Motor Project Report

1. Objective

The goal of this project was to test and control a DC motor using a Raspberry Pi and a motor driver (H-bridge), troubleshoot faulty outputs, and implement basic forward/backward movement with speed control.

---

2. Hardware Setup

- **Raspberry Pi GPIO pins:**
    - Left motor (Motor A): ENA = 18, IN1 = 23, IN2 = 24
    - Right motor (Motor B, faulty): ENB = 13, IN3 = 27, IN4 = 22
- **Motor driver:** H-bridge with 12 V motor power and logic powered from Pi
- **Motors:** Left motor connected, right motor disconnected (faulty driver)
- **Other:** Multimeter used to measure voltage at outputs; motors removed during testing

---

3. Troubleshooting and Testing

1. **Testing Pi GPIO pins:**
    - Ran scripts to set pins HIGH individually and measured outputs with a multimeter
    - Observed that only **Motor A outputs (OUT1/OUT2)** responded reliably
    - Motor B outputs showed 0–0.1 V → driver partially damaged
2. **EN and IN pins:**
    - Observed that outputs require **EN pin HIGH** to activate

    - IN pins control **direction** (forward/reverse)
    - Without EN HIGH, no output appears even if IN pins are set
3. **Motor driver behavior:**
    - When 12 V motor power was applied, some outputs turned on immediately if EN pins floated HIGH
    - Solved by tying EN LOW and IN pins LOW initially

---

4. Motor A Control Code

- Python code using **pigpio** library
- Functions implemented:
    - `run_motor(value)` → move forward (positive) or backward (negative)
    - `stop_motor()` → stop motor safely
- **PWM used to control speed:** 0–255 duty cycle

Example usage:

```
run_motor(150)   # forward at ~60% speed
run_motor(-150)  # backward at ~60% speed
stop_motor()     # stop
```

- ENA controls speed; IN1/IN2 control direction

To control the motor with the Raspberry Pi and the L298N driver, I used Python with the `pigpio` library. The following program demonstrates how to run **Motor A** both forward and backward with adjustable speed using PWM (Pulse Width Modulation).

```
import pigpio
import time

pi = pigpio.pi()
if not pi.connected:
    raise RuntimeError("pigpio daemon not running")

# Motor A pins (left motor)
ENA, IN1, IN2 = 18, 23, 24

# Set pins as outputs
for p in (ENA, IN1, IN2):
    pi.set_mode(p, pigpio.OUTPUT)

# Set PWM frequency
pi.set_PWM_frequency(ENA, 2000)

def run_motor(value):
    """Run Motor A forward or backward.
    Positive value → forward
    Negative value → backward
    Value range: -255 to 255
    """
```

```
    forward = value >= 0
    pi.write(IN1, 1 if forward else 0)
    pi.write(IN2, 0 if forward else 1)
    pi.set_PWM_dutycycle(ENA, min(255, abs(int(value))))

def stop_motor():
    pi.set_PWM_dutycycle(ENA, 0)
    pi.write(IN1, 0)
    pi.write(IN2, 0)

try:
    # Forward
    print("Motor A forward...")
    run_motor(100)
    time.sleep(2)
    stop_motor()
    time.sleep(1)

    # Backward
    print("Motor A backward...")
    run_motor(-210)
    time.sleep(2)
    stop_motor()
```

```
finally:
    stop_motor()
    pi.stop()
    print("Motor A stopped. Cleanup complete.")
```

Explanation:

- The **ENA pin** is used for speed control via PWM.
- **IN1** and **IN2** control the motor's direction.
- The function `run_motor(value)` takes a positive value for forward motion and a negative value for reverse motion, where the magnitude controls the speed (0–255).
- `stop_motor()` ensures the motor halts and the GPIO pins are safely reset.
- The code runs the motor forward for 2 seconds, pauses, and then runs it backward for 2 seconds before stopping completely.

---

5. Requirements

Item	Quantity / Notes
Lithium batteries	2
Battery charger / holder	1
DC motors	2
L298N dual H-bridge motor driver	1

Breadboard	1
Raspberry Pi	1
Jumper wires	>20

---

6. Observations

- Only **Motor A (left motor)** works; Motor B driver is faulty
- PWM allows **adjustable speed**
- Safe testing requires **motors disconnected** or low voltage, especially when outputs behave unpredictably

---

7. Conclusion

This project demonstrated how to safely test motor driver outputs using a Raspberry Pi, isolate a faulty channel, and control a single working motor with forward/backward motion and speed control. It highlights the importance of **EN pins, input logic, and careful power sequencing** when working with H-bridges.

---

8. References / Notes

- Raspberry Pi GPIO and pigpio library documentation
- H-bridge motor driver datasheet (for pin mapping and enable logic)
- Multimeter for voltage measurements and troubleshooting