Implementation of Closed-Loop DC Motor Speed Control

Sourish Vijayamadhavan (24140), DESE, IISc

I PROJECT DESCRIPTION

Closed-loop speed control is essential for DC motors to maintain a consistent and precise speed despite variations in load, supply voltage, or external disturbances. Unlike open-loop control, which cannot compensate for such variations, a closed-loop system continuously monitors the motor speed and adjusts the control input accordingly. This can be implemented using a digital Proportional-Integral (PI) controller, where the motor speed is measured using an encoder or tachometer, and the error between the desired and actual speed is fed into the PI controller. The proportional term provides immediate correction based on the error, while the integral term eliminates steady-state error by considering the accumulated past errors. The output of the PI controller is used to adjust the duty cycle of a PWM signal that controls the motor driver, ensuring smooth and stable speed regulation.

The user can set the desired speed of the DC motor by varying a potentiometer, whose output terminal is connected to the ADC of the ATTiny85 microcontroller. The measured voltage from the ADC is mapped to a reference speed ranging from 1000 to 3000 RPM.

The actual speed of the DC motor is measured using the AMT103 incremental rotary encoder. The rotary encoder is a device which helps measure the angular speed and direction of rotation of the DC motor by generating 2 quadrature pulse trains A and B. The phase shift between A and B helps determine the direction of rotation, while the frequency of the pulse train A (number of rising edges in a second) determines the angular speed of the motor. The microcontroller uses interrupts to continuously count the number of rising edges on pulse train A every 0.5 seconds, and uses this information to update the measured speed of the motor in real-time.

The digital PI controller implemented in the ATTiny85 takes the desired speed from the ADC and actual speed from the encoder, to regulate the duty cycle of the PWM signal used to drive the DC motor. The proportional and integral constants are chosen so as to achieve zero steady-state error along with high-speed transient tracking.

The speed of the motor is regulated by providing a PWM signal whose pulse width can be varied by the microcontroller. Increasing the pulse width of the PWM signal increases the average DC power provided to the motor, hence increasing its speed, while reducing the pulse width has an opposite effect. The control signal generated by the PI controller is used to regulate the pulse width of the PWM signal, to achieve the desired speed.

The output pins of the microcontroller can source currents of upto a few 100 μA, while the DC motor has an average current draw of 1 A and requires a voltage supply of 12 V, which can damage the microcontroller pins if connected directly. Hence, a current amplification circuit using the IRF620 power MOSFET is used to provide the desired voltage and current drive to the DC motor. This circuit provides an amplified version of the 5 V PWM signal from the microcontroller to the gate of the power MOSFET, and has an average current drive of 1 A, which can be used to drive the motor safely.

II CIRCUIT SCHEMATIC

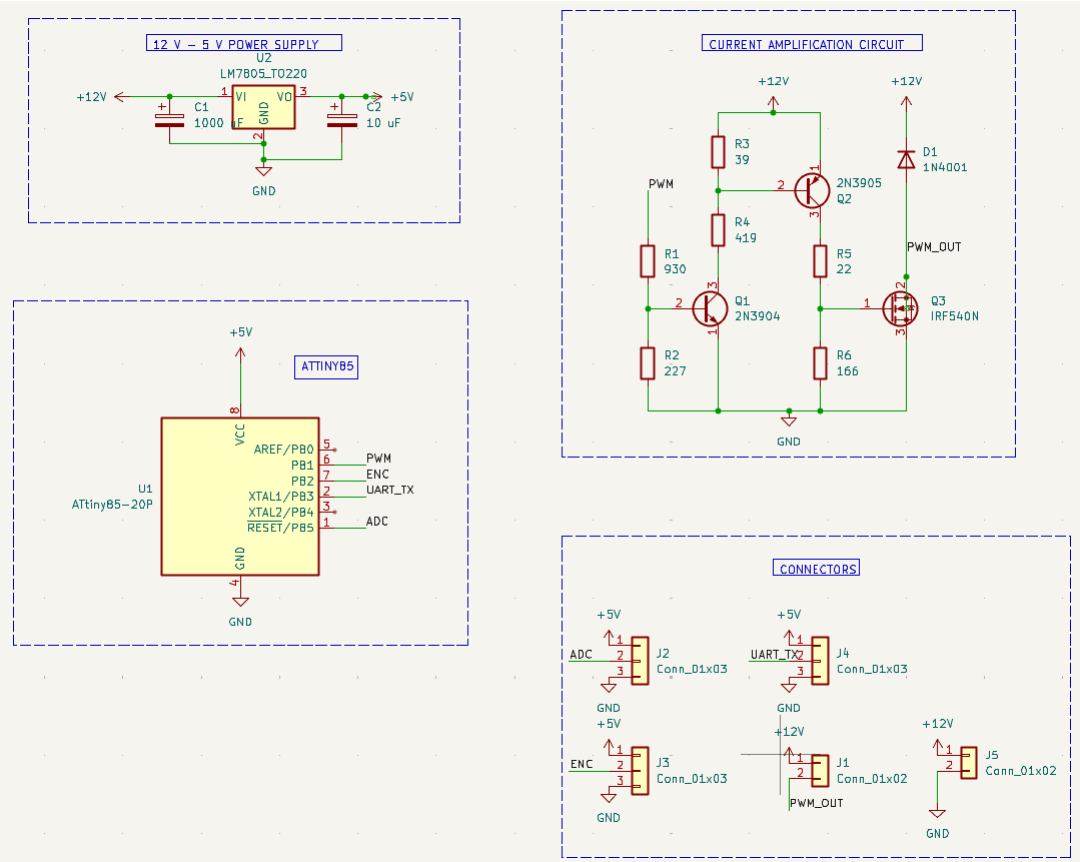

Figure 1: Circuit Schematic

The terminals of the DC motor are connected across the 12 V supply and PWM_OUT pins, in parallel with the freewheeling diode. The A pin from the optical encoder is connected to ENC on PB2, while the centre tap of the potentiometer is connected to the ADC on PB5.

The above circuit is implemented on a single-layer printed circuit board, and the PCB layout is as follows -

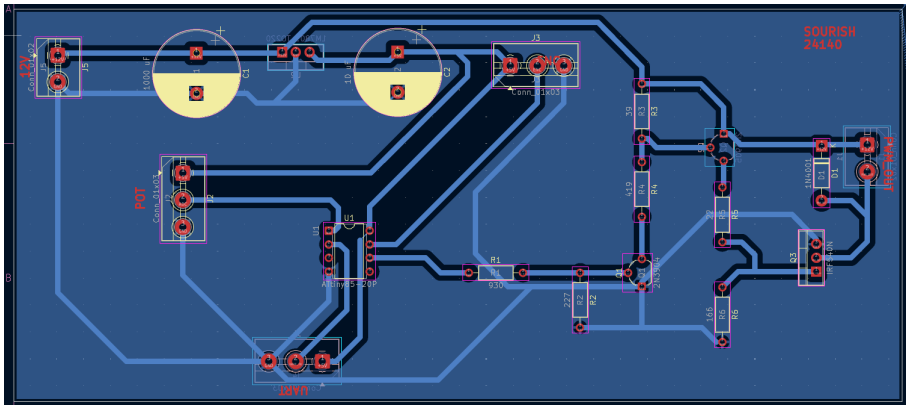

Figure 2: PCB for Closed-Loop DC Motor Speed Control

III DESIGN EQUATIONS

The current amplification circuit must be designed so as to provide a current drive capability of 1 A at the drain terminal of the MOSFET. The components used for this current amplification circuit are calculated as follows -

The average gate charge of the IRF620 power MOSFET when in triode region is found to be 14 nC as per the datasheet. The input waveform from the microcontroller is a square wave of 20 kHz frequency and 50% pulse width. Assuming the rise time required for turning on the MOSFET is 10% of the duty cycle, we get t_{on} = 2 us. Hence, the maximum gate current that must be supplied to the MOSFET is $I_{gm} = \frac{2Q_g}{t_{on}}$ = 30 mA, assuming a worst-case gate charge of 30 nC

The current through R_6 must be atleast twice this value, so as to support rapid transitions from on to off state and vice versa. Hence, I_{R6} = 60 mA and I_{R5} = 90 mA. As per the datasheet, the gate-to-source voltage of the power MOSFET must be atleast 10 V to support a wide range of load currents, hence $R_6 = \frac{10V}{60mA} = 166\,\Omega$. Assuming zero drop across the PMOS due to saturation, $R_5 = \frac{12V-10V}{90mA} = 22\,\Omega$

The base current of Q2 (2N4403 PNP BJT) is calculated by assuming it is in saturation, and its h_{fe} = 20 according to datasheet specifications. Hence, the base current of Q2 is $I_{b2} = \frac{2I_{c2}}{h_{fe2}} = \frac{2\times 90\,mA}{20} = 9\,mA$. To support quick rise and fall in waveform, the current through R_3 must be twice this value, hence I_{R3} = 18 mA and I_{R4} = 27 mA. Due to the emitter base junction, V_R3 = 0.7 V and V_R4 = 11.3 V, hence $R_3 = \frac{0.7V}{18mA} = 39\,\Omega$ and $R_4 = \frac{11.3V}{27mA} = 419\,\Omega$

The base current of Q1 (2N2222 NPN BJT) is found using its saturation h_{fe}, which is 35 according to the datasheet. Hence, $I_{b1} = \frac{2I_{c1}}{h_{fe1}} = \frac{2\times 27\,mA}{35} = 1.54\,mA$. Due to base-emitter drop of NPN BJT, V_{R2} = 0.7 V, and $R_2 = \frac{0.7V}{3mA} = 227\,\Omega$. Finally, as the output from the microcontroller digital I/O pin has a maximum value of 5 V, $R_1 = \frac{5V-0.7V}{4.5mA} = 930\,\Omega$

The values of the resistors used in the circuit can be summarized as follows -

Parameter	Value
R_1	930 Ω
R_2	227 Ω
R_3	39 Ω
R_4	419 Ω
R_5	22 Ω
R_6	166 Ω

IV COMPONENT SPECIFICATIONS

The list of components used and their specifications are given below -

Component	Specifications
DC Motor	12 V, 1.2 A, 3750 RPM
Optical Encoder	AMT103, 2048 PPR
Potentiometer	0-10 kΩ
Microcontroller	ATTiny85
5 V Regulator	LM7805
NPN BJT	2N2222
PNP BJT	2N4403
Power MOSFET	IRF620
R_1	930 Ω
R_2	227 Ω
R_3	39 Ω
R_4	419 Ω
R_5	22 Ω
R_6	166 Ω
C_1	1000 μF
C_2	10 μF

V SOFTWARE CODE

The ATTiny85 is programmed using AVR C Code, in the PlatformIO environment of Visual Studio Code.

```
void setupMillis() {
    TCCR1 = (1 << CTC1) | (1 << CS12) | (1 << CS10); // CTC mode with OCR1A as TOP, Prescaler 64
    OCR1A = 124;  // (8 MHz / 64) / 125 = 1000 Hz (1 ms tick)
    //OCR1A = 62;
    //OCR1A = 249; //trying to get 2x delay
    TIMSK |= (1 << OCIE1A); // Enable Timer1 compare interrupt
}

// Configure External Interrupt for Encoder
void setupInterrupt() {
    GIMSK |= (1 << INT0); // Enable INT0 interrupt
    MCUCR |= (1 << ISC00) | (1 << ISC01); // Rising edge trigger
    sei(); // Enable global interrupts
}

ISR(INT0_vect) {
    pulseCount++;
}

// Timer1 ISR (fires every 1ms to simulate millis())
ISR(TIMER1_COMPA_vect) {
    millisCounter++;
    PORTB ^= (1<<PB4);
}

// Returns the current time in milliseconds
unsigned long millis() {
    uint8_t oldSREG = SREG;
    cli(); // Disable interrupts to read atomic value
    unsigned long currentMillis = millisCounter;
    SREG = oldSREG; // Restore interrupt state
    return currentMillis;
}
```

Figure 3: Configuring ISR on INT0 and timer for 0.5 second duration

The centre tap of the 10 kΩ potentiometer is given as input to ADC0 (PB5) of the ATTiny85. The input to the ADC thus varies from 0 to 5 V, which can be used to set the desired speed from 1000 to 3000 RPM. The ADC is configured to run in polling mode, rather than free-running, to avoid changing the reference speed frequently, and is called once every second.

```
void ADC_init(){
    ADMUX = (0 << REFS1) | (0 << REFS0) | (0 << MUX2) | (0 << MUX1) | (0 << MUX0);

    ADCSRA = (1 << ADEN)  | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
}

uint16_t readADC(){
    ADCSRA |= (1<<ADSC);
    while(ADCSRA&(1<<ADSC));
    return ADC;
}
```

Figure 4: Configuring ADC0 for polling mode operation, and defining ADC read function which runs once a second

The speed of the DC motor is controlled by the ATTiny85 generating a PWM signal of 20 kHz frequency on pin PB1. The PWM signal is initialized with the frequency of 20 kHz.

```
// Configure Timer0 for PWM on PB1 (OC0B)
void setupPWM() {
    TCCR0A = (1 << COM0B1) | (1 << WGM00) | (1 << WGM01); // Fast PWM, non-inverting on OC0B
    TCCR0B = (1 << CS01); // Prescaler 8
    OCR0B = (uint8_t)duty; // Set initial duty cycle
}
```

Figure 5: Configuring non-inverting PWM of 20 kHz frequency

The reference and actual RPM of the DC motor is sent over UART to the console, to log and track the transient and steady-state response of the closed-loop control system. The ATTiny85 sends data to the serial port from its PB3 pin.

```
// Software UART: Send one character
void softUART_sendChar(char c) {
    uint8_t i;
    cli();
    PORTB &= ~(1 << TX_PIN); // Start bit
    _delay_us(1000000 / BAUD_RATE);

    for (i = 0; i < 8; i++) {
        if (c & (1 << i)) PORTB |= (1 << TX_PIN);
        else PORTB &= ~(1 << TX_PIN);
        _delay_us(1000000 / BAUD_RATE);
    }

    PORTB |= (1 << TX_PIN); // Stop bit
    _delay_us(1000000 / BAUD_RATE);
    sei();
}

// Software UART: Send a string
void softUART_sendString(const char* str) {
    while (*str) {
        softUART_sendChar(*str++);
    }
}

// Convert integer to string and send over UART
void softUART_sendInt(int num) {
    char buffer[10];
    itoa(num, buffer, 10);
    softUART_sendString(buffer);
}
```

Figure 6: Configuring non-inverting PWM of 20 kHz frequency

The while loop in the main code contains the PI control logic required to implement closed-loop speed control, which runs once every 0.5 s (chosen for better steady state and transient tracking). The values for Kp and Ki are chosen for fast transient tracking and accurate stead-state tracking (here Kp = 0.07, Ki = 0.0375). A two-tap moving window averaging filter is added to filter the measured speed from the encoder, thereby reducing the effect of high-frequency noise in the steady-state tracking

```
int main() {
    cli(); // Disable interrupts during setup

    ADC_init();
    setupGPIO();
    setupPWM();
    setupMillis();
    setupInterrupt();

    sei(); // Enable global interrupts

    while (1) {
        unsigned long currentTime = millis();
        if (currentTime - prevTime >= 1000) { // Calculate RPM every second
            cli();
            //rpm = (pulseCount / (float)pulsesPerRevolution) * 60.0;
            rpm = ((pulseCount*60.0)/((float)pulsesPerRevolution))*2;
            //Edit
            rpm_avg = (rpm+prev_rpm)/2;
            prev_rpm = rpm;
            //
            pulseCount = 0;
            sei();
            adcVal = readADC();
            referenceRPM = (adcVal-ADC_min)*(RPM_max-RPM_min)/(ADC_max-ADC_min)+1000.0;
            if(referenceRPM<1000){
                referenceRPM = 1000.0;
            }
```

Figure 7: Preprocessing of measured speed to reduce high-frequency noise

```
            // PI Controller
            //error = referenceRPM - rpm;
            //Edit
            error = referenceRPM - rpm_avg;
            //
            duty = Kp * error + Ki * integral;
            integral += error;
            if (integral > 5000) integral = 5000;
            if (integral < -5000) integral = -5000;
            if (duty < 0) duty = 0;
            if (duty > 255) duty = 255;

            OCR0B = (uint8_t)duty; // Update PWM duty cycle

            softUART_sendInt((int)referenceRPM);
            softUART_sendChar(',');
            //Edit
            softUART_sendInt((int)rpm_avg);
            //softUART_sendInt((int)rpm);
            softUART_sendChar('\n');


            prevTime = currentTime;
        }
    }
}
```

Figure 8: Implementation of digital PI controller

VI RESULTS

The reference speed is swept using the potentiometer from 1000 to 3000 RPM, and the measured speed of the DC motor is plotted as against the input reference speed on Serial Studio. The transient response of the system is given below -

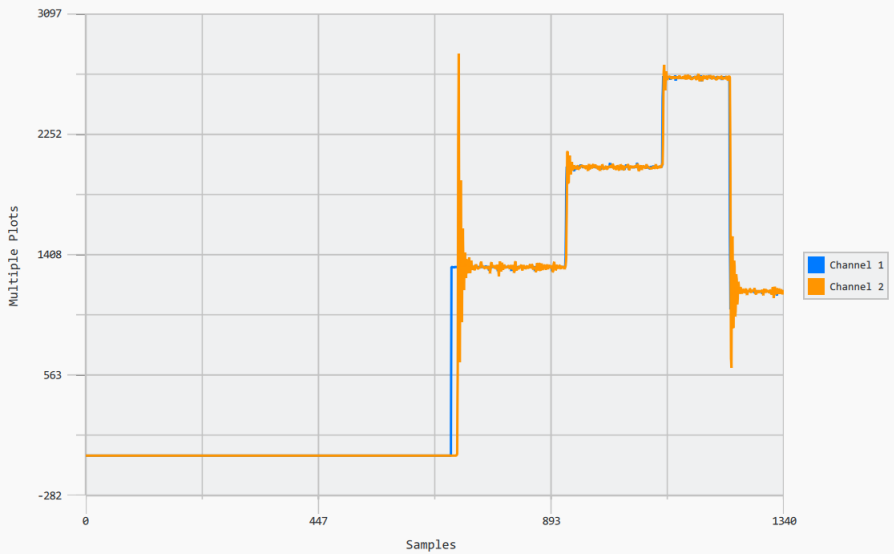

Figure 9: Steady-state tracking of reference RPM (blue) by actual motor RPM (orange)

Thus, it is observed that the steady-state tracking of the PI control system is accurate, and it is able to respond to changes in the reference RPM as long as it lies between 1000 and 3000 RPM.

VII CONCLUSION

Thus, a closed-loop system has been designed to control the speed of the DC motor from 1000 to 3000 RPM. This project made use of the following design principles -

1. Design of 12 V and 5 V power supply to power the motor driver circuit and the microcontroller, respectively.
2. Design of current amplification circuit to provide the current drive of 1.2 A required by the motor.
3. Design and tuning of PI controller to achieve high-speed and accurate response.

4. Digital preprocessing of input signals for reduced output noise.

These principles are still being used today by embedded and control engineers to design high-speed, fault-tolerant control systems for various industrial and domestic applications.