MCD6A Brushed DC Motor Driver

User Manual

Version 0.0.1

MekRobo Technology

20-Dec-2025

Contents

1 Overview

Dual channel, H-bridge, brushed DC motor driver with quadrature encoder decoding, current sensing and onboard micro-controller for control, automation and robotics application.

Dual channel motor driver, Toshiba TB9054FTG integrated circuit (IC) incorporates each channels with two outputs for direct drive of a brushed DC motor. PWM control with low-resistance enables highly energy efficient drive output. The PWM1A and PWM1B pins specify forward, reverse and brake modes for motor 1, and the PWM2A and PWM2B pins specify these modes for motor 2.

The output current capacity is 6.5 A (typical), which is suitable for a wide range of application like small size ground robots and a lot of automation projects.

Inbuilt ESP32 micro-controller makes easier programming with Arduino like programming interface with hassle-free use.

2 Application

Robotics application such as ground robots, self-balancing robot and automation application etc.

3 Specifications

- Dual channel, H-bridge brushed DC motor driver.
- Control Input: PWM with frequency from $1 - 10$ *khz*.
- Forward/reverse/brake modes.
- High voltage side output current monitoring.
- Protection: Stop output for supply under voltage, over current, over temperature.
- Operating voltage: V_{BAT} = 4.5 to 28 *V*, V_{CC} = 4.5 to 5.5 *V*, V_{DDIO} = 3.0 to 5.5 *V*.
- Operating temperature range: *Ta* = -40 to 125^{o}C
- Encoder Decoder: Onboard dual channel quadrature digital encoder decoding for motor speed measurement with onboard power supply to encoder sensor.
- PID control with onboard motor speed and current sensing.
- USB serial interface between micro-controller and the computer for two-way communication firmware upload, control, communication and sensors data streaming.

3.1 Technical Data

The power rating the MekRobo brushed DC motor controller is listed in Table 3.1

Table 1: Rating

Symbol	Parameter	Unit	Min	Typ	Max
V_{BAT}	Power Supply/Battery Voltage	V	4.5	12/24	27
I_1/I_2	Output Current to Motor Channel A/B	A	0	-	6.5

3.2 Schematic

Fig. 1 indicates the relevant pin-out, configuration jumper, USB-C connections and other relevant components.

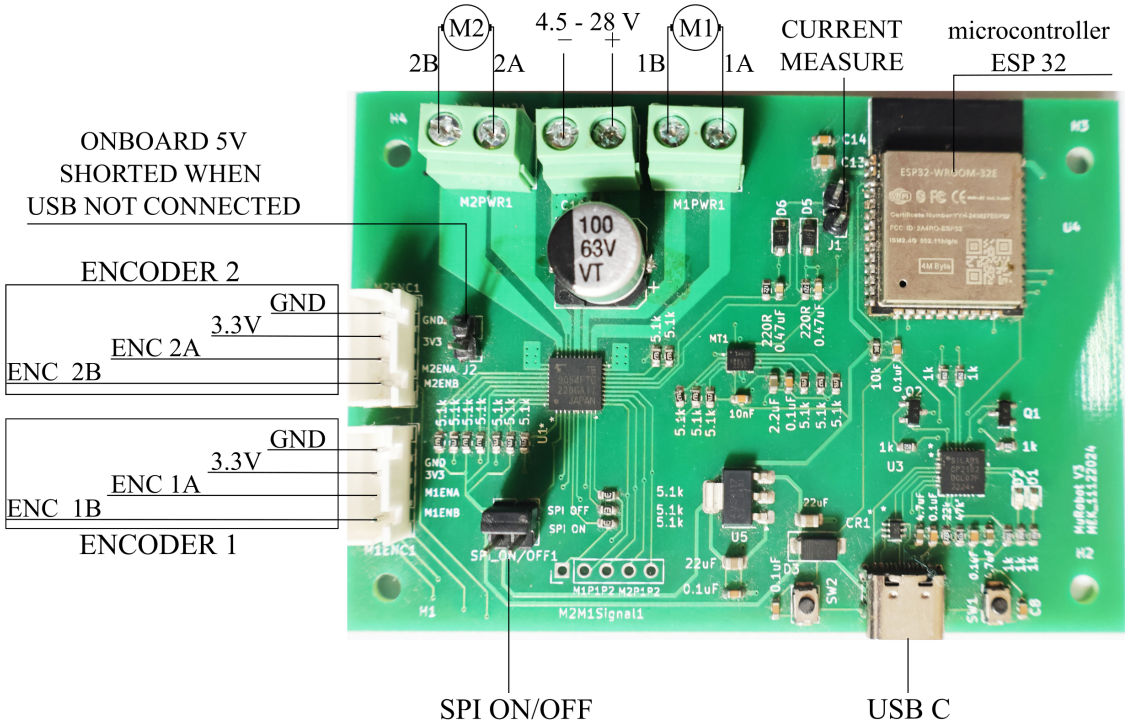


Figure 1: Motor Driver Schematics.

Table 2: Description

Parameter	**Symbol**	**IN/OUT**	**Description**
Motor 1, 2	1A, 2A	OUT	Motor 1, 2 Power A
	1B, 2B	OUT	Motor 1, 2 Power B
Encoder 1, 2	GND	GND	Ground
	3.3V	Power	Power to Encoder
	ENC 1A, 2A	IN	Encoder Channel A for Motor 1, 2
	ENC 1B, 2B	IN	Encoder Channel B for Motor 1, 2
SPI ON/OFF	-	Config.	Enable/Disable SPI of Driver IC TB9054FTG
Motor Current 1, 2	I_1, I_2	OUT	Measurement terminal of Motor 1, 2 current
USB	-	IN/OUT	Connection to Computer for Data read/write
Power Source/Supply	V_{BAT}	IN	Battery Power/ External Power Sources

3.3 Dimensions

In addition to the dimension specified in Fig. 2, sufficient clearance should be maintained for heat dissipation and avoiding electrical short circuit. There should be at-least 10 mm clearance in the bottom side and 20 mm clearance in the upper side.

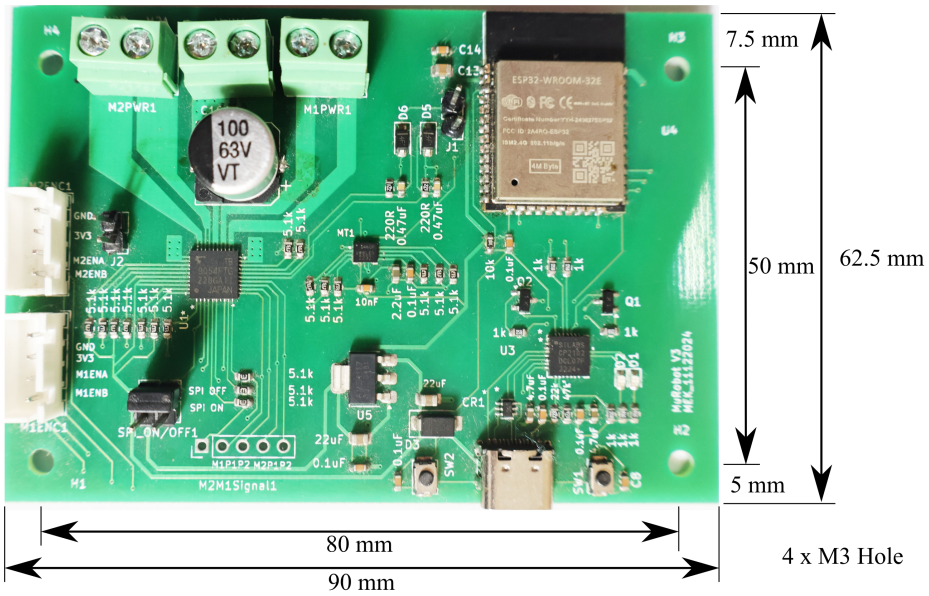


Figure 2: Controller Board Dimensions.

4 Application Case

One minimum application each for direct PWM sweep and RPM PID control for single as well dual channel has been included as example of the library.

4.1 Connection

The minimal circuit diagram has been presented in Fig. 3. The controller must be connected to a PC through the USB-C for serial monitoring.

4.2 Software Setup

MekRobo BDC driver can be controlled by onboard ESP32 micro-controller. It can also be controller with external micro-controller through external PWM pins.

Install Arduino-ESP32 support if it is not installed, following the step by step instruction in any of the follwing links.

ESPRESSIF Official https://docs.espressif.com/projects/arduino-esp32/en/latest/installing.html

Unofficial link: https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/

Download MekRob's ESP32-motor-controller Arduino from the official MekRobo github repository using the link: https://github.com/mekrobo/ESP32-motor-controller.git.

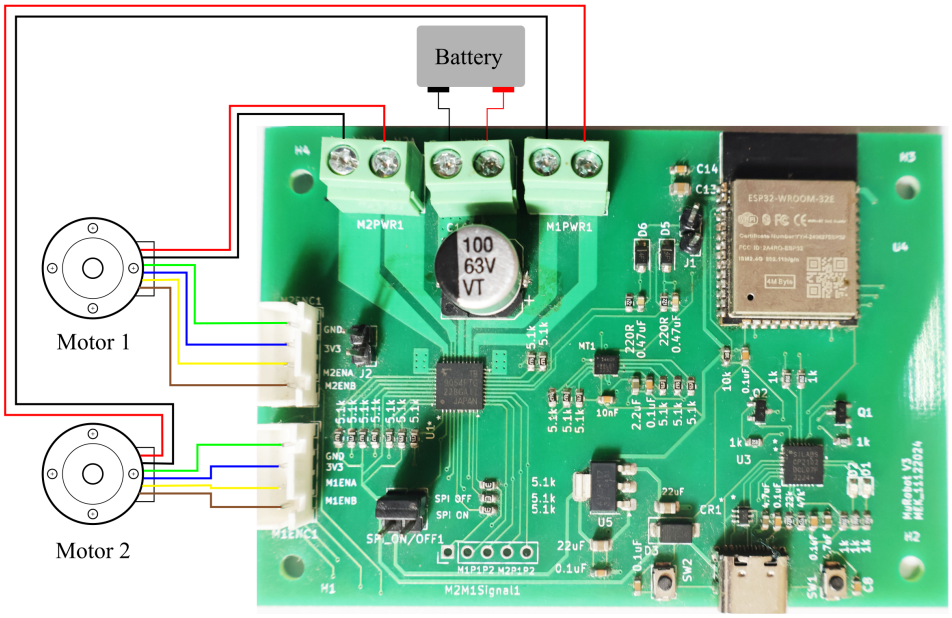
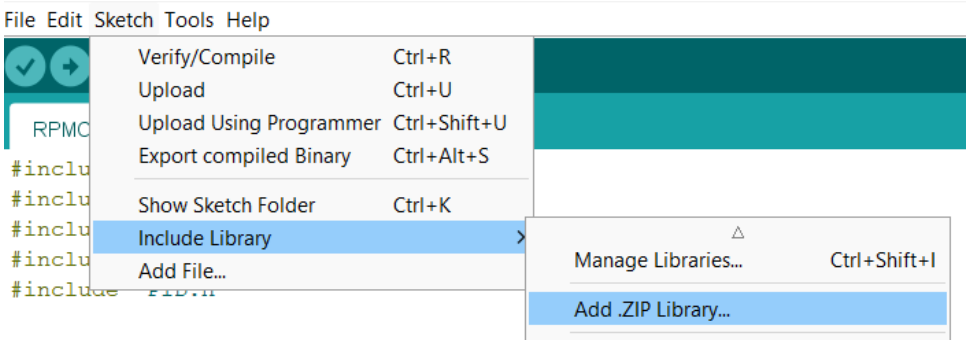
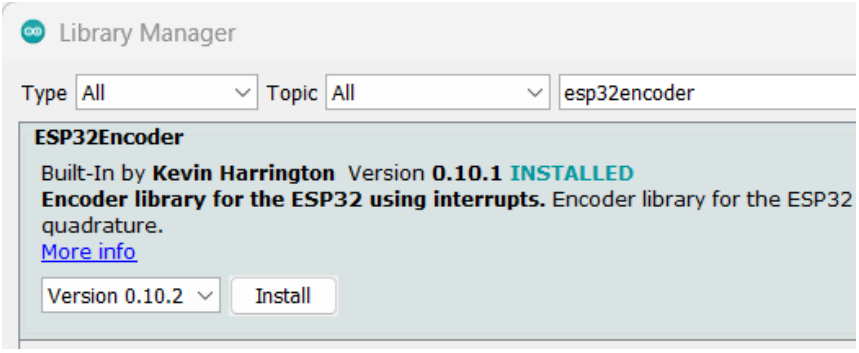


Figure 3: Motor Driver Connection.

It should download ESP32-motor-controller.zip file. In Arduino IDE go to Menu - `Sketch > Include Library > Add .ZIP Library...` and choose the downloaded zip file. You can see the confirmation if the library import is successful.

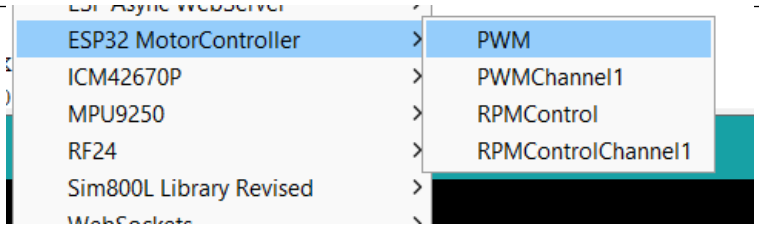


Then Install the dependency library ESP32Encoder from menu - `Sketch > Include Library > Manage Libraries...`



After successful import of library Open examples from Menu - `File > Examples > ESP32 MotorController...`

There are four examples: `PWM, PWMChannel1, RPMControl and RPMControlChannel1`.



The required header files for all the examples are as follows. The `config.h` has all the pin-out, motor specification, PID parameters etc. `motor.h` has the class `MotorController` that control the motors.

Listing 1: header files

```
#include <Arduino.h>
#include <ESP32Encoder.h>
#include "config.h"
#include "motor.h"
```

The configuration parameter like PWM_FREQ, COUNTS_PER_REV, Timer Settings, PID parameters are presented as follows. These parameters may need changes for various application scenarios.

Listing 2: Editable parameter in config.h

```
#define PWM_FREQ 7000 // PWM frequency in Hz

// Motor Specification
#define MAX_RPM 300 // motor  maximum RPM at output shaft
#define MAX_ACC 20000 // motor Acceleration at base motor
#define COUNTS_PER_REV_1 2709.2 // motor 1 encoder's no of ticks per rev motor
  shaft
#define COUNTS_PER_REV_2 2709.2 // motor 2 encoder's no of ticks per rev motor
  shaft

// Timer Settings
#define SENSOR_PERIOD_MS 20 // Sensor read period in ms (>=20 ms)
#define CONTROL_PERIOD_MS 40 // Control loop period in ms

// PID parameters for RPM control
#define Kff_1 0.7245 // RPM to PWM mapping for motor 1
#define Kfr_1 4.128 // PWM for drive friction of motor 1
#define KP_1 2 // P constant for motor 1
#define KI_1 0.2 // I constant for motor 1
#define KD_1 0 // D constant for motor 1

#define Kff_2 0.7187 // RPM to PWM mapping for motor 2
#define Kfr_2 4.0236 // PWM for drive friction of motor 2
#define KP_2 2 // P constant
#define KI_2 0.2 // I constant
#define KD_2 0 // D constant
```

The class `ESP32Encoder` from `<ESP32Encoder.h>` is initialized as `motor1_encoder`, `motor2_encoder` for **motor 1** and **motor 2** encoder ticks reading. The motor controller class `MotorController` from `"motor.h"` is initialized with channel selection, `Channel_1`, `Channel_2` for using **motor 1** and **motor 2** respectively.

Listing 3: Object calling

```
ESP32Encoder motor1_encoder, motor2_encoder; // Encoder objects for motor 1 and motor 2
MotorController motor1(Channel_1); // Motor Controller object for channel 1
MotorController motor2(Channel_2); // Motor Controller object for channel 2

TaskHandle_t SensorReadHandle = NULL; // Sensor Reading Task Handle

```

The constant parameter for unit conversion factor are computed to avoid repeated computation during runtime.

Listing 4: Constant parameter

```
float TICKS2RPM_1 = 60000.0 / (COUNTS_PER_REV_1 * SENSOR_PERIOD_MS); // RPM calculation factor for motor 1
float TICKS2RPM_2 = 60000.0 / (COUNTS_PER_REV_2 * SENSOR_PERIOD_MS); // RPM calculation factor for motor 2
float VALUE2AMP = (3.3 * 1.7483) / 4095.0; // Assuming a 12-bit ADC and 3.3V reference
```

Initialize the variables

Listing 5: Variables

```
rpm des_rpm = {0.0,0.0}, mes_rpm = {0.0, 0.0}; // Desired and Measured RPM structures
current mes_current;              // Measured current structure
int64_t last_pulse_count_m1 = 0; // store previous pulse count for motor 1
int64_t last_pulse_count_m2 = 0; // store previous pulse count for motor 2
int64_t pulse_count_m1 = 0;      // current pulse count for motor 1
int64_t pulse_count_m2 = 0;      // current pulse count for motor 2
char data_sensor[64];            // serial print buffer
// Additional variable required for RPM PWM control paste below
```

The sensor_read function which is attached to a dedicated core runs at an interval of SENSOR_PERIOD_MS in *miliseconds*.

Listing 6: Sensor reading

```
/**
* @brief Sensor reading in a periodic interval of SENSOR_PERIOD_MS (miliseconds)
*
* @param parameter Empty
*/
void sensor_read(void *parameter) {
  const TickType_t xDelay = pdMS_TO_TICKS(SENSOR_PERIOD_MS); // read at every SENSOR_PERIOD_MS
  TickType_t xLastWakeTime = xTaskGetTickCount();
  int i = 0;
  for (;;)
  {
    // Read sensors here
    pulse_count_m1 = motor1_encoder.getCount();
    pulse_count_m2 = motor2_encoder.getCount();
    mes_rpm.motor1 = (pulse_count_m1 - last_pulse_count_m1) * TICKS2RPM_1; // RPM calculation channel 1
```

```
      mes_rpm.motor2 = (pulse_count_m2 - last_pulse_count_m2) * TICKS2RPM_1; // RPM calculation channel 2
      last_pulse_count_m1 = pulse_count_m1;
      last_pulse_count_m2 = pulse_count_m2;
      mes_current.motor1 = analogRead(MOTOR1_C) * VALUE2AMP; // Assuming a 12-bit ADC and 3.3V reference
      mes_current.motor2 = analogRead(MOTOR2_C) * VALUE2AMP; // Assuming a 12-bit ADC and 3.3V reference
      vTaskDelayUntil(&xLastWakeTime, xDelay);
    }
  }
```

Setup required for Serial printing, encoder attachment, and attaching the sensor reading function to a dedicated core.

Listing 7: Setup

```
  void setup() {
    Serial.begin(115200);  // For Serial printing
    delay(100);
    // Initialize encoders
    ESP32Encoder::useInternalWeakPullResistors = UP;
    motor1_encoder.attachFullQuad(MOTOR1_ENC_A, MOTOR1_ENC_B);
    motor2_encoder.attachFullQuad(MOTOR2_ENC_A, MOTOR2_ENC_B);

    // Additional setup code can be added here
    delay(50);
    xTaskCreatePinnedToCore(sensor_read, "SensorRead", 4096, NULL, 1, &SensorReadHandle, 0); // Core 0
    // RPM PID control setup code paste below
  }
```

4.2.1 PWM Run

PWM sweep and sensor data printing for both the channel. For using only a single channel, the variables and functions calling for the other channel should be cleaned.

Listing 8: PWM Sweep

```
void loop() {
  // Changing PWM
  for(int dutyCycle = 0; dutyCycle <= 80; dutyCycle++){

    motor1.rotate(dutyCycle);
    motor2.rotate(-dutyCycle);
    data_sensor[0] = '\0';
    snprintf(data_sensor, sizeof(data_sensor), "Motor1:: PWM:%d, RPM: %0.2f, Current: %0.3f\n",
    dutyCycle,mes_rpm.motor1, mes_current.motor1);
    Serial.print(data_sensor);
    data_sensor[0] = '\0';
    snprintf(data_sensor, sizeof(data_sensor), "Motor2:: PWM:%d, RPM: %0.2f, Current: %0.3f\n",
    -dutyCycle,mes_rpm.motor2, mes_current.motor2);
    Serial.print(data_sensor);
```

```
    delay(100);
  }

  for(int dutyCycle = 80; dutyCycle >= -80; dutyCycle--){

    motor1.rotate(dutyCycle);
    motor2.rotate(-dutyCycle);

    data_sensor[0] = '\0';
    snprintf(data_sensor, sizeof(data_sensor), "Motor1:: PWM:%d, RPM: %0.2f, Current
    : %0.3f\n",
    dutyCycle,mes_rpm.motor1, mes_current.motor1);
    Serial.print(data_sensor);
    data_sensor[0] = '\0';
    snprintf(data_sensor, sizeof(data_sensor), "Motor2:: PWM:%d, RPM: %0.2f, Current
    : %0.3f\n",
    -dutyCycle,mes_rpm.motor2, mes_current.motor2);
    Serial.print(data_sensor);

    delay(100);
  }

  for(int dutyCycle = -80; dutyCycle <= 0; dutyCycle++){

    motor1.rotate(dutyCycle);
    motor2.rotate(-dutyCycle);

    data_sensor[0] = '\0';
    snprintf(data_sensor, sizeof(data_sensor), "Motor1:: PWM:%d, RPM: %0.2f, Current
    : %0.3f\n",
    dutyCycle,mes_rpm.motor1, mes_current.motor1);
    Serial.print(data_sensor);
    data_sensor[0] = '\0';
    snprintf(data_sensor, sizeof(data_sensor), "Motor2:: PWM:%d, RPM: %0.2f, Current
    : %0.3f\n",
    -dutyCycle,mes_rpm.motor2, mes_current.motor2);
    Serial.print(data_sensor);

    delay(100);
  }

}
```

4.2.2 RPM Control

The variable like PID parameters etc. specific to the RPM PID control should be also declared in addition to PWM run cases.

Listing 9: Variables for RPM Control

```
int ctrl_pwm1 = 0;  // control input PWM for motor 1
int ctrl_pwm2 = 0;  // control input PWM for motor 2

// PID Parameters for Motor 1
```

```
pid_param pid_param_m1 = {
  .kff = Kff_1,
  .fr = Kfr_1,
  .kp = KP_1,
  .ki = KI_1,
  .kd = KD_1,
  .pwm_max = PWM_MAX,
  .integral_max = 100,
  .integral_error = 0,
  .prev_error = 0,
};
// PID Parameters for Motor 2
pid_param pid_param_m2 = {
  .kff = Kff_2,
  .fr = Kfr_2,
  .kp = KP_2,
  .ki = KI_2,
  .kd = KD_2,
  .pwm_max = PWM_MAX,
  .integral_max = 100,
  .integral_error = 0,
  .prev_error = 0,
};
```

The callback function `static void pid_loop_cb (void *args)` attached to a timer interrupt, computes the required control input as PWM and run motors accordingly by calling `.rotate` member function at a set interval of `CONTROL_PERIOD_MS`.

Listing 10: RPM PID control loop

```
/** PID loop callback for RPM control with CONTROL_PERIOD_MS interval
*
* @param args desired rpm structure pointer
*/
static void pid_loop_cb(void *args)
{
  rpm *req_rpm = (rpm *)args;
  ctrl_pwm1 = pid_compute(&pid_param_m1, req_rpm->motor1, mes_rpm.motor1);
  ctrl_pwm2 = pid_compute(&pid_param_m2, req_rpm->motor2, mes_rpm.motor2);
  motor1.rotate(ctrl_pwm1);
  motor2.rotate(ctrl_pwm2);
}
```

Create a periodic timer interrupt with interval time of `CONTROL_PERIOD_MS` millisecond and attached the callback function `static void pid_loop_cb (void *args)` in setup.

Listing 11: RPM Control Setup

```
void setup() {
  // Paste the common setup code here

  // RPM PID control setup code
  // PID control timer setup at interval of CONTROL_PERIOD_MS
  const esp_timer_create_args_t periodic_timer_args = {
    .callback = pid_loop_cb,
    .arg = &des_rpm,
```

```
      .name = "pid_loop"
  };
  esp_timer_handle_t pid_loop_timer = NULL;
  esp_timer_create(&periodic_timer_args, &pid_loop_timer);
  esp_timer_start_periodic(pid_loop_timer, CONTROL_PERIOD_MS * 1000); //! interrupt
    function callback for pid control
  }
```

Runs a RPM sweeps in loop and print sensor readings for both the channel. For using only a single channel, the variables and functions calling for the other channel should be cleaned.

Listing 12: RPM Sweep

```
void loop() {
  // changing the RPM
  for(int i = 0; i <= 80; i++){

    des_rpm.motor1 = (float)(i*2); // Desired RPM for motor 1
    des_rpm.motor2 = (float)(i*2); // Desired RPM for motor 2

    //Sensor data print
    data_sensor[0] = '\0';
    snprintf(data_sensor, sizeof(data_sensor), "Motor1: PWM:%d, RPM:%0.2f, RPM: %0.2
    f, Current: %0.3f\n",
    ctrl_pwm1, des_rpm.motor1, mes_rpm.motor1, mes_current.motor1);
    Serial.print(data_sensor);
    data_sensor[0] = '\0';
    snprintf(data_sensor, sizeof(data_sensor), "Motor2: PWM:%d, RPM:%0.2f, RPM: %0.2
    f, Current: %0.3f\n",
    ctrl_pwm2, des_rpm.motor2, mes_rpm.motor2, mes_current.motor2);
    Serial.print(data_sensor);

    delay(100);
  }

  for(int i = 80; i >= -80; i--){

    des_rpm.motor1 = (float)(i*2); // Desired RPM for motor 1
    des_rpm.motor2 = (float)(i*2); // Desired RPM for motor 2

    // Sensor data print
    data_sensor[0] = '\0';
    snprintf(data_sensor, sizeof(data_sensor), "Motor1: PWM:%d, RPM:%0.2f, RPM: %0.2
    f, Current: %0.3f\n",
    ctrl_pwm1, des_rpm.motor1, mes_rpm.motor1, mes_current.motor1);
    Serial.print(data_sensor);
    data_sensor[0] = '\0';
    snprintf(data_sensor, sizeof(data_sensor), "Motor2: PWM:%d, RPM:%0.2f, RPM: %0.2
    f, Current: %0.3f\n",
    ctrl_pwm2, des_rpm.motor2, mes_rpm.motor2, mes_current.motor2);
    Serial.print(data_sensor);

    delay(100);
  }

  for(int i = -80; i <= 0; i++){
```

```
40
41      des_rpm.motor1 = (float)(i*2); // Desired RPM for motor 1
42      des_rpm.motor2 = (float)(i*2); // Desired RPM for motor 2
43
44      // Sensor data print
45      data_sensor[0] = '\0';
46      snprintf(data_sensor, sizeof(data_sensor), "Motor1: PWM:%d, RPM:%0.2f, RPM: %0.2
        f, Current: %0.3f\n",
47      ctrl_pwm1, des_rpm.motor1, mes_rpm.motor1, mes_current.motor1);
48      Serial.print(data_sensor);
49      data_sensor[0] = '\0';
50      snprintf(data_sensor, sizeof(data_sensor), "Motor2: PWM:%d, RPM:%0.2f, RPM: %0.2
        f, Current: %0.3f\n",
51      ctrl_pwm2, des_rpm.motor2, mes_rpm.motor2, mes_current.motor2);
52      Serial.print(data_sensor);
53
54      delay(100);
55    }
56  }
```

5 User Agreement

6 Revision History

Table 3: Revision History

Version	Date	Changes
Version 0.0.1	20-Dec-2025	Initial Release