MSG gripper

Description	Documentation for the MSG compliant AI stepper gripper by Source Robotics.
Repository	https://github.com/Source-Robotics/MSG-gripper-docs
Copyright	© 2026 Source Robotics d.o.o. · Text and images CC BY-SA 4.0 · Code examples MIT

Table of Contents

- MSG Gripper Manual
  - MSG Gripper Manual
  - Before you start
    - Quick start
    - About this manual
  - Chapters
    - 1. Overview
      - Overview
      - The gripper at a glance
      - Variants
      - 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

MSG Gripper Manual



Document number: SR-5003

MSG gripper release: 1.2

Firmware: STEPFOC 108

Revision: 2026.B

Published: 2026-11-04

Applies to: MSG gripper release 1.2 with the STEPFOC driver PCB v1, STEPFOC firmware 108

Original instructions

Quick start | About this manual

Source Robotics d.o.o. publishes this document for the MSG compliant AI stepper 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 STEPFOC 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 MSG 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.

Made for teleoperation and robot learning. We designed the MSG specifically for teleoperation, VLA (vision-language-action) training and physical intelligence. The camera on top of the body is placed with care: it looks down between the jaws at an angle that keeps both jaws, and the object between them, in view, the picture a teleoperator and a learning model need.

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

API reference

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.

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 MSG 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 stepper 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 stepper	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.

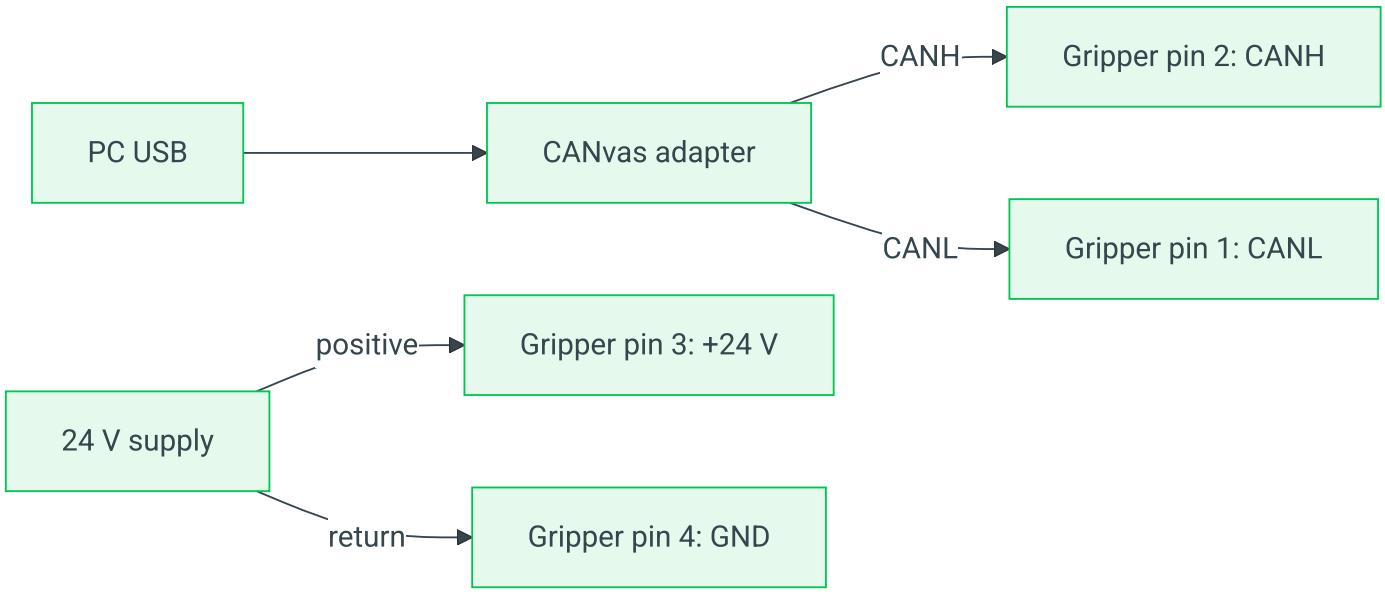
Check

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**.

The illustrated Motor GUI guide shows the panel.

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.

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 24 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.

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

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.

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 MSG Gripper Manual covers the MSG compliant AI stepper 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 STEPFOC stepper driver in gripper mode. The driver itself, from its full command set to its calibration and its bootloader, is documented on the STEPFOC site; this manual links there instead of repeating it.

Finding your way

Four tabs: **MSG 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 STEPFOC 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
MSG Gripper Manual	SR-5003	the MSG compliant AI stepper 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 MSG gripper release 1.2 running STEPFOC firmware 108. 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/STEPFOC-docs/. The SSG-48 gripper, the MSG's predecessor on the Spectral Micro driver, is at docs.source-robotics.com/SSG48-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 STEPFOC 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 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 1.2, with the files as downloads on the release; dimension drawings and what changes per variant; data for robot integration (frame, mass, inertia, TCP); known issues; the downloads with their versions; safety for clothing, hair and eyes, a caught finger, static and cleaning; check boxes after each step and Windows, Linux and macOS tabs; numbered figure captions; the 380 g variant
2026.A	2026-11-03	First version in this form: the manual chapters, the generated CAN and UART reference; gripper release 1.1, firmware 108

Chapters

1. Overview

Overview

What the MSG gripper is and how it works, the variants it comes in, 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
- Variants
- What you need

The gripper at a glance

The MSG is a two-jaw parallel gripper driven by a NEMA 17 stepper under field-oriented control. The STEPFOC driver on the back of the stepper 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. The jaws stop on an object, hold it with that force, and the driver reports that an object was found. We designed the MSG specifically for teleoperation, VLA (vision-language-action) training and physical intelligence. A camera on top of the body looks down between the jaws, from a place chosen for a clear view of both jaws and the object between them.



Figure 1.1: The MSG gripper with the 150 mm rail and the camera mount.

The parts

Part	What it does
NEMA 17 stepper motor, 21.5, 40 or 60 mm long	the actuator; its length sets the grip force: about 31, 50 or 76 N at its maximum current
STEPFOC driver on the back of the stepper	field-oriented control of the stepper, the gripper state machine, the CAN and UART interfaces
Pinion on the motor shaft, 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, 150 or 200 mm	carries the jaws; its length sets the stroke

Part	What it does
Jaws	printed, in several shapes, with a soft TPU pad or solid; yours to adapt
Thermistor on the motor housing	temperature protection: the gripper stops at 75 °C
M8 4-pin connector	24 V and the CAN bus, one cable to the robot
Camera mount	an OV9732 USB camera module looking between the jaws
Coupler plate	mounts the gripper on an 8 mm shaft, or on the PAROL6 wrist with its own cover

The driver's status LED and its UART port are inside the housing; the CAN termination switch is reachable through the stepper cover (CAN termination).

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 housing: for configuration, and for a self-built gripper. See Over UART.

Variants

The gripper is modular in two ways: the length of the linear rail sets the stroke, the length of the stepper sets the force. Any rail goes with any stepper, and the body comes in black or grey; the body, the jaw carriers and the stepper cover are printed for the chosen combination.

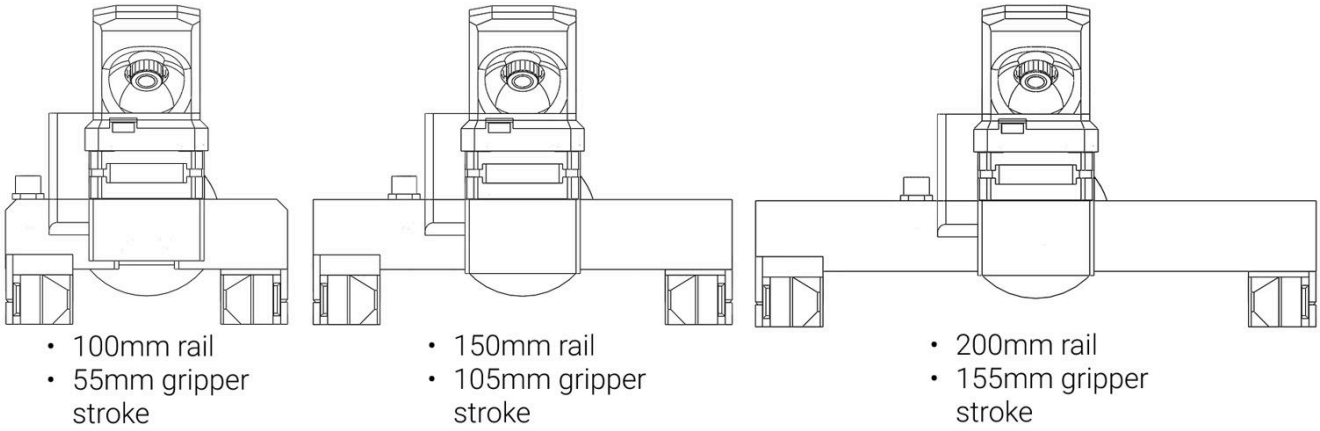


Figure 1.2: The three rail lengths and the stroke each gives.

Black

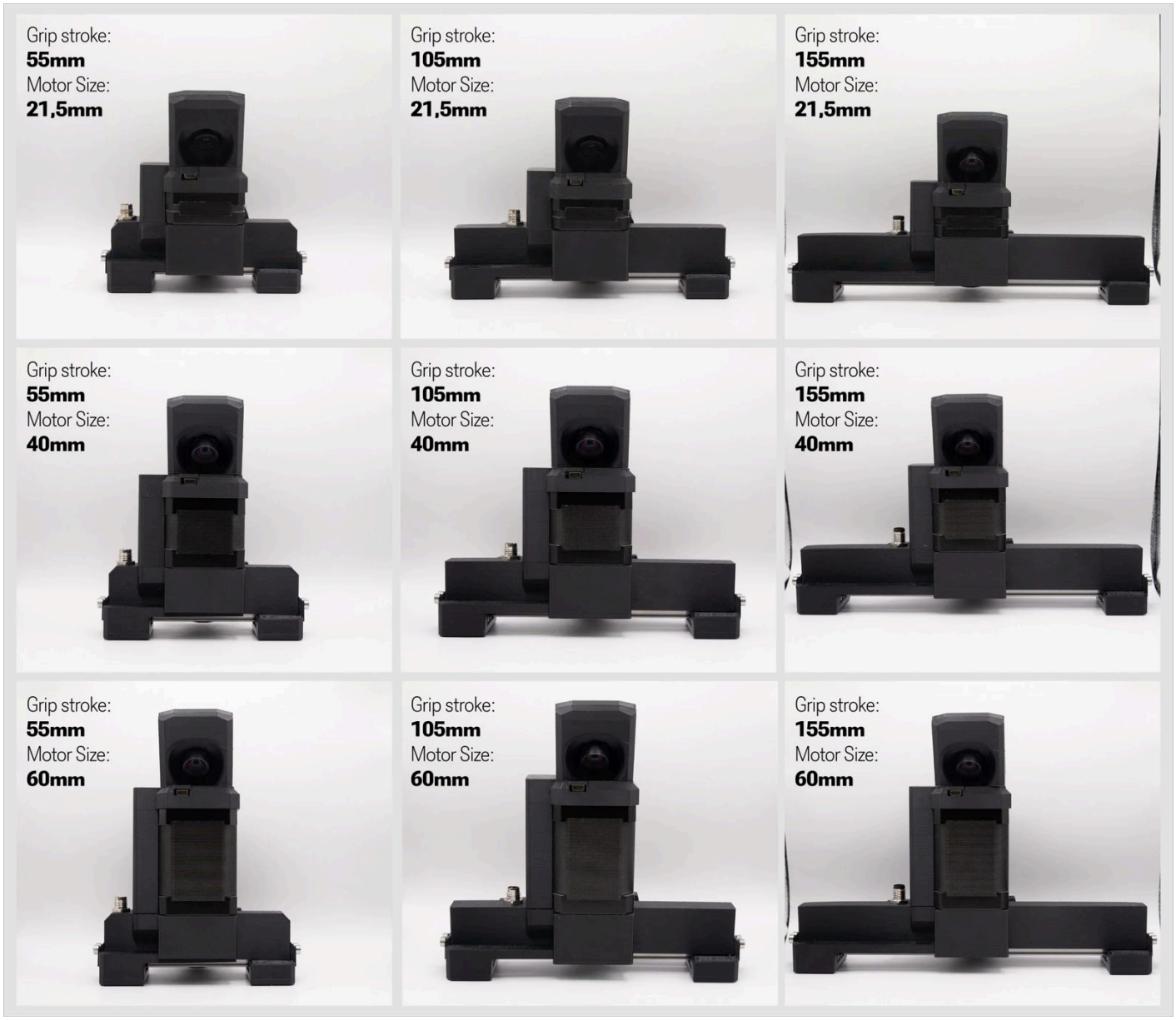
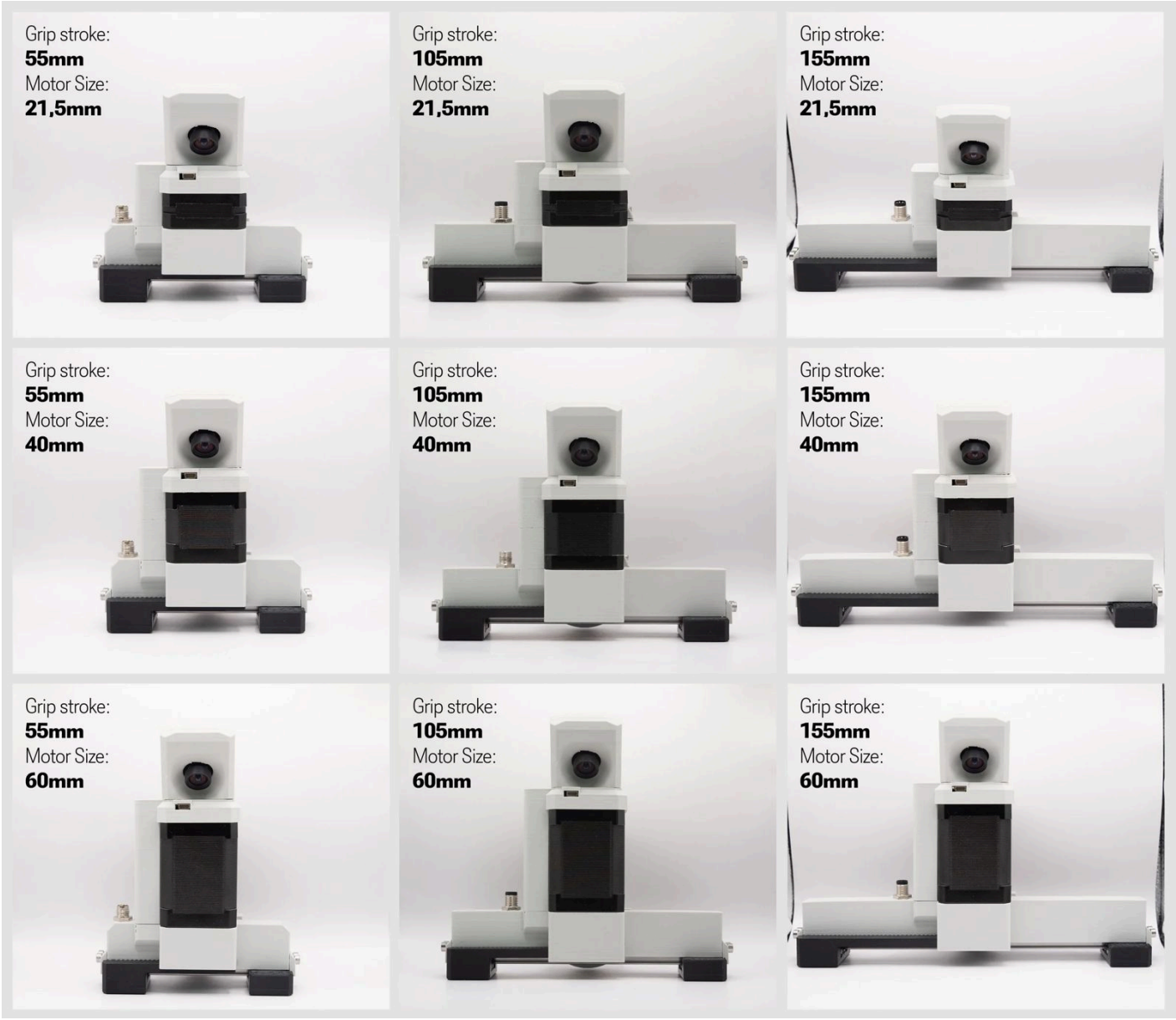


Figure 1.3: The nine combinations of rail and stepper, black body: 55, 105 and 155 mm of stroke across, the 21.5, 40 and 60 mm steppers down.

Grey



The same nine combinations, grey body.

Rail length and stroke

Rail	Stroke	Printed parts
100 mm	55 mm	`MSG_Gripper_body_100mm`, `gripper_jaw_100mm_L/R`
150 mm	105 mm	`MSG_Gripper_body_150mm`, `gripper_jaw_150mm_L/R`
200 mm	155 mm	`MSG_Gripper_body_200mm`, `gripper_jaw_200mm_L/R`

The stroke is the distance between the jaw carriers from fully open to fully closed; the opening between the jaw faces depends on the jaws you fit (Jaws). A longer rail makes the gripper wider and a little heavier and does not change the force.

Stepper length and force

Stepper	Motor	Maximum current	Assumed torque constant	Ideal combined force at the maximum current	Printed parts
21.5 mm	17HS08-1004S	1.5 A	0.17 Nm/A	about 31 N	`Stepper_cover_21_5mm`, `Stepper_spacer_21_5`
40 mm	17HS16-2004S1	2.0 A	0.21 Nm/A	about 50 N	`Stepper_cover_40mm`
60 mm	17HS24-2104S	2.1 A	0.30 Nm/A	about 76 N	`Stepper_cover_60mm`

A longer stepper gives more torque and so more grip force, at the price of weight and length behind the gripper. The force is proportional to the current limit you command, with the slope of the stepper's torque constant; the curves and the formula are in Force and speed. The current limit must stay within the stepper's maximum.

Grasps

The gripper grips from the outside (the jaws close on the object) and from the inside (the jaws open inside a hollow object). Both are the same move command with a different target; object detection reports *while closing* for the first and *while opening* for the second.

The camera

The camera mount on top of the body carries an OV9732 USB camera module (100° version in the BOM), looking down between the jaws. Its place and angle were chosen with care: the camera sees both jaws and the object between them, the view that teleoperation and VLA training need. It is a plain USB camera with its own cable, independent of the gripper's electronics; the mount and its lid are printed parts (`camera extension`, `camera_lid`), and a gripper without a camera simply leaves them off.



Figure 1.4: The gripper on a PAROL6 and what its camera sees.

What you need

The gripper

The MSG gripper is sold assembled, calibrated and configured in the Source Robotics shop, in the rail and stepper combination you choose, in black or grey. It ships with the current STEPFOC firmware and the bootloader installed, its driver in gripper mode and its stepper calibrated; the gripper calibration (finding the end stops) runs after every power-up and is part of normal use.

Or build it. The MSG is open source, and that is important to us: the CAD and print files, the bill of materials, the assembly 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 the rail, stepper and colour you ordered, with the camera module when ordered. **Not in the box:** the jaws, which you print from the repository or buy in the shop; the cable, sold separately; 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

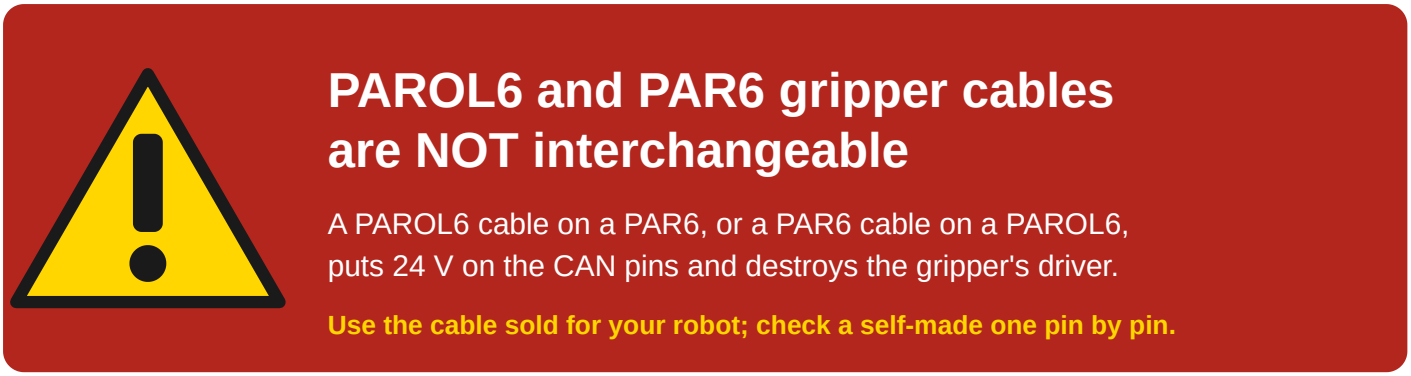


Figure 1.5: PAROL6 and PAR6 gripper cables are not interchangeable.

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 driver. 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 MSG and the SSG-48; see Motor GUI. The earlier SSG-gripper-GUI, a Python program, is legacy.
- **Python**: `pip install Spectral-BLDC`, the library the ROS 2 driver is built on; see With Python.
- **ROS 2**: the msg_gripper_ros2 package; 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 STEPFOC starter kit: it has this adapter and the rest of what the driver's setup needs (Kits and the shop).

Optional

- **Jaws** for your objects: the printed jaws in the repository (solid, hollow, or with a TPU pad), sponge tape for the gripping faces, or your own design; see Jaws.
- **A camera**: an OV9732 USB module for the camera mount; see Variants.

Safety

The MSG 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. With the current limit at the stepper's rating they close with tens of newtons, enough to crush a finger, and at full speed they travel the whole stroke in well under 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 to 500 mA, speed 30. Raise them step by step once the application behaves, and never above the stepper's rated current (Force and speed).

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.

Clothing, hair and eyes

The jaws, and the open rack and pinion behind them, catch more than fingers: long hair, loose sleeves, gloves, lanyards, rings and bracelets. Tie back long hair and take those off before you work at a powered gripper, and keep them out of the jaws' travel while it calibrates or moves. Wear safety glasses when a gripped part can break, slip or be thrown.

If a finger is caught

The E-stop does not open the jaws

The gripper's E-stop bit and Send_ESTOP stop the jaws where they are; they do not open them, and a powered gripper holds its grip. To free a finger, switch off the gripper's supply: without current the stepper holds nothing beyond its own friction, and the jaws can be pushed apart by hand. A gripper without an error opens with a move to position 0 as well (**Open** in Motor GUI). Keep the supply's switch within reach while you test.

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.

Temperature

A thermistor on the motor housing measures the motor's temperature. 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: for a long hold, use the lowest current that holds the object. A self-built gripper without the thermistor on its motor 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 stepper without current holds nothing beyond its own 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 MSG 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, the dimension drawings and the CAD files, and the data a robot's software needs: the tool centre point, the mass and the inertia.

In this chapter

- Specifications
- Force and speed
- Connector and pinout
- Dimensions and CAD
- Data for robot integration

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	up to 1.5 A (21.5 mm stepper), 2.0 A (40 mm), 2.1 A (60 mm); set per move as the grip force
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, reachable through the stepper cover
Service port	the driver's UART, 3.3 V, 256000 baud, inside the housing
Driver	STEPFOC stepper controller, field-oriented control, 14-bit encoder on the motor shaft
Temperature protection	thermistor on the motor housing, 75 °C limit (changeable with `#Tlim`)

Mechanical

	Value
Stroke	55 mm (100 mm rail), 105 mm (150 mm rail), 155 mm (200 mm rail)
Mass	about 380 g with the 150 mm rail and the 21.5 mm stepper; a longer rail or stepper adds to it; centre of mass and inertia on Data for robot integration
Position resolution	1/255 of the stroke: 0.2 to 0.6 mm per step of the position byte
Drive	NEMA 17 stepper, pinion (32 DP, 21 teeth, 8.3 mm pitch radius) on two racks, MGN7 linear rail

	Value
Grip force	ideal combined force up to about 31 N (21.5 mm stepper), 50 N (40 mm) or 76 N (60 mm) at the stepper's maximum current; see the model assumptions in Force and speed
Jaw speed	up to about 190 mm/s per jaw at speed 255; see Force and speed
Grasps	external (closing on the object) and internal (opening inside a hollow object)
Mounting	coupler plate for an 8 mm shaft (shaft coupler with M4 clamp screws); a ring cover for the PAROL6 wrist
Body material	PETG, printed with 25 % infill
Jaws	printed: solid, hollow, or a PETG body with a TPU pad; sponge tape on the faces
Camera	mount for an OV9732 USB camera module (100°), looking between the jaws

Environmental

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

Variants

Rail	Stroke	Stepper	Motor	Maximum current	Assumed torque constant
100 mm	55 mm	21.5 mm	17HS08-1004S	1.5 A	0.17 Nm/A

Rail	Stroke	Stepper	Motor	Maximum current	Assumed torque constant
150 mm	105 mm	40 mm	17HS16-2004S1	2.0 A	0.21 Nm/A
200 mm	155 mm	60 mm	17HS24-2104S	2.1 A	0.30 Nm/A

Any rail goes with any stepper, and the body comes in black or grey; see Variants.

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 (`#Kt` prints what the driver uses) 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:

$$F = \frac{K_t\, I_q}{r}, \qquad r = 8.3\ \text{mm}$$

r is the pitch radius of the pinion (32 DP, 21 teeth). The force is proportional to the current, with a slope set by the stepper:

Stepper	Motor	Assumed K_t	Ideal combined force per ampere	Maximum current	Ideal combined force at the maximum current
21.5 mm	17HS08-1004S	0.17 Nm/A	20 N/A	1.5 A	about 31 N
40 mm	17HS16-2004S1	0.21 Nm/A	25 N/A	2.0 A	about 50 N
60 mm	17HS24-2104S	0.30 Nm/A	36 N/A	2.1 A	about 76 N

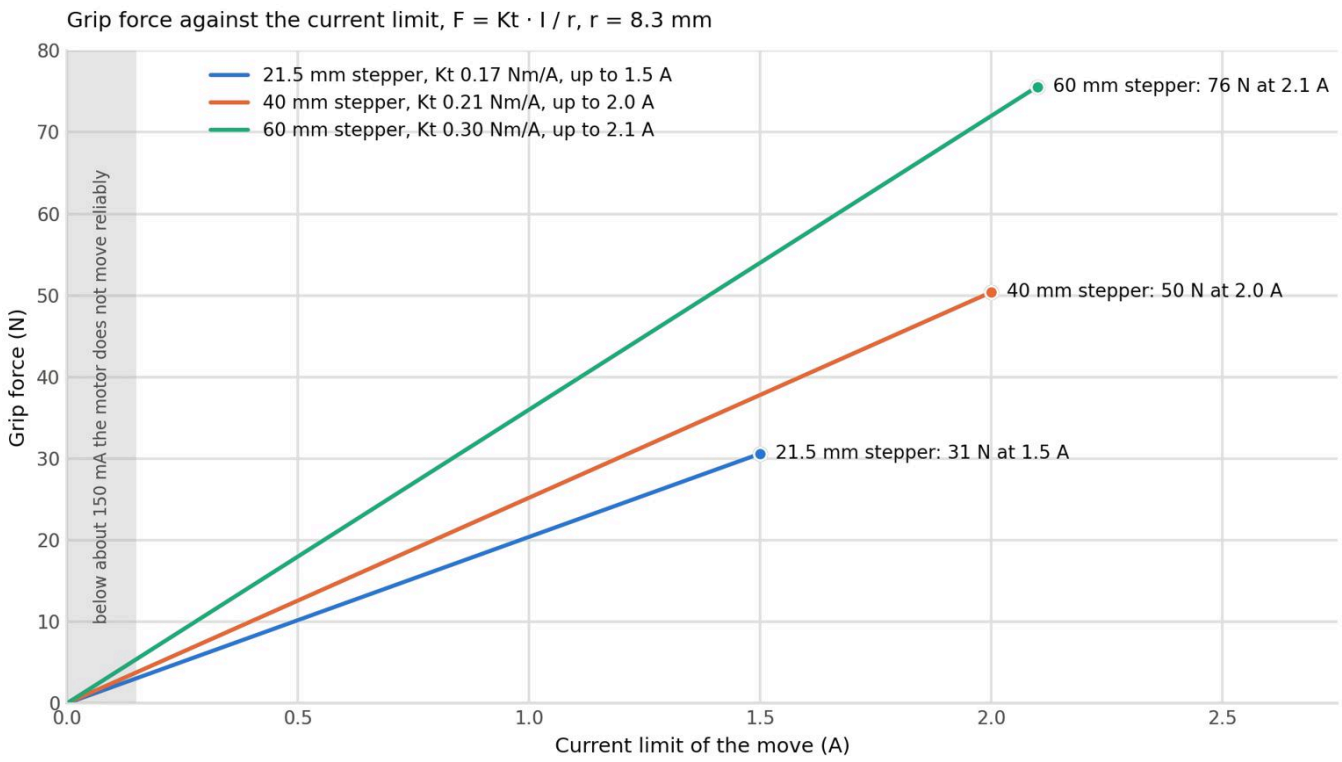


Figure 3.1: The grip force against the current limit for the three steppers, each up to its maximum current, from the formula above.

The curves use the listed torque constants; they are not force-gauge measurements. STEPFOC estimates `#Kt` from a no-load speed/voltage test, rather than measuring shaft torque. The constant must match the driver's q-axis current convention; a motor datasheet's current rating or supply current cannot be substituted directly. A self-built gripper with another stepper needs its own force calibration. Below about 150 mA the motor does not move reliably, so the usable range starts at a few newtons. Motor GUI commands up to 1500 mA; the full force of the 40 mm and 60 mm steppers needs a command from Python or over CAN.

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.

Stay within the stepper's maximum

The firmware does not clamp the gripper's current. A current above the stepper's maximum overheats the motor; the thermistor stops the gripper at 75 °C, but a motor driven far over its rating heats faster than the thermistor follows. The maximums are in the table above.

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 and the stepper's torque constant, the weight held by friction with the safety factor of the next section, and the speed from the speed byte over the rail's stroke.

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 jaw faces 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 11 N of supported weight (1.2 kg) at 76 N of combined grip force, or 4.7 N (0.47 kg) at 31 N. These are static estimates. 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 jaws 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 500 at speed 0 to 60000 at speed 255:

$$\text{ticks/s} = 500 + \frac{60000 - 500}{255}\, C$$

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	500 ticks/s, 0.03 rev/s	1.6 mm/s	3 mm/s
30	7500 ticks/s, 0.46 rev/s	24 mm/s	48 mm/s
128	30400 ticks/s, 1.9 rev/s	97 mm/s	190 mm/s
255	60000 ticks/s, 3.7 rev/s	190 mm/s	380 mm/s

Both jaws move, so the gap closes at twice the speed of one jaw. At speed 255 a 105 mm stroke takes about 0.3 s. 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 the stroke divided by 255: 0.22 mm on the 100 mm rail, 0.41 mm on the 150 mm rail, 0.61 mm on the 200 mm rail. The driver considers the position reached within 5 encoder ticks, about 0.03 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.

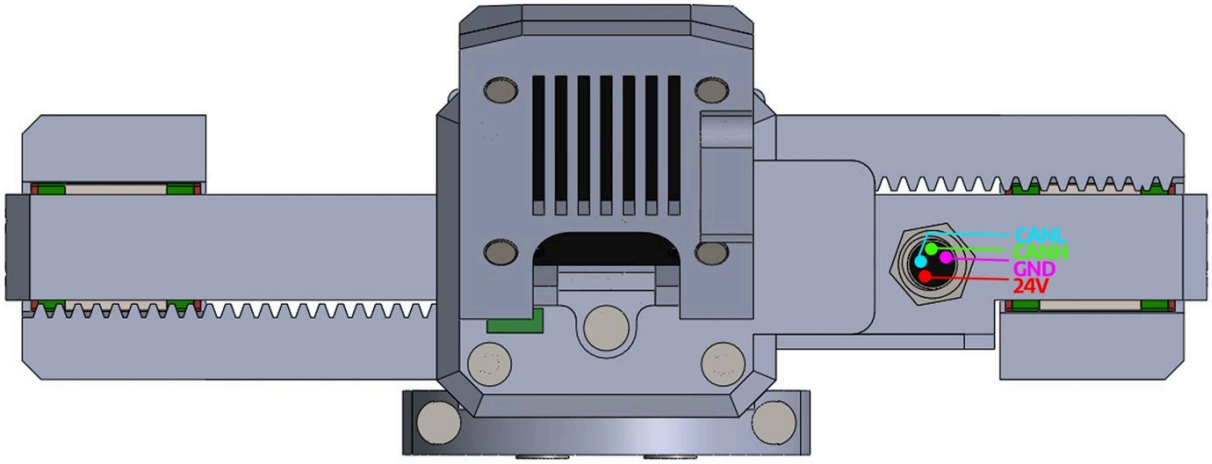


Figure 3.2: The M8 connector on the gripper body and its four signals: 24V, GND, CANH, CANL.

Pin numbers

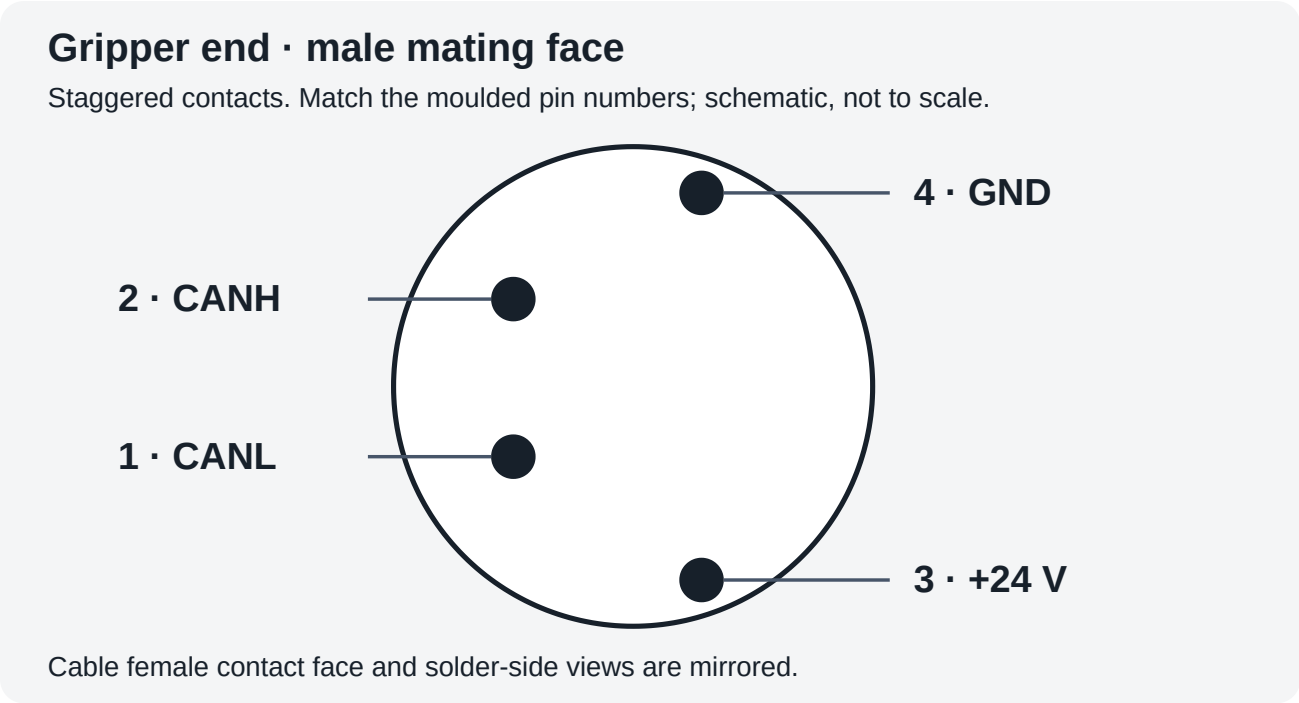


Figure 3.3: M8 male on the gripper: front mating face and numbered signals.

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 pin table above and the wiring diagram before you plug it in.

Inside the housing

The STEPFOC driver's own connectors are inside: its UART port (3.3 V, JST SH, 4 pins) under the coupler plate, used to configure a self-built gripper (Over UART), and its CAN termination switch, reachable through an opening in the stepper cover (CAN termination). The driver's pinout is in the STEPFOC documentation.

Dimensions and CAD

The drawings show the variant of the 3D model: the 150 mm rail, the 21.5 mm stepper and the solid jaws, with the jaws fully open. All dimensions are in millimetres, measured on the gripper's CAD as the robot model carries it. Heights are counted from the coupler's back face, the face that rests against the robot; the coupler axis is the gripper's axis on the robot.

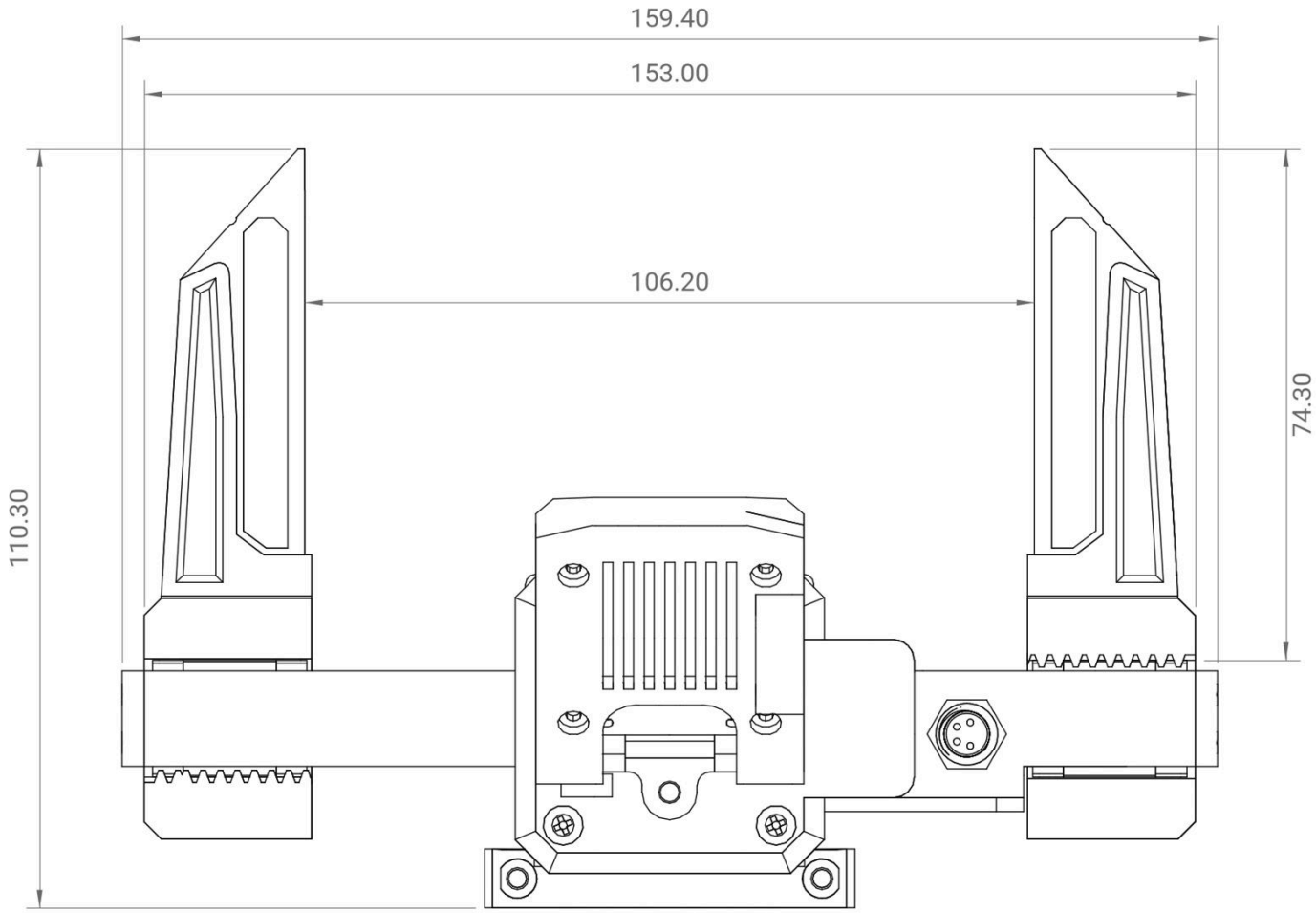


Figure 3.4: Front view, from the camera side, jaws fully open: 159.40 mm over the jaw blockers at the ends of the rail, 153.00 mm over the jaws, 106.20 mm between the faces of the solid jaws, 110.30 mm from the coupler's back face to the jaw tips, which reach 74.30 mm beyond the rail block.

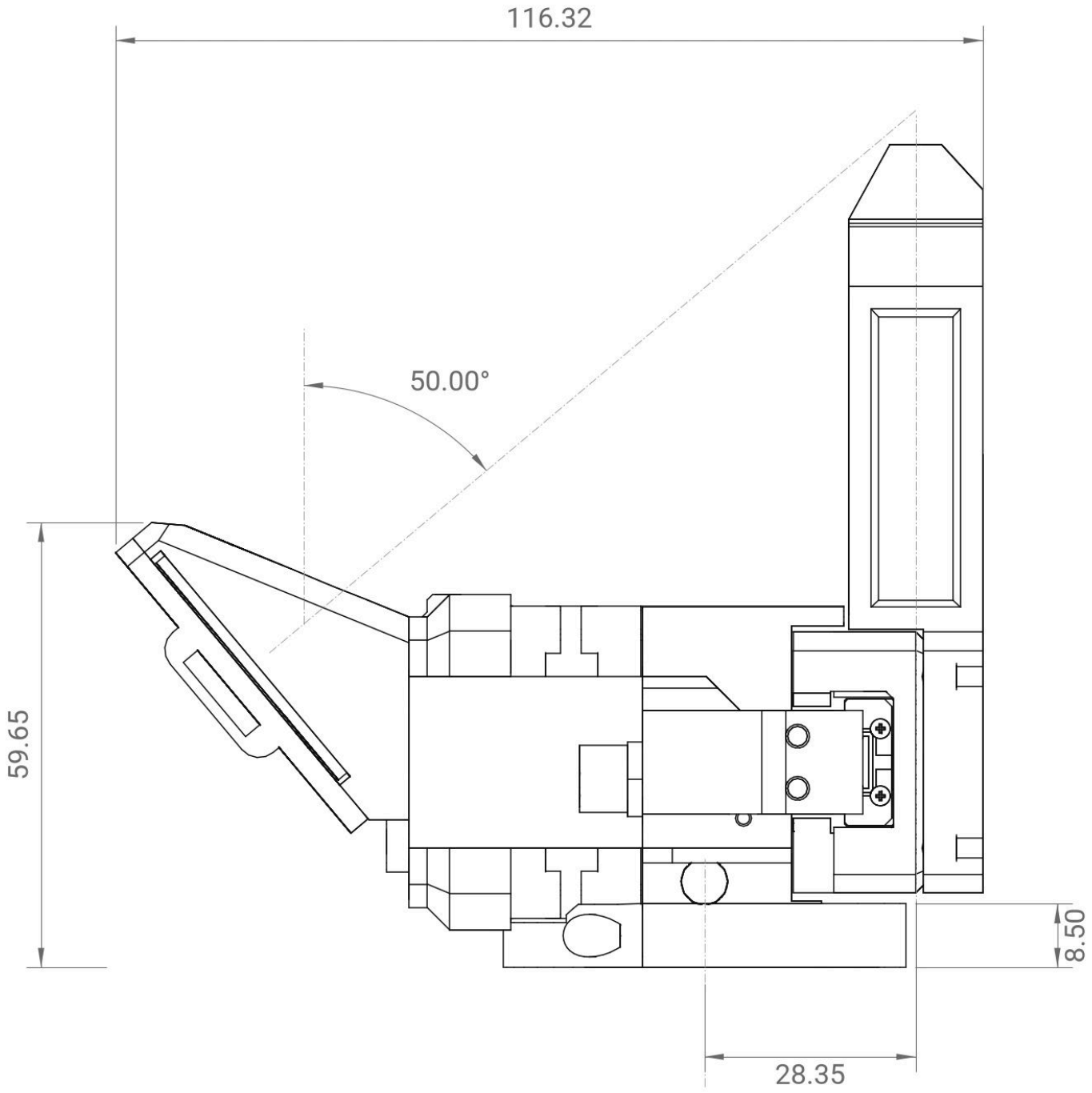


Figure 3.5: Side view: 116.32 mm from the jaws to the top of the camera mount, the jaws' centre line 28.35 mm from the coupler axis, an 8.50 mm thick coupler ring, the camera mount 59.65 mm in front of the coupler's back face, and the camera looking 50.00° off the coupler axis toward the jaws.

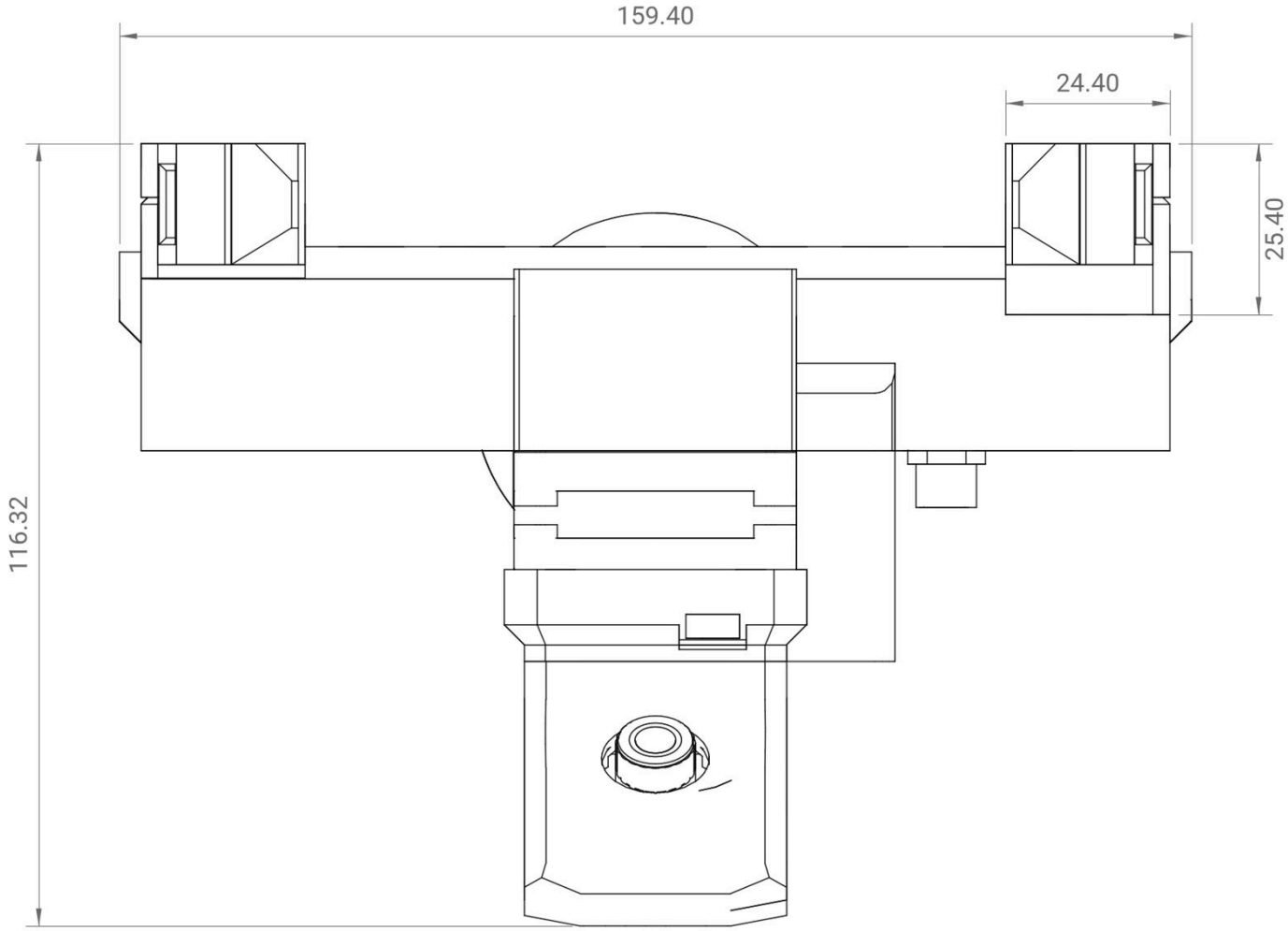


Figure 3.6: Top view, from the jaw tips: 159.40 mm across, 116.32 mm from the jaws to the top of the camera mount, each jaw 24.40 by 25.40 mm at its carrier.

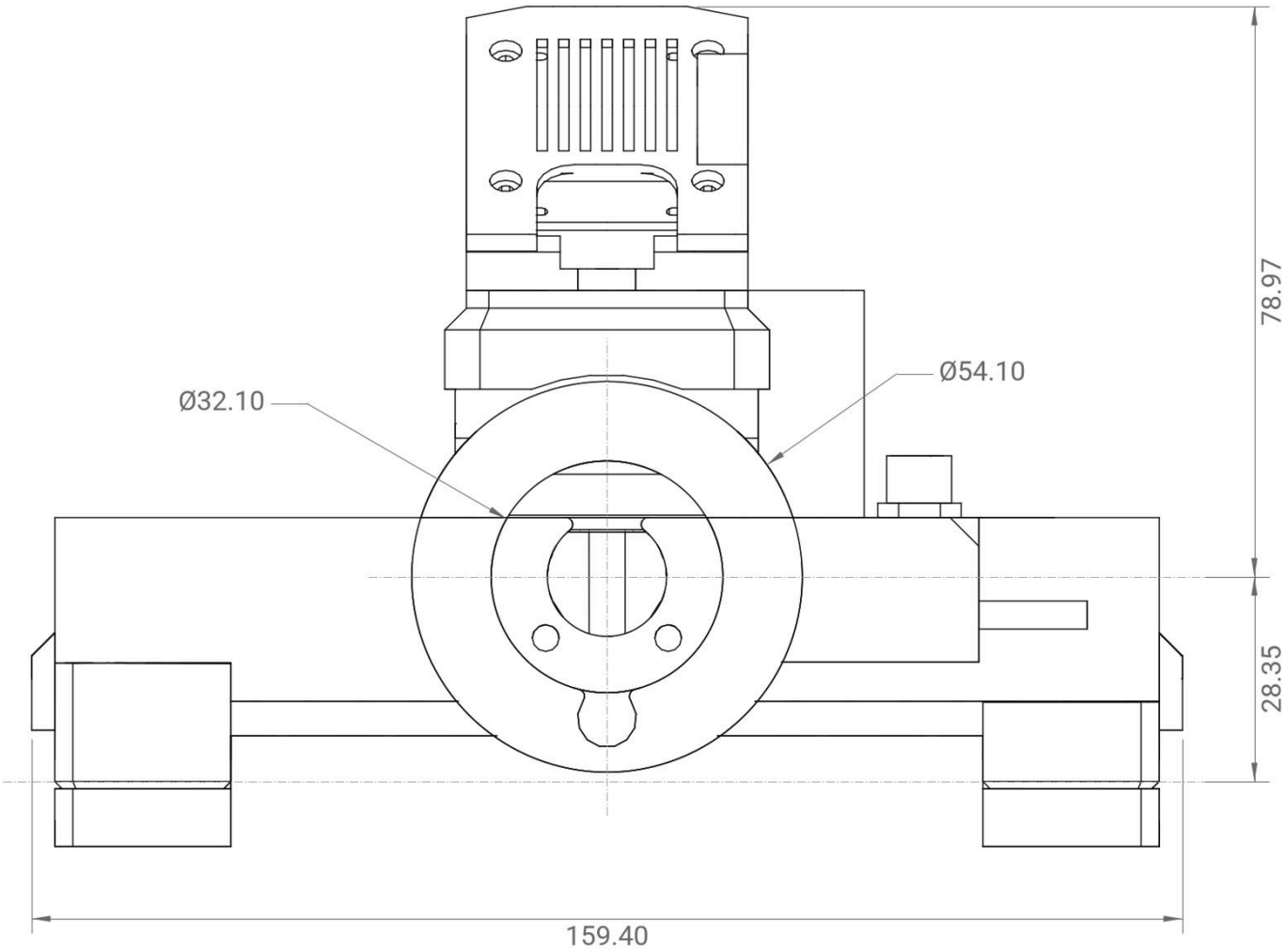


Figure 3.7: The coupler, seen from the robot: a ring of 54.10 mm diameter with a recess of 32.10 mm, round the coupler axis; the jaws' centre line 28.35 mm to one side of the axis, the top of the camera mount 78.97 mm to the other.

The recess is 6.50 mm deep; the flange of the shaft coupler sits in it. The PAROL6 ring cover has the same 54.10 and 32.10 mm circles (Mounting on a robot). `tools/dimension_drawings.py` in this site's repository draws the four views from the model and prints every number on them.

Other rails and steppers

The rail sets the width and the stroke; the stepper sets how far the stepper cover, and the camera mount on top of it, stand away from the rail. Neither changes the coupler, the jaws or the 110.30 mm to the jaw tips. The lengths below are measured in the parts' STEP files; the overall sizes add them to the drawn variant.

Rail	Stroke	Body	Over the jaw blockers	Printed parts
100 mm	55 mm	103.00 mm	109.40 mm	`MSG_Gripper_body_100mm`, `gripper_jaw_100mm_L/R`
150 mm (drawn)	105 mm	153.00 mm	159.40 mm	`MSG_Gripper_body_150mm`, `gripper_jaw_150mm_L/R`
200 mm	155 mm	203.00 mm	209.40 mm	`MSG_Gripper_body_200mm`, `gripper_jaw_200mm_L/R`

The jaw blockers at the two ends of the rail are 3.20 mm thick each. On the drawn 150 mm rail the open jaws stand flush with the ends of the body (153.00 mm over the jaws).

Stepper	Motor	Stepper cover	Cover's outer end, from the 21.5 mm one	Jaws to the top of the camera mount	Coupler axis to the top of the camera mount
21.5 mm (drawn)	17HS08-1004S	34.40 mm	0	116.32 mm	78.97 mm
40 mm	17HS16-2004S1	51.40 mm	+18.50 mm	134.82 mm	97.47 mm
60 mm	17HS24-2104S	71.40 mm	+38.50 mm	154.82 mm	117.47 mm

CAD

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

- STEP files: `Common parts` (the ring covers, the camera mount and its lid, the connector lid, the jaw blocker, the stepper spacer), `Linear rail dependant parts` (the body and the jaw carriers per rail length), `Stepper dependant parts` (the covers and spacers per stepper length), `Gripper jaws attachment` (the jaw designs), `Tools` (the magnet tool). There is no STEP file of the whole gripper; the robot model below is the assembled gripper.
- Print files: the same parts as 3MF, ready to slice.

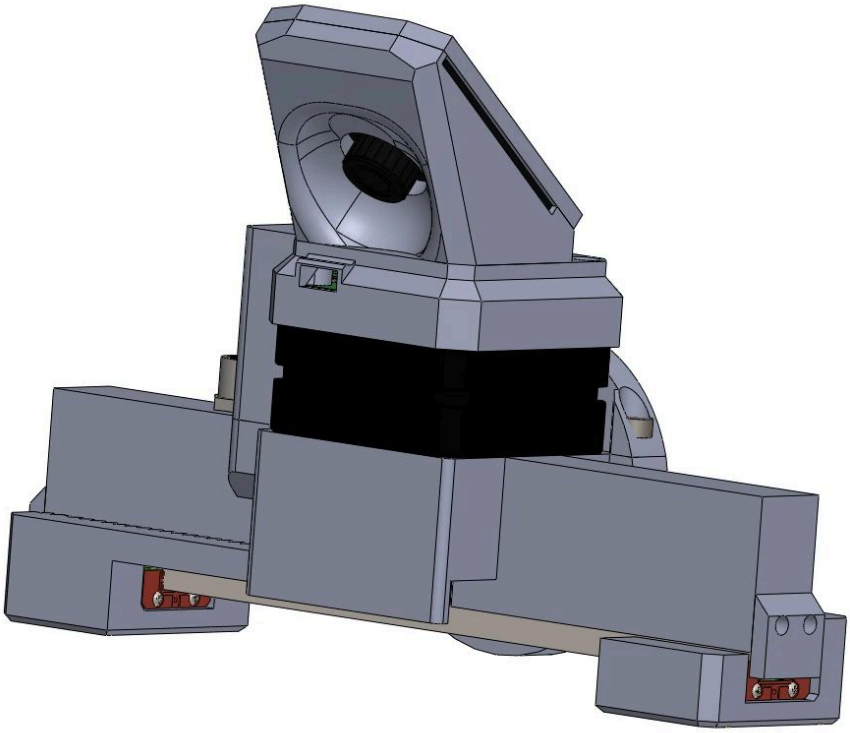


Figure 3.8: The STEP model of the gripper without its jaws, seen from the front, into the camera.

For simulation

The product repository's msg_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 soft TPU jaws, with their meshes, collision shapes and mass properties, and the PAROL6 and PAR6 models with the gripper mounted. The numbers a robot needs, the tool centre point, the mass and the inertia, are on Data for robot integration.

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 the numbers are given in, the mass, the centre of mass and the inertia, and the tool centre point (TCP). The numbers are taken from the gripper's robot model in robot_descriptions (version 2.18.1, `msg_description` in URDF and `msg` in MJCF) and moved into the frame below. The model is the 150 mm rail with the 21.5 mm stepper, the variant of the drawings.

The frame

The origin is on the coupler axis, at the centre of the coupler's back face: the face that rests against the robot.

- **Z** runs along the coupler axis, from the robot toward the jaw tips: the approach direction.
- **Y** runs along the jaw travel, parallel to the rail, positive toward the side of the M8 connector.
- **X** completes the frame: from the rail and the jaws toward the stepper and the camera. The jaws' centre line is at X = −28.35 mm, the top of the camera mount at X = +78.97 mm.

The axes are those of the model's `flange` frame. That frame's origin lies 15.50 mm further along Z, inside the body behind the coupler ring: subtract 15.50 mm from a Z below to have it in the model's `flange` frame.

Mass and inertia

Model, jaws	Mass measured	Mass in the model	Centre of mass x, y, z (mm)	Ixx, Iyy, Izz (10^{-6} kg·m²)	Ixy, Ixz, Iyz (10^{-6} kg·m²)
URDF, jaws open	about 380 g	380.0 g	−0.70, 0.22, 32.34	508.5, 147.6, 525.5	0.4, 42.0, −1.9
URDF, jaws closed	about 380 g	380.0 g	−0.70, 0.22, 32.34	200.2, 147.6, 217.2	0.4, 42.0, 0.2
MJCF, jaws open	about 380 g	380.0 g	−0.42, 0.24, 31.89	471.7, 137.5, 488.0	0.6, 35.7, −5.7
MJCF, jaws closed	about 380 g	380.0 g	−0.42, 0.24, 31.89	199.6, 137.5, 215.9	0.5, 35.7, 1.6

The model's mass is the body, 301.4 g, and two moving assemblies of 39.3 g each (the rail carriage, the jaw carrier and the jaw). The inertia is about the centre of mass, with the axes of the frame above, as the elements of the inertia tensor that a URDF's `<inertia>` takes. The solid and the soft TPU jaws have the same mass data in the model, so one row serves both. Closed is the model's closed end of the travel. Opening and closing move the two jaws along Y by the same amount in opposite directions: the centre of mass stays where it is, Ixx and Izz change. The URDF and the MJCF model give the jaws different centres of mass and inertias, hence the two pairs of rows.

Tool centre point

Jaws	The point	x (mm)	y (mm)	z (mm)	Orientation	Opening
Solid	the middle between the jaw tips	−28.35	0	110.30	the frame's axes	any
Solid	the middle between the gripping faces	−28.43	0	79.65	the frame's axes	any
Soft TPU	the middle between the jaw tips	−28.35	0	110.30	the frame's axes	any
Soft TPU	the middle between the TPU pads	−28.43	0	80.15	the frame's axes	any
Both	the model's `tcp` frame	−28.35	0	99.30	the frame's axes	any

The two jaws move by the same amount in opposite directions, so each point stays on the gripper's centre plane (y = 0) at every opening. A gripping face is the centre of its area: the solid jaw's face is 18.00 mm wide along X and runs from z = 51.30 to 110.30 mm, the TPU pad from z = 52.30 to 110.30 mm. Fully open, the solid jaws' faces are 106.20 mm apart, the TPU pads 104.20 mm. The model's `tcp` is its control point on the jaws' centre line, 11.00 mm short of the tips, the same for both jaws; robot_descriptions says it is not a measured grasp centre.

How accurate this is

The masses are measured: about 380 g for the whole gripper with the 150 mm rail and the 21.5 mm stepper (the variant the model shows), 39.3 g for each moving assembly (an approximate figure). The centres of mass and the inertias are not: robot_descriptions 2.18.1 calls them existing, unverified estimates, not scaled to the measured masses or checked against the CAD's mass distribution. Use them for simulation and for a first payload setting, not as a measurement. The model has one variant; a longer rail or stepper is heavier and moves the centre of mass toward the extra part, and the model does not say by how much.

Where the numbers go

In the PAROL6 Python API the MSG is a built-in tool, `rbt.select_tool("MSG", variant_key="150mm")`, with its TCP at (−29, 0, −109.5) mm in the API's flange frame, whose Z points back into the wrist (Tools and the TCP); in a URDF, the mass, the centre of mass and the inertia go into the gripper link's `<inertial>` and the TCP into a fixed joint from the mounting frame.

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 plate, the round plate on the back of the body behind the stepper. Two covers exist for it, and the STEP files let you make a third.

An 8 mm shaft

As shipped, the coupler plate carries a shaft coupler with an 8 mm bore and M4 clamp screws under the `ring_cover` part: the gripper slides onto an 8 mm shaft and is clamped with the two M4 screws. Any robot or fixture that offers an 8 mm shaft takes the gripper this way.

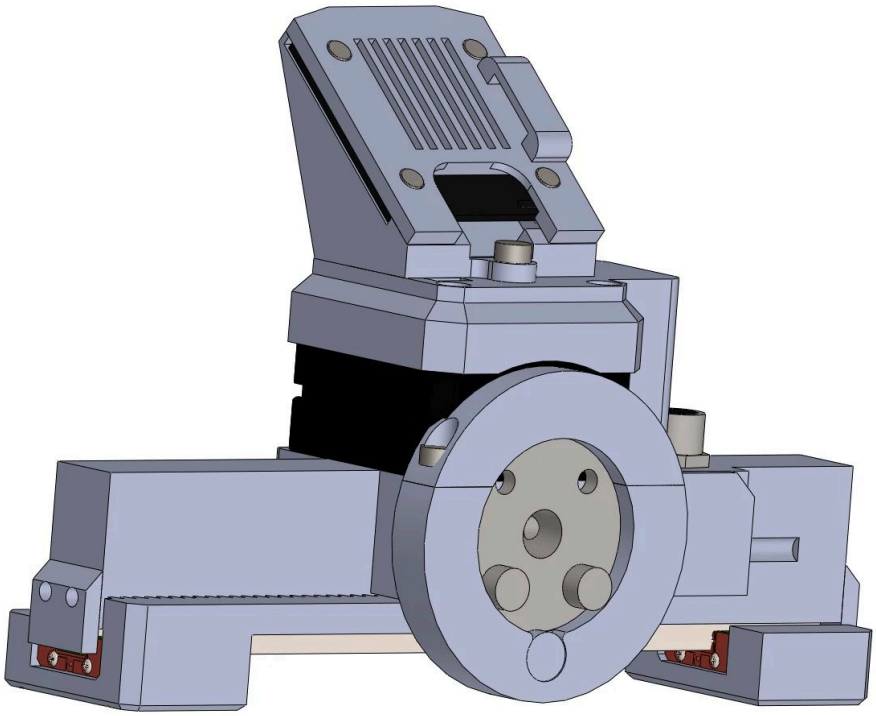


Figure 4.1: The coupler plate with the 8 mm shaft coupler.

PAROL6

For the PAROL6 wrist, replace `ring_cover` with `ring_cover-PAROL6` from the STEP files (`Common parts`): it bolts to the robot's last joint. Tighten the shaft coupler's two clamp screws with blue threadlocker. The PAROL6 gripper cable then runs from the gripper to the robot's gripper port; see The cable and On PAROL6 and PAR6, which also has the joint 5 homing offset.

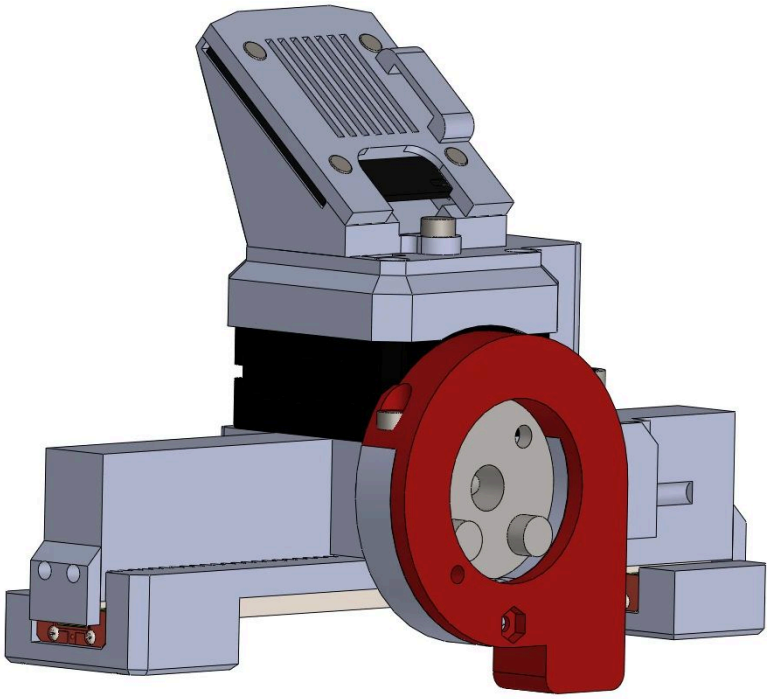


Figure 4.2: The gripper with the PAROL6 ring cover (red).

Another robot

Adapt `ring_cover`, or the body itself, to your flange: the STEP files are open. Keep the gripper's connector reachable and the camera's view free, 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 jaws 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

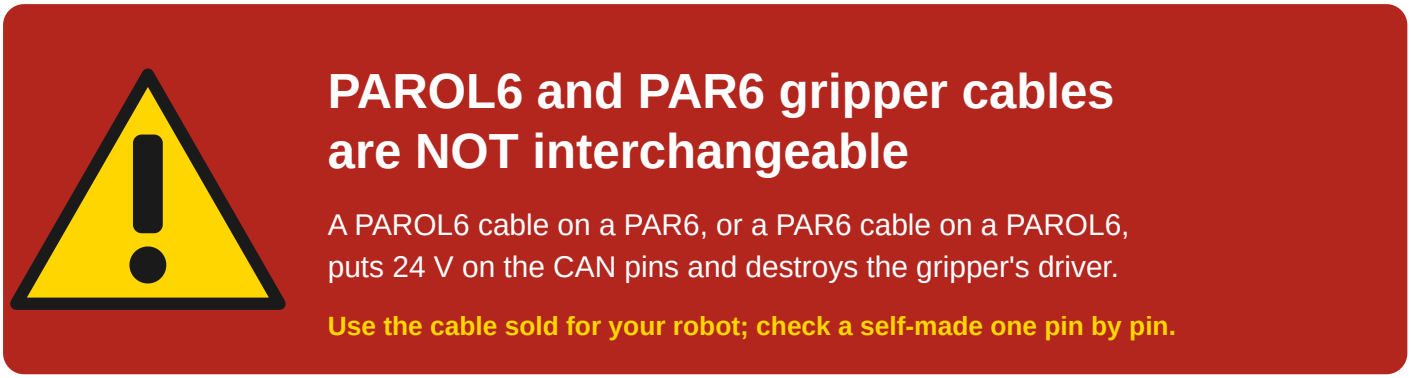


Figure 4.3: PAROL6 and PAR6 gripper cables are not interchangeable.

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:

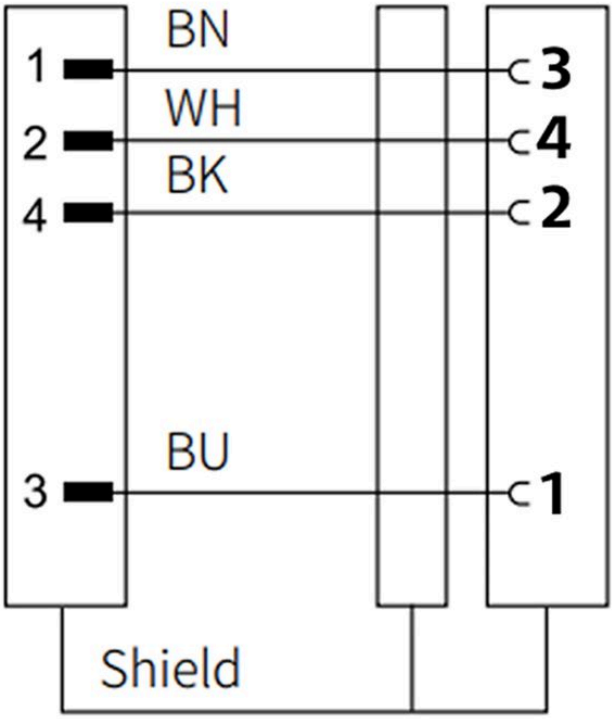
Part	Quantity	BOM
26 AWG 4-wire twisted-pair cable, 0.15 mm², shielded	about 0.4 m	cable
M8 4-pin female cable connector (the gripper has the male)	1	connector
M8 4-pin male cable connector, for a robot with an M8 port	1	the same listing





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



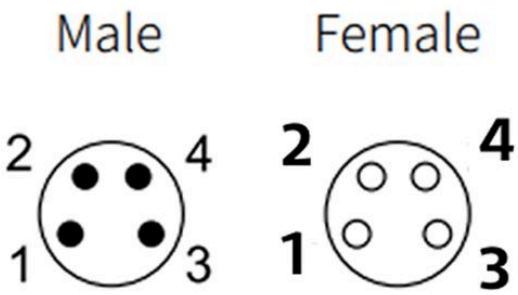


Figure 4.4: The PAROL6 cable: the pins are crossed between the two ends.

PAR6

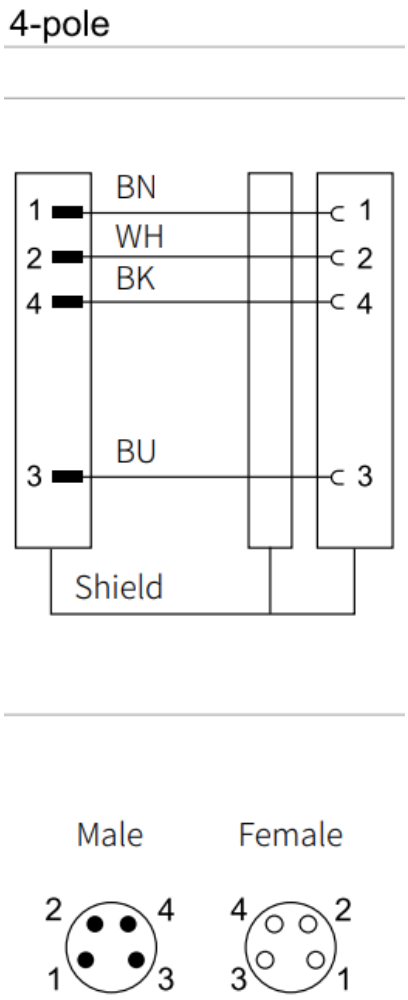


Figure 4.5: The PAR6 cable: pin to pin.

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 jaws bolt to the two carriers on the rail and are yours to adapt. The product repository has three designs, all in Gripper jaws attachment; each file is one jaw, mirrored in the slicer for the other side, except the solid jaw, which has a mirrored file.

Design	File	Use
PETG body with a glued-on TPU pad	`MSG_JAW_TPU_GLUE`	the recommended jaw: the soft pad deforms around the object and holds it better
Solid jaw	`MSG_solid_jaw`, `MSG_solid_jaw_mirror`	one part, quick to print
Hollow jaw	`MSG_JAW_HOLLOW`	lighter, and suited to multi-colour prints

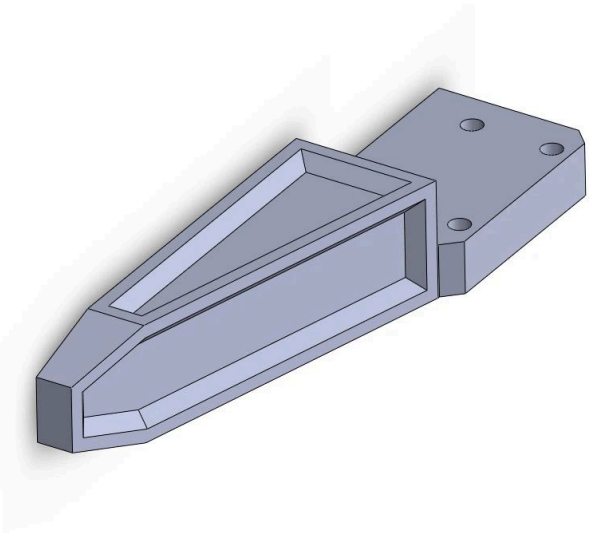


Figure 4.6: The jaw body, with the mounting holes for the carrier.

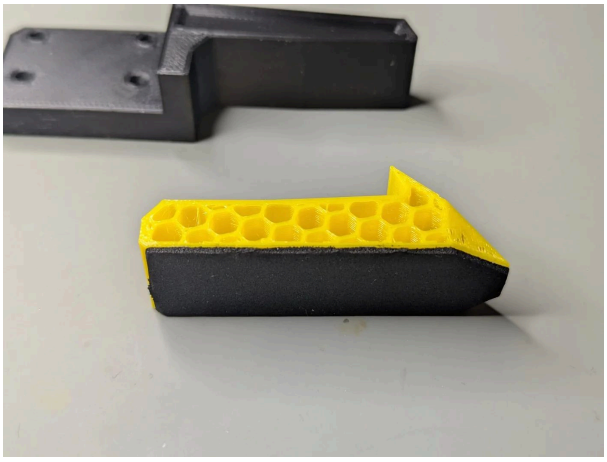
The TPU jaw



Figure 4.7: The TPU jaw: a PETG body (black) and a TPU pad (yellow).

Print the body in PETG with your usual settings and the pad in TPU with 16 % infill, no top or bottom layers and two walls, for the softest pad. Then glue the pad to the body with ordinary super glue, in the orientation the pictures show: a pad glued the wrong way round leaves the jaws facing the wrong direction.







Sponge pads

A strip of 1 mm sponge tape (30 mm wide, in the BOM) on the gripping face of any jaw improves the hold on hard and smooth objects a great deal. Cut it to length, peel the backing and press it on.

Your own jaws

Keep the carrier's hole pattern from one of the designs and shape the rest for your objects. Long jaws multiply the lever arm on the carriers: keep them short and stiff, and remember that the grip force is set at the jaw faces, not at the rail.

> **Custom jaws on a PAR6**
>
> On a PAR6, custom jaws must keep the size of the original jaws: 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 jaw reaches where the robot does not expect it.

Calibrate after changing the jaws

The gripper calibration finds the end stops with the jaws fitted; jaws 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 that is reachable through an opening in the stepper cover: slide it to the **ON** mark to connect the resistor, away from it to disconnect it.

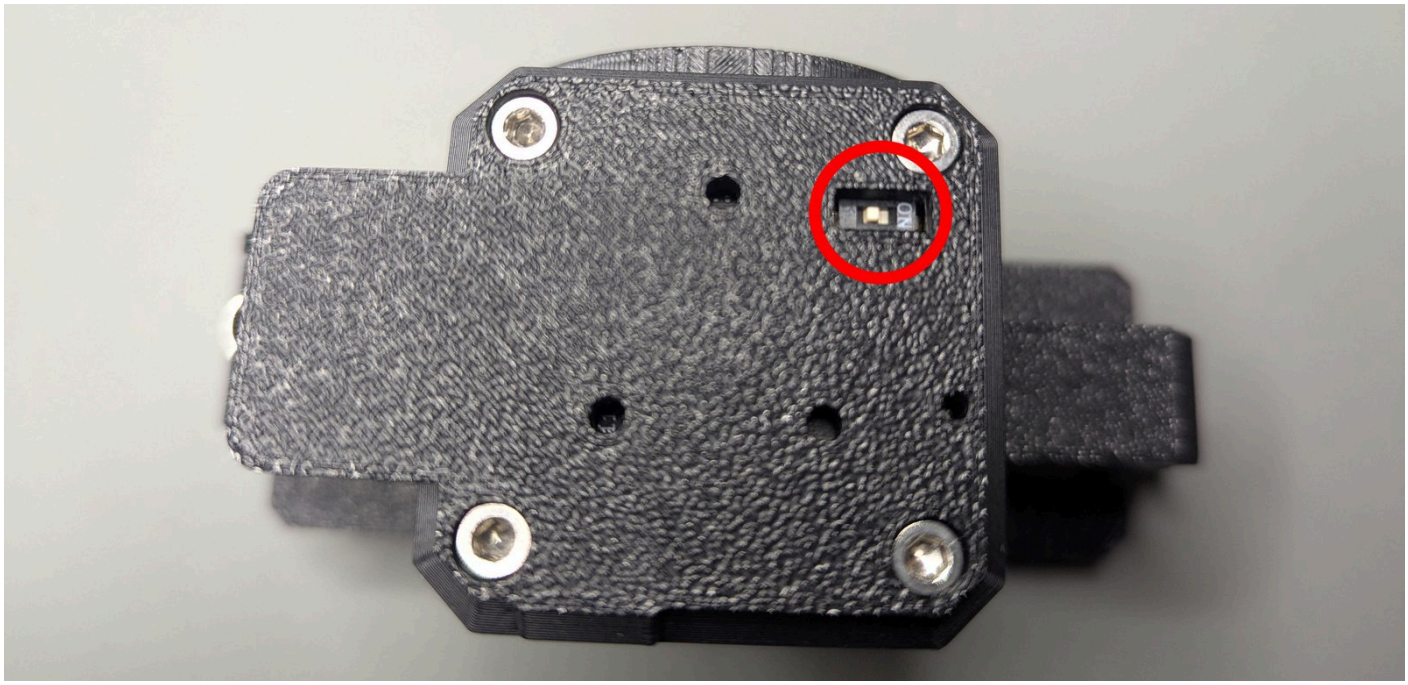


Figure 4.8: The termination switch of the driver, seen through the stepper cover.

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

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.

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 STEPFOC 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.

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.

Check

The gripper panel shows the gripper's position and current, and *Not calibrated* after a power-up. If the browser's port list does not show the adapter, see the notes for your system below; if the panel keeps waiting for feedback, step 3 and Troubleshooting.

The adapter's port on your system

Windows

The CANvas adapter appears in Device Manager under **Ports (COM & LPT)** as a COM port, and the browser's port list shows it. Close any other program that holds the port, a Python script or another browser tab: the connection fails at once while it is open elsewhere.

Linux

The browser opens the port only if your user may. Add yourself to the `dialout` group once, then log out and in again:

```
sudo usermod -aG dialout $USER
```

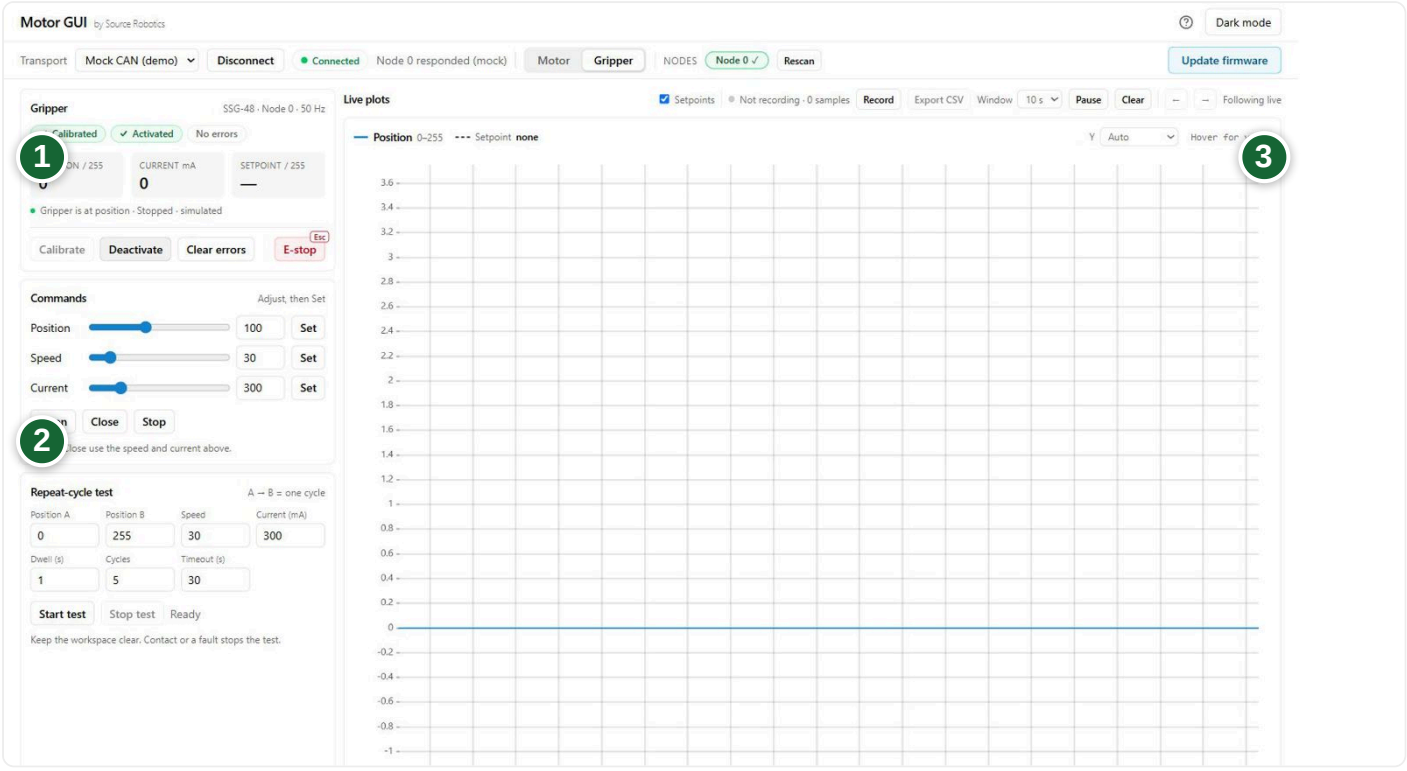
macOS

Nothing to install. A browser installed with Homebrew or from the App Store may need USB access: Motor GUI's own guide points to System Settings, **Privacy & Security**.

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
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

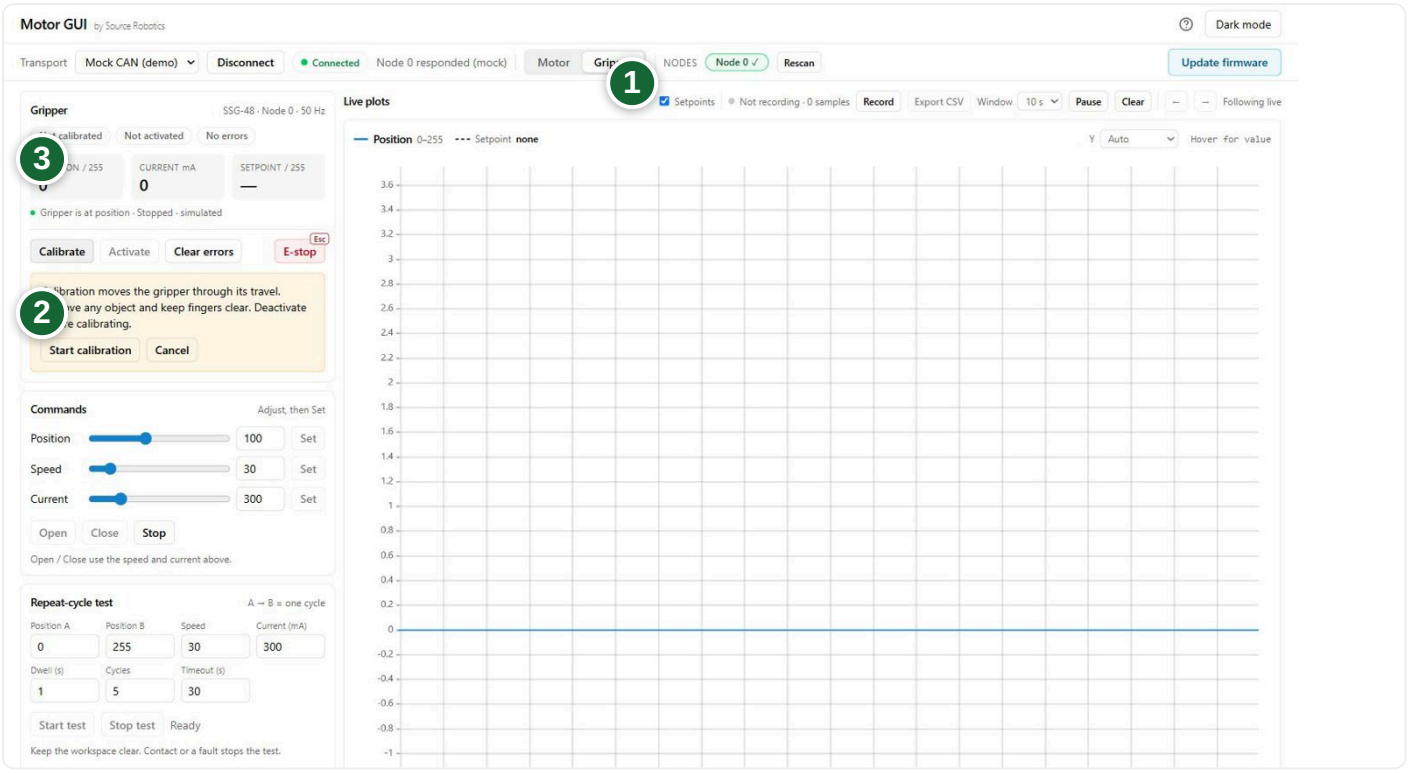
Control	What it does
E-stop, or Esc	stops the gripper at once: the move command with the E-stop bit, after which the gripper reports the E-stop error until a command with the bit at 0 has been sent and the error cleared
Clear errors	clears the errors; an error whose cause remains comes straight back
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

Motor GUI's current limit stops at 1500 mA. The 40 mm and 60 mm steppers take more (Force and speed); for their full force, command the move from Python or over CAN.

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: *MSG 21.5 mm motor*, *MSG 40 mm motor* and *MSG 60 mm motor*, and *SSG-48* 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 stepper must be calibrated too, which a bought gripper is from the factory; a self-built gripper gets that in Driver configuration.

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

Motor GUI shows no error; in Python the three error attributes read 0 after the next status poll; over UART, `#Error` prints `No errors!`. An error that is back at once still has its cause: Errors and recovery.

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 half a second. It takes about ten 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(10)       # 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 stepper is not calibrated. If the routine runs for 24 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*: in Motor GUI's panel, as `gripper_calibrated` = 1 in Python, or `Gripper calibrated is: 1` from `#Gripinfo`. If not after 24 s: 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 panel shows *Activated*; in Python `gripper_activated` reads 1; `#Gripinfo` prints `Gripper activated is: 1`.

5. Move

A move is three numbers: the position (0 open, 255 closed), the speed (0 to 255: linear from 1.6 to 190 mm/s at each jaw) and the current limit in mA, which is the grip force (Force and speed). Start with speed 30, about 24 mm/s at each jaw, and 300 mA.

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, within the stepper's rating, and the speed once the motion is what you want.

Check

The jaws go to each position and the state reads *at position*; on the object they stop and the state reads *object detected while closing*. If they do not move: 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

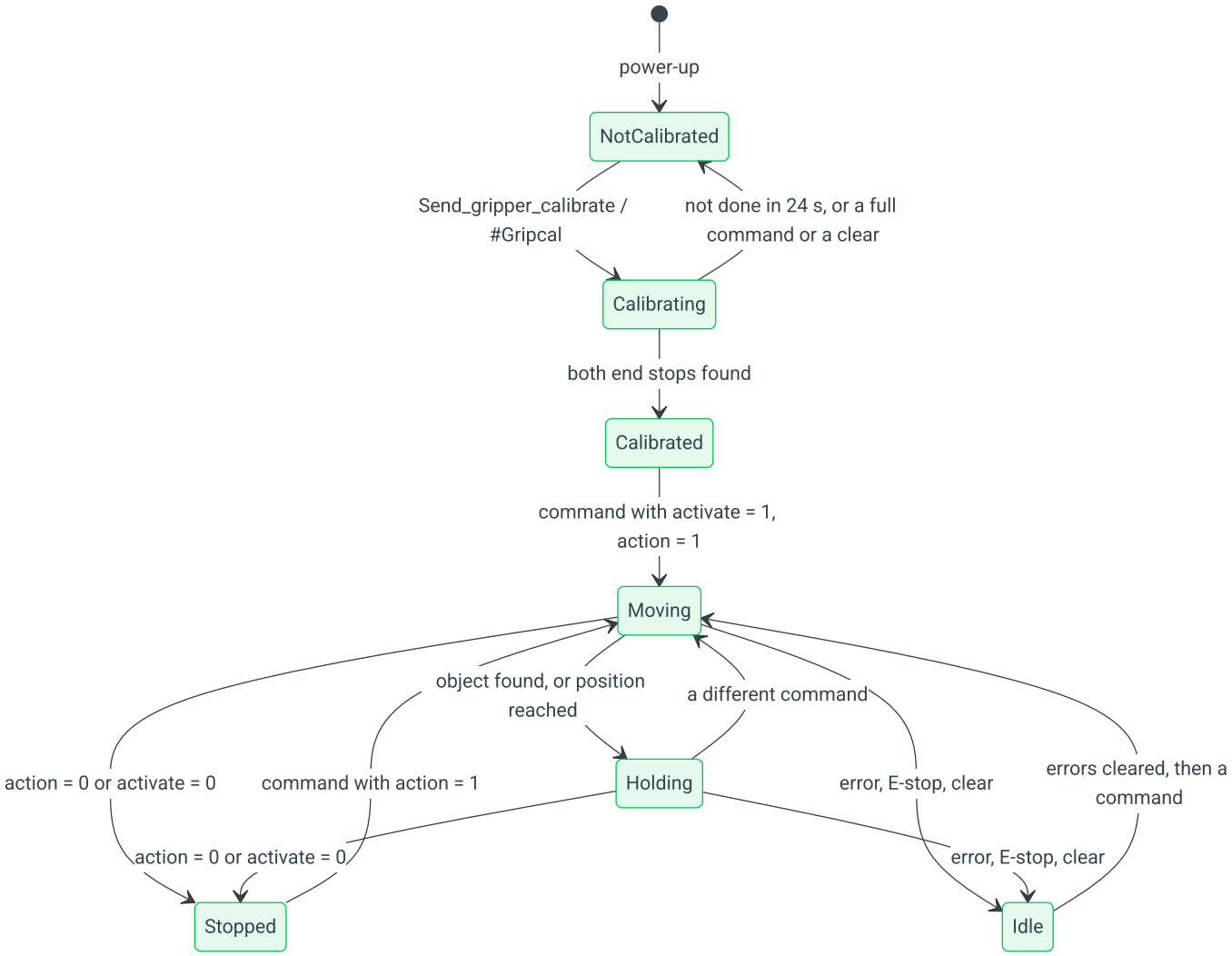
- How the gripper works
- Over CAN
- Over UART
- With Python
- With ROS 2
- On PAROL6 and PAR6
- Errors and recovery

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 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 24 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.

Stopped

- A command with the action bit at 0, or with the activate bit at 0, stops the move.
- The driver keeps the jaws still: it runs its velocity loop at zero speed with the command's current limit, so the jaws resist being moved, and the object detection keeps reporting.
- The next command with the activate and action bits set moves the jaws again.

Idle

- An error, the E-stop bit, Send_Clear_Error, Send_Idle or `#Idle` 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	1.6 to 190 mm/s at each jaw, linear (500 to 60000 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: the move stops and the jaws are held still
Action	bit	1 carries out the move; 0 stores the command and holds the jaws still where they are
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` (50 and 400 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

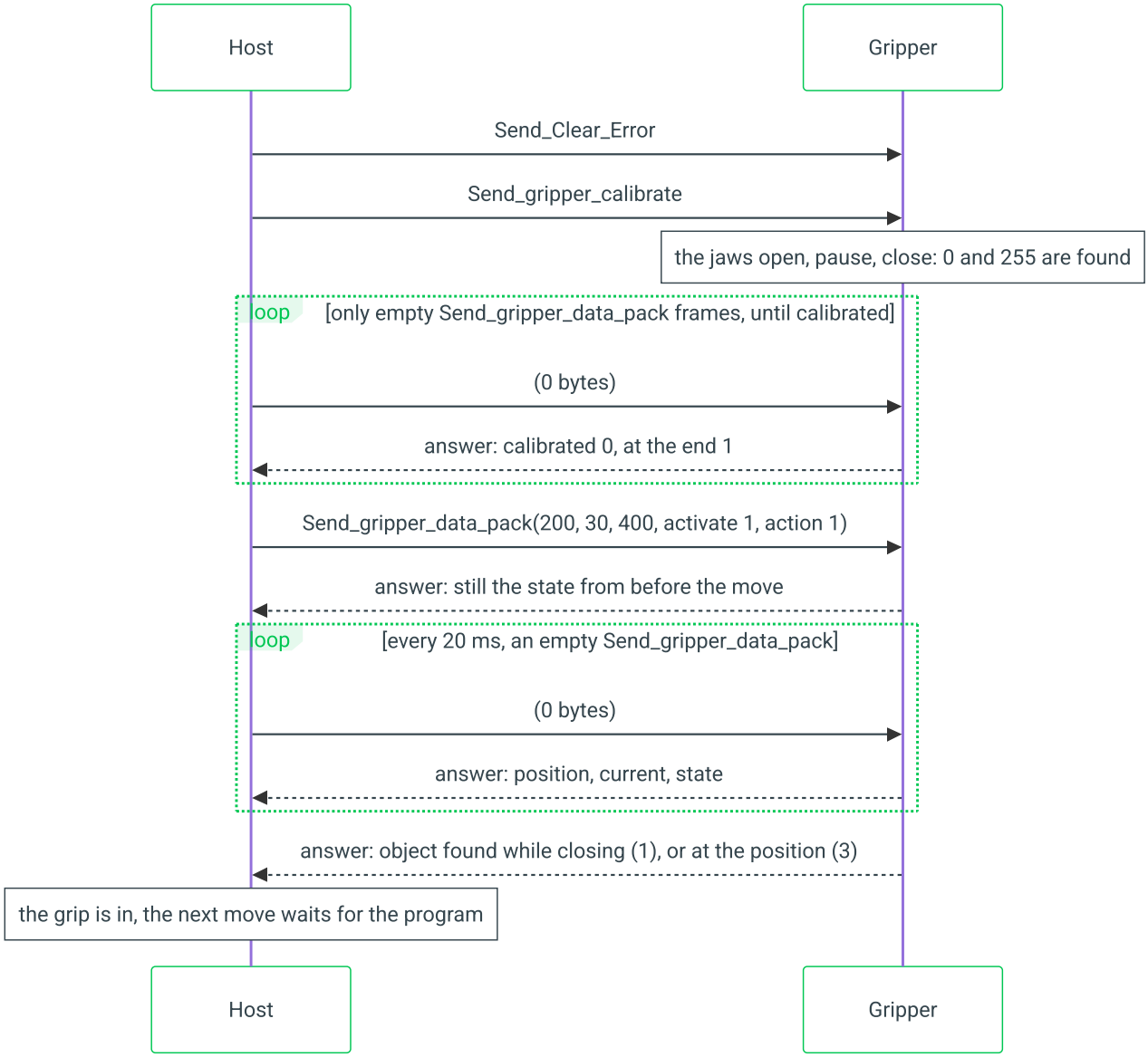
Field	Meaning
Action status	the action bit of the last command, as received; it stays 1 while the gripper holds, and the calibration does not change it
Object detection	0 moving, 1 object found while closing, 2 object found while opening, 3 at the position
Temperature error	the motor's thermistor is 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 jaws move faster than 500 encoder ticks per second, about 1.6 mm/s. When they are 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 driver 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 6250 Hz; 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 STEPFOC 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:

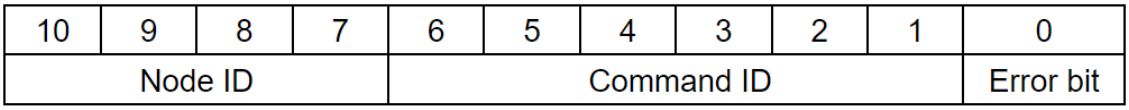


Figure 6.1: Bits 10 to 7 of the identifier are the node ID, bits 6 to 1 the command ID, bit 0 the error bit.

- **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 STEPFOC 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 (up to about ten seconds; the routine gives up after 24 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 24 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 within the stepper's maximum, 1.5, 2.0 or 2.1 A; 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 housing, under the coupler plate. Unscrew the plate, 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 STEPFOC 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 STEPFOC 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, with Python 3.10 or newer:

Windows

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

The CANvas adapter's port is `COM` and a number: Device Manager lists it under **Ports (COM & LPT)**. Give it as `channel='COM20'`.

Linux

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

The adapter's port is `/dev/ttyACM0` (`ls /dev/ttyACM*` lists it).

macOS

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

The adapter's port is `/dev/tty.usbmodem` and a number (`ls /dev/tty.usbmodem*` lists it).

The library pulls `python-can` and `pyserial`. The source and the examples are in Spectral-BLDC-Python.

The first program

```python
import Spectral_BLDC as Spectral
import time

# the adapter: a CANvas on COM20 (Windows) or /dev/ttyACM0 (Linux), 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(120):                          # wait for the calibrated bit, up to 24 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

Attribute	Meaning
`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

The msg_gripper_ros2 package is a ROS 2 Humble driver for the gripper, built on the Python library and a `python-can` adapter. It was developed by the UCLA Biomechatronics Lab and Ben Forbes, and is MIT licensed.

What it offers

Direction	Topic or action	Type
Command	`/gripper/command_joint_states`	`sensor_msgs/JointState`, the position as 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`

Installing

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
```

On Linux, add yourself to the `dialout` group to open the adapter's port without `sudo`, and log in again.

Running

```
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. Copy the file, change it, and pass it at launch:

```
ros2 launch msg_gripper_ros2 msg_gripper_launch.py config_file:=~/my_gripper.yaml
node_name:=my_gripper namespace:=robot
```

The package's README has the rest: the parameters, the node names, and the platforms it was tested on.

The SSG-48 package

The ssg48_adaptive_electric_gripper_ros2 package, written for the SSG-48 gripper, speaks the same protocol and should work with the MSG as well.

The description files

The gripper's standalone URDF and MJCF (MuJoCo) packages, with solid or soft TPU jaws, come from robot_descriptions; the product repository's msg_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. What any robot's software needs to carry the gripper, the tool centre point, the mass, the centre of mass and the inertia, is on Data for robot integration.

PAROL6

Mounting and connecting

1. Replace the gripper's `ring_cover` with `ring_cover-PAROL6` from the STEP files' `Common parts` (Mounting on a robot). It bolts to the robot's last joint, and the coupler plate's shaft coupler (8 mm bore, two M4 clamp screws) takes the joint 6 shaft.
2. Slide the gripper onto the shaft with the camera looking down between the jaws, and tighten the two clamp screws 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. Route the camera's USB cable outside the arm, with slack for joints 4, 5 and 6, and fix it so that the arm cannot catch it.
5. Keep the gripper's node ID at 0 and set its termination switch ON; it is the last node on the robot's bus.

The joint 5 homing offset

Joint 5 homes against the tool, and the control board firmware's default offset (8900) is for the pneumatic gripper holder, not for the MSG's ring cover. After the first homing, check joint 5 with the pin through its witness marks. If it is off, measure the offset and set `j5_homing_offset` in `main.cpp` of the control board firmware, as the PAROL6 documentation's mastering page describes.

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, up to about ten 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(10)`, then `Gripper(position, speed, current)`, with the current in mA.

Python API

```
import time

rbt.select_tool("MSG", variant_key="150mm")           # the parol6 package; 100mm, 150mm or
200mm: the rail
rbt.tool.calibrate()                                   # with the robot at rest
time.sleep(10)                                         # the calibration takes up to about ten
seconds
rbt.tool.set_position(0.8, speed=0.5, current=0.4)     # fractions; current 0.4 is 1180 mA
print(rbt.tool.status())
```

The API allows the calibration 2 s before it sends the next gripper action, which is too short for the MSG; the pause above waits for the calibration to finish.

The Python API's `current` is a fraction of the range 100 to 2800 mA: the gripper gets 100 + 2700 × `current` mA, so 1.0 is 2800 mA, more than any of the steppers takes. Keep it within the stepper's maximum (Force and speed):

Stepper	Maximum current	Largest `current`
21.5 mm	1.5 A	0.51
40 mm	2.0 A	0.70
60 mm	2.1 A	0.74

`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 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.

Another robot

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 stepper 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 on the motor housing 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 stepper 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 24 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 stepper cover or touch the driver, unplug the supply and touch earthed metal: static electricity damages the driver board. Hold the board by its edges.

Before each session

- The jaws sit tight on their carriers, nothing is cracked, the pads or the sponge tape are whole.
- 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 and the racks of chips and dust, as Cleaning describes.
- **Screws**: the jaw screws, the rail screws and the clamp screws of the shaft coupler; threadlocker where the assembly instructions use it.
- **The camera lens**, with a lens cloth.

Cleaning

Clean the gripper switched off and unplugged. Wipe the printed parts with a dry or slightly damp cloth, or with a little isopropyl alcohol on the cloth for grease; use no other solvents, such as acetone: the body is PETG, the pads are TPU. Never spray or pour anything onto the gripper; the driver behind the stepper cover and the M8 connector are open to it, and it is not waterproof or dustproof. Brush or blow chips out of the rail and the racks rather than wash them out, then grease them as above.

Replacing worn parts

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

A stepper or a driver is not serviced but replaced: the stepper by one of the three in the BOM, after which the motor calibration of Driver configuration is run again; the driver by a new STEPFOC driver, configured the same way.

Firmware

The gripper has no firmware of its own: it runs the STEPFOC 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 104 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 `stepfoc-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 STEPFOC 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 `Software version:` with the new release (over CAN, the third byte of Send_Respond_Device_Info), and the gripper calibrates and moves as before. If it no longer answers as a gripper, its configuration was cleared: restore it as in Driver setup and first run.

A gripper whose driver runs firmware 103 or older has no bootloader; it is updated once with a programming adapter on the driver's JTAG pads, inside the housing, as described in Installing the bootloader on an older board. From then on it updates over CAN like any other.

Release notes

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

9. Troubleshooting

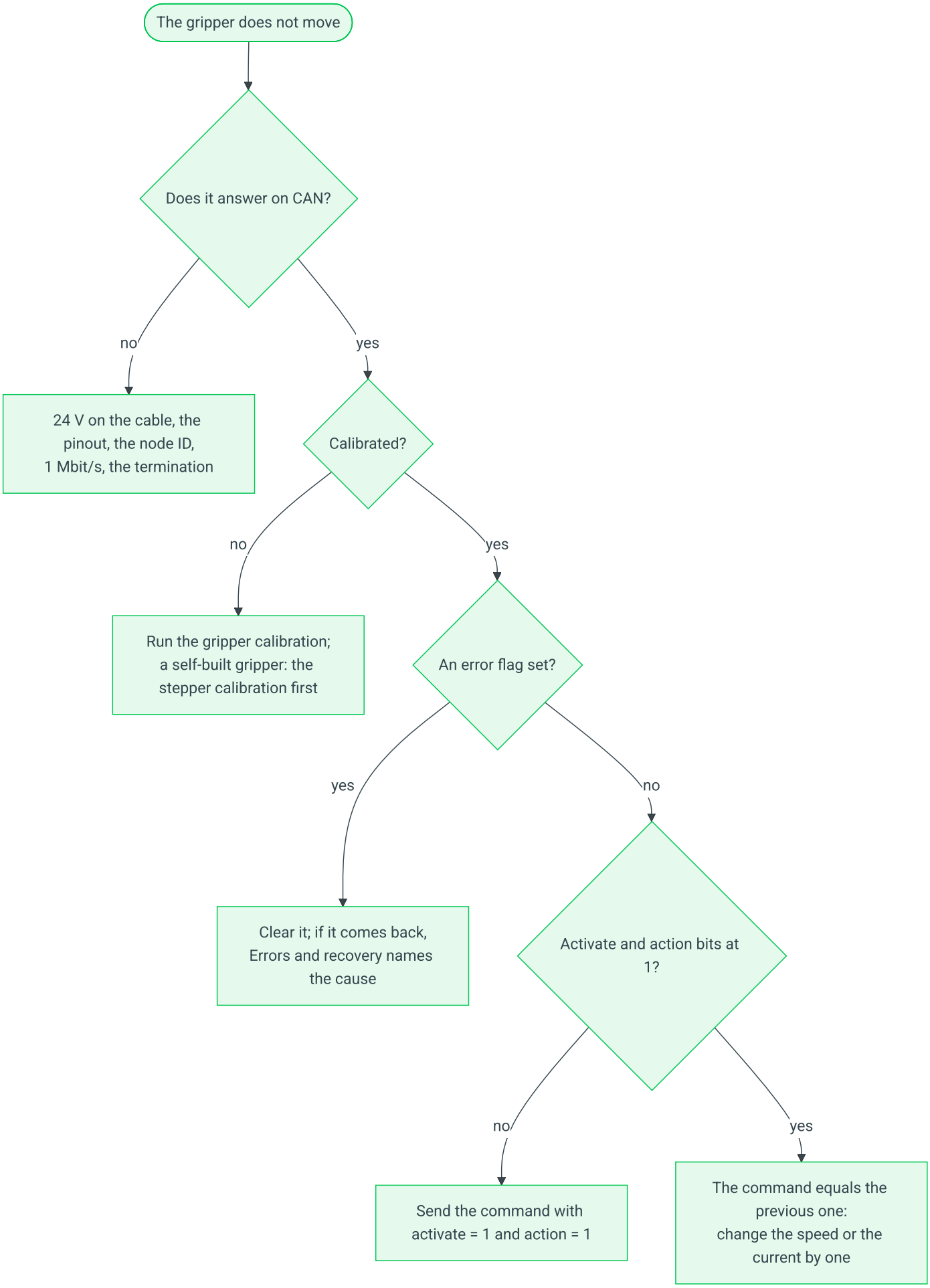
Troubleshooting

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, with what to do until they are, are in 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 stepper 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 stepper itself is not calibrated (a self-built gripper): Driver configuration

Calibration

Symptom	Cause
the jaws move to one end and stop; after 24 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 stepper not connected (self-built), or the stepper 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 jaw faces: sponge tape or the TPU pad
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 stepper 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 the thermistor on its motor 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 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.

STEPFOC firmware

Seen in: 108 (the driver's firmware, gripper mode).

	What happens	What to do	More
KI-1	After a gripper command with the E-stop bit set, the E-stop error comes back at every control cycle for as long as the last gripper command carries the bit, so a clear seems to do nothing.	First send a gripper command with the E-stop bit at 0 and the action bit at 0 (`#Gripstop 0` over UART), then clear.	Errors and recovery, clearing
KI-2	The answer to a full gripper command is sent before the gripper acts on it, so it carries the state from before the command, for example object detection 3 of the move before.	Ignore that answer. Poll with empty Send_gripper_data_pack frames, and give a move about 100 ms before you trust its object detection.	How the gripper works, the answer
KI-3	A full gripper command or a clear sent while the gripper calibration runs ends it, and the gripper stays uncalibrated. The routine's progress and its 24 s timer are not reset, so the next calibration can start part way through or give up early.	Clear first, then calibrate, and send only empty status polls until the calibrated bit is 1. After an interrupted calibration, switch the gripper off and on before you calibrate again.	How the gripper works, the states
KI-4	The action status in the gripper's answer is the action bit of the last command as it was received; the gripper never clears it, so it stays 1 after a move has finished and while the gripper holds.	Follow a move by its object detection (0 moving, 1 or 2 object found, 3 at the position), not by the action status.	How the gripper works, the answer
KI-5	The motor calibration (`#Cal`) leaves the driver's current limit at 1600 mA,	After `#Cal`, set `#Ilim` to the stepper's maximum current (1500,	Driver setup, current limit

	What happens	What to do	More
	whatever it was before, which is more than the 21.5 mm stepper's 1.5 A. Gripper moves use the current of each command; the limit applies when the driver runs as a motor (Motor GUI's Motor side, the UART motor modes).	2000 or 2100) before `#Save`, or apply Motor GUI's MSG preset, which sets the current limit.	

Python library (Spectral-BLDC)

Seen in: 1.20.

	What happens	What to do	More
KI-6	`UnpackData` assembles `gripper_object_detection` with its two bits swapped: 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 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-7	`tool.status()` does not follow the gripper's status byte: `engaged` reads true at all times, and `part_detected` comes from bits 3 and 2 of the byte, which hold the temperature and timeout errors, not the object detection. The calibrated and error bits are not in it.	Use the jaw position (`positions`) and the current (`channels`), and check where the jaws stopped to know that a part is held.	On PAROL6 and PAR6, driving it

	What happens	What to do	More
KI-8	`calibrate()` counts the calibration done after a fixed 2 s without reading the calibrated bit; the next gripper action then ends a calibration that is still running, and the MSG's takes up to about ten seconds. The API has no action that clears the gripper's errors.	Wait about 10 s after `calibrate()` before the next gripper action. Clear a gripper error with the commander's **Clear gripper error** or with Motor GUI, then calibrate.	On PAROL6 and PAR6, driving it
KI-9	The MSG's `current` is a fraction of 100 to 2800 mA, so `current=1.0` commands 2800 mA, more than any of the three steppers takes, and the firmware does not clamp it.	Keep `current` at or below 0.51 (21.5 mm stepper), 0.70 (40 mm) or 0.74 (60 mm).	On PAROL6 and PAR6, driving it

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 MSG is open source, and that is important to us. Everything the gripper is made of is published in the product repository under the CERN Open Hardware Licence: the STEP and print files of every part, the bill of materials with a link for every item, the assembly video, 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; change the jaws, the camera mount or the body for your application, or fit the gripper to a robot it 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 on the motor, the current within the stepper's maximum, and the Safety chapter apply to it as to a bought one. The gripper is in beta: expect rough edges in the hardware and the software.

The steps

1. Choose a rail length for the stroke and a stepper length for the force: Variants.
2. Bill of materials: the repository's BOM, with the reference images.
3. Printing the parts: PETG at 25 % infill, the folders for your rail and stepper, the jaws.
4. Mechanical assembly: the video, the order, the cable.
5. Driver setup and first run: the motor calibration before the pinion goes in, gripper mode, the thermistor, the first moves.

Kits and the shop

You do not have to source everything yourself:

	What is in it	You add
STEPFOC starter kit	the STEPFOC 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 stepper, the rail and its carriages, the pinion, the shaft coupler, the M8 connector, the screws and inserts, the camera, the printed parts
STEPFOC driver alone	the driver	the rest, and the adapters its setup needs

	What is in it	You add
Assembled MSG gripper	the gripper, built, calibrated and configured, in the rail, stepper and colour you choose	the cable

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.md in the gripper's repository; this page is that file as of 2026-10-31 (commit `9862f62`), 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 MSG 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
M3 nut		1		
M4	10 mm	2	Screw	
M3	6 mm	4	Cheese head screw	
M3	25 mm	4	Screw	
M3	8 mm	6	Screw	
M2	10 mm	1	Screw	
M2	4 mm	8	Screw	
M2.5	8 mm	6	Screw	
M2	6 mm	4	Screw	
M3	22 mm	4	For 17HS08-1004S stepper	

Diameter / Type	Length	Quantity	Description	Link
M3	40 mm	4	For 17HS16-2004S1 stepper	
M3	60 mm	4	For 17HS24-2104S stepper	

Brass Inserts

Thread	Length	Outer Diameter	Quantity	Description	Link
M3	5 mm	4.5 mm	8		Link

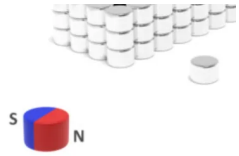
Motors (You need one of these per gripper!)

Type	Quantity	Description	Link
17HS08-1004S	1	21.5mm stepper 	Link
17HS16-2004S1	1	40mm stepper 	Link

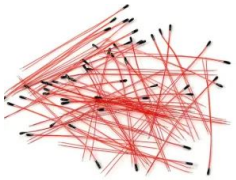
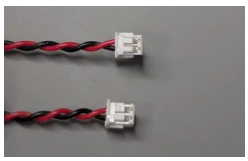
Type	Quantity	Description	Link
17HS24-2104S	1	60mm stepper 	Link

Mechanical Components

Type	Quantity	Description	Link
Shaft coupler	1	8 mm bore, M4 screw holes 	Link
MGN7C slider	2	Linear guide carriage 	Link
Linear guide	1	100 mm, 150 mm and 200 length works 	Link

Type	Quantity	Description	Link
Diametrical Neodymium Magnet	1	6 mm diameter, 2.5 mm thickness 	Link
Magnet 8 x 2.5mm or 8 x 3mm	1		Link
Gear 32P 21T for 5mm shaft	1		Link
Thermal cement	1		Link
Sponge tape — 30 mm width, 1 mm thickness	1		Link

Electronics

Type	Quantity	Description	Link
STEPFOC motor controller	1	Motor controller 	Link
100K Thermistor	1	NTC temperature sensor 	Link
XH 2.54 mm cable	1	300 mm, 2P, double-ended (24 V power) 	Link
ZH 1.5 mm cable	1	300 mm, 2P, double-ended (CAN bus) 	Link
M8 Male 4-pin connector	1		Link
OV9732 Camera	1	100 Degree version 	Link

Cables

> **Note**
>
> **PAROL6 and PAR6 robots use different cables for SSG48 and MSG grippers. Using incorrect one can damage your robot**

Official cables from Source Robotics:

- PAROL6 cable
- PAR6 cable - Coming soon

If you want to build your own cable, use the parts below and follow the wiring guide.

> **Tip**
>
> See the **cable building guide** in the Assembly manual for full wiring diagrams for both the PAR6 and PAROL6 robots.

Type	Length / Spec	Quantity	Description	Link
26 AWG cable	0.15 mm², 4-wire twisted pair	0.4 m	~40 cm usually sufficient 	Link
M8 4-pin male wire connector	—	1	M8 4Pin Male 	Link

Type	Length / Spec	Quantity	Description	Link
M8 4-pin female wire connector	—	1	M8 4Pin Female 	Link

Note

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

Printing the parts

Material and settings

PETG for every part but the TPU jaw pads. The repository's print table is one line, and it is the whole rule: **use 25 % infill for all** (Print_table.md). Supports where your slicer asks for them; the body prints on its back.

The files

Variant	Folders in STEP files
every gripper	`Common parts`: the coupler plate and ring cover (or `ring_cover-PAROL6`), the camera mount and lid, the jaw blocker, the stepper spacer, the connector lid
your rail	`Linear rail dependant parts /<rail>`: the body and the two jaw carriers
your stepper	`Stepper dependant parts /<stepper>`: the stepper cover, and the spacer for the 21.5 mm stepper
the jaws	`Gripper jaws attachment`: one of the designs, mirrored for the other side

The Print files folder has the same parts as 3MF, ready to slice.

The jaws

Two kinds, both in `Gripper jaws attachment`; mirror the file in the slicer for the other side, except the solid jaw, which has a mirrored file:

- **TPU jaws** (recommended): a PETG body, `MSG_JAW_TPU_GLUE`, and a soft TPU pad printed separately and glued on. The pad deforms around the object for a gentler, surer grip. Print the pad with **16 % infill, no top or bottom layers and 2 walls** for the softest grip; the body with the normal settings.
- **Solid jaws**: `MSG_solid_jaw` and `MSG_solid_jaw_mirror` in one piece, or `MSG_JAW_HOLLOW` for a two-colour print; `MSG_JAW_TPU_GLUE` prints solid too.

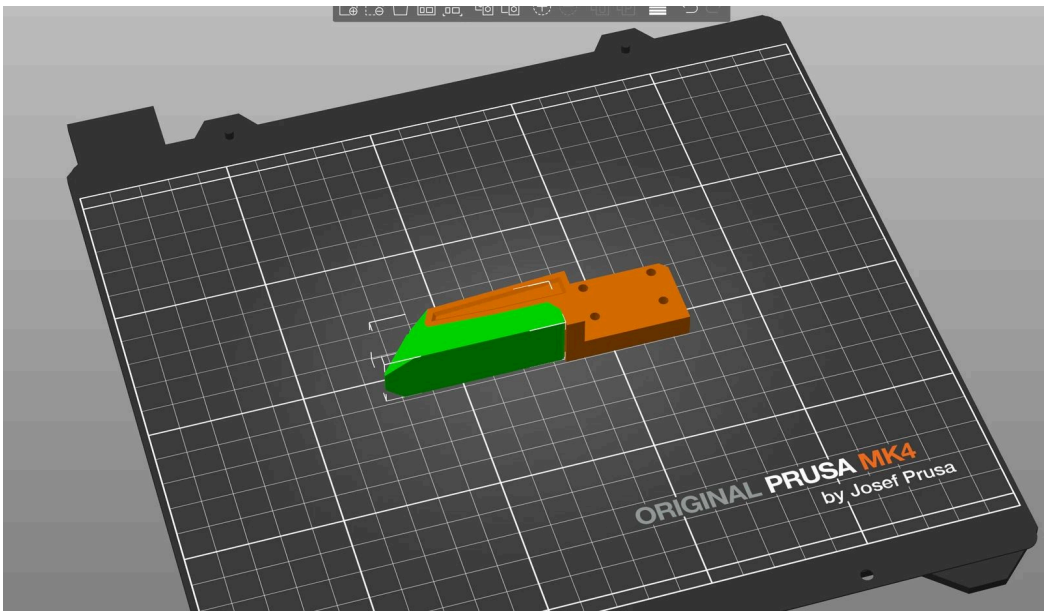


Figure 1: The TPU jaw in the slicer: the PETG body (orange) and the TPU pad (green) as separate parts.

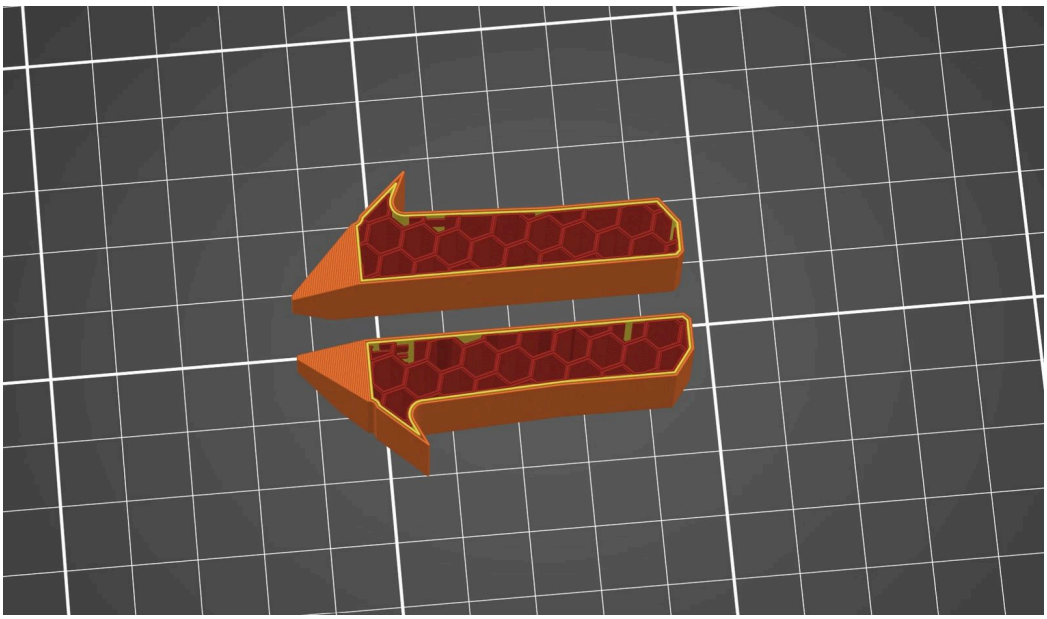
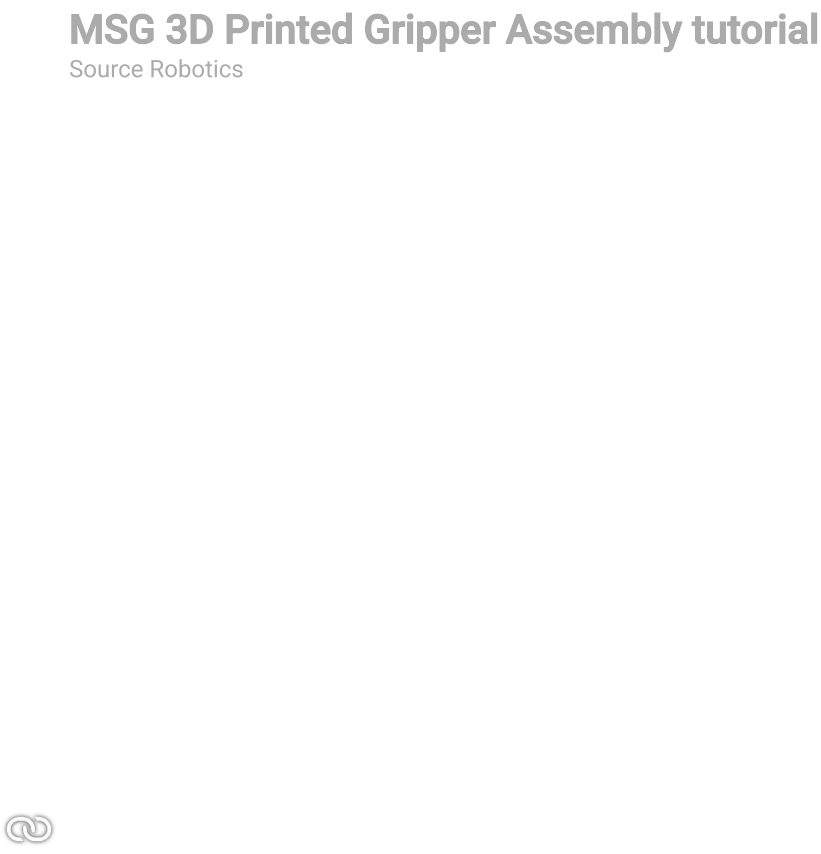


Figure 2: Two TPU pads sliced with the pad's settings: the honeycomb infill shows, with no top layer.

Gluing the pad on, its orientation and the sponge tape for the gripping faces are in Jaws; the repository's jaw guide is the original.

Mechanical assembly

The assembly is shown in the repository's video; a PDF manual is in preparation. Watch the video once before you start, with the parts in front of you.



Open the video on YouTube

The order

Touch earthed metal before you pick up the driver, and hold it by its edges: static electricity damages it.

1. The driver first: calibrated, in gripper mode, with the thermistor, as Driver setup and first run describes. The motor calibration turns the stepper several revolutions, which the racks cannot follow, so it happens before the pinion meshes.
2. The brass inserts into the printed parts with a soldering iron.
3. The rail with its two carriages into the body, the racks onto the carriers.
4. The stepper, with the pinion on its shaft and the driver on its back, into the body so that the pinion meshes with both racks; the thermistor's leads to the driver.

5. The driver's power and CAN cables to the M8 connector in its lid; the lid onto the body.
6. The stepper cover over the driver, with the termination switch reachable through its opening.
7. The coupler plate with the shaft coupler on the back, or `ring_cover-PAROL6` for the PAROL6 wrist.
8. The jaws onto the carriers, the camera into its mount.

> ✓ **Check**
>
> With the supply off, push one jaw by hand over the whole stroke: both jaws move together, smoothly, without a tight spot, and nothing rubs on the cable. The termination switch is reachable through the stepper cover. A stiff spot now makes the gripper calibration stop short later (Calibration).

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 jaws

Printing, gluing and padding the jaws: Printing the parts and Jaws.

Driver setup and first run

A gripper from the shop arrives with its driver configured: the stepper calibrated, gripper mode on, the thermistor on, the gains set, everything saved. A gripper you build has a bare STEPFOC driver that must be brought to the same state once, over its UART port. This page is that procedure; the commands are the driver's, and the STEPFOC documentation explains each in depth.

What you need

- The driver mounted on the stepper with its encoder magnet, before the pinion is engaged in the racks: the motor calibration turns the motor several revolutions in both directions, which the racks cannot follow. Do this step with the stepper out of the gripper or with the pinion not yet fitted.
- 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 STEPFOC site shows the terminal setup.
- The thermistor from the BOM, soldered to the driver's thermistor connector and fixed to the motor housing with thermal cement.

Static electricity damages the driver: touch earthed metal before you handle it, hold it by its edges, and connect and disconnect its cables with the supply off.

With Motor GUI

Motor GUI, connected over **UART** through the serial adapter, has a preset per stepper on its setup page: *MSG 21.5 mm motor*, *MSG 40 mm motor* and *MSG 60 mm motor*. A preset writes the stepper's resistance, inductance and 50 pole pairs, the current limit, the loop gains, gripper mode, the thermistor and the CAN ID, then saves; its CAN ID is 6, the PAR6 gripper's, so change it before applying for another robot. Apply it after the motor calibration of step 1: it does step 2 for you.

1. The motor calibration

Restore the defaults first, so that nothing from another motor or junk in the EEPROM goes into the calibration, then enter the stepper's parameters and calibrate:

Stepper	#PP	#R	#L
21.5 mm, 17HS08-1004S	50	3.7	0.0045
40 mm, 17HS16-2004S1	50	1.1	0.0026
60 mm, 17HS24-2104S	50	1.6	0.0030

```
#Default
#PP 50
#R 1.1
#L 0.0026
#Cal
```

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

Check

The terminal ends with `Calibration success!`, and `#Info` prints `Calibrated`, `Number of pole pairs: 50` and the resistance and inductance you entered. If it ends with `Calibration failed!`, the STEPFOC page Calibration explains each message.

2. The gripper settings

```
#Gripper 1
#Term 1
#Tlim 75
#Kpiq 1.4047
#Kiiq 0.1149
#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
`#Kpiq 1.4047`, `#Kiiq 0.1149`	the current loop gains of Motor GUI's gripper presets, the same for the three steppers; the gains the calibration computes for the stepper are sharper than a gripper needs
`#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

Check

After a power cycle, `#Info` prints `CAN ID is: 0` and `Gripper: 1`, and `#Gripinfo` prints `I am a gripper`. If not, the settings were not saved: enter them again and end with `#Save`.

3. Assemble and calibrate the gripper

Fit the pinion and assemble the gripper, power it up, 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 opened fully and closed fully, `#Gripinfo` prints `Gripper calibrated is: 1`, and a move at a low current goes to its position. If the calibration gives up after 24 s: Calibration.

Current limit

The gripper's moves use the current limit of each command, not the driver's `#Ilim`. Keep the commanded current at or below the stepper's maximum current (Force and speed); the firmware does not clamp it.

Appendix

Performance tests

A thermal test of the MSG gripper. The thermal limit is 75 °C at the thermistor on the motor housing.

> **Warning**
>
> A gripper built without the thermistor on its motor 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

Firmware release notes

What changed in each release of the STEPFOC 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.

V108 (2026-08-26)

STEPFOC firmware **V108** — application image for flashing via the Spectral bootloader.

Bug fixes

- Fixed calibration (`#Cal`) spuriously failing with `Phase order is wrong!` on otherwise-healthy motors. The "confirm commutation" pre-check (before the real angle-offset search) commands a pure torque-axis voltage using the not-yet-calibrated encoder angle — depending on a given unit's actual rotor-to-encoder mounting angle, that command could land almost entirely on the wrong axis and produce ~zero torque, failing the check even though nothing was actually broken. Bench-confirmed on a real unit.
- Fix: retry the pre-check once at a 90° angle bias before declaring failure — guaranteed to land within 45° of the true torque axis on at least one of the two tries. The downstream angle-offset search is unaffected and already independent of this pre-check.
- Added `Spin-confirm delta` / `Vbus` diagnostics to `#Cal`'s failure report, for the rare case both tries still fail.

Assets

- `stepfoc-firmware-V108.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: `25cbc4887ca2c192eba6e708930eb6b6c677fd1e76299313c3b3ce3459e0e8a1`
- Size: 99188 bytes

Built from commit 46b4911 on `bootloader-version`. Includes bootloader entry support (CAN/UART reset-into-bootloader).

V107 (2026-08-24)

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

Bug fixes

- Fixed PD mode (`IN_DATA_PACK_PD` over CAN): the commanded speed was written into `Feedforward_speed`, but `PD_mode()` reads `Velocity_setpoint` for its KD term — so the KD (velocity-damping) term of the impedance controller was always evaluated against a stale/zero setpoint and had no effect. The CAN handler now writes the

field the control law actually reads. `IN_DATA_PACK_1` / position mode is unaffected — `Feedforward_speed` there is correct and continues to feed the position cascade.

CAN bus

- Added a generic settings command (`IN_SETTINGS`) so future boolean settings don't each need their own command ID. Payload is `mask` + `value` (one byte each), one bit per setting — only bits set in the mask are applied, so a sender only needs to know about the bits it cares about.
- First setting on this command: brake vs. coast (bit 0), previously only selectable over UART (`Brake` / `Coast`), now selectable over CAN as well.

Assets

- `stepfoc-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: `63699037c79318e1d78738edcb13e9f9bc00a3393dc5803f920b244a60622ae2`
- Size: 98972 bytes

Built from commit 04ec180 on `bootloader-version`. Includes bootloader entry support (CAN/UART reset-into-bootloader).

V106 (2026-07-05)

STEPFOC 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).
- `#Error` now also reports driver faults (undervoltage/thermal/overcurrent), not just the other error types.
- `#Info` now always reports live ISR execution time, useful for checking control-loop headroom on the bench.

Assets

- `stepfoc-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: `bc061fb273e7c836e3180fa597ecd373e7e357b0eab9130b63a7fd19b70a6073`
- Size: 98940 bytes

Built from commit 661e7d3 on `bootloader-version`. Includes bootloader entry support (CAN/UART reset-into-bootloader).

V105 (2026-07-04)

STEPFOC firmware **V105** — application image for flashing via the Spectral bootloader.

What's new

- Reversed stepper phase wiring now calibrates automatically — no more "switch phases and try again." The firmware detects the wiring and compensates it in software, so **all 8 phase-wiring permutations calibrate identically and symmetrically**.
- `#Info` now reports the firmware's own build version, so the version is correct immediately after flashing (no `#Default` needed).

Assets

- `stepfoc-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: `0d80ce3cf78109f6ff54697425ed475ffca4e97e220d72fe0823ce1bd60a4908`
- Size: 95924 bytes

Built from commit bda57d6 on `bootloader-version`. Includes bootloader entry support (CAN/UART reset-into-bootloader).

V103 (2025-12-28)

First stable release for public

V102 (2025-12-28)

First official release for the public

V101 (2025-09-29)

Initial fully functional release based on spectral micro firmware.

Versions and compatibility

Which STEPFOC 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 (STEPFOC)	Released	Bootloader	MSG gripper	Python library (Spectral-BLDC)	ROS 2 driver	Docs (gripper release)
108	2026-08-26	required (installed on grippers shipped with 104 or newer)	V1.2, V1.1	1.20 or newer	msg_gripper_ros2 (ROS 2 Humble)	1.2
107	2026-08-24	required	V1.1	1.20 or newer	msg_gripper_ros2	not published
106	2026-07-05	required	V1.1	1.19 or newer	msg_gripper_ros2	not published
105	2026-07-04	required (the first release that needs it)	V1.1	1.19 or newer	msg_gripper_ros2	not published
103	2025-12-28	none (flashed with a programming adapter)	V1.1	1.17 or newer	msg_gripper_ros2	the STEPFOC site, versic 103

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 stops the move and holds the jaws still, 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 driver use.

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, so the current limit sets the force with which the jaws press on an object.

Driver

The STEPFOC board on the back of the gripper's stepper: 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 stepper's current as a vector relative to the rotor, from the driver's encoder. It turns the stepper into a motor whose torque is known and limited, which is what makes a force-controlled gripper out of it.

Firmware release

One build of the STEPFOC firmware, numbered 103, 104, 105, … and tagged `V103`, `V104`, … 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 stepper (`#Cal`): the pole pairs, resistance and inductance entered, 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 shaft 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 500 to 60000 encoder ticks per second of the motor.

Stroke

How much the distance between the jaw carriers changes from fully open to fully closed: 55, 105 or 155 mm, set by the rail length. 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 MSG gripper, 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, and the faults known in each are on Known issues.

Firmware and software

What	Version	Download	How to install
The driver's firmware (STEPFOC), which is the gripper's firmware	108	the image `stepfoc-firmware-V<number>.bin` and its manifest in the releases; the source at the tag, STEPFOC firmware	Firmware
Bootloader and its host tool, `host_tools/bl_flash.py`		stm32f103c-bootloader	Firmware: updating
Motor GUI, the browser tool for the driver and the gripper	runs in the browser, always the latest	motorgui.com	Motor GUI
Python library (`Spectral-BLDC`)	the one in Versions and compatibility	Spectral-BLDC-Python, `pip install Spectral-BLDC`	With Python
ROS 2 driver	main branch	source_robotics_msg_gripper_ros2	With ROS 2
CAN definitions (JSON, DBC) for this firmware release, and the frame tool		CAN frame tool and downloads	
The legacy gripper GUI, a Python program		SSG-gripper-GUI	

The firmware's version link opens the source at exactly that release. Motor GUI and the ROS 2 driver have no releases these pages are tied to: Motor GUI is served in its latest version, the ROS 2 driver from its main branch.

CAD and build files

The latest files, on the main branch of the product repository; the gripper release these pages describe is 1.2 (2026-11-04).

What	Where
STEP files of every part, by rail length and stepper length	STEP files in the product repository
Print files (3MF) for the body, the covers and the jaws	Print files
Jaw designs: solid, hollow, with a TPU pad	Gripper jaws attachment
Bill of materials	BOM.md, and on this site: Bill of materials
Assembly video	YouTube
Cable wiring diagrams	Building_cables.md
URDF and MJCF (MuJoCo) models, standalone and on PAROL6 and PAR6	the msg_description guide; the files are in robot_descriptions

More

What	Where
The driver's documentation	docs.source-robotics.com/STEPFOC-docs
The driver's repository	STEPFOC-stepper-controller
CANvas USB-to-CAN adapter	CANvas-adapter
Safety warning and disclaimer	SAFETY_WARNING_AND_DISCLAIMER.md
This documentation	MSG-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 STEPFOC stepper 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 STEPFOC 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 STEPFOC 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, keep it at or below the stepper's rated current (Force and speed)
4	7	Activate	bit	1 activate the gripper, 0 deactivate
4	6	Action	bit	1 go to the position, 0 stop and hold the jaws still
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. 108
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
`#L`	Set/Get	float	Driver setup
`#LED`	Set/Get	int	Status and utility
`#Magnet`	Get		Status and utility
`#Mode`	Get		Status and utility
`#PP`	Set/Get	int	Driver setup
`#R`	Set/Get	float	Driver setup
`#Reset`	Action		Status and utility
`#Save`	Action		Driver setup

Command	Kind	Argument	Group
`#Temp`	Get		Status and utility
`#Term`	Set/Get	int	Driver setup
`#Tlim`	Set/Get	int	Driver setup
`#Vbus`	Get		Status and utility

The driver's other commands (motion, gains, the calibration settings) are in the STEPFOC 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 grip force. The firmware does not clamp it; keep it at or below the stepper's rated current (see Force and speed).	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
`#R`	Set/Get	float: resistance in Ω	The phase resistance of the motor (one coil). STEPFOC does not measure it; enter the datasheet value before Cal.	the current value
`#L`	Set/Get	float: inductance in H	The phase inductance of the motor (one coil); enter the datasheet value before Cal.	the current value
`#Kt`	Set/Get	float: Nm/A	The torque constant, found by calibration.	the current value
`#Cal`	Action		Runs the driver's motor calibration (see Driver configuration): enter PP, R and L first, with the pinion not yet in the rack.	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	`Loaded default config!`

Command	Kind	Argument	What it does	Response
			gripper flag, heartbeat and watchdog. Back those up first.	
`#Term`	Set/Get	int: 0 or 1	Enables the thermistor measurement and the temperature error (off by default).	the setting
`#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`
`#PP`	Set/Get	int: pole pairs	The motor's pole pairs: 50 for a 1.8° stepper, 75 for 1.2°, 100 for 0.9°, 125 for 0.72°. Calibration accepts only these values.	the current value

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. In gripper mode (mode 11) it prints nothing; `#Gripinfo` shows the gripper's state.	`Idle mode`, `Position control`, `Speed control`, `Current control`, `PD control`, `Open loop`, `Gripper calib mode` during the gripper calibration; `Gripper mode` is what it prints for mode 6, voltage torque
`#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	`Clear errors`

Command	Kind	Argument	What it does	Response
			`#Calibrated 1`). Enters **idle** mode.	
`#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 on the motor housing 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 STEPFOC 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, with a longer wait for the calibration: the README's 4 s is too short for the 150 and 200 mm rails, and a move command sent while the calibration still runs ends it. `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 14 N with the 21.5 mm stepper and 25 N with the 60 mm. 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(10)                 # the calibration: up to about ten seconds; send nothing meanwhile

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 stepper'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 stepper'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 STEPFOC 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 rail and the stepper, the jaws.
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/MSG-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 STEPFOC 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 CERN-OHL-S v2 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.