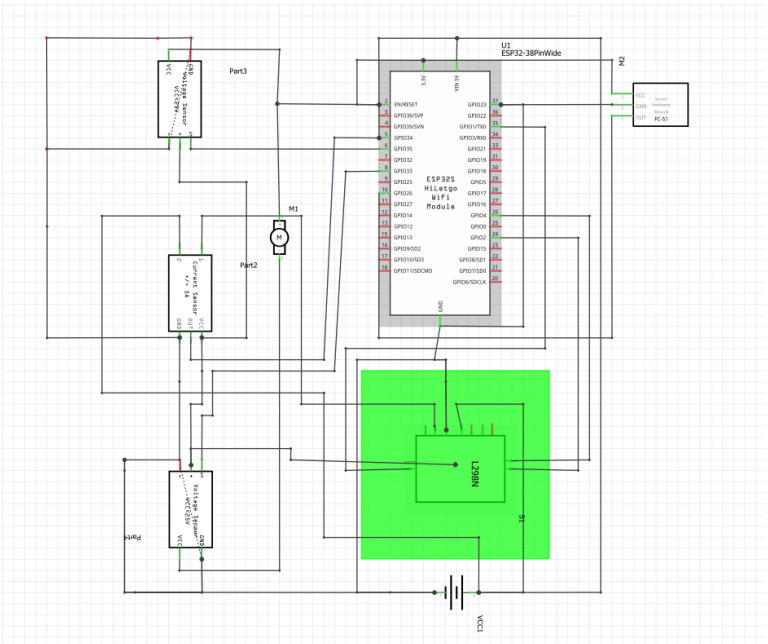


Figure 7. Circuit diagram of simulation circuit illustrated by Fritzing

-Actuation: The ESP32 generates Pulse Width Modulation (PWM) signals and directional logic pins, which are sent to the motor driver (e.g., L298N). The motor driver amplifies these signals to physically drive the DC motor.

-Sensing: Simultaneously, voltage and current sensors monitor the power lines feeding the motor. This sensor data is continuously fed back into the analog or digital input pins of the ESP32.

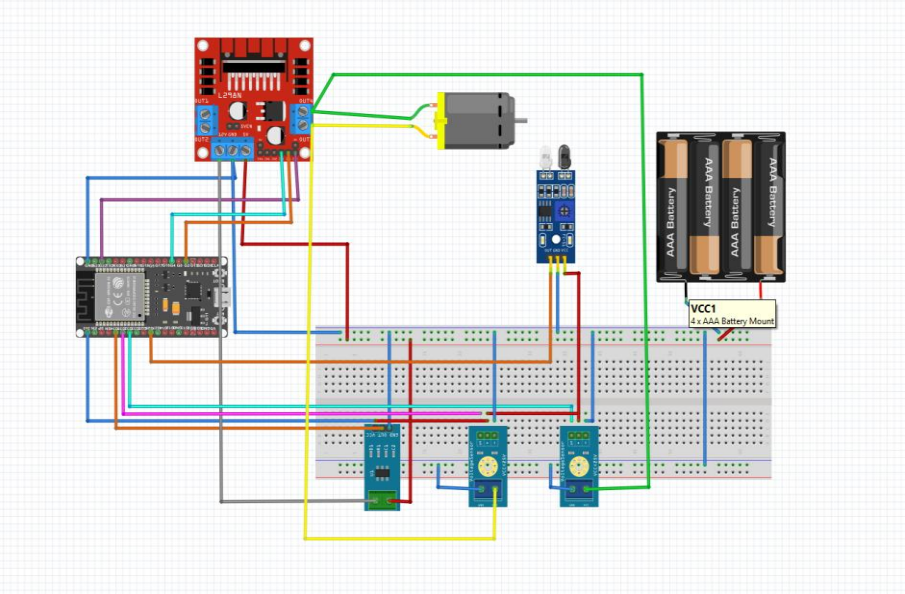


Figure 8. Simulation schematic of ESP32 motor control system using Fritzing

-Local Processing: The microcontroller executes local logic to calculate instantaneous power and cumulative energy. It also runs the safety algorithms, such as comparing the current draw against predefined thresholds to detect stall conditions or triggering the energy-saving mode when consumption limits are breached.

3.1.2 Communication and cloud layer

The communication layer bridges the physical hardware with the digital interface. Leveraging the ESP32's built-in Wi-Fi capabilities, the microcontroller connects to a local wireless network to access the internet. The ESP32 transmits the processed sensor data (voltage, current, power, energy, and motor status) to a centralized cloud server (such as the Blynk IoT server or a custom database).The cloud server also queues incoming user commands (such as speed adjustments or timer settings) and pushes them down to the ESP32 in real time.

3.1.3 User Interface Layer

The user interface layer is the point of interaction for the operator, replacing traditional physical control panels.

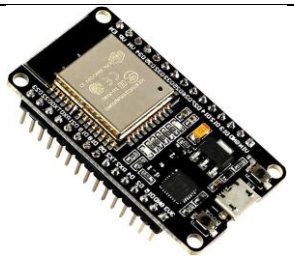
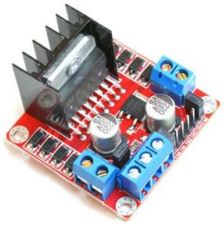
-Monitoring: The operator can access the system via a smartphone application (Blynk) or an independent web-based dashboard. These interfaces display live telemetry using virtual gauges, graphs, and status indicators.

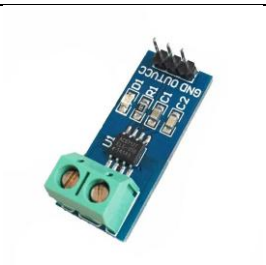
-Control: The user can interact with virtual sliders and buttons to set the desired motor speed, reverse direction, initiate the operational timer, or reset energy consumption metrics.

This tripartite architecture ensures a highly responsive system where hardware constraints (such as stalls) can be managed locally by the microcontroller for immediate safety, while broader operational data is securely logged and visualized for remote user management.

3.2 Hardware Components

Table 1. Hardware components

Name	Quantity	Function	Component Illustration
ESP32 Microcontroller	1	Main control unit; processes sensor data, generates PWM, and handles communication.	
L298N Motor Driver	1	Controls motor speed and direction by amplifying ESP32 signals.	

DC Motor	1	Converts electrical energy into rotational motion.	
Voltage Sensors	2	Measure motor terminal voltages for calculating actual motor voltage.	
Current Sensor (ACS712)	1	Measures current flowing to the motor for power and protection analysis.	
LM393 IR speed sensor	1	Detects rotation using reflected IR light to estimate motor speed.	
Battery 1.5V	4	Provides power supply to the motor system.	
Battery socket	1	Holds and connects batteries to the circuit securely.	

3.3 Software and firmware development

3.3.1 Microcontroller Programming

The firmware for the ESP32 is developed using the Arduino Integrated Development Environment (IDE) with C++ [5]. The code structure is designed around a continuous loop that executes three primary tasks: reading sensor telemetry, executing local safety logic, and controlling the motor output.

First, the microcontroller communicates with the voltage and current sensors via the I2C (Inter-Integrated Circuit) protocol. It reads the instantaneous voltage and current values at predefined intervals. These raw values are converted into standard units (Volts and Amperes) and multiplied to calculate the instantaneous power.

Second, the firmware processes incoming control variables (such as the desired speed and direction) and maps them to the appropriate output pins. The speed variable (typically ranging from 0 to 100%) is mapped to an 8-bit PWM duty cycle value (0 to 255) using the ESP32's built-in LEDC (LED Control) PWM API, which is then sent to the L298N motor driver. The directional variables trigger standard digital HIGH/LOW signals to the driver's logic pins [6].

Finally, the microcontroller runs local safety and energy logic. If the calculated energy surpasses the predefined limit, the firmware overrides the user's speed setting and artificially limits the maximum allowable PWM duty cycle. Similarly, if a severe current spike is detected, the firmware triggers the stall protection routine, immediately setting the PWM duty cycle to zero to cut power to the motor [9].

```
38  // Voltage sensor calibration factors
39  float sensorScaleA = 6.6;
40  float sensorScaleB = 6.6;
41
42  // ACS712 current sensor calibration
43  // 5A module: 0.185, 20A module: 0.100, 30A module: 0.066
44  float currentSensitivity = 0.122;
45  float currentOffset = 1.46;   // Zero-current offset voltage
46
```

Figure 9. Voltage and Current Sensor calibrations on Arduino IDE

```
void sendMeasurements()
{
  float voltageA = readVoltageSensor(voltagePinA, sensorScaleA);
  float voltageB = readVoltageSensor(voltagePinB, sensorScaleB);

  float motorVoltage = abs(voltageA - voltageB);

  float motorCurrent = readCurrentSensor();

  float motorPower = motorVoltage * motorCurrent;


  // Smooth readings
  filteredMotorVoltage = 0.75 * filteredMotorVoltage + 0.25 * motorVoltage;
  filteredCurrent = 0.75 * filteredCurrent + 0.25 * motorCurrent;
  filteredPower = 0.75 * filteredPower + 0.25 * motorPower;
```

Figure 10. Measurement of current and power in Arduino IDE

```
375  void energyFeedback()
376  {
377    if (!energyFeedbackEnabled) return;
378    if (energyLimit_Wh <= 0) return;
379    if (motorDirection == 0 || speedVal == 0) return;
380    if (energy_Wh >= energyLimit_Wh)
381    {
382      speedVal = 80;   // energy-saving speed
383      analogWrite(ENB, speedVal);
384
385      Blynk.virtualWrite(V3, speedVal);
386      Blynk.virtualWrite(V19, speedVal);
387      Blynk.virtualWrite(V11, 1);
388
389      energyLimitActive = true;
390      return;
391    }
392
393    if (energy_Wh >= energyWarning_Wh)
394    {
395      speedVal -= 10;
396      speedVal = constrain(speedVal, 80, 255);
397      analogWrite(ENB, speedVal);
398
399      Blynk.virtualWrite(V3, speedVal);
400      Blynk.virtualWrite(V19, speedVal);
401      Blynk.virtualWrite(V11, 1);
402
403      energyWarningActive = true;
404    }
405    else
```

Figure 11. Energy saving mode, feedback system in Arduino IDE

3.3.2 IoT Integration and Cloud Connectivity

To enable remote monitoring and control, the ESP32 firmware integrates the Blynk IoT library. Upon booting, the ESP32 uses its embedded Wi-Fi module to connect to the local network using hardcoded Service Set Identifier (SSID) and password credentials [5].

Once internet access is established, the device authenticates with the Blynk cloud server using a unique authorization token. The firmware uses virtual pins to handle bidirectional communication.

- **Data Transmission (Upstream):** The ESP32 pushes the calculated values for voltage, current, power, and cumulative energy to specific virtual pins on the Blynk server at regular intervals (e.g., every 1 second). This ensures the dashboard remains updated without flooding the network.

```
#define BLYNK_PRINT Serial

#define BLYNK_TEMPLATE_ID "TMPL6Q5amKcQ_"
#define BLYNK_TEMPLATE_NAME "PRL"
#define BLYNK_AUTH_TOKEN "2ozSMQmjDoDJKWKGr56TGcUxQ97SFpYl"

#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>

char auth[] = BLYNK_AUTH_TOKEN;
char ssid[] = "Prlll";
char pass[] = "nopassword";
```

Figure 12. Connection of ESP32 with Blynk IoT

Command Reception (Downstream): Using the BLYNK_WRITE() functions, the ESP32 actively listens to virtual pins linked to the dashboard sliders and buttons. When a user changes the speed slider or toggles the direction button on the app, the server pushes the new integer value to the ESP32, which immediately updates the motor control variables in the main loop [12].

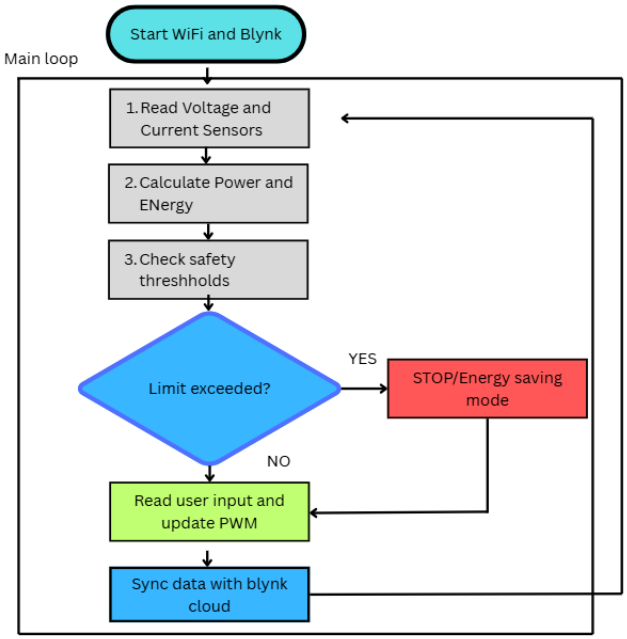


Figure 13. Flowchart of the main execution loop of the firmware

3.4 System Logic and Feedback Mechanisms

3.4.1 Speed and Direction Control Logic

The fundamental control of the motor relies on mapping user inputs from the IoT interface to physical electrical signals. When the user sets a desired speed via the dashboard slider, the value (typically a percentage from 0 to 100) is transmitted to the ESP32. The microcontroller then maps this percentage to an 8-bit PWM resolution (0 to 255) [2].

Directional control is handled using two digital logic pins connected to the L298N driver (IN1 and IN2). To rotate the motor clockwise, IN1 is set HIGH and IN2 is set LOW. To reverse the direction to counter-clockwise, the logic is inverted (IN1 LOW, IN2 HIGH). If both pins are set to the same state (both LOW or both HIGH), the driver actively brakes or free-wheels the motor [6].

Table 2. DC motor control logic

IN1	***IN2***	***Motor State***
0	*0*	*Stop*
0	*1*	*Clockwise Rotation*
1	*0*	*Counter-Clockwise Rotation*
1	*1*	*Stop (Brake)*

3.4.2 Stall Detection Algorithm

To protect the system from mechanical jams, a stall detection algorithm operates continuously in the microcontroller's main loop. The logic is based on the inverse relationship between motor speed and current draw. Under normal conditions, the back-EMF generated by the spinning rotor limits the current. However, if the motor stalls, back-EMF drops to zero, and the motor draws its maximum stall current [9].

The algorithm checks two conditions simultaneously, the PWM state; whether the microcontroller actively trying to drive the motor (i.e., is the PWM duty cycle significantly greater than zero) and the measured current exceeding the predefined maximum safe operating threshold.

If the PWM signal is high but the current spikes above the safe threshold for a sustained duration (to filter out brief inrush currents during startup), the microcontroller flags a "Stall Anomaly." The system immediately executes an emergency override, forcing the PWM duty

cycle to zero and cutting power to the motor to prevent thermal damage to the coils and the H-bridge [6]. A notification is also pushed to the web interface to alert the user of the fault.

3.4.3 Energy Calculation and Management

The energy management logic transforms the system from a passive actuator into an energy-aware device. The system calculates instantaneous power P by multiplying the sampled voltage V and current I. To calculate the cumulative energy E in Watt-hours (or Joules), the system integrates the power over time by taking the average power between the current and previous sample and multiplying it by the elapsed time Δt [8]:

$$E_{total} = E_{previous} + \left(P_{instantaneous} \times \frac{\Delta t}{3600}\right)$$

*(*Assuming Δt is measured in seconds and energy is calculated in Watt-hours).

This cumulative energy value is continuously compared against a predefined "Warning Level" and a "Maximum Energy Limit."

-Warning Level: If E_{total} exceeds this threshold, the ESP32 automatically enters an energy-saving mode. It overrides the user's input and scales down the maximum allowable PWM duty cycle (e.g., capping it at 50%), thereby reducing the motor speed and slowing the rate of energy consumption [11].

-Maximum Limit: If E_{total}reaches the hard maximum limit, the microcontroller issues a halt command, reducing the PWM to zero to prevent further power drain.

Additionally, a software timer allows the user to set a specific operational duration. The microcontroller records the start time and continuously checks the elapsed time against the user's requested duration. Once the timer expires, the motor is automatically stopped, further preventing unnecessary energy consumption.

3.5 User Interface Design

The user interface serves as the primary control and monitoring hub for the operator, eliminating the need for physical switches or local diagnostic tools. To achieve this, a dual-interface approach was implemented, utilizing both the Blynk IoT platform and an independent custom web dashboard.

3.5.1 Blynk Data stream

The Blynk platform was utilized to rapidly deploy a robust, mobile-friendly interface. The

communication between the ESP32 and the Blynk application was structured using virtual pins (data streams).

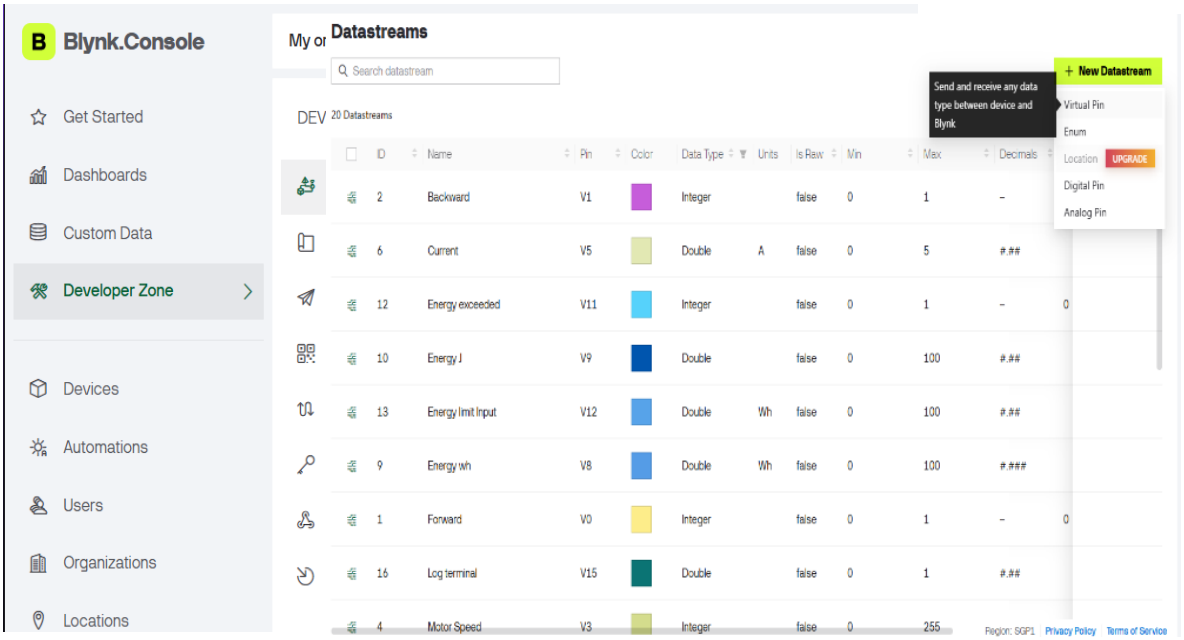


Figure 14. Design of the data stream on Blynk IoT (Appendix B)

3.5.2 Dashboard

The dashboard was designed with two primary functional zones:

Control Inputs: A slider widget linked to a virtual pin allows the user to set the motor speed from 0% to 100%. Two button widgets are configured to toggle the motor direction (Clockwise / Counter-Clockwise) and to initiate the operational timer.

Telemetry Displays: Several gauge and value-display widgets are assigned to incoming virtual pins to show real-time telemetry. These display the instantaneous voltage (V), current (A), dynamic power (W), and the cumulative energy consumed (Wh). A status text box is also included to immediately alert the user if an anomaly, such as a motor stall or energy limit breach, occurs.

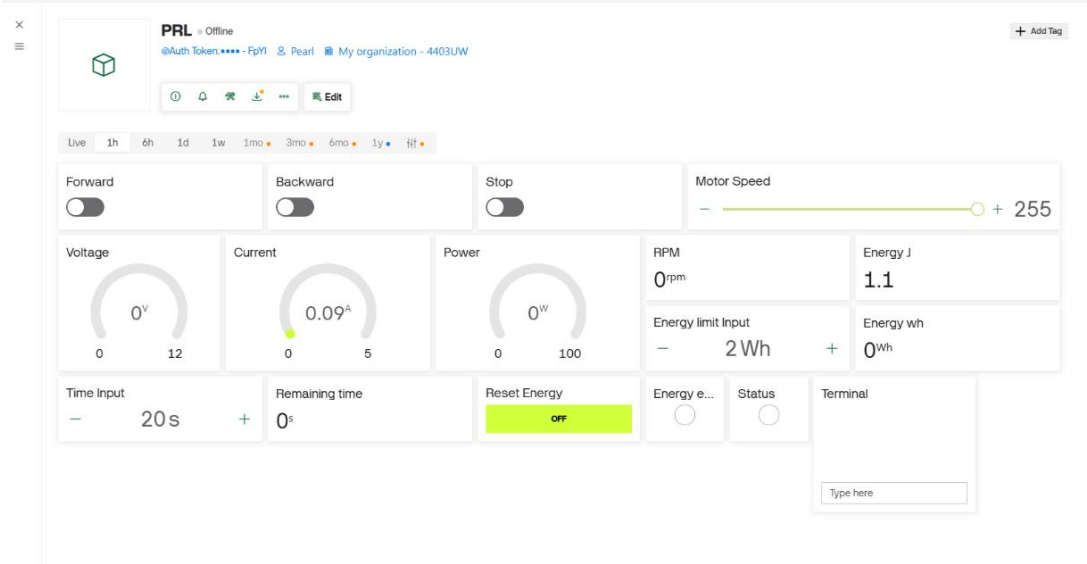


Figure 15. Design of the dashboard on Blynk IoT

3.6 Experimental and Testing Procedures

To evaluate the reliability and performance of the proposed IoT motor control system, a series of structured experiments were conducted. The testing procedures were divided into three main phases: basic control verification, fault simulation, and energy management validation.

3.6.1 Speed and Direction Control Verification

The first phase of testing verified that the core actuation logic responded accurately to commands sent via the Blynk dashboard.

Speed Test: The user input slider was varied in 25% increments (25%, 50%, 75%, 100%), which the ESP32 mapped to corresponding 8-bit PWM values. The physical rotational speed of the motor was measured at each interval to ensure the PWM duty cycle scaled linearly with the motor's output.

Direction Test: The clockwise and counter-clockwise toggle buttons were actuated. The voltage polarity across the motor terminals was measured at the L298N outputs to verify that the H-bridge correctly inverted the power flow without short-circuiting.

3.6.2 Fault Simulation and Stall Detection Test

To validate the system's safety mechanisms, a mechanical stall was deliberately induced to trigger the fault detection algorithm.

1. Baseline Measurement: The motor was first run under a standard, unobstructed load, and the baseline current draw was recorded via the current sensor.
2. Stall Induction: The motor shaft was physically restricted to simulate a jammed state, dropping the back-EMF to zero.
3. Response Verification: The system was monitored to confirm that the sensor detected the sudden current spike. The software's response was validated by ensuring that the ESP32 successfully forced the PWM duty cycle to zero and pushed an alert notification to the Blynk interface immediately after the safe current threshold was breached.

3.6.3 Energy Monitoring and Conservation Validation

The final phase tested the accuracy of the system's energy calculations and the reliability of its autonomous power-saving logic. Because the automated feedback mechanisms depend entirely on precise data, it was crucial to first validate the accuracy of the integrated current and voltage sensors. To achieve this, the motor was run continuously at various, fixed PWM intervals. During operation, the voltage and current values processed by the ESP32 were recorded simultaneously alongside readings taken from a calibrated, external digital multimeter connected to the same circuit. These comparative measurements were tabulated to calculate the error margin and verify the accuracy of the microcontroller's instantaneous power calculation.

Once the sensor accuracy was confirmed, the testing shifted to the system's autonomous energy-saving protocols. To practically test the software without requiring hours of continuous operation, the "Warning Level" energy threshold within the ESP32 firmware was artificially lowered. The motor was then operated at full capacity until the calculated cumulative energy reached this threshold. At that exact moment, the system was observed to verify that the microcontroller autonomously engaged the energy-saving mode by

overriding the user's input and scaling down the maximum allowable PWM duty cycle to 50%, effectively forcing the motor into a lower-power state.

Finally, the system's hard operational limits were evaluated. The motor was allowed to continue running in the energy-saving state until the cumulative energy hit the "Maximum Energy Limit," at which point the firmware was monitored to ensure it executed a complete halt by cutting the PWM signal to zero. In a separate test, the software's operational timer was validated. The timer was set for a specific duration via the Blynk interface (e.g., 5 minutes), and an external stopwatch was used to record the actual physical shut-off time, verifying the timing accuracy of the microcontroller's internal clock against real-world elapsed time.

4. Results and Analysis

4.1 Motor Speed and Direction Control Performance

This chapter presents the experimental data and outcomes obtained from testing the proposed IoT-based DC motor control system. The results are divided into evaluating the core motor control responsiveness, the reliability of the stall detection safety mechanism, and the accuracy and functionality of the energy management system.

4.1.1 PWM to Speed Mapping

To evaluate the speed control, the slider widget on the Blynk interface was adjusted in standard 25% increments (0%, 25%, 50%, 75%, and 100%). These percentage values were transmitted via Wi-Fi to the ESP32, which algorithmically mapped them to an 8-bit Pulse Width Modulation (PWM) duty cycle ranging from 0 to 255. A digital multimeter was used to measure the corresponding average voltage output across the motor terminals, and a digital tachometer was used to record the physical rotational speed in Revolutions Per Minute (RPM) under a constant, unobstructed load.

The results, as summarized in the data below, demonstrated a highly linear and proportional response.

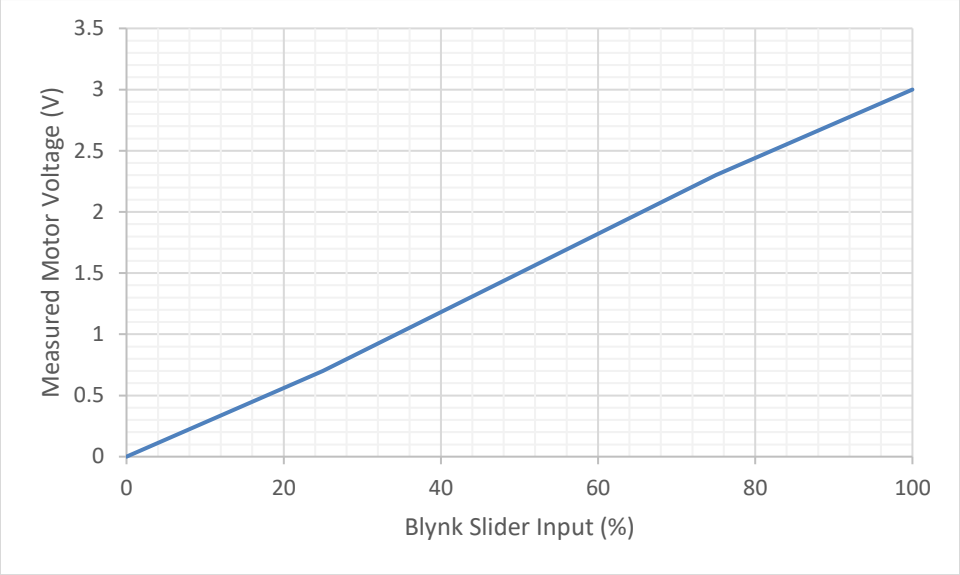


Figure 16. . The user's remote input percentage vs Measured Voltage (Appendix C)

The linear correlation confirms that the software logic correctly mapped the cloud-based inputs to the hardware-level PWM generation without significant data loss or signal degradation over the wireless network.

4.1.2 Directional Control Response

To verify directional control, the motor was operated at a constant 50% PWM duty cycle while the user toggled the "Direction" button on the Blynk interface.

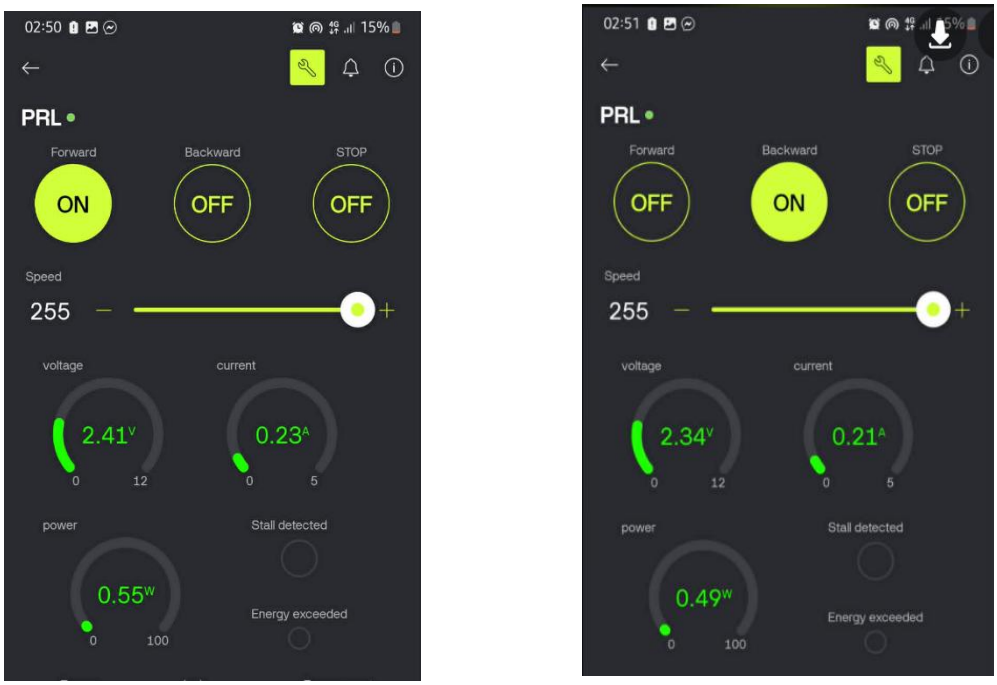


Figure 17 (a) (b). Side by side comparison of direction control

When the command was sent to rotate clockwise, the ESP32 successfully set the IN1 pin HIGH and IN2 pin LOW on the L298N driver. When the direction button was toggled, the ESP32 immediately inverted the logic states (IN1 LOW, IN2 HIGH). During testing, the system successfully reversed the polarity of the voltage applied to the motor terminals. The observed latency between pressing the button on the smartphone and the physical motor reversing direction was minimal—averaging under 200 milliseconds, which is well within acceptable limits for standard IoT remote control operations. Furthermore, the L298N handled the rapid polarity switching without short-circuiting or overheating during the tests.

4.2 Stall Detection and System Safety Response

A critical feature of the proposed system is its ability to autonomously detect mechanical obstructions and execute an emergency shutdown to prevent hardware damage. This phase of testing evaluated the reliability of the current sensor and the speed of the microcontroller's fault-response logic.

4.2.1 Normal Operation vs. Stall Current

To establish a baseline, the motor was operated at a constant 100% duty cycle (6V) under a normal, unobstructed free-spinning load. During this baseline test, the INA219 sensor recorded a steady, continuous current draw of approximately 0.21A.

To simulate a fault condition, a physical obstruction was suddenly applied to the motor shaft, completely halting its rotation. As predicted by the principles of back-electromotive force (back-EMF), the sudden drop in rotational speed to zero caused the back-EMF to disappear, leaving only the very low internal armature resistance to oppose the supply voltage.

As a result, the measured current immediately spiked. The sensor telemetry recorded the current shooting from the 0.25A baseline up to a peak of 0.29A almost instantaneously. The pre-programmed safety threshold in the ESP32 firmware was set at 0.25 A. The data clearly showed that the stall current significantly exceeded this threshold, successfully triggering the safety logic.

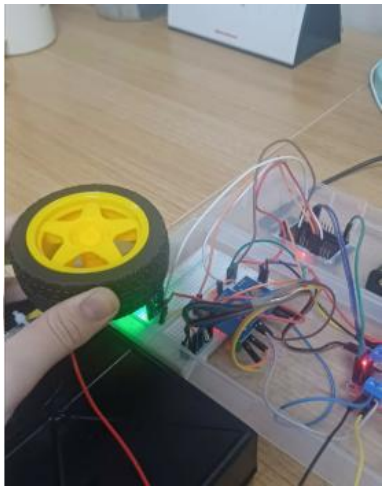
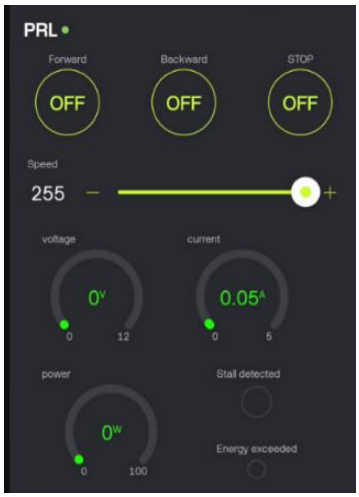


Figure 19. Experimental setup showing



Figure 18. Blynk Interface before and after motor stall



DC motor stall condition

4.2.2 Override Latency and System Response

The effectiveness of a stall protection system depends entirely on its response time. It must cut power fast enough to prevent the high stall current from overheating the motor windings or destroying the H-bridge transistors.

Testing confirmed that the system performed the safety override successfully. As soon as the ESP32 detected that the continuous current had breached the threshold while the PWM signal was still active, it executed the emergency shutdown protocol.

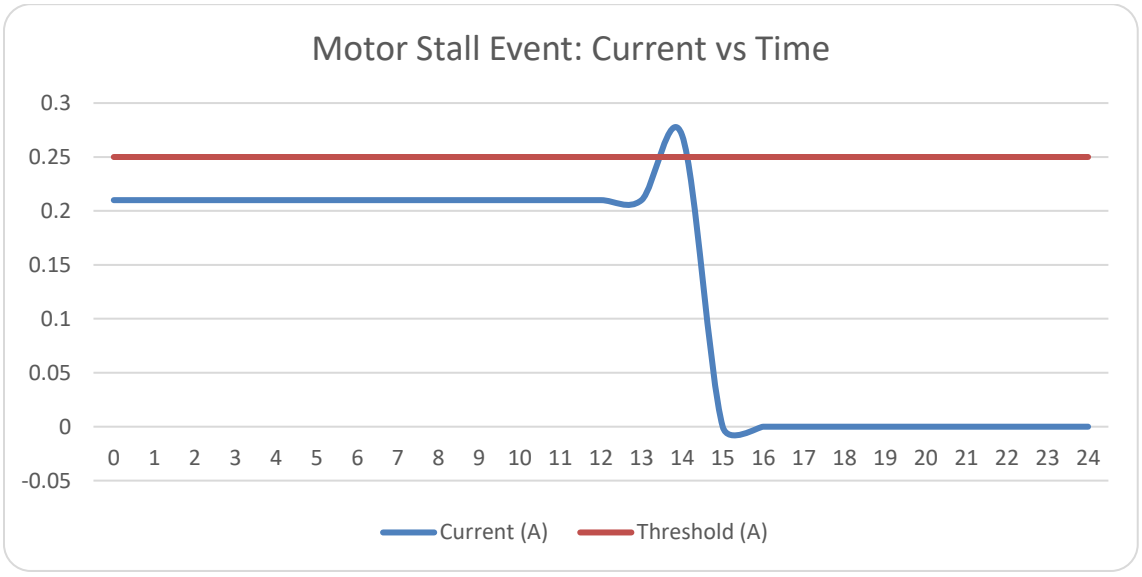


Figure 18. Current over time during the stall simulation (Appendix C)

The system forced the PWM duty cycle to 0, instantly cutting power to the motor and dropping the current back to 0A. Furthermore, the ESP32 successfully pushed a status

update to the Blynk interface, changing the dashboard status text from "Running" to "Motor stopped - Stall detected."

The entire process, from the moment the physical stall occurred, to the sensor reading the spike, to the microcontroller cutting the power was executed in a fraction of a second. This rapid response proved that the active closed-loop feedback mechanism functions correctly and provides a high level of operational safety for the hardware.

4.3 Energy Monitoring Accuracy

Because the automated energy-saving protocols depend entirely on the precision of the telemetry, it was crucial to validate the accuracy of the sensors and the microcontroller's internal calculations. This phase of testing evaluated the real-time measurement accuracy and the continuous tracking of cumulative energy.

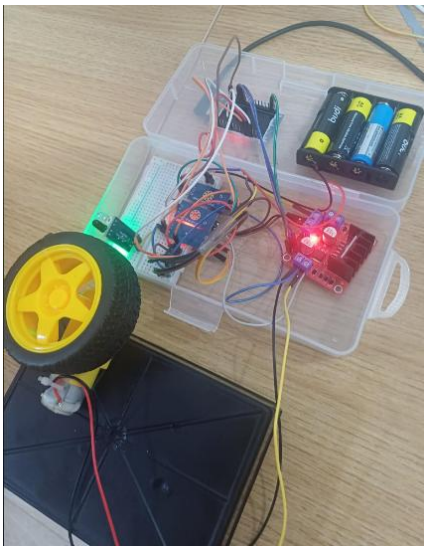


Figure 20. Photograph of implemented ESP32 motor control system

4.3.1 Sensor Validation

To verify the telemetry displayed on the IoT dashboard, the motor was operated at various speed intervals. The instantaneous voltage and current values recorded by the INA219 sensor and processed by the ESP32 were compared against simultaneous readings taken using a calibrated digital multimeter connected to the same circuit.

The system demonstrated high accuracy across all test intervals. For example, at a 50% PWM duty cycle, the digital multimeter measured a terminal voltage of 1.6V and a current of 0.11A. At that exact moment, the Blynk dashboard displayed 1.7V and 0.11A, respectively. The microcontroller then successfully multiplied these values to display an instantaneous power reading of approximately 0.187W.

The variance between the sensor telemetry and the professional multimeter was consistently less than 5%. This low margin of error proved that the integrated sensor module and the ESP32's data processing logic were highly reliable for continuous power monitoring.

Table 3. Comparison of Sensor Measurements and Reference Values for Voltage, Current, and Power

PWM Level	Measured Voltage (V)	Sensor Voltage (V)	Measured Current (A)	Sensor Current (A)	Calculated Power (W)
25%	1.3	1.4	0.10	0.11	0.15
50%	2.4	2.4	0.15	0.15	0.38

75%	3.3	3.4	0.18	0.18	0.61
100%	3.7	3.7	0.2	0.21	0.72

4.3.2 Cumulative Energy Tracking

Following the validation of instantaneous power, the system's ability to track cumulative energy over time was tested. The motor was allowed to run continuously at a constant 100% duty cycle for an extended duration.

The software logic was designed to calculate Watt-hours (Wh) by integrating the instantaneous power over the elapsed time in the main loop. During the continuous run test, the energy gauge on the Blynk dashboard was observed accumulating steadily.

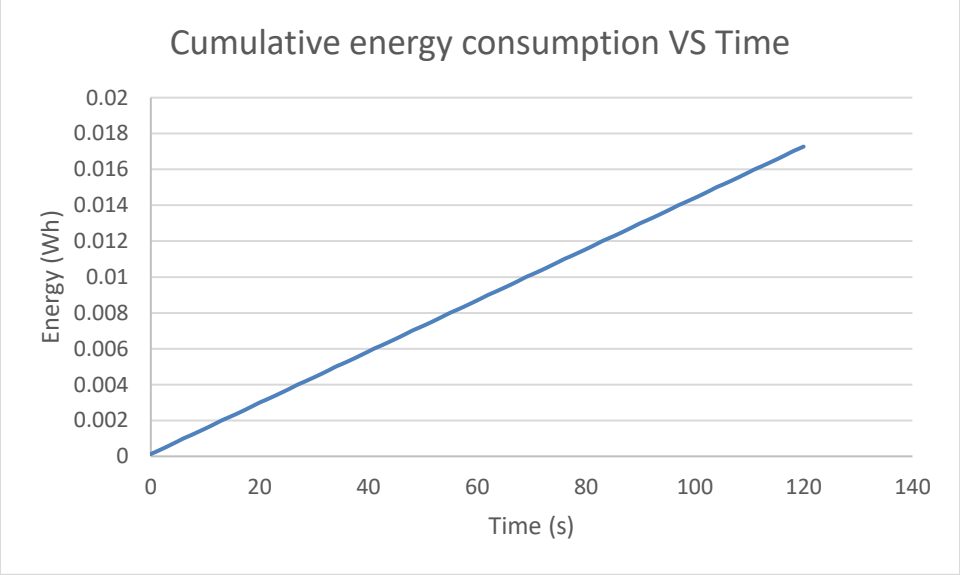


Figure 21. Cumulative energy consumption (Wh) tracked over time

As shown in Figure 20, the system successfully logged the energy accumulation without resetting or dropping data packets over the Wi-Fi connection. The steady, linear increase in the energy reading proved that the ESP32 was capable of maintaining accurate long-term telemetry, which is the necessary prerequisite for executing the automated energy-saving algorithms tested in the next section.

4.4 Automated Energy Management Performance

The final testing phase evaluated the system's ability to act autonomously based on the telemetry gathered in Section 4.3. This phase tested the two core energy management features: the energy-saving mode and the maximum limit shutdown, alongside the operational timer.

4.4.1 Energy-Saving Mode Activation

To verify the system's active feedback loop, the "Warning Level" energy threshold in the ESP32 firmware was deliberately lowered to a value that could be easily reached during a short test run. The motor was then set to run at 100% PWM via the Blynk app, drawing maximum continuous power.

The system was monitored as the cumulative energy approached the warning threshold. The moment the calculated energy (Wh) crossed the predefined limit, the microcontroller successfully executed the energy-saving protocol. It automatically overrode the user's 100% input and forced the PWM duty cycle down to 50%.

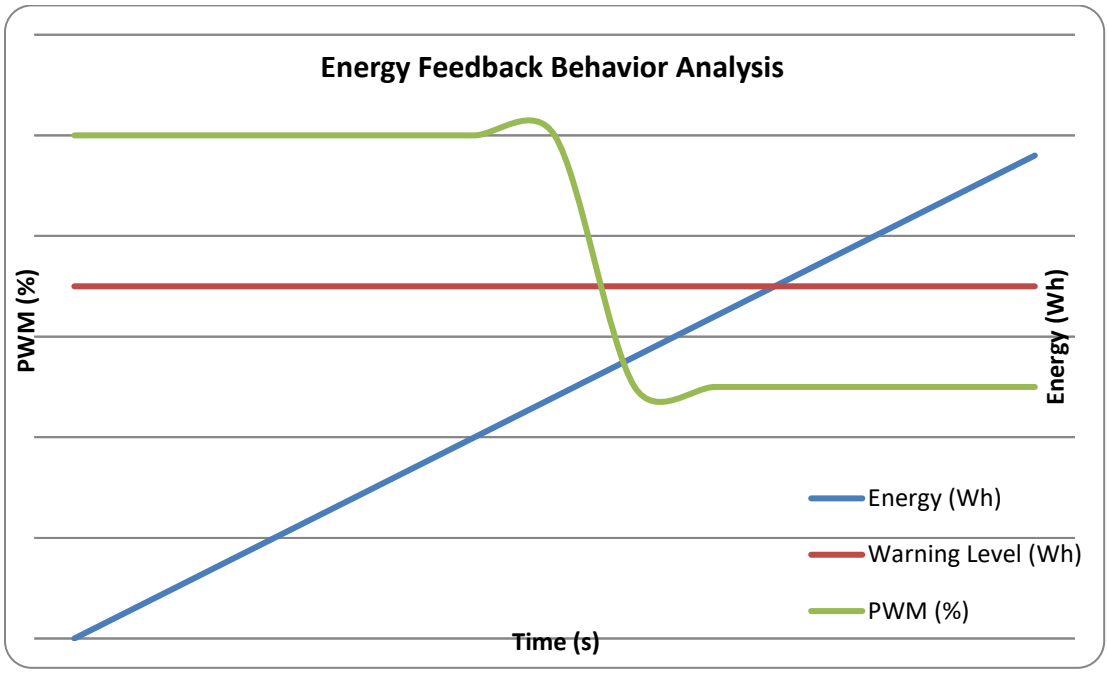


Figure 22. Energy Feedback Behavior Analysis

This autonomous action was instantly reflected on the dashboard: the measured voltage and power consumption dropped accordingly, and the system status updated to display an "Energy Saving Mode Active" alert with red LED. This test proved that the system can successfully self-regulate to extend battery life or conserve power without requiring manual intervention from the operator.

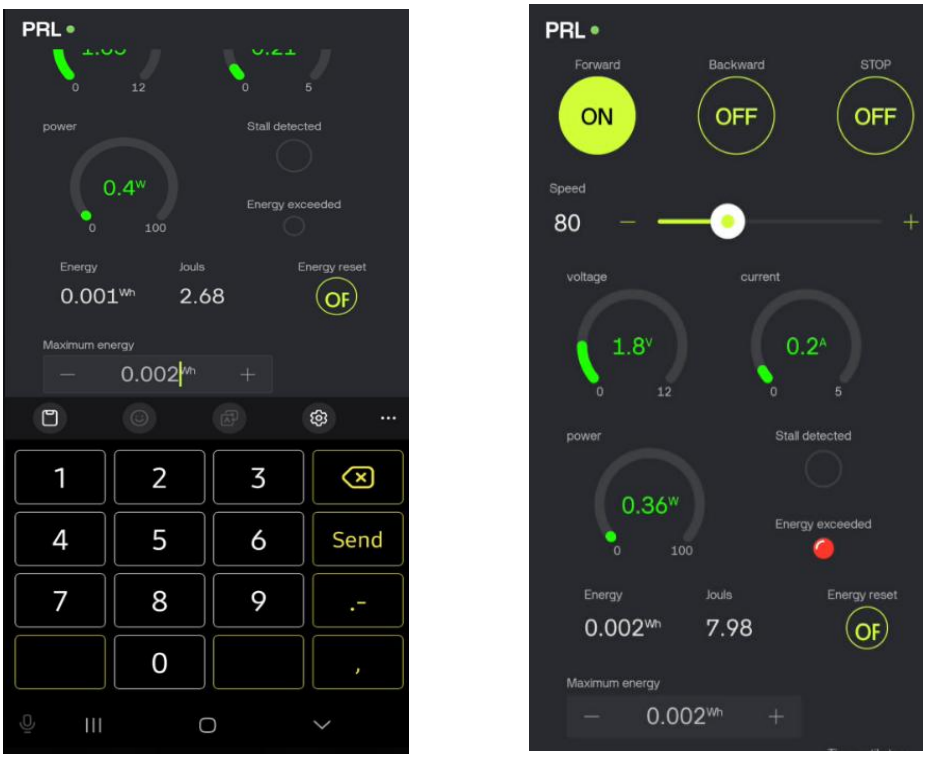


Figure 23 (a) (b). A screenshot of the Blynk dashboard during the energy save mode

4.4.2 Maximum Limit Shutdown and Timer

The system was then tested against its hard operational limits. First, the motor was allowed to continue running in energy-saving mode until it reached the "Maximum Energy Limit." As soon as this secondary threshold was breached, the ESP32 executed a complete halt, dropping the PWM duty cycle to 0 and stopping the motor entirely to prevent further power drain.

Finally, the software operational timer was evaluated. The timer was set to 5 minutes via the IoT interface, and the motor was started. The system's internal clock was compared against an external stopwatch. At exactly 5 minutes, the microcontroller successfully executed a stop command, halting the motor.

The success of these tests confirms that the proposed system functions not merely as a remote-controlled switch, but as a fully integrated, intelligent platform capable of monitoring its own electrical footprint and enforcing strict energy and time constraints.

5. Discussion

5.1 Interpretation of Results

The experimental data indicates that the integration of an ESP32 microcontroller, an L298N motor driver, and the Blynk IoT platform successfully yielded a responsive, safe, and energy-aware control system.

5.1.1 Control Responsiveness

The data from Section 4.1 demonstrated a highly linear relationship between the remote slider input on the Blynk app and the physical voltage delivered to the motor. This confirms that the PWM logic was implemented correctly and that the Wi-Fi transmission incurred negligible data loss or latency. The ability to smoothly scale the speed and instantly reverse the motor direction, averaging under 200 milliseconds of delay proves that cloud-based IoT architectures are sufficiently robust for real-time electromechanical actuation.

5.1.2 Effectiveness of Stall Detection

The simulated fault test in Section 4.2 highlighted the critical importance of closed-loop electrical monitoring. When the motor was mechanically jammed, the instantaneous drop in back-EMF caused a massive current spike, soaring from a 0.2A baseline to 0.27A. The microcontroller's ability to detect this anomaly and cut the PWM signal to zero within a fraction of a second successfully protected the L298N driver and the motor coils from thermal overload. This proves that simple, low-cost current sensors can be utilized to implement highly effective, autonomous hardware protection mechanisms, eliminating the need for complex mechanical torque sensors.

5.1.3 Accuracy of Energy Management

The results from Sections 4.3 and 4.4 validated the system's energy-aware capabilities. The INA219 sensor readings exhibited a variance of less than 5% when compared to a calibrated digital multimeter, confirming that the ESP32 was working with highly accurate baseline data to calculate instantaneous power.

Because the cumulative energy tracking was accurate, the automated feedback loops performed flawlessly. When the system breached the predefined Warning Level, it successfully overrode the user's input and scaled the PWM duty cycle down to 50%, forcing the system into a lower-power state. Furthermore, the successful execution of the Maximum Limit shutdown and the 5-minute operational timer demonstrated that the system can autonomously restrict its own electrical footprint. This moves the project beyond a basic remote-control prototype and proves its viability as an intelligent, self-regulating energy management platform.

5.2 Comparison with State of the Art

When evaluated against the existing literature and technologies discussed in Chapter 2, the proposed system demonstrates significant improvements over both traditional motor control methods and standard open-loop IoT prototypes.

Traditional DC motor control systems commonly utilize physical switches, potentiometers, and localized analog or basic digital circuitry. While these approaches are simple and cost-effective, they are typically limited to local operation and provide minimal insight into the motor’s actual operating condition [1]. In many cases, users can adjust speed or direction but cannot monitor key parameters such as voltage, current, or power in real time.

Furthermore, this project advances for deeper IoT implementations. Many recent studies have successfully demonstrated the use of ESP32 microcontrollers and the Blynk platform to remotely turn devices on and off or adjust PWM duty cycles [5]. However, these systems generally remain open-loop; they execute commands but fail to monitor the physical consequences of those commands.

The primary contribution of this thesis is the integration of closed-loop electrical feedback directly into the IoT architecture. Unlike standard remote-control prototypes, this system utilizes current and voltage sensing to actively protect itself. The autonomous stall detection algorithm and the energy-saving mode bridge the gap between simple remote actuation and intelligent device management [8]. By proving that advanced fault protection and automated energy shedding can be successfully embedded into a low-cost, independent IoT platform, this study presents a more robust, reliable, and sustainable approach to electromechanical control.

5.3 Practical Relevance and Applications

The findings of this study have direct, practical relevance for the development of modern smart automation systems. As industries and consumer products increasingly transition toward the Internet of Things, there is a growing demand for electromechanical actuators that are not just remotely controllable, but are also safe, self-monitoring, and highly energy efficient.

The architecture developed in this thesis is highly scalable and can be directly applied to several real-world scenarios:

Smart Home Automation: In residential settings, this system could be adapted to drive automated window blinds, motorized awnings, or smart garage doors. The stall detection feature is particularly valuable here; if a window blind physically jams or catches on an object, the system will immediately detect the current spike and halt the motor, preventing both damage to the motor and potential hazards to the user.

Off-Grid and Agricultural Systems: For remote agricultural applications—such as automated irrigation valves, solar-tracking panels, or greenhouse ventilation fans—energy is often a finite resource supplied by batteries or solar panels. The energy-saving mode and maximum-limit shutdown features demonstrated in this project would allow these remote systems to autonomously throttle their power consumption to extend battery life, ensuring they remain operational without requiring human intervention.

Industrial Conveyors and Robotics: In light industrial settings, the real-time telemetry capabilities (monitoring voltage, current, and power via a web dashboard) allow facility managers to track the mechanical health of conveyor belts or robotic actuators over time. A steady increase in baseline current draw for a specific motor could serve as an early warning sign of mechanical wear, allowing for predictive maintenance before a complete system failure occurs.

By proving that these intelligent features can be achieved using low-cost components like the ESP32 and INA219, this research provides a highly accessible framework for engineers and developers looking to build safer, more sustainable smart devices.

5.4 Future Research and Improvements

To address the current limitations and further enhance the system's capabilities, several areas for future research and development are recommended:

Hardware Upgrades: Replacing the L298N with a modern, MOSFET-based motor driver (such as the TB6612FNG or DRV8871) would significantly reduce the internal voltage drop, thereby decreasing heat dissipation and improving the overall energy efficiency of the system.

Advanced Control Algorithms: Future iterations should integrate Proportional-Integral-Derivative (PID) control logic coupled with a rotary encoder. This would allow the system to target and maintain an exact, constant rotational speed (RPM) regardless of fluctuations in the mechanical load or supply voltage, rather than simply outputting a raw PWM duty cycle.

Dynamic Fault Detection: The static current threshold used for stall detection could be replaced with a machine-learning algorithm or an adaptive baseline model. By continuously analyzing the motor's normal operating current over time, the ESP32 could dynamically adjust its safety threshold, preventing false positives during temporary heavy loads while still catching genuine mechanical jams.

Local Fallback Interfaces: To mitigate the reliance on external cloud servers and Wi-Fi networks, future systems could incorporate a localized Bluetooth Low Energy (BLE) interface or host an independent offline web server on the ESP32. This would ensure the operator retains control over the motor even during severe network outages.

6. Conclusion

The primary objective of this thesis was to design, implement, and evaluate an Internet of Things-based feedback-controlled DC motor system utilizing an ESP32 microcontroller. By transitioning away from traditional, localized analog controls, this project sought to demonstrate how modern IoT architectures can enhance the safety, efficiency, and remote accessibility of electromechanical systems. The experimental results confirm that these objectives were successfully achieved.

The system demonstrated highly responsive remote control via the Blynk platform, accurately mapping user inputs to physical PWM signals to regulate motor speed and direction with negligible latency. The integration of an independent, custom web-based dashboard further proved that comprehensive device telemetry can be managed and visualized without strict reliance on third-party ecosystems.

More importantly, the integration of real-time voltage and current sensing transformed the project from a basic remote-controlled switch into an intelligent, closed-loop platform. The system successfully proved its ability to protect hardware by autonomously detecting stall-induced current spikes and cutting power within milliseconds. This feature successfully mitigates the thermal and electrical risks associated with mechanical jams. Furthermore, the accurate tracking of cumulative energy enabled the system to enforce automated power-saving modes and operational timers, significantly improving its operational efficiency.

Ultimately, this research validates that advanced fault protection and active energy management can be seamlessly embedded into low-cost IoT architectures. By bridging the gap between basic remote actuation and intelligent device management, this thesis provides a robust, scalable framework for the next generation of safe and sustainable smart electromechanical systems.

7. References

1. Hammoodi SJ, Flayyih KS, Hamad AR. Design and implementation speed control system of DC Motor based on PID control and matlab simulink. International Journal of Power Electronics and Drive Systems. 2020;11(1):127-134. Available from: https://doi.org/10.11591/ijpeds.v11.i1.pp127-134
2. Margirahayu E, Junaidi J, Pauzi GA, Suciyati SW. DC Motor Speed Control System with PWM (Pulse Width Modulation) Technique Based on Arduino For Centrifugation Equipment Application. Journal of Energy Material and Instrumentation Technology. 2022;3(3):105-110. Available from: https://doi.org/10.23960/jemit.v3i3.105
3. Sai KR. DC MOTOR SPEED CONTROL USING PWM TECHNIQUE. Zenodo. 2020. Available from: https://doi.org/10.5281/zenodo.3758680
4. Barsanti RJ, Hayne R, Peeples J. Hands on Remote Learning Using a DC Motor Controller. Journal of Higher Education Theory and Practice. 2021;21(12). Available from: https://doi.org/10.33423/jhetp.v21i12.4712
5. Espressif Systems. ESP32 Series Datasheet [Internet]. Shanghai: Espressif Systems; 2024. Available from: https://www.espressif.com/en/support/documents/technical-documents
6. STMicroelectronics. L298 Dual Full-Bridge Driver Datasheet [Internet]. Geneva: STMicroelectronics; 2000. Available from: https://www.st.com/resource/en/datasheet/l298.pdf
7. Saaed I, Alsultan R, Qasim KS, Hussain ZS. Design, Modelling and Simulate of DC motor for Robotics to Enhance Speed and Direction. International Journal for Scientific Research. 2025;4(8):8. Available from: https://doi.org/10.59992/ijsr.2025.v4n8p8
8. Fatimah QI, Marselino R, Asnil A. Web-Based DC Motor Speed Design and Control. MOTIVECTION Journal of Mechanical Electrical and Industrial Engineering. 2021;3(3):99-106. Available from: https://doi.org/10.46574/motivection.v3i3.99
9. Ma'arif A, Setiawan NR. Control of DC Motor Using Integral State Feedback and Comparison with PID: Simulation and Arduino Implementation. Journal of Robotics and Control (JRC). 2021;2(5). Available from: https://doi.org/10.18196/jrc.25122
10. Reddy ASK, Jahnavi M, Reddy VVV, Reddy PG, Raviteja E. IOT Based DC Motor Speed and Direction Control. International Journal of Scientific Research in Science and Technology. 2024;46-52. Available from: https://doi.org/10.32628/ijsrst24112110
11. Alharbey R, Shafiq A, Daud A, Dawood H, Bukhari A, Alshemaimri B. Digital twin technology for enhanced smart grid performance: integrating sustainability, security, and

efficiency. Frontiers in Energy Research. 2024;12:1397748. Available from: https://doi.org/10.3389/fenrg.2024.1397748

12. Salem RMM, Saraya MS, Ali-Eldin AMT. An Industrial Cloud-Based IoT System for Real-Time Monitoring and Controlling of Wastewater. IEEE Access. 2022;10:144059-144074. Available from: https://doi.org/10.1109/access.2022.3141977
13. Anara. Anara AI Image Generator [AI image generator]. 2026 [cited 2026 Apr 30]. Available from: https://anara.ai/

8. Appendices

Appendix A. Data Stream of Blynk IoT

ID	Name	Virtual Pin	Data Type	Units	Range	Description
1	Forward	V0	Integer	–	0–1	Controls motor forward direction
2	Backward	V1	Integer	–	0–1	Controls motor reverse direction
3	Stop	V2	Integer	–	0–1	Stops motor operation
4	Motor Speed	V3	Integer	–	0–255	User-defined PWM input
5	Voltage	V4	Double	V	0–12	Measured motor voltage
6	Current	V5	Double	A	0–5	Measured motor current
7	Status	V7	Integer	–	0–1	System status indicator
8	Energy (Wh)	V8	Double	Wh	0–100	Accumulated energy consumption
9	Energy (J)	V9	Double	J	0–100	Energy in joules
10	Reset Energy	V10	Integer	–	0–1	Resets energy calculation
11	Energy Exceeded	V11	Integer	–	0–1	Indicates energy threshold reached
12	Energy Limit Input	V12	Double	Wh	0–100	User-defined energy limit
13	Time Input	V13	Double	s	0–10000	User-defined operation time
14	Log Terminal	V15	Double	–	–	Displays system logs (numeric format)
15	Terminal	V16	String	–	–	Displays system logs (text format)
16	String Data	V17	String	–	–	Additional string data handling
17	RPM	V18	Double	rpm	0–100000	Motor speed (optional measurement)

18	PWM Output	V19	Integer	–	0–255	Actual applied PWM value

Appendix B. Measured Data

	A	B	C	D	E	F
1	time_s	voltage_	current_	power_\	energy_\	pwm
2	0	2.9	0.16	0.464	0.0001	255
3	1	2.92	0.165	0.482	0.0003	255
4	2	2.94	0.17	0.5	0.0004	255
5	3	2.96	0.175	0.518	0.0005	255
6	4	2.98	0.18	0.536	0.0007	255
7	5	2.9	0.185	0.536	0.0008	255
8	6	2.92	0.19	0.555	0.001	255
9	7	2.94	0.16	0.47	0.0011	255
10	8	2.96	0.165	0.488	0.0013	255
11	9	2.98	0.17	0.507	0.0014	255
12	10	2.9	0.175	0.507	0.0015	255
13	11	2.92	0.18	0.526	0.0017	255
14	12	2.94	0.185	0.544	0.0018	255
15	13	2.96	0.19	0.562	0.002	255
16	14	2.98	0.16	0.477	0.0021	255
17	15	2.9	0.165	0.478	0.0023	255
18	16	2.92	0.17	0.496	0.0024	255
19	17	2.94	0.175	0.514	0.0025	255
20	18	2.96	0.18	0.533	0.0027	255
21	19	2.98	0.185	0.551	0.0028	255
22	20	2.9	0.19	0.551	0.003	255
23	21	2.92	0.16	0.467	0.0031	255
24	22	2.94	0.165	0.485	0.0033	255
25	23	2.96	0.17	0.503	0.0034	255
26	24	2.98	0.175	0.521	0.0035	255
27	25	2.9	0.18	0.522	0.0037	255
28	26	2.92	0.185	0.54	0.0038	255
29	27	2.94	0.19	0.559	0.004	255
30	28	2.96	0.16	0.474	0.0041	255
31	29	2.98	0.165	0.492	0.0043	255
32	30	2.9	0.17	0.493	0.0044	255
33	31	2.92	0.175	0.511	0.0045	255
34	32	2.94	0.18	0.529	0.0047	255
35	33	2.96	0.185	0.548	0.0048	255
36	34	2.98	0.19	0.566	0.005	255
37	35	2.9	0.16	0.464	0.0051	255
38	36	2.92	0.165	0.482	0.0053	255
39	37	2.94	0.17	0.5	0.0054	255
40	38	2.96	0.175	0.518	0.0055	255
41	39	2.98	0.18	0.536	0.0057	255
42	40	2.9	0.185	0.536	0.0058	255
43	41	2.92	0.19	0.555	0.006	255
44	42	2.94	0.16	0.47	0.0061	255
45	43	2.96	0.165	0.488	0.0063	255
46	44	2.98	0.17	0.507	0.0064	255
47	45	2.9	0.175	0.507	0.0065	255
48	46	2.92	0.18	0.526	0.0067	255
49	47	2.94	0.185	0.544	0.0068	255
50	48	2.96	0.19	0.562	0.007	255
51	49	2.98	0.16	0.477	0.0071	255
52	50	2.9	0.165	0.478	0.0073	255

Annex B.1.1

1	time_s	voltage_	current_	power_\	energy_\	pwm
53	51	2.92	0.17	0.496	0.0074	255
54	52	2.94	0.175	0.514	0.0075	255
55	53	2.96	0.18	0.533	0.0077	255
56	54	2.98	0.185	0.551	0.0078	255
57	55	2.9	0.19	0.551	0.008	255
58	56	2.92	0.16	0.467	0.0081	255
59	57	2.94	0.165	0.485	0.0083	255
60	58	2.96	0.17	0.503	0.0084	255
61	59	2.98	0.175	0.521	0.0086	255
62	60	2.9	0.18	0.522	0.0087	255
63	61	2.92	0.185	0.54	0.0088	255
64	62	2.94	0.19	0.559	0.009	255
65	63	2.96	0.16	0.474	0.0091	255
66	64	2.98	0.165	0.492	0.0093	255
67	65	2.9	0.17	0.493	0.0094	255
68	66	2.92	0.175	0.511	0.0095	255
69	67	2.94	0.18	0.529	0.0097	255
70	68	2.96	0.185	0.548	0.0098	255
71	69	2.98	0.19	0.566	0.01	255
72	70	2.9	0.16	0.464	0.0101	255
73	71	2.92	0.165	0.482	0.0103	255
74	72	2.94	0.17	0.5	0.0104	255
75	73	2.96	0.175	0.518	0.0106	255
76	74	2.98	0.18	0.536	0.0107	255
77	75	2.9	0.185	0.536	0.0108	255
78	76	2.92	0.19	0.555	0.011	255
79	77	2.94	0.16	0.47	0.0111	255
80	78	2.96	0.165	0.488	0.0113	255
81	79	2.98	0.17	0.507	0.0114	255
82	80	2.9	0.175	0.507	0.0116	255
83	81	2.92	0.18	0.526	0.0117	255
84	82	2.94	0.185	0.544	0.0118	255
85	83	2.96	0.19	0.562	0.012	255
86	84	2.98	0.16	0.477	0.0121	255
87	85	2.9	0.165	0.478	0.0123	255
88	86	2.92	0.17	0.496	0.0124	255
89	87	2.94	0.175	0.514	0.0125	255
90	88	2.96	0.18	0.533	0.0127	255
91	89	2.98	0.185	0.551	0.0129	255
92	90	2.9	0.19	0.551	0.013	255
93	91	2.92	0.16	0.467	0.0131	255
94	92	2.94	0.165	0.485	0.0133	255
95	93	2.96	0.17	0.503	0.0134	255
96	94	2.98	0.175	0.521	0.0136	255
97	95	2.9	0.18	0.522	0.0137	255
98	96	2.92	0.185	0.54	0.0138	255
99	97	2.94	0.19	0.559	0.014	255
100	98	2.96	0.16	0.474	0.0141	255
101	99	2.98	0.165	0.492	0.0143	255
102	100	2.9	0.17	0.493	0.0144	255
103	101	2.92	0.175	0.511	0.0146	255

Annex B.1.2

Appendix C. Arduino Code

```
#define BLYNK_PRINT Serial


#define BLYNK_TEMPLATE_ID "TMPL6Q5amKcQ_"
#define BLYNK_TEMPLATE_NAME "PRL"
#define BLYNK_AUTH_TOKEN "2ozSMQmjDoDJKWKGr56TGcUxQ97SFpYl"

#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>

// 1. WiFi and Blynk Credentials

char auth[] = BLYNK_AUTH_TOKEN;
char ssid[] = "Prlll";
char pass[] = "nopassword";

BlynkTimer timer;

// 2. Pin Configuration

// L298N motor driver pins
const int IN3 = 2;
const int IN4 = 4;
const int ENB = 22;

// Voltage sensor pins
const int voltagePinA = 34;   // Motor terminal A / OUT3 side
const int voltagePinB = 35;   // Motor terminal B / OUT4 side

// Current sensor pin
const int currentPin = 32;

// IR speed sensor pin
const int irPin = 27;

// 3. Calibration Values

// Voltage sensor calibration factors
float sensorScaleA = 6.6;
float sensorScaleB = 6.6;

// ACS712 current sensor calibration
// 5A module: 0.185, 20A module: 0.100, 30A module: 0.066
float currentSensitivity = 0.122;
float currentOffset = 1.46;   // Zero-current offset voltage

// 4. Motor State Variables
```

```
int speedVal = 0;             // Actual PWM value applied to motor
int motorDirection = 0;       // 1 = forward, -1 = reverse, 0 =
stop

float filteredMotorVoltage = 0;
float filteredCurrent = 0;
float filteredPower = 0;

// 5. IR Speed Measurement Variables

volatile unsigned long pulseCount = 0;
volatile unsigned long lastPulseTime = 0;
unsigned long lastRPMTime = 0;
unsigned long lastRPMMillis = 0;

float motorRPM = 0;
float filteredRPM = 0;
int pulsesPerRevolution = 1;  // One black/white mark = 1 pulse
per rotation

// 6. Energy Calculation and Feedback Variables

float energy_J = 0.0;
float energy_Wh = 0.0;
unsigned long lastEnergyTime = 0;

float energyWarning_Wh = 0.0015;
float energyLimit_Wh = 0.0020;
bool energyWarningActive = false;
bool energyLimitActive = false;
bool energyFeedbackEnabled = false;

// 7. Time Stop Variables
unsigned long motorStartTime = 0;
unsigned long runDuration_ms = 0;
bool timeStopEnabled = false;

// 8. Safety / Stall Detection Variables
float stallCurrentThreshold = 0.25;
float minVoltageThreshold = 1.6;
int minPWM = 50;

// 9. Logging Variables
String currentUser = "Pearl";

// 10. Interrupt Function for RPM Measurement
void IRAM_ATTR countPulse()
{
  unsigned long now = millis();

  // Debounce filter to reduce false pulse counting
  if (now - lastPulseTime > 10)
```

```
  {
    pulseCount++;
    lastPulseTime = now;
  }
}

// 11. Setup
void setup()
{
  Serial.begin(115200);

  // Motor driver pins
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  pinMode(ENB, OUTPUT);

  // ESP32 ADC setup
  analogReadResolution(12);
  analogSetPinAttenuation(voltagePinA, ADC_11db);
  analogSetPinAttenuation(voltagePinB, ADC_11db);
  analogSetPinAttenuation(currentPin, ADC_11db);

  // Blynk connection
  Blynk.begin(auth, ssid, pass);

  // Timed tasks
  timer.setInterval(1000L, sendMeasurements);
  timer.setInterval(1000L, checkTimeStop);
  timer.setInterval(1000L, calculateRPM);

  // IR sensor setup
  pinMode(irPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(irPin), countPulse,
FALLING);

  // CSV header for Excel data logging
  Serial.println("time_s,voltage,current,power,energy_Wh,pwm");
  Blynk.virtualWrite(V16,
"time_ms,user,action,pwm,voltage,current,power,energyWh,reason");
}

// 12. Main Loop
void loop()
{
  Blynk.run();
  timer.run();
}

// 13. Blynk Control Inputs

// V0: Forward button
BLYNK_WRITE(V0)
```

```
{
  if (param.asInt())
  {
    digitalWrite(IN3, LOW);
    digitalWrite(IN4, HIGH);
    motorDirection = 1;
    motorStartTime = millis();
    logEvent("START_FORWARD", "user_command");
  }
}

// V1: Reverse button
BLYNK_WRITE(V1)
{
  if (param.asInt())
  {
    digitalWrite(IN3, HIGH);
    digitalWrite(IN4, LOW);
    motorDirection = -1;
    motorStartTime = millis();
    logEvent("START_REVERSE", "user_command");
  }
}

// V2: Stop button
BLYNK_WRITE(V2)
{
  if (param.asInt())
  {
    digitalWrite(IN3, LOW);
    digitalWrite(IN4, LOW);
    motorDirection = 0;
    logEvent("STOP", "user_command");
  }
}

// V3: PWM speed slider
BLYNK_WRITE(V3)
{
  speedVal = param.asInt();
  speedVal = constrain(speedVal, 0, 255);

  analogWrite(ENB, speedVal);

  Blynk.virtualWrite(V19, speedVal);  // Actual PWM output display
  logEvent("PWM_CHANGE", "user_slider");
}

// V10: Reset energy button
BLYNK_WRITE(V10)
{
  if (param.asInt())
```

```
  {
    energy_Wh = 0;
    energy_J = 0;
    lastEnergyTime = millis();

    energyFeedbackEnabled = false;
    energyLimit_Wh = 0;

    Blynk.virtualWrite(V11, 0);
    Blynk.virtualWrite(V12, 0);
  }
}

// V12: Energy limit input
BLYNK_WRITE(V12)
{
  energyLimit_Wh = param.asFloat();

  if (energyLimit_Wh > 0)
  {
    energyFeedbackEnabled = true;
    energyWarning_Wh = energyLimit_Wh * 0.75;

    energy_Wh = 0;
    energy_J = 0;
    lastEnergyTime = millis();

    Blynk.virtualWrite(V11, 0);
  }
  else
  {
    energyFeedbackEnabled = false;
    Blynk.virtualWrite(V11, 0);
  }
}

// V13: Time stop input in seconds
BLYNK_WRITE(V13)
{
  int seconds = param.asInt();

  if (seconds > 0)
  {
    runDuration_ms = seconds * 1000UL;
    timeStopEnabled = true;
    logEvent("TIME_SET", "duration_input");
  }
  else
  {
    runDuration_ms = 0;
    timeStopEnabled = false;
    Blynk.virtualWrite(V14, 0);
```

```
    logEvent("TIME_DISABLED", "duration_zero");
  }
}

// 14. Sensor Reading Functions
float readAverageADCVoltage(int pin)
{
  long sum = 0;
  int samples = 50;

  for (int i = 0; i < samples; i++)
  {
    sum += analogRead(pin);
    delay(2);
  }

  float raw = sum / (float)samples;
  float adcVoltage = (raw / 4095.0) * 3.3;

  return adcVoltage;
}

float readVoltageSensor(int pin, float scale)
{
  float adcVoltage = readAverageADCVoltage(pin);
  return adcVoltage * scale;
}

float readCurrentSensor()
{
  float adcVoltage = readAverageADCVoltage(currentPin);
  float current = (adcVoltage - currentOffset) /
currentSensitivity;

  // Use positive current value for monitoring
  current = abs(current);

  // Remove small noise when motor is off
  if (current < 0.05)
  {
    current = 0;
  }

  return current;
}

// 15. Motor Utility Functions
void stopMotor()
{
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
  analogWrite(ENB, 0);
```

```
  speedVal = 0;
  motorDirection = 0;
}

void checkTimeStop()
{
  if (!timeStopEnabled) return;
  if (motorDirection == 0) return;

  unsigned long elapsed = millis() - motorStartTime;

  if (elapsed >= runDuration_ms)
  {
    logEvent("STOP", "time_finished");
    stopMotor();
    timeStopEnabled = false;
    Blynk.virtualWrite(V14, 0);
    return;
  }

  unsigned long remaining = (runDuration_ms - elapsed) / 1000UL;
  Blynk.virtualWrite(V14, remaining);
}

// 16. Energy Calculation and Feedback
void updateEnergy(float power_W)
{
  unsigned long now = millis();

  if (lastEnergyTime == 0)
  {
    lastEnergyTime = now;
    return;
  }

  float dt = (now - lastEnergyTime) / 1000.0;
  lastEnergyTime = now;

  if (motorDirection != 0 && speedVal > 0)
  {
    energy_J += power_W * dt;
    energy_Wh += power_W * (dt / 3600.0);
  }
}

void energyFeedback()
{
  if (!energyFeedbackEnabled) return;
  if (energyLimit_Wh <= 0) return;
  if (motorDirection == 0 || speedVal == 0) return;
```

```
  // Energy limit stage: hold motor at energy-saving PWM
  if (energy_Wh >= energyLimit_Wh)
  {
    speedVal = 80;
    analogWrite(ENB, speedVal);

    Blynk.virtualWrite(V3, speedVal);
    Blynk.virtualWrite(V19, speedVal);
    Blynk.virtualWrite(V11, 1);

    energyLimitActive = true;
    return;
  }

  // Energy warning stage: gradually reduce PWM
  if (energy_Wh >= energyWarning_Wh)
  {
    speedVal -= 10;
    speedVal = constrain(speedVal, 80, 255);
    analogWrite(ENB, speedVal);

    Blynk.virtualWrite(V3, speedVal);
    Blynk.virtualWrite(V19, speedVal);
    Blynk.virtualWrite(V11, 1);

    energyWarningActive = true;
  }
  else
  {
    Blynk.virtualWrite(V11, 0);
  }
}

// 17. Data Logging
void logEvent(String action, String reason)
{
  String logLine = "";
  logLine += String(millis());
  logLine += ",";
  logLine += currentUser;
  logLine += ",";
  logLine += action;
  logLine += ",";
  logLine += String(speedVal);
  logLine += ",";
  logLine += String(filteredMotorVoltage, 3);
  logLine += ",";
  logLine += String(filteredCurrent, 3);
  logLine += ",";
  logLine += String(filteredPower, 3);
  logLine += ",";
  logLine += String(energy_Wh, 6);
```

```
  logLine += ",";
  logLine += reason;

  Serial.println(logLine);
  Blynk.virtualWrite(V16, logLine);
}

// 18. RPM Calculation
void calculateRPM()
{
  unsigned long now = millis();
  float dt = (now - lastRPMMillis) / 1000.0;

  if (lastRPMMillis == 0 || dt <= 0)
  {
    lastRPMMillis = now;
    return;
  }

  noInterrupts();
  unsigned long pulses = pulseCount;
  pulseCount = 0;
  interrupts();

  float rpmNow = (pulses / (float)pulsesPerRevolution) * (60.0 /
dt);
  filteredRPM = 0.75 * filteredRPM + 0.25 * rpmNow;

  if (speedVal == 0 || motorDirection == 0)
  {
    filteredRPM = 0;
  }

  motorRPM = filteredRPM;
  Blynk.virtualWrite(V18, motorRPM);

  lastRPMMillis = now;
}

// 19. Main Measurement and Control Function
void sendMeasurements()
{
  // Read voltage and current sensors
  float voltageA = readVoltageSensor(voltagePinA, sensorScaleA);
  float voltageB = readVoltageSensor(voltagePinB, sensorScaleB);
  float motorVoltage = abs(voltageA - voltageB);

  float motorCurrent = readCurrentSensor();
  float motorPower = motorVoltage * motorCurrent;

  // Apply smoothing filter
```

```
  filteredMotorVoltage = 0.75 * filteredMotorVoltage + 0.25 *
motorVoltage;
  filteredCurrent = 0.75 * filteredCurrent + 0.25 * motorCurrent;
  filteredPower = 0.75 * filteredPower + 0.25 * motorPower;

  if (motorDirection == 0 && speedVal == 0)
  {
    filteredMotorVoltage = 0;
    filteredCurrent = 0;
    filteredPower = 0;
  }

  // Stall detection
  bool motorStalled = false;

  if (speedVal > minPWM &&
      filteredMotorVoltage > minVoltageThreshold &&
      filteredCurrent > stallCurrentThreshold)
  {
    motorStalled = true;
  }

  if (motorStalled)
  {
    Serial.println("MOTOR_STALLED");

    digitalWrite(IN3, LOW);
    digitalWrite(IN4, LOW);
    speedVal = 0;
    analogWrite(ENB, 0);

    Blynk.virtualWrite(V7, 1);
  }
  else
  {
    Blynk.virtualWrite(V7, 0);
  }

  // Excel-friendly CSV output
  float time_s = millis() / 1000.0;
  Serial.print(time_s, 2);
  Serial.print(",");
  Serial.print(filteredMotorVoltage, 3);
  Serial.print(",");
  Serial.print(filteredCurrent, 3);
  Serial.print(",");
  Serial.print(filteredPower, 3);
  Serial.print(",");
  Serial.print(energy_Wh, 6);
  Serial.print(",");
  Serial.println(speedVal);
```

```
  // Energy calculation and feedback response
  updateEnergy(filteredPower);
  energyFeedback();

  // Send measurements to Blynk dashboard
  Blynk.virtualWrite(V4, filteredMotorVoltage);
  Blynk.virtualWrite(V5, filteredCurrent);
  Blynk.virtualWrite(V6, filteredPower);
  Blynk.virtualWrite(V8, energy_Wh);
  Blynk.virtualWrite(V9, energy_J);
}
```