SPEED CONTROL OF DC MOTOR USING STM32 IN CLOSED LOOP

A PROJECT REPORT

21ES602 – Embedded System Design Using ARM

submitted by

Keerthana M G	**CB.EN.P2EBS24011**
Saraan J	**CB.EN.P2EBS24019**

in partial fulfilment for the award of the degree

of

MASTER OF TECHNOLOGY

IN

EMBEDDED SYSTEMS

AMRITA SCHOOL OF ENGINEERING, COIMBATORE

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE – 641 112

Submitted on - 6th November, 2024

TABLE OF CONTENTS

S. No.	Chapter	Page no:
1.	Introduction	3
2.	Components required	4
3.	Theory	5
4.	Block diagram	7
5.	Working of the project	8
6.	Code	9
7.	Results	11
8.	Future scope	12
9.	Conclusion	13

CHAPTER 1

1.1 INTRODUCTION:

The control of DC motor speed is a critical aspect of modern industrial automation, robotics, and various electronic applications. Precise speed control in a DC motor enables consistent performance, accuracy, and energy efficiency, which are essential in applications ranging from consumer electronics to advanced manufacturing. The STM32 microcontroller, a powerful and versatile embedded platform, is particularly well-suited for implementing closed-loop control systems for DC motors due to its real-time processing capabilities, advanced peripherals, and ease of integration with various sensors and control algorithms.

In a closed-loop control system, the motor's speed is constantly monitored and compared with a desired reference speed. Any deviation between the actual and desired speeds results in a corrective adjustment, achieved through feedback mechanisms. This approach offers significant advantages, including improved stability, accuracy, and robustness against external disturbances or load changes.

The purpose of this project is to design and implement a closed-loop speed control system for a DC motor using an STM32 microcontroller. The system utilizes Pulse Width Modulation (PWM) to control motor speed, while an encoder measures the actual speed of the motor. Through a Proportional-Integral-Derivative (PID) controller algorithm, the STM32 adjusts the PWM duty cycle to maintain the desired speed, ensuring accurate and reliable motor operation.

CHAPTER – 2

2.1 COMPONENTS REQUIRED:

1. DC Motor
2. STM32 Microcontroller
3. Encoder (Rotary encoder or optical encoder)
4. Motor driver
5. Power supply
6. Speed sensor

DC Motor:

A motor that operates with direct current power. Motor specifications such as rated voltage, current and speed are essential for determining the required power supply and driving circuit.

STM32 Microcontroller:

A powerful ARM cortex – M based microcontroller known for its high speed operation and versatility. Provides necessary peripherals for implementing closed – loop control including timers, PWM (Pulse Width Modulation) and ADC (Analog to Digital Converter).

Encoder:

Measures the motor's speed by generating pulses proportional to the rotational movement. Outputs digital signals (pulses) that the STM32 can read to determine speed.

Motor driver:

A power electronic device that drives the DC motor. Capable of handling the current and voltage required by the motor and providing control inputs for speed and direction.

Power supply:

Supplies adequate voltage and current for the DC motor and STM32 microcontroller. Ensure it matches the motor specifications and is stable to avoid fluctuations.

Speed sensor:

An optical encoder or a Hall effect sensor can be used as a speed sensor to measure the DC motor's speed. These sensors generate pulses as the motor shaft rotates, allowing the STM32 microcontroller to monitor speed in real-time

CHAPTER – 3

3.1 THEORY:

DC Motor Control Basics:

A DC motor operates by converting electrical energy into mechanical motion, with the speed of rotation controlled by the voltage applied to the motor's terminals. Speed control can be achieved by adjusting the voltage or current supplied to the motor, typically through Pulse Width Modulation (PWM). In PWM, a high-frequency digital signal switches the motor's power on and off rapidly, effectively controlling the average voltage.

Pulse Width Modulation (PWM):

PWM is a technique used to regulate power to the motor by controlling the duty cycle of the signal—i.e., the ratio of the "on" time to the total time period. By varying the duty cycle, you can control the motor's average voltage and, hence, its speed. STM32 microcontrollers have dedicated timer peripherals that can generate PWM signals. Configuring the timer allows precise control over PWM frequency and duty cycle.

Closed-Loop Control and Feedback System:

A closed-loop control system adjusts the motor's speed to match a desired setpoint by continuously measuring and adjusting based on feedback. The feedback loop compares the measured speed with the desired speed (setpoint) and applies corrections. In this project, the feedback system uses a rotary encoder or tachometer to measure the motor's actual speed. The encoder generates pulses as the motor shaft rotates, allowing the system to calculate the speed.

Proportional-Integral-Derivative (PID) Control:

PID control is a popular control algorithm used in closed-loop systems to minimize error and stabilize control. It consists of three terms:

- Proportional (P): Adjusts the control signal based on the current error.
- Integral (I): Accounts for past errors and corrects for any steady-state error.
- Derivative (D): Predicts future errors based on the rate of change of the error.

The PID algorithm calculates a control output that adjusts the PWM duty cycle to keep the motor speed close to the desired value.

STM32 Microcontroller and Peripherals:

STM32 microcontrollers are powerful for motor control due to features like PWM generation, high-speed timers, and digital input pins that can read encoder signals. The timer module can be configured to read encoder pulses and calculate motor speed by counting the frequency of these pulses. The STM32's ADC (Analog-to-Digital Converter) and PWM output make it possible to read sensor inputs and control the motor, respectively.

Software Implementation:

In the software, you'll configure the STM32 to generate PWM signals for the motor driver and set up a timer for reading encoder pulses. The PID algorithm is implemented to

calculate the difference between the desired speed and actual speed (feedback). The PWM duty cycle is adjusted in real time based on the PID output to maintain the motor speed.

Motor Driver Circuit:

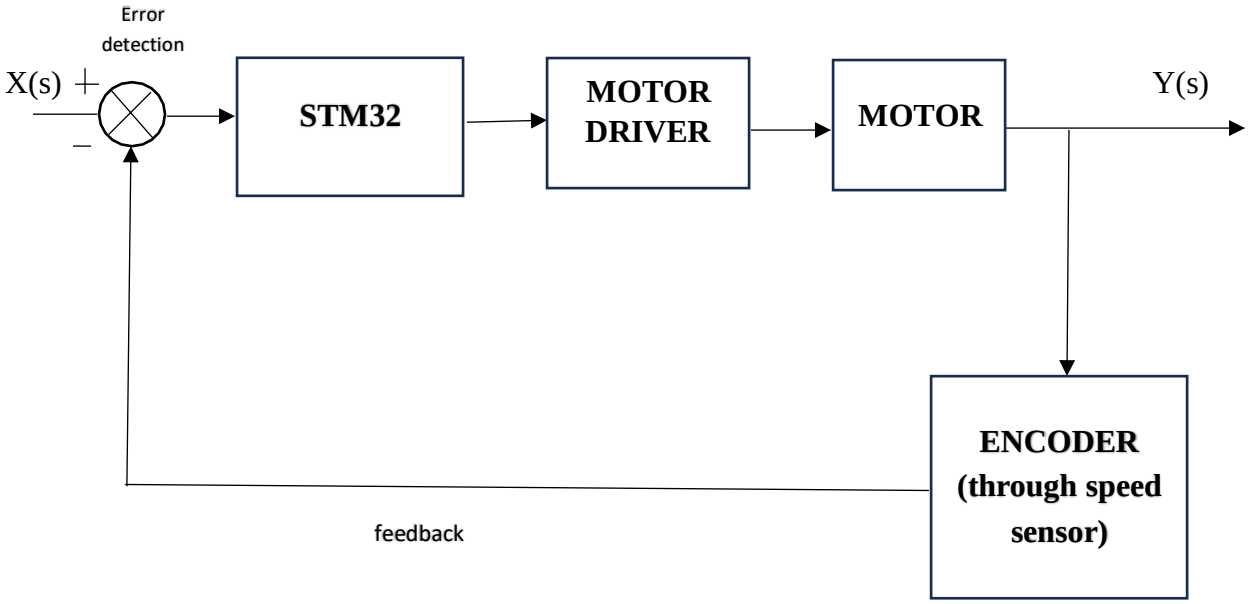
The motor driver (e.g., L298, L293, or MOSFET-based circuit) acts as an interface between the STM32 and the DC motor. The motor driver circuit amplifies the PWM signal and provides enough current to the motor to drive it effectively without overloading the STM32.

Error Handling and Stability:

Stability is crucial in closed-loop systems. The PID parameters (P, I, D gains) must be tuned to ensure the motor reaches and maintains the desired speed without excessive oscillation or overshoot. Using methods like Ziegler–Nichols or trial-and-error can help find suitable PID values for your specific motor and load.

CHAPTER – 4

4.1 BLOCK DIAGRAM:



General closed loop transfer function,

$$T(s) = \frac{G(s)}{1 + G(s)H(s)}$$

CHAPTER – 5

5.1 WORKING OF THE PROJECT:

In this closed-loop DC motor speed control system, the STM32 microcontroller receives feedback from the motor's encoder to regulate the motor's speed accurately. Here's a breakdown of how each component and process works together to achieve this.

The system begins with a target speed input, which can be set through various means, such as a potentiometer or a digital input. This target speed represents the desired motor rotationspeed, usually measured in revolutions per minute (RPM). Once the target speed is set, the STM32 continuously monitors the motor's actual speed to compare it with this reference.

The encoder attached to the motor shaft plays a crucial role in providing real-time speed feedback. As the motor spins, the encoder generates pulses, with each pulse representing a small increment of rotation. These pulses are fed into the STM32's GPIO pins configured to count them. The STM32 measures the frequency of these pulses, which corresponds directly to the motor's rotational speed.

To control the motor's speed effectively, the STM32 needs to calculate the speed error,which is the difference between the target and actual speed. This error value is then used in a PID (Proportional-Integral-Derivative) control algorithm to determine the corrective action needed. The PID algorithm continuously adjusts the motor's PWM (Pulse Width Modulation) signal to correct any deviations from the target speed. The proportional term (P) reacts to the current error, the integral term (I) accumulates past errors to eliminate steady-state errors, and the derivative term (D) predicts future errors, smoothing the response.

The PWM signal generated by the STM32 modulates the power sent to the motor through a motor driver, such as an L298N or similar. By adjusting the PWM duty cycle, the STM32 effectively controls the motor's speed. When the motor's speed lags behind the target, the STM32 increases the PWM duty cycle to boost power; conversely, if the speed exceeds the target, the STM32 reduces the duty cycle, slowing the motor down.

This feedback loop runs continuously, with the STM32 frequently recalculating the speed and adjusting the PWM output. As a result, the motor's speed remains stable even under varying loads or other disturbances, thanks to the real-time adjustments made by the PID controller. With careful tuning of the PID coefficients, the system can achieve smooth and precise speed control, ideal for applications requiring stable motor speeds.

CHAPTER – 6

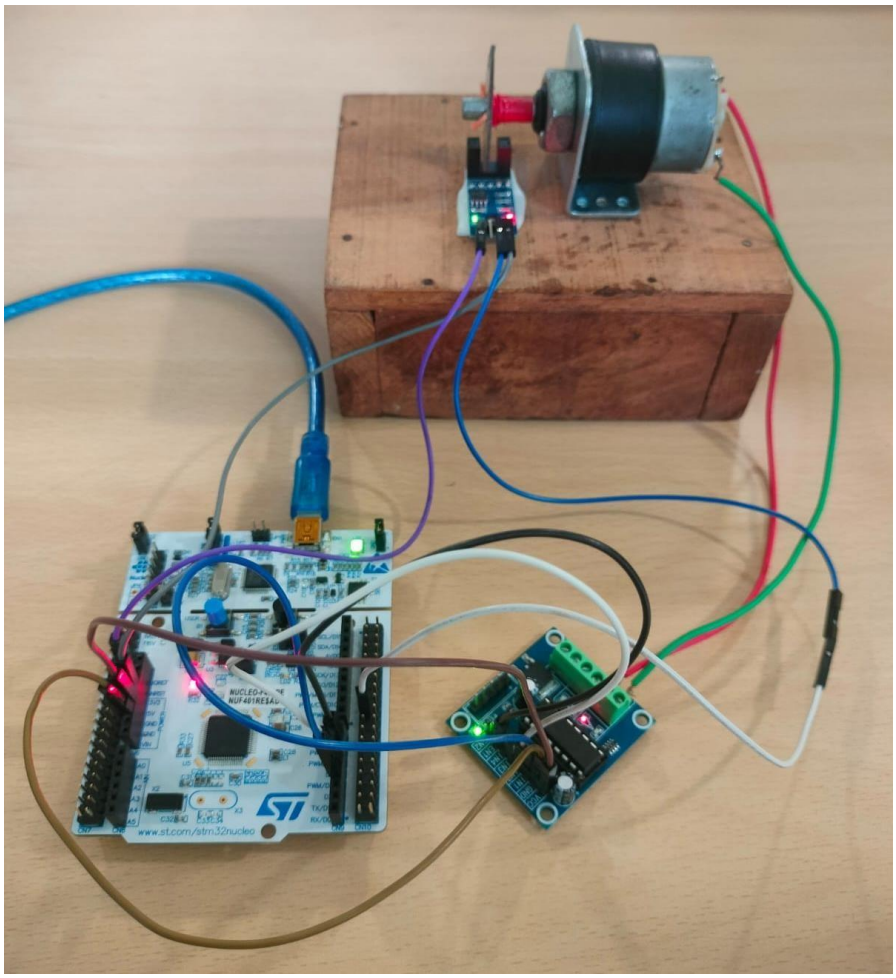
6.1 CODE

```
// Define motor control pins
#define IN1_PIN 4
#define IN2_PIN 5
#define ENA_PIN 6

// Define encoder feedback pin (LM393 output)
#define SENSOR_PIN 2

// Variables for closed-loop control
volatile unsigned long pulseCount = 0; // Pulse count from LM393 encoder
unsigned long previousPulseCount = 0;  // Previous pulse count for speed calculation
unsigned long timeInterval = 1000;     // Time interval to calculate speed in ms (1 second)
unsigned long actualSpeed = 0;         // Actual motor speed (RPM)
unsigned long targetSpeed = 100;       // Desired motor speed (RPM)
unsigned long previousTime = 0;        // Time tracking for speed control

void setup() {
  // Initialize serial communication for debugging
  Serial.begin(9600);

  // Set motor control pins as output
  pinMode(IN1_PIN, OUTPUT);
  pinMode(IN2_PIN, OUTPUT);
  pinMode(ENA_PIN, OUTPUT);

  // Set sensor pin as input (LM393 output)
  pinMode(SENSOR_PIN, INPUT);

  // Attach interrupt to sensor pin (LM393 output) to count pulses
  attachInterrupt(digitalPinToInterrupt(SENSOR_PIN), countPulses, RISING);

  // Set initial motor direction and speed
  digitalWrite(IN1_PIN, HIGH);  // Motor direction forward
  digitalWrite(IN2_PIN, LOW);   // Motor direction forward
  analogWrite(ENA_PIN, 128);    // Set initial motor speed (50% PWM)
}

void loop() {
  unsigned long currentTime = millis();

  // Calculate actual speed (RPM) every timeInterval (1 second)
  if (currentTime - previousTime >= timeInterval) {
    // Calculate the actual speed (in RPM) based on pulse count
    unsigned long pulseDelta = pulseCount - previousPulseCount;
    actualSpeed = (pulseDelta * 60) / (timeInterval / 1000); // Convert pulse count to RPM

    // Display actual speed on Serial Monitor for debugging
```

```
    Serial.print("Actual Speed: ");
    Serial.print(actualSpeed);
    Serial.print(" RPM, ");

    // Simple proportional controller to adjust motor speed
    int error = targetSpeed - actualSpeed;
    int pwmDutyCycle = 128 + error;  // Basic proportional control

    // Constrain PWM duty cycle within valid range (0-255)
    if (pwmDutyCycle > 255) {
      pwmDutyCycle = 255;
    } else if (pwmDutyCycle < 0) {
      pwmDutyCycle = 0;
    }

    // Set motor speed by adjusting PWM duty cycle
    analogWrite(ENA_PIN, pwmDutyCycle);

    // Display PWM duty cycle on Serial Monitor
    Serial.print("PWM Duty Cycle: ");
    Serial.println(pwmDutyCycle);

    // Update pulse count for the next calculation
    previousPulseCount = pulseCount;
    previousTime = currentTime;
  }

  // Add any additional logic or delay as required for your application
}

// Interrupt function to count pulses from LM393
void countPulses() {
  pulseCount++;  // Increment pulse count each time the sensor sends a pulse
}
```

CHAPTER - 7

7.1 RESULTS:



```
Actual Speed: 0 RPM, Target Speed: 100 RPM, PWM Duty Cycle: 140
Actual Speed: 0 RPM, Target Speed: 100 RPM, PWM Duty Cycle: 150
Actual Speed: 0 RPM, Target Speed: 100 RPM, PWM Duty Cycle: 160
Actual Speed: 0 RPM, Target Speed: 100 RPM, PWM Duty Cycle: 170
Actual Speed: 0 RPM, Target Speed: 100 RPM, PWM Duty Cycle: 180
Actual Speed: 0 RPM, Target Speed: 100 RPM, PWM Duty Cycle: 190
Actual Speed: 0 RPM, Target Speed: 100 RPM, PWM Duty Cycle: 200
Actual Speed: 0 RPM, Target Speed: 100 RPM, PWM Duty Cycle: 210
Actual Speed: 0 RPM, Target Speed: 100 RPM, PWM Duty Cycle: 220
Actual Speed: 0 RPM, Target Speed: 100 RPM, PWM Duty Cycle: 230
Actual Speed: 0 RPM, Target Speed: 100 RPM, PWM Duty Cycle: 240
Actual Speed: 0 RPM, Target Speed: 100 RPM, PWM Duty Cycle: 250
Actual Speed: 0 RPM, Target Speed: 100 RPM, PWM Duty Cycle: 255
Actual Speed: 0 RPM, Target Speed: 100 RPM, PWM Duty Cycle: 255
Actual Speed: 0 RPM, Target Speed: 100 RPM, PWM Duty Cycle: 255
Actual Speed: 0 RPM, Target Speed: 100 RPM, PWM Duty Cycle: 255
Actual Speed: 0 RPM, Target Speed: 100 RPM, PWM Duty Cycle: 255
Actual Speed: 0 RPM, Target Speed: 100 RPM, PWM Duty Cycle: 255
Actual Speed: 0 RPM, Target Speed: 100 RPM, PWM Duty Cycle: 255
```

CHAPTER – 8

8.1 FUTURE SCOPE:

The closed-loop speed control of a DC motor using an STM32 microcontroller holds considerable promise for future advancements across various fields. One potential direction is the development of more sophisticated control algorithms, such as adaptive control, fuzzy logic, or model predictive control, which could significantly enhance the system's ability to manage complex dynamics and disturbances. Additionally, integrating this project with the Internet of Things (IoT) could open up possibilities for remote monitoring, control, and data analysis, facilitating predictive maintenance and operational efficiency. The inclusion of more advanced feedback mechanisms, such as high-resolution encoders or additional sensors, could further improve precision and accuracy in speed regulation. Shifting from a basic loop to a real-time operating system (RTOS) would allow for more efficient multitasking, making the system suitable for complex applications where multiple processes must run concurrently, like in industrial automation or robotics.

On the hardware side, the use of sophisticated motor drivers and improved encoders would boost performance and reliability, making the system more applicable in high-stakes environments such as electric vehicles or automated manufacturing systems. Application areas are vast, ranging from robotics, where precise control is necessary, to automated machinery, CNC machines, and even electric vehicles, where efficiency and reliability are crucial. User interface development, such as a graphical interface for monitoring and control, would make the system more accessible, allowing for easier parameter tuning and diagnostics. Additionally, implementing energy-efficient features, safety mechanisms, and redundancy would enhance the system's reliability and make it suitable for critical applications. Altogether, this project offers a strong foundation for exploring embedded systems, control engineering, and software development, equipping you with valuable skills and insights for advanced industrial applications and academic research.

CHAPTER – 9

9.1 CONCLUSION:

In conclusion, the closed-loop speed control of a DC motor using an STM32 microcontroller represents a robust and versatile project with significant applications in various fields requiring precision motor control. Through the implementation of PID control, real-time feedback from an encoder, and PWM modulation, this project demonstrates effective speed regulation in a closed-loop system. It also serves as an excellent platform for exploring advanced control techniques, real-time systems, and IoT integration, all of which could greatly enhance its functionality and adaptability in industrial and research settings. The potential for future improvements—such as adaptive control algorithms, enhanced sensor feedback, and IoT connectivity—highlights the scalability of this project, making it a valuable foundation for applications in robotics, automated machinery, electric vehicles, and more. By refining this project, one can gain practical experience with embedded systems, signal processing, and control engineering, which are essential skills in today's technology-driven world. Overall, this project not only achieves precise and reliable motor control but also lays the groundwork for a range of advanced applications in modern automation and intelligent systems.