SSG-48 gripper

Description	Documentation for the SSG-48 adaptive electric gripper by Source Robotics.
Repository	https://github.com/Source-Robotics/SSG48-gripper-docs
Copyright	© 2026 Source Robotics d.o.o. · Text and images CC BY-SA 4.0 · Code examples MIT

Table of Contents

- SSG-48 Gripper Manual
  - SSG-48 Gripper Manual
  - Before you start
    - Quick start
    - About this manual
  - Chapters
    - 1. Overview
      - Overview
      - The gripper at a glance
      - Hardware versions
      - What you need
    - 2. Safety
    - 3. Technical data
      - Technical data
      - Specifications
      - Force and speed
      - Connector and pinout
      - Dimensions and CAD
      - Data for robot integration
    - 4. Installation
      - Installation
      - Mounting on a robot
      - The cable
      - Jaws
      - CAN termination
    - 5. First setup
      - First setup
      - Motor GUI
      - Calibrate, activate, move
    - 6. Operation

- Operation
- How the gripper works
- Over CAN
- Over UART
- With Python
- With ROS 2
- On PAROL6 and PAR6
- Errors and recovery
- 7. Maintenance
- 8. Firmware
- 9. Troubleshooting
- Troubleshooting
- Known issues

Build guide

- Build guide
- Bill of materials
- Printing the parts
- Mechanical assembly
- Driver setup and first run

Appendix

- Performance tests
- Firmware release notes
- Versions and compatibility

Resources

- Glossary
- CAD and downloads

API reference

- API reference
- CAN frame tool and downloads
- CAN messages
- CAN messages
- Gripper
- System

- Status requests
- UART commands
  - UART commands
  - Gripper
  - Driver setup
  - Status and utility
- Errors and status bits
- Python library

Guides

- Guides
- Basic control in Python
- Teleoperation with two grippers

Support

- Getting help
- Report a problem
- Contributing

SSG-48 Gripper Manual



Document number: SR-5004

SSG-48 gripper release: 2

Firmware: Spectral 107

Revision: 2026.B

Published: 2026-11-04

Applies to: SSG-48 gripper release 2 with the Spectral Micro driver PCB v2.0, Spectral firmware 107

Original instructions

Quick start | About this manual

Source Robotics d.o.o. publishes this document for the SSG-48 adaptive electric gripper. It is revised as the gripper and its firmware change; the document number, the gripper release, the firmware, the revision and the date above identify the version you are reading, and the document history on the About page says what changed. The version selector in the header switches between gripper releases. The gripper runs the Spectral Micro driver's firmware: take care that the firmware named under *Applies to* matches what the gripper reports. The text and the images are licensed CC BY-SA 4.0, the code examples MIT.

The SSG-48 is open source, and that is important to us. Everything the gripper is made of is published: the CAD and print files, the bill of materials, the assembly instructions, the driver's firmware and the software that drives it. Buy the gripper assembled, or build one yourself; change the jaws or the body for your application and fit it to any robot. The build guide is the order to do it in.

Quick start

One-page quick start: choose the cable, check the bus, connect Motor GUI, calibrate, and make the first gentle open and close. Each step says what to check before continuing.

The manual

Safety, Technical data, Installation, First setup, Operation, Maintenance, Firmware, Troubleshooting.

On a robot

PAROL6 and PAR6, ROS 2, Python: the gripper from the robot's own software, from a ROS 2 node, or from your program over CAN.

API reference

The gripper's CAN messages and UART commands, the status bits and the Python library, read from the firmware.

Open source: build your own

The build guide: the repository's bill of materials, printing, the assembly, the driver's setup and the first run. The same manual applies.

Before you start

Quick start

From an assembled SSG-48 gripper to its first gentle open and close, using Motor GUI on a PC. For a gripper already connected to a robot, use On PAROL6 and PAR6.

Before connecting power

Keep fingers and loose objects out of the stroke. Calibration opens and closes the jaws fully. A wrong cable can put 24 V on the CAN pins and destroy the driver: check the numbered pinout.

1. Choose your starting point

Your gripper	Start here
Bought assembled	Its motor is already calibrated and the driver configured. Continue below; the jaws still need calibration after each power-up.
Self-built, or a replacement driver or motor	Finish Driver setup and first run first, then return here. Do not apply a factory preset to a bought gripper just to try this page.
No hardware yet	Open Motor GUI, choose **Mock CAN (demo)** and **Connect**, then switch to **Gripper**. The demo is an SSG-48; the control panel is shared with MSG.

2. Gather and mount

You need a mounted gripper with jaws/fingers fitted, a **24 V supply**, a correctly wired **M8 cable**, a **CANvas USB-to-CAN adapter**, and Chrome or Edge. The What you need page lists the parts; Mounting and Jaws cover the mechanics.

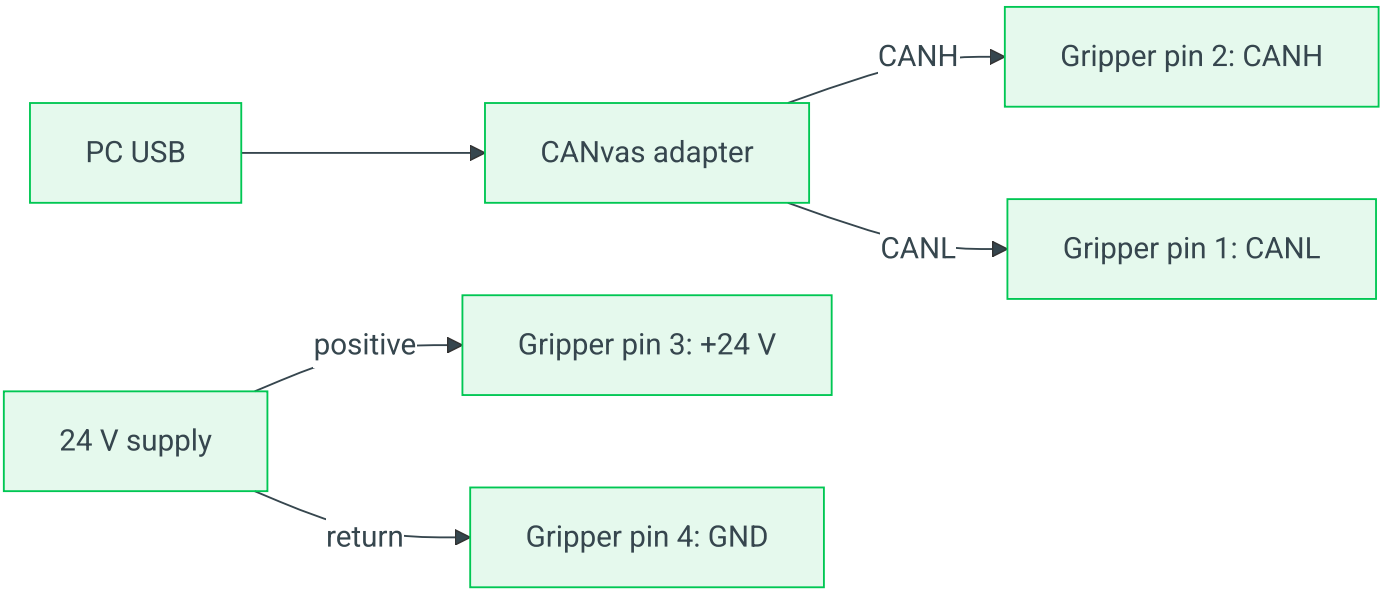
Check

The body cannot shift, the jaws travel freely, and the cable cannot enter their path.

3. Wire with power off

1. Check the cable against Connector and pinout: gripper pin 3 is +24 V, pin 4 GND, pin 2 CANH, pin 1 CANL. Check continuity before the first power-up.
2. Connect the supply and the CANvas adapter as shown below.

3. Set the gripper's CAN termination ON for this two-node bench bus; the CANvas adapter provides the other end. Leave the gripper at its shipped node ID, **0**, unless you already assigned another.



Check, unpowered

The supply polarity and cable pinout match, and CANH-to-CANL measures about **60 Ω** with the bus connected. Otherwise check The cable and CAN termination before powering it.

4. Connect Motor GUI

Power the gripper, plug the CANvas adapter into USB, and open motorgui.com. Choose **CAN (slcan)** at **1 Mbit/s**, press **Connect**, and choose the adapter's serial port. Select node **0** (or your assigned ID) and switch from **Motor** to **Gripper**.

Check

Position and current feedback arrive. *Not calibrated* after power-up is normal. If feedback is missing, check the power, port, node ID and termination: Troubleshooting. If a fault disables the controls, resolve it using Errors and recovery.

The illustrated Motor GUI guide shows the panel.

5. Calibrate and activate

If the panel shows an error, press **Clear errors** first. With the stroke empty and the gripper deactivated, press **Calibrate**, read the prompt, and press **Start calibration**. Let the jaws find both end stops, and press nothing else until the panel says **Calibrated**: a move or a clear during the calibration ends it. Follow Calibrate, activate, move for the full procedure, then **Activate** and check for faults.

Check

The panel says **Calibrated**, **Activated**, and **No errors** before you command a move. A calibration that times out after 30 seconds has not found its end stops; do not continue as if it passed. Use Calibration troubleshooting.

6. Make the first move

Set **Speed 30** and **Current 300 mA**, applying each with its **Set** button. Press **Open**, then **Close**. After that works, put a light test object between the jaws and repeat, keeping fingers clear.

Check

The position follows the command; on an object the jaws stop before fully closed and the panel reports an object. If it does not move, slips or reports an error, use Troubleshooting.

The current limit sets an approximate force limit; Force and speed explains the relationship and the motor's limits.

7. Finish and continue

Remove the test object before deactivating or switching off: the grip may release. Disconnect Motor GUI and switch off the supply before changing cables.

Next task	Guide
Control it from a program	With Python, With ROS 2
Put it on a robot	On PAROL6 and PAR6
Inspect a CAN frame or download DBC definitions	CAN frame tool and downloads
Diagnose a problem	Troubleshooting, Errors and recovery

About this manual

The SSG-48 Gripper Manual covers the SSG-48 adaptive electric gripper from unpacking to a gripper that works on a robot: mounting and cabling, the first setup with Motor GUI, control over CAN and UART, from Python, from ROS 2 and from the PAROL6 and PAR6 robots, and firmware updates. It is written for a maker, a student or an engineer who can plug in a cable and run a Python script. Read Safety before you power anything; First setup is the shortest path to a gripper that moves.

The gripper is a Spectral Micro BLDC driver in gripper mode. The driver itself, from its full command set to its calibration and its bootloader, is documented on the Spectral site; this manual links there instead of repeating it.

Finding your way

Four tabs: **SSG-48 Gripper Manual**, the numbered chapters, read from the start; **API reference**, the gripper's CAN messages, UART commands, status bits and Python calls, for looking things up (the driver's complete reference is on the Spectral site); **Guides**, worked examples; **Support**, help and problem reports.

The dropdown next to the site name switches between gripper releases, and a page of an older release says so in a bar at the top. The repository card in the header opens the product repository; every page ends with a link to edit it and one to report a problem with it.

Warnings and notes

Note

Points out something that is easy to miss or that changes what follows: a default, a limit, a side effect.

Notice

The robot, the tool, or data on the control box can be damaged or lost if this is ignored. Nobody is at risk.

Danger

Someone can be injured if this is ignored. Stop and read the whole box before acting on the step it belongs to.

Related documents

Document	Number	Covers
SSG-48 Gripper Manual	SR-5004	the SSG-48 adaptive electric gripper: safety, technical data, installation, setup, operation over CAN, UART, Python and ROS 2, firmware, the build guide; the gripper's API reference; guides

A document marked with a firmware release describes SSG-48 gripper release 2 running Spectral firmware 107. The version selector in the header switches between gripper releases; a new firmware release for the same gripper updates the pages in place. The compatibility table says which firmware, Python library and hardware revisions work together.

The driver inside the gripper is documented at https://docs.source-robotics.com/Spectral-BLDC-docs/. The MSG gripper, the SSG-48's successor on the STEPFOC stepper driver, is at docs.source-robotics.com/MSG-gripper-docs. The robots the gripper is made for: docs.source-robotics.com/PAROL-docs and docs.source-robotics.com/par6.

Document history

The manual is versioned by gripper release, the version the product repository tags (the selector in the header); within a release it follows the Spectral firmware the gripper runs, with a new revision letter per update. The text and the images are licensed CC BY-SA 4.0, the code examples MIT; the hardware design is published under the CERN Open Hardware Licence v2, strongly reciprocal (CERN-OHL-S-2.0), in the product repository. The Source Robotics name, the product names and the logos are not covered by any of these.

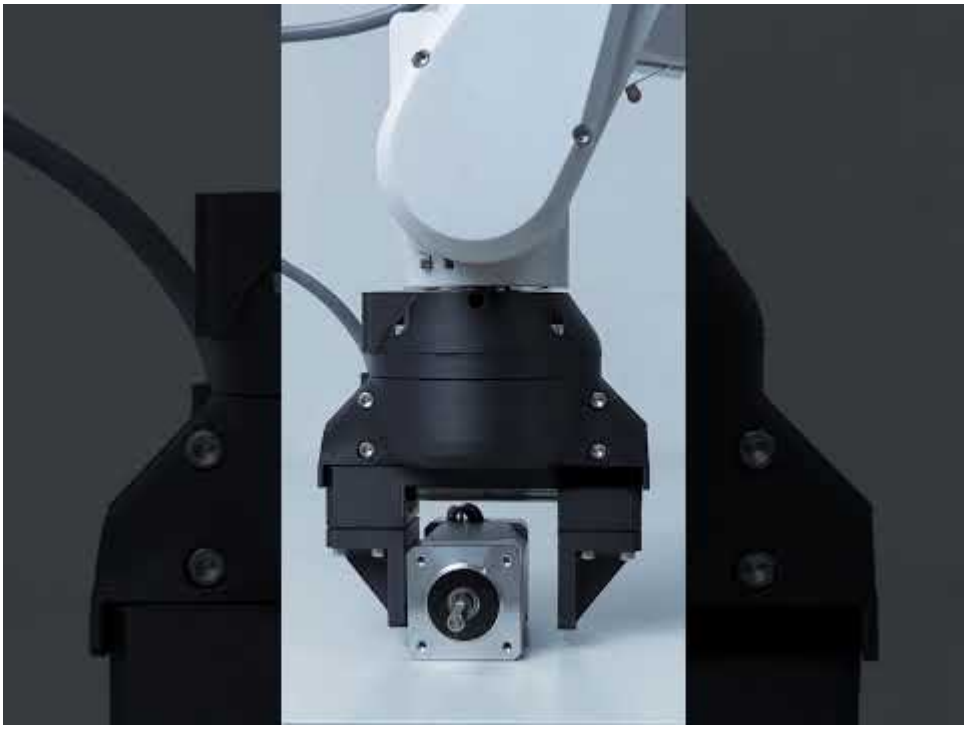
Revision	Date	Changes
2026.B	2026-11-04	Gripper release 2, with the files as downloads on the release; known issues; data for robot integration (frame, mass, inertia, TCP); check boxes after each step and Windows, Linux and macOS tabs; numbered figure captions; the downloads with their versions; safety at the jaws, static and cleaning; the thermistor and the thermal cement in the BOM; the hardware licence CERN-OHL-S-2.0
2026.A	2026-11-03	First version in this form: the manual chapters, the generated CAN and UART reference; gripper release 1.1, firmware 107

Chapters

1. Overview

Overview

What the SSG-48 gripper is and how it works, the versions of its hardware, and what you need around it to get it gripping.



The video of the gripper in action.

In this chapter

- The gripper at a glance
- Hardware versions
- What you need

The gripper at a glance

The SSG-48 is a two-jaw parallel gripper driven by a brushless gimbal motor under field-oriented control. The Spectral Micro driver inside the body regulates the motor's current, so a move sets not only where the jaws go and how fast, but how hard they press: the current limit is the grip force, from about 8 N at 150 mA to about 80 N at 1500 mA. The jaws stop on an object, hold it with that force, and the driver reports that an object was found.



Figure 1.1: The SSG-48 gripper with pinch fingers.

The parts

Part	What it does
GBM5208-75 gimbal motor	the actuator: a brushless motor wound for high torque at low speed and low current
Spectral Micro driver, inside the body	field-oriented control of the motor, the gripper state machine, the CAN and UART interfaces
Pinion on the motor, two racks on a linear rail	the pinion turns, the two racks move in opposite directions: the jaws open and close symmetrically
MGN7 linear rail, 100 mm	carries the two jaw plates; 48 mm of stroke
Jaw plates	printed, with a four-hole pattern for your own fingers; fin-ray and pinch fingers in the repository
Thermistor in the motor windings	temperature protection: the gripper stops at 75 °C
M8 4-pin connector	24 V and the CAN bus, one cable to the robot
Coupler connector	mounts the gripper on the PAROL6 or PAR6 wrist, on a UR5 flange, or on your own with the template

The driver's status LED, its UART port and its CAN termination switch are inside the body, under the coupler connector.

What the firmware does

After power-up the gripper calibrates by finding both end stops. Then it takes move commands: a position from 0 (open) to 255 (closed), a speed from 0 to 255 and a current limit in mA. When the jaws stop short of the target with the current at its limit, an object is in the way and is held; when they reach the target, the position is held. Every answer carries the state: moving, object found while closing or opening, or at the position. The gripper stops itself on an over-temperature, an E-stop bit from the host, a CAN watchdog that runs out, and every fault the driver detects. How the gripper works and Errors and recovery have the details.

How it is controlled

- **CAN**, 1 Mbit/s, on the same cable as the power: one message moves the gripper and is answered with its state. From a PC with the CANvas adapter, from a ROS 2 node, from PAROL6 or PAR6, or from your own controller. See Over CAN.
- **Python**, the `Spectral-BLDC` library on top of CAN. See With Python.

- **Motor GUI** in the browser: connect, calibrate, open and close, watch the plots. See Motor GUI.
- **UART**, the driver's serial port inside the body: for configuration, and for a self-built gripper. See Over UART.

The MSG gripper is the SSG-48's successor: the same protocol and software, with a stepper and the STEPFOC driver, three rail lengths and a camera mount. Programs written for one run on the other.

Hardware versions

The SSG-48 exists in two mechanical versions, both in the product repository's Gripper STL parts:

Version	Folder	What differs
V1	`V1`	the original body, lid, gear and magnet shaft
V2	`V2`	the "large bearing" body, lid, gear and magnet shaft: a larger bearing on the pinion for a stiffer, longer-lasting mechanism

The electronics, the motor, the rail, the jaw plates, the coupler connectors and everything about control are the same in both; a gripper from the shop is a V2. The product release V2 (November 2026) carries the parts of both versions; the release V1.1 (2024) is the version tested with PAROL6.

The jaw plates, the coupler connectors (`coupler_connector_PAROL6` , `coupler_connector_PAR6` , `connector_coupler_generic`) and the fingers are common to both versions.

What you need

The gripper

The SSG-48 is sold assembled, calibrated and configured in the Source Robotics shop. It ships with the current Spectral firmware and the bootloader installed, its driver in gripper mode and its motor calibrated; the gripper calibration (finding the end stops) runs after every power-up and is part of normal use.

Or build it. The SSG-48 is open source, and that is important to us: the CAD and printable files, the bill of materials, the assembly manual and video, the firmware and the software are all published. The build guide is the order to do it in.

What is in the box

The gripper, assembled, calibrated and configured, in black or grey. **Not in the box:** the fingers, which you print from the repository; the cable, sold separately; the 8 mm shaft coupler for a PAROL6, which you ask for when ordering; a CAN adapter for a PC.

A cable and a 24 V supply

The gripper is powered and controlled through one M8 4-pin cable: 24 V, GND, CANH, CANL.

Where the gripper goes	Cable
PAROL6	the PAROL6 gripper cable from the shop
PAR6	the PAR6 gripper cable (the shop; coming soon), wired differently from the PAROL6 one
a bench, or your own robot	a cable you make from an M8 4-pin connector and four wires, see The cable

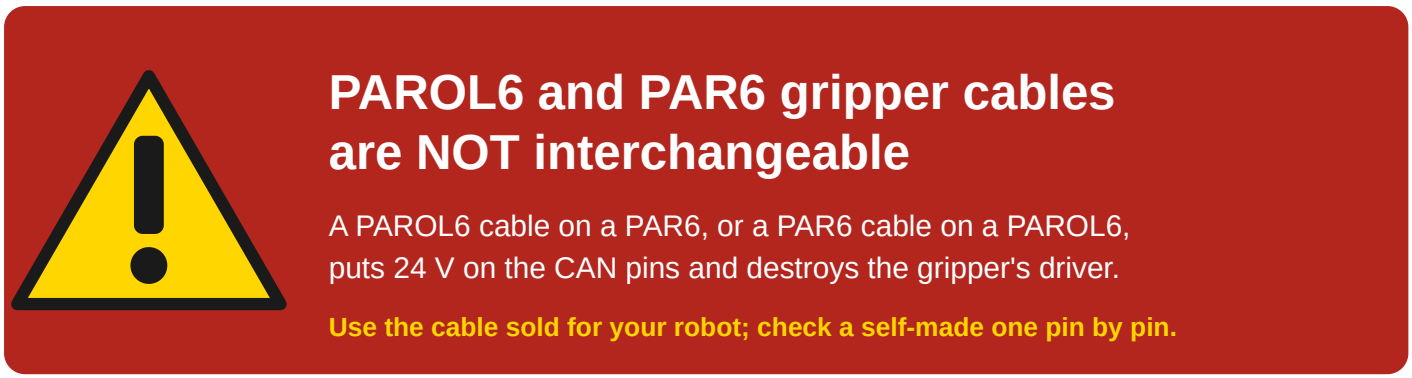


Figure 1.2: PAROL6 and PAR6 gripper cables are not interchangeable.

The two robot cables differ

PAROL6 and PAR6 put the four signals on different pins. A cable for the wrong robot, or a self-made cable with a wrong pin, puts 24 V where the CAN bus belongs and destroys the gripper's driver. Check the pinout before you plug in a cable for the first time.

On the bench, a 24 V supply with a current limit (2 A is plenty for one gripper) is the safest source.

A CAN interface

The gripper talks CAN. On a robot, the robot's control board provides the bus. On a PC, the CANvas USB-to-CAN adapter does; it shows up as a serial port and works with Motor GUI, the Python library and the ROS 2 drivers. Any adapter that `python-can` supports works with the library.

Software

- **Motor GUI**, motorgui.com, in the browser over the CANvas adapter: the recommended tool to test and set up a gripper, with a gripper panel for the SSG-48 and the MSG; see Motor GUI. The earlier SSG-gripper-GUI, a Python program, is legacy.
- **Python**: `pip install Spectral-BLDC`, the library the ROS 2 drivers are built on; see With Python.
- **ROS 2**: the ssg48_adaptive_electric_gripper_ros2 package, or the MSG gripper's driver, which speaks the same protocol; see With ROS 2.
- **PAROL6 and PAR6**: the robots' own software drives the gripper; see On PAROL6 and PAR6.
- **For a self-built gripper**: a 3.3 V USB-to-serial adapter (serial adapter) for the driver's UART port, to configure the driver with Motor GUI or a terminal; see Driver configuration. For a self-build we recommend the Spectral Micro BLDC starter kit: it has this adapter and the rest of what the driver's setup needs (Kits and the shop).

Optional

- **Fingers** for your objects: the fin-ray and pinch fingers in the repository, or your own on the jaw plate template; see Jaws.

Safety

The SSG-48 gripper is a force-controlled gripper for collaborative and research robots, not a certified safety component. Whoever puts it on a machine is responsible for the risk assessment of the gripper, the robot and everything around them. Read this chapter before you power the gripper; the rules here are the ones that injure a hand or destroy the gripper when they are ignored.

Mechanical

The jaws

The jaws are a pinch point. At a high current limit they close with up to 80 N, enough to crush a finger, and at full speed they travel the whole stroke in a fraction of a second. Never put a finger between the jaws of a powered gripper. Keep hands away during calibration, which moves the jaws over the whole stroke without any command from you.

Start low

Test every new setup with a low current and speed: current 300 mA, speed 30. Raise them step by step once the application behaves; Force and speed says what each value means.

Objects

A gripped object that slips, or that the jaws crush, can fly or fall. The grip force is a current limit, not a measurement: a hard object takes the full force, a soft one deforms until the force is reached.

Hair, clothing and jewellery

The jaws catch more than fingers: long hair, loose sleeves, gloves, lanyards, rings and bracelets. Tie back long hair and take off what can be caught before you work at a powered gripper, and keep it out of the jaws' reach while they move. Wear safety glasses where a part can slip out of the grip and fly.

Electrical

The cable

The gripper's M8 connector carries 24 V next to the CAN bus. A cable with the wrong pinout, including the cable of the other Source Robotics robot, puts 24 V on the CAN pins and destroys the driver. Use the cable made for your robot or check every pin of your own; see The cable.

Supply

24 V DC. The driver inside tolerates 12 to 29 V, but the gripper's force and speed figures assume 24 V.

Do not unplug a powered gripper

Connect and disconnect the cable with the supply off. Hot-plugging the connector can damage the driver.

Static

Static electricity damages the Spectral Micro driver inside the body. Unplug the supply and touch earthed metal before you open the body, set the termination switch or handle the driver, and hold the board by its edges.

Temperature

The motor has a thermistor in its windings. At 75 °C the driver raises a temperature error and the gripper goes idle; it clears when the motor has cooled and the error is cleared. A gripper holding an object at a high current for long heats up: a hold at 300 mA ran for 24 hours at 62 °C, a hold at 1600 mA tripped the limit in four minutes (Performance tests). For a long hold, use the lowest current that holds the object. A self-built gripper without a thermistor in the windings has no protection and can burn out its motor; see Driver configuration.

Stopping the gripper

- The E-stop bit of the move command (`Send_gripper_data_pack`) stops the gripper and raises the gripper E-stop error. The error stays as long as the last command carries the bit: send a command with the bit at 0, then clear it (Errors and recovery).
- `Send_ESTOP` to the driver, or `#Idle` over UART, stops it as well.
- The CAN watchdog (Send_Watchdog) stops the gripper by itself when the host stops talking, which is the right protection for a program that may crash mid-grip.
- Cutting the supply releases the grip: a motor without current holds nothing beyond the mechanism's friction, so an object can drop.

None of these is a certified safety function. A machine that can hurt someone needs an emergency stop that cuts the power, independent of the gripper and the robot software.

Care

The gripper is not waterproof or dustproof: the rail and the racks are open to dust, chips and liquids. Keep it out of wet or dusty places. What to check, and when to grease the mechanism, is in Maintenance.

Liability

The SSG-48 gripper is sold as a component for people who build their own machines, and its designs are published for people who build the gripper itself. The hardware designs, the firmware and this documentation are under development and may contain errors. Whoever builds or operates a machine with the gripper is responsible for its safe design, assembly and operation. The full disclaimer is in the product repository.

3. Technical data

Technical data

The numbers: the specifications of the gripper and its variants, what the current and speed settings mean in newtons and millimetres per second, the connector and its pinout, where the CAD files are, and what a robot's software needs to carry the gripper.

In this chapter

- Specifications
- Force and speed
- Connector and pinout
- Dimensions and CAD
- Data for robot integration: the frame, the mass, the centre of mass, the inertia and the tool centre point

Specifications

Electrical

	Value
Supply voltage	24 V DC; the driver tolerates 12 to 29 V, the force and speed figures assume 24 V
Idle power	0.5 W
Motor current	set per move: up to 1500 mA for a grip, 300 mA for a long hold; see Force and speed
Interface	CAN 2.0, 1 Mbit/s, 11-bit identifiers; node ID 0 as shipped
Connector	M8, 4 pins: 24 V, GND, CANH, CANL
CAN termination	120 Ω, switchable; the switch is on the driver inside the body
Service port	the driver's UART, 3.3 V, 256000 baud, inside the body
Driver	Spectral Micro BLDC controller, field-oriented control, 14-bit encoder on the motor
Temperature protection	thermistor in the motor windings, 75 °C limit (changeable with `#Tlim`)

Mechanical

	Value
Stroke	48 mm
Grip force	ideal combined force up to about 80 N at 1500 mA; see the model assumptions in Force and speed
Jaw speed	up to about 180 mm/s per jaw at speed 255; see Force and speed
Position resolution	1/255 of the stroke: 0.19 mm per step of the position byte

	Value
Mass	440 g
Width	104 mm over the jaw plates; 60.7 mm from the top of the jaw plates to the driver board, without a coupler connector
Drive	GBM5208-75 gimbal motor, pinion (6 mm pitch radius) on two racks, MGN7 linear rail
Grasps	external (closing on the object) and internal (opening inside a hollow object)
Mounting	coupler connectors for PAROL6 and PAR6, a UR5 adapter plate, a template for your own
Fingers	on the jaw plates' four-hole pattern; at most 80 mm tall and 60 mm wide
Body material	PETG, printed

Environmental

	Value
Operating temperature	−5 to 65 °C
Protection	none: not waterproof or dustproof

Force and speed

A move command carries a current limit in mA and a speed from 0 to 255. This page estimates the resulting force and speed. Force and payload values are ideal model estimates, not measured ratings.

Grip force

The driver limits the motor's current to the value you command, so the torque the motor produces is limited to $T = K_t I_q$, with K_t the motor's torque constant in Nm/A and I_q the current limit in amperes (the command is in mA). The pinion shares that torque between two racks. Here F is the **combined normal force of both jaws**, not the force at one jaw:

$$F = \frac{K_t I_q}{r}, \qquad K_t = 0.325\ \mathrm{Nm/A}, \quad r = 6\ \mathrm{mm}$$

K_t = 0.325 Nm/A is the assumed torque constant used by this model and r the pitch radius of the pinion. They give an ideal combined force of about 54 N per ampere: about 8 N at 150 mA and 80 N at 1.5 A.

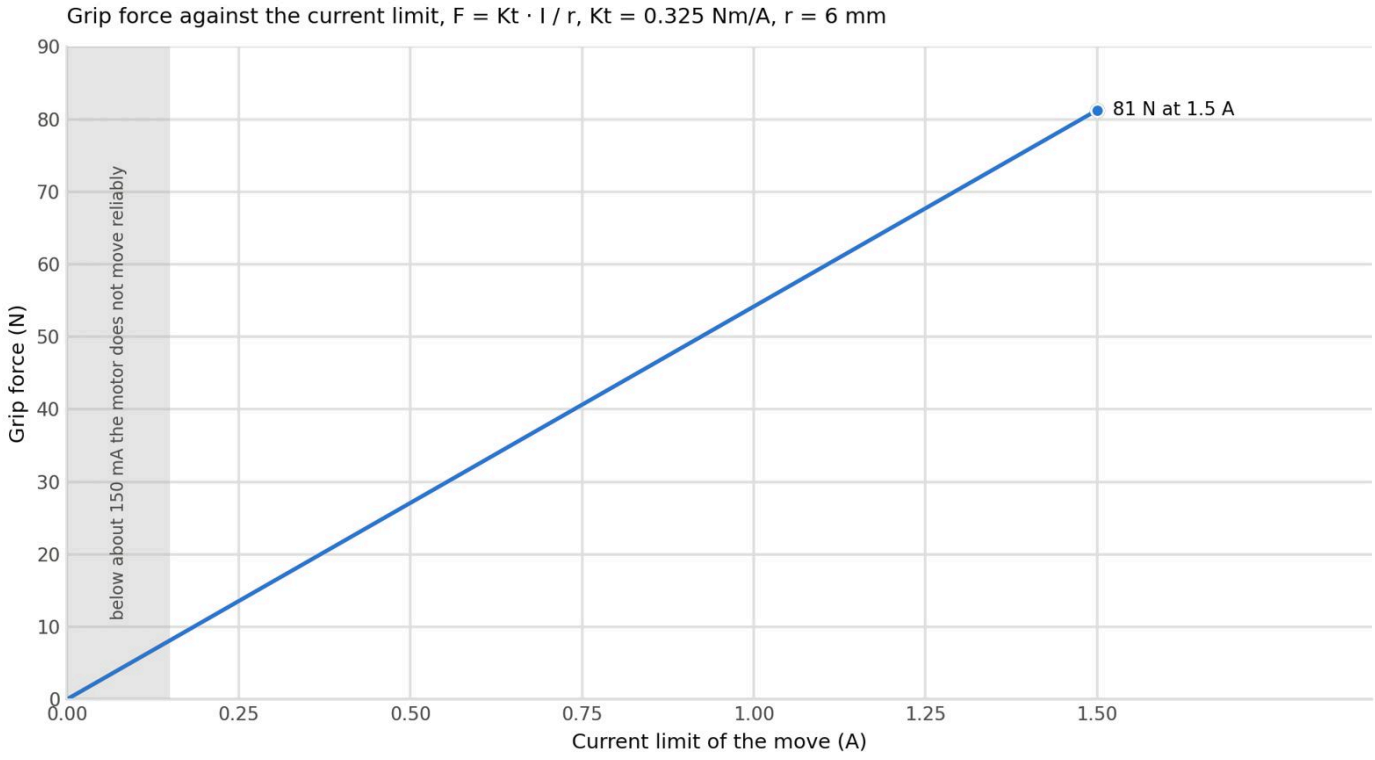


Figure 3.1: The grip force against the current limit, from the formula above, up to the 1.5 A a grip uses.

The curve is computed from the assumed torque constant, rather than a measured force calibration. `#Kt` over UART prints the driver's torque constant; that value must match its q-axis current convention. A datasheet current rating or supply current cannot be substituted directly. Measure your gripper with the fingers and contact position you will use; below about 150 mA the motor does not move reliably.

The reported force-gauge result is about 48 N/A, but its per-jaw versus combined-force convention is not specified. Keep it separate from the theoretical combined-force curve until the measurement setup is known.

The current limit caps motor torque; it does not guarantee that the commanded current or the estimated contact force is reached. Mechanism friction, finger flex and the contact position affect the actual force.

Heat

The firmware does not clamp the gripper's current. A hold at 300 mA ran for 24 hours (Performance tests); a hold at 1600 mA trips the 75 °C thermistor limit after four minutes. Use high currents for the grip and a low current for the hold.

For a symmetric grip, each jaw applies $F/2$. The torque balance is $T = F_{\text{left}} r + F_{\text{right}} r = Fr$; counting F once for each jaw would count the motor's torque twice. These are theoretical forces before losses in the mechanism. A compression gauge between the jaws reads the force at **one contact**; double that reading before comparing it with the combined force here. Record the current, fingers and contact position when measuring; the same setup is needed for a useful comparison.

Try it

The formulas of this page, with your numbers: the force from the current, the weight held by friction with the safety factor of the next section, and the speed from the speed byte.

Payload

The grip force is not the weight the gripper can lift. An object held by friction between the two jaws is held by $\mu(F_{\text{left}} + F_{\text{right}}) = \mu F$, with μ the friction coefficient between the fingers and the object. Dividing the available friction force by a chosen safety factor S gives the static estimate:

$$W = \frac{\mu F}{S}, \qquad m = \frac{W}{g}$$

Using an assumed μ = 0.3 and S = 2, the model gives about 12 N of supported weight (1.2 kg) at 80 N of combined grip force. This is a static estimate. Upward acceleration of 1 g doubles the load; acceleration and emergency-stop loads need a separate check. Friction can be lower than 0.3, so measure it with your fingers on your object. A form-fit grip needs its own load assessment.

Speed

The speed byte of a move is mapped by the firmware to the motor's speed in encoder ticks per second, linearly from 40 at speed 0 to 80000 at speed 255:

$$\text{ticks/s} = 40 + \frac{80000 - 40}{255}\,C$$

The motor's encoder has 16384 ticks per revolution, so the angular speed is $\omega = \text{ticks/s} / 16384 \times 2\pi$ and each jaw moves at

$$v = \omega\, r$$

Speed byte	Motor speed	Each jaw moves at	The jaws close at
0	40 ticks/s	0.1 mm/s	0.2 mm/s
20 (the power-up default)	6300 ticks/s, 0.4 rev/s	15 mm/s	29 mm/s
30	9400 ticks/s, 0.6 rev/s	22 mm/s	43 mm/s
128	40200 ticks/s, 2.5 rev/s	92 mm/s	185 mm/s
255	80000 ticks/s, 4.9 rev/s	184 mm/s	370 mm/s

Both jaws move, so the gap closes at twice the speed of one jaw. At speed 255 the 48 mm stroke takes well under a quarter of a second. The gripper calibration runs at speed 40 and 500 mA.

Position

The position byte is linear over the stroke: 0 is the fully open end stop found by the gripper calibration, 255 the fully closed one. One step of the byte is 48 mm divided by 255, 0.19 mm. The driver considers the position reached within 5 encoder ticks, about 0.01 mm.

Connector and pinout

The gripper has one connector, an M8 4-pin male on the side of the body, and one cable: 24 V and the CAN bus.

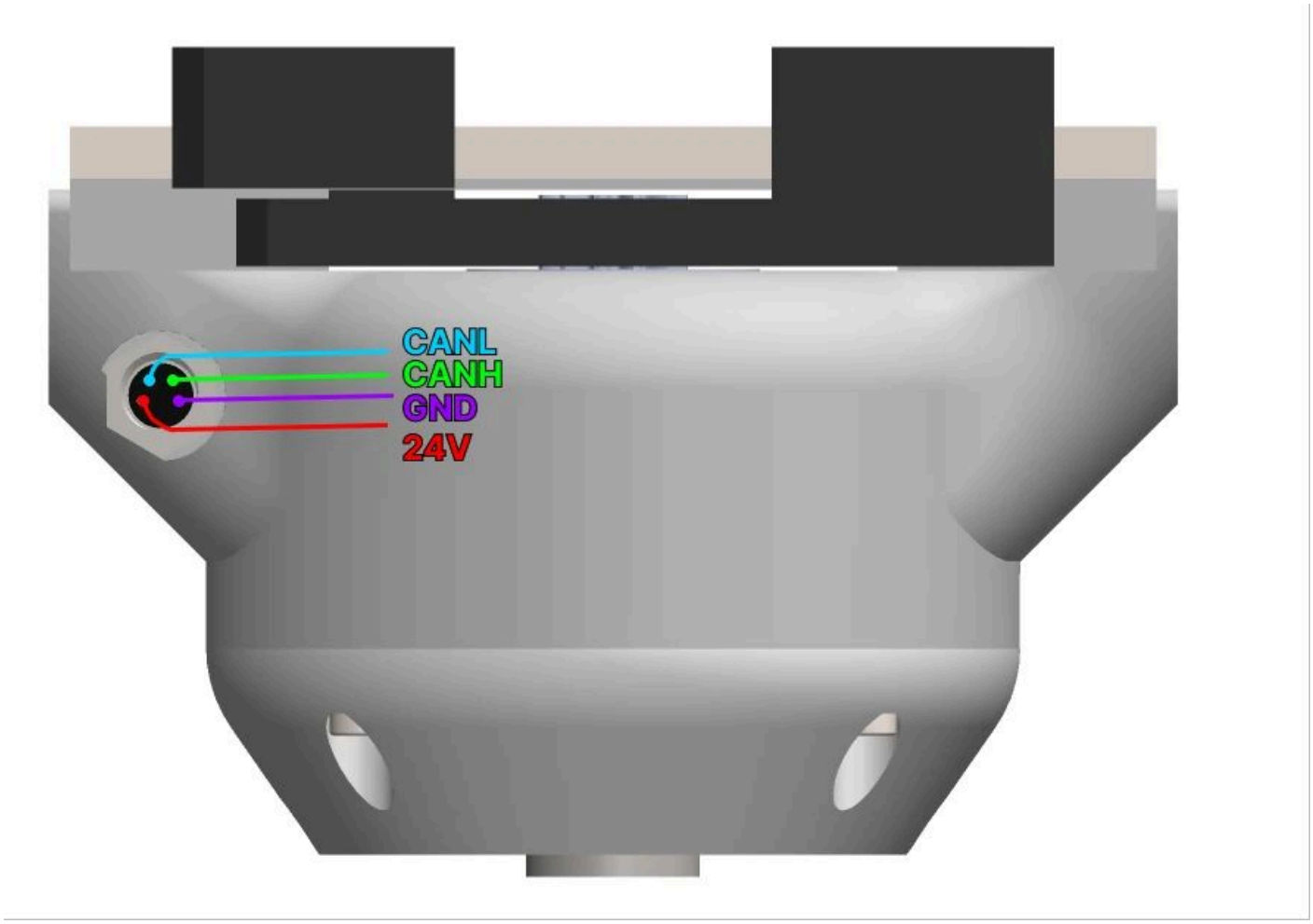


Figure 3.2: The M8 connector on the gripper body and its four signals: CANL, CANH, GND, 24V.

Pin numbers

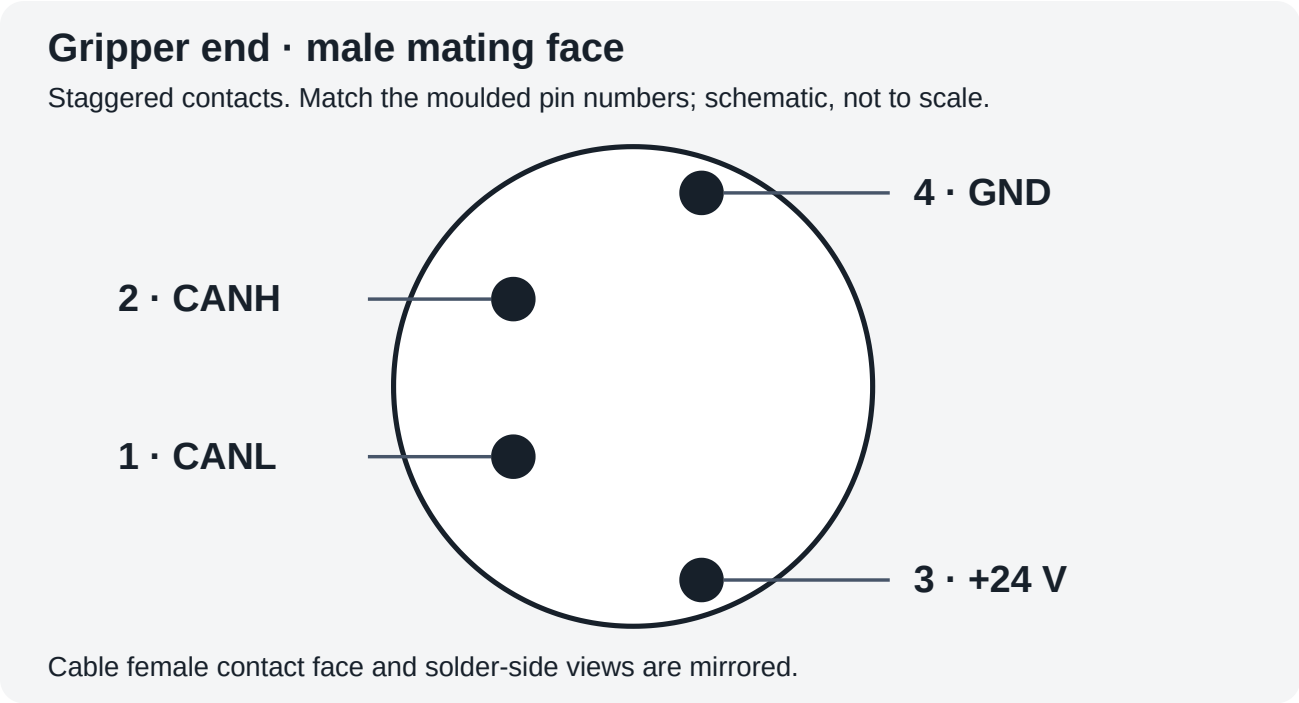


Figure 3.3: M8 male on the gripper: front mating face and numbered signals.

The drawing shows the connector's staggered contact layout. Look **into the male connector on the gripper**, with its pins visible. The female cable connector has the mirrored contact view. The solder side is also mirrored: use the numbers moulded into the connector, rather than matching wire colours or the installed housing's rotation.

Gripper pin	Signal	Connection
1	**CANL**	CAN low
2	**CANH**	CAN high, 1 Mbit/s with CANL
3	**+24 V**	positive motor supply
4	**GND**	power-supply return

These pin numbers combine the gripper's signal drawing above with the male contact numbering in the repository's cable drawing. They describe the **gripper end**. PAROL6 and PAR6 use different connections at the robot end: The cable gives both diagrams. Check continuity with the supply disconnected.

The wrong cable destroys the driver

A cable with 24 V on a CAN pin destroys the driver inside the gripper the moment it is powered. Use the cable made for your robot; check a self-made cable pin by pin against the pinout above and the wiring diagram before you plug it in.

Inside the body

The Spectral Micro driver's own connectors are inside, reachable when the coupler connector is unscrewed: its UART port (3.3 V, JST SH, 4 pins), used to configure a self-built gripper (Over UART), and its CAN termination switch (CAN termination). The driver's pinout is in the Spectral Micro documentation.

Dimensions and CAD

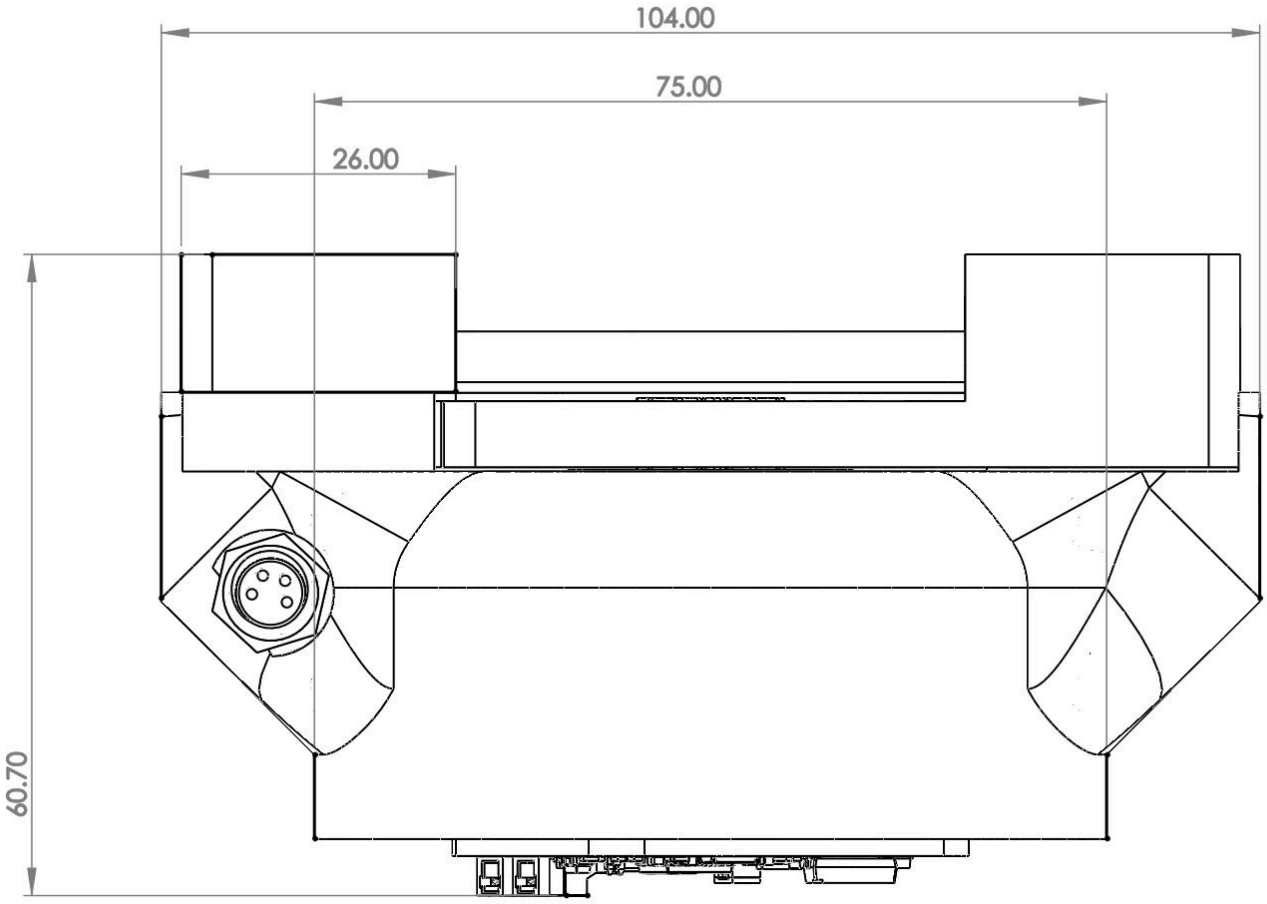


Figure 3.4: Front view, in millimetres, without a coupler connector: 104 mm over the jaw plates, 75 mm across the body, 26 mm jaw plates, 60.7 mm from the top of the jaw plates to the bottom of the driver board.

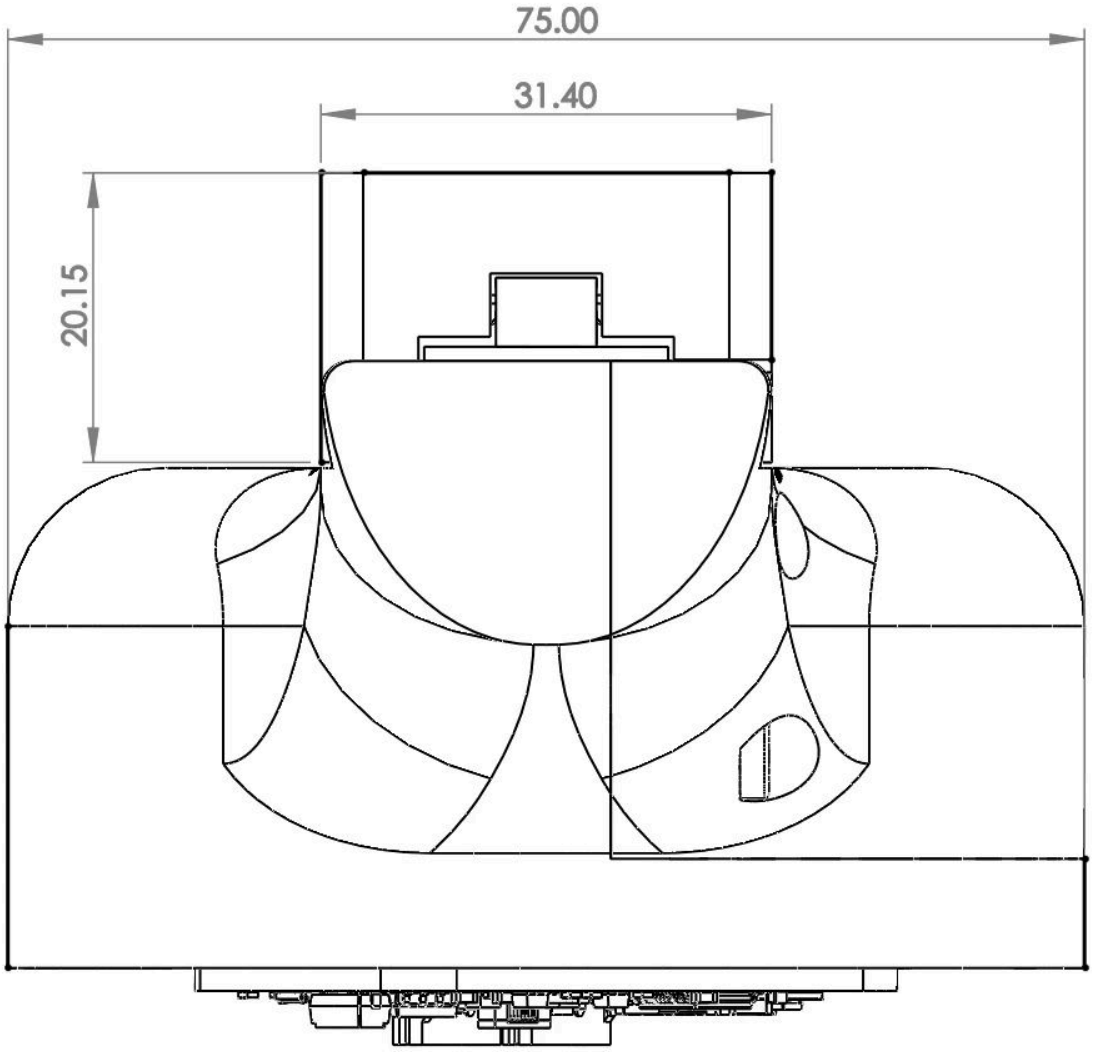


Figure 3.5: Side view, across the jaw travel: 75 mm across the body, 31.4 mm jaw plates.

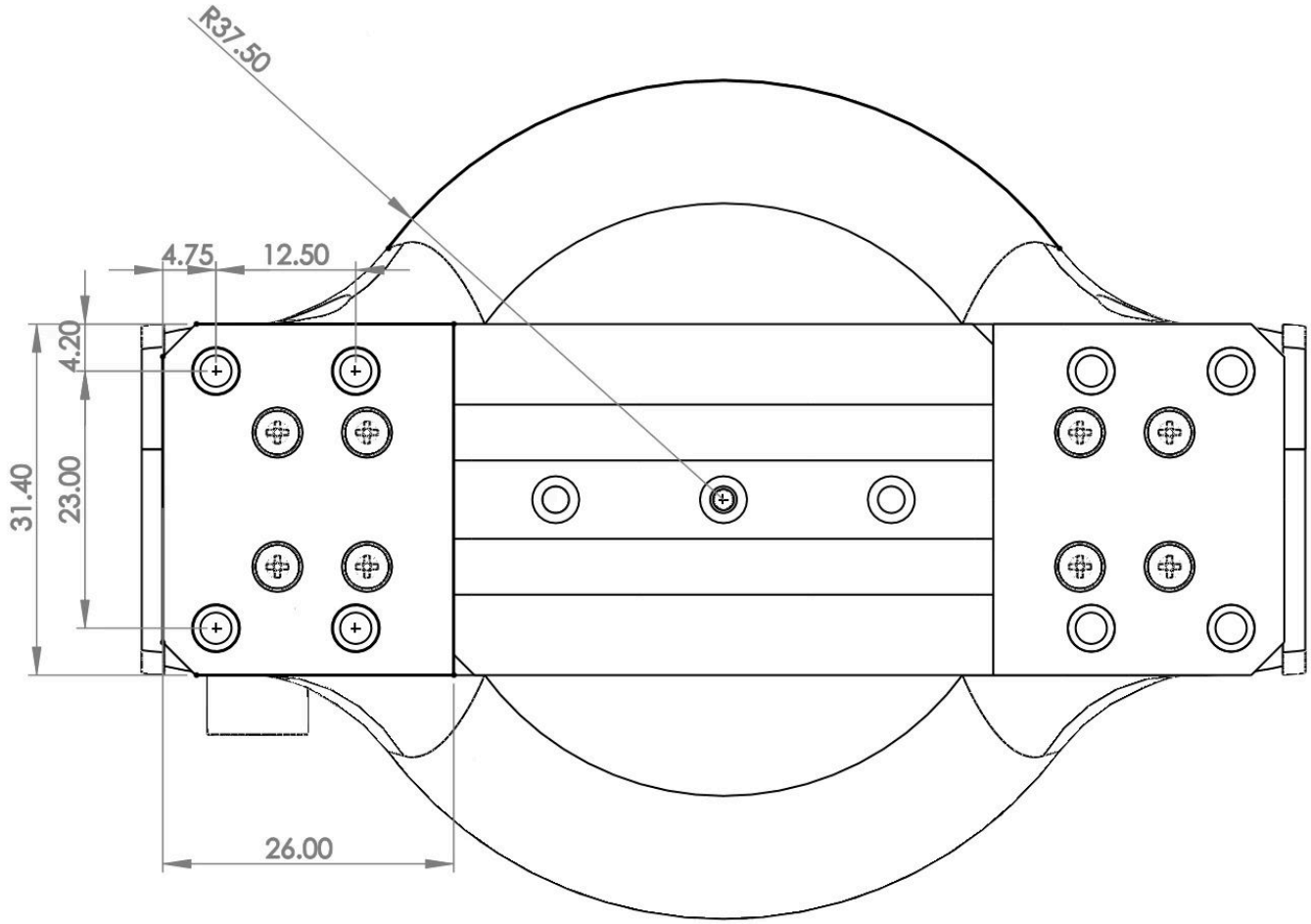


Figure 3.6: Top view, with the jaw plates and their hole pattern.

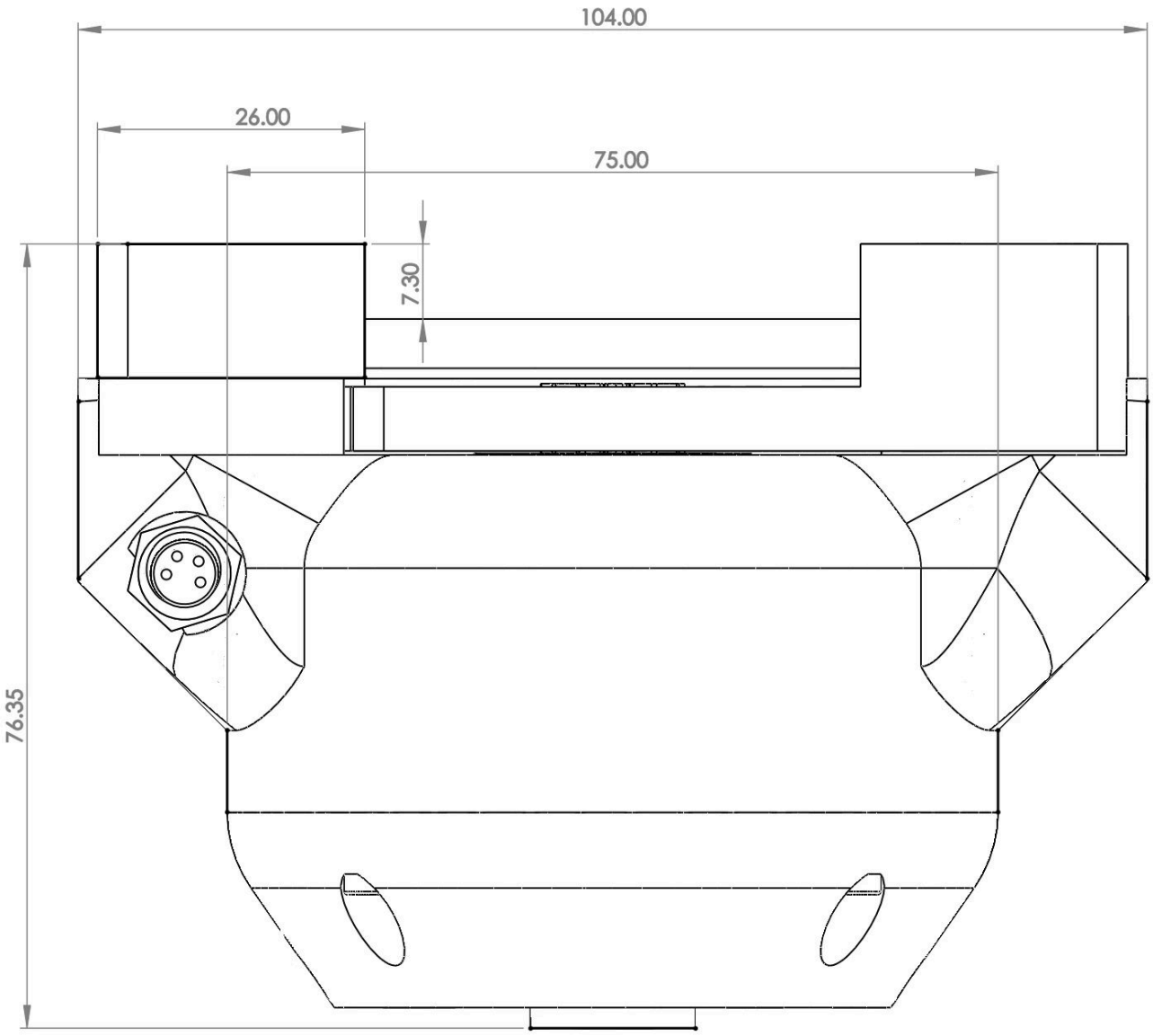


Figure 3.7: The gripper with the PAROL6 coupler connector: 76.35 mm from the top of the jaw plates to the end of the coupler.

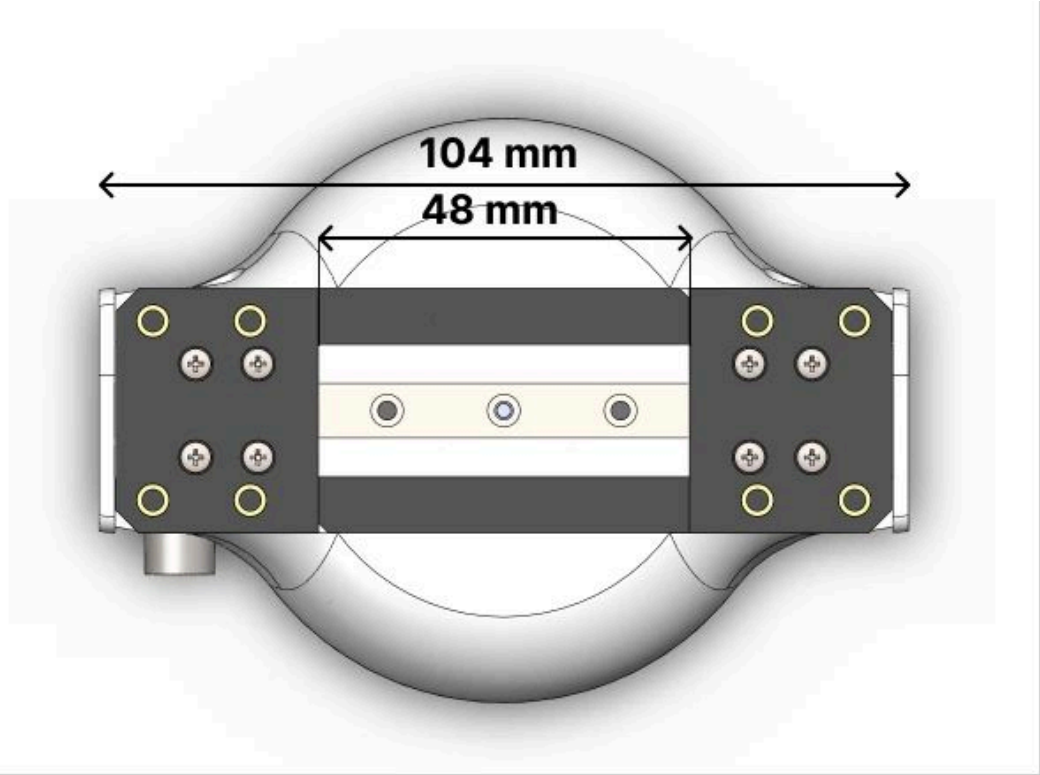


Figure 3.8: 48 mm of stroke between the jaw plates, 104 mm overall.

CAD

The STEP files of every part are in the product repository, the reference for anything that has to fit:

- STEPS: the body parts (`Gripper STEP files`), the whole gripper (`3D MODEL`), the jaw plate template, the pinch jaws, the generic coupler connector template and the UR5 adapter plate.
- Gripper STL parts: the printable parts of both hardware versions and the coupler connectors for PAROL6 and PAR6.
- gripper jaws: the fin-ray fingers and their jaw plate.

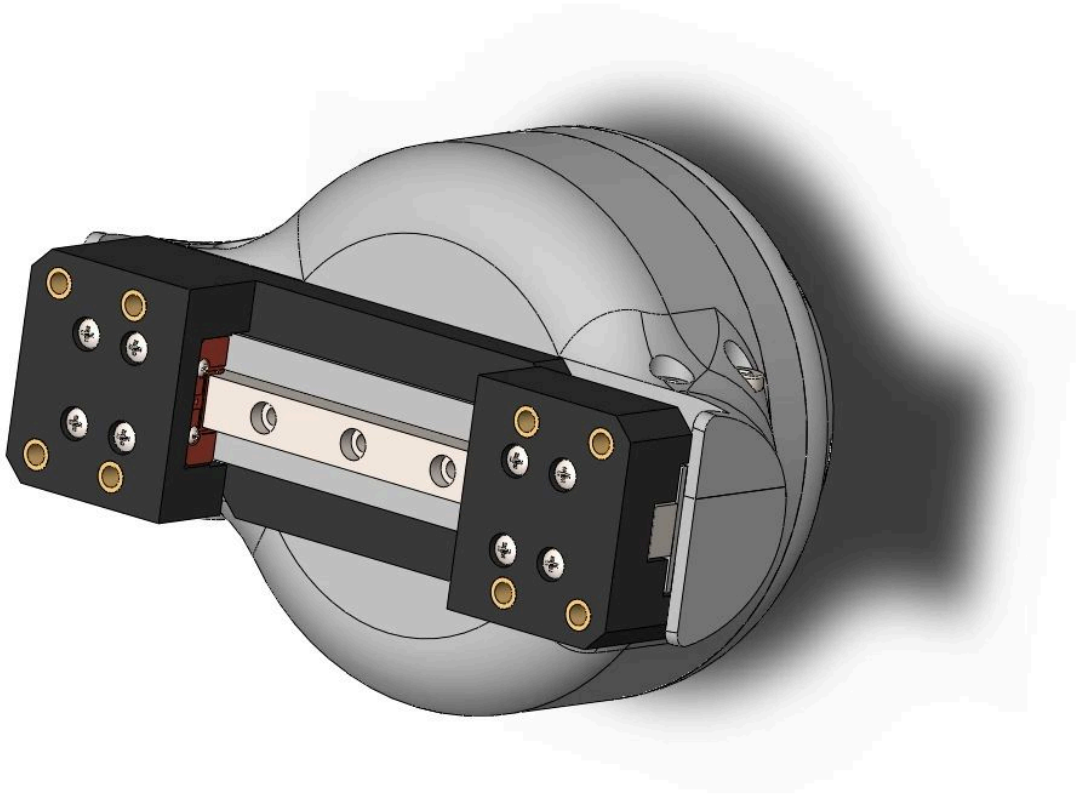


Figure 3.9: The STEP model of the gripper.

For simulation

The product repository's ssg48_description guide links the gripper's models, which are kept in robot_descriptions: standalone URDF and MJCF (MuJoCo) packages of the gripper with solid or fin-ray jaws, with their meshes, collision shapes and mass properties, and the PAROL6 and PAR6 models with the gripper mounted.

CAD and downloads lists everything that can be downloaded.

Data for robot integration

What a robot's software needs to carry the gripper: the frame it is mounted by, the mass, the centre of mass, the inertia and the tool centre point (TCP). The numbers come from the gripper's URDF and MJCF (MuJoCo) models in robot_descriptions, version 2.18.1, moved into the frame below; the two files give the same values. Both models are on CAD and downloads.

The frame

The origin is the centre of the mounting face, on the coupler side: the plane where the robot's flange meets the gripper, and the `flange` frame of the URDF and MJCF models. On PAROL6 it is the flange face, 50.5 mm from the joint 6 centre (Dimensions and kinematics).

- **Z** points out of the mounting face, through the body to the fingers: the approach direction. On the robot it is the flange's Z.
- **Y** is the jaw travel: the jaws move apart and together along Y, symmetrically about Z.
- **X** is square to both, across the 31.4 mm width of the jaw plates. The M8 connector on the body points along X.

Mass and inertia

The centre of mass in mm, the inertia in kg·m² about the centre of mass, with axes parallel to the frame, for fully open jaws.

Fingers	Mass, measured	Mass in the model	Centre of mass x, y, z	Ixx, Iyy, Izz	Ixy, Ixz, Iyz
Pinch jaws (the model's Solid jaws, `ssg48_solid`)	440 g	440.0 g	0.06, 0.14, 25.71	2.615e-4, 1.220e-4, 2.245e-4	-6.14e-6, -2.69e-6, 0.86e-6
Fin-ray fingers (`ssg48_fin_ray`)	440 g	440.0 g	0.06, 0.14, 25.71	2.615e-4, 1.220e-4, 2.245e-4	-6.14e-6, -2.69e-6, 0.86e-6

The jaws do not move the centre of mass: they travel symmetrically. Closing them brings each finger 23.5 mm towards the Z axis, which lowers Ixx and Izz by about 0.65e-4 kg·m² at the closed end.

Tool centre point

Fingers	x, y, z	Orientation	Opening	The point
Pinch jaws	0, 0, 91.4 mm	the frame's: roll, pitch and yaw 0	any	the middle between the finger tips, where the tips meet when the jaws close
Fin-ray fingers	0, 0, 114.8 mm	the frame's: roll, pitch and yaw 0	any, with the fingers unloaded	the middle between the finger tips

The point stays on the Z axis at every opening, because the jaws move symmetrically. The middle of the gripping faces is lower: at z = 75.9 mm for the pinch jaws (faces from 60.4 to 91.4 mm) and 87.6 mm for the fin-ray fingers (60.4 to 114.8 mm). The models have one `tcp` frame for both finger types, at 0, 0, 83.8 mm with the frame's orientation; the PAROL6 and PAR6 robot models move it (on PAROL6, a6 = 134.3 mm = 50.5 + 83.8 mm). robot_descriptions keeps it as a control frame, not a measured grasp centre: it is 7.6 mm short of the pinch tips and 31.0 mm short of the fin-ray tips.

How accurate the numbers are

The masses are measured: 440 g for the whole gripper, 39.3 g for each moving assembly (carriage, jaw and finger) and 361.4 g for the rest. The centre of mass and the inertia are not: robot_descriptions (2.18.1, MODEL_DETAILS.md) says that the gripper COM and central inertia remain unverified estimates. Both finger types share one set of mass properties in the model, so the two rows are the same. In both files each finger's centre of mass lies on the other side of the Z axis from the finger, and 8 to 13 mm closer to the mounting face than the finger meshes' centres: the open-jaw values above come out close, but the files' own inertia at other openings is not reliable. The geometry, and with it the frame and the finger tips, is the CAD's.

Where the numbers go

On PAROL6, `rbt.select_tool("SSG-48")` (with `variant_key="pinch"` for the pinch jaws) sets the Python API's own TCP for the gripper, and `set_tcp_offset` or `set_tcp_transform` moves it (Tools and the TCP); in a URDF, the frame is a fixed joint from the robot's flange, the mass table an `<inertial>` and the TCP a `tcp` link, as in the models' `ssg48_description` package.

4. Installation

Installation

Putting the gripper on a robot: the mounting, the cable that carries its power and its CAN bus, the jaws for your objects, and the CAN termination.

In this chapter

- Mounting on a robot
- The cable
- Jaws
- CAN termination

When all of it is done, continue with First setup.

Mounting on a robot

The gripper mounts by its coupler connector, the round part on the back of the body. The repository has three, for PAROL6, PAR6 and the UR5 flange, and a STEP template for your own.

Robot	Part	Where
PAROL6	`coupler_connector_PAROL6.STL`	Gripper STL parts
PAR6	`coupler_connector_PAR6.STL`	the same folder
UR5	`UR_coupler_gripper.STEP`	STEPS/UR5 adapter plate
your own	`coupler_connector_generic_STEP.STEP`, the template	STEPS

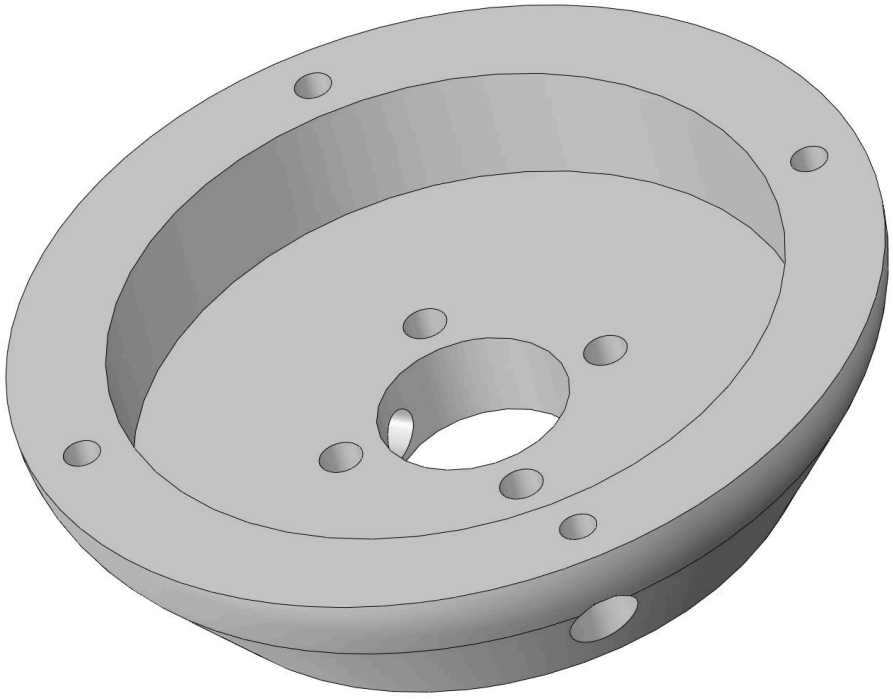


Figure 4.1: The generic coupler connector template: the bolt circle of the gripper on the outside, your flange's pattern in the middle.

PAROL6

The PAROL6 coupler connector carries a shaft coupler with an 8 mm bore and two M4 clamp screws for the robot's last joint, and the homing tab that joint 5 homes against.

1. Fit `coupler_connector_PAROL6` to the gripper, with the homing tab on the same side as the gripper's cable connector and an M3 × 12 screw in the marked hole, as the last pages of the assembly manual show.
2. Slide the gripper onto the J6 motor shaft.
3. Tighten the two screws of the shaft coupler, one on each side, with blue threadlocker.
4. Connect the PAROL6 gripper cable to the gripper and to the robot's gripper port (The cable).





The PAROL6 software needs to know the gripper is there, and the control board firmware needs the joint 5 homing offset for it: On PAROL6 and PAR6.

PAR6

`coupler_connector_PAR6` bolts to the PAR6 wrist; the PAR6 gripper cable, wired differently from the PAROL6 one, goes to the robot's tool connector.

Another robot

Adapt the template to your flange, or use the UR5 adapter plate as a starting point. Keep the gripper's connector reachable, and route the cable so that it cannot be pulled by the robot's motion.

On the bench

For testing, clamp the body, not the jaw plates or the rail, and leave the whole stroke free: calibration moves the jaws from one end stop to the other without any command from you.

The cable

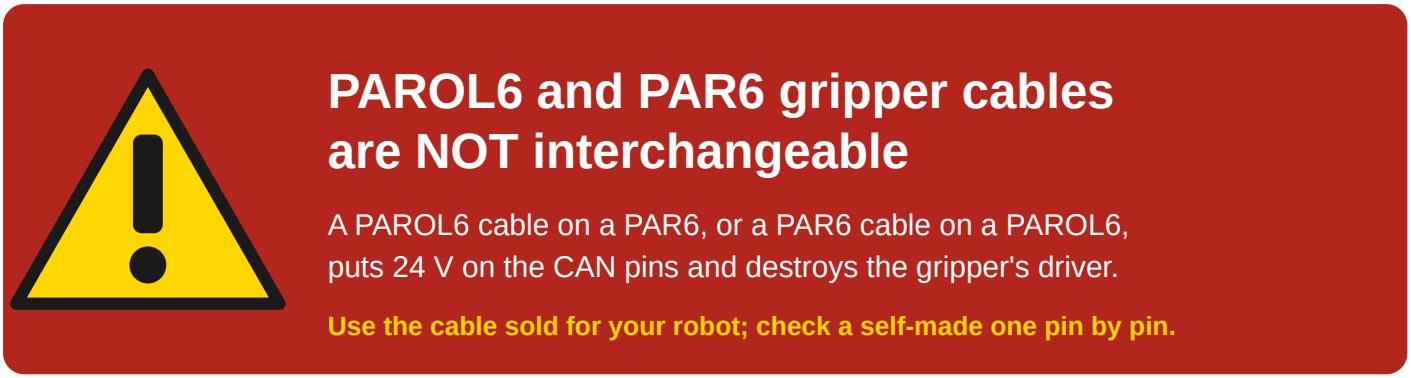


Figure 4.2: PAROL6 and PAR6 gripper cables are not interchangeable.

One cable with an M8 4-pin connector powers the gripper and carries its CAN bus. The robot end differs from robot to robot.

PAROL6 and PAR6 cables are different

The two robots put the four signals on different pins of their gripper port. The PAROL6 cable in a PAR6, or the other way round, puts 24 V on the CAN bus and destroys the gripper's driver. Use the cable sold for your robot, and check a self-made cable pin by pin.

The cables from the shop

Robot	Cable
PAROL6	PAROL6 gripper cable
PAR6	the PAR6 gripper cable, coming to the shop

Making your own

For a bench setup, another robot, or a replacement, the cable is four wires between an M8 4-pin female connector at the gripper and whatever the other end needs:

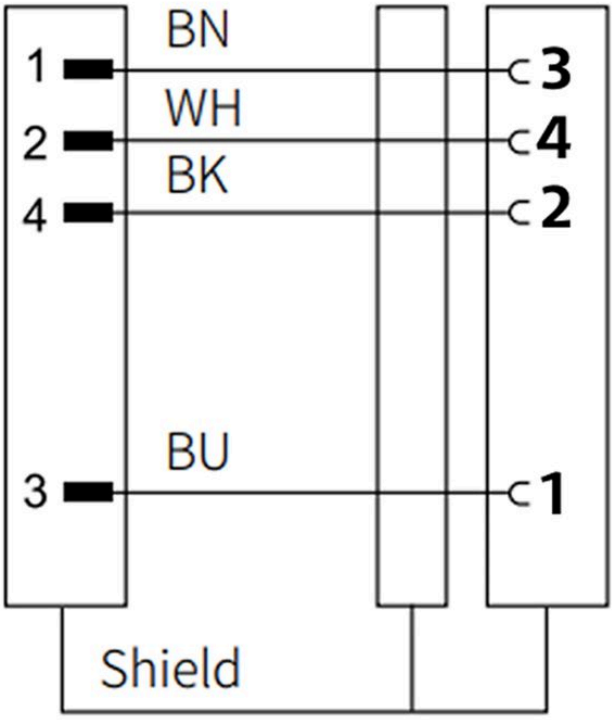
Part	Quantity	BOM
26 AWG 4-wire twisted-pair cable, 0.15 mm², shielded	about 0.4 m	cable
M8 4-pin female cable connector (the gripper has the male)	1	connector
M8 4-pin male cable connector, for a robot with an M8 port	1	the same listing





Wire it as the diagram for your robot shows. Pin 1 to 4 are the numbers moulded into the M8 connectors; the colours are those of the cable in the BOM.

4-pole



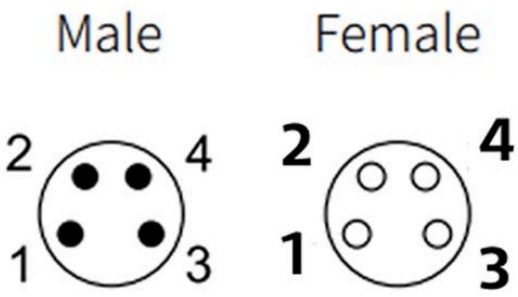


Figure 4.3: The PAROL6 cable: the pins are crossed between the two ends.

PAR6

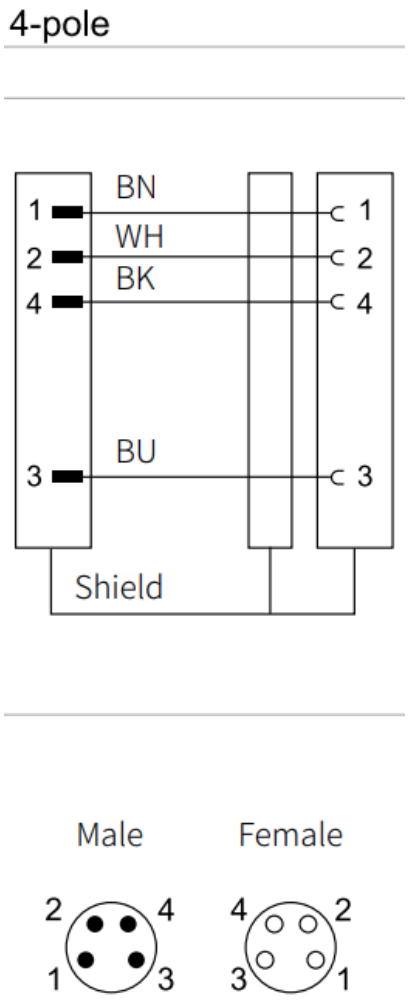


Figure 4.4: The PAR6 cable: pin to pin.

Bench or your own robot

Connect the gripper's 24V and GND to a 24 V supply and its CANH and CANL to your CAN interface; on a PC that is the CANvas adapter's CAN terminals. Connector and pinout shows which pin of the gripper's connector is which signal. Twist the CAN pair, keep the cable short at 1 Mbit/s, and terminate the bus at both ends (CAN termination).

Connect the shield at one end only. Check continuity and the absence of shorts between all four pins before the first power-up.

Handling

Plug and unplug with the supply off. Route the cable along the robot with enough slack for the wrist's full motion, and keep it away from the jaws' travel.

Jaws

The gripper's two jaw plates carry a four-hole pattern for fingers. The product repository has fingers to print and a template for your own.

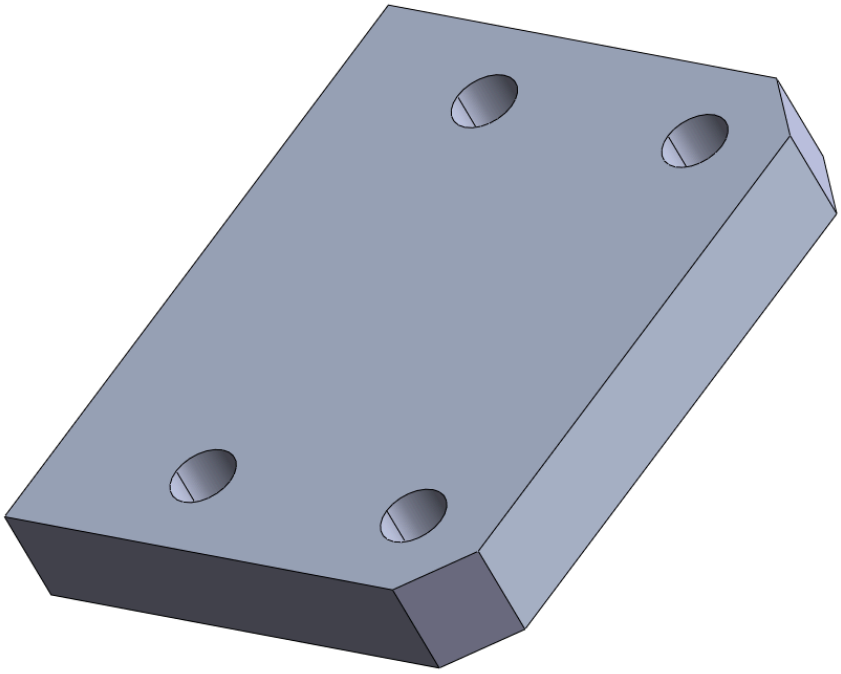


Figure 4.5: The jaw plate template, with the four mounting holes.

Fingers	Files	Use
Fin-ray fingers	`fin_ray_1`, `fin_ray_3`, `Finger_1` and `Jaw_plate_fin_ray` in gripper jaws	fingers that wrap around round and irregular objects; printed in a flexible filament
Pinch jaws	`solid_jaws_pinch` and `solid_jaws_pinch_foam` in STEPS	solid jaws for small and flat objects, with or without a foam pad
Your own	`Jaw_plate.STEP`, the template	keep the hole pattern, shape the rest

Limits

- Fingers at most **80 mm** tall and **60 mm** wide: longer fingers multiply the lever arm on the jaw plates and the rail, and foul the body at full opening.
- The grip force is set at the finger faces. A finger that bends takes some of it.

> **Custom fingers on a PAR6**
>
> On a PAR6, custom fingers must keep the size of the original fingers: no longer, no wider and no thicker. The PAR6 software works with the gripper's original dimensions (its collision model of the tool and the tool centre point), so a larger finger reaches where the robot does not expect it.

Calibrate after changing the fingers

The gripper calibration finds the end stops with the fingers fitted; fingers that meet earlier or later than the old ones change where 255 is. Calibrate again after a change, which happens at the next power-up anyway.

CAN termination

A CAN bus needs a 120 Ω resistor at each of its two ends. The gripper's driver has one, on a switch. The gripper is normally the last node on a robot's bus, where the switch must be **ON**.

The gripper is	Switch
the last node on a robot's bus, at the end of the arm	ON
alone on a bench bus with the CANvas adapter (which has its own termination)	ON
in the middle of a bus, with other nodes after it	off

To change it, unplug the supply, touch earthed metal (static electricity damages the driver) and unscrew the coupler connector: the switch is on the Spectral Micro driver inside the body, next to its CAN connectors; **ON** connects the resistor. The driver's connector page shows where it is on the board.

A bus with the right termination measures about 60 Ω between CANH and CANL with everything unpowered. A missing or doubled termination shows up as lost frames and a gripper that answers only sometimes.

Check

About 60 Ω between CANH and CANL, unpowered. About 120 Ω: one end is not terminated; about 40 Ω: three terminations are on.

5. First setup

First setup

For the complete bench procedure, use the one-page quick start.

From a mounted and cabled gripper to one that opens and closes on command. Motor GUI in the browser is the usual tool for it; every step also gives the Python call and the UART command.

1. Motor GUI: the browser, the CANvas adapter, connecting to the gripper, the gripper panel.
2. Calibrate, activate, move: the three things every session starts with, and the first grip.

A self-built gripper first needs its driver set up, which a bought one already has: Driver setup and first run in the build guide.

A gripper that has been through these pages is ready for Operation.

Motor GUI

Motor GUI is Source Robotics' tool for its drivers and for the grippers built on them: a page in the browser that reaches the hardware through the browser's serial port access, so there is nothing to install. With the CANvas adapter on the gripper's CAN bus it calibrates and activates the gripper, opens and closes it, shows its state and plots its position and current. It is the recommended tool for the first setup and for testing; everything it does is one CAN message, so what you learn here carries over to Python and the other interfaces.

> **The gripper GUI is legacy**
>
> The earlier SSG-gripper-GUI, a Python program with a Windows build, still works with both grippers but is not developed further. Use Motor GUI.

What you need

- A Chromium-based browser: Chrome, Edge, Brave or Opera. The serial port access Motor GUI relies on (Web Serial) is not in Firefox or Safari.
- The CANvas adapter in a USB port and on the gripper's CAN bus, with the bus terminated (CAN termination), and the gripper powered.
- Nothing on the gripper itself: a bought gripper is ready. The browser's port access and the adapter's installation are described on the Spectral site, Motor GUI.

Connecting

1. Open motorgui.com. In the connection bar choose **CAN (slcan)** and the bitrate 1 Mbit/s, then connect: the browser asks which serial port, pick the CANvas adapter.
2. Choose the gripper's node in the node picker: its CAN ID, 0 unless you changed it. Switch the bar from **Motor** to **Gripper**.
3. The gripper panel fills in as soon as the gripper's feedback arrives: calibrated or not, activated or not, the position, the current and the errors. While it says that it is waiting for feedback, check the node ID, the power and that the driver is in gripper mode.

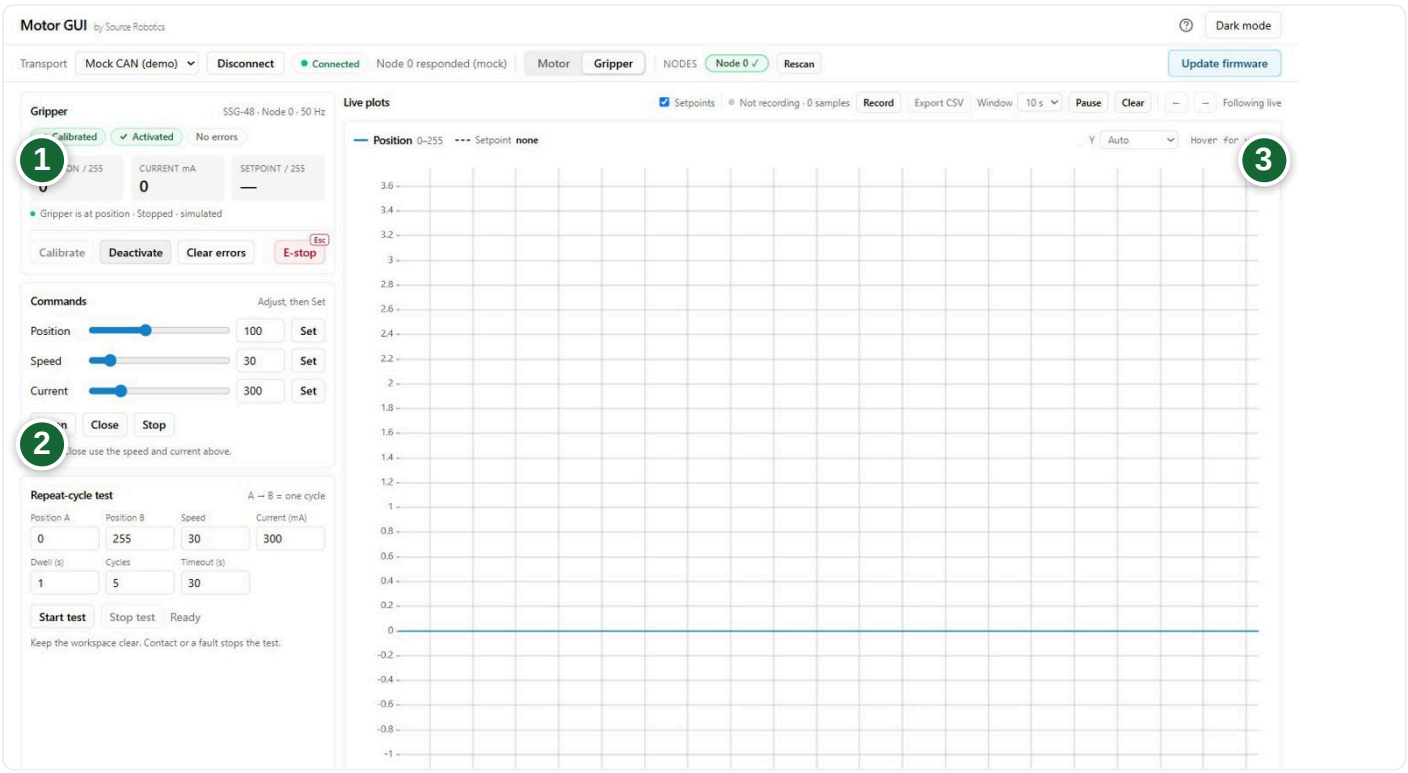
> **Check**
>
> The gripper panel shows the position and the current, and *Not calibrated* after a power-up. If it keeps waiting for feedback: The gripper does not answer.

The panel works with one gripper at a time; to work with another on the same bus, pick its node. The **Motor** side of the switch shows the same driver as a motor, which the gripper does not need.

Try it without a gripper

Open Motor GUI, choose **Mock CAN (demo)** and **Connect**, then switch to **Gripper**. The whole panel runs in the browser without an adapter or a gripper. The mock device identifies itself as SSG-48; MSG uses the same controls.

The gripper panel



1. State: calibrated, activated, and no errors. **2. Commands:** position, speed and current, each applied with its Set button. **3. Plots:** position and current feedback. Captured from the **SSG-48 mock demo**; MSG uses the same controls.

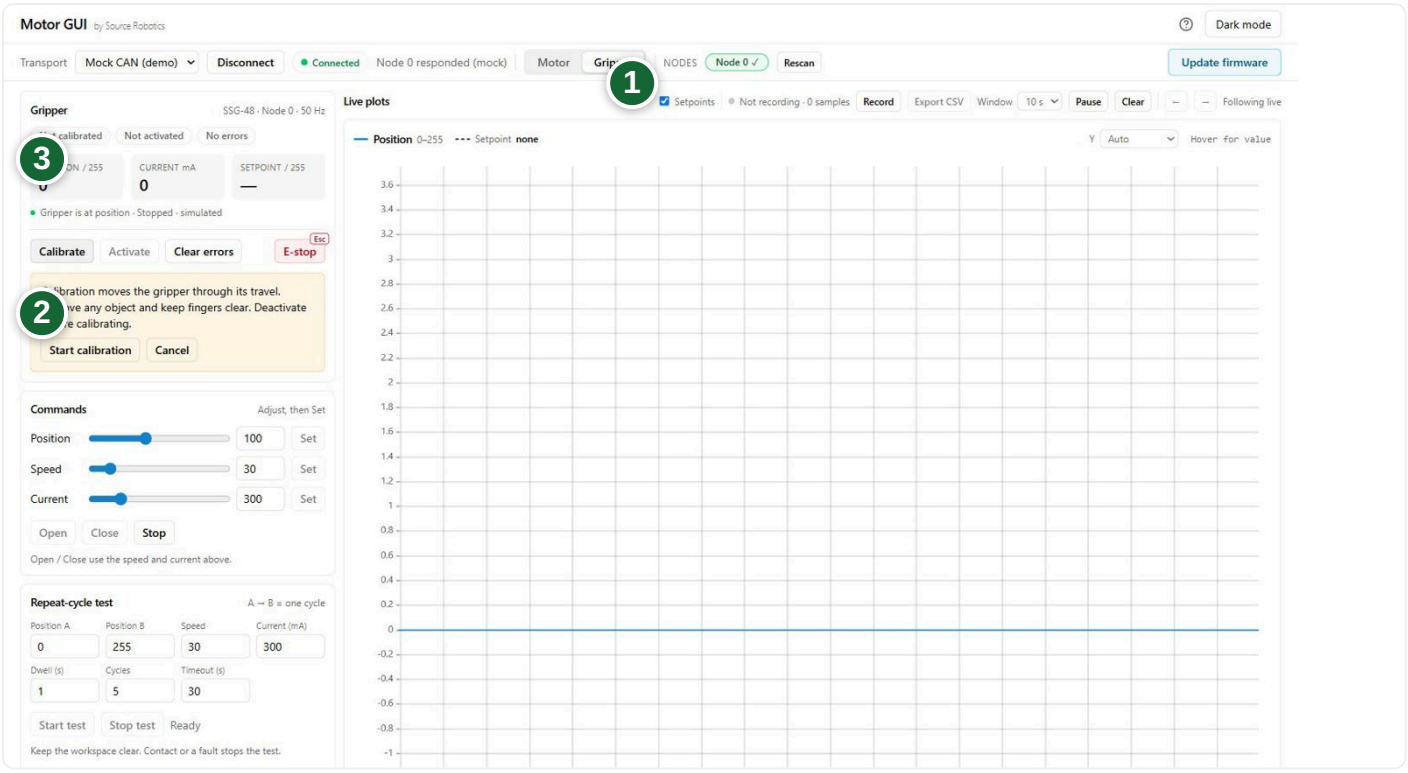
Control	What it does
Calibrate	runs the gripper calibration: the jaws travel the whole stroke to find the end stops. The panel asks you to confirm; clear any error, deactivate the gripper and take everything out of the jaws first. Press nothing else until the panel says *Calibrated*: a move or **Clear errors** during the calibration ends it
Activate, **Deactivate**	the activate bit: a calibrated gripper carries out moves only while it is activated

Control	What it does
Position, *Speed*, *Current*, each with **Set**	the move: position 0 (open) to 255 (closed), speed 0 to 255, current limit 0 to 1500 mA, which is the force. Each **Set** sends its value together with the other two as last sent
Open, **Close**	a move to position 0 or 255 with the set speed and current
Stop	ends the move where the jaws are
E-stop, or Esc	stops the gripper at once: the move command with the E-stop bit, after which the gripper reports the E-stop error until a command with the bit at 0 has been sent and the error cleared
Clear errors	clears the errors; an error whose cause remains comes straight back
Plots	position and current, live, with the requested position drawn in; the window and the scale are adjustable, and **Export CSV** saves a recorded run
Events	the feedback rate and the recent gripper events, with **Export events** for diagnostics
Cycle test	moves between two positions at a speed and a current, a number of times with a dwell: for endurance tests and demonstrations. Keep the tab visible while it runs

Presets for a self-built gripper

Connected over **UART** instead, through the serial adapter on the driver's UART port, Motor GUI's setup page has tuned presets for the grippers: *SSG-48*, and the three *MSG* steppers for the other gripper. A preset writes the motor's parameters, the loop gains, gripper mode, the thermistor and the CAN ID to the driver and saves them; its CAN ID is 6, the PAR6 gripper's, so change it before applying for another robot. Driver configuration says when to apply it, and lists the same settings as commands for a terminal.

Calibration prompt



1. Gripper view: select the intended node and Gripper panel. **2. Calibration prompt:** clear the stroke and confirm only when ready. **3. State:** wait for Calibrated before moving. This is the SSG-48 simulation; it is not a measurement of calibration time.

What next

Calibrate, activate, move is the first session with a connected gripper.

Calibrate, activate, move

Every session with the gripper starts the same way: clear what errors there are, calibrate, activate. Then it moves. This page does it once with Motor GUI, with Python and over UART; How the gripper works explains what happens inside.

> **Danger**
>
> The jaws move over the whole stroke during calibration, and close with the set force afterwards. Nothing in reach of them, fingers included.

1. Power up

At power-up the gripper reports *Not calibrated* and does not move. Its motor must be calibrated too, which a bought gripper is from the factory; a self-built gripper gets that during its assembly, Driver configuration.

> **Check**
>
> The gripper answers: Motor GUI's panel fills in, or a status poll returns a frame. If not: The gripper does not answer.

2. Clear errors

The driver powers up in an error state until its motor is calibrated and it has run through its checks; an E-stop, an over-temperature or a supply fault leave an error too. A gripper with an error does not move, and does not calibrate either: clear first.

Motor GUI

Clear errors, when the panel shows an error.

Python

```
gripper.Send_Clear_Error()
```

UART

```
#Clear
```

An error whose cause is still present comes straight back; after an E-stop, send a command with the E-stop bit at 0 before you clear. Errors and recovery says what each needs.

Check

No error is reported: Motor GUI shows none, and the temperature, timeout and E-stop bits of the answer are 0.

3. Calibrate

The gripper calibration finds the two end stops: the jaws open fully, pause, then close fully, at low force (500 mA) and low speed. Each end is recognised when the jaws have stood still against it for a little over half a second. It takes a few seconds; the result is not kept over a power cycle, so this runs at every power-up.

Motor GUI

Calibrate, and confirm. When the jaws stop, the panel shows *Calibrated*; press nothing else until then.

Python

```
gripper.Send_gripper_calibrate()
time.sleep(6)       # send nothing meanwhile; With Python waits for the calibrated bit instead
```

UART

```
#Gripcal
```

If the jaws do not move at all, or move to one end and stop: the gripper is in an error state (clear it first, see step 2), or the motor is not calibrated. If the routine runs for 30 seconds and gives up, an end stop was not found: something blocks the jaws short of the end, or the mechanism is too stiff for 500 mA.

Until the gripper reports *Calibrated*, send nothing but status polls: a move command or a clear during the calibration ends it, and the gripper stays uncalibrated.

Check

The jaws opened fully, then closed fully, and the gripper reports *Calibrated* (the calibrated bit is 1; `#Gripinfo` prints the end stops it found). If not: Calibration.

4. Activate

A calibrated gripper carries out a move only while it is activated.

Motor GUI

Activate. The panel shows *Activated*.

Python

The activate bit is part of every move command, its fourth argument; send it as 1. To activate without moving, send a command with the action bit, the fifth argument, at 0:

```
gripper.Send_gripper_data_pack(0, 30, 300, 1, 0, 0, 0)   # position, speed, current, activate =
1, action = 0, E-stop, release direction
```

UART

`#Grippos` activates by itself; `#Gripact 1` activates without moving.

Check

The gripper reports *Activated* and the jaws have not moved.

5. Move

A move is three numbers: the position (0 open, 255 closed), the speed (0 to 255: linear from 0.1 to 184 mm/s at each jaw) and the current limit in mA, which is the grip force (Force and speed). Start with speed 30, about 22 mm/s at each jaw, and 300 mA, about 16 N.

Motor GUI

Set *Speed* 30 and *Current* 300 with their **Set** buttons, then **Close** and **Open**, or set a *Position*: the jaws follow, the plots show the position and the current, and the state says whether the jaws reached the position or stopped on something.

Python

```
gripper.Send_gripper_data_pack(200, 30, 300, 1, 1, 0, 0)   # position, speed, current, activate, action, E-stop, release direction
time.sleep(2)
gripper.Send_gripper_data_pack(20, 30, 300, 1, 1, 0, 0)
```

UART

```
#Gripvel 30
#Gripcur 300
#Grippos 200
#Grippos 20
```

Put an object between the jaws and close on it: the jaws stop on it and hold it with 300 mA, and the state reads *object detected while closing*. Raise the current for a firmer grip, up to 1500 mA for a short grip, and the speed once the motion is what you want.

Check

The jaws reach each position and the state reads *at the position*; on an object they stop and the state reads *object detected while closing*. If not: It does not move.

What the gripper reports

Every move command is answered with the gripper's state, which Motor GUI shows in its panel and the library puts into attributes: the position, the current, whether it is calibrated and activated, whether a move is in progress, the object-detection result, and the three error bits. How the gripper works goes through them.

6. Operation

Operation

How a working gripper is used: what happens inside when it calibrates and moves, the interfaces it is driven through, and what to do when it reports an error.

In this chapter

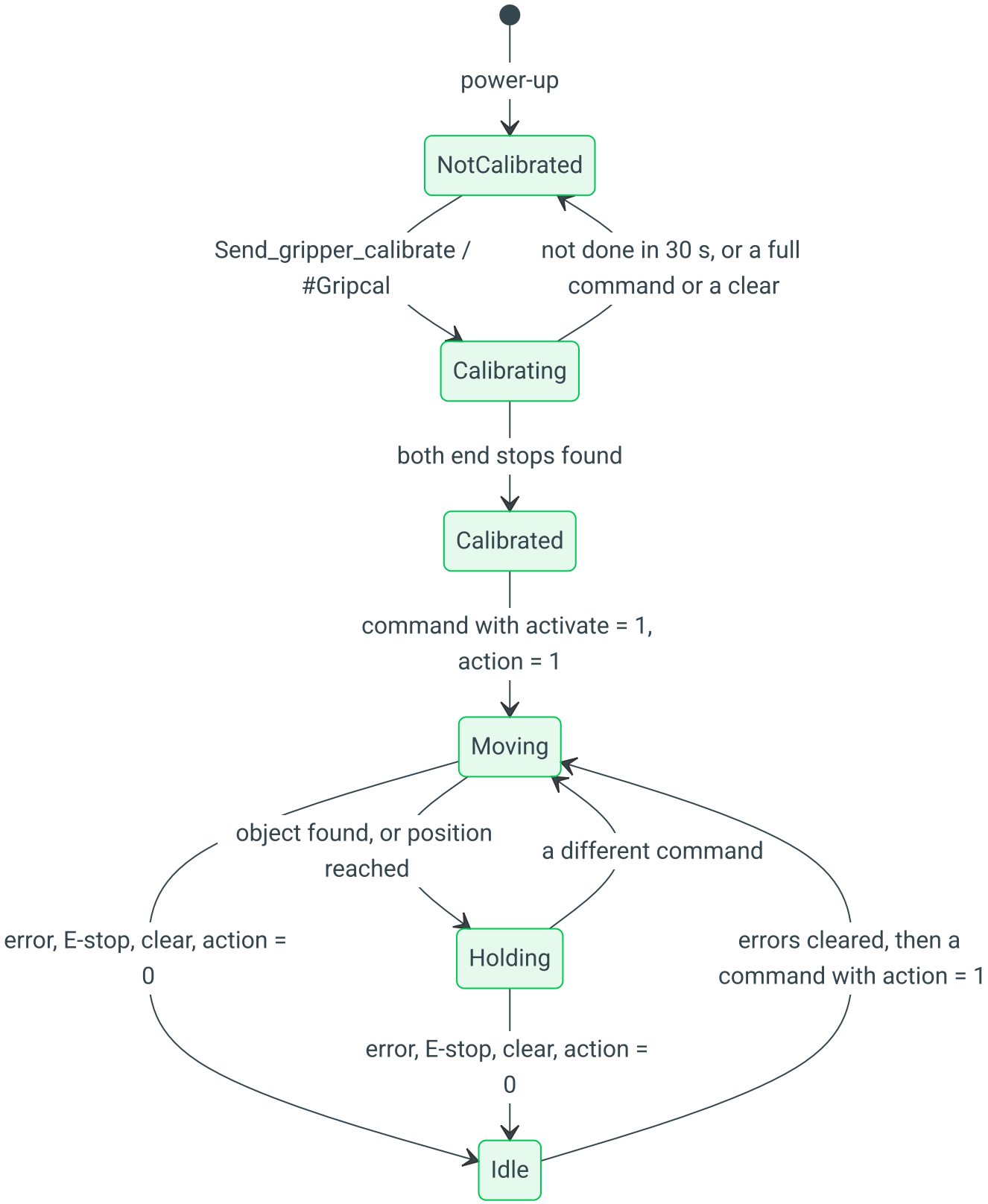
- How the gripper works
- Over CAN
- Over UART
- With Python
- With ROS 2
- On PAROL6 and PAR6
- Errors and recovery

The exact layout of every message and command is in the API reference.

How the gripper works

The gripper is a state machine in the driver's firmware, fed by one command and answering with one status. This page is what it does with them.

The states



Not calibrated

- After power-up the gripper does not know where its jaws are.
- It answers status requests and refuses moves.
- The gripper calibration is not kept over a power cycle.

Calibrating

- The jaws open at speed 40 and 500 mA until they have stood still against the end stop for a little over half a second.
- They pause briefly, then close the same way.
- The two end positions become 0 (open) and 255 (closed).
- If the whole routine has not found both end stops within 30 seconds of its start, it gives up and the gripper stays uncalibrated.
- While it runs, send only empty Send_gripper_data_pack frames, the status poll. A full gripper command or Send_Clear_Error ends the calibration where it is, and the gripper stays uncalibrated.
- Send_gripper_calibrate has no answer; the calibrated bit of the status turns 1 when the routine is done.

Moving

- A command with the activate and action bits set starts a move towards the position, at the speed.
- The current is limited to the commanded value.
- The driver runs its velocity loop towards the target and watches two things: the speed of the jaws and the motor current.

Holding: the move ends in one of two ways.

- **Position reached** (within 5 encoder ticks): the driver switches to position control and holds the jaws there. Object detection reads 3.
- **Object in the way**: the jaws stop before the target with the current at its limit. The driver keeps pushing with the limited current, which is the grip. Object detection reads 1 if the jaws were closing, 2 if they were opening.
- Either way, nothing moves until a different command arrives.

Idle

- An error, the E-stop bit, Send_Clear_Error, Send_Idle or `#Idle`, or a command with the action bit or the activate bit at 0 takes the gripper to idle.
- The motor is no longer driven and holds nothing beyond the mechanism's friction, so an object can slip.
- After an error, clear it; then the next command with the action bit set moves the gripper again.

The command

Send_gripper_data_pack carries, in 5 bytes:

Field	Range	Meaning
Position	0 to 255	0 fully open, 255 fully closed, linear over the stroke

Field	Range	Meaning
Speed	0 to 255	0.1 to 184 mm/s at each jaw, linear (40 to 80000 encoder ticks per second); see Force and speed
Current	mA	the current limit of the move, so the grip force; not clamped by the firmware
Activate	bit	1 activates the gripper; 0 deactivates it, and it stops
Action	bit	1 carries out the move; 0 stores the command and leaves the gripper idle
E-stop	bit	1 stops the gripper and raises the E-stop error, again at every cycle while the bit stays 1
Release direction	bit	read by the firmware but not used

The same command sent twice does nothing new: the driver compares every command with the previous one, and only a changed command resets the "at position" state and starts a move. So a host may repeat the command at any rate, which the CAN watchdog needs; to move the jaws from a held position to the same position again, change the speed or the current by one.

An empty Send_gripper_data_pack, 0 bytes, changes nothing and is answered like a full one: it is the status poll, and it costs the bus almost nothing. A host usually sends one full command per move and polls with empty ones.

Over UART, `#Grippos` is the same command: it sets the position, activates the gripper, sets the action bit and starts the move, with the speed and current last set by `#Gripvel` and `#Gripcur` (20 and 200 mA after power-up).

The answer

Every Send_gripper_data_pack is answered with Respond_Gripper_data_pack, 4 bytes:

Field	Meaning
Position	where the jaws are, 0 to 255
Current	the motor current, mA; positive while opening, negative while closing
Activated	the activate flag
Action status	the action bit of the last command, as received; it stays 1 while the gripper holds, and the calibration does not change it

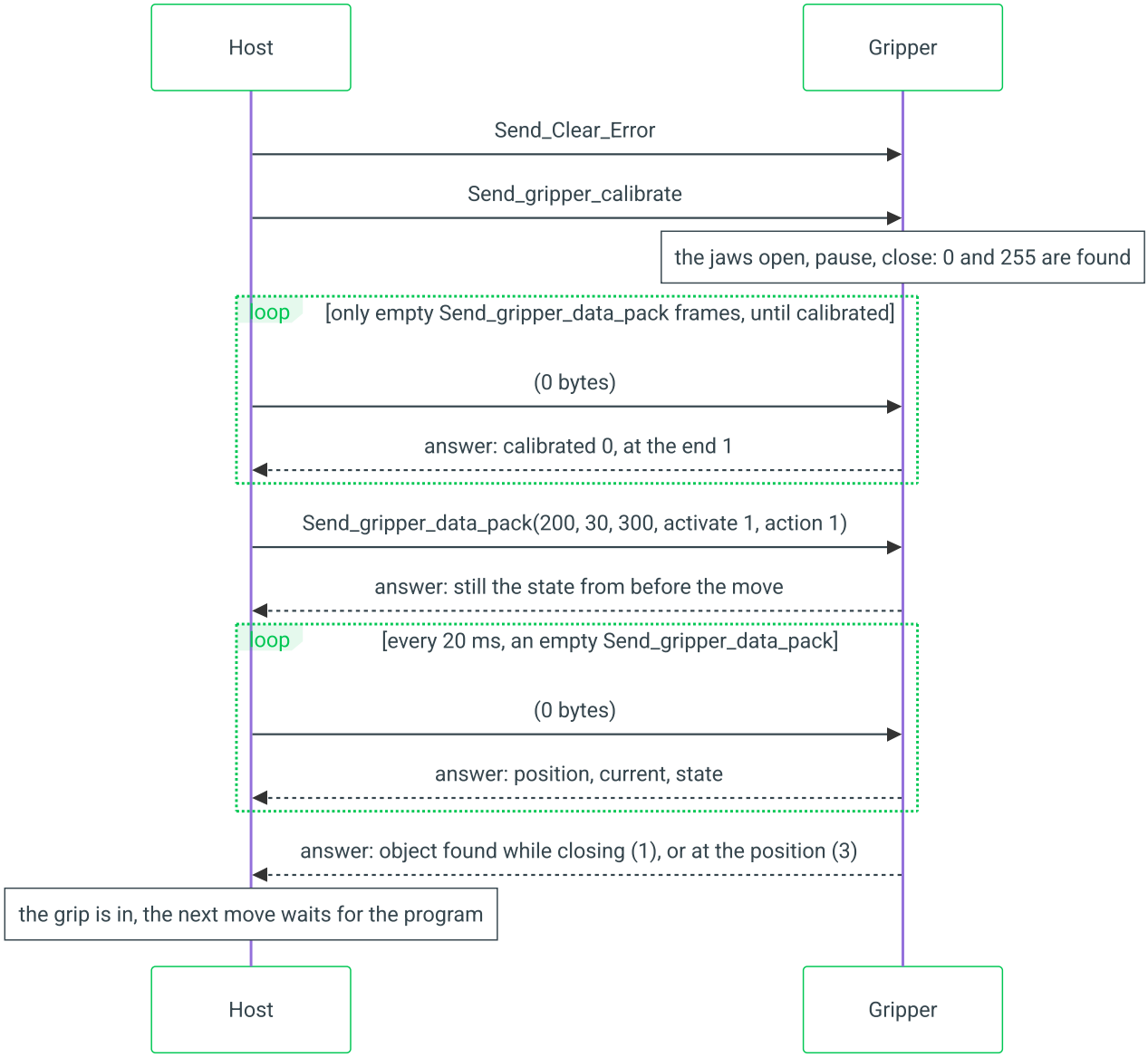
Field	Meaning
Object detection	0 moving, 1 object found while closing, 2 object found while opening, 3 at the position
Temperature error	the motor's thermistor is above the limit
Timeout error	the driver's CAN watchdog ran out
E-stop error	the E-stop bit was received
Calibrated	the gripper calibration is done

The answer to a full command is sent before the gripper acts on that command, so it still carries the state from before it; the answers to the polls that follow show the move. To follow a move, use the object detection: the action status only repeats the bit you sent.

The bits are listed in Errors and status bits, with the byte layout in the CAN reference.

A grip, message by message

What a host sends and what comes back, from power-up to a held object:



Object detection, exactly

While a move is in progress and the jaws are not yet at the target, the driver reads *moving* (0) as long as the motor turns faster than 500 encoder ticks per second, about 1 mm/s at the jaws. When it is slower than that and the motor current is above 80 % of the commanded limit, something stops them: *object found*, 1 or 2 by the sign of the current. Below 80 % with the jaws slow, it still reads *moving*: the object is soft and still being squeezed. At the very start of a move the jaws are slow too, and with the current already near its limit the state can briefly read 1 or 2; a host gives a move about 100 ms before it trusts the state. Once the target is reached, 3, and it stays 3 until a different command arrives.

The state is therefore a snapshot of each answer. A host that wants to know "did the grip succeed" sends the move, then polls until the state is 1, 2 or 3, with a timeout of its own; the ROS 2 drivers and the PAROL6 software do exactly that.

What the current limit does

The current limit of a move is the driver's Iq limit for the duration of the move and the hold. It bounds the torque, so the force on an object, and it also bounds how fast the jaws accelerate: a low limit with a high speed gives a gentle, slow-starting move. The temperature of the motor is the other bound: holding an object at a high current for minutes heats the windings, and at 75 °C the thermistor stops the gripper.

Timing

The driver runs its loops at 5 kHz; a move command takes effect at the next cycle, and the answer leaves within a millisecond. The CAN bus at 1 Mbit/s carries a command and its answer in well under half a millisecond, so a host can run the gripper at several hundred commands per second; 50 Hz is a good rate for a controller that polls the state.

Over CAN

CAN is the gripper's interface: the one cable carries it next to the power, and a robot's control board, a PC with the CANvas adapter or any microcontroller with a CAN transceiver drives the gripper with it. The gripper is a Spectral Micro driver on the bus, so the frames are the driver's; this page is the part a gripper needs.

The frame

CAN 2.0 data frames with 11-bit identifiers at 1 Mbit/s. The identifier carries the node, the command and the error state:

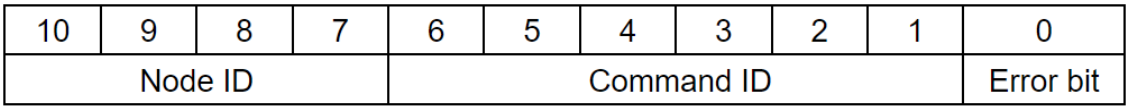


Figure 6.1: Bits 10 to 7 of the identifier are the node ID, bits 6 to 1 the command ID, bit 0 the error bit.

- **Node ID**, bits 10 to 7: which device. Every gripper ships as node 0; set another with Send_CAN_ID or `#CANID` and save it. Use 0 to 13: the driver's bootloader reserves 14 and 15, and a device on one of those cannot be updated over CAN.
- **Command ID**, bits 6 to 1: which message. The gripper command is 61, its answer 60, the calibration 62.
- **Error bit**, bit 0: in a frame from the gripper, 1 while any error is active. In a frame from the host it is ignored.

So the gripper command to node 0 travels on identifier `(0 << 7) | (61 << 1)` = 0x7A, and the answer comes back on `(60 << 1) | error` = 0x78 or 0x79; to node 3 the command is 0x1FA and the answer 0x1F8. Multi-byte values are big-endian; flags inside a byte are numbered from the most significant bit.

The messages a gripper needs

Message	ID	What for
Send_gripper_data_pack	61	the move command: position, speed, current, the activate, action and E-stop bits; 0 bytes to poll
Respond_Gripper_data_pack	60	the answer: position, current, the state and error bits

Message	ID	What for
Send_gripper_calibrate	62	the gripper calibration, after every power-up
Send_Clear_Error	1	clears the errors
Send_ESTOP, Send_Idle	0, 12	stop
Send_Watchdog	15	the gripper stops when the host stops talking
Send_CAN_ID, Send_Save_config	11, 13	another node ID, saved
Send_Respond_State_of_Errors, Send_Respond_Temperature	26, 23	the driver's error flags, the motor temperature

The CAN reference has the bytes of each; the driver's other messages, which a gripper does not need, are on the Spectral Micro site.

A control loop

1. At power-up: Send_Clear_Error, then Send_gripper_calibrate; poll with empty Send_gripper_data_pack frames until the calibrated bit is 1 (a few seconds; the routine gives up after 30 s). Send nothing else meanwhile.
2. For each grip: one Send_gripper_data_pack with the position, speed and current and the activate and action bits set. Its answer still carries the state from before it; ignore it. Then poll with empty frames until the object-detection field reads 1, 2 or 3, or your own timeout passes.
3. Keep polling at a steady rate (50 Hz is plenty) if you set the watchdog: every frame resets it, and the gripper stops by itself when your program stops.

The answer to every frame tells the whole state, so a host never has to ask twice.

Host checklist

- **Clear, then calibrate.** With an error active, the driver drops the calibration request; a clear sent during the calibration ends it.
- **During the calibration, only empty frames.** A full Send_gripper_data_pack ends the calibration as well, and the gripper stays uncalibrated. Wait for the calibrated bit; the routine gives up after 30 s.
- **The first answer is old.** The answer to a full command is sent before the gripper acts on it. Ignore it, and give the move about 100 ms before you trust the object detection: at the start of a move the jaws are slow, which can briefly read as an object found.

- **The action status repeats your bit.** It is the action bit you last sent and stays 1 while the gripper holds. Follow a move by its object detection.
- **An identical command does nothing.** To repeat a move to the position the gripper holds, change the speed or the current by one.
- **E-stop recovery:** first a command with the E-stop bit at 0 (and the action bit at 0), then Send_Clear_Error. While the last command carries the E-stop bit, a clear does not hold.
- **Watchdog:** with Send_Watchdog set, keep the polls coming. The gripper commands and the status requests reset it; Send_Clear_Error and Send_gripper_calibrate do not. After a timeout, fix the host, clear, and resume.
- **Current:** keep the commanded current at 1500 mA or less for a grip, and low for a long hold; the firmware does not clamp it (Force and speed).

From a PC

The CANvas adapter shows up as a serial port and speaks slcan; `python-can` opens it with `bustype='slcan'`, and the Python library wraps the messages. Any CAN adapter that `python-can` supports works the same way.

On a bus with other devices

A gripper on a robot shares the bus with the robot's joints or other grippers. Give each device its own node ID (connect one at a time, Send_CAN_ID, Send_Save_config), terminate both ends of the bus (CAN termination), and keep in mind that a lower node ID wins the bus arbitration.

Over UART

The driver inside the gripper has a serial port. It is not the gripper's everyday interface, which is CAN, but it is the way to configure the driver of a self-built gripper and the quickest way to look at what the driver reports when something is wrong.

Reaching the port

The UART connector of the driver (JST SH, 4 pins, 3.3 V) is inside the body, under the coupler connector. Unscrew the coupler connector, plug in the USB-to-serial adapter with the UART cable, and open a terminal at 256000 baud with CR+LF line endings; the gripper's 24 V must be on for anything but reading settings. The serial terminal on the Spectral Micro site walks through the terminal setup.

> **Danger**
>
> The port is 3.3 V. A 5 V adapter destroys the driver.

The gripper commands

Command	What it does
`#Gripper 1`	gripper mode on (set and saved on every gripper); `#Gripper` prints it
`#Gripcal`	the gripper calibration
`#Gripvel 30`	the speed of the next moves, 0 to 255
`#Gripcur 300`	the current limit of the next moves, mA
`#Grippos 200`	move to the position, 0 open to 255 closed; activates the gripper and starts the move
`#Gripact 1`	the activate flag without moving
`#Gripstop 1`	the E-stop bit
`#Gripinfo`	the gripper's state: calibrated, activated, the setpoints, the object detection, the end stops

Only `#Grippos` starts a move. A session:

```
#Clear
#Gripcal
#Gripinfo
#Gripvel 30
#Gripcur 300
#Grippos 200
#Grippos 20
```

Wait until `#Gripinfo` reports the gripper calibrated before the first `#Grippos`: a `#Grippos` or a `#Clear` during the calibration ends it. After `#Gripstop 1`, send `#Gripstop 0` before `#Clear`, or the E-stop error comes straight back.

The UART reference has these and the driver commands a gripper needs; the driver's whole command set is on the Spectral Micro site.

With Python

The `Spectral-BLDC` library wraps the driver's CAN protocol, gripper messages included: one method per message, and the gripper's answers unpacked into attributes. The ROS 2 driver and the legacy gripper GUI are built on it.

Installing

In a virtual environment:

Windows

```
py -m venv .venv
.venv\Scripts\activate
pip install Spectral-BLDC
```

The CANvas adapter is `COM` and a number (Device Manager, **Ports (COM & LPT)**): `channel='COM20'`.

Linux

```
python3 -m venv .venv
source .venv/bin/activate
pip install Spectral-BLDC
sudo usermod -aG dialout $USER     # access to the serial port; log out and in again once
```

The CANvas adapter is `/dev/ttyACM0` (`ls /dev/ttyACM*` lists it): `channel='/dev/ttyACM0'`.

macOS

```
python3 -m venv .venv
source .venv/bin/activate
pip install Spectral-BLDC
```

The CANvas adapter is `/dev/cu.usbmodem` and a number (`ls /dev/cu.usbmodem*` lists it): `channel='/dev/cu.usbmodem1101'`, with your number.

Check

`python -c "import Spectral_BLDC"` returns without an error message.

The source and the examples are in Spectral-BLDC-Python.

The first program

```
import Spectral_BLDC as Spectral
import time

# the adapter: a CANvas on COM20 (Windows), /dev/ttyACM0 (Linux) or /dev/cu.usbmodem... (macOS),
1 Mbit/s
bus = Spectral.CanCommunication(bustype='slcan', channel='COM20', bitrate=1000000)
gripper = Spectral.SpectralCAN(node_id=0, communication=bus)

gripper.Send_Clear_Error()                      # clear first: with an error active, the
calibration does not run
gripper.Send_gripper_calibrate()                # the jaws open fully, then close fully
for _ in range(150):                            # wait for the calibrated bit, up to 30 s
    time.sleep(0.2)
    gripper.Send_gripper_data_pack()            # only empty frames now: a move or a clear ends
the calibration
    message, ids = bus.receive_can_messages(timeout=0.05)
    if message is not None:
        gripper.UnpackData(message, ids)
        if gripper.gripper_calibrated:
            break

for position in (200, 20, 200, 20):
    # position 0..255, speed 0..255, current mA, activate, action, E-stop, release direction
    gripper.Send_gripper_data_pack(position, 30, 300, 1, 1, 0, 0)
    bus.receive_can_messages(timeout=0.05)      # its answer still carries the state from before
the move
    time.sleep(0.1)                             # give the move time to start
    for _ in range(40):                         # poll the state for up to 2 s
        gripper.Send_gripper_data_pack()        # an empty frame: status only
        message, ids = bus.receive_can_messages(timeout=0.05)
        if message is not None:
            gripper.UnpackData(message, ids)
            print(gripper.gripper_position, gripper.gripper_current,
gripper.gripper_object_detection)
            if gripper.gripper_object_detection in (1, 2, 3):   # object found, or at the
position
                break
```

The answer to a move command is sent before the gripper acts on it, so it still carries the state from before: after a finished move, object detection 3. The example reads that answer and drops it. At the very start of a move the jaws are still slow, and with the current near its limit the object detection can briefly read 1 or 2; the pause of 0.1 s covers that. While the gripper calibration runs, send only empty frames: a full command or Send_Clear_Error ends the calibration, and the gripper stays uncalibrated.

`Send_gripper_data_pack` with all seven arguments sends the move; without arguments it sends the empty status poll. `receive_can_messages` returns the raw `python-can` message and the node ID, command ID and error bit from the identifier; `UnpackData` fills the gripper attributes:

Attribute	Meaning
`gripper_position`	0 open to 255 closed
`gripper_current`	the motor current in mA
`gripper_activated`, `gripper_action_status`	the activate flag; the action bit of the last command, which stays 1 while the gripper holds
`gripper_object_detection`	0 moving, 1 object found while closing, 2 object found while opening, 3 at the position
`gripper_temperature_error`, `gripper_timeout_error`, `gripper_estop_error`	the three error bits
`gripper_calibrated`	the gripper calibration is done

Object detection in library 1.20

Library version 1.20 assembles `gripper_object_detection` with its two bits swapped relative to the firmware: it reports 1 where the gripper sends 2 (object found while opening) and 2 where it sends 1 (object found while closing); 0 and 3 are unaffected. Until the library is fixed, swap the two values in your program, or test for "1 or 2" as this example does.

Stopping

```
gripper.Send_gripper_data_pack(20, 30, 300, 1, 1, 1, 0)   # the E-stop bit: stops, raises the E-stop error
gripper.Send_gripper_data_pack(20, 30, 300, 1, 0, 0, 0)   # to recover: first the bit back at 0, action 0
gripper.Send_Clear_Error()                                 # then clear; while the bit is 1, the error comes back
gripper.Send_Watchdog(500)                                 # from now on the gripper stops 500 ms after the last frame
```

With the watchdog set, keep the status poll running at a steady rate; a program that crashes releases the grip within the watchdog time.

Several grippers

One `CanCommunication` per bus, one `SpectralCAN` per gripper with its node ID; every answer carries the node ID in `ids.node_id`, so route it to the right object. The teleoperation guide couples two grippers this way.

The complete map of calls to messages is in the Python library reference.

With ROS 2

Two ROS 2 packages drive the gripper; both are built on the Python library and a `python-can` adapter.

The SSG-48 package

ssg48_adaptive_electric_gripper_ros2, written for this gripper by Lasse Stubbe Ebbesen, with example nodes. Its README has the installation, the topics and the parameters.

The MSG gripper's package

msg_gripper_ros2, a ROS 2 Humble driver written for the MSG gripper by the UCLA Biomechatronics Lab and Ben Forbes. The SSG-48 speaks the same protocol, so it works with it as well:

Direction	Topic or action	Type
Command	`/gripper/command_joint_states`	`sensor_msgs/JointState`, the position in the gripper's own units, 0 to 255
Feedback	`/gripper/feedback_joint_states`	`sensor_msgs/JointState`, the position in the same units and the effort in mA
Action	`/gripper_command`	`control_msgs/GripperCommand`

Ubuntu 22.04 with ROS 2 Humble, Python 3.10 or newer, and a USB-to-CAN adapter that speaks slcan, such as the CANvas:

```
pip install Spectral_BLDC python-can
cd ~/ros2_ws/src
git clone https://github.com/Source-Robotics/source_robotics_msg_gripper_ros2.git
cd ~/ros2_ws
colcon build --packages-select msg_gripper_ros2
source install/setup.bash
ros2 launch msg_gripper_ros2 msg_gripper_launch.py
```

The only setting that depends on your machine is the adapter's port, `channel` in `config/default.yaml`: `/dev/ttyACM0` or `/dev/ttyUSB0` on Linux, `/dev/tty.usbmodem*` on macOS, `COM3` on Windows. On Linux, add yourself to the `dialout` group to open the port without `sudo`.

The description files

The gripper's standalone URDF and MJCF (MuJoCo) packages, with solid or fin-ray jaws, come from robot_descriptions; the product repository's ssg48_description guide links the downloads and says how to open them. Its display launch file is for ROS 1; in ROS 2, load the URDF with a launch file of your own.

On PAROL6 and PAR6

The gripper is made for the Source Robotics arms. Both drive it over their own CAN bus; their software calibrates it, moves it and reads its state, so a program for the robot does not talk CAN itself. The tool centre point, the mass, the centre of mass and the inertia for the robot's tool settings, on these robots or any other, are on Data for robot integration.

PAROL6

Mounting and connecting

1. Fit `coupler_connector_PAROL6` to the gripper (Mounting on a robot). It carries a shaft coupler with an 8 mm bore and two M4 clamp screws for the joint 6 shaft, and the homing tab that joint 5 homes against: the tab goes on the same side as the gripper's cable connector, with an M3 × 12 screw in the marked hole, as the last pages of the assembly manual show.
2. Slide the gripper onto the joint 6 shaft, orient it, and tighten the two clamp screws, one on each side, with blue threadlocker.
3. Connect the PAROL6 gripper cable between the gripper and the M8 connector at the forearm. Only this cable: the PAR6 cable is wired differently and destroys the gripper (The cable).
4. Keep the gripper's node ID at 0 and switch its termination on; it is the last node on the robot's bus.

The joint 5 homing offset

Joint 5 homes against the tool, and the SSG-48's coupler sits differently from the pneumatic gripper holder that the control board firmware's default offset (8900) is for. Set `j5_homing_offset` to `8035` in `main.cpp` of the control board firmware and upload it, as the PAROL6 documentation describes for the SSG-48. After the first homing, check joint 5 with the pin through its witness marks, and master the robot if needed.

Driving it

The gripper calibrates after every power-up and does not move before that. Clear the gripper's errors first, then calibrate, then wait until the calibration is done, a few seconds: a gripper move or a clear sent while it runs ends the calibration.

Commander software

The **Gripper** tab: after homing the robot, press **Clear gripper error**, then **Calibrate gripper**, and wait until the status says calibrated. Then set the position, speed and current with the sliders and press **Move GoTo**. In RBTscript: `Gripper_cal()`, `Delay(6)`, then `Gripper(position, speed, current)`, with the current in mA.

Python API

```
import time

rbt.select_tool("SSG-48")                              # the parol6 package; pinch fingers:
variant_key="pinch"
rbt.tool.calibrate()                                   # with the robot at rest
time.sleep(6)                                          # the calibration takes a few seconds
rbt.tool.set_position(0.8, speed=0.5, current=0.4)     # fractions; current 0.4 is 580 mA
print(rbt.tool.status())
```

The API allows the calibration 2 s before it sends the next gripper action, which can be too short when the jaws start far from the open end; the pause above waits for the calibration to finish.

The Python API's `current` is a fraction of the range 100 to 1300 mA: the gripper gets 100 + 1200 × `current` mA, so 1.0 is 1300 mA, within the 1500 mA a grip may use. Use a low value for a long hold (Force and speed). `position` and `speed` are fractions of the position and speed bytes: 0.8 is position 204, 0.5 is speed 128.

The PAROL6 documentation covers the robot's side: the gripper port, the commander software, the API and the tool centre point.

PAR6

The PAR6 arm's control box runs the gripper on its CAN bus as a tool, with `coupler_connector_PAR6` and the PAR6 gripper cable (different from the PAROL6 one). The PAR6 documentation covers the tool configuration, the runtime's gripper commands and the firmware updates the box can do for the gripper.

UR5 and other robots

The repository has an adapter plate for the UR5 flange. Any controller with a CAN interface drives the gripper directly: Over CAN is the protocol, With Python and With ROS 2 are two ready implementations, and the CAN reference has every byte.

Errors and recovery

When the driver inside the gripper detects a fault it stops the motor, goes to idle and sets the error bit in every CAN frame it sends. The grip is released: an idle motor holds nothing beyond the mechanism's friction.

Seeing the error

- **The error bit** of every answer's identifier: 1 while any error is active.
- **Respond_Gripper_data_pack** carries the three gripper error bits: temperature, timeout, E-stop.
- **Send_Respond_State_of_Errors** returns every flag of the driver, which is where the other causes show.
- Motor GUI shows the errors in its gripper panel; `#Error` over UART prints them.

The errors

Error	Reported in	Cause	What to do
Temperature	the gripper's temperature bit	the thermistor in the windings reads more than 75 °C (`#Tlim`)	let the motor cool; hold with less current
Timeout	the gripper's timeout bit	the driver's CAN watchdog ran out: no valid frame within the time set by Send_Watchdog	the host stopped sending; fix the host, then clear
E-stop	the gripper's E-stop bit	a move command with the E-stop bit set, or `#Gripstop 1`	send a command with the bit at 0 (or `#Gripstop 0`), then clear: while the last command's bit is 1, a clear does not hold
Not calibrated (motor)	State_of_Errors, `#Error`	the driver's own motor calibration is missing: a self-built gripper before Driver configuration, or after `#Default`	calibrate the motor
Vbus	State_of_Errors	the supply is below 9 V or above the driver's `#Vmax` limit, 29 V after `#Default`	the supply, the cable
Encoder	State_of_Errors	the driver's encoder does not see its magnet	the magnet on the motor shaft (a self-built gripper)

Error	Reported in	Cause	What to do
Driver	State_of_Errors	the driver stage's own protection tripped: overcurrent, overtemperature	lower the current; check the motor leads
Velocity	State_of_Errors	the motor exceeded the driver's speed limit	unlikely in a gripper; clear
Driver E-stop	State_of_Errors	Send_ESTOP was received	clear

A gripper that is not **gripper-calibrated** is not in error: it just refuses moves until Send_gripper_calibrate has found the end stops.

Clearing

Send_Clear_Error, **Clear** in the GUI or `#Clear` clears every flag, the gripper's E-stop included, and goes to idle. An error whose cause is still present comes straight back: a hot motor, a supply that is still low.

The gripper's E-stop counts as such a cause while the E-stop bit of the last gripper command is 1: the driver raises the error again at every cycle. So after an E-stop, first send a gripper command with the E-stop bit at 0 and the action bit at 0 (or `#Gripstop 0` over UART), then clear. The next command with the action bit set moves the gripper again.

After clearing, the gripper is idle and still calibrated and activated; the next move command moves it. A clear sent while the gripper calibration runs ends the calibration, and the gripper stays uncalibrated.

When the gripper does not answer

- No answer at all: the supply, the cable (and its pinout), the bus termination (about 60 Ω across CANH and CANL with everything unpowered), 1 Mbit/s on the adapter, the node ID.
- Answers but does not move: not calibrated, not activated, an error active, or the action bit at 0.
- Moves to one end during calibration and stops there: the other end stop is blocked, or the mechanism is too stiff for the calibration's 500 mA; after 30 seconds the routine gives up.

Troubleshooting goes through the symptoms one by one.

Maintenance

The gripper needs little: every wearing part is printed, so a worn part is replaced by printing a new one, and the mechanism is a rail, two racks and a pinion that want a little grease now and then. Power the gripper off before any work on it. Before you open the body, to reach the driver or its termination switch, unplug the supply and touch earthed metal: static electricity damages the driver.

Before each session

- The fingers sit tight on their jaw plates and nothing is cracked.
- The cable's M8 nut is tight, the cable has slack for the wrist's full motion and runs clear of the jaws.
- Nothing lies in the jaws' travel, and the gripper calibration finds both end stops.

When the motion gets rough, and every few months of regular use

- **Grease.** A little lithium grease on the rail and on the rack and pinion. Remove the four screws that hold the linear rail to reach the mechanism, wipe off the old grease and the dust, grease, refit, and move the jaws over the whole stroke by hand before you power up. A grinding noise, a stiff motion or a calibration that stops short of an end are the signs that it is due.
- **Clean** the rail, the racks and the body, as in Cleaning.
- **Screws**: the finger screws, the rail screws, the lid and the coupler connector; threadlocker where the assembly manual uses it.

Cleaning

Clean the gripper switched off and unplugged. Wipe the body and the fingers with a dry or slightly damp cloth, or with a little isopropyl alcohol on the cloth for grease; use no other solvents on the printed parts, and never spray or pour anything into the body, where the driver and the motor are open to it. The gripper is not waterproof or dustproof: keep it out of wet and dusty places, and blow chips out of the rail rather than wash them out. Grease the rail again if the cleaning took its grease off.

Replacing worn parts

The fingers, the jaw plates and every other printed part are in the product repository, Gripper STL parts and gripper jaws; the rail, the carriages and the screws are in the BOM. A replacement finger is the same part as a new one, and the gripper calibration at the next power-up takes care of the end stops.

A motor or a driver is not serviced but replaced: the motor by the GBM5208-75 of the BOM, after which the motor calibration of Driver configuration is run again (the body has to be opened as in the assembly manual); the driver by a new Spectral Micro, configured the same way.

Firmware

The gripper has no firmware of its own: it runs the Spectral Micro driver's firmware with gripper mode turned on. This site describes the firmware release named in the version box on the landing page, for the gripper release the site is versioned by; a new firmware release for the same gripper updates these pages in place.

Which release does my gripper run?

Over CAN, Send_Respond_Device_Info returns the firmware version as its third byte; over UART, `#Info` prints `Software version: <number>`. The compatibility table says which releases go with which Python library and bootloader.

Updating

A gripper shipped with firmware 105 or newer carries the driver's bootloader and updates over its CAN cable: the host tells the driver to reset, the bootloader waits two seconds for a host, and the host streams the new image into it. Nothing is opened. The image is `spectral-bldc-firmware-V<number>.bin` from the driver's GitHub releases, with a manifest `firmware.json` and the release notes.

From	How
a PC with the CANvas adapter	Motor GUI over CAN: choose the release, the GUI fetches the image and flashes it
a PC, scripted	`host_tools/bl_flash.py` from the bootloader repository, over CAN (slcan or gs_usb adapters) or over the driver's UART port
a PAR6 control box	the box updates every device on its bus, the gripper included, from its dashboard or `tools/flash_motor_firmware.py`

The procedure, the bootloader's protocol and what to do when a flash is interrupted are in the Spectral Micro documentation: Updating through the bootloader and The bootloader.

Before you update

- Open the gripper or take the object out: it loses its grip during the update and until it is calibrated and commanded again.
- An update may clear the driver's configuration (the release notes say so when it does). A bought gripper's configuration is in Driver configuration, so it can be restored; note what `#Info` prints before you start.
- Give the gripper a node ID from 0 to 13: the bootloader cannot address 14 or 15.

Check

After the update, `#Info` prints the new release as `Software version: <number>` (Send_Respond_Device_Info over CAN), and the gripper calibrates. If its configuration was cleared, restore it: Driver configuration.

Grippers shipped before 2026

Grippers shipped with firmware 104 or older, which is every SSG-48 before the bootloader releases of 2026, have no bootloader. Their driver is updated once with a programming adapter on its JTAG pads, inside the body under the coupler connector, as described in Installing the bootloader on an older board. From then on the gripper updates over CAN like any other. A gripper on firmware 101 or 104 works as this manual describes, with the differences the Spectral Micro site's version 101 documents: the calibration messages and a few commands.

Release notes

The release notes of every Spectral release up to the one this site describes are in the appendix, Firmware release notes.

9. Troubleshooting

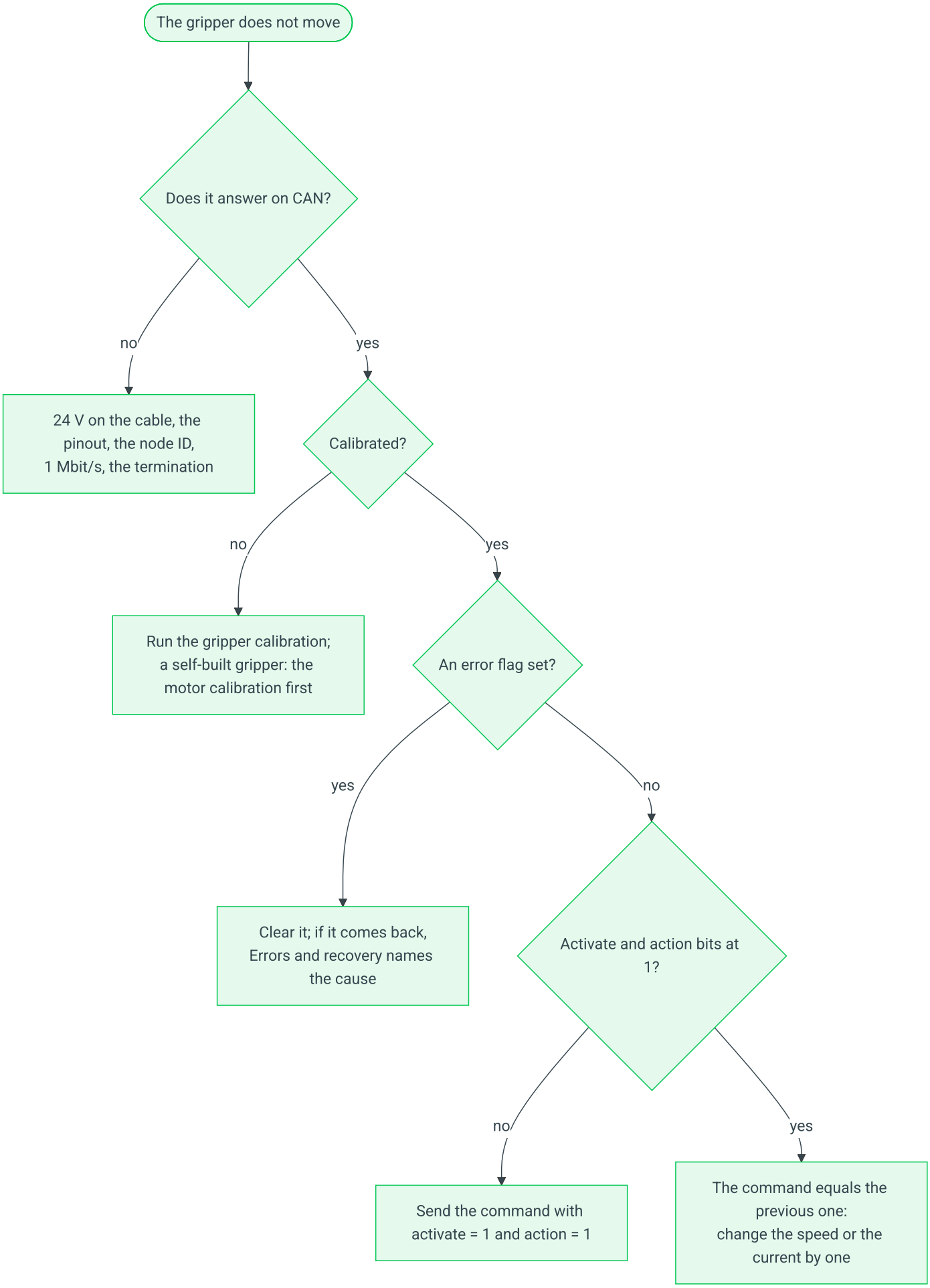
Troubleshooting

By symptom. The error flags and what clears them are in Errors and recovery. The faults in the firmware and the software that are known and not yet fixed, and what to do until they are, are on Known issues.

The gripper does not answer

Symptom	Check
nothing at all	the 24 V on the cable; the cable's pinout (a wrong one has destroyed the driver already); the node ID the host uses; 1 Mbit/s on the adapter; the termination (about 60 Ω across CANH and CANL, unpowered)
answers, then stops answering	the watchdog ran out and the host did not notice; a loose M8 connector; a second device with the same node ID
the GUI shows *No connection*	the COM port of the CANvas adapter; another program holding the port; the CAN ID in the GUI

It does not move


The gripper does not move
Does it answer on CAN?
no
yes
24 V on the cable, the pinout, the node ID, 1 Mbit/s, the termination
Calibrated?
no
yes
Run the gripper calibration; a self-built gripper: the motor calibration first
An error flag set?
yes
no
Clear it; if it comes back, Errors and recovery names the cause
Activate and action bits at 1?
no
yes
Send the command with activate = 1 and action = 1
The command equals the previous one: change the speed or the current by one

Symptom	Check
calibrated, activated, no error, no motion	the action bit of the command is 0; the command is identical to the previous one (change the speed or the current by one to repeat a move)
Not calibrated after power-up	normal: run the gripper calibration
an error is shown	Errors and recovery: clear it, then look at the cause if it comes back
the driver reports *Not calibrated* in State_of_Errors and `#Error`	the motor itself is not calibrated (a self-built gripper): Driver configuration

Calibration

Symptom	Cause
the jaws move to one end and stop; after 30 s the gripper is still not calibrated	the other end is blocked, or the mechanism is too stiff for the calibration's 500 mA: a tight rail, a pinion pressed too hard into the racks, debris in the racks
the jaws jitter and do not travel	a phase of the motor not connected (self-built), or the motor not calibrated
255 is not fully closed, or 0 not fully open	something stopped the jaws short of the end during calibration; calibrate again with the stroke free

The grip

Symptom	Cause
the object slips	the current too low for the object; the finger faces: foam on the pinch jaws, or the fin-ray fingers
the object is crushed	the current too high; a hard object takes the full force at once
object detected while nothing is there	the current limit too low for the mechanism's friction at that speed; raise the current a little or lower the speed

Symptom	Cause
the gripper stops with a temperature error after a while	a long hold at a high current; hold with the lowest current that keeps the object
the jaws creep or hum when holding	the position loop against a hard object; lower the current, or hold a soft object with a position slightly beyond the contact

Noise and heat

A gimbal motor under field-oriented control is quiet; a grinding noise is the rack and pinion without grease, or debris. The motor gets warm in use and hot under a long grip; the thermistor stops it at 75 °C. A self-built gripper without a thermistor in the windings has no such protection.

After a firmware update

If the gripper does not calibrate or behaves oddly after an update, check the driver's configuration over UART: `#Gripper` (gripper mode, 1), `#Term` (the thermistor, 1) and `#Info` (the motor calibration and parameters). A release note says when an update clears the configuration; Driver configuration restores it.

Getting help

Support says where to ask and what to include; the gripper's status and `#Info`'s output are the first things.

Known issues

The faults in the gripper's firmware and in the software that drives it that are known and not yet fixed, and what to do until they are. The gripper has no firmware of its own: these are the Spectral Micro driver's, in gripper mode. The pages that describe each part say the same where it matters; this page collects them in one place. Which versions work together is in Versions and compatibility.

Spectral driver firmware

Seen in: 107.

	What happens	What to do	More
KI-1	A gripper E-stop raises the E-stop error again at every control cycle for as long as the last gripper command carries the E-stop bit, so Send_Clear_Error (or `#Clear`) seems to do nothing.	Send a gripper command with the E-stop bit at 0 and the action bit at 0 (or `#Gripstop 0`), then clear.	Errors and recovery, clearing
KI-2	The answer to a full Send_gripper_data_pack is sent before the gripper acts on the command, so it carries the state from before it.	Ignore that answer; poll with empty frames, and give a move about 100 ms before you trust its object detection.	Over CAN, host checklist
KI-3	A full gripper command or a clear sent during the gripper calibration ends it, and the gripper stays uncalibrated. The routine's counters are not reset, so the next calibration starts where the stopped one was (it can skip the opening) and counts the time already spent towards its 30 s limit.	Send only empty frames until the calibrated bit is 1. If a calibration was stopped, calibrate again and check the bit; if it does not come, switch the gripper off and on and calibrate.	Over CAN, host checklist
KI-4	The action status in the gripper's answer is the action bit of the last command; the firmware never clears it, neither at the end of a move nor in the calibration.	Follow a move by its object detection and the position, not by the action status.	The gripper's state byte
KI-5	The motor calibration (`#Cal`) sets the driver's current limit to 1600 mA and leaves it there, and `#Save` stores it. Gripper moves are not affected, they use each command's current; the	On a self-built gripper, set `#Ilim` (1000 mA, the default, suits the	Driver setup and first run, current limit

	What happens	What to do	More
	driver's own modes, such as the current and position modes of the teleoperation guide, run with up to 1600 mA.	GBM5208-75) after `#Cal` and before `#Save`.	
KI-6	With braking in idle switched on (Send_Settings, or `#Brake 1`), the current that turning the motor drives through its shorted phases is not limited, and the setting is not saved; the gripper coasts again after every power-up.	Leave the default, coasting, unless you need the brake; set it again after each power-up, and do not force the jaws by hand while it brakes.	Send_Idle

Python library (Spectral-BLDC)

Seen in: 1.20.

	What happens	What to do	More
KI-7	`gripper_object_detection` has its two bits swapped: the library reports 1 where the gripper sends 2 (object found while opening) and 2 where it sends 1 (object found while closing); 0 and 3 are right.	Swap 1 and 2 in your program, or test for "1 or 2" when any object will do.	With Python

PAROL6 Python API (parol6)

Seen in: 0.9.1.

	What happens	What to do	More
KI-8	In `rbt.tool.status()`, `engaged` is always true (it is read from the speed field, which the robot always reports as 1), and `part_detected` reads bits 3 and 2 of the gripper's state byte, the temperature and timeout errors, not the object detection in bits 5 and 4.	To know that a part is held, check the position the jaws stopped at, or use a sensor.	On PAROL6 and PAR6, driving it
KI-9	`rbt.tool.calibrate()` counts the calibration done after a fixed 2 s without reading the calibrated bit, and the SSG-48	Wait for the calibration before the first move, as the example on the PAROL6 page does; clear	On PAROL6 and PAR6, driving it

	What happens	What to do	More
	takes about 3 s, longer when the jaws start far from the open end. The API has no action to clear a gripper error.	an error in the commander's Gripper tab or with Motor GUI, then calibrate.	

Something that is not here

Check Troubleshooting first. If it is still not explained, Report a problem says what to include so it can be reproduced. A fixed issue stays on this page, under "Fixed", with the version that fixes it.

Build guide

Build guide

The SSG-48 is open source, and that is important to us. Everything the gripper is made of is published in the product repository under the CERN-OHL-S-2.0 licence: the STEP files, the printable parts of both hardware versions, the bill of materials with a link for every item, the assembly manual and video, the fin-ray and pinch fingers, and the URDF and MJCF models. The driver's firmware and the software that drives the gripper are open source too. A gripper you build is the same gripper as the one from the shop, and this manual applies to it; design your own fingers on the jaw plate template, or a coupler connector for a robot the gripper was not designed for. This guide is the order to do it in.

> **Warning**
>
> A self-built gripper is yours to make safe: the thermistor in the windings, the current within what the motor stands, and the Safety chapter apply to it as to a bought one.

The steps

1. Bill of materials: the repository's BOM, with the reference images.
2. Printing the parts: PETG, the V2 set, the fingers in a flexible filament.
3. Mechanical assembly: the manual, page by page, the video, the cable.
4. Driver setup and first run: the motor calibration and gripper mode during the assembly, then the first moves.

Kits and the shop

You do not have to source everything yourself:

	What is in it	You add
Spectral Micro BLDC starter kit	the Spectral Micro driver, the CANvas USB-to-CAN adapter, the USB-to-serial adapter, an ST-Link with the programming adapter, the UART, power and CAN cables, the 6 × 2.5 mm encoder magnet, the 100K thermistor	the GBM5208-75 motor, the rail and its carriages, the shaft coupler, the M8 connector, the screws and inserts, the printed parts
Spectral Micro driver alone	the driver	the rest, and the adapters its setup needs
Assembled SSG-48 gripper	the gripper, built, calibrated and configured, in black or grey	the fingers, the cable, the shaft coupler for PAROL6

The starter kit is the good-value start for a build: it has the driver and everything the driver's setup needs, the CAN adapter for Motor GUI, the serial adapter for the UART, the magnet, the thermistor and the two internal cables, for little more than the driver alone. The PAROL6 gripper cable is in the shop too.

Bill of materials

This is a copy

The original is BOM/BOM.md in the gripper's repository; this page is that file as of 2026-11-04 (commit `37e3488`), with the reference images shown inline. If a link is broken or a part is wrong, open an issue there, and the fix lands here with the next sync. The shop's starter kit covers the driver and everything its setup needs.

This BOM lists all components required to build the SSG48 gripper. Quantities are for **one gripper**.

Note

If any product links are broken, unavailable, or no longer point to the correct item, please open an issue, send us a message on Discord, or contact us via email. You can also use the reference images to find the same or an equivalent part from another supplier.

Screws and Nuts

Diameter / Type	Length	Quantity	Description	Link
M2.5	8 mm	4		
M2	4 mm	8		
M3	5 mm	5	Cheese head screw	
M2	10 mm	1		
M3	8 mm	7	Cheese head screw	
M4	10 mm	4		
M3	12 mm	4		
M3	10 mm	4		

Brass Inserts

Thread	Length	Outer Diameter	Quantity	Description	Link
M3	3 mm	4.5 mm	1		Link
M3	5 mm	4.5 mm	15		Link

Motors

> **Note**
>
> **If you are having a hard time sourcing motors consider building our MSG gripper!**

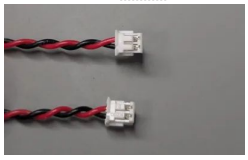
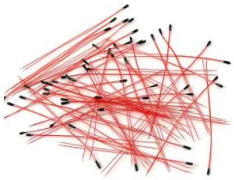
Type	Quantity	Description	Link
GBM5208-75	1		Link // Alibaba link

Mechanical Components

Type	Quantity	Description	Link
Shaft coupler	1	8 mm hole shaft, M4 screw holes 	Link
MGN7C slider	2		Link
Linear guide	1	100 mm 	Link
Magnet	1	6 mm diameter, 2.5 mm thickness, diametrically magnetised  	Link

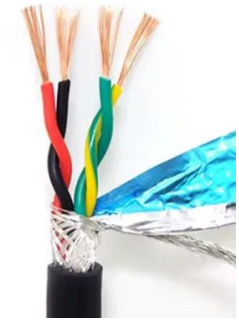
Electronics

Type	Quantity	Description	Link
Spectral micro BLDC driver	1	source-robotics.com 	Link

Type	Quantity	Description	Link
XH 2.54 mm, 300 mm, 2P, double-ended	1	Cable for 24 V 	Link
ZH 1.5 mm, 300 mm, 2P, double-ended	1	Cable for CAN bus 	Link
Male 4-pin connector	1		Link
100K NTC thermistor	1	NTC 100K 3950, glued into the motor windings: the driver stops the gripper at 75 °C 	Link
Thermal cement	1	Holds the thermistor in the windings and conducts the heat to it 	Link

Cables

Parts for the DIY cable version below:

Type	Length / Spec	Quantity	Description	Link
26 AWG cable	0.15 mm², 4-wire twisted pair	0.4 m	Usually 35 cm cable is enough 	Link
M8 4-pin male wire connector	–	1	M8 4Pin Male 	Link
M8 4-pin female wire connector	–	1	M8 4Pin Female 	Link

Check out the cable building guide here

Note

Equivalent parts from other suppliers may be used as long as specifications match.

Printing the parts

Material and settings

PETG for the body, the lids, the jaw plates and the coupler connector; the fin-ray fingers in a flexible filament. The assembly manual's reference prints were made on a Prusa MK2S in PETG with 25 % infill, supports only where a part needs them: clean every support off before assembly, the bearing seats and the rail slot in particular.



Figure 1: The body printed in black and in grey, the two colours the shop sells.

The files

What	Where
The body, the lid, the gear and the magnet shaft, in two hardware versions: print the `V2` set, the large-bearing version (Hardware versions)	Gripper STL parts
The jaws, the guide cover, and the coupler connectors for PAROL6 and PAR6	the same folder
The fin-ray fingers and their jaw plate	gripper jaws
The STEP files, for a coupler connector of your own (the generic template) or your own fingers on the jaw plate	STEPS

The brass inserts go in with a soldering iron after printing, as the assembly manual shows for each part.

Mechanical assembly

The repository's assembly manual, page by page: the jaws, the gear and magnet on the motor, the motor into the body, the connector and its wires, the driver and the thermistor, the calibration, the linear guide, the lid, the coupler connector, the cable. Read it once before you start: some steps cannot be undone without taking the body apart again. Static electricity damages the driver: touch earthed metal before you handle it, and hold it by the edges.



Download the assembly manual (PDF, 3 MB) | The assembly video

The thermistor

The motor has a 100K NTC thermistor in its windings; the driver stops the gripper when it reads 75 °C. Put the bead between the windings and fix it with a drop of thermal cement, which sets hard and conducts the heat: thermal paste stays soft and lets the bead move, ordinary glue insulates it. Solder the leads to the driver's TERM and 3V3 pads and keep them clear of the rotor. The thermistor and the cement are in the Bill of materials, and the starter kit has the thermistor; Thermistor on the Spectral Micro site has photos of it on the board and in a gimbal motor.

Calibrate on the way

The motor is calibrated during the assembly, not before it: once the motor and the driver are in the body, the wires soldered and the thermistor in the windings, and before the linear guide with the jaws goes on (page 22 of the manual), connect the serial adapter and run Driver setup and first run. The racks would block the calibration's turns, so the order matters.

The cable

The gripper's cable is not in the box. The cable has the parts, the wiring diagrams for PAROL6 and PAR6, and the warning that the two are not interchangeable; the repository's cable guide is the original.

The fingers

The fin-ray and pinch fingers, and fingers of your own on the jaw plate: Jaws.

Driver setup and first run

A gripper from the shop arrives with its driver configured: the motor calibrated, gripper mode on, the thermistor on, everything saved. A gripper you build has a bare Spectral Micro driver that is brought to the same state once, over its UART port, **during the assembly**: after the motor and the driver are in the body, the wires soldered and the thermistor in the windings, and before the linear guide with the jaws goes on (page 22 of the assembly manual). The motor calibration turns the motor several revolutions in both directions, which the racks could not follow. This page is that procedure; the commands are the driver's, and the Spectral Micro documentation explains each in depth.

What you need

- The gripper assembled up to that point, with the encoder magnet on the motor shaft and the driver mounted.
- The 100K NTC thermistor in the motor windings, held by thermal cement, its leads on the driver's TERM and 3V3 pads. Both parts are in the Bill of materials; Thermistor on the Spectral Micro site shows how it is fitted. `#Term 1` below needs it: without it the driver reads below −20 °C and stops with a temperature error.
- A 3.3 V USB-to-serial adapter on the driver's UART connector, a terminal at 256000 baud with CR+LF line endings, and 24 V on the driver's power connector. The serial terminal on the Spectral Micro site shows the terminal setup; Motor GUI over UART does the same job in the browser.

1. The motor calibration

Restore the defaults first, so that nothing from another motor or junk in the EEPROM goes into the calibration, then calibrate. The Spectral Micro measures the motor's resistance and inductance and finds its pole pairs by itself; nothing is entered:

```
#Default
#Cal
```

The motor clicks, then turns, the driver prints each step and ends with `Calibration success!`. The whole routine and its messages are in Calibration on the Spectral Micro site. `#Default` also resets the serial number the driver reports; note what `#SERNUM` prints before you start if you need it.

Check

The driver ends with `Calibration success!`. If it stops with an error, check the motor's three wires and the encoder magnet, and see Calibration.

2. The gripper settings

```
#Gripper 1
#Term 1
#Tlim 75
#CANID 0
#Save
```

Command	What it does
`#Gripper 1`	gripper mode: the driver runs the gripper state machine and answers the gripper messages
`#Term 1`, `#Tlim 75`	the thermistor on, the temperature limit at 75 °C. Without it the motor can burn out under a long grip
`#CANID 0`	the node ID on the robot's bus; 0 to 13, see Over CAN
`#Save`	nothing above survives a power cycle until it is saved

The current loop gains the calibration computes for the motor are what the shipped grippers use. Motor GUI's *SSG-48* preset, on its setup page over UART, writes this step for you: the gains of a calibrated shop gripper (`#Kpiq 7.0381`, `#Kiiq 4.2557`), gripper mode, the thermistor and the CAN ID (6, the PAR6 gripper's; change it before applying for another robot), then saves.

Check

After a power cycle, `#Gripper` and `#Term` print 1, `#Tlim` 75 and `#CANID` the node ID; `#Temp` prints the room temperature, not `Temp Off`. A temperature error at once means the thermistor is not connected (The thermistor).

3. Finish the assembly and calibrate the gripper

Fit the linear guide with the jaws, the lid and the coupler connector, power the gripper up through its cable, and run the gripper calibration (`#Gripcal`, or Motor GUI's **Calibrate**): the jaws open fully and close fully. Then a first move at a low current, as in Calibrate, activate, move. `#Gripinfo` prints the gripper's state, including the end stops the calibration found.

Check

The jaws open fully and close fully, the gripper reports *Calibrated*, and the first moves reach their positions. If the calibration gives up after 30 s: Calibration.

Current limit

The gripper's moves use the current limit of each command, not the driver's `#Ilim`. The firmware does not clamp it; Force and speed says what is safe for the motor.

Appendix

Performance tests

Thermal tests of the SSG-48 gripper under different conditions. The thermal limit is 75 °C at the thermistor in the motor windings.

> **Warning**
>
> A gripper built without a thermistor in the windings has no temperature protection and can damage its motor in normal operation.

Several days of cycling

- Open and close every 2.5 s, no load: the current near 0
- 7 days without interruption
- Motor temperature: 42 °C
- Ambient temperature: 33 °C

Holding at a constant current

- Hold an object at 300 mA for 24 hours
- Stable motor temperature: 62 °C
- Ambient temperature: 33 °C

The current that trips the thermal limit

- Hold an object at 1600 mA
- The temperature error trips after 4 minutes
- Ambient temperature: 33 °C

What this means for use: a hold at 300 mA (about 16 N) ran 24 hours without reaching the limit; a grip at 1500 mA (about 80 N, ideal) is for seconds, not minutes. Between the two, the motor heats in proportion to the square of the current.

Firmware release notes

What changed in each release of the Spectral firmware the gripper runs, from the release notes in the driver's repository, up to the release this site describes. Entries about the driver's motion modes do not concern a gripper; the gripper mode, the CAN protocol and the bootloader do.

V107 (2026-08-26)

Spectral BLDC firmware **V107** — application image for flashing via the Spectral bootloader.

CAN bus

- Fixed `IN_DATA_PACK_PD` writing the commanded speed into `Feedforward_speed` instead of `Velocity_setpoint` — PD mode's KD (velocity-damping) term was being evaluated against a stale/zero setpoint, so it silently ignored the commanded speed.
- Added a generic mask+value settings command (`IN_SETTINGS`) for CAN-unreachable boolean settings, starting with idle-mode brake/coast selection.

Usability

- Idle mode can now be set to brake (driver stays enabled, phases held shorted via a zero-voltage PWM vector) or coast (Hi-Z, previous/default behavior), via `IN_SETTINGS` over CAN or `#Brake <0|1>` over UART.

Calibration

- Fixed `#Cal` spuriously failing with "Commutation/alignment check failed!" on some individual motor units. The pre-check that confirms the motor responds to a current command was driving a fixed voltage at the raw (uncalibrated) encoder angle; whether that produced torque depended on each unit's random magnet-to-encoder mounting angle. It now retries once at a 90-degree angle bias before failing.

Assets

- `spectral-bldc-firmware-V107.bin` — application firmware image (flash at `0x08003000` , target STM32F103CB).
- `firmware.json` — manifest with version, size, and SHA-256 for download verification by the web flasher.

Integrity

- SHA-256: `2ca1f2e3ad3ee7848cf47d8733b084be2daa87fc2e49f954dd41c76405d48ada`
- Size: 95404 bytes

Built from commit `2e7febc` on `Bootloader_main` .

V106 (2026-07-06)

Spectral BLDC firmware **V106** — application image for flashing via the Spectral bootloader.

CAN bus - The controller now drains up to 8 CAN messages per pass instead of just 1, so short bursts (multiple commands sent back-to-back, or a busy multi-node bus) no longer silently drop messages once the 3-deep hardware FIFO fills. - A CAN peripheral that fails to initialize no longer bricks the board — UART stays fully usable, only the CAN bus is skipped.

Safety - Added a hardware watchdog (100ms). If the control loop ever stops running, the board resets instead of leaving the motor driven at a stale duty cycle. - Fixed a bug where `Torque_mode` could silently overwrite your commanded Iq setpoint when clamping to the current limit — reading it back after clamping now shows what you actually asked for, and raising the current limit afterward immediately restores the original commanded torque instead of requiring the command to be resent. - Fixed undefined behavior in a serial-protocol integer conversion (`fourBytesToInt`). - Synced default configuration values that had drifted between two locations in the code, so a fresh `#Default` and a freshly-flashed board's built-in defaults now agree.

EEPROM / configuration integrity - Added CRC validation over the saved configuration, split into two independent halves: calibration data (resistance, inductance, Kt/KV, angle offset, current-loop gains, etc.) and settings (PID tuning, limits, watchdog config, CAN ID, etc.). - This means a corrupted save (e.g. from a power loss mid-write) is detected and safely defaulted instead of silently running with garbage values — and critically, a future firmware update that only adds a new setting can no longer wipe your calibration, since the two are validated independently. - Boards upgrading from older, pre-CRC firmware keep their existing calibration automatically — no need to recalibrate after this update. - A few unused slots were reserved in the calibration block for upcoming features (e.g. magnet alignment), so those can be added later without another migration.

Performance - PWM output no longer re-resolves which timer/channel it's writing to on every single call (it was doing this 4x per control tick) — resolved once and cached, freeing up ISR headroom. - Velocity calculation switched from a floating-point divide to an integer multiply.

Usability - New `#Help` command — prints a summary of the available commands. - Serial now responds with `"Unknown command"` for unrecognized input instead of silently ignoring it. - Calibration now prints progress during the angle-offset search (previously up to ~40 seconds of silence that could look like the board had hung). - `#Info` now always reports live ISR execution time, useful for checking control-loop headroom on the bench. - Fixed a leftover calibration-failure message that incorrectly suggested swapping motor phases — that failure mode no longer exists (reversed wiring has been auto-detected/compensated in software since V105); the message now accurately describes what actually happened (a commutation/current-response check, not a wiring-direction problem).

Assets - `spectral-bldc-firmware-V106.bin` — application firmware image (flash at 0x08003000, target STM32F103CB). - `firmware.json` — manifest with version, size, and SHA-256 for download verification by the web flasher.

Integrity - SHA-256: `edb06ca112a7d24b0581e0b157a4524928067a46f8dc850ce22df8128092a758` - Size: 94908 bytes

Built from commit `e1f8611` on `Bootloader_main`.

V105 (2026-07-05)

Spectral BLDC firmware **V105** — reversed-wiring auto-calibration

`#Cal` now auto-detects reversed motor phase wiring during the open-loop spin and compensates it in software as a phase B<->C reflection, instead of failing and telling you to physically swap phases. Angle-offset calibration is now found automatically as part of `#Cal` itself (deterministic rotor-lock + verify + symmetry-sweep), so the standalone `#Calangle` command has been removed. `#Info` now reports the true firmware version instead of a stale EEPROM value. Existing calibrated units are unaffected and do not need `#Default`.

Assets - `spectral-bldc-firmware-V105.bin` — application firmware image (flash at 0x08003000, target STM32F103CB). - `firmware.json` — manifest with version, size, and SHA-256 for download verification by the web flasher.

Integrity - SHA-256: `0bd80ea038b8497fca8ab6a613b083d28e3473b360fa152b512b8ab0304dedb5` - Size: 91732 bytes

Built from commit `b7dded3` on `Bootloader_main`. Includes reversed-wiring auto-calibration (`commutation_dir` detection + phase B<->C reflection) and deterministic rotor-lock angle-offset calibration.

V104 (2026-06-19)

Spectral BLDC firmware **V104** — bootloader build.

Assets

- `spectral-bldc-firmware-V104.bin` — firmware image (90,968 bytes).
- `firmware.json` — manifest with version, size, and SHA-256.

Integrity

- SHA-256: `76d70a6091218d74021eeef55c31d39f19fe97f68e2b236169f84c708518a360`
- Size: 90968 bytes

Built from commit 20765c9 on `Bootloader_main`. Includes CAN Vlim command and bootloader support.

v101 (2025-09-29)

Some small fixes added based on STEPFOC code that is a clone of this code.

1. Added theta_offset calib.
2. Fixed some issues related to gripper mode where it vibrates

NOTE added another entry to EEPROM memory. Loading this code to update your previous code you will need to run #Default to load that new entry in EEPROM!

v100 (2024-03-27)

Software version 100

Versions and compatibility

Which Spectral firmware release works with which bootloader, gripper hardware, Python library and ROS 2 driver. One row per firmware release, newest first; the last column names the version of this site, a gripper release, that describes it.

Firmware (Spectral)	Released	Bootloader	SSG-48 gripper	Python library (Spectral-BLDC)	ROS 2 driver
107	2026-08-26	required (installed on grippers shipped with 105 or newer)	V2, V1.1	1.20 or newer	ssg48_adaptive_electric_gripper_r msg_gripper_ros2
106	2026-07-06	required	V1.1	1.19 or newer	ssg48_adaptive_electric_gripper_r
105	2026-07-05	required (the first release that needs it)	V1.1	1.19 or newer	ssg48_adaptive_electric_gripper_r
104	2026-06-19	none (flashed with a programming adapter); the last release of the main branch	V1.1	1.19 or newer	ssg48_adaptive_electric_gripper_r
101	2025-09-29	none (flashed with a programming adapter)	V1.1	1.17 or newer	ssg48_adaptive_electric_gripper_r

Send_Respond_Device_Info returns the firmware release a gripper runs; `#Info` prints it over UART.

Resources

Glossary

Action bit

A bit of the move command: 1 carries out the move, 0 leaves the gripper idle with the command stored. The action status of the answer repeats this bit.

Activated

A flag the gripper keeps, set by the activate bit of the move command or by `#Gripact`. A move is carried out only while the gripper is activated and calibrated.

CAN ID (node ID)

The address of the gripper on the CAN bus, 0 to 13; every gripper ships as 0. It occupies bits 10 to 7 of the 11-bit CAN frame identifier; the command ID occupies bits 6 to 1 and the error bit is bit 0.

CANvas

Source Robotics' USB-to-CAN adapter. It appears as a serial port and speaks slcan, which `python-can`, Motor GUI and the ROS 2 drivers use.

Coupler connector

The round part on the back of the body that mounts the gripper: one for PAROL6, one for PAR6, a UR5 adapter plate, and a template for your own flange.

Current limit (grip force)

The current, in mA, the driver may put into the motor during a move. Torque is current times the motor's torque constant. It limits grip force; the conversion to newtons depends on the motor and mechanism. See Force and speed for the ideal model and measurement limits.

Driver

The Spectral Micro board inside the body: field-oriented control of the motor, the gripper state machine and the CAN and UART interfaces. The gripper's firmware is the driver's firmware.

E-stop bit

A bit of the move command. 1 stops the gripper and raises the gripper E-stop error, which comes back at every cycle while the last command's bit is 1: send a command with the bit at 0, then clear the error.

Error bit

Bit 0 of every CAN frame identifier the driver sends: 1 while any error is active.

Field-oriented control (FOC)

Controlling the motor's current as a vector relative to the rotor, from the driver's encoder. It gives a brushless motor a known, limited torque at standstill, which is what makes a force-controlled gripper out of it.

Fin-ray finger

A finger whose two flexible sides are joined by ribs, so that it bends around an object pressed into it. The repository has printable ones for the jaw plates.

Firmware release

One build of the Spectral firmware, numbered 101, 104, 105, … and tagged `v101`, `V104`, `V105`, … in the driver's repository. `#Info` prints the release a gripper runs. This documentation describes one release; the version selector switches between them.

Gripper calibration

The routine (`Send_gripper_calibrate`, `#Gripcal`) that opens the jaws fully, then closes them fully, at low force, and records both end stops as positions 0 and 255. Needed after every power-up.

Gripper mode

The driver's mode in which it runs a gripper (`#Gripper 1`, saved): move commands with a jaw position, a speed and a current limit, and object detection.

Motor calibration

The driver's own calibration of the motor (`#Cal`): the resistance and inductance measured, the pole pairs found, the encoder aligned, the motor constants measured. Done once, at the factory or when you build the gripper; not to be confused with the gripper calibration.

Object detection

The gripper's report of why the jaws stopped: 0 still moving, 1 an object stopped them while closing, 2 an object stopped them while opening, 3 they reached the commanded position.

Position

The jaw position as the gripper commands and reports it: 0 fully open, 255 fully closed, linear in between.

Rack and pinion

The gripper's mechanism: a pinion on the motor meshes with two racks that run on the linear rail in opposite directions, so one motor moves both jaws symmetrically.

Speed

The speed of a move, 0 to 255, mapped by the firmware to 40 to 80000 encoder ticks per second of the motor.

Stroke

How much the distance between the jaw plates changes from fully open to fully closed: 48 mm. Each jaw travels half of it.

Timeout error (watchdog)

The driver's CAN watchdog, as the gripper reports it: when the watchdog is set (`Send_Watchdog`) and no gripper command or status request arrives in time, the gripper stops and raises it.

Watchdog

A timer the driver resets with every gripper command (full or empty) and every status request it receives. When it is set and runs out, the gripper reports a timeout error and stops.

CAD and downloads

Everything to download for the SSG-48, at the versions these pages describe. The versions come from one file, `versions.yml`, so this page changes with them; which versions work together is in Versions and compatibility.

Firmware, software and models

What	Version	Download	How to install
Driver firmware (Spectral), the gripper's only firmware: `spectral-bldc-firmware-V<number>.bin` and `firmware.json`	107 (2026-08-26)	The release, or by Motor GUI, which fetches it	Firmware
Python library (`Spectral-BLDC`)	1.20	`pip install Spectral-BLDC` ; the source and examples in Spectral-BLDC-Python	With Python
URDF and MJCF (MuJoCo) models: the gripper alone with pinch or fin-ray fingers (`ssg48_urdf.zip` , `ssg48_mujoco.zip`), and PAROL6 and PAR6 with the gripper mounted	2.18.1 (2026-11-03)	The release; the guide: ssg48.md	Data for robot integration
CAN message definitions (JSON, DBC) for this firmware release	the same release	CAN frame tool and downloads	
Motor GUI, the browser tool for the driver and the gripper	the current one	motorgui.com	Motor GUI

The version links open the files at exactly that release. Known faults in each are on Known issues.

CAD and build files

The latest files, on the main branch of the product repository.

What	Where
STEP files: the body parts, the whole gripper, the jaw plate and pinch jaws, the coupler connector template, the UR5 adapter plate	STEPS in the product repository

What	Where
Printable parts (STL) of both hardware versions and the coupler connectors for PAROL6 and PAR6	Gripper STL parts
Fin-ray fingers and their jaw plate	gripper jaws
Bill of materials	BOM.md, and on this site: Bill of materials
Assembly manual (PDF) and the motor's datasheet	Assembly manual
Assembly video	YouTube
Cable wiring diagrams	Building_cables.md

More

What	Where
The driver's documentation	docs.source-robotics.com/Spectral-BLDC-docs
ROS 2 drivers	ssg48_adaptive_electric_gripper_ros2, source_robotics_msg_gripper_ros2
The legacy gripper GUI, a Python program	SSG-gripper-GUI
CANvas USB-to-CAN adapter	CANvas-adapter
This documentation	SSG48-gripper-docs
Shop	source-robotics.com

API reference

API reference

The gripper's CAN messages and UART commands, its status and error bits, and the Python library calls that send them, for the firmware named on the landing page. The gripper is a Spectral Micro BLDC driver in gripper mode and accepts the driver's whole command set; only what a gripper host or a self-built gripper needs is here, the rest is in the Spectral reference.

- CAN messages: the identifier, and the gripper, system and status messages
- UART commands: the gripper commands, the driver setup commands, status and utility
- Errors and status bits
- Python library: which library call sends which message

Conventions. Bytes are numbered from 0 and multi-byte values are big-endian. Bits are numbered from the most significant: bit 7 is the first flag in a byte. The position is 0 (open) to 255 (closed), the speed 0 to 255, the current in mA. UART command names are case-sensitive.

The command IDs, frame lengths and responses are read from the firmware's source at its release tag when the site is built, so they cannot drift from the firmware; the descriptions and byte layouts are written by hand.

CAN frame tool and downloads

Build or inspect a frame using the command definitions of this manual's firmware release. All calculations run in your browser; the tool does not connect to an adapter or send commands to hardware.

Downloads

Command definitions (JSON) · Gripper DBC

The JSON includes the firmware tag, source commit, message IDs, every documented layout and its bit order. The DBC files cover **nodes 0 to 13**, with both the clean and fault-bit identifiers of driver replies. Reserved commands are omitted. Values use the firmware's native units: ticks, ticks/s, mA and the units in the CAN reference.

An RTR request uses the same identifier as its data reply. DBC describes the **data reply**; it does not represent the request's RTR flag or the empty gripper status poll. Both are supported by the browser tool.

The gripper's two object-detection bits have an unusual order: bit 5 is the low bit, bit 4 the high bit. The DBC signal `gripper_object_detection_raw` retains the wire value and supplies text labels (raw 2 = object while closing, raw 1 = object while opening). The browser decoder converts that pair to the firmware's numeric status (1 = closing, 2 = opening).

Try a frame

The example `0x07A`, data `C8 1E 01 2C C0`, is a command to gripper node 0: position 200, speed 30, current 300 mA, activate and action set, E-stop and release direction clear. These are an illustration of the encoding; the device's current and motion limits still apply. Select another message above the fields to build one of its documented forms. To inspect a reply, enter its identifier and data bytes below.

Check: the example decodes to the values above. Change the identifier to `0x1FA` to see the same command addressed to node 3. An invalid length or out-of-range field produces an explanation instead of a frame.

For wiring, termination, host timing and recovery, use Over CAN. To look up a message's purpose and accepted lengths, use the CAN reference.

CAN messages

CAN messages

CAN 2.0, 1 Mbit/s, 11-bit identifiers. The identifier carries the node ID (bits 10 to 7, 0 to 13), the command ID (bits 6 to 1, 0 to 63) and, in frames from the gripper, the error bit (bit 0). Over CAN explains how the pieces are used; this page lists them.

Three kinds of message:

- **Send_…** — the host sends data to the gripper. The gripper command is answered with Respond_Gripper_data_pack; the others are not answered.
- **Send_Respond_…** — the host asks for data with a **remote frame** (RTR bit set, no data) under the command ID; the gripper answers with a data frame under the same command ID.
- **Respond_…** — frames only the gripper sends.

Multi-byte values are big-endian (data[0] first); bits inside a byte are numbered from the most significant bit (bit 7 first).

The messages a gripper needs

ID	Message	Direction	Length	Group
0	Send_ESTOP	host → driver	0	System
1	Send_Clear_Error	host → driver	0	System
9	Respond_Heartbeat	driver → host	0	Status requests
10	Send_Respond_Ping	host → driver as a remote frame, driver → host as a data frame	0 (remote) / 0	Status requests
11	Send_CAN_ID	host → driver	1	System
12	Send_Idle	host → driver	0	System
13	Send_Save_config	host → driver	0	System
14	Send_Reset	host → driver	0	System
15	Send_Watchdog	host → driver	5	System

ID	Message	Direction	Length	Group
23	Send_Respond_Temperature	host → driver as a remote frame, driver → host as a data frame	0 (remote) / 2	Status requests
24	Send_Respond_Voltage	host → driver as a remote frame, driver → host as a data frame	0 (remote) / 2	Status requests
25	Send_Respond_Device_Info	host → driver as a remote frame, driver → host as a data frame	0 (remote) / 7	Status requests
26	Send_Respond_State_of_Errors	host → driver as a remote frame, driver → host as a data frame	0 (remote) / 2	Status requests
30	Send_Heartbeat_Setup	host → driver	4	System
60	Respond_Gripper_data_pack	driver → host	4	Gripper
61	Send_gripper_data_pack	host → driver	0, 5	Gripper
62	Send_gripper_calibrate	host → driver	0	Gripper

The driver's other messages (the motion commands, the gains and limits, the encoder and current requests) are not listed here; they are in the Spectral CAN reference.

By group

- Gripper: the move command and its answer, the gripper calibration
- System: node ID, E-stop, idle, save, reset, clear, watchdog, heartbeat setup
- Status requests: temperature, voltage, identity, the driver's error flags, the heartbeat

The Python library

Every message the host can send has a method of the same name in `Spectral_BLDC.SpectralCAN`, listed in Python library.

Tools and downloads

Build or decode a CAN frame, and download this firmware's JSON and DBC definitions.

Gripper

The move command, its answer, and the gripper calibration. How the gripper works explains what the gripper does with them.

Send_gripper_data_pack

ID 61 | host → driver | 0 or 5 bytes | Python: `Send_gripper_data_pack()`

The gripper command, for a driver set up as a gripper (`#Gripper 1`, saved). Position, speed and the current limit of the move, plus the activate, action and E-stop bits. A move is carried out only while the gripper is calibrated, activated and the action bit is 1. An empty frame (0 bytes) only asks for the state. The answer is sent before the driver acts on the frame, so it carries the state from before it. While the gripper calibration runs, send only the empty frame: a full one ends the calibration.

5 bytes

Byte	Bit	Field	Type	Meaning
0		Position	unsigned 8-bit	0 fully open, 255 fully closed
1		Speed	unsigned 8-bit	0 slowest, 255 fastest
2-3		Current	signed 16-bit	current limit of the move in mA, the grip force; not clamped by the firmware, see Force and speed for the safe range
4	7	Activate	bit	1 activate the gripper, 0 deactivate
4	6	Action	bit	1 go to the position, 0 idle
4	5	E-stop	bit	1 raises the gripper E-stop error, again at every cycle while the last command's bit is 1
4	4	Release direction	bit	read by the firmware but not used

The driver answers with **Respond_Gripper_data_pack**.

Respond_Gripper_data_pack

ID 60 | driver → host | 4 bytes

The driver's answer to Send_gripper_data_pack: position, measured current, and the state bits. Python attributes `gripper_position`, `gripper_current`, `gripper_activated`, `gripper_action_status`, `gripper_object_detection`, `gripper_temperature_error`, `gripper_timeout_error`, `gripper_estop_error`, `gripper_calibrated`.

The frame (4 bytes)

Byte	Bit	Field	Type	Meaning
0		gripper_position	unsigned 8-bit	0 fully open, 255 fully closed
1-2		gripper_current	signed 16-bit	measured current in mA
3	7	gripper_activated	bit	1 activated
3	6	gripper_action_status	bit	the action bit of the last command, as received; stays 1 while holding, not changed by the calibration
3	5-4	gripper_object_detection	2 bits	bit 5 is the low bit, bit 4 the high bit: 0 moving, 1 object detected while closing, 2 object detected while opening, 3 at the position
3	3	gripper_temperature_error	bit	
3	2	gripper_timeout_error	bit	
3	1	gripper_estop_error	bit	
3	0	gripper_calibrated	bit	1 calibrated

Send_gripper_calibrate

ID 62 | host → driver | 0 bytes | Python: `Send_gripper_calibrate()`

Starts the gripper calibration (the jaws find their end stops). Needed after every power-up. It has no answer: poll with empty Send_gripper_data_pack frames until the calibrated bit is 1, and send nothing else meanwhile, since a full gripper command or Send_Clear_Error ends the calibration. With an error active the request is dropped: clear first.

The driver does not answer this command.

System

Node ID, stopping, saving, resetting, clearing errors, the watchdog and the heartbeat setup: the driver's system messages a gripper host uses.

Send_CAN_ID

ID 11 | host → driver | 1 bytes | Python: `Send_CAN_ID()`

Gives the driver a new node ID, 0 to 13 (the firmware accepts up to 15, but the bootloader reserves 14 and 15: a driver on one of those cannot be updated over CAN). The driver answers to the new ID at once; Send_Save_config keeps it over a power cycle and also writes it into the bootloader's board ID.

1 bytes

Byte	Bit	Field	Type	Meaning
0		ID	unsigned 8-bit	0 to 15

The driver does not answer this command.

Send_ESTOP

ID 0 | host → driver | 0 bytes | Python: `Send_ESTOP()`

Stops the motor and raises the E-stop error, which stays until Send_Clear_Error.

The driver does not answer this command.

Send_Idle

ID 12 | host → driver | 0 bytes | Python: `Send_Idle()`

Puts the driver in idle mode (the motor coasts or brakes, see Send_Settings).

The driver does not answer this command.

Send_Save_config

ID 13 | host → driver | 0 bytes | Python: `Send_Save_config()`

Writes the current configuration (gains, limits, calibration, CAN ID, heartbeat, watchdog, gripper flag) to the EEPROM.

The driver does not answer this command.

Send_Reset

ID 14 | host → driver | 0 bytes | Python: `Send_Reset()`

Resets the driver. From firmware 105 the reset lands in the bootloader, which waits 2 s for a flashing host and then starts the firmware again; this is how a firmware update begins.

The driver does not answer this command.

Send_Clear_Error

ID 1 | host → driver | 0 bytes | Python: `Send_Clear_Error()`

Clears the error flags and goes to idle. An error whose cause is still present comes straight back; the gripper's E-stop error comes back while the last gripper command's E-stop bit is 1, so send a command with the bit at 0 first. A clear also ends a running gripper calibration. A calibration error is cleared only by calibrating.

The driver does not answer this command.

Send_Watchdog

ID 15 | host → driver | 5 bytes | Python: `Send_Watchdog()`

Sets the watchdog: the gripper command (full or empty) and the status requests reset it, Send_Clear_Error, Send_gripper_calibrate and the setup messages do not; when the time runs out the driver reports a watchdog error and stops. 0 turns it off (the default). The action byte is stored but not acted on by this firmware: the driver always stops into its idle behaviour.

5 bytes

Byte	Bit	Field	Type	Meaning
0-3		Time	unsigned 32-bit	ms; 0 = off
4		Action	unsigned 8-bit	0 idle, 1 hold, 2 brake (stored only)

The driver does not answer this command.

Send_Heartbeat_Setup

ID 30 | host → driver | 4 bytes | Python: `Send_Heartbeat_Setup()`

Sets the heartbeat interval in ms; 0 turns the heartbeat off (the default). Kept by Send_Save_config.

4 bytes

Byte	Bit	Field	Type	Meaning
0-3		Interval	unsigned 32-bit	ms; 0 = off

Status requests

Remote frames the host sends under a command ID, answered by the gripper with a data frame under the same ID, and the heartbeat the gripper sends on its own when it is set up.

Send_Respond_Ping

ID 10 | host → driver as a remote frame, driver → host as a data frame | remote frame, 0 bytes | Python: `Send_Respond_Ping()`

Asks the driver to answer with an empty frame, to see that it is on the bus.

The frame (0 bytes)

Byte	Bit	Field	Type	Meaning

The driver answers with **Send_Respond_Ping**.

Send_Respond_Temperature

ID 23 | host → driver as a remote frame, driver → host as a data frame | remote frame, 0 bytes | Python: `Send_Respond_Temperature()`

Asks for the thermistor temperature. Meaningful only with the thermistor enabled (`#Term 1`).

The frame (2 bytes)

Byte	Bit	Field	Type	Meaning
0-1		temperature	signed 16-bit	°C

The driver answers with **Send_Respond_Temperature**.

Send_Respond_Voltage

ID 24 | host → driver as a remote frame, driver → host as a data frame | remote frame, 0 bytes | Python: `Send_Respond_Voltage()`

Asks for the supply voltage.

The frame (2 bytes)

Byte	Bit	Field	Type	Meaning
0-1		voltage	unsigned 16-bit	mV

The driver answers with **Send_Respond_Voltage**.

Send_Respond_Device_Info

ID 25 | host → driver as a remote frame, driver → host as a data frame | remote frame, 0 bytes | Python: `Send_Respond_Device_Info()`

Asks for the driver's identity: hardware version, batch date and serial number from the EEPROM, and the firmware release (from firmware 105 on this is the build's own version; before, a value from the EEPROM).

The frame (7 bytes)

Byte	Bit	Field	Type	Meaning
0		HARDWARE_VERSION	unsigned 8-bit	PCB revision
1		BATCH_DATE	unsigned 8-bit	production batch
2		SOFTWARE_VERSION	unsigned 8-bit	firmware release, e.g. 107
3-6		SERIAL_NUMBER	unsigned 32-bit	serial number

The driver answers with **Send_Respond_Device_Info**.

Send_Respond_State_of_Errors

ID 26 | host → driver as a remote frame, driver → host as a data frame | remote frame, 0 bytes | Python: `Send_Respond_State_of_Errors()`

Asks for every error flag and the calibrated, activated and watchdog states.

The frame (2 bytes)

Byte	Bit	Field	Type	Meaning
0	7	error	bit	any error is active
0	6	temperature_error	bit	motor over the temperature limit
0	5	encoder_error	bit	magnet not detected or misaligned
0	4	vbus_error	bit	supply voltage outside the limits
0	3	driver_error	bit	the MOSFET driver reports undervoltage, thermal shutdown or overcurrent
0	2	velocity_error	bit	speed over the limit
0	1	current_error	bit	reserved; never set by this firmware
0	0	estop_error	bit	Send_ESTOP was received
1	7	calibrated	bit	the motor is calibrated
1	6	activated	bit	the controller is activated
1	5	watchdog_error	bit	the watchdog ran out

The driver answers with **Send_Respond_State_of_Errors**.

Respond_Heartbeat

ID 9 | driver → host | 0 bytes

The frame the driver sends on its own at the heartbeat interval (Send_Heartbeat_Setup); empty. Off by default. Any other frame the driver sends resets the heartbeat timer, so a driver that answers motion commands at a high rate never sends a heartbeat.

The frame (0 bytes)

Byte	Bit	Field	Type	Meaning

UART commands

UART commands

The commands a gripper needs over the driver's serial port, for the firmware release this site describes. The kind, the argument type, the mode a command enters and what it prints are read from the firmware; the descriptions are written by hand. Over UART says where the port is and how it is used.

Syntax

```
#<Name> [number]
```

- A **Set/Get** command sets the value when a number follows and prints the current value either way.
- A **Get** command only prints.
- An **Action** command does something; a number after it is ignored.
- Names are case-sensitive, at most 19 characters; the number at most 19; the line ends with LF (CR before it is fine). 256000 baud, 8N1, 3.3 V.
- A name the parser does not know is answered `Unknown command`.

The commands a gripper needs

Command	Kind	Argument	Group
`#Cal`	Action		Driver setup
`#Calibrated`	Set/Get	int	Status and utility
`#CANID`	Set/Get	int	Driver setup
`#Cbw`	Set/Get	float	Driver setup
`#Clear`	Action		Status and utility
`#Default`	Action		Driver setup
`#Error`	Get		Status and utility
`#Gripact`	Set/Get	int	Gripper
`#Gripcal`	Action		Gripper

Command	Kind	Argument	Group
`#Gripcur`	Set/Get	int	Gripper
`#Gripinfo`	Get		Gripper
`#Gripper`	Set/Get	int	Gripper
`#Grippos`	Set/Get	int	Gripper
`#Gripstop`	Set/Get	int	Gripper
`#Gripvel`	Set/Get	int	Gripper
`#Help`	Action		Status and utility
`#Ilim`	Set/Get	int	Driver setup
`#Info`	Get		Status and utility
`#Kiiq`	Set/Get	float	Driver setup
`#Kpiq`	Set/Get	float	Driver setup
`#Kt`	Set/Get	float	Driver setup
`#LED`	Set/Get	int	Status and utility
`#Magnet`	Get		Status and utility
`#Mode`	Get		Status and utility
`#Reset`	Action		Status and utility
`#Save`	Action		Driver setup
`#Temp`	Get		Status and utility
`#Term`	Set/Get	int	Driver setup
`#Tlim`	Set/Get	int	Driver setup

Command	Kind	Argument	Group
`#Vbus`	Get		Status and utility

The driver's other commands (motion, gains, the calibration settings) are in the Spectral UART reference.

By group

- Gripper: gripper mode, the gripper calibration, the setpoints, the state
- Driver setup: the defaults, the motor parameters and calibration, the gains, the thermistor, the node ID, saving
- Status and utility: identity, errors, temperature, help

Gripper

Command	Kind	Argument	What it does	Response
`#Gripper`	Set/Get	int: 0 or 1	Whether this driver is a gripper. A gripper has it set and saved; `#Default` clears it, so set it again after restoring the defaults.	the flag
`#Gripcal`	Action		Starts the gripper calibration (the jaws find their end stops). Clear any error first; until `#Gripinfo` reports it calibrated, send no `#Grippos` or `#Clear`, which end it. Enters **gripper calibration** mode.	`Gripper calibration !`
`#Gripinfo`	Get		Prints the gripper state: gripper mode, calibrated, activated, the position, speed and current setpoints, the E-stop and action bits, the object detection, and the end stops in encoder ticks.	the report
`#Gripstop`	Set/Get	int: 0 or 1	The gripper E-stop bit. While it is 1, the E-stop error is raised again at every cycle: set it to 0 before `#Clear`.	the bit
`#Gripact`	Set/Get	int: 0 or 1	The activate bit.	the bit
`#Gripcur`	Set/Get	int: current in mA	The current limit of gripper moves	the setpoint
`#Gripvel`	Set/Get	int: 0 to 255	The speed setpoint of gripper moves.	the setpoint
`#Grippos`	Set/Get	int: 0 (open) to 255 (closed)	The position setpoint; also activates the gripper, sets the action bit and goes to gripper mode. Enters **gripper** mode.	the current position

Driver setup

The driver commands a self-built gripper is configured with; Driver configuration is the procedure.

Command	Kind	Argument	What it does	Response
`#Kpiq`	Set/Get	float: gain	Kp of the Iq current loop; also sets Kpid.	the current value
`#Kiiq`	Set/Get	float: gain	Ki of the Iq current loop; also sets Kiid.	the current value
`#Kt`	Set/Get	float: Nm/A	The torque constant, found by calibration.	the current value
`#Cal`	Action		Runs the calibration routine (see Calibration): it measures R and L, finds the pole pairs, the phase order and the angle offset, and measures Kt, KV and the flux linkage.	the calibration log
`#Save`	Action		Writes the configuration to the EEPROM. Do it after calibration and after tuning.	`Save`
`#Ilim`	Set/Get	int: current in mA	The driver's Iq current limit outside gripper mode; the firmware clamps it to 2800 mA. Gripper moves use Gripcur instead.	the current limit
`#Tlim`	Set/Get	int: temperature in °C	The temperature at which the temperature error is raised; 75 by default.	`Temp limit N`
`#Default`	Action		Writes the factory defaults to the EEPROM, including the serial number, hardware version and batch date, and clears the gripper flag, heartbeat and watchdog. Back those up first.	`Loaded default config!`
`#Term`	Set/Get	int: 0 or 1	Enables the thermistor measurement and the temperature error (off by default).	the setting

Command	Kind	Argument	What it does	Response
`#Cbw`	Set/Get	float: bandwidth in Hz	The current loop bandwidth calibration uses to compute Kpiq and Kiiq.	the bandwidth
`#CANID`	Set/Get	int: 0 to 13	The CAN node ID. Use 0 to 13: the firmware accepts 14 and 15, but the bootloader reserves them and a driver on one of those cannot be updated over CAN. Save keeps it over a power cycle and also writes it into the bootloader's board ID.	the ID, or `Invalid CAN ID`

Status and utility

Command	Kind	Argument	What it does	Response
`#Magnet`	Get		Reads the encoder's magnet status.	`Magnet Good` or `Magnet Bad`
`#Reset`	Action		Resets the driver. From firmware 105 the reset lands in the bootloader, which waits 2 s for a flashing host and then starts the firmware again.	nothing
`#Temp`	Get		Prints the thermistor temperature.	°C, or `Temp Off` while the thermistor is disabled
`#Mode`	Get		Prints the control mode.	`Idle mode`, `Position control`, `Speed control`, `Current control`, `PD control`, `Open loop`, `Gripper mode`, `Gripper calib mode`
`#Vbus`	Get		Prints the supply voltage.	mV
`#Error`	Get		Prints which errors are active.	`No errors!` or `In error state!` followed by one line per active error
`#Clear`	Action		Clears the errors and goes to idle, and ends a running gripper calibration. An error whose cause is still present comes straight back, the gripper's E-stop too while `#Gripstop` is 1; a calibration error is cleared only by calibrating (or `#Calibrated 1`). Enters **idle** mode.	`Clear errors`

Command	Kind	Argument	What it does	Response
`#Help`	Action		Prints a short command reference. Added in firmware 106.	the reference
`#Info`	Get		Prints the driver's identity and state: hardware version, batch date and firmware version, the ISR time, the CAN ID, the motor calibration, the thermistor and temperature, the supply voltage and the motor parameters (phase direction and order, pole pairs, R, L, Kt, KV). The serial number is printed by `#SERNUM`, the errors by `#Error`, gripper mode by `#Gripper`.	the report
`#Calibrated`	Set/Get	int: 0 or 1	The calibrated flag. `#Calibrated 1` clears a calibration error without calibrating; the motor may still be uncalibrated.	the flag
`#LED`	Set/Get	int: 0 or 1	Turns the status LED on or off.	the LED state

Errors and status bits

The gripper's state byte

Byte 3 of Respond_Gripper_data_pack, the answer to every gripper command, bits numbered from the most significant:

Bit	Flag	Meaning
7	`activated`	the activate flag is set
6	`action_status`	the action bit of the last gripper command, as received (`#Grippos` sets it to 1); it stays 1 while the gripper holds, and the calibration does not change it. The state of a move is in `object_detection`
5, 4	`object_detection`	bit 5 is the low bit, bit 4 the high bit: 0 moving, 1 object found while closing, 2 object found while opening, 3 at the position
3	`temperature_error`	the thermistor in the windings is above the limit
2	`timeout_error`	the driver's CAN watchdog ran out
1	`estop_error`	the E-stop bit of a command was received
0	`calibrated`	the gripper calibration is done

The Python library's attributes are `gripper_activated`, `gripper_action_status`, `gripper_object_detection`, `gripper_temperature_error`, `gripper_timeout_error`, `gripper_estop_error` and `gripper_calibrated`; library version 1.20 assembles `gripper_object_detection` with its two bits swapped (1 and 2 exchanged).

The driver's error flags

The flags as the firmware keeps them, the line `#Error` prints for each, and where each sits in the two bytes of Send_Respond_State_of_Errors (bits numbered from the most significant).

Flag	`#Error` prints	Byte, bit	Raised when
`Error`	`In error state!`	0, 7	any of the flags below is set (also the error bit of every CAN

Flag	`#Error` prints	Byte, bit	Raised when
			identifier)
`temperature_error`	`Temp error`	0, 6	the thermistor reads above `#Tlim` (75 °C)
`encoder_error`	`Encoder error`	0, 5	the driver's encoder does not see its magnet
`Vbus_error`	`Vbus error`	0, 4	the supply is below 9 V or above `#Vmax` (29 V after `#Default`)
`Driver_error`	`Driver error (undervoltage, thermal shutdown, or overcurrent)`	0, 3	the driver stage's fault output is active
`Velocity_error`	`Velocity error`	0, 2	the motor speed exceeds the driver's limit
`Current_error`	`Current error`	0, 1	never in this firmware; reserved
`ESTOP_error`	`Received ESTOP command`	0, 0	Send_ESTOP was received
`Calibrated` (0)	`Not calibrated`	1, 7 (1 = calibrated)	the driver's motor calibration is missing
`Activated`	—	1, 6	the activated flag, not an error
`Watchdog_error`	`Watchdog error`	1, 5	the watchdog time passed without a valid CAN frame

The gripper's `timeout_error` is the driver's `Watchdog_error`; the gripper's `temperature_error` is the driver's. The gripper's `estop_error` is its own, raised by the E-stop bit of the gripper command, and is distinct from the driver's `ESTOP_error` raised by Send_ESTOP.

Clearing

Send_Clear_Error and `#Clear` clear every flag, the gripper's E-stop included, and go to idle; a flag whose cause persists is set again at once. The gripper's `estop_error` is raised again at every cycle while the E-stop bit of the last gripper command (or `#Gripstop`) is 1: send a gripper command with the bit at 0 first, then clear. The driver's `Calibrated` flag is set only by the motor calibration (`#Cal`).

The LED

The driver's status LED is inside the housing: three short flashes and a pause in normal operation, flashing every 0.5 s during the motor calibration, on steadily in error mode or while the motor is not calibrated.

Python library

The `Spectral-BLDC` package (Spectral-BLDC-Python, `pip install Spectral-BLDC`) has one method per CAN message in `Spectral_BLDC.SpectralCAN`, and `Spectral_BLDC.CanCommunication` for the bus. With Python shows how they are used with the gripper.

Calls and messages

The calls for the messages this reference describes; the library has one for every message of the driver.

Library call	Sends	Answered with	Reference
`Send_gripper_data_pack()`	Send_gripper_data_pack (ID 61)	Respond_Gripper_data_pack	Gripper
`Send_gripper_calibrate()`	Send_gripper_calibrate (ID 62)	—	Gripper
`Send_CAN_ID()`	Send_CAN_ID (ID 11)	—	System
`Send_ESTOP()`	Send_ESTOP (ID 0)	—	System
`Send_Idle()`	Send_Idle (ID 12)	—	System
`Send_Save_config()`	Send_Save_config (ID 13)	—	System
`Send_Reset()`	Send_Reset (ID 14)	—	System
`Send_Clear_Error()`	Send_Clear_Error (ID 1)	—	System
`Send_Watchdog()`	Send_Watchdog (ID 15)	—	System
`Send_Heartbeat_Setup()`	Send_Heartbeat_Setup (ID 30)	—	System
`Send_Respond_Ping()`	Send_Respond_Ping (ID 10)	Send_Respond_Ping (data frame)	Status requests

Library call	Sends	Answered with	Reference
`Send_Respond_Temperature()`	Send_Respond_Temperature (ID 23)	Send_Respond_Temperature (data frame)	Status requests
`Send_Respond_Voltage()`	Send_Respond_Voltage (ID 24)	Send_Respond_Voltage (data frame)	Status requests
`Send_Respond_Device_Info()`	Send_Respond_Device_Info (ID 25)	Send_Respond_Device_Info (data frame)	Status requests
`Send_Respond_State_of_Errors()`	Send_Respond_State_of_Errors (ID 26)	Send_Respond_State_of_Errors (data frame)	Status requests

`Send_gripper_data_pack(position, speed, current, activate, action, estop, release_direction)` sends the 5-byte move; called without arguments it sends the empty frame that only asks for the state.

Receiving

`CanCommunication.receive_can_messages(timeout)` returns `(message, ids)`: the raw `python-can` message and a tuple `ids` with `node_id`, `command_id` and `error_bit` from the identifier, or `(None, None)` after the timeout. `SpectralCAN.UnpackData(message, ids)` decodes the message into the object's attributes:

Message	Attributes
Respond_Gripper_data_pack	`gripper_position`, `gripper_current`, `gripper_activated`, `gripper_action_status`, `gripper_object_detection`, `gripper_temperature_error`, `gripper_timeout_error`, `gripper_estop_error`, `gripper_calibrated`
Send_Respond_State_of_Errors	`error`, `temperature_error`, `encoder_error`, `vbus_error`, `driver_error`, `velocity_error`, `current_error`, `estop_error`, `calibrated`, `activated`, `watchdog_error`
Send_Respond_Temperature	`temperature`
Send_Respond_Voltage	`voltage`
Send_Respond_Device_Info	`HARDWARE_VERSION`, `BATCH_DATE`, `SOFTWARE_VERSION`, `SERIAL_NUMBER`

Object detection in the library

Library version 1.20 assembles `gripper_object_detection` with the two bits swapped relative to the firmware, so it reports 1 where the gripper sends 2 (object detected while opening) and 2 where it sends 1 (object detected while closing); 0 and 3 are unaffected. Until the library is fixed, swap the two values in your program, or read the bits yourself from the frame.

Guides

Guides

Worked examples around the gripper.

- Basic control in Python: the shortest program that calibrates the gripper and opens and closes it, from the product repository's README.
- Teleoperation with two grippers: one gripper follows the other and pushes back what it feels, over one CAN bus and a few lines of Python.

Building a gripper yourself is in the manual, the Build guide. More examples, for the driver as much as for the gripper, are in the Spectral-BLDC-Python examples and in the Spectral guides.

Basic control in Python

The shortest program that drives the gripper: it connects through the CANvas adapter, clears the errors, calibrates, then closes and opens every three seconds and prints what the gripper answers. It is the example from the product repository's README; `pip install Spectral-BLDC` is the only requirement. Change `COM20` to your adapter's port (`/dev/ttyACM0` on Linux) and `node_id` to the gripper's CAN ID if you changed it from 0.

> **Danger**
>
> 700 mA closes the jaws with real force, about 38 N. Keep fingers out of the jaws, and start with a lower current if the gripper is new to you.

```
import Spectral_BLDC as Spectral
import time

Communication1 = Spectral.CanCommunication(bustype='slcan', channel='COM20', bitrate=1000000)
Motor1 = Spectral.SpectralCAN(node_id=0, communication=Communication1)

Motor1.Send_Clear_Error()
Motor1.Send_gripper_calibrate()
time.sleep(4)

temp_var = 0
while True:
   if temp_var == 0:
      Motor1.Send_gripper_data_pack(50,100,700,1,1,0,0)
      temp_var = 1
   elif temp_var == 1:
      Motor1.Send_gripper_data_pack(240,100,700,1,1,0,0)
      temp_var = 0

   message, UnpackedMessageID = Communication1.receive_can_messages(timeout=0.2)

   if message is not None:
      print(f"Message is: {message}")
      print(f"Node ID is : {UnpackedMessageID.node_id}")
      print(f"Message ID is: {UnpackedMessageID.command_id}")
      print(f"Error bit is: {UnpackedMessageID.error_bit}")
      print(f"Timestamp is: {message.timestamp}")

   time.sleep(3)
```

Each answer printed is the one to the command just sent, so it still shows the state from before that move. The arguments of `Send_gripper_data_pack`, in order: the position (0 open, 255 closed), the speed (0 to 255), the current limit in mA (the force), then the activate bit, the action bit, the E-stop bit and the release direction. With Python explains the library's calls, How the gripper works the state the gripper answers with, and Force and speed what the current means in newtons.

Teleoperation with two grippers

Two grippers on one CAN bus, coupled so that each follows the other: squeeze one by hand and the other closes; when the other meets an object, the one in your hand pushes back with the force it feels. A hundred lines of Python, and the basis of a teleoperation setup with force feedback.

This guide does not use the gripper mode. Both drivers run in **current (torque) mode**, the driver's own motion command, and the PC closes a position loop between them at a fixed rate. The gripper mode's position, speed and object detection are not involved; the drivers are plain torque sources here.

What you need

- Two grippers, calibrated as motors (a bought gripper is), with different node IDs: 0 and 1 (`#CANID 1` , `#Save` on one of them over UART).
- Both on one bus with the CANvas adapter, 24 V on both, termination at the two ends.
- `pip install Spectral-BLDC SourceRoboticsToolbox` .

> **Warning**
>
> In current mode the gripper has no end-stop knowledge and no object detection: it pushes with whatever current the loop commands. Set the driver's current limit to the motor's rating (`#Ilim`) on both grippers, start with low gains, and keep fingers out of the jaws.

The idea

Each driver is commanded a current proportional to the difference between the two positions, plus a damping term on the speed difference and a little damping of its own motion:

$$I_0 = K_p\,(p_1 - p_0) + K_d\,(v_1 - v_0) - K_k\,v_0, \qquad I_1 = K_p\,(p_0 - p_1) + K_d\,(v_0 - v_1) - K_k\,v_1$$

p is the jaw position of each gripper in the same units, v its speed. When the two positions agree, both currents are zero and the grippers rest. Move one by hand and the difference drives the other after it; block the other, and the difference drives the one in your hand back with the same current, which is the force you feel. K_p is the stiffness of the virtual spring between them, K_d its damping.

The two drivers' encoders do not read the same number at the same jaw position, so each position is referred to a reference reading taken once with both grippers in the same position; the Source Robotics Toolbox's `Joint` does that bookkeeping and unwraps the encoder over its 16384-tick turns.

The code

```
import time
import numpy as np
import Spectral_BLDC as Spectral
import SourceRoboticsToolbox as Toolbox

INTERVAL_S = 0.002                # the loop rate: 500 Hz
REFERENCE = [10440, 13555]        # each gripper's raw encoder reading with both jaws in the
same position

Kp, Kd, Kk = 650, 0.0055, 0.0003   # stiffness, damping between the two, damping of each

bus = Spectral.CanCommunication(bustype='slcan', channel='COM123', bitrate=1000000)
motors = [Spectral.SpectralCAN(node_id=i, communication=bus) for i in (0, 1)]
joints = [Toolbox.Joint(encoder_resolution=14, master_position=REFERENCE[i], gear_ratio=1,
offset=0, dir=0)
          for i in (0, 1)]

position = np.zeros(2)            # each gripper's position, referred to its reference
sector_known = [False, False]
current = [0, 0]

while True:
    t0 = time.perf_counter()
    for i in (0, 1):              # command the currents; each answer carries position and speed
        motors[i].Send_data_pack_1(Position=None, Speed=None, Current=int(current[i]))
    for _ in (0, 1):
        message, ids = bus.receive_can_messages(timeout=0.001)
        if ids is None:
            continue
        i = ids[0]
        motors[i].UnpackData(message, ids)
        if not sector_known[i]:
            joints[i].determine_sector(motors[i].position)
            sector_known[i] = True
        position[i] = joints[i].get_joint_position(motors[i].position)

    current[0] = Kp * (position[1] - position[0]) + Kd * (motors[1].speed - motors[0].speed) - Kk
* motors[0].speed
    current[1] = Kp * (position[0] - position[1]) + Kd * (motors[0].speed - motors[1].speed) - Kk
* motors[1].speed

    time.sleep(max(0.0, INTERVAL_S - (time.perf_counter() - t0)))
```

The reference readings: with both grippers powered and their jaws set by hand to the same opening, read each driver's position as the loop receives it (print `motors[i].position` with both currents at 0) and put the two numbers in `REFERENCE`.

Tuning

Start with `Kp` at a tenth of the value above and raise it until the follower tracks the leader without lagging; raise `Kd` if the pair oscillates. The current the formula produces is in mA and is clamped by each driver's `#Ilim`, so the motor's rating is the ceiling of the force feedback. A fixed loop rate matters: a loop that runs at an uneven rate makes the damping terms noisy.

The full example, with a timer for a precise loop rate, is gripper_teleop_PC_calc.py in the Python library's examples, and the bilateral teleoperation guide on the Spectral Micro site explains the control law with a joint instead of a gripper.

Support

Getting help

- **Questions and discussion**: the Source Robotics Discord.
- **A problem with the gripper's hardware or its files**: an issue in the product repository; see Report a problem for what to include.
- **A problem with the firmware**: an issue in the driver's repository; with the Python library, in Spectral-BLDC-Python; with the ROS 2 driver, in its repository.
- **A mistake on a page of this site**: every page ends with a link to edit it and one to report it; see Contributing.
- **Orders and hardware**: source-robotics.com.

Before you ask, Troubleshooting goes through the common symptoms, and the gripper's status tells most of what anyone will ask you for.

Report a problem

Open an issue in the product repository for the gripper and its files, in the driver's repository for the firmware, or in Spectral-BLDC-Python for the Python library. Include:

1. **The gripper**: bought or self-built, the hardware version, the fingers.
2. **The firmware release** the gripper runs: Send_Respond_Device_Info, or `#Info` over UART, which also prints the driver's configuration; add what `#Error` prints.
3. **The gripper's status** when the problem happens: the state and error bits of Respond_Gripper_data_pack, or `#Gripinfo`.
4. **What you did and what happened**: the commands, in order, with their position, speed and current, and the answers.
5. **How it is connected**: the robot or the adapter, the cable, the termination; a photo helps more than a description.

For a problem with this documentation, the *Report it* link at the end of the page opens an issue in the documentation repository with the page already named.

Contributing

This documentation lives in Source-Robotics/SSG48-gripper-docs. A small fix can be made from the browser: the *Edit it on GitHub* link at the end of every page opens the page's source, and GitHub turns your change into a pull request. For anything larger, open an issue first.

How the site is built

- The pages are Markdown, built with MkDocs and the Material theme; the repository's `README.md` says how to preview it locally.
- The site describes one release of the Spectral firmware, named in `versions.yml`. The command IDs, frame lengths, the UART command list and what each prints in the API reference are read from the firmware's source at that release's tag by `tools/sync_firmware.py`; do not edit those facts in the pages. The descriptions of the commands are in `docs/includes/can_commands.yml` and `uart_commands.yml`, which list only the commands a gripper needs.
- Every push runs a strict build (a broken link fails it) and the tool tests. A push to `main` publishes the site with mike, under the gripper release named by `docs_version` in `versions.yml`.

House rules

- Facts come from the firmware and the product repository. When a page and the firmware disagree, the firmware is right; fix the page, or report the firmware.
- Spell commands exactly as the firmware does, even where the spelling looks wrong.
- Write for someone who has the gripper in front of them and wants to do one thing: say what to do, in order, then why.
- Three kinds of boxes: a note, a notice (damage) and a danger (injury). Use them for those, not for emphasis.
- Images: under 2400 px on the long side, with a caption that says what the reader should see.

Licence

Text and images: CC BY-SA 4.0: copy, adapt and share them, also commercially, if you credit Source Robotics and share your changes under the same licence. Code: the MIT licence, which covers the code examples in the pages and the site's own code. The gripper's hardware design is under the Apache 2.0 licence in the product repository. The Source Robotics name, the product names and the logos are not covered by any of these. By contributing you agree that your contribution is published under these licences.