Cre – AID Labs

Problem: Write code for the control system for a motor controller that uses the ESP32 microcontroller. The system must have the following features:

a. The motor speed is the input to the controller.

b. The motor speed shouldn't change, irrespective of the load applied.

c. The motor uses an H-Bridge driver which, in turn, is driven using PWM by the ESP32.

d. The motor has an incremental hall-effect pulse encoder (See OE-37 Encoder) which is interfaced with ESP32 for feedback.

e. The controller may communicate to the host device using serial port, where the user can set

The speed and direction in the following format:

i. F<speed> for forward, speed is a value from 0 to 255. E.g. F40
ii. B<speed> for backward, speed is a value from 0 to 255. E.g. B200
iii. S for stop

f. Any kind of mapping from the speed values to angular speed of the motor may be used.

GitHub Link: https://github.com/KarthikT23/Cre-AID-Labs

Code:

```
// Motor control pins

const int PWM_Pin = 5;     // PWM pin for motor speed control

const int dir1_Pin = 18;   // H-Bridge input 1

const int dir2_Pin = 19;   // H-Bridge input 2

const int encoderA_Pin = 2;  // Encoder channel A pin

const int encoderB_Pin = 3;  // Encoder channel B pin
```

```
// PID controller variables
float kp = 1.0, ki = 0.1, kd = 0.05;  // PID gains
float integral = 0, derivative = 0, lastError = 0;
unsigned long lastTime = 0;

// Motor speed variables
int targetSpeed = 0;
volatile int encoderCount = 0;



void setupMotor();
void processSerialCommand();
void runPIDControl();
int parseSpeed();
void setMotorDirection(bool forward);
void setMotorSpeed(int speed);
void stopMotor();
void handleEncoderInterrupt();
int calculateEncoderSpeed();

void setup() {
  setupMotor();
}

void loop() {
  processSerialCommand();
```

```
  runPIDControl();
}

void setupMotor() {
  pinMode(PWM_Pin, OUTPUT);
  pinMode(dir1_Pin, OUTPUT);
  pinMode(dir2_Pin, OUTPUT);
  pinMode(encoderA_Pin, INPUT_PULLUP);
  pinMode(encoderB_Pin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(encoderA_Pin), handleEncoderInterrupt, RISING);
  attachInterrupt(digitalPinToInterrupt(encoderB_Pin), handleEncoderInterrupt, RISING);

  Serial.begin(9600);

  stopMotor();
}

void processSerialCommand() {
  if (Serial.available() > 0) {
    char command = Serial.read();
    switch (command) {
      case 'F':
        setMotorDirection(true);
        targetSpeed = parseSpeed();
        break;
      case 'B':
```

```
      setMotorDirection(false);
      targetSpeed = parseSpeed();
      break;
    case 'S':
      stopMotor();
      break;
    default:
      Serial.println("Invalid command");
  }
 }
}

void runPIDControl() {
 unsigned long currentTime = millis();
 float deltaTime = (currentTime - lastTime) / 1000.0;
 int currentSpeed = calculateEncoderSpeed();
 float error = targetSpeed - currentSpeed;

 integral += error * deltaTime;
 derivative = (error - lastError) / deltaTime;
 lastError = error;
 lastTime = currentTime;

 // PID output mapping and constraint
 int pwmValue = targetSpeed + kp * error + ki * integral + kd * derivative;
 pwmValue = constrain(pwmValue, 0, 255);
```

```
  setMotorSpeed(pwmValue);
}

int parseSpeed() {
  String speedString = Serial.readStringUntil('\n');
  return speedString.toInt();
}

void setMotorDirection(bool forward) {
  digitalWrite(dir1_Pin, forward ? HIGH : LOW);
  digitalWrite(dir2_Pin, forward ? LOW : HIGH);
}

void setMotorSpeed(int speed) {
  analogWrite(PWM_Pin, speed);
}

void stopMotor() {
  setMotorSpeed(0);
}

void handleEncoderInterrupt() {
  if (digitalRead(encoderB_Pin) == HIGH) {
    encoderCount++;
  } else {
```

```
    encoderCount--;
  }
}

int calculateEncoderSpeed() {
  int currentCount = encoderCount;
  int countsPerSecond = static_cast<int>(currentCount / (millis() - lastTime) * 1000.0);
  return countsPerSecond;
}
```

Assumptions:

Hardware Configuration:

i. The motor is controlled by an H-Bridge driver connected to the PWM pins on the ESP32 (pins 5, 18, and 19).
ii. An incremental hall-effect pulse encoder is connected to the ESP32 to provide feedback on the motor's speed (pins 2 and 3).

Serial Communication: The user can send commands ('F, 'B, 'S') and speed values to the system via a serial interface.

PID Controller:

1. The PID controller is used to adjust the motor speed based on the difference between the target speed and the actual speed obtained from the encoder feedback.
2. The PID parameters (kp, ki, kd) are assumed to be initially set to 1.0, 0.1, and 0.05 respectively.

Speed Mapping:

1) The motor speed is mapped to a PWM value within the range of 0 to 255.
2) Constraints are applied to ensure that the PWM value stays within the valid range.

Theoretical Basis:

PID Control:

The basic idea behind a PID controller is to read a sensor, then compute the desired actuator output by calculating proportional, integral, and derivative responses and summing those three components to compute the output.

1. Proportional (P): The proportional term adjusts the motor speed in proportion to the current speed error (difference between target and actual speed).
2. Integral (I): The integral term accounts for accumulated past errors, preventing a steady-state error and improving system stability.
3. Derivative (D): The derivative term anticipates future errors by considering the rate of change of the current error.

Encoder Feedback:

i. The encoder interrupts track the pulses generated by the hall-effect encoder, providing information about the motor's position and speed.
ii. The speed is calculated based on the time between encoder pulses.

Serial Communication:

1) The user communicates with the system through the serial port, providing commands ('F', 'B', 'S') and corresponding speed values.
2) Commands are interpreted in the loop, adjusting the target speed accordingly.

PWM Control: The analogWrite function is used to control the PWM signal to the motor, adjusting the duty cycle and the motor speed.

Engineering Calculations:

PID Controller Calculations:

1) Proportional Term (P): P = Kp x error, where Kp is the proportional gain and error is the difference between the target speed and the actual speed.
2) Integral Term (I): I = I + Ki x error x delta_t, where Ki is the integral gain and delta_t is the time elapsed since last iteration.
3) Derivative Term (D): D = Kd x (error-lastError)/delta_t, where Kd is the derivative gain.

PWM Mapping and Constrain:

1) The PID output, along with the target speed, is mapped to a PWM value within the range of 0 to 255.
2) The PWM value is constrained to stay within the valid PWM range.

Encoder Speed Calculation:

1) Encoder interrupts track the pulses generated by the hall-effect encoder.
2) The speed is calculated based on the time between encoder pulses: speed = encoderCount/delta_t, where delta_t is the time elapsed since the last encoder count.

Serial Communication Parsing:

1) The user provides speed values through the serial interface.
2) The parseSpeed function reads the speed value as a string and converts it to an integer for further use.

Formula

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_p \frac{de}{dt}$$

$u(t)$ = PID control variable

K_p = proportional gain

$e(t)$ = error value

K_i = integral gain

de = change in error value

dt = change in time

Circuit Connections:

ESP32 to Motor Driver (H-Bridge):

1. Connect PWM_Pin (e.g., GPIO 5) on the ESP32 to the PWM input of the motor driver.
2. Connect dir1_Pin and dir2_Pin (e.g., GPIO 18 and GPIO 19) on the ESP32 to the input pins of the H-Bridge to control the motor direction.

ESP32 to Incremental Hall-Effect Pulse Encoder:

1. Connect encoder_Pin and encoder_Pin (e.g., GPIO 2 and GPIO 3) on the ESP32 to the channels A and B of the incremental hall-effect pulse encoder.
2. Check encoder's power and ground are connected properly.

Motor Driver to DC Motor:

1. Connect the motor to the output terminals of the H-Bridge. Connect one motor terminal to dir1_Pin and the other terminal to dir2_Pin.
2. Connect the other terminals of the motor to the H-Bridge output.

Power Supply:

1. Connect the power supply's positive terminal to the positive voltage input of the H-Bridge.
2. Connect the power supply's negative terminal to the ground of the ESP32, motor driver, and the negative terminal of the incremental hall-effect pulse encoder

Block Diagram: