ABSTRACT

The DC Motors have been popular in the industry control area for a long time owing to their simple design and reliability.They are widely used in many industrial applications where wide speed ranges are required and therefore they are used in many adjustable speed drives. A PID Controller feedback system is integrated with DC Motor to form a digital control system. In this paper, a digital speed control of a DC motor using pulse width modulation technique was implemented by replacing the analog circuit with an Arduino microcontroller circuit.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

In today's technology, DC motors are a very important industrial component for effective and efficient handling of loads/tasks.

A DC motor is an electrical machine that converts electrical power into mechanical power as the electric field generates a force on the coil that pushes the rotor.DC motors have advantage in terms of smoothness of speed change,ease of control,and rapid dynamic response to change in load torque.

The maximum speed of the motor should not exceed the rated speed. Manufacturing industries and automobile systems such as textile,conveyor systems,paper mills,robotic systems are required to maintain variable rotation speeds of the motor with load changes.Therefore,it is necessary to control the speed of the motor by changing the magnetic flux.

CHAPTER 2

LITERATURE REVIEW

Digital input DC motor speed controller regulates motor speed based on digital signals enabling precise control.Speed control can be achieved through varying voltage, current, or pulse width modulation (PWM).

[1]K. Ravindra Sai , Ch. V. D. Krishna Vamsi , V. Vijay Kumar, B. Pradeep Kumar & P. Sudheer, "*DC Motor speed control using PWM technique*", Global Journal of Engineering Science and Researched, 7(4), 27–31.

The paper proposes a precise speed control method for DC motors using Pulse Width Modulation (PWM) with a 555 timer in astable mode. This setup generates continuous HIGH and LOW pulses to create a PWM signal, controlling motor speed by adjusting resistors and capacitors. Modern power electronic devices like MOSFETs and IGBTs enhance efficiency, compactness, and reliability of variable speed drives. DC motors offer high starting torque and versatile speed control, making them suitable for various industrial applications. Compared to AC motors, DC motors have simpler and less expensive speed control methods, with PWM being efficient and cost-effective**.**

[2]Moleykutty George, "*Speed Control of Separately Excited DC Motor*", Faculty of Engineering and Technology, Multimedia University Melaka Campus, 75450 Melaka,Malaysia.

This paper suggests several advanced methods for the precise speed control of DC motors, emphasizing the application of modern control techniques and technologies.

Direct current (DC) motors are widely used in industrial applications like electric vehicles, steel rolling mills, electric cranes, and robotic manipulators due to their precise control characteristics. Traditionally, rheostatic armature control was used for low-power DC motor speed control.

However, static power converters have improved efficiency, controllability, and performance. Conventional PID controllers achieve desired torque-speed characteristics but struggle with parameter variations. Recently, neural network controllers (NNC) have been introduced to enhance nonlinear system performance due to their learning ability and adaptability.

A constant-power field weakening controller and a single-phase uniform PWM ac-dc buck-boost converter have been proposed for effective speed control of DC motors. Simulation methods and open-loop control systems have been designed to improve dynamic behavior prediction and control speed. The superior performance of AI-based controllers is prompting a shift from conventional methods to intelligent speed controllers. This paper proposes a NARMA-L2 controller for speed control of a separately excited DC motor, presenting simulation results that demonstrate its effectiveness.

CHAPTER 3

PROBLEM DEFINITION & OPTIMIZED SOLUTION

3.1 PROBLEM DEFINITION

Huge power loss due to the usage of resistor in series with armature:-

In DC motor speed control systems, integrating a resistor in series with the armature can result in significant power loss. This is mainly because the resistor, utilized for regulating speed, dissipates surplus electrical energy as heat. As a consequence, the system's efficiency diminishes, leading to wasted power and potentially heightened operational expenses. The resistor restricts the current flowing through the motor, diminishing its torque and speed to achieve speed control. However, the voltage drop across the resistor leads to substantial power dissipation ($P = I^2R$), as the resistor typically possesses a high value to attain the desired speed reduction. This dissipated power not only reduces overall efficiency but may also necessitate additional cooling measures, increasing system complexity and costs. Alternative speed control methods like pulse-width modulation (PWM) techniques or electronic motor controllers offer more efficient solutions by adjusting the effective voltage without dissipating excess power, thus enhancing overall efficiency and minimizing power loss.

PLL method drawbacks:

Tuning challenges :- PLL (Phase-Locked Loop) methods can be challenging to tune accurately due to their complex feedback loops and sensitivity to parameters such as loop bandwidth, phase margin, and loop gain. Achieving optimal performance often requires precise adjustment of these parameters, which can be time-consuming and require specialized expertise.

Susceptibility to noise:- PLLs are sensitive to noise and disturbances in the input signal, which can affect their ability to lock onto the desired frequency or phase. Noise can introduce jitter or phase errors, leading to degraded performance, especially in high-frequency or noisy

environments. Filtering and noise-rejection techniques may be necessary to mitigate these effects.

Limited adaptability & flexibility :-While PLLs are effective for tasks like frequency synthesis, clock recovery, and demodulation, they may lack the adaptability and flexibility required for certain dynamic or rapidly changing environments. Once configured, PLL parameters may be fixed, making it difficult to adapt to variations in input conditions or requirements without re-tuning or re-designing the system. This limitation can be problematic in applications where flexibility and adaptability are critical, such as software-defined radios or agile communication systems.

3.2 OPTIMIZED SOLUTION

Utilizing encoders for real-time monitoring, feedback mechanisms enable precise adjustments to DC motor operation based on actual conditions. Motor speed control methods involve varying DC voltage or employing Pulse Width Modulation (PWM) signals, each offering distinct advantages. Controllers like P, PI, and PID are commonly employed, with PID offering comprehensive control by considering current error, past errors, and future error trends, resulting in precise speed regulation and robust performance.

CHAPTER 4

BLOCK DIAGRAM & DESCRIPTION

The figure 4.1 shows the block diagram of a digitally controlled dc motor. Here, arduino uno acts as the microcontroller. Digital input is given by the matrix keypad and the output can be displayed in the OLED display .A motor driver (L298N) is used to drive the motor (G12-N20).

4.1 BASIC BLOCK DIAGRAM

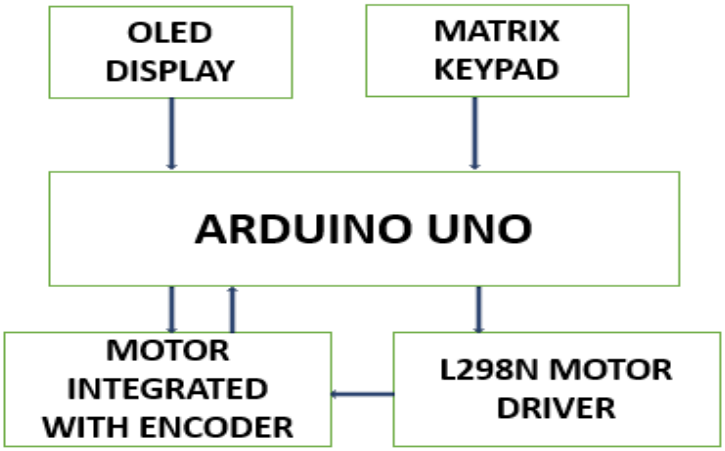


Fig: 4.1 Block Diagram

4.2 BLOCK DESCRIPTION

OLED Display

An OLED (Organic Light-Emitting Diode) display is a modern technology using organic compounds that emit light when electrified, allowing each pixel to be individually controlled. This results in vivid colors and true blacks due to the high contrast ratio, as pixels can be turned

off completely. OLED displays also offer wide viewing angles, fast response times, and thin, flexible designs, making them ideal for innovative applications like foldable screens.

Keypad Matrix

A 4x3 keypad matrix is a common input device that allows users to input numerical and other characters by pressing buttons arranged in a grid. Each button corresponds to a unique key, and the arrangement is usually in a 4x3 grid, providing 12 keys in total.

Motor driver

The L298N is a popular dual H-bridge motor driver integrated circuit (IC) commonly used to control DC motors or stepper motors. It allows bidirectional control of two motors simultaneously, making it ideal for robotics, mechatronics, and other projects requiring motor control.The module can drive DC motors that have voltages between 5 and 35V, with a peak current up to 2A.

Motor Integrated with Encoder

The G12-N20 motor is a small, low-voltage DC motor with a gearbox attached to the output shaft. The gearbox contains gears that reduce the speed of the motor while increasing its torque output.

The motor's physical dimensions are typically compact, making it suitable for applications where space is limited.The operating voltage of the G12-N20 motor typically ranges from 3V to 12V, although specific models may have different voltage ratingsThe motor's speed depends on the voltage applied and the gear ratio of the gearbox. Common speed ranges for G12-N20 motors are between 100 and 300 rpm.The gearbox of the G12-N20 motor provides increased torque output compared to a standard DC motor without a gearbox. The torque output depends on factors such as the gear ratio and the motor's operating voltage. The motor's current draw varies depending on

the load and operating conditions but is typically in the range of a few hundred milliamps (mA).The gear ratio determines the relationship between the motor's rotational speed and the output shaft's speed. A higher gear ratio results in lower output speed but higher torque.G12-N20 motors are typically controlled using pulse-width modulation (PWM) to adjust their speed.These motors can be integrated into Arduino or other microcontroller-based projects using appropriate motor driver circuits and programming.

Arduino UNO

The Arduino Uno is an open-source microcontroller. Arduino developed UNO and it is a board based on the Microchip ATmega328P microcontroller. The board is provided with sets of analog and digital input/output (I/O) pins that may be interfaced to other circuits and various expansion boards (shields). The board has 14 digital I/O pins (six capable of PWM output), 6 analog I/O pins.It is programmable with the Arduino IDE (Integrated Development Environment), through a type B USB cable. It can be powered by an external 9-volt battery or by the USB cable. Its working voltage is between 7 and 20 volts.

CHAPTER 5

COMPONENT SPECIFICATIONS

The components and its specifications used in the project is given below:

HARDWARE

Arduino UNO

Microcontroller: ATmega328

Operating Voltage: 5V

Input Voltage (recommended): 7-12V

Input Voltage (limits): 6-20V

Digital I/O Pins: 14 (of which 6 provide PWM output)

Analog Input Pins: 6

DC Current per I/O Pin: 40 mA

DC Current for 3.3V Pin: 50 mA

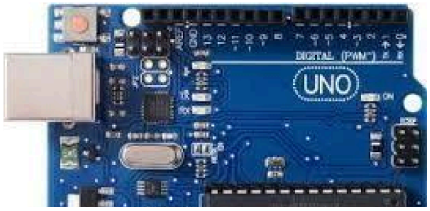
Flash Memory: 32 KB of which 0.5 KB used by boot loader

SRAM: 2 KB (Atmega328)

EEPROM: 1 KB (Atmega328)

Clock Speed: 16 MHz

Fig: 5.1 Arduino UNO



OLED Display

OLED driver- SSD1306

Supply voltage- 3.3V-6V

Diagonal screen size- 24.384mm (0.96”)

Active area size- 21.744mm*10.864mm

Pixel resolution- 128*64

Pixel Pitch- 0.17mm*0.17mm

Pixel size- 0.154mm * 0.154mm

Viewing angle- 160 degree

Fig 5.2 OLED Display

Matrix Keypad (4 x 3)

Size: 68.9 x 76 x 0.8mm

Cable length: 85mm (include connector)

Connector: dupont 7 pins, 0.1 inch (2.54mm) Pitch

Max. circuit rating: 35VDC, 100mA

Mounting: Self-Adherence

Life expectancy: 1 million closures

Contact bounce:

Weight: 7g

Dielectric withstand: 250 Vrms (60Hz, 1min)

Fig: 5.3 Matrix Keypad (4 x 3)

Motor Driver

Driver Model: L298N

Driver Chip: Double H Bridge L298N

Motor Supply Voltage (Maximum): 46V

Motor Supply Current (Maximum): 2A

Logic Voltage: 5V

Driver Voltage: 5-35V

Driver Current: 2A

Logical Current: 0-36mA

Maximum Power (W): 25W

Current Sense for each motor

Heatsink for better performance

Power-On LED indicator

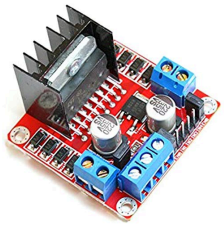


Fig: 5.4 L298N Motor Driver

Encoder Specifications

PPR (Pulses Per Revolution): 7 pulses per revolution.

CPR (Counts Per Revolution): PPR x 4 (since it's a quadrature encoder) = 7 x 4 = 28.

CPR at output shaft: Due to the gear ratio, CPR at output shaft = 28 x 30 = 840.

DC Motor

Rated Voltage : 6~12V

Revolving Speed : 100 RPM @ 6V

Load Speed: 80 RPM

Rated Torque: 2 kg.cm

Stall Torque: 16 kg.cm

Rated Current: 0.07A

Stall Current: 1A

Reduction Ratio: 1:10

Total Length : 34mm

Gear Material: Full Metal

Gearbox Size : 15 x 12 x 10mm (L*W*H)

Shaft Size : 3 x 10mm(D*L)

Fig: 5.5 Motor Integrated with Encoder

SOFTWARE

Arduino IDE

The Arduino IDE is an open-source software, which is used to write and upload code to the Arduino boards. The IDE application is suitable for different operating systems such as **Windows, Mac OS X, and Linux**. It supports the programming languages C and C++. Here, IDE stands for **Integrated Development Environment**.

The program or code written in the Arduino IDE is often called sketching. We need to connect the Genuino and Arduino board with the IDE to upload the sketch written in the Arduino IDE software. The sketch is saved with the extension '.ino.'

Proteus 8

Proteus 8 Professional is a software application that allows you to design, test, and simulate electronic circuits and microcontrollers. It is developed by Labcenter Electronics Ltd. and it is widely used by electronic engineers, hobbyists, and students.

CHAPTER 6

ALGORITHM

The provided algorithm describes the steps for implementing a motor control system that varies the motor speed based on keypad input, using a PID controller for feedback control and an OLED to display relevant information.

Algorithm/flowchart with explanation

1. Initialization (`setup` function):

- Initialize serial communication.

- Initialize the OLED display.

- Set motor control pins as output.

- Stop the motor initially.

- Set encoder pins as input.

- Attach an interrupt to the encoder A pin.

2. Main Loop (`loop` function):

- Read the pressed key from the keypad.

- Based on the pressed key:

- If `0` is pressed, stop the motor.

- If `1` to `9` is pressed, map the key value to the target RPM and set the motor running state to true.

- Calculate the elapsed time.

- If the elapsed time exceeds the interval:

  - Calculate the change in encoder position.

  - Calculate the actual RPM based on encoder pulses.

  - Compute the RPM error.

  - If the motor is running:

    - Compute the PID control output.

    - Adjust the motor speed based on the PID output.

  - If the motor is not running, stop the motor.

  - Display the set RPM, actual RPM, and error on the OLED display.

  - If there is an RPM error, display an adjustment indicator on the OLED.

- Add a small delay for keypad debounce.

3. **Motor Control Functions**:

- `forward`: Move the motor forward at the current speed.

- `backward`: Move the motor backward at the current speed.

- `stopMotor`: Stop the motor by disabling the motor driver.

- `setMotorSpeed`: Constrain the speed within the range and move the motor forward if speed is non-zero, otherwise stop the motor.

- `calculateRPM`: Calculate the RPM based on the PWM value.

4. Display Function:

- `displayRPM`: Clear the OLED display and print the set RPM, actual RPM, and error.

5. Encoder Interrupt Service Routine (`readEncoder` function):

- Read the encoder pulses and update the encoder position accordingly.

CHAPTER 7

CIRCUIT DIAGRAM AND DESCRIPTION

This setup involves connecting various components to the Arduino for controlling a motor and reading inputs from a keypad and an encoder, as well as displaying information on an I2C OLED.

7.1 Circuit design and explanations for each designs

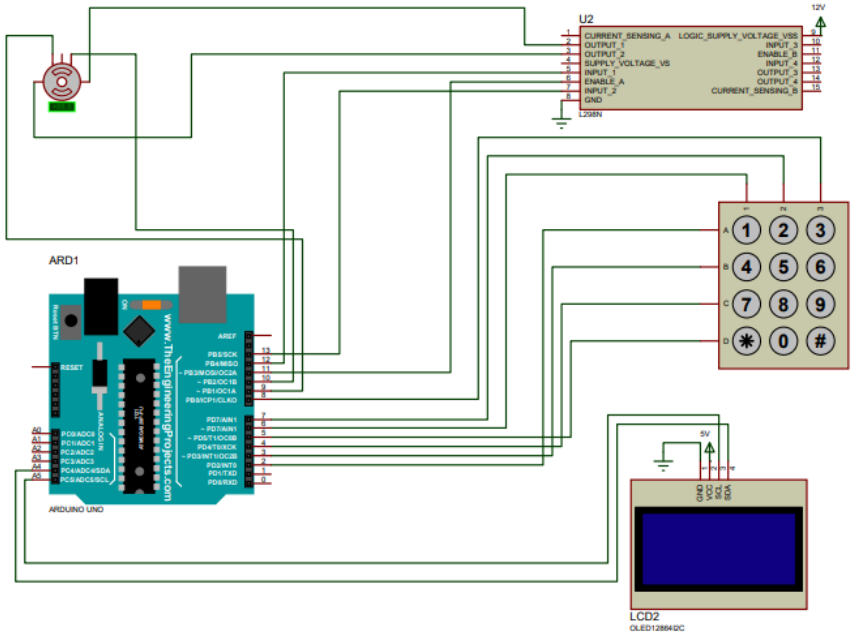


Fig: 7.1 Circuit Diagram

7.2 OLED INTERFACING CIRCUIT

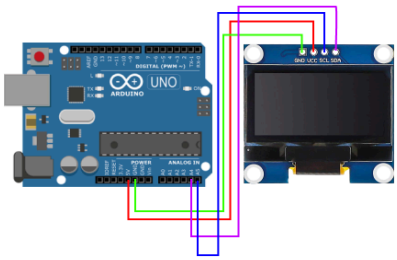


Fig: 7.2 OLED interfacing with Arduino UNO

Connecting an OLED display to an Arduino is straightforward and involves using a few essential components: the Arduino board, the OLED display, connecting wires, and optionally, a breadboard for organizing the connections. The OLED display typically has four pins: VCC (power), GND (ground), SDA (serial data), and SCL (serial clock). To wire the display to the Arduino, connect the VCC pin of the OLED to the 3.3V or 5V power pin on the Arduino, ensuring compatibility with the display's voltage requirements. The GND pin of the OLED connects to the GND pin on the Arduino. For data communication, connect the SDA pin of the OLED to the A4 pin on the Arduino and the SCL pin of the OLED to the A5 pin on the Arduino. These connections allow the Arduino to communicate with the OLED display using the I2C protocol, enabling it to send data and commands to the display for rendering text and graphics.

7.3 ARDUINO TO KEYPAD

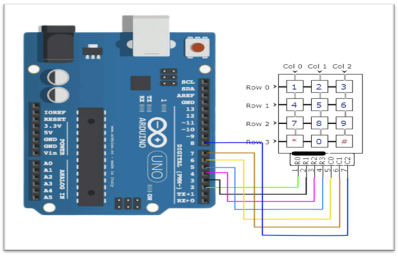


Fig: 7.2 4x3 Keypad Matrix with Arduino UNO

The keypad matrix consists of rows and columns, each connected to specific digital pins on the Arduino. When a button is pressed, it creates a connection between a row and a column pin, allowing the Arduino to detect which button was pressed.

In the accompanying code, the layout of the keypad matrix is defined, specifying the pins connected to the rows and columns of the keypad. This setup enables the Arduino to scan the keypad by sequentially energizing the row pins and reading the column pins. When a button is pressed, the Arduino identifies the specific row and column that are connected, thereby determining the pressed key.

For example, if the keypad has four rows connected to pins 2, 3, 4, and 5, and three columns connected to pins 6, 7, and 8, the code will define these connections and the matrix layout. The Arduino will continuously scan the rows and columns to detect any button press. When a key is pressed, the corresponding value is determined based on the intersection of the activated row and column.

The detected key value is then printed to the serial monitor, providing real-time feedback. This setup is ideal for projects requiring user input, such as security systems, menu navigation, and data entry applications.

7.4 ARDUINO TO L298N AND G12-N20

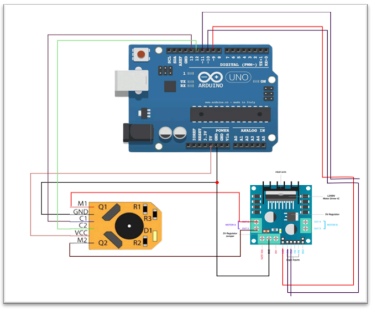


Fig: 7.4 Arduino to L298N and G12-N20

The L298N motor driver is an H-bridge motor driver, which is capable of controlling the direction and speed of DC motors by receiving signals from the Arduino. The motor driver acts as an intermediary between the Arduino and the motor, providing the necessary power and control signals to the motor.

In this setup, the Arduino sends pulse-width modulation (PWM) signals to the motor driver's ENA pin to control the motor's speed. The IN1 and IN2 pins on the motor driver receive digital signals from the Arduino to determine the motor's direction. By varying the duty cycle of the PWM signal, the voltage supplied to the motor can be adjusted, thereby controlling the motor's speed proportionally.

The speed controller in this system adjusts the voltage supplied to the G12-N20 motor based on the PWM signals. As the PWM duty cycle increases, the average voltage supplied to the motor increases, resulting in a higher motor speed. Conversely, decreasing the duty cycle reduces the motor speed. This proportional control allows for precise speed regulation of the motor, making it suitable for applications that require variable speed and direction control, such as robotic systems, conveyor belts, and other automated systems.

CHAPTER 8

RESULTS

The PID based DC motor speed controller allows the user to set the appropriate speed using the 4*3 matrix keypad. The user is able to control the rpm corresponding to the PWM values enabled through the arduino microcontroller as per the requirement. The user can choose between the PWM values between 0 to 255 for speed adjustment. The voltage applied to the motor based on the PWM value is given by the equation:

Voltage = (Supply Voltage * pwmValue) / 255.0

Theoretically to estimate the RPM of the motor based on the applied voltage is given by the equation:

RPM = (Max RPM of motor* Voltage) / Supply Voltage

To calculate the actual RPM of the motor obtained as feedback from the encoder based on the encoder pulses is given by the equation:

Actual RPM = (deltaPos * 60.0) / (CPR * (interval / 1000.0))

* deltaPos: The change in encoder position within the given interval.
* 60.0: Converts the time from seconds to minutes (since RPM is revolutions per minute).
* CPR: Counts Per Revolution of the encoder at the output shaft. For this encoder it is 840.
* interval: The time interval in milliseconds over which the encoder position is sampled

The feedback from the encoder is obtained in the form of digital pulses. to calculate the rpm the frequency of the pulses are calculated over a period of time.The quadrature encoder provides two output signals (A and B) which are 90 degrees out of phase.The number of pulses is counted using an interrupt service routine (ISR) that is triggered on changes in the state of the encoder signal A. Each change in the state of signal A corresponds to a pulse. This is handled by the ‘readEncoder’ function. The encoder provides multiple pulses per revolution (PPR), and the total counts per revolution at the output shaft is given by the CPR. In this case, CPR is 840. By dividing the total number of pulses (deltaPos) by the CPR, the number of revolutions is determined.

The PID controller is used as a feedback controller which analyzes the error in the system. The error is calculated by finding the difference between the setpoint, which is the rpm set by the user and the input, which is obtained as feedback from the encoder attached to the dc motor.

Error =Setpoint- Input

The rpm set by the user and error status is displayed in the oled display giving the user real time awareness of the speed of the motor in isolated environments.Performance and stability of the control system can be maintained by suitable calibration of PID parameters K_p,K_i,K_d using methods like Ziegler nicholls method and trial & error methods.The simulation of stability in steady state error can be displayed using MATLAB Software.

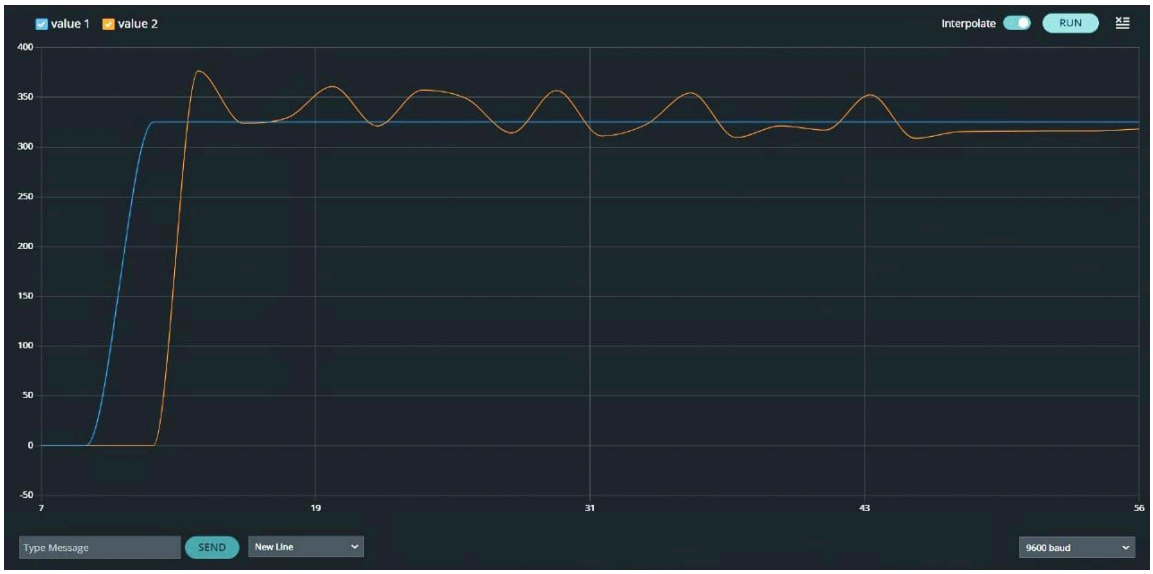


Fig 8.1 PID output response

- Set RPM
- Damped oscillatory response of the system(PID output)

Initial Transient State:When a particular keypad button is pressed, the set RPM is updated to the corresponding mapped value.Initially, there is a period where the PID controller is adjusting the motor speed to reach the set RPM.This period is characterized by fluctuations or oscillations in the PID output, representing the PID controller's effort to minimize the error between the set RPM and the actual RPM.

Error Correction:The PID error is plotted, showing the difference between the set RPM and the

actual RPM.As the motor speed approaches the desired set RPM, the error decreases.The PID output will show significant changes during this period, indicating the adjustments made by the PID controller.

Final Steady State:After some time, the system reaches a steady state where the actual RPM matches the set RPM.In this state, the error becomes minimal or zero, indicating that the PID controller has successfully stabilized the motor speed.The PID output will also stabilize, showing fewer fluctuations.

Example setup: Keypad value:3

SetRPM: 325

The feedback from the encoder is detected and the error obtained through the above calculations is corrected by the PID controller at realtime and the RPM is adjusted until the system attains steady state condition or as close as possible to the SetRPM value.

Fig 8.2 Response at keypad value '3'

The theoretical values of the SetRPM is represented as follows:

Keypad	Set RPM	Voltage	PWM Values
0	0	0	0
1	100	1.2	26

2	**212**	**2.54**	**54**
3	**325**	**3.9**	**83**
4	**437**	**5.24**	**111**
5	**550**	**6.6**	**140**
6	**662**	**7.94**	**169**
7	**775**	**9.3**	**198**
8	**887**	**10.64**	**226**
9	**1000**	**12**	**255**

Table 8.1 theoretical values of the SetRPM

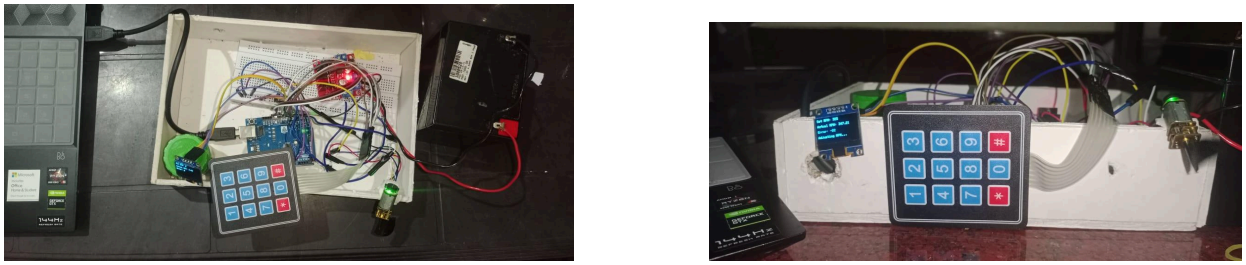


Fig 8.3 Final Result

CHAPTER 9

CONCLUSION & FUTURE SCOPE

PID-based DC motor speed controllers are versatile and essential in various applications, from industrial automation to consumer electronics. The future scope of these controllers is promising, with advancements in IoT integration, adaptive control algorithms, energy efficiency, and cost reduction. These developments will enhance the precision, reliability, and functionality of PID-based DC motor speed controllers, making them even more integral to modern technology and industry. PID based dc motor controllers have various applications in various fields:

1.**Automotive industry:**

Cruise control: where the driver requires precise and constant speed monitoring at real time

Electric Vehicles (EVs): Smooth acceleration and deceleration control.

2.**Industrial Automation:**

Conveyor belts:Maintain constant speed for material handling

Robotic Arms, Machinery

3.**Integration with IoT:**

Remote Monitoring and Control: IoT integration allows for real-time monitoring and control over the internet

4.**Advanced Algorithms**:

Adaptive PID Controllers: Adjust PID parameters in real-time based on changing conditions.

Hybrid Controllers: Combining PID with other control strategies (e.g., fuzzy logic, neural networks) for improved performance.

REFERENCES

1. K. Ravindra Sai , Ch. V. D. Krishna Vamsi , V. Vijay Kumar, B. Pradeep Kumar & P. Sudheer, ”*DC Motor speed control using PWM technique*”, Global Journal of Engineering Science and Researched, 7(4), 27–31.

2.Moleykutty George*” Speed Control of Separately Excited DC Motor”* Faculty of Engineering and Technology, Multimedia University Melaka Campus, 75450 Melaka, Malaysia.

3.Zuo Z. Liu, Fang L. Luo, and Muhammad H. Rasid, “*High performance nonlinear MIMO field weakening controller of a separately excited dc motor*,” Electric Power Systems Research, vol. 55, issue 3, Sep. 2000, pp. 157-164.

4. Nabil A. Ahmed, “*Modeling and simulation of AC-DC buck-boost converter fed dc motor with uniform PWM technique*,” Electric Power Systems Research, vol.73, issue 3, Mar. 2005, pp. 363-372

5.Heru Supriyono, Fedrik Fajar Alanro, Agus Supardi, “*Development of DC Motor Speed Control Using PID Based on Arduino and Matlab For Laboratory Trainer*”, Department of Electrical Engineering Universitas Muhammadiyah Surakarta, Surakarta, Indonesia

APPENDIX

Program Codes

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Keypad.h>
 #define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET    -1 // Reset pin # (or -1 if sharing Arduino reset pin)


 // Create an OLED display object
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
 // Define the keypad size and the keypad matrix
const byte ROWS = 4;
const byte COLS = 3;
char keys[ROWS][COLS] = {
  {'1', '2', '3'},
  {'4', '5', '6'},
  {'7', '8', '9'},
  {'*', '0', '#'}
};
 // Define the keypad pins
byte rowPins[ROWS] = {4, 5, 6, 7}; // Connect the row pins to these Arduino pins
byte colPins[COLS] = {8, 9, 10};   // Connect the column pins to these Arduino pins
 // Define motor control pins
const int ENA = 11;  // Enable A pin of the L298N motor driver
```

```
const int IN1 = 12; // IN1 pin of the L298N motor driver
const int IN2 = 13; // IN2 pin of the L298N motor driver
 // Encoder pins
const int ENCODER_A = 3; // Encoder A pin
const int ENCODER_B = 2; // Encoder B pin
volatile int encoderPos = 0;
int prevEncoderPos = 0;
unsigned long prevTime = 0;
unsigned long interval = 1000; // 1 second interval for RPM calculation
 // Motor speed control variables
const int MAX_SPEED = 255; // Maximum speed (PWM value)
const int MIN_SPEED = 0;  // Minimum speed (PWM value)
int currentSpeed = 0; // Current motor speed
int setRPM = 0; // Target RPM set by keypad
bool motorRunning = false; // Motor running state
unsigned long previousMillis = 0; // Time tracking
 // Create the Keypad object
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);
 // Motor specifications
const int MOTOR_MAX_RPM = 1000; // Maximum RPM at rated voltage
const int RATED_VOLTAGE = 12; // Rated voltage of the motor
const int SUPPLY_VOLTAGE = 12; // Ensure this is the actual voltage supplied to the motor driver
 // Encoder specifications
const int CPR = 840; // Counts per revolution at output shaft
 // PID controller variables
float kp = 1.5, ki = 0.8, kd = 0.09; // PID coefficients
float integral = 0, previousError = 0;
```

```
void setup() {
 Serial.begin(9600); // Initialize serial communication
 // Initialize the OLED display
 if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
   Serial.println(F("SSD1306 allocation failed"));
            for(;;);
 }
 display.display();
 delay(2000); // Pause for 2 seconds
// Clear the buffer
 display.clearDisplay();
 // Set motor control pins as output
 pinMode(ENA, OUTPUT);
 pinMode(IN1, OUTPUT);
 pinMode(IN2, OUTPUT);
 // Initially, stop the motor
 stopMotor();
 // Set encoder pins as input
 pinMode(ENCODER_A, INPUT);
 pinMode(ENCODER_B, INPUT);
// Attach interrupt to encoder A pin
 attachInterrupt(digitalPinToInterrupt(ENCODER_A), readEncoder, CHANGE);
}

void loop() {
  char key = keypad.getKey(); // Read the pressed key
   if (key != NO_KEY) { // Check if a key is pressed
```

```
    Serial.print("Pressed: ");
    Serial.println(key); // Print the pressed key to Serial Monitor
// Set motor speed based on key pressed
            if (key == '0') {
            stopMotor();
            setRPM = 0;
            motorRunning = false;
            } else if (key >= '1' && key <= '9') {
            setRPM = map(key - '0', 1, 9, 100, MOTOR_MAX_RPM); // Map key to RPM range (100 to
1000)
            motorRunning = true;
            }
  }
  unsigned long currentTime = millis();
  if (currentTime - prevTime >= interval) {
            int deltaPos = encoderPos - prevEncoderPos;
            prevEncoderPos = encoderPos;
            prevTime = currentTime;
// Calculate actual RPM
            float actualRPM = (deltaPos * 60.0) / (CPR * (interval / 1000.0));
            int errorRPM = setRPM - actualRPM;
            if (motorRunning) {
            // PID controller
            float error = setRPM - actualRPM;
            integral += error * (interval / 1000.0);
            float derivative = (error - previousError) / (interval / 1000.0);
            float output = kp * error + ki * integral + kd * derivative;
```

```
            previousError = error;
 // Adjust motor speed based on PID output
            int adjustedSpeed = map(output, 0, MOTOR_MAX_RPM, MIN_SPEED, MAX_SPEED);
      setMotorSpeed(constrain(adjustedSpeed, MIN_SPEED, MAX_SPEED));
            } else {
            stopMotor();
            }

  displayRPM(setRPM, actualRPM, errorRPM);
    Serial.print("Set RPM: ");
    Serial.print(setRPM);
    Serial.print(" | Actual RPM: ");
    Serial.print(actualRPM);
    Serial.print(" | Error: ");
    Serial.println(errorRPM);
// Print values for Serial Plotter
    Serial.print(setRPM);
    Serial.print(",");
    Serial.println(actualRPM);
// Show an indicator on OLED if RPM is being adjusted
            if (errorRPM != 0) {
      display.setCursor(0, 48); // Move to next line
      display.print("Adjusting RPM...");
      display.display();
            }
  }
   delay(100); // Add a small delay to debounce the keypad
```

```
}
 // Function to move the motor forward
void forward() {
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  analogWrite(ENA, currentSpeed); // Set speed
}
// Function to move the motor backward
void backward() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  analogWrite(ENA, currentSpeed); // Set speed
}
 // Function to stop the motor
void stopMotor() {
  currentSpeed = 0;
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  analogWrite(ENA, 0); // Disable motor driver (stop motor)
}
 // Function to set motor speed
void setMotorSpeed(int speed) {
  currentSpeed = constrain(speed, MIN_SPEED, MAX_SPEED); // Constrain speed within range
  if (currentSpeed == 0) {
        stopMotor(); // If speed is 0, stop the motor
  } else {
        forward(); // Move the motor forward with the new speed
```

```
  }
}
 // Function to calculate RPM from PWM value
int calculateRPM(int pwmValue) {
  // Calculate the voltage applied to the motor
  float voltage = (SUPPLY_VOLTAGE * pwmValue) / 255.0;
  // Calculate the RPM based on the voltage applied
  int rpm = (MOTOR_MAX_RPM * voltage) / RATED_VOLTAGE;
  return rpm;
}
 // Function to display RPM on the OLED screen
void displayRPM(int setRPM, float actualRPM, int errorRPM) {
  display.clearDisplay();
  display.setTextSize(1); // Set text size to 1
  display.setTextColor(SSD1306_WHITE); // Set text color to white
  display.setCursor(0, 0); // Set cursor position
  display.print("Set RPM: ");
  display.println(setRPM); // Display the set RPM
  display.setCursor(0, 16); // Move to next line
  display.print("Actual RPM: ");
  display.println(actualRPM); // Display the actual RPM
  display.setCursor(0, 32); // Move to next line
  display.print("Error: ");
  display.println(errorRPM); // Display the error RPM
  display.display();
}
 // Interrupt service routine for reading encoder pulses
```

```
void readEncoder() {
  int stateA = digitalRead(ENCODER_A);
  int stateB = digitalRead(ENCODER_B);
  if (stateA == stateB) {
        encoderPos++;
  } else {
        encoderPos--;
  }
}
```

Specification sheets

Arduino uno

https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf

L298N

https://components101.com/modules/l293n-motor-driver-module