Ministry of Education and Research of the Republic of Moldova
Technical University of Moldova
Department of Software and Automation Engineering

REPORT

Laboratory work No. 4.2

Discipline: Embedded Systems

Student: Revenco Victor, FAF - 221

Checked: asist. univ., Martiniuc A.

Chișinău 2025

Analysis of the Situation in the Field

1. Description of the Technologies Used and Application Context

Developing a modular application for a microcontroller (MCU) to control a DC motor using the L298 driver. Commands will be received through the STDIO interface (serial terminal or keypad), while also reporting the motor's parameters via the serial interface and/or LCD display.

Hardware Components

- **Microcontroller**
- **Jumper Wires**
- **USB Power Supply**
- **DC motor**
- **L298 driver**
- **Breadboard**

Software Components

- **PlatformIO with Visual Studio Code**: Integrated Development Environment (IDE)
- **C++ for Embedded Systems**: Programming language used for implementation
- **STDIO Functions (printf)**: For system reporting and monitoring
- **Serial Monitor**: For displaying system status

2. System Architecture Explanation and Solution Justification

The application is organized into several modules:

- **Main Module**: Sets up the system, creates FreeRTOS tasks, and initializes modules
- **Command Processor Module**: Offers methods to read line, process command and print help commands
- **Serial I/O Module**: Configures STDIO for serial communication
- **Motor Control Module**: Sets up motor system, sets motor power and return current power

3. Case Study

The system can have several practical uses, for example in factories for conveyor belts which would need to be set to spin the belt at a certain speed and in a certain direction which would automate the task of moving products throughout the factory to each worker.

4. Design

1. Architectural Sketch and Component Interconnection

- Serial Monitor for system status output
- DC Motor for the main hardware piece

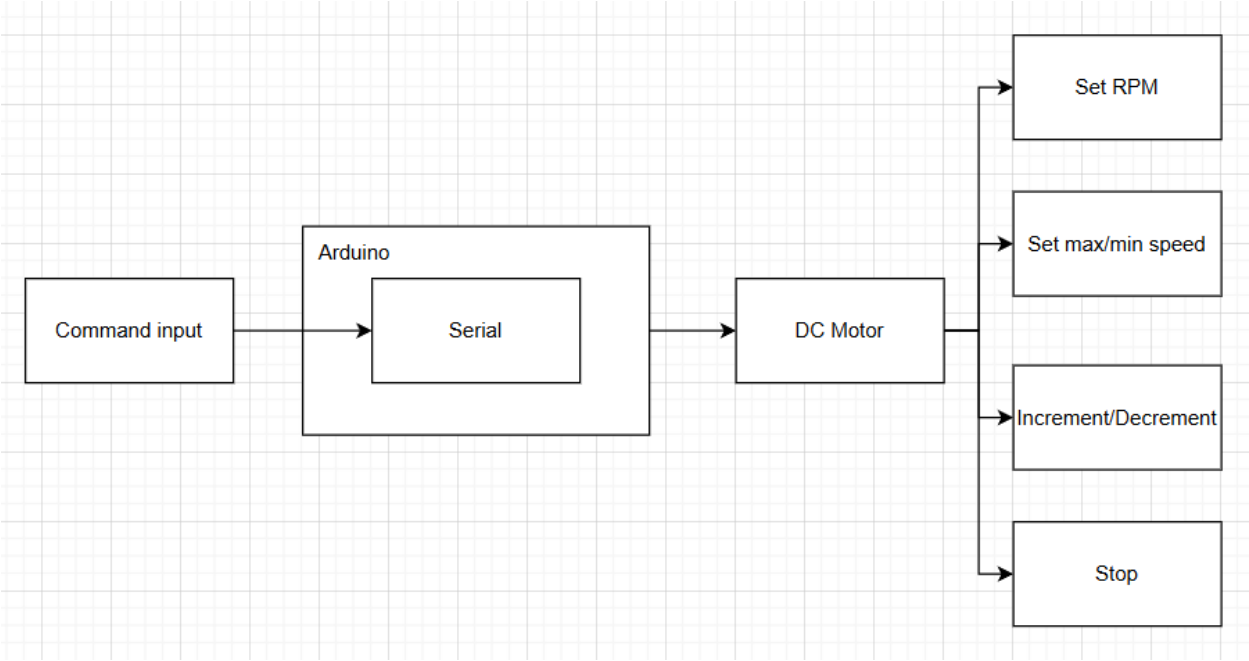


Figure 1 Architectural sketch

Component Description and Their Roles:

- **Microcontroller**: Core processing unit executing tasks
- **DC Motor:** Spins according to the values set

The sketch shows how the components interact.

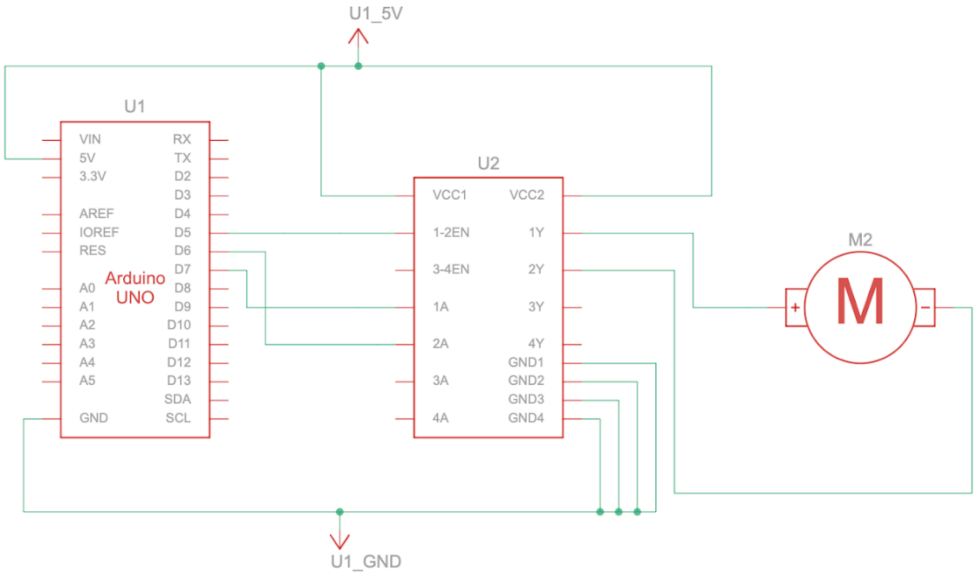


Figure 2 Electrical schematic

This image shows an **Arduino Uno** connected to a driver module through the breadboard. The driver itself is connected to the DC motor.

2. Schematic diagrams

To understand the system's behavior, a Flowchart and a Finite State Machine (FSM) are used.

Flowchart – Serial Command Processing

The FSM includes the following states:

1. Idle – The motor is stopped
2. Forward – The motor is running forward
3. Reverse – The motor is running in reverse

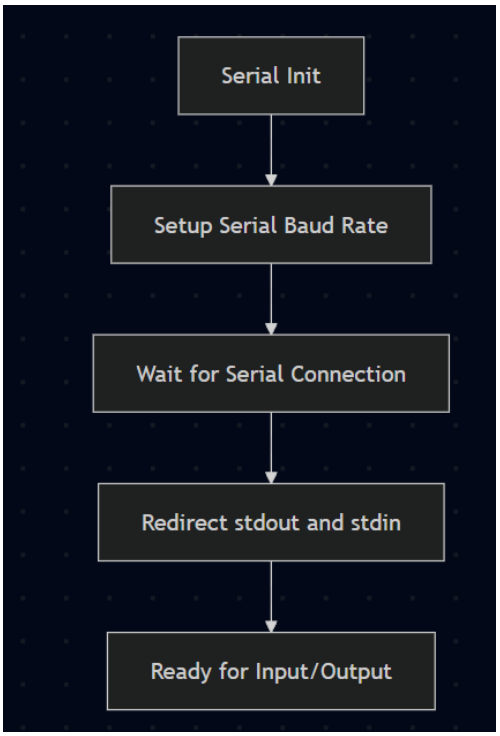


Figure 3 SerialIO

The **SerialIO Flowchart** shows the steps involved in setting up serial communication. The system begins by initializing the serial interface with a specified baud rate. It then waits for the serial connection to be established and redirects the standard output (`stdout`) and input (`stdin`) to the serial interface, allowing the program to interact with the user via the serial console.

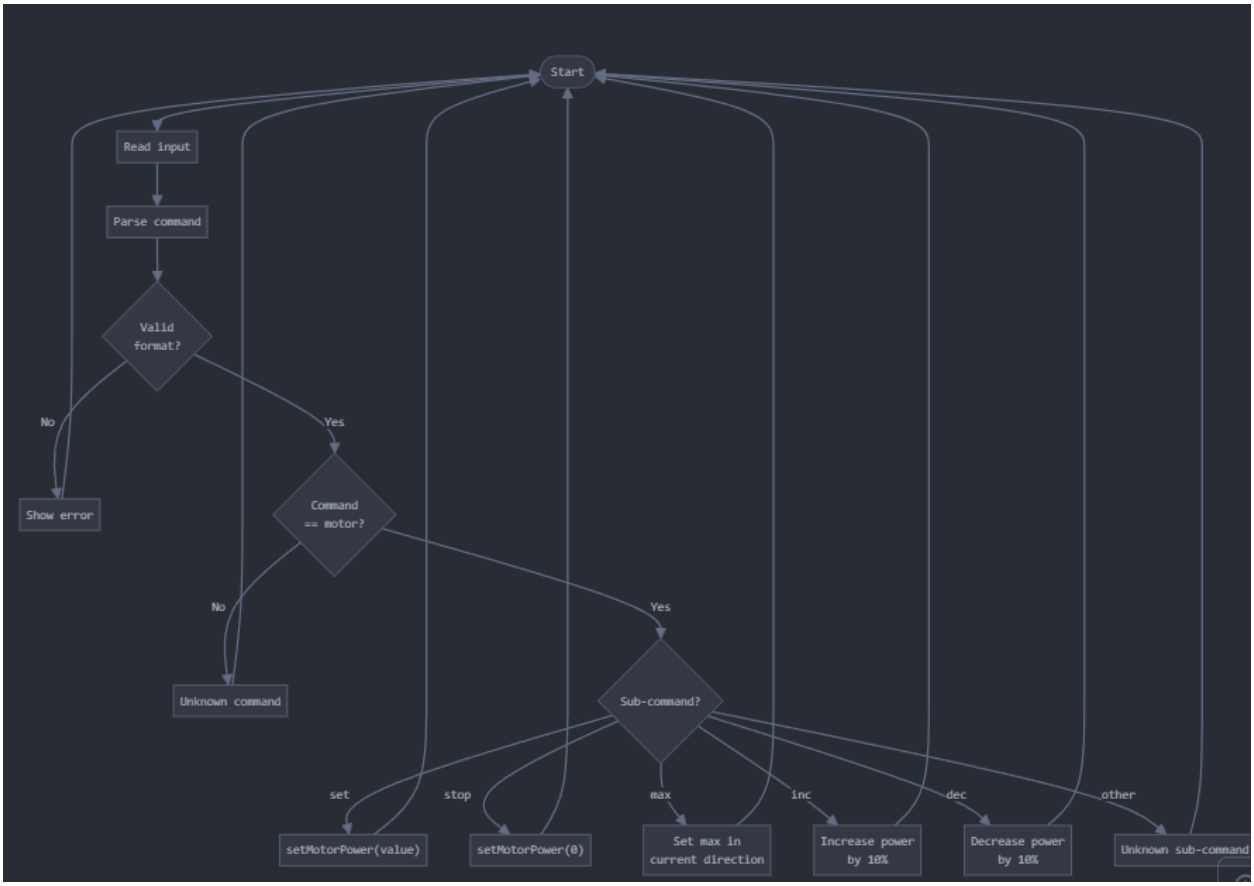


Figure 4 Command processing

The flow begins with reading the command line input, then parsing it using sscanf. Based on the parsing result, the system determines the command type and sub-command type. For the "motor" command, various sub-commands are handled differently, such as setting power, stopping, maximizing power, or incrementing/decrementing power. Invalid inputs or unknown commands result in appropriate error messages before returning to the prompt for a new command.

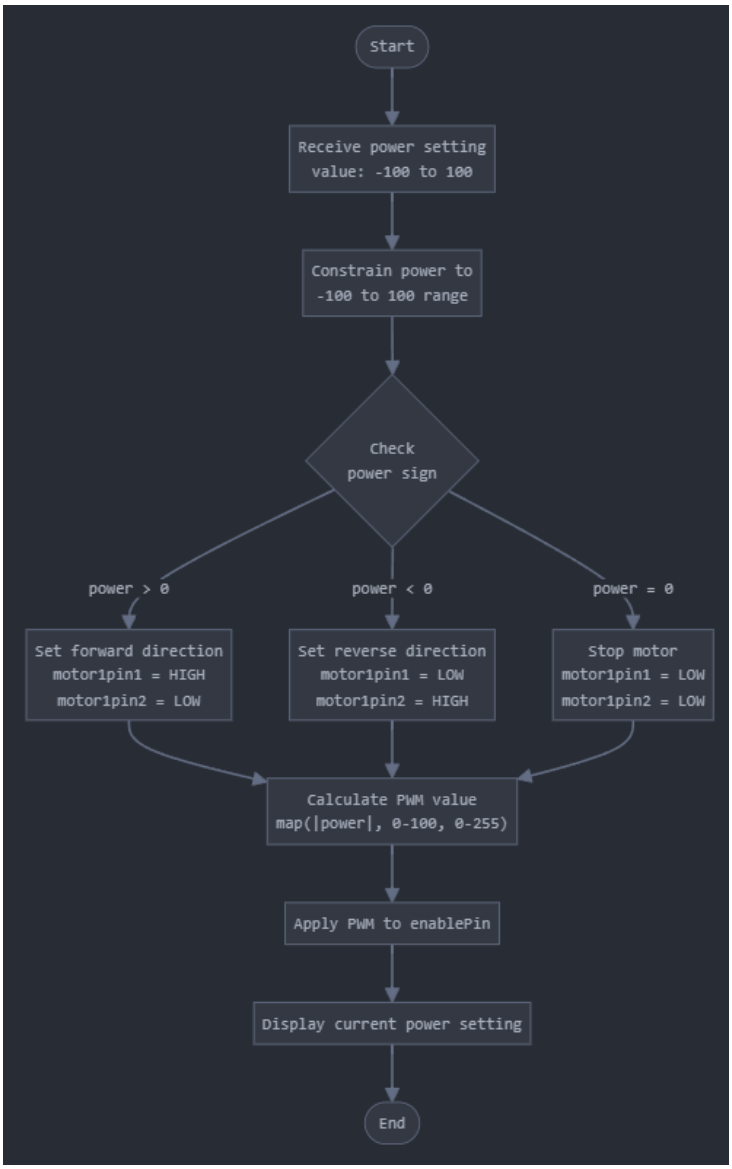


Figure 5 Motor control logic

When a power value is received, it first constrains the value to the valid range (-100 to 100). Based on the sign of the power, it sets the appropriate direction pins (forward, reverse, or stop). The absolute power value is then mapped from the 0-100 range to the 0-255 PWM range and applied to the enable pin. Finally, the current power setting is displayed to the user.

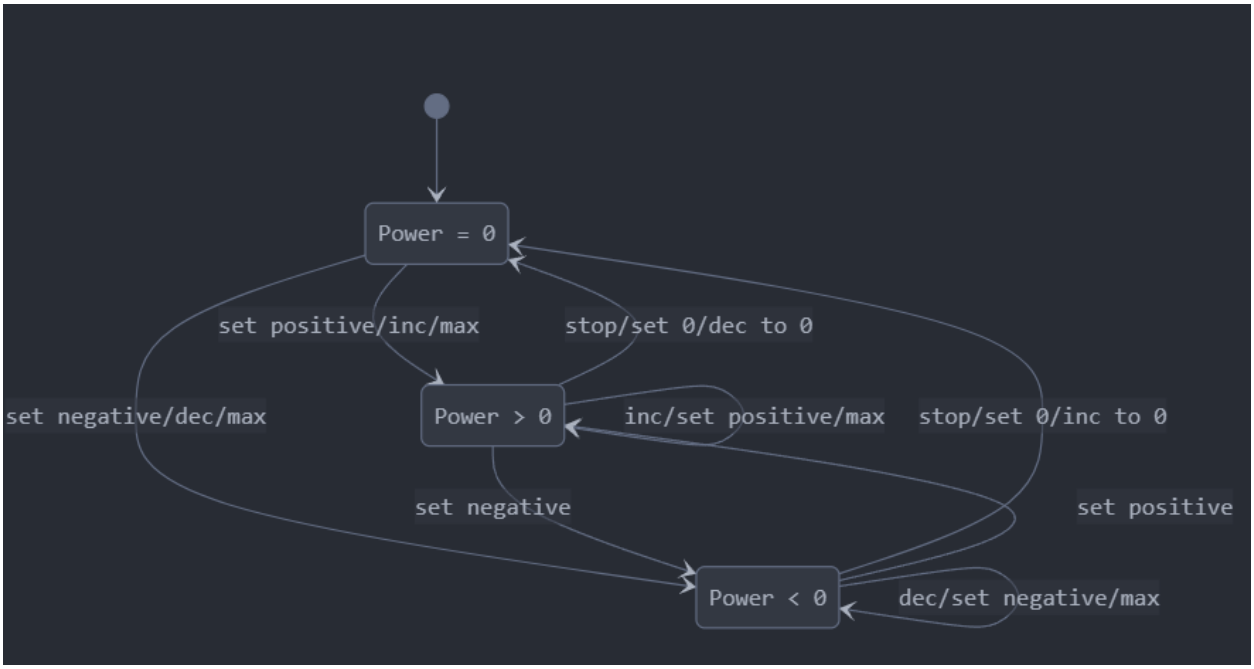


Figure 6 FSM

Modular implementation

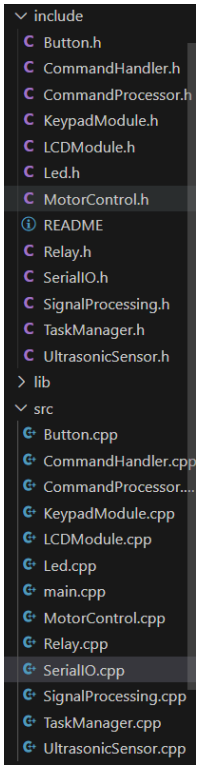
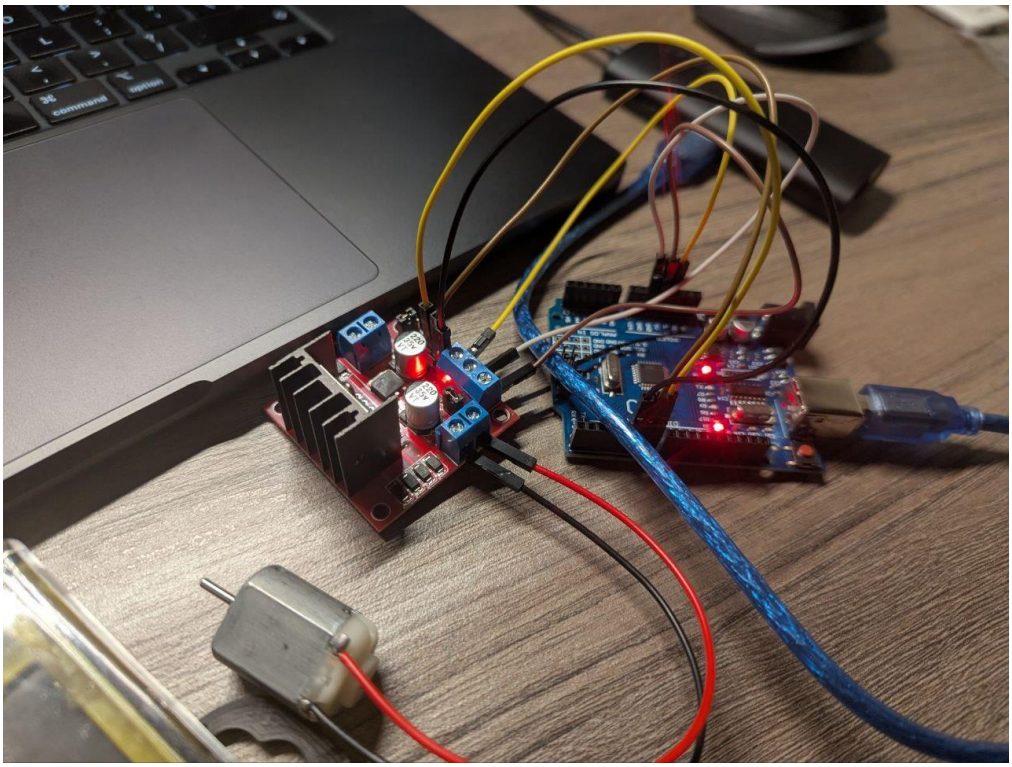
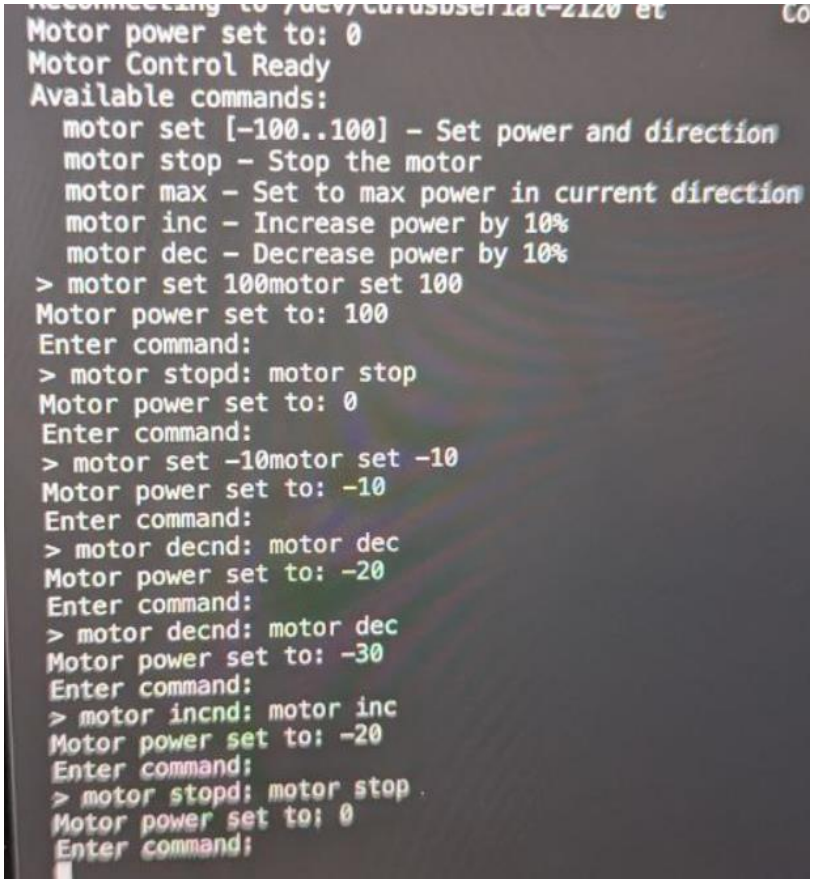


Figure 7 Project organization

Results





Conclusions

In this laboratory work, I developed a modular application for controlling a DC motor using an L298 driver and a microcontroller, with commands received through the STDIO interface and status reporting via Serial. I implemented analog control of the motor's power using PWM, as well as commands for adjusting and stopping the motor. I created separate drivers following abstraction principles to ensure modularity and reusability. Through this activity, I became familiar with DC motor control techniques.

Bibliography

1. Official Arduino Documentation
    - Arduino Reference – Serial Communication https://www.arduino.cc/reference/en/#communication
    - Arduino Mega 1280 Pinout & Datasheet https://docs.arduino.cc/hardware/mega-1280
2. PlatformIO Official Documentation
    - PlatformIO for Arduino Development https://docs.platformio.org/en/latest/platforms/atmelavr.html
3. TUM Courses
    - Introducere în Sistemele Embedded și Programarea Microcontrolerelor
    - Principiile comunicației seriale și utilizarea interfeței UART

Appendix

1. **GitHub**: https://github.com/Kipitokisk/SI_Lab

```
#include "SerialIO.h"
#include "MotorControl.h"
#include "CommandProcessor.h"


void setup() {
    serialInit();
    setup_motor();
    printCommandHelp();
}

void loop() {
    if (Serial.available() > 0) {
        readLine();
      }
      delay(10);
}
```

```
#include "CommandProcessor.h"
#include "MotorControl.h"
#include <string.h>
```

```
char inputBuffer[64];

void processCommand(const char* commandStr) {
  char command[32];
  char subCommand[32] = "";
  int value = 0;

  // Parse the command using sscanf
  if (sscanf(commandStr, "%s %s %d", command, subCommand, &value) >= 1) {
    printf("> %s", command);
    if (strlen(subCommand) > 0) printf(" %s", subCommand);
    if (value != 0 || strcmp(subCommand, "set") == 0) printf(" %d", value);
    printf("\n");

    if (strcmp(command, "motor") == 0) {
      if (strcmp(subCommand, "set") == 0) {
        setMotorPower(value);
      }
      else if (strcmp(subCommand, "stop") == 0) {
        setMotorPower(0);
      }
      else if (strcmp(subCommand, "max") == 0) {
        // Set to max power in current direction
        int currentPower = getCurrentPower();
        if (currentPower >= 0) {
          setMotorPower(100);
        } else {
          setMotorPower(-100);
        }
      }
      else if (strcmp(subCommand, "inc") == 0) {
        int currentPower = getCurrentPower();
        setMotorPower(currentPower + 10);
      }
      else if (strcmp(subCommand, "dec") == 0) {
        int currentPower = getCurrentPower();
        setMotorPower(currentPower - 10);
      }
      else {
        printf("Unknown sub-command\n");
      }
    }
    else {
      printf("Unknown command\n");
    }
```

```
  }
  else {
    printf("Invalid command format\n");
  }
}

// Read a line from Serial using stdio functions
void readLine() {
  int i = 0;
  char c;

  printf("Enter command: ");

  // Clear the buffer
  memset(inputBuffer, 0, sizeof(inputBuffer));

  // Read until newline or buffer is full
  while (i < (int)(sizeof(inputBuffer) - 1)) {
    c = getchar();

    // Echo the character back
    putchar(c);

    if (c == '\n' || c == '\r') {
      break;
    }

    inputBuffer[i++] = c;
  }

  inputBuffer[i] = '\0';  // Null terminate the string

  if (strlen(inputBuffer) > 0) {
    processCommand(inputBuffer);
  }
}

void printCommandHelp() {
  printf("Motor Control Ready\n");
  printf("Available commands:\n");
  printf("  motor set [-100..100] - Set power and direction\n");
  printf("  motor stop - Stop the motor\n");
  printf("  motor max - Set to max power in current direction\n");
  printf("  motor inc - Increase power by 10%%\n");
  printf("  motor dec - Decrease power by 10%%\n");
```

```
}
```

```
#include "MotorControl.h"
#include <stdio.h>

// Motor pins
int motor1pin1 = 6;
int motor1pin2 = 7;
int enablePin = 5;
int currentPower = 0; // Range: -100 to 100

void setup_motor() {
  // Configure motor pins
  pinMode(motor1pin1, OUTPUT);
  pinMode(motor1pin2, OUTPUT);
  pinMode(enablePin, OUTPUT);

  // Initial state: motor stopped
  setMotorPower(0);
}

void setMotorPower(int power) {
  // Constrain power to -100 to 100 range
  currentPower = constrain(power, -100, 100);

  // Set direction based on power sign
  if (currentPower > 0) {
    digitalWrite(motor1pin1, HIGH);
    digitalWrite(motor1pin2, LOW);
  } else if (currentPower < 0) {
    digitalWrite(motor1pin1, LOW);
    digitalWrite(motor1pin2, HIGH);
  } else {
    // Stop if power is 0
    digitalWrite(motor1pin1, LOW);
    digitalWrite(motor1pin2, LOW);
  }

  // Convert percentage (0-100) to PWM (0-255)
  int pwmValue = map(abs(currentPower), 0, 100, 0, 255);
  analogWrite(enablePin, pwmValue);

  printf("Motor power set to: %d\n", currentPower);
}
```

```
int getCurrentPower() {
  return currentPower;
}
```