M17 Series Servomotors - DATASHEET

Affordable and Simple All-in-One Motion Control

From Education to Innovation

Introduction

The M17 Series Servomotors are all-in-one motion control solutions that integrate a motor, motor driver, motion controller, and encoder in a single compact package. These servomotors feature an RS-485 communication interface, enabling multiple units to be daisy-chained together and controlled from a single connection point. Available in four models - M17-60, M17-48, M17-40, and M17-34 - the series offers flexible options to match specific torque and size requirements while maintaining consistent control characteristics. Each M17 Series Servomotor comes with sophisticated control features including multiple operation modes, self-calibration capabilities, and built-in status monitoring through LED indicators. The motors can be easily controlled from any platform including Mac, PC, Raspberry Pi, or Arduino (requiring only a low-cost RS485 adapter), making them ideal for both educational and industrial applications. The series supports precise position control with trapezoid movement profiles, closed-loop control mode, and comprehensive error handling. With standardized NEMA 17 mounting dimensions, wide voltage range (12-24V), and robust communication protocol, M17 Series Servomotors provide a reliable and flexible solution for applications requiring precise motion control, from robotics and CNC machines to automated testing equipment and scientific instruments.

Send Us Feedback

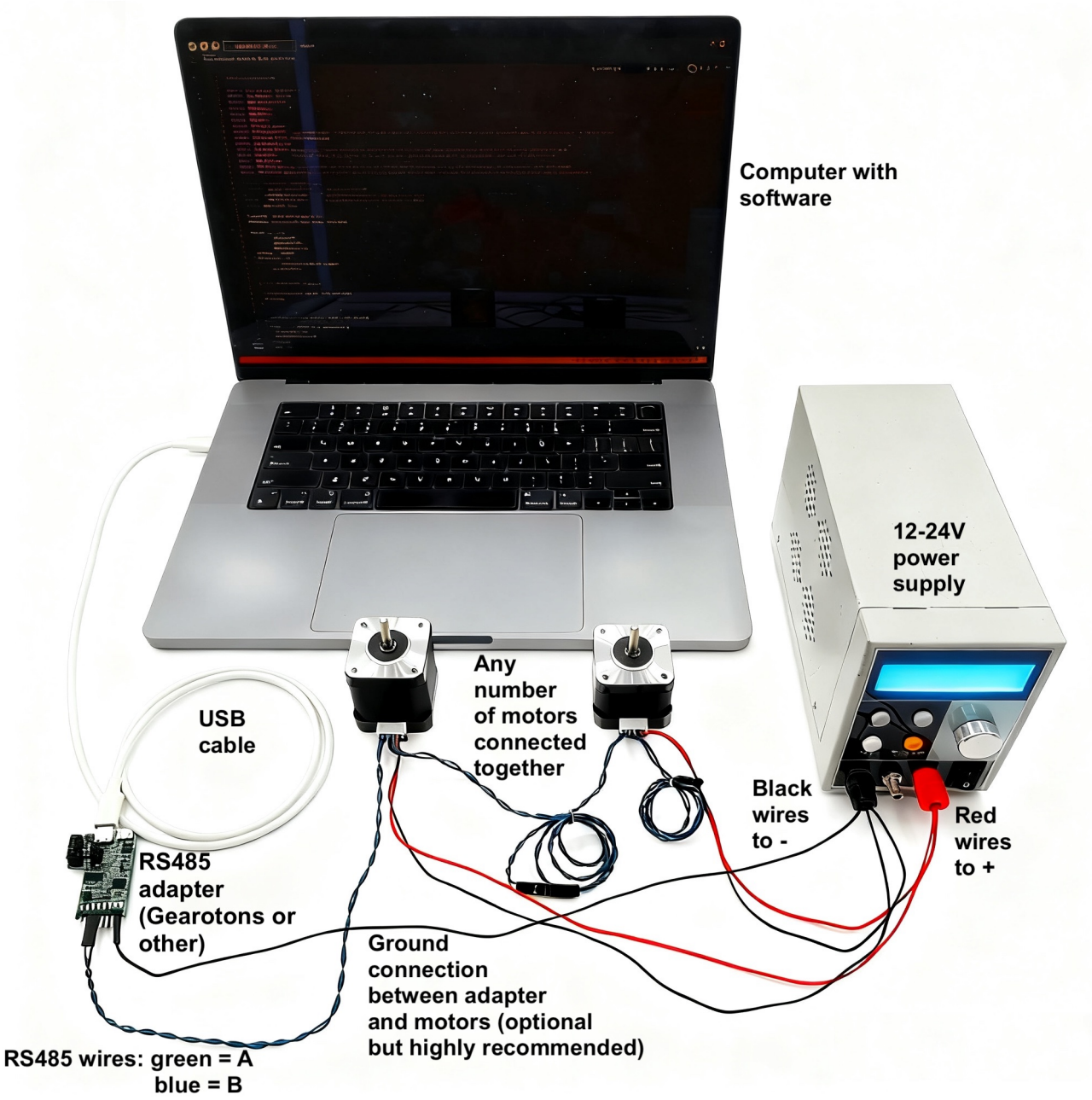
If you find errors in this document or have questions, please let us know by going to: tutorial.gearotons.com/feedback. We would really appreciate your suggestions on how we can improve the product or documentation. Thanks very much!

Visit our Feedback Page

Key Features

- High level of integration combines a motor, motor driver, motion control system, and encoder
- Control any number of motors from one simple controller over an RS-485 communication interface
- Control with high level commands such as "Enable mosfets" and "Trapezoid move" (avoid timing critical DIR/STEP control)
- Compact form factor nearly the same size as a NEMA 17 stepper motor with the same specifications (no protrusions)
- Standardized NEMA 17 mounting dimensions
- Wide voltage range (12-24V) for flexible power options
- High-precision closed loop control with built-in encoder and PID control loop that runs at 32 kHz
- Much more power efficient in closed loop control vs. conventional stepper
- Integrated over-current, over-voltage, and over-temperature protection
- Maximum speed can reach 560 RPM
- Torque-to-weight ratio is the same as an equivalent stepper motor
- Compatible with a wide variety of interfaces and hardware, such as Raspberry Pi, Arduino, ESP32, Mac, and PC
- We strive to provide excellent documentation and tutorials to get you up and running fast
- We provide AI friendly documentation in case you want your favorite AI to do all the work for you
- Suitable for robotics, CNC, automation, scientific instruments, testing jigs, 3D printers, and everything else
- Available in different sizes so you can find the right torque and price for your application

Connection Diagram



Unit System

The M17 Series Servomotors have certain internal units so that they can perform the calculations associated with motion efficiently (using integer math). It is the responsibility of the controlling software to support multiple units of measurement for various quantities. Our Python and Arduino libraries handle unit conversions automatically, allowing you to work with your preferred units. Below are the supported units for each quantity:

Quantity	Available Units
Time	seconds, milliseconds, minutes, microseconds, timesteps
Position	shaft rotations, degrees, radians, encoder counts
Velocity	rotations per second, rpm, degrees per second, radians per second, counts per second, counts per timestep
Acceleration	rotations per second squared, rpm per second, degrees per second squared, radians per second squared, counts per second squared, counts per timestep squared
Current	internal current units, milliamps, amps
Voltage	millivolts, volts
Temperature	celsius, fahrenheit, kelvin

Getting Started Guide

To help you get started with your M17 Series Servomotor, we provide a comprehensive online guide that covers everything from initial setup to advanced protocol implementations. This guide includes:

- Step-by-step setup instructions
- Detailed communication protocol documentation
- Programming examples and code snippets
- Description of error codes
- Troubleshooting tips and best practices

Click Here to Visit our Getting Started Guide: tutorial.gearotons.com

Indicator LEDs and Buttons

The servomotor has two status LEDs (Green and Red). The green LED flashes slowly to show a heart beat, flashes quickly to indicate that the bootloader is running rather than the application, and also lights up briefly while a packet is being received, so it flickers with communication on the bus. The red LED indicates fatal error codes by flashing a certain number of times.

The servomotor has two buttons labelled "Reset" and "Test". The Reset button will reset the internal microcontroller and all state will go back to default values. The Test button will cause the motor to spin. Press briefly to let it spin one way and press for more than 0.3 seconds and release to let it spin the other way. Hold down for at least 2 seconds and release to cause the motor to go to closed loop mode. Hold down for more than 15 seconds and release to let the motor perform a calibration on itself. Note that it will spin during calibration and must be able to spin freely for calibration to be successful, so remove any loads before doing this operation.

Communication Protocol Overview

The M17 Series Servomotors use RS-485 for communication, enabling multiple motors to be daisy-chained together. Each motor has a factory assigned 64-bit unique address so that it can be addressed individually on an RS-485 bus. Each motor can also be assigned a unique 1-byte alias (0-251) in order to save bytes in the communication packet and still get similar individual control. There is also a broadcast alias, which is 255. The protocol supports a comprehensive set of commands for motion control, configuration, and status monitoring. Firmware update through the RS-485 interfce is supported. Communication baud rate is fixed at 230400.

Command Reference Summary

For the up to date source of truth for all available commands, you can look at this document.

 https://github.com/tomrodinger/servomotor/blob/main/python_programs/servomotor/motor_commands.json

You can also run this command:

```
pip3 install servomotor   # run this just once to install the library and programs
servomotor_command -c
```

This will print out the information contained in the motor_commands.json file in a nicer way and give some usage information for sending commands to the motor from the command line.

Basic Control

Command	Description
Disable MOSFETs	Immediately disable the motor driver outputs (MOSFETs); the command executes at once (it is not queued) and is idempotent.
Enable MOSFETs	Enable the motor driver outputs (MOSFETs).
Reset time	Resets the device's absolute microsecond clock to zero and, as a major side effect, clears the entire movement queue and stops the motor instantly.
Emergency stop	Stop the motor immediately: this command executes at once (it is not queued), disables the MOSFETs, clears the movement queue, zeroes the current velocity, and snaps the internal position target to the current position.
Zero position	Make the current position the zero position (the origin).
System reset	Reset the device.

Motion Control

Command	Description
Trapezoid move	Travel the given signed displacement over exactly the given duration, relative to the position at the end of previously queued motion (for an absolute target use 'Go to position').
Go to position	Queue a smooth absolute move.
Homing	Home the motor by moving until it hits a hard stop.
Go to closed loop	Enter closed-loop position control mode.
Move with acceleration	Queue a constant-acceleration segment: the acceleration is applied once per time step for the given number of time steps.
Move with velocity	Rotate the motor at a constant velocity for a fixed duration.
Multimove	Enqueue up to 32 moves in one command; each move is an acceleration segment or an instant velocity change (selected per move by the moveTypes bitmask) executed for its given number of time steps.

Configuration

Command	Description
Set maximum velocity	Set the maximum allowed velocity.
Set maximum acceleration	Set the maximum acceleration used to plan all trapezoid moves queued afterwards ('Go to position', 'Trapezoid move', calibration moves); items already in the queue are unaffected.
Start calibration	Run a full motor calibration.
Set maximum motor current	Set the maximum motor current and maximum regeneration current.
Set safety limits	Set the position safety limits.
Test mode	Set or trigger a test mode.
Set PID constants	Set the P, I, and D constants of the closed-loop controller that tries to maintain the motion trajectory.
Set max allowable position deviation	Set the maximum distance that the actual motor position (as measured by the hall sensors) is allowed to deviate from the commanded position; exceeding it raises fatal error 45, ERROR_POSITION_DEVIATION_TOO_LARGE. Takes effect immediately (not queued).
CRC32 control	Enable or disable the CRC32 layer of the protocol for this device, in both directions.

Status & Monitoring

Command	Description
Get current time	Returns the device's absolute time as an unsigned 64-bit count of true microseconds elapsed since power-up or since the last 'Reset time'; it is not wall-clock time.
Get n queued items	Returns the number of items currently in the movement queue as a single byte between 0 and 32 (the queue holds at most 32 items).
Get hall sensor position	Get the position as measured by the hall sensors, which is the actual shaft position.
Get status	Get the motor's status flags and fatal error code.
Control hall sensor statistics	Turn hall sensor statistics gathering on or off.
Get hall sensor statistics	Read back the statistics gathered from the three analog hall sensors.
Get position	Get the current commanded (desired) position, i.e. the motion profile's internal target position that advances every control tick.
Get comprehensive position	Get the commanded position, the hall sensor (measured) position, and the external encoder count in one shot.
Get supply voltage	Get the measured voltage of the power supply.
Get max PID error	Get the minimum and maximum position error observed by the closed-loop PID controller since the last read, then reset the accumulators.
Get temperature	Get the temperature measured by a dedicated analog sensor on the motor driver PCB (not the microcontroller die and not the motor windings).
Get debug values	Get a snapshot of diagnostic values: the maximum acceleration, maximum velocity, and current commanded velocity (all in raw internal units; no unit conversion is applied to any output of this command), the measured velocity, the number of time steps remaining in the currently executing queue item, four general-purpose debug values, execution-time profiler counters for the motor control loop, raw hall sensor voltages, the commutation position offset and phase-reversal flag, hall position delta statistics, and the motor PWM voltage.
Get communication statistics	Get six communication error counters and optionally reset them.

Device Management

Command	Description
Time sync	Sends the master's clock reading to the motor so the motor can discipline its clock rate.
Get product specs	Get two product constants needed for motion math.
Detect devices	Detect all devices connected on the RS485 interface.
Set device alias	Set the device's one-byte alias, save it to nonvolatile flash, and then automatically reset the device.
Get product info	Get the product information: product code (model number), firmware compatibility code, hardware version, serial number, and unique ID, returned as a 28-byte response payload.
Firmware upgrade	Write one 2048-byte page of application firmware to flash.
Get product description	Get a short product description string.
Get firmware version	Get the firmware version, or the bootloader version if the device is currently running the bootloader.
Ping	Send a 10-byte payload and the device immediately responds with the same 10 bytes (executes at once, not queued).
Vibrate	Cause the motor to vibrate by rapidly alternating the open-loop PWM voltage between +50 and -50 (a fixed, hard-coded amplitude), or stop vibrating.
Identify	Identify a motor by flashing its green LED rapidly: 30 flashes at roughly 90 ms each, about 2.7 seconds total, after which the normal 1-second heartbeat blink resumes.

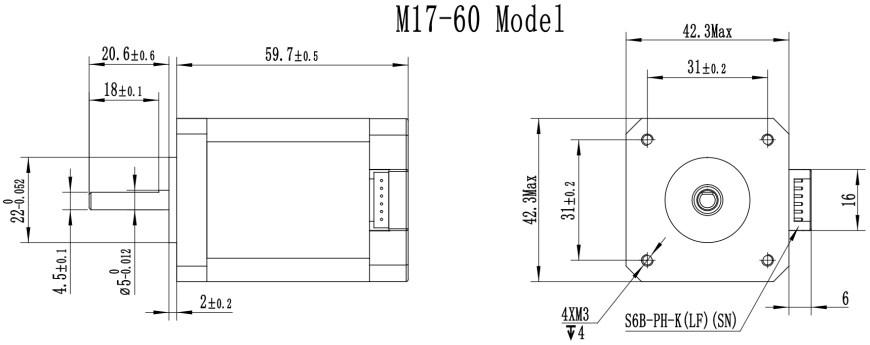
Other

Command	Description
Capture hall sensor data	Capture hall-sensor data and stream it back.
Read multipurpose buffer	Read out and clear the multipurpose data buffer, which holds at most one dataset at a time.

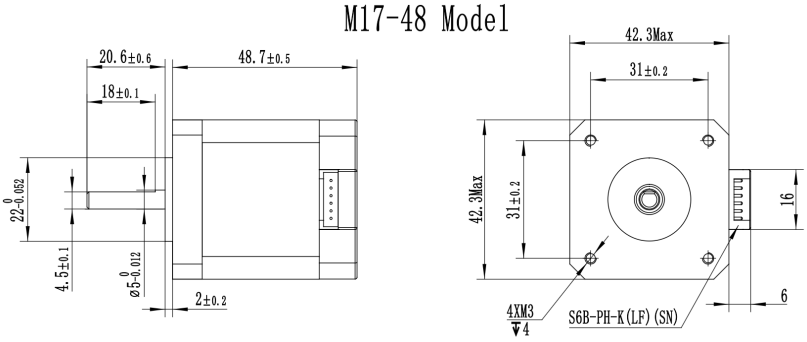
Mechanical Specifications

Parameter	M17-60	M17-48	M17-40	M17-34
Dimensions (LxW)	42.2x42.2 mm	42.2x42.2 mm	42.2x42.2 mm	42.2x42.2 mm
Height	59.7 mm	48.7 mm	40.1 mm	33.5 mm
Shaft Length	20.6 mm	20.6 mm	20.6 mm	20.6 mm
Weight	470g	360g	285g	210g
Protection Class	IP20	IP20	IP20	IP20

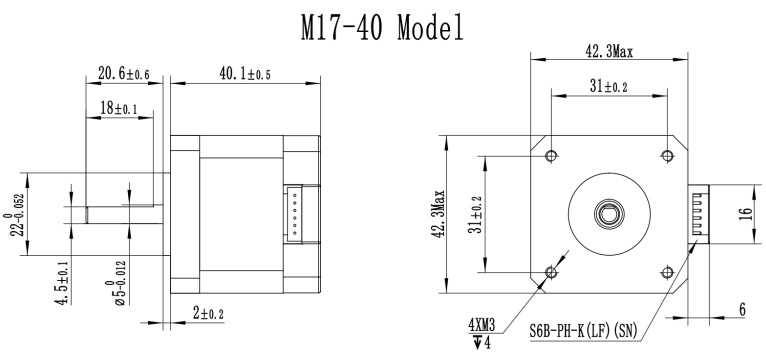
M17-60 Model



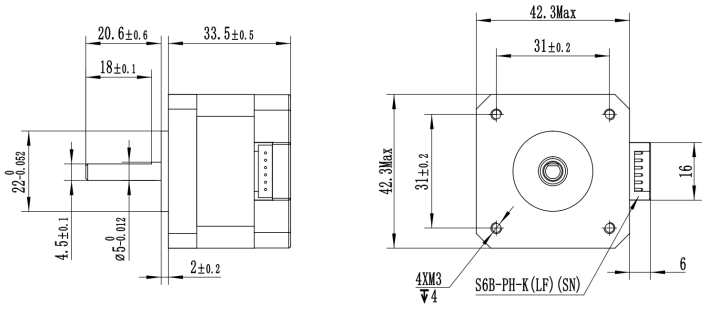
M17-48 Model



M17-40 Model



M17-34 Model



Technical Specifications

Parameter	M17-60	M17-48	M17-40	M17-34
Operating Voltage	12-24V	12-24V	12-24V	12-24V
Rated Torque	0.65 N.m	0.55 N.m	0.42 N.m	0.28 N.m
Maximum Speed	560 RPM	560 RPM	560 RPM	560 RPM
Maximum Current	1.1A	1.1A	1.1A	1.0A
Rated Power	26.4W	26.4W	26.4W	24.0W

Operating Conditions

Parameter	Specification
Operating Temperature	0C to +80C
Storage Temperature	-20C to +60C
Humidity Range	20% to 80% RH (non-condensing)
Installation Environment	Indoor use only

Python Library

Our Python library provides a comprehensive interface for controlling M17 Series Servomotors. The library handles all low-level communication protocols and unit conversions, making it easy to integrate servomotor control into your Python projects.

Installation

Install the servomotor library using pip:

```
pip3 install servomotor
```

Example: Basic Motor Control

The following example demonstrates how to connect to a servomotor, enable the mosfets, and perform a trapezoid move:

```
#!/usr/bin/env python3
"""
Minimal trapezoid move: rotate 1 turn in 1 second.
Edit ALIAS below if needed. Uses rotations and seconds.
"""
import time, servomotor
from servomotor import communication

# Hard-coded settings for a minimal demo
ALIAS = 'X'                               # Device alias, change if needed
SERIAL_PORT = "/dev/tty.usbserial-110"  # Serial device path; change if needed (e.g., "COM3" on
                                          #  Windows)
DISPLACEMENT_ROTATIONS = 1.0              # 1 rotation
DURATION_SECONDS = 1.0                    # 1 second
DELAY_MARGIN = 0.10                       # +10% wait margin because the motor's clock is not
                                          #  perfectly accurate

communication.serial_port = SERIAL_PORT # if you comment this out then the program
                                          #  should prompt you for the serial port or it will use
                                          #  the last used port from a file
servomotor.open_serial_port()

m = servomotor.M3(ALIAS, time_unit="seconds", position_unit="shaft_rotations", verbose=0)
m.enable_mosfets()
m.trapezoid_move(DISPLACEMENT_ROTATIONS, DURATION_SECONDS)
time.sleep(DURATION_SECONDS * (1.0 + DELAY_MARGIN))
m.disable_mosfets()

servomotor.close_serial_port()
```

Key Features

- Support for all motor commands
- Automatic unit conversion for time, position, velocity, acceleration, current, voltage, and temperature
- Error handling and status monitoring
- Support for multiple motors on the same bus
- Get up and running in just a few lines of code

Documentation

For complete API documentation and more examples, visit:
https://github.com/tomrodinger/servomotor/tree/main/python_programs

Arduino Library

The Arduino library for M17 Series Servomotors provides an easy-to-use interface for controlling motors from Arduino boards. The library supports hardware serial communication and includes all essential motor control functions.

Installation

Install the library through the Arduino Library Manager:

- 1. Open Arduino IDE
- 2. Go to Sketch → Include Library → Manage Libraries
- 3. Search for "Servomotor"
- 4. Find the library called "Servomotor by Gearotons" and click Install

Manual Installation

Alternatively, if you are an expert, you can grab the code for the Arduino library here: https://github.com/tomrodinger/Servomotor_Arduino_Library

Example: Basic Motor Control

This example shows how to initialize a motor, enable mosfets, and perform a trapezoid move:

```
// Minimal Arduino example: Trapezoid move using built-in unit conversions
// Goal: spin the motor exactly 1 rotation in 1 second, then stop.
// Sequence:
//  enable MOSFETs -> trapezoidMove(1.0 rotations, 1.0 seconds) -> wait 1.1s -> disable MOSFETs.
//
// Notes:
// - This uses the library's unit conversion (no raw counts/timesteps).
// - Configure Serial1 pins for your board (ESP32 example pins below).
// - Motor is created AFTER Serial1.begin(...) so hardware UART pins are set first.

#include <Servomotor.h>

#define ALIAS 'X'                   // Device alias
#define BAUD 230400                 // RS485 UART baud rate
#define DISPLACEMENT_ROTATIONS 1.0f // 1 rotation
#define DURATION_SECONDS 1.0f       // 1 second
#define TOLERANCE_PERCENT 10        // +10% wait margin because the motor's clock is not
                                    //  perfectly accurate
#define WAIT_MS ((unsigned long)(DURATION_SECONDS * 1000.0f * (100 + TOLERANCE_PERCENT) / 100))

// Example RS485 pins for ESP32 DevKit (change as needed for your board)
#if defined(ESP32)
#define RS485_TXD 4             // TX pin to RS485 transceiver
#define RS485_RXD 5             // RX pin from RS485 transceiver
#endif

void setup() {
  Serial.begin(115200); // Console serial for debugging

  // Create the motor; serial port opens on first instantiation.
#if defined(ESP32)
  Servomotor motor(ALIAS, Serial1, RS485_RXD, RS485_TXD);
#else
  Servomotor motor(ALIAS, Serial1);
#endif

  // Use units: rotations for position, seconds for time
  motor.setPositionUnit(PositionUnit::SHAFT_ROTATIONS);
  motor.setTimeUnit(TimeUnit::SECONDS);

  motor.enableMosfets();
  motor.trapezoidMove(DISPLACEMENT_ROTATIONS, DURATION_SECONDS);
  delay(WAIT_MS);
  motor.disableMosfets();
}

void loop() {
}
```

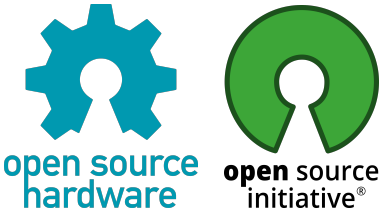
Our Mission

At Gearotons, we believe that the future belongs to those who understand and can work alongside artificial intelligence and automation systems. Our mission is to empower the next generation of innovators by making advanced motion control technology accessible, understandable, and affordable for makers, educators, and engineers worldwide. We are committed to breaking down the barriers that have traditionally separated students and hobbyists from professional-grade automation tools. By providing integrated solutions that eliminate complexity without sacrificing capability, we enable hands-on learning experiences that prepare young minds for a world where human creativity and machine precision work in harmony. Our core belief is that every student should have the opportunity to experiment with the building blocks of modern automation—from robotics and CNC systems to scientific instruments. Through our educational partnerships and open-source approach, we're fostering a generation that doesn't just use technology, but truly understands and shapes it. Together, we're building the foundation for innovators who will define the amazing future.

Open Source

We believe in making the world better through technology. Software, firmware, and PCB design files are available here:

https://github.com/tomrodinger/servomotor



Datasheet Version: 1.10 Release Date: Jul. 15, 2026

(c) 2026 All specifications subject to change without notice.