

SYSTEMS AND CONTROL ENGINEERING

SC649: EMBEDDED CONTROLS AND ROBOTICS

Assignment 1

Team

PRANAV GUPTA (22B2179)
ROHAN MEKALA (22B2106)
SAHIL SUDHAKAR (210010055)

Instructor

PROF. LEENA VACHHANI
leena.vachhani@iitb.ac.in



September 9, 2024

Contents

1	Setup	2
2	Problem Statement 1	3
2.1	Aim	3
2.2	Methodology	3
2.3	Observations	4
2.4	Results & Conclusions	4
3	Problem Statement 2	5
3.1	Aim	5
3.2	Methodology	5
3.3	Observations	7
3.4	Results	8
3.5	Conclusions	8

1 Setup

1. We have used IN3 and IN4 pins of the **L298N** motor driver to provide differential inputs to the Arduino to rotate the motor in clockwise/anticlockwise direction.
2. Since we have used IN3 and IN4, we have used the ENB pin of the motor driver to provide power to the motor via the Arduino.
3. C1 and C2 are the inputs that we receive from the encoder based on the direction of the motor(encoder) rotation.

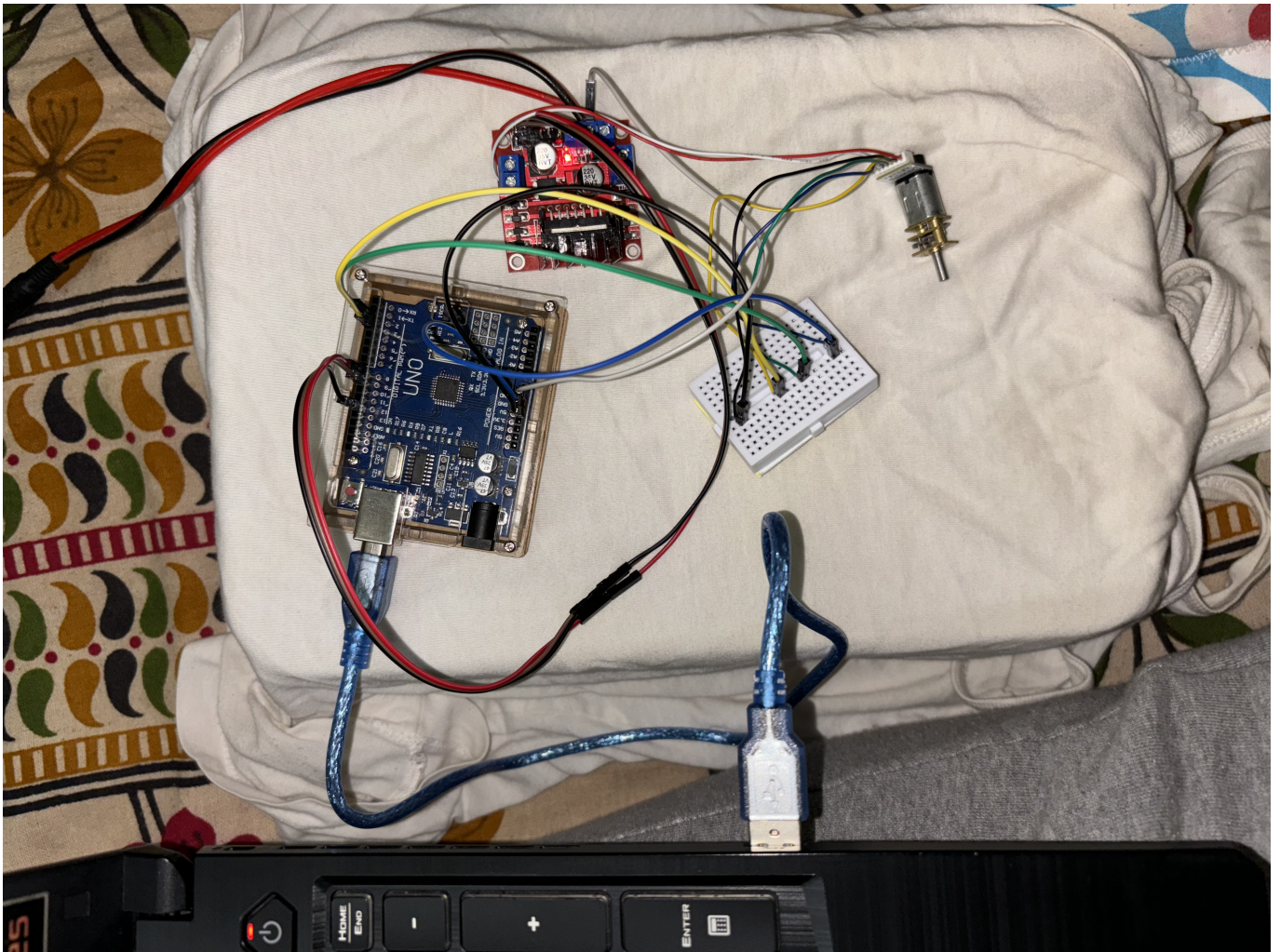


Figure 1: photograph of setup

2 Problem Statement 1

2.1 Aim

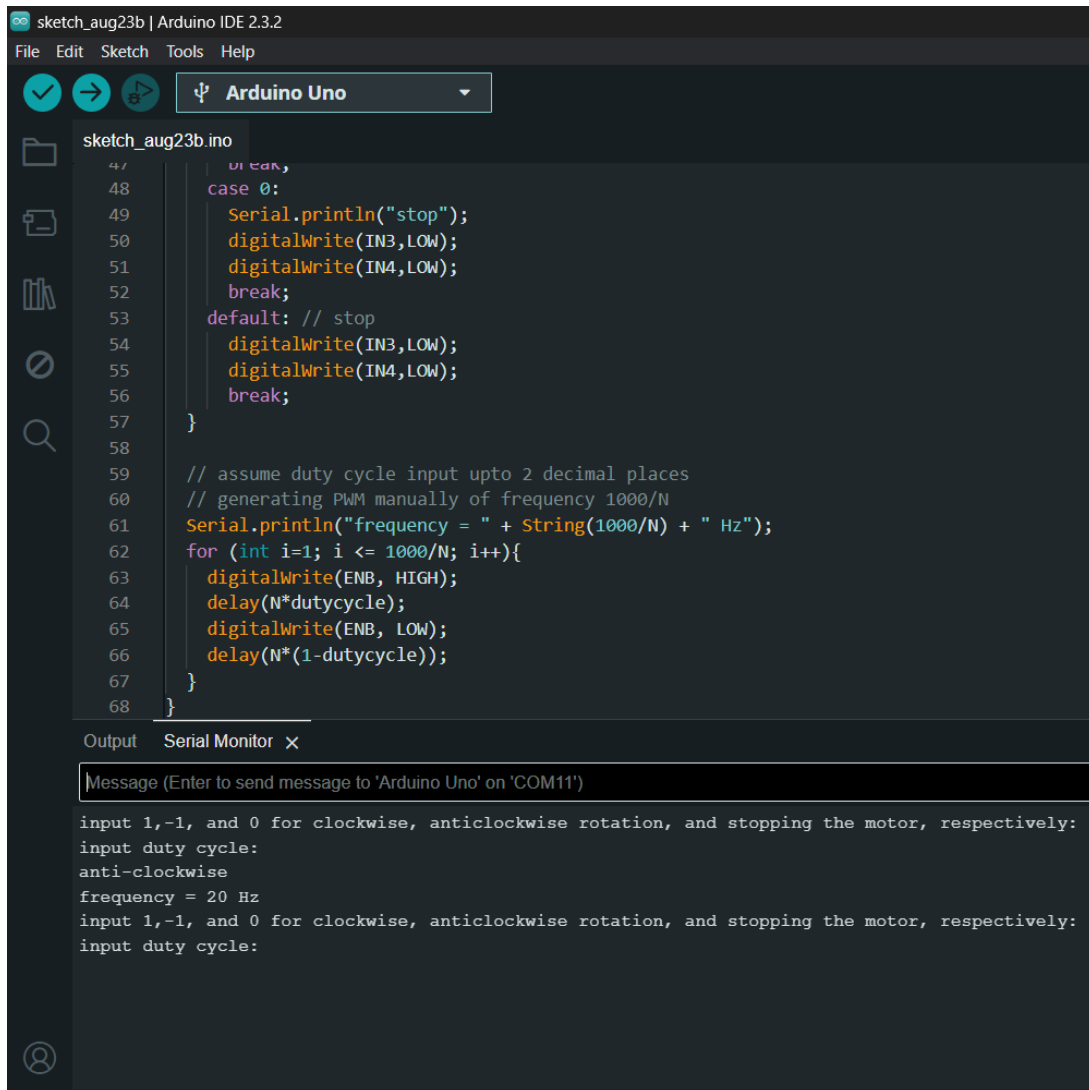
Implement an open-loop controller, as shown in Figure 1 to control the speed and direction of the motor.

1. Take user input from the serial monitor to determine the direction of motor rotation. You can take input as 1,-1, and 0 for clockwise, anticlockwise rotation, and stopping the motor, respectively.
2. Take user input from the serial monitor for the required Duty Cycle.
3. Generate PWM signals based on user input and control the speed and direction of the motor.
4. Try generating PWM signals at different frequencies and observe the effects on motor behavior. Share your results in detail and your conclusions from it

2.2 Methodology

1. To take user input from the serial monitor, we used the `Serial.available()` function to take the user's duty cycle and motor direction inputs.
2. We have set up a switch case block to allow the user the choice of clockwise rotation, anti-clockwise rotation or stopping the motor by inputting 1, -1 and 0 respectively.
3. For clockwise rotation, "HIGH LOW" input via IN3 and IN4 is required whereas for anti-clockwise rotation, "LOW HIGH" input is required. For stopping the motor, either "LOW LOW" should suffice since the encoder works on a differential input model to decide the direction of rotation.
4. Along with that, we have also asked the user to input the duty cycle. The assumption we have made here is that the input would be up to 2 decimal places (for eg: 0.25 duty cycle).
5. To generate PWM signals at different frequencies, we established a logic using the duty cycle. As the duty cycle is $\frac{T_{on}}{T_{on}+T_{off}}$ and frequency is simply $\frac{1}{T_{on}+T_{off}}$, we defined a constant N at the start of the code which sets the period of the PWM signal in milliseconds, so that the PWM frequency came out to be $1000/N$ Hz. We used a for loop that goes from $i=1$ to $i=50/N$ and in each iteration, we first gave HIGH input through ENB, then placed a delay of $N * \text{duty cycle}$, which would correspond to T_{on} , then gave Low input through ENB and placed a delay of $N * (1 - \text{duty cycle})$, which would correspond to T_{off} .

2.3 Observations



The screenshot displays the Arduino IDE 2.3.2 interface. The top menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with icons for checking, running, and uploading code, along with a dropdown menu for the board, currently set to 'Arduino Uno'. The main editor window shows a sketch named 'sketch_aug23b.ino' with the following C++ code:

```
47     break;
48   case 0:
49     Serial.println("stop");
50     digitalWrite(IN3, LOW);
51     digitalWrite(IN4, LOW);
52     break;
53   default: // stop
54     digitalWrite(IN3, LOW);
55     digitalWrite(IN4, LOW);
56     break;
57   }
58
59   // assume duty cycle input upto 2 decimal places
60   // generating PWM manually of frequency 1000/N
61   Serial.println("frequency = " + String(1000/N) + " Hz");
62   for (int i=1; i <= 1000/N; i++){
63     digitalWrite(ENB, HIGH);
64     delay(N*dutycycle);
65     digitalWrite(ENB, LOW);
66     delay(N*(1-dutycycle));
67   }
68 }
```

Below the code editor is the 'Serial Monitor' window, which is currently empty. The 'Output' tab is selected, and the 'Serial Monitor' title bar is visible. The 'Serial Monitor' window shows the following output:

```
input 1,-1, and 0 for clockwise, anticlockwise rotation, and stopping the motor, respectively:
input duty cycle:
anti-clockwise
frequency = 20 Hz
input 1,-1, and 0 for clockwise, anticlockwise rotation, and stopping the motor, respectively:
input duty cycle:
```

Figure 2: Serial Monitor output for taking direction and duty cycle from user and generating PWM signal for motor open loop control manually

2.4 Results & Conclusions

By taking user input via the `Serial.available()` function, we were able to successfully control the direction of the motor by varying the differential inputs via IN3 and IN4 from “HIGH LOW” to “LOW HIGH” and vice versa and also control the speed of the motor using the duty cycle input by the user. We were able to use the relation between frequency and duty cycle and place appropriate delays between the HIGH and LOW inputs through ENB to vary the PWM frequency.

3 Problem Statement 2

3.1 Aim

Interface the encoder with the motor, as shown in Figure 1, and read the data from it

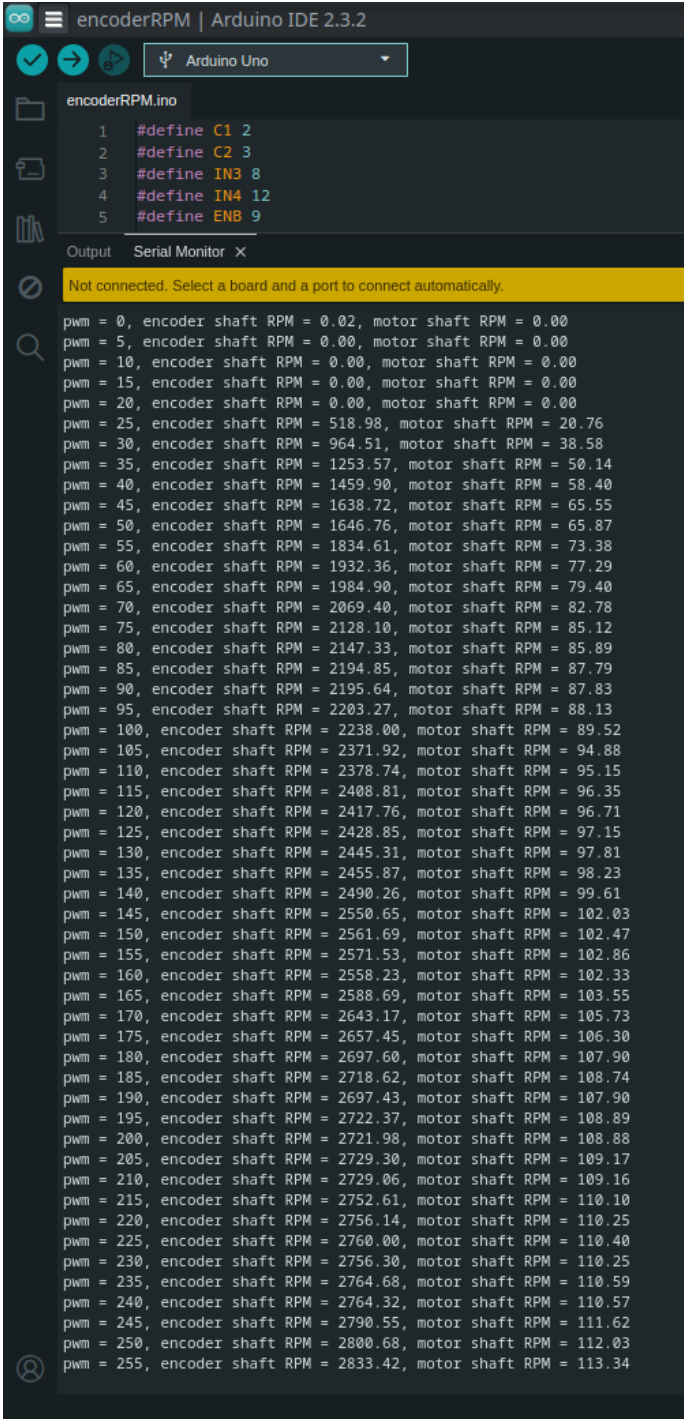
1. Do the necessary calculations and calibrations and based on that, calculate the motor RPM and make a map of duty-cycle to encoder count to motor speed(RPM).
2. Use the encoder data to determine if the motor is rotating clockwise or anticlockwise.
3. Display the necessary information on the serial monitor

3.2 Methodology

1. The state of the encoder is the input we get from C1 and C2. We have declared variables `acurr` and `bcurr` which are the current state of the encoder as they read the input from C1 and C2 respectively, and `aprev` and `bprev` which are the previous state of the encoder. These 4 variables have been initially defined as 0. We have also initialized a variable `tickCounter` to 0 which will act as our tick counter.
2. We observed that the C1 and C2 directional inputs (states) follow a certain pattern
 - Clockwise: 00 10 11 01 00
 - Anticlockwise: 00 01 11 10 00
3. We have created a nested if structure. The outer if checks whether `acurr` is different from `aprev` or `bcurr` is different from `bprev`. If either is different, that means the encoder (motor) is rotating, otherwise the C1 and C2 inputs would simply be 00 00 and so on.
4. According to our convention, clockwise rotation is negative and anticlockwise rotation is positive.
5. We have identified that the state pattern for clockwise rotation follows a pattern such that `acurr = 1-bprev` and `bcurr = aprev`.
6. The inner loop checks if this condition is satisfied. If it does, then it is classified as a clockwise rotation and the tick counter is reduced by 1 (since clockwise rotation is negative according to our configuration). If the condition is not satisfied, then it is classified as an anticlockwise rotation and the tick counter is incremented by 1 (since anticlockwise rotation is positive according to our configuration).
7. At the end of each `void loop()` iteration, `aprev` and `bprev` are given the values of `acurr` and `bcurr`.
8. We marked a line on the encoder and rotated it 50 times and then divided the number of ticks by 50 to get an average estimate of the ticks per encoder rotation which we found to be 25. We defined a variable `ticksPerEncoderRotation` to store this value.
9. To get the ticks per motor rotation, we multiplied the ticks per encoder rotation with the motor shaft gearbox ratio and stored this value in the variable `ticksPerMotorRotation`.
10. We have implemented a logic which provides the motor RPM at intervals of 1 second.
11. We have defined a for loop which has the PWM value as the loop variable which goes from 0 to 255 in steps of 5. We provide this PWM value to the motor via the ENB pin of the L298N motor driver.

12. For each iteration of this for loop, we have defined a nested for loop which runs 10 times.
13. In the inner for loop, we have placed a while loop which will execute when `updateRPM()` returns false. The `updateRPM()` function uses two variables, `currentTime` and `lastUpdateTime`, which we have initially initialised to 0, to keep track of intervals of 1 second. The `updateRPM()` function does this in the following manner: In `void setup()`, we have initialised `lastUpdateTime` to `millis()`, which is a function which gives us the current system time in milliseconds. In each iteration of the while loop, when `updateRPM()` function is called at the start of the iteration to check the loop condition, it will return false till the difference between `currentTime` and `lastUpdateTime` is lesser than 1 second and in each while iteration, the `currentTime` variable will be updated using the `millis()` function and we will also update the number of encoder ticks using the `updateEncoderTicks()` function which has the same `tickCounter` logic that we had used in the previous problem statement.
14. As soon as the difference between `currentTime` and `lastUpdateTime` is equal to 1 second, we exit the while loop and we store the value of RPM for that 1 second interval in the `avgRPM` variable while also dividing by `N` as we are calculating average RPM.
15. Here, `N` is basically the number of 1 second intervals for which we are calculating RPM, adding those values and dividing by `N` to report the average RPM for `N` seconds. In our code, we have arbitrarily chosen `N` to be 10.
16. So essentially we are incrementing the PWM input from 0 to 255 in steps of 5, and in each iteration, we are calculating 10 RPM values for 10 “1 second intervals” which we are averaging to report the average RPM for that 10 second duration.
17. In the `updateRPM()` function, the logic for calculating the RPM for that 1 second interval is as follows: Once a 1 second interval has passed, we will have the updated value of `tickCounter` variable (with the help of `updateEncoderTick()` function) which is essentially the `ticksPerSecond` as our time interval is of 1 second. We then reset the `tickCounter` value to 0 for the next 1 second interval. We then reset the `tickCounter` and the `avgRPM` variable to 0 for the next 1 second interval. We simply multiply the `ticksPerSecond` with `[360/ticksPerMotorRotation]` (which is basically the degree of rotation for each tick) and divide by 60. The `lastUpdateTime` variable is then given the value of the `currentTime` variable for the next 1 second interval.

3.3 Observations



```
encoderRPM.ino
1 #define C1 2
2 #define C2 3
3 #define IN3 8
4 #define IN4 12
5 #define ENB 9

Output Serial Monitor x
Not connected. Select a board and a port to connect automatically.

pwm = 0, encoder shaft RPM = 0.02, motor shaft RPM = 0.00
pwm = 5, encoder shaft RPM = 0.00, motor shaft RPM = 0.00
pwm = 10, encoder shaft RPM = 0.00, motor shaft RPM = 0.00
pwm = 15, encoder shaft RPM = 0.00, motor shaft RPM = 0.00
pwm = 20, encoder shaft RPM = 0.00, motor shaft RPM = 0.00
pwm = 25, encoder shaft RPM = 518.98, motor shaft RPM = 20.76
pwm = 30, encoder shaft RPM = 964.51, motor shaft RPM = 38.58
pwm = 35, encoder shaft RPM = 1253.57, motor shaft RPM = 50.14
pwm = 40, encoder shaft RPM = 1459.90, motor shaft RPM = 58.40
pwm = 45, encoder shaft RPM = 1638.72, motor shaft RPM = 65.55
pwm = 50, encoder shaft RPM = 1646.76, motor shaft RPM = 65.87
pwm = 55, encoder shaft RPM = 1834.61, motor shaft RPM = 73.38
pwm = 60, encoder shaft RPM = 1932.36, motor shaft RPM = 77.29
pwm = 65, encoder shaft RPM = 1984.90, motor shaft RPM = 79.40
pwm = 70, encoder shaft RPM = 2069.40, motor shaft RPM = 82.78
pwm = 75, encoder shaft RPM = 2128.10, motor shaft RPM = 85.12
pwm = 80, encoder shaft RPM = 2147.33, motor shaft RPM = 85.89
pwm = 85, encoder shaft RPM = 2194.85, motor shaft RPM = 87.79
pwm = 90, encoder shaft RPM = 2195.64, motor shaft RPM = 87.83
pwm = 95, encoder shaft RPM = 2203.27, motor shaft RPM = 88.13
pwm = 100, encoder shaft RPM = 2238.00, motor shaft RPM = 89.52
pwm = 105, encoder shaft RPM = 2371.92, motor shaft RPM = 94.88
pwm = 110, encoder shaft RPM = 2378.74, motor shaft RPM = 95.15
pwm = 115, encoder shaft RPM = 2408.81, motor shaft RPM = 96.35
pwm = 120, encoder shaft RPM = 2417.76, motor shaft RPM = 96.71
pwm = 125, encoder shaft RPM = 2428.85, motor shaft RPM = 97.15
pwm = 130, encoder shaft RPM = 2445.31, motor shaft RPM = 97.81
pwm = 135, encoder shaft RPM = 2455.87, motor shaft RPM = 98.23
pwm = 140, encoder shaft RPM = 2490.26, motor shaft RPM = 99.61
pwm = 145, encoder shaft RPM = 2550.65, motor shaft RPM = 102.03
pwm = 150, encoder shaft RPM = 2561.69, motor shaft RPM = 102.47
pwm = 155, encoder shaft RPM = 2571.53, motor shaft RPM = 102.86
pwm = 160, encoder shaft RPM = 2558.23, motor shaft RPM = 102.33
pwm = 165, encoder shaft RPM = 2588.69, motor shaft RPM = 103.55
pwm = 170, encoder shaft RPM = 2643.17, motor shaft RPM = 105.73
pwm = 175, encoder shaft RPM = 2657.45, motor shaft RPM = 106.30
pwm = 180, encoder shaft RPM = 2697.60, motor shaft RPM = 107.90
pwm = 185, encoder shaft RPM = 2718.62, motor shaft RPM = 108.74
pwm = 190, encoder shaft RPM = 2697.43, motor shaft RPM = 107.90
pwm = 195, encoder shaft RPM = 2722.37, motor shaft RPM = 108.89
pwm = 200, encoder shaft RPM = 2721.98, motor shaft RPM = 108.88
pwm = 205, encoder shaft RPM = 2729.30, motor shaft RPM = 109.17
pwm = 210, encoder shaft RPM = 2729.06, motor shaft RPM = 109.16
pwm = 215, encoder shaft RPM = 2752.61, motor shaft RPM = 110.10
pwm = 220, encoder shaft RPM = 2756.14, motor shaft RPM = 110.25
pwm = 225, encoder shaft RPM = 2760.00, motor shaft RPM = 110.40
pwm = 230, encoder shaft RPM = 2756.30, motor shaft RPM = 110.25
pwm = 235, encoder shaft RPM = 2764.68, motor shaft RPM = 110.59
pwm = 240, encoder shaft RPM = 2764.32, motor shaft RPM = 110.57
pwm = 245, encoder shaft RPM = 2790.55, motor shaft RPM = 111.62
pwm = 250, encoder shaft RPM = 2800.68, motor shaft RPM = 112.03
pwm = 255, encoder shaft RPM = 2833.42, motor shaft RPM = 113.34
```

Figure 3: Serial Monitor output for pwm-encoderRPM-motorRPM mapping

The screenshot shows the Arduino IDE interface with the 'motordirection.ino' sketch loaded. The code in the editor includes a loop that updates the encoder ticks and prints the RPM to the Serial Monitor. The Serial Monitor output shows a series of 'clockwise RPM' values ranging from 56.68 to 69.30.

```

40
41 void loop() {
42   currentTime = millis();
43
44   digitalWrite(IN3, HIGH);
45   digitalWrite(IN4, LOW);
46   analogWrite(ENB, 50);
47   updateEncoderTicks();
48
49   if(updateRPM()) {
50     Serial.print("clockwise RPM = ");
51     Serial.println(RPM);
52   }
53 }
54
Output Serial Monitor x
Not connected. Select a board and a port to connect automatically.
clockwise RPM = 56.68
clockwise RPM = 69.35
clockwise RPM = 69.32
clockwise RPM = 69.46
clockwise RPM = 69.36
clockwise RPM = 69.37
clockwise RPM = 69.38
clockwise RPM = 69.40
clockwise RPM = 69.30

```

(a) clockwise rotation

The screenshot shows the Arduino IDE interface with the 'motordirection.ino' sketch loaded. The code in the editor includes a loop that updates the encoder ticks and prints the RPM to the Serial Monitor. The Serial Monitor output shows a series of 'anticlockwise RPM' values ranging from -55.01 to -65.87.

```

40
41 void loop() {
42   currentTime = millis();
43
44   digitalWrite(IN3, LOW);
45   digitalWrite(IN4, HIGH);
46   analogWrite(ENB, 50);
47   updateEncoderTicks();
48
49   if(updateRPM()) {
50     Serial.print("anticlockwise RPM = ");
51     Serial.println(RPM);
52   }
53 }
54
Output Serial Monitor x
Message (Enter to send message to 'Arduino Uno' on '/dev/ttyUSB0')
anticlockwise RPM = -55.01
anticlockwise RPM = -65.87
anticlockwise RPM = -65.81
anticlockwise RPM = -65.88
anticlockwise RPM = -65.81
anticlockwise RPM = -65.87

```

(b) anticlockwise rotation

Figure 4: Serial Monitor output for detecting motor rotation direction using encoder readings

3.4 Results

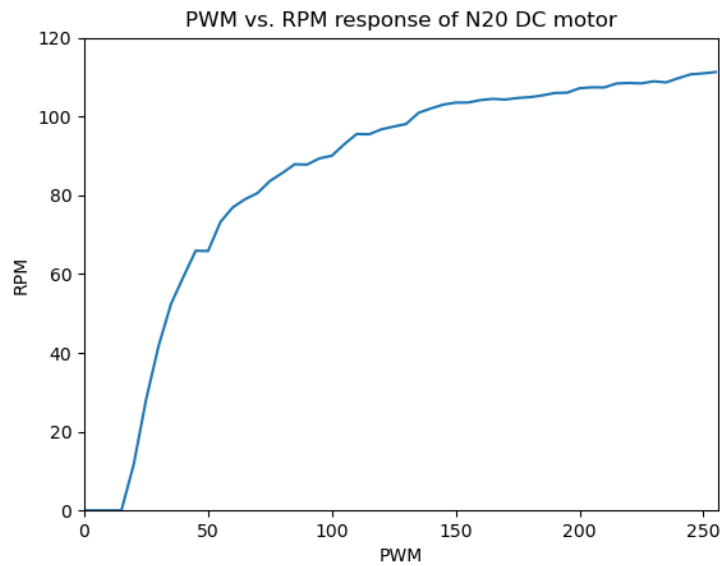


Figure 5: plot of duty-cycle to motor shaft RPM response measurements

3.5 Conclusions

We were able to successfully calculate the RPM of the motor using the 1 second interval logic that we developed. We were able to do so while mapping the duty cycle to the RPM by varying the PWM value from 0 to 255 in steps of 5. In our code, we have calculated the average RPM for 10 “1 second intervals” for each PWM iteration (the outer for loop). We have also successfully used the encoder data to determine if the motor is rotating clockwise or anticlockwise using the encoder state pattern logic (acurr, aprev, bcurr and bprev). We have also been able to map the encoder count to the motor speed(RPM) using the updateEncoderTicks() logic.